DIZAJN I RAZVOJ SOFTVERA

Rad sa stringovima

Regularni izrazi

Nešto više o klasama i objektima

Enumeracije

Klasa String. Konstruktori

- U Javi, stringovi se najčešće predstavljaju pomoću klase `String`.
- Način deklarisanja i inicijalizacije stringova koji smo koristili do sada je:

```
String s = "Petak, 9. mart";
```

- Pored ovog načina, možemo koristiti i standardni način kreiranja stringa pomoću ključne riječi `new`:

```
String s = new String("Petak, 22. februar");
```

- Novi string se može inicijalizovati postojećim stringom:

```
String prvi = "Programiranje";
String drugi = new String(prvi);
```

- Novi string se može inicijalizovati i nizom `char` vrijednosti, ili njegovim podnizom:

```
char prvi[] = {'P', 'e', 't', 'a', 'k'};
String drugi = new String(prvi);
String treci = new String(prvi,1,3); // treci = "eta"
```

- Jednom kreiran `String` objekat se ne može promijeniti. Klasa `String` je deklarisana kao `public final`.

Metode length, charAt i substring

- Dužina stringa se dobija pomoću metode `length()`, npr. `prvi.length()`.
- `charAt` metoda vraća karakter (podatak tipa `char`) sa određene pozicije u stringu. Na primjer, posljednji karakter stringa `prvi` se dobija na sljedeći način:

```
prvi.charAt(prvi.length()-1);
```

- Kao i kod nizova, indeksiranje elemenata stringa počinje od 0. U slučaju nepostojeće pozicije, baca se `StringIndexOutOfBoundsException` izuzetak.
- Pomoću metode `substring`, možemo izdvojiti dio stringa iz postojećeg stringa. Postoje dvije preklopljene verzije ove metode:

```
String prvi = "Petak, 22. februar";
String drugi = prvi.substring(7);    // drugi = "22. februar"
String treci = prvi.substring(7,10); // treci = "22."
```

- Prva verzija, koja prima jedan argument, vraća podstring stringa `prvi` počev od osmog karaktera, dok druga, koja prima dva argumenta, vraća podstring od 3 karaktera, počev od osmog karaktera.
- Obje verzije vraćaju objekat tipa `String`, pa se mogu iskoristiti u inicijalizaciji drugih stringova.

Poređenje stringova

- Stringovi se mogu porediti metodama `equals`, `equalsIgnoreCase`, `compareTo`, `regionMatches` i pomoću operatora `==`.
- Metoda `equals` vrši leksikografsko poređenje stringova, pri čemu se pravi razlika između malih i velikih slova. Koristi se na sljedeće načine:

```
s1.equals(s2);
s1.equals("program");
```

  i vraća `true` ako su stringovi isti, i `false` u suprotnom.
- Metoda `equalsIgnoreCase` leksikografski poredi stringove, ali ne pravi razliku između malih i velikih slova. Koristi se isto kao metoda `equals`.
- Metoda `compareTo` je deklarisana u interfejsu `Comparable` i implementirana je u klasi `String`. Metoda se poziva kao:

```
s1.compareTo(s2)
```

  i vraća 0 ako su `s1` i `s2` jednaki, negativan broj ako je `s1<s2`, i pozitivan broj ako je `s1>s2`.

Poređenje stringova. startsWith i endsWith

- Metoda `regionMatches` poredi djelove dva stringa. Pogledati dokumentaciju za bliže objašnjenje.
- Konačno, operator `==` poredi reference, a ne sadržaj stringova. Preciznije, izraz `s1==s2` će vratiti `true` samo ako `s1` i `s2` ukazuju na isti objekat u memoriji, i `false` u suprotnom.
- Metode `startsWith` i `endsWith` su korisne ako želimo da provjerimo da li dati string počinje ili se završava nekim podstringom. Metode vraćaju `true` ako to jeste slučaj i `false` u suprotnom. Na primjer, za string

  ```
  s = "Program"
  ```

  metode

  ```
  s.startsWith("Pro")
  s.endsWith("ram")
  ```

  će obje vratiti `true`.
- Ove metode mogu imati još jedan cjelobrojni parametar koji predstavlja poziciju u stringu od koje počinjemo poređenje.

indexOf i lastIndexOf

- Metode `indexOf` i `lastIndexOf` vraćaju poziciju prve i posljednje pojave traženog karaktera ili podstringa u datom stringu. Ukoliko traženi karakter ili string ne postoje, vraća se -1.
- Posmatrajmo sljedeći niz naredbi:

```
String s = "banana";
A = s.indexOf('a');
B = s.lastIndexOf('a');
C = s.indexOf("na");
D = s.lastIndexOf("na");
E = s.lastIndexOf("ena");
```

Nakon ovog niza naredbi, promjenljive `A`, `B`, `C`, `D` i `E` će imati vrijednosti `1`, `5`, `2`, `4` i `-1`, respektivno.

- Ove metode mogu imati još jedan cjelobrojni parametar koji predstavlja poziciju u stringu od koje počinjemo pretragu.

