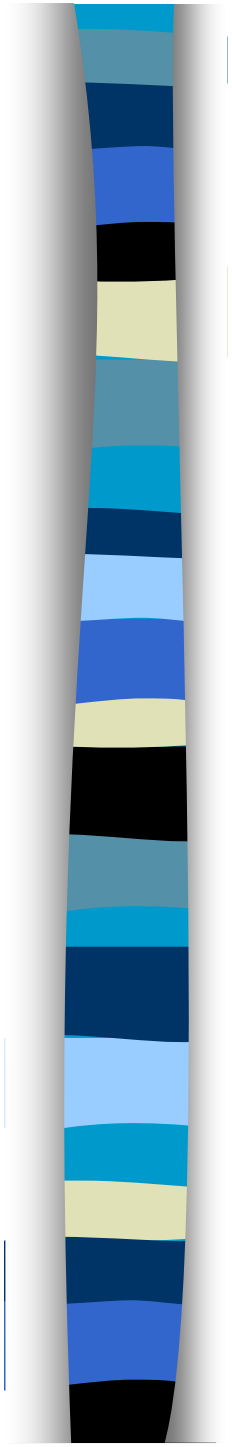


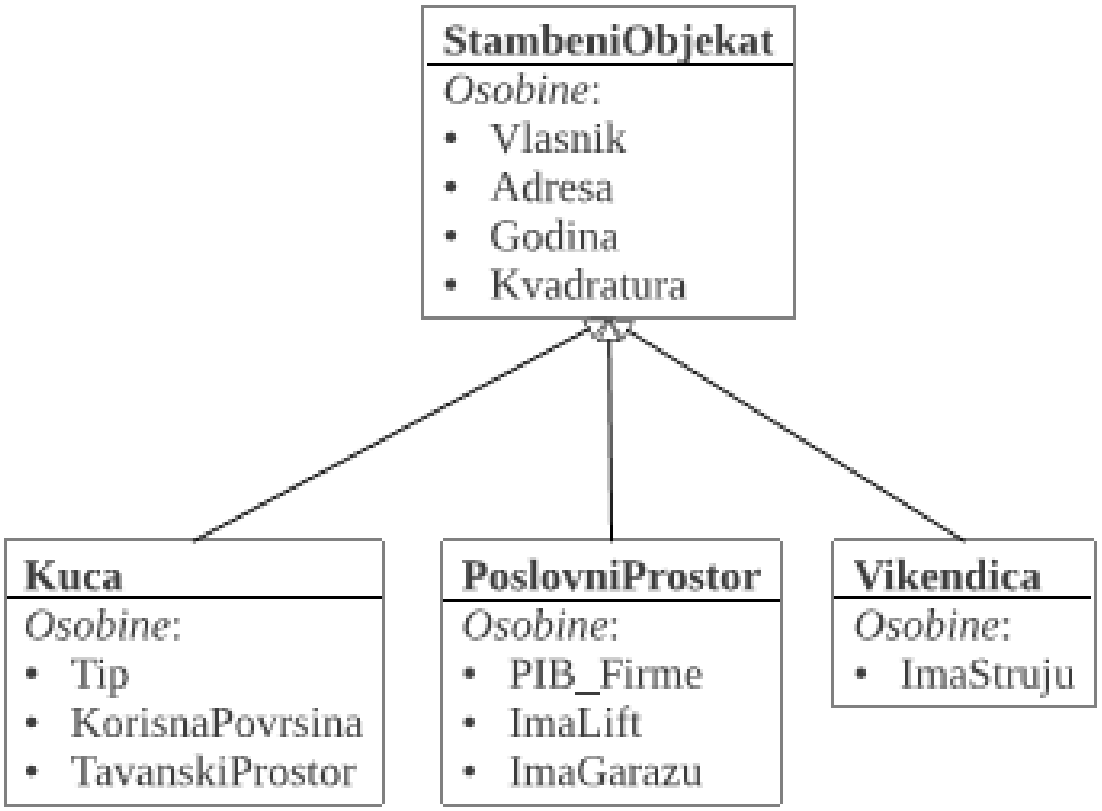
# Nasleđivanje

# Nasleđivanje

**Jednostruko nasleđivanje – single inheritance**

- Svaka klasa može da ima proizvoljan broj podklasa
- **Svaka klasa može imati jednu direktnu nadklasu.**
- Nasleđivanje dozvoljava definisanje nove klase na osnovu postojeće klase.
- Klasa koja se nasleđuje naziva se **osnovna klasa, klasa roditelj, ili nadklasa.**
- Klasa koja nasleđuje neku drugu klasu se naziva **klasa dete ili podklasa.**
- Nasleđivanje pojednostavljuje proces modeliranja hijerarhije objekata iz realnog okruženja
  - Opšta svojstva funkcionalnosti se implementiraju u osnovnoj klasi i time se omogućava bolje ponovno korišćenje koda
  - Podklase onda proširuju, specijalizuju ili menjaju (eng. *Override*) funkcionalnosti osnovne klase.
- Svaka podklasa dodaje specifično stanje i/ili ponašanje na postojeće stanje i ponašanje nasleđene klase – **definisanje pod-tipa**

# Nasleđivanje




Slika 2.4: Hijerarhija klasa stambenih objekata.

# Nasleđivanje

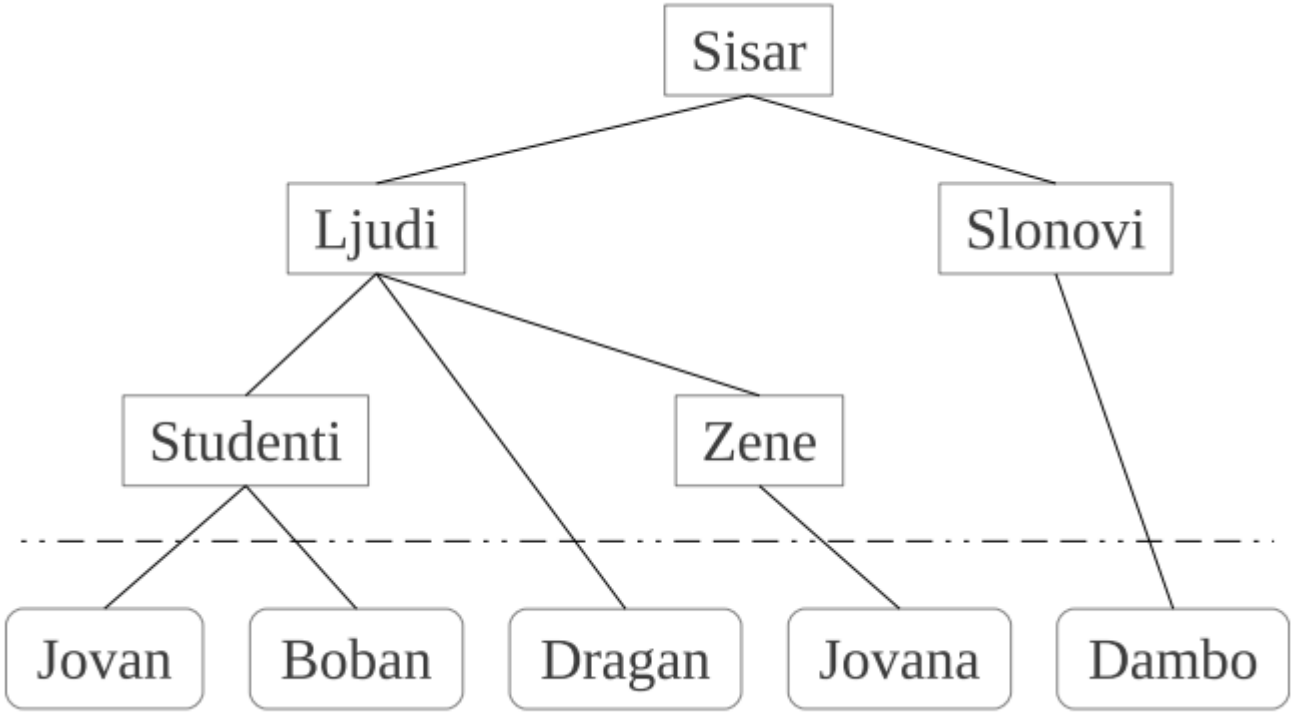
- Sintaksa u Javi za kreiranje podklase je jednostavna: na početku deklaracije klase koristi se ključna reč ***extends*** iz koje se navodi ime klase koja se nasleđuje.

Primer 2.3: Klasa Kuca nasleđuje klasu StambeniObjekat

```
class Kuca extends StambeniObjekat {
  // dodaju se nova polja i metodi koji su specificni za kucu
}
```

- Nova klasa Kuca poseduje  sva polja i metode koji su specificirani u klase i StambeniObjekat ali
- dozvoljava i mogućnost promene postojećih metoda kao i dodavanje niza novih metoda kako bi se bolje fokusirali na specifična svojstva nove klase

# Primer jednostrukog nasleđivanja




- Klasa Sisari ima klase Ljudi i Slonovi kao svoje direktne podklase.
- Klasa Ljudi ima klasu Sisar kao direktnu nadklasu, a klase Studenti i Zene kao svoje direktne podklase.
- Objekti Jovan, Boban, Dragan, Jovana, i Dambo predstavljaju instance tačno određenih klasa

# Nasleđivanje

- Nasleđivanje uspostavlja između klasa relaciju JE (eng. ***is-a***).
- Podklasa ima ista svojstva i funkcionalnosti kao nasleđena klasa, uz neka dodatna svojstva i funkcionalnosti.
- Na primer:
  - Klasa Automobil JE PrevoznoSredstvo,
  - Klasa Jabuka JE Voce.
- Kompozicije i agregacije omogućuju konstruisanje kompleksnih klasa koje inkorporiraju druge objekte. Oni uspostavljaju relaciju IMA (eng. ***has***) između klasa.
- Na primer:
  - Automobil IMA Masinu,
  - Kupac IMA KreditnuKarticu.

# Kompozicija vs. agregacija

- U **komponovanom** objektu, objekti komponente od kojih je napravljena kompozicija postoje i koriste se jedino preko komponovanog objekta.
- Ako se komponovani objekat uništi, onda se uništavaju i objekti komponente.
- Na primer, objekat Zaposleni IMA ime koje je realizovano kao String objekat. Nema potrebe da taj objekat ostane ukoliko se uništava objekat Zaposleni.

# Kompozicija vs. agregacija

- U **agregaciji**, objekti od kojih je napravljena agregacija mogu (ali nije neophodno) da egzistiraju nezavisno od njihove upotrebe u agregiranom objektu.

- U zavisnosti od strukture agregacije, uništavanje agregacije može, ali i ne mora da znači i uništavanje njenih komponenti (agregiranih objekata).

- Na primer, objekat Korisnik IMA u okviru sebe objekat BankovniRacun. Međutim, BankovniRacun može da predstavlja zajednički račun nekoliko korisnika (muža i žene, roditelja i deteta). U tom slučaju, nije dobro brisati objekat BankovniRacun samo zato što je jedan objekat tipa Korisnik izbrisan iz sistema.

# Nasleđivanje

- Pomoću nasleđivanja, postojeće klase se mogu koristiti za pravljenje novih klasa
- Nova klasa koja nasleđuje neku postojeću klasu zadržava sve članove postojeće klase osim privatnih članova
- Novu klasu zovemo ***direktna podklasa*** postojeće klase, a postojeću klasu zovemo ***direktna nadklasa*** nove klase
- Klasa može da dobije i nove članove, koje njena direktna nadklasa ne sadrži, a postojeći metodi diretne nadklase mogu biti promenjeni (redefinisani - overriden) u novoj klasi
- Pored termina direktna podklasa i direktna nadklasa, uvešćemo i termine ***podklasa*** i ***nadklasa*** (tranzitivna zatvorenja relacija direktna podklasa, odnosno direktna nadklasa)
- Pomoću ključne reči **extends** se deklariše da klasa nasleđuje neku postojeću klasu

# Nasleđivanje - klasa `Object`

- U Javi API postoji jedna klasa koja je nadklasa svim ostalim klasama. To je klasa **Object**.
- Sve ostale klase u Javi, i one koje već postoje i one koje mi pravimo, nasleđuju klasu **Object**.
- Ako se prilikom deklaracije neke nove klase ne navede ključna reč extends i ime direktne nadklase, tada se podrazumeva da je direktna nadklasa te nove klase klasa **Object**.
- Klasa koja ima direktnu nadklasu koja nije klasa **Object** takođe nasleđuje klasu **Object**, zato što njena nadklasa nasleđuje klasu **Object**.

# Nasleđivanje - klasa `Object`

- Klasa **Object** nema privatnih članova, tako da se svi njeni članovi nasleđuju u njenim podklasama, tj. u svim ostalim klasama
- Nizovi takođe nasleđuju klasu **Object**
- Klasa **Object** obezbeđuje nekoliko korisnih metoda za sve svoje podklase.
- Ove metode mogu biti redefinisane kako bi omogućile operacije koje su specifične za pojedinačne podklase.

# Nasleđivanje - primer Bycicle

- Deklarisaćemo klasu ***Bicycle***, čija tri privatna polja opisuju stanja, odnosno osobine bicikla:
  - broj obrtaja pedala (cadence),
  - stepen prenosa menjača (gear) i
  - brzina (speed).
- Klasa *Bicycle* ima jedan javni konstruktor i četiri javna metoda koja postavljaju i manipulišu vrednostima polja

# Nasleđivanje - primer Bicycle

```
class Bicycle {

  // the Bicycle class has three fields
  private int cadence;
  private int gear;
  private int speed;

  // the Bicycle class has one constructor
  public Bicycle(int startCadence, int startGear, int startSpeed) {
    cadence = startCadence;
    gear = startGear;
    speed = startSpeed;
  }

  // the Bicycle class has four methods
  public void setCadence(int newValue) {
    cadence = newValue;
  }

  public void setGear(int newValue) {
    gear = newValue;
  }

  public void applyBrake(int decrement) {
    speed -= decrement;
  }

  public void sppedUp(int increment) {
    speed += increment;
  }
}
```

# Nasleđivanje - primer Bicycle

- Sad recimo da želimo da deklarišemo klasu ***MountainBike*** da predstavimo bicikl „planinac“ koji ima sve osobine kao i opšti bicikl predstavljen klasom Bicycle.
- Umesto da unutar klase ***MountainBike*** iznova deklarišemo sve članove kao i u klasi Bicycle, klasa ***MountainBike*** može da nasledi klasu Bicycle i time jednostavno „preuzme“ sve njene članove (koje dozvoljavaju modifikatori pristupa), i da doda svoje nove članove, npr. visinu sica (seatHeight) i odgovarajući setSeatHeight metod, kao i nov konstruktor.
- Nova klasa ***MountainBike*** imaće ukupno:
  - četiri polja (tri nasleđena i jedno novo),
  - jedan konstruktor i
  - pet metoda (četiri nasleđena i jedan nov)

# Nasleđivanje - primer Bicycle

```
class MountainBike extends Bicycle {

  // the MountainBike subclass adds one field

  public int seatHeight;

  // the MountainBike subclass has one constructor
  public MountainBike(int startCadence, int startGear, int startSpeed,
       int startSeatHeight) {
    super(startCadence, startGear, startSpeed);
    seatHeight = startSeatHeight;
  }

  // the MountainBike subclass adds one method
  public void setSeatHeight(int newValue) {
    seatHeight = newValue;
  }
}
```

# Nasleđivanje – primer Employee

- Deklarisaćemo klasu ***Employee***, osnovnu klasu koja predstavlja radnika u preduzeću.
- Pored polja name sa imenom radnika, klasa sadrži i finalno polje ssn sa jedinstvenim brojem radnika kom se jednom postavljena vrednost neće dalje menjati.
- Pored polja za e-mail adresu (email) i godinu rođenja zaposlenog (yearOfBirth), tu je i statičko polje ***baseVacationDays*** koje predstavlja osnovni broj dana godišnjeg odmora zajednički za sve zaposlene.

# Nasleđivanje – primer Employee

- Metod ***computeVacationDays()*** računa broj dana odmora za zaposlenog sabirajući osnovnu vrednost iz ***baseVacationDays*** sa vrednošću ne-statičkog polja ***extraVacationDays***.
- Pored standardnih set i get metoda, klasa sadrži još i metod ***print*** koji štampa vrednosti polja objekta, stavljajući ispred ispisa header, a iza ispisa footer prosleđen kao parametar metoda.

# Nasleđivanje – primer Employee

- Klasa ***Manager*** proširuje klasu Employee novim poljem ***responsibility*** koje opisuje zaduženje menadžera kao zaposlenog.

- Pored dodatih standardnih set i get metoda za to polje i novog konstruktora, klasa Manager redefiniše i metod ***print*** u kom se pored vrednosti polja nasleđenih iz klase Employee sad štampa i vrednost novog polja.

# Nasleđivanje - primer Employee

```
class Employee {
  private String name;
  private final String ssn;
  private String email;
  private int yearOfBirth;
  private static int baseVacationDays;
  private int extraVacationDays;

  public Employee(String name, String ssn) {
    this.name = name;
    this.ssn = ssn;
  }

  public void setName(String name) {
    if (name != null && name.length() > 0) {
      this.name = name;
    }
  }

  public String getName() {
    return this.name;
  }

  public String getSsn() {
    return this.ssn;
  }

  public void setEmail(String email) {
    this.email = email;
  }

  public String getEmail() {
    return this.email;
  }
```

# Nasleđivanje - primer Employee

```
  public void setYearOfBirth(int yearOfBirth) {
    this.yearOfBirth = yearOfBirth;
  }

  public int getYearOfBirth() {
    return this.yearOfBirth;
  }

  public static void setBaseVacationDays(int baseVacationDays) {
    Employee.baseVacationDays = baseVacationDays;
  }

  public void setExtraVacationDays(int extraVacationDays) {
    this.extraVacationDays = extraVacationDays;
  }

  public int computeVacationDays() {
    return Employee.baseVacationDays + this.extraVacationDays;
  }

  public void print(String header, String footer) {
    if (header != null) {
      System.out.println(header);
    }
    System.out.println("Name: " + this.name);
    System.out.println("SSN: " + this.ssn);
    System.out.println("Email Address: " + this.email);
    System.out.println("Year Of Birth: " + this.yearOfBirth);
    System.out.println("Vacation Days: " + this.computeVacationDays());
    if (footer != null) {
      System.out.println(footer);
    }
  }
}
```

# Nasleđivanje - primer Employee

```
class Manager extends Employee {
  private String responsibility;

  public Manager(String name, String ssn, String responsibility) {
    super(name, ssn);
    this.responsibility = responsibility;
  }

  public void setResponsibility(String responsibility) {
    this.responsibility = responsibility;
  }

  public String getResponsibility() {
    return this.responsibility;
  }

  public void print(String header, String footer) {
    super.print(header, null);
    System.out.println("Responsibility: " + responsibility);
    if (footer != null) {
      System.out.println(footer);
    }
  }
}
```

# Teorijske vežbe 2

Objektno-orjentisano programiranje 1

# Parsiranje stringa

- Parsiranje stringa – izdvajanje pojedinačnih informacija o nekom objektu iz kompozitnog string opisa objekta
  - Jedan radnik je opisan stringom koji sadrži sve informacije o radniku razdvojene nekim delimiterom (npr. ",").
  - Kako iz takvog stringa izvući pojedinačne informacije o radniku?

- Npr. "Milovan, 3030.4 , 10 , DMI"
  - ime → "Milovan"
  - plata → 3030.4
  - staz → 10
  - radnaJedinica → "DMI"

# Metode split() i trim() klase String

- **String[] split(String regex)**
  - Od stringa kreira niz stringova (tokena) razdvojenih delimiterom koji je specificiran regularnim izrazom regex

```
String s = "Ana,Voli,Milovana";
String t[] = s.split(",");
```

t[0] je "Ana"

t[1] je "Voli"

t[2] je "Milovana"

- **String trim()**
  - Vraća string bez vodećih i krajnjih praznina

```
String s = "      pera     ";
String p = s.trim(); → p je "pera"
```

```java
public static void main(String[] args) {
        // informacije o radniku (ime, plata, staz, radna jedinica)
        String info = "    Milovan,     3030.4    ,   10  ,   DMI";

        String[] tokeni = info.split(",");

        String ime = tokeni[0].trim();
        double plata = Double.parseDouble(tokeni[1].trim());
        int staz = Integer.parseInt(tokeni[2].trim());
        String rj = tokeni[3].trim();

        System.out.println("Ime =  " + ime);
        System.out.println("Plata = " + plata);
        System.out.println("Staz = " + staz);
        System.out.println("Radna jedinica = " + rj);
}
```

# Zadatak 5

- Jedan student PMFa je predstavljen klasom Student koja ima sledeće atribute
  - id (broj indeksa) koji se automatski određuje – odgovara rednom broju kreiranog objekta klase Student
  - ime
  - prosek
  - departman na kome studira
- Klasa Student definiše get metode za sve gornje atribute i toString() metod
- Student se nagrađuje ukoliko mu je prosek jednak ili veći od 9.5

# Zadatak

- Klasom Generacija je predstavljena jedna generacija studenata. Generacija se inicijalizuje podacima iz tekstualnog fajla koji je organizovan na sledeći način
  - Prva linija: broj studenata u generaciji
  - Svaka sledeća linija opisuje jednog studenta. Linija sadrži redom ime studenta, prosek i ime departmana razdvojene ","

- Klasa Generacija takođe definiše sledeće metode
  - **void nagrade(String departman)**
    ispisuje sve studente sa nekog departmana koji se nagrađuju i koliko ima nagrađenih studenata
  - **Student najboljiNaDepartmanu(String departman)**
    vraća najboljeg studenta na nekom departmanu

# Zadatak

- Napraviti generaciju studenata na osnovu proizvoljnog fajla (za koga pretpostavljamo da je pravilno formatiran) i ispisati sve nagrađene studente, kao i najboljeg studenta na DMI

# Primer fajla

15
Pera, 8.6, DMI
Petra, 9.8, DF
Mika, 9.0, DF
Mara, 9.2, DMI
Zika, 9.55, DMI
Milica, 10.0, DMI
Stojan, 10.0, DF
Darko, 9.6, DMI
Slavica, 8.2, DMI
Marko, 8.0, DMI
Marica, 9.4, DMI
Zarko, 9.7, DH
Goran, 6.7, DMI
Zorana, 9.0, DF
Zoran,7.7, DH

```java
public class Student {
	private int id;
	private String ime;
	private double prosek;
	private String departman;

	private static int idCounter = 0;
	private static final double odlicanProsek = 9.5;

	public Student(String ime, double prosek, String departman) {
		id = ++idCounter;

		this.ime = ime;
		this.prosek = prosek;
		this.departman = departman;
	}

	// to be continued
}
```

```
public class Student {
        ...

        public String toString() {
                return id + ": " + ime +
                        "(" + prosek + ") -- " +
                        departman;
        }

        public int getId()              { return id;         }
        public String getIme()          { return ime;        }
        public double getProsek()       { return prosek;     }
        public String getDepartman()    { return departman;  }

        public boolean dobijaNagradu() {
                return prosek >= odlicanProsek;
        }
}
```

```java
import java.io.*;
public class Generacija {
	private Student[] studenti;

	public Generacija(String inputFile) throws IOException {
		BufferedReader br = new BufferedReader(
			new FileReader(inputFile));

		int brojStudenata = Integer.parseInt(br.readLine());
		studenti = new Student[brojStudenata];

		for (int i = 0; i < brojStudenata; i++) {
			String linija = br.readLine();
			String[] s = linija.split(",");
			String ime = s[0].trim();
			double prosek = Double.parseDouble(s[1].trim());
			String departman = s[2].trim();

			studenti[i] = new Student(ime, prosek, departman);
		}

		br.close();
	}
	// to be continued
}
```

```java
public void nagrade(String departman) {
	int brojac = 0;

	System.out.println("Nagradjeni studenti na " + departman);

	for (int i = 0; i < studenti.length; i++) {
		Student s = studenti[i];
		if (s.getDepartman().compareTo(departman) == 0) {
			if (s.dobijaNagradu()) {
				System.out.println(s);
				brojac++;
			}
		}
	}

	System.out.println("Ukupno nagrada: " + brojac);
}
```

```java
public Student najboljiNaDepartmanu(String departman) {
	Student najbolji = null;

	for (int i = 0; i < studenti.length; i++) {
		Student s = studenti[i];
		if (s.getDepartman().compareTo(departman) == 0) {
			if (najbolji == null) {
				najbolji = s;
			} else {
				if (s.getProsek() > najbolji.getProsek()) {
					najbolji = s;
				}
			}
		}
	}

	return najbolji;
}
```

```java
import java.io.IOException;

public class Zadatak  {
	public static void main(String[] args)
		throws IOException
	{
		Generacija g = new Generacija("studenti.txt");

		g.nagrade("DMI");
		Student najbolji = g.najboljiNaDepartmanu("DMI");
		System.out.println("Najbolji student na DMI: " + najbolji);
	}
}
```

```
Nagradjeni studenti na departmanu DMI
5: Zika(9.55) -- DMI
6: Milica(10.0) -- DMI
8: Darko(9.6) -- DMI
Ukupno nagrada: 3
Najbolji student na DMI: 6: Milica(10.0) -- DMI
```

# Nasleđivanje (inheritance)

- Mogućnost definisanja novih klasa koje proširuju funkcionalnost postojećih klasa
- class B extends A {}
  - Klasa B nasleđuje atribute i metode klase A
  - Klasa B dodaje nove atribute i metode koji proširuju funkcionalnost koja je nasleđena iz A
  - Klasa B može **redefinisati** nasleđene atribute i metode klase A (method/attribute overriding)
  - Klasa A: bazna klasa, super klasa, roditeljska klasa
  - Klasa B: izvedena klasa, podklasa, klasa dete

**Ako bazna klasa definiše bar jedan konstruktor, tada u svakom konstruktoru izvedene klase prva naredba mora biti poziv nekog konstruktora bazne klase kako bi se inicijalizovali nasleđeni atributi**

```
class Pravugaonik {
	protected int duz, sir;

	public Pravugaonik(int duz, int sir) {
		this.duz = duz;
		this.sir = sir;
	}
}

class Kvadrat extends Pravugaonik {
	public Kvadrat(int duz) {
		super(duz, duz);
	}
}
```

```java
public class Pravugaonik {

	protected int duz, sir;

	public Pravugaonik(int duz, int sir) {
		this.duz = duz;
		this.sir = sir;
	}

	public int obim() {
		return 4 * (duz + sir);
	}

	public boolean podudaranSa(Pravugaonik o) {
		return duz == o.duz && sir == o.sir;
	}
}
```

```
public class Kvadrat extends Pravugaonik {
        public Kvadrat(int duz) {
                super(duz, duz);
        }

        // redefinisana metoda obim
        public int obim() {
                return 4 * duz;
        }

        // nova funkcionalnost
        public boolean jedinicni() {
                return duz == 1;
        }
}
```

# Zadatak

- Klasom Radnik se opisuje jedan radnik radne organizacije. Za svakog radnika čuvamo ime i radni staž. Plata radnika je određena godinama radnog staža – za svaku godinu dobija 10000 $

- Klasa Rukovodilac nasleđuje klasu Radnik. Svaki rukovodilac upravlja nekim brojem radnika. Rukovodilac ima dodatak na platu – za svakog podređenog dobija 1000$.

- Naći prosečnu platu rukovodilaca koji imaju više od *k* podređenih.

# Zadatak

- Podatke o radnoj organizaciji učitavamo iz tekstualnog fajla organizovanog na sledeći način
  - Prva linija – broj radnika
  - Svaka sledeća linija sadrži informacije o jednom radniku
  - Ako je običan radnik: <Ime>,<RadniStaz>
  - Ako je rukovodilac: <Ime>,<RadniStaz>,<BrPodredjenih>

# Primer ulaznog fajla

8

Milovan, 10

**Ana,12, 3**

**Dijana, 10, 5**

Pera, 8

**Tijana, 4, 8**

Petra, 9

**Aca, 8, 2**

Mika, 9

```java
public class Radnik {

	private String ime;
	private int radniStaz;
	private static final double koeficijent = 10000;

	public Radnik(String ime, int radniStaz) {
		this.ime = ime;
		this.radniStaz = radniStaz;
	}

	public double plata() {
		return radniStaz * koeficijent;
	}

	public String toString() {
		return ime + ", " + radniStaz + ", " + plata();
	}
}
```

```
public class Rukovodilac extends Radnik {

	private int brPodredjenih;
	private static final double dodatak = 1000.0;

	public Rukovodilac(String ime, int radniStaz, int brPodredjenih) {
		super(ime, radniStaz);
		this.brPodredjenih = brPodredjenih;
	}

	public double plata() {
		return super.plata() + dodatak * brPodredjenih;
	}

	public int getBrPodredjenih() {
		return brPodredjenih;
	}

	public String toString() {
		return "Rukovodilac " + super.toString();
	}
}
```

```
public class RadnaOrganizacija {
	private Radnik[] radnici;

	public RadnaOrganizacija(String imeFajla) throws IOException {
		BufferedReader br = new BufferedReader(
			new FileReader(imeFajla));

		int brRadnika = Integer.parseInt(br.readLine());
		radnici = new Radnik[brRadnika];

		for (int i = 0; i < brRadnika; i++) {
			String[] s = br.readLine().split(",");
			String ime = s[0].trim();
			int radniStaz = Integer.parseInt(s[1].trim());

			if (s.length == 3) {
				int brPodredjenih = Integer.parseInt(s[2].trim());
				radnici[i] =
					new Rukovodilac(ime, radniStaz, brPodredjenih);
			} else {
				radnici[i] = new Radnik(ime, radniStaz);
			}
		}

		br.close();
	}
	...
```

```java
public double prosecnaPlataRukovodilaca(int brPodredjenih) {
	double sum = 0.0;
	int num = 0;

	for (int i = 0; i < radnici.length; i++) {
		if (radnici[i] instanceof Rukovodilac) {
			Rukovodilac r = (Rukovodilac) radnici[i];
			if (r.getBrPodredjenih() >= brPodredjenih) {
				sum += r.plata();
				num++;
			}
		}
	}

	return num == 0 ? Double.NaN : sum / num;
}
```

```
import java.io.IOException;

public class Zadatak6 {

	public static void main(String[] args)
		throws IOException
	{
		RadnaOrganizacija ro = new RadnaOrganizacija("RadnaOrganizacija.txt");
		System.out.println("Prosecna plata rukovodilaca sa >= 5 radnika");
		System.out.println(ro.prosecnaPlataRukovodilaca(5));
	}
}
```