

1. Napišite klasu `Account` za baratanje s bankovnim računima. Za svaki račun pohranjuju se informacije o trenutnom stanju računa (atribut `balance`) i trenutnom ukupnom broju provedenih transakcija (atribut `transactions`). Klasa ima sljedeće metode:

- konstruktor s jednim argumentom koji služi za postavljanje početnog stanja računa, a broj transakcija se prilikom kreiranja računa postavlja na 0.
- getterske metode za oba atributa.
- `void deposit(double amount)` metoda za uplatu novca i `void withdraw(double amount)` metoda za podizanje novca. Broj transakcija se uvećava pri svakoj uplati i podizanju novca.
- `void onEndOfMonth()` metoda se poziva nad objektom jednom na kraju svakog mjeseca. U metodi se najprije poziva metoda `endOfMonthCharge` te se broj transakcija postavlja na 0.
- `void endOfMonthCharge()` je apstraktna metoda. Služi baci za računanje i naplatu troškove usluge vođenja računa. Metoda "skida" (withdraw) novce s računa. Postoje 3 vrste računa. Za svaku vrstu postoji drugačija politika naplate troškova.

2. Napišite klasu `FixedFeeAccount` koja je `Account`. Klasa ima konstruktor s jednim argumentom preko kojeg se postavlja početno stanje računa. Za ovu klasu vrijedi da se troškovi usluge naplaćuju tj. skidaju s računa u fiksnom iznosu od 10 Kn mjesечно.

3. Napišite klasu `WithdrawAccount` koja je `Account`. Klasa ima atribut `withdrawCounter` i konstruktor s jednim argumentom preko kojeg se postavlja početno stanje računa, a u konstruktoru se postavlja i vrijednost atributa `withdrawCounter` na 0. Klasa mora nadjačati metodu `withdraw` u kojoj se inkrementira vrijednost brojača `withdrawCounter` (broj isplate). Za ovu klasu vrijedi da se troškovi usluge naplaćuju prema broju isplate u mjesecu i to 0.10 Kn po isplati. Ako je `withdrawCounter > 0` s računa se isplaćuje iznos od (`withdrawCounter * 0.1`). Ova zadnja isplata u korist banke se ne naplaćuje. Na kraju se `withdrawCounter` resetira na 0.

4. Napišite klasu `VipAccount` koja je `Account`. Klasa ima konstruktor s jednim argumentom preko kojeg se postavlja početno stanje računa. Klasa mora nadjačati metodu `deposit` jer se kod ovog računa potiču uplate na način da ukoliko je iznos uplate veći od 10000 Kn tada se uplata uvećava za 10 Kn. Kod ovog računa nema naplate troškova usluge na kraju mjeseca.

Napomena: sve klase napisati bez modifikatora vidljivosti i bez navođenja paketa kojem pripadaju te ih sve zajedno kopirati u prostor za rješenje. Sve metode svih klasa imaju public vidljivost. Na vrhu možete dodavati proizvoljne importe.

```
104 }  
105 }  
106 }  
107 }  
108 }  
109 }  
110 }  
111 }  
112 }  
113 }  
114 }  
115 }  
116 }  
117 }
```

Napisati apstraktnu klasu `Reference` koja predstavlja referencu u znanstvenoj literaturi i ima sljedeće privatne atribute: `authors` tipa `Author[]`, `title` tipa `String`, `year` tipa `int` i `label` tipa `String`. Ova klasa je opremljena konstruktorom i `getterima` za inicijalizaciju i pristup vrijednosti navedenih atributa. Osim toga, ova klasa implementira sučelje `Citable` koje ima jednu jedinu metodu `getCitation` koja vraća citat reference koji se koristi pri navođenju reference u znanstvenom članku. Programski kod klase `Author` i sučelja `Citable` (koje **ne treba implementirati**) je prikazan u nastavku:

```
class Author {  
    String name;  
    String surname;  
  
    public Author(String name, String surname) {  
        this.name = name;  
        this.surname = surname;  
    }  
  
    public String getName() {  
        return name;  
    }  
  
    public String getSurname() {  
        return surname;  
    }  
}  
  
interface Citable {  
    String getCitation();  
}
```

Nakon toga je potrebno napisati klasu `Book` koja nasljeđuje klasu `Reference`, a ima dodatne privatne atribute `publisher` i `address` tipa `String`. Ova klasa je opremljena konstruktorom i `getterima` za inicijalizaciju i pristup vrijednosti navedenih atributa. Pozivom metode `getCitation` nad ovom klasom se treba vratiti jedna linija teksta koja je formirana od sljedećih dijelova bez razmaka:

- ime klase
- zadnje dvije znamenke godine izdavanja
- prezimena svih autora bez razmaka
- znak ''
- atribut `label` iz nadklase `Reference`

Zatim je potrebno napisati klasu `InProceedings` koja nasljeđuje klasu `Book`, a ima dodatne privatne atribute `booktitle` tipa `String` te `startPage` i `endPage` tipa `int`. Ova klasa je također opremljena konstruktorom i `getterima` za inicijalizaciju i pristup vrijednosti navedenih atributa. Pozivom metode `getCitation` nad ovom klasom se treba vratiti jedne linija teksta koja je formirana od sljedećih dijelova bez razmaka:

- ime klase
- prvo slovo prezimena svih autora bez razmaka
- godina izdavanja
- znak ''
- atribut `label` iz nadklase `Reference`

Primjer isječka koda metode `main` koja demonstrira upotrebu ovih klasa je sljedeći:

```
Author ws = new Author("Walter", "Savitch");  
Author km = new Author("Kenrick", "Mock");  
  
Book book = new Book(new Author[]{ws, km}, "Absolute Java", 2015,  
                    "0134041674", "Pearson", "PEL, Edinburgh Gate, Harlow, Essex CM20 2JE, England");  
  
System.out.println(book.getCitation());//Book15SavitchMock-0134041674  
  
Author gb = new Author("Gilad", "Bracha");  
Author mo = new Author("Martin", "Odersky");  
Author ds = new Author("David", "Stoutamire");  
Author pw = new Author("Philip", "Wadler");  
  
InProceedings inProceedings = new InProceedings(new Author[]{gb, mo, ds, pw},  
                                                "Making the future safe for the past: Adding genericity to the Java programming language", 1998,  
                                                "286957", "ACM",  
                                                "ACM, 2 Penn Plaza, Suite 701, New York, NY 10121-0701",  
                                                "Proceedings of the 13th ACM SIGPLAN conference on Object-oriented programming, systems, languages and  
                                                applications", 183, 200);  
  
System.out.println(inProceedings.getCitation());//InProceedingsBOSW1998-286957
```

Napomena: Klase `Reference`, `Book` i `InProceedings` napisati bez modifikatora vidljivosti i kopirati u prostor za rješenje bez navođenja paketa kojem pripadaju. Za klase nije potrebno provjeravati jesu li predane ispravne vrijednosti argumenata.

```
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13
```

Run
Save

1. Napišite klasu `Account` za baratanje s bankovnim računima. Za svaki račun pohranjuju se informacije o trenutnom stanju računa (atribut `balance`) i trenutnom ukupnom broju provedenih transakcija (atribut `transactions`). Klasa ima sljedeće metode:
 - konstruktor s jednim argumentom koji služi za postavljanje početnog stanja računa, a broj transakcija se prilikom kreiranja računa postavlja na 0.
 - getterske metode za oba atributa.
 - `void deposit(double amount)` metoda za uplatu novca i `void withdraw(double amount)` metoda za podizanje novca. Broj transakcija se uvećava pri svakoj uplati i podizanju novca.
 - `void onEndOfMonth()` metoda se poziva nad objektom jednom na kraju svakog mjeseca. U metodi se najprije poziva metoda `endOfMonthCharge` te se broj transakcija postavlja na 0.
 - `void endOfMonthCharge()` je apstraktna metoda. Služi banci za računanje i naplatu troškove usluge vođenja računa. Metoda "skida" (`withdraw`) novce s računa. Postoje 3 vrste računa. Za svaku vrstu postoji drugačija politika naplate troškova.
2. Napišite klasu `FixedFeeAccount` koja je `Account`. Klasa ima konstruktor s jednim argumentom preko kojeg se postavlja početno stanje računa. Za ovu klasu vrijedi da se troškovi usluge naplaćuju tj. skidaju s računa u fiksnom iznosu od 10 Kn mjesечно.
3. Napišite klasu `WithdrawAccount` koja je `Account`. Klasa ima atribut `withdrawCounter` i konstruktor s jednim argumentom preko kojeg se postavlja početno stanje računa, a u konstruktoru se postavlja i vrijednost atributa `withdrawCounter` na 0. Klasa mora nadjačati metodu `withdraw` u kojoj se inkrementira vrijednost brojača `withdrawCounter` (broj isplata). Za ovu klasu vrijedi da se troškovi usluge naplaćuju prema broju isplata u mjesecu i to 0.10 Kn po isplati. Ako je `withdrawCounter > 0` s računa se isplaćuje iznos od `(withdrawCounter * 0.1)`. Ova zadnja isplata u korist banke se ne naplaćuje. Na kraju se `withdrawCounter` resetira na 0.
4. Napišite klasu `VipAccount` koja je `Account`. Klasa ima konstruktor s jednim argumentom preko kojeg se postavlja početno stanje računa. Klasa mora nadjačati metodu `deposit` jer se kod ovog računa potiču uplate na način da ukoliko je iznos uplate veći od 10000 Kn tada se uplata uvećava za 10 Kn. Kod ovog računa nema naplate troškova usluge na kraju mjeseca.

Napomena: sve klase napisati bez modifikatora vidljivosti i bez navođenja paketa kojem pripadaju te ih sve zajedno kopirati u prostor za rješenje. Sve metode svih klasa imaju public vidljivost. Na vrhu možete dodavati proizvoljne importe.

Napisati klasu *KeyValue* parametriziranu po 2 parametra koja predstavlja parove ključ vrijednost. Za klasu *KeyValue* definirati *gettere* i *settere* za ključ i vrijednost te konstruktor koji prima inicijalne vrijednosti ključa i vrijednosti. Npr. sljedećim retkom

```
KeyValue<String, Integer> kv = new KeyValue<>("pero", 5);
```

stvorit će se novi objekt u kojem će ključ biti **pero**, a za vrijednost **5**.

Nakon toga iz klase *KeyValue* izvesti klasu *SingleTypeKeyValue* koja predstavlja parametrizirani par ključ-vrijednost u kojem su ključ i vrijednost istog tipa. Klasa *SingleTypeKeyValue* ne definirane nikakve dodatne metode u odnosu na *KeyValue*.

Nakon toga iz klase *SingleTypeKeyValue* izvesti klasu *XY* koja predstavlja par vrijednosti tipa *Double*. Smisao ove klase je da predstavlja vrijednost neke funkcije za neku vrijednost x koja je zapisana kao ključ.

U klasi *XY* napisati statičku metodu `public static boolean isIncreasingFunction(XY[] data)` koja za primljeno polje provjerava jesu li parovi koji predstavljaju parove (x,y) neke funkcije takvi da je funkcija rastuća. Poredak elemenata u polju može biti proizvoljan i **polje se ne smije mijenjati**.