Nadovezivanje stringova

- Već smo vidjeli da se nadovezivanje stringova može vršiti operatorom +.
- Pored ovog operatora, postoji i metoda `concat`, koja se koristi na sljedeći način:

```
s1.concat(s2)
```

- Iako je za očekivati da će se string `s1` promijeniti nakon ove naredbe, to se neće desiti. Jednom kreiran objekat klase `String` se ne može promijeniti.
- Ovo je ilustrovano na sljedećem primjeru:

```
String s1 = "Nova ", s2 = "godina", s3;
s3 = s1.concat(s2);
System.out.printf("s1 = %s\ns2 = %s\ns3 = %s",s1,s2,s3);
```

```
s1 = Nova
s2 = godina
s3 = Nova godina
```

Ispis

Još metoda iz klase String

- `replace` mijenja jedan karakter drugim u čitavom stringu.
- `toLowerCase` sva velika slova prebacuje u mala, ostale karaktere ne mijenja.
- `toUpperCase` sva mala slova prebacuje u velika, ostale karaktere ne mijenja.
- `trim` uklanja sve bjeline kojima počinje i završava dati string.
- Pošto se jednom kreiran `String` objekat ne može promijeniti, metode `replace`, `toLowerCase`, `toUpperCase` i `trim` vraćaju kopiju stringa sa izvršenom izmjenom.
- `valueOf` je statička metoda klase `String` koja vraća `String` reprezentaciju argumenta, koji može biti tipa `boolean`, `char`, `int`, `long`, `float`, `double`, kao i niz `char` vrijednosti.
- Za suprotnu operaciju, konverziju stringa u određenu numeričku vrijednost, koriste se metode `parseBoolean`, `parseInt`, ..., `parseDouble`, iz omotačkih klasa `Boolean`, `Integer`, ..., `Double`.

Klasa StringBuilder. Konstruktori

- Klasa `StringBuilder` služi za rad sa promjenljivim stringovima, što nije dozvoljavala klasa `String`.
- Interno, `StringBuilder` objekti se tretiraju kao nizovi promjenljive dužine koji sadrže sekvencu karaktera.
- Postoje 4 konstruktora klase `StringBuilder`, od kojih ćemo pokazati 3:

```
StringBuilder s = new StringBuilder();
StringBuilder s = new StringBuilder(10);
StringBuilder s = new StringBuilder("Petak i Java");
```

- Konstruktor bez argumenata pravi `StringBuilder` objekat bez karaktera i sa podrazumijevanim kapacitetom od 16 karaktera.
- Ako se navede cio broj u pozivu konstruktora, taj broj će predstavljati kapacitet `StringBuilder` objekta.
- Ako se u konstruktoru navede string, kreira se `StringBuilder` objekat sa karakterima tog stringa i kapacitetom jednakom dužini stringa uvećanom za 16.

Dužina i kapacitet StringBuilder-a

- Metoda `length` vraća dužinu `StringBuilder` stringa (broj karaktera).
- Metoda `capacity` vraća broj karaktera koji se može smjestiti u `StringBuilder` objekat bez alokacije dodatne memorije.
- Metoda `ensureCapacity` osigurava da kapacitet `StringBuilder` stringa bude jednak argumentu te metode.
- Metoda `setLength` povećava ili smanjuje dužinu `StringBuilder` stringa. Ako je argument metode manji od dužine `StringBuilder` stringa, dolazi do odsijecanja karaktera. U suprotnom, ako je argument metode veći od dužine `StringBuilder` stringa, dolazi do nadovezivanja `null` karaktera (`'\0'`) na kraj tog stringa do tražene dužine.

```
StringBuilder s1 = new StringBuilder("Java");
StringBuilder s2 = new StringBuilder("Java");
System.out.printf("Dužina %d, kapacitet %d\n",s1.length(), s1.capacity());
s1.setLength(8);
s2.setLength(2);
System.out.printf("String veće dužine %s, a manje %s\n",s1,s2);
```

```
Dužina 4, kapacitet 20
String veće dužine Java    , a manje Ja
```

Ispis

StringBuilder metode

- Metoda `charAt` vraća karakter na poziciji jednakoj argumentu metode. Pokušaj pristupanju karakteru van dozvoljenog opsega [0, dužina-1] dovodi do `StringIndexOutOfBoundsException` izuzetka.
- Metoda `setCharAt` na poziciji (prvi argument metode) postavlja karakter (drugi argument metode).
- Metoda `getChars` kopira karaktere iz `StringBuilder` stringa u niz karaktera, koji je argument metode. Pogledati dokumentaciju za više detalja.
- Metoda `reverse` obrće redosljed karaktera stringa, i nema argumenata.
- Metoda `append` nadovezuje argument metode (primitivni i referencijski tipovi) na predmetni string. Ova metoda se može lančano pozivati, na sljedeći način:

```
StringBuilder s1 = new StringBuilder("Java"), s2;
s2 = s1.append(" je").append(" moćan").append(" jezik.");
```

- Operatori nadovezivanja stringova u Javi + i += se implementiraju pomoću `StringBuilder` ili `StringBuffer` objekata.

