DIZAJN I RAZVOJ SOFTVERA

Kontrola toka: Ciklusi

Osnovno o klasama

Statičke klase

switch naredba

```
switch(izraz) {
    case vrijednost1:
         naredba1 ili blok_naredbi1;
         break;
    case vrijednost2:
         naredba2 ili blok_naredbi2;
         break;
         ...
    case vrijednostN:
         naredbaN ili blok_naredbiN;
         break;
    default:
         naredba ili blok naredbi;
}
```

- Tip izraza izraz može biti jedan od cjelobrojnih tipova `byte`, `short`, `int` ili `char`, a od Jave SE 7, to može biti i tip `String`.
- Zavisno od vrijednosti izraza, izvršavaju se odgovarajuće naredbe. Ako izraz nije jednak nijednoj od specificiranih vrijednosti, izvršava se `default` dio.

switch naredba – Primjeri

Ocjena studenta:

```
String poruka = "Student je dobio ocjenu
";
switch(ispit/10){
  case 10:
  case 9:
    poruka += "A";
    break;
  case 8:
    poruka += "B";
    break;
  case 7:
    poruka += "C";
    break;
  case 6:
    poruka += "D";
    break;
  case 5:
    poruka += "E";
    break;
  default:
    poruka += "F";
}
```

propadanje

U Javi, string nije niz `char` podataka, već je objekat klasnog tipa.

Operator **+** predstavlja nadovezivanje stringova.

`switch` sa stringom:

```
switch(prezimeStudenta) {
  case "Marković":
  case "Ilić":
  case "Petrović":
    System.out.println(imeStudenta);
}
```

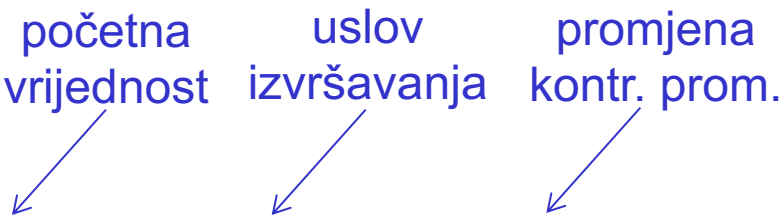
Ciklusi: while, do...while i for

while

```
while(izraz)
  naredba ili blokNaredbi;
```

do...while

```
do
  naredba ili blokNaredbi;
while(izraz);
```



for

```
for(izraz1; izraz2; izraz3)
  naredba ili blokNaredbi;
```

Primjer while petlje

```
import java.util.Scanner;

public class PrimjerWhile {

  public static void main(String[] args) {
    int broj, suma = 0, brojUnosa = 0;
    double aritmSred, aritmSredCast;
    Scanner unos = new Scanner(System.in);
    System.out.print("Unijeti prirodne brojeve (-1 prekida unos): ");
    broj = unos.nextInt();
    while(broj != -1) {
      suma += broj;
      brojUnosa++;
      broj = unos.nextInt();
    }
    aritmSred = suma/brojUnosa;
    aritmSredCast = (double)suma/brojUnosa;
    System.out.printf("Uneseno %d brojeva.\n", brojUnosa);
    System.out.printf("Aritmetička sredina je %.3f (sa cast-om %.3f)",
        aritmSred, aritmSredCast);
  }
}
```

Ispis

```
Unijeti prirodne brojeve (-1 prekida unos): 2
3
11
-1
Unešeno 3 brojeva.
Aritmetička sredina je 5.000 (sa cast-om 5.333)
```

Primjer for petlje

```
public class PrimjerFor {

  public static void main(String[] args) {

    System.out.printf("%8s%8s%8s\n", "Broj", "Kvadrat", "Kub");
    for(int broj=1; broj<=5; broj++)
      System.out.printf("%8d%8d%8d\n", broj, broj*broj, broj*broj*broj);
  }
}
```

Brojačka promjenljiva deklarisana u for petlji. Ne postoji van petlje.

Štampanje cijelog broja sa 8 polja (nedostajući karakteri se dopunjavaju spejsovima)

Ispis

Broj	Kvadrat	Kub
1	1	1
2	4	8
3	9	27
4	16	64
5	25	125

- Svi izrazi u zaglavlju `for` petlje su opcioni. Dozvoljeno je i

  `for( ; ; )`

  što predstavlja beskonačnu petlju.
- Kad se izostavi uslov petlje, Java podrazumijeva da je taj uslov ispunjen.

break i continue

- Naredba `break` je naredba bezuslovnog skoka.
- Pomoću `break` se izlazi iz ciklusa ili `switch` naredbe. Greška je ako se break nađe u nekom drugom dijelu koda.
- Kad se u `for` petlji naiđe na naredbu `continue`, preskaču se naredbe iz tijela petlje nakon `continue`, i nastavlja se sa sljedećom iteracijom petlje. U tom slučaju, brojačka petlja se ažurira u skladu sa trećim izrazom u zaglavlju `for` petlje.
- Naredba `continue` se može naći i u `while` i `do...while` petljama, kada se nakon te naredbe odmah prelazi na provjeru ispravnosti upravljačkog izraza (uslova petlje).

