Sve principe naučene na prethodnim vežbama (kreiranje nove klase, apstraktne klase, apstraktne metode, nasleđivanje, *override* metode, kreiranje novih objekata klasa, poziv metoda definisanih u klasama i sl.), kada smo prošli osnove OOP u programskom jeziku JAVA, nastavljamo da koristimo i nadograđujemo i ove nedelje. Dodatno, konzolna aplikacija za vođenje evidencije o raspoloživim artiklima u jednoj prodavnici računara, koju treba da isprogramiramo, ima značajno veću programersku kompleksnost od zadatka sa prethodnih vežbi i zato moramo paziti na svaki detalj.

Prvi zadatak predstavlja kreiranje apstraktne klase Artikal (podsetite se sa predavanja i prethodnih vežbi šta su apstraktne klase i kako se kreiraju). Iz teksta zadatka vidimo da je svaki artikal opisan sa tri atributa: Šifra artikal (tipa String), Naziv artikla (tipa String) i Cena artikla (tipa double), koji će imati *protected* modifikator pristupa. Kao što smo već rekli na prethodnim vežbama, svaka klasa mora imati konstruktor (istog imena kao i klasa), i mi biramo da konstruktor klase Artikal kao ulazne parametre ima sva tri atrubuta date klase (za razliku od prethodnog zadatka gde su prikazani različiti tipovi konstruktora, od ovog zadatka ćemo koristiti samo jedan tip koji kao ulazne parametre prima sve atribute odgovarajuće klase).

```
13  public abstract class Artikal {
14
15      protected String sifra;
16      protected String naziv;
17      protected double cena;
18
19      public Artikal (String sifra, String naziv, double cena)/*konstruktor koji
20                                                               kao ulazne param.
21                                                               prima sve atribute
22                                                               klase Artikal*/
23      {
24          this.cena=cena;
25          this.naziv=naziv;
26          this.sifra=sifra;
27      }
```

Ispis podataka o svakom artiklu treba da bude jedinstven, u obliku **„šifra, naziv, cena“**, što znači da moramo izvršiti *override* metode **String toString()** na sledeći način:

```
29      @Override
        public String toString()
31      {
32          return sifra + " , "+naziv+" , "+cena;
33      }
34
35  }
```

Možemo primetiti da klasa Artikal nema nijednu apstraktnu metodu. Da li onda ona može biti apstraktna? Naravno, potrebno je samo staviti ključnu reč **abstract** ispred imena klase, kao što možemo videti, i tada sva pravila vezana za apstraktne klase u potpunosti važe. Jedini slučaj kada klasa **mora** biti

apstraktna, tj. kada nemamo mogućnost odlučivanja, je kada ona **ima makar jednu apstraktnu metodu** (kao što smo već objasnili na prethodnim vežbama).

Najvažniji korak u ovakvim zadacima predstavlja jasno razumevanje samog teksta zadatka, tj. pravilno shvatanje odnosa između klasa (ko koga nasleđuje). Ako pažljivo pročitamo tačku 2. zadatka, vidimo da se kaže sledeće: „Kreirati klasu koja predstavlja komponentu računara. **Komponenta je artikal** koji se prodaje i opsana je sledećim podacima: Kategorija.". Ovde je jako bitno da budemo pažljivi, pošto nam boldovani deo jasno kaže da komponenta predstavlja jednu vrstu artikla, odnosno programerskim jezikom, klasa Komponenta **nasleđuje** klasu Artikal (ključna reč **extends**). Takođe, rečeno je da je svaki objekat klase Komponenta opisan jednim atributom - kategorijom (tipa String, postoje dve moguće kategorije – Memorija i Procesor).

```
public class komponenta extends Artikal {

    protected String kategrija;
```