StringBuilder metode. StringBuffer

- Metoda `insert` umeće vrijednosti različitih tipova (drugi argument) na bilo koju poziciju (prvi argument) u `StringBuilder` stringu. Metoda je preklopljena za sve primitivne tipove, nizove karaktera, objekte klasa `String` i `Object`, kao i u interfejsu `CharSequence`.
- Metode `delete` i `deleteCharAt` brišu karaktere iz stringa. Metoda `delete` ima dva argumenta koji definišu opseg karaktera za brisanje, dok `deleteCharAt` ima jedan argument – indeks karaktera koji se briše.
- Ukoliko su argumenti metoda `insert`, `delete` i `deleteCharAt` van dozvoljenog opsega, baca se `StringIndexOutOfBoundsException` izuzetak.
- `StringBuilder` klasa nije nitno bezbjedna (eng. *thread safe*), tj. ukoliko više programskih niti pokušava pristupiti istom `StringBuilder` objektu, one mogu simultano mijenjati objekat, što može dovesti do nepredviđenih rezultata.
- Rješenje za ovaj problem je `StringBuffer` klasa, koja je nitno bezbjedna, i koja ima istu funkcionalnost kao `StringBuilder` klasa.

Character klasa

- Klasa `Character` je omotačka klasa za primitivni tip `char`.
- Većina metoda ove klase su statičke, radi jednostavne obrade pojedinačnih `char` vrijednosti.
- Metoda `isDefined` vraća podatak (`true` ili `false`) o tome da li je karakter argument definisan u Unicode skupu karaktera.
- Metode `isDigit`, `isLetter` i `isLetterOrDigit` vraćaju podatak (`true` ili `false`) o tome da li je karakter argument cifra, slovo ili jedno od to dvoje.
- Metode `isLowerCase` i `isUpperCase` vraćaju podatak (`true` ili `false`) o tome da li je karakter argument malo ili veliko slovo.
- Metode `toLowerCase` i `toUpperCase` pretvaraju veliko slovo u malo i obrnuto, ili vraćaju originalni karakter ukoliko se date operacije ne mogu izvršiti.
- Metode `digit` i `forDigit` pretvaraju karakter u cifre i cifre u karaktere, respektivno, u različitim brojnim sistemima. Osnova brojnog sistema (eng. *radix*) je drugi argument metoda, dok je predmetni karakter prvi argument.

split metoda klase String

- Metoda `split` klase `String` „razbija“ predmetni string na podstringove razdvojene delimiterima (spejs, tab, zarez, Enter...), i vraća niz `String` objekata.

```
String recenica = "Java je moćan jezik.";
String rijeci[] = recenica.split(" ");
for(String s : rijeci)
     System.out.println(s);
```

```
Java
je
moćan
jezik.
```

Ispis

- Argument metode `split` može biti regularan izraz, čime se može izvršiti složenije razbijanje na podstringove.

join metoda klase String

- Metoda `join` klase `String` radi suprotnu radnju od metode `split`, tj. kreira string sastavljajući stringove argumente, razdvojene delimiterom. Delimiter je prvi argument metode.

```
String mocnaJava = String.join("-", "Java", "je", "moćan", "jezik");
```

```
Java-je-moćan-jezik
```
← Rezultat

Regularni izrazi

- Regularan izraz ili regex (eng. *regular expression*) je sekvenca karaktera koja predstavlja šablon za pretragu. Obično se koristi u *string matching* operacijama, tj. operacijama tipa "pronađi i zamijeni".
- Koncept regularnih izraza je uveo američki matematičar Stiven Klin 1950.
- Regularne izraze koriste mnogi tekst editori i programi za pretragu i manipulaciju tekstom.
- Mnogi programski jezici, uključujući Javu, PHP, Perl, Python, Ruby, JavaScript, podržavaju regularne izraze za manipulisanje stringova.
- Regularni izrazi pružaju vrlo elegantno rješenje za validaciju unosa korisnika i provjeru da li podaci zadovoljavaju određeni format. Na primjer, regularni izrazi nam mogu poslužiti za provjeru formata email adrese, broja telefona, korisničke lozinke koja mora da sadrži određene karaktere, imena i prezimena osobe itd.
- Vrlo važna primjena regularnih izraza je u kreiranju kompajlera. Često se provjera sintakse programa vrši pomoću vrlo složenog regularnog izraza, pa ukoliko program ne odgovara tom izrazu javlja se sintaksna greška.

matches metoda

- U Javi, regularan izraz je `String` objekat.
- Klasa `String` ima nekoliko metoda za izvršavanje operacija sa regularnim izrazima, od kojih je najjednostavnija `matches` metoda. Ova metoda ima za argument `String` objekat koji specificira regularan izraz i poziva se na `String` objektu za koji želimo da utvrdimo da li odgovara predmetnom regularnom izrazu. Metoda vraća `true` ili `false`.
- Na primjer, provjera da li se string `str` sastoji samo od cifara, samo od malih slova ili počinje riječju Java, bi bila:

```
str.matches("\\d*")       // nula ili više cifara
str.matches("[a-z]+")     // jedno ili više malih slova
str.matches("Java.+")     // string počinje riječju Java
```

- Regularni izraz se sastoji od karaktera literala i specijalnih simbola.
- Na sljedećem slajdu su prikazane klase karaktera koje se mogu koristiti u regularnim izrazima.

