

OSNOVI OBJEKTNO- ORIJENTISANOG PROGRAMIRANJA

prof. dr Dušan Gajić

prof. dr Dinu Dragan



Pregled

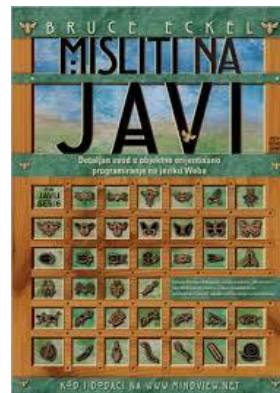
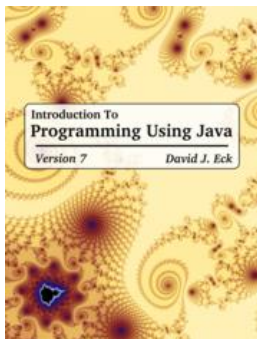
- Pojam objektno-orijentisanog programiranja (OOP), modelovanje problema u OOP - objekat i klasa
- Kontrola pristupa, enkapsulacija
- Nasleđivanje, slaganje
- Polimorfizam, apstraktne klase i interfejsi
- Tačnost, robusnost i efikasnost OO programa – obrada grešaka pomoću izuzetaka, ulaz/izlaz (rad sa tokovima i datotekama)
- Java biblioteka klasa
- UML, objektno-orijentisano projektovanje softvera

Ciljevi

- Upoznati se sa **osnovnim konceptima objektno-orijentisane paradigme u programiranju i njenim prednostima**
- Osposobiti se za **projektovanje i pisanje programa na jeziku Java korišćenjem objektno-orijentisanih koncepata**
- Upoznati se sa **osnovnim metodama objektno-orijentisanog projektovanja softvera**

Literatura

1. Brošura sa slajdovima, prateći kod + **vaše beleške**
2. David J. Eck, Introduction to Programming using Java, 7th edition, 2014. <http://math.hws.edu/javanotes/>
3. Bruce Eckel, Thinking in Java, 4th edition, Prentice Hall, 2005 – prevod 4. izdanja, “[Misliti na Javi](#)”, Mikroknjiga, Beograd.
4. Matt Weisfeld, “[Objektno orijentisani način mišljenja](#)”, CET, Beograd.



UVOD U OBJEKTNO- ORIJENTISANO PROGRAMIRANJE (OOP)

Objektno-orijentisano programiranje

- Pokušaj da **programi prirodnije modeluju način na koji ljudi razmišljaju o svetu i deluju u njemu**
- U srcu OOP umesto zadataka nalazimo **objekte**
- **Programiranje** se svodi na **projektovanje skupa objekata** koji adekvatno **modeluju problem koji se rešava**
- **Softverski objekti u programu** mogu da predstavljaju **stvarne ili apstraktne entitete** u problemskom domenu
- Na ovaj način, **proces razvoja softvera** postaje **prirodniji**, a samim tim **lakši i produktivniji**

Proceduralno i objektno-orijentisano programiranje

Proceduralno programiranje:

program = niz poziva procedura
(potprograma) koji obavljaju operacije nad promenljivama
(program = strukture podataka + algoritmi)

Objektno-orijentisano programiranje:

program = skup objekata koji međusobno interaguju šaljući poruke
(objekat obuhvata i podatke i funkcije)

Proceduralno i objektno-orijentisano programiranje

- Prelazak sa proceduralnog programiranja na OOP zahteva **promenu načina razmišljanja**
- Objektno-orijentisani pristup sadrži **koncepte višeg nivoa apstrakcije** koji su **bliži problemskom domenu**
- **Težište se prebacuje sa implementacije na interfejse i veze** između delova softvera (cilj je oslabiti veze između delova programa i time ih učiniti lakšim za kontrolu i modifikaciju)
- Umesto **algoritamske (funkcionalne) dekompozicije** koristi se pre svega **objektna dekompozicija**

Prednosti OOP

- **OOP se javio kao pokušaj odgovara putem unapređenja koncepata na uočene probleme u razvoju softvera:**
 - Zahtevi korisnika su *složeni* i stalno se povećavaju. Softverski sistemi su *složeni*. **OOP omogućuje bolju kontrolu složenosti.**
 - Kako povećati produktivnost proizvodnje softvera. Povećanjem broja programera u timu? Problemi – interakcija između delova softvera! Način povećanja produktivnosti – ponovna upotreba softvera (engl. *software reuse*). **OOP povećava produktivnost.**
 - Problemi održavanja softvera: ispravljanje grešaka, promena zahteva i dodavanje zahteva. **OOP olakšava održavanje i unapređenje softvera.**

Objekat

- **Objekat** je skup promenljivih i pridruženih metoda za manipulaciju tih promenljivih
- Objekat ima **osobine (atribute, polja)** i **ponašanja (metode, funkcije)**, reaguje na **dogadjaje**
- Objekti međusobno **interaguju šaljući poruke**
- Objekti su usko povezani sa klasama – **klase se koriste kako bi se kreirali objekti**

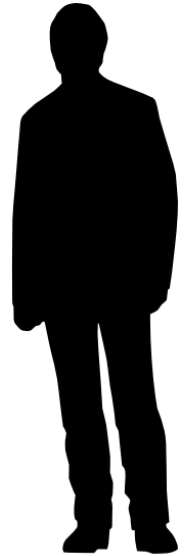
Objekat

- **Atribut** – osobina objekta; u kodu se obično može identifikovati kao opisna reč – *ime, starost, omogućeno, zabranjeno, bojaPozadine...*
- **Metod** – nešto što objekat može uraditi, u kodu se obično može identifikovati na osnovu glagola – *prikaži, postavi...*
- **Događaj** – spoljašnji faktor u odnosu na koje objekat može reagovati

Objekat - primeri

Realan objekat:

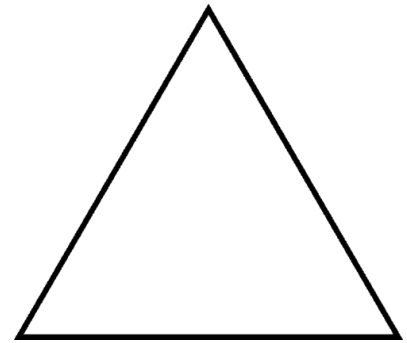
- Osoba
 - Atributi: ime, prezime, visina, težina, pol, starost, boja kose...
 - Metode: predstaviSe(), ofarbajKosu()...
- Sijalica
 - Atributi: svetli (da/ne, true/false), snaga (npr. 40 W), tip (obicna, LED...)
 - Metode: ukljuciSvetlo(), iskljuciSvetlo(), proverisvetlo(),...



Objekat - primeri

Apstraktni objekti:

- Oblik
 - Atributi: broj strana, veličina uglova, prečnik, tip (otvoren/zatvoren)...
 - Metode: racunajPovrsinu(), racunajObim(), iscrtaj()
- Fajl
 - Atributi: ime, tip, veličina, lokacija...
 - Metode: snimiFajl(), učitajFajl(), obrišiFajl()...



Definicija OO jezika – Alan Kay

- **Sve je objekat.**
- **Program je skup objekata** koji jedni drugima porukama saopštavaju šta da rade.
- **Svaki objekat ima memorijski prostor** koji se sastoji od drugih objekata.
- **Svaki objekat ima tip (klasu).**
- **Svi objekti određenog tipa** mogu da **primaju iste poruke.**

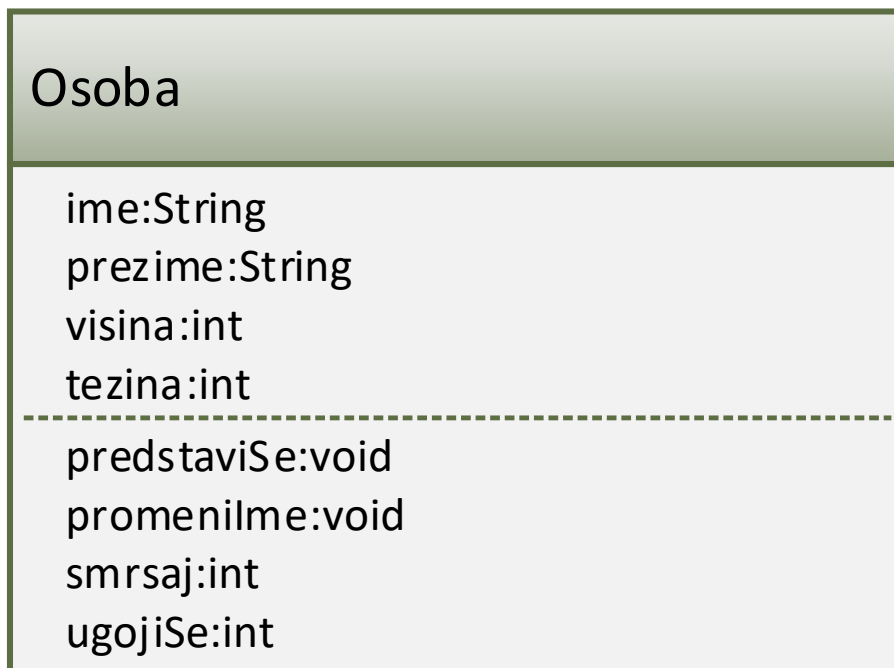
Klasa

- Klasa je osnovna organizaciona jedinica programa u objektno-orijentisanim programskim jezicima
- Klasa predstavlja strukturu u koju su grupisani **podaci (atributi)** i **funkcije (metode)**
- Klasa je **apstraktni tip podataka** koji omogućava **pravljenje objekata pružajući njihovu definiciju** kroz opis njihovih atributa (podataka, svojstava) i metoda (funkcija, operacija, ponašanja).

Klasa

- **Objekat je instanca klase. Sa jednom klasom možete napraviti koliko god objekata je neophodno.**
- **Ovo je analogno promenljivoj i njenom tipu – klasa je tip promenljive, a objekat je promenljiva.**
- Klase se često grafički predstavljaju pomoću UML (engl. Unified Modeling Language) **dijagrama klasa** (engl. class diagrams)
- Dijagram klasa sadrži ime klase, kao i nazive i tipove njenih atributa i metoda

Dijagram klasa za klasu Osoba



Klasa

- Dvostruka uloga klasa:
 1. **šablon za pravljenje objekata**
 2. **kontejner za statičke promenljive i metode**
- Atributi i metodi klase – ključna reč `static`
- Primeri korišćenja klase kao kontejnera:
 - Atribut klase: `System.in`
 - Metoda klase: `System.in.println()`
 - Poziv statičkog metoda `abs()` klase `Math`: `b = Math.abs(a);`
- **Obično se jedna klasa koristi za samo jednu od dve moguće uloge!**

Primer 1.1

- Prvi projekat u Javi koji smo pravili na osnovama programiranja (ispis poruke “Zdravo svete!”), napisan u stilu objektno-orijentisanog programiranja
- Potrebno je kreirati klasu Poruka sa atributom tekst i metodama postaviTekst i pribaviTekst
- Potom testiramo našu klasu kreiranjem objekta klase (tipa) Poruka u okviru glavnog programa, praćenog postavljanjem i prikazivanjem njenog sadržaja

Primer 1.1 – klasa Poruka

```
public class Poruka {  
    String tekst;  
  
    void postaviTekst(String noviTekst){  
        tekst = noviTekst;  
    }  
  
    void pribaviTekst(){  
        System.out.println(tekst);  
    }  
}
```

Primer 1.1 – klasa Main

```
public class Main {  
  
    public static void main(String[] args){  
  
        Poruka prvaPoruka = new Poruka();  
        prvaPoruka.tekst = "Zdravo svete!";  
        prvaPoruka.postaviTekst("Zdravo OOP svete!");  
        prvaPoruka.pribaviTekst();  
    }  
}
```

Promenljive i objekti

- Deklarisanjem promenljive ne kreira se objekat!

Poruka p;

**Nijedna promenljiva u Javi nikada
ne može sadržati objekat.
Promenljiva samo čuva referencu na objekat!**

- Objekti se čuvaju u posebnom delu memorije koji se zove gomila (engl. heap)

Promenljive i objekti

- Promenljiva samo čuva informaciju koja je neophodna da bi se objekat pronašao u memoriji. Ova informacija se naziva **referenca** ili **pokazivač na objekat**.
- Objekti se kreiraju pomoću operatora **new**, koji stvara objekat i vraća referencu na taj objekat

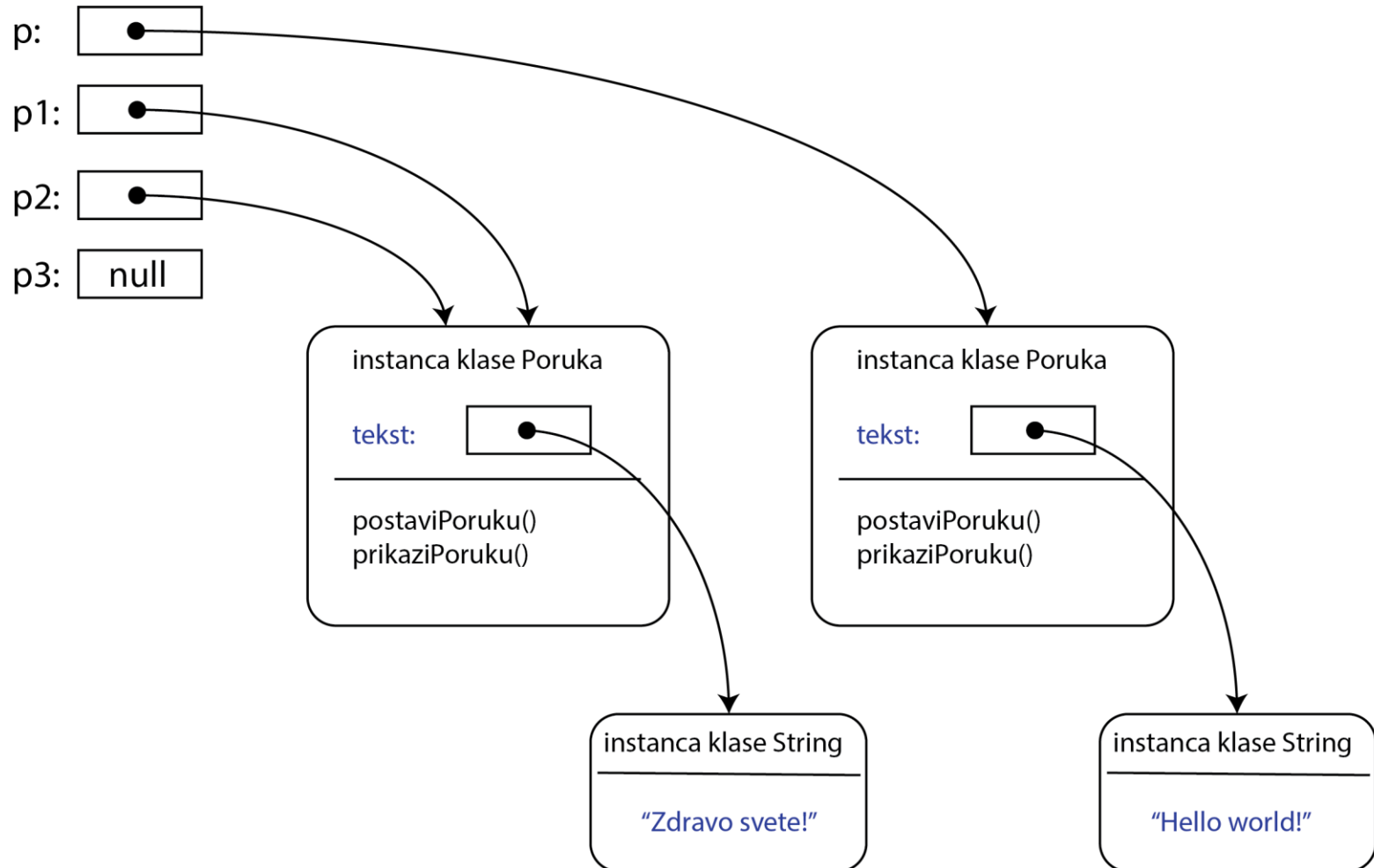
Poruka **p** = **new** Poruka();

- Nakon prethodne naredbe promenljiva **p** će sadržati referencu na objekat kreiran primenom operatora **new**

Promenljive i objekti - primer

```
Poruka p, p1, p2, p3;           // Deklarisanje 4 prom. klase Poruka
p = new Poruka();               // Kreiramo novi objekat klase
                                // Poruka i čuvamo referencu na taj
                                // objekat u promenljivoj p.
p1 = new Poruka();              // Kreiramo još jedan objekat klase
                                // Poruka i čuvamo referencu na njega
                                // u promenljivoj p1.
p2 = p1;                        // Kopiramo vrednost reference iz p1
                                // u promenljivu p2.
p3 = null;                      // Postavljamo null referencu u
                                // promenljivu p3.
p.tekst = " Hello world!";      // Postavljamo vrednost atributa
p1.tekst = "Zdravo svete!";     //tekst instanci na neke vrednosti
```


Promenljive i objekti - primer



Promenljive i objekti - primer

Kada se promenljiva jednog objekta dodeli drugoj, kopira se samo referenca. Referisani objekat se ne kopira!

Primer 1.2

- Kreirati klasu Student sa atributima ime, prezime, broj poena na testu 1 i testu 2, prosečan broj poena, kao i metodama za postavljanje imena i prezimena studenta, broja poena i računanje prosečnog broja poena na testovima i štampanje uspeha studenata
- Potom testirati klasu kreiranjem dva objekta u okviru glavnog programa, kojima postavljamo imena i prezimena i broj poena na testovima i za koje potom računamo i prikazujemo uspeh

Primer 1.2 – klasa Student

```
public class Student {  
    String ime;           //ime studenta  
    String prezime;       //prezime studenta  
    double test1, test2, prosek; //ocene na testovima  
  
    void postaviImePrezime(String i, String p){  
        ime = i;  
        prezime = p;  
    }  
  
    void postaviPoene(double t1, double t2){  
        test1 = t1;  
        test2 = t2;  
    }  
  
    ...  
}
```

Primer 1.2 – klasa Student

...

```
void racunajProsek() {                // prosek poena
    prosek = (test1 + test2) / 2;
}
```

```
void stampaPoene(){                  // stampa rezultata
    System.out.println("Student " + ime + " "+ prezime
        + " - prosecan broj poena: " + prosek);
}
```

```
}
```

Primer 1.2 – klasa Main

```
public class Main {  
    public static void main(String[] args){  
        Student prvi = new Student();  
        Student drugi = new Student();  
        prvi.postaviImePrezime("Petar", "Petrovic");  
        drugi.postaviImePrezime("Ivana", "Ivanovic");  
        prvi.postaviPoene(39.5,53.8);  
        drugi.postaviPoene(38.7, 57.5);  
        prvi.racunajProsek();  
        drugi.racunajProsek();  
        prvi.stampajPoene();  
        drugi.stampajPoene();  
    }  
}
```

Zadatak za rad na času

- Kreirati klasu Zaposleni sa atributima ime (tipa String), prezime (tipa String) i koefRadnogMesta (tipa double), kao i metodama za postavljanje imena i prezimena zaposlenog i postavljanje koeficijenta radnog mesta, računanje plate po formuli $plata = 2500 * koefRadnogMesta$ i štampanje plate zaposlenog.
- Potom testirati klasu kreiranjem tri objekta u okviru glavnog programa, kojima postavljamo imena i prezimena i koeficijente radnog mesta i za koje potom računamo i prikazujemo platu

Zadatak za vežbanje

- Kreirati klasu `Racunar` sa atributima `procesor` (tipa `String`), `radniTakt` (tipa `double`), `kapacitetMemorije` (tipa `int`), kao i metodama za postavljanje naziva i radnog takta procesora, postavljanje kapaciteta memorije u GB, računanje indeksa performansi računara po formuli:
$$\text{indeksPerformansi} = 10 * \text{radniTakt} + \text{kapacitetMemorije}$$
i štampanje indeksa performansi računara.
- Potom testirati klasu kreiranjem pet objekta u okviru glavnog programa, kojima prvo postavljamo vrednosti atributa, a potom računamo i prikazujemo njihovu rang listu uređenu po indeksu performansi u opadajućem redosledu.

KONSTRUKTORI

Konstruktori i inicijalizacija objekata

- **Konstruktor je specijalna metoda** kojom se **kreira novi objekat**, tj. **nova instanca neke klase**
- Svaka klasa ima najmanje jedan konstruktor
- Ako programer ne napiše konstruktor, onda sistem sam kreira standardni (engl. default) konstruktor za datu klasu
- Standardni konstruktor samo alocira memoriju i inicijalizuje promenljive instance
- Specifičnosti definicije konstruktora:
 - Nema povratni tip (čak ni void!)
 - Mora se zvati isto kao i klasa
 - Prilikom definicije konstruktora, mogu se navesti samo atributi za kontrolu pristupa

Konstruktor - primer

- Kreiramo klasu `Valjak`
- Atributi valjka su poluprečnik osnove `r` i visina valjka `H`
- U klasi definišemo dva konstruktora: standardni konstruktor `Valjak()`, kojim se alocira memorija i konstruktor `Valjak(float r1, float H1)`, čijim se korišćenjem prilikom kreiranja novih objekata klase `Valjak` definišu i početne vrednosti atributa `r` i `H`

Primer 1.3 – klasa Valjak

```
public class Valjak {  
    float r;  
    float H;  
  
    Valjak() {}  
  
    Valjak(float r1, float H1){  
        r=r1;  
        H=H1;  
    }  
}  
  
...  
public static void main(String[] args){  
    ...  
    Valjak cev = new Valjak(2.0f, 200.0f);  
    ...  
}
```

Primer 1.4 – klasa Student sa konstruktorom

```
public class Student {  
    String ime;           //ime studenta  
    String prezime;       //prezime studenta  
    double test1, test2, prosek; //ocene na testovima i  
                                //njihov prosek  
  
    Student(String i, String p, double t1, double t2){  
        ime = i;  
        prezime = p;  
        test1 = t1;  
        test2 = t2;  
    }  
}
```

...

Primer 1.4 – klasa Student sa konstruktorom

...

```
void racunajProsek() {    //prosek poena
    prosek = (test1 + test2) / 2;
}
```

```
void stampajPoene(){
    System.out.println("Student "+pribaviIme()+" "+
        pribaviPrezime()+" - prosecan broj
        poena: " + racunajProsek());
}
```

```
}
```

Primer 1.4 – klasa Main

```
public class Main {  
  
    public static void main(String[] args){  
  
        Student prvi = new Student("Petar", "Petrovic",  
                                    39.5, 53.8);  
        Student drugi = new Student("Ivana", "Ivanovic",  
                                    38.7, 57.5);  
  
        prvi.racunajProsek();  
        drugi.racunajProsek();  
        prvi.stampajPoene();  
        drugi.stampajPoene();  
    }  
}
```

Specijalna promenljiva `this`

- **`this`** je specijalna promenljiva koja čuva referencu na objekat nad kojim je pozvan metod
- Klasa može da pristupi atributu ili pozove metodu/konstruktor iz iste klase:

<code>this</code> .atribut;	//pristup atributu
<code>this</code> .metod(parametri);	//pristup metodi
<code>this</code> (parametri);	//konstruktor

Primer 1.5 – klasa Student sa konstruktorom i korišćenjem this

```
public class Student {  
    String ime;                //ime studenta  
    String prezime;           //prezime studenta  
    double test1, test2, prosek; //ocene na testovima  
  
    Student(String ime, String prezime, double test1, double  
        test2){  
        this.ime = ime;  
        this.prezime = prezime;  
        this.test1 = test1;  
        this.test2 = test2;  
    }  
}
```

...

Primer 1.5 – klasa Student sa konstruktorom i korišćenjem this

...

```
void racunajProsek() { // metoda racuna prosek poena
    prosek = (test1 + test2) / 2;
}
void stampajPoene(){ // metoda za stampu
    System.out.println("Student " + ime + " "+
        prezime + " - prosecan broj poena: " + prosek);
}
}
```

Objekti i klase – Zadatak

- Zadatak: Realizovati klasu `Zaposleni` sa atributima `ime`, `prezime`, `koefRadnogMesta`, `plata`, standardnim konstruktorom i konstruktorom koji postavlja inicijalne vrednosti atributa, metodom za računanje plate na osnovu broja radnih dana u mesecu i radnog mesta, kao i metodom za štampanje podataka o zaposlenom i njegovoj plati

- Formula za računanje plate:

$$\text{plata} = \text{koefRadnogMesta} * \text{brojRadnihDana} * 100$$

- Potom testirati klasu kreiranjem tri objekta u okviru glavnog programa, za koje izračunavamo platu i potom prikazujemo podatke o zaposlenima i njihovim platama

Rešenje – dijagram klasa



Rešenje - Klasa Zaposleni

```
public class Zaposleni {  
  
    String ime;  
    String prezime;  
    double koefRadnogMesta;  
    double plata;  
  
    Zaposleni() { }  
  
    Zaposleni(String ime, String prezime,  
               double koefRadnogMesta){  
        this.ime = ime;  
        this.prezime = prezime;  
        this.koefRadnogMesta = koefRadnogMesta;  
    }  
}
```

...

Rešenje - Klasa Zaposleni

...

```
void racunajPlatu(int brojDana){  
    plata = koefRadnogMesta * brojDana * 100;  
}  
  
void stampajPlatu() {  
    System.out.println(ime + " " + prezime + ",  
    na radnom mestu sa koeficijentom " +  
    koefRadnogMesta + " ima platu " + plata);  
}  
}
```

Rešenje - Klasa Main

```
public class Main {  
  
    public static void main(String[] args){  
  
        Zaposleni z1 = new Zaposleni("Petar", "Petrovic",  
                                     3800);  
        Zaposleni z2 = new Zaposleni("Ivana", "Ivanovic",  
                                     3900);  
  
        z1.racunajPlatu(22);  
        z2.racunajPlatu(23);  
        z1.stampajPlatu();  
        z2.stampajPlatu();  
    }  
}
```

Zadatak za rad na času

- Zadatak: Realizovati klasu `Vozilo` sa atributima `marka (String)`, `tip (String)`, `godiste (int)`, `registracija (String)`, `maksBrzina (int)`, `snaga (int)`, standardnim konstruktorom i konstruktorom koji postavlja inicijalne vrednosti, metodom za računanje vrednosti automobila na osnovu formule:

$$\text{cena} = \text{maksBrzina} * \text{snaga} / (2017 - \text{godiste})$$

kao i metodama za štampanje podataka o vozilu i poređenje cene dva vozila.

Klasu testirati kreiranjem više objekata u glavnom programu i pozivanjem odgovarajućih metoda. Nacrtati i dijagram klasa za klasu `Vozilo`.

KONTROLA PRISTUPA I ENKAPSULACIJA

Objektna paradigma

- OOP je deo **objektne paradigme** koja obuhvata osnovne objektne koncepte:
 - **apstraktni tipovi podataka** (engl. *abstract data types*)
 - **enkapsulacija** (engl. *encapsulation*)
 - **nasleđivanje** (engl. *inheritance*)
 - **polimorfizam** (engl. *polymorphism*)

Apstraktni tip podataka

- **Apstraktni tipovi podataka** (engl. abstract data types - ADT) su **matematički model** u kojima je **tip podatka definisan njegovim ponašanjem iz tačke gledišta korisnika podatka**, tj. u smislu mogućih vrednosti podataka, mogućih operacija nad podacima tog tipa i ponašanja ovih operacija.
- Pojam ADT stoji **nasuprot strukturama podataka**, koje su konkretne reprezentacije podataka iz tačke gledišta programera tj. implementatora, a ne korisnika
- ADT je **tip koji je definisao programer**, za koji se mogu **kreirati primerci (instance)** i koji je **predstavljen strukturom i ponašanjem**
- **Koncept klase u OOP** je primer **apstraktnog tipa podataka**

Enkapsulacija

- Enkapsulacija (engl. *encapsulation*): deo softvera ima jasno definisan **interfejs** i **implementaciju**; **interfejs je svima dostupan, implementacija je nedostupna**
- Omogućava sakrivanje informacija (engl. information hiding)
- **Klasa** praktično **enkapsulira podatke i operacije u jedan paket**
- Primenom atributa za kontrolu pristupa, podacima se može pristupu samo preko odgovarajućih metoda
- Atributi su privatni za klasu i zajedno sa privatnim metodama čine **implementaciju klase**
- **Javne metode** čine **interfejs klase**

Pristupni atributi

- Postoje četiri moguće vrednosti pristupnih atributa:
 - **bez pristupnog atributa (default, friend)** – dopušten pristup iz metoda proizvoljne klase iz istog paketa
 - **public** – dopušten pristup iz metoda proizvoljne klase (ne nužno iz istog paketa), sve dok je klasa čiji je to član deklarirana kao *public*
 - **private** – dostupan samo iz metoda unutar klase. Nema pristupa izvan klase.
 - **protected** – dopušten pristup iz metoda proizvoljne klase istog paketa i iz proizvoljne podklase (ne nužno iz istog paketa)

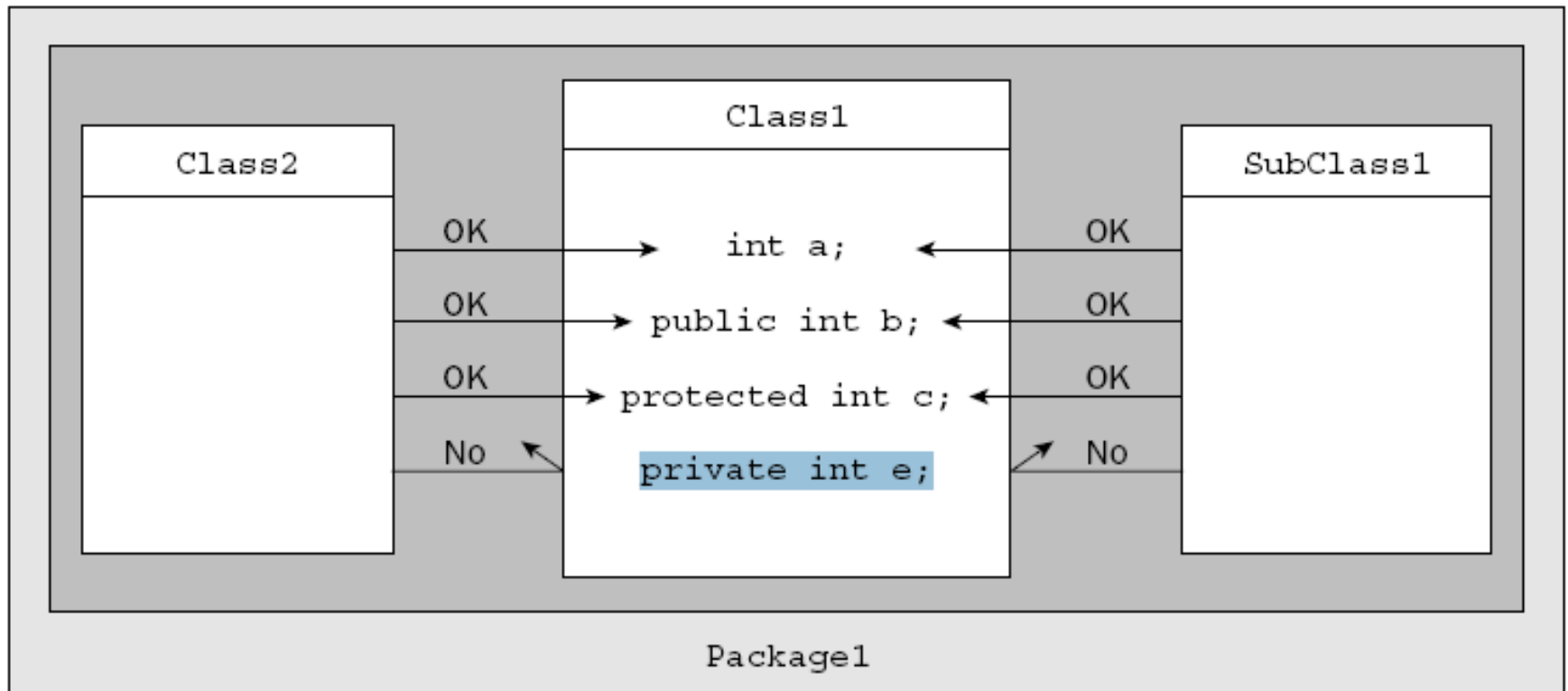
Primer 2.1 - klasa Poruka

```
public class Poruka {  
    private String tekst;  
  
    public void postaviPoruku(String poruka){  
        this.tekst = poruka;  
    }  
  
    public void prikaziPoruku(){  
        System.out.println(this.tekst);  
    }  
}
```

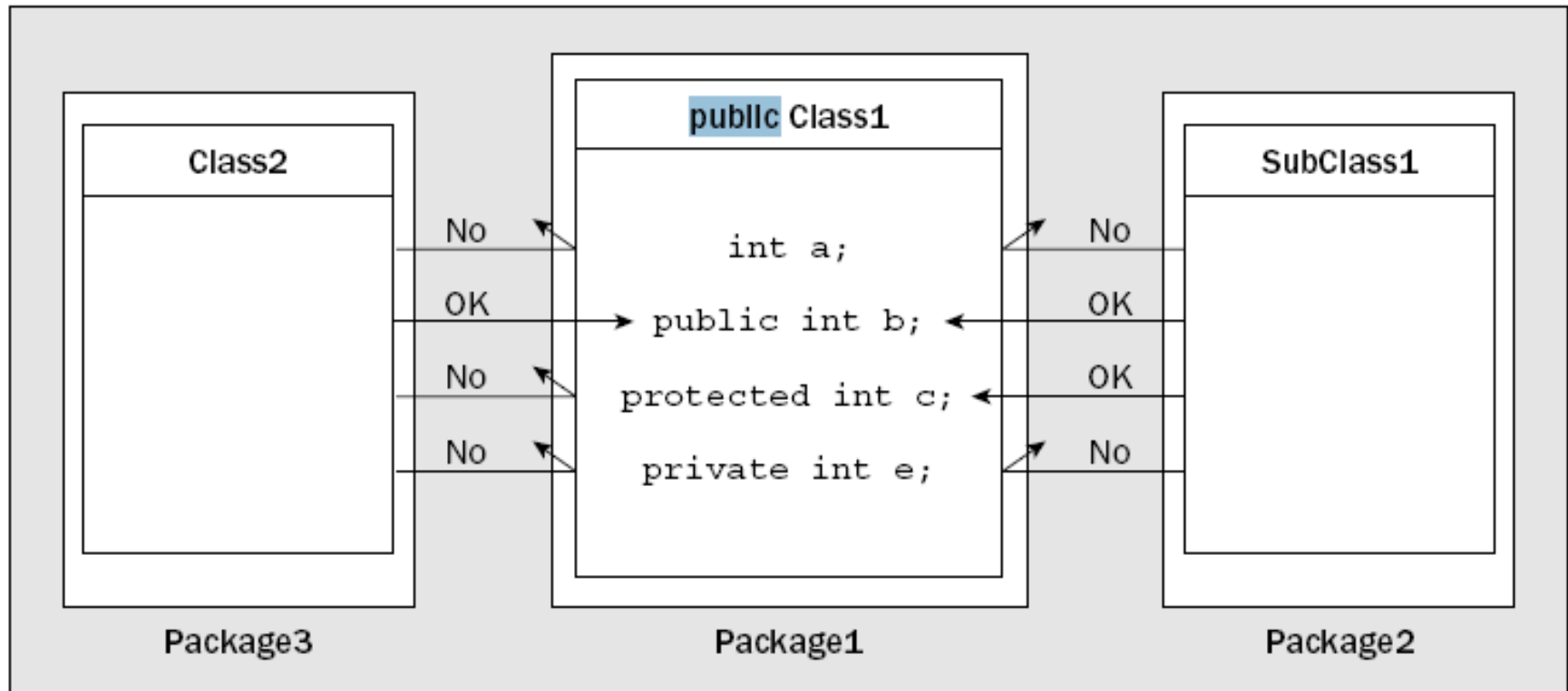
Primer 2.1 – klasa Main

```
public class Main {  
  
    public static void main(String[] args){  
  
        Poruka p = new Poruka();  
        //p.tekst = "Zdravo svete!"; greska!!!  
        p.postaviPoruku("Zdravo OOP svete!");  
        p.prikaziPoruku();  
    }  
}
```

Primer - klase u istom paketu



Primer - klase u različitim paketima



Uobičajeni izbor pristupnih atributa

- Uobičajeno je da instance promenljive budu **private**, tako da im se ne može direktno pristupati, niti se mogu direktno menjati izvan klase. Jedini način da im se pristupi ili da se njihove vrednosti promene jeste pomoću metoda iste klase.
- Ukoliko je potrebno pristupiti vrednostima **private** atributa izvan klase, to se postiže **pristupnom metodom** za pribavljanje (engl. *getter* ili *accessor*) atributa klase
- Ukoliko je potrebno promeniti vrednost **private** atributa izvan klase, to se postiže **pristupnom metodom** za postavljanje (engl. *setter* ili *mutator*) atributa klase

Pristupne metode

- **Metode za pribavljanje atributa** (engl. get) – poznate kao **accessor** ili **getter** metode, omogućavaju pristup privatnim atributima klase iz koda koji se nalazi van klase (npr. iz metode main klase Main)

```
public String pribaviPoruku(){  
    return this.tekst;  
}
```

- **Metode za postavljanje atributa** (engl. set) – poznate kao **mutator** ili **setter** metode, omogućava promenu privatnih atributa klase iz koda koji se nalazi van klase (npr. iz metode main klase Main)

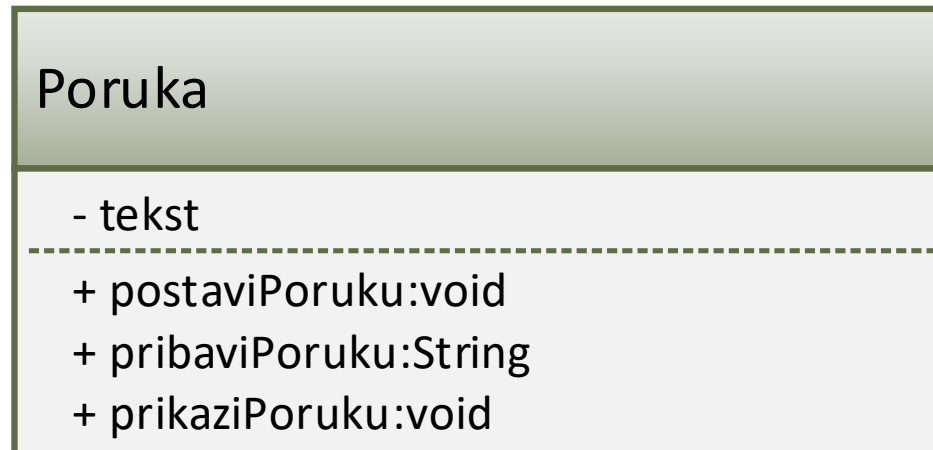
```
public void postaviPoruku(String poruka){  
    this.tekst = poruka;  
}
```

Primer 2.1 - klasa Poruka sa pristupnim metodama

```
public class Poruka {  
    private String tekst;  
  
    public void postaviPoruku(String poruka){  
        this.tekst = poruka;  
    }  
  
    public String pribaviPoruku(){  
        return this.tekst;  
    }  
  
    public void prikaziPoruku(){  
        System.out.println(pribaviPoruku());  
    }  
}
```

Enkapsulacija – rezime

- Atributi za kontrolu pristupa i pristupe metode su jedna od tehnika za odvajanje implementacije klase od njenog interfejsa – omogućavaju sakrivanje informacija, tj. **enkapsulaciju**
- Dijagram klasa za dobro enkapsuliranu klasu Poruka:



- private -, protected #, public +, default (paket) ~

Zadatak za rad na času

- Realizovati prethodno implementirane klase (Student, Zaposleni, Racunar, Vozilo) tako da pruže dobru enkapsulaciju podataka kroz uvođenje pristupnih atributa i realizaciju pristupnih metoda (za postavljanje i pribavljanje vrednosti) za svaki od atributa.

Zadaci za razmišljanje i vežbanje

- Osmisliti, nacrtati dijagram klase i realizovati u Javi klasu `Klijent` za čuvanje i rad sa podacima o klijentima neke banke. Koje attribute bi trebalo posmatrati? Koje osnovne metode bi trebalo implementirati?
- Osmisliti, nacrtati dijagrame klasa i realizovati u Javi klase `Ucionica` i `Racunar` koje bi se mogle koristiti u programu za evidenciju inventara neke institucije. U svakoj učionici može se nalaziti određeni broj računara. Koje attribute bi trebalo posmatrati? Koje osnovne metode bi trebalo implementirati?

NASLEĐIVANJE

Nasleđivanje

- **Nasleđivanje** (engl. inheritance) omogućava **stvaranje nove klase koja je zasnovana na postojećoj klasi**
- Jedna klasa može da nasledi drugu, sa značenjem da su njene instance jedna vrsta instanci osnovne (bazne) klase
- Relacija između **roditeljske klase (bazne klase, nadklase)** i klase potomka (**izvedene klase, podklase**) je **jeste** (engl. *is-a*) – npr. zaposleni jeste osoba
- Atributi i metode iz roditeljske klase se nasleđuju od strane novokreirane klase potomka

Nasleđivanje

- Novi atributi i metodi mogu biti kreirani u novoj klasi, ali oni ne utiču na definiciju roditeljske klase

```
public class Potomak extends Roditelj{  
    //novi atributi i metodi ili izmene  
}
```

- U Javi je omogućeno samo **jednostruko nasleđivanje**, tj. klasa može imati samo jednog roditelja
- Situacije u kojima je potrebno višestruko nasleđivanje se u Javi rešavaju primenom **interfejsa** – klasa može da ima samo jednog roditelja, ali može da implementira više interfejsa
- Sve klase u Javi su izvedene iz klase Object

Nasleđivanje - primer

```
class Osoba {  
    String ime;  
    String prezime;  
    int starost;  
  
    void predstaviSe(){  
        ...  
    }  
}
```

```
class Zaposleni extends Osoba {  
    double plata;  
  
    void racunajPlatu(){  
        ...  
    }  
}
```

Svaki Zaposleni ima atribute ime i starost i metodu predstaviSe, *kao i* atribut plata i metodu racunajPlatu

Specijalna promenljiva **super**

- **super** je specijalna promenljiva koja čuva referencu na objekat roditeljske klase za objekat nad kojim je pozvan metod
- Korišćenjem specijalne promenljive **super**, klasa potomak može da pristupi atributu ili pozove metodu/konstruktor roditeljske klase:

```
super.atribut;           // pristup atributu
```

```
super.metod(parametri);  // pristup metodi
```

```
super(parametri);       // konstruktor
```

Nasleđivanje – primer 2.2

- Roditeljska klasa Osoba

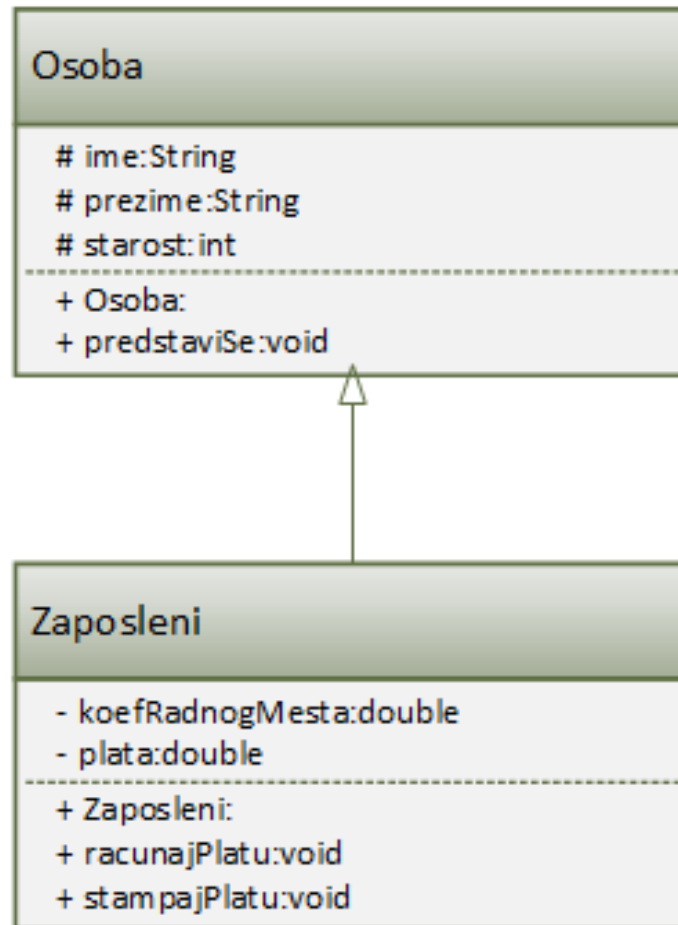
```
public class Osoba
```

- Klasa potomak Zaposleni

```
public class Zaposleni extends Osoba
```

- Prethodne klase testirati kreiranjem dva objekta klase Zaposleni u glavnom programu. Potom odštampati podatke o zaposlenima i izračunati i odštampati njihovu platu pozivom odgovarajućih metoda.

Primer 2.2 – diagram klasa



Primer 2.2 – klasa Osoba

```
public class Osoba {  
    protected String ime;  
    protected String prezime;  
    protected int starost;  
  
    Osoba() {}  
  
    Osoba(String ime, String prezime, int starost){  
        this.ime = ime;  
        this.prezime = prezime;  
        this.starost = starost;  
    }  
  
    ...  
}
```


Primer 2.2 – klasa Osoba

bez metoda za pribavljanje i postavljanje

...

```
public void predstaviSe() {  
    System.out.println("Ime: " + this.ime  
                        + " Prezime: " + this.prezime  
                        + " Starost: " + this.starost  
                        + " godina");  
}  
}
```

Primer 2.2 – klasa Osoba

sa metodama za pribavljanje i postavljanje

...

```
public void predstaviSe() {  
    System.out.println("Ime: " + pribaviIme() +  
        " Prezime: " + pribaviPrezime() +  
        " Starost: " + pribaviStarost() +  
        " godina");  
}  
}
```

Primer 2.2 – klasa Zaposleni

```
public class Zaposleni extends Osoba {  
    private double koefRadnogMesta;  
    private double plata;  
  
    Zaposleni(String ime, String prezime, int starost,  
               double koefRadnogMesta) {  
        super(ime, prezime, starost);  
        this.koefRadnogMesta = koefRadnogMesta;  
    }  
    ...  
}
```


Primer 2.2 – klasa Zaposleni

sa metodama za pribavljanje i postavljanje

...

```
public void postaviPlatu(int brojDana){  
    plata = pribaviKoefRadnogMesta() * brojDana;  
}
```

```
public void stampajPlatu() {  
    System.out.println(pribaviIme() + " " +  
        pribaviPrezime() +  
        ", na radnom mestu sa koeficijentom "  
        + pribaviKoefRadnogMesta() +  
        " ima platu " + pribaviPlatu());  
}
```

Primer 2.2 – klasa Main

```
public class Main {  
    public static void main(String[] args) {  
  
        Zaposleni z1 = new Zaposleni("Petar",  
                                       "Petrovic",  
                                       35, 3800);  
        Zaposleni z2 = new Zaposleni("Ivana",  
                                       "Ivanovic",  
                                       32, 3900);  
  
        ...  
    }  
}
```

Primer 2.2 – klasa Main

```
...  
  
z1.predstaviSe();  
z2.predstaviSe();  
  
z1.racunajPlatu(22);  
z2.racunajPlatu(23);  
  
z1.stampajPlatu();  
z2.stampajPlatu();  
}  
}
```

Nasleđivanje – primer 2.3

- Roditeljska klasa Osoba

```
public class Osoba
```

- Klasa potomak Zaposleni

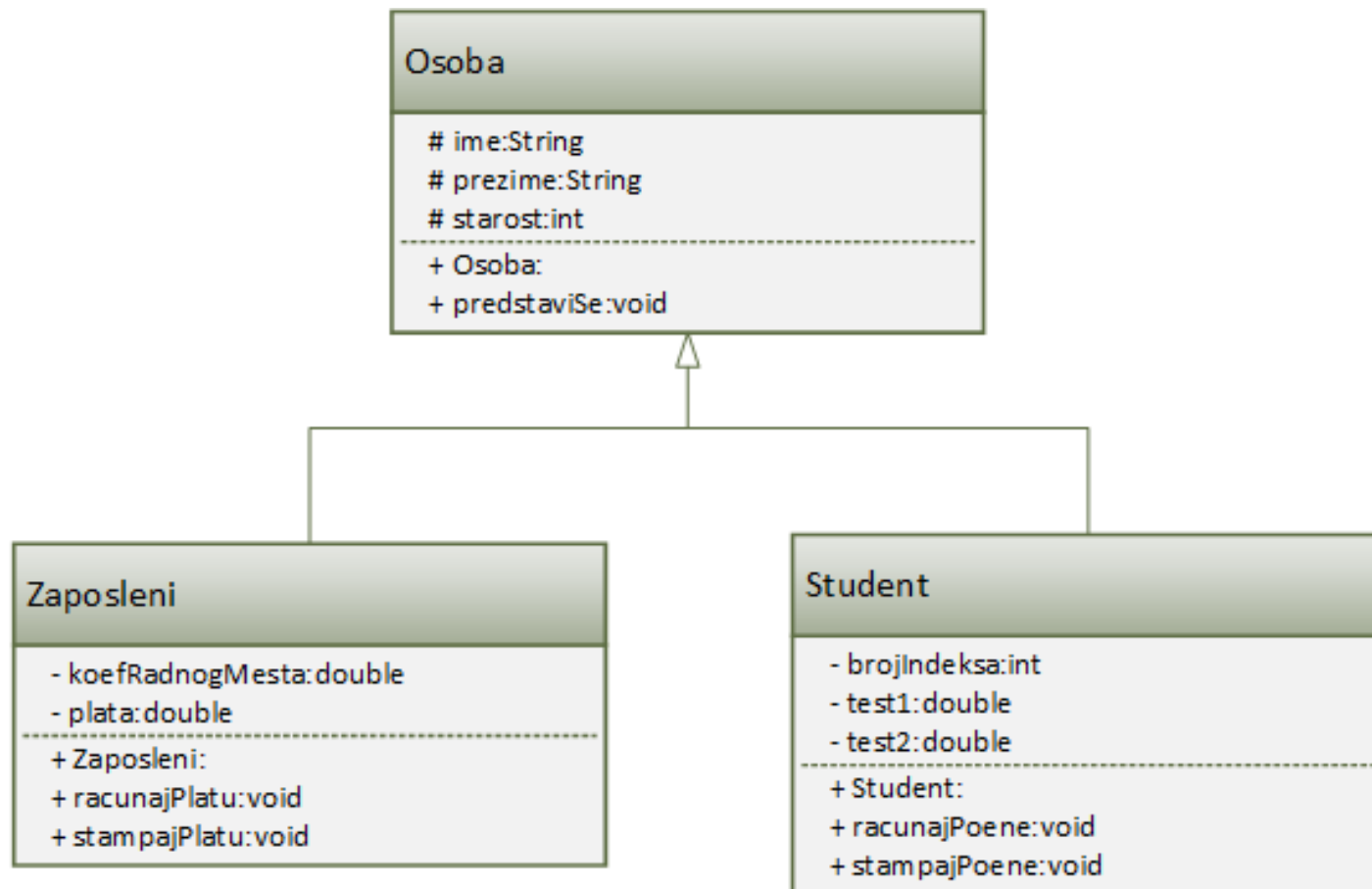
```
public class Zaposleni extends Osoba
```

- Klasa potomak Student

```
public class Student extends Osoba
```

- Dodati i dva objekta klase Student u glavnom programu. Potom odštampati podatke i o studentima i izračunati i odštampati prosečan broj poena koje su osvojili pozivom odgovarajućih metoda.

Primer 2.3 – dijagram klasa



Primer 2.3 – klasa Student

```
public class Student extends Osoba {  
    private int brojIndeksa;  
    private double test1, test2; //broj poena na testovima  
  
    Student(String ime, String prezime, int starost,  
            int brojIndeksa, double test1, double test2){  
        super(ime, prezime, starost);  
        this.brojIndeksa = brojIndeksa;  
        this.test1 = test1;  
        this.test2 = test2;  
    }  
    ...  
}
```

Primer 2.3 – klasa Student

sa metodama za pribavljanje i postavljanje

...

```
public double racunajPoene() { // metoda za prosek poena
    double prosek = (test1 + test2) / 2;
    return prosek;
}
public void stampajPoene(){ // metoda za stampu
    System.out.println("Student " + pribaviIme() +
        " "+ pribaviPrezime()
        + " - prosecan broj poena: "
        + racunajPoene());
}
}
```

Primer 2.3 – klasa Main

```
public class Main {  
    public static void main(String[] args) {  
  
        Zaposleni z1 = new Zaposleni("Petar", "Petrovic",  
                                     35, 3800);  
        Zaposleni z2 = new Zaposleni("Ivana", "Ivanovic",  
                                     32, 3900);  
  
        Student s1 = new Student("Marko", "Markovic",  
                                 21, 10482, 25.5, 28.7);  
        Student s2 = new Student("Marina", "Marinovic",  
                                 20, 10505, 28.8, 30.5);  
  
        ...  
    }  
}
```

Primer 2.3 – klasa Main

...

```
z1.predstaviSe();  
z2.predstaviSe();  
s1.predstaviSe();  
s2.predstaviSe();
```

```
z1.postaviPlatu(22);  
z2.postaviPlatu(23);  
s1.racunajPoene();  
s2.racunajPoene();
```

```
z1.stampajPlatu();  
z2.stampajPlatu();  
s1.stampajPoene();  
s2.stampajPoene();
```

```
}
```

```
}
```

Primer 2.3 – Zadatak za rad na času

- Roditeljska klasa Osoba

```
public class Osoba
```

- Klasa potomak Klijent

```
public class Klijent extends Osoba
```

- Kreirati dva objekta klase Klijent u glavnom programu. Potom odštampati osnovne podatke o klijentima. Zatim izvršiti transfer sa računa jednog klijenta na račun drugog klijenta. Prikazati stanja računa klijenata pre i posle transfera pozivom odgovarajućih metoda.

Nasleđivanje – primer 2.4

- Roditeljska klasa Vozilo

```
public class Vozilo
```

- Klasa potomak Automobil

```
public class Automobil extends Vozilo
```

- Klasa potomak Kamion

```
public class Kamion extends Vozilo
```

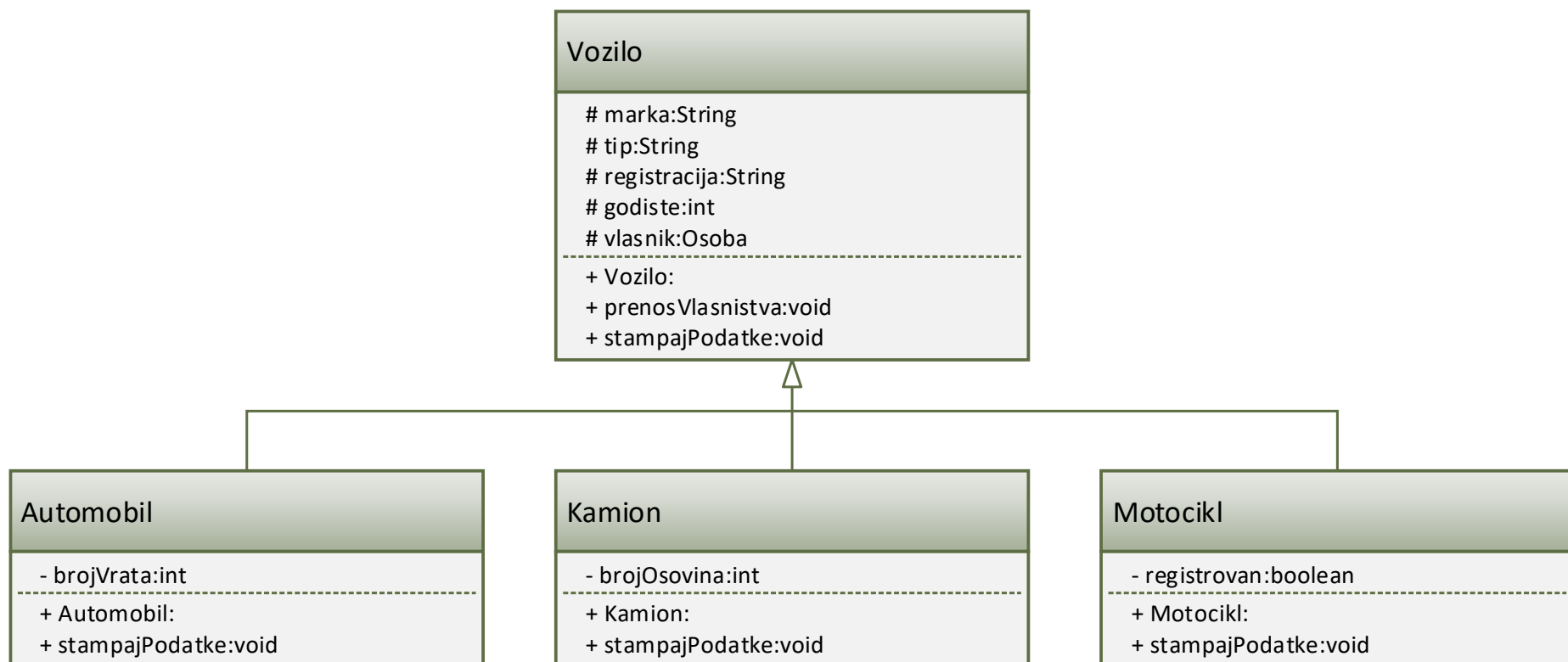
- Klasa potomak Motocikl

```
public class Motocikl extends Vozilo
```

Nasleđivanje – primer 2.4

- Prethodne klase testirati kreiranjem dva objekta klase Osoba (iz prethodnog primera), i po jednog objekta klase Automobil, Kamion i Motocikl.
- Potom odštampati podatke o kreiranim objektima i izvršiti prenos vlasništva svakog od vozila sa jednog vlasnika na drugog. Zatim ponovo prikazati podatke o vozilima.

Primer 2.4 – dijagram klasa



Primer 2.4 – klasa Vozilo

```
public class Vozilo {  
    protected String marka;  
    protected String tip;  
    protected String registracija;  
    protected int godiste;  
    protected Osoba vlasnik;  
  
    Vozilo(String marka, String tip, String registracija,  
           int godiste, Osoba vlasnik){  
        this.marka = marka;  
        this.tip = tip;  
        this.registracija = registracija;  
        this.godiste = godiste;  
        this.vlasnik = vlasnik;  
    }  
    ...  
}
```

Primer 2.4 – klasa Vozilo

...

```
protected void prenosVlasnistva(Osoba noviVlasnik) {  
    this.vlasnik = noviVlasnik;  
}
```

```
protected void stampaJPodatke() {  
    System.out.println(pribaviMarku() + " " +  
        pribaviTip() + " " +  
        pribaviGodiste() + " " +  
        pribaviRegistraciju() + " " +  
        vlasnik.pribaviIme() + " " +  
        vlasnik.pribaviPrezime());  
}
```

```
}
```

Primer 2.4 – klasa Automobil

```
public class Automobil extends Vozilo {  
    private int brojVrata;  
  
    Automobil(String marka, String tip, String registracija,  
        int godiste, Osoba vlasnik, int brojVrata){  
        super(marka, tip, registracija, godiste, vlasnik);  
        this.brojVrata = brojVrata;  
    }  
  
    public void stampajPodatke() {  
        System.out.println(pribaviMarku() + " " +  
            pribaviTip() + " " +  
            pribaviGodiste() + " " +  
            pribaviRegistraciju() + " " +  
            pribaviBrojVrata() + " " +  
            vlasnik.pribaviIme() + " " +  
            vlasnik.pribaviPrezime());  
    }  
}
```

Primer 2.4 – klasa Kamion

```
public class Kamion extends Vozilo {  
    private int brojOsovina;  
  
    Kamion(String marka, String tip, String registracija,  
            int godiste, Osoba vlasnik, int brojOsovina){  
        super(marka, tip, registracija, godiste, vlasnik);  
        this.brojOsovina = brojOsovina;  
    }  
  
    public void stampajPodatke() {  
        System.out.println(pribaviMarku() + " " +  
            pribaviTip() + " " +  
            pribaviGodiste() + " " +  
            pribaviRegistraciju() + " " +  
            pribaviBrojOsovina() + " " +  
            vlasnik.pribaviIme() + " " +  
            vlasnik.pribaviPrezime());  
    }  
}
```

Primer 2.4 – klasa Motocikl

```
public class Motocikl extends Vozilo {  
    private boolean registrovan;  
  
    Motocikl(String marka, String tip, String registracija,  
        int godiste, Osoba vlasnik, boolean registrovan){  
        super(marka, tip, registracija, godiste, vlasnik);  
        this.registrovan = registrovan;  
    }  
  
    public void stampajPodatke() {  
        System.out.println(pribaviMarku() + " " +  
            pribaviTip() + " " +  
            pribaviGodiste() + " " +  
            pribaviRegistraciju() + " " +  
            pribaviRegistrovan() + " " +  
            vlasnik.pribaviIme() + " " +  
            vlasnik.pribaviPrezime());  
    }  
}
```

Primer 2.4 – klasa Main

```
public class Main {  
    public static void main(String[] args) {  
  
        Osoba o1 = new Osoba("Petar", "Petrovic", 35);  
        Osoba o2 = new Osoba("Ivana", "Ivanovic", 32);  
  
        //Vozilo v = new Vozilo("Opel", "Corsa", "NS021IT",  
        //                      2004, o1);  
        Automobil a = new Automobil("Opel", "Corsa", "NS021IT",  
                                     2004, o1, 5);  
        Kamion k = new Kamion("Volvo", "FH", "NS021RS", 2016,  
                              o1, 3);  
        Motocikl m = new Motocikl("Honda", "Rebel", "ZA018RS",  
                                   1998, o1, false);  
  
        ...  
    }  
}
```

Primer 2.4 – klasa Main

...

```
//v.stampajPodatke();  
a.stampajPodatke();  
k.stampajPodatke();  
m.stampajPodatke();
```

```
a.prenosVlasnistva(o2);  
k.prenosVlasnistva(o2);  
m.prenosVlasnistva(o2);
```

```
a.stampajPodatke();  
k.stampajPodatke();  
m.stampajPodatke();
```

```
}
```

```
}
```


Nasleđivanje

Promenljiva koja može da sadrži referenca na objekat klase A može da sadrži i referencu na objekat koji pripada i bilo kojoj od podklasa klase A.

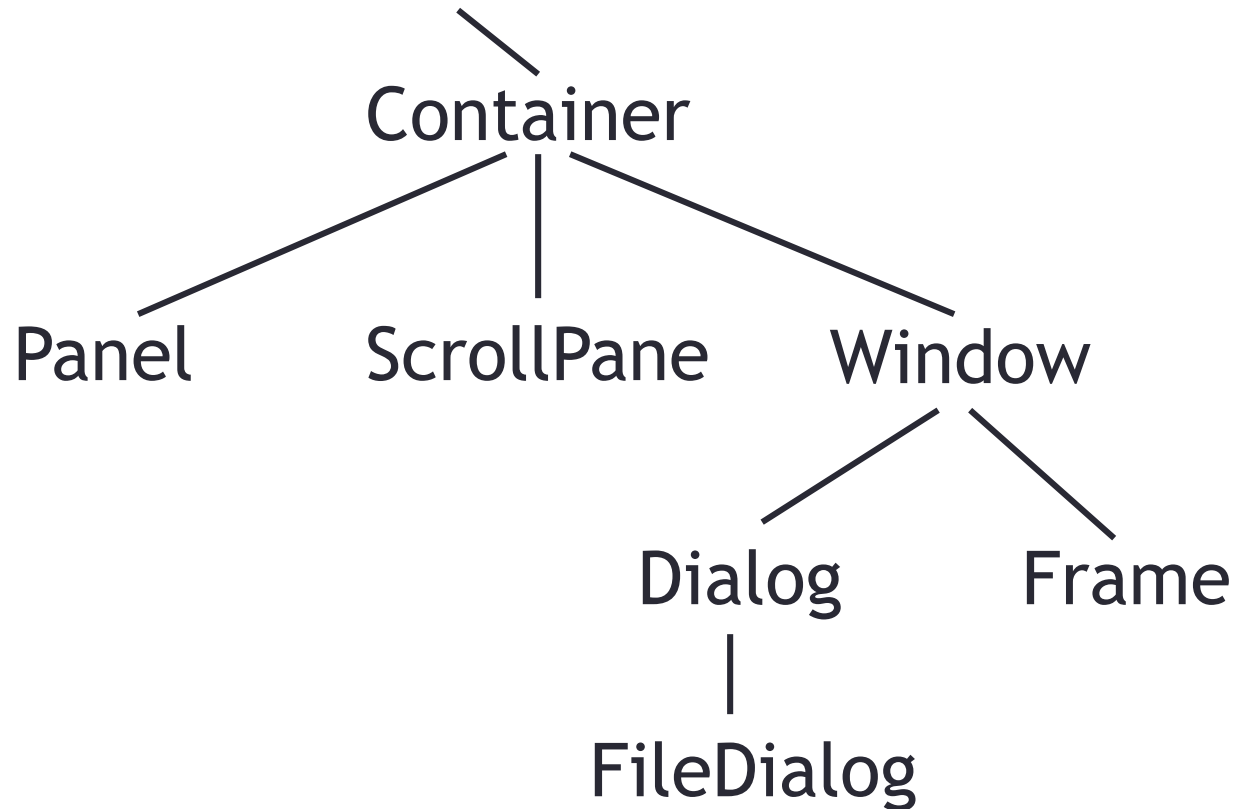
Dakle, potpuno su validne sledeće naredbe:

```
Vozilo v = a;  
Vozilo v1 = new Automobil("Opel", "Corsa", "NS021IT", 2004, o1, 5);
```

Promenljive `v` i `v1` sadrže referencu na objekat tipa `Vozilo` koji je instanca njene potklase `Automobil`. Informacija o stvarnoj klasi objekta se čuva kao deo objekta i može se dobiti pomoću `instanceof` operatora npr.:

```
if (v instanceof Automobil) ...
```

Nasleđivanje – primer 3 - GUI



FileDialog je Dialog, Dialog je Window, Window je Container

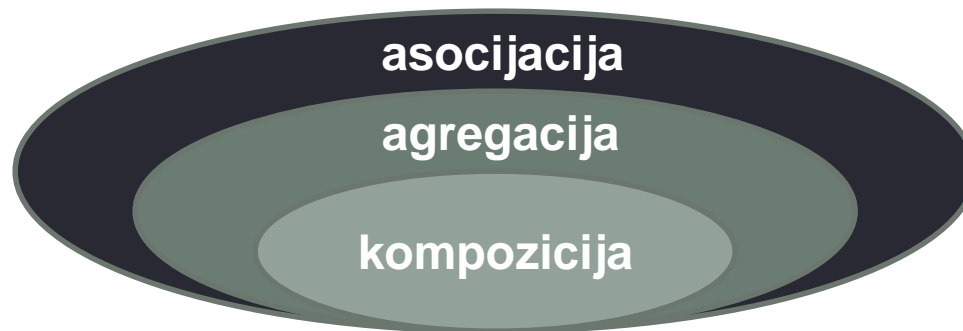
Zadatak za rad na času

- Zadatak 1: Realizovati klasu `Oblik` sa atributima `boja` (`String`), `tip` (`String`), `brojStrana` (`int`), `površina` (`double`), standardnim konstruktorom i konstruktorom koji postavlja inicijalne vrednosti, metodama za postavljanje i pribavljanje atributa, kao i metodama za promenu boje oblika i štampanje podataka o obliku. Realizovati klase `Kvadrat` (dodatni atribut `duzinaStranice` tipa `double`) i `Krug` (dodatni atribut `poluprecnik` tipa `double`) koje nasleđuju klasu `Oblik` i implementiraju metode za računanje površine.
- Klasu testirati kreiranjem više objekata u glavnom programu i pozivanjem odgovarajućih metoda. Nacrtati i dijagram klasa.

SLAGANJE

Slaganje

- **Slaganje** takođe omogućava kreiranje nove klase od postojećih, ali ne nasleđivanjem, već njihovim “slaganjem” – “složeni” objekat sadrži “prostije” objekte, relacija klasa je **ima** (engl. *has-a*) – npr. **vozilo ima motor**
- Slaganje je jednostavnije i fleksibilnije od nasleđivanja
- Vrste slaganja: **asocijacija**, **agregacija** i **kompozicija**



Slaganje

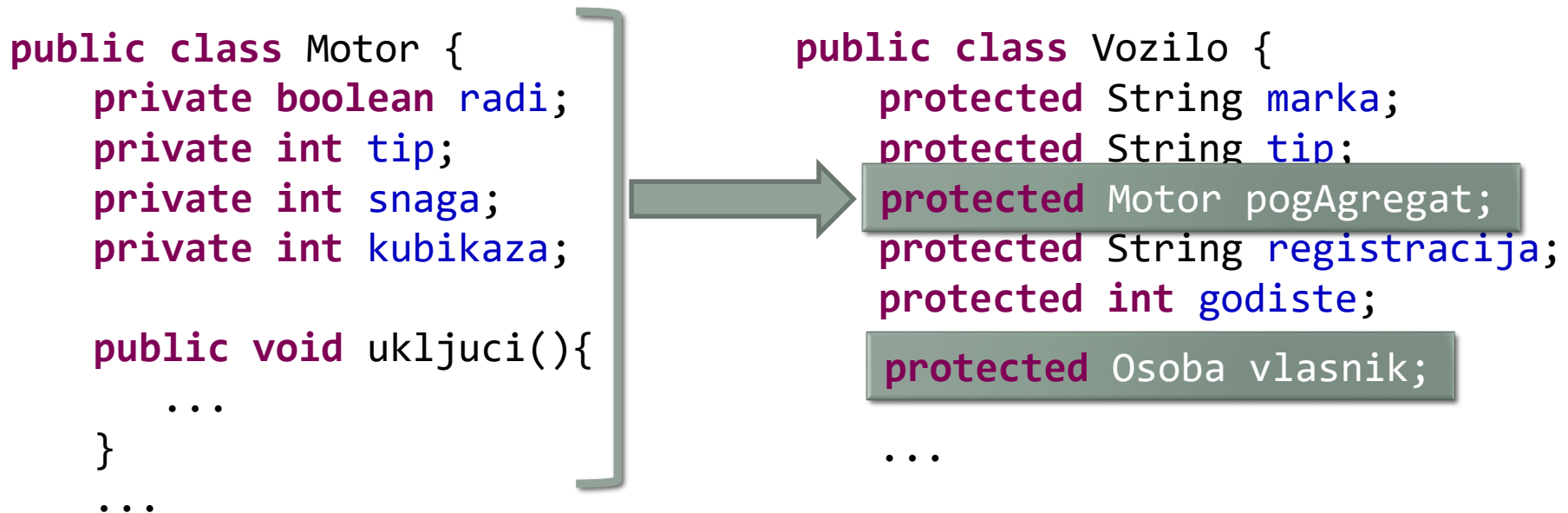
Vrste slaganja:

- **Asocijacija** (engl. association) – semantički slaba veza – primer osoba-vozilo, doktor-pacijent
- **Agregacija** (engl. aggregation) – specijalni vid unidirekcione asocijacije – tipično odnos celina/komponenta, primer vozilo-motor
- **Kompozicija** (engl. composition) – specijalni vid “jake” agregacije – primer kuća-sobe

	Asocijacija	Agregacija	Kompozicija
Vlasnik	Nema vlasnika	Jedan vlasnik	Jedan vlasnik
Životni vek	Poseban vek	Poseban vek	Vek vlasnika

Slaganje – primer 2.5 Vozilo i Motor

- Klasa Vozilo i klasa Motor



- Agregacija Vozilo-Motor, asocijacija Vozilo-Osoba
- Za vežbu: proširićemo primer 2.4 uvodeći klasu Motor primenom agregacije

Slaganje – primer 2.5 Vozilo i Motor

```
public class Motor {  
    boolean radi;           //pokrenut ili ne  
    private String tip;     //dizel ili benzin  
    private int snaga;      // snaga u kW  
    private int kubikaza;   // kubikaza u ccm  
  
    Motor() {}  
  
    Motor(boolean radi, String tip, int snaga, int kubikaza){  
        this.radi = radi;  
        this.tip = tip;  
        this.snaga = snaga;  
        this.kubikaza = kubikaza;  
    }  
  
    ...  
}
```

Slaganje – primer 2.5 Vozilo i Motor

...

```
public void postaviRadi(boolean radi) {  
    this.radi = radi;  
}
```

```
public boolean pribaviRadi() {  
    return this.radi;  
}
```

```
public void ukljuci() {  
    if (this.pribaviRadi()==false)  
        this.postaviRadi(true);  
}
```

```
public void iskljuci() {  
    if (this.pribaviRadi()==true)  
        this.postaviRadi(false);  
}
```

...

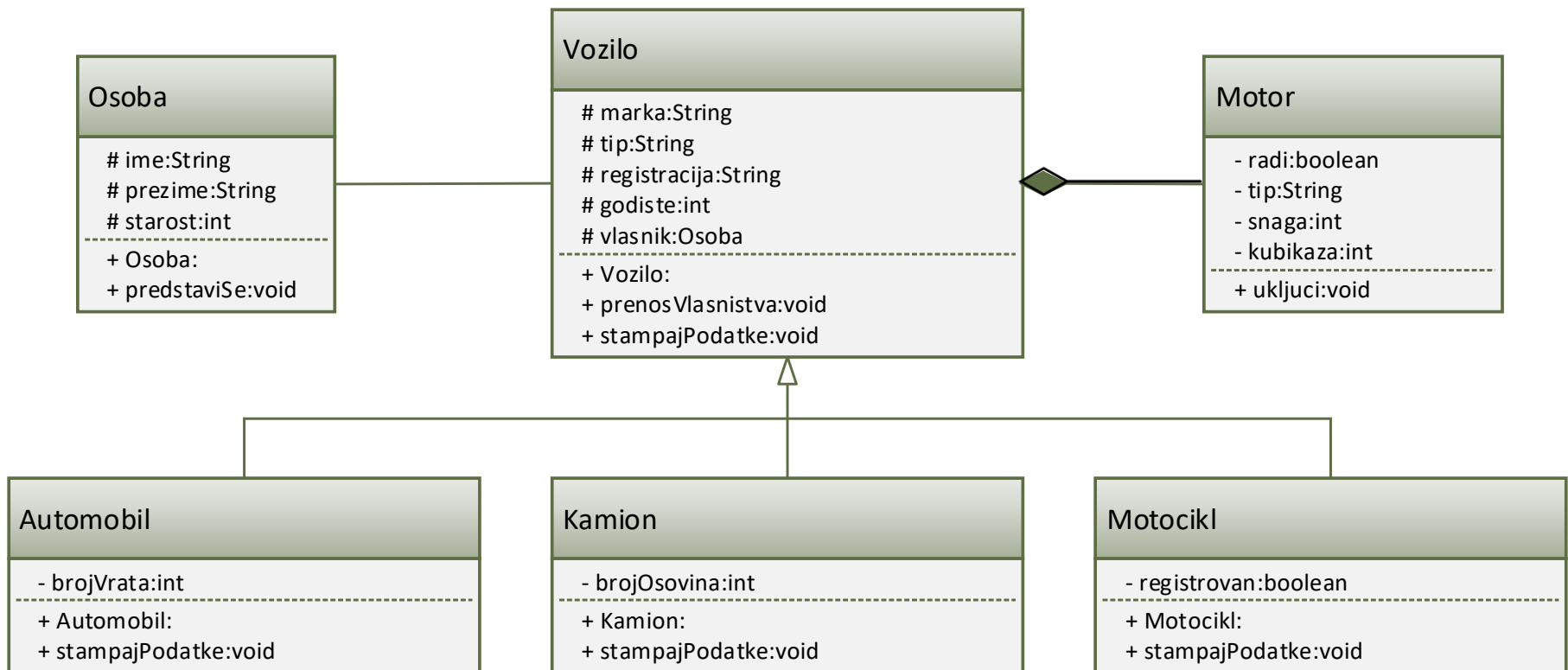
Slaganje – primer 2.5 Vozilo i Motor

...

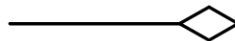
```
public void stampajPodatke() {  
    System.out.println("Informacije o motoru:");  
    System.out.println("Ukljucen:" + pribaviRadi() +  
        " Tip: " + pribaviTip() +  
        " Snaga: " + pribaviSnagu() +  
        " Kubikaza: " + pribaviKubikazu());  
}  
}
```

Slaganje – dijagram klasa

- Klasa Vozilo i klase Motor i Osoba



nasleđivanje



agregacija



asocijacija

Zadatak za vežbanje

- Zadatak 1: Osmisliti, nacrtati dijagrame klasa i realizovati u Javi klase Institucija, Ucionica, Zaposleni (koja nasleđuje klasu Osoba) i Racunar koje bi se mogle koristiti u programu za evidenciju zaposlenih i inventara neke obrazovne institucije. Spiskove učionica, zaposlenih i računara po učionicama čuvati u odgovarajućim nizovima (svaki sa najviše 20 članova). U kojim međusobnim odnosima se nalaze pomenute klase (nasleđivanje, asocijacija, agregacija, kompozicija)?
- Klase testirati kreiranjem objekata u glavnom programu i pozivanjem izabranih metoda. Nacrtati UML dijagram klasa rešenja.

POLIMORFIZAM

Objektna paradigma

- OOP je deo **objektne paradigme** koja obuhvata osnovne objektne koncepte:
 - **apstraktni tipovi podataka** (engl. *abstract data types*)
 - **enkapsulacija** (engl. *encapsulation*)
 - **nasleđivanje** (engl. *inheritance*)
 - **polimorfizam** (engl. *polymorphism*)



Polimorfizam

- Četvrti ključni koncept u okviru **objektne paradigme**
- **Polimorfizam** (engl. polymorphism) je **svojstvo da različiti objekti mogu da odgovaraju na iste poruke na različite načine**. Sam termin znači “mnoštvo oblika”.
- Polimorfizam praktično omogućava “virtuelizaciju” objekata.
- Polimorfni metodi mogu se **adaptirati na specifičnosti objekta** nad kojim su pozvani. Polimorfizam se može definisati i kao svojstvo da se prilikom poziva odaziva odgovarajuća verzija metode klase čiji su naslednici dali nove verzije.

Polimorfizam – primer 3.1

- Realizovati klasu `Oblik` sa atributima `boja` (`String`), `tip` (`String`), `brojStrana` (`int`), standardnim konstruktorom i konstruktorom koji postavlja inicijalne vrednosti, kao i metodama za promenu boje oblika i štampanje podataka o obliku.
- Realizovati klase `Kvadrat` (dodatni atribut `duzinaStranice` tipa `double`) i `Krug` (dodatni atribut `poluprecnik` tipa `double`) koje nasleđuju klasu `Oblik` i implementiraju metode za računanje površine i obima.
- Klase testirati kreiranjem više objekata u glavnom programu i pozivanjem metoda za računanje površine i obima i štampu podataka.

Primer 3.1 – klasa Oblik

```
public class Oblik {  
    private String boja;  
    private String tip;  
    private int brojStrana;  
  
    Oblik(){}  
  
    Oblik(String boja, String tip, int brojStrana){  
        this.boja = boja;  
        this.tip = tip;  
        this.brojStrana = brojStrana;  
    }  
  
    ...  
}
```

Primer 3.1 – klasa Oblik

```
...  
public void postaviBoju(String novaBoja) {  
    this.boja = novaBoja;  
}  
  
public String pribaviBoju() {  
    return this.boja;  
}  
  
public String pribaviTip() {  
    return this.tip;  
}  
  
...
```


Primer 3.1 – klasa Krug

```
public class Krug extends Oblik {  
    private double poluprecnik;  
  
    Krug(String boja, int brojStrana, double poluprecnik){  
        super(boja, "Krug", brojStrana);  
        this.poluprecnik = poluprecnik;  
    }  
  
    public double racunajPovrsinu() {  
        return this.poluprecnik*this.poluprecnik*Math.PI;  
    }  
  
    public double racunajObim() {  
        return 2*this.poluprecnik*Math.PI;  
    }  
  
    public void stampajPodatke() {  
        System.out.println(pribaviBoju() + " " + pribaviTip() + " "  
            + pribaviBrojStrana()+" " +  
            racunajPovrsinu()+" "  
            racunajObim());  
    }  
}
```


Primer 3.1 – klasa Main

```
public class Main {  
    public static void main(String[] args) {  
  
        Krug kr = new Krug("Crvena",1, 2.0);  
        Kvadrat kv = new Kvadrat("Bela",4, 1.5);  
  
        kr.stampajPodatke();  
        kv.stampajPodatke();  
    }  
}
```


Polimorfizam – primer 3.2

- Realizovati klasu Zena, izvedenu iz klase Osoba, koja ima i atribut devojackoPrezime. Objekti klase Zena treba da odgovaraju na poruku `predstaviSe`, ali dame skoro nikada ne otkrivaju svoje godine. Zato objekat klase Zena treba da ima funkciju `predstaviSe`, samo što će ona izgledati nešto drugačije, svojstveno izvedenoj klasi Zena – bez saopštavanja podataka o starosti, ali sa devojačkim prezimenom.
- Klase testirati kreiranjem više objekata u glavnom programu i pozivanjem metoda za predstavljanje.

Primer 3.2 – klasa Osoba

```
public class Osoba {  
    private String ime;  
    private String prezime;  
    private int starost;  
  
    Osoba() {}  
  
    Osoba(String ime, String prezime, int starost){  
        this.ime = ime;  
        this.prezime = prezime;  
        this.starost = starost;  
    }  
  
    public String pribaviIme(){  
        return this.ime;  
    }  
  
    ...  
}
```

Primer 3.2 – klasa Osoba

```
...  
public String pribaviPrezime(){  
    return this.prezime;  
}  
  
public int pribaviStarost(){  
    return this.starost;  
}  
  
public void postaviIme(String ime){  
    this.ime = ime;  
}  
  
public void postaviPrezime(String prezime){  
    this.prezime = prezime;  
}  
...
```

Primer 3.2 – klasa Osoba

...

```
public void postaviStarost(int starost){  
    this.starost = starost;  
}
```

```
void predstaviSe() {  
    System.out.println("Ime: " + pribaviIme() +  
        " Prezime: " + pribaviPrezime() +  
        " Starost: " + pribaviStarost());  
}
```

```
}
```

Primer 3.2 – klasa Zena

```
public class Zena extends Osoba {
    String devojackoPrezime;

    Zena(String ime, String prezime, String devojackoPrezime,
        int starost){
        super(ime, prezime, starost);
        this.devojackoPrezime = devojackoPrezime;
    }

    public String pribaviDevojackoPrezime(){
        return this.devojackoPrezime;
    }

    public void postaviDevojackoPrezime(String devojackoPrezime){
        this.devojackoPrezime = devojackoPrezime;
    }
    ...
}
```

Primer 3.2 – klasa Zena

...

```
void predstaviSe() {  
    System.out.println("Ime: " + pribaviIme() +  
        " Prezime: " + pribaviPrezime() +  
        " Devojacko prezime: " +  
        pribaviDevojackoPrezime());  
}
```

Primer 3.2 – klasa Main

```
public class Main {  
    public static void main(String[] args) {  
  
        Osoba o = new Osoba("Ivana", "Ivanovic", 32);  
        Zena z = new Zena("Ivana", "Ivanovic", "Petrovic", 32);  
  
        o.predstaviSe();  
        z.predstaviSe();  
  
    }  
}
```

Zadatak za rad na času

- Realizovati klase Nastavnik, Asistent i NenastavniRadnik, izvedene nasleđivanjem iz klase Zaposleni. Klasa Nastavnik ima dodatne attribute zvanje (tipa String) i brojSCIRadova (tipa int), klasa Asistent ima dodatne attribute mentor (tipa String) i godinaDoktorskihStudija (tipa int), a klasa NenastavniRadnik ima dodatne attribute radnoMesto (tipa String) i godineStaza (tipa int). Za svaku od klasa realizovati metodu predstaviSe i racunajPlatu uzimajući u obzir specifične attribute za svaku od klasa. Napomena: platu za nastavnike računati kao $60000 + \text{brojSCIRadova} * 3000$, kod asistenata kao $40000 + \text{godinaDoktorskihStudija} * 2000$, a kod nenastavnih radnika kao $30000 + \text{godineStaza} * 500$.
- Klase testirati kreiranjem više objekata u glavnom programu i pozivanjem metoda za predstavljanje i računanje plate.

APSTRAKTNE KLASSE I INTERFEJSI

Apstraktne klase

- Oblik predstavlja samo apstraktnu ideju
- Samo za konkretne oblike, kao što su krug i kvadrat, znamo kako da računamo površinu i obim
- Čemu onda služe metode `racunajPovrsinu` i `racunajObim` u klasi `Oblik`? Kako bi mogli da ih implementiramo?
- U suštini nikada nemamo razloga da pravimo objekat klase `Oblik`! Možemo da imamo promenljive tipa `Oblik`, ali one će uvek referencirati na konkretan oblik iz neke podklase
- Rešenje: definisati klasu `Oblik` kao **apstraktnu klasu**

Apstraktne klase

- **Apstraktna klasa je klasa koja ne služi za kreiranje objekata, već samo kao osnova za izvođenje podklasa**
- **Apstraktna klasa postoji samo kako bi izrazila zajednička svojstva svih njenih podklasa.** Klasa koja nije apstraktna naziva se **konkretnom**. Objekti se mogu kreirati samo na osnovu konkretnih klasa
- Slično, možemo reći da su metode `racunajPovrsinu` i `racunajObim` u klasi `Oblik` **apstraktne metode** zato što nikada ne treba da se pozivaju – **sav njihov posao u stvari rade istoimene metode u podklasama!**
- Njihova **jedina uloga** je da **kažu računaru da svi oblici znaju da računaju svoju površinu i obim!**

Apstraktne klase

- Klasa `Oblik` i njeni metodi `racunajPovrsinu` i `racunajObim` su i u Primeru 3.1 apstraktni na semantičkom nivou
- Kako bi i sintaksno postali apstraktni, neophodno je dodati ključnu reč **abstract** ispred njihove definicije
- Kod apstraktnih metoda se blok naredbi koje čine implementaciju zamenjuje samo sa ;
- U svakoj od konkretnih podklasa mora biti realizovana implementacija za sve apstraktne metode, kako bi mogao biti kreiran objekat odgovarajuće klase
- Čim se klasa i sintaksno proglasi apstraktnom, više nije moguće kreirati objekte na osnovu nje

Primer 3.3 – apstraktna klasa Oblik

```
public abstract class Oblik {  
    ...  
    public abstract double racunajPovrsinu();  
  
    public abstract double racunajObim();  
    ...  
}
```

Objekat klase Oblik ne može se više kreirati u glavnom programu – sistem prijavljuje sintaksnu grešku!

```
Oblik o = new Oblik("Plava", "Krug", 1);  
//greska – Cannot instantiate the type Oblik  
//sto je i zeljeno ponasanje jer je klasa Oblik  
//apstraktna i na osnovu nje se ne mogu praviti  
//objekti jer ona samo izrazava zajednicka svojstva!
```

Primer 3.3 – apstraktna klasa Oblik

```
public abstract class Oblik {
    ...

    ...

    public abstract double racunajObim();

    public abstract double racunajPovrsinu();

    public void stampajPodatke() {
        System.out.println(    pribaviBoju() + " " +
                               pribaviTip() + " " +
                               pribaviBrojStrana() + " " +
                               racunajPovrsinu()+ " "+
                               racunajObim());
    }
}
```

Primer 3.3 – klasa Krug

```
public class Krug extends Oblik {
    private double poluprecnik;

    Krug(String boja, int brojStrana, double poluprecnik){
        super(boja, "Krug", brojStrana);
        this.poluprecnik = poluprecnik;
    }

    public double racunajPovrsinu() {
        return this.poluprecnik*this.poluprecnik*Math.PI;
    }

    public double racunajObim() {
        return 2*this.poluprecnik*Math.PI;
    }
}
```

Primer 3.3 – klasa Kvadrat

```
public class Kvadrat extends Oblik {  
    private double stranica;  
  
    Kvadrat(String boja, int brojStrana, double stranica){  
        super(boja, "Kvadrat", brojStrana);  
        this.stranica = stranica;  
    }  
  
    public double racunajPovrsinu() {  
        return this.stranica*this.stranica;  
    }  
  
    public double racunajObim() {  
        return 4*this.stranica;  
    }  
}
```


Primer 3.3 – klasa Main

```
public class Main {  
    public static void main(String[] args) {  
        //Oblik o = new Oblik("Plava", "Krug", 1);  
        //greska - Cannot instantiate the type Oblik  
  
        Krug kr = new Krug("Crvena",1, 2.0);  
        Kvadrat kv = new Kvadrat("Bela",4, 1.5);  
  
        kr.stampajPodatke();  
        kv.stampajPodatke();  
    }  
}
```

Interfejs i Java interfajs

- Važno je praviti **razliku** između **pojma interfejsa**, koji se odnosi na **OOP uopšte** i na **javne interfejse klasa**, i **pojma Java interfejsa**, koji je **specifična jezička konstrukcija** u programskom jeziku Java
- **Interfejs potprograma** sastoji se od njegovog imena, povratnog tipa i broja i tipova njegovih parametara. Ovo su podaci neophodni za poziv potprograma. **Interfejs klase** sastoji se od njenih javno dostupnih atributa i metoda
- **Java interface** je rezervisana reč sa dodatnim tehničkim značenjem. **Interfejs se u ovom smislu sastoji od skupa interfejsa metoda instanci, bez pridruženih implementacija**

Interfejsi

- **Java ne dozvoljava “klasično” višestruko nasleđivanje**, pa Java interfejsi otklanjaju u izvesnoj meri ovo ograničenje
- **Iako u Javi klasa može naslediti samo jednu klasu, ona može implementirati više interfejsa**
- Klasa implementira interfejs pružajući implementacije svih metoda koji su specificirani interfejsom
- Kako bi implementirala interfejs, klasa mora izjaviti da implementira neki interfejs koristeći ključnu reč **implements** npr.

public class Krug **implements** Figura

- Interfejsi su **vrlo korisni programerima klijentskih aplikacija** jer omogućavaju **polimorfizam** (isti kod može da radi sa različitim tipovima objekata)

Primer – interfejs Figura

```
public interface Figura {  
    public double racunajPovrsinu();  
    public double racunajObim();  
}
```

Klasa koja implementira interfejs Figura mora obezbediti implementacije svih metoda ovog interfejsa. Naravno, ta klasa pored implementacije ovih metoda može sadržati i druge attribute i metode. Ona može naslediti jednu klasu, ali implementirati više interfejsa, kao npr.:

```
public class Kvadrat extends Oblik  
    implements Figura, Element {
```

Interfejsi i nasleđivanje

Nasleđivanje daje *jeste* relaciju *i deljenje koda*

- Zaposleni ili Student mogu da se tretiraju kao objekti klase Osoba i nasleđuju njen kod
- Automobil ili Kamion ili Motocikl mogu da se tretiraju kao objekti klase Vozilo i nasleđuju njen kod

Interfejs daje *jeste* relaciju *bez deljenja koda*

- Kvadrat može da se tretira kao Figura, ali ne nasleđuje nikakav kod.

Ponekad se **nasleđivanje** označava kao **nasleđivanje implementacije**, a **interfejs** kao **nasleđivanje definicije**

Interfejsi i apstraktne klase

- Iako **interfejsi nisu klase**, oni su im na određeni način **slični**
- Interfejs vrlo **podseća na apstraktnu klasu**, tj. klasu koja se nikada ne koristi za pravljenje objekata, ali služi kao osnova za pravljenje podklasa
- Metode u interfejsu su apstraktne i moraju biti implementirane u svakoj konkretnoj klasi koja implementira dati interfejs
- Glavna razlika je u tome **što klasa koja nasleđuje neku apstraktnu klasu ne može naslediti i neku drugu klasu**, dok **klasa koje implementira interfejs može da nasledi neku klasu, ali i da implementira još neki interfejs**
- Apstraktna klasa, pored apstraktnih, može sadržati i ne-apstraktne metode
- **Interfejs je "čista" apstraktna klasa jer sadrži samo apstraktne metode**

Interfejsi i pristupni atributi

- Svi metodi interfejsa moraju biti **public**, šta više to se podrazumeva i ne mora se uopšte navoditi
- Interfejs može sadržati i deklaracije promenljivih. Te promenljive moraju biti **public static final** u interfejsu, kao i u svim klasama koje implementiraju taj interfejs. Ovi modifikatori su jedina moguća opcija u interfejsima, pa se takođe podrazumevaju i njihovo navođenje je opciono.
- Primer:

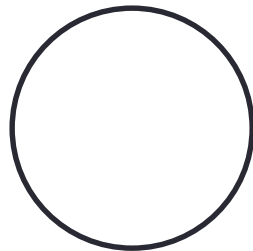
```
public interface KonverzioniFaktori {  
    int METARA_U_MILJI = 1609;  
    double KILOGRAMA_U_FUNTI = 0.45;  
}
```

Interfejs kao ugovor

- Interfejs se može posmatrati i kao **ugovor**, tj. analogno ne-programerskoj ideji uloga, protokola ili sertifikata:
 - *„Ja sam sertifikovan kao figura, zato što implementiram interfejs Figura. Ovim garantujem da znam kako da izračunam moj obim i površinu.“*
- Za definisanje ugovora važi jednostavno **pravilo da treba implementirati sve metode iz interfejsa u klasi koja sačinjava ugovor sa njim**
- Jedna od prednosti ugovora je **standardizovanje pravila programiranja kroz kreiranje komunikacionog protokola između klasa**
- **Interfejsi kao ugovori “postaju” pristupne tačke programskog koda**

Interfejsi – primer 3.4

- Realizovati interfejs `Figura` sa metodama za računanje površine i obima.
- Izmeniti klase `Kvadrat` i `Krug` tako da implementiraju interfejs `Figura` i nasleđuju klasu `Oblik`.
- Klase testirati kreiranjem više objekata u glavnom programu i pozivanjem metoda za računanje površine i obima i štampu podataka.



Interfejsi – primer 3.4

- Kako sada treba da izgledaju deklaracije klasa Kvadrat i Krug?

```
public class Kvadrat extends Oblik implements Figura {  
    ...  
}  
public class Krug extends Oblik implements Figura {  
    ...  
}
```

- Šta bi se desilo kada bi iz deklaracije ovih klasa izostavili klasu Oblik?

Interfejs daje *jeste* relaciju *bez deljenja koda*!

Promenljive i interfejsi – primer 3.4

- Može se napraviti promenljiva tipa interfejsa, ali se ne može kreirati objekat, već ova promenljiva može da referencira na objekat klase koji implementira interfejs

```
public class Main {  
    public static void main(String[] args) {  
        Figura f1, f2;  
  
        //f1 = new Figura();      //nije dozvoljeno!  
        f1 = new Krug("Crvena", "Krug", 1, 2.0);  
        f2 = new Kvadrat("Bela", "Kvadrat", 4, 1.5);  
  
        f1.racunajPovrsinu(); //mogu se pozivati  
        f2.racunajPovrsinu(); //samo metode iz interfejsa  
        ...  
    }  
}
```


RADNO OKRUŽENJE I API

Radna okruženja i ponovna upotreba koda

- Cilj softverskog inženjerstva je razvoj **robustnog i ponovo upotrebljivog softvera - višekratna upotrebljivost koda** se može **ostvariti** kroz **standardizaciju – koncept “prikluči i radi”**
- Koncept **radnog okruženja** (engl. **framework**) se zasniva na principima “prikluči i radi” i višekratne upotrebljivosti
- Primeri: Microsoft Office – Word, Excel, PowerPoint imaju većinu zajedničkih stavki u menijima (File, Edit, View, Format...), Windows – rad sa prozorima – radno okruženje sadrži gotove elemente, **ne treba stalno ponovo izmišljati točak!**
- **Programer koristi radno okruženje za pravljenje aplikacija tako što upotrebljava gotove interfejse**

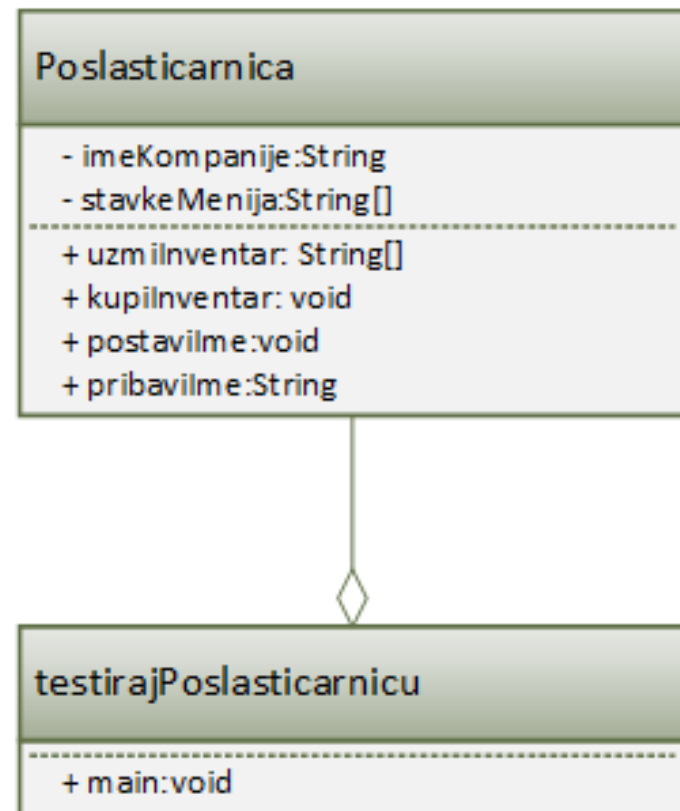
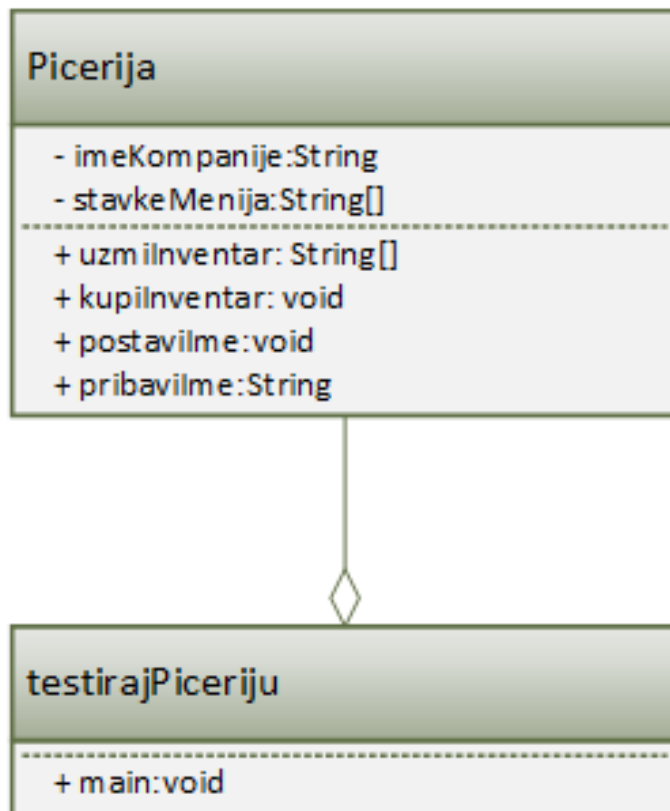
Radna okruženja i API

- **Prednosti: osećaj usaglašenosti i jednoobraznosti, programer može koristiti kod koji je već napisan i testiran**
- Kako se koriste gotovi okviri za dijalog iz radnog okruženja? Postoje pravila koje postavlja radno okruženje, obično organizovana u dokumentaciji napisanoj od strane kreatora okruženja, što dovodi do pojma **programskog interfejsa aplikacije** (engl. **Application Programming Interface – API**)
- Korišćenjem API-ja, pravite “važeće” aplikacije koristeći gotove elemente radnog okruženja i time se prilagođavate standardima – primer Java API, applet i web pretraživači

Primer 3.5 – elektronsko poslovanje

- Razvoj veb sajta picerije koji omogućava on-line narudžbine
- Cilj nam je da napravimo **radno okruženje koje bi uvelo višekratnu upotrebljivost koda u praksi**
- Treba razviti **funkcionalno radno okruženje** upotrebom **nasleđivanja, spajanja, apstraktnih klasa i interfejsa**
- Šta ako sutradan dobijemo zahtev za razvoj veb sajta poslastičarnice? Da li ćemo pisati novu aplikaciju? Koliko još porodičnih radnji može koristiti naše okruženje na vebu?
- Ako imamo **dobro i pouzdano radno okruženje, programe** bi mogli da nudimo po **povoljnijim cenama**, a pri tom bi oni bili **već isprobani i primenjeni**, što **smanjuje posao oko održavanja i otklanjanja grešaka**

Primer 3.5 – pristup bez ponovne upotrebe koda

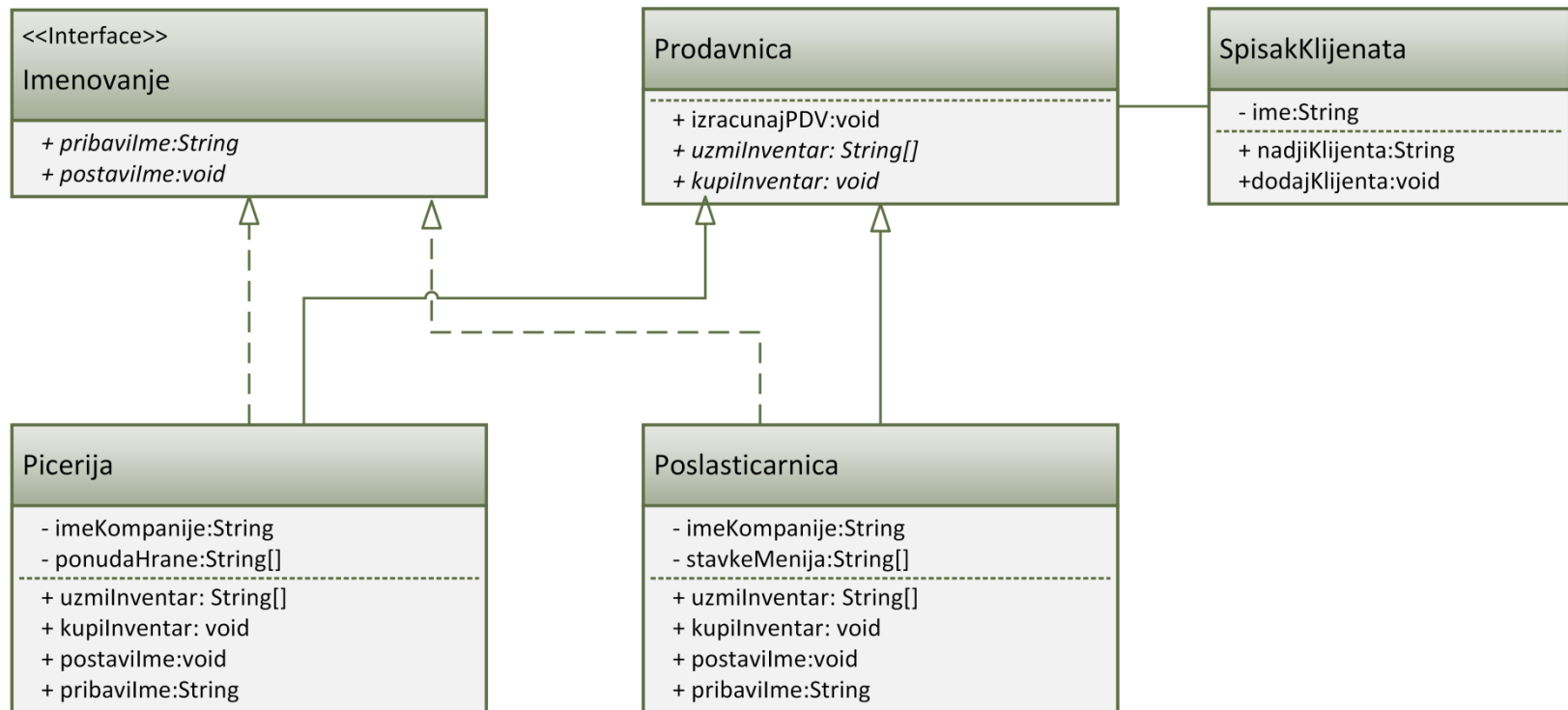


Primer 3.5 – rešenje za elektronsko poslovanje

- Napravićemo **apstraktnu klasu** koja **izdvaja određenu realizaciju i interfejs koji modeluje neka ponašanja**
- **Cilj – radno okruženje kojim višekratno koristimo kod**
- Svaka specifična aplikacija biće **vezana samo ugovorom**, nema njenog strogog vezivanja za prilagođene klase
- Sastoji se od:
 - Interfejsa **Imenovanje**, modeluje ponašanja, deo ugovora,
 - Apstraktne klase **Prodavnica**, izdvaja implementaciju, deo ugovora,
 - Klase **SpisakKlijenata**, koju koristimo kroz spajanje i
 - Nove realizacije klase **Prodavnica** za svakog klijenta kroz klase potomke koji je nasleđuju

Primer 3.5 – diagram klasa

Apstraktni metodi se obeležavaju *italic* slovima



-----> realizacija interfejsa

Primer 3.5 – apstraktna klasa Prodavnica

```
public abstract class Prodavnica {  
    private SpisakKlijenata spisakKlijenata;  
  
    public void izracunajPDV() {  
        System.out.println("Stopa PDV je 20%!");  
    }  
  
    public abstract String[] uzmiInventar();  
  
    public abstract void kupiInventar(String artikal);  
}
```

Primer 3.5 – klasa SpisakKlijenata

```
public class SpisakKlijenata {  
    private String[] ime;  
    private int trenutniBrojKlijenata;  
    private int maxBrojKlijenata;  
  
    SpisakKlijenata(){}  
  
    SpisakKlijenata(int maxBrojKlijenata){  
        this.maxBrojKlijenata = maxBrojKlijenata;  
        this.trenutniBrojKlijenata = 0;  
        this.ime = new String[maxBrojKlijenata];  
    }  
  
    public String nadjiKlijenta(String ime) {  
        for (int i = 0; i < this.trenutniBrojKlijenata; i++) {  
            if (this.ime[i].equals(ime)) {  
                return this.ime[i];  
            }  
        }  
        return ("Klijent nije pronadjen!");  
    }  
    ...  
}
```

Primer 3.5 – klasa SpisakKlijenata

• • •

```
public void dodajKlijenta(String ime) {
    if (this.trenutniBrojKlijenata < this.maxBrojKlijenata) {
        this.ime[this.trenutniBrojKlijenata++] = ime;
    }
    else{
        System.out.println("Nema vise mesta u spisku
                           klijenata!");
    }
}
}
```

Primer 3.5 – interfejs Imenovanje

```
public interface Imenovanje {  
  
    String pribaviIme();  
    void postaviIme(String ime);  
  
}
```

Primer 3.5 – klasa Picerija

```
public class Picerija extends Prodavnica implements Imenovanje {  
  
    private String imeKompanije;  
  
    private String[] ponudaHrane = {  
        "Pica",  
        "Pasta",  
        "Salata",  
        "Kalcona",  
        "Sok",  
        "Pivo"  
    };  
  
    public String[] uzmiInventar() {  
        return ponudaHrane;  
    }  
  
    ...  
}
```


Primer 3.5 – klasa Picerija

...

```
public void kupiInventar(String artikal) {  
    System.out.println("\nUpravo ste narucili artikal “  
                        + artikal);  
}
```

```
public String pribaviIme() {  
    return imeKompanije;  
}
```

```
public void postaviIme(String ime) {  
    imeKompanije = ime;  
}
```

```
}
```

Primer 3.5 – klasa Poslasticarnica

```
public class Poslasticarnica extends Prodavnica implements Imenovanje{

    private String imeKompanije;

    private String[] stavkaMenija = {
        "Sladoled",
        "Torta",
        "Krofna",
        "Kafa",
        "Caj",
        "Limunada"
    };

    public String[] uzmiInventar() {
        return stavkaMenija;
    }

    ...
}
```

Primer 3.5 – klasa Poslasticarnica

...

```
public void kupiInventar(String artikal) {  
    System.out.println("\nUpravo ste narucili artikal “  
                        + artikal);  
}
```

```
public String pribaviIme() {  
    return imeKompanije;  
}
```

```
public void postaviIme(String ime) {  
    imeKompanije = ime;  
}
```

```
}
```

Primer 3.5 – klasa Main

```
public class Main {  
    public static void main(String[] args) {  
  
        Poslasticarnica carigrad = new Poslasticarnica();  
        Picerija ciao = new Picerija();  
  
        carigrad.postaviIme("Evropa");  
        ciao.postaviIme("Ciao");  
  
        carigrad.kupiInventar("Sladoled");  
        ciao.kupiInventar("Pica");  
        ...  
    }  
}
```

Zadatak za rad na času

- Zadatak: dopuniti apstraktnu klasu Prodavnica, interfejs Imenovanje, klasu SpisakKlijenata, kao i izvedene klase Picerija i Poslasticarnica novim atributima i metodima i proširiti implementacije postojećih metoda, tako da se realizuju funkcionalnosti koje bi se mogle zahtevati u svakodnevnom poslovanju
- Realizovati nove klase Restoran i Knjizara koje takođe nasleđuju apstraktnu klasu Prodavnica i implementiraju interfejs Imenovanje
- Klase testirati kreiranjem više objekata u glavnom programu i pozivanjem odgovarajućih izabranih metoda

TAČNOST I ROBUSNOST OO PROGRAMA – GENERISANJE I OBRADA IZUZETAKA

Tačnost i robusnost

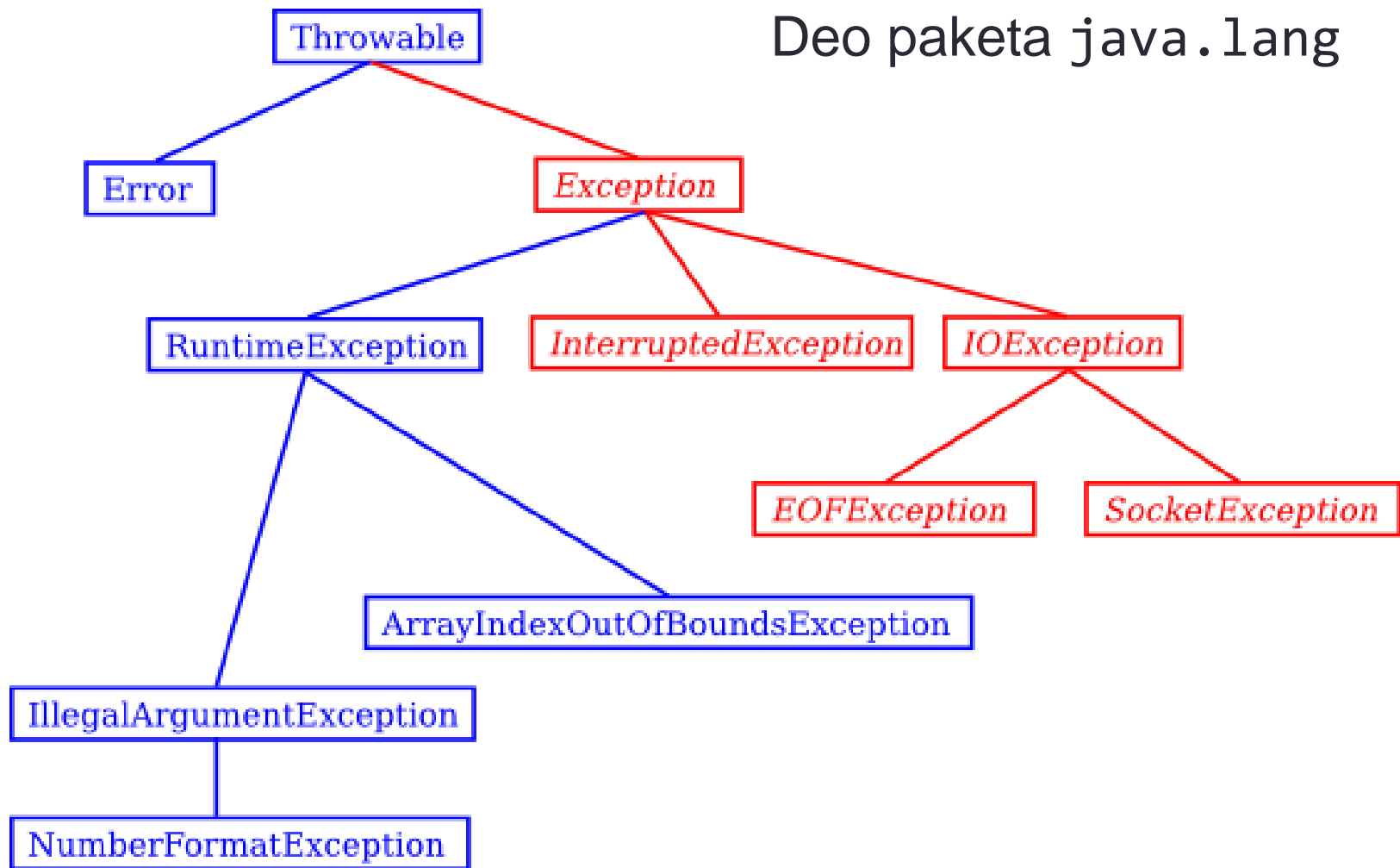
- Osnovna filozofija Jave je da se **loše napisan kod neće izvršavati!**
- Program je **tačan** ako uspešno realizuje zadatak za koji je projektovan
- Program je **robustan** ako je u stanju da reaguje na neočekivane situacije (kao što su nevalidni ulazni podaci) na razuman način
- Da bismo napravili **robustan sistem**, **svaka njegova komponenta mora biti robusna**

Postupanje sa greškama

- Uz pretpostavku da je kod sposoban da uoči i pronade mesto gde se desila greška, tada se sa njom može postupati na nekoliko načina:
 1. Ignorisanje greške – nije dobar način!
 2. Provera pojave mogućih problema i prekid programa kada se pojavi problem
 3. Provera pojave mogućih problema, hvatanje greške i pokušaj da se problem reši – pokušaj “oporavka”
 4. Obrada greške pomoću izuzetaka – “bacanje” izuzetaka – poželjan način!

Throwable i neke njene podklase

Deo paketa `java.lang`



Obrada grešaka pomoću izuzetaka

- Izuzeci pružaju način za otkrivanje grešaka i njihovu obradu
- Za hvatanje i obradu izuzetaka postoji poseban blok
- Sintaksa bloka za hvatanje i obradu izuzetaka:

```
try {  
    //ovde ide potencijalno problematican kod  
} catch (Exception e) {  
    //kod koji obradjuje izuzetak  
}
```

- Ako je unutar try bloka generisan izuzetak, blok catch će ga obraditi.

Obrada grešaka pomoću izuzetaka

- Opšti oblik:

```
try {  
    //problematican kod  
} catch (Exception e) {  
    //kod koji obradjuje izuzetak  
    System.out.println(e.getMessage());  
}  
System.out.println("Izuzetak je obradjen!");
```

Primer 4.1 – hvatanje i obrada izuzetka

- Determinanta matrice jednaka je razlici proizvoda elemenata na glavnoj i sporednoj dijagonali

```
try {  
    double determinanta = M[0][0]*M[1][1] - M[0][1]*M[1][0];  
    ...  
    System.out.println("Determinanta M je " + determinanta);  
}  
catch (ArrayIndexOutOfBoundsException e) {  
    System.out.println("M nije ispravne velicine!");  
}  
catch (NullPointerException e) {  
    System.out.print("Programska greska! M ne postoji!");  
}
```

- Ako se u try bloku generiše izuzetak, tada se on hvata u odgovarajućem catch bloku

Obrada grešaka pomoću izuzetaka

- Ako se izuzetak pojavi tokom izvršavanja bloka, dešava se sledeće:
 - Izvršavanje bloka se prekida
 - Proveravaju se uslovi iz klauzula `catch` (može ih biti više), da bi se utvrdilo da li za taj izuzetak postoji odgovarajući `catch` blok
 - Ako se nijedan od `catch` blokova ne tiče tog izuzetka, onda se on prosleđuje `try` bloku višeg nivoa – ako se izuzetak ne uhvati u kodu biće uhvaćen od strane sistema sa nepredvidivim ishodom!
 - Ako se pronađe odgovarajući `catch` blok, izvršavaju se naredbe iz tog bloka
 - Potom se nastavlja sa izvršavanjem programa počev od prve naredbe iza bloka `try`

Opšti oblik try – catch bloka

```
try {  
    // kod koji može da izazove izuzetak  
} catch(<tip1> <ime1>){  
    // kod kojim se obrađuje izuzetak 1. tipa  
}  
} catch(<tip2> <ime1>){  
    // kod kojim se obrađuje izuzetak 2. tipa  
...[finally {blok}] // tipično ovde idu  
                        // aktivnosti koje se uvek  
                        // izvršavaju, bez obzira  
                        // da li se desio izuzetak
```

Čišćenje pomoću **finally**

- Često postoje delovi koda koje želimo da izvršimo bez obzira na to da li je prethodno nastao izuzetak ili ne
- To je obično slučaj u operacijama koje ne predstavljaju oporavak memorije (npr. uvek želimo da zatvorimo prethodno otvorenu datoteku)
- Dakle, u `finally` blok idu aktivnosti koje se uvek dešavaju
- Pošto Java ima sakupljač smeća, ovo naredba je neophodna kada u prvobitno stanje moramo da vratimo nešto što nije memorija – primeri: zatvaranje datoteke, raskid veze sa mrežom
- Blok `finally` posmatraćemo nešto kasnije u okviru primera za rad sa datotekama

Specifikacija izuzetka

- U Javi ste dužni da u specifikaciji metode navedete koje izuzetke ona može da generiše
- Za tu svrhu je rezervisana reč **throws**, npr.:

```
void f() throws ArithmeticException, IOException {  
    //kod metode koja moze da izazove izuzetke  
}
```

- Ako se u specifikaciji ne navede **throws**, podrazumeva se da metoda ne generiše izuzetke
- Hvatanje bilo kog tipa izuzetka vrši se pomoću osnovne klase Exception – **catch** (Exception e) { ...

Generisanje izuzetka bez obrade

- Ima situacija kada je smisleno generisati izuzetak bez njegovog hvatanja i obrade
- Kada program otkrije neko stanje greške, ali nema razumnog načina za obradu greške, tada program može generisati izuzetak u nadi da će neki drugi deo programa da ga uhvati i obradi. U ove svrhe je rezervisan ključna reč **throw**
- Izuzeci mogu da se generišu pomoću uslova i **if** bloka u kome se, ako je ispunjen uslov, generiše izuzetak

Primer 4.2 – generisanje izuzetka

```
/**
 * Vraca veci od dva korena kvadratne jednacine
 *  $A*x*x + B*x + C = 0$ , ako ona ima korena. Ako je  $A == 0$  ili
 * je diskriminanta  $B*B - 4*A*C$  negativna onda se generise
 * izuzetak tipa IllegalArgumentException.
 */
static public double koren( double A, double B, double C )
    throws IllegalArgumentException {
    if (A == 0) {
        throw new IllegalArgumentException("A ne moze biti nula!");
    }
    else {
        double disk = B*B - 4*A*C;
        if (disk < 0)
            throw new IllegalArgumentException("Diskriminanta
                                                manja od nule!");
        return (-B + Math.sqrt(disk)) / (2*A);
    }
}
```

Uputstvo za korišćenje izuzetaka

- Izuzetke koristimo da:
 - Rešimo probleme i ponovo pozovemo metodu koja je prouzrokovala izuzetak
 - “Zakrpimo” grešku i nastavimo rad bez ponovnog isprobavanja metode
 - Pojednostavimo kod (ako način na koji generišemo izuzetke još više komplikuje kod, onda ga nije lako i poželjno koristiti)
 - Završimo program
 - Povećamo pouzdanost biblioteke i programa – kratkoročno ulaganje napora u pisanje koda za otklanjanje grešaka je dugoročna investicija u snagu i stabilnost aplikacije

Pretpostavke (engl. assertions)

- Pretpostavke (engl. assertions) obezbeđuju da je ispunjen neki preduslov kako bi se omogućilo dalje izvršavanje programa
- Koristi se rezervisana reč **assert**
- Oblici naredbe:
 - assert** uslov;
 - assert** uslov : poruka_o_gresci;
- Uključivanje pretpostavki u Eclipse-u: – Run As – Run Configurations... – Arguments tab – VM arguments – “-ea”
- Primer:
assert (fakt==1) : "Faktorijel nije inicijalizovan na 1!";

Primer 4.3 - pretpostavke

```
/**
 * Vraca veci od dva korena kvadratne jednacine
 *  $A*x*x + B*x + C = 0$ , ako ona ima korena.
 * Preduslovi:  $A \neq 0$  i  $B*B - 4*A*C > 0$ 
 */
static public double koren( double A, double B, double C ) {
    assert A != 0 : "Vodeci koeficijent kvadratne jednacine
                      ne sme biti nula!";
    double disk = B*B - 4*A*C;
    assert disk >= 0 : "Diskriminanta kvadratne jednacine
                      ne sme biti negativna!";
    return (-B + Math.sqrt(disk)) / (2*A);
}
```

Beleške (engl. annotations)

- Beleške (engl. annotations) su metapodaci (podaci o podacima)
- Postaje od Java 5 – beleške proverava kompajler kako bi osigurao da je kod u skladu sa namerama programera
- Primeri:
 - `@Override`
 - `@Deprecated`
 - `@SuppressWarnings`
- U kodu se koriste vrlo slično kao `static`, `final` i sl. – ako se napiše npr. `@Override` u definiciji nekog metoda, onda bi on trebalo da redefiniše istoimenu metodu iz neke nadklase – ako ovakva metoda ne postoji kompajler prijavljuje grešku!

RAD SA TOKOVIMA I DATOTEKAMA

Ulaz i izlaz programa

- Računarski programi su korisni samo ako na neki način interaguju sa ostatkom sveta – ova interakcija se naziva ulaz/izlaz (engl. input/output – I/O)
- U Javi, najčešće se koristi ulaz/izlaz koji uključuje fajlove i računarske mreže putem mehanizma tokova (engl. streams) – tokovi su objekti koji podržavaju I/O naredbe
- Standardni izlaz (**System.out**) i standardni ulaz (**System.in**) su primeri tokova
- Rad sa tokovima i datotekama u Javi zahteva poznavanje mehanizma obrade grešaka pomoću izuzetaka

Rad sa tokovima

- Kada radimo sa ulazom/izlazom, razlikujemo dve osnovne kategorije podataka:
 - mašinski-formatirani podaci – sastavljeni od bajtova i
 - tekst koji mogu da čitaju ljudi – sastavljen od znakova
- Tako u Javi postoje i dve osnovne vrste tokova:
 - **Tokovi bajtova (byte streams)** i
 - **Tokovi znakova (character streams)**
- Klase za rad sa tokovima su deo paketa `java.io` koji se mora uvesti na početku programa
- Tokovi su neophodni u Java programima za rad sa fajlovima i komunikaciju preko mreže

Standardni tokovi podataka u Javi

- Java uključuje tri standardna toka podataka:
 1. Standardni ulaz – tipično se koristi za učitavanje sa tastature preko `System.in`
 2. Standardni izlaz – tipično se koristi za ispis na ekran preko `System.out`
 3. Standardna greška – takođe se tipično vezuje za ekran, služi za ispis grešaka preko `System.err`

Rad sa tokovima

- Objekat koji upisuje podatke u **tok bajtova** pripada nekoj od podklasa apstraktne klase `OutputStream`, dok onaj koji čita iz toka bajtova pripada nekoj od podklasa apstraktne klase `InputStream`.
- Objekat koji upisuje podatke u **tok znakova** pripada nekoj od podklasa apstraktne klase `Writer`, dok onaj koji čita iz toka znakova pripada nekoj od podklasa apstraktne klase `Reader`.
- Tokovi bajtova su korisni u mašinskoj komunikaciji, kao i za efikasno čuvanje vrlo velike količine podataka, npr. u velikim bazama podataka, ali njima se mogu obrađivati i ASCII znakovi (pošto su veličine 1 B), ali ne i UNICODE karakteri za koje je neophodan tok karaktera!

Primer 4.3 – bajt tok U/I

- Program koji učitava bajtove iz ulaznog fajla i upisuje ih u izlazni fajl – koristimo za binarne podatke i ASCII

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

public class KopiranjeBajtova {
    public static void main(String[] args) throws IOException {
        FileInputStream ulaz = null;
        FileOutputStream izlaz = null;
        try {
            ulaz = new FileInputStream("ulazBajt.txt");
            izlaz = new FileOutputStream("izlazBajt.txt");
            int c;
            while ((c = ulaz.read()) != -1) {
                izlaz.write(c);
            }
        }
        ...
    }
}
```

Primer 4.3 – bajt tok U/I

- Program koji učitava bajtove iz ulaznog fajla i upisuje ih u izlazni fajl – koristimo za binarne podatke i ASCII

```
...
    catch (IOException e) {
        System.out.println(e.getMessage());
    }
    finally {
        if (ulaz != null) {
            ulaz.close();
        }
        if (izlaz != null) {
            izlaz.close();
        }
    }
}
```

Primer 4.4 – karakter tok U/I

- Program koji učitava bajtove iz ulaznog fajla i upisuje ih u izlazni fajl – koristiti za UNICODE karaktere

```
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

public class KopiranjeBajtova {
    public static void main(String[] args) throws IOException {
        FileReader ulaz = null;
        FileWriter izlaz = null;
        try {
            ulaz = new FileReader("ulazKarakter.txt");
            izlaz = new FileWriter("izlazKarakter.txt");
            int c;
            while ((c = ulaz.read()) != -1) {
                izlaz.write(c);
            }
        }
        ...
    }
}
```


Primer 4.4 – karakter tok U/I

- Program koji učitava bajtove iz ulaznog fajla i upisuje ih u izlazni fajl – binarni za UNICODE karaktere

```
...
catch (IOException e) {
    System.out.println(e.getMessage());
}
finally {
    if (ulaz != null) {
        ulaz.close();
    }
    if (izlaz != null) {
        izlaz.close();
    }
}
}
```

Baferovani tokovi podataka

- Baferovani tokovi podataka su posebno važni prilikom rada sa velikim fajlovima
- Baferovani ulazni tok čita podatke iz dela memorije poznatog kao bafer; nativni ulazni API se poziva samo kada je bafer prazan. Slično, baferovani izlazni tok upisuje podatke u bafer i nativni izlazni API se poziva samo kada je bafer pun
- Da bi se u prethodnom primeru koristio baferovani U/I potrebno je da se pozovu odgovarajući konstruktori:

```
ulaz = new BufferedReader(new FileReader("unos.txt"));  
izlaz = new BufferedWriter(new FileWriter("ispis.txt"));
```

- `BufferedInputStream` i `BufferedOutputStream` kreiraju baferovane tokove bajtova, dok `BufferedReader` i `BufferedWriter` kreiraju baferovane tokove karaktera

Klasa Scanner

- Klasa Scanner radi kao omotač oko izvora ulaznih podataka. Izvor može biti Reader, InputStream, String ili File
- Scanner radi sa tokenima (najkraći smisleni niz karaktera) i delimiterima
- Primer korišćenja delimitera:

```
String ulaz = "10 caj 20 kafa 30 vocni sok";  
Scanner s = new Scanner(ulaz).useDelimiter("\\s");  
System.out.println(s.nextInt());  
System.out.println(s.next());  
System.out.println(s.nextInt());  
System.out.println(s.next());  
s.close();
```

Izlaz:
10
caj
20
kafa

Primer 4.5 - klasa Scanner

- Primer:

```
import java.util.Scanner;
```

```
class SkenerTest {  
    public static void main(String args[]) {  
  
        Scanner sc = new Scanner(System.in);  
        System.out.println("Unesite Vas JMBG: ");  
        String jmbg = sc.next();  
        System.out.println("Unesite Vase ime: ");  
        String ime = sc.next();  
        System.out.println("Unesite Vasu platu: ");  
        double plata = sc.nextDouble();  
        System.out.println("JMBG:" + jmbg + " Ime:" + ime +  
                           " Plata:" + plata);  
  
        sc.close();  
    }  
}
```

Serijalizacija objekata prilikom U/I

- Serijalizacija objekata je postupak predstavljanja objekata kao sekvence podataka primitivnih tipova koji mogu postati elementi tokova bajtova ili karaktera. Prilikom ulaza, treba učitati serijalizovane podatke i na osnovu njih rekonstruisati kopiju originalnog objekta
- U Javi za ovu svrhu postoje gotove klase `ObjectInputStream` i `ObjectOutputStream`
- Metode za U/I rad sa objektima su `readObject()`, u `ObjectInputStream`, i `writeObject(Object obj)` u `ObjectOutputStream`. Ove metode mogu generisati `IOException`
- `ObjectInputStream` i `ObjectOutputStream` rade samo sa objektima klasa koje implementiraju interfejs `Serializable`

Zadatak za rad na času

- Napraviti paket zaposleni i u okviru njega implementirati:
 1. Apstraktnu klasu Radnik čiji su zaštićeni podaci: ime radnika, prezime radnika, JMBG, broj tekućeg računa i koeficijent stručne sprema, a javni: metod za učitavanje podataka o radniku iz tekstualne datoteke, metod za upis imena, prezimena, broja tekućeg računa i plate radnika (za zadatak vrednost cene rada) u jedan red tekstualne datoteke i apstraktni metod za izračunavanje plate radnika.
 2. klasu AktivanRadnik izvedenu iz apstraktne klase Radnik, koja kao privatni podatak sadrži varijabilni koeficijent (u skladu sa tim treba predefinisati i metod za učitavanje).
 3. klasu RadnikNaBolovanju takođe izvedenu iz klase Radnik.
U klasi Main kreirati platni spisak radnika jednog preduzeća na osnovu sadržaja ulazne datoteke *spisak.txt*.

Zadatak za rad na času

NAPOMENA: Platu aktivnog radnika računati po obrascu:

$$plata = (koefStrucneSpreme + varijabilniKoef) * cenaRada$$

a platu radnika na bolovanju po obrascu:

$$0.8 * koeficijentStrucneSpreme * cenaRada$$

U ulaznoj datoteci zapisana je najpre cena rada, zatim broj radnika u preduzeću, a zatim slede podaci o svim radnicima. Podaci o jednom radniku počinju linijom u kojoj je zapisan simbol + ili -. Simbol + označava da slede podaci o aktivnom radniku, a – da slede podaci o radniku koji se trenutno nalazi na bolovanju. Platni spisak ispisati u izlaznu datoteku *plate.txt*.

Napomena: za konverziju se mogu koristiti omotač klase za primitivne tipove iz `java.lang` kao što je `Integer`:

Primer za ulaz: `n = Integer.parseInt(bafUlaz.readLine());`

Primer za izlaz: `dat.write(new Double(plata).toString());`

Zadatak za vežbanje

- Napraviti paket matematika i u okviru njega implementirati:
 1. Interfejs Funkcija koji sadrži metode za učitavanje parametara funkcije iz tekstualne datoteke, izračunavanje vrednosti funkcije u zadatoj tački, ispitivanje da li funkcija ima realne nule i nalaženje nula funkcije.
 2. Klasu LinearnaFunkcija (za predstavljanje funkcija oblika $y=ax+b$) koja implementira interfejs Funkcija.
 3. Klasu KvadratnaFunkcija (za predstavljanje funkcija oblika $y=ax^2+bx+c$) koja takođe implementira interfejs Funkcija.

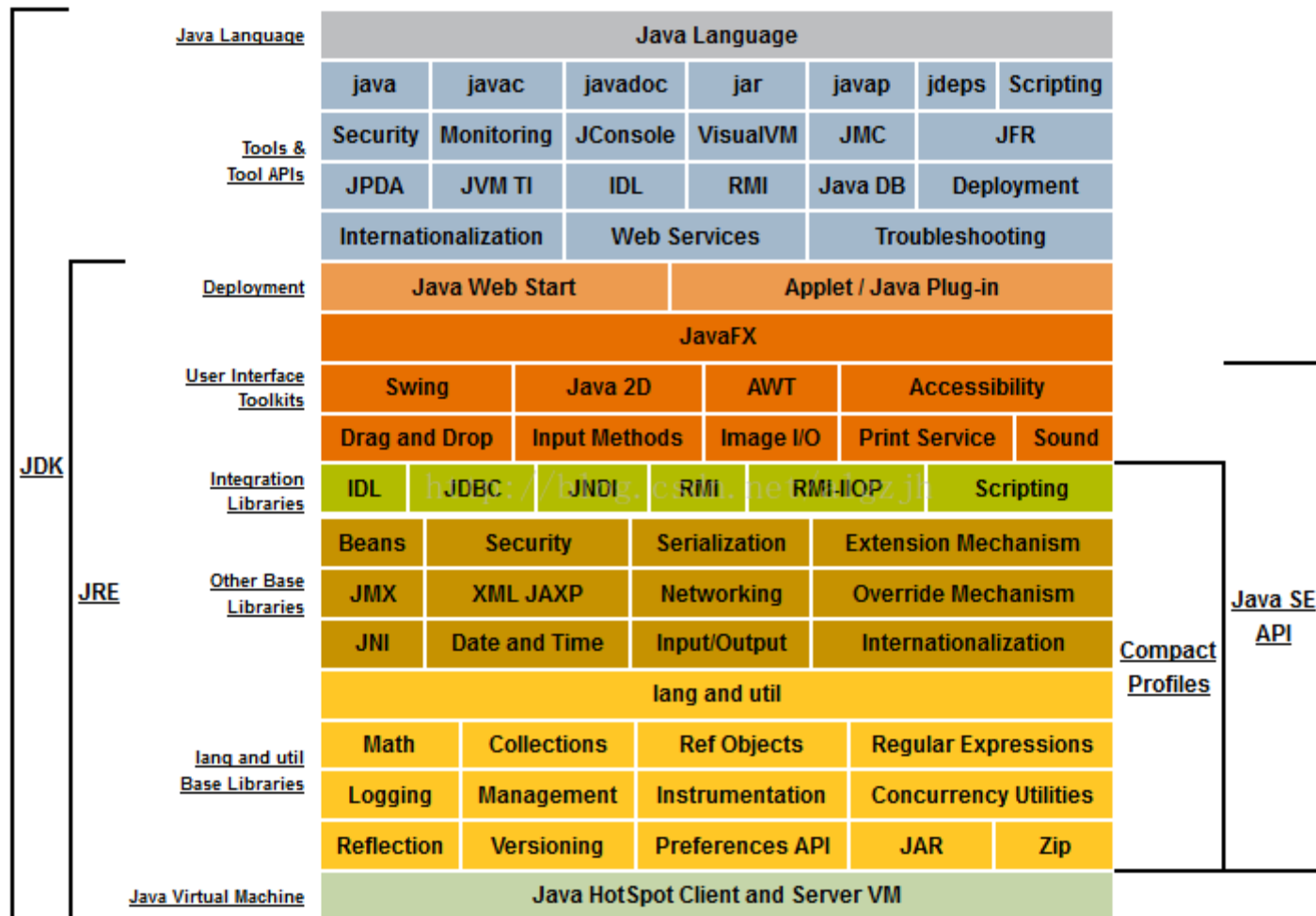
U klasi Main definisati dve promenljive tipa interfejsa Funkcija, jednoj dodeliti objekat tipa LinearnaFunkcija, drugoj tipa KvadratnaFunkcija. Parametre funkcija učitati iz datoteka *funkcija1.txt* i *funkcija2.txt*, respektivno, i štampati njihove nule (ukoliko postoje) u izlazne datoteke *nule1.txt* i *nule2.txt*, respektivno.

JAVA PLATFORMA

Specifičnosti OOP u Javi

- Sve klase izvedene su iz klase Object
- U Javi je sve objekat, osim promenljivih nekog od osam primitivnih tipova
- Nikad ne morate da uništite objekat - automatski sakupljač smeća (engl. garbage collector)
- Zabranjeno višestruko nasleđivanje – koriste se interfejsi
- Tip može biti klasa, interfejs ili neki od osam primitivnih. Ovo su jedine mogućnosti. Samo klase se mogu koristiti za kreiranje novih objekata
- Filozofija Jave – “napiši jednom, pokreni bilo gde” (“write once, run anywhere” - WORA)

Konceptualni diagram Java komponenti



Java archive - Java ARchive (JAR)

- JAR je fajl format koji se tipično koristi za agregiranje više Java .class fajlova i pridruženih metapodataka i resursa (tekst, slike, itd.) u jedan fajl radi distribucije
- JAR se zasniva na ZIP formatu, ima ekstenziju .jar
- JAR fajlovi omogućavaju da se efikasno dopremi i pokrene čitava aplikacija, uključujući sve prateće resurse, u jednom zahtevu – primene: web i mobilno programiranje
- Kreiranje .jar iz Eclipse: File ➡ Export ➡ Java ➡ JAR file ➡ ➡ izbor željenih klasa i resursa za uključivanje u JAR
- Sadržaj .jar fajla može biti raspakovan bilo kojim standardnim alatom za dekompresiju ili korišćenjem jar komandnog alata

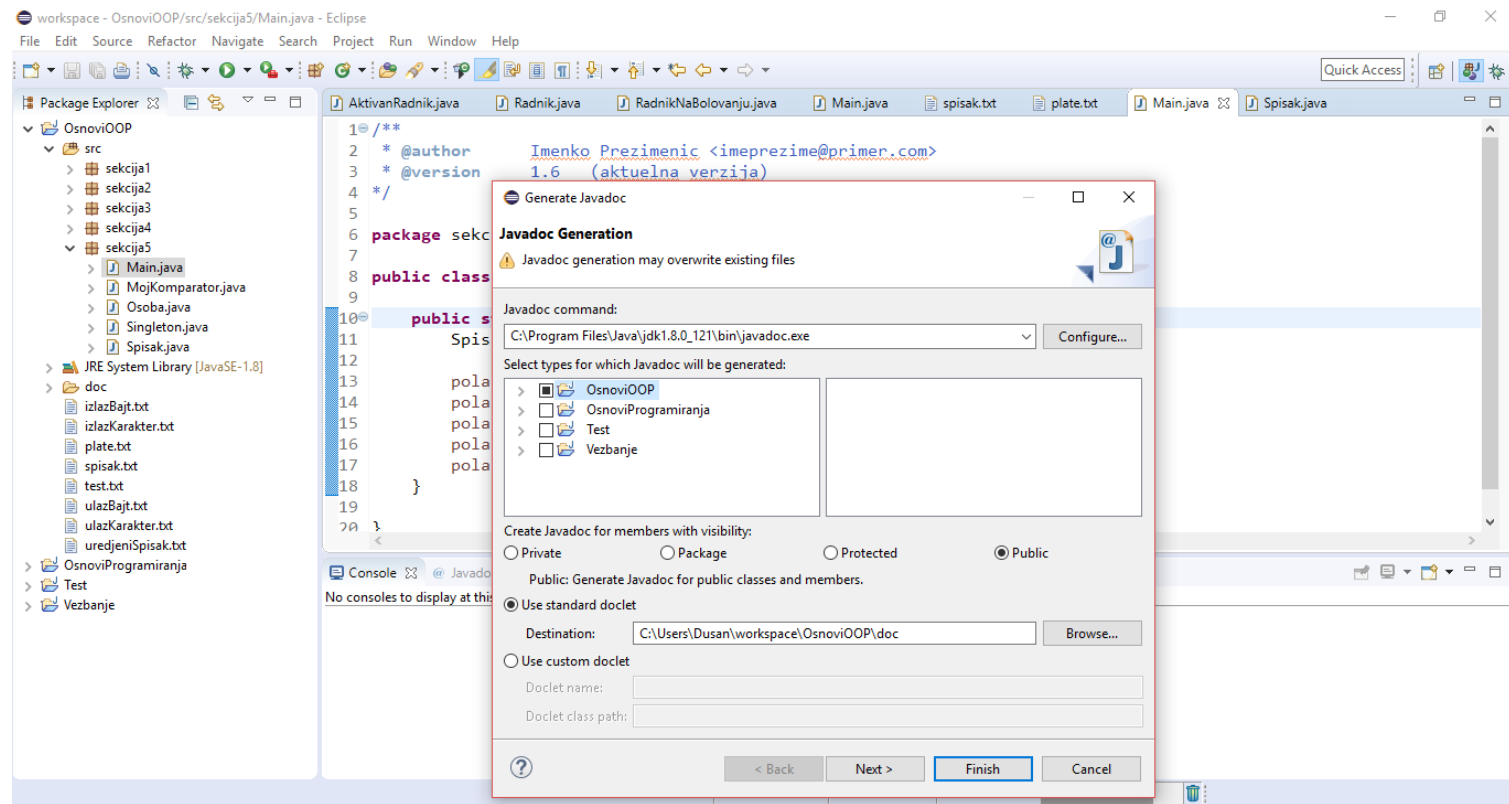
Javadoc

- Javadoc je generator dokumentacije koji služi za generisanje API dokumentacije u HTML obliku direktno iz fajlova sa Java izvornim kodom
- Komentari oblika `/** ... */`
- Koristi i tagove - `@author`, `@version`, `@param...`
- De facto industrijski standard za dokumentovanje u Javi
- Primer korišćenja:

```
/**
 * @author      Imenko Prezimenic <imeprezime@primer.com>
 * @version     1.6      (aktuelna verzija)
 */
public class Test {
    // telo klase
}
```

Javadoc

- Kod metoda postaviti `@param` i `@return`
- Pokretanje: Project ➔ Generate Javadoc (prethodno podesiti putanju do javadoc.exe)



Java platforma

Java biblioteka klasa može se podeliti u dve osnovne grupe paketa:

1. Prvu grupu čine standardni paketi sa klasama neophodnim za programiranje u Javi

Primeri:

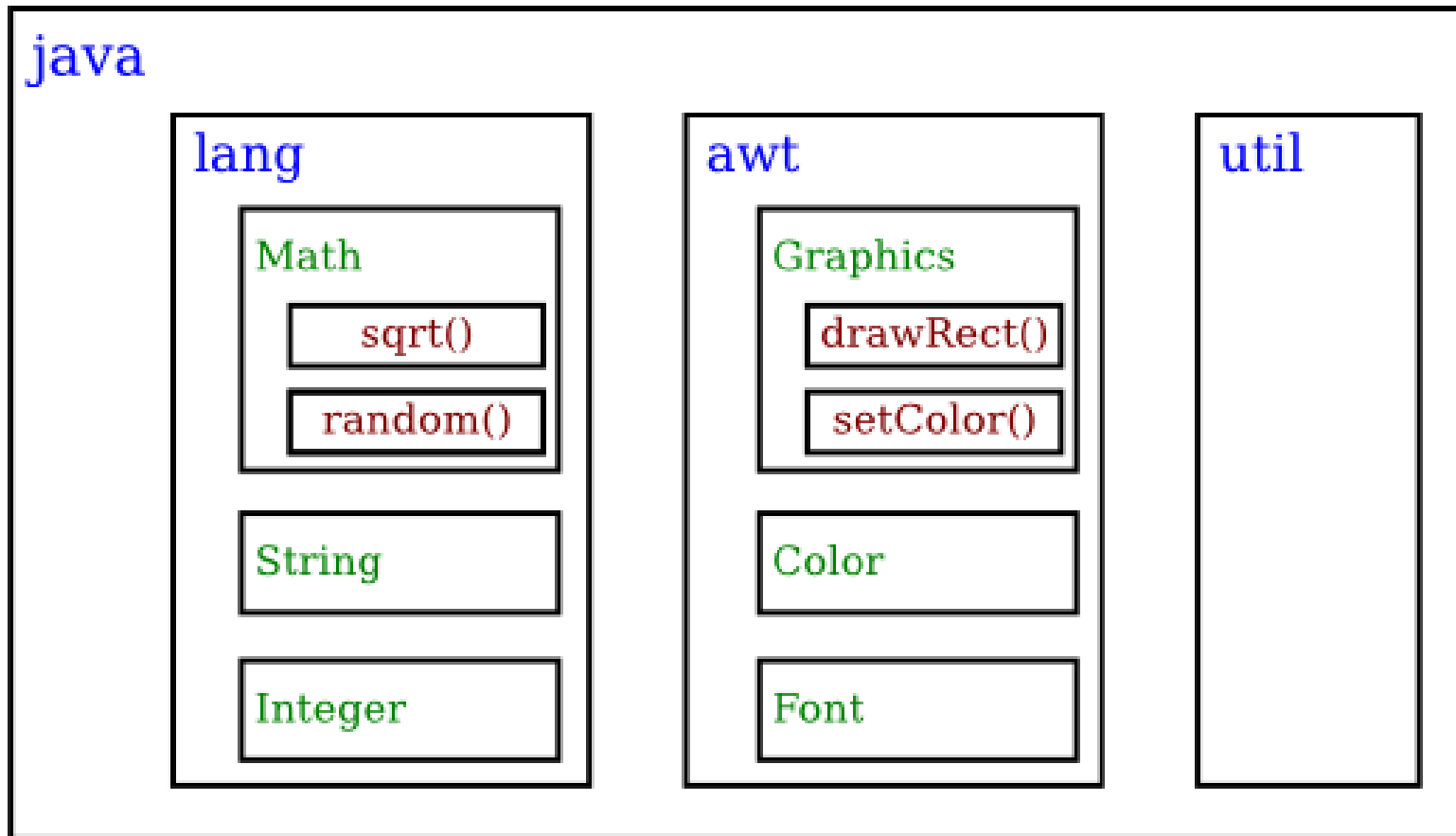
```
java.lang  
java.io  
java.util
```

2. Drugu grupu čine dodatni paketi sa klasama za kreiranje apleta, rad sa mrežom itd.

Primeri:

```
java.applet  
java.net  
...
```

Java platforma – standardni paketi



Metode ugnježdene u **klase** ugnježdene u dva sloja **paketa**.
Puno ime metode `sqrt()` je `java.lang.Math.sqrt()`.

Java platforma – `java.lang`

- Paket `java.lang` sadrži osnovne interfejse i klase koji su neophodni za programiranje u Javi. Ovde spadaju hijerarhija klasa, tipovi koji su deo definicije jezika, osnovni izuzetci, matematičke funkcije itd. Klase iz ovog paketa su **automatski uključene** u svaki Java izvorni fajl.
- Najvažnije klase u `java.lang` su:
 - `Object` - korenska klasa svih klasa
 - `System` – klasa koja pruža sistemske operacije
 - `Math` – klasa sa osnovnim matematičkim funkcijama
 - `Throwable`, `Exception`, `Error` – klase za rad sa greškama i izuzecima
 - `String` – klasa za rad sa stringovima
 - `Character`, `Integer`, `Float`... – omotač (engl. *wrapper*) klase za primitivne tipove

Java niti – `java.lang`

- Java ima odličnu podršku za multiprocessing i rad sa nitima koji su veoma važni na savremenim računarima
- Niti (engl. thread) se predstavljaju objektom koji pripada klasi `java.lang.Thread` (ili nekoj podklasi ove klase) ili objektom klase koja implementira interfejs `java.lang.Runnable`
- Svrha objekta `Thread` je da samo jednom izvrši neki metod. Ovaj metod predstavlja zadatak koji nit treba da izvrši. Više niti može da se izvršava paralelno
- Niti se mogu programirati tako što se kreira klasa izvedena iz klase `Thread` ili klasa koja implementira interfejs `Runnable` i u njoj definiše metod `public void run()`. Implementacija ovog metoda definiše zadatak koji će nit izvršavati

Java platforma – java.io

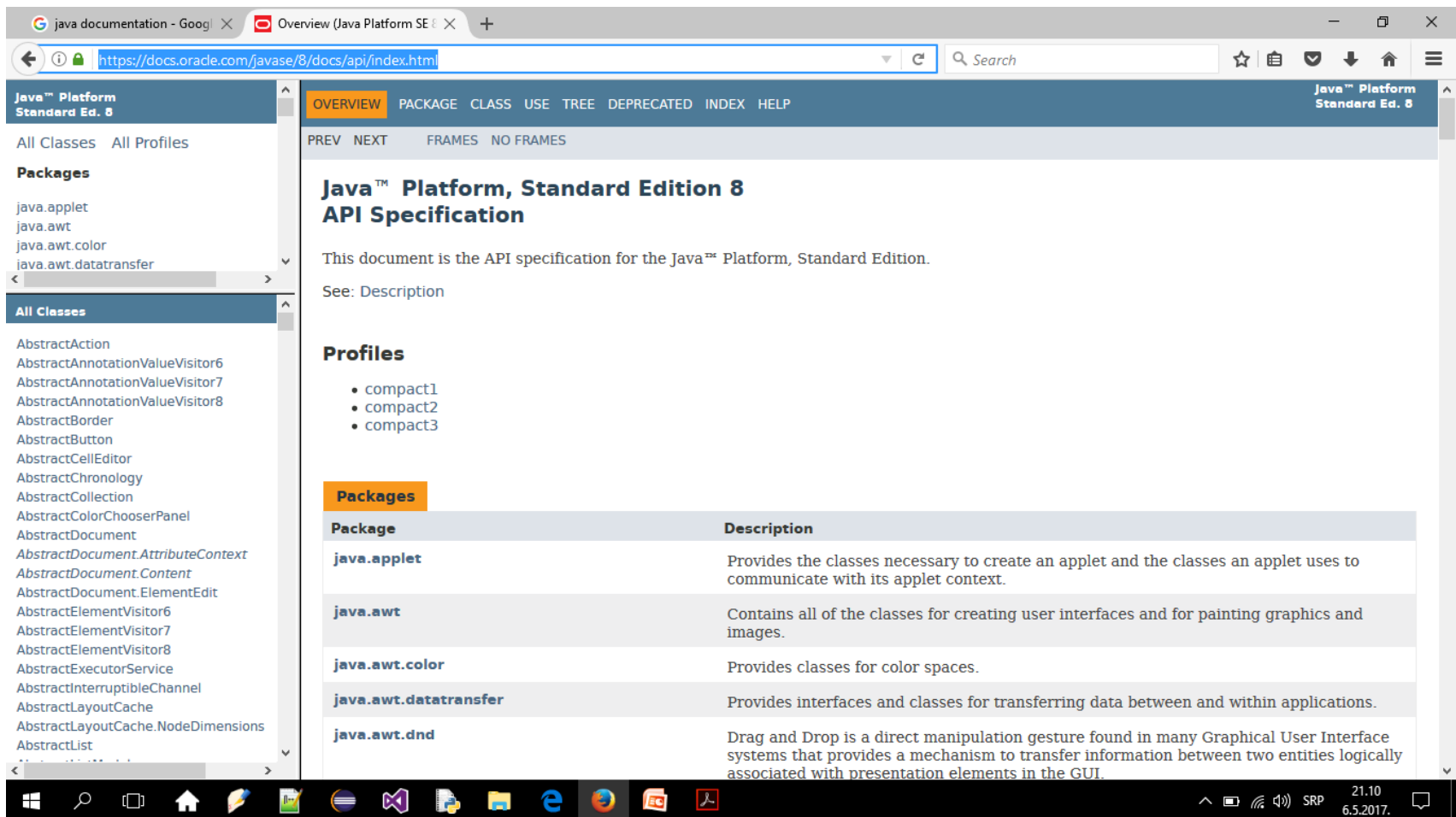
- Paket `java.io` sadrži interfejse i klase za rad sa ulazom i izlazom
- Klase u okviru ovog paketa realizuju rad sa tokovima
- Najvažnije klase su:
 - Za rad sa bajt tokovima – apstraktne klase `InputStream` i `OutputStream`
 - Za rad sa karakter tokovima – apstraktne klase `Reader` i `Writer`
- Metodi klasa ovog paketa generišu izuzetke tipa `IOException` u slučaju da ne mogu biti izvršeni - treba ih pozivati u okviru `try-catch-finally` struktura
- Paket `java.io` sadrži i klase kao što su `RandomAccessFile` (rad sa fajlovima sa slučajnim pristupom) i `File` (predstavlja fajl ili putanju u fajl sistemu)

Java platforma – `java.util`

- Paket `java.util` sadrži interfejse i klase sa strukturama podataka, generatom slučajnih brojeva, vremenom i datum i drugim pomoćnim alatima.
- Najvažniji deo ovog paketa je Collections radno okruženje – organizovana hijerarhija struktura podataka koja je projektovana pod jakim uticajem projektnih obrazaca, sadrži npr. `ArrayList`, `LinkedList`, `HashTable`, itd.
- U ovom paketu se nalaze važni intefejsi `Iterator`, `Comparator`, `Collection` ili `Map`, kao i klase kao što su `Scanner`, `Vector` ili `Calendar`

Java API dokumentacija

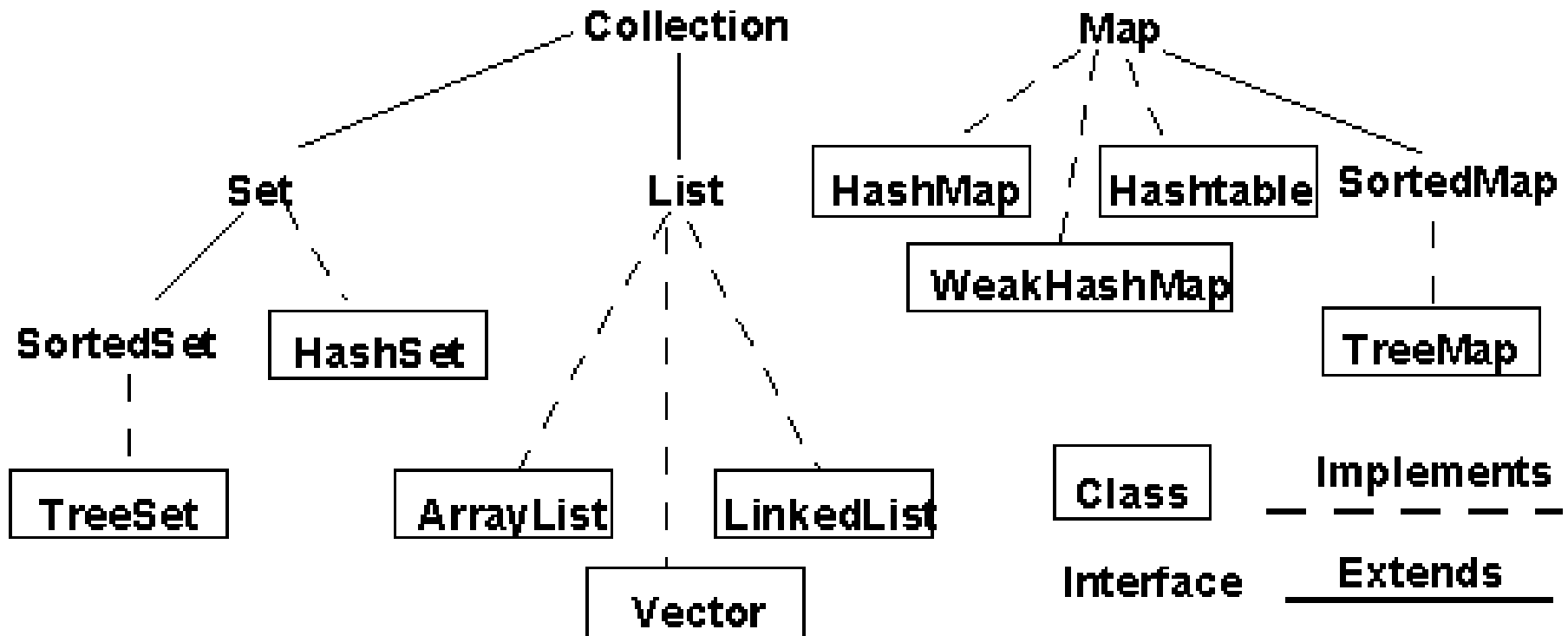
<https://docs.oracle.com/javase/8/docs/api/index.html>



The screenshot displays the Java Platform, Standard Edition 8 API Specification website. The page is titled "Java™ Platform, Standard Edition 8 API Specification". It includes a navigation bar with tabs for OVERVIEW, PACKAGE, CLASS, USE, TREE, DEPRECATED, INDEX, and HELP. The main content area shows the "Overview" section, which includes a description of the document and a list of profiles (compact1, compact2, compact3). Below this, there is a table of packages with their descriptions.

Package	Description
java.applet	Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context.
java.awt	Contains all of the classes for creating user interfaces and for painting graphics and images.
java.awt.color	Provides classes for color spaces.
java.awt.datatransfer	Provides interfaces and classes for transferring data between and within applications.
java.awt.dnd	Drag and Drop is a direct manipulation gesture found in many Graphical User Interface systems that provides a mechanism to transfer information between two entities logically associated with presentation elements in the GUI.

Java Collections Framework - JCF



Liste i skupovi u JCF

- Dva osnovna tipa kolekcija elemenata u Javi su **lista** i **skup**
- **Lista se sastoji od sekvence elemenata u linearnom uređenju.** Lista ima tačno određeno uređenje, ali to ne znači da su vrednosti elemenata u njoj sortirane
- **Skup je kolekcija u kojoj ne postoje duplirani elementi.** Elementi skupa mogu, ali i ne moraju da imaju neko uređenje
- Treći tip kolekcija koji se nešto ređe koristi nego liste i skupovi su **redovi sa prioritetom** (engl. priority queues)

Liste, skupovi i mape u JCF

- Dva standardne strukture podataka za predstavljanje listi su **dinamički niz** i **lančana lista**
- Skupovi u Javi, za razliku od matematičkog pojma skupa, moraju biti konačni i sadržati samo elemente istog tipa
- **Mape su vid generalizovanih nizova.** Sastoje se od elemenata u vidu parova (ključ, vrednost). Osnova za rad sa mapama u Javi je interfejs `Map<K, V>`
- Savremeni sistemi za rad sa velikim skupovima podataka kao što su Hadoop i Spark, zasnovani su na Javi i radu sa mapama – MapReduce programski model

JCF liste - ArrayList, LinkedList

- Objekat tipa `ArrayList<T>` predstavlja uređenu sekvencu objekata tipa `T`, smeštenih u nizu koji može da raste po potrebi – kad god se doda novi element
- Objekat tipa `LinkedList<T>` takođe predstavlja uređenu sekvencu objekata tipa `T`, ali objekti se čuvaju u čvorovima (engl. nodes) koji su međusobno uvezani pokazivačima
- Klasa `LinkedList` je efikasnija u primenama gde se često dodaju ili uklanjaju elementi na početku ili u sredini liste, dok je klasa `ArrayList` efikasnija kada je potreban čest slučajan pristup elementima liste
- Obe liste implementiraju metode interfejsa `Collection`, pa je moguće njihovo lako sortiranje (`sort`), okretanje (`reverse`), itd.

JCF skupovi - TreeSet, HashSet

- Skupovi implementiraju sve metode interfejsa `Collection`, ali na takav način da obezbede da se nijedan elemenat ne može pojaviti dva puta u skupu
- Skup `TreeSet` ima svojstvo da su njegovi elementi uređeni u rastući redosled
- Skup `HashSet` čuva svoje elemente u posebnoj strukturi podataka poznatoj kao heš tabela (engl. hash table)
- Kod heš tabela su operacije pronalaženja, dodavanja i brisanja elementa vrlo efikasne (dosta brže nego kod `TreeSets`). Elementi `HashSet`-a se ne čuvaju u nikakvom posebnom uređenju

Zadatak 5.1 – Spisak polaznika

- Napraviti program koji čitanjem iz ulaznog tekstualnog fajla *spisak.txt* prihvata podatke o polaznicima (ime, prezime, JMBG) i prikazuje ih na ekranu. Potom treba spisak polaznika sortirati po JMBG-u, ponovo ga prikazati na ekranu i na kraju ga upisati i u izlazni fajl *uredjeniSpisak.txt*
- Klase testirati u glavnom programu kreiranjem objekta sa spiskom polaznika i pozivanjem odgovarajućih metoda

Zadatak 5.1 – klasa Osoba

```
public class Osoba {  
    private String ime;  
    private String prezime;  
    private String jmbg;  
  
    Osoba() {}  
  
    public Osoba(String ime, String prezime, String jmbg){  
        this.ime = ime;  
        this.prezime = prezime;  
        this.jmbg = jmbg;  
    }  
  
    public String pribaviIme(){  
        return this.ime;  
    }  
    ...  
}
```

Zadatak 5.1 – klasa Osoba

...

```
public String pribaviPrezime(){  
    return this.prezime;  
}
```

```
public String pribaviJMBG(){  
    return this.jmbg;  
}
```

```
public void postaviIme(String ime){  
    this.ime = ime;  
}
```

```
public void postaviPrezime(String prezime){  
    this.prezime = prezime;  
}
```

...

Zadatak 5.1 – klasa Osoba

...

```
public void postaviJMBG(String jmbg){  
    this.jmbg = jmbg;  
}
```

```
@Override public String toString() {  
    return ("Ime:" + this.pribaviIme() + " Prezime: "  
        + this.pribaviPrezime() + " JMBG: "  
        + this.pribaviJMBG());  
}
```

```
}
```

Zadatak 5.1 – klasa MojKomparator

```
import java.util.*;

class MojKomparator implements Comparator<Osoba> {

    @Override public int compare(Osoba o1, Osoba o2) {
        int i = o1.pribaviJMBG().compareTo(o2.pribaviJMBG())
        if (i > 0) {
            return -1;
        }
        else if (i < 0) {
            return 1;
        }
        return 0;
    }
}
```

Zadatak 5.1 – klasa Spisak

```
import java.io.*;
import java.util.*;

public class Spisak {
    ArrayList<Osoba> listaPolaznika;

    public void učitajListu(String imeFajla) {
        Scanner s = null;
        ArrayList<Osoba> listaPolaznika = new ArrayList<Osoba>();
        try {
            s = new Scanner(new File(imeFajla));
            do {
                String ime = s.next();
                String prezime = s.next();
                String jmbg = s.next();
                Osoba noviPolaznik = new Osoba(ime, prezime, jmbg);
                listaPolaznika.add(noviPolaznik);
            } while (s.hasNext());
        } catch (IOException e) {
            System.out.println(e.getMessage());
        } ...
    }
}
```


Zadatak 5.1 – klasa Spisak

```
...
    finally {
        if (s != null) {
            s.close();
        }
    }
    this.listaPolaznika = listaPolaznika;
}

public void sortirajListu() {
    Collections.sort(this.listaPolaznika, new MojKomparator());
}

public void stampajListu() {
    System.out.println(Arrays.toString(this.listaPolaznika.toArray()));
}

...
```

Zadatak 5.1 – klasa Spisak

```
...
public void upisiListu(String imeFajla) {
    PrintWriter pw = null;
    try {
        pw = new PrintWriter(new FileOutputStream(imeFajla));
        for (Osoba polaznik : this.listaPolaznika)
            pw.println(polaznik.pribaviIme() + " " +
                        polaznik.pribaviPrezime() + " " +
                        polaznik.pribaviJMBG());
    } catch (IOException e) {
        System.out.println(e.getMessage());
    }
    finally {
        if (pw != null) {
            pw.close();
        }
    }
}
```

```
}
```

Zadatak 5.1 – klasa Main

```
public class Main {  
  
    public static void main(String[] args) {  
  
        Spisak polaznici = new Spisak();  
  
        polaznici.ucitajListu("spisak.txt");  
        polaznici.stampajListu();  
        polaznici.sortirajListu();  
        polaznici.stampajListu();  
        polaznici.upisiListu("uredjeniSpisak.txt");  
    }  
}
```

Zadaci za rad na času

- Modifikovati paket zaposleni tako da uključuje i klasu Spisak. Za čuvanje spiska radnika upotrebiti pogodnu strukturu iz Java Collections Framework-a.
- Modifikovati klase Institucija, Ucionica, Zaposleni (koja nasleđuje klasu Osoba) i Racunar tako da koriste gotove strukture podataka iz Java Collections Framework. Pod kojim uslovima je za čuvanje sekvence objekata efikasnije koristiti ArrayList, a pod kojima LinkedList?

UNIFIED MODELING LANGUAGE (UML)

UML

- Objedinjeni jezik za modelovanje - UML (Unified Modeling Language)
- UML predstavlja standardizovani jezik i grafičku notaciju za
 - vizuelizaciju,
 - specifikaciju,
 - modelovanje i
 - dokumentovanjedelova softverskog sistema koji se projektuje
- UML predstavlja zajednički “rečnik” za sporazumevanje između osoba uključenih u projekovanje i razvoj nekog softverskog sistema

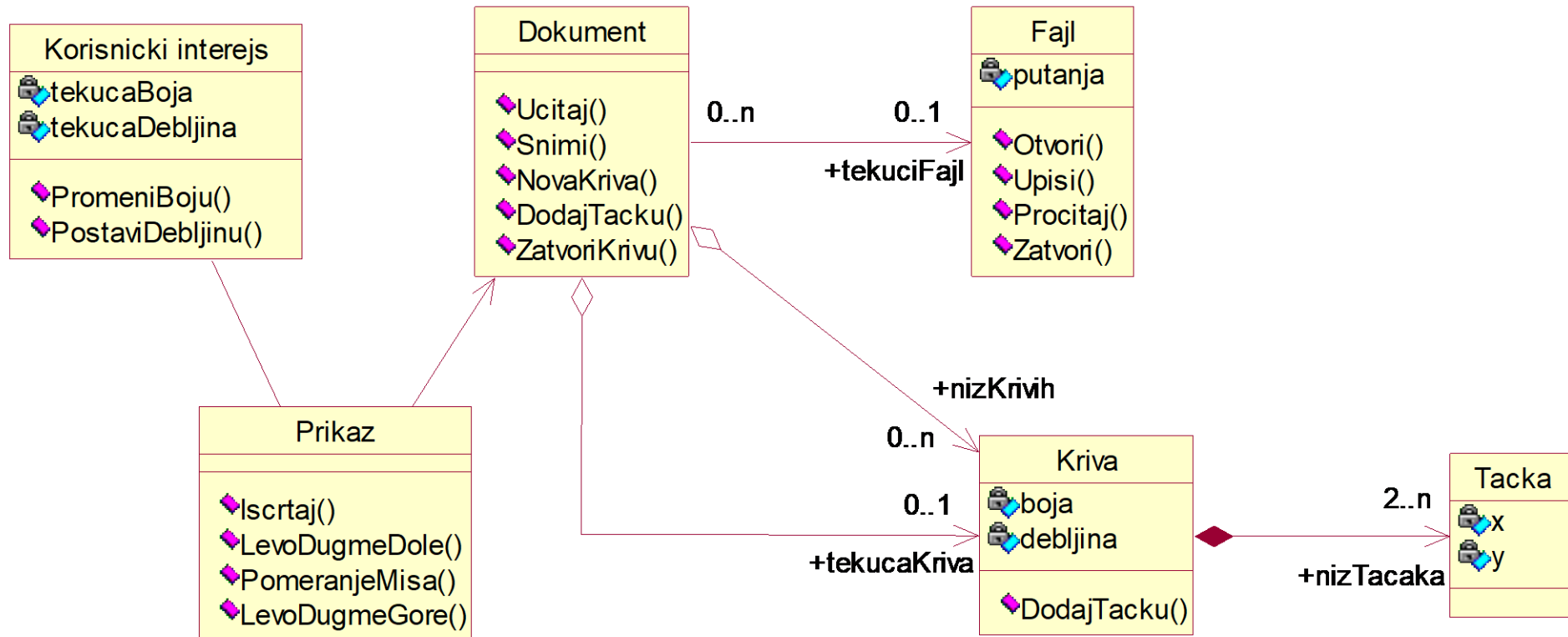
Tipovi UML dijagrama

- Dijagrami klasa (engl. *Class Diagram*)
- Dijagrami slučajeve korišćenja (engl. *Use-Case Diagram*)
- Dijagrami sekvence (engl. *Sequence Diagram*)
- Dijagrami saradnje (engl. *Collaboration Diagram*)
- Dijagrami stanja (engl. *Statechart Diagram*)
- Dijagrami aktivnosti (engl. *Activity Diagram*)
- Dijagrami komponenti (engl. *Component Diagram*)
- Dijagrami razmeštaja (engl. *Deployment Diagram*)

Dijagrami klasa

- Koriste se za predstavljanje klasa i njihove organizacije u pakete
- Dijagrami klasa se koriste za modelovanje
 - domena sistema
 - aplikacije
- Elementi dijagrama su:
 - klase
 - veze između klasa
 - nasleđivanje
 - asocijacija
 - agregacija
 - kompozicija
 - paketi
 - veze zavisnosti između paketa

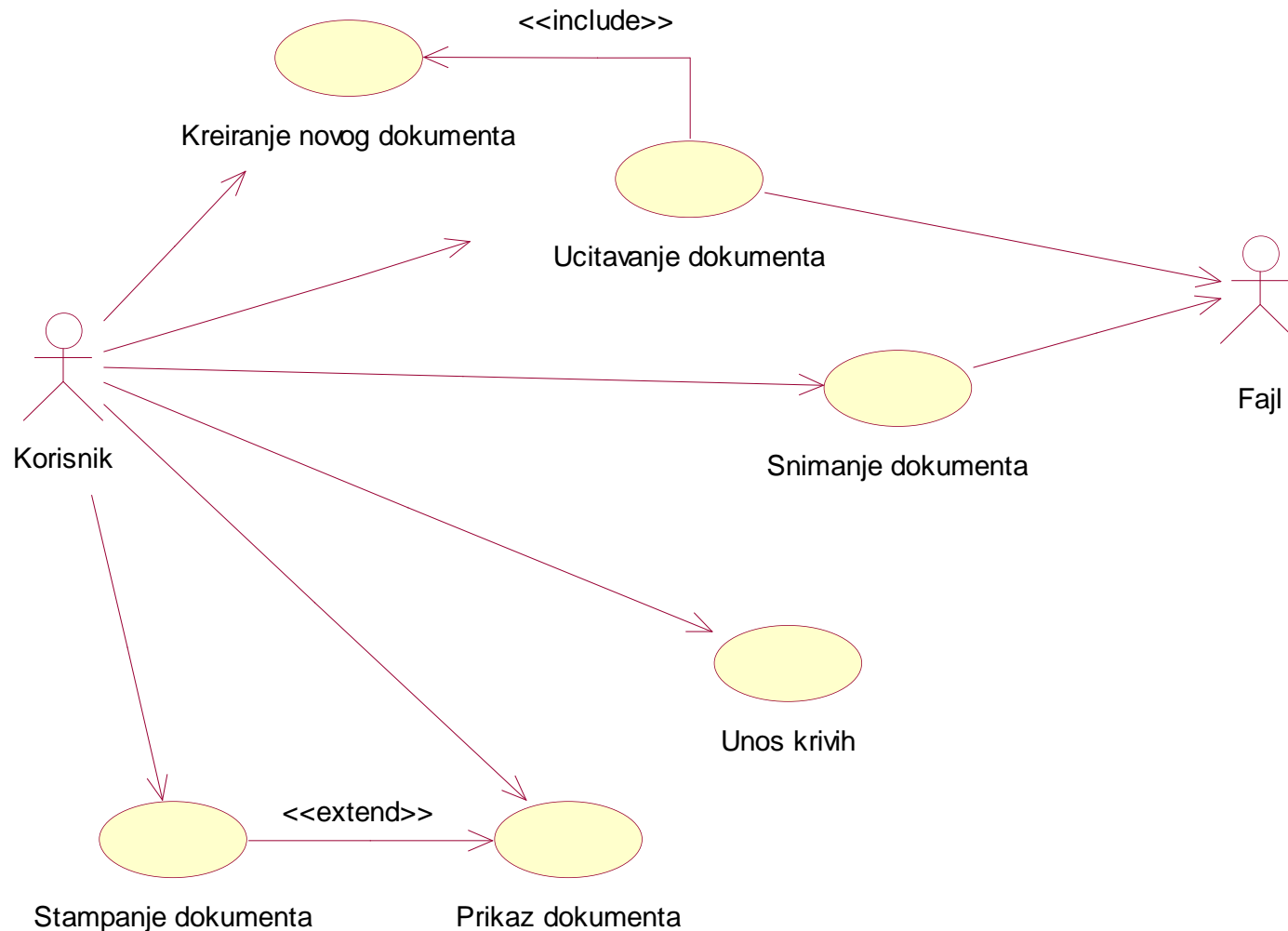
Dijagram klasa - primer



Dijagrami slučajeva korišćenja

- Koriste se u procesu prikupljanja i dokumentovanja korisničkih zahteva
- Elementi dijagrama su:
 - akteri
 - korisnici sistema
 - drugi sistemi iz okruženja
 - slučajevi korišćenja sistema
 - veze između aktera i slučajeva korišćenja
 - asocijacija
 - generalizacija
 - paketi
 - veze zavisnosti između paketa

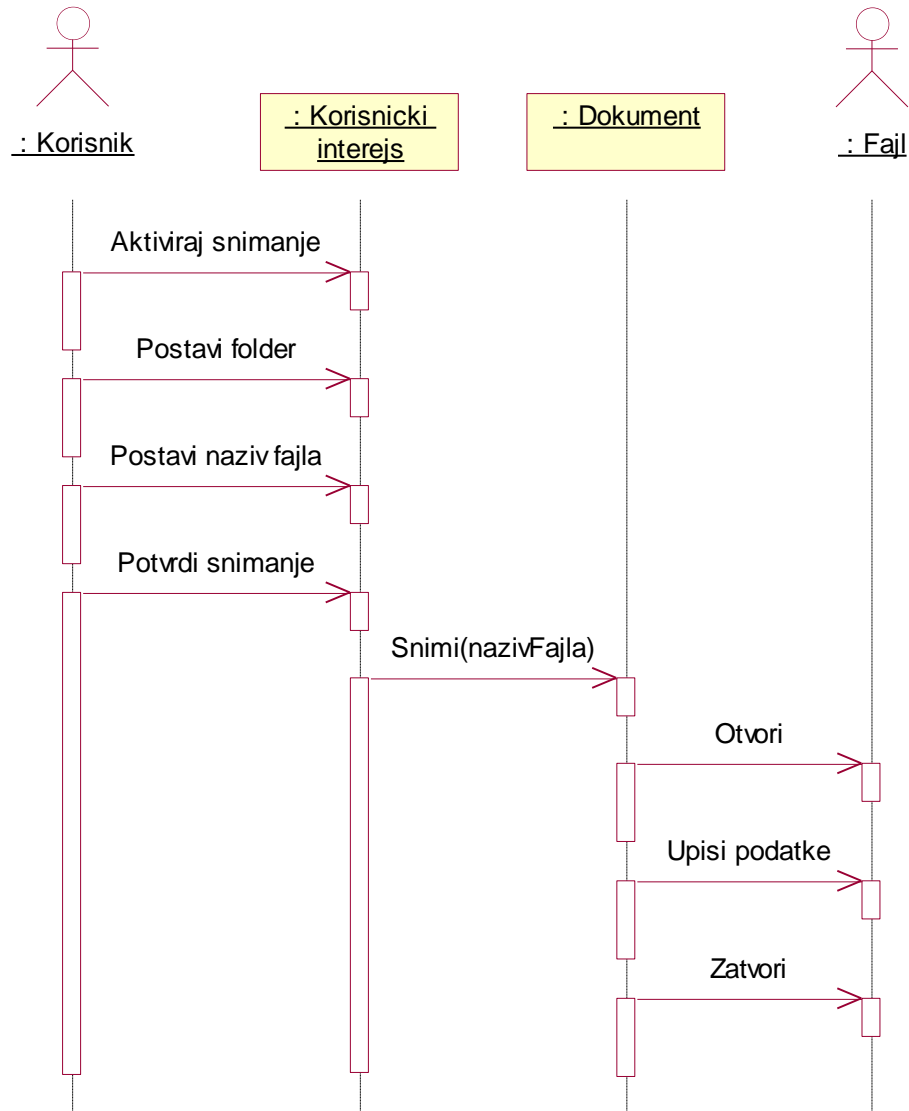
Dijagram slučajeja korišćenja - primer



Dijagrami sekvence

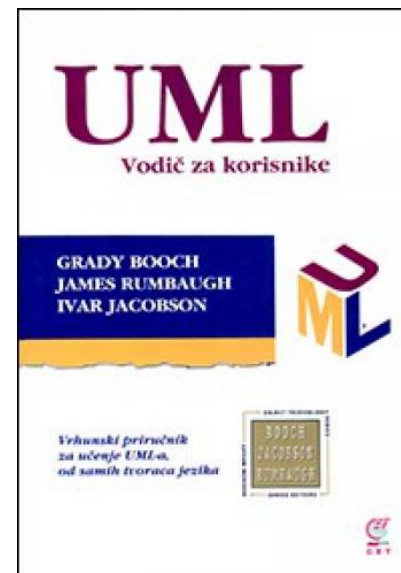
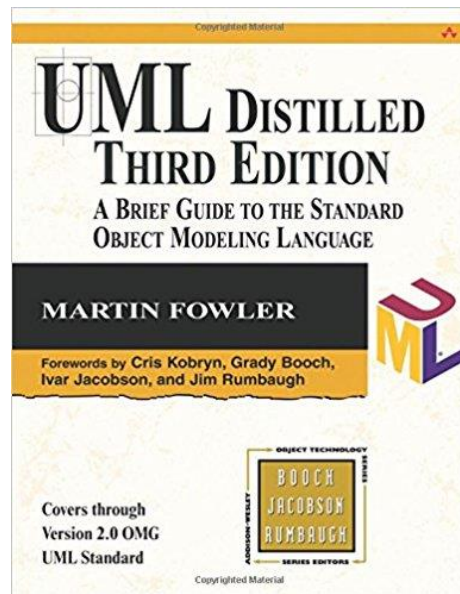
- Koriste se za predstavljanje scenarija interakcije između objekata u sistemu
 - Najčešće se ovi scenariji odnose na slučajeve korišćenja sistema
- Elementi dijagrama su:
 - objekti
 - vremenska linija
 - poruke između objekata

Dijagram sekvence - primer



Literatura za UML

- Martin Fowler, UML Distilled: A Brief Guide to the Standard Object Modeling Language, 3rd edition, Addison Wesley, 2003.
- Grady Booch, James Rumbaugh, Ivar Jacobson, UML - Vodič za korisnike, CET, 2001.



Zadaci za vežbanje

- Nacrtati UML dijagrame klasa i dijagrame slučajeva korišćenja za klase ranije implementirane u okviru paketa zaposleni, matematika i institucija. Identifikovati tipične slučajeve korišćenja za aplikacije koje koriste prethodne pakete.

OBJEKTNO-ORIJENTISANO PROJEKTOVANJE SOFTVERA

Projektovanje OO programa

*“Postoje dva načina za projektovanje i razvoj programa. Jedan je da ih učinite toliko jednostavnim da je očigledno da nemaju nedostatke. Drugi je da ih učinite toliko komplikovanim da nemaju **očiglednih** nedostataka.”*

C.A.R. Hoare

- Ključ za **uspešno OO projektovanje programa** je **dobro osmišljena apstrakcija problema** kroz objektne koncepte:

“Suština apstrakcije je čuvanje informacija koje su relevantne u datom kontekstu i ignorisanje informacija koje su irelevantne za dati kontekst”

John Guttag

Ulazni podaci za OO projektovanje

- **Konceptualni model** je rezultat objektno-orijentisane analize i opisuje koncepte u problemskom domenu. Eksplicitno se kreira tako da bude nezavisan od implementacionih detalja, kao što su konkurentnost ili čuvanje podataka
- **Slučajevi korišćenja** (use cases) su opisi sekvence događaja koji zajedno dovode do toga da sistem realizuje neku korisnu aktivnost
- **Dijagrami sekvence** grafički prikazuju, za određeni scenario u okviru nekog slučaja korišćenja, događaje koje generišu eksterni akteri, njihov redosled i moguće događaje unutar sistema

Ulazni podaci za OO projektovanje

- **Dokumentacija za korisnički interfejs** (engl. user interface – UI) prikazuje i opisuje izgled i tok rada sa programom putem korisničkog interfejsa finalnog softverskog proizvoda
- **Relacioni model podataka** – model podataka je apstraktni model koji opisuje kako se podaci predstavljaju i koriste. Ako se ne koristi objektna baza podataka, tada je relacioni model podataka neophodno unapred kreirati, pošto je izabrana strategija za objektno-relaciono mapiranje jedan od izlaza procesa objektno-orijentisanog projektovanja

Projektovanje OO programa

- **Definisati objekte** (kreirati dijagrame klase na osnovu konceptualnih dijagrama). Tom prilikom se entiteti tipično mapiraju na klase
- **Definisati elemente klasa** (atributi i metode)
- **Definisati broj objekata, trenutak njihovog nastajanja i nestajanja, kao i način medjusobne interakcije** tokom vremena
- **Definisati odgovornosti** svakog dela sistema

Projektovanje OO programa

- **Upotrebiti projektne obrazce** (ako su primenljivi) - glavna prednost primene projektnih obrazaca je mogućnost njihovog ponovnog korišćenja u više aplikacija. Objektno-orijentisani projektni obrazci obično prikazuju odnose i interakcije između klasa i objekata, bez specifikacije konačnih aplikacionih klasa i objekata
- **Izabrati radno okruženje** (ako je primenljivo) - radna okruženja uključuju veliki broj biblioteka i klasa koje se mogu iskoristi za implementaciju standardnih struktura u aplikaciji. Na ovaj način se može dosta uštedeti na vremenu razvoja softvera pošto se izbegava ponovno pisanje velikog dela koda prilikom razvoja novih aplikacija
- **Identifikovati perzistentne objekte/podatke** (ako je primenljivo) – potrebno je identifikovati objekte koji treba da postoje duže od trajanja jednog izvršenja aplikacija. Ako aplikacija koristi relacionu bazu podataka, projektovati objektno-relaciono mapiranje

Projektni obrasci

- **Projektni obrasci su opšta, ponovo upotrebljiva rešenja za probleme koji se često javljaju** u određenom kontekstu projektovanja softvera
- Oni **nisu kompletan projekat koji se može direktno transformisati u izvorni ili mašinski kod**, već su opis ili šablon za rešavanje problema koji može da se koristi u mnogo različitih situacija
- Projektni obrasci su **formalizovani postupci najbolje prakse** (engl. best practices) koje programeri mogu koristiti kako bi efikasno rešili tipične probleme koji se javljaju prilikom projektovanja aplikacije ili sistema

Projektni obrasci

- Projektni obrasci su originalno (prema GoF) podeljeni na: **stvaralačke** (engl. creational), **strukturalne** (engl. structural) i **bihevijoralne** (engl. behavioral), a danas se koriste i **konkurentni** (npr. blockchain) i **arhitekturni** (npr. Model-View-Controller - MVC)
- **Stvaralački**: Singleton (osigurava da klasa ima samo jednu instancu za koju postoji globalni pristup), Builder, Factory...
- **Strukturalni**: Adapter, Facade, Decorator...
- **Bihevijoralni**: Iterator, Interpreter, Visitor...

Primer 5.1 – obrazac Singleton u Javi

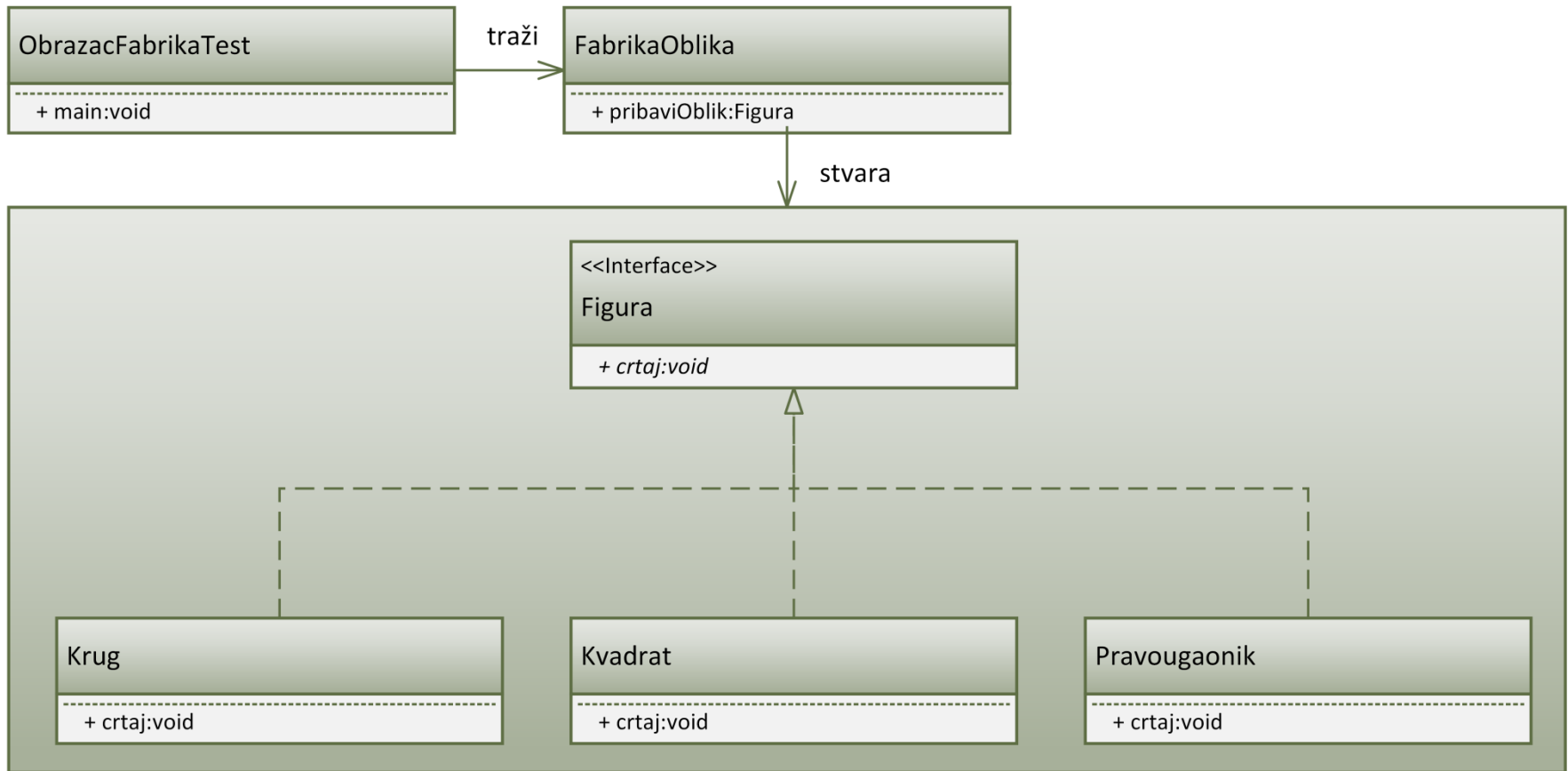
```
public class Singleton {  
  
    private static Singleton instanca = null;  
  
    protected Singleton() {  
        // Postoji samo kako bi sprecili instanciranje  
    }  
  
    public static Singleton pribaviInstancu() {  
        if(instanca == null) {  
            instanca = new Singleton();  
        }  
        return instanca;  
    }  
}
```

Tipične primene: čuvanje podešavanja aplikacije (konfiguracioni fajlovi), logovanje podataka, itd.

Primer 5.2 – obrazac Factory u Javi

- Sa Factory obrascem, kreiramo objekat bez potrebe da izložimo logiku kreiranja objekta (engl. *creation logic*) klijentu i potom se obraćamo novostvorenom objektu korišćenjem zajedničkog interfejsa
- Kreiraćemo interfejs `Figura` i konkretne klase koje implementiraju ovaj interfejs. Potom ćemo u narednom koraku definisati klasu „fabriku“ `FabrikaOblika`
- Napravićemo i test klasu `ObrazacFabrikaTest` koja će koristiti klasu `FabrikaOblika` kako bi pribavila odgovarajući objekat nekog oblika. Test klasa će samo prosleđivati informaciju fabrici oblika da li je u pitanju krug, kvadrat ili pravouganik, a klasa `FabrikaOblika` će potom „isporučivati“ traženi oblik test klasi

Primer 5.2 – obrazac Factory u Javi



Primer 5.2 – interfejs Oblik, klase Krug, Pravougaonik i Kvadrat

```
public interface Figura {                                //Figura.java
    void crtaj();
}

public class Pravougaonik implements Figura {           //Pravougaonik.java
    @Override public void crtaj() {
        System.out.println("Unutar Pravougaonik::crtaj() metode!");
    }
}

public class Kvadrat implements Figura {                //Kvadrat.java
    @Override public void crtaj() {
        System.out.println("Unutar Kvadrat::crtaj() metode!");
    }
}

public class Krug implements Figura {                   //Krug.java
    @Override public void crtaj() {
        System.out.println("Unutar Krug::crtaj() metode!");
    }
}
```

Primer 5.2 – klasa FabrikaOblika

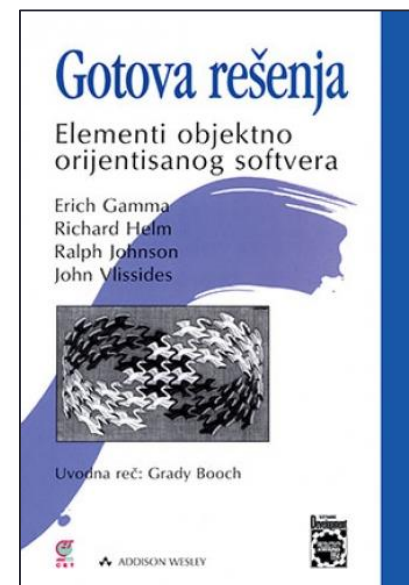
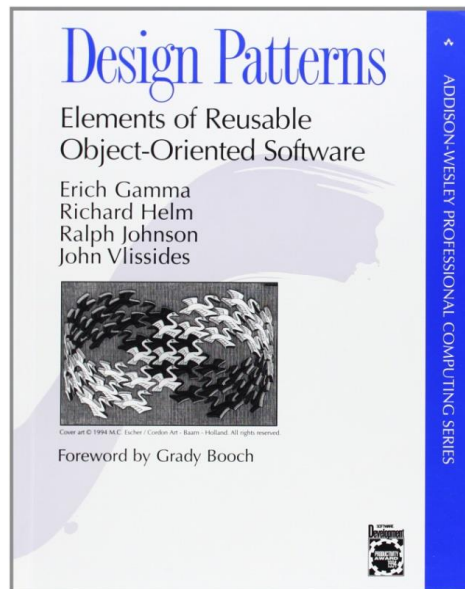
```
public class FabrikaOblika {  
    //metod pribaviOblik dobavlja oblik zeljenog tipa  
    public Figura pribaviOblik(String tipOblika){  
        if (tipOblika == null){  
            return null;  
        }  
        if (tipOblika.equalsIgnoreCase("KRUG")){  
            return new Krug();  
        } else if (tipOblika.equalsIgnoreCase("PRAVOUGAONIK")){  
            return new Pravouganik();  
        } else if (tipOblika.equalsIgnoreCase("KVADRAT")){  
            return new Kvadrat();  
        }  
        return null;  
    }  
}
```

Primer 5.2 – klasa ObrazacFabrikaTest

```
public class ObrazacFabrikaTest {  
    public static void main(String[] args) {  
        FabrikaOblika fabrikaOblika = new FabrikaOblika();  
  
        // pribavi oblik Krug i pozovi njegov metod crtaj  
        Figura oblik1 = fabrikaOblika.pribaviOblik ("KRUG");  
        // pozovi metod crtaj za krug  
        oblik1.crtaj();  
  
        // pribavi oblik Pravougaonik i pozovi njegov metod crtaj  
        Figura oblik2 = fabrikaOblika.pribaviOblik ("PRAVOUGAONIK");  
        // pozovi metod crtaj za pravougaonik  
        oblik2.crtaj();  
  
        // pribavi oblik Kvadrat i pozovi njegov metod crtaj  
        Figura oblik3 = fabrikaOblika.pribaviOblik ("KVADRAT");  
        // pozovi metod crtaj za kvadrat  
        oblik3.crtaj();  
    }  
}
```

Projektni obrasci

- Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison Wesley 1994. – poznata kao “Gang of Four” – “GoF”
- Srpsko izdanje: Gotova rešenja, CET Beograd, 2002.





How the customer explained it



How the project leader understood it



How the engineer designed it



How the programmer wrote it



How the sales executive described it



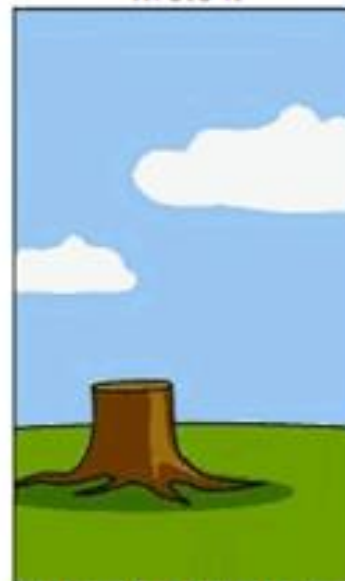
How the project was documented



What operations installed



How the customer was billed



How the helpdesk supported it



What the customer really needed

Zadatak za rad na času

- Korišćenjem projektnog obrasca Factory napraviti “fabriku vozila”.
- Treba kreirati interfejs `Proizvod` i konkretne klase koje implementiraju ovaj interfejs. Potrebno je i definisati klasu „fabriku“ - `FabrikaVozila`
- Treba kreirati i test klasu `ObrazacFabrikaTest` koja će koristiti klasu `FabrikaVozila` kako bi pribavila odgovarajući objekat nekog tipa vozila. Test klasa će samo prosleđivati informaciju fabriki vozila da li je u pitanju automobil, kamion ili motocikl, a klasa `FabrikaVozila` će potom „isporučivati“ traženo vozilo test klasi

Zadaci za vežbanje

- Proučiti ostale najvažnije projektne obrasce (pored Singleton i Factory, to su npr. Builder, Adapter, Facade, Iterator, Visitor, MVC) i modifikovati klase u paketu zaposleni tako da koriste projektne obrasce u situacijama gde je to adekvatno.
- Modifikovati preostale klase razvijene tokom kursa iz OOP tako da se primene projektni obrasci u situacijama gde je to adekvatno.