Збирка решени задачи по предметот напредно програмирање

Ѓорѓи Маџаров
Стефан Андонов

Факултет за информатички науки и компјутерско инженерство - Скопје
Скопје, 2023

Издавач:
Универзитет „Св. Кирил и Методиј“ во Скопје
Бул. „Гоце Делчев“ бр. 9, 1000 Скопје
www.ukim@ukim.edu.mk

Уредник за издавачка дејност на УКИМ:
проф. д-р Никола Јанкуловски, ректор

Уредник на публикацијата:
проф. д-р Ѓорѓи Мацаров, Факултет за информатички науки и компјутерско инженерство – Скопје

Рецензенти:
1. д-р Ивица Димитровси
   редовен професор
   Факултет за информатички науки и компјутерско инженерство - Скопје
   Универзитет „Св. Кирил и Методиј“ - Скопје
2. д-р Иван Китановски
   вонреден професор
   Факултет за информатички науки и компјутерско инженерство - Скопје
   Универзитет „Св. Кирил и Методиј“ - Скопје

Техничка обработка:
д-р Ѓорѓи Мацаров
м-р Стефан Андонов

Лектура на македонски јазик:
м-р Маја Тефова

CIP - Каталогизација во публикација
Национална и универзитетска библиотека „Св. Климент Охридски Скопје

004.434.045Java(075.8)(076)

МАЦАРОВ, Горѓи
Збирка решени задачи по предметот напредно програмирање [Електронски извор] / Горѓи Мацаров, Стефан Андонов. - Скопје : Универзитет "Св. Кирил и Методиј Скопје, Факултет за информатички науки и компјутерско инженерство, 2023

Начин на пристапување (URL):
http://www.ukim.edu.mk/mk_content.php?meni=53&glavno=41. - Текст во PDF формат, содржи 151 стр., илустр. - Наслов преземен од екранот. - Опис на изворот на ден 08.02.2023. - Регистар

ISBN 978-9989-43-484-6

1. Андонов, Стефан [автор]
а) Компјутерско програмирање -- Java(програмски јазик) --
Објектно-ориентирано програмирање -- Високошколски учебници -- Вежби

COBISS.MK-ID 59320837

Содржина

1. Објектно-ориентирано програмирање во Јава

Објектно-ориентираното програмирање или скратено ООП е доминантна програмска парадигма што ги заменува структурираните или процедуралните техники на програмирање, развиени во седумдесетите години на дваесетиот век. Објектно-ориентираната програма е претставена преку објекти кои меѓусебно комуницираат со испраќање на пораки.

Секој објект има одредена специфична функционалност, изложена пред корисниците и скриена имплементација. Многу објекти кои се користат во рамки на програмите се веќе дефинирани во одредени библиотеки и се користат согласно нивната постоечка дефиниција и имплементација.

Објектите кои сè уште не се дефинирани или имплементирани, корисникот треба да се погрижи за нивната дефиниција и имплементација. Дали кориникот ќе креира свој објект или истиот ќе го преземе од некоја постоечка библиотека зависи од повеќе фактори, како на пример време или буџет со кои располага. Но, во основа, сè додека некој објект ги задоволува корисничките спецификации, не е значајно кој и како ја имплементира неговата функционалност.

1.1 Класи и објекти

Во рамките на ова поглавје е дадено решение на едноставен проблем од објектно-ориентирано програмирање. Претставени се основните концепти за креирање на класа, дефинирање на нејзините членови (атрибути и методи) како и начинот на инстанцирање на објекти од соодветната класа. Опишани се основните критериуми за енкапсулација и интеракција.

> Проблем 1.1 Да се напише класа која чува низа на цели броеви `IntegerArray`. Класата треба да е `immutable`. Тоа значи дека, откако еднаш ќе се инстанцира, не може да се менува состојбата на објектот, односно не може да се менуваат податоците зачувани во него и не може да се наследува од неа `final`. За потребите

на класата треба да се имплементираат следните методи:

- `IntegerArray(int a[])` - конструктор кој прима низа од цели броеви
- `length():int` - метод кој ја враќа должината на низата
- `getElementAt(int i):int` - го враќа елементот на позиција `i`, може да претпоставите дека индексот `i` секогаш ќе има вредност во интервалот `[0,length()-1]`
- `sum():int` - метод кој ја враќа сумата на сите елементи во низата
- `average():double` - метод кој ја враќа средната вредност на елементите во низата
- `getSorted():IntegerArray` - враќа нов објект од истата класа кој ги содржи истите вредности од тековниот објект, но сортирани во растечки редослед
- `concat(IntegerArray ia):IntegerArray` - враќа нов објект од истата класа во кој се содржат сите елементи од `this` објектот и по нив сите елементи од `ia` објектот, притоа запазувајќи на нивниот редослед
- `toString():String` - враќа текстуална репрезентација на објектот каде елементите се одделени со запирка и едно празно место после запирката и на почетокот и крајот на стрингот има средни загради. Пример за низа која ги содржи боревите 2,1 и 4 враќа "[2, 1, 4]"

*Забелешка сите методи треба да се `public`

Покрај класата `IntegerArray` треба да се напише дополнително уште една класа која ќе служи за читање на низа од цели броеви од влезен тек на податоци. Оваа класа треба да се вика `ArrayReader` и во неа треба да се имплементира `public static` метод за читање на низа од цели броеви од `InputStream`.

- `readIntegerArray(InputStream input):IntegerArray` - чита низа од цели броеви од `input` зададена во следниот формат: Во првата линија има еден цел број којшто кажува колку елементи има во низата, а во наредниот ред се дадени елементите на низата одделени со едно или повеќе празни места.

```
1  import java.io.ByteArrayInputStream;
2  import java.io.IOException;
3  import java.io.InputStream;
4  import java.util.Arrays;
5  import java.util.Random;
6  import java.util.Scanner;
7
8
9  class IntegerArray {
10
11   private int a[];
12
13   public IntegerArray(int a[]) {
14     this.a = Arrays.copyOf(a, a.length);
15   }
16
17   private IntegerArray(int a[],boolean to_copy) {
18     if ( to_copy )
19       this.a = a;
20     else
21       this.a = Arrays.copyOf(a, a.length);
22   }
23
```

```
24    public int length() {
25      return a.length;
26    }
27
28    public int getElementAt(int i) {
29       return a[i];
30    }
31
32    public int sum() {
33      int sum = 0;
34      for (int e : a) sum += e;
35      return sum;
36    }
37
38    public double average() {
39      return sum() * 1.0 / length();
40    }
41
42    public IntegerArray getSorted() {
43      int sorted_a[] = Arrays.copyOf(a, length());
44      Arrays.sort(sorted_a);
45      return new IntegerArray(sorted_a);
46    }
47
48    public IntegerArray concat(IntegerArray ia) {
49      int res_a[] = new int[a.length+ia.a.length];
50      System.arraycopy(a, 0, res_a, 0, a.length);
51      System.arraycopy(ia.a, 0, res_a, a.length, ia.a.length);
52      return new IntegerArray(res_a,true);
53    }
54
55    public String toString() {
56      return Arrays.toString(a);
57    }
58
59    @Override
60    public int hashCode() {
61      final int prime = 31;
62      int result = 1;
63      result = prime * result + Arrays.hashCode(a);
64      return result;
65    }
66
67    @Override
68    public boolean equals(Object obj) {
69      if (this == obj)
70        return true;
71      if (obj == null)
72        return false;
73      if (getClass() != obj.getClass())
74        return false;
75      IntegerArray other = (IntegerArray) obj;
76      if (!Arrays.equals(a, other.a))
77        return false;
78      return true;
79    }
80  }
81
82  class ArrayReader  {
83
84    public static IntegerArray readIntegerArray(InputStream input) {
```

```
85      Scanner jin = new Scanner(input);
86      int n = jin.nextInt();
87      int a[] = new int[n];
88      for ( int i = 0 ; i < n ; ++i ) {
89        a[i] = jin.nextInt();
90      }
91      jin.close();
92      return new IntegerArray(a);
93    }
94  }
95
96  public class IntegerArrayTester {
97
98      public static void main(String[] args) {
99          Scanner scanner = new Scanner(System.in);
100         String s = scanner.nextLine();
101         IntegerArray ia = null;
102         switch (s) {
103             case "testSimpleMethods":
104                 ia = new IntegerArray(generateRandomArray(scanner.nextInt()
    ));
105                 testSimpleMethods(ia);
106                 break;
107             case "testConcat":
108                 testConcat(scanner);
109                 break;
110             case "testEquals":
111                 testEquals(scanner);
112                 break;
113             case "testSorting":
114                 testSorting(scanner);
115                 break;
116             case "testReading":
117                 testReading(new ByteArrayInputStream(scanner.nextLine().
    getBytes()));
118                 break;
119             case "testImmutability":
120                 int a[] = generateRandomArray(scanner.nextInt());
121                 ia = new IntegerArray(a);
122                 testSimpleMethods(ia);
123                 testSimpleMethods(ia);
124                 IntegerArray sorted_ia = ia.getSorted();
125                 testSimpleMethods(ia);
126                 testSimpleMethods(sorted_ia);
127                 sorted_ia.getSorted();
128                 testSimpleMethods(sorted_ia);
129                 testSimpleMethods(ia);
130                 a[0] += 2;
131                 testSimpleMethods(ia);
132                 ia = ArrayReader.readIntegerArray(new ByteArrayInputStream(
    integerArrayToString(ia).getBytes()));
133                 testSimpleMethods(ia);
134                 break;
135         }
136         scanner.close();
137     }
138
139     static void testReading(InputStream in) {
140         IntegerArray read = ArrayReader.readIntegerArray(in);
141         System.out.println(read);
142     }
```

```
143
144     static void testSorting(Scanner scanner) {
145         int[] a = readArray(scanner);
146         IntegerArray ia = new IntegerArray(a);
147         System.out.println(ia.getSorted());
148     }
149
150     static void testEquals(Scanner scanner) {
151         int[] a = readArray(scanner);
152         int[] b = readArray(scanner);
153         int[] c = readArray(scanner);
154         IntegerArray ia = new IntegerArray(a);
155         IntegerArray ib = new IntegerArray(b);
156         IntegerArray ic = new IntegerArray(c);
157         System.out.println(ia.equals(ib));
158         System.out.println(ia.equals(ic));
159         System.out.println(ib.equals(ic));
160     }
161
162     static void testConcat(Scanner scanner) {
163         int[] a = readArray(scanner);
164         int[] b = readArray(scanner);
165         IntegerArray array1 = new IntegerArray(a);
166         IntegerArray array2 = new IntegerArray(b);
167         IntegerArray concatenated = array1.concat(array2);
168         System.out.println(concatenated);
169     }
170
171     static void testSimpleMethods(IntegerArray ia) {
172         System.out.print(integerArrayToString(ia));
173         System.out.println(ia);
174         System.out.println(ia.sum());
175         System.out.printf("%.2f\n", ia.average());
176     }
177
178
179     static String integerArrayToString(IntegerArray ia) {
180         StringBuilder sb = new StringBuilder();
181         sb.append(ia.length()).append('\n');
182         for (int i = 0; i < ia.length(); ++i)
183             sb.append(ia.getElementAt(i)).append(' ');
184         sb.append('\n');
185         return sb.toString();
186     }
187
188     static int[] readArray(Scanner scanner) {
189         int n = scanner.nextInt();
190         int[] a = new int[n];
191         for (int i = 0; i < n; ++i) {
192             a[i] = scanner.nextInt();
193         }
194         return a;
195     }
196
197
198     static int[] generateRandomArray(int k) {
199         Random rnd = new Random(k);
200         int n = rnd.nextInt(8) + 2;
201         int a[] = new int[n];
202         for (int i = 0; i < n; ++i) {
203             a[i] = rnd.nextInt(20) - 5;
```

```
204         }
205         return a;
206     }
207
208 }
```

Прво се дефинира класата `IntegerArray` во која се чува само една променлива на инстанца - низа од цели броеви `a`. Со цел класата да се прогласи за `immutable`, потребно е класата да се декларира како `final`.

За класата е потребно да се дефинира конструктор со еден аргумент - низа од цели броеви чија содржина ќе биде копирана во низата од цели броеви што се чува како променлива на инстанца. За таа цел се користи готовата функционалност `Arrays.copyOf(...)`.

Методите `length()`, `getElementAt(i)`, `sum()` и `average()` имплементираат тривијални операции со низи од цели броеви.

Методот `IntegerArray getSorted()` соодветно треба да врати нов објект од класата `IntegerArray` во кој низата со цели броеви ќе биде со истите вредности од низата од `this` објектот, но истите ќе бидат сортирани. Тука е важно да се креира копија од низата што се чува во `this` објектот. Копијата да се подреди (со готовата функционалност `Arrays.sort()`) и да се испрати како аргумент во конструктор при креирање на нов објект од класата `IntegerArray`. Честа грешка при решавање на ова задача може да биде повикот на `Arrays.sort()` на низата што се чува во `this` објектот, што ќе придонесе до промена на низата во `this` објектот и во објектот што е резултат од методот.

Во методот `IntegerArray concat (IntegerArray ia)` потребно е да се врати нов објект којшто во себе ќе има низа која пак, ќе се добие како резултат на спојување на низата од `this` објектот и низата од објектот `ia`. За да не се случи промена на низата во `this` објектот, прво е потребно да се постави низа од цели броеви со должина еднаква на сумата на должините на двете низи. Потоа, со функционалноста `System.arraycopy()` се копира содржината на низата од `this` објектот, а веднаш по неа и содржината на низата во `ia` објектот. На ваков начин добиената низа се праќа како аргумент во конструкторот при креирање на резултантиот објект.

Функционалностите со влезни и излезни потоци на податоци, што се потребни во класата `ArrayReader`, ќе бидат подетално објаснати во следните поглавја.

1.2 Наследување и композиција

Во рамките на ова поглавје ќе бидат разгледани два примери кои укажуваат на начинот на кој правилно се имплементира наследување, полиморфизам и композиција во Јава. Во првата задача е ставен акцент на наследување и полиморфизам, додека во втората задача е претставен концептот на композиција на објекти.

Проблем 1.2 Треба да се креира апликација за банка која ќе управува со сметките на повеќе корисници и ќе врши трансакции помеѓу нив. Банката работи со долари.

За потребите на ваквата апликација треба да се напишат класите `Account`, `Transaction` и `Bank`. Класата `Account` претставува една сметка на еден корисник и треба да ги чува следните податоци:

- Име на корисникот,
- единствен идентификациски број (`long`)

- тековното салдо на сметката.

Оваа класа исто така треба да ги имплементира и следниве методи

- `Account(String name, double balance)` – конструктор со параметри (id-то треба да се генерира со помош на класата `java.util.Random`)
- `getBalance():double`
- `getName():String`
- `getId():long`
- `setBalance(double balance)`
- `toString():String` – враќа стринг во следниот формат

```
Name:FirstName LastName
Balance:20.00\$
```

Класата `Transaction` претставува трансакција (префрлување пари од една на друга сметка), од страна на банката за што честопати се наплаќа провизија. За почеток треба да се напише класата `Transaction` со податочни членови за идентификациските броеви на две сметки Едната од која се одземаат парите и другата на која се додаваат парите, текстуален опис и износ на трансакцијата. За оваа класа треба да се имплементираат методите:

- `Transaction(long fromId, long toId, Stirng desc, double amount)`
- `getAmount():double`
- `getFromId():long`
- `getToId():long`

Оваа класа треба да биде апстрактна, бидејќи не е наменета директно да се користи.Таа е само како основна класа за изведување на други класи. Како што беше претходно напомнато, банката наплаќа провизија за одредени трансакции. Има два типа на провизија: фиксна сума и процент. Кај фиксна сума за која било трансакција без
разлика на износот на трансакцијата се наплаќа иста провизија (пример 10$). Кај процент за секој еден долар од трансакцијата банката наплаќа одреден процент провизија (на пример 5%, или 5 центи на секој долар – процентите секогаш се цели броеви и провизија се наплаќа само на цели долари).

За да се прави разлика меѓу различните типови на провизија, треба да се напишат уште две класи кои ќе наследуваат од `Transaction` и кои треба да се именуваат како `FlatAmountProvisionTransaction` и `FlatPercentProvisionTransaction`. Првата класа `FlatAmountProvisionTransaction` треба да содржи:

- `FlatAmountProvisionTransaction(long fromId, long toId, double amount, double flatProvision)` - соодветен конструктор кој го поставува полето за опис на `FlatAmount` и
- `getFlatAmount():double`.

Слично, класата `FlatPercentProvisionTransaction` треба да има:

- `FlatPercentProvisionTransaction (long fromId, long toId, double amount, int centsPerDolar)` - конструктор кој го поставува полето за опис на `FlatPercent` и

- `getPercent():int`.

Исто така, треба да се препокрие `equals(Object o):boolean` методот и за двете класи.

Накрај треба да се имплементира класата `Bank` којашто ги чува сметките од своите корисници и дополнително врши трансакции. Класатам освен сметките на своите корисници, треба да ги чува и сопственото име и вкупната сума на трансфери, како и вкупната наплатена провизија од страна на банката за сите трансакции. Класата `Bank` треба да ги има следните методи:

- `Bank(String name, Account accounts[])` – конструктор со соодветните параметри (направете сопствена копија на низата од сметки)
- `makeTransaction(Transaction t):boolean` – врши проверка дали корисникот ги има потребните средства на сметка и дали и двете сметки на кои се однесува трансакцијата се навистина во банката. Ако и двата услови се исполнети ја извршува трансакцијата и враќа true, а во спротивно враќа false
- `totalTransfers():double` – ја дава вкупната сума на пари кои се префрлени во сите трансакции досега
- `totalProvision():double` – ја дава вкупната провизија наплатена од банката за сите извршени трансакции досега
- `toString():String` - го враќа името на банката во посебна линија во формат

```
Name: Stopanska Banka A.D. Skopje
```

по што следат податоците за сите корисници.

Провизијата се наплаќа така што на основната сума на трансакцијата се додава вредноста не провизијата и таа сума се одзема од првата сметка. За сите класи да се напишат соодветни `equals` и `hashCode` методи. ■

За задачите во кои се очекува да се користат концептите на наследување и полиморфизам, најважно е на почетокот да се идентификуваат врските помеѓу податоците и соодветно да се дизајнираат класите и нивните зависности. Во оваа задача, особено важни се дефинициите на класите за трансакции. Прво, се дефинира класата `Transaction` која е апстрактна класа и од неа се изведуваат две класи за конкретните типови на трансакции. Во задачата не е побарано, но, за класата `Transaction` да биде апстрактна потребно е да се дефинира најмалку еден апстрактен метод. Апстрактни методи се методите кои имаат дефиниција, но, не и имплементација. Во овој случај, тоа би бил методот којшто треба да ја врати провизијата за дадената трансакција (`double getProvision()`). Овој метод треба соодветно да биде препокриен во изведените класи во согласност со барањата на задачата. На овој начин, со употреба на полиморфизам, сите објекти инстанцирани од изведените класи на `Transacation` би имале различно однесување, во согласност со имплементацијата на методот `getProvision`.

```
1 import java.util.*;
2
3 class Bank {
4     private String name;
5     private Account[] accounts;
6
7     private double totalTransfers;
8     private double totalProvision;
9
```

```
10      public Bank(String name, Account[] accounts) {
11          this.name = name;
12          this.accounts = Arrays.copyOf(accounts, accounts.length);
13          this.totalTransfers = 0.0;
14          this.totalProvision = 0.0;
15      }
16
17      public boolean makeTransaction(Transaction transaction) {
18          Optional<Account> from = findAccount(transaction.getFromId());
19          Optional<Account> to = findAccount(transaction.getToId());
20          if (from.isPresent() && to.isPresent()) {
21              Account fromAccount = from.get();
22              Account toAccount = to.get();
23              double fromBalance = fromAccount.getBalance();
24              double toBalance = toAccount.getBalance();
25              double amount = transaction.getAmount();
26              double provision = transaction.getProvision();
27              if (provision + amount <= fromBalance) {
28                  if (fromAccount.getId() == toAccount.getId()) {
29                      fromBalance =
30                              toBalance = (fromBalance - provision);
31                  } else {
32                      fromBalance -= provision + amount;
33                      toBalance += amount;
34                  }
35                  fromAccount.setBalance(fromBalance);
36                  toAccount.setBalance(toBalance);
37                  updateTotals(amount, provision);
38                  return true;
39              }
40          }
41          return false;
42      }
43
44      void updateTotals(double amount, double provision) {
45          this.totalTransfers += amount;
46          this.totalProvision += provision;
47      }
48
49      Optional<Account> findAccount(long id) {
50          return Arrays.stream(accounts)
51                  .filter(each -> each.getId() == id)
52                  .findAny();
53      }
54
55      public static String toString(long amount) {
56          return String.format("%.2f$", amount / 100.);
57      }
58
59      @override
60      public String toString() {
61          StringBuilder result = new StringBuilder();
62          result.append("Name: ");
63          result.append(name);
64          result.append("\n\n");
65          Arrays.stream(accounts)
66                  .forEach(each -> result.append(each.toString()));
67          return result.toString();
68      }
69
70      @override
```

```
71      public boolean equals(Object o) {
72          if (this == o) return true;
73          if (o == null || getClass() != o.getClass()) return false;
74          Bank bank = (Bank) o;
75          return Objects.equals(name, bank.name) &&
76                  Arrays.equals(accounts, bank.accounts) &&
77                  Objects.equals(totalTransfers, bank.totalTransfers) &&
78                  Objects.equals(totalProvision, bank.totalProvision);
79      }
80
81      @override
82      public int hashCode() {
83          return Objects
84                  .hash(name, accounts, totalTransfers, totalProvision);
85      }
86
87      public double totalProvision() {
88          return totalProvision;
89      }
90
91      public double totalTransfers() {
92          return totalTransfers;
93      }
94
95      public Account[] getAccounts() {
96          return accounts;
97      }
98  }
99
100 class Account {
101     private String name;
102     private long id;
103     private double balance;
104
105     public Account(String name, double balance) {
106         this.name = name;
107         this.balance = balance;
108         this.id = new Random().nextLong();
109     }
110
111     public String getName() {
112         return name;
113     }
114
115     public void setName(String name) {
116         this.name = name;
117     }
118
119     public long getId() {
120         return id;
121     }
122
123     public void setId(long id) {
124         this.id = id;
125     }
126
127     public double getBalance() {
128         return balance;
129     }
130
131     public void setBalance(double balance) {
```

```
132         this.balance = balance;
133     }
134 
135     @override
136     public String toString() {
137         return String
138                 .format("Name: %s\nBalance: %s\n", name, balance);
139     }
140 
141     @override
142     public boolean equals(Object o) {
143         if (this == o) return true;
144         if (o == null || getClass() != o.getClass()) return false;
145         Account account = (Account) o;
146         return Objects.equals(id, account.id);
147     }
148 
149     @override
150     public int hashCode() {
151         return Objects.hash(id);
152     }
153 }
154 
155 class FlatAmountProvisionTransaction extends Transaction {
156 
157     private final double flatAmount;
158 
159     public FlatAmountProvisionTransaction(long fromId, long toId,
160                                           double amount,
161                                           double flatAmount) {
162         super(fromId, toId, amount, "FlatAmount");
163         this.flatAmount = flatAmount;
164     }
165 
166     @override
167     public double getProvision() {
168         return flatAmount;
169     }
170 
171     public double getFlatAmount() {
172         return flatAmount;
173     }
174 }
175 
176 class FlatPercentProvisionTransaction extends Transaction {
177     private final int percent;
178 
179     public FlatPercentProvisionTransaction(long fromId, long toId,
180                                            double amount,
181                                            int percent) {
182         super(fromId, toId, amount, "FlatPercent");
183         this.percent = percent;
184     }
185 
186     @override
187     public double getProvision() {
188         long amount = (long) this.amount / 100;
189         return percent * amount;
190     }
191 
192     public int getPercent() {
```

```
193         return percent;
194     }
195 }
196
197
198 abstract class Transaction {
199     final long fromId;
200     final long toId;
201     final double amount;
202     final String description;
203
204     public Transaction(long fromId, long toId, double amount,
205                        String description) {
206         this.fromId = fromId;
207         this.toId = toId;
208         this.amount = amount;
209         this.description = description;
210     }
211
212     public long getFromId() {
213         return fromId;
214     }
215
216     public long getToId() {
217         return toId;
218     }
219
220     public double getAmount() {
221         return amount;
222     }
223
224     public abstract double getProvision();
225
226     @override
227     public boolean equals(Object o) {
228         if (this == o) return true;
229         if (o == null || getClass() != o.getClass()) return false;
230         Transaction that = (Transaction) o;
231         return Objects.equals(fromId, that.fromId) &&
232                 Objects.equals(toId, that.toId) &&
233                 Objects.equals(amount, that.amount) &&
234                 Objects.equals(description, that.description);
235     }
236
237     @override
238     public int hashCode() {
239         return Objects.hash(fromId, toId, amount, description);
240     }
241
242     public String getDescription() {
243         return description;
244     }
245 }
246
247 public class BankTester {
248
249     public static void main(String[] args) {
250         Scanner jin = new Scanner(System.in);
251         String bank_name = jin.nextLine();
252         int num_accounts = jin.nextInt();
253         jin.nextLine();
```

```
254         Account accounts[] = new Account[num_accounts];
255         for (int i = 0; i < num_accounts; ++i)
256             accounts[i] = new Account(jin.nextLine(),
257                     Double.parseDouble(jin.nextLine()));
258         Bank bank = new Bank(bank_name, accounts);
259         while (true) {
260             String line = jin.nextLine();
261             switch (line) {
262                 case "stop":
263                     return;
264                 case "transaction":
265                     String descrption = jin.nextLine();
266                     double amount =
267                             Double.parseDouble(jin.nextLine());
268                     double parameter =
269                             Double.parseDouble(jin.nextLine());
270                     int from_idx = jin.nextInt();
271                     int to_idx = jin.nextInt();
272                     jin.nextLine();
273                     Transaction t =
274                             getTransaction(descrption, from_idx,
275                                     to_idx, amount, parameter, bank);
276                     System.out.println(
277                             "Transaction amount: " + t.getAmount());
278                     System.out.println("Transaction description: " +
279                             t.getDescription());
280                     System.out.println("Transaction successful? " +
281                             bank.makeTransaction(t));
282                     break;
283                 case "print":
284                     System.out.println(bank.toString());
285                     System.out.println("Total provisions: " +
286                             bank.totalProvision());
287                     System.out.println("Total transfers: " +
288                             bank.totalTransfers());
289                     System.out.println();
290                     break;
291             }
292         }
293     }
294
295     private static Transaction getTransaction(String description,
296                                               int from_idx,
297                                               int to_idx,
298                                               double amount, double o,
299                                               Bank bank) {
300         switch (description) {
301             case "FlatAmount":
302                 return new FlatAmountProvisionTransaction(
303                         bank.getAccounts()[from_idx].getId(),
304                         bank.getAccounts()[to_idx].getId(), amount,
305                         o);
306             case "FlatPercent":
307                 return new FlatPercentProvisionTransaction(
308                         bank.getAccounts()[from_idx].getId(),
309                         bank.getAccounts()[to_idx].getId(), amount,
310                         (int) o);
311         }
312         return null;
313     }
314 }
```

Методот `makeTransacation(Transaction t)` во класата `Bank` е најкомплексен за имплементација. Во истиот, потребно е да се извршат неколку валидации пред да се реализира самата трансакција. Дополнително, потребно е да се внимава методот да не стане предолг, а со тоа и тежок за разбирање и тестирање. За таа цел, за првата валидација (дали постојат сметките на испраќачот и примачот) се користи помошен приватен метод `findAccount(long id)`. Доколку се најдат и двете сметки, се проверува дали испраќачот има средства за да го покрие износот на трансакција, но, и провизијата за истата. Доколку тој услов е исполнет, од сметката на испраќачот се одзема износот и провизијата на трансакцијата, а на сметката на примачот се додава износот на трансакцијата. Соодветно се ажурираат и записите на банката со методот `updateTotals(long amount, long provision)`.

Проблем 1.3 Да се имплементира класа `Canvas` на која ќе чуваат различни форми. За секоја форма се чува:

- `id:String`
- `color:Color` (enum дадена)

Притоа сите форми треба да имплементираат два интерфејси:

- `Scalable` - дефиниран со еден метод `void scale(float scaleFactor)` за соодветно зголемување/намалување на формата за дадениот фактор
- `Stackable` - дефиниран со еден метод `float weight()` кој враќа тежина на формата (се пресметува како плоштина на соодветната форма)

Во класата `Canvas` да се имплементираат следните методи:

- `void add(String id, Color color, float radius)` за додавање круг
- `void add(String id, Color color, float width, float height)` за додавање правоаголник
  - При додавањето на нова форма, во листата со форми таа треба да се смести на соодветното место според нејзината тежина. Елементите постојано се подредени според тежината во опаѓачки редослед.
- `void scale(String id, float scaleFactor)` - метод кој ја скалира формата со даденото `id` за соодветниот `scaleFactor`. Притоа ако има потреба, треба да се изврши преместување на соодветните форми, за да се задржи подреденоста на елементите.
  Не смее да се користи сортирање на листата.
- `toString()` - враќа стринг составен од сите фигури во нов ред. За секоја фигура се додава:
  - `C: [id:\%5s] [color:\%10s] [weight:\%10.2f]` ако е круг
  - `R: [id:\%5s] [color:\%10s] [weight:\%10.2f]` ако е правоаголник

■

Во оваа задача треба да постојат два типа на форми (кругови и правоаголници), при што и двете форми ќе бидат скалабилни и ќе може да се измери нивната тежина. Скалабилноста на формите и мерењето на нивната тежина е овозможено преку два интерфејси `Scalable` и `Stackable` што треба да ги имплементираат и двете форми.

Бидејќи круговите и правоаголниците имаат два заеднички податоци (ИД и боја), се дефинира основна класа `Shape` во која се дефинирани овие полиња. Класите `Circle` и `Rectangle` соодветно наследуваат од класата `Shape`.

Бидејќи сите типови на форми треба да ги имплементираат интерфејсите `Scalable` и `Stackable`, нивната имплементација се декларира директно во класата `Shape`. Во

оваа класа, методите нема да се имплементираат, па со тоа класата станува апстрактна класа.

Класите `Rectangle` и `Circle` наследуваат од `Shape` и ги имплементираат методите дефинирани во рамки на интерфејсите. Во методот `scale(float scaleFactor)` се множат сите димензии на формата со `scaleFactor` , додека пак во методот `weight()` се пресметува плоштината на формата според соодветната математичка формула.

Во класата `Canvas` се чува листа од форми (објекти инстанцирани од изведените класи на класата `Shape`) кои се референцирани од референци од класата `Shape` .

Во ова задача е претставен и концептот на преопотоварување (анг. overloading) на методи, преку имплементацијата на методите со име `add` и нивниот различен потпис.

```
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

enum Color {
    RED, GREEN, BLUE
}

abstract class Shape implements Scaleble, Stackable {
    protected String id;
    protected Color color;

    public Shape(String id, Color color) {
        this.id = id;
        this.color = color;
    }
}

interface Stackable {
    public float weight();
}

interface Scaleble {
    public void scale(float scaleFactor);
}

class Circle extends Shape {
    float r;

    public Circle(String id, Color color, float r) {
        super(id, color);
        this.r = r;
    }

    @Override
    public void scale(float scaleFactor) {
        r *= scaleFactor;
    }

    @Override
    public float weight() {
        return (float) (Math.PI * r * r);
    }

    @Override
    public String toString() {
        return String
                .format("C: %-5s%-10s%10.2f\n", id, color, weight());
```

```
49      }
50
51  }
52
53  class Rectangle extends Shape {
54      float width, height;
55
56      public Rectangle(String id, Color color, float width,
57                       float height) {
58          super(id, color);
59          this.width = width;
60          this.height = height;
61      }
62
63      @Override
64      public void scale(float scaleFactor) {
65          width *= scaleFactor;
66          height *= scaleFactor;
67      }
68
69      @Override
70      public float weight() {
71          return width * height;
72      }
73
74      @Override
75      public String toString() {
76          return String
77                  .format("R: %-5s%-10s%10.2f\n", id, color, weight());
78      }
79
80  }
81
82  class Canvas {
83      List<Shape> shapes;
84
85      public Canvas() {
86          shapes = new ArrayList<Shape>();
87      }
88
89      int find(float weight) {
90          for (int i = 0; i < shapes.size(); ++i) {
91              if (shapes.get(i).weight() < weight) {
92                  return i;
93              }
94          }
95          return shapes.size();
96      }
97
98      public void add(String id, Color color, float radius) {
99          Circle c = new Circle(id, color, radius);
100         int index = find(c.weight());
101         this.shapes.add(index, c);
102     }
103
104     public void add(String id, Color color, float width,
105                     float height) {
106         Rectangle rect = new Rectangle(id, color, width, height);
107         int index = find(rect.weight());
108         this.shapes.add(index, rect);
109     }
```

```
110
111     public void scale(String id, float scaleFactor) {
112         Shape s = null;
113         for (int i = shapes.size() - 1; i >= 0; i--) {
114             if (shapes.get(i).id.equals(id)) {
115                 s = shapes.get(i);
116                 shapes.remove(i);
117                 break;
118             }
119         }
120         s.scale(scaleFactor);
121         int index = find(s.weight());
122         shapes.add(index, s);
123     }
124
125     @Override
126     public String toString() {
127         StringBuilder sb = new StringBuilder();
128         for (Shape shape : shapes) {
129             sb.append(shape);
130         }
131         return sb.toString();
132     }
133 }
134
135 public class ShapesTest {
136     public static void main(String[] args) {
137         Scanner scanner = new Scanner(System.in);
138         Canvas canvas = new Canvas();
139         while (scanner.hasNextLine()) {
140             String line = scanner.nextLine();
141             String[] parts = line.split(" ");
142             int type = Integer.parseInt(parts[0]);
143             String id = parts[1];
144             if (type == 1) {
145                 Color color = Color.valueOf(parts[2]);
146                 float radius = Float.parseFloat(parts[3]);
147                 canvas.add(id, color, radius);
148             } else if (type == 2) {
149                 Color color = Color.valueOf(parts[2]);
150                 float width = Float.parseFloat(parts[3]);
151                 float height = Float.parseFloat(parts[4]);
152                 canvas.add(id, color, width, height);
153             } else if (type == 3) {
154                 float scaleFactor = Float.parseFloat(parts[2]);
155                 System.out.println("ORIGNAL:");
156                 System.out.print(canvas);
157                 canvas.scale(id, scaleFactor);
158                 System.out.printf("AFTER SCALING: %s %.2f\n", id,
159                         scaleFactor);
160                 System.out.print(canvas);
161             }
162         }
163     }
164 }
```

2. Исклучоци

Вообичаено, при решавање на одредени проблеми потребно е да се изградат две одделни стратегии. Првата стратегија се однесува на посакуваниот тек на настани, кога решавањето на проблемот се одвива во согласност со направениот план и реализацијата на парцијалните активности е според предвидените очекувања. Втората стратегија се однесува на предвидување и решавање на ситуациите кои не се во согласност со посакуваниот тек на настани. Најчесто, оваа стратегија се однесува на справување со исклучителните ситуации во кои можеме да се најдеме при решавање на проблемите. Овој пристап, целосно може да се преслика при решавање на софтверските задачи. Повеќе програмски јазици нудат директна поддршка за справување со т.н. исклучителни ситуации во коишто може да се најде нашата програма. Тоа се прави со помош на механизамот исклучок. Исклучоците се настани што се случуваат за време на извршувањето на програмите и истите го менуваат нејзиниот нормален тек (на пр. делење со нула, пристап до низа од елементи надвор од нејзините граници, итн.). Во Јава, исклучок е објект што енкапсулира настан за грешка што се случил во методот и содржи:

- информации за грешката, вклучувајќи го и неговиот тип
- состојбата на програмата пред појавата на грешката
- други дополнителни информации

Исклучокот може да биде фрлен во даден метод и истиот да биде фатен, се со цел правилно и навремено да се справиме со него. Исклучоците се користат за опишување на многу различни типови на состојби на грешка.

