Objekti i klase

Objektno-orjentisano programiranje u Javi

JOŠ JEDNOM

- Standardno, **proceduralno** programiranje (npr. C):

  Program započinje izvršavanjem funkcije **main** koja izvršava postavljeni zadatak pozivanjem drugih funkcija. Program završava kad se izvrše sve instrukcije funkcije main. Osnovni građevni blok programa je, dakle, **funkcija**. Postavljeni zadatak se rešava tako da što se razbije na niz manjih zadataka od kojih se svaka može implementirati u jednoj funkciji, tako da je program niz funkcijskih poziva.

- U **objektno-orijentisanom** programiranju

  osnovnu ulogu imaju **objekti koji sadrže i podatke i funkcije (metode)**. Program se konstruiše kao skup objekata koji međusobno komuniciraju. Podaci koje objekat sadrži predstavljaju njegovo **stanje**, dok pomoću metoda on to stanje da može menja i komunicira sa drugim objektima.

Objektno-orjentisano programiranje u Javi

- Java je potpuno objektno-orjentisan jezik i zato je
  sav izvorni kod u Javi podeljen u klase
  (koje se nalaze u istoj ili različitim izvornim datotekama).

```
// Podsecanje
public class HelloWorld{
   public static void main(String[] args){
          System.out.println("Hello, World");
      }
   } // kraj definicije klase HelloWorld
```

- Dakle, u osnovi svega je objekat koji:
  - ima podatke
  - ima metode
  - ima jedinstvenu adresu u memoriji
  - predstavlja instancu neke konkretne klase

Klasa

Definisanje klase

```
<modifikator> class <className> {

        <modifikator> <tip> <imepromenljive1>;
        <modifikator> <tip> <imepromenljive2>;
        ...

    <modifikator> <povratni tip> <imemetoda1> (<tip> <arg1>,...) {
            ... //implementacija metoda 1
    }
    <modifikator> <povratni tip> <imemetoda2> (<tip> <arg1>,...) {
            ... //implementacija metoda 2
    }
    ...

}   // kraj definicije klase
```

podaci članovi

funkcije članice - metodi

Objekti

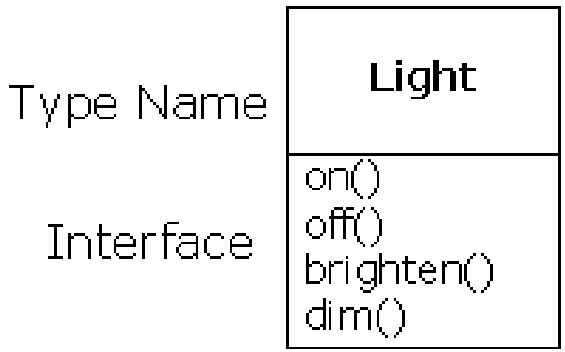
- Klasom se opisuju objekti sa istim
  - karakteristikama (podaci članovi)
  - ponašanjem (funkcionalnostima – metode)
- Podaci članovi (atributi)
  - svaki objekat ima sopstvene vrednosti podataka članova
  - trenutne vrednosti podataka objekta čine trenutno **stanje objekta**
- Funkcije članice (metodi)
  - njima je su definisana **ponašanja objekta**
  - poziv metoda jednog objekta – **slanje poruke**
  - obrada zahteva tj. **odgovaranje na poruku**

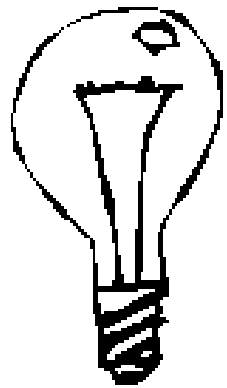
PRIMER.

`lt` – objekat klase `Light`

`lt.on();` - slanje poruke, tj.

poziv metoda on objekta `lt`





Metodi, overloading

- skup metoda - interface
- Povratni tip i ime metoda su obavezni, kao i ()

```
<modifier> <returnType> <methodName> (<Parameters>) {
// body of the method. The code statements go here.
}
```

- Metodi se definišu **samo kao deo klase**. Pozivi pogrešnih metoda za neki objekat se registruju pri kompajliranju.

```
int x = a.f(); // a je objekat odgovarajuce klase
```

- Poseban tretman imaju static metodi, kao i static podaci (o tome malo kasnije).

```
float naturalLogBase() { return 2.718f; }
void nothing() { return; }
```

- Vraćanje sa bilo koje tačke, ali sa odgovarajućim tipom.