break i continue sa labelama

- Java ne podržava `goto` naredbu, ali postoji mogućnost da prekinemo izvršenje proizvoljne petlje, a ne samo one u kojoj se nalazi `break`.
- Slično važi za `continue`.

```
public class Test {
  public static void main(String[] args) {
    int niz[][] = {{12,34,101},{33,34,56},{201,202,203}};
    int i, j = 0;

    spoljna:
    for(i = 0; i < niz.length; i++) {
      for(j = 0; j < niz[i].length; j++) {
        if(niz[i][j] == 33)
          break spoljna;
      }
    }
    System.out.printf("Nađen broj 33 na poziciji (%d,%d).", i + 1, j + 1);
  }
}
```

Ispis

```
Nađen broj 33 na poziciji (2,1).
```

Uvod u klase – klasa Knjiga

podaci klase (atributi)

metode klase

```
public class Knjiga {

  private String ime;
  private int brojStrana;

  public String getIme() {
    return ime;
  }

  public void setIme(String s) {
    ime = s;
  }

  public int getBrojStrana() {
    return brojStrana;
  }

  public void setBrojStrana(int n) {
    brojStrana = n;
  }

  public void prikaziKnjigu(){
    System.out.printf("Ime knjige je %s i ima %d strana.\n",
        getIme(), getBrojStrana());
  }
}
```

mixedCase imenovanje metoda i podataka.

Podaci deklarisani kao **private** – može im se pristupiti samo u okviru iste klase. Na ovaj način se promoviše enkapsulacija.

Metode deklarisane kao **public** – može im se pristupiti spolja, iz bilo koje klase.

Testna klasa

```
import java.util.Scanner;

public class KnjigaTest {

   public static void main(String[] args) {
      Scanner unos = new Scanner(System.in);
      // Kreiranje objekta klase Knjiga
      Knjiga knjiga = new Knjiga();

      // Prikaz podrazumijevanog imena knjige i broja strana
      System.out.println("Podrazumijevano ime i broj strana: ");
      knjiga.prikaziKnjigu();

      // Unos imena knjige i broja strana
      System.out.println("Unesite ime knjige i broj strana: ");
      String imeKnjige = unos.nextLine(); // Čitanje linije teksta
      int brStr = unos.nextInt();
      knjiga.setIme(imeKnjige);
      knjiga.setBrojStrana(brStr);
      knjiga.prikaziKnjigu();
   }
}
```

Klasa koja manipuliše instancama drugih klasa se naziva **upravljačka klasa**.

Čitanje linije teksta do karaktera za prelazak u novi red. Vraća pročitane karaktere kao `String`, ne uključujući karakter za prelazak u novi red.

Postoji i metoda **`next()`**, koja čita do prve bjeline.

Ispis

```
Podrazumijevano ime knjige i broj strana:
Ime knjige je null i ima 0 strana.
Unesite ime knjige i broj strana:
Robinzon Kruso
234
Ime knjige je Robinzon Kruso i ima 234 strana.
```

static metoda main

- Klase koje su deklarisane kao `public` moraju biti smještene u posebnim fajlovima koji nose isto ime kao i klasa, i ekstenziju `java`.
- Klasa `Knjiga` nema metodu `main`, što znači da izvršenje programa ne može započeti iz te klase.
- Pomoću ključne riječi `static` u deklaraciji metode `main`, JVM locira i poziva ovu metodu, čime počinje izvršavanje Java aplikacije.
- `static` metode su posebne po tome što se mogu pozivati, a bez da je kreiran ijedan objekat klase u kojoj se nalaze static metode.
- `main` metoda se mora deklarisati kao statična jer se poziva prije kreiranja ijednog objekta predmetne klase (klasa `KnjigaTest` na prethodnom slajdu).
- Bilo koja klasa može da sadrži metodu `main`. Ako aplikacija sadrži više klasa, svaka od klasa može sadržati metodu `main`. Poziva se ona metoda `main` čija se klasa navede u `java` komandi.

Kreiranje instanci klase

- Novi objekat klase (instanca) se kreira pomoću ključne riječi `new`:

```
Knjiga knjiga = new Knjiga();
```

- Iako nema nikakvih argumenata unutar zagrada, zagrade su obavezne.
- Dio koda `Knjiga()` u prethodnoj liniji predstavlja poziv konstruktora, metode koja se koristi samo pri kreiranju objekta da ga inicijalizuje.
- Konstruktor mora imati isto ime kao i sama klasa, a može imati i argumente.
- Ako u realizaciji klase ne deklarišemo konstruktore, JVM će našoj klasi dodijeliti **podrazumijevani konstruktor**.
- Podrazumijevani konstruktor daje podacima klase podrazumijevane početne vrijednosti, koje su za primitivne tipove `byte`, `char`, `short`, `int`, `long`, `float` i `double` jednake 0, za tip `boolean` je `false`, dok je za referencijske tipove vrijednost `null`.
- Podaci klase dobijaju podrazumijevane vrijednosti, a lokalne promjenljive metoda ne.