Исклучоците го одделуваат кодот за справување со настанатите грешки од регуларниот код. Користењето на исклучоците обезбедува почист и почитлив код. Со помош на исклучоците, грешките се пропагираат, односно вгнездените методи не мора експлицитно да се справуваат со грешките што доведува до помалку работа и посигурен код. Дефинирањето на класи за исклучоци овозможува групирање на типовите на грешки според нивната родител-класа и/или нивна идентификација според нивната вистинска класа. Со други зборови, исклучоците го стандардизираат справувањето со

грешки во рамките на програмите.

Во програмскиот јазик Јава, исклучок е објект којшто е секогаш инстанца на класа изведена од `Throwable` . Јава дефинира неколку типови на исклучоци во рамки на постоечките библиотеки. Исто така, Јава им овозможува на корисниците да дефинираат сопствени типови на исклучоци, ако вградените исклучоци не одговараат на нивните потреби.

2.1 Вградени исклучоци

Вградените исклучоци се достапни во стандардните библиотеки на Јава. Овие исклучоци се погодни за објаснување на одредени ситуации во кои се генерираат грешки. Во продолжение, даден е списокот на важни вградени исклучоци во Јава.

- `ArithmeticException` - се фрла при појава на исклучителна состојба во аритметичка операција.
- `ArrayIndexOutOfBoundsException` - се фрла за да се покаже дека пристапот до елемент од низата се прави со невалиден индекс. Индексот е негативен, поголем од или еднаков на големината на низата.
- `ClassNotFoundException` - овој исклучок се фрла кога се обидуваме да пристапиме до класа чија дефиниција не е пронајдена
- `FileNotFoundException` - овој исклучок се фрла кога датотеката не е достапна или истата не се отвора.
- `IOException` - се фрла кога операцијата за влез-излез нема да успее или истата ќе биде прекината.
- `InterruptedException` - се фрла кога конкретна нишка (анг. thread) чека, спие или процесира, и истата е прекината.
- `NoSuchFieldException` - се фрла кога дадена класа не го содржи специфицираното поле (или променлива).
- `NoSuchMethodException` - се фрла при обид за пристап до метод што не постои.
- `NullPointerException` - овој исклучок се фрла при пристап до празен (анг. null) објект.
- `NumberFormatException` - овој исклучок се фрла кога методот не може да конвертира низа од знаци во нумерички формат.
- `RuntimeException` - претставува каков било исклучок што се случува за време на извршување на програмата.

Проблем 2.1 Да се напише пример: програма за користење на вградената имплементација на исклучокот `ArrayIndexOutOfBoundException` ■

```
1  class ArrayIndexOutOfBoundTester
2  {
3    public static void main(String args[])
4    {
5      try{
6        int a[] = new int[10];
7        a[15] = 1;
8      }
9      catch(ArrayIndexOutOfBoundsException e){
10       System.out.println ("Array Index is Out Of Bounds");
11     }
12   }
13 }
```

Проблем 2.2 Да се напише пример програма за користење на вградената имплементација на исклучокот `FileNotFoundException` ■

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
class File_notFound_Demo {

  public static void main(String args[]) {
    try {
      // Following file does not exist
      File file = new File("E://file.txt");
      FileReader fr = new FileReader(file);
    } catch (FileNotFoundException e) {
        System.out.println("File does not exist");
    }
  }
}
```

2.2 Кориснички дефинирани исклучоци

Кодот што се извршува може да наиде на проблем што не може да се опише од стандардните (вградени) класи на исклучоци. Во овој случај, потребно е корисникот да дефинира свој тип на исклучок преку наследување на класата `Exception` или која било друга класа, којашто директно или индиректно наследува од класата `Exception`. Вообичаено, се имплементира стандардниот конструктор и преотоварен конструктор со сите детали за проблемот што корисникот сака да ги пренесе.

Проблем 2.3 Треба да се развие систем за електронска нарачка од пицерија. Менито на пицеријата се состои од следново:

- Pizza:
  - Standard: 10$
  - Pepperoni: 12$
  - Vegetarian: 8$
- Extra
  - Ketchup 3$
  - Coke 5$

За да се претстави менито, секоја ставка треба да имплементира интерфејс `Item` кој опишува една ставка од менито и ги дефинира следниве методи:

- `int getPrice()` - ја дава цената за конкретната ставка

Следно, потребно е да се дефинираат две класи `ExtraItem` и `PizzaItem` за да може да правите разлика меѓу пици и останатите работи во нарачката. И двете класи треба да имаат еден конструктор кој прима еден `String` аргумент.

- `ExtraItem(String type)` - валидни вредности за `type` се `Coke,` и `Ketchup`
- `PizzaItem(String type)` - валидни вредности за `type` се `Standard`, `Pepperoni` и `Vegetarian`

Ако за `type` се проследи некоја невалидна вредност (која ја нема на менито) треба да се фрли исклучок `InvalidExtraTypeException`, односно

`InvalidPizzaTypeException`.

Последно имплементирајте ја класата `Order`. Таа треба да ги нуди следните функционалности:

- `Order()` - креира нова празна нарачка
- `addItem(Item item, int count)` - соодветната ставка се додава во нарачката (`count` означува колку примероци сакаме од дадената ставка). Ако `count` е поголем од 10 се фрла исклучок `ItemOutOfStockException(item)`. Доколку во нарачката веќе ја има соодветната ставка `Item`, тогаш истата се заменува со нова. Следниот код резултира со нарачка со една стандардна пица:

```
Order order = new Order();
order.addItem(new PizzaItem("Standard"), 2);
order.addItem(new PizzaItem("Standard"), 1);
```

- `getPrice():int` - ја враќа вкупната цена на нарачката
- `displayOrder()` - ја печати содржината на нарачката со соодветни редни броеви пред секоја ставка, името, количината и збирна сума на ставката, како и вкупна сума за целата нарачка. За редниот број се резервирани 3(три) места порамнети надесно, за имињата на ставките се резервирани 15(петнаесет) места со порам-нување налево, за кардиналноста две места порамнети надесно и за цената на една
ставка 5(пет) места порамнети надесно. За "Total:" се резервирани 22(дваесет и две) места со
порамнување налево и за вкупната цена 5(пет) места порамнети надесно. Пример:

```
  1.Standard       x 2   20$
  2.Vegetarian     x 1    8$
  3.Coke           x 3   15$
Total:                   43$
```

Редоследот по кој се печатат ставките е оној по кој тие се внесувани во нарачката. Доколку некоја ставка се внесе повторно, нејзиното место не се менува.

- `removeItem(int idx)` - се отстранува нарачката со даден индекс (сите нарачки со поголеми индекси се поместуваат налево). Доколку не постои нарачка со таков индекс треба да се фрли исклучок `ArrayIndexOutOfBoundsException(idx)`
- `lock()` - ја заклучува нарачката. За да може нарачката да се заклучи треба истата да има барем една ставка, во спротивно фрлете исклучок `EmptyOrderException`.

Откако ќе се заклучи нарачката треба веќе да не може да се менува со методите `removeItem`, `addItem`. Повикот на овие методи резултира со исклучок од типот `OrderLockedException`.

■

При решавање на задачи во коишто има можност да се случи неочекуван тек на настани, потребно е да се употреби механизмот за фрлање и справување на исклучоци. При решавање на вакви задачи важно е:

1. Да се идентификува каде во кодот би можел да се појави неочекуваниот настан, па соодветно таму да се креира и фрли исклучок.
2. Да се опфати повикот кон потенцијално ризичниот код (метод, конструктор) со `try/catch` блок за да се дефинира однесување при исклучок. Треба да се има во предвид дека интерпретерите на повеќето околини за развој, сами даваат сугестија во кој дел од кодот треба да се додаде овој блок.

```
import java.util.ArrayList;
import java.util.Optional;
import java.util.Scanner;
import java.util.stream.IntStream;

interface Item {
    boolean isPizza();

    int getPrice();

    String getType();
}

enum ExtraType {
    Coke(0), OrangeJuice(1), Ketchup(2);

    private int value;
    private int cost;

    ExtraType(int value) {
        this.value = value;
        if (value == 0) cost = 5;
        else if (value == 1) cost = 8;
        else if (value == 2) cost = 3;
    }

    public int getCost() {
        return cost;
    }

    public int getValue() {
        return value;
    }
}

enum PizzaType {
    Standard(0), Pepperoni(1), Extra_cheese(2), Vegetarian(3);
    private int value;
    private int cost;

    PizzaType(int value) {
        this.value = value;
        switch (value) {
            case 0:
                cost = 10;
                break;
            case 1:
                cost = 12;
                break;
            case 2:
                cost = 15;
                break;
            case 3:
```

```
54                  cost = 8;
55                  break;
56          }
57      }
58
59      public int getCost() {
60          return cost;
61      }
62
63      public int getValue() {
64          return value;
65      }
66  }
67
68  class EmptyOrder extends Exception {
69      public EmptyOrder() {
70          this("The order must not be empty");
71      }
72
73      public EmptyOrder(String message) {
74          super(message);
75      }
76  }
77
78
79  class ExtraItem implements Item {
80      private ExtraType type;
81
82      public ExtraItem(String type) throws InvalidExtraTypeException {
83          try {
84              this.type = ExtraType.valueOf(type);
85          } catch (IllegalArgumentException e) {
86              throw new InvalidExtraTypeException(type);
87          }
88      }
89
90      public ExtraItem(ExtraType type) {
91          this.type = type;
92      }
93
94      public int getPrice() {
95          return type.getCost();
96      }
97
98      public boolean isPizza() {
99          return false;
100     }
101
102     @Override
103     public String getType() {
104         return type.name();
105     }
106 }
107
108 class PizzaItem implements Item {
109     private PizzaType type;
110
111     public PizzaItem(String type) throws InvalidPizzaTypeException {
112         try {
113             this.type = PizzaType.valueOf(type);
114         } catch (IllegalArgumentException e) {
```

```
115             throw new InvalidPizzaTypeException();
116         }
117     }
118
119     @Override
120     public int getPrice() {
121         return type.getCost();
122     }
123
124     public boolean isPizza() {
125         return true;
126     }
127
128     @Override
129     public String getType() {
130         return type.name();
131     }
132 }
133
134 class InvalidExtraTypeException extends Exception {
135     public InvalidExtraTypeException(String type) {
136         super(type);
137     }
138
139     public InvalidExtraTypeException() {
140         this("Nepoznat tip");
141     }
142 }
143
144 class ItemOutOfStockException extends Exception {
145     private Item item;
146
147     public ItemOutOfStockException(Item item) {
148         this.item = item;
149     }
150
151     public ItemOutOfStockException() {
152         this("Unknown item is out of stock");
153     }
154
155     public ItemOutOfStockException(String message) {
156         super(message);
157     }
158
159     public Item getItem() {
160         return item;
161     }
162 }
163
164 class Order {
165     private class OrderItem {
166         private final Item item;
167         private int count;
168
169         public Item getItem() {
170             return item;
171         }
172
173         public int getCount() {
174             return count;
175         }
```

```
176
177         public OrderItem(Item item, int count) {
178             this.item = item;
179             this.count = count;
180         }
181
182         public void setCount(int count) {
183             this.count = count;
184         }
185
186         public int getPrice() {
187             return getItem().getPrice() * getCount();
188         }
189
190     }
191
192     private ArrayList<OrderItem> items;
193
194     private boolean locked;
195
196     public Order() {
197         items = new ArrayList<>();
198         locked = false;
199     }
200
201     public void addItem(Item item, int count)
202             throws OrderLockedException, ItemOutOfStockException {
203         if (locked) {
204             throw new OrderLockedException();
205         }
206         if (count > 10) {
207             throw new ItemOutOfStockException(item);
208         }
209         Optional<OrderItem> orderItem = items.stream()
210                 .filter(each -> each.getItem().getType()
211                         .equals(item.getType()))
212                 .findFirst();
213         if (orderItem.isPresent()) {
214             orderItem.ifPresent(oi -> oi.setCount(count));
215             return;
216         }
217         items.add(new OrderItem(item, count));
218     }
219
220     public void removeItem(int index) throws OrderLockedException {
221         if (locked) {
222             throw new OrderLockedException();
223         }
224         items.remove(index);
225     }
226
227     public int getPrice() {
228         return items.stream()
229                 .mapToInt(OrderItem::getPrice)
230                 .sum();
231     }
232
233     public void displayOrder() {
234         IntStream.range(0, items.size())
235                 .forEach(i -> {
236                     OrderItem order = items.get(i);
```

```
                    System.out.printf("%3d.%-15sx%2d%5d$\n", i + 1,
                            order.getItem().getType(),
                            order.getCount(), order.getPrice());
                });
        System.out.printf("%-22s%5d$\n", "Total:", getPrice());
    }

    public void lock() throws EmptyOrder {
        if (items.size() == 0) {
            throw new EmptyOrder();
        }
        locked = true;
    }
}

class InvalidPizzaTypeException extends Exception {
    public InvalidPizzaTypeException() {
    }
}

class OrderLockedException extends Exception {
    private static final long serialVersionUID = 1L;

    public OrderLockedException() {
    }
}

public class PizzaOrderTest {

    public static void main(String[] args) {
        Scanner jin = new Scanner(System.in);
        int k = jin.nextInt();
        if (k == 0) { // test order with removing
            Order order = new Order();
            while (true) {
                try {
                    String type = jin.next();
                    String name = jin.next();
                    Item item = null;
                    if (type.equals("Pizza")) item = new PizzaItem(name);
                    else item = new ExtraItem(name);
                    if (!jin.hasNextInt()) break;
                    order.addItem(item, jin.nextInt());
                } catch (Exception e) {
                    System.out.println(e.getClass().getSimpleName());
                }
            }
            jin.next();
            System.out.println(order.getPrice());
            order.displayOrder();
            while (jin.hasNextInt()) {
                try {
                    int idx = jin.nextInt();
                    order.removeItem(idx);
                } catch (Exception e) {
                    System.out.println(e.getClass().getSimpleName());
                }
            }
            System.out.println(order.getPrice());
            order.displayOrder();
        }
```

```
298         if (k == 1) { //test locking & exceptions
299             Order order = new Order();
300             try {
301                 order.lock();
302             } catch (Exception e) {
303                 System.out.println(e.getClass().getSimpleName());
304             }
305             try {
306                 order.addItem(new ExtraItem("Coke"), 1);
307             } catch (Exception e) {
308                 System.out.println(e.getClass().getSimpleName());
309             }
310             try {
311                 order.lock();
312             } catch (Exception e) {
313                 System.out.println(e.getClass().getSimpleName());
314             }
315             try {
316                 order.removeItem(0);
317             } catch (Exception e) {
318                 System.out.println(e.getClass().getSimpleName());
319             }
320         }
321     }
322
323 }
```

Во оваа задача можат да се идентификуваат неколку ситуации во кои може да се наруши очекуваниот тек на извршување на настаните:

1. При креирање на објекти од тип `ExtraItem` или `PizzaItem`, во конструкторите на овие две класи, текстот што се добива како аргумент потребно е да се конвертира во објект од соодветната енумерација. Овој дел од кодот е опфатен со `try/catch` блок. Доколку успешно се креира објектот од тип `enum`, ќе се креира и објект од тип `ExtraItem/PizzaItem`. Во спротивно, методот `valueOf` од соодветните енумерации ќе фрли исклучок од тип `InvalidArgumentException` и програмата ќе продолжи во `catch` делот дефиниран за овој тип на исклучок. Во `catch` блокот, дополнително ќе биде фрлен кориснички дефинираниот исклучок од тип `InvalidPizzaTypeException`, односно `InvalidExtraTypeException`. Ова ќе направи прекин на креирањето на објектите во соодветните конструктори, односно, ќе спречи креирање на невалиден објект. За овој исклучок треба да се обезбеди механизам за справување со исклучоци (`try/catch` блок), каде што има опасност исклучокот да биде фрлен. А, тоа е насекаде каде што има повици до конструкторите на класите `ExtraItem` и `PizzaItem`. Вообичаено, не е добра практика конструкторите да фрлаат исклучок. Валидацијата на вредностите на аргументите вообичаено се прави претходно во соодветни валидатори, но заради поедноставување на решението, истото е направено во рамки на конструкторот
2. Во методот `addItem` се фрла исклучок од тип `ItemOutOfStockException` при обид да се додаде ставка со количина поголема од 10. Соодветно, се обезбедува механизам за справување со исклучокот (`try/catch` блок) секаде каде се повикува методот `addItem`.
3. Во класата нарачка се чува `bool` променлива `locked` која иницијално е поставена на `false` (нарачката е отворена), но со повик на методот `lock()` може да се заклучи нарачката, односно променливата `locked` да се постави на `true`.

Во методите `addItem` и `removeItem` потребно е да се фрли исклучок од тип `OrderLockedException` доколку нарачката е веќе заклучена. За справување со овој исклучок потребно е повиците на методите `addItem` и `removeItem` да бидат опфатени `try/catch` блок.

4. Во методот `lock()` треба да се направи проверка дали нарачката има барем една ставка. Во случај кога листата со ставки е празна треба да се фрли исклучок од тип `EmptyOrderException` .
5. При обид за бришење на нарачка на позиција што е надвор од граници на низата. Во овој случај постојат две опции:
   - Доколку структурата за чување на податоци за нарачки е обична низа, во кодот на методот `removeItem` треба да се направи проверка дали индексот `idx` е во граници од 0(нула) до големината на низата со ставки. Доколку ова не е исполнето, да се фрли исклучок од тип `ArrayIndexOutOfBounds(idx)` .
   - Доколку се користи класата `ArrayList` за чување на нарачки, нејзините методи за пристап до елементите преку нивната позиција во рамки на листата директно фрлаат исклучок од тип `ArrayIndexOutOfBounds` (овој исклучок е веќе дефиниран во пакетот `java.lang`). Справувањето со овој исклучок е единствен за двата пристапи, односно, употреба на `try/catch` блок насекаде каде се повикува методот `removeItem` .

3. Читање и запишување на податоци

Во оваа глава ќе бидат презентирани класите кои се поставени во `java.util` пакетот и се наменети за читање од влезен поток од податоци и запишување на излезен поток од податоци. Овие класи најчесто се употребуваат за читање од тастатура или влезна датотека, односно запишување на екран или во излезна датотека. Во заедницата се познати и како класи за влез/излез (анг. I/O). Во задачите се покажува како правилно да се користат класите наменети за читање од влезен поток на податоци и како правилно да се форматираат и постават податоци на излезен поток.

3.1 Користење на класата `Scanner` за читање на податоци

Класата `Scanner` најчесто се користи за читање на податоци од тастатура и текстуална датотека. Едноставна замена на аргументот `System.in` (во конструкторот на класата `Scanner`) со соодветен објект за репрезентација на поток на податоци којшто е поврзан со текстуалната датотека овозможува читање на текстуалната датотека.

```
Scanner StreamObject = new Scanner(new FileInputStream(FileName));
```

Методите на `Scanner` класата овозможуваат читање од кој било влезен поток на податоци (во случајот читање од тастатура и текстуална датотека). Класата `Scanner` го дели влезот од податоци на токени користејќи сепаратор (празно место како предефинирана вредност). Добиените токени можат да се конвертираат во вредности од различни типови со користење на неговите интерфејсни методи.

Проблем 3.1 Да се имплементира класа `Subtitles` која ќе чита од влезен тек (стандарден влез, датотека, ...) превод, во стандарден `srt` формат. Секој еден елемент од преводот се состои од реден број, време на почеток на прикажување, време на крај на прикажување и текст кој е во следниот формат (пример):

```
2
```

```
00:00:48,321 --> 00:00:50,837
Let's see a real bet.
```

Делот со текстот може да има повеќе редови. Сите елементи се разделени со еден нов ред.

Ваша задача е да ги имплементирате методите:

- `Subtitles()` - предефиниран конструктор
- `int loadSubtitles(InputStream inputStream)` - метод за читање на преводот (враќа резултат колку елементи се прочитани)
- `void print()` - го печати прочитаниот превод во истиот формат како и при читањето.
- `void shift(int ms)` - ги поместува времињата на сите елементи од преводот за бројот на милисекунди кои се проследува како аргумент (може да биде негативен, со што се поместуваат времињата наназад).

■

При решавање на задачи во кои е потребно да се прочитаат податоци од влезен поток од податоци, многу е важно прво да се согледа форматот на запишаните влезни податоци. Тој може да биде структуриран и неструктуриран. Стратегијата и начинот на читање на тие податоци можат да бидат различни:

- да се чита влезниот поток на податоци ред по ред
- да се чита знак по знак (бајт по бајт)
- влезниот поток на податоци директно да се токенизира доколку е познат форматот на запишување
- да се обезбеди директна конверзија на податоците доколку однапред ја знаеме структурата на запис (пример: читање на вредности на матрица, добро позната податочна структура, итн.).

Откако ќе се согледа форматот на запишаните податоци, може да се донесе одлука со кои класи и методи од тие класи ќе бидат читани податоците.

```
1  import java.io.InputStream;
2  import java.util.ArrayList;
3  import java.util.List;
4  import java.util.Scanner;
5
6  class Subtitles {
7      List<Element> elements;
8
9      public Subtitles() {
10         elements = new ArrayList<Element>();
11     }
12
13     public int loadSubtitles(InputStream inputStream) {
14         Scanner scanner = new Scanner(inputStream);
15         while (scanner.hasNextLine()) {
16             String line = scanner.nextLine();
17             int number = Integer.parseInt(line);
18             String time = scanner.nextLine();
19             StringBuilder text = new StringBuilder();
20             while (true) {
21                 if (!scanner.hasNextLine()) break;
22                 line = scanner.nextLine();
23                 if (line.trim().length() == 0)
24                     break;
25                 text.append(line);
26                 text.append("\n");
```

```
27              }
28              Element element =
29                      new Element(number, time, text.toString());
30              elements.add(element);
31          }
32          return elements.size();
33      }
34
35      public void shift(int ms) {
36          for (Element e : elements) {
37              e.shift(ms);
38          }
39      }
40
41      public void find(String text) {
42          for (Element e : elements) {
43              if (e.findText(text)) {
44                  System.out.println(e.number);
45              }
46          }
47      }
48
49      public void print() {
50          for (Element e : elements) {
51              System.out.println(e);
52          }
53      }
54  }
55
56  class Element {
57      public int timeFrom;
58      public int timeTo;
59      public String text;
60      public int number;
61
62      public Element(int number, String time, String text) {
63          this.number = number;
64          String[] parts = time.split("-->");
65          timeFrom = stringToTime(parts[0].trim());
66          timeTo = stringToTime(parts[1].trim());
67          this.text = text;
68      }
69
70      public void shift(int ms) {
71          timeFrom += ms;
72          timeTo += ms;
73      }
74
75      public boolean findText(String someText) {
76          return text.contains(someText);
77      }
78
79      static int stringToTime(String time) {
80          String[] parts = time.split(",");
81          int res = Integer.parseInt(parts[1]);
82          parts = parts[0].split(":");
83          int sec = Integer.parseInt(parts[2]);
84          int min = Integer.parseInt(parts[1]);
85          int h = Integer.parseInt(parts[0]);
86          res += sec * 1000;
87          res += min * 60 * 1000;
```

```
88          res += h * 60 * 60 * 1000;
89          return res;
90      }
91
92      static String timeToString(int time) {
93          int h = time / (60 * 60 * 1000);
94          time = time % (60 * 60 * 1000);
95          int m = time / (60 * 1000);
96          time = time % (60 * 1000);
97          int s = time / 1000;
98          int ms = time % 1000;
99          return String.format("%02d:%02d:%02d,%03d", h, m, s, ms);
100     }
101
102     @Override
103     public String toString() {
104         return String.format("%d\n%s --> %s\n%s", number,
105                 timeToString(timeFrom), timeToString(timeTo), text);
106     }
107 }
108
109 public class SubtitlesTest {
110     public static void main(String[] args) {
111         Subtitles subtitles = new Subtitles();
112         int n = subtitles.loadSubtitles(System.in);
113         System.out.println("+++++ ORIGINIAL SUBTITLES +++++");
114         subtitles.print();
115         int shift = n * 37;
116         shift = (shift % 2 == 1) ? -shift : shift;
117         System.out.println(String.format("SHIFT FOR %d ms", shift));
118         subtitles.shift(shift);
119         System.out.println("+++++ SHIFTED SUBTITLES +++++");
120         subtitles.print();
121     }
122 }
```

Во оваа задача, лесно може да се согледа дека податоците за еден превод се организирани во најмалку три редови. Притоа, во првиот ред е запишан редниот број, во вториот ред е запишан временскиот опсег во кој треба да се прикаже преводот и во наредните редови е запишан преводот. Кога се чита ред по ред (како во оваа ситуација) може да се искористат класите `Scanner` и `BufferedReader`. Оваа задача е решена со помош на класата `Scanner`.

Прво се креира објект од класата `Scanner` преку предавање на објектот од тип `InputStream` во конструкторот на класата `Scanner`.

Бидејќи влезот треба да се прочита ред по ред, прво потребно е да се напише циклус што ќе се извршува сè додека има редови во датотеката за читање. За проверка на условот дали сме стигнале до крајот на датотеката се користи методот `scanner.hasNextLine()`. Во рамки на циклусот, прво се чита еден ред и истиот се парсира во цел број (бидејќи тој ред го означува редниот број на преводот). Потоа се чита уште еден ред и истиот се зачувува како текстуална низа, што подоцна се трансформира во два временски интервали. Следно, со помош на вгнезден циклус се читаат редовите во кои е запишан преводот, сè додека не дојдеме до празен ред (преводите се одделени со празен ред). Секој од прочитаните редови се додава во `StringBuilder` со цел да се избегне постојано креирање на нови текстуални низи. Како краен резултат од извршувањето на вгнездениот циклус е читањето на целосниот превод во една текстуална низа што се добива со повикувањето на методот `toString()`

од `StringBuilder` објектот. За читање од влезниот поток од податоци ред по ред, се користи методот `nextLine()` од класата `Scanner`.

Дополнително, во задачата потребно е да се обработи и прочитаниот запис што се однесува на почетното и крајното време на преводот. За таа цел, во рамки на конструкторот на класата `Element` потребно е редот во кој е запишано почетното и крајното време на преводот да се раздели на две текстуални низи со сепаратор „стрелки“. Добиените две текстуални низи се конвертираат во милисекунди поминати од почетокот на филмот.

3.1.1 Користење на класата `PrintWriter` за запишување на податоци

Класата `PrintWriter` се користи за форматирано печатење на податоци во излезен поток.

Проблем 3.2 Да се имплементира класа `CheckIn` која ќе чита од влезен тек (стандарден влез, датотека, ...) податоци за пријавување/одјавување на корисник. Податоците содржат (Име - единствено), почетен и краен датум (dd.mm.YYYY) со време (во 24-часовен формат) на пријавување и одјавување. Сите податоци се разделени со едно празно место. Датумот и времето се разделени со `-`, во датумот денот, месецот и годината може да се разделени со `.` или `/`, а во самото време часот и минутите може да бидат разделени со `:` или `.`. Пример за форматот на податоците:

```
John 03.06.2015-11:15 03/06/2015-20.45
```

За потребите на класата потребно е да се имплементираат следните методи:

- `CheckIn()` - предефиниран конструктор
- `void readTimes(InputStream inputStream)` - метод за читање на податоците
- `void writeTimes(OutputStream outputStream)` - метод кој ги печати сите времиња на пријавување сортирани во растечки редослед според времето на пријавување на корисникот во растечки редослед. При печатењето името се печати со 15(петнаесет) места порамнето налево, датумот во формат dd.mm.YYYY (само еднаш), па следуваат времињата на пријавување/одјавување и вкупно времетраење на присуство во минути.

Методот за читање `readTimes` фрла исклучок од тип `InvalidCheckInTimes` ако датумот на пријавување и одјавување е различен или пак времето на пријавување е после времето на одјавување. Исклучокот во пораката `getMessage()` треба да го врати влезниот податок којшто го предизвикал исклучокот. Сите пријавувања/одјавувања до моментот кога ќе се фрли некој исклучок треба да си останат прочитани.

■

Слично како и во претходната задача, така и во оваа, читањето на податоци најдобро е да се направи ред по ред во кои се запишани информациите за работните времиња на вработените.

Прво, се креира објект од класата `Scanner`, а потоа со помош на циклус за повторување се чита ред по ред од влезниот поток. Секоја од прочитаните текстуални низи се дели на три низи користејќи го празното место како сепаратор. Во првата текстуална низа е запишано името на вработениот, во втората текстуална низа почетокот на смената, а во третата текстуална низа е запишан крајот на смената. На втората и третата текстуална низа дополнително се прави поделба каде како сепаратор се користи „—“. Ова резултира со добивање на две текстуални низи со должина 2, каде првата текстуална низа е датумот, а втората текстуална низа е времето. Откако ќе се

добијат датумите и времињата за почетокот и крајот на смената, потребно е истите да се форматираат. Бидејќи денот, месецот и годината во датумите може да бидат разделени со „/“ или „.“, а во излезот на програмата се очекува да бидат разделени со „.“, со помош на методот `replace` од класата `String`, секаде каде што има „/“ се заменува со „.“. За парсирање на времето, прво се обидуваме да ја поделиме текстуалната низа со „:“, а доколку тоа не успее (не врати низа од текстуални низи со големина 2), се прави делење на текстуалната низа со „.“.

Во оваа задача, освен читањето од влезен поток, потребно е резултатите да се запишат во излезен поток (објект од класата `outputStream`), со помош на методот `writeTimes`. За таа цел, се креира објект од класата `PrintWriter`, така што на конструкторот на класата се предава објектот од тип `OutputStream`. Податоците за смените (објекти од класата `CheckInTime`) се подредуваат и се печатат со методот `println()` од класа `PrintWriter`.

При работа со `PrintWriter` важно е да се повика методот `flush` со цел креираниот податочен поток во самиот `PrintWriter` објект да се запише на излезниот поток. На крајот, секој отворен поток на податоци треба да се затвори со повик на методот `close`.

```
import java.io.InputStream;
import java.io.OutputStream;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Scanner;

class CheckIn {
    List<CheckInTime> elements;

    public CheckIn() {
        elements = new ArrayList<CheckInTime>();
    }

    public void loadTimes(InputStream inputStream)
            throws InvalidCheckInTimes {
        Scanner scanner = new Scanner(inputStream);
        while (scanner.hasNextLine()) {
            String line = scanner.nextLine();
            String[] parts = line.split(" ");
            String name = parts[0];
            String[] time1 = parts[1].split("-");
            String[] time2 = parts[2].split("-");
            CheckInTime time = new CheckInTime(name,
                    CheckInTime.setDateFormat(time1[0]),
                    CheckInTime.setDateFormat(time2[0]),
                    CheckInTime.strToTime(time1[1]),
                    CheckInTime.strToTime(time2[1]));
            if (!time.valid()) throw new InvalidCheckInTimes(line);
            elements.add(time);
        }
    }

    public void printTimes(OutputStream outputStream) {
        PrintWriter writer = new PrintWriter(outputStream);
        Collections.sort(elements);
        for (CheckInTime checkInTime : elements) {
            writer.println(checkInTime);
```

```
        }
        writer.flush();
        writer.close();
    }


}

class CheckInTime implements Comparable<CheckInTime> {
    private String name;
    private String dateFrom;
    private String dateTo;
    private int timeFrom;
    private int timeTo;

    public CheckInTime(String name, String dateFrom, String dateTo,
                       int timeFrom, int timeTo) {
        this.name = name;
        this.dateFrom = dateFrom;
        this.dateTo = dateTo;
        this.timeFrom = timeFrom;
        this.timeTo = timeTo;
    }

    public boolean valid() {
        return dateFrom.equals(dateTo) && timeFrom <= timeTo;
    }

    public static int strToTime(String time) {
        String[] t = time.split(":");
        if (t.length == 1) {
            t = time.split("\\.");
        }
        return Integer.parseInt(t[0]) * 60 + Integer.parseInt(t[1]);
    }

    public static String setDateFormat(String date) {
        date = date.replace("/", ".");
        return date;
    }

    @Override
    public String toString() {
        return String
                .format("%-15s%s %02d:%02d - %02d:%02d (%d)", name,
                        dateFrom, timeFrom / 60, timeFrom % 60,
                        timeTo / 60, timeTo % 60, timeTo - timeFrom);
    }

    @Override
    public int compareTo(CheckInTime o) {
        return Integer.compare(timeFrom, o.timeFrom);
    }
}

class UnsupportedFormatException extends Exception {
    String data;

    public UnsupportedFormatException(String data) {
        this.data = data;
    }
```

```
101
102     @Override
103     public String getMessage() {
104         return data;
105     }
106 }
107
108 class InvalidCheckInTimes extends Exception {
109     String data;
110
111     public InvalidCheckInTimes(String data) {
112         this.data = data;
113     }
114
115     @Override
116     public String getMessage() {
117         return data;
118     }
119 }
120
121 public class CheckInTest {
122     public static void main(String[] args) {
123         CheckIn checkIns = new CheckIn();
124         try {
125             checkIns.loadTimes(System.in);
126         } catch (InvalidCheckInTimes e2) {
127             System.out.println(
128                     "InvalidCheckInTimes: " + e2.getMessage());
129         }
130         System.out.println(
131                 String.format("====== PRINT CHECK INS ======"));
132         checkIns.printTimes(System.out);
133     }
134 }
```

3.2 Користење на класата `BufferedReader` за читање од влезен поток

Класата `BufferedReader` може да се користи за читање од влезен поток на податоци (вклучително и текстуална датотека). Објект од класата `BufferedReader` ги има методите `read` и `readLine`. За користење на класата `BufferedReader` за читање од текстуална датотека, потребно е да бидат внесени следниве библиотеки:

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.FileNotFoundException;
import java.io.IOException;
```

Класата `BufferedReader` како аргумент во конструкторот користи објект од класата `FileReader`. Поток на податоци од класата `BufferedReader` се креира и поврзува со текстуална датотека на следниов начин:

```
BufferedReader readerObject;
readerObject = new BufferedReader(new FileReader(FileName));
```

По дефинирањето на објектот од класата `BufferedReader`, методите `read` и `readLIne` можат да се користат за читање од датотеката со име `FileName`. Методот `readLine` е ист како и методот за читање од тастатура, со таа разлика што овде се

чита од датотека. Методот `read` чита еден знак и враќа вредност (од тип `int`), кој соодветствува на прочитаниот знак. Поради тоа, најчесто е потребно и дополнително кастирање:

```
char next = (char)(readerObject.read());
```

Дополнително, започнувајќи од Јава 8 постои и методот `lines()` што враќа објект `Stream<String>`, и претставува поток од текстуални низи каде што секоја текстуална низа е еден ред од потокот на податоци.