- **OVERLOADING**
  U jednoj klasi (ili pri nasleđivanju) se može definisati više metoda sa istim imenom ali se oni **moraju razlikovati po broju ili bar po tipu argumenata**. Dozvoljeno je i da vraćaju različite tipove (ali pod uslovom da se razlikuju po argumentima).

Kreiranje objekta

- Sve čime se manipuliše je objekat neke klase♀ (čak i sama aplikacija/applet, naravno ne računajući primitivne tipove), odnosno **referenca na objekat**.

  Objekat – TV

  Referenca – daljinski

- Sa sobom nosite daljinski, a ne televizor. Možete kupiti daljinski i bez televizora.

```
Light lt; // kreirana je samo referenca
Light lt = new Light(); // kreiran je i objekat
```

```
int i = 2;
```

i
Primitive variable name

```
Light lt = new Light();
```

lt
Reference variable name

Zadatak 1.

- Definisati klasu `Robot` (bez modifikatora) na sledeći način

Robot – bez modifikatora	
`rbr`	– tipa integer, bez modifikatora
`setrbr(int br)`	– metod koji postavlja vrednost promenljive rbr
`sayHello()`	– metod koji ispisuje poruku *Hello, I am robo no. ___*

Definisati aplikaciju `UseRobot` (preciznije `public` klasu `UseRobot`) koja u svom main metodu:

- kreira objekat klase `Robot`,
- postavlja vrednost `rbr` na 1,
- šalje poruku kreiranom objektu da se javi (`sayHello`).

- **Primer 1a.**

Prepraviti aplikaciju `UseRobot` tako da kreira u petlji 3 objekta tipa Robot, pri čemu svakom dodeljuje redni broj po redosledu kreiranja i, naravno svaki od njim mora da se “javi“.

Konstruktori

Primer 1.

```
class Tacka{
      private double x;
      private double y;
      public Tacka() { x=0.0; y=0.0; }
      public Tacka(double a, double b) { x=a; y=b; }
      public double getX() { return x; }
      public double getY() { return y; }
}
```

konstruktori

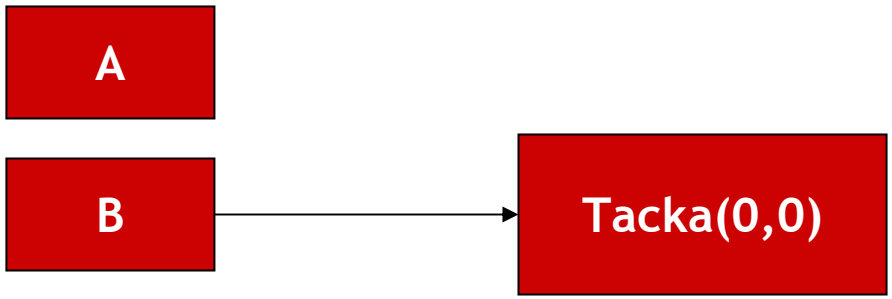
- Konstruktor je metod ima isto ime kao i klasa, pozivase isključivo pri instanciranju objekata (dakle, operatorom **new**),
- **nema povratne vrednosti**
- Klasa može imati više konstruktora (overloading na delu)
- Konstruktor bez parametara je **default konstruktor** i on postoji kada klasa nema posebno implementiran (naveden) ni jedan konstuktor videti Primer 1., u suprotnom neće postojati.

Konstruktori

```
// Test.java
public class Test {
 public static void main(String[ ] args){
  Tacka a = new Tacka();
  Tacka b = new Tacka(1,1);
  System.out.println("A(" + a.getX() + "," + a.getY() + ")\n");
  System.out.println("B(" + b.getX() + "," + b.getY() + ")\n");
   }
}
```

Object variables - Reference

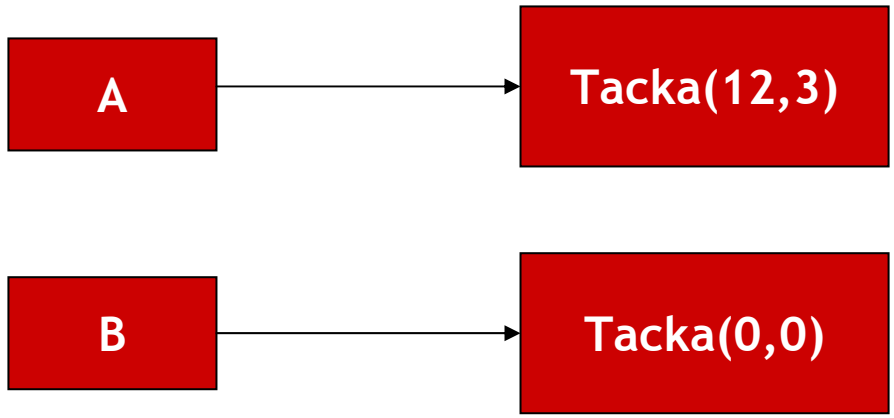
```
int x;       // x je varijabla primitivnog tipa
Tacka A;     // A je objektna varijabla (neinicijalizovana)
Tacka B=new Tacka(0,0); // B je objektna varijabla (inicijalizovana)
```