Primjer: Ako je ulazno poje sadržavalo slijedeće vrijednosti za XY **(6, 8), (7, 8), (9.1, 13), (-3, -2), (-1, 4)**, funkcija je rastuća (primjetiti da ne mora biti stroga rastuća), a npr. za poje sadržaje **(6, 8), (7.5, 12), (9, 9), (10, 10), (11, 11)** ukazuje da funkcija nije rastuća.

Ako je u metodu *isIncreasingFunction* poslano prazno polje ili polje sa samo jednim elementom, metoda mora vratiti *true*. Možete prepostaviti da su vrijednost za x ispravno zadane (tj. ne može se dvaput pojaviti ista vrijednost za x).

Napomena: Klase *KeyValue*, *SingleTypeKeyValue* i *XY* napisati bez modifikatora vidljivosti i kopirati u prostor za rješenje bez navođenja paketa kojem pripada. Na vrhu možete dodavati *importe* iz standardnih Javinih biblioteka.

```
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13
```

Run

Save

Napisati program koji će provjeriti osnovnu validnost kreditne kartice. Potrebno je napisati klasu:

```
public static class CreditCard
```

i u njoj metode:

```
public static boolean checkCreditCard(String stringPAN, String cardType)
public static int getChecksum(Long numbPAN)
```

Metoda `checkCreditCard` provjerava PAN (16-znamenkasti broj na kartici) i tip kartice (Stringovi koji mogu biti vrijednosti "Diners", "Visa", "MasterCard" ili "AmEx").

Provjera se obavlja na sljedeći način:

- Ako je duljina broja neispravna, baca se iznimka: `InvalidPANLengthException`
- Ako se u ulaznom stringu pojavio nedozvoljeni znak (nešto što nije znamenka), baca se iznimka `InvalidPANCharacterException`
- Pomoću metode `getChecksum` (koja je objašnjena niže) potrebno je odrediti odgovara li očekivana znamenka unesenoj zadnjoj znamenki. Ako ne, baca se `InvalidPANChecksumException`.
- Ako prva znamenka ne odgovara pripadajućoj kartici, baca se `InvalidCardException`

Za ispravnu karticu, metoda mora vratiti `true`.

Prve znamenke i njihove odgovarajuće kartice su:

- 3 za Diners
- 4 za Visa
- 5 za MasterCard
- 6 za AmEx

Metoda `getChecksum` za ulazni parametar tipa `long` sumira sve znamenke osim zadnje. Zatim vraća zadnju znamenku broja izračunate sume.

NIJE potrebno pisati programski kod za klase:

```
InvalidCardException
InvalidPANLengthException
InvalidPANCharacterException
InvalidPANLengthException
```

A screenshot of a code editor interface. On the left, there is a vertical line number column from 1 to 13. The main area contains the following code:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
```

On the right side of the editor, there are two buttons: a green "Run" button and a white "Save" button.

```
KeyValue<String, Integer> kv = new KeyValue<>("pero", 5);
```

Nakon toga iz klase `KeyValuePair` izvesti klasu `SingleTypeKeyValuePair` koja

SingleTypeKeyValue ne definirane nikakve dodatne metode u odnosu na *KeyValue*.

Nakon toga iz klase `SingleHyperkeyValue` izvesti klasu `XY` koja predstavlja par vrijednosti tipa `Double`. Smisao ove klase je da predstavlja vrijednost neke funkcije za neku vrijednost x koja je zapisana kao ključ.

U klasi `XY` napisati statičku metodu `public static boolean isIncreasingFunction(XY[] data)` koja za primljeno polje provjerava jesu li parovi koji predstavljaju parove (x,y) neke funkcije takvi da je funkcija rastuća. Poredak elemenata u polju može biti proizvoljan i **polje se ne smije mijenjati**.

Ako je u metodu *isIncreasingFunction* poslano prazno polje ili polje sa samo jednim elementom, metoda mora vratiti *true*. Možete prepostaviti da su vrijednost za ispravno zadane (tj. ne može se dyanut pojaviti ista vrijednost za x).

Napomena: Klase *KeyValue*, *SingleTypeKeyValue* i *XY* napisati bez modifikatora vidljivosti i kopirati u prostor za rješenje bez navođenja paketa kojem pripada. Na vrhu možete dodavati importe iz standardnih Java-ovih biblioteka.

Run

Save

Digitized by srujanika@gmail.com

5
5
5
5
5
5
5
60
61
62
63
64
65

Con

eKeyValue is public,
eTypeKeyValue
in a file named
be converted to
oatible types: int
ain.java:141: error:
ethod getKey()
getKey(),
error: cannot find
location: class XY
y())); ^ symbol:

ation: class XY
alue())); ^
oes: int cannot
oes: int cannot
: incompatible
in.java:201:
data[2*i + 1]);
new
onverted to
ol if
method
<

Status: Com

| Language | Compiler |
|-----------|----------|
| /usr/lib/ | |

Main.java:1^

Main.java:1^

Main.java:1^

symbol:
location:
Main.java:1^

Napisati program koji će provjeriti osnovnu validnost kreditne kartice. Potrebno je napisati klasu:

```
public static class CreditCard
```

i u njoj metode:

```
public static boolean checkCreditCard(String stringPAN, String cardType)
public static int getChecksum(Long numbPAN)
```

Metoda `checkCreditCard` provjerava PAN (16-znamenkasti broj na kartici) i tip kartice (Stringovi koji mogu biti vrijednosti "Diners", "Visa", "MasterCard" ili "AmEx").

Provjera se obavlja na sljedeći način:

- Ako je duljina broja neispravna, baca se iznimka: `InvalidPANLengthException`
- Ako se u ulaznom stringu pojavio nedozvoljeni znak (nešto što nije znamenka), baca se iznimka `InvalidPANCharacterException`
- Pomoću metode `getChecksum` (koja je objašnjena niže) potrebno je odrediti odgovara li očekivana znamenka unesenoj zadnjoj znamenki. Ako ne, baca se `InvalidPANChecksumException`.
- Ako prva znamenka ne odgovara pripadajućoj kartici, baca se `InvalidCardException`

Za ispravnu karticu, metoda mora vratiti `true`.

Prve znamenke i njihove odgovarajuće kartice su:

- 3 za Diners
- 4 za Visa
- 5 za MasterCard
- 6 za AmEx

Metoda `getChecksum` za ulazni parametar tipa `long` sumira sve znamenke osim zadnje. Zatim vraća zadnju znamenku broja izračunate sume.

NIJE potrebno pisati programski kod za klase:

```
InvalidCardException
InvalidPANLengthException
InvalidPANCharacterException
InvalidPANLengthException
```

The screenshot shows a code editor interface with a dark theme. On the left, there is a vertical line number column from 1 to 13. The main area is empty, representing a new Java file. In the top right corner, there are two buttons: a green 'Run' button and a white 'Save' button.

Napisati klasu `InvalidPeppaPigCharacterException` koja naslijeduje `RuntimeException` i sadrži konstruktor kojim se može postaviti poruke iznimke.

Napisati klasu `CartoonChecker` i u njoj javnu statičku metodu `processCharacter` koja prima jedan podatak tipa String i provjerava postoji li taj string u polju stringova koji se nalaze u klasi `Peppa`. Referenca na polje se može dobiti preko statičke metode `Peppa.getAllCharacters()`.

Ako predano ime `postoji` u polju stringova, metoda `processCharacter` na standardni izlaz ispisuje poruku `<IME> JE lik u crtanim filmu Peppa Pig`

Ako predano ime `ne postoji` u listi, metoda treba baciti `InvalidPeppaPigCharacterException` s porukom `<IME> NIJE lik u crtanim filmu Peppa Pig`.

Npr. nek polje dobiveno s `Peppa.getAllCharacters()` sadrži samo dva stringa, `{"Peppa", "George"}`. Ako iz glavnog programa metodu `processCharacter` pozivamo redom s parametrima "Peppa", "George" i "Traktor Tom", na standardni izlaz će se ispisati:

```
Peppa JE lik u crtanim filmu Peppa Pig
```

```
1
2 class InvalidPeppaPigCharacterException extends RuntimeException{
3
4     /**
5      *
6     */
```

Run

Save

Potrebno je napraviti klasu `JavaProjectFileVisitor` koja nasljeđuje `SimpleFileVisitor<Path>`. Klasa mora imati navedene public metode:

```
public JavaProjectFileVisitor(Set<String> extensionFilter);
public Map<String, Set<String>> getProjectNamesPerExtension()
```

Kroz konstruktor klasa dobiva skup koji sadrži popis ekstenzija. Metoda `getProjectNamesPerExtension()` se izvodi isključivo za datoteke koje NE odgovaraju filteru ekstenzija (tj. njihove ekstenzije se ne nalaze u skupu). Ključevi u mapi su ekstenzije datoteka (npr. ".txt", ".java", ".class", ...) dok su vrijednosti skupovi koji sadrže imena datoteka. Možete prepostaviti da ne postoje dvije datoteke istog imena.

Pokretanjem koda nad direktorijem koji je prikazan u nastavku očekujemo prikazane rezultate:

```
JavaProjectFileVisitor visitor = new JavaProjectFileVisitor(Set.of(".pdf"));
File f = new File("folder1");
Files.walkFileTree(f.toPath(), visitor);
Map<String, Set<String>> map = visitor.getProjectNamesPerExtension();
for(String key : map.keySet()){
    Set<String> list = map.get(key);
    for(String s : list)
        System.out.println("Key: " + key + " - " + s);
}
```

Ispis:

```
Key: .txt - a.txt
Key: .txt - b3.txt
Key: .java - b1.java
Key: .java - b2.java
```

Izgled direktorija:

```
+--folder1
|   +-a.txt (10 B)
|   +-folder2
|   |   +-b1.java (11 B)
|   |   +-b2.java (15 B)
|   |   +-b3.txt (100 B)
|   |   +-b4.pdf (150 B)
|   |   +-folder3
```

Napomena: Klasu `JavaProjectFileVisitor` napisati bez modifikatora vidljivosti i kopirati u prostor za rješenje bez navođenja paketa kojem pripada. Na vrhu možete dodavati *importe* iz standardnih Javinih biblioteka. Importe na klase iz predavanja, auditornih i pripreme obrisati prije predaje rješenja. Uočite da programski kod koji predajete **ne smije ništa ispisivati**, nego samo vraća tražene podatke.

Napišite klasu `TextFileAnalyser` sa statičkom metodom koja detektira na kojem je jeziku datoteka:

```
public static Language detectLanguage(Path file) throws IOException
```

Ova metoda može detektirati 3 jezika:

- hrvatski - ako datoteka ima samo hrvatska slova: šđčćžŠĐĆĆŽ
- njemački - ako datoteka ima samo njemačka slova: äöüÄÖÜß
- engleski - ako datoteka ima samo engleska slova (ASCII)
- nepoznato - ako ima slova iz više jezika

Treba napraviti enumeraciju `Language` koja ima sljedeće vrijednosti: `HR`, `DE`, `EN` i `UNKNOWN`.

Npr. ako ima slova iz HR i DE onda je rezultat nepoznato, a ako ima slova iz EN i HR onda je rezultat HR.

Primjer korištenja:

```
Path.of("example.txt");
TextFileAnalyser.detectLanguage(inputFile);
```

Napomena: Sve klase i sučelja imaju vidljivost postavljenu na *package private*.

Napomena: Prije predaje iz Vašeg koda izbacite sve ispise na standardni izlaz kako bi testovi ispravno radili.

41 }

| Correct | Incorrect | Partial | Unanswered | Score | Score % | Hint |
|---------|-----------|---------|------------|-------|---------|---------------------|
| ✓ | | | | 1 | 100 | Correct. Well done! |

L

J

T

U sljedećem programskom odsječku, ulazna lista redom predstavlja znamenke nekog decimalnog broja oblika A.BCDE... (npr. 3.1425976).

Nadopunite odsječak koda

```
class Solution{

    public static Integer getPiSequenceBreakpoint(List<Integer> input){
        return IntStream.range(0, input.size()) /*finish with your code*/

    }

}
```

prema sljedećim zahtjevima:

- metoda za ulaznu listu brojeva vraća redni broj pozicije (počevši od 0) prve znamenke koja ne odgovara znamenici u broju Pi.
- U slučaju znamenka razlike ne postoji baca se iznimka `NoSuchElementException`
- pretpostavite da su dostupni sljedeći importi: `import java.util.*; import java.util.stream.Stream; import java.util.stream.IntStream; import java.util.stream.Collectors; import java.lang.Math.*;`
- uputa za predaju: u Edgar zalijepiti samo dio koji nedostaje u odsječku (tj. dio koji mora doći umjesto `/* finish with your code */`).

Primjer

Za listu s brojevima `3, 1, 4, 1, 5, 9, 2, 5, 5, 5` vratit će se `7`.

NAPOMENA:

- početni dio koda `IntStream.range(0, input.length)` generira tok prirodnih brojeva koji odgovaraju pozicijama u ulaznoj listi. U ostatku izraza ih možete koristiti kao indeksere.

The screenshot shows a dark-themed Java code editor. On the left, there is a vertical line of numbers from 1 to 13, likely representing line numbers. The main area contains the provided code snippet. In the bottom right corner of the code area, there are two buttons: a green "Run" button and a white "Save" button with black text. The code itself is as follows:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
```

Na raspolaganju imate razred `Garden`.

```
class Garden {  
    private String name; // garden's name  
    private List<Plant> plants; // plants in the garden  
  
    public Garden(String name, List<Plant> plants) {  
        this.name = name;  
        this.plants = plants;  
    }  
  
    // GETTERS & SETTERS EXIST  
  
    @Override  
    public String toString() {  
        return "Garden [name=" + name + ", plants=" + plants + "]";  
    }  
}
```

Potrebno je dopuniti i predati razred `Plant` tako da implementirat metode `equals` i `hashCode`. Biljke su jednake ako imaju jednak latinski naziv. U razred trebate dodati i geterske i seterske metode.

```
class Plant {  
    private String latinName; // latin name for the plant  
    private String croName; // croatian name for the plant  
  
    public Plant(String latinName, String croName) {  
        super();  
        this.latinName = latinName;  
        this.croName = croName;  
    }  
  
    // GETTERS & SETTERS  
  
    @Override  
    public String toString() {  
        return "Plant [latinName=" + latinName + ", croName=" + croName + "]";  
    }  
}
```

Također, potrebno je implementirati i predati razred `Task` u kojem su definirane sljedeće dvije metode:

```
public static BiConsumer<Garden, List<Plant>> deletePlantsFunction()
```

- Metoda vraća `BiConsumer` koji djeluje nad objektom tipa `Garden` (predstavlja vrt) i `List<Plant>` (predstavlja uginule biljke).
- Consumer iz liste biljaka objekta `Garden` briše biljke (ako ih posjeduje) sadržane u listi `List<Plant>`.
- `BiConsumer` implementirati **lambda** izrazom.

```
public static BiFunction<Garden, Plant, Boolean> checkIfPlantIsInGardenFunction()
```

- Metoda vraća `BiFunction` koja djeluje nad objektom tipa `Garden` (predstavlja vrt) i objektom tipa `Plant` (predstavlja biljku).
- Funkcija provjerava nalazi li se u vrtu navedena biljka.
- `BiFunction` implementirati **lambda** izrazom.

NAPOMENA: Razrede `Plant` i `Task` napisati bez modifikatora vidljivosti, kao package-private.

PRIMJER IZVOĐENJA:

```
List<Plant> pList1 = new ArrayList<>();  
pList1.add(new Plant("Leucanthemum vulgare", "Ivancica"));  
pList1.add(new Plant("Pinus cembra", "Limba"));  
Garden g1 = new Garden("Bakin", pList1);  
  
List<Plant> pList2 = new ArrayList<>();  
pList2.add(new Plant("Calendula officinalis", "Neven"));  
pList2.add(new Plant("Pinus cembra", "Limba"));  
  
BiFunction<Garden, Plant, Boolean> checkBiFunction = Task.checkIfPlantIsInGardenFunction();  
System.out.println(checkBiFunction.apply(g1, new Plant("Broussonetia papyrifera", "Dudovac")));  
System.out.println(checkBiFunction.apply(g1, new Plant("Pinus cembra", "Limba")));  
  
System.out.println(g1.toString());  
BiConsumer<Garden, List<Plant>> checkBiConsumer = Task.deletePlantsFunction();  
checkBiConsumer.accept(g1, pList2);  
System.out.println(g1.toString());
```

IZLAZ:

```
false  
true  
Garden [name=Bakin, plants=[Plant [latinName=Leucanthemum vulgare, croName=Ivancica], Plant [latinName=Pinus cembra, croName=Limba]]]  
Garden [name=Bakin, plants=[Plant [latinName=Leucanthemum vulgare, croName=Ivancica]]]
```

```
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13
```

Run
Save

Potrebno je implementirati klasu `FruitFinder` kojom se upravlja kolekcija voća.

Podaci o voću su pohranjeni u strukturi `Fruit`. Voće ima svoje ime te svoju boju (jedan od vrijednosti enumeracije `FruitColor`).

```
class Fruit {  
  
    private final String name;  
    private final FruitColor color;  
  
    public Fruit(String name, FruitColor color) {  
        this.name = name;  
        this.color = color;  
    }  
  
    public String getName() {  
        return name;  
    }  
  
    public FruitColor getColor() {  
        return color;  
    }  
  
    @Override  
    public boolean equals(Object o) {  
        if (this == o)  
            return true;  
        if (o == null || getClass() != o.getClass())  
            return false;  
        Fruit fruit = (Fruit) o;  
        return Objects.equals(name, fruit.name) &&  
               color == fruit.color;  
    }  
  
    @Override  
    public int hashCode() {  
        return Objects.hash(name, color);  
    }  
  
    @Override  
    public String toString() {  
        final StringBuffer sb = new StringBuffer("Fruit{");  
        sb.append("name='").append(name).append('');  
        sb.append('}');  
        return sb.toString();  
    }  
}  
  
enum FruitColor {  
    ORANGE, RED, YELLOW  
}
```

Zadatak je implementirati sljedeće metode u klasi `FruitFinder`, definirane prototipom:

```
/**  
 * Returns a {@link Predicate<Fruit>} instance that will match a fruit by the provided color and  
 * check a fruit name contains the given pattern, at any point in the name, case sensitive.  
 *  
 * @param color the color to be matched  
 * @param pattern the pattern that is to be contained within the name  
 * @return the {@link Predicate} instance  
 */  
public static Predicate<Fruit> allFruitThatMatchColorAndPattern(  
    FruitColor color, String pattern  
)  
  
/**  
 * Returns a {@link Predicate<Fruit>} instance that will match a fruit if it is not the same  
 * color as provided and if the fruits name does not contain the given pattern, at any point in  
 * the name, case sensitive.  
 *  
 * @param color the color to be excluded  
 * @param pattern the pattern that can not be contained within the name  
 * @return the {@link Predicate} instance  
 */  
public static Predicate<Fruit> allFruitThatDoesNotMatchColorAndDoesNotContainPattern(  
    FruitColor color, String pattern  
)
```

NAPOMENA: klasu `FruitFinder` napisati bez modifikatora vidljivosti. Klasa `Fruit` i enumeracija `FruitColor` nalazi se u sustavu i ne trebate je predavati, nego samo koristiti.

Primjer korištenja vidljiv je u sljedećem isječku koda:

```
Fruit apple = new Fruit("Apple", FruitColor.RED);  
Fruit tangerine = new Fruit("Tangerine", FruitColor.ORANGE);  
  
List<Fruit> fruits = new ArrayList<>(Arrays.asList(apple, tangerine));  
List<Fruit> actual1 = fruits.stream()  
    .filter(FruitFinder.allFruitThatMatchColorAndPattern(FruitColor.RED, "pp"))  
    .collect(  
        Collectors.toList());  
  
System.out.println(actual1); // [Fruit{name='Apple'}]  
  
List<Fruit> actual2 = fruits.stream()  
    .filter(  
        FruitFinder  
            .allFruitThatDoesNotMatchColorAndDoesNotContainPattern(FruitColor.RED, "pp")  
    )  
    .collect(  
        Collectors.toList());  
  
System.out.println(actual2); // [Fruit{name='Tangerine'}]
```

1
2
3
4
5
6
7
8
9
10
11
12
13

Run

Save

U sljedećem programskom odsječku, ulazna lista redom predstavlja znamenke nekog decimalnog broja oblika A.BCDE... (npr. 3.1425976).

Nadopunite odsječak koda

```
class Solution{

    public static Integer getPiSequenceBreakpoint(List<Integer> input){
        return IntStream.range(0, input.size()) /*finish with your code*/

    }

}
```

prema sljedećim zahtjevima:

- metoda za ulaznu listu brojeva vraća redni broj pozicije (počevši od 0) prve znamenke koja ne odgovara znamencu u broju Pi.
- U slučaju znamenka razlike ne postoji baca se iznimka `NoSuchElementException`
- pretpostavite da su dostupni sljedeći importi: `import java.util.*; import java.util.stream.Stream; import java.util.stream.IntStream; import java.util.stream.Collectors; import java.lang.Math.*;`
- uputa za predaju: u Edgar zalijepiti samo dio koji nedostaje u odsječku (tj. dio koji mora doći umjesto `/* finish with your code */`).

Primjer

Za listu s brojevima `3, 1, 4, 1, 5, 9, 2, 5, 5, 5` vratit će se `7`.

NAPOMENA:

- početni dio koda `IntStream.range(0, input.length)` generira tok prirodnih brojeva koji odgovaraju pozicijama u ulaznoj listi. U ostatku izraza ih možete koristiti kao indeksere.