Проблем 3.3 Да се дефинира класа `CakeShopApplication` во која ќе се чуваат информации за сите тековни нарачки на торти и пити.

За класата да се дефинира:

- `CakeShopApplication()` - конструктор
- `int readCakeOrders(InputStream inputStream)` - метод којшто од влезен поток на податоци ќе прочита информации за повеќе нарачки. Во секој еден ред се наоѓа информација за една нарачка во формат: `orderId item1Name item1Price item2Name item2Price .... itemNName itemNPrice`, каде што `orderId` е ID на нарачката, а по неа следуваат паровите нарачан производ и цена. Методот треба да врати цел број којшто означува колку точно производи се нарачани во рамки на сите нарачки кои се успешно прочитани.
- `void printLongestOrder(OutputStream outputStream)` - метод кој на излезен поток ќе ја испечати нарачката со најголем број на нарачани продукти, во формат: `orderId totalOrderItems`, каде `totalOrderItems` е бројот на сите нарачани продукти.

■

Од барањата на задачата во кои јасно е наведено дека информациите за нарачките се дадени во посебни редови, може да се заклучи дека читањето на влезниот поток треба да биде реализирано ред по ред.

Иако задачата може да се реши и со употреба на `Scanner`, овде ќе биде прикажан начинот на употреба на `BufferedReader` класата за читање од влезен поток на податоци.

Во функцијата `int readCakeOrders(InputStream inputStream)` прво се креира објект од тип `BufferedReader`. Во конструкторот на класата `BufferedReader` се предава објект од тип `InputStreamReader`, додека во конструкторот на `InputStreamReader` се предава објект од тип `InputStream`. Ова е генерално решение што може да функционира за читање од каков било `InputStream` (од класата `InputStream` се изведени стотина потткласи). Потоа, со помош на циклус за повторување се чита ред по ред, сè додека методот `readLine()` од класата `BufferedReader` не врати `null`. За секој прочитан ред се повикува статичкиот метод `createOrder` од класата `Order` кој ги парсира сите информации за нарачката и ставките што се дел од неа.

Релевантните податоци за една нарачка запишани во еден ред од влезниот поток се одделени со празно место, при што на нултата позиција се наоѓа ИД-то на нарачката, а потоа на секоја непарна позиција (1, 3, итн.) се наоѓа името на ставката. На секоја парна позиција (2, 4, итн.) се наоѓа цената на ставката. Соодветно, се креира листа од сите ставки и истата се предава како аргумент во конструкторот на класата `Order`, заедно со ИД-то на нарачката што се наоѓа на нултата позиција.

На крајот од читањето се повикува методот `close`, за да се затвори читањето од влезниот поток на податоци.

Во коментари е дадено решение со употреба на методот `lines()` од класата `BufferedReader`.

```
import java.io.*;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Comparator;
import java.util.List;
import java.util.stream.Collectors;

class Item {
    private String name;
    private int price;

    public Item(String name) {
        this.name = name;
        this.price = 0;
    }

    public void setPrice(int price) {
        this.price = price;
    }
}

class Order implements Comparable<Order> {
    private int id;
    private List<Item> items;

    public Order() {
        this.id = -1;
        items = new ArrayList<>();
    }

    public Order(int id, List<Item> items) {
        this.id = id;
        this.items = items;
    }

    public static Order createOrder(String line) {
        String[] parts = line.split("\\s+");
        int orderId = Integer.parseInt(parts[0]);
        List<Item> items = new ArrayList<>();
        Item item = null;
        for (int i = 1; i < parts.length; i++) {
            if (i % 2 == 1) { //item name is here
                item = new Item(parts[i]);
            } else {
                item.setPrice(Integer.parseInt(parts[i]));
                items.add(item);
            }
        }
        return new Order(orderId, items);
    }

    public List<Item> getItems() {
        return items;
    }

    public int getNumberOfItems() {
        return items.size();
    }

    $@override
    public int compareTo(Order o) {
```

```
62          return Integer.compare(this.items.size(), o.items.size());
63      }
64
65      $@override
66      public String toString() {
67          return this.id + " " + this.items.size();
68      }
69  }
70
71  class CakeShopApplication {
72      private List<Order> orders;
73
74      public CakeShopApplication() {
75          this.orders = new ArrayList<>();
76      }
77
78      public int readCakeOrders(InputStream inputStream)
79              throws IOException {
80          BufferedReader br = new BufferedReader(
81                  new InputStreamReader(inputStream));
82
83          /* JAVA 8 Stream API solution
84          this.orders = br.lines()
85                  .map(Order::createOrder)
86                  .collect(Collectors.toList());
87
88          br.close()
89
90          return this.orders.stream().mapToInt(order ->
91                                      order.getNumberOfItems()).sum();
92          */
93
94          String line;
95          while ((line = br.readLine()) != null) {
96              orders.add(Order.createOrder(line));
97          }
98
99          int sum = 0;
100         for (Order order : orders) {
101             sum += order.getNumberOfItems();
102         }
103
104         br.close();
105         return sum;
106     }
107
108     public void printLongestOrder(OutputStream outputStream) {
109         PrintWriter printWriter = new PrintWriter(outputStream);
110         Order longestOrder = orders.get(0);
111
112         /* JAVA 8 Stream API solution
113         Order longestOrder = this.orders.stream()
114                                     .max(Comparator.naturalOrder())
115                                     .orElseGet(Order::new);
116          */
117
118         for (Order order : orders) {
119             if (order.compareTo(longestOrder) > 0) {
120                 longestOrder = order;
121             }
122         }
```

```
123
124         printWriter.println(longestOrder);
125
126         printWriter.flush();
127         printWriter.close();
128     }
129 }
130
131 public class CakeShopApplicationTest1 {
132
133     public static void main(String[] args) {
134         CakeShopApplication cakeShopApplication =
135                 new CakeShopApplication();
136
137         System.out.println("--- READING FROM INPUT STREAM ---");
138         try {
139             System.out.println(
140                     cakeShopApplication.readCakeOrders(System.in));
141         } catch (IOException e) {
142             e.printStackTrace();
143         }
144
145         System.out.println(
146                 "--- PRINTING LARGEST ORDER TO OUTPUT STREAM ---");
147         cakeShopApplication.printLongestOrder(System.out);
148     }
149 }
```

4. Генерици

Започнувајќи со верзијата 5.0, Јава дозволува дефинирање на класи и методи што вклучуваат параметри за типови. Таквите дефиниции се нарекуваат генерици. Воведувањето на параметарски тип овозможува креирање на класи, методи и интерфејси независни од конкретни податочни типови. Ова значи дека една класа, метод или интерфејс може да се употребува со повеке податочни типови без да се пишува конкретна, специфична имплементација. Иако `Object` е класа предок на сите други класи чија референца може да се однесува на кој било тип на објект, сепак на таков начин не може да се обезбеди безбедност на тип. Со воведувањето на параметарските типови успешно можеме да спречиме појава на грешки настанати како резултат на кастирање при извршување на програмата.

4.1 Генерички класи, интерфејси и методи

4.1.1 Генерички класи

Класата која е дефинирана со параметар како тип се нарекува генеричка класа или параметризирана класа. Параметарскиот тип се дава во аглести загради по името на класата во нејзиното заглавие. Секој не-клучен збор може да биде употребен како идентификатор за параметарски тип. По конвенција параметарот стартува со голема буква. Параметарскиот тип може да се употребува како и секој друг тип во дефиницијата на класата.

Проблем 4.1 Да се имплементира генеричка класа `Triple` (тројка) од нумерички вредности (кои се поттип на класата `Number`). За класата да се имплементираат:

- конструктор со 3(три) аргументи,
- `double max()` - го враќа најголемиот од трите броја
- `double average()` - кој враќа просек на трите броја
- `void sort()` - кој ги сортира елементите во растечки редослед

- да се преоптовари методот `toString()` којшто враќа форматиран стринг со две децимални места за секој елемент и празно место помеѓу нив.

■

Во оваа задача потребно е да се чуваат три променливи на инстанца во класата `Triple` и сите три да се од ист тип. Типот на овие променливи треба да биде поттип на класата `Number`.

Со употреба на генерици, се дефинираат овие три променливи од генерички тип `T`. Додека пак, во дефиницијата на класата се додава `<T extends Number>`, што означува дека T мора да биде тип што наследува од класата `Number`.

Со ова се овозможува класата `Triple` да се користи за тројки од тип `Short`, `Integer`, `Float`, `Double`, `Long` итн.

```
class Triple<T extends Number> {
  T first, second, third;

  public Triple(T first, T second, T third) {
    this.first = first;
    this.second = second;
    this.third = third;
  }

  public double max() {
    double max = first.doubleValue();
    if (second.doubleValue() > max) {
      max = second.doubleValue();
    }
    if (third.doubleValue() > max) {
      max = third.doubleValue();
    }
    return max;
  }

  public double avarage() {
    double sum = first.doubleValue() + second.doubleValue()
        + third.doubleValue();
    return sum / 3;
  }

  public void sort() {
    if (first.doubleValue() > second.doubleValue()) {
      T temp = second;
      second = first;
      first = temp;
    }
    if (second.doubleValue() > third.doubleValue()) {
      T temp = third;
      third = second;
      second = temp;
    }
    if (first.doubleValue() > second.doubleValue()) {
      T temp = second;
      second = first;
      first = temp;
    }
  }

  @Override
  public String toString() {
```

```
47      return String.format("%.2f %.2f %.2f", first.doubleValue(),
48          second.doubleValue(), third.doubleValue());
49    }
50
51    public static void main(String[] args) {
52      Scanner scanner = new Scanner(System.in);
53      int a = scanner.nextInt();
54      int b = scanner.nextInt();
55      int c = scanner.nextInt();
56
57      Triple<Integer> tInt = new Triple<Integer>(a, b, c);
58
59      System.out.printf("%.2f\n", tInt.max());
60      System.out.printf("%.2f\n", tInt.avarage());
61
62      tInt.sort();
63      System.out.println(tInt);
64
65      float fa = scanner.nextFloat();
66      float fb = scanner.nextFloat();
67      float fc = scanner.nextFloat();
68
69      Triple<Float> tFloat = new Triple<Float>(fa, fb, fc);
70
71      System.out.printf("%.2f\n", tFloat.max());
72      System.out.printf("%.2f\n", tFloat.avarage());
73
74      tFloat.sort();
75      System.out.println(tFloat);
76
77      double da = scanner.nextDouble();
78      double db = scanner.nextDouble();
79      double dc = scanner.nextDouble();
80
81      Triple<Double> tDouble = new Triple<Double>(da, db, dc);
82
83      System.out.printf("%.2f\n", tDouble.max());
84      System.out.printf("%.2f\n", tDouble.avarage());
85
86      tDouble.sort();
87      System.out.println(tDouble);
88    }
89  }
```

Проблем 4.2 Да се имплементира класа за генеричка табела `GenericTable` со генерички имиња на редовите кои се споредливи вредности (`Comparable`) и генерички вредности кои мора да се поткласа на `Number` .

Во класата треба да се имплементираат следните методи:

- `void addRow(RowKey key, Value... values)` - за додавање на низа од вредности `values` за редот со даден клуч `key`
- `double max(RowKey key)` - ја враќа вредноста (`double`) на елементот со максимална вредност за даден ред со клуч `key`
- `String toString()` - враќа `String` репрезентација на табелата со формат на секој ред `key: v1 v2 . . . vn` каде што елементите `v` се печатат со 6(шест) места од кои две децимални и се разделени со таб `\t` .

Од барањето специфицирано за методот `addRow(RowKey rowKey, Value ... values)`, треба да се забележи дека за решавање на оваа задача ќе бидат потребни два генерички параметри `RowKey` и `Value`. Дополнително, може да се согледа дека клучевите на редиците треба да бидат споредливи објекти, односно генеричкиот тип `RowKey` треба да наследува (односно имплементира) од `Comparable` (`RowKey extends Comparable`) интерфејсот. Вредностите треба да бидат нумерички типови, односно класите коишто ќе се користат на местото на генеричкиот тип `Value`, треба да се поттип на класата `Number` (`Value extends Number`).

Во согласност со тоа, дефиницијата на генеричката класа `GenericTable` ќе биде `class GenericTable<RowKey extends Comparable, Value extends Number>`. Дефиницијата на оваа класа може да биде и `class GenericTable<RowKey extends Comparable <RowKey>, Value extends Number>`. Практично тоа значи дека наместо да го имплементираме методот `compareTo(Object obj)`, ќе биде потребно да го имплементираме методот `compareTo (RowKey rowKey)`.

Ако не се наметнат ограничувањета на типовите `RowKey` и `Value`, тогаш е невозможно генеричката табела да биде претставена преку `TreeMap`, бидејќи елементите од `RowKey` класата нема да можат да се споредуваат меѓусебно. Исто така, методот `double max (RowKey key)` ќе биде невозможно да се импелементира од истите причини (класата `Number` го имплементира интерфејсот `Comparable<Number>` и објектите од таа класа ќе можат да се споредуваат меѓусебно).

```
import java.util.*;

class Key implements Comparable<Key> {
    int index;
    String name;

    public Key(int index, String name) {
        this.index = index;
        this.name = name;
    }

    @Override
    public int compareTo(Key o) {
        return Integer.compare(index, o.index);
    }

    @Override
    public String toString() {
        return String.format("%d (%s)", index, name);
    }
}

class GenericTable<RowKey extends Comparable, Value extends Number> {
    Map<RowKey, List<Value>> rows;

    public GenericTable() {
        rows = new TreeMap<>();
    }

    public void addRow(RowKey key, Value... values) {
        List<Value> row = rows.get(key);
        if (row == null) {
            row = new ArrayList<>();
        }
```

```
35          row.addAll(Arrays.asList(values));
36          rows.put(key, row);
37      }
38
39      public double max(RowKey key) {
40          List<Value> row = rows.get(key);
41          double max = Double.MIN_VALUE;
42          for (Number number : row) {
43              max = Math.max(max, number.doubleValue());
44          }
45          return max;
46      }
47
48      @Override
49      public String toString() {
50          StringBuilder res = new StringBuilder();
51          for (RowKey key : rows.keySet()) {
52              List<Value> row = rows.get(key);
53              res.append(String.format("%s: ", key));
54              for (Value value : row) {
55                  res.append(String.format("%6.2f\t", value.doubleValue()));
56              }
57              res.append('\n');
58          }
59          return res.toString();
60      }
61
62      public static void main(String[] args) {
63          Scanner scanner = new Scanner(System.in);
64          GenericTable<String, Integer> stringTable = new GenericTable<>();
65          GenericTable<Integer, Double> integerTable = new GenericTable<>();
66          GenericTable<Key, Float> keyTable = new GenericTable<>();
67          int n = scanner.nextInt();
68          scanner.nextLine();
69          for (int i = 0; i < n; ++i) {
70              String[] parts = scanner.nextLine().split("\\s+");
71              Integer[] values = new Integer[parts.length - 1];
72              for (int j = 0; j < values.length; ++j) {
73                  values[j] = Integer.parseInt(parts[j + 1]);
74              }
75              stringTable.addRow(parts[0], values);
76          }
77          System.out.println("=== STRING TABLE ===");
78          System.out.println(stringTable);
79          String k = String.format("row%d", n / 2);
80          System.out.printf("MAX (%s): %.2f\n", k, stringTable.max(k));
81          n = scanner.nextInt();
82          scanner.nextLine();
83          for (int i = 0; i < n; ++i) {
84              String[] parts = scanner.nextLine().split("\\s+");
85              Double[] values = new Double[parts.length - 1];
86              for (int j = 0; j < values.length; ++j) {
87                  values[j] = Double.parseDouble(parts[j + 1]);
88              }
89              integerTable.addRow(Integer.parseInt(parts[0]), values);
90          }
91          System.out.println("=== INTEGER TABLE ===");
92          System.out.println(integerTable);
93          System.out.printf("MAX (%d): %.2f\n", n / 2,
94                            integerTable.max(n / 2));
95          n = scanner.nextInt();
```

```
        scanner.nextLine();
        for (int i = 0; i < n; ++i) {
            String[] parts = scanner.nextLine().split("\\s+");
            Float[] values = new Float[parts.length - 1];
            for (int j = 0; j < values.length; ++j) {
                values[j] = Float.parseFloat(parts[j + 1]);
            }
            String[] keys = parts[0].split(":");
            Key key = new Key(Integer.parseInt(keys[0]), keys[1]);
            keyTable.addRow(key, values);
        }
        System.out.println("=== KEY TABLE ===");
        System.out.println(keyTable);
        Key key = new Key(1, "a");
        System.out.printf("MAX (%s): %.2f\n", key, keyTable.max(key));
        scanner.close();
    }
}
```

Проблем 4.3 Да се имплементира `GenericCounter<T>` за броење на исти елементи. Класата треба да имплементира два методи:

- `void count(T element)` - метод кој прима елемент за броење
- `String toString()` - кој враќа стринг репрезентација на избројените елементи

```
[елемент]|[повторување на знакот *
онолку пати колку што е избројан дадениот елемент]
```

Решението треба да има мемориска комплексност $O(n)$ и комплексност $O(n)$ за броење на елементите (линеарно пребарување) ■

Во оваа задача класата `GenericCounter` се дефинира со генеричкиот параметар `T`. Дополнително, се креира помошна генеричка класа `CountableElement<T>` којашто во себе содржи елемент од тип `T` и бројач на појавување на тој елемент.

При броењето на елементите во методот `void count(T element)`, ако се појави нов елемент, истиот се додава во листата на објекти од тип `CountableElement<T>`, а ако елементот се појавил претходно, бројачот за неговото појавување се зголемува за еден.

```
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

class GenericCounter<T> {
    List<CountableElement<T>> elements;

    GenericCounter() {
        elements = new ArrayList<CountableElement<T>>();
    }

    public void count(T element) {
        for (int i = 0; i < elements.size(); ++i) {
            CountableElement<T> el = elements.get(i);
            if (el.isEqual(element)) {
                el.increment();
                return;
            }
        }
```

```
20          elements.add(new CountableElement<T>(element));
21      }
22
23      @Override
24      public String toString() {
25          StringBuilder sb = new StringBuilder();
26          for (int i = 0; i < elements.size(); ++i) {
27              CountableElement<T> el = elements.get(i);
28              sb.append(String.format("%s|%s\n", el.element.toString(),
29                      countToStar(el.count)));
30          }
31          return sb.toString();
32      }
33
34      static String countToStar(int c) {
35          StringBuilder sb = new StringBuilder();
36          for (int i = 0; i < c; ++i) {
37              sb.append("*");
38          }
39          return sb.toString();
40      }
41  }
42
43  class CountableElement<T> {
44      int count;
45      T element;
46
47      public CountableElement(T element) {
48          this.element = element;
49          count = 1;
50      }
51
52      public void increment() {
53          ++count;
54      }
55
56      public boolean isEqual(T element) {
57          return this.element.equals(element);
58      }
59  }
60
61  public class GenericCounterTest {
62      public static void main(String[] args) {
63          Scanner scanner = new Scanner(System.in);
64          int n = scanner.nextInt();
65          GenericCounter<Integer> counterInt =
66                  new GenericCounter<Integer>();
67          scanner.nextLine();
68          for (int i = 0; i < n; ++i) {
69              int x = scanner.nextInt();
70              counterInt.count(x);
71          }
72          System.out.println("=====INTEGERS=====");
73          System.out.println(counterInt);
74          n = scanner.nextInt();
75          scanner.nextLine();
76          GenericCounter<String> counterString =
77                  new GenericCounter<String>();
78          for (int i = 0; i < n; i++) {
79              String s = scanner.nextLine();
80              counterString.count(s);
```

```
81          }
82          System.out.println("=====STRINGS=====");
83          System.out.println(counterString);
84          n = scanner.nextInt();
85          scanner.nextLine();
86          GenericCounter<Float> counterFloat =
87                  new GenericCounter<Float>();
88          for (int i = 0; i < n; i++) {
89              float f = scanner.nextFloat();
90              counterFloat.count(f);
91          }
92          System.out.println("=====FLOATS=====");
93          System.out.println(counterFloat);
94          scanner.close();
95      }
96  }
```

Проблем 4.4 Да се развие генеричка класа `GenericFraction` за работа со дропки. Класата има два генерички параметри `T` и `U` коишто мора да бидат од некоја класа која наследува од класата `Number`. `GenericFraction` има две променливи:

- numerator - броител
- denominator - именител.

Во класата да се имплементираат следните методи:

- `GenericFraction(T numerator, U denominator)` - конструктор кој ги иницијализира броителот и именителот на дропката. Ако се обидиме да поставиме дропка со вредност 0 за именителот, треба да се фрли исклучок од тип `ZeroDenominatorException`
- `GenericFraction<Double, Double> add(GenericFraction<? extends Number,? extends Number> gf)` - собирање на две дропки
- `double toDouble()` - враќа вредност на дропката како реален број
- `String toString()` - ја печати дропката во следниот формат `[numerator] / [denominator]`, скратена (нормализирана) и секој со две децимални места.

Во согласност со барањето дефинирано во задачата, генеричката класа за дропка потребно е да има два генерички параметри кои наследуваат од класата `Number`. Барањето е дефинирано на овој начин, со цел броителот и именителот на дропката да можат да бидат претставени со различна прецизност (на пр. броителот да биде објект од тип `Short`, а именителот да биде објект од тип `Double`).

Иако броителот и именителот можат да бидат репрезентирани со различна прецизност, сепак наследувањето од класата `Number` гарантира конверзија на нивната вредност во `Double` преку имплементацијата методот `doubleValue()`. Ова е покажано во методите `toDouble()` и `gcd()`.

Употребата на џокер-знакот (`?`) во методот `add` означува дека типот на соодветниот податок може да биде која било класа што наследува од класата `Number`. Ова овозможува да се направи сума на две дропки во кои броителот и именителот не се инстанци од иста класа (пр. дропка со броител од тип `Short` и именител од тип `Long` и друга дропка со броител од тип `Long` и именител од тип `Integer`). Ако се употребат генеричките параметри `T` и `V` наместо џокер-знакот `?`, тогаш методот `add` ќе може да се користи само за дропки со исти типови на броител и именител.

```
import java.util.Scanner;

class GenericFraction<T extends Number, U extends Number> {
    private T numerator;
    private U denominator;

    public GenericFraction(T numerator, U denominator)
            throws ZeroDenominatorException {
        if (denominator.doubleValue() == 0) {
            throw new ZeroDenominatorException();
        }
        this.numerator = numerator;
        this.denominator = denominator;

    }

    double toDouble() {
        return this.numerator.doubleValue() /
                this.denominator.doubleValue();
    }

    static double gcd(double a, double b) {
        if (b == 0)
            return a;
        if (a < b)
            return gcd(a, b - a);
        else
            return gcd(b, a - b);
    }

    public double gcd() {
        return gcd(this.numerator.doubleValue(),
                this.denominator.doubleValue());
    }

    public GenericFraction<Double, Double> add(
            GenericFraction<? extends Number, ? extends Number> gf)
            throws ZeroDenominatorException {
        return new GenericFraction<Double, Double>(
                this.numerator.doubleValue()
                        * gf.denominator.doubleValue() +
                        this.denominator.doubleValue()
                                * gf.numerator.doubleValue(),
                this.denominator.doubleValue()
                        * gf.denominator.doubleValue());

    }

    @Override
    public String toString() {
        double gcd = this.gcd();
        return String.format("%.2f / %.2f",
                this.numerator.doubleValue() / gcd,
                this.denominator.doubleValue() / gcd);
    }
}

class ZeroDenominatorException extends Exception {
    public ZeroDenominatorException() {
        super("Denominator cannot be zero");
    }
```

```
}

public class GenericFractionTest {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        double n1 = scanner.nextDouble();
        double d1 = scanner.nextDouble();
        float n2 = scanner.nextFloat();
        float d2 = scanner.nextFloat();
        int n3 = scanner.nextInt();
        int d3 = scanner.nextInt();
        try {
            GenericFraction<Double, Double> gfDouble =
                    new GenericFraction<Double, Double>(n1, d1);
            GenericFraction<Float, Float> gfFloat =
                    new GenericFraction<Float, Float>(n2, d2);
            GenericFraction<Integer, Integer> gfInt =
                    new GenericFraction<Integer, Integer>(n3, d3);
            System.out.printf("%.2f\n", gfDouble.toDouble());
            System.out.println(gfDouble.add(gfFloat));
            System.out.println(gfInt.add(gfFloat));
            System.out.println(gfDouble.add(gfInt));
            gfInt = new GenericFraction<Integer, Integer>(n3, 0);
        } catch (ZeroDenominatorException e) {
            System.out.println(e.getMessage());
        }
        scanner.close();
    }
}
```

4.1.2 Генерички методи и интефејси

Интерфејсите може да имаат еден или повеќе параметарски типови. Деталите и нотацијата се исти како и во класите со параметарски типови. Параметарскиот тип може да се користи и кај методите од генеричка класа. Како дополнување, генеричкиот метод може да има свој параметарски тип, различен од оној на класата. Генеричкиот метод може да биде член на обична или на генеричка класа и да има свој параметарски тип. Параметарскиот тип на генеричкиот метод е локален и важи за самиот метод, но не и за класата.

> Проблем 4.5 Да се имплементира статички генерички метод за печатење на низа од елементи `static <E> void printArray(E[] inputArray)`. Методот треба да поддржува печатење на низи од различни податочни типови. ■

Кога работиме со генерички функции потребно е сите декларирани генерички типови во аргументите на генеричката функција и во типот на резултатот од генеричката функција, да се наведат пред типот на резултатот од функцијата во `<>`.

Во оваа задача е употребен само еден генерички тип (`E`) и истиот е аргумент на функцијата (пред `void` се декларира само `<E>`).

```
public class PrintArray {

    public static <E> void printArray(E[] inputArray) {
      for(E element : inputArray) {
         System.out.printf("%s ", element);
      }
      System.out.println();
```

```
    }

    public static void main(String args[]) {
        Integer[] intArray = { 1, 2, 3, 4, 5 };
        Double[] doubleArray = { 1.1, 2.2, 3.3, 4.4 };
        Character[] charArray = { 'H', 'E', 'L', 'L', 'O' };

        System.out.println("Array integerArray contains:");
        printArray(intArray);

        System.out.println("\nArray doubleArray contains:");
        printArray(doubleArray);

        System.out.println("\nArray characterArray contains:");
        printArray(charArray);
    }
}
```

Проблем 4.6 Доколку во претходната задача за генеричка дропка има дополнително барање кое гласи: „Да се напише статичка генеричка функција `create` којашто има два аргументи (именител и броител), објекти кои се од класа која наследува од класата `Number`, а враќа објект од тип генеричка дропка потписот и имплементацијата на таа функција би бил: ■

```
static <T extends Number, U extends Number>
GenericFraction<T,U> create (T numerator, U denominator)
        throws ZeroDenominatorException {
    return new GenericFraction(numerator, denominator);
}
```

Проблем 4.7 Да се имплементира статички генерички метод за наоѓање на максимум од три елементи `static <T extends Comparable<T> > T maximum(T x, T y, T z)`. Методот треба да поддржува споредување на елементи од различни податочни типови. ■

Во оваа функција имаме три аргументи кои се од генерички тип `T`. Затоа во декларацијата на генеричките параметри на почетокот на функцијата се наоѓа само генеричкиот параметар `T`.

Целта на оваа функција е да најде максимум од три елементи. За да може да се најде максимумот, мора генеричкиот тип `T` да биде споредлив. Па, затоа генеричкиот параметар е деклариран како `<T extends Comparable<T>>` во дефиницијата на методот.

```
public class Max {
    public static <T extends Comparable<T>> T maximum(T x, T y, T z) {
        T max = x;

        if(y.compareTo(max) > 0) {
            max = y;
        }

        if(z.compareTo(max) > 0) {
            max = z;
        }

        return max;
    }

```

```
16    public static void main(String args[]) {
17       System.out.printf("Max of %d, %d and %d is %d\n\n",
18                         3, 4, 5, maximum( 3, 4, 5 ));
19
20       System.out.printf("Max of %.1f,%.1f and %.1f is %.1f\n\n",
21                         6.6, 8.8, 7.7, maximum( 6.6, 8.8, 7.7 ));
22
23       System.out.printf("Max of %s, %s and %s is %s\n",
24                         "pear", "apple", "orange",
25                         maximum("pear", "apple", "orange"));
26    }
27 }
```

4.2 Генерички податочни структури

4.2.1 Колекции во Јава (листи и множества)

Јава колекција е секоја класа којашто организирано чува објекти и го имплементира `Collection` интерфејсот. На пример `ArrayList<T>` е Јава колекциска класа и ги имплементира сите методи на `Collection` интерфејсот. `Collection` интерфејсот е највисоко ниво на рамката за колекциски класи (класи-колекции) во Јава. Сите класи колекции се наоѓаат во пакетот `java.util` .

Проблем 4.8 Да се имплементира класа `BodyForm` која од влезен тек (стандарден влез, датотека, ...), ќе чита податоци за мерења на тежината на неколку луѓе. Податоците за луѓето и нивната тежина (секој човек може да има различен број на мерења) се дадени во следниот формат:

```
name weight1 weight2 weight3...
```

Пример:

```
James 81.3 90.8 87.9
```

Ваша задача е да ги имплементирате методите:

- `BodyForm()` - default конструктор
- `void readData(InputStream inputStream)` - метод за читање на податоците
- `void printByWeight(OutputStream outputStream, int type)` - метод кој ги печати сите луѓе сортирани според тежината во растечки редослед. Притоа ако аргументот `type` има вредност:
  - 1 се подредуваат според максималната измерена тежина
  - 2 се подредуваат според просечната измерена тежина (просек од сите измерени тежини).

Печатењето на луѓето е во следниот формат:

```
[name] MAX : [max.0] kg, AVG : [avg.0] kg
```

■

Во оваа задача, во класата `BodyForm` потребно е да се чува колекција од мерењата направени за секоја од личностите (објекти од класата `Person`). Бидејќи нема никакви барања за групирање на личностите или пак, за нивно чување во колекција којашто овозможува конкретен начин на управување, објектите од класата `Person` се чуваат во генеричка податочна структура - листа (`List<Person>`). Конкретната имплементација

на оваа генеричка податочна структура која се користи во решението е класата `ArrayList<Person>`.

Со читање на податоците за личностите (за секоја личност информациите се дадени во еден ред), се креира објект од класа `Person` и истиот се додава во листата од објекти од тип `Person`.

Во методот `printByWeight` треба да се овозможи подредување на елементите во листата по два критериуми. Доколку се бара подредување по само еден критериум, најчесто тоа се решава со имплементација на генеричкиот интерфејс `Comparable <Person>`. Кога се бара подредување по повеќе од еден критериум најдобро е да се имплементираат компаратори (имплементација на генеричкиот интерфејс `Comparator <Person>`), коишто би се користеле при повик на методот `Collections.sort()`. Во задачата се имплементирани два компаратори: `MaxComparator` (споредува според максимална измерена тежина) и `AvgComparator` (споредува според просечна измерена тежина).

Бидејќи компараторот е функциски интерфејс, истиот може да биде имплементиран со ламбда-израз. Како и секој друг интерфејс, може да биде имплементиран и со анонимни класи.

```
import java.io.*;
import java.util.*;

class BodyForm {
    private List<Person> persons;

    public BodyForm() {
        persons = new ArrayList<Person>();
    }

    void readData(InputStream inputStream) {
        Scanner scanner = new Scanner(inputStream);
        while (scanner.hasNext()) {
            String line = scanner.nextLine();
            String[] parts = line.split(" ");
            String name = parts[0];
            Person person = new Person(name);
            for (int i = 1; i < parts.length; ++i) {
                person.addMeasurment(Float.parseFloat(parts[i]));
            }
            persons.add(person);
        }
        scanner.close();
    }

    void printByWeight(OutputStream outputStream, int type) {
        PrintWriter pw = new PrintWriter(outputStream);
        Collections.sort(persons, (type == 1) ?
                new MaxComparator() : new AvgComparator());

        for (Person person : persons) {
            pw.println(person);
        }
        pw.flush();
    }

}

class Person {
    String name;
```

```
41      List<Float> measurements;
42      float max;
43      float sum;
44      float avg;
45
46      public Person(String name) {
47          this.name = name;
48          measurements = new ArrayList<Float>();
49          sum = 0;
50          max = Float.MIN_VALUE;
51      }
52
53      public void addMeasurment(float weightMeasurement) {
54          measurements.add(weightMeasurement);
55          sum += weightMeasurement;
56          if (weightMeasurement > max) {
57              max = weightMeasurement;
58          }
59          avg = sum / measurements.size();
60      }
61
62      @Override
63
64      public String toString() {
65          return String
66                  .format("%s MAX : %.1f kg, " +
67                          "AVG : %.1f kg", name, max, avg);
68      }
69  }
70
71  class MaxComparator implements Comparator<Person> {
72      @Override
73
74      public int compare(Person p1, Person p2) {
75          if (p1.max < p2.max) return -1;
76          else if (p1.max > p2.max) return 1;
77          else return 0;
78      }
79  }
80
81  class AvgComparator implements Comparator<Person> {
82      @Override
83
84      public int compare(Person p1, Person p2) {
85          if (p1.avg < p2.avg) return -1;
86          else if (p1.avg > p2.avg) return 1;
87          else return 0;
88      }
89  }
90
91  public class BodyFormTest {
92      public static void main(String[] args) {
93          BodyForm bodyForm = new BodyForm();
94          bodyForm.readData(System.in);
95          System.out.println("BY MAX");
96          bodyForm.printByWeight(System.out, 1);
97          System.out.println("BY AVG");
98          bodyForm.printByWeight(System.out, 2);
99      }
100 }
```

Проблем 4.9 Да се имплементира класа `ArchiveStore` во која се чува листа на архиви (елементи за архивирање).