Parametri i argumenti metoda

- Kada radimo sa metodama, pojmovi **parametar** i **argument** se često koriste kao sinonimi. Ipak, mi ćemo praviti razliku između njih.
- Parametrom ćemo zvati promjenljivu koja je deklarisana u zaglavlju metode, i koju metoda koristi u svom tijelu. Na primjer, u klasi `Knjiga`, parametar metode `setIme` je `String s`.
- Sa druge strane, argument se prosljeđuje metodi tokom izvršavanja aplikacije. Na primjer, argument metode `setIme` je promjenljiva `imeKnjige`, deklarisana i učitana u okviru metode `main` klase `KnjigaTest`. Ovaj argument je proslijeđen metodi `setIme` tokom izvršavanja.
- Prilikom poziva, vrijednosti parametara postaju jednake vrijednostima argumenata metode.
- Broj argumenata u pozivu metode mora biti jednak broju parametara metode. Tipovi argumenata u pozivu moraju biti konzistentni sa tipom odgovarajućih parametara. Konzistencija, u ovom smislu, ne znači da tipovi moraju biti isti, već da se tip svakog argumenta može konvertovati u tip odgovarajućeg parametra.

Kompajliranje aplikacije sa više klasa

- Da bi izvršili aplikaciju sa više klasa, potrebno je iskompajlirati sve klase aplikacije.
- Konkretno, u našem primjeru sa dvije klase `Knjiga` i `KnjigaTest`, obje klase se mogu istovremeno iskompajlirati na sljedeći način:

```
javac Knjiga.java KnjigaTest.java
```

- Sve klase u tekućem folderu se mogu iskompajlirati sa

```
javac *.java
```

import i paketi

- Ključnom riječju `import` smo na početku klase `KnjigaTest` uveli klasu `Scanner`.
- Klase `System` i `String` nismo morali uvoditi na ovaj način jer se one nalaze u paketu `java.lang`, koji sadrži osnovne klase potrebne svakoj Java aplikaciji.
- Kao takav, paket `java.lang` se implicitno uvodi u sve aplikacije.
- Za sve klase koje se kompajliraju u istom folderu se smatra da pripadaju istom paketu, tzv. **podrazumijevanom paketu**.
- Klase iz jednog paketa se implicitno uvode u izvorni kôd drugih klasa iz istog paketa, pa ih nije potrebno uvoditi ključnom riječju `import`. Iz tog razloga, klasu `Knjiga` nismo morali uvoditi u klasu `KnjigaTest`.

import i paketi

- Ako treba uvesti više klasa iz određenog paketa, za svaku od tih klasa treba navesti zasebnu `import` deklaraciju, jednu za drugom. Alternativa ovom pristupu je korišćenje džoker karaktera * čime se u aplikaciju uvode sve klase iz datog paketa.
- Na primjer, uvođenje svih klasa iz paketa `java.lang` bi se izvršilo sa

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

  Ovime se ne povećava veličina programa, tj. ne dodaju se sve klase iz predmetnog paketa našoj aplikaciji. Deklaracija `import` ukazuje u kom paketu treba tražiti klase koje koristimo i samo se te klase dodaju aplikaciju.
- Riječ `import` se ne mora koristiti ako se u deklaraciji umjesto riječi `Scanner` navede i ime paketa kojem pripada klasa `Scanner`. Na primjer, promjenljivu `unos` iz klase `KnjigaTest` možemo deklarisati sa

  ```
  java.util.Scanner unos = new java.util.Scanner(System.in);
  ```

  bez linije

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

Reference i pokazivači

- Promjenljive u Javi se dele na **primitivne** (`byte`, `char`, `short`, `int`, `long`, `float`, `double` i `boolean`) i **referencijske** (sve ostale).
- U promjenljive referencijskog tipa se smještaju memorijske adrese objekata. Referenca ukazuje na objekat u memoriji računara.
- Jedan objekat može da sadrži više referenci na njega.
- Reference su jako slične pokazivačima, koji postoje u jezicima C i C++, jer i pokazivači mogu da sadrže adresu objekata.
- **Java ne podržava rad sa pokazivačima!**
- Glavna razlika između referenci i pokazivača je da se sa referencama **ne može** raditi kao sa pokazivačima.
- Za razliku od pokazivača, referenca ne može ukazati na proizvoljnu memorijsku lokaciju, i ne mogu se vršiti elementarne operacije sa referencama. Ovo je ključno za bezbjednost Jave.
- Pokazivače možemo sabirati i oduzimati sa konstantama, pokazivače možemo oduzimati itd. čime možemo pristupiti proizvoljnoj memorijskoj lokaciji, što predstavlja potencijalnu opasnost.

Reference i objekti

- U liniji

  ```
  Knjiga knjiga = new Knjiga();
  ```

  smo u jednom koraku kreirali novi objekat klase `Knjiga`, referencijsku promjenljivu `knjiga` i u nju upisali referencu na kreirani objekat. Isto smo mogli postići sa:

  ```
  Knjiga knjiga;
  knjiga = new Knjiga();
  ```

- Kaže se još da je referencijska promjenljiva vezana (eng. *attached*) za objekat. Referencijska promjenljiva je vezana za objekat ako ga nedvosmisleno identifikuje.
- Ako se ne izvrši inicijalizacija pri kreiranju referencijske promjenljive, ona ima podrazumijevanu vrijednost `null`.
- Riječ `null` je rezervisana riječ koja označava referencu ni na jedan objekat.
- Kad završimo rad sa datim objektom i zaključimo da nam više neće trebati, možemo razdvojiti (eng. *detach*) referencijsku promjenljivu od objekta upisivanjem vrijednosti `null` u nju.

