Objektno orijentirano programiranje
Završni ispit
17.6.2020.

Ispit nosi ukupno 50 bodova i piše se 150 minuta.
U zadacima nije potrebno pisati dio u kojem se uključuju klase ili paketi klasa (import)

1. zadatak (10 bodova)

Potrebno je modelirati klase, sučelja i/ili enumeracije i odnose među njima temeljem donjeg opisa. Umjesto pisanja koda, možete nacrtati UML dijagram (po uzoru na sliku iz 4. zadatka).

Svaka tvornica ima ime i resurse, pri čemu resursi mogu biti ljudi i strojevi. Resursi se mogu dodavati i uklanjati iz tvornice. Za svaki resurs treba moći evidentirati broj sati (dohvatiti trenutne, upisati nove ili poništiti na nulu), a ljudima se mora moći isplatiti plaća temeljem broja sati, jedinične cijene sata (argument metode) i koeficijenta pojedine vrste radnika (vrijednosti tipa *double*). Roboti imaju svoj identifikator (*String*) te im se može poslati programski kod zadatka koji treba obaviti (*byte[]*). Svaka osoba opisana je svojim OIB-om, imenom i datumom rođenja. Svaki ljudski resurs ima svoju posebnu kutiju alata. Svaki alat je opisan imenom i vrstom alata (postavlja se u konstruktoru i ima gettere) i u kutiji može biti više istih primjeraka. Za tvornicu iz primjera evidentiraju se majstori i poljoprivredni radnici, pri čemu su majstorski alati neki od navedenih: bušilica, čekić, kliješta, odvijač, ključ, ostalo, a poljoprivredni: lopata, kramp, motika, grablje, ostalo.

2. zadatak (10 bodova)

Vozilo (`Vehicle`) je definirano svojom težinom (`weight:int`), konjskim snagama (`hp:int`) i imenom (`name:String`) koji se postavljaju u konstruktoru i nakon toga se ne mogu mijenjati.

Vozila se moraju moći dodati u uobičajene kolekcije u Javi. Vozila se smatraju jednakim ako su im sva 3 atributa jednaka.

Prirodni poredak vozila je takav da se prvo gleda koliko vozilo ima konjskih snaga, potom se vrši usporedba po težini, a nakon toga po imenu. Prilikom usporedbe po imenu treba obratiti pažnju da imena mogu sadržavati hrvatske znakove.

Klasa `Vehicle` treba sadržavati i statičke varijable koji predstavljaju komparatore po pojedinom atributu.

Dovršite kod klase `Vehicle` unutar okvira na sljedećoj stranici.

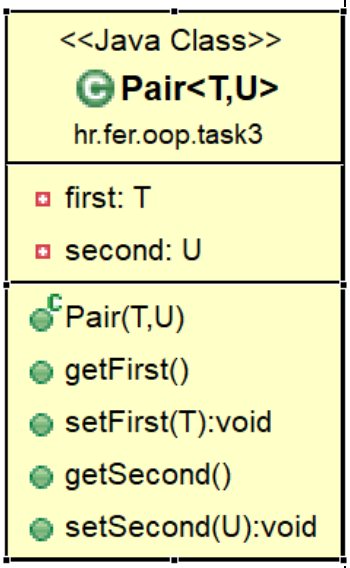
```
public class Vehicle _____ {
  private static Comparator<Object> hrcomparator
                              = Collator.getInstance(Locale.forLanguageTag("hr"));
  private String name; private int hp; private int weight;
  public Vehicle(int weight, int hp, String name){
       this.name = name; this.weight = weight; this.hp = hp;
  }
  public String getName() {      return name; }
  public int getWeight() { return weight; }
  public int getHp() { return hp; }
  @Override
  public String toString() {
       return String.format("%s %dkg %dKS", getName(), getWeight(),getHp());
  }
  // TODO: Nadopuniti
```

```
  public static _____ BY_NAME = _____;

  public static _____ BY_HP = _____;

  public static _____ BY_WEIGHT = _____;
}
```

3. zadatak (10 bodova)