Секој елемент за архивирање `Archive` има:

- `id` - цел број
- `dateArchived` - датум на архивирање.

Постојат два вида на елементи за архивирање, `LockedArchive` за кој дополнително се чува датум до кој не смее да се отвори `dateToOpen` и `SpecialArchive` за кој се чуваат максимален број на дозволени отворања `maxOpen`. За елементите за архивирање треба да се обезбедат следните методи:

- `LockedArchive(int id, LocalDate dateToOpen)` - конструктор за заклучена архива
- `SpecialArchive(int id, int maxOpen)` - конструктор за специјална архива

За класата `ArchiveStore` да се обезбедат следните методи:

- `ArchiveStore()` - предефиниран конструктор
- `void archiveItem(Archive item, LocalDate date)` - метод за архивирање елемент item на одреден датум date
- `void openItem(int id, LocalDate date)` - метод за отворање елемент од архивата со зададен id и одреден датум date. Ако не постои елемент со даденото id треба да се фрли исклучок од тип `NonExistingItemException` со порака `Item with id [id] doesn't exist`.
- `String getLog()` - враќа стринг со сите пораки запишани при архивирањето и отворањето архиви во посебен ред.

За секоја акција на архивирање во текст треба да се додаде следната порака `Item [id] archived at [date]`, додека за секоја акција на отворање архива треба да се додаде `Item [id] opened at [date]`. При отварање ако се работи за `LockedArhive` и датумот на отворање е пред датумот кога може да се отвори, да се додаде порака `Item [id] cannot be opened before [date]`. Ако се работи за `SpecialArhive` и се обидеме да ја отвориме повеќепати од дозволениот број (`maxOpen`) да се додаде порака `Item [id] cannot be opened more than [maxOpen] times`.

Во решението на оваа задача потребно да се употребат концептите на наследување и полиморфизам. Прво треба да се дефинира апстрактната класа `Archive` со еден апстрактен метод `void open (Date date)`. Од оваа класа треба да се изведат класи за двата типа на архиви: `LockedArchive` и `SpecialArchive`. Во овие две класи треба соодветно да се препокрие методот `open()`, во согласност со барањата на задачата.

Слично како и во претходната задача, архивите (објекти од класата `Archive`) ќе се чуваат во генеричката податочната структура `ArrayList<Archive>` во рамки на класата `ArchiveStore`.

```
1 import java.time.LocalDate;
2 import java.util.ArrayList;
3 import java.util.Scanner;
4
5 class ArchiveStore {
6     private ArrayList<Archive> items;
7     private StringBuilder log;
8
```

```
9     public ArchiveStore() {
10        items = new ArrayList<Archive>();
11        log = new StringBuilder();
12    }
13
14    public void archiveItem(Archive item, LocalDate date) {
15        item.archive(date);
16        items.add(item);
17        log.append(String.format("Item %d archived at %s\n",
18                item.getId(), date.toString()));
19    }
20
21    public Archive openItem(int id, LocalDate date)
22            throws NonExistingItemException {
23        for (Archive item : items) {
24            if (item.getId() == id) {
25                try {
26                    item.open(date);
27                } catch (InvalidArchiveOpenException e) {
28                    log.append(e.getMessage());
29                    log.append("\n");
30                    return item;
31                }
32                log.append(String.format("Item %d opened at %s\n",
33                        item.getId(), date));
34                return item;
35            }
36        }
37        throw new NonExistingItemException(id);
38    }
39
40    public String getLog() {
41        return log.toString();
42    }
43 }
44
45 class NonExistingItemException extends Exception {
46    public NonExistingItemException(int id) {
47        super(String.format("Item with id %d doesn't exist", id));
48    }
49 }
50
51 abstract class Archive {
52    protected int id;
53    protected LocalDate dateArchived;
54
55    public void archive(LocalDate date) {
56        this.dateArchived = date;
57    }
58
59    public abstract LocalDate open(LocalDate date)
60            throws InvalidArchiveOpenException;
61
62    public int getId() {
63        return id;
64    }
65
66    @override
67    public boolean equals(Object obj) {
68        Archive archive = (Archive) obj;
69        return this.id == archive.id;
```

```
70      }
71  }
72
73  class LockedArchive extends Archive {
74      private LocalDate dateToOpen;
75
76      public LockedArchive(int id, LocalDate dateToOpen) {
77          this.id = id;
78          this.dateToOpen = dateToOpen;
79      }
80
81      @override
82      public LocalDate open(LocalDate date)
83              throws InvalidArchiveOpenException {
84          if (date.isBefore(dateToOpen))
85              throw new InvalidArchiveOpenException(String.format(
86                      "Item %d cannot be opened before %s", id,
87                      dateToOpen));
88          return date;
89
90      }
91  }
92
93  class SpecialArchive extends Archive {
94      private int countOpened;
95      private int maxOpen;
96
97      public SpecialArchive(int id, int maxOpen) {
98          this.id = id;
99          countOpened = 0;
100         this.maxOpen = maxOpen;
101     }
102
103     @override
104     public LocalDate open(LocalDate date)
105             throws InvalidArchiveOpenException {
106         if (countOpened >= maxOpen)
107             throw new InvalidArchiveOpenException(String.format(
108                     "Item %d cannot be opened more than %d times", id,
109                     maxOpen));
110         ++countOpened;
111         return date;
112     }
113
114 }
115
116 class InvalidArchiveOpenException extends Exception {
117     public InvalidArchiveOpenException(String message) {
118         super(message);
119     }
120 }
121
122 public class ArchiveStoreTest {
123     public static void main(String[] args) {
124         ArchiveStore store = new ArchiveStore();
125         LocalDate date = LocalDate.of(2013, 10, 7);
126         Scanner scanner = new Scanner(System.in);
127         scanner.nextLine();
128         int n = scanner.nextInt();
129         scanner.nextLine();
130         scanner.nextLine();
```

```
131         int i;
132         for (i = 0; i < n; ++i) {
133             int id = scanner.nextInt();
134             long days = scanner.nextLong();
135 
136             LocalDate dateToOpen = date.atStartOfDay()
137                     .plusSeconds(days * 24 * 60 * 60).toLocalDate();
138             LockedArchive lockedArchive =
139                     new LockedArchive(id, dateToOpen);
140             store.archiveItem(lockedArchive, date);
141         }
142         scanner.nextLine();
143         scanner.nextLine();
144         n = scanner.nextInt();
145         scanner.nextLine();
146         scanner.nextLine();
147         for (i = 0; i < n; ++i) {
148             int id = scanner.nextInt();
149             int maxOpen = scanner.nextInt();
150             SpecialArchive specialArchive =
151                     new SpecialArchive(id, maxOpen);
152             store.archiveItem(specialArchive, date);
153         }
154         scanner.nextLine();
155         scanner.nextLine();
156         while (scanner.hasNext()) {
157             int open = scanner.nextInt();
158             try {
159                 store.openItem(open, date);
160             } catch (NonExistingItemException e) {
161                 System.out.println(e.getMessage());
162             }
163         }
164         System.out.println(store.getLog());
165     }
166 }
```

Во методот `openItem(int id, LocalDate date)` , потребно е да се измине листата од архиви за да се најде архивата со соодветно `id` . За пребарување на елементи според некој критериум (како што е побарано во рамки на оваа задача), најчесто се препорачува прво да се индексираат елементите за да се овозможи побрзо пребарување. Во продолжение се дадени примери во кои се користат податочни структури коишто овозможуваат индексирање на елементите.

Проблем 4.10 Да се напише класа за книга `Book` во која се чува: Наслов, категорија и цена. За оваа класа, да се имплементира преоптоварен конструктор со следните аргументи `Book(String title, String category, float price)` .

Потоа да се напише класа `BookCollection` во која се чува колекција од книги. Во оваа класа треба да се имплементираат следните методи:

- `public void addBook(Book book)` - додавање книга во колекцијата
- `public void printByCategory(String category)` - ги печати сите книги од проследената категорија (се споредува стрингот без разлика на мали и големи букви), сортирани според насловот на книгата (ако насловот е ист, се сортираат според цената).
- `public List<Book> getCheapestN(int n)` - враќа листа на најевтините N

книги (ако има помалку од N книги во колекцијата, ги враќа сите).

Во оваа задача, потребно е да се подредат книгите по два критериуми. Затоа се креираат два компаратори (класи што го имплементираат интерфејсот `Comparator`): `TitleComparator` и `PriceComparator`.

Оваа задача е решена со користење на податочната структура множество којашто овозможува чување само на уникатни елементи. Со користење на `TreeSet` имплементацијата, книгите директно се подредуваат при нивното додавање во множеството според употребениот компаратор. На овој начин се гарантира елементите во податочната структура `TreeSet` секогаш да бидат подредени во согласност со зададениот критериум.

Компараторот (`PriceComparator`) се користи при креирање множество од тип `TreeSet`. Компараторот `TitleComparator` се користи во методот `printByCategory (String category)` за да се подредат книгите според категоријата во која припаѓаат. Подредувањето на книгите според категорија е реализирано со помошно множество од тип `TreeSet`.

```
import java.util.*;

class Book {
    private String title;
    private String category;
    private float price;

    public Book(String title, String category, float price) {
        this.title = title;
        this.category = category;
        this.price = price;
    }

    public String getTitle() {
        return title;
    }

    public String getCategory() {
        return category;
    }

    public float getPrice() {
        return price;
    }

    @override
    public String toString() {
        return String.format("%s (%s) %.2f", title, category, price);
    }
}

class TitleComparator implements Comparator<Book> {

    @override
    public int compare(Book o1, Book o2) {
        int result = o1.getTitle().compareTo(o2.getTitle());
        if (result != 0)
            return result;
        return Float.compare(o1.getPrice(), o2.getPrice());

    }
}
```

```
43
44 class PriceComparator implements Comparator<Book> {
45
46     @override
47     public int compare(Book o1, Book o2) {
48         int result = Float.compare(o1.getPrice(), o2.getPrice());
49         if (result != 0) {
50             return result;
51         }
52         return o1.getTitle().compareTo(o2.getTitle());
53     }
54 }
55
56 class BookCollection {
57     private TreeSet<Book> books;
58
59     public BookCollection() {
60         books = new TreeSet<>(new PriceComparator());
61     }
62
63
64     public void addBook(Book book) {
65         books.add(book);
66     }
67
68     public void printByCategory(String category) {
69         Set<Book> booksByTitle = new TreeSet<>(new TitleComparator());
70         booksByTitle.addAll(books);
71         for (Book book : booksByTitle) {
72             if (book.getCategory().equalsIgnoreCase(category)) {
73                 System.out.println(book);
74             }
75         }
76     }
77
78
79     public List<Book> getCheapestN(int n) {
80         List<Book> firstNBooks = new ArrayList<>();
81         int i = 0;
82         for (Book b : books) {
83             if (i < n) {
84                 firstNBooks.add(b);
85             }
86             ++i;
87         }
88         return firstNBooks;
89     }
90 }
91
92 public class BooksTest {
93     public static void main(String[] args) {
94         Scanner scanner = new Scanner(System.in);
95         int n = scanner.nextInt();
96         scanner.nextLine();
97         BookCollection booksCollection = new BookCollection();
98         Set<String> categories =
99                 fillCollection(scanner, booksCollection);
100        System.out.println("=== PRINT BY CATEGORY ===");
101        for (String category : categories) {
102            System.out.println("CATEGORY: " + category);
103            booksCollection.printByCategory(category);
```

```
104         }
105         System.out.println("=== TOP N BY PRICE ===");
106         print(booksCollection.getCheapestN(n));
107     }
108
109     static void print(List<Book> books) {
110         for (Book book : books) {
111             System.out.println(book);
112         }
113     }
114
115     static TreeSet<String> fillCollection(Scanner scanner,
116                                           BookCollection collection) {
117         TreeSet<String> categories = new TreeSet<String>();
118         while (scanner.hasNext()) {
119             String line = scanner.nextLine();
120             String[] parts = line.split(":");
121             Book book = new Book(parts[0], parts[1],
122                     Float.parseFloat(parts[2]));
123             collection.addBook(book);
124             categories.add(parts[1]);
125         }
126         return categories;
127     }
128 }
```

При користење на множества од типот (`TreeSet`) треба да се внимава на имплементацијата на компараторите, односно на имплементацијата на `compareTo()` методот во самите класи. Бидејќи множествата чуваат уникатни елементи. Во овој случај, доколку компараторот за цена споредбата ја прави само по цената на книгите, во множество ќе биде додадена само една книга од сите книги со иста цена. Тоа ќе биде книгата која последно сме се обиделе да ја внесеме во множеството. Иако не е наведено во барањата на задачата, во компараторите покрај споредба на цена/наслов се прави и споредба на преостанатите полиња со цел да се избегне оваа ситуација.

Проблем 4.11 Да се дефинира класа `Component` за која се чуваат боја, тежина и колекција од внатрешни компоненти (референци од класата `Component`). Во оваа класа да се дефинираат методите:

- `Component(String color, int weight)` - конструктор со аргументи боја и тежина
- `void addComponent(Component component)` - за додавање нова компонента во внатрешната колекција (во оваа колекција компонентите секогаш се подредени според тежината во растечки редослед. Ако имаат иста тежина подредени се алфабетски според бојата).
- `void changeColor(int weight, String color)` - ја менува бојата на сите компоненти со тежина помала од проследената.
- `String toString()` - враќа стринг репрезентација на објектот, во следниот формат:

```
weight1:color1
---weight2:color2
------weight3:color3
------weight4:color4
...
```

```
---weight5:color5
------weight6:color6
------weight7:color7
---------weight8:color8
```

Во оваа задача, класата којашто треба да се имплементира, треба да чува (управува со) референци од објекти од самата таа класа. Овој концепт овозможува креирање на структура од објекти во форма на дрво или граф. Во поглавјето „Шаблони за развој на софтвер“ се воведува шаблон што користи слична парадигма за решавање на конкретен тип на проблеми.

За да се овозможи чување на подредени елементи во согласност со одреден критериум, колекцијата од компоненти е множество од тип `TreeSet`. Изборот на овој тип на множество наметнува имплементација на `compareTo` методот во рамките на класата `Component` или имплементација на соодветен `Comparator`.

При имплементација на конкретни функционалности кај овие класи, најчесто се употребува рекурзија, со цел да се овозможи изминување (посетување) на елементите од сите нивоа на податочната структура.

Во решението на задачата, два методи се имплементирани со помош на рекурзија:

- `changeColor(int weight , String color)` - Во овој метод прво се проверува тежината на компонента. Ако таа е помала од аргументот тежина (`weight`) се менува бојата на компонентата. Потоа, рекурзивно се посетуваат сите внатрешни компоненти со помош на статичкиот рекурзивен метод `change(Component component, int weight, String color)`, кој исто така ја менува бојата на сите компоненти ако нивната тежина е помала од аргументот тежина (`weight`).
- `toString()` - Во овој метод, при креирањето на текстуалната репрезентација на компонентите треба да се овозможи додавање на дополнителни црти за секое следно ниво во хиерархијата од компоненти. За таа цел е имплементиран помошниот статички рекурзивен метод `createString(StringBuilder sb, Component component, int level)` што рекурзивно ги поминува внатрешните компоненти на секоја компонен-та. При процесирањето на внатрешните компоненти се менува и нивото (`level`). Оваа променлива ја чува информацијата за бројот на црти коишто треба да се додадат при креирањето на текстуалната репрезентација. При повик на `toString` методот на една компонента, се повикува помошниот метод `createString` со празен `StringBuilder` на којшто ќе се додаваат текстуалните репрезентации на сите внатрешни компоненти.

```
1  import java.util.Scanner;
2  import java.util.Set;
3  import java.util.TreeSet;
4
5
6  class Component implements Comparable<Component> {
7      String color;
8      int weight;
9      Set<Component> inner;
10
11     public Component(String color, int weight) {
12         this.color = color;
13         this.weight = weight;
14         this.inner = new TreeSet<Component>();
15     }
16
```

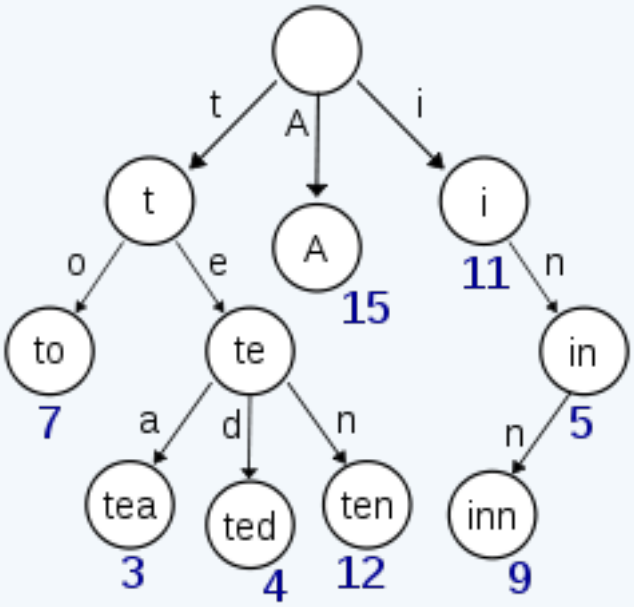
```
17      public void addComponent(Component component) {
18          inner.add(component);
19      }
20
21      public void changeColor(int weight, String color) {
22          if (this.weight < weight) {
23              this.color = color;
24          }
25          for (Component composite : inner) {
26              change(composite, weight, color);
27          }
28      }
29
30      static void change(Component component, int weight, String color) {
31          if (component.weight < weight) {
32              component.color = color;
33          }
34          for (Component c : component.inner) {
35              change(c, weight, color);
36          }
37      }
38
39      static void createString(StringBuilder sb, Component component,
40                               int level) {
41          for (int i = 0; i < level; ++i)
42              sb.append("---");
43          sb.append(String.format("%d:%s\n", component.weight,
44                  component.color));
45          for (Component c : component.inner) {
46              createString(sb, c, level + 1);
47          }
48      }
49
50      @Override
51
52      public int compareTo(Component o) {
53          int w = this.weight - o.weight;
54          if (w == 0) return this.color.compareTo(o.color);
55          return w;
56      }
57
58      @Override
59
60      public String toString() {
61          StringBuilder sb = new StringBuilder();
62          Component.createString(sb, this, 0);
63          return sb.toString();
64      }
65  }
66
67  public class ComponentTest {
68      public static void main(String[] args) {
69          Scanner scanner = new Scanner(System.in);
70          String name = scanner.nextLine();
71          Component prev = null;
72          while (true) {
73              int what = scanner.nextInt();
74              scanner.nextLine();
75              if (what == 1) {
76                  String color = scanner.nextLine();
77                  int weight = scanner.nextInt();
```

```
78                  Component component = new Component(color, weight);
79                  prev = component;
80              } else if (what == 2) {
81                  String color = scanner.nextLine();
82                  int weight = scanner.nextInt();
83                  Component component = new Component(color, weight);
84                  prev.addComponent(component);
85                  prev = component;
86              } else if (what == 3) {
87                  String color = scanner.nextLine();
88                  int weight = scanner.nextInt();
89                  Component component = new Component(color, weight);
90                  prev.addComponent(component);
91              } else if (what == 4) {
92                  break;
93              }
94              scanner.nextLine();
95          }
96      }
97  }
```

4.2.2 Мапи

Мапа е генеричка податочна структура којашто чува парови од клуч и вредност. Клучевите во мапата се уникатни и еден клуч може да биде поврзан единствено со една вредност. Мапите како податочни структури се корисни при пребарување, ажурирање или бришење на елементи врз основа на даден клуч. Интерфејсот `Map` вклучува методи за основните операции како додавање на пар од клуч и вредност, читање на вредност преку неговиот клуч, отстранување на пар од клуч и вредност, тестирање на присуство на клуч или вредност, број на парови во структурата и тестирање дали податочната структура е празна или не.

> Проблем 4.12 Да се имплементира податочната структура `Trie`. `Trie`, или како што се нарекува префикс-дрво (анг. prefix tree), (бидејќи може да се пребарува преку префикси), претставува вид на подредено пребарувачко дрво во кое позициите на јазлите ги дефинираат клучевите со кои истите се асоцирани. Сите наследници на даден јазол имаат ист (заеднички) префикс на зборот (текстуалната низа) асоциран со тој јазол. Коренот е асоциран со празна текстуална низа.



Пример:

Во `Trie` податочната структура прикажана на сликата се додадени зборовите (клучевите): `A`, `to`, `tea`, `ted`, `ten`, `i`, `in`, и `inn`.

Trie податочната структура која треба да се имплементира треба да има два методи:

- Методот `void addString(String word)` - метод за додавање на нов збор во индексираната структура.
- `List<String> complete(String word)` - метод којшто треба да ги врати сите зборови внесени во податочната структура кои започнуваат со зборот (имаат префикс) `word`. Зборовите содржани во други зборови не се земаат во предвид.

Пример: Ако зборовите `accord` и `according` се внесени во податочната структура, се враќа само зборот `according`. ■

Од цртежот даден во описот на задачата, може да се согледа дека ќе треба да дефинираме податочна структура слична на дрво. За да се дефинира дрво податочната структура, прво потребно е да се дефинира класа јазол (`Node`), којашто претставува основен градбен елемент на дрво податочната структура. јазолот е дефиниран со класата `Node` во која се чува вредност на јазолот (знакот, `value`) и референци кон неговите деца-јазли. За чување на референците на деца-јазлите се користи мапа во која клуч е знакот запишан во дете-јазолот, а вредност е објект од класата `Node` што го претставува соодветниот дете-јазол. Мапата е од тип `TreeMap` со цел клучевите да бидат лексикографски подредени. На тој начин како резултат од изминувањето на дрвото во методот `complete` се добие листа од лексикографски подредени зборови. Податочната структура `Trie` е претставена само преку јазолот-корен.

1. `void addString(String word)` - За да се имплементира овој метод креиран е помошниот метод `void addStringRecursive(Node node, String word, int position)` со следните аргументи:
   - Првиот аргумент се однесува на јазолот којшто моментално се проверува и овој аргумент се менува постојано во секој рекурзивен повик.
   - Вториот аргумент го означува зборот кој се додава во структурата `Trie` и истиот е константен, односно не се менува низ рекурзивните повици.
   - Третиот аргумент е позицијата на буквата од зборот којашто се процесира. Позицијата се менува низ секој рекурзивен повик.

Прво, методот се повикува за коренот на дрвото и за позиција 0 (првата буква). Методот рекурзивно се извршува сè додека позицијата не се изедначи со бројот на букви во зборот, односно, сè додека не се поминати сите букви од зборот. Доколку јазолот `node` има дете со вредност иста како буквата што се наоѓа на позиција `position`, тогаш методот се повикува рекурзивно за детето-јазол. Доколку јазолот `node` нема дете-јазол со вредност иста како буквата на позиција `position`, тогаш методот се повикува рекурзивно на ново-креирано дете јазол со вредност еднаква на буквата на позиција `position`. На овој начин дрвото од тип `Trie` рекурзивно се полни за секој додаден збор.

2. `List<String> complete(String word)` - Во методот `complete` прво се бара јазолот во кој се наоѓа последната буква од зборот `word`, проверувајќи ги сите претходни букви од зборот. За таа цел, јазолот `prefixNode` се поставува првично на коренот на дрвото. Потоа, се итерираат сите букви од зборот и се бара детето на `prefixNode` со вредност како таа буква. Доколку не се најде дете јазол со таква вредност (методот `getChid` врати `null`), тоа значи дека во дрвото нема запишано збор со префикс `word` и се враќа празна листа како резултат. Доколку се врати јазол различен од `null`, тој јазол се запишува во `prefixNode` и постапката продолжува со следната буква од зборот. Откако е најден јазолот во кој се наоѓа последната буква од зборот `word`, се повикува помошниот рекурзивен метод `searchWordsFromNode(String word, Node node, List<String> results)` со следните аргументи:
   - Првиот аргумент е зборот кој го пребаруваме. Тоа е почетниот префикс за сите зборови кои се додадени во структурата. Истиот се менува низ секој рекурзивен повик на тој начин што му се додава буквата, којашто е запишана во јазолот кој се процесира. Зборот целосно се оформува кога ќе стигнеме до јазол-лист во дрвото.
   - Вториот аргумент е јазолот кој се процесира
   - Третиот аргумент е листата од сите зборови со префикс `word` кои се дел од изградената `Trie` структурата.

   Методот `searchWordsFromNode` е помошен метод којшто се повикува во `complete` методот. Истиот овозможува креирање на зборовите кои се дел од `Trie` структурата.

```
1  import java.util.*;
2
3  class Node {
4      Character value;
5      Map<Character, Node> children;
6
7      public Node() {
8          value = ' ';
9          children = new TreeMap<>();
10     }
11
12     public Node(Character value) {
13         this();
14         this.value = value;
15     }
16
17     public Node getChild(Character c) {
18         return children.get(c);
19     }
20
21     public char getValue() {
```

```
22          return value;
23      }
24
25      public Node addChild(char c) {
26          Node node = new Node(c);
27          children.put(c, node);
28          return node;
29      }
30
31      @Override
32      public String toString() {
33          return toString(this, 0);
34      }
35
36      public static String toString(Node node, int level) {
37          StringBuilder sb = new StringBuilder();
38          for (int i = 0; i < level; i++)
39              sb.append("-");
40          sb.append(node.value).append("\n");
41          node.children.values().forEach(children -> sb
42                  .append(Node.toString(children, level + 1)));
43          return sb.toString();
44      }
45
46      public boolean hasChildren() {
47          return children.size() != 0;
48      }
49
50      public boolean hasChild(char c) {
51          return (children.get(c) != null);
52      }
53  }
54
55  class Trie {
56
57      Node root;
58
59      public Trie() {
60          root = new Node();
61      }
62
63      public void addString(String word) {
64          addStringRecursive(root, word, 0);
65      }
66
67      public static void addStringRecursive(Node node, String word,
68                                            int position) {
69          if (position != word.length()) {
70              char c = word.charAt(position);
71              if (node.hasChild(c)) {
72                  addStringRecursive(node.getChild(c), word,
73                          position + 1);
74              } else {
75                  addStringRecursive(node.addChild(c), word,
76                          position + 1);
77              }
78          }
79      }
80
81      List<String> complete(String word) {
82          Node prefixNode = root;
```

```
83          List<String> results = new ArrayList<>();
84          for (int i = 0; i < word.length(); i++) {
85              if (!prefixNode.hasChild(word.charAt(i)))
86                  return new ArrayList<>();
87              prefixNode = prefixNode.getChild(word.charAt(i));
88          }
89          searchWordsFromNode(word, prefixNode, results);
90          return results;
91      }
92
93      public void searchWordsFromNode(String word, Node node,
94                                      List<String> results) {
95          if (!node.hasChildren()) {
96              results.add(word);
97          } else {
98              for (char c : node.children.keySet()) {
99                  searchWordsFromNode(word + c, node.getChild(c),
100                         results);
101             }
102         }
103     }
104
105     @Override
106     public String toString() {
107         return root.toString();
108     }
109 }
110
111 public class TrieTest {
112
113     public static void main(String[] args) {
114
115         Trie trie = new Trie();
116
117         List<String> completeWords = new ArrayList<>();
118         boolean complete = false;
119
120         Scanner scanner = new Scanner(System.in);
121         while (scanner.hasNext()) {
122             String word = scanner.next();
123             if (word.equals("COMPLETE:")) {
124                 complete = true;
125                 continue;
126             }
127             if (complete) {
128                 completeWords.add(word);
129             } else {
130                 trie.addString(word);
131             }
132         }
133         scanner.close();
134
135         for (String word : completeWords) {
136             System.out.println(word);
137             List<String> completedWords = trie.complete(word);
138             System.out.println(completedWords);
139         }
140     }
141 }
```

Проблем 4.13 Да се имплементира класа `UserGroups` во која ќе се чуваат информациите за корисници (имиња) и групите во кои тие се членови.

Во класата да се имплементираат следните методи:

- `void addUsers(String[] users, String[] groups)` - метод за додавање на корисниците (низите `users` и `groups` се со иста должина и за секој корисник на истиот индекс во низата `groups` е групата во која припаѓа.
- `void setParentGroups(String[] groups, String[] parentGroups)` - во овој метод на група од низата `groups` се поставува родител-група од `parentGroups`. Ова означува дека сите корисници кои се членови на некоја група `group` треба да бидат членови и на евентуалниот родител на оваа група (ако го има).
- `void printUsers()` - се печатат сите корисници подредени според бројот во името во растечки редослед и групите во кои припаѓа, разделени со запирка
- `void printGroups()` - се печатат сите групи подредени лексикографски и сите корисници во групата разделени со запирка.

Напомена: Имињата на корисниците се уникатни. ■

При решавање на овој проблем (како и за сите други), прво треба да одлучиме кои податочни структури ќе се користат за чување и менаџирање со корисниците на системот и групите заедно со нивните членови. Со цел да се покаже употребата на мапа податочната структура, корисниците независно се чуваат во една мапа (`users` , каде што клучот е името на корисникот, а вредност е објект од класата `User`). Корисниците по групи се чуваат во друга мапа (`groups` , каде што клучот е името на групата, а вредност е множество од сите корисници кои се дел од таа група). Имплементацијата којашто се користи за мапата `groups` (`TreeMap`) е со цел паровите од клуч и вредност да се чуваат подредени според вредноста на клуч. Односно, според името на групата (во согласност со поставеното барање во функцијата `printGroups()` .

Дополнително, за секој корисник се чува информација за групите во кои корисникот припаѓа. Групите на корисникот се чуваат во подредено множество (`TreeSet`), со цел истите да бидат лексикографски подредени за потребите на методот за печатење `printGroups()` .

Во методот `printUsers()` сите корисници од мапата се додаваат во подредено множество (`TreeSet`) за да се подредат според името на корисникот.

```
1  import java.util.*;
2
3  class UserGroups {
4      Map<String, User> users;
5      Map<String, Set<User>> groups;
6
7      public UserGroups() {
8          users = new HashMap<>();
9          groups = new TreeMap<>();
10     }
11
12     public void addUsers(String[] users, String[] groups) {
13         for (int i = 0; i < users.length; ++i) {
14             User user = new User(users[i]);
15             user.addGroup(groups[i]);
16             this.users.put(users[i], user);
17             Set<User> groupUsers = this.groups.get(groups[i]);
18             if (groupUsers == null) {
19                 groupUsers = new TreeSet<>();
20             }
```

```
            groupUsers.add(user);
            this.groups.put(groups[i], groupUsers);
        }
    }

    public void setParentGroups(String[] groups,
                                String[] parentGroups) {
        for (int i = 0; i < groups.length; ++i) {
            Set<User> users = this.groups.get(groups[i]);
            Set<User> pgUsers = this.groups.get(parentGroups[i]);
            if (users != null && pgUsers != null) {
                pgUsers.addAll(users);
                for (User user : users) {
                    user.groups.add(parentGroups[i]);
                }
            }
        }
    }

    public void printUsers() {
        TreeSet<User> users = new TreeSet<>(this.users.values());
        for (User user : users) {
            System.out.println(user);
        }
    }

    public void printGroups() {
        for (String group : groups.keySet()) {
            System.out.print(group);
            System.out.print(" : ");
            int i = 0;
            Set<User> users = groups.get(group);
            for (User user : users) {
                System.out.print(user.name);
                if (++i != users.size()) {
                    System.out.print(", ");
                }
            }
            System.out.println();
        }
    }
}

class User implements Comparable<User> {
    String name;
    int number;
    Set<String> groups;

    public User(String name) {
        this.name = name;
        number = Integer.parseInt(name.split("_")[1]);
        this.groups = new TreeSet<>();
    }

    public void addGroup(String name) {
        this.groups.add(name);
    }

    @Override
    public String toString() {
        StringBuilder res = new StringBuilder();
```

```
82          res.append(name);
83          res.append(" : ");
84          for (String g : groups) {
85              res.append(g);
86              res.append(", ");
87          }
88          res.delete(res.length() - 2, res.length());
89          return res.toString();
90      }
91
92      @Override
93      public int compareTo(User o) {
94          return Integer.compare(number, o.number);
95      }
96  }
97
98  public class UserGroupsTest {
99      public static void main(String[] args) {
100         Scanner scanner = new Scanner(System.in);
101         UserGroups userGroups = new UserGroups();
102         int n = scanner.nextInt();
103         scanner.nextLine();
104         String[] users = new String[n];
105         String[] groups = new String[n];
106         for (int i = 0; i < n; ++i) {
107             String[] parts = scanner.nextLine().split(":");
108             users[i] = parts[0];
109             groups[i] = parts[1];
110         }
111         userGroups.addUsers(users, groups);
112         n = scanner.nextInt();
113         scanner.nextLine();
114         String[] groups1 = new String[n];
115         String[] parentGroups = new String[n];
116         for (int i = 0; i < n; ++i) {
117             String[] parts = scanner.nextLine().split(":");
118             groups1[i] = parts[0];
119             parentGroups[i] = parts[1];
120         }
121         userGroups.setParentGroups(groups1, parentGroups);
122         System.out.println("--- USERS ---");
123         userGroups.printUsers();
124         System.out.println("--- GROUPS ---");
125         userGroups.printGroups();
126         scanner.close();
127     }
128 }
```

Проблем 4.14 Да се имплементира класа за именик `PhoneBook` со следните методи:

- `void addContact(String name, String number)` - додава нов контакт со име и број во именикот. Ако се обидеме да додадеме контакт со веќе постоечки број, треба да се фрли исклучок од класа `DuplicateNumberException` со порака `Duplicate number: [number]`. Комплексноста на овој метод не треба да надминува $O(logN)$ за N контакти.
- `void contactsByNumber(String number)` - ги печати сите контакти кои во бројот го содржат бројот пренесен како аргумент во методот (минимална

должина на бројот [number] е 3). Комплексноста на овој метод не треба да надминува $O(NlogN)$ за N контакти.