The screenshot shows a dark-themed code editor with line numbers from 1 to 13 on the left. The main area contains the Java code provided above. To the right of the editor are two buttons: a green "Run" button and a white "Save" button.

```
1
2
3
4
5
6
7
8
9
10
11
12
13
```

1. Napišite klasu `Account` za baratanje s bankovnim računima. Za svaki račun pohranjuju se informacije o trenutnom stanju računa (atribut `balance`) i trenutnom ukupnom broju provedenih transakcija (atribut `transactions`). Klasa ima sljedeće metode:
 - konstruktor s jednim argumentom koji služi za postavljanje početnog stanja računa, a broj transakcija se prilikom kreiranja računa postavlja na 0.
 - getterske metode za oba atributa.
 - `void deposit(double amount)` metoda za uplatu novca i `void withdraw(double amount)` metoda za podizanje novca. Broj transakcija se uvećava pri svakoj uplati i podizanju novca.
 - `void onEndOfMonth()` metoda se poziva nad objektom jednom na kraju svakog mjeseca. U metodi se najprije poziva metoda `endOfMonthCharge` te se broj transakcija postavlja na 0.
 - `void endOfMonthCharge()` je apstraktna metoda. Služi banci za računanje i naplatu troškove usluge vođenja računa. Metoda "skida" (`withdraw`) novce s računa. Postoje 3 vrste računa. Za svaku vrstu postoji drugačija politika naplate troškova.
2. Napišite klasu `FixedFeeAccount` koja je `Account`. Klasa ima konstruktor s jednim argumentom preko kojeg se postavlja početno stanje računa. Za ovu klasu vrijedi da se troškovi usluge naplaćuju tj. skidaju s računa u fiksnom iznosu od 10 Kn mjesечно.
3. Napišite klasu `WithdrawAccount` koja je `Account`. Klasa ima atribut `withdrawCounter` i konstruktor s jednim argumentom preko kojeg se postavlja početno stanje računa, a u konstruktoru se postavlja i vrijednost atributa `withdrawCounter` na 0. Klasa mora nadjačati metodu `withdraw` u kojoj se inkrementira vrijednost brojača `withdrawCounter` (broj isplata). Za ovu klasu vrijedi da se troškovi usluge naplaćuju prema broju isplata u mjesecu i to 0.10 Kn po isplati. Ako je `withdrawCounter > 0` s računa se isplaćuje iznos od `(withdrawCounter * 0.1)`. Ova zadnja isplata u korist banke se ne naplaćuje. Na kraju se `withdrawCounter` resetira na 0.
4. Napišite klasu `VipAccount` koja je `Account`. Klasa ima konstruktor s jednim argumentom preko kojeg se postavlja početno stanje računa. Klasa mora nadjačati metodu `deposit` jer se kod ovog računa potiču uplate na način da ukoliko je iznos uplate veći od 10000 Kn tada se uplata uvećava za 10 Kn. Kod ovog računa nema naplate troškova usluge na kraju mjeseca.

Napomena: sve klase napisati bez modifikatora vidljivosti i bez navođenja paketa kojem pripadaju te ih sve zajedno kopirati u prostor za rješenje. Sve metode svih klasa imaju public vidljivost. Na vrhu možete dodavati proizvoljne importe.

Napisati apstraktnu klasu `Reference` koja predstavlja referencu u znanstvenoj literaturi i ima sljedeće privatne atribute: `authors` tipa `Author[]`, `title` tipa `String`, `year` tipa `int` i `label` tipa `String`. Ova klasa je opremljena konstruktorom i `getterima` za inicijalizaciju i pristup vrijednosti navedenih atributa. Osim toga, ova klasa implementira sučelje `Citable` koje ima jednu jedinu metodu `getCitation` koja vraća citat reference koji se koristi pri navođenju reference u znanstvenom članku. Programski kod klase `Author` i sučelja `Citable` (koje **ne treba implementirati**) je prikazan u nastavku:

```
class Author {
    String name;
    String surname;

    public Author(String name, String surname) {
        this.name = name;
        this.surname = surname;
    }

    public String getName() {
        return name;
    }

    public String getSurname() {
        return surname;
    }
}

interface Citable {
    String getCitation();
}
```

Nakon toga je potrebno napisati klasu `Book` koja nasljeđuje klasu `Reference`, a ima dodatne privatne atribute `publisher` i `address` tipa `String`. Ova klasa je opremljena konstruktorom i `getterima` za inicijalizaciju i pristup vrijednosti navedenih atributa. Pozivom metode `getCitation` nad ovom klasom se treba vratiti jedna linija teksta koja je formirana od sljedećih dijelova bez razmaka:

- ime klase
- zadnje dvije znamenke godine izdavanja
- prezimena svih autora bez razmaka
- znak ''
- atribut `label` iz nadklase `Reference`

Zatim je potrebno napisati klasu `InProceedings` koja nasljeđuje klasu `Book`, a ima dodatne privatne atribute `booktitle` tipa `String` te `startPage` i `endPage` tipa `int`. Ova klasa je također opremljena konstruktorom i `getterima` za inicijalizaciju i pristup vrijednosti navedenih atributa. Pozivom metode `getCitation` nad ovom klasom se treba vratiti jedne linija teksta koja je formirana od sljedećih dijelova bez razmaka:

- ime klase
- prvo slovo prezimena svih autora bez razmaka
- godina izdavanja
- znak ''
- atribut `label` iz nadklase `Reference`

Primjer isječka koda metode `main` koja demonstrira upotrebu ovih klasa je sljedeći:

```
Author ws = new Author("Walter", "Savitch");
Author km = new Author("Kenrick", "Mock");

Book book = new Book(new Author[]{ws, km}, "Absolute Java", 2015,
                    "0134041674", "Pearson", "PEL, Edinburgh Gate, Harlow, Essex CM20 2JE, England");

System.out.println(book.getCitation());//Book15SavitchMock-0134041674

Author gb = new Author("Gilad", "Bracha");
Author mo = new Author("Martin", "Odersky");
Author ds = new Author("David", "Stoutamire");
Author pw = new Author("Philip", "Wadler");

InProceedings inProceedings = new InProceedings(new Author[]{gb, mo, ds, pw},
                                                "Making the future safe for the past: Adding genericity to the Java programming language", 1998,
                                                "286957", "ACM",
                                                "ACM, 2 Penn Plaza, Suite 701, New York, NY 10121-0701",
                                                "Proceedings of the 13th ACM SIGPLAN conference on Object-oriented programming, systems, languages and
                                                applications", 183, 200);

System.out.println(inProceedings.getCitation());//InProceedingsBOSW1998-286957
```

Napomena: Klase `Reference`, `Book` i `InProceedings` napisati bez modifikatora vidljivosti i kopirati u prostor za rješenje bez navođenja paketa kojem pripadaju. Za klase nije potrebno provjeravati jesu li predane ispravne vrijednosti argumenata.

```
1
2
3
4
5
6
7
8
9
10
11
12
13
```

- Napišite klasu `Account` za baratanje s bankovnim računima. Za svaki račun pohranjuju se informacije o trenutnom stanju računa (atribut `balance`) i trenutnom ukupnom broju provedenih transakcija (atribut `transactions`). Klasa ima sljedeće metode:
 - konstruktor s jednim argumentom koji služi za postavljanje početnog stanja računa, a broj transakcija se prilikom kreiranja računa postavlja na 0.
 - getterske metode za oba atributa.
 - `void deposit(double amount)` metoda za uplatu novca i `void withdraw(double amount)` metoda za podizanje novca. Broj transakcija se uvećava pri svakoj uplati i podizanju novca.
 - `void onEndOfMonth()` metoda se poziva nad objektom jednom na kraju svakog mjeseca. U metodi se najprije poziva metoda `endOfMonthCharge` te se broj transakcija postavlja na 0.
 - `void endOfMonthCharge()` je apstraktna metoda. Služi banci za računanje i naplatu troškove usluge vođenja računa. Metoda "skida" (`withdraw`) novce s računa. Postoje 3 vrste računa. Za svaku vrstu postoji drugačija politika naplate troškova.
- Napišite klasu `FixedFeeAccount` koja je `Account`. Klasa ima konstruktor s jednim argumentom preko kojeg se postavlja početno stanje računa. Za ovu klasu vrijedi da se troškovi usluge naplaćuju tj. skidaju s računa u fiksnom iznosu od 10 Kn mjesечно.
- Napišite klasu `WithdrawAccount` koja je `Account`. Klasa ima atribut `withdrawCounter` i konstruktor s jednim argumentom preko kojeg se postavlja početno stanje računa, a u konstruktoru se postavlja i vrijednost atributa `withdrawCounter` na 0. Klasa mora nadjačati metodu `withdraw` u kojoj se inkrementira vrijednost brojača `withdrawCounter` (broj isplate). Za ovu klasu vrijedi da se troškovi usluge naplaćuju prema broju isplata u mjesecu i to 0.10 Kn po isplati. Ako je `withdrawCounter > 0` s računa se isplaćuje iznos od `(withdrawCounter * 0.1)`. Ova zadnja isplata u korist banke se ne naplaćuje. Na kraju se `withdrawCounter` resetira na 0.
- Napišite klasu `VipAccount` koja je `Account`. Klasa ima konstruktor s jednim argumentom preko kojeg se postavlja početno stanje računa. Klasa mora nadjačati metodu `deposit` jer se kod ovog računa potiču uplate na način da ukoliko je iznos uplate veći od 10000 Kn tada se uplata uvećava za 10 Kn. Kod ovog računa nema naplate troškova usluge na kraju mjeseca.