Klase karaktera u regularnim izrazima

Konstrukcija	Opis
`[abc]`	Karakter a, b ili c
`[^abc]`	Sve osim karaktera a, b ili c
`[a-z]`	Sva mala slova (opseg)
`[^a-z]`	Sve osim malih slova
`[a-zA-Z]` ili `[a-z[A-Z]]`	Sva slova, mala ili velika
`[a-z&&[bcd]]`	Presjek skupova malih slova i karaktera b, c ili d

Karakter	Opis
`.`	Bilo koji karakter
`\d`	Bilo koja cifra `[0-9]`
`\D`	Sve osim cifri `[^0-9]`
`\w`	Bilo koji word karakter `[a-zA-Z_0-9]`
`\W`	Sve osim word karaktera `[^\w]`
`\s`	Bjelina
`\S`	Sve osim bjelina `[^\s]`

Predefinisane klase karaktera

Kvantifikatori u regularnim izrazima

- Kvantifikatori u regularnim izrazima su specijalni izrazi koji određuju broj pojavljivanja podstringa (jednog ili više uzastopnih karaktera) u stringu.
- Kvantifikator se odnosi samo na podstring koji prethodi kvantifikatoru.
- Ako želimo provjeru da li se složeniji izraz ponavlja u stringu, možemo koristiti male zagrade da grupišemo taj izraz.
- Kvantifikatori su pohlepni (eng. *greedy*), tj. traže maksimalan broj pojavljivanja podstringa (nastavljaju pretragu sve dok ima poklapanja). Ako se iza njih stavi karakter `?`, kvantifikatori postaju lijeni, tj. traže minimalan broj pojavljivanja.

Kvantifikator	Opis
`*`	0 ili više uzastopnih pojavljivanja podstringa
`+`	1 ili više uzastopnih pojavljivanja podstringa
`?`	0 ili 1 pojavljivanja podstringa
`{n}`	Tačno n uzastopnih pojavljivanja podstringa
`{n,}`	Barem n uzastopnih pojavljivanja podstringa
`{n,m}`	Između n i m uzastopnih pojavljivanja podstringa (uključeni n i m)

Primjeri korišćenja matches metode

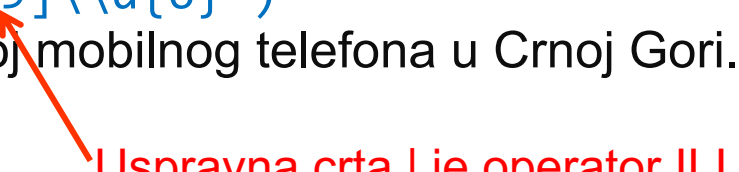
`str.matches("Java.*")`
String koji počinje riječju `Java` i nakon toga može da sadrži proizvoljan broj (uključujući 0) proizvoljnih karaktera.

`str.matches("(Ta)+")`
Stringovi `"Ta"`, `"TaTa"`, `"TaTaTa"` ...

`str.matches("[A-Z][a-z]+")`
String koji predstavlja ime osobe, tj. string koji počinje velikim slovom, a nakon toga jedno ili više malih slova.

`str.matches("[A-Z][a-z]+(-[A-Z][a-z]+)?")`
String koji predstavlja prezime osobe, tj. string koji može da sadrži jednu ili dvije riječi. Ako ima dvije riječi, razdvojene su karakterom `-`. Riječi počinju velikim slovom, a ostalo su mala slova. Na primjer, `"Popović"` ili `"Popović-Bugarin"`.

`str.matches("06[7|8|9]\\d{6}")`
String koji predstavlja broj mobilnog telefona u Crnoj Gori.



Metode replaceAll i replaceFirst

- Metoda `replaceAll` mijenja sve pojave jednog podstringa u predmetnom stringu drugim podstringom.
- Metoda `replaceFirst` mijenja prvu pojavu jednog podstringa u predmetnom stringu drugim podstringom.
- Traženi string može biti regularni izraz.
- Metode `replaceAll` i `replaceFirst` su metode klase `String`, pa ove metode ne mogu da promijene predmetni string, već vraćaju string sa izvršenim zamjenama.
- primjeri:

```
str.replaceAll("abc","Q")
str.replaceAll("\\d","#")
str.replaceAll("[\\.,;]","\n")
```

zamjena podstringa `abc` karakterom `Q`
zamjena svih cifara karakterom #
zamjena svih tački, zareza i tačka-zareza karakterom za prelaz u novi red

Karakter tačka se predstavlja sa `\\.`

Metoda split

- U ranije pomenutoj metodi `split`, delimiter može biti regularan izraz.
- Na primjer, ako želimo da izdvojimo sve riječi u nekom tekstu, možemo koristiti `split` metodu sa delimiterom koji će vršiti razbijanje teksta na bjelinama i znacima interpunkcije, na sljedeći način:

```
tekst.split("\\s+|,\\s*|\\.\\s*|;\\s*|\\?\\s*|!\\s*")
```

1 ili više bjelina

Zarez praćen sa 0 ili više bjelina

Tačka praćena sa 0 ili više bjelina

Tačka-zarez praćen sa 0 ili više bjelina

Znak pitanja praćen sa 0 ili više bjelina

Znak uzvika praćen sa 0 ili više bjelina

- Java klase `Pattern` i `Matcher` se također mogu koristiti za rad sa regularnim izrazima.