- `void contactsByName(String name)` - ги печати сите контакти кои даденото име. Комплексноста на пристапот до контактите со име `name` треба да биде $O(1)$.

Во двата методи контактите се печатат сортирани лексикографски според името, а оние со исто име и потоа според бројот. ■

За да се избегне додавање на дупликати телефонски броеви со методот `addContact(String name, String number)` од класата `PhoneBook`, телефонските броеви треба да се чуваат во множество. Бидејќи немаме барање за подредување на телефонските броеви во функционалните барања, телефонските броеви ќе бидат чувани во `HashSet`, чија комплексноста за пристап до елемент е $O(1)$.

Од барањата за методот `contactsByNumber(String number)` може да се заклучи дека телефонските броеви треба да се чуваат во посебна податочна структура, каде истите ќе бидат групирани според сите можни комбинации од 3-цифрени до 9-цифрени броеви кои се дел од самите броеви (пр. телефонскиот број 076123456 ги содржи следните комбинации на цифри од 3-цифрени до 9-цифрени броеви: 076, 761, 612, 123, 234, 345, 456, 0761, 7612, ..., 07612345, 76123456, 076123456). За таа цел, телефонските броеви/контактите се чуваат во мапа во која клуч ќе бидат сите 3-9 цифрени комбинации на броеви од броевите што се додаваат во методот `addContact`. Вредност ќе биде подредено множество од објекти од класата `Contact` чиј телефонски број го содржи клучот. Мапата е од тип `HashMap` и гарантира комплексноста од $O(1)$ за пристап до броевите што го содржат бројот `number`. Бидејќи множеството е подредено (`TreeSet`), комплексноста за печатење на контактите е $O()$ каде е бројот на пронајдени контакти и $<= N$ (контактите се веќе индексирани според пребарувачкиот критериум).

За потребите на методот `contactsByName(String name)` контактите треба да се индексираат според нивното име. Индексирачката податочна структура е мапа во која клуч е името на контактот, а вредност е подредено множество од сите контакти со име `name`. Мапата е од тип `HashMap` која гарантира комплексност од $O(1)$ за пристап до контактите со име `name`.

За да се постигне целосна функционалност при додавање на нови контакти со методот `addContact(String name, String number)`, сите податочни структури мора да се пополнат и обноват во согласност со новодобиените податоци.

```
1  import java.util.ArrayList;
2  import java.util.HashMap;
3  import java.util.HashSet;
4  import java.util.List;
5  import java.util.Scanner;
6  import java.util.Set;
7  import java.util.TreeMap;
8  import java.util.TreeSet;
9
10 class PhoneBook {
11     TreeMap<String, Set<Contact>> contactsNumber;
12     HashMap<String, Set<Contact>> contactsName;
13     HashSet<String> numbers;
14
15     public PhoneBook() {
16         contactsNumber = new TreeMap<String, Set<Contact>>();
17         contactsName = new HashMap<String, Set<Contact>>();
18         numbers = new HashSet<String>();
```

```
19      }
20
21      public void addContact(String name, String number)
22              throws DuplicateNumberException {
23          if (numbers.contains(number)) {
24              throw new DuplicateNumberException(number);
25          }
26          numbers.add(number);
27          Contact contact = new Contact(name, number);
28          List<String> keys = contact.getPhoneKeys();
29          for (String key : keys) {
30              Set<Contact> contacts = contactsNumber.get(key);
31              if (contacts == null) {
32                  contacts = new TreeSet<Contact>();
33                  contactsNumber.put(key, contacts);
34              }
35              contacts.add(contact);
36          }
37          Set<Contact> nameContacts = contactsName.get(name);
38          if (nameContacts == null) {
39              nameContacts = new TreeSet<Contact>();
40              contactsName.put(name, nameContacts);
41          }
42          nameContacts.add(contact);
43      }
44
45      public void contactsByNumber(String number) {
46          Set<Contact> contacts = contactsNumber.get(number);
47          if(contacts == null) {
48              System.out.println("NOT FOUND");
49              return;
50          }
51          for (Contact contact : contacts) {
52              System.out.println(contact);
53          }
54      }
55
56      public void contactsByName(String name) {
57          Set<Contact> contacts = contactsName.get(name);
58          if(contacts == null) {
59              System.out.println("NOT FOUND");
60              return;
61          }
62          for (Contact contact : contacts) {
63              System.out.println(contact);
64          }
65      }
66  }
67
68  class Contact implements Comparable<Contact> {
69      String name;
70      String number;
71
72      public Contact(String name, String number) {
73          this.name = name;
74          this.number = number;
75      }
76
77      @Override
78      public String toString() {
79          return String.format("%s %s", name, number);
```

```
80      }
81
82      public List<String> getPhoneKeys() {
83          List<String> keys = new ArrayList<String>();
84          int len = number.length();
85          for (int i = 0; i <= len - 3; ++i) {
86              for (int j = i + 3; j <= len; ++j) {
87                  String key = number.substring(i, j);
88                  keys.add(key);
89              }
90          }
91          return keys;
92      }
93
94      @Override
95      public int compareTo(Contact o) {
96          if (name.equals(o.name)) {
97              return number.compareTo(o.number);
98          }
99          return name.compareTo(o.name);
100     }
101
102 }
103
104 class DuplicateNumberException extends Exception {
105     public DuplicateNumberException(String number) {
106         super(String.format("Duplicate number: %s", number));
107     }
108 }
109
110 public class PhoneBookTest {
111     public static void main(String[] args) {
112         PhoneBook phoneBook = new PhoneBook();
113         Scanner scanner = new Scanner(System.in);
114         int n = scanner.nextInt();
115         scanner.nextLine();
116         for (int i = 0; i < n; ++i) {
117             String line = scanner.nextLine();
118             String[] parts = line.split(":");
119             try {
120                 phoneBook.addContact(parts[0], parts[1]);
121             } catch (DuplicateNumberException e) {
122                 System.out.println(e.getMessage());
123             }
124         }
125         while (scanner.hasNextLine()) {
126             String line = scanner.nextLine();
127             System.out.println(line);
128             String[] parts = line.split(":");
129             if (parts[0].equals("NUM")) {
130                 phoneBook.contactsByNumber(parts[1]);
131             } else {
132                 phoneBook.contactsByName(parts[1]);
133             }
134         }
135     }
136 }
```

Проблем 4.15 Да се имплементира паркинг-системот на еден град. За таа цел треба да се имплементираат класите:

- `ParkingSector` во која се чуваат информации за:
  - кодот на секторот `String`
  - бројот на паркинг-места `int`
  - сите автомобили (регистрации) во овој сектор `?`
- `CityParking` во која се чуваат информации за:
  - името на градот `String`
  - и сите паркинг-сектори во тој град `?`

Во класата `CityParking` треба да се имплементираат следните методи:

- `CityParking(String name)` конструктор со аргумент име на градот
- `void createSectors(String[] sectorNames, int[] counts)` креирање на паркинг-сектори со имиња `String[] sectorNames` и број на паркинг-места `int[] counts` (двете низи се со иста големина)
- `void addCar(String sectorName, int spotNumber, String rNumber)` за додавање автомобил со регистрација `rNumber` на позиција `spotNumber` (притоа ако позицијата не е во опсегот од $1 \leq$ `spotNumber` $\leq$ број-на-паркинг-места. Треба да се фрли исклучок од тип `InvalidSpotNumberException`, ако позицијата е зафатена од некој друг автомобил тогаш се фрла исклучок од тип `SpotTakenException`. Ако не постои сектор со даденото име, се фрла исклучок од тип `NoSuchSectorException`)
- `void findCar(String registrationNumber)` за печатење на секторот и бројот на местото каде што е паркиран автомобилот со дадената регистрација (ако не се пронајде таков автомобил се фрла исклучок од тип `CarNotFoundException`) Забелешка: комплексноста на методот не треба да биде поголема од $O(logN)$
- `toString()` враќа стринг во формат:

```
Ime_na_grad

Kod_na_sektor : zafateni_mesta/parking_mesta  //za site sektori vo nov red
```

Во класата `ParkingSector` треба да се имплементираат следните методи:

- `ParkingSector(String code, int count)` конструктор со аргументи код на секторот и број на слободни места
- и останати потребни методи за имплементација на претходната класа...

■

Во класата `ParkingSector` автомобилите/регистрациите се чуваат во мапа (`spots`) во која клуч е редниот број на паркинг-местото (од 1(еден) до бројот на паркинг места), а вредност е регистрацијата на паркираното возило на тоа паркинг-место. Мапата е од тип `HashMap` што гарантира комплексност од $O(1)$ при проверка за пополнетост на паркинг-место. За проверка дали возило со дадена регистрација е паркирано во секторот, дополнително се користи инверзна мапа (`index`) во која клуч е регистрацијата на возилото, а вредност е редниот број на паркинг-местото).

Во класата `CityParking` , паркинг-секторите се чуваат во мапа (`sectors`) во која клуч е кодот на секторот, а вредност е објект од класата `ParkingSector` . Оваа мапа е од

тип `TreeMap` , со цел паровите (клуч, вредност) да бидат подредени според вредноста на клучот, односно според кодот на паркинг-секторот. Во оваа класа дополнително се чува и мапа (`index`) во која клуч е бројот на регистрацијата, а вредност е паркинг-секторот каде што е паркирано возилото со регистрација иста како клучот (објект од тип `ParkingSector`). Мапата `index` е од тип `HashMap` и истата гарантира комплексност од $O(1)$ за пристап до паркинг-секторот во кој е паркирано возилото.

Бидејќи комплексноста за пристап до паркинг секторот каде е паркирано некое возило е $O(1)$ и комплексноста за пристап до редниот број на паркинг местото во секторот каде е паркирано возилото е $O(1)$, вкупната комплексност на методот `findCar(String registrationNumber)` изнесува исто така $O(1)$.

```
import java.util.HashMap;
import java.util.Map;
import java.util.Map.Entry;
import java.util.Scanner;
import java.util.TreeMap;

class ParkingSector {
    private String code;
    private int count;
    private Map<Integer, String> spots;
    private Map<String, Integer> index;

    public ParkingSector(String code, int count) {
        this.code = code;
        this.count = count;
        this.spots = new HashMap<Integer, String>();
        this.index = new HashMap<String, Integer>();
    }

    public void addCar(int spotNumber, String registrationNumber)
            throws InvalidSpotNumberException, SpotTakenException {
        if (spotNumber <= 0 || spotNumber > count) {
            throw new InvalidSpotNumberException();
        }
        if (spots.containsKey(spotNumber)) {
            throw new SpotTakenException();
        }
        spots.put(spotNumber, registrationNumber);
        index.put(registrationNumber, spotNumber);
    }

    public int findSpotNumber(String registrationNumber)
            throws CarNotFoundException {
        if (index.containsKey(registrationNumber)) {
            return index.get(registrationNumber);
        }
        throw new CarNotFoundException(registrationNumber);
    }

    public void clearSpot(int spotNumber) {
        spots.remove(spotNumber);
    }

    public String getCode() {
        return this.code;
    }

    public Map<Integer, String> getSpots() {
```

```
49          return spots;
50      }
51
52      @Override
53      public String toString() {
54          return String.format("%s : %d/%d", code, spots.size(), count);
55      }
56
57  }
58
59  class InvalidSpotNumberException extends Exception {
60      public InvalidSpotNumberException() {
61          super("Invalid spot number");
62      }
63  }
64
65  class SpotTakenException extends Exception {
66      public SpotTakenException() {
67          super("Spot is taken already!");
68      }
69  }
70
71  class CityParking {
72      private String name;
73      private Map<String, ParkingSector> sectors;
74      private Map<String, ParkingSector> index;
75
76      public CityParking(String name) {
77          this.name = name;
78          this.sectors = new TreeMap<String, ParkingSector>();
79          this.index = new HashMap<String, ParkingSector>();
80      }
81
82      public void createSectors(String[] sectorNames, int[] counts) {
83          for (int i = 0; i < sectorNames.length; ++i) {
84              ParkingSector parkingSector =
85                      new ParkingSector(sectorNames[i],
86                              counts[i]);
87              sectors.put(parkingSector.getCode(), parkingSector);
88          }
89      }
90
91      public void addCar(String sectorName, int spotNumber,
92                         String registrationNumber)
93              throws InvalidSpotNumberException,
94              SpotTakenException, NoSuchSectorException {
95          ParkingSector parkingSector = sectors.get(sectorName);
96          if (parkingSector != null) {
97              parkingSector.addCar(spotNumber, registrationNumber);
98              index.put(registrationNumber, parkingSector);
99          } else {
100             throw new NoSuchSectorException(sectorName);
101         }
102     }
103
104     public void findCar(String registrationNumber)
105             throws CarNotFoundException {
106         if (index.containsKey(registrationNumber)) {
107             ParkingSector parkingSector =
108                     index.get(registrationNumber);
109             int spotNumber =
```

```
                    parkingSector.findSpotNumber(registrationNumber);
            System.out.println(String.format("%s : %d",
                    parkingSector.getCode(), spotNumber));
        } else {
            throw new CarNotFoundException(registrationNumber);
        }
    }

    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder();
        sb.append(name);
        sb.append("\n");
        for (Entry<String, ParkingSector> sector : sectors
                .entrySet()) {
            sb.append(sector.getValue().toString());
            sb.append("\n");
        }
        return sb.toString();
    }
}

class NoSuchSectorException extends Exception {
    public NoSuchSectorException(String sectorName) {
        super(String.format("No sector with name %s", sectorName));
    }
}

class CarNotFoundException extends Exception {
    public CarNotFoundException(String registrationNumber) {
        super(String.format("Car with RN %s not found",
                registrationNumber));
    }
}

public class ParkingTest {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt();
        scanner.nextLine();
        String[] sectorNames = new String[n];
        int[] counts = new int[n];
        for (int i = 0; i < n; ++i) {
            String[] parts = scanner.nextLine().split(" ");
            sectorNames[i] = parts[0];
            counts[i] = Integer.parseInt(parts[1]);
        }
        String name = scanner.nextLine();
        CityParking cityParking = new CityParking(name);
        cityParking.createSectors(sectorNames, counts);
        n = scanner.nextInt();
        scanner.nextLine();
        for (int i = 0; i < n; ++i) {
            String[] parts = scanner.nextLine().split(" ");
            String sectorName = parts[0];
            int spotNumber = Integer.parseInt(parts[1]);
            String registrationNumber = parts[2];
            try {
                cityParking.addCar(sectorName, spotNumber,
                        registrationNumber);
            } catch (InvalidSpotNumberException e) {
```

```
171                 System.out.println(e.getMessage());
172             } catch (SpotTakenException e) {
173                 System.out.println(e.getMessage());
174             } catch (NoSuchSectorException e) {
175                 System.out.println(e.getMessage());
176             }
177         }
178         n = scanner.nextInt();
179         scanner.nextLine();
180         for (int i = 0; i < n; ++i) {
181             String registrationNumber = scanner.nextLine();
182             try {
183                 cityParking.findCar(registrationNumber);
184             } catch (CarNotFoundException e) {
185                 System.out.println(e.getMessage());
186             }
187         }
188         System.out.println(cityParking);
189     }
190 }
```

5. Stream API

Апликацискиот програмски интерфејс за потоци (Stream API) е додаден во Јава 8 за да се олесни последователното и паралелното извршување на операциите. Овој интерфејс обезбедува две клучни апстракции: поток од податоци, што претставува конечна или бесконечна низа на елементи и поточни операции за процесирање на потокот од елементи. Како Потокот од елементи може да биде креиран на повеќе различни начини. Вообичаено потоците од елементи се креираат од колекции, низи, датотеки, генератори на псевдослучајни броеви и други. Елементите во потокот може да бидат инстанци на примитивни податочни типови или референцни типови. Од примитивните податочни типови поддржани се `int` , `long` и `double` . Процесирачките операции може да бидат класични и терминални. Класичните операции го трансформираат влезниот поток од елементи во друг поток од елементи од ист или различен тип (пропуштање на одредени елементи кои задоволуваат одреден услов или мапирање на елементи со помош на некоја функција). Терминалните операции се последните операции коишто се спроведуваат на потокот од елементи добиен како резултат од последната класична операција (агрегација на сите елементи од потокот во единствена вредност, односно колекција од елементи или печатење на сите елементи). Евалуацијата на класичните операции не започнувам додека не биде поставена терминална операција.

Проблем 5.1 Да се имплементира апликација за евидентирање на оценките на студентите на еден факултет. Студентите на факултетот може да бидат запишани на тригодишни или четиригодишни студии. Во текот на студиите, студентите имаат два семестри во секоја година и во секој од семестрите имаат по најмногу три предмети. За таа цел, дефинирајте класа `Faculty` во којашто ќе чувате информации за студентите и нивните оценки во сите семестри. За класата да се имплементираат:

- `Faculty()` - предефиниран конструктор
- метод `void addStudent(String id, int yearsOfStudies)` за додавање на студент на факултетот со индекс `ID` и години на студии `yearsOfStudies` .

- метод `void addGradeToStudent(String studentId, int term, String courseName, int grade)` - за додавање на оцена по предметот `courseName` на студентот со индекс `studentId` во семестар `term`.
- Со помош на исклучок од тип `OperationNotAllowedException` да се спречи додавање на повеќе од 3(три) оценки по семестар. Во таков случај да се испечати порака од формат:

  ```
  Student [studentID] already has 3 grades in term [term]
  ```

  Со истиот тип на исклучок да се спречи додавање на оценка во семестар поголем од 6(шест) за тригодишни студии, односно во семестар поголем од 8(осум) за четиригодишни студии. Во овој случај да се испечати порака:

  ```
  Term [term] is not possible for student with ID [studentId]
  ```

- Да се детектира дипломирање на студентот. Студентот дипломира тогаш кога ќе положи 18(осумнаесет) или 24(дваесет и четири) предмети во зависност од тоа колку години студира. Во моментот на дипломирање на студентот истиот треба да се избрише од евиденцијата и да се зачува лог за него во формат:

  ```
  Student with ID [studentID] graduated with average grade
  [averageGrade] in [yearsOfStudies] years
  ```

- Метод `String getFacultyLogs ()` - што ќе ги врати логовите за дипломираните студенти.
- Метод `String getDetailedReportForStudent (String id)` - метод што ќе врати детален извештај студентот со индекс id. Пристапот до студентот со индекс ИД да има комплексност `O(1)`! Деталниот извештај е во формат:

  ```
  Student: [id]
  Term 1:
  Courses for term: [count]
  Average grade for term: [average]
  ...
  .....
  Term n:
  Courses: [count]
  Average grade for term: [average]
  Average grade: [average grade for student]
  Courses attended: [all-attended-courses, comma-separated]
  ```

- Метод `void printFirstNStudents (int n)` - метод којшто ќе испечати краток извештај за најдобрите `n` студенти (според бројот на положени предмети, а доколку е ист бројот на положени предмети според просечната оцена), подредени во опаѓачки редослед. Краткиот извештај треба да го има следниот формат:

  ```
  Student: [id] Courses passed: [coursesPassed] Average grade:
  [averageGrade]
  ```

- Метод `void printCourses ()` - метод којшто ќе ги испечати сите предмети во следниот формат:

  `[course_name] [count_of_students] [average_grade]}`

  Предметите треба да бидат подредени според бројот на слушатели, а доколку тој број е ист, според просечната оцена.

Во решението на оваа задача потребно е да се употреби наследување и полиморфизам со цел да се имплементира барањето за постоење на студент од тригодишни или четиригодишни студии (дадена е дефиниција на класите `StudentOnThreeYearStudies` и `StudentOnFourYearStudies`). За да се овозможи групирање на оценките по семестар, како и подредување на семестрите според редниот број во растечки редослед, оценките на секој студент се чуваат во колекција `TreeMap` . Таму клучот е цел број што означува реден број на семестарот, а вредноста е листа од цели броеви што ги претставува сите оцени на студентот во соодветниот семестар. За да се овозможи лексикографско подредување на предметите што секој студент ги има слушано, имињата на предметите се чуваат во подредено множество `TreeSet` .

Во продолжение се објаснети дел од методите со посебен акцент на `stream` операторите:

- Методот `double averageGrade()` во класата `Student` - Во овој метод прво се креира поток од вредностите од мапата со оцени по семестри `gradesByTerm` . Потоа се користи операторот `flatMap(Collections::stream)` за да се постигне израмнување на потокот од листи од цели броеви во поток од цели броеви. Како резултат од овие два оператори се добива `Stream<Integer>` . За да се искористи постоечкиот оператор за пресметување на средна вредност `average()` потребно е потокот од податоци да биде конвертиран во `IntStream` со помош на методот `mapToInt` .
- Методот `double averageGrade()` во класата `Student` - Во овој метод се користи аналоген пристап како и во претходниот метод со единствена разлика што тука се земаат само оценките за конкретен семестар од мапата `gradesByTerm` . На тој начин директно се добива поток од цели броеви и нема потреба од повикување на операторот `flatMap` за израмнување на потокот.
- Методот `printFirstNStudents(int n)` во класата `Faculty` - Во овој метод прво се креира компаратор за студенти што се користи за подредување на студентите. Потоа, сите студенти се додаваат во подередено множество `TreeSet` со цел студентите да се чуваат подредени според дефинираниот компаратор. Од подреденото множество се креира поток на студенти и од истиот со помош на операторот `limit` се земаат првите `N` . Со операторот `forEach` студентите кои се добиваат како резултат од претходната операција, се печатат во нов ред.
- Методот `printCourses()` во класата `Faculty` - Овој метод ја следи истата парадигма како и претходниот метод (`printFirstNStudents(int n)`). Прво се креира компаратор за споредување и подредување на курсевите во резултантното множество. Потоа, со операторот `forEach` секој курс директно се печати во нов ред.

```
1 import java.util.*;
2 import java.util.function.Function;
3 import java.util.stream.Collectors;
```

```
4  import java.util.stream.IntStream;
5
6  class OperationNotAllowedException extends Exception {
7      OperationNotAllowedException(String message) {
8          super(message);
9      }
10 }
11
12 abstract class Student {
13     String id;
14     Map<Integer, List<Integer>> gradesByTerm;
15     Set<String> courses;
16
17     public Student(String id) {
18         this.id = id;
19         gradesByTerm = new TreeMap<>();
20         courses = new TreeSet<>();
21     }
22
23     String getGraduationLog() {
24         return String
25                 .format("Student with ID %s graduated with average " +
26                         "grade %.2f", id, averageGrade());
27     }
28
29     double averageGrade() {
30         return gradesByTerm.values().stream()
31                 .flatMap(Collection::stream)
32                 .mapToInt(i -> i)
33                 .average()
34                 .orElse(5.0);
35     }
36
37     double averageGradeForTerm(int term) {
38         return gradesByTerm.get(term).stream()
39                 .mapToInt(i -> i)
40                 .average()
41                 .orElse(5.0);
42     }
43
44     abstract boolean addGrade(int term, String courseName, int grade)
45             throws OperationNotAllowedException;
46
47     void validate(int term) throws OperationNotAllowedException {
48         if (!gradesByTerm.containsKey(term))
49             throw new OperationNotAllowedException(String.format(
50                     "Term %d is not possible for student with ID %s",
51                     term,
52                     id));
53         if (gradesByTerm.get(term).size() == 3)
54             throw new OperationNotAllowedException(String.format(
55                     "Student %s already has 3 grades in term %d", id,
56                     term));
57     }
58
59     int countOfCoursesPassed() {
60         return gradesByTerm.values().stream()
61                 .mapToInt(List::size)
62                 .sum();
63     }
64
```

```
65      public String getDetailedReport() {
66          StringBuilder sb = new StringBuilder();
67          sb.append(String.format("Student: %s\n", id));
68          gradesByTerm.keySet().forEach(
69                  term -> sb.append(getTermReport(term)).append("\n"));
70          sb.append(
71                  String.format(
72                          "Average grade: %.2f\nCourses attended: %s",
73                          averageGrade(),
74                          String.join(",", courses)
75                  ));
76          return sb.toString();
77      }
78
79      public String getShortReport() {
80          return String
81                  .format("Student: %s Courses passed: " +
82                                  "%d Average grade: %.2f",
83                          id,
84                          countOfCoursesPassed(),
85                          averageGrade());
86      }
87
88      String getTermReport(int term) {
89          return String
90                  .format("Term %d\nCourses: " +
91                                  "%d\nAverage grade for term: %.2f",
92                          term,
93                          gradesByTerm.get(term).size(),
94                          averageGradeForTerm(term)
95                  );
96      }
97
98      String getId() {
99          return id;
100     }
101 }
102
103 class StudentOnThreeYearsStudies extends Student {
104
105     public StudentOnThreeYearsStudies(String id) {
106         super(id);
107         IntStream.range(1, 7).forEach(
108                 i -> gradesByTerm.putIfAbsent(i, new ArrayList<>()));
109     }
110
111     @Override
112     boolean addGrade(int term, String courseName, int grade)
113             throws OperationNotAllowedException {
114         validate(term);
115         gradesByTerm.get(term).add(grade);
116         courses.add(courseName);
117         return countOfCoursesPassed() == 18;
118     }
119
120     @Override
121     String getGraduationLog() {
122         return super.getGraduationLog() + " in 3 years.";
123     }
124 }
125
```

```
126 class StudentOnFourYearsStudies extends Student {
127
128     public StudentOnFourYearsStudies(String id) {
129         super(id);
130         IntStream.range(1, 9).forEach(
131                 i -> gradesByTerm.putIfAbsent(i, new ArrayList<>()));
132     }
133
134     @Override
135     boolean addGrade(int term, String courseName, int grade)
136             throws OperationNotAllowedException {
137         validate(term);
138         gradesByTerm.get(term).add(grade);
139         courses.add(courseName);
140         return countOfCoursesPassed() == 24;
141     }
142
143     @Override
144     String getGraduationLog() {
145         return super.getGraduationLog() + " in 4 years.";
146     }
147 }
148
149 class Course {
150     String courseName;
151     IntSummaryStatistics statistics;
152
153     public Course(String courseName) {
154         this.courseName = courseName;
155         statistics = new IntSummaryStatistics();
156     }
157
158     void addGrade(int grade) {
159         statistics.accept(grade);
160     }
161
162     @Override
163     public String toString() {
164         return String
165                 .format("%s %d %.2f", courseName,
166                         statistics.getCount(),
167                         statistics.getAverage());
168     }
169
170     int getStudentsCount() {
171         return (int) statistics.getCount();
172     }
173
174     double getCourseAverageGrade() {
175         return statistics.getAverage();
176     }
177
178     public String getCourseName() {
179         return courseName;
180     }
181 }
182
183 class Faculty {
184     Map<String, Student> studentsById;
185     Map<String, Course> coursesByName;
186     StringBuilder logs;
```

```
187
188     public Faculty() {
189         studentsById = new HashMap<>();
190         coursesByName = new HashMap<>();
191         logs = new StringBuilder();
192     }
193
194     void addStudent(String id, int yearsOfStudies) {
195         studentsById.put(id,
196                 yearsOfStudies == 3 ?
197                         new StudentOnThreeYearsStudies(id) :
198                         new StudentOnFourYearsStudies(id));
199     }
200
201     void addGradeToStudent(String studentId, int term,
202                            String courseName,
203                            int grade)
204             throws OperationNotAllowedException {
205         Student student = studentsById.get(studentId);
206         boolean graduated = student.addGrade(term, courseName, grade);
207         coursesByName.putIfAbsent(courseName, new Course(courseName));
208         coursesByName.get(courseName).addGrade(grade);
209         if (graduated) {
210             logs.append(student.getGraduationLog()).append("\n");
211             studentsById.remove(studentId);
212         }
213     }
214
215     String getFacultyLogs() {
216         return logs.deleteCharAt(logs.length() - 1).toString();
217     }
218
219     String getDetailedReportForStudent(String id) {
220         return studentsById.get(id).getDetailedReport();
221     }
222
223     void printFirstNStudents(int n) {
224         Comparator<Student> studentComparator =
225                 Comparator.comparing(Student::countOfCoursesPassed)
226                         .thenComparing(Student::averageGrade)
227                         .thenComparing(Student::getId).reversed();
228         TreeSet<Student> students = new TreeSet<>(studentComparator);
229         students.addAll(studentsById.values());
230         students.stream()
231                 .limit(n)
232                 .forEach(student -> System.out
233                         .println(student.getShortReport()));
234     }
235
236     void printCourses() {
237         Comparator<Course> courseComparator =
238                 Comparator.comparing(Course::getStudentsCount)
239                         .thenComparing(Course::getCourseAverageGrade)
240                         .thenComparing(Course::getCourseName);
241         TreeSet<Course> coursesSet = new TreeSet<>(courseComparator);
242         coursesSet.addAll(coursesByName.values());
243         coursesSet.forEach(System.out::println);
244     }
245 }
246
247 public class FacultyTest {
```

```
248
249     public static void main(String[] args) {
250         Scanner sc = new Scanner(System.in);
251         Faculty faculty = new Faculty();
252         for (int i = 1; i <= 10; i++) {
253             faculty.addStudent("student" + i,
254                     ((i % 2) == 1 ? 3 : 4));
255             int courseCounter = 1;
256             for (int j = 1; j <= ((i % 2 == 1) ? 6 : 8); j++) {
257                 for (int k = 1; k <= ((j % 2 == 1) ? 2 : 3);
258                      k++) {
259                     int grade = sc.nextInt();
260                     try {
261                         faculty.addGradeToStudent("student" + i,
262                                 j,
263                                 ("course" + courseCounter),
264                                 grade);
265                     } catch (OperationNotAllowedException e) {
266                         System.out.println(e.getMessage());
267                     }
268                     ++courseCounter;
269                 }
270             }
271
272         }
273
274         for (int i = 11; i < 15; i++) {
275             faculty.addStudent("student" + i,
276                     ((i % 2) == 1 ? 3 : 4));
277             int courseCounter = 1;
278             for (int j = 1; j <= ((i % 2 == 1) ? 6 : 8); j++) {
279                 for (int k = 1; k <= 3; k++) {
280                     int grade = sc.nextInt();
281                     try {
282                         faculty.addGradeToStudent("student" + i,
283                                 j,
284                                 ("course" + courseCounter),
285                                 grade);
286                     } catch (OperationNotAllowedException e) {
287                         System.out.println(e.getMessage());
288                     }
289                     ++courseCounter;
290                 }
291             }
292         }
293         System.out.println("LOGS");
294         System.out.println(faculty.getFacultyLogs());
295         System.out.println("DETAILED REPORT FOR STUDENT");
296         System.out.println(
297                 faculty.getDetailedReportForStudent("student2"));
298         try {
299             System.out.println(
300                     faculty.getDetailedReportForStudent(
301                             "student11"));
302             System.out.println(
303                     "The graduated students should be deleted!!!");
304         } catch (NullPointerException e) {
305             System.out.println(
306                     "The graduated students are really deleted");
307         }
308         System.out.println("FIRST N STUDENTS");
```

```
309         faculty.printFirstNStudents(10);
310         System.out.println("COURSES");
311         faculty.printCourses();
312     }
313 }
```

Проблем 5.2 Да се имплементира систем за чување на пораки според определени теми. За секоја порака во системот се чува:

- временски печат (`timestamp`), објект од класата `LocalDateTime` - содржина на пораката `String` - на која партиција треба да биде зачувана пораката (`Integer` којшто може да биде и null). - клуч на пораката (`String`)

Системот за чување на овие пораки функционира на следниов начин:

- Пораките се чуваат во рамки на еден `Broker`. Во брокерот може да има повеќе теми (`topics`).
- Во секоја тема (`Topic`) има одреден број на партиции каде што се чуваат пораки. Пораките се чуваат сортирани според датумот на креирање. На секоја од партициите има исто ограничување за тоа кои пораки може да се чуваат, и тоа:
- може да се чуваат максимум `limit` пораки. Доколку пристигне порака кога капацитетот е исполнет, се брише најстарата порака, а се додава новата.
- не може да се чуваат пораки постари од `startDate`.
- На секоја тема може да му биде зголемен бројот на партиции, но не смее да биде намален!

За да се имплементира системот потребно е да се имплементираат следните класи со соодветни функционалности:

- `Message` - класа за репрезентација на порака,
- `Topic` - класа за чување на пораки по теми. За секоја тема се чува името на темата, како и колекција од сите пораки распределени по партиции. Оваа класа треба да ги има следните методи:
  - `Topic(String topicName, int partitionsCount)` конструктор
  - `void addMessage(Message message) throws PartitionDoesNotExistException` - метод за додавање на нова порака во оваа тема. Додавањето се прави во соодветна партиција којашто се чува како информација во пораката `message`. Доколку таму не е специфицирана, треба да се користи методот `assignPartition` од класата `PartitionAssigner`. Доколку не постои партицијата којашто е наведена во пораката, да се фрли исклучок од тип `PartitionDoesNotExistException`.
  - `void changeNumberOfPartitions(int newPartitionsNumber) throws UnsupportedOperationException` - метод за промена на бројот на партиции. Доколку се проба да се намали бројот на партиции да се фрли исклучок како во потписот на функцијата.
  - `String toString()` - метод којшто ќе дава `toString` репрезентација на темата така што за него ќе се испечати неговото име, бројот на партиции и содржината на соодветните партиции. Партициите да се сортираат според реден број.
- `MessageBroker` - класа за чување на повеќе теми. Дополнително во оваа класа се чуваат и стартниот датум, како и максималниот капацитет на секоја

партиција за сите теми во овој брокер. Да се имплементираат следните методи:
- `MessageBroker(LocalDateTime minimumDate, Integer capacityPerTopic)` - конструктор
- `void addTopic (String topic, int partitionsCount)` - метод за додавање на нова тема со одреден број на партиции во неа. Не смее да се додаде тема со исто име.
- `void addMessage (String topic, Message message)` - метод за додавање на порака на определена тема.
- `void changeTopicSettings (String topic, int partitionsCount)` - метод за промена на бројот на партиции на определена тема.
- `String toString()` - метод што ќе дава текстуална репрезентација на брокерот. Најпрво ќе се испечати колку теми има, а потоа за сите теми ќе се даде нивната текстуална репрезентација. Темите да се подредени според нивното име.