Objekat, referenca, entitet

- Pored pojma objekta i reference, u OO programiranju se koristi pojam **entitet**.
- Entitet predstavlja ime u programskom kôdu koje se odnosi na objekat tokom izvršavanja programa, tj. koje je u svakom trenutku izvršavanja programa vezano za neki objekat. Entitet predstavlja generalizaciju tradicionalnog pojma promjenljive, koja odgovara OO okruženju.
- Pojasnimo razliku između pojmova objekta, reference i entiteta.
  - Objekat je pojam vezan za vrijeme izvršavanja (eng. *run time*). Svaki objekat predstavlja instancu određene klase, kreiran tokom izvršavanja i sastavljen od određenog broja polja (podataka).
  - Referenca je takođe pojam vezan za vrijeme izvršavanja, i predstavlja vrijednost koja ukazuje na objekat u memoriji. Preko referencijske promjenljive, koja sadrži vrijednost reference, ostvarujemo interakciju sa objektom.
  - Entitet predstavlja statički pojam, tj. tiče se programskog kôda. Entitet je identifikator koji se pojavljuje u tekstu klase, i predstavlja jednu ili niz vrijednosti objekata za vrijeme izvršavanja programa.
- Ako je x entitet referencijskog tipa, za njega kažemo da je vezan za objekat O ako je njegova vrijednost tokom izvršavanja referenca na taj objekat. Mi ćemo, jednostavnosti radi, umesto pojma entiteta koristiti tradicionalni pojam promjenljive.

Operacije sa referencama

- Referencijska promjenljiva može mijenjati svoju vrijednost tokom izvršenja programa. Ukoliko se ne inicijalizuje, referencijska promjenljiva ima vrijednost `null`. Vezivanjem za objekat ili razdvajanjem od njega, ona mijenja svoju vrijednost.
- U nastavku dajemo koje operacije su dozvoljene sa referencijskim promjenljivim.
- Dozvoljene operacije sa referencama su:
  - Vezivanje referencijske promjenljive za objekat

    Referenca se veže za objekat naredbom tipa `Klasa var = new Klasa();`
  - Razdvajanje referencijske promjenljive od objekta

    Referenca se razdvaja od objekta upisivanjem `null` u referencijsku promjenljivu. Na ovaj način, referenca nije vezana ni za jedan objekat u memoriji.
  - Poređenje referencijskih promjenljivih

    Dvije reference se mogu porediti koristeći operator ==, na potpuno isti način kao primitivni tipovi. U tom slučaju, samo utvrđujemo da li su reference vezane za isti objekat u memoriji.

Operacije sa referencama

- Poređenje referencijskih promjenljivih

```
Knjiga knjiga1 = new Knjiga(), knjiga2 = new Knjiga(), knjiga3;
knjiga3 = knjiga2;

if(knjiga1 == knjiga2)
 System.out.println("Reference knjiga1 i knjiga2 su iste");
else
 System.out.println("Reference knjiga1 i knjiga2 nisu iste");

if(knjiga3 == knjiga2)
 System.out.println("Reference knjiga3 i knjiga2 su iste");
else
 System.out.println("Reference knjiga3 i knjiga2 nisu iste");
```

Promjenljive `knjiga1` i `knjiga2` nisu iste, jer su vezane za različite objekte, dok su `knjiga3` i `knjiga2` iste jer smo sa `knjiga3 = knjiga2;` vezali `knjiga3` za isti objekat na koji ukazuje `knjiga2`. Ovom naredbom nije kreiran novi objekat, već smo postigli da su dvije ref. promjenljive vezane za isti objekat.

Operacije sa referencama

- ## Poređenje jednakosti objekata

  Dva objekta su jednaka ako su im odgovarajući atributi jednaki, tj. poklapaju im se vrijednosti primitivnih promjenljivih, i odgovarajuće referencijske promjenljive vezane za iste objekte. U tom smislu, provjera jednakosti objekata se ne može izvršiti operatorom ==.

- ## Kloniranje objekata

  U OO terminologiji, postoji koncept **kloniranja objekta**. Kloniranje predstavlja kreiranje novog objekta određene klase, sa identičnim vrijednostima atributa kao predmetni objekat te klase. Metoda koja bi klonirala objekat neke klase, uzimala bi taj objekat kao argument i vraćala referencu na objekat iste klase, sa identičnim vrijednostima atributa kao proslijeđeni objekat

- ## Kopiranje objekata

  Vrlo sličan konceptu kloniranja je koncept **kopiranja objekata**. Razlika je u tome da objekat u koji želimo da kopiramo predmetni objekat (tj. ciljni objekat) već postoji.

Upućivanje poruka. Klijenti i serveri

- U okviru metode `main` klase `KnjigaTest` poziva se metoda `prikaziKnjigu()` klase `Knjiga` preko referencijske promjenljive `knjiga`:

```
knjiga.prikaziKnjigu();
```

- Klasa koja sadrži poziv metode neke klase predstavlja **klijenta** te klase. U našem slučaju, klasa `KnjigaTest` predstavlja klijenta klase `Knjiga`.
- Nasuprot tome, klasa `Knjiga` je **server** klase `KnjigaTest`.
- Pozivanje metode neke klase u okviru druge klase se još naziva i **upućivanje poruke** (eng. *message passing*). Dakle, klasa `KnjigaTest` upućuje poruku klasi `Knjiga` pozivanjem metode `prikaziKnjigu()`.
- Objekat klase može da mijenja svoje stanje, predstavljeno atributima, kada se pozove njegov metod, tj. kada mu se uputi poruka.