Ključna riječ this

- Svi objekti jedne klase dijele istu kopiju metoda klase.
- Sa druge strane, svaki objekat ima svoju kopiju podataka, isključujući statičke podatke koji su jedinstveni za čitavu klasu.
- Pomoću ključne riječi `this`, objekat može referencirati samog sebe. Na ovaj način, metoda zna koji tačno objekat je pozvao metodu, tj. sa čijim podacima da manipuliše. Naziva se još i `this` referenca.
- Kada se pozove nestatička metoda za određeni objekat, tijelo metode implicitno koristi `this` referencu za pristup promjenljivim objekta i drugim metodama klase.
- `this` se može koristiti za pristup podacima objekta koji su zasjenjeni lokalnim promjenljivim ili parametrima metode.

this i zasjenjivanje podataka klase

```
public class TackaTest {
  public static void main(String[] args) {
    Tacka t = new Tacka(2.21,4.74);
    System.out.println(t.toString());
  }
}

class Tacka {
  double x, y;
  public Tacka(double x, double y){
    this.x = x;
    this.y = y;
  }
  public String toString(){
    return String.format("(%.2f,%.2f)", x, this.y);
  }
}
```

Ispis

```
(2.21,4.74)
```

String.format metoda kreira string

this i zasjenjivanje podataka klase

- Klase `Tacka` i `TackaTest` su smještene u istom fajlu.
- Više klasa se može smjestiti u isti fajl samo pod uslovom da je samo jedna klasa javna, a sve ostale nisu javne. U suprotnom dolazi do greške kompajliranja.
- Javin kompajler će, u ovom slučaju, kreirati poseban `class` fajl, u istom folderu, za svaku kompajliranu klasu.
- Klasa `Tacka` nije javna. Podrazumijevano, klase nisu javne, već javnim postaju kad se deklarišu ključnom riječju `public`.
- `this` referenca omogućava pristup zasjenjenim podacima klase.
- `this` referencu možemo koristiti i kad podatak nije zasjenjen.

this i preklapanje konstruktora

```
public class Tacka {

   private double x, y;

   public Tacka(){
      this(0, 0);
   }

   public Tacka(double x){
      this(x, 0);
   }

   public Tacka(Tacka t){
      this(t.getX(), t.getY());
   }

   public Tacka(double x, double y){
      postaviTacku(x, y);
   }

   public void postaviTacku(double x, double y){
      setX(x);
      setY(y);
   }

   public void postaviTacku(Tacka t){
      this.postaviTacku(t.getX(), t.getY());
   }

   public double getX() {
      return x;
   }

   public void setX(double x) {
      this.x = x;
   }
   public double getY() {
      return y;
   }
   public void setY(double y) {
      this.y = y;
   }
   public double rastojanje(Tacka t){
      return Math.sqrt((getX()-t.getX())*(getX()-t.getX())
                     + (getY()-t.getY())*(getY()-t.getY()));
   }
   public String toString(){
      return String.format("(%.2f,%.2f)", getX(), getY());
   }
}
```

```
public class TackaTest2 {
  public static void main(String[] args) {
    Tacka t1 = new Tacka();
    Tacka t2 = new Tacka(3.4);
    Tacka t3 = new Tacka(2.3,-1.17);
    Tacka t4 = new Tacka(t3);
    System.out.printf("Prva tačka %s\n",t1.toString());
    System.out.printf("Druga tačka %s\n",t2.toString());
    System.out.printf("Treća tačka %s\n",t3.toString());
    System.out.printf("Četvrta tačka %s\n",t4.toString());
    t1.postaviTacku(4.5, -2.1);
    t2.postaviTacku(t3);
    t3.setX(11.59);
    t4.setY(-5.6);
    System.out.printf("Prva tačka nakon izmjene %s\n",t1);
    System.out.printf("Druga tačka nakon izmjene %s\n",t2);
    System.out.printf("Treća tačka nakon izmjene %s\n",t3);
    System.out.printf("Četvrta tačka nakon izmjene %s\n",t4);
    System.out.printf("Rastojanje prve i druge tačke %.2f\n",t1.rastojanje(t2));
  }
}
```

```
Prva tačka (0.00,0.00)
Druga tačka (3.40,0.00)
Treća tačka (2.30,-1.17)
Četvrta tačka (2.30,-1.17)
Prva tačka nakon izmjene (4.50,-2.10)
Druga tačka nakon izmjene (2.30,-1.17)
Treća tačka nakon izmjene (11.59,-1.17)
Četvrta tačka nakon izmjene (2.30,-5.60)
Rastojanje prve i druge tačke 2.39
```

← Ispis

this i preklapanje konstruktora

- Prvi konstruktor nema argumenata. Ako definišemo bar jedan konstruktor, Javin kompajler neće kreirati podrazumijevani konstruktor.
- Ukoliko mi ne definišemo konstruktor bez argumenata, klijenti klase `Tacka` neće moći kreirati nove `Tacka` objekte naredbom `new Tacka()`.
- Konstruktor bez argumenata obezbeđuje da klijenti klase `Tacka` mogu kreirati objekat `Tacka` sa podrazumijevanim vrijednostima.
- `this` referenca se, unutar konstruktora, može iskoristiti za pozivanje drugih konstruktora. Na ovaj način, korišćenjem koda za inicijalizaciju iz nekog drugog konstruktora iste klase eliminišemo ponavljanje sličnih naredbi u konstruktorima.
- Poziv konstruktora pomoću `this` reference se mora izvršiti u prvoj naredbi konstruktora, inače dolazi do greške kompajliranja.
- Pokušaj pozivanja konstruktora iz neke metode klase pomoću `this` reference uzrokuje grešku.