Во решението на оваа задача пораките се чуваат во различни партиции на различни теми. За да се овозможи групирање на пораките според партиција во рамките на една тема, како и подредување на партициите според нивниот реден број, пораките се чуваат во мапа (`TreeMap`) каде клучот претставува бројот на партицијата, а вредноста претставува подредено множество од пораки `TreeSet` . Вредноста на мапата е од тип `TreeSet` за да се овозможи подредување на пораките според нивниот временски печат. Сите теми во брокерот се чуваат во една мапа. Клучот е текстуална низа, којашто го означува името на темата, а вредноста е објект од класа `Topic` .

Во оваа задача на неколку места се користени `stream` оператори и тоа:

- Конструкторот `Topic(String topicName, int partitionsCount)`)
  - Во конструкторот се користи статичкиот метод `range(start,end)` од класата `IntStream` за да се генерира поток од цели броеви во опсег од 1(еден) до бројот на партиции. Потоа секој број од овој поток со помош на операторот `forEach` се користи за да се додаде празна колекција од пораки за соодветните партиции.
- Методот `toString()` во класата `Topic` - Во овој метод покрај користењето на `stream` оператори, дополнително се користат вгнездени `stream` оператори (на елемент од потокот на податоци). Прво, се креира поток од `Map.Entry< Integer, TreeSet<Message> >` , каде што секој елемент од потокот претставува една пар-тиција со сите нејзини пораки. Секој од паровите од мапата со помош на операторот `map` прво се трансформира во текстуална низа, а потоа со помош на операторот `collect` , текстуалните низи се спојуваат во една текстуална низа разделени со празен ред меѓу нив. Во рамките на `map` операторот се користат вгнездени `stream` оператори. Прво, се креира поток од пораките во партицијата, истите се мапираат во нивната текстуална репрезентација, а потоа со користење на `collect)` , се спојуваат со нов ред помеѓу нив.

```
1 import java.time.LocalDateTime;
2 import java.util.*;
3 import java.util.stream.Collectors;
4 import java.util.stream.IntStream;
5
6 class PartitionDoesNotExistException extends Exception {
7     PartitionDoesNotExistException(String topic, int partition) {
8         super(String.format("The topic %s does not have " +
```

```
                "a partition with number %d", topic, partition));
    }
}

class UnsupportedOperationException extends Exception {

    public UnsupportedOperationException(String s) {
        super(s);
    }
}

class Message implements Comparable<Message> {
    LocalDateTime timestamp;
    String message;
    Integer partition;
    String key;

    public Message(LocalDateTime timestamp, String message,
                   Integer partition, String key) {
        this.timestamp = timestamp;
        this.message = message;
        this.partition = partition;
        this.key = key;
    }

    public Message(LocalDateTime timestamp, String message,
                   String key) {
        this.timestamp = timestamp;
        this.message = message;
        this.key = key;
    }


    @Override
    public int compareTo(Message message) {
        return this.timestamp.compareTo(message.timestamp);
    }

    @Override
    public String toString() {
        final StringBuilder sb = new StringBuilder("Message{");
        sb.append("timestamp=").append(timestamp);
        sb.append(", message='").append(message).append('\'');
        sb.append('}');
        return sb.toString();
    }
}

class Topic {
    String topicName;
    Map<Integer, TreeSet<Message>> messagesByPartition;
    int partitionsCount;

    public Topic(String topicName, int partitionsCount) {
        this.topicName = topicName;
        messagesByPartition = new TreeMap<>();
        this.partitionsCount = partitionsCount;
        IntStream.range(1, partitionsCount + 1).forEach(
                i -> messagesByPartition.put(i, new TreeSet<>()));
    }

```

```
70      void addMessage(Message message)
71              throws PartitionDoesNotExistException {
72          Integer messagePartition = message.partition;
73          if (messagePartition == null) {
74              messagePartition = (Math.abs(message.key.hashCode()) %
75                      this.partitionsCount) + 1;
76          }
77
78          if (!messagesByPartition.containsKey(messagePartition))
79              throw new PartitionDoesNotExistException(topicName,
80                      messagePartition);
81
82          messagesByPartition
83                  .computeIfPresent(messagePartition, (k, v) -> {
84                      if (v.size() == MessageBroker.capacityPerTopic)
85                          v.remove(v.first());
86                      v.add(message);
87                      return v;
88                  });
89      }
90
91      void changeNumberOfPartitions(int newPartitionsNumber)
92              throws UnsupportedOperationException {
93          if (newPartitionsNumber < this.partitionsCount)
94              throw new UnsupportedOperationException(
95                      "Partitions number cannot be decreased!");
96
97          else {
98              int diff = newPartitionsNumber - this.partitionsCount;
99              int size = this.messagesByPartition.size();
100             for (int i = 1; i <= diff; i++)
101                 this.messagesByPartition
102                         .putIfAbsent(size + i, new TreeSet<>());
103             this.partitionsCount = newPartitionsNumber;
104         }
105     }
106
107     public String toString() {
108         return String.format("Topic: %10s Partitions: %5d\n%s",
109             topicName,
110             partitionsCount,
111             messagesByPartition.entrySet().stream()
112                 .map(entry -> String
113                     .format("%2d : Count of messages: %5d\n%s",
114                         entry.getKey(),
115                         entry.getValue().size(),
116                         !entry.getValue().isEmpty() ?
117                             "Messages:\n" +
118                                     entry.getValue()
119                                             .stream()
120                                             .map(Message::toString)
121                                             .collect(
122                                                Collectors
123                                                    .joining("\n")) :
124                             "")
125                 ).collect(Collectors.joining("\n"))
126         );
127     }
128 }
129
130 class MessageBroker {
```

```
131     Map<String, Topic> topicMap;
132     static LocalDateTime minimumDate;
133     static Integer capacityPerTopic;
134 
135     public MessageBroker(LocalDateTime minimumDate,
136                          Integer capacityPerTopic) {
137         topicMap = new TreeMap<>();
138         MessageBroker.minimumDate = minimumDate;
139         MessageBroker.capacityPerTopic = capacityPerTopic;
140 
141     }
142 
143     public void addTopic(String topic, int partitionsCount) {
144         topicMap.put(topic, new Topic(topic, partitionsCount));
145     }
146 
147     public void addMessage(String topic, Message message)
148             throws UnsupportedOperationException,
149             PartitionDoesNotExistException {
150         if (message.timestamp.isBefore(minimumDate))
151             return;
152 
153         topicMap.get(topic).addMessage(message);
154     }
155 
156     public void changeTopicSettings(String topic, int partitionsCount)
157             throws UnsupportedOperationException {
158         topicMap.get(topic).changeNumberOfPartitions(partitionsCount);
159     }
160 
161     public String toString() {
162         return String.format("Broker with %2d topics:\n%s",
163                 topicMap.size(),
164                 topicMap.values().stream().map(Topic::toString)
165                         .collect(Collectors.joining("\n"))
166         );
167     }
168 }
169 
170 public class MessageBrokersTest {
171 
172     public static void main(String[] args) {
173         Scanner sc = new Scanner(System.in);
174 
175         String date = sc.nextLine();
176         LocalDateTime localDateTime = LocalDateTime.parse(date);
177         Integer partitionsLimit = Integer.parseInt(sc.nextLine());
178         MessageBroker broker =
179                 new MessageBroker(localDateTime, partitionsLimit);
180         int topicsCount = Integer.parseInt(sc.nextLine());
181 
182         //Adding topics
183         for (int i = 0; i < topicsCount; i++) {
184             String line = sc.nextLine();
185             String[] parts = line.split(";");
186             String topicName = parts[0];
187             int partitionsCount = Integer.parseInt(parts[1]);
188             broker.addTopic(topicName, partitionsCount);
189         }
190 
191         //Reading messages
```

```
192         int messagesCount = Integer.parseInt(sc.nextLine());
193
194         System.out.println("===ADDING MESSAGES TO TOPICS===");
195         for (int i = 0; i < messagesCount; i++) {
196             String line = sc.nextLine();
197             String[] parts = line.split(";");
198             String topic = parts[0];
199             LocalDateTime timestamp = LocalDateTime.parse(parts[1]);
200             String message = parts[2];
201             if (parts.length == 4) {
202                 String key = parts[3];
203                 try {
204                     broker.addMessage(topic,
205                             new Message(timestamp, message, key));
206                 } catch (UnsupportedOperationException
207                         | PartitionDoesNotExistException e) {
208                     System.out.println(e.getMessage());
209                 }
210             } else {
211                 Integer partition = Integer.parseInt(parts[3]);
212                 String key = parts[4];
213                 try {
214                     broker.addMessage(topic,
215                             new Message(timestamp, message, partition,
216                                     key));
217                 } catch (UnsupportedOperationException
218                         | PartitionDoesNotExistException e) {
219                     System.out.println(e.getMessage());
220                 }
221             }
222         }
223
224         System.out.println(
225                 "===BROKER STATE AFTER ADDITION OF MESSAGES===");
226         System.out.println(broker);
227
228         System.out.println("===CHANGE OF TOPICS CONFIGURATION===");
229         //topics changes
230         int changesCount = Integer.parseInt(sc.nextLine());
231         for (int i = 0; i < changesCount; i++) {
232             String line = sc.nextLine();
233             String[] parts = line.split(";");
234             String topicName = parts[0];
235             Integer partitions = Integer.parseInt(parts[1]);
236             try {
237                 broker.changeTopicSettings(topicName, partitions);
238             } catch (UnsupportedOperationException e) {
239                 System.out.println(e.getMessage());
240             }
241         }
242
243         System.out.println("===ADDING NEW MESSAGES TO TOPICS===");
244         messagesCount = Integer.parseInt(sc.nextLine());
245         for (int i = 0; i < messagesCount; i++) {
246             String line = sc.nextLine();
247             String[] parts = line.split(";");
248             String topic = parts[0];
249             LocalDateTime timestamp = LocalDateTime.parse(parts[1]);
250             String message = parts[2];
251             if (parts.length == 4) {
252                 String key = parts[3];
```

```
253                 try {
254                     broker.addMessage(topic,
255                             new Message(timestamp, message, key));
256                 } catch (UnsupportedOperationException
257                         | PartitionDoesNotExistException e) {
258                     System.out.println(e.getMessage());
259                 }
260             } else {
261                 Integer partition = Integer.parseInt(parts[3]);
262                 String key = parts[4];
263                 try {
264                     broker.addMessage(topic,
265                             new Message(timestamp, message, partition,
266                                     key));
267                 } catch (UnsupportedOperationException
268                         | PartitionDoesNotExistException e) {
269                     System.out.println(e.getMessage());
270                 }
271             }
272         }
273
274         System.out.println(
275                 "===BROKER STATE AFTER CONFIGURATION CHANGE===");
276         System.out.println(broker);
277
278
279     }
280 }
```

Проблем 5.3 Да се напише класа `Canvas` во којашто ќе се чува колекција од форми од различен тип. Секоја форма треба да може да се добијат информации за ИД на корисникот што ја додал во колекцијата, колкава плоштина и периметар има, како и да се овозможи формата да биде скалирана за некој коефициент. Во класата `Canvas` да се имплементираат:

- предефиниран конструктор
- `void readShapes (InputStream is)` - метод за читање на информации за формите од влезен поток.
  - Информациите за секоја форма се дадени во секој ред. При читање на формите прво се чита број (1 = круг/2 = квадрат/3 = правоаголник), па потоа се чита ИД-то на корисникот којшто ја креирал формата. Па, потоа доколку станува збор за круг/квадрат се чита број за радиусот/страна-та на кругот/квадартот, а доколку е правоаголник се читаат два броја за должина и висина на правоаголнкот.
  - ИД на корисникот мора да биде стринг со должина од 6(шест) знаци, при што не се дозволени специјални знаци (само букви и бројки). Доколку некој ИД не е во ред, да се фрли исклучок од тип `InvalidIDException` при креирањето на формата, а со истиот справете се во рамки на функцијата readShapes. Односно, неправилно ИД да не повлече прекинување на читањето на формите.
  - Димензија на форма не смее да биде 0(нула). Во таков случај да се фрли исклучок од тип `InvalidDimensionException`. Овој исклучок треба да го прекине понатамошното читање на останатите форми.
- `void scaleShapes (String userID, double coef)` - метод што ќе ги скалира

сите форми креирани од корисникот `userID` со коефициентот `coef` (ќе ги помножи сите димензии на формата со тој коефициент).

- `void printAllShapes (OutputStream os, boolean ascending)` - метод што ќе ги испечати формите на излезен поток сортирани според нивната плоштина во растечки/опаѓачки редослед според вредноста на променливата `ascending`.
- `void printByUserId (OutputStream os)` - метод што ќе ги испечати формите групирани според корисникот, при што корисниците се подредени според бројот на форми што ги имаат креирани (доколку тој број е ист, тогаш според сумата на плоштините на формите). Формите на даден корисник треба да се подредени според периметарот во опаѓачки редослед.
- `void statistics (OutputStream os)` - метод што ќе испечати статистики за плоштините на сите форми во колекцијата (min, max, average, sum, count).

Бидејќи постојат три различни типови на форми во здачата, прво се дефинира `IShape` интерфејсот во кој се декларирани три методи за основните функционалности на која било форма. Потоа, се дефинира апстрактната класа `Shape` која го имплементира интерфејсот `IShape` и во неа се чуваат сите заеднички полиња на различните форми. Од оваа класа се изведени класите `Circle`, `Square` и `Rectangle` за соодветните типови на форми. Формите се чуваат како листа од објекти од тип `IShape` во рамки на класата `Canvas`.

Во класата `Canvas`, `stream` операторите се употребени во:

- Методот `scaleShapes(String userId, double coef)` - Прво, се креира поток од сите форми и истите се филтрираат според наметнатиот критериум (вредноста на променливата `ID` е еднаква на вредноста на првиот аргумент на методот. Потоа со операторот `forEach`, секоја од пропуштените форми се скалира со коефициентот даден како втор аргумент на овој метод.
- Методот `printByUserId(OutputStream os)` - Во овој метод со помош на колекторот `groupingBy`, прво се креира мапа во која клуч е `ID`-то на корисникот, а вредноста на мапата е множество (`TreeSet`) од сите форми креирани од страна на тој корисник. Потоа, сите форми во согласност со креираниот компаратор се подредуваат и со помош на операторот `foreach` се печатат потребните информации за секоја форма.
- Методот `statistics(OutputStream os)` - Со помош на операторот `mapToDouble`, потокот на форми во овој метод се конвертира во поток од децимални броеви (секој децимален број ја претставува плоштината на соодветната форма). На резултантниот поток, со помош на терминалниот оператор `summaryStatistics` се добива информација за минимумот, максимумот, просекот, сумата и бројот на децималните броеви (запишани во објект од класата `DoubleSummaryStatistcs`).

```
1  import java.io.*;
2  import java.util.*;
3  import java.util.stream.Collectors;
4
5  interface IShape {
6      String getID();
7
8      double getArea();
9
10     double getPerimeter();
11
```

```
12      void scale(double coef);
13  }
14
15  class InvalidDimensionException extends Exception {
16      InvalidDimensionException() {
17          super("Dimension 0 is not allowed!");
18      }
19  }
20
21  class InvalidIDException extends Exception {
22
23      public InvalidIDException(String id) {
24          super(String.format("ID %s is not valid", id));
25      }
26  }
27
28  abstract class Shape implements IShape {
29      String id;
30
31      public Shape(String id) {
32          this.id = id;
33      }
34
35      @Override
36      public String getID() {
37          return id;
38      }
39
40      @Override
41      public String toString() {
42          return id;
43      }
44  }
45
46  class Circle extends Shape {
47
48      private double radius;
49
50      public Circle(String id, double radius) {
51          super(id);
52          this.radius = radius;
53      }
54
55      @Override
56      public double getArea() {
57          return Math.pow(radius, 2) * Math.PI;
58      }
59
60      @Override
61      public double getPerimeter() {
62          return 2 * radius * Math.PI;
63      }
64
65      @Override
66      public void scale(double coef) {
67          radius *= coef;
68      }
69
70      @Override
71      public String toString() {
72          return String
```

```
                .format("Circle: %s Radius: %.2f Area: " +
                                "%.2f Perimeter: %.2f",
                        id, radius, getArea(), getPerimeter());
    }
}

class Square extends Shape {

    double a;

    public Square(String id, double a) {
        super(id);
        this.a = a;
    }

    @Override
    public double getArea() {
        return Math.pow(a, 2);
    }

    @Override
    public double getPerimeter() {
        return 4 * a;
    }

    @Override
    public void scale(double coef) {
        a *= coef;
    }

    @Override
    public String toString() {
        return String
                .format("Square: %s Side: %.2f Area: " +
                                "%.2f Perimeter: %.2f",
                        id, a, getArea(), getPerimeter());
    }
}

class Rectangle extends Square {

    double b;

    public Rectangle(String id, double a, double b) {
        super(id, a);
        this.b = b;
    }

    @Override
    public double getArea() {
        return a * b;
    }

    @verride
    public double getPerimeter() {
        return 2 * a + 2 * b;
    }

    @Override
    public void scale(double coef) {
        super.scale(coef);
```

```
134         b *= coef;
135     }
136 
137     @Override
138     public String toString() {
139         return String
140                 .format("Rectangle: %s Sides: %.2f, %.2f Area: " +
141                                 "%.2f Perimeter: %.2f",
142                         id, a, b, getArea(), getPerimeter());
143     }
144 }
145 
146 class ShapeFactory {
147 
148     private static boolean checkId(String id) {
149         if (id.length() != 6)
150             return false;
151 
152         for (char c : id.toCharArray()) {
153             if (!Character.isLetterOrDigit(c))
154                 return false;
155         }
156 
157         return true;
158     }
159 
160     public static IShape createShape(String line)
161             throws InvalidDimensionException, InvalidIDException {
162         String[] parts = line.split("\\s+");
163         int type = Integer.parseInt(parts[0]);
164         String id = parts[1];
165         if (!checkId(id))
166             throw new InvalidIDException(id);
167         double firstDimension = Double.parseDouble(parts[2]);
168         if (firstDimension == 0.0)
169             throw new InvalidDimensionException();
170 
171         if (type == 1) {
172             return new Circle(id, firstDimension);
173         } else if (type == 2) {
174             return new Square(id, firstDimension);
175         } else {
176             double secondDimension = Double.parseDouble(parts[3]);
177             if (secondDimension == 0.0)
178                 throw new InvalidDimensionException();
179             return new Rectangle(id, firstDimension, secondDimension);
180         }
181     }
182 }
183 
184 class Canvas {
185     List<IShape> shapes;
186 
187     public Canvas() {
188         shapes = new ArrayList<>();
189     }
190 
191     public void readShapes(InputStream is)
192             throws InvalidDimensionException {
193         Scanner sc = new Scanner(is);
194         while (sc.hasNextLine()) {
```

```
195             String line = sc.nextLine();
196             try {
197                 shapes.add(ShapeFactory.createShape(line));
198             } catch (InvalidIDException e) {
199                 System.out.println(e.getMessage());
200             }
201         }
202     }
203
204     public void scaleShapes(String userID, double coef) {
205         shapes.stream().filter(shape -> shape.getID().equals(userID))
206                 .forEach(shape -> shape.scale(coef));
207     }
208
209     public void printAllShapes(OutputStream os, boolean ascending) {
210         Comparator<IShape> shapeComparator =
211                 Comparator.comparing(IShape::getArea)
212                         .thenComparing(IShape::getID);
213         if (!ascending)
214             shapeComparator = shapeComparator.reversed();
215
216         PrintWriter pw = new PrintWriter(os);
217         shapes.stream()
218                 .sorted(shapeComparator)
219                 .forEach(pw::println);
220         pw.flush();
221     }
222
223     public void printByUserId(OutputStream os) {
224         PrintWriter pw = new PrintWriter(os);
225         Comparator<IShape> shapeComparator =
226                 Comparator.comparing(IShape::getPerimeter)
227                         .thenComparing(IShape::getID).reversed();
228         Map<String, Set<IShape>> result = shapes.stream()
229                 .collect(Collectors.groupingBy(
230                         IShape::getID,
231                         Collectors.toCollection(
232                                 () -> new TreeSet<>(shapeComparator))
233                 ));
234
235         Comparator<Map.Entry<String, Set<IShape>>> entryComparator =
236                 Comparator
237                         .comparing(entry -> entry.getValue().size());
238
239         result.entrySet().stream()
240                 .sorted(entryComparator.reversed().thenComparing(
241                         entry -> entry.getValue().stream()
242                                 .mapToDouble(IShape::getArea).sum()))
243                 .forEach(entry -> {
244                     pw.println("Shapes of user: " + entry.getKey());
245                     entry.getValue().forEach(pw::println);
246                 });
247
248         pw.flush();
249     }
250
251     public void statistics(OutputStream os) {
252         PrintWriter pw = new PrintWriter(os);
253         pw.println(shapes.stream().mapToDouble(IShape::getArea)
254                 .summaryStatistics());
255         pw.flush();
```

```
    }
}

public class CanvasTest {

    public static void main(String[] args) {
        Canvas canvas = new Canvas();

        System.out.println("READ SHAPES AND EXCEPTIONS TESTING");
        try {
            canvas.readShapes(System.in);
        } catch (InvalidDimensionException e) {
            System.out.println(e.getMessage());
        }

        System.out.println("BEFORE SCALING");
        canvas.printAllShapes(System.out, true);
        canvas.scaleShapes("123456", 1.5);
        System.out.println("AFTER SCALING");
        canvas.printAllShapes(System.out, false);

        System.out.println("PRINT BY USER ID TESTING");
        canvas.printByUserId(System.out);

        System.out.println("PRINT STATISTICS");
        canvas.statistics(System.out);
    }
}
```

Проблем 5.4 За потребите на Министерството за здравство потребно е да се направи апликација која ќе ги менаџира корисниците на апликацијата и нивните контакти со кои биле во близина. Да се дефинира класа `StopCorona` и за истата да се имплементираат:

- предефиниран конструктор
- метод `void addUser(String name, String id)` - што ќе регистрира нов корисник на апликацијата. Доколку веќе постои корисник со такво `id`, методот треба да фрли исклучок од тип `UserIdAlreadyExistsException`.
- Метод `void addLocations (String id, List<ILocation> iLocations)` - што за корисникот со ИД исто како првиот аргумент, ќе ги регистрира сите негови детектирани локации. `ILocation` е интерфејс и истиот обезбедува информации за должината и ширината на локацијата, како и времето кога е детектирана таа локација.
- Метод `void detectNewCase (String id, LocalDateTime timestamp)` - што симулира пријавување на даден корисник дека е носител на вирусот. Првиот аргумент е неговото ИД, а вториот е времето кога корисникот пријавил дека е носител.
- Метод `Map<User, Integer> getDirectContacts (User u)` - што ќе враќа мапа во која клучеви се сите блиски контакти на корисикот `u`, а соодветните вредности во мапата се бројот на остварени блиски контакти со корисникот `u`.
- Метод `Collection<User> getIndirectContacts (User u)` - што ќе враќа колекција од индиректните контакти на корисникот `u`. За индиректни контакти се мислат блиските контакти на директните контакти на `u`, при што еден

корисник не може да биде и директен и индиректен контакт на некој друг корисник.

- Метод `void createReport ()` - што ќе креира и испечати извештај за МЗ во кој за сите корисници носители на вирусот ќе испечати информации во следниот формат:

```
[user_name] [user_id] [timestamp_detected]
Direct contacts:
[contact1_name] [contact1_first_five_letters_of_id]
[number_of_detected_contacts1]
[contact2_name] [contact2_first_five_letters_of_id]
[number_of_detected_contacts1]
...
[contactN_name] [contactN_first_five_letters_of_id]
[number_of_detected_contactsN]
Count of direct contacts: [sum]
Indirect contacts:
[contact1_name] [contact1_first_five_letters_of_id]
[contact2_name] [contact2_first_five_letters_of_id]
...
[contactN_name] [contactN_first_five_letters_of_id]
Count of indirect contacts: [count]
```

- Дополнително, на крајот на извештајот да се испечати просечниот број на директни и индиректни контакти на корисниците, коишто се носители на Корона вирусот.

Забелешка:

- Близок контакт се мисли контактот на двајца корисници кога евклидовото растојание помеѓу некоја од нивните локации е `<=2`, а временското растојание на соодветно измерените локации е помало од 5(пет) минути.
- Носителите на вирусот да се сортирани според времето кога се детектирани дека се носители. Директните контакти на носителите да бидат сортирани според бројот на остварени блиски контакти во опаѓачки редослед. Индиректните контакти да се сортирани лексикографски според нивното име, а доколку се исти според ИД на корисникот.

Во класата `StopCoronaApp` информациите за сите корисници на апликацијата се чуваат во мапата `userByIdMap`, во која клуч е ИД-то на корисникот, а вредност е објект од класата `User`. Дополнително, во оваа класа чуваме уште една мапа за корисниците на апликацијата кои се инфицирани од вирусот КОВИД-19. Во оваа мапа, клучот е ИД-то на инфицираниот корисник и вредноста е временски печат кога корисникот е пријавен како инфициран (објект од класата `LocalDateTime`).

`Stream` оператори во класата `User` се користат во `countCloseContacts(User otherUser)` методот. Тој како резултат го враќа бројот на блиски контакти на корисникот со `otherUser`. Прво, се користи `flatMapToInt` операторот којшто ја мапира секоја i-та локација на `this` корисникот во поток од цели броеви (`IntStream`). Овој поток е составен од броевите 0 и 1 и означува дали i-тата и j-тата локација ги исполнуваат критериумот за близок контакт (j-тата локација ги претставува

локациите на корисникот `otherUser`). На крајот, откако е добиен `IntStream` се повикува терминалниот оператор `sum` којшто враќа вкупен број на сите блиски контакти меѓу двата корисници.

Во класата `StopCoronaApp`, `stream` операторите се употребени во методите:

- Методот `Map<User, Integer> getDirectContacts(User u)` - Во рамки на овој метод, прво се креира поток од сите корисници на апликацијата (вредностите од мапата `userByIdMap`). Од тој поток се отстранува корисникот `u` како и сите корисници кои немаат отстварено близок контакт со корисникот `u` (со употреба на операторот `filter`. Потоа, со операторот `forEach` секој од останатите корисници се сместува во мапа, кадешто клуч е корисникот, а вредност е бројот на блиски контакти со корисникот `u`.
- Методот `Collection<User> getIndirectContact(User u)` - Во согласност со барањата наведени во задачата за индиректни контакти на еден корисник се сметаат сите блиски контакти на неговите директни контакти. За таа цел, резултатот од методот `getDirectContacts` со операторот `flatMap` се мапира секој директен контакт на `u` во неговите директни контакти и сите тие се агрегираат во `Stream<User>`. Од овие корисници се остранууваат тие што се веќе евидентирани како директни контакти, како и самиот корисник `u` со помош на операторот `filter`. Пропуштените корисници се собираат во подредено множество `TreeSet` со помош на колектор.
- Методот `void createReport()` - Во овој метод се креира поток на парови клуч-вредност од мапата на инфицирани корисници. Истиот поток се подредува со операторот `sorted` според вредноста на парот клуч-вредност. Потоа, со операторот `forEach` за секој пар се повикува методот `printInfectedUserEntry`.

```
1  import java.time.Duration;
2  import java.util.*;
3  import java.time.LocalDateTime;
4  import java.util.stream.Collectors;
5
6  interface ILocation {
7      double getLongitude();
8
9      double getLatitude();
10
11     LocalDateTime getTimestamp();
12 }
13
14 class LocationUtils {
15     public static double distanceBetween(ILocation location1,
16                                          ILocation location2) {
17         return Math.sqrt(Math.pow(location1.getLatitude() -
18                 location2.getLatitude(), 2)
19                 + Math.pow(
20                 location1.getLongitude() - location2.getLongitude(),
21                 2));
22     }
23
24     public static double timeBetweenInSeconds(ILocation location1,
25                                               ILocation location2) {
26         return Math.abs(Duration.between(location1.getTimestamp(),
27                 location2.getTimestamp()).getSeconds());
28     }
29
30     public static boolean isDanger(ILocation location1,
```

```
                                    ILocation location2) {
        return distanceBetween(location1, location2) <= 2.0 &&
                timeBetweenInSeconds(location1, location2) <= 300;
    }
}

class User {
    String id;
    String name;
    List<ILocation> locations;

    public User(String id, String name) {
        this.id = id;
        this.name = name;
        locations = new ArrayList<>();
    }

    public void addLocations(List<ILocation> iLocations) {
        locations.addAll(iLocations);
    }

    public String complete() {
        return String.format("%s %s", name, id);
    }

    public String hidden() {
        return String.format("%s %s***", name, id.substring(0, 4));
    }

    public int countCloseContacts(User otherUser) {
        return locations.stream()
                .flatMapToInt(i -> otherUser.locations.stream()
                        .mapToInt(j -> LocationUtils.isDanger(i, j) ?
                                1 : 0))
                .sum();
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        User user = (User) o;
        return Objects.equals(id, user.id);
    }

    @Override
    public int hashCode() {
        return Objects.hash(id);
    }

    public String getId() {
        return id;
    }

    public String getName() {
        return name;
    }
}

class UserAlreadyExistException extends Exception {
    String id;
```

```
92
93      public UserAlreadyExistException(String id) {
94          super(String.format("User with id %s already exists", id));
95      }
96  }
97
98  class StopCoronaApp {
99
100     Map<String, User> userByIdMap;
101     Map<String, LocalDateTime> infectedUsersByIdMap;
102
103     StopCoronaApp() {
104         userByIdMap = new HashMap<>();
105         infectedUsersByIdMap = new HashMap<>();
106     }
107
108
109     public void addUser(String name, String id)
110             throws UserAlreadyExistException {
111         if (userByIdMap.containsKey(id))
112             throw new UserAlreadyExistException(id);
113         userByIdMap.put(id, new User(id, name));
114     }
115
116     public void addLocations(String id, List<ILocation> locations) {
117         userByIdMap.get(id).addLocations(locations);
118     }
119
120
121     public void detectNewCase(String id, LocalDateTime timestamp) {
122         infectedUsersByIdMap.put(id, timestamp);
123     }
124
125     public Map<User, Integer> getDirectContacts(User u) {
126         Map<User, Integer> result =
127                 new TreeMap<>(Comparator.comparing(User::getId));
128         userByIdMap.values().stream()
129                 .filter(user -> !user.equals(u))
130                 .filter(user -> user.countCloseContacts(u) != 0)
131                 .forEach(user -> result
132                         .put(user, u.countCloseContacts(user)));
133         return result;
134     }
135
136     public Collection<User> getIndirectContact(User u) {
137         Comparator<User> comparator =
138                 Comparator.comparing(User::getName)
139                         .thenComparing(User::getId));
140         Map<User, Integer> directContact = getDirectContacts(u);
141         return directContact.keySet().stream()
142                 .flatMap(user -> getDirectContacts(user).keySet()
143                         .stream())
144                 .filter(user -> !directContact.containsKey(user) &&
145                         !user.equals(u))
146                 .collect(Collectors.toCollection(
147                         () -> new TreeSet<>(comparator)));
148
149     }
150
151
152     public void createReport() {
```

```
153
154         List<Integer> countOfDirectContacts = new ArrayList<>();
155         List<Integer> countOfIndirectContacts = new ArrayList<>();
156
157         infectedUsersByIdMap.entrySet().stream()
158                 .sorted(Map.Entry.comparingByValue())
159                 .forEach(entry -> printInfectedUserEntry(entry,
160                         countOfDirectContacts,
161                         countOfIndirectContacts));
162
163         System.out.printf("Average direct contacts: %.4f\n",
164                 countOfDirectContacts.stream().mapToInt(i -> i)
165                         .average().getAsDouble());
166         System.out.printf("Average indirect contacts: %.4f\n",
167                 countOfIndirectContacts.stream().mapToInt(i -> i)
168                         .average().getAsDouble());
169
170
171     }
172
173     private void printInfectedUserEntry(
174             Map.Entry<String, LocalDateTime> entry,
175             List<Integer> countsOfDirectContacts,
176             List<Integer> countsOfIndirectContacts) {
177         User user = userByIdMap.get(entry.getKey());
178         System.out
179                 .printf("%s %s\n", user.complete(), entry.getValue());
180         System.out.println("Direct contacts:");
181
182         Map<User, Integer> directContacts = getDirectContacts(user);
183
184         directContacts.entrySet().stream()
185                 .sorted(Map.Entry
186                         .comparingByValue(Comparator.reverseOrder()))
187                 .forEach(e -> System.out
188                         .printf("%s %d\n", e.getKey().hidden(),
189                                 e.getValue()));
190
191         int countOfDirectContact =
192                 directContacts.values().stream().mapToInt(i -> i)
193                         .sum();
194         System.out.printf("Count of direct contacts: %d\n",
195                 countOfDirectContact);
196         countsOfDirectContacts.add(countOfDirectContact);
197         System.out.println("Indirect contacts: ");
198
199         Collection<User> indirectContacts = getIndirectContact(user);
200         indirectContacts.forEach(u -> System.out.println(u.hidden()));
201         System.out.printf("Count of indirect contacts: %d\n",
202                 indirectContacts.size());
203         countsOfIndirectContacts.add(indirectContacts.size());
204     }
205 }
206
207 public class StopCoronaTest {
208
209     public static double timeBetweenInSeconds(ILocation location1,
210                                               ILocation location2) {
211         return Math.abs(Duration.between(location1.getTimestamp(),
212                 location2.getTimestamp()).getSeconds());
213     }
```

```
214
215     public static void main(String[] args) {
216         Scanner sc = new Scanner(System.in);
217
218         StopCoronaApp stopCoronaApp = new StopCoronaApp();
219
220         while (sc.hasNext()) {
221             String line = sc.nextLine();
222             String[] parts = line.split("\\s+");
223
224             switch (parts[0]) {
225                 case "REG": //register
226                     String name = parts[1];
227                     String id = parts[2];
228                     try {
229                         stopCoronaApp.addUser(name, id);
230                     } catch (UserAlreadyExistException e) {
231                         System.out.println(e.getMessage());
232                     }
233                     break;
234                 case "LOC": //add locations
235                     id = parts[1];
236                     List<ILocation> locations = new ArrayList<>();
237                     for (int i = 2; i < parts.length; i += 3) {
238                         locations.add(createLocationObject(parts[i],
239                                 parts[i + 1], parts[i + 2]));
240                     }
241                     stopCoronaApp.addLocations(id, locations);
242
243                     break;
244                 case "DET": //detect new cases
245                     id = parts[1];
246                     LocalDateTime timestamp =
247                             LocalDateTime.parse(parts[2]);
248                     stopCoronaApp.detectNewCase(id, timestamp);
249
250                     break;
251                 case "REP": //print report
252                     stopCoronaApp.createReport();
253                     break;
254                 default:
255                     break;
256             }
257         }
258     }
259
260     private static ILocation createLocationObject(String lon,
261                                                   String lat,
262                                                   String timestamp) {
263         return new ILocation() {
264             @Override
265             public double getLongitude() {
266                 return Double.parseDouble(lon);
267             }
268
269             @Override
270             public double getLatitude() {
271                 return Double.parseDouble(lat);
272             }
273
274             @Override
```

```
275             public LocalDateTime getTimestamp() {
276                 return LocalDateTime.parse(timestamp);
277             }
278         };
279     }
280 }
```

Проблем 5.5 Да се дефинира класата `GenericCollection` во којашто ќе се чуваат елементи кои треба да се споредливи и елементи кои имаат временски момент на креирање. Класата да ги овозможи следните методи:

- `void addGenericItem (String category, T element)` - метод за додавање на нов елемент во дадена категорија
- `Collection<T> findAllBetween (LocalDateTime from, LocalDateTime to)` - метод којшто ќе врати колекција од сите елементи што се наоѓаат во интервалот на датуми даден како аргументи на функцијата.
- `Collection<T> itemsFromCategories (List<String> categories)` - метод што ќе врати број на елементи што се наоѓаат во категориите дадени како аргумент на функцијата.
- `public Map< String, Set<T> > byMonthAndDay()` - враќа мапа во којашто елементите се групирани според нивниот timestamp (поточно месецот и денот споени со `-` измеѓу нив пр. `12-30`, без разлика на годината). Месецот се добива со повик на методот `getMonth()`, а денот `getDayOfMonth()`.
- `public Map<Integer, Long> countByYear()` - враќа мапа во којашто клучеви се сите години кога има креирани некој елемент, а соодветната вредност е бројот на елементи креирани во таа година.

