Razvoj softvera

dr.sc. Emir Mešković

II predavanje

Sadržaj predavanja

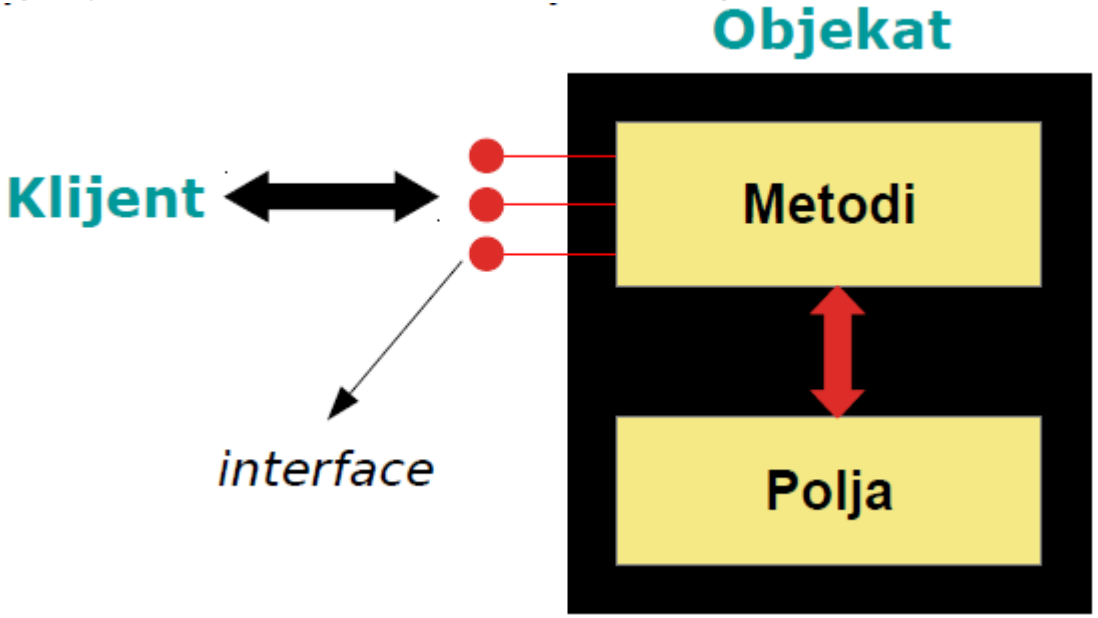
- Klase i objekti
- Paketi

Java i OOP

- Java je objektno orjentirani programski jezik
  - Java programi sastoje se od klasa i objekata
- Objekat predstavlja tijesno vezivanje:
  - *Stanja* – definisano vrijednostima varijabli koje sadrži objekat (varijable se nazivaju članovi ili polja objekta). Članovi mogu biti primitvne Java varijable (tj numeričke vrijednosti) ili reference (tj drugi objekti)
  - *Ponašanja* – programski kod (metod) koji operiše nad stanjem konkretnog objekta.
- Klasa je nacrt za objekat
  - Definira članove i ponašanje objekata koji nastaju na osnovu klase.
  - Od jedne klase može se napraviti proizvoljan broj objekata. Svaki objekat može imati različite vrijednosti članova (tj. različito stanje)

Enkapsulacija

- OOP programi
  - Program se sastoji od skupa objekata koji međusobnom komunikacijom riješavaju određeni problem.
  - Jedan objekat (*klijent)* može zahtijevati neku funkcionalnost koji pruža drugi objekat. Klijent ne treba da ima bilo kakvu informaciju o tome kako se ta funkcionalnost realizira.
  - Funkcionalnost se pruža kroz interface metode, a unutrašnje stanje (tj članovi ili polja) treba biti direktno dostupno samo unutar objekta.



Klase

- Java klase se definiraju u obliku:

```
modifikator class ImeKlase
{
    metod1
    metod2
    . . .
    polje1
    polje2
    . . .
}
```

- Redosljed definiranja metoda i polja nije bitan
- Klase se po Java konvenciji imenuju velikim prvim slovom.
- Modifikator označava tip klase (npr public, abstract itd...) i nije obavezan.

Klase i Java fajlovi

- Klase se snimaju u tekstualne fajlove sa `java` ekstenzijom.
  - `java` fajl može da sadrži proizvoljan broj klasa. Svaka klasa će se iskompajlirati u svoj bytecode fajl (tj `class` fajl) sa imenom identičnim imenu klase.
  - Ako je neka klasa u java fajlu označena kao javna (pomoću modifikatora `public`) onda java fajl mora imati isto ime kao i javna klasa.
    - iz gornjeg proizilazi da samo jedna javna klasa može biti snimljena unutar jednog java fajla

Paketi

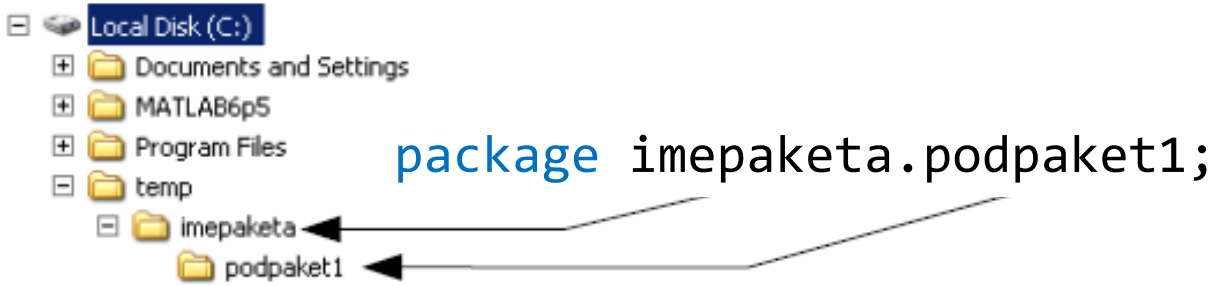
- Java klase se organizuju u pakete
  - Svaka klasa pripada nekom paketu
  - Sve klase unutar istog java fajla pripadaju istom paketu
  - Klase iz određenog fajla postaju dio paketa tako što se na početku tog fajla upisuje linija:

```
package imepaketa;
```

  - Ako se izostavi ime paketa, sve klase unutar java fajla postaju dio neimenovanog tzv *default* paketa.

Paketi i direktoriji

- Struktura paketa
  - Paketi su vezani sa direktorijima na način da `java` fajl koji pripada paketu *imepaketa* mora biti snimljen u istoimenom poddirektoriju.
  - Paket *imepaketa* može da sadrži podpakete. Podpaket *podpaket1* koji pripada paketu *imepaketa* treba biti snimljen u poddirektoriju *podpaket1* unutar poddirektorija *imepaketa.*
    - `java` fajlovi u paketu `podpaket1` tada počinju sa:



    - Više java fajlova može specificirati pripadnost istom paketu (svi fajlovi moraju biti u odgovarajućem direktoriju)

Definiranje i kreiranje objekata

- Objekat `objekat1` klase `ImeKlase` definira se:

```
ImeKlase objekat1;
```

- Kao i slučaju sa Java nizovima, `objekat1` je tipa Java reference (pointer u C++ terminologiji) na lokaciju u memoriji gdje će se nalaziti objekat klase `ImeKlase`
- Da bi konstruisali objekat na toj lokaciji potrebno je pozvati neki od konstruktor metoda klase `ImeKlase` pomoću operatora `new`, koji će alocirati memoriju potrebnu za objekat klase `ImeKlase`.

```
ImeKlase objekat1 = new ImeKlase();
```

- Kao i u C++, Java klasa može imati različite konstruktore koji primaju različiti broj i tip parametara.

Klase objekti i paketi

- Da bi mogli kreirati objekte određene klase moramo imati pristup klasi.
- Pristup klasama ovisi o njihovim pripadnostima paketima.
- Objekat `objekat1`, klase `Klasa1`, iz paketa `paket1`, možemo kreirati:
  - U kodu neke klase iz paketa `paket1` sa
    - `Klasa1 objekat1;`
  - U kodu neke klase iz bilo kojeg drugog paketa sa
    - `paket1.Klasa1 objekat1;`
  - `objekat1` moguće je kreirati van paketa `paket1` samo ukoliko je klasa `Klasa1` definirana kao javna (modifikator `public`) u suprotnom nije moguće kreirati objekat klase `Klasa1` van paketa `paket1`
- Klase iz default paketa moguće je koristiti samo unutar drugih klasa koje pripadaju default paketu

Ključna riječ `import`

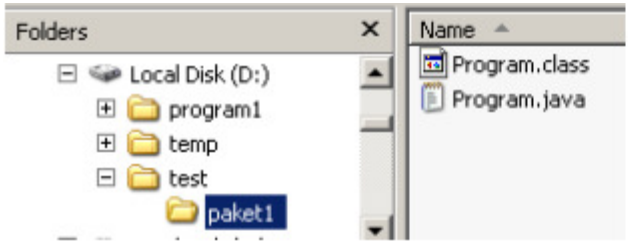
- Pretpostavimo da želimo konstruisati objekat klase `Date` iz paketa `java.util`
  - `java.util.Date danas = new java.util.Date();`
- Alternativno, odmah nakon deklaracije paketa možemo koristiti `import` instrukciju i to na dva načina:
  - Prvi način
    - `import java.util.Date;`
    - Nakon čega možemo koristiti klasu `Date` kao da je definirana unutar trenutnog paketa tj.
    - `Date danas = new Date();`
  - Drugi način
    - `import java.util.*;`
    - Nakon čega možemo koristiti klasu `Date` kao i sve ostale klase iz paketa `java.util` kao da su definirane unutar trenutnog paketa.
    - Gornje ne važi za klase koje se nalaze u eventualnim podaketima paketa `java.util`

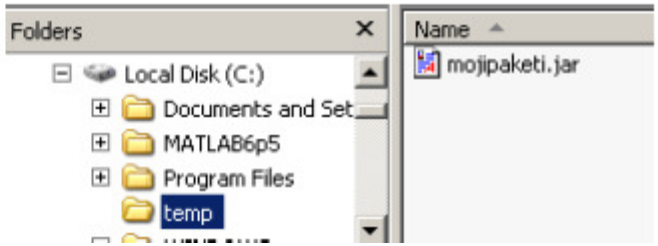
JVM, paketi i `jar` arhive

- Java programi se obično distribuiraju u obliku arhive sa `jar` ekstenzijom
  - `jar` arhiva je zip arhiva koja sadrži pakete (class fajlovi skupa sa direktorijima) koji se koriste u programu
  - `rt.jar` (obično se nalazi u direktoriju `C:\ProgramFiles\Java\jre1.5.0_06\lib`) sadrži pakete iz standardne Java biblioteke (npr. paketi `java` i `javax`)
- Startanje programa
  - Da bi mogli startati (ili kompajlirati) neki Java program koji koristi pakete izvan standardne biblioteke, JVM mora znati lokacije na kojim se nalaze paketi sa klasama koje koristi program.
  - Za obavještavanje JVM o lokaciji paketa
    - može se postaviti sistemska varijabla `CLASSPATH`
    - Ili koristiti `-cp` parametar pri pozivu jvm-a

Primjer startanja programa

- Postavka
  - Pretpostavimo da želimo da startamo program definiran u `main` metodu klase `Program` iz paketa `paket1` koji se nalazi u direktoriju `test` na disku D. Pored standardne bibiloteke klasa koristi klase iz paketa koji se nalaze u arhivi `mojipaketi.jar` koja se nalazi u direktoriju `temp` na disku C.





- Opcija 1 (pomoću `CLASSPATH`) u komandnoj liniji upisati:

```
C:\>set CLASSPATH=c:\temp\mojipaketi.jar;d:\test
C:\>java paket1.Program
```

- Opcija 2 (pomoću `-cp` opcije) u komandnoj liniji upisati:

```
C:\>java -cp d:\test;c:\temp\mojipaketi.jar paket1.Program
```

Definiranje članova klase

- Članovi (polja)
  - mogu se definirati bilo gdje unutar tijela klase u liniji oblika:
    - `modifikatori tip ime;`
  - npr
    - `public String prezime;`
  - **Modifikatori za pristup:**
    - `public` - javno dostupan član
    - `private` – član dostupan samo unutar klase
    - `protected` – član dostupan u unutar klase i klasa koje nasljeđuju od date klase
    - Ako se modifikator izostavi pristup je na nivou paketa tj. članu se može direktno pristupiti bilo gdje unutar paketa (nije preporučljivo)
  - **Tip**
    - Može biti bilo koji validan Java tip (primitivni ili klasa) npr `int`, `String`, `double[]` ...

Metodi

- Mogu se definirati bilo gdje unutar tijela klase u obliku:

```
modifikator povratni_tip ime(tip_1 par_1, ..., tip_n par_n)
{
  //tijelo
}
```

- Modifikatori za pristup metodima su identični kao za članove
- povratni_tip – funkcija može da vraća bilo koji validni Java tip
- Metod može da prima proizvoljan broj parametara za svaki primljeni parametar potrebno je specificirati ime i njegov tip
- Metodi mogu pristupiti instanci trenutnog objekta na kojem operiraju preko ključne riječi `this` (slično kao C++)
- Metodi imaju direktni pristup svim članovima objekta na kojem operišu (`this` objekat) kao i svim članovima objekata iste klase.