Konstruktori klase

- Konstruktor klase, ili samo konstruktor, je specijalna metoda koja se poziva pri kreiranju objekta da inicijalizuje objekat.
- Konstruktor mora imati isto ime kao i sama klasa, a može imati i argumente.
- Druga važna razlika između konstruktora i ostalih metoda je da konstruktor ne vraća vrijednost, čak se ne može deklarisati ni kao `void`.
- Ukoliko naša realizacija klase ne uključuje konstruktor klase, kompajler obezbjeđuje podrazumijevani konstruktor, koji nema parametara, i koji podacima objekta daje podrazumijevane vrijednosti.
- Ukoliko kreiramo bar jedan konstruktor, JVM neće kreirati podrazumijevani konstruktor!
- Klasa može sadržati više konstruktora, što je poznato kao **preklapanje konstruktora** (eng. *constructor overloading*), o čemu će biti više riječi kasnije.

Konstruktori klase

Konstruktori klase `Knjiga`:

```
public Knjiga(String imeKnjige, int brStr) {
   ime = imeKnjige;
   brojStrana = brStr;
}

public Knjiga(String imeKnjige) {
   ime = imeKnjige;
}

public Knjiga(int brStr) {
   brojStrana = brStr;
}

public Knjiga() {
   brojStrana = 0;
   ime = null;
}
```

Konstruktori klase - Primjer

```
public class KnjigaTest2 {

   public static void main(String[] args) {
         // Kreiranje objekta klase Knjiga
         Knjiga knjiga1 = new Knjiga("Robinzon Kruso",234);
         Knjiga knjiga2 = new Knjiga("Robinzon Kruso");
         Knjiga knjiga3 = new Knjiga(234);
         Knjiga knjiga4 = new Knjiga();

         knjiga1.prikaziKnjigu();
         knjiga2.prikaziKnjigu();
         knjiga3.prikaziKnjigu();
         knjiga4.prikaziKnjigu();
   }
}
```

Ispis

```
Ime knjige je Robinzon Kruso i ima 234 strana.
Ime knjige je Robinzon Kruso i ima 0 strana.
Ime knjige je null i ima 234 strana.
Ime knjige je null i ima 0 strana.
```

Modularnost

- Jedni od ključnih principa OO programiranja su **proširivost** (eng. *extendibility*) i **ponovna upotreba kôda** (eng. *code reusability*).
- Proširivost predstavlja sposobnost softvera da se jednostavno prilagodi promjenama u njegovoj specifikaciji.
- Ponovna upotreba kôda predstavlja mogućnost da postojeći softverski elementi mogu poslužiti pri razvoju drugih aplikacija.
- Ova dva pojma se zajedno tretiraju kao **modularnost programskog koda**.
- Java API ili Javina klasna biblioteka sadrži veliki broj klasa i pripadajućih metoda, uključujući metode za matematičke proračune, rad sa stringovima i karakterima, ulazno/izlazne operacije, rad sa bazama podataka, rad sa fajlovima, mrežne aplikacije i druge operacije.
- Javina API specifikacija (Java Platform SE 15) se može naći na adresi

  http://docs.oracle.com/javase/15/docs/api/
- Poznavanje postojećih Javinih klasa i metoda značajno može ubrzati razvoj aplikacije.

Modularnost

- **Metode** (u drugim jezicima se nazivaju i **rutine**, **procedure** ili **funkcije**) predstavljaju sredstvo modularizacije programa u smislu razdvajanja pojedinačnih zadataka u posebne jedinice.
- Metode su izolovane od ostatka programa, edituju se na jednom mjestu, mogu se koristiti proizvoljan broj puta sa različitih lokacija u programu.
- Programe treba modularizovati da bi se složeniji zadaci podijelili na niz manjih (princip podijeli pa vladaj), što softverski dizajn čini lakšim. Lakše je riješiti problem uklapajući male jednostavne cjeline nego rješavati ga od početka. Na ovaj način se promoviše i ponovna upotreba koda.
- U tom smislu, preporučuje se upotreba postojećih Javinih metoda gdje god je to moguće.
- Modularizacijom programa se izbjegava ponavljanje koda, što predstavlja čest izvor grešaka pri editovanju programa. Greške se mnogo lakše uklanjaju i program se mnogo lakše održava ako je podijeljen na manje izolovane cjeline.

Statičke metode

- Jedan način pozivanja metoda klase je preko referencijske promjenljive u koju je upisana referenca na kreiranu instancu. Ovo nije i jedini način.
- Ponekad se metoda može pozvati bez da je kreirana ijedna instanca klase. Ovo je slučaj sa metodom `main`, koja se poziva prije nego je kreirana ijedna instanca predmetne klase.
- Metode koje se mogu pozvati bez prethodnog kreiranja instanci klase se nazivaju **statičkim** ili **klasnim** metodama, i deklarišu se pomoću ključne riječi **static**. Statičke metode se pozivaju na sledeći način:

```
ImeKlase.imeMetode(argumenti)
```

- Uzmimo, na primjer, klasu `Math` koja sadrži niz korisnih matematičkih metoda (neke su navedene na sljedećem slajdu). Ova klasa je dio paketa `java.lang`, pa se ne mora eksplicitno uvoditi `import` deklaracijom.
- Sve metode iz `Math` klase su statičke, pa se ove metode mogu pozvati bilo gdje u našem kodu na sljedeći način:

```
Math.imeMetode(argumenti)
```