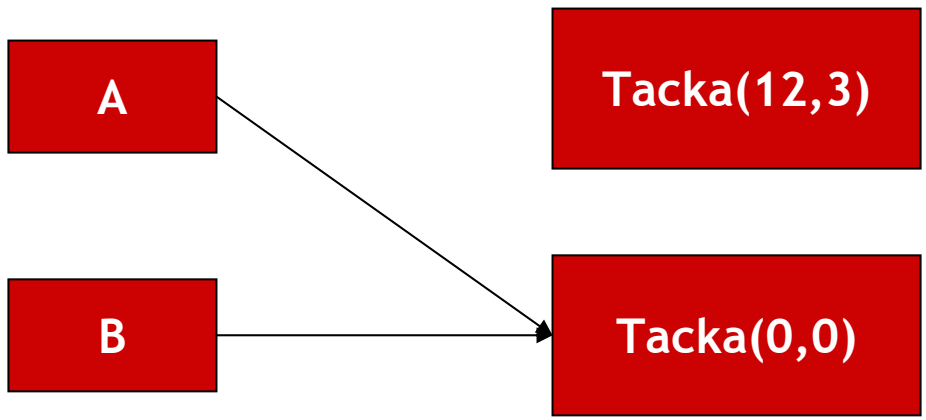

```
double z = A.getX();  // greska, objekat ne postoji
double w = B.getX(); // OK
```

Object variables - Reference

```
A = new Tacka(12,3);
```



```
A = B;
```



Oblast važenja

- Oblast važenja (**scope**) podrazumeva vidljivost i životni vek ‘imena’ i Java je definiše slično C-u i C++-u

```
{ int x = 12;
 /* samo je x dostupno*/
       { int q = 96;
              /* x i q su dostupni */
        }
 /* samo x je dostupno a q ‘ne postoji’ */
 }
```

domaćice

- Za razliku od C-a u Javi **nije dozvoljeno sledeće**

```
{ int x = 12;
      {
        int x = 96; /* illegal */
      }
}
```

Oblast važenja

- **Životni vek objekata nije isti životnom veku primitivnih tipova.** Nakon kreiranja objekat postoji i posle }, jer je iz oblasti važenja ‘izašla‘ samo referenca

```
{
        Tacka s = new Tacka();
}
```

- Kada objekat više nije potreban, tj. nije referenciran ni jednom referencom onda biva automatski oslobođen **garbage collector**-om

Još malo o objektima i referencama

Pošto se svim 'podacima' (osim onih koji su primitivnog tipa) pristupa preko reference očigledno je da:

- operator dodele vrednosti (=) i poređenja (==) 'rade' samo sa referencama na objekte (naravno, kada se primenjuju na objekte)

za kopiranje stanja objekta korisiti se metoda clone(), a za poređenje metoda equals()

- promenjliva koja pripada složenom tipu ("reference type") ne može nikada biti dodeljena promenljivoj primitivnog tipa i obrnuto

```
MyClass copy = (MyClass) obj.clone();
```

```
String a = "Christian Cantrell";
String b = "Christian Cantrell";
if (a.equals(b)) { // This code will be executed. }
```

Originalno equals metod radi isto što i == ako nije posebno definisan (overriden) za klasu, za većinu klasa iz javinih paketa jeste.

Nizovi

- U Javi, niz je sam po sebi objekat (or reference type) bez obzira na to kakvi su mu elementi, a elementi mogu pripadati nekom primitivnom tipu podatka ili biti objekti neke klase.
- Tri koraka u kreiranju niza
  - Deklaracija promenljive niza (array variable)

    ```
    int primes[]; int[] primes;
    Sijalica luster[]; Sijalica[] luster;
    ```

