Objektno orijentirano programiranje
1. međuispit

27.11.2015.

Ispit nosi ukupno 25 bodova i piše se 120 minuta. Ispit ima 5 zadataka.

1. zadatak (5 bodova)

Napišite sadržaj metoda `toString` u klasama `A`, `B` i `C`, a zatim napišite jedan ili više konstruktora u klasi `B` da bi sljedeći programski odsječak dao ispis kao što je navedeno u komentarima. Napomena: ostatak koda ne smijete mijenjati. **Rješenje upisati u prazne pravokutnike na ovom ispitu.**

```
public static void main(String [] args) {
   C c1 = new C();
   C c2 = new C(1, 2, 3, 4);
   System.out.println(c1); // ispisuje: x=0 y=0 z=0 w=0
   System.out.println(c2); // ispisuje: x=1 y=2 z=3 w=4
}
```

```
public class A {
   private int x=0, y=0;
   public A(int x, int y){
      this.x = x; this.y = y;
   }
   public String toString(){
      // TODO napisati sadržaj metode
```

```
   }
}
public class B extends A {
   private int z = 0;
      // TODO dodati konstruktore
```

```
   public String toString() {
      // TODO napisati sadržaj metode
```

```
   }
}
```

```
public class C extends B {
   private int w=0;
   public C() {
   }
   public C(int x, int y, int z, int w) {
      super(x, y, z);
      this.w = w;
   }
   public String toString(){
      // TODO napisati sadržaj metode
```

```
   }
}
```

2. zadatak (5 bodova)

U prvom dijelu je naveden programski kôd u kojem je nestatička klasa `CarLight` ugniježđena u klasi `Car`.

```
public abstract class Part {
	public Integer serialNum;
	public abstract void powerUp();
	public Part(Integer serialNum){
		System.out.println("Part created!");
		this.serialNum = serialNum;
	}
	 public String toString() {
		return this.serialNum.toString();
	}
}
```

```
public class Car {
	private boolean isLocked=false;
	private String type;

	public Car(String type) {
		this.type = type;
	}
	public Part createLight(int serialNum) {
		return new CarLight(serialNum);
	}
	private class CarLight extends Part {
		private boolean isLightsOn = false;

		private CarLight (int serialNum) {
		  super(serialNum);
		}
		@Override
		public void powerUp() {
		  if(!isLocked){
		   this.isLightsOn = true;
		   System.out.println("Lights on!");
		     }
		}
		public boolean getLightsOn() {
			return this.isLightsOn;
		}
		public void setLightsOn(boolean val) {
			this.isLightsOn = val;
		}
	}

	 public static void main(String[] args) {
		Car fiat = new Car("Fiat");
		Car skoda = new Car("Skoda");
		Part fiatLight = fiat.createLight(126985);
		Part skodaLight = skoda.createLight(439580);
		fiatLight.powerUp();
		skodaLight.powerUp();
		display (fiatLight);
		display (skodaLight);
	 }
	private static void display(Part part) {
	   System.out.println(part.toString() + ": " +
			((CarLight)part).getLightsOn());
	}
}
```

a) Potrebno je navesti što se ispisuje nakon pokretanja navedenog odsječka kôda. **Rješenje upisati u prazni pravokutnik.**

b) Navedeni je odsječak potrebno nadopuniti kôdom na za to označenim mjestima kako bi program bio sintaksno ispravan, izvodio se bez greški te ispisivao isto kao i gore zadani program. Napomena: u ovom je odsječku **statička klasa** `CarLight` ugniježdena u klasi `Car`.
Rješenje upisati na prazne linije na ovom ispitu.

```
public abstract class Part{
       public Integer serialNum;
       public abstract void powerUp();
       public Part(_____________________________________){
              System.out.println("Part created!");
              this.serialNum = _____________________________________;
       }
        public String toString() {
              return ____________________________________
       }
}
```

```
public class Car {
       private boolean isLocked=false;
       private String type;

       public Car(String type) {
              this.type = type;
       }
       public Part createLight(_____________________________________) {

              return new CarLight(_____________________________________);
       }
       public static class CarLight extends Part {
              private boolean isLightsOn = false;


             ____________________________________

              public CarLight (_____________________________________) {
                super(serialNum);
                ____________________________________
              }
             @Override
              public void powerUp() {
                if(!car.isLocked){
                  this.isLightsOn = true;
                  System.out.println("Lights on!");
                }
              }
              public boolean getLightsOn() {
                     return this.isLightsOn;
              }
              public void setLightsOn(boolean val) {
                     this.isLightsOn = val;
              }
       }

       public static void main(String[] args) {
              Car fiat = new Car("Fiat");
              Car skoda = new Car("Skoda");
              Part fiatLight = ___________________________________

             Part skodaLight = ____________________________________
              fiatLight.powerUp();
              skodaLight.powerUp();
              display (fiatLight);
              display (skodaLight);
       }
        private static void display(Part part) {
          System.out.println(part.toString() + ": " +
                 ((CarLight)part).getLightsOn());
        }
}
```

3. zadatak (5 bodova)