Neke metode iz klase Math

Metoda	Opis
`abs(x)`	apsolutna vrijednost broja x
`sqrt(x)`	kvadratni korijen broja x
`sin(x)`	sinus broja x
`cos(x)`	kosinus broja x
`tan(x)`	tangens broja x
`ceil(x)`	prvi veći cio broj od x
`floor(x)`	prvi manji cio broj od x
`exp(x)`	eksponent broja x, tj. e^x
`log(x)`	prirodni logaritam broja x
`min(x,y)`	manji od x i y
`max(x,y)`	veći od x i y
`pow(x,y)`	x na y, tj. x^y

- Na primjer, kvadratni koren broja 891.2 i treći stepen broja 12.1 se mogu dobiti sa:

```
Math.sqrt(891.2)
Math.pow(12.1,3)
```

Statički podaci

- Pored statičkih metoda, postoje i statički podaci klase.
- Na primjer, matematičke konstante `Math.PI` i `Math.E` (π i e) su deklarisane u klasi `Math` kao `public`, `final` i `static`.
- Ključna riječ `final` označava da su te vrijednosti konstante, tj. da se ne mogu mijenjati. Ključna riječ `static` znači da se ovim podacima može pristupiti preko imena klase i operatora tačka, tj. kao `Math.PI` i `Math.E`.
- Promjenljive koje nisu deklarisane kao statičke, kao što su `ime` i `brojStrana` u klasi `Knjiga`, se nazivaju **promjenljivima instanci** (eng. *instance variables*) jer svaka instanca klase ima svoju kopiju ovih promjenljivih.
- Sa druge strane, statičke promjenljive ili promjenljive klase, nemaju svoje kopije za pojedinačne instance, već sve instance dijele jednu kopiju statičke promjenljive.
- Ova dva tipa promjenljivih čine **podatke klase** ili **polja klase** (eng. *class fields*).
- Pomoću statičkih promjenljivih možemo, recimo, da brojimo koliko je kreirano instanci date klase. Po kreiranju svake instance, konstruktor klase uvećava vrijednost statičke promjenljive za 1.

Pristupanje statičkim članovima klase

- Ako u klasi postoji više statičkih metoda, jedna drugu mogu zvati direktno preko imena, i mogu manipulisati statičkim podacima te klase direktno preko imena.
- Sa druge strane, statička metoda ne može pristupiti direktno nestatičkim članovima klase, već to mora uraditi preko reference na instancu klase.
- Uzmimo primjer statičke metode `maks` koja vraća maksimalni od 3 argumenta. Recimo da ova metoda pripada klasi `Maksimum3Broja`.

```
public static double maks(double a, double b, double c){
    double m = a;
    if(b > m) m = b;
    if(c > m) m = c;
    return m;
}
```

- Bilo koja statička metoda iz klase `Maksimum3Broja`, recimo metoda `main`, može pozvati ovu metodu samo preko imena `maks(x,y,z)`.
- Sve druge klase koje bi koristile klasu `Maksimum3Broja` bi morale pozvati tu metodu kao `Maksimum3Broja.maks(x,y,z)`.
- `Math.max(Math.max(x,y),z)` radi isto (ponovna upotreba koda!)

Pristupanje statičkim članovima klase

- Zašto statička metoda ne može da pozove nestatičke metode direktno?
- Tokom izvršavanja aplikacije, može postojati veliki broj instanci date klase, svaka sa svojim podacima. Statička metoda ne može da zna na koju instancu bi se primijenio nestatički metod pozvan direktno, bez reference na konkretnu instancu.
- Da bi pristupili `public static` članu klase kad nije kreiran nijedan objekat klase, koristimo ime klase, operator tačka i ime promjenljive ili metode (`Math.PI` ili `Math.random()`). Ovako se može pristupati i kad ima kreiranih objekata klase.
- Da bi pristupili `private static` članu klase kad nije kreiran nijedan objekat klase, potrebno je kreirati `public static` metodu kojoj ćemo pristupiti preko imena klase.

Osnovno o stringovima

- Posmatrajmo naredbu koja bi štampala rezultat metode `maks`:

```
System.out.println("Maksimum brojeva " + maks(x, y, z));
```

- String koji se štampa nastaje nadovezivanjem fiksnog stringa `"Maksimum brojeva "` i rezultata koji vraća metoda `maks`.
- Operator + vrši nadovezivanje stringova u Javi. Ukoliko su oba operanda ove operacije objekti tipa `String`, nadovezivanjem se kreira novi string dobijen kada se drugi string nadoveže na prvi.
- Jednom kreiran `String` objekat se ne može mijenjati!
- Svaki put kad treba izvršiti neku modifikaciju stringa (promjena karaktera, nadovezivanje), kreira se novi `String` objekat koji sadrži modifikaciju. Dakle, u naredbi `S += P`, gdje su `S` i `P` objekti tipa `String`, `P` se neće jednostavno nadovezati na `S`, već se kreira novi string `S` koji sadrži ova dva stringa nadovezana. Originalni `S` se ne mijenja.
- Razlog ovome je efikasnost rada – nepromjenljivi stringovi se efikasnije implementiraju od promjenljivih.