Kod konstruisanja konstruktora klase Komponenta, postavlja sa pitanje broja ulaznih parametara. Da li će to biti samo jedan String (čiju vrednost ćemo dodeliti kategorji) ili ipak postoje još neki podaci koji su nam neophodni? Jedna od najvažnijih stvari koje traba da zamaptite sa ovih vežbi kaže da svaka klasa naslednica **preuzima sve atribute roditeljske klase. To znači da će se osim String-a koji opisuje kategoriju, kao ulazni parametari konstruktora klase Komponenta naći i svi atributi roditeljske klase Artikal (sifra, naziv, cena) čiji konstruktor će biti pozvan unutar konstruktora klase Komponenta. Poziv konstruktora roditeljske klase se izvršava pomoću ključne reči super, i jako je bitno napomenuti da redosled parametara mora ostati identičan (u ovom slučaju String sifra, String naziv, double cena), pošto će u suprotnom (npr. String sifra, double cena, String naziv) kompajler prijaviti grešku.**

```
    public komponenta (String kategorija, String sifra, String naziv, double cena) /* konstruktor
            klase komponenta preuzima sve atribute roditeljske klase (Artikal)*/
    {
        super (sifra, naziv,cena); // poziv konstruktora roditeljske klase
                                   // VODITI RACUNA O REDOSLEDU NAVODJENJA ATRIBUTA
        this.kategrija=kategorija;
    }
```

Takođe, opet je neophodno obezbediti jedinstveni ispis (u obliku **„kategorija, šifra, naziv, cena"**), tj. *override* metode toString(). Deo **„šifra, naziv, cena"** smo već *override*-ovali u klasi Artikal, pa je potrebno samo pozvati metodu toString() roditeljske klase (pomoću ključne reči **super**).

```
    @Override
    public String toString()
    {
        return kategrija+" , "+super.toString();
    }
```

Sledeći korak predstavlja impelentaciju klase Memorija. Opet, posebnu pažnju posvećujemo delu teksta u kom se kaže „**Memorija je jedna od specijalizovanih komponenti računara**", što automatski znači da

klasa Memorija predstavlja **klasu naslednicu** klase **Komponenta**. Svaki objekat klase Memorija opisan je pomoću njegovog kapaciteta (tipa int).

```
12    public class Memorija extends komponenta {
13
14        protected int kapacitet;
```

Što se tiče konstruktora klase Memorija, situacija je ista kao za klasu Komponenta, tj. svi atributi roditeljske klase (u ovom slučaju klase Komponenta) se nasleđuju. To faktički znači da će se kao ulazni parametri u konstruktoru klase Memorija, osim kapaciteta, naći i svi parametri konstruktora roditeljske klase (klase **Komponenta**), ali s obzirom da znamo da je, u slučaju klase Memorija, kategorija upravo Memorija, dati parametar se može fiksirati. Redosled navođenja parametara u konstruktoru roditeljske klase (koji se poziva ključnom reči **super**) mora biti **identičan** kao u samoj roditeljskoj klasi.

```
15
16        //konstruktor klase Memorija preuzima sve atribute roditeljske klase -
17        //komponenta
18        public Memorija ( String sifra, String naziv, Double cena, int kapacitet )
19        {
20            super ("Memorija", sifra, naziv, cena); //parametar kategorija moze da
21                                                    //se fiksira, posto znamo da ce
22                                                    //uvek biti Memorija
23                                                    //kategorija="Memorija"
24            this.kapacitet=kapacitet;
25        }
26
27        /*ALTERNATIVA*/
28        /*
29            public Memorija (String kategorija, String sifra,
30                             String naziv, Double cena, int kapacitet )
31        {
32            super (kategorija, sifra, naziv, cena); // bez fiksiranja kategorije na
33                                                    // Memoriju
34            this.kapacitet=kapacitet;
35        }
36        */
```

Ispis svakog objekta klase Memorija biće u obliku **„kategorija, šifra, naziv, cena, kapacitet“.** Opet je neophodno pozvati metodu toString() roditeljske klase (pokriva deo **„kategorija, šifra, naziv, cena“**)) i sa njom konkatenovati podatak o kapacitetu, na sledeći način:

```
38        @Override
39        public String toString()
40        {
41            return super.toString()+" , "+kapacitet;
42        }
```

Klasa Procesor predstavlja još jednu **komponentu** računara (što znači da nasleđuje klasu Komponenta), i sve što smo naveli za klasu Memorija odnosi se i na klasu Procesor. Opet su nasleđeni svi atributi

roditeljske klase, pri čemu se kategorija može fiksirati na Procesor. Svaki objekat klase Procesor je opisan pomoću dva integer-a – radnim taktom i brojem jezgara.

```
12  public class procesor extends komponenta {
13
14      protected Integer RadniTakt, BrojJezgara;
15
16
17      public procesor (String sifra, String naziv, Double Cena,
18                       Integer RadniTakt, Integer BrojJezgara)
19      {
20          super ("Procesor", sifra, naziv, Cena);//parametar kategorija moze da
21                                                 //se fiksira, posto znamo da ce
22                                                 //uvek biti Procesor
23                                                 //kategorija="Procesor"
24          this.RadniTakt=RadniTakt;
25          this.BrojJezgara=BrojJezgara;
26
27      }
28
29       /*ALTERNATIVA*/
30      /*
31          public procesor (String kategorija, String sifra,
32                           String naziv, Double cena, int kapacitet )
33      {
34          super (kategorija, sifra, naziv, cena); // bez fiksiranja kategorije na
35                                                  // Procesor
36          this.RadniTakt=RadniTakt;
37          this.BrojJezgara=BrojJezgara;
38      }
39      */
```

Ispis podataka o svakom objektu klase Procesor je oblika **„kategorija, šifra, naziv, cena, radni takt, broj jezgara“,** pa primenjujemo identičan princip kao za klasu Memorija:

```
40
41      @Override
        public String toString()
43      {
44          return super.toString() + "," + RadniTakt + "," + BrojJezgara;
45      }
```

Klasa **Prodavnica** predstavlja zasebnu klasu, koja treba da omogući korisniku prodaju artikala, dodavanja novog artikla, proveru preostalog broja artikala i ispis svih artikala koji se trenutno nalaze u prodavnici. Ovde takođe moramo obratiti pažnju na tekst zadatka, koji kaže da je prodavnica opisana **listom** artikala. Automatski, to nas navodi na korišćenje *ArrayList*-e (pogledati prethodne vežbe) za skladištenje različitih artikala. Neophodno je opet uključiti (importovati) odgovarajući paket koji podržava rad sa *ArrayList*-ama. Prilikom kreiranja novog objekta klase Prodavnica, data lista je prazna, tj. unutar konstruktora klase Prodavnica kreira se novi objekat klase *ArrayList*.

```
14    public class Prodavnica {
15
16        private ArrayList<Artikal> prodavnica; //atribut klase Prodavnica
17
18        public Prodavnica () //konstruktor klase prodavnica
19        {
20            prodavnica = new ArrayList(); //kreira se novi objekat klase ArrayList
21                                          //tj. prayna lista artikala
22        }
```

Prodaja odgovarajućeg artikla u stvari predstavlja samo uklanjanje datog artikla iz naše liste. Funkcija koja će ovo omogućiti nema povratnu vrednost (samo se uklanja artikal), pa je tipa *void*, a kao ulazni parametar moramo navesti objekat klase Artikal koji želimo da prodamo (uklonimo). Data funkcija mora biti *public* kako bi joj pristupili iz drugih funkcija (za testiranje). Uklanjanje artikla iz liste se vrši pozivom metode *remove()* klase *ArrayList.*

```
29        public void prodaja(Artikal a)
30        {
31            prodavnica.remove(a);
32        }
```

Ako govorimo o funkciji za dodavanje novog artikla, ona je takođe *public* i *void,* a kao ulazni parametar prima objekat klase Artikal koji želimo da dodamo. Dodavanje novog artikla u listu se vrši pozivom metode *add()* klase *ArrayList.*

```
34        public void dodaj(Artikal a)
35        {
36            prodavnica.add(a);
37        }
```

Provera preostalog broja artikala može se dobiti kao veličina liste artikala (dati artikli su još uvek u prodavnici). S obzirom da ova funkcija mora da vrati **broj** artikala, tj. ima povratnu vrednost tipa integer, i sama funkcija mora biti tipa integer, pri čemu data funkcija nemi nijedan ulazni parametar. Provera veličine liste artikala se vrši pozivom metode *size()* klase *ArrayList.*

```
24        public int TrenutniBrojArtikala()
25        {
26            return prodavnica.size();
27        }
```

Za ispis svih artikala koji se nalaze u prodavnici, neophodno je proći kroz celu listu, i u tu svrhu koristićemo *for-each* petlju:

```
39      public void ispis()
40      {
            for (Artikal a:prodavnica ) //for-each petlja
42
43          {
44              System.out.println(a);
45          }
46      }
```

Testiranje našeg programa ćemo izvršiti u funkciji *main()* gde ćemo kreirati nekoliko objekata klase Procesor, klase Memorija kao i jedan objekat klase Prodavnica. Takođe, istestiraćemo metode klase Prodavnica za dodavanje artikla, prodaju artikla, ispis svih artikala u prodavnici kao i proveru trenutnog broja artikala u prodavnici.

```
19      public static void main(String[] args) {
20          Memorija m1= new Memorija("E1025", "Intel", 2500., 2500); //kreiranje
21                                                                    //nove
22                                                                    //memorije
23          Memorija m2= new Memorija("E1026", "Intel", 2102., 2500);
24          Memorija m3= new Memorija("E1027", "Intel", 2700., 2500);
25
26          procesor p1= new procesor("E1025", "Intel", 2500., 2500,5);//kreiranje
27                                                                     //novog
28                                                                     //procesora
29          procesor p2= new procesor("E1025", "Intel", 2500., 2500,5);
30          procesor p3= new procesor("E1025", "Intel", 2500., 2500,5);
31
32          Prodavnica p=new Prodavnica();//kreiranje nove prodavnice
33
```

```
34
35          p.dodaj(m1);//dodavanje novog artikla u prodavnicu (Memorije m1)
36          p.dodaj(m2);
37          p.dodaj(m3);
38
39          p.dodaj(p2);
40
41          System.out.println("Pocetno stanje");
42          System.out.println("Broj artikala: " + p.TrenutniBrojArtikala());
43          p.ispis();//ispis svih artikala u prodavnici
44
45          p.prodaja(m1);//prodaja odgovaraju'eg artikla (Memorije m1)
46
47          p.prodaja(p2);
48
49          System.out.println("Nakon prodaje");
50          System.out.println("Broj artikala: " + p.TrenutniBrojArtikala());
51          p.ispis(); //ispis artikala u prodavnici nakon prodaje Memorije m1
52                     // i procesora p2
53      }
```

Kao izlaz našeg programa, na konzoli dobijamo sledeći ispis:

```
Pocetno stanje
Broj artikala: 4
Memorija , M1025 , Intel , 2500.0 , 2500
Memorija , M1026 , Intel , 2102.0 , 2500
Memorija , M1027 , Intel , 2700.0 , 2500
Procesor , P1029 , Intel , 2500.0,2500,5
Nakon prodaje
Broj artikala: 2
Memorija , M1026 , Intel , 2102.0 , 2500
Memorija , M1027 , Intel , 2700.0 , 2500
```

Na sajtu je postavljen kako tekst zadatka, tako i njegovo celokupno rešenje (sa video snimkom).

Za bilo koju vrstu pitanja možete mi se obratiti na mail vukan.ninkovic@gmail.com.