set i get metode

- Unutar klase `Tacka`, podacima te klase ne pristupamo direktno već koristeći `set` i `get` metode. Iako na prvi pogled ovo može izgledati kao nepotrebno pozivanje metoda, što je sporije od direktnog pristupa podacima, to nije slučaj.
- Jedan od principa modularnosti u OO dizajnu softvera je **princip uniformnog pristupa** (eng. *uniform access principle*) po kome svi servisi koje pruža neki modul trebaju biti dostupni kroz jedinstven zapis, bez obzira na to kakva je unutrašnja organizacija klase.
- Pretpostavimo da želimo da promijenimo reprezentaciju klase, pa da umjesto x i y koordinata tačaka u pravougaonom koordinatnom sistemu, za podatke imamo ugao phi i radijus ro u polarnom koordinatnom sistemu. U tom slučaju bi dobijanje x i y koordinata tačke zahtijevalo određeni matematički račun, koji bi trebalo sprovesti u svakoj naredbi metoda klase gdje pristupamo tim koordinatama. Svakako bi ovu izmjenu bilo poželjno lokalizovati na jedno mesto, za šta nam idealno mogu poslužiti `set` i `get` metode. Samo te dvije metode znaju unutrašnju organizaciju klase, dok sve ostale metode pristupaju željenim podacima koristeći te dvije metode.

set i get metode

- Svi konstruktori u klasi `Tacka` se svode na poziv metode `postaviTacku`, koja opet poziva `setX` i `setY` metode.
- Ovo jeste manje efikasno, jer zahtijeva poziv metode umjesto direktnog pristupa podacima koji je brži, ali je svakako i manje podložno greškama jer se promjene vrše samo na jednom mestu.
- Kompajler može optimizovati programe tako što jednostavne metode realizuje **inline**, tj. svaki njihov poziv u programu zamijeni čitavim kodom metode.
- Dodatna funkcionalnost koja se može inkorporirati unutar `set` i `get` metoda je provjera ispravnosti proslijeđenih argumenata. Naime, kod nekih klasa, postoje nedozvoljene vrijednosti podataka. Na primjer, iznos koeficijenta plate radnika ne može biti negativan, prosjek studenta takođe ne može biti negativan. U tom slučaju, `set` i `get` metode treba da sadrže dio koda koji će baciti izuzetak tipa `IllegalArgumentException`.

Još par činjenica o this referenci

- `this` ne može biti korišćena u statičkim metodama. Razmislite zašto.
- `this` je `final` promjenljiva i samim tim joj se ne može dodijeliti vrijednost. Na primjer, naredba tipa `this = new Tacka();` bi dovela do greške kompajliranja.
- `this` može biti vraćena vrijednost metode, tj. dozvoljena je naredba `return this`.
- `this` može biti argument metode.
- Kad se upotrijebi `this` tamo gdje se očekuje string, implicitno se poziva `toString` metoda tekućeg objekta.

Kompozicija

- Generalno govoreći, objektna kompozicija predstavlja način kombinovanja jednostavnijih objekata u cilju dobijanja složenijih.
- Posmatrajmo objekat Kompjuter. Ovaj objekat će sadržati objekte tipa Memorija, CPU, Matična ploča itd. tj. taj objekat ima druge objekte.
- Kompozicija odražava **vezu tipa ima** (eng. *has-a relationship*).
- Klasa može sadržati reference na objekte druge klase i ovo se naziva **kompozicijom**.
- Kreirajmo klasu `Krug` koja će sadržati referencu na objekat klase `Tacka` i jedan realan broj koji predstavlja poluprečnik kruga. Svaki krug je jednoznačno određen položajem centra kruga (`Tacka` objekat) i poluprečnikom. Kompozicija odražava vezu tipa ima između kruga i tačke, jer krug ima centar, koji je `Tacka`.

Kompozicija

```
public class Krug {
  Tacka centar;
  double radijus;
  public Krug(Tacka t, double r){
    centar = t;
    radijus = r;
  }
  public String toString(){
    return String.format("Centar %s, radijus %.2f", centar, radijus);
  }
}

public class KrugTest {
  public static void main(String[] args) {
    Tacka t = new Tacka(2.3,3.9);
    Krug k = new Krug(t,1.6);

    System.out.println(k);
  }
}
```