Napomena: sve klase napisati bez modifikatora vidljivosti i bez navođenja paketa kojem pripadaju te ih sve zajedno kopirati u prostor za rješenje. Sve metode svih klasa imaju public vidljivost. Na vrhu možete dodavati proizvoljne importe.

```

1  package com.sage;
2
3  public abstract class Account {
4
5     protected double balance;
6
7     protected int transactions;
8
9     public Account(double balance) {
10        this.balance = balance;
11        this.transactions = 0;
12    }
13
14    public double getBalance() {
15        return balance;
16    }
17
18    public int getTransactions() {
19        return transactions;
20    }
21
22    public void deposit(double amount) {
23        balance += amount;
24        transactions++;
25    }
26
27    public void withdraw(double amount) {
28        if (amount < 0) {
29            throw new IllegalArgumentException("Amount must be positive");
30        }
31
32        if (balance - amount < 0) {
33            throw new IllegalStateException("Insufficient funds");
34        }
35
36        balance -= amount;
37        transactions++;
38    }
39
40    public abstract void onEndOfMonth();
41
42    public void endOfMonthCharge() {
43        onEndOfMonth();
44    }
45
46    protected void resetWithdrawCounter() {
47        withdrawCounter = 0;
48    }
49
50    protected int withdrawCounter;
51
52    protected void incrementWithdrawCounter() {
53        withdrawCounter++;
54    }
55
56    protected void withdraw(double amount) {
57        if (amount < 0) {
58            throw new IllegalArgumentException("Amount must be positive");
59        }
60
61        if (withdrawCounter > 0) {
62            balance -= amount;
63        }
64    }
65
66    protected void withdraw(double amount, double fee) {
67        if (amount < 0) {
68            throw new IllegalArgumentException("Amount must be positive");
69        }
70
71        if (fee < 0) {
72            throw new IllegalArgumentException("Fee must be positive");
73        }
74
75        if (balance - amount - fee < 0) {
76            throw new IllegalStateException("Insufficient funds");
77        }
78
79        balance -= amount + fee;
80        withdrawCounter++;
81    }
82
83    protected void withdraw(double amount, double fee, double minFee) {
84        if (amount < 0) {
85            throw new IllegalArgumentException("Amount must be positive");
86        }
87
88        if (fee < 0) {
89            throw new IllegalArgumentException("Fee must be positive");
90        }
91
92        if (minFee < 0) {
93            throw new IllegalArgumentException("MinFee must be positive");
94        }
95
96        if (balance - amount - fee - minFee < 0) {
97            throw new IllegalStateException("Insufficient funds");
98        }
99
100       balance -= amount + fee + minFee;
101       withdrawCounter++;
102   }
103
104   protected void withdraw(double amount, double fee, double minFee, double maxFee) {
105       if (amount < 0) {
106           throw new IllegalArgumentException("Amount must be positive");
107       }
108
109       if (fee < 0) {
110           throw new IllegalArgumentException("Fee must be positive");
111       }
112
113       if (minFee < 0) {
114           throw new IllegalArgumentException("MinFee must be positive");
115       }
116
117       if (maxFee < 0) {
118           throw new IllegalArgumentException("MaxFee must be positive");
119       }
120
121       if (balance - amount - fee - minFee - maxFee < 0) {
122           throw new IllegalStateException("Insufficient funds");
123       }
124
125       balance -= amount + fee + minFee + maxFee;
126       withdrawCounter++;
127   }
128
129   protected void withdraw(double amount, double fee, double minFee, double maxFee, double maxFeePercentage) {
130       if (amount < 0) {
131           throw new IllegalArgumentException("Amount must be positive");
132       }
133
134       if (fee < 0) {
135           throw new IllegalArgumentException("Fee must be positive");
136       }
137
138       if (minFee < 0) {
139           throw new IllegalArgumentException("MinFee must be positive");
140       }
141
142       if (maxFee < 0) {
143           throw new IllegalArgumentException("MaxFee must be positive");
144       }
145
146       if (maxFeePercentage < 0) {
147           throw new IllegalArgumentException("MaxFeePercentage must be positive");
148       }
149
150       if (balance - amount - fee - minFee - maxFee - maxFeePercentage * amount < 0) {
151           throw new IllegalStateException("Insufficient funds");
152       }
153
154       balance -= amount + fee + minFee + maxFee + maxFeePercentage * amount;
155       withdrawCounter++;
156   }
157
158   protected void withdraw(double amount, double fee, double minFee, double maxFee, double maxFeePercentage, double maxFeePercentagePercentage) {
159       if (amount < 0) {
160           throw new IllegalArgumentException("Amount must be positive");
161       }
162
163       if (fee < 0) {
164           throw new IllegalArgumentException("Fee must be positive");
165       }
166
167       if (minFee < 0) {
168           throw new IllegalArgumentException("MinFee must be positive");
169       }
170
171       if (maxFee < 0) {
172           throw new IllegalArgumentException("MaxFee must be positive");
173       }
174
175       if (maxFeePercentage < 0) {
176           throw new IllegalArgumentException("MaxFeePercentage must be positive");
177       }
178
179       if (maxFeePercentagePercentage < 0) {
180           throw new IllegalArgumentException("MaxFeePercentagePercentage must be positive");
181       }
182
183       if (balance - amount - fee - minFee - maxFee - maxFeePercentage * amount - maxFeePercentagePercentage * amount < 0) {
184           throw new IllegalStateException("Insufficient funds");
185       }
186
187       balance -= amount + fee + minFee + maxFee + maxFeePercentage * amount + maxFeePercentagePercentage * amount;
188       withdrawCounter++;
189   }
190
191   protected void withdraw(double amount, double fee, double minFee, double maxFee, double maxFeePercentage, double maxFeePercentagePercentage, double maxFeePercentagePercentagePercentage) {
192       if (amount < 0) {
193           throw new IllegalArgumentException("Amount must be positive");
194       }
195
196       if (fee < 0) {
197           throw new IllegalArgumentException("Fee must be positive");
198       }
199
200       if (minFee < 0) {
201           throw new IllegalArgumentException("MinFee must be positive");
202       }
203
204       if (maxFee < 0) {
205           throw new IllegalArgumentException("MaxFee must be positive");
206       }
207
208       if (maxFeePercentage < 0) {
209           throw new IllegalArgumentException("MaxFeePercentage must be positive");
210       }
211
212       if (maxFeePercentagePercentage < 0) {
213           throw new IllegalArgumentException("MaxFeePercentagePercentage must be positive");
214       }
215
216       if (maxFeePercentagePercentagePercentage < 0) {
217           throw new IllegalArgumentException("MaxFeePercentagePercentagePercentage must be positive");
218       }
219
220       if (balance - amount - fee - minFee - maxFee - maxFeePercentage * amount - maxFeePercentagePercentage * amount - maxFeePercentagePercentagePercentage * amount < 0) {
221           throw new IllegalStateException("Insufficient funds");
222       }
223
224       balance -= amount + fee + minFee + maxFee + maxFeePercentage * amount + maxFeePercentagePercentage * amount + maxFeePercentagePercentagePercentage * amount;
225       withdrawCounter++;
226   }
227
228   protected void withdraw(double amount, double fee, double minFee, double maxFee, double maxFeePercentage, double maxFeePercentagePercentage, double maxFeePercentagePercentagePercentage, double maxFeePercentagePercentagePercentagePercentage) {
229       if (amount < 0) {
230           throw new IllegalArgumentException("Amount must be positive");
231       }
232
233       if (fee < 0) {
234           throw new IllegalArgumentException("Fee must be positive");
235       }
236
237       if (minFee < 0) {
238           throw new IllegalArgumentException("MinFee must be positive");
239       }
240
241       if (maxFee < 0) {
242           throw new IllegalArgumentException("MaxFee must be positive");
243       }
244
245       if (maxFeePercentage < 0) {
246           throw new IllegalArgumentException("MaxFeePercentage must be positive");
247       }
248
249       if (maxFeePercentagePercentage < 0) {
250           throw new IllegalArgumentException("MaxFeePercentagePercentage must be positive");
251       }
252
253       if (maxFeePercentagePercentagePercentage < 0) {
254           throw new IllegalArgumentException("MaxFeePercentagePercentagePercentage must be positive");
255       }
256
257       if (maxFeePercentagePercentagePercentagePercentage < 0) {
258           throw new IllegalArgumentException("MaxFeePercentagePercentagePercentagePercentage must be positive");
259       }
260
261       if (balance - amount - fee - minFee - maxFee - maxFeePercentage * amount - maxFeePercentagePercentage * amount - maxFeePercentagePercentagePercentage * amount - maxFeePercentagePercentagePercentagePercentage * amount < 0) {
262           throw new IllegalStateException("Insufficient funds");
263       }
264
265       balance -= amount + fee + minFee + maxFee + maxFeePercentage * amount + maxFeePercentagePercentage * amount + maxFeePercentagePercentagePercentage * amount + maxFeePercentagePercentagePercentagePercentage * amount;
266       withdrawCounter++;
267   }
268
269   protected void withdraw(double amount, double fee, double minFee, double maxFee, double maxFeePercentage, double maxFeePercentagePercentage, double maxFeePercentagePercentagePercentage, double maxFeePercentagePercentagePercentagePercentage, double maxFeePercentagePercentagePercentagePercentagePercentage) {
270       if (amount < 0) {
271           throw new IllegalArgumentException("Amount must be positive");
272       }
273
274       if (fee < 0) {
275           throw new IllegalArgumentException("Fee must be positive");
276       }
277
278       if (minFee < 0) {
279           throw new IllegalArgumentException("MinFee must be positive");
280       }
281
282       if (maxFee < 0) {
283           throw new IllegalArgumentException("MaxFee must be positive");
284       }
285
286       if (maxFeePercentage < 0) {
287           throw new IllegalArgumentException("MaxFeePercentage must be positive");
288       }
289
290       if (maxFeePercentagePercentage < 0) {
291           throw new IllegalArgumentException("MaxFeePercentagePercentage must be positive");
292       }
293
294       if (maxFeePercentagePercentagePercentage < 0) {
295           throw new IllegalArgumentException("MaxFeePercentagePercentagePercentage must be positive");
296       }
297
298       if (maxFeePercentagePercentagePercentagePercentage < 0) {
299           throw new IllegalArgumentException("MaxFeePercentagePercentagePercentagePercentage must be positive");
300       }
301
302       if (maxFeePercentagePercentagePercentagePercentagePercentage < 0) {
303           throw new IllegalArgumentException("MaxFeePercentagePercentagePercentagePercentagePercentage must be positive");
304       }
305
306       if (balance - amount - fee - minFee - maxFee - maxFeePercentage * amount - maxFeePercentagePercentage * amount - maxFeePercentagePercentagePercentage * amount - maxFeePercentagePercentagePercentagePercentage * amount - maxFeePercentagePercentagePercentagePercentagePercentage * amount < 0) {
307           throw new IllegalStateException("Insufficient funds");
308       }
309
310       balance -= amount + fee + minFee + maxFee + maxFeePercentage * amount + maxFeePercentagePercentage * amount + maxFeePercentagePercentagePercentage * amount + maxFeePercentagePercentagePercentagePercentage * amount + maxFeePercentagePercentagePercentagePercentagePercentage * amount;
311       withdrawCounter++;
312   }
313
314   protected void withdraw(double amount, double fee, double minFee, double maxFee, double maxFeePercentage, double maxFeePercentagePercentage, double maxFeePercentagePercentagePercentage, double maxFeePercentagePercentagePercentagePercentage, double maxFeePercentagePercentagePercentagePercentagePercentage, double maxFeePercentagePercentagePercentagePercentagePercentagePercentage) {
315       if (amount < 0) {
316           throw new IllegalArgumentException("Amount must be positive");
317       }
318
319       if (fee < 0) {
320           throw new IllegalArgumentException("Fee must be positive");
321       }
322
323       if (minFee < 0) {
324           throw new IllegalArgumentException("MinFee must be positive");
325       }
326
327       if (maxFee < 0) {
328           throw new IllegalArgumentException("MaxFee must be positive");
329       }
330
331       if (maxFeePercentage < 0) {
332           throw new IllegalArgumentException("MaxFeePercentage must be positive");
333       }
334
335       if (maxFeePercentagePercentage < 0) {
336           throw new IllegalArgumentException("MaxFeePercentagePercentage must be positive");
337       }
338
339       if (maxFeePercentagePercentagePercentage < 0) {