Potrebno je modelirati klase i sučelja koje se koriste u aplikaciji za prodaju vozila (`vehicle`). Postoje tri vrste vozila: automobil (`car`), kamion (`truck`) i motocikl (`motorcycle`). Svaka vrsta vozila ima definiran naziv proizvođača (`manufacturer`), oznaku modela (`model`), jedinstveni serijski broj šasije (`id`), snagu (`power`), broj putnika (`numberOfPassangers`) i cijenu (`regularPrice`). Gettere i settere nije potrebno pisati. Automobili dodatno imaju oznaku posjedovanja klimatizacije (`hasAC`) i broja vrata (`numberOfDoors`). Kamioni imaju definiranu nosivost (`maxWeight`), a motocikli oznaku tipa (`type`) koja mora biti jedna od sljedećih: `chopper, sport, racing i off-road`. Za vozila je potrebno napraviti metodu `public double getPrice(boolean isVIP)` koja za korisnike koji nisu VIP vraća regularnu (tj. definiranu) cijenu vozila, a za VIP korisnike vraća regularnu cijenu umanjenu za određeni postotak ovisno o tipu vozila tako da je popust na motocikle 15%, na automobile 10%, a na kamione 5%. Prilikom rješavanja zadatka po potrebi je moguće dodati nove metode. Vrste vozila koje se mogu iznajmljivati su automobili i motocikli, a njih je potrebno modelirati sučeljem koje ima dvije metode: `public void addRentingPrice(double rentingPrice)` i `public double getRentingPrice()`.

U glavnom programu je potrebno ispisati ukupnu sumu cijena iznajmljivanja svih koja se mogu iznajmiti, a nalaze se u bazi podataka. Pri tome pretpostavite da postoji metoda `loadVehicles()` klase `DBLoader` koja učitava sva vozila iz baze podataka i vraća ih u obliku skupa `Set<Vehicle>;` tu metodu nije potrebno pisati.

Rješenje napisati posebnom listu papira i označiti da se radi o 3. zadatku.

4. zadatak (5 bodova)

Parametrizirani razred `Tuple` predstavlja model uređenog para (e1,e2). Pri tome tip prvog i drugog elementa ne mora biti jednak. Kod u nastavku ilustrira uporabu ovog razreda.

```
public class Main {
	public static void main(String[] args) {
		Iterable<Tuple<String,Integer>> grades = Arrays.asList(
			new Tuple<>("Ivo", 5),
			new Tuple<>("Ivo", 3),
			new Tuple<>("Anica", 5),
			new Tuple<>("Ana", 4),
			new Tuple<>("Ana", 3),
			new Tuple<>("Jasna", 5)
		);

		Tuple<String,Integer> first = grades.iterator().next();
		System.out.printf("Ime: %s, ocjena: %d%n", first.getE1(), first.getE2());

		TupleHelper.processTuples(
			grades,
			(e1,e2) -> e1.length()==3 && e2>3,
			(e1,e2) -> System.out.printf("(%s,%d)%n", e1, e2)
		);
	}
}
```

Razred `TupleHelper` sadrži metodu *`processTuples`* koja nad nizom uređenih parova koji zadovoljavaju uvjet zadan drugim argumentom primjenjuje akciju definiranu trećim argumentom. Pogledajte primjer koji je dan u glavnom programu. Ispis programa će biti:

```
Ime: Ivo, ocjena: 5
(Ivo,5)
(Ana,4)
```

Vaši zadatci su sljedeći.

a) Napišite izvorni kod parametriziranog razreda `Tuple`.
b) Uvjet i akciju modelirajte prikladnim parametriziranim sučeljima. Za broj i vrstu argumenata metoda u tim sučeljima pogledajte primjer uporabe u prikazanoj metodi `main`. Za oba sučelja napišite njihov izvorni kod.
c) Napišite izvorni kod razreda `TupleHelper` s navedenom metodom.
d) Napišite ponovno poziv metode `processTuples` koji je prikazan u metodi `main`, ali bez uporabe lambda izraza: umjesto toga napišite kod koji eksplicitno stvara objekt (anonimnog razreda) čiju referencu šalje u metodu.

Rješenja napisati posebnom listu papira i označiti da se radi o 4. zadatku.

5. zadatak (5 bodova)

Za razred *A* definiran kao u lijevom okviru ispišite što ispisuje program iz desnog okvira. Dijeljenje s nulom izaziva iznimku `ArithmeticException`, a pokušaj pristupa elementu van polja izaziva `ArrayIndexOutOfBoundsException`.

Rješenje upisati u iscrtkani okvir na ovom ispitu.

```
public class A implements AutoCloseable {
  private int x;
  public A(int x){
       this.x = x;
       System.out.println("ctor " + x);
  }
  @Override
  public void close() throws Exception {
       System.out.println("close " + x);
  }
}
```

```
public static void main(String[] args) {
  int[] arr = new int[] { 15, 10 };
  try {
       A a40 = new A(40);
       for(int i=1; i>=-1; i--)
           m1(arr, i);
       A a50 = new A(50);
  }
  catch(Exception e) {
       System.out.println("main exc");
  }
  finally {
       System.out.println("main finally");
  }
  System.out.println("main done");
}

private static void m1(int[] arr, int i) throws Exception {
  try(A a1 = new A(i)) {
       m2(arr[i] + arr[i+1], i);
  }
  catch(ArrayIndexOutOfBoundsException e) {
       System.out.println("m1 exc");
  }
  System.out.println("m1 done");
}

private static int m2(int x, int y) throws Exception {
   int r = 0;
   try(A a2 = new A(x+y)) {
       r = x / y;
   }
   finally {
       System.out.println("m2 finally");
   }
   System.out.println("m2 done");
   return r;
}
```