String reprezentacija objekta

- Svi primitivni tipovi i objekti u Javi imaju `String` reprezentaciju!
- Vrijednosti primitivnih tipova i objekata se prije nadovezivanja konvertuju u odgovarajući string. Na primjer, `double` vrijednost 5.8700 bi se konvertovala u string `"5.87"` (odbacuju se prateće nule).
- Primitivne numeričke vrijednosti se konvertuju u odgovarajuće stringove, dok se `boolean` vrijednosti konvertuju u string `"true"` ili `"false"`.
- Svi Javini objekti imaju **`toString`** metod koji vraća `String` reprezentaciju objekta. Kad se objekat koristi pri nadovezivanju stringova, njegova `toString` metoda se poziva implicitno. Može se pozvati i eksplicitno.
- Pri nadovezivanju stringova, treba povesti računa o asocijativnosti operatora `+` (s lijeva na desno). Na primjer, izraz

```
"Rezultat je " + 4 + 7
```

će vratiti string `"Rezultat je 47"`.

- Ako želimo da prvo izvršimo sabiranje, potrebno je zagradama odvojiti ono što želimo da se zasebno izvrši, tj.

```
"Rezultat je " + (4 + 7)
```

Konverzija argumenata

- Pri pozivu metode, može se desiti da tipovi argumenata ne odgovaraju tipovima parametara metode. Na primjer, metoda `maks`, koja ima tri `double` parametra, se može pozvati i kao `maks(1,2,3)`, iako ona očekuje tri realna broja kao argumente.
- Ukoliko se ovo desi, dolazi do tzv. **implicitne konverzije argumenata**, tj. do konvertovanja tipa argumenta u tip parametra.
- Može se desiti da konverzija nije dozvoljena, što dovodi do grešaka u kompajliranju.
- Pri konverziji može doći i do gubitka podataka. Na primjer, pri konverziji `double` vrijednosti u `int`, odbacuje se decimalni dio broja. Obrnuto, pri konverziji `int` u `double`, ne dolazi do gubitka podataka. Takođe, pri konverziji `int` u `short`, može doći do promjene vrijednosti podataka zbog većeg opsega tipa `int`.
- Slično se dešava kada imamo izraz koji sadrži operande istog tipa. Prost primjer je dijeljenje dva cijela broja koje će rezultirati cijelim brojem. Ako su svi operandi jednog tipa, onda će i tip rezultata biti tog tipa.

Pravila unapređenja

- **Pravila unapređenja** (eng. *promotion rules*) se primjenjuju na izraze koji sadrže dva ili više primitivna tipa, kao i na primitivne tipove koji se prosljeđuju kao argumenti kod metoda.
- Svaka vrijednost je unapređena u najviši tip koji figuriše u izrazu. Pri izračunavanju izraza, ne dolazi do promjene tipa promjenljivih koje figurišu u izrazu, već se pravi privremena kopija vrijednosti svake promjenljive.
- Pravila unapređenja primitivnih tipova su data u tabeli.

Tip	Tip unapređenja
`double`	Nijedan
`float`	double
`long`	float ili double
`int`	long, float ili double
`char`	int, long, float ili double
`short`	int, long, float ili double (ne i char)
`byte`	short, int, long, float ili double (ne i char)
`boolean`	Nijedan (boolean vrijednosti nisu numeričke u Javi)

Pravila unapređenja

- Konvertovanje vrijednosti u tipove niže od ovih naznačenih u tabeli može rezultirati u promjeni vrijednosti ukoliko niži tip ne može predstaviti vrijednost višeg tipa. Na primjer, ako bi pokušali da smjestimo vrijednost `int` promjenljive od 50000 u `short` promjenljivu, došlo bi do gubitka tog broja jer je maksimalan broj koji se može smjestiti u `short` promjenljivu 32767.
- U slučaju gdje može doći do gubitka podatka, Javin kompajler zahtijeva da se koristi operator eksplicitne konverzije, tj. `cast` operator. U suprotnom, dolazi do greške komajliranja. Uzmimo, na primjer, metodu maks čiji su parametri tipa `int`:

```
int maks(int a, int b, int c)
```

- Ukoliko bi ovu metodu pozvali sa

```
maks(2,3,4.5)
```

došlo bi do greške kompajliranja jer je treći argument višeg tipa od očekivanog i može doći do gubitka tačnosti. U tom slučaju, moramo naglasiti da smo sigurni da želimo da proslijedimo tu vrijednost, tj. da smo svesni gubitka tačnosti, navođenjem `cast` operatora ispred trećeg argumenta:

```
maks(2,3,(int)4.5)
```