339           throw new IllegalArgumentException("MaxFeePercentagePercentagePercentage must be positive");
340       }
341
342       if (maxFeePercentagePercentagePercentagePercentage < 0) {
343           throw new IllegalArgumentException("MaxFeePercentagePercentagePercentagePercentage must be positive");
344       }
345
346       if (maxFeePercentagePercentagePercentagePercentagePercentage < 0) {
347           throw new IllegalArgumentException("MaxFeePercentagePercentagePercentagePercentagePercentage must be positive");
348       }
349
350       if (balance - amount - fee - minFee - maxFee - maxFeePercentage * amount - maxFeePercentagePercentage * amount - maxFeePercentagePercentagePercentage * amount - maxFeePercentagePercentagePercentagePercentage * amount - maxFeePercentagePercentagePercentagePercentagePercentage * amount < 0) {
351           throw new IllegalStateException("Insufficient funds");
352       }
353
354       balance -= amount + fee + minFee + maxFee + maxFeePercentage * amount + maxFeePercentagePercentage * amount + maxFeePercentagePercentagePercentage * amount + maxFeePercentagePercentagePercentagePercentage * amount + maxFeePercentagePercentagePercentagePercentagePercentage * amount;
355       withdrawCounter++;
356   }
357
358   protected void withdraw(double amount, double fee, double minFee, double maxFee, double maxFeePercentage, double maxFeePercentagePercentage, double maxFeePercentagePercentagePercentage, double maxFeePercentagePercentagePercentagePercentage, double maxFeePercentagePercentagePercentagePercentagePercentage, double maxFeePercentagePercentagePercentagePercentagePercentagePercentage) {
359       if (amount < 0) {
360           throw new IllegalArgumentException("Amount must be positive");
361       }
362
363       if (fee < 0) {
364           throw new IllegalArgumentException("Fee must be positive");
365       }
366
367       if (minFee < 0) {
368           throw new IllegalArgumentException("MinFee must be positive");
369       }
370
371       if (maxFee < 0) {
372           throw new IllegalArgumentException("MaxFee must be positive");
373       }
374
375       if (maxFeePercentage < 0) {
376           throw new IllegalArgumentException("MaxFeePercentage must be positive");
377       }
378
379       if (maxFeePercentagePercentage < 0) {
379           throw new IllegalArgumentException("MaxFeePercentagePercentage must be positive");
380       }
381
382       if (maxFeePercentagePercentagePercentage < 0) {
383           throw new IllegalArgumentException("MaxFeePercentagePercentagePercentage must be positive");
384       }
385
386       if (maxFeePercentagePercentagePercentagePercentage < 0) {
387           throw new IllegalArgumentException("MaxFeePercentagePercentagePercentagePercentage must be positive");
388       }
389
390       if (maxFeePercentagePercentagePercentagePercentagePercentage < 0) {
391           throw new IllegalArgumentException("MaxFeePercentagePercentagePercentagePercentagePercentage must be positive");
392       }
393
394       if (balance - amount - fee - minFee - maxFee - maxFeePercentage * amount - maxFeePercentagePercentage * amount - maxFeePercentagePercentagePercentage * amount - maxFeePercentagePercentagePercentagePercentage * amount - maxFeePercentagePercentagePercentagePercentagePercentage * amount < 0) {
395           throw new IllegalStateException("Insufficient funds");
396       }
397
398       balance -= amount + fee + minFee + maxFee + maxFeePercentage * amount + maxFeePercentagePercentage * amount + maxFeePercentagePercentagePercentage * amount + maxFeePercentagePercentagePercentagePercentage * amount + maxFeePercentagePercentagePercentagePercentagePercentage * amount;
399       withdrawCounter++;
400   }
401
402   protected void withdraw(double amount, double fee, double minFee, double maxFee, double maxFeePercentage, double maxFeePercentagePercentage, double maxFeePercentagePercentagePercentage, double maxFeePercentagePercentagePercentagePercentage, double maxFeePercentagePercentagePercentagePercentagePercentage, double maxFeePercentagePercentagePercentagePercentagePercentagePercentage) {
403       if (amount < 0) {
404           throw new IllegalArgumentException("Amount must be positive");
405       }
406
407       if (fee < 0) {
408           throw new IllegalArgumentException("Fee must be positive");
409       }
410
411       if (minFee < 0) {
412           throw new IllegalArgumentException("MinFee must be positive");
413       }
414
415       if (maxFee < 0) {
416           throw new IllegalArgumentException("MaxFee must be positive");
417       }
418
419       if (maxFeePercentage < 0) {
419           throw new IllegalArgumentException("MaxFeePercentage must be positive");
420       }
421
422       if (maxFeePercentagePercentage < 0) {
422           throw new IllegalArgumentException("MaxFeePercentagePercentage must be positive");
423       }
424
425       if (maxFeePercentagePercentagePercentage < 0) {
425           throw new IllegalArgumentException("MaxFeePercentagePercentagePercentage must be positive");
426       }
427
428       if (maxFeePercentagePercentagePercentagePercentage < 0) {
428           throw new IllegalArgumentException("MaxFeePercentagePercentagePercentagePercentage must be positive");
429       }
429
430       if (maxFeePercentagePercentagePercentagePercentagePercentage < 0) {
430           throw new IllegalArgumentException("MaxFeePercentagePercentagePercentagePercentagePercentage must be positive");
431       }
431
432       if (balance - amount - fee - minFee - maxFee - maxFeePercentage * amount - maxFeePercentagePercentage * amount - maxFeePercentagePercentagePercentage * amount - maxFeePercentagePercentagePercentagePercentage * amount - maxFeePercentagePercentagePercentagePercentagePercentage * amount < 0) {
433           throw new IllegalStateException("Insufficient funds");
434       }
435
436       balance -= amount + fee + minFee + maxFee + maxFeePercentage * amount + maxFeePercentagePercentage * amount + maxFeePercentagePercentagePercentage * amount + maxFeePercentagePercentagePercentagePercentage * amount + maxFeePercentagePercentagePercentagePercentagePercentage * amount;
437       withdrawCounter++;
438   }
439
440   protected void withdraw(double amount, double fee, double minFee, double maxFee, double maxFeePercentage, double maxFeePercentagePercentage, double maxFeePercentagePercentagePercentage, double maxFeePercentagePercentagePercentagePercentage, double maxFeePercentagePercentagePercentagePercentagePercentage, double maxFeePercentagePercentagePercentagePercentagePercentagePercentage) {
441       if (amount < 0) {
442           throw new IllegalArgumentException("Amount must be positive");
443       }
444
445       if (fee < 0) {
446           throw new IllegalArgumentException("Fee must be positive");
447       }
448
449       if (minFee < 0) {
450           throw new IllegalArgumentException("MinFee must be positive");
451       }
452
453       if (maxFee < 0) {
454           throw new IllegalArgumentException("MaxFee must be positive");
455       }
456
457       if (maxFeePercentage < 0) {
457           throw new IllegalArgumentException("MaxFeePercentage must be positive");
458       }
459
460       if (maxFeePercentagePercentage < 0) {
460           throw new IllegalArgumentException("MaxFeePercentagePercentage must be positive");
461       }
462
463       if (maxFeePercentagePercentagePercentage < 0) {
463           throw new IllegalArgumentException("MaxFeePercentagePercentagePercentage must be positive");
464       }
465
466       if (maxFeePercentagePercentagePercentagePercentage < 0) {
466           throw new IllegalArgumentException("MaxFeePercentagePercentagePercentagePercentage must be positive");
467       }
467
468       if (maxFeePercentagePercentagePercentagePercentagePercentage < 0) {
468           throw new IllegalArgumentException("MaxFeePercentagePercentagePercentagePercentagePercentage must be positive");
469       }
469
470       if (balance - amount - fee - minFee - maxFee - maxFeePercentage * amount - maxFeePercentagePercentage * amount - maxFeePercentagePercentagePercentage * amount - maxFeePercentagePercentagePercentagePercentage * amount - maxFeePercentagePercentagePercentagePercentagePercentage * amount < 0) {
471           throw new IllegalStateException("Insufficient funds");
472       }
473
474       balance -= amount + fee + minFee + maxFee + maxFeePercentage * amount + maxFeePercentagePercentage * amount + maxFeePercentagePercentagePercentage * amount + maxFeePercentagePercentagePercentagePercentage * amount + maxFeePercentagePercentagePercentagePercentagePercentage * amount;
475       withdrawCounter++;
476   }
477
478   protected void withdraw(double amount, double fee, double minFee, double maxFee, double maxFeePercentage, double maxFeePercentagePercentage, double maxFeePercentagePercentagePercentage, double maxFeePercentagePercentagePercentagePercentage, double maxFeePercentagePercentagePercentagePercentagePercentage, double maxFeePercentagePercentagePercentagePercentagePercentagePercentage) {
479       if (amount < 0) {
480           throw new IllegalArgumentException("Amount must be positive");
481       }
482
483       if (fee < 0) {
484           throw new IllegalArgumentException("Fee must be positive");
485       }
486
487       if (minFee < 0) {
488           throw new IllegalArgumentException("MinFee must be positive");
489       }
490
491       if (maxFee < 0) {
492           throw new IllegalArgumentException("MaxFee must be positive");
493       }
494
495       if (maxFeePercentage < 0) {
495           throw new IllegalArgumentException("MaxFeePercentage must be positive");
496       }
497
498       if (maxFeePercentagePercentage < 0) {
498           throw new IllegalArgumentException("MaxFeePercentagePercentage must be positive");
499       }
499
500       if (maxFeePercentagePercentagePercentage < 0) {
500           throw new IllegalArgumentException("MaxFeePercentagePercentagePercentage must be positive");
501       }
501
502       if (maxFeePercentagePercentagePercentagePercentage < 0) {
502           throw new IllegalArgumentException("MaxFeePercentagePercentagePercentagePercentage must be positive");
503       }
503
504       if (maxFeePercentagePercentagePercentagePercentagePercentage < 0) {
504           throw new IllegalArgumentException("MaxFeePercentagePercentagePercentagePercentagePercentage must be positive");
505       }
505
506       if (balance - amount - fee - minFee - maxFee - maxFeePercentage * amount - maxFeePercentagePercentage * amount - maxFeePercentagePercentagePercentage * amount - maxFeePercentagePercentagePercentagePercentage * amount - maxFeePercentagePercentagePercentagePercentagePercentage * amount < 0) {
507           throw new IllegalStateException("Insufficient funds");
508       }
509
510       balance -= amount + fee + minFee + maxFee + maxFeePercentage * amount + maxFeePercentagePercentage * amount + maxFeePercentagePercentagePercentage * amount + maxFeePercentagePercentagePercentagePercentage * amount + maxFeePercentagePercentagePercentagePercentagePercentage * amount;
511       withdrawCounter++;
512   }
513
514   protected void withdraw(double amount, double fee, double minFee, double maxFee, double maxFeePercentage, double maxFeePercentagePercentage, double maxFeePercentagePercentagePercentage, double maxFeePercentagePercentagePercentagePercentage, double maxFeePercentagePercentagePercentagePercentagePercentage, double maxFeePercentagePercentagePercentagePercentagePercentagePercentage) {
515       if (amount < 0) {
516           throw new IllegalArgumentException("Amount must be positive");
517       }
518
519       if (fee < 0) {
520           throw new IllegalArgumentException("Fee must be positive");
521       }
522
523       if (minFee < 0) {
524           throw new IllegalArgumentException("MinFee must be positive");
525       }
526
527       if (maxFee < 0) {
528           throw new IllegalArgumentException("MaxFee must be positive");
529       }
530
531       if (maxFeePercentage < 0) {
531           throw new IllegalArgumentException("MaxFeePercentage must be positive");
532       }
533
534       if (maxFeePercentagePercentage < 0) {
534           throw new IllegalArgumentException("MaxFeePercentagePercentage must be positive");
535       }
535
536       if (maxFeePercentagePercentagePercentage < 0) {
536           throw new IllegalArgumentException("MaxFeePercentagePercentagePercentage must be positive");
537       }
537
538       if (maxFeePercentagePercentagePercentagePercentage < 0) {
538           throw new IllegalArgumentException("MaxFeePercentagePercentagePercentagePercentage must be positive");
539       }
539
540       if (maxFeePercentagePercentagePercentagePercentagePercentage < 0) {
540           throw new IllegalArgumentException("MaxFeePercentagePercentagePercentagePercentagePercentage must be positive");
541       }
541
542       if (balance - amount - fee - minFee - maxFee - maxFeePercentage * amount - maxFeePercentagePercentage * amount - maxFeePercentagePercentagePercentage * amount - maxFeePercentagePercentagePercentagePercentage * amount - maxFeePercentagePercentagePercentagePercentagePercentage *
```