Kompozicija

```
Centar (2.30,3.90), radijus 1.60
```

Ispis

Ugnježdene klase. Statičke ugnj. klase

- Klasa u Javi za član može imati drugu klasu - **ugnježdenu klasu** (eng. *nested class*). Koncept ugnježdenih klasa omogućava logičko grupisanje klasa koje se koriste na jednom mjestu čime se promoviše enkapsulacija. Takođe, doprinose čitljivosti i održavanju programskog koda jer se ugnježdavanjem klase smještaju bliže mjestu gdje se koriste.
- Opseg ugnježdene klase je ograničen opsegom spoljašnje klase. Ugnježdena klasa ima pristup članovima spoljašnje klase, uključujući i privatne članove. Suprotno ne važi, tj. spoljašnja klasa nema pristup članovima ugnježdene klase.
- Ugnježdena klasa može biti deklarisana kao privatna, javna, zaštićena ili može imati podrazumijevano pravo pristupa.
- Ugnježdene klase se dijele na **statičke** i **nestatičke**.
- Kod statičkih ugnježdenih klasa, instanca se može kreirati bez prethodnog kreiranja instance spoljašnjih klasa. Iz statičke ugnježdene klase, možemo pristupiti samo statičkim članovima spoljašnje klase.
- primjer sa statičkim ugnježdenim klasama je dat na sljedećem slajdu.

Primjer sa statičkim ugnježdenim klasama

```
public class Spoljasnja {

  static int xStatic = 1;
  private static int xPrivateStatic = 2;
  int xNonStatic = 3;

  static class StaticUgnjezdena {      // Statička ugnježdena klasa
    void stampaj() {
      // Možemo pristupiti statičkom članu spoljašnje klase
      System.out.println("xStatic = " + xStatic);
      // Možemo pristupiti privatnom statičkom članu spoljašnje klase
      System.out.println("xPrivateStatic = " + xPrivateStatic);
      // Ne možemo pristupiti nestatičkom članu spoljašnje klase
      // Sljedeća naredba bi uzrokovala grešku kompajliranja
      // System.out.println("xNonStatic = " + xNonStatic);
    }
  }

  public static void main(String[] args) {
    // Kreiranje instance statičke ugnježdene klase
    Spoljasnja.StaticUgnjezdena su = new Spoljasnja.StaticUgnjezdena();
    su.stampaj();
  }
}
```

```
xStatic = 1
xPrivateStatic = 2
```

← Ispis

Nestatičke ugnježđene (unutrašnje) klase

- Nestatičke ugnježđene klase se još nazivaju **unutrašnje klase**.
- Instanca unutrašnje klase se ne može kreirati bez prethodno kreirane instance spoljašnje klase.
- Iz unutrašnje klase, možemo pristupiti i statičkim i nestatičkim članovima spoljašnje klase.
- Unutrašnja klasa ne može sadržati statičke metode.
- Na sljedećem slajdu je primjer sa unutrašnjom klasom.

Primjer sa unutrašnjim klasama

```
public class Spoljasnja {

  static int xStatic = 1;
  private static int xPrivateStatic = 2;
  int xNonStatic = 3;

  static class Unutrasnja {
    void stampaj() {
      // Možemo pristupiti statičkom članu spoljašnje klase
      System.out.println("xStatic = " + xStatic);
      // Možemo pristupiti privatnom statičkom članu spoljašnje klase
      System.out.println("xPrivateStatic = " + xPrivateStatic);
      // Možemo pristupiti nestatičkom članu spoljašnje klase
      System.out.println("xNonStatic = " + xNonStatic);
    }
  }

  public static void main(String[] args) {
    // Kreiranje instance unutrašnje klase
    Spoljasnja spoljasnja = new Spoljasnja();
    Spoljasnja.Unutrasnja un = spoljasnja.new Unutrasnja();
    un.stampaj();
  }
}
```

```
xStatic = 1
xPrivateStatic = 2
xNonStatic = 3
```

← Ispis

Nabrojani tipovi podataka

- U svom najjednostavnijem obliku, enumeracija predstavlja listu imenovanih konstanti, i time podsjeća na enumeracije u C i C++. Međutim, ova sličnost je samo površinska.
- U Javi, enumeracija definiše svoj tip klase, čime je koncept enumeracije znatno proširen.
- Enumeracija se kreira pomoću ključne riječi **`enum`**. Na primjer, enumeracija koja bi obuhvatala dane u sedmici bi izgledala:

```
enum Dani {
  PONEDELJAK, UTORAK, SRIJEDA, CETVRTAK, PETAK, SUBOTA,
NEDJELJA
}
```

- Identifikatori `PONEDELJAK`, `UTORAK`, ..., se nazivaju konstante enumeracije, pri čemu svaka predstavlja jedinstvenu vrijednost.
- Svaka konstanta je implicitno deklarisana kao `public static final` član klase `Dani`, i njihov tip odgovara tipu enumeracije u kojoj su deklarisani, u ovom slučaji tipu `Dani`.

Nabrojani tipovi podataka

- Konvencionalno, konstante enumeracije sadrže samo velika slova u svom imenu, čime se naglašava da nisu u pitanju promjenljive, već konstante.
- Nakon deklarisanja enumeracije, možemo kreirati promjenljive tog tipa. Iako su klasnog tipa, instance enumeracije se ne kreiraju pomoću ključne riječi new, već isto kao primitivni tipovi. Na primjer, sa

  ```
  Dani dan;
  ```

  se deklariše promjenljiva `dan` tipa `Dani`.
- Jedine vrijednosti koje može imati promjenljiva dan su one definisane enumeracijom (`PONEDELJAK`, ..., `NEDJELJA`), i pri dodjeli vrijednosti moramo navesti ime enumeracije. Na primjer:

  ```
  Dani dan = Dani.SRIJEDA;
  ```
- Dvije konstante nabrajanja se mogu porediti:

  ```
  if(dan == Dani.SRIJEDA) System.out.println("Ipak je srijeda");
  ```
- Konstanta nabrajanja se može naći u upravljačkom izrazu `switch` naredbe.