Секаде каде што има колекција од елементи, истите треба да бидат подредени во опаѓачки редослед! ■

Во решението на оваа задача треба да се употребат генерици. За таа цел, прво треба да се утврди како точно да се дефинира генеричкиот тип `T` и кои се неговите ограничувања. Од барањата во задачата, произлегуваат две ограничувања: Елементите да може да се споредуваат и да имаат свој временски печат `LocalDateTime`. Од ограничувањето дека елементите треба да се споредуваат, може да се заклучи дека `T` треба да го имплементира генеричкиот интерфејс `Comparable<T>`. Од второто ограничување, може да се заклучи дека типот `T` треба да имплементира интерфејс со метод `getTimestamp` (интерфејсот `IHasTimestamp`). Од ова произлегува дека `T extends Comparable<T> & IHasTimestamp` е дефиницијата на генеричкиот тип `T`.

Во класата `GenericCollection` се чуваат генеричките елементи групирани според нивната категорија во мапа каде што клучот е името на категоријата, а вредноста е множество од генеричките елементи што припаѓаат во соодветната категорија.

Во класата `GenericCollection<T>` на неколку места се употребени `stream` оператори:

- Методот `Collection<T> findAllBetween(LocalDateTime from, LocalDateTime to)` - Во овој метод, со помош на операторот `flatMap` прво се израмнува колекцијата од множества од генерички елементи во колекција од генерички елементи. По израмнувањето се филтрираат елементите чии временски печат е помеѓу временскиот печат `from` и `to`. На крајот, со помош на операторот `collect` се собираат сите пропуштени елементи во подредено множество (`TreeSet`).
- Методот `Collection<T> itemsFromCategories (List<String> categories)`

- Во овој метод, прво се филтрираат категориите од мапата што припаѓаат во листата `categories`. Потоа, пропуштените категории се мапираат со помош на `flatMap` операторот во поток од генеричките елементи кои припаѓаат на соодветните категории. На крајот, повторно како и во претходниот метод истите се агрегираат во подредено множество.

- Методот `Map< String, Set<T> > byMonthAndDay()` - Во овој метод, исто како и во првиот метод, прво се израмнуваат сите генерички елементи во поток од генерички елементи. Потоа, со помош на `collect` операторот и `groupBy` колекторот се собираат сите елементи во мапа каде клучот е текстуална низа во формат месец-ден (првиот аргумент во `groupBy` повикот). Вредност е подредено множество од сите елементи чии временски печат е во тој ден и месец (третиот аргумент во `groupBy` повикот). Резултантната мапа е од тип `TreeMap` (дефинирана преку вториот аргумент во `groupBy` повикот).

- Методот `Map<Integer,Long> countByYear()` - Овој метод е комплетно аналоген на претходниот и истиот се разликува само во однос на агрегацијата на елементите направена со `groupBy` операторот. Во овој метод, елементите се групираат според годината од временскиот печат (првиот аргумент во `groupBy` повикот), а резултатот е вкупниот број на елементи во рамки на таа година.

```
import java.time.LocalDateTime;
import java.util.*;
import java.util.stream.Collectors;

interface IHasTimestamp {
    LocalDateTime getTimestamp();
}

class IntegerElement
        implements Comparable<IntegerElement>, IHasTimestamp {

    int value;
    LocalDateTime timestamp;


    public IntegerElement(int value, LocalDateTime timestamp) {
        this.value = value;
        this.timestamp = timestamp;
    }

    @Override
    public LocalDateTime getTimestamp() {
        return timestamp;
    }

    @Override
    public int compareTo(IntegerElement o) {
        return Integer.compare(this.value, o.value);
    }

    @Override
    public String toString() {
        return "IntegerElement{" +
                "value=" + value +
                ", timestamp=" + timestamp +
                '}';
    }
```

```
}

class StringElement
        implements Comparable<StringElement>, IHasTimestamp {

    String value;
    LocalDateTime timestamp;


    public StringElement(String value, LocalDateTime timestamp) {
        this.value = value;
        this.timestamp = timestamp;
    }

    @Override
    public LocalDateTime getTimestamp() {
        return timestamp;
    }

    @Override
    public int compareTo(StringElement o) {
        return this.value.compareTo(o.value);
    }

    @Override
    public String toString() {
        return "StringElement{" +
                "value='" + value + '\'' +
                ", timestamp=" + timestamp +
                '}';
    }
}

class TwoIntegersElement
        implements Comparable<TwoIntegersElement>, IHasTimestamp {

    int value1;
    int value2;
    LocalDateTime timestamp;

    public TwoIntegersElement(int value1, int value2,
                              LocalDateTime timestamp) {
        this.value1 = value1;
        this.value2 = value2;
        this.timestamp = timestamp;
    }

    @Override
    public LocalDateTime getTimestamp() {
        return timestamp;
    }

    @Override
    public int compareTo(TwoIntegersElement o) {
        int cmp = Integer.compare(this.value1, o.value1);
        if (cmp != 0)
            return cmp;
        else
            return Integer.compare(this.value2, o.value2);
    }

```

```
99      @Override
100     public String toString() {
101         return "TwoIntegersElement{" +
102                 "value1=" + value1 +
103                 ", value2=" + value2 +
104                 ", timestamp=" + timestamp +
105                 '}';
106     }
107 }
108 
109 class GenericCollection<T extends Comparable<T> & IHasTimestamp> {
110 
111     Map<String, Set<T>> mapByCategory;
112 
113     GenericCollection() {
114         mapByCategory = new HashMap<>();
115     }
116 
117     public void addGenericItem(String category, T element) {
118         mapByCategory.putIfAbsent(category, new TreeSet<>());
119         mapByCategory.computeIfPresent(category, (k, v) -> {
120             v.add(element);
121             return v;
122         });
123     }
124 
125     public Collection<T> findAllBetween(LocalDateTime from,
126                                         LocalDateTime to) {
127         return mapByCategory.values().stream()
128                 .flatMap(Collection::stream)
129                 .filter(t -> t.getTimestamp().isAfter(from) &&
130                         t.getTimestamp().isBefore(to))
131                 .collect(Collectors.toCollection(() -> new TreeSet<T>(
132                         Comparator.reverseOrder())));
133     }
134 
135     public Collection<T> itemsFromCategories(
136             List<String> categories) {
137         return mapByCategory.keySet().stream()
138                 .filter(categories::contains)
139                 .flatMap(category -> mapByCategory.get(category)
140                         .stream())
141                 .collect(Collectors.toCollection(() -> new TreeSet<T>(
142                         Comparator.reverseOrder())));
143     }
144 
145     public Map<String, Set<T>> byMonthAndDay() {
146         return mapByCategory.values().stream()
147                 .flatMap(Collection::stream)
148                 .collect(Collectors.groupingBy(
149                         element -> String.format("%02d-%02d",
150                                 element.getTimestamp()
151                                         .getMonthValue(),
152                                 element.getTimestamp()
153                                         .getDayOfMonth()),
154                         TreeMap::new,
155                         Collectors.toCollection(() -> new TreeSet<T>(
156                                 Comparator.reverseOrder()))
157                 ));
158     }
159 
```

```
160     public Map<Integer, Long> countByYear() {
161         return mapByCategory.values().stream()
162                 .flatMap(Collection::stream)
163                 .collect(Collectors.groupingBy(
164                         element -> element.getTimestamp().getYear(),
165                         TreeMap::new,
166                         Collectors.counting()
167                 ));
168     }
169 
170     private String getDayAndMonth(T element) {
171         return String.format("%02d-%02d",
172                 element.getTimestamp().getMonthValue(),
173                 element.getTimestamp().getDayOfMonth());
174     }
175 
176     private Integer getYear(T element) {
177         return element.getTimestamp().getYear();
178     }
179 
180 }
181 
182 public class GenericCollectionTest {
183 
184     public static void main(String[] args) {
185 
186         int type1, type2;
187         GenericCollection<IntegerElement> integerCollection =
188                 new GenericCollection<IntegerElement>();
189         GenericCollection<StringElement> stringCollection =
190                 new GenericCollection<StringElement>();
191         GenericCollection<TwoIntegersElement> twoIntegersCollection =
192                 new GenericCollection<TwoIntegersElement>();
193         Scanner sc = new Scanner(System.in);
194 
195         type1 = sc.nextInt();
196 
197         int count = sc.nextInt();
198 
199         for (int i = 0; i < count; i++) {
200             if (type1 == 1) { //integer element
201                 int value = sc.nextInt();
202                 LocalDateTime timestamp =
203                         LocalDateTime.parse(sc.next());
204                 String category = sc.next();
205                 integerCollection.addGenericItem(category,
206                         new IntegerElement(value, timestamp));
207             } else if (type1 == 2) { //string element
208                 String value = sc.next();
209                 LocalDateTime timestamp =
210                         LocalDateTime.parse(sc.next());
211                 String category = sc.next();
212                 stringCollection.addGenericItem(category,
213                         new StringElement(value, timestamp));
214             } else { //two integer element
215                 int value1 = sc.nextInt();
216                 int value2 = sc.nextInt();
217                 LocalDateTime timestamp =
218                         LocalDateTime.parse(sc.next());
219                 String category = sc.next();
220                 twoIntegersCollection.addGenericItem(category,
```

```
                        new TwoIntegersElement(value1, value2,
                                timestamp));
            }
        }

        type2 = sc.nextInt();

        if (type2 == 1) { //findAllBetween
            LocalDateTime start = LocalDateTime.of(2008, 1, 1, 0, 0);
            LocalDateTime end = LocalDateTime.of(2020, 1, 30, 23, 59);
            if (type1 == 1)
                printResultsFromFindAllBetween(integerCollection,
                        start, end);
            else if (type1 == 2)
                printResultsFromFindAllBetween(stringCollection,
                        start, end);
            else
                printResultsFromFindAllBetween(twoIntegersCollection,
                        start, end);
        } else if (type2 == 2) { //itemsFromCategories
            List<String> categories = new ArrayList<>();
            int n = sc.nextInt();
            while (n != 0) {
                categories.add(sc.next());
                n--;
            }
            if (type1 == 1)
                printResultsFromItemsFromCategories(integerCollection,
                        categories);
            else if (type1 == 2)
                printResultsFromItemsFromCategories(stringCollection,
                        categories);
            else
                printResultsFromItemsFromCategories(
                        twoIntegersCollection, categories);
        } else if (type2 == 3) { //byMonthAndDay
            if (type1 == 1)
                printResultsFromByMonthAndDay(integerCollection);
            else if (type1 == 2)
                printResultsFromByMonthAndDay(stringCollection);
            else
                printResultsFromByMonthAndDay(twoIntegersCollection);
        } else { //countByYear
            if (type1 == 1)
                printResultsFromCountByYear(integerCollection);
            else if (type1 == 2)
                printResultsFromCountByYear(stringCollection);
            else
                printResultsFromCountByYear(twoIntegersCollection);
        }
    }

    private static void printResultsFromItemsFromCategories(
            GenericCollection<?> collection,
            List<String> categories) {
        collection.itemsFromCategories(categories).forEach(
                element -> System.out.println(element.toString()));
    }

    private static void printResultsFromFindAllBetween(
            GenericCollection<?> collection, LocalDateTime start,
```

```
282             LocalDateTime end) {
283         collection.findAllBetween(start, end).forEach(
284                 element -> System.out.println(element.toString()));
285     }
286 
287     private static void printSetOfElements(Set<?> set) {
288         System.out.print("[");
289         System.out.print(set.stream().map(Object::toString)
290                 .collect(Collectors.joining(", ")));
291         System.out.println("]");
292     }
293 
294     private static void printResultsFromByMonthAndDay(
295             GenericCollection<?> collection) {
296         collection.byMonthAndDay().forEach((key, value) -> {
297             System.out.print(key + " -> ");
298             printSetOfElements(value);
299         });
300     }
301 
302     private static void printResultsFromCountByYear(
303             GenericCollection<?> collection) {
304         collection.countByYear().forEach((key, value) -> {
305             System.out.println(key + " -> " + value);
306         });
307     }
308 }
```

6. Вовед во шаблони за дизајн на софтвер

Дизајнот на софтвер го опфаќа процесот на планирање и решавање на проблем при развој на софтверско решение. По одредувањето на целите и спецификацијата на барањата, развивачите на софтвер (или дизајнерите) треба да развијат план за решавање на проблемот. Тој вклучува имплементација на алгоритми, развој на компоненти од ниско ниво и архитектонски поглед на решението. Во практика, честопати се појавуваат исти (слични) проблеми што се решаваат на ист (сличен) начин. Идејата е еднаш развиеното докажано решение повторно да се употреби

Во софтверското инженерство, шаблоните за дизајн претставуваат генерално, реупотребливо решение за вообичаени (повторливи) проблеми кои се појавуваат во секојдневието при дизајнот на софтвер. Реупотребата на вакви шаблони може да го забрза процесот на развој на софтвер, бидејќи би се применувале претходно докажани и испробани решенија.

Шаблонот не е готов и директно употребив дизајн кој може директно да се трансфор-мира во код. Шаблонот претставува само опис, рецепт, мостра за решавање на проблем што може да се употреби во повеќе различни ситуации. Објектно-ориентираните шаблони за дизајн вообичаено ги прикажуваат релациите и интеракциите помеѓу класите или објектите, без прецизна спецификација на конечните класи или објекти што ќе бидат вклучени во апликацијата. Алгоритмите не се сметаат за шаблони за дизајн, бидејќи тие решаваат пресметковни, а не дизајнерски проблеми.

Проблем 6.1 Да се напише класа за `MP3Player` во која се чуваат листа со песни (`List<Song>`) и песната којаашто моментално се слуша (е на ред да се пушти). MP3Player-от има четири копчиња `Play`, `Stop`, `FWD` и `REW`.

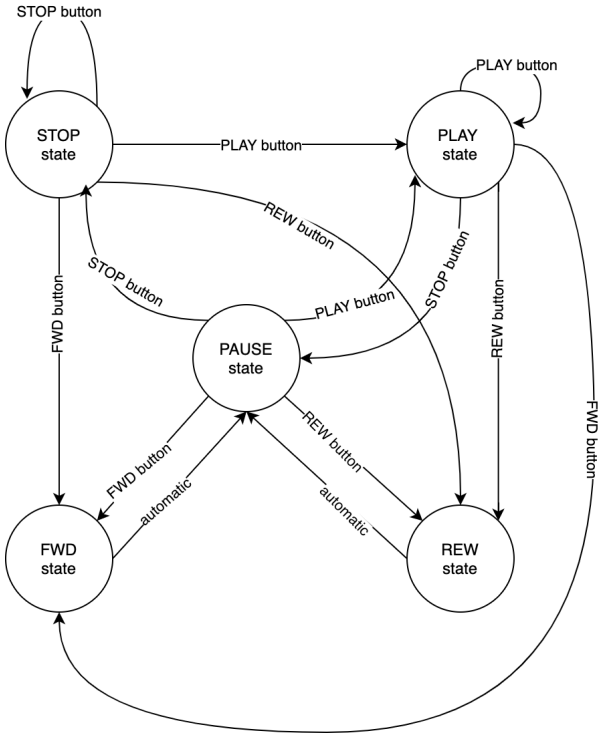
- Ако се притисне копчето `Play` се пушта моменталната песна (на екранот се пишува `"Song i is playing"`, каде што `i` е редниот број на моменталната песна, почнувајќи од 0).
- Ако се притисне копчето `Stop`:

  - моменталната песна којашто е пуштена се паузира (на екран се пишува `"Song i is paused"` каде што `i` е моменталната песна што била пуштена).
  - листата целосно се ресетира од почеток, ако моменталната песна веќе била паузирана (на екран се испишува `"Songs are stopped"`).
- Ако се притисне копчето `FWD` песната се паузира и следната песна од листата станува моментална (да се земе во предвид кружното повторување на песните).
- Ако се притисне копчето REW песната се паузира и претходната песна од листата станува моментална (да се земе во предвид кружното повторување на песните).

За секоја песна се чуваат насловот на песната (како `String`) и изведувачот на песната (како `String`). ■

Задачата може да се реши на многу начини. Со цел да се обезбеди поголема флексибилност и робустност во решението треба да се употребат добри дизајнерски практики. Без употреба на шаблон за развој на софтвер, решението може да стане преобемно со користење на многу if-else услови (од кои голем дел ќе бидат вгнездени). За да се избегне сето тоа, потребно е оваа задача да биде решена со употреба на Состојба (анг. State) шаблонот за развој на софтвер.

При решавање на оваа задача, прво треба да се идентификуваат сите можни состојби. Состојбите кои се идентификувани кај MP3 player-от се: пуштена песна (play), стопирано (stop), паузирано (pause), следна песна (fwd), претходна песна (rew). Откако ќе бидат идентификувани состојбите, треба да се идентификуваат акциите за премин меѓу состојбите. Во овој случај, премин меѓу состојбите може да се реализира со притискање на некој од четирите тастери: STOP, FWD, REW и Play. Најдобар начин за визуелизација на состојбите и премините меѓу состојбите е со помош на конечен автомат (анг. finite state machine). Иницијално MP3 player-от е во состојбата стопирано.



Слика 6.1: Конечен автомат

Откако се идентифукувани сите состојби и можните премини меѓу состојбите, се дефинираат класи за состојбите во согласност со шаблонот за развој на софтвер. Прво, се дефинира интерфејс `State` што како методи ги има сите можни акции со кои се дефинира конкретен премин меѓу различните состојби (во случајов тоа се притискања на одредени тастери). Овој интерфејс го имплементира класата `AbstractState` во која се чува и инстанца од класата `MP3Player` . Потоа, од класата `AbstractState` се изведуваат класите за петте можни состојби. Во секоја од изведените класи се препокриваат методите од интерфејсот `State` и истите се имплементираат во согласност со начинот на справување со акциите за конкретната состојба.

Во класата `MP3Player` , освен потребните информации за песните, се чуваат и сите различни состојби во кои MP3 player-от може да се најде како и референца на моменталната активна состојба (состојбата во којашто се наоѓа MP3 player-от).

```
import java.util.ArrayList;
import java.util.List;

public class PatternTest {
    public static void main(String args[]) {
        List<Song> listSongs = new ArrayList<Song>();
        listSongs.add(new Song("first-title", "first-artist"));
        listSongs.add(new Song("second-title", "second-artist"));
        listSongs.add(new Song("third-title", "third-artist"));
        listSongs.add(new Song("fourth-title", "fourth-artist"));
        listSongs.add(new Song("fifth-title", "fifth-artist"));
        MP3Player player = new MP3Player(listSongs);

        System.out.println(player.toString());
        System.out.println("First test");


        player.pressPlay();
        player.printCurrentSong();
        player.pressPlay();
        player.printCurrentSong();

        player.pressPlay();
        player.printCurrentSong();
        player.pressStop();
        player.printCurrentSong();

        player.pressPlay();
        player.printCurrentSong();
        player.pressFWD();
        player.printCurrentSong();

        player.pressPlay();
        player.printCurrentSong();
        player.pressREW();
        player.printCurrentSong();


        System.out.println(player.toString());
        System.out.println("Second test");


        player.pressStop();
        player.printCurrentSong();
        player.pressStop();
        player.printCurrentSong();
```

```
47
48          player.pressStop();
49          player.printCurrentSong();
50          player.pressPlay();
51          player.printCurrentSong();
52
53          player.pressStop();
54          player.printCurrentSong();
55          player.pressFWD();
56          player.printCurrentSong();
57
58          player.pressStop();
59          player.printCurrentSong();
60          player.pressREW();
61          player.printCurrentSong();
62
63
64          System.out.println(player.toString());
65          System.out.println("Third test");
66
67
68          player.pressFWD();
69          player.printCurrentSong();
70          player.pressFWD();
71          player.printCurrentSong();
72
73          player.pressFWD();
74          player.printCurrentSong();
75          player.pressPlay();
76          player.printCurrentSong();
77
78          player.pressFWD();
79          player.printCurrentSong();
80          player.pressStop();
81          player.printCurrentSong();
82
83          player.pressFWD();
84          player.printCurrentSong();
85          player.pressREW();
86          player.printCurrentSong();
87
88
89          System.out.println(player.toString());
90      }
91  }
92
93  class Song {
94      String title;
95      String artist;
96
97      public Song(String title, String artist) {
98          this.title = title;
99          this.artist = artist;
100     }
101
102
103     public String getTitle() {
104         return title;
105     }
106
107     public void setTitle(String title) {
```

```
108         this.title = title;
109     }
110 
111     public String getArtist() {
112         return artist;
113     }
114 
115     public void setArtist(String artist) {
116         this.artist = artist;
117     }
118 
119     @Override
120     public String toString() {
121         return "Song{" + "title=" + title + ", artist=" +
122                 artist + '}';
123     }
124 
125 }
126 
127 class MP3Player {
128     List<Song> songList;
129     int currentSong;
130 
131     State play;
132     State pause;
133     State stop;
134     State fwd;
135     State rew;
136 
137     final void createStates() {
138         play = new PlayState(this);
139         pause = new PauseState(this);
140         stop = new StopState(this);
141         fwd = new FWDState(this);
142         rew = new REWState(this);
143         state = stop;
144     }
145 
146     public MP3Player(List<Song> songList) {
147         this.songList = songList;
148         currentSong = 0;
149         createStates();
150     }
151 
152     public State getPlay() {
153         return play;
154     }
155 
156     public void setPlay(State play) {
157         this.play = play;
158     }
159 
160     public State getPause() {
161         return pause;
162     }
163 
164     public void setPause(State pause) {
165         this.pause = pause;
166     }
167 
168     public State getStop() {
```

```
169         return stop;
170     }
171
172     public void setStop(State stop) {
173         this.stop = stop;
174     }
175
176     public State getFwd() {
177         return fwd;
178     }
179
180     public void setFwd(State fwd) {
181         this.fwd = fwd;
182     }
183
184     public State getRew() {
185         return rew;
186     }
187
188     public void setRew(State rew) {
189         this.rew = rew;
190     }
191
192     State state;
193
194     public State getState() {
195         return state;
196     }
197
198     public void setState(State state) {
199         this.state = state;
200     }
201
202     public Song getCurrentSong() {
203         return songList.get(currentSong);
204     }
205
206     public void setSongIndex(int currentSong) {
207         this.currentSong = currentSong % songList.size();
208     }
209
210     public int getSongIndex() {
211         return currentSong;
212     }
213
214
215     public void songFWD() {
216         currentSong = (currentSong + 1) % songList.size();
217     }
218
219     public void songREW() {
220         currentSong =
221                 (currentSong + songList.size() - 1) % songList.size();
222     }
223
224     public void pressPlay() {
225         state.pressPlay();
226     }
227
228     public void pressStop() {
229         state.pressStop();
```

```
230     }
231
232     public void pressFWD() {
233         state.pressFwd();
234         state.forward();
235     }
236
237     public void pressREW() {
238         state.pressRew();
239         state.reward();
240     }
241
242     void printCurrentSong() {
243         System.out.println(getCurrentSong());
244     }
245
246     @Override
247     public String toString() {
248         return "MP3Player{currentSong = " + currentSong +
249                 ", songList = " + songList + "}";
250     }
251
252
253 }
254
255 interface State {
256     void pressPlay();
257
258     void pressStop();
259
260     void pressFwd();
261
262     void pressRew();
263
264     void forward();
265
266     void reward();
267 }
268
269
270 abstract class AbstractState implements State {
271     MP3Player mp3;
272
273     public AbstractState(MP3Player mp3) {
274         this.mp3 = mp3;
275     }
276
277 }
278
279 class FWDState extends AbstractState {
280     public FWDState(MP3Player mp3) {
281         super(mp3);
282     }
283
284     @Override
285     public void pressPlay() {
286         System.out.println("Illegal action");
287     }
288
289     @Override
290     public void pressStop() {
```

```
291         System.out.println("Illegal action");
292     }
293
294     @Override
295     public void pressFwd() {
296         System.out.println("Illegal action");
297     }
298
299     @Override
300     public void pressRew() {
301         System.out.println("Illegal action");
302     }
303
304     @Override
305     public void forward() {
306         mp3.songFWD();
307         mp3.setState(mp3.getPause());
308     }
309
310     @Override
311     public void reward() {
312         System.out.println("Illegal action");
313     }
314
315 }
316
317
318 class REWState extends AbstractState {
319     public REWState(MP3Player mp3) {
320         super(mp3);
321     }
322
323     @Override
324     public void pressPlay() {
325         System.out.println("Illegal action");
326     }
327
328     @Override
329     public void pressStop() {
330         System.out.println("Illegal action");
331     }
332
333     @Override
334     public void pressFwd() {
335         System.out.println("Illegal action");
336     }
337
338     @Override
339     public void pressRew() {
340         System.out.println("Illegal action");
341     }
342
343     @Override
344     public void forward() {
345         System.out.println("Illegal action");
346     }
347
348     @Override
349     public void reward() {
350         mp3.songREW();
351         mp3.setState(mp3.getPause());
```

```
352     }
353
354 }
355
356
357 class PlayState extends AbstractState {
358
359     public PlayState(MP3Player mp3) {
360         super(mp3);
361     }
362
363     @Override
364     public void pressPlay() {
365         System.out.println("Song is already playing");
366     }
367
368     @Override
369     public void pressStop() {
370         System.out
371                 .println("Song " + mp3.getSongIndex() + " is paused");
372         mp3.setState(mp3.getPause());
373     }
374
375     @Override
376     public void pressFwd() {
377         System.out.println("Forward...");
378         mp3.setState(mp3.getFwd());
379     }
380
381     @Override
382     public void pressRew() {
383         System.out.println("Reward...");
384         mp3.setState(mp3.getRew());
385     }
386
387     @Override
388     public void forward() {
389         System.out.println("Illegal action");
390     }
391
392     @Override
393     public void reward() {
394         System.out.println("Illegal action");
395     }
396
397 }
398
399
400 class StopState extends AbstractState {
401
402     public StopState(MP3Player mp3) {
403         super(mp3);
404     }
405
406     @Override
407     public void pressPlay() {
408         System.out.println(
409                 "Song " + mp3.getSongIndex() + " is playing");
410         mp3.setState(mp3.getPlay());
411     }
412
```

```
413     @Override
414     public void pressStop() {
415         System.out.println("Songs are already stopped");
416     }
417 
418     @Override
419     public void pressFwd() {
420         System.out.println("Forward...");
421         mp3.setState(mp3.getFwd());
422     }
423 
424     @Override
425     public void pressRew() {
426         System.out.println("Reward...");
427         mp3.setState(mp3.getRew());
428     }
429 
430     @Override
431     public void forward() {
432         System.out.println("Illegal action");
433     }
434 
435     @Override
436     public void reward() {
437         System.out.println("Illegal action");
438     }
439 
440 }
441 
442 class PauseState extends AbstractState {
443 
444     public PauseState(MP3Player mp3) {
445         super(mp3);
446     }
447 
448     @Override
449     public void pressPlay() {
450         System.out.println(
451                 "Song " + mp3.getSongIndex() + " is playing");
452         mp3.setState(mp3.getPlay());
453     }
454 
455     @Override
456     public void pressStop() {
457         System.out.println("Songs are stopped");
458         mp3.setSongIndex(0);
459         mp3.setState(mp3.getStop());
460     }
461 
462     @Override
463     public void pressFwd() {
464         System.out.println("Forward...");
465         mp3.setState(mp3.getFwd());
466     }
467 
468     @Override
469     public void pressRew() {
470         System.out.println("Reward...");
471         mp3.setState(mp3.getRew());
472     }
473 
```

```
474     @Override
475     public void forward() {
476         System.out.println("Illegal action");
477     }
478
479     @Override
480     public void reward() {
481         System.out.println("Illegal action");
482     }
483 }
```

Проблем 6.2 Да се имплементира класа пошта `PostOffice` во којашто се чуваат името на поштата, локацијата и листа од пратки. За пратките да се имплементира апстрактна класа `Package` во којашто се чуваат информации за пратка: `name` (String) за кого е наменета, `address` (`String`), `trackingNumber` (`int`) за следење и `weight` (`int`) изразена во грамови.

Во системот постојат интернационални пратки (`InternationalPackage`) за кои дополнително се чува за кој регион се испратени (`Africa`, `Asia`, `Europe` и `America`) и локални (`LocalPackage`) пратки за кои се чува дали пратката е со приоритет или без приоритет. Цената на интернационалните пратки се наплаќа според масата, и тоа: по 1.5 долари за грам, а локалните пратки со приоритет се наплаќаат по 5 долари, а по 3 долари пратките без приоритет.

Дополнително, постои и групна пратка (`GroupPackage`) што се состои од една или повеќе пратки од кој било тип (интернационална, локална или групна пратка). За групните пратки треба да постои метод за додавање на нова пратка (во рамки на групната пратка). Цената на групните пратки се пресметува како сума од цените на сите пратки коишто се дел од групната пратка и 2(два) долари како дополнителен трошок. Тежината на групните пратки се пресметува како сума од тежините на сите пратки коишто се дел од групната пратка.