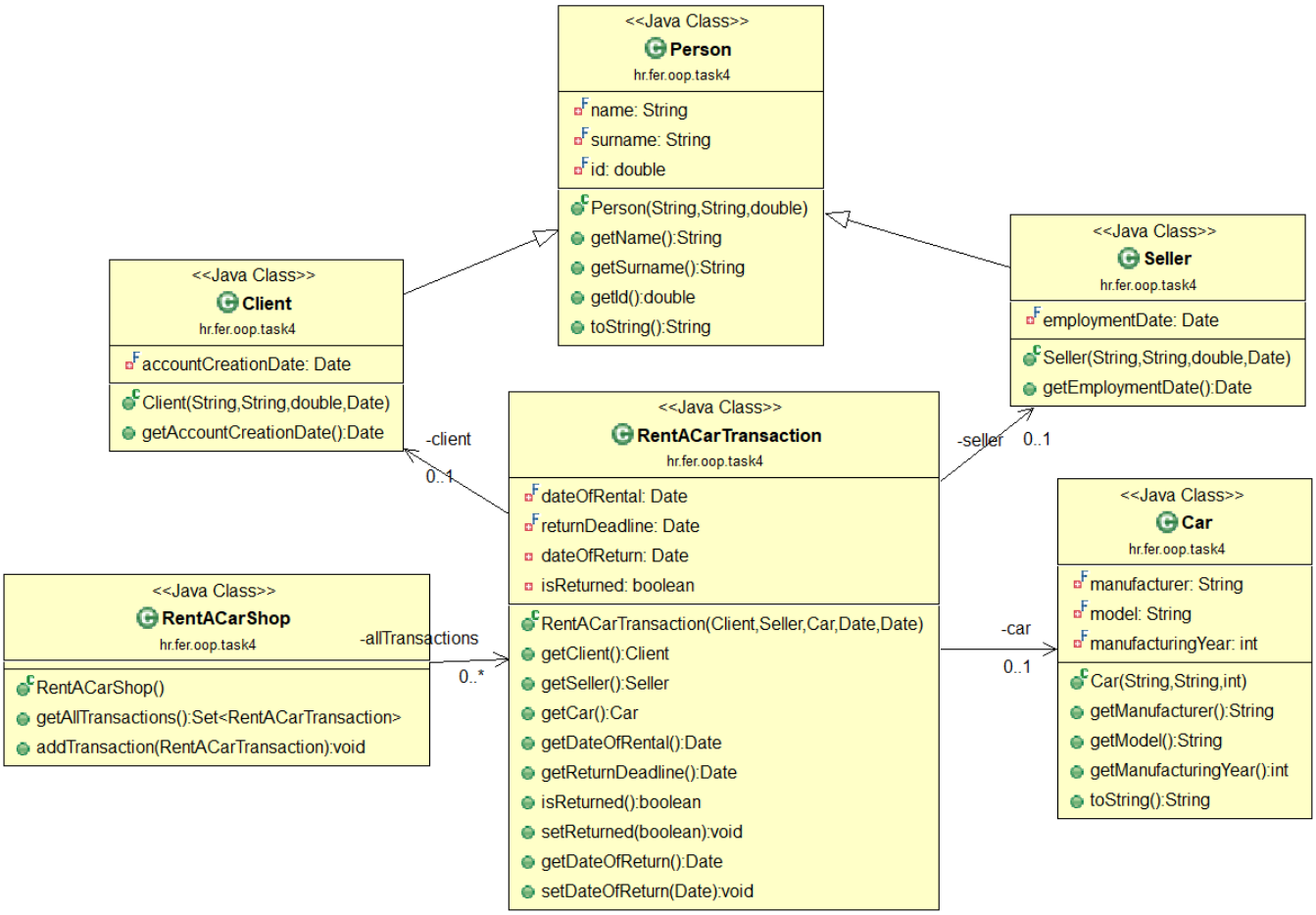
Napisati klasu `VehicleCollection` koja predstavlja kolekciju vozila (duplikati su dozvoljeni, a poredak je nebitan) i omogućava dodavanje jednog ili varijabilnog broja vozila u kolekciju i uklanjanje jednog primjerka nekog vozila iz kolekcije. Iteriranje kroz kolekciju se vrši po obrnutom prirodnom poretku vozila. Klasa `Pair`, nije izvedena iz neke druge klase, ne implementira nijedno sučelje, ne može se mijenjati ni naslijediti. *Napomena. U zadatku smijete koristiti lambda izraze, anonimne klase, reference na metode, ugrađene komparatore, ... ali nije dozvoljeno koristiti kolekcijske tokove.*

```
public class VehicleCollection implements Iterable<Pair<Vehicle, Integer>> {


      public void add(Vehicle vehicle) {



       }
       public void add(_____________________________________) {


       }
       public void remove(Vehicle vehicle) {




       }
       @Override
       public Iterator<Pair<Vehicle, Integer>> iterator() {









}
```