Opseg deklaracija. Zasjenjivanje

- Opseg deklaracije metode ili entiteta (podatka klase, parametra metode, lokalne promjenljive) predstavlja dio programa iz kog možemo pristupiti toj metodi ili entitetu. Osnovna pravila vezana za opseg su:
  - Opseg deklaracije parametra metode je tijelo metode.
  - Opseg deklaracije lokalne promjenljive je od tačke gdje je promjenljiva deklarisana do kraja predmetnog bloka.
  - Opseg lokalne promjenljive koja se deklariše u zaglavlju (dio za inicijalizaciju) `for` petlje je tijelo petlje i ostale naredbe u zaglavlju petlje.
  - Opseg metode ili podatka klase je tijelo klase.
- Promjenljive se mogu deklarisati u okviru bilo kog bloka. Ako lokalna promjenljiva ili parametar metode imaju isto ime kao podatak klase, podatak klase je sakriven do završetka bloka. Ova se pojava naziva **zasjenjivanje** (eng. *shadowing*). Primjer zasjenjivanja je dat na sljedećem slajdu.
- U metodi se ne mogu deklarisati dve istoimene promjenljive u ugnježdenim blokovima `{ int x; { int x;}}`, za razliku od C i C++.
- Sa druge strane, dozvoljeno je imati dvije ili više `for` petlji sa istoimenom lokalnom promjenljivom.

```
public class Zasjenjivanje {
  private static int x = 50;

  public static void main(String[] args) {
    int x = 3;
    System.out.printf("Lokalna (main): %d\n",x);
    prva();  druga();  treca(33);
    System.out.printf("Lokalna (main): %d\n",x);
  }

  public static void prva(){
    int x = 10;
    x++;
    System.out.printf("Lokalna (prva): %d\n",x);
  }

  public static void druga(){
    x++;
    System.out.printf("Podatak klase (druga): %d\n",x);
    for(int x = 2; x < 10; x+=2)
      System.out.printf("Lokalna (for petlja u metodi druga): %d\n",x);
    System.out.printf("Podatak klase (druga): %d\n",x);
  }

  public static void treca(int x){
    x++;
    System.out.printf("Parametar (treca): %d\n",x);
  }
}
```

```
Lokalna (main): 3
Lokalna (prva): 11
Podatak klase (druga): 51
Lokalna (for petlja u metodi druga): 2
Lokalna (for petlja u metodi druga): 4
Lokalna (for petlja u metodi druga): 6
Lokalna (for petlja u metodi druga): 8
Podatak klase (druga): 51
Parametar (treca): 34
Lokalna (main): 3
```

Ispis

Preklapanje metoda

- **Preklapanje metoda** (eng. *method overloading*) je situacija kada više metoda jedne klase ima isto ime.
- Metode se mogu preklapati pod uslovom da imaju različitu listu parametara. Listu parametara određuje broj, tip i redosljed parametara.
- Pri pozivu preklopljene metode, kompajler, na osnovu broja, tipova i redosljeda proslijeđenih argumenata, bira odgovarajuću metodu.
- Preklapanje metoda se obično koristi za kreiranje metoda koje izvršavaju iste ili slične zadatke, ali sa različitim argumentima. Na primjer, metoda `abs` iz klase `Math` je preklopljene u četiri verzije:

```
public static int abs(int a)
public static long abs(long a)
public static float abs(float a)
public static double abs(double a)
```

Preklapanje i potpis metode

- Kombinacija imena metode sa brojem, tipom i redosljedom parametara predstavlja **potpis metode** (eng. *method's signature*).
- Kompajler pravi razliku između preklopljenih metoda na osnovu potpisa metoda, jer se samo na osnovu imena ne može napraviti razlika.
- Interno, kompajler koristi duža imena metoda koja uključuju ime metode, tip i redosljed parametara. Ako se ovakva produžena imena u potpunosti slažu, kompajler zaključuje da je došlo do konflikta pri preklapanju. U tom smislu, metode čiji su potpisi

```
int fun(int a, double b)
int fun(double a, int b)
```

se međusobno razlikuju i dozvoljene su sa stanovišta preklapanja.

- Pri odlučivanju da li je došlo do preklapanja, ne posmatra se tip vraćene vrijednosti, tj. preklopljene vrijednosti se ne mogu razlikovati na osnovu tipa vraćene vrijednosti. U tom smislu, deklaracija metoda

```
int fun(int a, double b)
double fun(int a, double b)
```

nije dozvoljena.

Preklapanje – Problemi oko razrješenja

- Posmatrajmo preklapanje

```
int fun(int a, double b)
int fun(double a, int b)
```

- Poziv `fun(3,4)` u prethodnom slučaju bi predstavljao sintaksnu grešku jer JVM ne može da odredi koju od metoda da pozove. Pošto nema potpunog poklapanja tipova argumenata sa parametrima, gleda se da li postoji parametar koji je istog tipa za sve metode. Ukoliko takav ne postoji, dolazi do greške. U prethodnom slučaju, i prvi i drugi parametri metoda su različitog tipa (`int` i `double`, `double` i `int`), te dolazi do greške.
- Ukoliko postoji parametar koji je istog tipa za sve metode, on se eliminiše iz razmatranja i posmatraju se ostali parametri kako bi se razriješilo koju metodu treba zvati. Ukoliko nema potpunog poklapanja sa ostalim parametrima, procedura se ponavlja, tj. gleda se da li postoji parametar koji je istog tipa za sve metode itd.

Preklapanje – Problemi oko razrješenja

- Posmatrajmo sljedeće preklapanje:

```
int fun(int a, double b, double c)
int fun(int a, int b, double c)
int fun(double a, int b, double c)
```

```
int fun(int a, double b, double c)
int fun(int a, int b, double c)
int fun(double a, double b, int c)
```

- Pozivom `fun(3,4,5)` u prvom preklapanju (plava pozadina) bi se pozvala druga metoda (`int-int-double`). Pošto je treći parametar istog tipa kod sve tri metode, tj. `double`, on se eliminiše iz razmatranja i posmatraju se ostala dva parametra. Kod druge metode, ostala dva se potpuno poklapaju sa tipom argumenata (dva `int`-a) i time je razriješen poziv preklopljene metode.
- U drugom preklapanju (crvena pozadina) dolazi do greške, jer ne postoji parametar istog tipa za sve tri metode.