Во класата `PostOffice` да се имплементираат следните методи:

- `void loadPackages(Scanner scanner)` – метод преку кој се внесуваат пратките во следен формат: тип на пратка (`I` или `L`), име, адреса, број, тежина, за локалните – true ако е со приритет и false ако не е, за интернационалните – регион. Доколку записот на пратка не одговара со опишаниот формат (типот на пратката е различна од `I` и `L`, или тежината е помала или еднаква на нула) се фрла исклучок од класата `InvalidPackageException` во кој се проследува невалидниот запис.
- `boolean addPackage(Package p)` - за додавање на пакет во листата со пакети
- `Package mostExpensive()` - ја враќа најскапата пратка
- `void printPackages(OutputStream out)` – метод кој ги печати пратките според цената (од највисока кон најниска) во следен формат:

```
L, name, address, number, weight, true/false за приоритет - локална пратка
I, name, address, number, weight, region - интернационална пратка
G, number, weight - групна пратка
```

Пратките кои се дел од групната пратка се печатат вовлечени за едно празно место надесно. ■

Во оваа задача е потребно да се дефинираат и имплементираат класи за три

различни типа на поштенски пратки (локални, меѓународни и групни). Сиве овие класи наследуваат од апстрактна класа `Package`. Локалните и меѓународните пратки имаат едноставна дефиниција со мали разлики од основната класа, но групните пратки претставуваат композициона податочна структура. За групните пратки треба да се чува листа од пратки (локални, меѓународни и групни пратки). Групната пратка може во себе да содржи друга групна пратка и да формира композиција од различните пратки.

```
import java.io.OutputStream;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.Scanner;

class PostOffice {
    String name;
    String location;
    private ArrayList<Package> packages;

    public PostOffice(String name, String location) {
        this.name = name;
        this.location = location;
        this.packages = new ArrayList<>();
    }

    public boolean addPackage(Package p) {
        return packages.add(p);
    }

    public void loadPackages(Scanner scanner)
            throws InvalidPackageException {
        while (scanner.hasNext()) {
            String line = scanner.nextLine();
            String[] parts = line.split("\\s+");
            if (!parts[0].equals("I") && !parts[0].equals("L")) {
                scanner.close();
                throw new InvalidPackageException(line);
            }
            int weight = Integer.parseInt(parts[4]);
            if (weight <= 0) {
                scanner.close();
                throw new InvalidPackageException(line);
            }
            if (parts[0].equals("I")) {
                packages.add(
                        new InternationalPackage(parts[1], parts[2],
                                Integer.parseInt(parts[3]),
                                Integer.parseInt(parts[4]),
                                parts[5]));
            }
            if (parts[0].equals("L")) {
                packages.add(new LocalPackage(parts[1], parts[2],
                        Integer.parseInt(parts[3]),
                        Integer.parseInt(parts[4]),
                        parts[5].equals("true")));
            }
        }
    }

```

```
53      public Package mostExpensive() {
54          return packages.stream().max(Comparator.naturalOrder()).get();
55      }
56
57      public void printPackages(OutputStream out) {
58          PrintWriter printWriter = new PrintWriter(out);
59          packages.stream().sorted().forEach(printWriter::println);
60          printWriter.flush();
61      }
62  }
63
64  abstract class Package implements Comparable<Package> {
65      String name;
66      String address;
67      int trackingNumber;
68      int weight;
69
70      public Package(String name, String address, int trackingNumber,
71                     int weight) {
72          this.name = name;
73          this.address = address;
74          this.trackingNumber = trackingNumber;
75          this.weight = weight;
76      }
77
78      public int getWeight() {
79          return weight;
80      }
81
82      abstract public double getPrice();
83
84      @Override
85      public int compareTo(Package o) {
86          return Double.compare(o.getPrice(), getPrice());
87      }
88
89      abstract protected String format(String spaces);
90
91      @Override
92      public String toString() {
93          return format("");
94      }
95
96  }
97
98  class InternationalPackage extends Package {
99      String region;
100
101     public InternationalPackage(String name, String address,
102                                 int trackingNumber, int weight,
103                                 String region) {
104         super(name, address, trackingNumber, weight);
105         this.region = region.toString();
106     }
107
108     @Override
109     public double getPrice() {
110         return weight * 1.5;
111     }
112
113     @Override
```

```
114     protected String format(String spaces) {
115         return String.format(spaces + "I, %s, %s, %d, %d, %s", name,
116                 address, trackingNumber, getWeight(), region);
117     }
118 }
119 
120 class LocalPackage extends Package {
121     boolean priorityPackage;
122 
123     public LocalPackage(String name, String address,
124                         int trackingNumber, int weight,
125                         boolean priorityPackage) {
126         super(name, address, trackingNumber, weight);
127         this.priorityPackage = priorityPackage;
128     }
129 
130     @Override
131     public double getPrice() {
132         return priorityPackage ? 5 : 3;
133     }
134 
135     @Override
136     protected String format(String spaces) {
137         return String.format(spaces + "L, %s, %s, %d, %d, %s", name,
138                 address, trackingNumber, getWeight(),
139                 priorityPackage);
140     }
141 }
142 
143 class GroupPackage extends Package {
144     ArrayList<Package> packages;
145 
146     public GroupPackage(String name, String address,
147                         int trackingNumber) {
148         super(name, address, trackingNumber, -1);
149         packages = new ArrayList<>();
150     }
151 
152     @Override
153     public double getPrice() {
154         return packages.stream().mapToDouble(Package::getPrice).sum()+2;
155     }
156 
157     public void addPackage(Package p) {
158         packages.add(p);
159     }
160 
161     @Override
162     public int getWeight() {
163         return packages.stream().mapToInt(Package::getWeight).sum();
164     }
165 
166     @Override
167     protected String format(String spaces) {
168         StringBuilder sBuilder = new StringBuilder();
169         sBuilder.append(
170                 String.format(spaces + "G, %d, %d\n", trackingNumber,
171                         getWeight()));
172 
173         packages.forEach(p -> sBuilder.append(p.format(spaces + " ")).
    append("\n"));
```

```
174
175         sBuilder.deleteCharAt(sBuilder.length() - 1);
176         return sBuilder.toString();
177     }
178 }
179
180 class InvalidPackageException extends Exception {
181
182     public InvalidPackageException(String message) {
183         super(message);
184     }
185
186 }
187
188 public class PostOfficeTester {
189     public static void main(String[] args) {
190         Scanner scanner = new Scanner(System.in);
191         int test = Integer.parseInt(scanner.nextLine());
192         if (test == 1) {
193             //Group Package printing
194             System.out.println("======Packages======");
195             PostOffice office = new PostOffice("Poshta", "Skopje");
196             office.addPackage(new InternationalPackage("John_Doe",
197                     "Main_St_123", 111, 4, "America"));
198             GroupPackage groupPackage =
199                     new GroupPackage("John_Done", "Main_St_123", 232);
200             groupPackage.addPackage(
201                     new LocalPackage("Jon_Snow", "The_Wall", 432, 5,
202                             true));
203             office.addPackage(groupPackage);
204             office.printPackages(System.out);
205         }
206         if (test == 2) {
207             //Nested Group Package printing
208             System.out.println("======Packages======");
209             PostOffice office = new PostOffice("Poshta", "Skopje");
210             office.addPackage(
211                     new InternationalPackage("Richard_Hendricks",
212                             "Noble_Pathway", 325, 4, "Africa"));
213             office.addPackage(new LocalPackage("Jalisa_Acheson",
214                     "Emerald_Harbour", 600, 14, false));
215
216             GroupPackage groupPackage =
217                     new GroupPackage("Meagan_Schuette", "Westeros",
218                             232);
219             groupPackage.addPackage(
220                     new LocalPackage("Meagan_Schuette", "Westeros",
221                             432, 5, true));
222             groupPackage.addPackage(
223                     new InternationalPackage("Sansa_Stark",
224                             "Westeros", 332, 3, "Asia"));
225
226             GroupPackage nestedGroupPackage =
227                     new GroupPackage("Marline_Bohling",
228                             "Crystal_Hills", 284);
229             nestedGroupPackage.addPackage(
230                     new InternationalPackage("Edie_Bramblett",
231                             "Lazy_Treasure", 382, 7, "Europe"));
232             nestedGroupPackage.addPackage(
233                     new InternationalPackage("Cassaundra_Huff",
234                             "Sleepy_Farms", 696, 1, "Asia"));
```

```
235             nestedGroupPackage.addPackage(
236                     new InternationalPackage("German_Sabbagh",
237                             "Tawny_Heath", 963, 12, "Africa"));
238
239             groupPackage.addPackage(nestedGroupPackage);
240             office.addPackage(groupPackage);
241             office.addPackage(
242                     new LocalPackage("Clemmie_Reves", "Little_Cloud",
243                             217, 5, true));
244             office.printPackages(System.out);
245         }
246         if (test == 3) {
247             //Most expensive Group Package
248             System.out.println("======Packages======");
249             PostOffice office = new PostOffice("Poshta", "Skopje");
250             office.addPackage(new InternationalPackage("Dohn_Joe",
251                     "Main_St_321", 444, 4, "Europe"));
252             GroupPackage groupPackage =
253                     new GroupPackage("John_Jon", "First_St_123", 232);
254             groupPackage.addPackage(
255                     new LocalPackage("Jon_Snow", "Westeros", 432, 5,
256                             true));
257             groupPackage.addPackage(
258                     new InternationalPackage("Sansa_Stark",
259                             "Westeros", 332, 3, "Asia"));
260             office.addPackage(groupPackage);
261             office.addPackage(
262                     new LocalPackage("Littlefinger", "The_Eyrie", 987,
263                             7, false));
264             office.printPackages(System.out);
265             System.out.println();
266             System.out.println("======MostExpensive======");
267             System.out.println(office.mostExpensive());
268         }
269         if (test == 4) {
270             //Most expensive International Package
271             System.out.println("======Packages======");
272             PostOffice office = new PostOffice("Poshta", "Skopje");
273             office.addPackage(new InternationalPackage("Dohn_Joe",
274                     "Main_St_321", 444, 15, "Europe"));
275             GroupPackage groupPackage =
276                     new GroupPackage("John_Jon", "First_St_123", 232);
277             groupPackage.addPackage(
278                     new LocalPackage("Jon_Snow", "Westeros", 432, 5,
279                             true));
280             groupPackage.addPackage(
281                     new InternationalPackage("Sansa_Stark",
282                             "Westeros", 332, 3, "Asia"));
283             office.addPackage(groupPackage);
284             office.addPackage(
285                     new LocalPackage("Littlefinger", "The_Eyrie", 987,
286                             7, false));
287             office.printPackages(System.out);
288             System.out.println();
289             System.out.println("======MostExpensive======");
290             System.out.println(office.mostExpensive());
291         }
292         scanner.close();
293     }
294 }
```

Во вакви ситуации се препорачува употребата на Композиција (анг. Composite) шаблонот за развој на софтвер којшто овозможува претставување на композициски податочни структури (во форма на дрво или граф). При користење на овој шаблон за развој на софтвер, најчесто се јавува потреба од рекурзивно изминување на структурата за да се изврши некоја операција. Во оваа задача, рекурзивното изминување на композицијата од елементи се прави при пресметувањето на цената на групната пратка, пресметувањето на тежината на групната пратка, како и при печатењето на пратките коишто се составен дел од групната пратка.

Проблем 6.3 Да се напише класа `TaskManager` што ќе служи за менаџирање на задачи `tasks` на даден корисник. За класата да се имплементираат методите:

- `readTasks (InputStream inputStream)` - метод за читање на задачите на корисникот при што секоја задача е во следниот формат: `[kategorija][ime_na_zadaca],[opis],[rok_na_zadacata],[prioritet]` . Рокот за задачата и приоритетот се опциони полиња. Не смее да се дозволи дадена задача да има рок којшто е веќе поминат. Во ваков случај да се фрли исклучот од тип `DeadlineNotValidException`. Да се фати исклучокот на соодветно место, така што нема да се спречи читањето на останатите задачи!!!
- `void printTasks(OutputStream os, boolean includePriority, boolean includeCategory)` - метод за печатење на задачите.
  - Доколку `includeCategory` е `true` да се испечатат задачите групирани според категории. Во спротивност, се печатат сите внесени задачи
  - Доколку `includePriority` е `true` да се испечатат задачите сортирани според приоритетот (при што 1 е највисок приоритет), а доколку немаат приоритет или имаат ист приоритет се сортираат растечки според временското растојание помеѓу рокот и моменталниот датум. Односно задачите со рок најблизок до денешниот датум се печатат први.
  - Доколку `includePriority` е `false` се печатат во растечки редослед според временското растојание помеѓу рокот и моменталниот датум.
  - При печатењето на задачите се користи default опцијата за toString (доколку работите во IntelliJ), со тоа што треба да внимавате на името на променливите.

■

Од дефиницијата на методот `readTasks` може да се заклучи дека за секоја задача се знае нејзината категоријата и нејзиното име. Дополнително, задачите имаат и опциони полиња за краен рок за извршување на задачата и приоритет. Ова значи дека во рамките на системот за менаџирање на задачи може да постојат 4(четири) типа на задачи: обична задача, задача со приоритет, задача со краен рок за извршување, задача со приоритет и краен рок за извршување. Бидејќи постојат неколку типови на задачи, интуитивен пристап при решавање на овој проблем би бил да се направи еден интерфејс што би го дефинирал однесувањето на задачите и да се дефинираат 4(четири) класи коишто ќе го имплементираат тој интерфејс. Овој пристап би работел за оваа ситуација.

Но, ако претпоставиме дека во иднина се очекува на задачите да се доделат уште 3(три) опционални полиња (на пр. вработен на кого е доделена задачата, епоха во која припаѓа, проценето време за завршување). Во оваа ситуација би требало да се дефинираат 32(триесет и две) класи, односно уште дополнителни 28(дваесет и осум) класи.

Во вакви ситуации се користи шаблонот за дизајн на софтвер Декоратор (анг. Decorator). Декораторот предвидува дефинирање на еден интефејс (`Component`) којшто ги дефинира сите својства на објектите. Дополнително, се имплементира(ат) класа/класи за конкретни компоненти `ConcreteComponent` , како и една апстрактна класа за декоратор `BaseDecorator` . Во апстрактната класа за декораторот се чува инстанца од објект од тип `Component` . Таа инстанца е објектот што се декорира со дополнителни својства/променливи.

Во решението на оваа задача прво, е дефиниран интерфејсот `ITask` . Овој интерфејс има три методи: `getCategory` , `getPriority` , `getDeadline` . Потоа е дефинирана конкретна компонента - класа за обичната задача `SimpleTask` . `TaskDecorator` е апстрактна класа декоратор, а конкретните класи за двата типа на декорирање се `PriorityTaskDecorator` и `TimeTaskDecorator` .

Во согласност со ваквиот дизајн, креирањето на објекти од тип `ITask` што имаат приоритет и краен рок на завршување се прави на следниот начин:

```
ITask base = new SimpleTask (category, name , description);
ITask result = new PriorityTaskDecorator(new TimeTaskDecorator (base,
    deadline), priority );
```

Доколку се вратиме повторно на сценариото, во кое една задача има 5(пет) опциони полиња, со користење на Декоратор шаблонот, наместо да се креираат 32(триесет и две) класи, ќе се креираат само 5(пет) класи за конкретните декоратори.

```
import java.io.*;
import java.time.Duration;
import java.time.LocalDateTime;
import java.util.*;
import java.util.stream.Collectors;
import java.util.stream.Stream;

interface ITask {
    LocalDateTime getDeadline();

    int getPriority();

    String getCategory();
}

class DeadlineNotValidException extends Exception {
    public DeadlineNotValidException(LocalDateTime deadline) {
        super(String.format("The deadline %s has already passed",
                deadline));
    }
}

class SimpleTask implements ITask {
    String category;
    String name;
    String description;

    public SimpleTask(String category, String name,
                      String description) {
        this.category = category;
        this.name = name;
        this.description = description;
    }

    @Override
```

```
36      public LocalDateTime getDeadline() {
37          return LocalDateTime.MAX;
38      }
39
40      @Override
41      public int getPriority() {
42          return Integer.MAX_VALUE;
43      }
44
45      @Override
46      public String getCategory() {
47          return category;
48      }
49
50      @Override
51      public String toString() {
52          final StringBuilder sb = new StringBuilder("Task{");
53          sb.append("name='").append(name).append('\'');
54          sb.append(", description='").append(description).append('\'');
55          sb.append('}');
56          return sb.toString();
57      }
58  }
59
60  abstract class TaskDecorator implements ITask {
61      ITask iTask;
62
63      public TaskDecorator(ITask iTask) {
64          this.iTask = iTask;
65      }
66  }
67
68  class PriorityTaskDecorator extends TaskDecorator {
69
70      int priority;
71
72      public PriorityTaskDecorator(ITask iTask, int priority) {
73          super(iTask);
74          this.priority = priority;
75      }
76
77      @Override
78      public LocalDateTime getDeadline() {
79          return iTask.getDeadline();
80      }
81
82      @Override
83      public int getPriority() {
84          return priority;
85      }
86
87      @Override
88      public String getCategory() {
89          return iTask.getCategory();
90      }
91
92      @Override
93      public String toString() {
94          StringBuilder sb = new StringBuilder();
95          sb.append(iTask.toString(), 0, iTask.toString().length() - 1);
96          sb.append(", priority=").append(priority);
```

```
97          sb.append('}');
98          return sb.toString();
99      }
100 }
101
102 class TimeTaskDecorator extends TaskDecorator {
103
104     LocalDateTime deadline;
105
106     public TimeTaskDecorator(ITask iTask, LocalDateTime deadline) {
107         super(iTask);
108         this.deadline = deadline;
109     }
110
111     @Override
112     public LocalDateTime getDeadline() {
113         return deadline;
114     }
115
116     @Override
117     public int getPriority() {
118         return iTask.getPriority();
119     }
120
121     @Override
122     public String getCategory() {
123         return iTask.getCategory();
124     }
125
126     @Override
127     public String toString() {
128         StringBuilder sb = new StringBuilder();
129         sb.append(iTask.toString(), 0, iTask.toString().length() - 1);
130         sb.append(", deadline=").append(deadline);
131         sb.append('}');
132         return sb.toString();
133     }
134 }
135
136 class TaskFactory {
137     public static ITask createTask(String line)
138             throws DeadlineNotValidException {
139         String[] parts = line.split(",");
140         String category = parts[0];
141         String name = parts[1];
142         String description = parts[2];
143         SimpleTask base = new SimpleTask(category, name, description);
144         if (parts.length == 3) {
145             return base;
146         } else if (parts.length == 4) {
147             try {
148                 int priority = Integer.parseInt(parts[3]);
149                 return new PriorityTaskDecorator(base, priority);
150             } catch (Exception e) { //parsing failed, it's a date
151                 LocalDateTime deadline =
152                         LocalDateTime.parse(parts[3]);
153                 checkDeadline(deadline);
154                 return new TimeTaskDecorator(base, deadline);
155             }
156         } else {
157             LocalDateTime deadline = LocalDateTime.parse(parts[3]);
```

```
158             checkDeadline(deadline);
159             int priority = Integer.parseInt(parts[4]);
160             return new PriorityTaskDecorator(
161                     new TimeTaskDecorator(base, deadline), priority);
162         }
163     }
164
165     private static void checkDeadline(LocalDateTime deadline)
166             throws DeadlineNotValidException {
167         if (deadline.isBefore(LocalDateTime.now()))
168             throw new DeadlineNotValidException(deadline);
169     }
170 }
171
172 class TaskManager {
173     Map<String, List<ITask>> tasks;
174
175     public TaskManager() {
176         tasks = new TreeMap<>();
177     }
178
179     public void readTasks(InputStream inputStream) {
180         tasks = new BufferedReader(new InputStreamReader(inputStream))
181                 .lines()
182                 .map(line -> {
183                     try {
184                         return TaskFactory.createTask(line);
185                     } catch (DeadlineNotValidException e) {
186                         System.out.println(e.getMessage());
187                     }
188                     return null;
189                 })
190                 .filter(Objects::nonNull)
191                 .collect(Collectors.groupingBy(
192                         ITask::getCategory,
193                         TreeMap::new,
194                         Collectors.toList())
195                 );
196     }
197
198     public void addTask(ITask iTask) {
199         tasks.computeIfAbsent(iTask.getCategory(),
200                 k -> new ArrayList<>());
201         tasks.computeIfPresent(iTask.getCategory(), (k, v) -> {
202             v.add(iTask);
203             return v;
204         });
205     }
206
207     public void printTasks(OutputStream os, boolean includePriority,
208                            boolean byCategory) {
209         PrintWriter pw = new PrintWriter(os);
210
211         Comparator<ITask> priorityComparator =
212                 Comparator.comparing(ITask::getPriority)
213                         .thenComparing(task -> Duration
214                                 .between(LocalDateTime.now(),
215                                         task.getDeadline()));
216         Comparator<ITask> simpleComparator = Comparator.comparing(
217                 task -> Duration.between(LocalDateTime.now(),
218                         task.getDeadline()));
```

```
219
220         if (byCategory) {
221             tasks.forEach((category, t) -> {
222                 pw.println(category.toUpperCase());
223                 t.stream()
224                         .sorted(includePriority ? priorityComparator :
225                                 simpleComparator)
226                         .forEach(pw::println);
227             });
228         } else {
229             tasks.values().stream()
230                     .flatMap(Collection::stream)
231                     .sorted(includePriority ? priorityComparator :
232                             simpleComparator)
233                     .forEach(pw::println);
234         }
235
236         pw.flush();
237     }
238 }
239
240 public class TasksManagerTest {
241     public static void main(String[] args) {
242         TaskManager manager = new TaskManager();
243
244         System.out.println("Tasks reading");
245         manager.readTasks(System.in);
246         System.out.println("By categories with priority");
247         manager.printTasks(System.out, true, true);
248         System.out.println("-------------------------");
249         System.out.println("By categories without priority");
250         manager.printTasks(System.out, false, true);
251         System.out.println("-------------------------");
252         System.out.println("All tasks without priority");
253         manager.printTasks(System.out, false, false);
254         System.out.println("-------------------------");
255         System.out.println("All tasks with priority");
256         manager.printTasks(System.out, true, false);
257         System.out.println("-------------------------");
258
259     }
260 }
```

Проблем 6.4 Да се имплементира апликацијата за евиденција на работниот ангажман на вработени во една ИТ компанија.

За таа цел, да се имплементира класата `PayrollSystem` во којашто ќе се чуваат информации за вработени во компанијата. Постојат два типа на вработени `HourlyEmployee` и `FreelanceEmployee`. `HourlyEmployee` добиваат плата базирана на вкупниот број на изработени часови, додека пак `FreelanceEmployee` добиваат плата базирана на поените на тикетите што ги решиле. За класата `PayrollSystem` да се имплементираат:За класата `PayrollSystem` да се имплементираат:

- `PayrollSystem(Map<String,Double> hourlyRateByLevel, Map<String, Double> ticketRateByLevel)` - конструктор со два аргументи. Првиот аргумент означува колку е саатницата за соодветно ниво за вработените коишто земаат плата по час работа, а вториот аргумент означува колку е платата по

поен од тикет за соодветното ниво за фриленсерите.

- `void readEmployeesData (InputStream is)` - метод за читање на податоците за вработените во компанијата, при што за секој вработен податоците се дадени во нов ред.
- `Employee createEmployee (String line)` - метод што врз основа на влезен стринг (еден читан ред) во кој се запишани информациите за даден вработен, ќе креира и врати објект од класа Employee. Податоците за вработените се внесуваат во следниот формат:
  - Доколку вработениот е `HourlyEmployee`:
    `H;ID;level;hours; bonus`
  - Доколку вработениот е `FreelanceEmployee`:
    `F;ID;level;ticketPoints1;ticketPoints2;...;ticketPointsN; bonus`

  За вработените постојат два типа на бонуси:
  - Фиксен паричен бонус (запишан како бројка).
    пр. H;ID;level;hours; 100
    (во овој случај добива фиксен бонус на плата од 100$)
  - Процентуален паричен бонус (запишан како број со знак процент).
    пр. F;ID;level;ticketPoints1;ticketPoints2;...;ticketPointsN; 10%
    (во овој случај добива процентуален бонус од 10% од неговата плата).
- Во претходниот метод, со исклучок од тип `BonusNotAllowedException` да се спречи креирање на вработен на којшто му е доделен фиксен бонус поголем од 1000$ или процентуален бонус поголем од 20(дваесет)% .
- `Map<String, Double> getOvertimeSalaryForLevels ()` - метод којшто ќе врати мапа каде што клучот е нивото на вработениот, а вредноста е вкупниот износ којашто компанијата го исплатила за прекувремена работа за вработените од тоа ниво.
- `void printStatisticsForOvertimeSalary ()` - метод којшто ќе испечати статистики (минимум, максимум, сума, просек) на исплатените додатоци за прекувремена работа на сите вработени во компанијата.
- `Map<String, Integer> ticketsDoneByLevel()` - метод којшто ќе врати мапа каде што клучот е нивото на вработените, а вредноста е бројот на поени за тикети што се сработени од вработените од соодветното ниво.
- `Collection<Employee> getFirstNEmployeesByBonus (int n)` - метод којшто ќе врати сортирана колекција од првите n вработени сортирани во опаѓачки редослед според бонусот којшто го добиле на платата.

Во оваа задача повторно се употребува шаблонот за дизајн на софтвер Декоратор. Дефиниран е `Employee` интерфејсот и истиот има две конкретни имплементации: `HourlyEmployee` и `FreelanceEmployee` . Целта е да се декорира објект од тип `Employee` со дополнително својство - бонус на плата. Бонусот може да биде фиксен износ којшто се додава на платата или пак, процентуален бонус што се пресметува на основната плата. Се дефинира апстракната класа `BonusDecorator` што го имплементира интерфејсот `Employee` и чува инстанца од тип `Employee` , односно вработениот кој ќе биде декориран со соодветниот бонус. Во апстрактната класа `BonusDecorator` се имплементирани поголемиот дел од методите од интерфејсот `Employee` , со исклучок на методите `getBonus()` и `calculateSalary()` , бидејќи нивната имплементација зависи од кон-кретниот декоратор.

Од класата `BonusDecorator` се изведени декораторите за фиксни бонуси и проценти

туални бонуси (`FixedBonusDecorator` и `PercentageBonusDecorator`). Соодветно, во овие два декоратори се имплементирани методите `getBonus()` и `calculateSalary()` .

Креирањето на објекти од тип `Employee` што се декорирани со бонус декоратор се прави на следниот начин:

```
Employee baseEmployee = new HourlyEmployee("testID", "level1", 11, 43);
Employee decoratedEmployee = new FixedBonusDecorator(baseEmployee, 100);
Employee decoratedEmployee2 = new PercentageBonusDecorator(baseEmployee,5);
```

Целосното решение на задачата е дадено во продолжение.

```
import java.util.*;
import java.util.stream.Collectors;

class BonusNotAllowedException extends Exception {
    BonusNotAllowedException(String bonus) {
        super(String.format("Bonus of %s is not allowed", bonus));
    }

}

interface Employee {
    double calculateSalary();

    double getBonus();

    double getOvertime();

    int getTicketsCount();

    void updateBonus(double amount);

    String getLevel();
}

abstract class EmployeeBase
        implements Employee, Comparable<EmployeeBase> {
    String ID;
    String level;
    double rate;
    double totalBonus;


    public EmployeeBase(String ID, String level, double rate) {
        this.ID = ID;
        this.level = level;
        this.rate = rate;
        this.totalBonus = 0.0;
    }

    public String getLevel() {
        return level;
    }

    @Override
    public int compareTo(EmployeeBase o) {
        return Comparator.comparing(EmployeeBase::calculateSalary)
                .thenComparing(EmployeeBase::getLevel)
                .compare(this, o);
    }

```

```
51      @Override
52      public String toString() {
53          return String
54                  .format("Employee ID: %s Level: %s Salary: %.2f", ID,
55                          level, calculateSalary() + totalBonus);
56      }
57
58      @Override
59      public double getBonus() {
60          return totalBonus;
61      }
62
63      @Override
64      public void updateBonus(double amount) {
65          this.totalBonus += amount;
66      }
67  }
68
69  class HourlyEmployee extends EmployeeBase {
70      double hours;
71      double overtime;
72      double regular;
73
74      public HourlyEmployee(String ID, String level, double rate,
75                            double hours) {
76          super(ID, level, rate);
77          this.hours = hours;
78          overtime = Math.max(0, hours - 40);
79          regular = hours - overtime;
80      }
81
82      @Override
83      public double calculateSalary() {
84          return regular * rate + overtime * rate * 1.5;
85      }
86
87      @Override
88      public double getOvertime() {
89          return overtime * rate * 1.5;
90      }
91
92      @Override
93      public int getTicketsCount() {
94          return -1;
95      }
96
97      @Override
98      public String toString() {
99          return super.toString() + String.format(
100                 " Regular hours: %.2f Overtime hours: %.2f", regular,
101                 overtime);
102     }
103 }
104
105 class FreelanceEmployee extends EmployeeBase {
106     List<Integer> ticketPoints;
107
108     public FreelanceEmployee(String ID, String level, double rate,
109                              List<Integer> ticketPoints) {
110         super(ID, level, rate);
111         this.ticketPoints = ticketPoints;
```

```
112     }
113
114     @Override
115     public double calculateSalary() {
116         return ticketPoints.stream().mapToInt(tp -> tp).sum() * rate;
117     }
118
119     @Override
120     public double getOvertime() {
121         return -1;
122     }
123
124     @Override
125     public int getTicketsCount() {
126         return ticketPoints.size();
127     }
128
129     @Override
130     public String toString() {
131         return super.toString() +
132                 String.format(" Tickets count: %d Tickets points: %d",
133                         ticketPoints.size(),
134                         ticketPoints.stream().mapToInt(i -> i).sum()
135                 );
136     }
137 }
138
139 abstract class BonusDecorator implements Employee {
140     Employee employee;
141
142     public BonusDecorator(Employee employee) {
143         this.employee = employee;
144     }
145
146     @Override
147     public double getOvertime() {
148         return employee.getOvertime();
149     }
150
151     @Override
152     public int getTicketsCount() {
153         return employee.getTicketsCount();
154     }
155
156     @Override
157     public void updateBonus(double amount) {
158         employee.updateBonus(amount);
159     }
160
161     @Override
162     public String getLevel() {
163         return employee.getLevel();
164     }
165
166     @Override
167     public String toString() {
168         return employee.toString() +
169                 String.format(" Bonus: %.2f", getBonus());
170     }
171 }
172
```

```
173 class FixedBonusDecorator extends BonusDecorator {
174
175     double fixedAmount;
176
177     public FixedBonusDecorator(Employee employee,
178                                double fixedAmount) {
179         super(employee);
180         this.fixedAmount = fixedAmount;
181         employee.updateBonus(fixedAmount);
182     }
183
184     @Override
185     public double calculateSalary() {
186         double salaryWithoutBonus = employee.calculateSalary();
187         return salaryWithoutBonus + fixedAmount;
188     }
189
190     @Override
191     public double getBonus() {
192         return fixedAmount;
193     }
194 }
195
196
197 class PercentageBonusDecorator extends BonusDecorator {
198     double percent;
199     double bonus;
200
201     public PercentageBonusDecorator(Employee employee,
202                                     double percent) {
203         super(employee);
204         this.percent = percent;
205         bonus = employee.calculateSalary() * percent / 100.0;
206         employee.updateBonus(bonus);
207     }
208
209     @Override
210     public double calculateSalary() {
211         double salaryWithoutBonus = employee.calculateSalary();
212         return salaryWithoutBonus + bonus;
213     }
214
215     @Override
216     public double getBonus() {
217         return bonus;
218     }
219 }
220
221 class EmployeeFactory {
222     public static Employee createEmployee(String line,
223                                           Map<String, Double> hourlyRate,
224                                           Map<String, Double> ticketRate)
225             throws BonusNotAllowedException {
226         String[] partsBySpace = line.split("\\s+");
227
228         Employee e = createSimpleEmployee(partsBySpace[0], hourlyRate,
229                 ticketRate);
230
231         if (partsBySpace.length > 1) {
232             if (partsBySpace[1].contains("%")) { //percentage bonus
233                 double percentage = Double.parseDouble(partsBySpace[1]
```

```
                        .substring(0, partsBySpace[1].length() - 1));
                if (percentage > 20)
                    throw new BonusNotAllowedException(
                            partsBySpace[1]);
                e = new PercentageBonusDecorator(e, percentage);

            } else { //fixed bonus
                double bonusAmount =
                        Double.parseDouble(partsBySpace[1]);
                if (bonusAmount > 1000)
                    throw new BonusNotAllowedException(
                            partsBySpace[1] + "$");
                e = new FixedBonusDecorator(e, bonusAmount);

            }
        }
        return e;
    }

    public static Employee createSimpleEmployee(String subline,
                                        Map<String, Double> hourlyRate,
                                        Map<String, Double> ticketRate) {
        String[] parts = subline.split(";");
        String id = parts[1];
        String level = parts[2];
        if (parts[0].equalsIgnoreCase("H")) { //hourly
            double hours = Double.parseDouble(parts[3]);
            return new HourlyEmployee(id, level,
                    hourlyRate.get(level), hours);
        } else { //freelance
            List<Integer> ticketPoints = Arrays.stream(parts).skip(3)
                    .map(Integer::parseInt)
                    .collect(Collectors.toList());
            return new FreelanceEmployee(id, level,
                    ticketRate.get(level), ticketPoints);
        }
    }
}

class PayrollSystem {
    Map<String, Double> hourlyRate;
    Map<String, Double> ticketRate;
    List<Employee> employees;

    public PayrollSystem(Map<String, Double> hourlyRate,
                         Map<String, Double> ticketRate) {
        this.hourlyRate = hourlyRate;
        this.ticketRate = ticketRate;
        employees = new ArrayList<>();
    }

    Employee createEmployee(String line)
            throws BonusNotAllowedException {
        Employee e = EmployeeFactory
                .createEmployee(line, hourlyRate, ticketRate);
        employees.add(e);
        return e;
    }

    Map<String, Double> getOvertimeSalaryForLevels() {
        Map<String, Double> result =
```

```
                employees.stream().collect(Collectors.groupingBy(
                        Employee::getLevel,
                        Collectors
                                .summingDouble(Employee::getOvertime)
                ));

        List<String> keysWithZeros = result.keySet().stream()
                .filter(key -> result.get(key) == -1)
                .collect(Collectors.toList());

        keysWithZeros.forEach(result::remove);

        return result;
    }

    void printStatisticsForOvertimeSalary() {
        DoubleSummaryStatistics dss = employees.stream()
                .filter(e -> e.getOvertime() != -1)
                .mapToDouble(Employee::getOvertime)
                .summaryStatistics();

        System.out
                .printf("Statistics for overtime salary: Min: %.2f " +
                                "Average: %.2f Max: %.2f Sum: %.2f",
                        dss.getMin(), dss.getAverage(), dss.getMax(),
                        dss.getSum());
    }

    Map<String, Integer> ticketsDoneByLevel() {
        return employees.stream()
                .filter(e -> e.getTicketsCount() != -1)
                .collect(Collectors.groupingBy(
                        Employee::getLevel,
                        Collectors
                                .summingInt(Employee::getTicketsCount)
                ));
    }

    Collection<Employee> getFirstNEmployeesByBonus(int n) {
        return employees.stream()
                .sorted(Comparator.comparing(Employee::getBonus)
                        .reversed())
                .limit(n)
                .collect(Collectors.toList());
    }

}

public class PayrollSystemTest {

    public static void main(String[] args) {

        Map<String, Double> hourlyRateByLevel = new LinkedHashMap<>();
        Map<String, Double> ticketRateByLevel = new LinkedHashMap<>();
        for (int i = 1; i <= 10; i++) {
            hourlyRateByLevel.put("level" + i, 11 + i * 2.2);
            ticketRateByLevel.put("level" + i, 5.5 + i * 2.5);
        }

        Scanner sc = new Scanner(System.in);

```

```
356         int employeesCount = Integer.parseInt(sc.nextLine());
357
358         PayrollSystem ps = new PayrollSystem(hourlyRateByLevel,
359                 ticketRateByLevel);
360         Employee emp = null;
361         for (int i = 0; i < employeesCount; i++) {
362             try {
363                 emp = ps.createEmployee(sc.nextLine());
364             } catch (BonusNotAllowedException e) {
365                 System.out.println(e.getMessage());
366             }
367         }
368
369         int testCase = Integer.parseInt(sc.nextLine());
370
371         switch (testCase) {
372             case 1: //Testing createEmployee
373                 if (emp != null)
374                     System.out.println(emp);
375                 break;
376             case 2: //Testing getOvertimeSalaryForLevels()
377                 ps.getOvertimeSalaryForLevels()
378                         .forEach((level, overtimeSalary) -> {
379                             System.out
380                                 .printf("Level: %s Overtime " +
381                                                 "salary: %.2f\n",
382                                         level, overtimeSalary);
383                         });
384                 break;
385             case 3: //Testing printStatisticsForOvertimeSalary()
386                 ps.printStatisticsForOvertimeSalary();
387                 break;
388             case 4: //Testing ticketsDoneByLevel
389                 ps.ticketsDoneByLevel()
390                         .forEach((level, overtimeSalary) -> {
391                             System.out
392                                 .printf("Level: %s Tickets by level: %d\n",
393                                         level, overtimeSalary);
394                         });
395                 break;
396             case 5: //Testing getFirstNEmployeesByBonus (int n)
397                 ps.getFirstNEmployeesByBonus(
398                         Integer.parseInt(sc.nextLine()))
399                         .forEach(System.out::println);
400                 break;
401         }
402
403     }
404 }
```

Индекс на термини

Е-издание: `http://www.ukim.edu.mk/mk_content.php?meni=53&glavno=41`