Napisati klasu *KeyValue* parametriziranu po 2 parametra koja predstavlja parove ključ vrijednost. Za klasu *KeyValue* definirati *gettere* i *settere* za ključ i vrijednost te konstruktor koji prima inicijalne vrijednosti ključa i vrijednosti. Npr. sljedećim retkom

```
KeyValue<String, Integer> kv = new KeyValue<>("pero", 5);
```

stvorit će se novi objekt u kojem će ključ biti **pero**, a za vrijednost **5**.

Nakon toga iz klase *KeyValue* izvesti klasu *SingleTypeKeyValue* koja predstavlja parametrizirani par ključ-vrijednost u kojem su ključ i vrijednost istog tipa. Klasa *SingleTypeKeyValue* ne definirane nikakve dodatne metode u odnosu na *KeyValue*.

Nakon toga iz klase *SingleTypeKeyValue* izvesti klasu *XY* koja predstavlja par vrijednosti tipa *Double*. Smisao ove klase je da predstavlja vrijednost neke funkcije za neku vrijednost x koja je zapisana kao ključ.

U klasi *XY* napisati statičku metodu `public static boolean isIncreasingFunction(XY[] data)` koja za primljeno polje provjerava jesu li parovi koji predstavljaju parove (x,y) neke funkcije takvi da je funkcija rastuća. Poredak elemenata u polju može biti proizvoljan i **polje se ne smije mijenjati**.

Primjer: Ako je ulazno poje sadržavalo slijedeće vrijednosti za XY **(6, 8), (7, 8), (9.1, 13), (-3, -2), (-1, 4)**, funkcija je rastuća (primjetiti da ne mora biti stroga rastuća), a npr. za poje sadržaje **(6, 8), (7.5, 12), (9, 9), (10, 10), (11, 11)** ukazuje da funkcija nije rastuća.

Ako je u metodu *isIncreasingFunction* poslano prazno polje ili polje sa samo jednim elementom, metoda mora vratiti *true*. Možete prepostaviti da su vrijednost za x ispravno zadane (tj. ne može se dvaput pojaviti ista vrijednost za x).

Napomena: Klase *KeyValue*, *SingleTypeKeyValue* i *XY* napisati bez modifikatora vidljivosti i kopirati u prostor za rješenje bez navođenja paketa kojem pripada. Na vrhu možete dodavati *importe* iz standardnih Javinih biblioteka.

```
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13
```

Run

Save

Napisati program koji će provjeriti osnovnu validnost kreditne kartice. Potrebno je napisati klasu:

```
public static class CreditCard
```

i u njoj metode:

```
public static boolean checkCreditCard(String stringPAN, String cardType)
public static int getChecksum(Long numbPAN)
```

Metoda `checkCreditCard` provjerava PAN (16-znamenkasti broj na kartici) i tip kartice (Stringovi koji mogu biti vrijednosti "Diners", "Visa", "MasterCard" ili "AmEx").

Provjera se obavlja na sljedeći način:

- Ako je duljina broja neispravna, baca se iznimka: `InvalidPANLengthException`
- Ako se u ulaznom stringu pojavio nedozvoljeni znak (nešto što nije znamenka), baca se iznimka `InvalidPANCharacterException`
- Pomoću metode `getChecksum` (koja je objašnjena niže) potrebno je odrediti odgovara li očekivana znamenka unesenoj zadnjoj znamenki. Ako ne, baca se `InvalidPANChecksumException`.
- Ako prva znamenka ne odgovara pripadajućoj kartici, baca se `InvalidCardException`

Za ispravnu karticu, metoda mora vratiti `true`.

Prve znamenke i njihove odgovarajuće kartice su:

- 3 za Diners
- 4 za Visa
- 5 za MasterCard
- 6 za AmEx

Metoda `getChecksum` za ulazni parametar tipa `long` sumira sve znamenke osim zadnje. Zatim vraća zadnju znamenku broja izračunate sume.

NIJE potrebno pisati programski kod za klase:

```
InvalidCardException
InvalidPANLengthException
InvalidPANCharacterException
InvalidPANLengthException
```

A screenshot of a code editor interface. On the left, there is a vertical line number column from 1 to 13. The main area contains the following code:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
```

On the right side of the editor, there are two buttons: a green "Run" button and a white "Save" button.

Napisati klasu *KeyValue* parametriziranu po 2 parametra koja predstavlja parove ključ vrijednost. Za klasu *KeyValue* definirati *gettere* i *settere* za ključ i vrijednost te konstruktor koji prima inicijalne vrijednosti ključa i vrijednosti. Npr. sljedećim retkom

```
KeyValue<String, Integer> kv = new KeyValue<>("pero", 5);
```

stvorit će se novi objekt u kojem će ključ biti `pero`, a za vrijednost `5`.

Nakon toga iz klase *KeyValue* izvesti klasu *SingleTypeKeyValue* koja predstavlja parametrizirani par ključ-vrijednost u kojem su ključ i vrijednost istog tipa. Klasa *SingleTypeKeyValue* ne definirane nikakve dodatne metode u odnosu na *KeyValue*.

Nakon toga iz klase *SingleTypeKeyValue* izvesti klasu *XY* koja predstavlja par vrijednosti tipa *Double*. Smisao ove klase je da predstavlja vrijednost neke funkcije za neku vrijednost x koja je zapisana kao ključ.

U klasi *XY* napisati static metodu `public static boolean isIncreasingFunction(XY[] data)` koja za primljeno polje provjerava jesu li parovi koji predstavljaju parove (*x*, *y*) u polju *data* u redoslijedu rastuća. Poredak elemenata u polju može biti proizvoljan i **polje se ne smije mijenjati**.

Primjer:
Ako je u polju `XY[] data = {{6, 8}, {7, 8}, {9.1, 13}, {-3, -2}, {-1, 4}}`, funkcija je rastuća (primjetiti da ne mora biti straga redoslijeda).
Ako je u polju `XY[] data = {{6, 8}, {7.5, 12}, {9, 9}, {10, 10}, {11, 11}}` ukazuje da funkcija nije rastuća.

Ako je u polje poslano prazno polje ili polje sa samo jednim elementom, metoda mora vratiti `true`. Možete prepostaviti da su vrijednost za x ispravno zadati (može se pojaviti ista vrijednost za x).

Napomena: U zadatku niste trebali napisati *KeyValue* i *XY* napisati bez modifikatora vidljivosti i kopirati u prostor za rješenje bez navođenja paketa kojem pripada. Na kraju zadatka učitajte ugradnjene Javine biblioteka.

Run

Save

Napisati program koji će provjeriti osnovnu validnost kreditne kartice. Potrebno je napisati klasu:

```
public static class CreditCard
```

i u njoj metode:

```
public static boolean checkCreditCard(String stringPAN, String cardType)
public static int getChecksum(Long numbPAN)
```

Metoda `checkCreditCard` provjerava PAN (16-znamenkasti broj na kartici) i tip kartice (Stringovi koji mogu biti vrijednosti "Diners", "Visa", "MasterCard" ili "AmEx").

Provjera se obavlja na sljedeći način:

- Ako je duljina broja neispravna, baca se iznimka: `InvalidPANLengthException`
- Ako se u ulaznom stringu pojavio nedozvoljeni znak (nešto što nije znamenka), baca se iznimka `InvalidPANCharacterException`
- Pomoću metode `getChecksum` (koja je objašnjena niže) potrebno je odrediti odgovara li očekivana znamenka unesenoj zadnjoj znamenki. Ako ne, baca se `InvalidPANChecksumException`.
- Ako prva znamenka ne odgovara pripadajućoj kartici, baca se `InvalidCardException`

Za ispravnu karticu, metoda mora vratiti `true`.

Prve znamenke i njihove odgovarajuće kartice su:

- 3 za Diners
- 4 za Visa
- 5 za MasterCard
- 6 za AmEx

Metoda `getChecksum` za ulazni parametar tipa `long` sumira sve znamenke osim zadnje. Zatim vraća zadnju znamenku broja izračunate sume.

NIJE potrebno pisati programski kod za klase:

```
InvalidCardException
InvalidPANLengthException
InvalidPANCharacterException
InvalidPANLengthException
```

The screenshot shows a code editor interface with a dark theme. On the left, there is a vertical line number column from 1 to 13. The main area contains 13 empty lines of code. To the right of the code area, there is a green 'Run' button and a white 'Save' button.

Napisati klasu `InvalidPeppaPigCharacterException` koja naslijeduje `RuntimeException` i sadrži konstruktor kojim se može postaviti poruke iznimke.

Napisati klasu `CartoonChecker` i u njoj javnu statičku metodu `processCharacter` koja prima jedan podatak tipa String i provjerava postoji li taj string u polju stringova koji se nalaze u klasi `Peppa`. Referenca na polje se može dobiti preko statičke metode `Peppa.getAllCharacters()`.

Ako predano ime `postoji` u polju stringova, metoda `processCharacter` na standardni izlaz ispisuje poruku `<IME> JE lik u crtanim filmu Peppa Pig`

Ako predano ime `ne postoji` u listi, metoda treba baciti `InvalidPeppaPigCharacterException` s porukom `<IME> NIJE lik u crtanim filmu Peppa Pig`.

Npr. nek polje dobiveno s `Peppa.getAllCharacters()` sadrži samo dva stringa, `{"Peppa", "George"}`. Ako iz glavnog programa metodu `processCharacter` pozivamo redom s parametrima "Peppa", "George" i "Traktor Tom", na standardni izlaz će se ispisati:

```
Peppa JE lik u crtanim filmu Peppa Pig
```

```
1
2 class InvalidPeppaPigCharacterException extends RuntimeException{
3
4     /**
5      *
6     */
```

Run

Save

Napišite klasu `TextFileAnalyser` sa statičkom metodom koja detektira na kojem je jeziku datoteka:

```
public static Language detectLanguage(Path file) throws IOException
```

Ova metoda može detektirati 3 jezika:

- hrvatski - ako datoteka ima samo hrvatska slova: šđčćžŠĐĆĆŽ
- njemački - ako datoteka ima samo njemačka slova: äöüÄÖÜß
- engleski - ako datoteka ima samo engleska slova (ASCII)
- nepoznato - ako ima slova iz više jezika

Treba napraviti enumeraciju `Language` koja ima sljedeće vrijednosti: `HR`, `DE`, `EN` i `UNKNOWN`.

Npr. ako ima slova iz HR i DE onda je rezultat nepoznato, a ako ima slova iz EN i HR onda je rezultat HR.

Primjer korištenja:

```
Path.of("example.txt");
TextFileAnalyser.detectLanguage(inputFile);
```

Napomena: Sve klase i sučelja imaju vidljivost postavljenu na *package private*.

Napomena: Prije predaje iz Vašeg koda izbacite sve ispise na standardni izlaz kako bi testovi ispravno radili.

41 }

| Correct | Incorrect | Partial | Unanswered | Score | Score % | Hint |
|---------|-----------|---------|------------|-------|---------|---------------------|
| ✓ | | | | 1 | 100 | Correct. Well done! |

sage

U sljedećem programskom odsječku, ulazna lista redom predstavlja znamenke nekog decimalnog broja oblika A.BCDE... (npr. 3.1425976).

Nadopunite odsječak koda

```
class Solution{

    public static Integer getPiSequenceBreakpoint(List<Integer> input){
        return IntStream.range(0, input.size()) /*finish with your code*/

    }

}
```

prema sljedećim zahtjevima:

- metoda za ulaznu listu brojeva vraća redni broj pozicije (počevši od 0) prve znamenke koja ne odgovara znamenici u broju Pi.
- U slučaju znamenka razlike ne postoji baca se iznimka `NoSuchElementException`
- pretpostavite da su dostupni sljedeći importi: `import java.util.*; import java.util.stream.Stream; import java.util.stream.IntStream; import java.util.stream.Collectors; import java.lang.Math.*;`
- uputa za predaju: u Edgar zalijepiti samo dio koji nedostaje u odsječku (tj. dio koji mora doći umjesto `/* finish with your code */`).

Primjer

Za listu s brojevima `3, 1, 4, 1, 5, 9, 2, 5, 5, 5` vratit će se `7`.

NAPOMENA:

- početni dio koda `IntStream.range(0, input.length)` generira tok prirodnih brojeva koji odgovaraju pozicijama u ulaznoj listi. U ostatku izraza ih možete koristiti kao indeksere.

The screenshot shows a dark-themed Java code editor. On the left, there is a vertical line of numbers from 1 to 13, likely representing line numbers. The main area contains the provided code snippet. In the bottom right corner of the code area, there are two buttons: a green "Run" button and a white "Save" button.

```
1
2
3
4
5
6
7
8
9
10
11
12
13
```

Na raspolaganju imate razred `Garden`.

```
class Garden {  
    private String name; // garden's name  
    private List<Plant> plants; // plants in the garden  
  
    public Garden(String name, List<Plant> plants) {  
        this.name = name;  
        this.plants = plants;  
    }  
  
    // GETTERS & SETTERS EXIST  
  
    @Override  
    public String toString() {  
        return "Garden [name=" + name + ", plants=" + plants + "]";  
    }  
}
```

Potrebno je dopuniti i predati razred `Plant` tako da implementirat metode `equals` i `hashCode`. Biljke su jednake ako imaju jednak latinski naziv. U razred trebate dodati i geterske i seterske metode.

```
class Plant {  
    private String latinName; // latin name for the plant  
    private String croName; // croatian name for the plant  
  
    public Plant(String latinName, String croName) {  
        super();  
        this.latinName = latinName;  
        this.croName = croName;  
    }  
  
    // GETTERS & SETTERS  
  
    @Override  
    public String toString() {  
        return "Plant [latinName=" + latinName + ", croName=" + croName + "]";  
    }  
}
```

Također, potrebno je implementirati i predati razred `Task` u kojem su definirane sljedeće dvije metode:

```
public static BiConsumer<Garden, List<Plant>> deletePlantsFunction()
```

- Metoda vraća `BiConsumer` koji djeluje nad objektom tipa `Garden` (predstavlja vrt) i `List<Plant>` (predstavlja uginule biljke).
- Consumer iz liste biljaka objekta `Garden` briše biljke (ako ih posjeduje) sadržane u listi `List<Plant>`.
- `BiConsumer` implementirati **lambda** izrazom.

```
public static BiFunction<Garden, Plant, Boolean> checkIfPlantIsInGardenFunction()
```

- Metoda vraća `BiFunction` koja djeluje nad objektom tipa `Garden` (predstavlja vrt) i objektom tipa `Plant` (predstavlja biljku).
- Funkcija provjerava nalazi li se u vrtu navedena biljka.
- `BiFunction` implementirati **lambda** izrazom.

NAPOMENA: Razrede `Plant` i `Task` napisati bez modifikatora vidljivosti, kao package-private.

PRIMJER IZVOĐENJA:

```
List<Plant> pList1 = new ArrayList<>();  
pList1.add(new Plant("Leucanthemum vulgare", "Ivancica"));  
pList1.add(new Plant("Pinus cembra", "Limba"));  
Garden g1 = new Garden("Bakin", pList1);  
  
List<Plant> pList2 = new ArrayList<>();  
pList2.add(new Plant("Calendula officinalis", "Neven"));  
pList2.add(new Plant("Pinus cembra", "Limba"));  
  
BiFunction<Garden, Plant, Boolean> checkBiFunction = Task.checkIfPlantIsInGardenFunction();  
System.out.println(checkBiFunction.apply(g1, new Plant("Broussonetia papyrifera", "Dudovac")));  
System.out.println(checkBiFunction.apply(g1, new Plant("Pinus cembra", "Limba")));  
  
System.out.println(g1.toString());  
BiConsumer<Garden, List<Plant>> checkBiConsumer = Task.deletePlantsFunction();  
checkBiConsumer.accept(g1, pList2);  
System.out.println(g1.toString());
```

IZLAZ:

```
false  
true  
Garden [name=Bakin, plants=[Plant [latinName=Leucanthemum vulgare, croName=Ivancica], Plant [latinName=Pinus cembra, croName=Limba]]]  
Garden [name=Bakin, plants=[Plant [latinName=Leucanthemum vulgare, croName=Ivancica]]]
```

```
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13
```

Run

Save

Potrebno je implementirati klasu `FruitFinder` kojom se upravlja kolekcija voća.

Podaci o voću su pohranjeni u strukturi `Fruit`. Voće ima svoje ime te svoju boju (jedan od vrijednosti enumeracije `FruitColor`).

```
class Fruit {  
  
    private final String name;  
    private final FruitColor color;  
  
    public Fruit(String name, FruitColor color) {  
        this.name = name;  
        this.color = color;  
    }  
  
    public String getName() {  
        return name;  
    }  
  
    public FruitColor getColor() {  
        return color;  
    }  
  
    @Override  
    public boolean equals(Object o) {  
        if (this == o)  
            return true;  
        if (o == null || getClass() != o.getClass())  
            return false;  
        Fruit fruit = (Fruit) o;  
        return Objects.equals(name, fruit.name) &&  
               color == fruit.color;  
    }  
  
    @Override  
    public int hashCode() {  
        return Objects.hash(name, color);  
    }  
  
    @Override  
    public String toString() {  
        final StringBuffer sb = new StringBuffer("Fruit{");  
        sb.append("name='").append(name).append('');  
        sb.append('}');  
        return sb.toString();  
    }  
}  
  
enum FruitColor {  
    ORANGE, RED, YELLOW  
}
```

Zadatak je implementirati sljedeće metode u klasi `FruitFinder`, definirane prototipom:

```
/**  
 * Returns a {@link Predicate<Fruit>} instance that will match a fruit by the provided color and  
 * check a fruit name contains the given pattern, at any point in the name, case sensitive.  
 *  
 * @param color the color to be matched  
 * @param pattern the pattern that is to be contained within the name  
 * @return the {@link Predicate} instance  
 */  
public static Predicate<Fruit> allFruitThatMatchColorAndPattern(  
    FruitColor color, String pattern  
)  
  
/**  
 * Returns a {@link Predicate<Fruit>} instance that will match a fruit if it is not the same  
 * color as provided and if the fruits name does not contain the given pattern, at any point in  
 * the name, case sensitive.  
 *  
 * @param color the color to be excluded  
 * @param pattern the pattern that can not be contained within the name  
 * @return the {@link Predicate} instance  
 */  
public static Predicate<Fruit> allFruitThatDoesNotMatchColorAndDoesNotContainPattern(  
    FruitColor color, String pattern  
)
```

NAPOMENA: klasu `FruitFinder` napisati bez modifikatora vidljivosti. Klasa `Fruit` i enumeracija `FruitColor` nalazi se u sustavu i ne trebate je predavati, nego samo koristiti.

Primjer korištenja vidljiv je u sljedećem isječku koda:

```
Fruit apple = new Fruit("Apple", FruitColor.RED);  
Fruit tangerine = new Fruit("Tangerine", FruitColor.ORANGE);  
  
List<Fruit> fruits = new ArrayList<>(Arrays.asList(apple, tangerine));  
List<Fruit> actual1 = fruits.stream()  
    .filter(FruitFinder.allFruitThatMatchColorAndPattern(FruitColor.RED, "pp"))  
    .collect(  
        Collectors.toList());  
  
System.out.println(actual1); // [Fruit{name='Apple'}]  
  
List<Fruit> actual2 = fruits.stream()  
    .filter(  
        FruitFinder  
            .allFruitThatDoesNotMatchColorAndDoesNotContainPattern(FruitColor.RED, "pp")  
    )  
    .collect(  
        Collectors.toList());  
  
System.out.println(actual2); // [Fruit{name='Tangerine'}]
```

1
2
3
4
5
6
7
8
9
10
11
12
13

Run

Save

U sljedećem programskom odsječku, ulazna lista redom predstavlja znamenke nekog decimalnog broja oblika A.BCDE... (npr. 3.1425976).

Nadopunite odsječak koda

```
class Solution{

    public static Integer getPiSequenceBreakpoint(List<Integer> input){
        return IntStream.range(0, input.size()) /*finish with your code*/

    }

}
```

prema sljedećim zahtjevima:

- metoda za ulaznu listu brojeva vraća redni broj pozicije (počevši od 0) prve znamenke koja ne odgovara znamencu u broju Pi.
- U slučaju znamenka razlike ne postoji baca se iznimka `NoSuchElementException`
- pretpostavite da su dostupni sljedeći importi: `import java.util.*; import java.util.stream.Stream; import java.util.stream.IntStream; import java.util.stream.Collectors; import java.lang.Math.*;`
- uputa za predaju: u Edgar zalijepiti samo dio koji nedostaje u odsječku (tj. dio koji mora doći umjesto `/* finish with your code */`).

Primjer

Za listu s brojevima `3, 1, 4, 1, 5, 9, 2, 5, 5, 5` vratit će se `7`.

NAPOMENA:

- početni dio koda `IntStream.range(0, input.length)` generira tok prirodnih brojeva koji odgovaraju pozicijama u ulaznoj listi. U ostatku izraza ih možete koristiti kao indeksere.

The screenshot shows a dark-themed Java code editor interface. On the left, there is a vertical column of line numbers from 1 to 13. The main code area contains the following Java code:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
```

To the right of the code area are two buttons: a green "Run" button and a white "Save" button.