Primjer 1 C++ kod


```
#include <iostream>
#include <cmath>
class Tacka
{
    public:
      Tacka(double x, double y) : x_(x), y_(y) {}
      Tacka() : x_(0), y_(0) {}
      double vratiX() { return x_; }
      double vratiY() { return y_; }
      void postaviX(double x) { x_ = x; }
      void postaviY(double y) { y_ = y; }
      double distanca(Tacka* druga)
      {
          return sqrt(pow((x_-druga->x_),2)+pow(y_-druga->y_,2));
      }
    private:
      double x_;
      double y_;
};
int main()
{
    Tacka* a = new Tacka(5,10);
    Tacka* b = new Tacka();
    std::cout << "Distanca je " << a->distanca(b);
    delete a;
    delete b;
    return 0;
}
```

Ekvivalentan Java kod


```
public class Tacka
{
    public Tacka(double x, double y)
    {
       x_ = x;
       y_ = y;
    }
    public Tacka() { this(0,0); }
    public double vratiX() { return x_; }
    public double vratiY() { return y_; }
    public void postaviX(double x) { this.x_ = x; }
    public void postaviY(double y) { y_ = y; }
    public double distanca(Tacka druga)
    {
       return Math.sqrt(Math.pow((x_-druga.x_),2)+Math.pow(y_-druga.y_,2));
    }
    private double x_;
    private double y_;
}
class Tester
{
    public static void main(String[] args)
    {
       Tacka a = new Tacka(5,10), b = new Tacka();
       System.out.println("Distanca je " + a.distanca(b));
    }
}
```

Konstruktori

- Objekti se konstruišu pomoću specijalnih metoda, *konstruktora* koji imaju identično ime kao i klasa.
- Svaki metod, uključujući i konstruktor, može se preopteretiti (*overload*), tj. definirati sa istim imenom samo različitom implementacijom i različitim ulaznim parametrima (slično kao C++)
- Default konstruktor, koji ne uzima parametre, biti će kreiran od strane kompajlera ako programer ne napravi niti jedan konstruktor. Ako programer napravi bilo koji konstruktor kompajler neće generirati default konstruktor
- Java konstruktor može pozivati drugi konstruktor iste klase pomoću ključne riječi `this` i parametara koje prima konstruktor koji se poziva.
- Članovi klase se obično inicijaliziraju unutar konstruktora. Članovi koji se ostave ne inicijalizirani poprimaju svoje default vrijednosti (kao za Java nizove)

Još o inicijalizaciji

- Članove je moguće inicijalizirati i eksplicitno, van konstruktor metoda, prilikom njihovog definiranja. npr

```
public class Tacka
{
        // ...
        private double x_ = 0.0;
        private double y_ = 1;
}
```

- Ova inicijlizacija vrši se prije poziva bilo kojeg konstruktora (tj. eksplicitno dodjeljena vrijednost će biti korištena umjesto default vrijednosti ako niti jedan konstruktor ne modificira vrijednost člana)

Ključna riječ `static` za članove

- `static` se koristi kao modifikator za članove i/ili metode
- Za članove klase, `static` se postavlja između modifikatora pristupa i tipa člana npr.
  - `private static int p = 0;`
- Ako je član deklariran kao statičan onda postoji samo jedna kopija tog člana za kompletnu klasu i svi objekti dijele tu kopiju. Za razliku od običnih članova, za koje svaki objekat ima svoju kopiju.
- Statičan član se može posmatrati kao globalna varijabla unutar neke klase, tj kao stanje dijeljeno između svih objekata neke klase)

Ključna riječ `static` za metode

- Za metode klase, `static` se postavlja između modifikatora pristupa i povratnog tipa metoda, npr:
  - `public static int metod1() { p++; }`
- Metod se treba deklarirati kao statički u slučajevima da metod operiše samo nad statičkim članovima klase.
- Metod nije moguće deklarirati kao statički ako poziva neki drugi metod koji nije statički ili ako pristupa nekom članu koji nije statički.
- Za pozivanje statičkih metoda nije potrebno kreirati niti jedan objekat klase. Ako je metod `metod1()` deklariran kao statički u klasi `Klasa1`, metod se poziva:
  - `Klasa1.metod1();`
- ili preko objekta klase `Klasa1`
  - `Klasa1 p = new Klasa1(); p.metod1();`

Primjer 2


```
public class Linija
{
    public Linija(Tacka a, Tacka b)
    {
      a_ = a;
      b_ = b;
    }
    public Tacka vratiA() { return a_; }
    public Tacka vratiB() { return b_; }
    public double duzina() { return a_.distanca(b_); }
    private Tacka a_ = new Tacka();
    private Tacka b_;

    public static void main(String[] args)
    {
      Linija linija = new Linija(new Tacka(2,3), new Tacka(5,4));
      Tacka c = linija.vratiA();
      System.out.println(linija.duzina());
      c.postaviX(10);
      System.out.println(linija.duzina());
    }
}
```