4. zadatak (10 bodova)

Slika prikazuje sustav jednostavne poslovnice za posudbu automobila. Datumi su evidentirani tipom podatka `Date` koji implementira sučelje `Comparable<Date>`, a posjeduje i metode `boolean before(Date)` i `boolean after(Date)`.



Neka je `s` tipa `Stream<RentACarTransaction>`. Nadopunite sljedeće izraze tako da vraćaju:

a) skup svih automobila koji su ikad bili posuđeni:

```
Set<Car> set = s.
```

b) skup svih automobila novijih od 2015.g. (automobili iz te godine nisu uključeni) koji su ikad bili posuđeni

```
Set<Car> set = s.
```

c) skup svih klijenata koji imaju barem jedan nevraćeni auto:

```
Set<Client> set = s.
```

d) skup svih klijenata koji su barem jednom prekoračili rok posudbe:

```
Set<Client> = s.
```

e) Listu svih prodavača koji su ikad napravili neku transakciju, sortiranih uzlazno po datumu zaposlenja

```
List<Seller> list = s.
```

5. zadatak (10 bodova)

U označeni okvir upišite što ispisuje sljedeći program. Sve definirane klase pripadaju istom paketu i u svakoj klasi na vrhu su uključeni potrebni paketi te još dodatno `import static java.lang.System.out;`

```
public class NoGasException
             extends Exception {
  public NoGasException(String msg) {
       super(msg);
  }
}

public class Vehicle implements Closeable {
  int remaining;
  String id;

  public Vehicle(String id, int mileage)
                    throws NoGasException {
       this.id = id;
       out.println(id);
       if (mileage < 0)
          throw new NoGasException(
                     "Remaining mileage "
                      + "must be above 0;");
       else
          this.remaining = mileage;
   }

   public void ride(int km) {
       if (km > remaining)
         throw new
             IllegalArgumentException(
                        "Ride to long!");
       else {
         out.printf("Riding %d km%n",km);
         remaining = remaining - km;
       }
   }

   public int getRemainingMileage() {
       return remaining;
   }

   @Override
   public void close() throws IOException {
       out.println("Ride over for " + id);
   }
}


public class Car extends Vehicle {
  public Car(String id, int remaining)
                  throws NoGasException {
       super(id, remaining);
  }

  @Override
  public void close() throws IOException {
       super.close();
       out.println("Car has finished" +
                   "the ride!");
       }
}
```

```
public class Main {
 public static void main(String[] args)
                          throws Exception {
   Vehicle v = null;
   Car c = null;
   Car c1 = null;

   try (Vehicle v1 = new Vehicle("V1", 5)) {
       v = new Vehicle("V", 12);
       c = new Car("C", -12);
       c1 = new Car("C1", -15);
   } catch (NoGasException e) {
         out.println(e.getMessage());
   } catch (Exception e) {
         out.println("Riding exception!");
   } finally {
         out.println("All vehicles created!");
         c1 = new Car("C1.1", 50);
   }

   try {
       c.ride(45);  c1.ride(51); v.ride(2);
   } catch (NullPointerException e) {
       out.println("Null pointer exception");
   } catch (Exception e) {
       out.println("Exception");
   } finally {
       out.println("Rides completed!");
       ((Vehicle)c1).close();
   }

   out.println(v.remaining);
 }
}
```

Ovdje upisati rješenje: