Siniša Nikolić

Java Web Development kurs – Termin 05

Sadržaj

- Upoznavanje sa mehanizmima nasleđivanja
  - Princip nasleđivanja
  - Apstraktne klase
  - Polimorfizam
- Implementacija interfejsa u Javi,
- Objašnjavanje Java *Collection* frejmvorka

Dodatni materijal:

- Generics in Java
- Statičko povezivanje (*static/early binding*) i dinamičko (*dynamic/late binding*) povezivanje

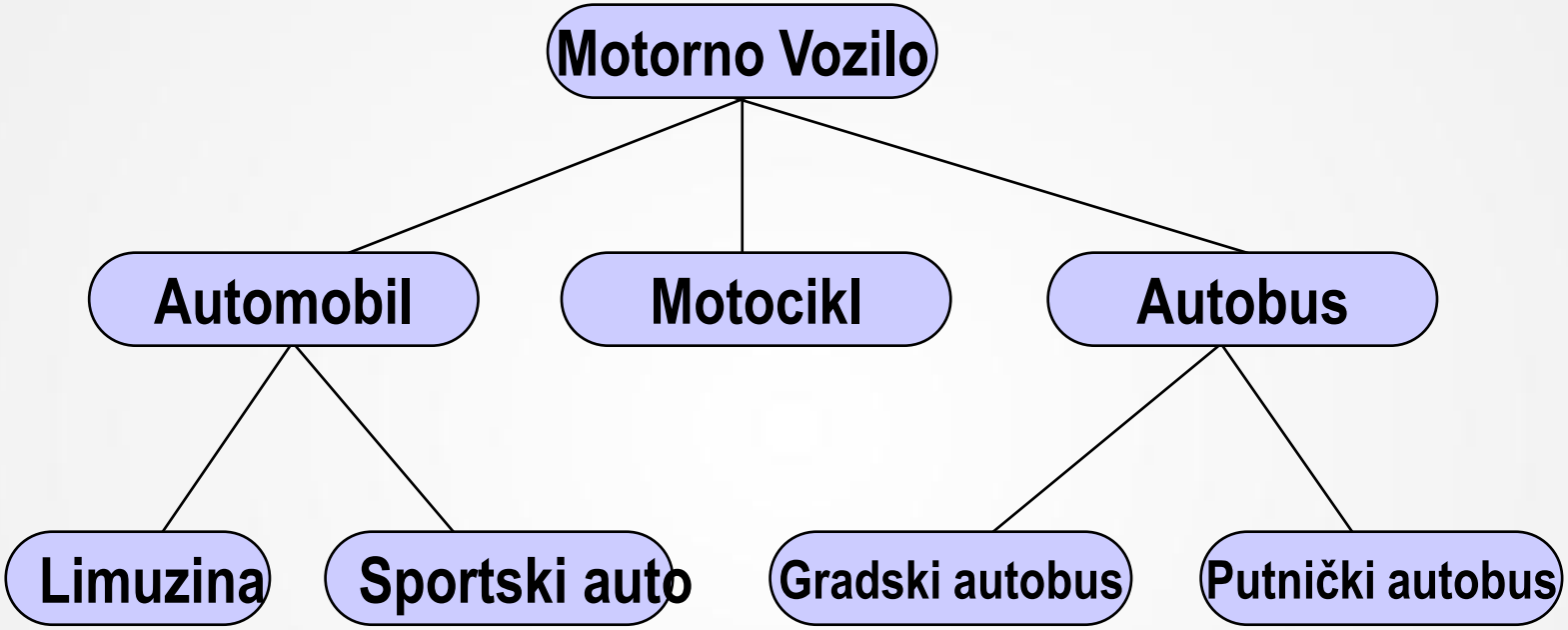
Princip nasleđivanja

- Vrlo često novi programi nastaju proširivanjem prethodnih. Najbolji način za stvaranje novog softvera je imitacija, doterivanje i proširivanje postojećeg.
- Zamislite slučaj kada imate izvorni kod neke klase. Postavite sebi pitanje, „Kako bi vi mogli da taj kod ponovo iskoristite?“ Mogli biste da ga iskopirate i u kopiji menjate ono što je potrebno.
- Kod 1 klase kopiramo na 10 mesta
- Postavite sebi pitanje "Koliki bi problem nastao ukoliko kod originalne klase sadrži neku grešku? Da li moramo tu grešku ispraviti u svim novim klasama?" Bez pažljivog planiranja završili biste sa reorganizovanom gomilom koda, prepunom bagova.

Princip nasleđivanja

- Relacija nasleđivanja omogućuje proširenje ponašanja postojeće klase.
- Generalizacija – Entiteti sa zajedničkim osobinama se grupišu tako da se njihove zajedničke osobine definišu samo jednom u osnovnoj klasi koja predstavlja njihovu generalizaciju.
- Specijalizacija – Sve ostale osobine entiteta koji su karakteristične za svaki posmatrani entitet se definišu u zasebnim klasama koje nasleđuju osnovnu klasu, te nove klase predstavljaju specijalizaciju entiteta osnovnih klasa.
- Nasleđivanje se može tumačiti kao “je vrsta” veza

Princip nasleđivanja – hijerarhija

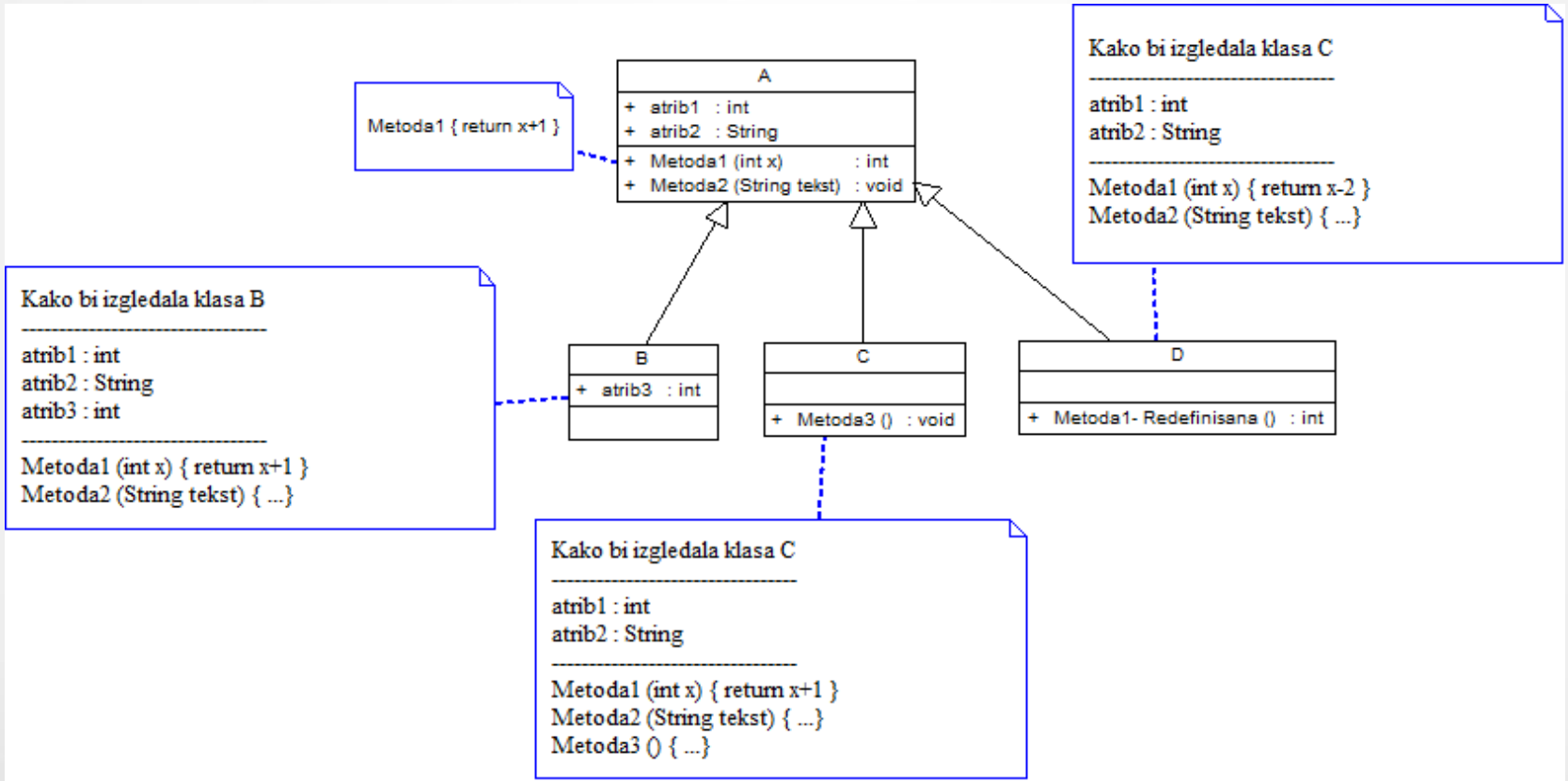


Princip nasleđivanja

- Klasa koja nasleđuje drugu klasu (izvedena klasa) preuzima sve atribute i metode klase koju nasleđuje (osnovna klasa), efekat je sličan kao kad bi mi ručno prekopirali kod osnovne klase u izvedenoj klasi – ali nismo.
- Npr. klase B,C,D nasleđuju klasu A. Izvedene klase B,C,D (potomak, podklasa – *subclass*, dete klasa – *child class,*) predstavljaju jednu specijalnu vrstu osnovne klase A (predak, nadklasa – *superclass*, roditaljska klasa – *parent class*), gde klase B,C,D nasleđuje sve atribute i sve metode od klase A.

Princip nasleđivanja

- Nova izvedena B,C ili D klasa može da:
  - proširi strukturu podataka osnovne klase A dodavanjem novih atributa (klasa B)
  - proširi funkcionalnost osnovne klase A dodavanjem novih metoda (klasa C)
  - izmeni funkcionalnost osnovne klase A redefinisanjem postojećih metoda (klasa D)

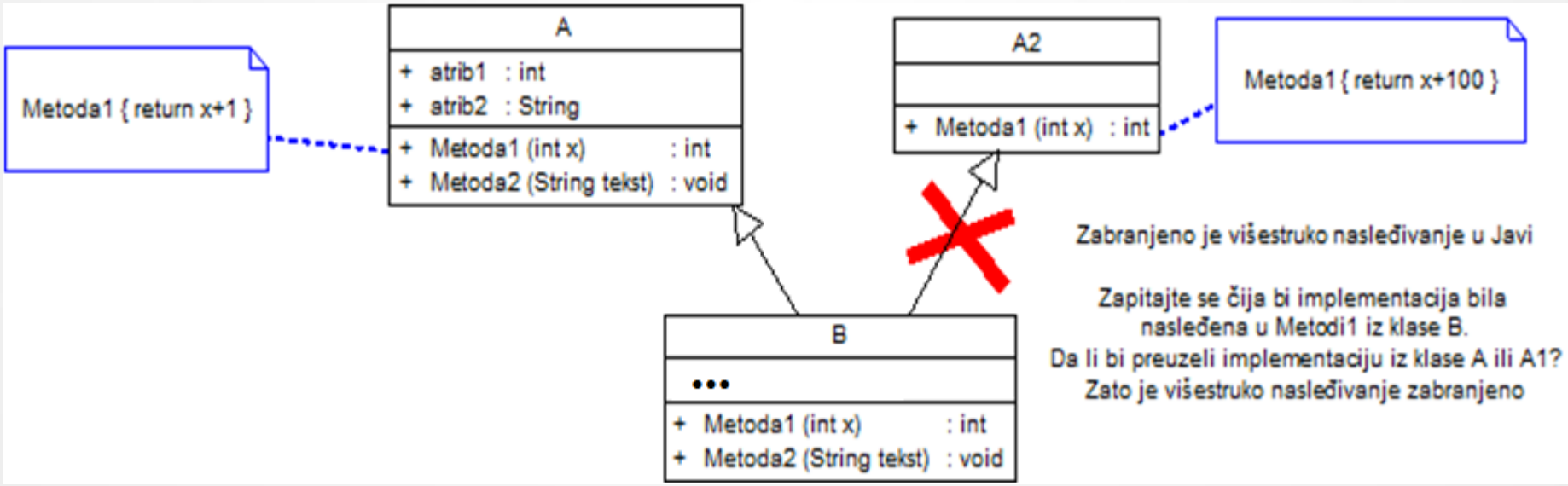


Princip nasleđivanja

- Primer nasleđivanja za osobu na fakultetu:
  - student je osoba koja studira,
  - profesor je osoba koja predaje na fakultetu.
- Imamo osnovnu klasu **osoba** (JMBG, ime i prezime, grad) i specijalizacije bi bile **student** (osoba koja ima indeks i ocene) i **profesor** (osoba koja radi na fakultetu i ima zvanje, platu, radno mesto, predmete koje drži).

Princip nasleđivanja

- Postoji samo jednostruko nasleđivanje
- Jedna klasa može samo jednu naslediti, ali više klasa može nasleđivati istu klasu
- Ako ništa ne napišemo klasa nasleđuje Object klasu
- Ključna reč **extends**

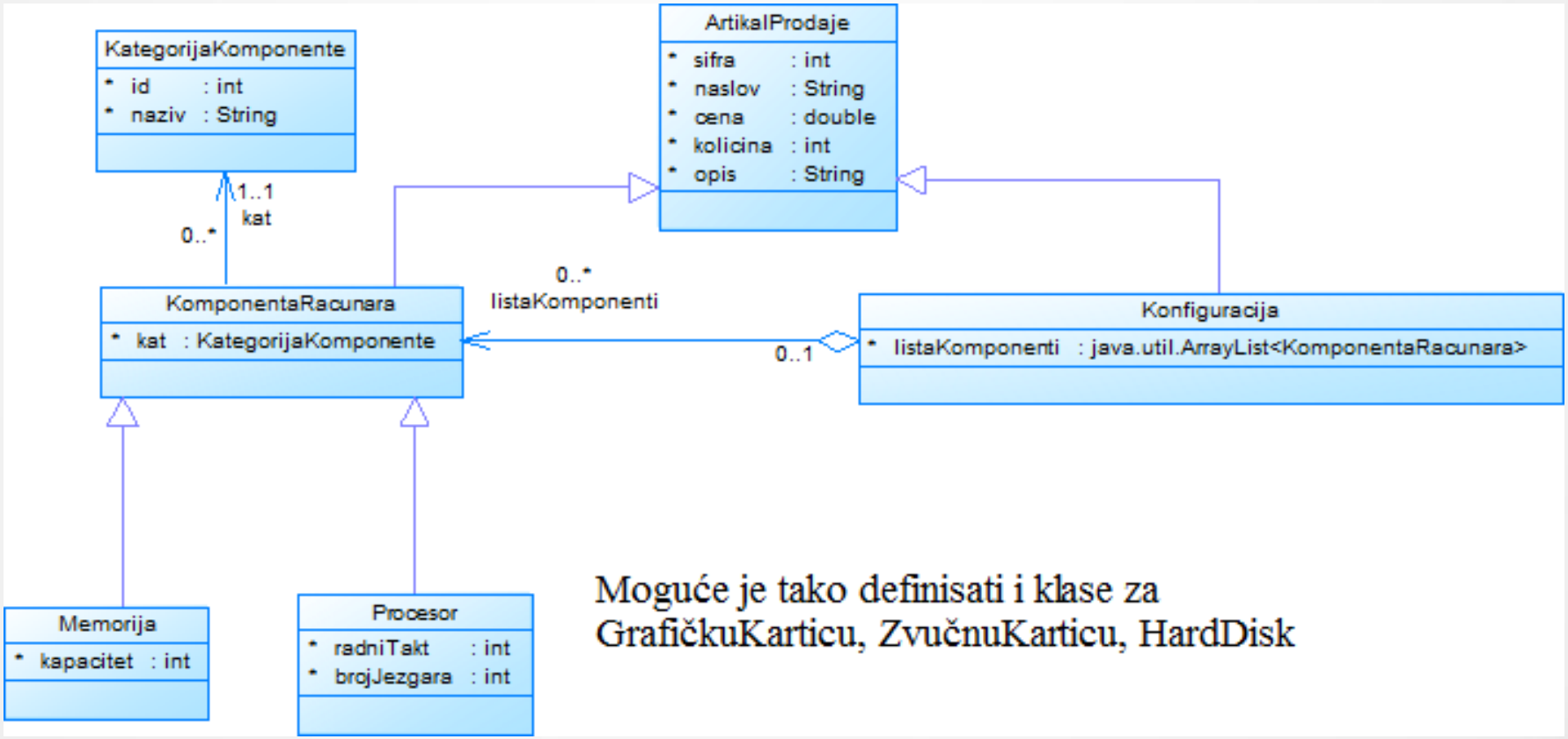


Princip nasleđivanja

- Primer nasleđivanja za artikle prodaje u prodavnici računara:
  - Artikal Prodaje koji predstavlja generalizaciju za sve proizvode koji se mogu prodavati u prodavnici računara. Opisan je šifrom, nazivom (naslovom), cenom, raspoloživom količinom i opisom.
  - Komponenta računara je artikal koji se prodaje i dodatno je opisana *kategorijom komponente.*
  - Gotova konfiguracija je artikl koji se prodaje i dodatno je opisana *listom komponenti računara* koji ulaze u konfiguraciju
  - Memorija računara je jedna od specijalizacija komponenti računara i dodatno je opisana *kapacitetom* memorije
  - Procesor računara je jedna od specijalizacija komponenti računara i dodatno je opisan *radnim taktom* i *brojem jezgara*

Princip nasleđivanja

- Primer nasleđivanja za artikle prodaje u prodavnici računara – Dijagram klasa:



primerNasladjivanjaProdavnicaRacunara

Princip nasleđivanja

- Ako napišemo ključna reč **final** ispred naziva klase nasleđivanje je zabranjeno
- Nasleđivanje je zavisno od modifikatora pristupa definisanih za metode i atribute klase pretka. Oni su:
  - vidljivi unutar metoda klasa naslednica i mogu se pozivati nad objektima klasa naslednica – public, protected
  - nisu vidljivi unutar metoda klasa naslednica i ne mogu se pozivati nad objektima klasa naslednica – private, unspecified

Princip nasleđivanja – redefinisanje metoda

- Method **overriding** – Redefinisanje metoda je pojava da u klasi naslednici postoji metoda istog imena i parametara kao i u baznoj klasi
- Cilj je definisati/izmeniti/proširiti funkcionalnost metode roditeljske klase
- Redefinisane metode mogu se anotirati u kodu Anotacija **@Override**
- Primer:
  - klasa A ima metode **metoda1**() i **metoda2**()
  - klasa B nasleđuje klasu A i takođe ima metode **metoda1**() i **metoda2**(), ali samo **metoda1**() je **redefinisana**

Princip nasleđivanja – redefinisanje metoda

```
class A {
      int metoda1() {
        System.out.println("metoda1 klase A");
      }
      int metoda2() {
        System.out.println("metoda2 klase A");
      }
}
class B extends A {
      @Override
      int metoda1() {
        System.out.println("metoda1 klase B");
      }
}
...
A varA = new A();
B varB = new B();
varA.metoda1();
varB.metoda1();
varA.metoda2();
varB.metoda2();
```

Princip nasleđivanja – redefinisanje metoda

- Konzola:

```
metoda1 klase A
metoda1 klase B
metoda2 klase A
metoda2 klase A
```

primer 01 – bez reči super

Princip nasleđivanja – reč super

- Ključna reč **super** označava roditeljsku klasu. Ona se može koristiti i u metodama i u konstruktorima.
- Ključna reč super u konstruktoru označava da pozivamo konstruktor roditeljske klase. Prva linija u konstruktoru klase naslednice **mora** biti poziv konstruktora roditeljske klase
- Korišćenjem reči super možemo pristupiti metodama roditeljske klase koje su redefinisane

primer 01 – sa super

Apstraktne klase

- Osnovna klasa koja nema nijedan konkretan (realan) objekat, već samo predstavlja generalizaciju izvedenih klasa, naziva se apstraktnom klasom.

- Apstraktna klasa može da sadrži apstraktne funkcije, koje su u ovoj klasi samo deklarisane, a nije implementirane

- Klase koje ne mogu imati svoje objekte, već samo njene klase naslednice mogu da imaju objekte (ako i one nisu apstraktne)

Apstraktne klase

- Ako klasa ima makar jednu apstraktnu metodu, mora da se deklariše kao apstraktna.
- Apstraktna klasa ne mora da ima apstraktne metode!

```
abstract class A {
      int i;
      public void metoda1() { ... }
      public abstract void metoda2();
      ...
    }
class B extends A {
  @Override
      public void metoda2() { ... }
}
```

primer 02

zadatak01 – svi zajedno radimo

Polimorfizam

- Opisuje koncept u kome se određena akcija može izvršiti na više načina. Polymorphism je nastao kombinacijom grčkih reči *poly* (više) i *morphs* (izgled/forma)
- Može biti:
  - *compile time polymorphism* (implementira sa *method overloading*)
  - *runtime polymorphism* (implementira sa *method overriding*)
- Naglasak na *Runtime polymorphism* koji se još zove *Dynamic Method Dispatch.*
- Situacija kada se poziva metoda nekog objekta, a ne zna se unapred kakav je to konkretan objekat
  - ono što se zna je koja mu je bazna klasa
- Tada je moguće u programu pozivati metode bazne klase, a da se zapravo pozivaju metode konkretne klase koja nasleđuje baznu klasu

Polimorfizam

- Prednost korišćenja polimorfizma ogleda se u toma da nam on omogućava kreiranje uniformnog pristupa/kontrole ka različitim objektima koji imaju zajednički podskup operacija.
- Rezultat polimorfizma je kod koji je više koncizan i lakši za održavanje.

Polimorfizam

```
abstract class Vozilo {
      abstract void vozi();
    }
class Automobil extends Vozilo {
   @Override
      void vozi() { ... }
}
class Kamion extends Vozilo {
   @Override
      void vozi() { ... }
}
class Vozac {
      void vozi(Vozilo v) {
        v.vozi();
      }
}
    ...
Vozac v = new Vozac();
v.vozi(new Automobil());
```

primer 03

Interfejsi

- Omogućavaju definisanje samo apstraktnih metoda, konstanti i statičkih atributa
- Ključna reč **implements**
- **Interfejs nije klasa**! On je spisak metoda i atributa koje klasa koja implementira interfejs mora da poseduje.
- Interfejsi se ne nasleđuju, već implementiraju
- Da bi klasa implementirala interfejs, mora da redefiniše sve njegove metode
- Jedan interfejs može da nasledi drugog
- Sve metode su implicitno public, a svi atributi su implicitno public static final

Interfejsi

- Jedna klasa može da implementira jedan … ili više interfejsa

```
interface USB {
  void init();
  byte[] getData();
}
interface Camera {
  void init();
  Picture getPicture();
}
class WebCam implements USB, Camera {
  @Override
  void init() { ... }
  @Override
  byte[] getData() { ... }
  @Override
  Picture getPicture() { ... }
}
```

primer 04

Kolekcije

- Nizovi imaju jednu manu – kada se jednom naprave nije moguće promeniti veličinu.
- Kolekcije rešavaju taj problem.
- Zajedničke metode:
  - dodavanje elemenata,
  - uklanjanje elemenata,
  - iteriranje kroz kolekciju elemenata

Kolekcije

Implementacija Koncept	Hash table	Resizable Array	Balanced Tree	Linked List	Hash table + Linked list
Set	HashSet		TreeSet		LinkedHashSet
List		ArrayList		LinkedList	
Map	HashMap		TreeMap		LinkedHashMap

Klasa ArrayList

- Predstavlja kolekciju, odn. dinamički niz
- U listu se dodaju Java objekti
- Elementi se u ArrayList dodaju metodom add()
- Elementi se iz ArrayList uklanjaju metodom remove()
- Elementi se iz ArrayList dobijaju (ne uklanjaju se, već se samo čitaju) metodom get()

Netipizirane kolekcije

- U netipiziranoj kolekciji možemo ubaciti objekte razlicitih tipova

```
ArrayList lista = new ArrayList();
lista.add(5);
lista.add(new Student(2,"Strahinja","Baki","Novi Sad","E1 02/2012"));
lista.add(new RacunUBanci(2, 0.0));
```

- Prilikom rada sa netipiziranom kolekcijom može potencionalno doći do greške prilikom izvršavanja Java koda

```
Integer i = (Integer)lista.get(2); // Run time error
```

Tipizirane kolekcije – Generics princip

- Tipizirane kolekcije omogućavaju smeštaj samo jednog tipa podatka u kolekciju.
- Tipizirane kolekcije se tumače kao „kolekcija objekata određenog tipa“

```
ArrayList<SProizvoljniTip> lista = new
    ArrayList<ProizvoljniTip>();
```

- Java kolekcije podržavaju generics princip putem tipiziracije njenih elemenata.
  - Generics omogućuje da se za kolekciju definiše tip objekata koji će se u kolekciji smeštiti u trenutku kada se instancira kolekcija.

For each kroz kolekcije

- Sintaksa

```
for (TipRlementa el : kolekcija) {
  System.out.println(el.toSting);
}
```

- Primer

```
for (Student el : listaStudenata) {
  System.out.println("Student sa id " + el.id + " čije
   je ime i prezime " + el.ime + " " + el.prezime + "
   ima indeks " + el.indeks + " i zivi u gradu " +
   el.grad);
}
```

primer 05

zadatak02

Dodatni Materijal

Generics in Java

- Generics je sastavni deo generičkog programiranja koji je u Javi dodato sa verzijom 1.5. u 2004. godini.
- Generics omogućuju klasi i njenim metodama da rade sa različitim tipovima podataka, da atributi klase budu različitih tipova, a korisniku je ostavljena sloboda da definiše tip koji će se koristiti.
- Generics omogućuje verifikacija objekta tako da on mora pripasti navedenom tipu, provera se izvršava u vreme kompajliranja Java koda.
- Motivacija za uključivanje generics principa u Javu se vidi iz sledećeg koda.

```
ArrayList lista = new ArrayList();
lista.add("String");
Integer i = (Integer)lista.get(0); // Run time error
```

- Iako se java kod kompajlira bez greške, prilikom izvršavanja 3 linije koda doći će do greške *java.lang.ClassCastException*

Generics in Java

- Problem se može izbeći ako se koristi generics princip.
- Kod prepravljen tako da podrži generics princip.

```
ArrayList<String> lista = new ArrayList<String>();
lista.add("String");
Integer i = lista.get(0); // (type error)
//compilation-time error
```

- Kod se neće moći kompajlirati bez greške, jer će se već kod treće 3 linije koda videti da preuzima String, a ne Integer što se ovečuje.
- Generics omogućuje da se iz liste preuzima tačno određeni tip podataka, eliminišući potrebu za castovanjem elemenata liste.
- Prethodno navedeno je primarna motivacija za korišćenje generics principa u Javi.

Generics in Java

- Primer generičke klase bi bio

```
public class GenerickaKlasa<TipVrednosti> {

    private String tekst; //može sadržati i druge atribute
    private TipVrednosti vred;

    public GenerickaKlasa(String tekst, TipVrednosti vred) {
        this.tekst = tekst;
        this.vred = vred;
    }
    ...
    public TipVrednosti getVred() {
        return vred;
    }
    public void setVred() {
        return vred;
    }
    public String toString() {
        return tekst + ", " + vred;
    }
}
```

Generics in Java

- Primer kreiranja objekta generičke klase bi bio

```
GenerickaKlasa<RacunUBanci> obj = new
    GenerickaKlasa<RacunUBanci>("Petar Petrovic", new
    RacunUBanci(1, 999.99));

System.out.println(obj.toString());

//iako je u deklaraciji metode getVred kao povratni tip
//natnačen tip TipVrednosti
//metoda getVred vratiće objekat tipa RacunUBanci
RacunUBanci rac = obj.getVred();
```

Statičko povezivanje (*static/early binding*) i dinamičko (*dynamic/late binding*) povezivanje

- Povezivanje poziva funkcije sa odgovarajućim telom metode (implementacijom) se naziva *binding.*
- Direktno povezano sa polimorfizmom za vreme kompajliranja i polimorfizmom za vreme izvršavanja.
- Povezivanje se odnosi na određivanje konkretnih vrednosti promenljivih i određivanje konkretnih implementacija metoda.

Statičko povezivanje (*static/early binding*) i dinamičko (*dynamic/late binding*) povezivanje

- Static binding
  - kada se tip objekta određen za vreme kompajliranja
  - kada je poziv metode određen za vreme kompajliranje (konkretna implementacija metoda zna za vreme kompajliranja).
    - Koristi informacije o tipu objekta
    - Sve metode deklarisane kao static, private ili final se sigurno određene za vreme kompajliranje (ne mogu da se redefinišu).

Statičko povezivanje (*static/early binding*) i dinamičko (*dynamic/late binding*) povezivanje

- Static binding

```
public class Test {
    public class Vozilo {
        public void pokreni(){System.out.println("Vozilo je upaljeno");}
    }
    public class Automobil extends Vozilo {
        @Override
        public void pokreni(){System.out.println("Automobil je upaljen");}
    }
    void ispisNesto(int a){System.out.println("Ispis parametar tipa int");}
    void ispisNesto(double d){System.out.println("Ispis parametar tipa double");}
    void ispisNesto(Vozilo v){System.out.println("Ispis parametar Vozilo");}
    void ispisNesto(Automobil a){System.out.println("Ispis parametar Automobil");}

    public void staticBindingTest() {
        System.out.println("Poziv konketne metode zavisi od tipa parametra");
        ispisNesto(4); //Ispis parametar tipa int
        ispisNesto(5.0); //Ispis parametar tipa double
        ispisNesto(new Vozilo()); //Ispis parametar tipa Vozilo
        ispisNesto(new Automobil()); //Ispis parametar tipa Automobil
        Vozilo v = new Automobil();
        ispisNesto(v); //Ispis parametar tipa Vozilo
    }
    public static void main(String[] args) {
        Test t = new Test();
        t.staticBindingTest();
    }
}
```

Statičko povezivanje (*static/early binding*) i dinamičko (*dynamic/late binding*) povezivanje

- Dynamic binding
  - kada se tip objekta određuje za vreme izvršavanja
  - kada je poziv metode određuje za vreme izvršavanja (konkretna implementacija metoda nije poznata za vreme kompajliranja i određuje se za vreme izvršavanja).
    - koristi konkretne objekte

```
public class Test {
    public class Vozilo {
        public void pokreni(){System.out.println("Vozilo je upaljeno");}
    }
    public class Automobil extends Vozilo {
        @Override
        public void pokreni(){System.out.println("Automobil je upaljen");}
    }
    public void dynamicBindingTest() {
        System.out.println("Poziv konketne metode zavisi od objekta koji poziva metodu");
        Vozilo v = new Automobil();
        v.pokreni(); //Automobil je upaljen
    }
    public static void main(String[] args) {
        Test t = new Test();
        t.dynamicBindingTest();
    }
}
```