Enumeracija kao klasa

- Svi `enum` tipovi su referencijski tipovi.
- Deklaracija enumeracije, pored konstanti, može uključiti i druge klasne komponente, kao što su metode, konstruktori, podaci.
- Enumeracije predstavljaju specijalan tip klase kod kojih su konstante deklarisane kao `final` i `static`.
- U nastavku dajemo primjer enumeracije `Opel` koja sadrži konstante `ASTRA`, `INSIGNIA`, `CORSA` i `ZAFIRA`, dva cjelobrojna podatka `godProiz` i `kubikaza`, konstruktor i `get` metode za podatke.

Primjer enumeracije kao klase

```
public enum Opel {
  ASTRA(2011,1590),
  INSIGNIA(2009,2000),
  CORSA(2008,1400),
  ZAFIRA(2012,2100);
  private final int godProiz;
  private final int kubikaza;

  Opel(int gP, int kub){
    godProiz = gP;
    kubikaza = kub;
  }
  public int getGodProiz() {
    return godProiz;
  }
  public int getKubikaza() {
    return kubikaza;
  }
}
```

Nakon svake konstante enumeracije, unutar zagrada navodimo vrijednosti kojima inicijalizujemo konstante. Konstante enumeracije se navode odmah na početku deklaracije enumeracije, tj. ispred podataka, konstruktora i metoda. U suprotnom, dobija se sintaksna greška.

Konstruktor. Inicijalizacija se, kao kod ostalih klasa, vrši pozivanjem konstruktora.

Primjer enumeracije kao klase

Metoda values vraća niz enum konstanti u redosljedu u kom su deklarisane.

```
public class OpelTest {

  public static void main(String[] args) {

    System.out.printf("%-10s%8s%10s\n", "Marka","Godina", "Kubikaža");
    for(Opel opelAuto: Opel.values())
      System.out.printf("%-10s%8d%10d\n",
        opelAuto,opelAuto.getGodProiz(),opelAuto.getKubikaza());
  }

}
```

```
Marka        Godina  Kubikaža
ASTRA          2011      1590
INSIGNIA       2009      2000
CORSA          2008      1400
ZAFIRA         2012      2100
```

← Ispis

Uklanjanje smeća

- Svaki objekat koristi resurse sistema, na primjer memoriju.
- Onog trenutka kada neki objekat u memoriji nema nijednu referencu na njega, tj. kad postane nedostupan programeru pa se više ne može koristiti, taj objekat postaje smeće. Ove situacije dovode do curenja memorije (eng. *memory leaks*).
- **Uklanjanje smeća** (eng. *garbage collection*) predstavlja proces uklanjanja nedostupnih objekata iz memorije.
- U Javi se uklanjanje smeća obavlja automatski, za razliku od, recimo, jezika C i C++, gdje to programer mora da radi ručno, što je izvor mnogih suptilnih fatalnih grešaka.
- U Javi se uklanjanje smeća obavlja pozivanjem posebnog programa koji se naziva garbage collector.
- Iako se automatskim uklanjenjem smeća značajno smanjuje curenje memorije, ono nije u potpunosti eliminisano.

Uklanjanje smeća

- Ostala nepotrebna trošenja resursa se mogu desiti. Na primjer, pri radu sa fajlovima, fajlove je potrebno zatvoriti čim smo završili željenu operaciju sa njima. Dok naša aplikacija čita ili modifikuje fajl, ostale aplikacije ne mogu otvoriti taj fajl.
- Klase koje koriste resurse sistema treba da obezbijede metode kojima programeri mogu da oslobode te resurse čim više nisu potrebni u programu. U tu svrhu, mnoge klase iz Java API-a imaju metode `close` ili `dispose`. Vrlo često korišćena klasa `Scanner` ima metodu `close`.
- Tokom izvršavanja programa, nije precizno definisano kada će se, i da li će se uopšte izvršiti uklanjanje smeća. Kad se to desi, JVM ne garantuje da će ukloniti sve nedostupne objekte iz memorije. Štaviše, može se desiti da se ne ukloni nijedan nedostupni objekat.

Uklanjanje smeća – gc i finalize

- Pomoću metode `System.gc()` možemo da „zamolimo“ JVM da pokrene uklanjanje smeća. JVM će se „potruditi“ da ukloni svo smeće, ali ne postoji garancija za to.
- Metodu `Object.finalize()` poziva garbage collector nad objektom kad se utvrdi da je to zaista smeće. U toj metodi se mogu osloboditi resursi.

```
public class Test {
  public static void main(String[] args){
    Test t1 = new Test();
    t1 = null;
    System.gc();
  }
  @Override
  protected void finalize(){
    System.out.println("Pozvan garbage collector");
    System.out.println("Uklonjen objekat: " + this);
  }
}
```

```
Pozvan garbage collector
Uklonjen objekat: Test@10fad0da
```

← Ispis