  - Kreiranje objekta niza i njegova dodela promenljivoj niza
    - Korišćenjem operatora new
      - `primes = new int[5];` `luster = new Sijalica[3];`
    - Direktnim inicijalizovanjem niza
      - `primes = {2, 3, 5, 7, 11};`
  - Smeštanje podataka u niz `primes[1]=2;`
- Višedimenzioni nizovi se deklarišu na sledeći način
  - `int brojevi = new int[10][10];`

konstruktor niza

Nije dozvoljeno i jedno i drugo

Nizovi

- Kao i u C-u prvi element ima indeks 0.
- **Nizovi su uvek inicijalizovani**
  - na nulu, ako su numerički (čak ako se radi o lokalnom nizu)
  - null, ako su elementi reference.
- Elementima se **ne može pristupiti van definisanog opsega** (pri izvršavanju se proverava indeks i diže `IndexOfBoundsException` u slučaju da je van opsega)
- Svaki niz ima public polje, length, koji sadrži broj elemenata u nizu
- Prosleđivanje niza metodu (poznat primer)

```
public static void main(String[] args){
      System.out.println(args[0]);
}
```

- Niz kao povratni tip metoda

```
public int[] makeArray(int howMany)
  {  int[] ar = new int[howMany];...
     return ar;
  }
```

Višedimenzionalni nizovi

- Deklaracija/inicijalizacija

```
int matricaint = new int[10][10];
int a[][] = { {1, 2, 3 },  {4, 5, 6 } };
int a[][] = new int[2][3];

int a[][] = new int[2][];
for(int i = 0; i < a.length; i++) { a[i] = new int[3];}

Automobil[][] a = {
   { new Automobil(), new Automobil() },
   { new Automobil(), new Automobil() }
};
```

- Niz nizova različite dužine

```
float[][] samples;
samples = new float[3][];
samples[0]=new float[2];
samples[1]=new float[4];
samples[2]=new float[10];
```

String klasa – immutable Stringovi

- Stringovi su objekti klase **String** koja je deo java.lang paketa.
- String literali

  “Ovo je string lliteral !“

  ```
  System.out.println("PI or \U03C0 ");
  ```

  Da li se kreira objekat tipa String ili ne?
- String klasa ne zahteva operator new pri instanciranju.
- Sadržaj String objekata se ne može menjati

```
public final class String
extends Object
implements Serializable, Comparable<String>, CharSequence
```

- znakovi u String objektu su Unicode znakovi (svaki zauzima 2 bajta)

Stringovi

- Neki metodi klase `String` (`public`)

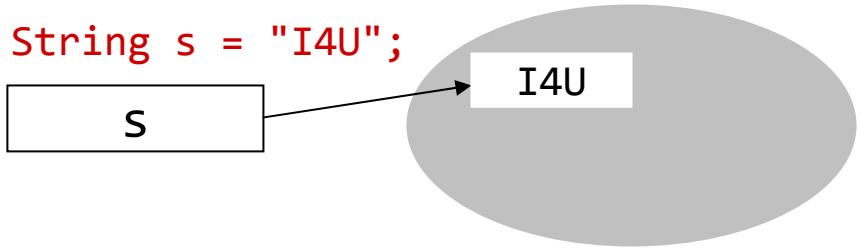
```
char charAt(int index)
String concat(String str)
int compareTo(String str)
```
Makes a lexical comparison. Returns a negative integer if the current string is less than str, 0 if the two strings are identical, and an integer greater than zero if the current string is greater than str.
```
Boolean endsWith(String suffix)
Boolean equals(Object obj)
```
Returns true if the string that obj refers to matches the string that the calling object refers to.
```
Boolean equalsIgnoreCase(String s)
int indexOf(int ch)
int lastIndexOf(int ch)
int length()
String replace(char oldchar, char newchar)
boolean startsWith(String prefix)
String substring(int beginIndex)
String substring(int beginIndex, int endIndex)
String toLowerCase()
String toString() Returns the calling object itself.
String toUpperCase()
String trim()
```

Sve izmene postojećeg stringa se izvode kreiranjem novog

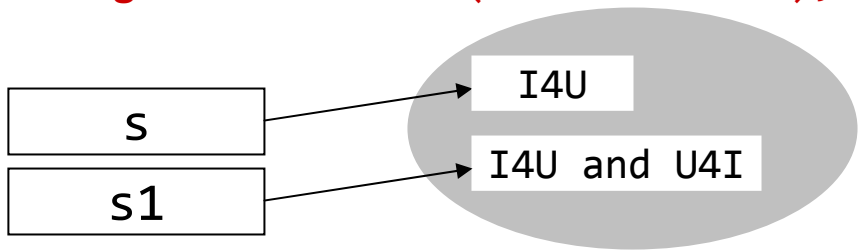
Stringovi

```
String s = "I4U";
String s1 = s.concat("I4U and U4I");
s1.toUpperCase();
```

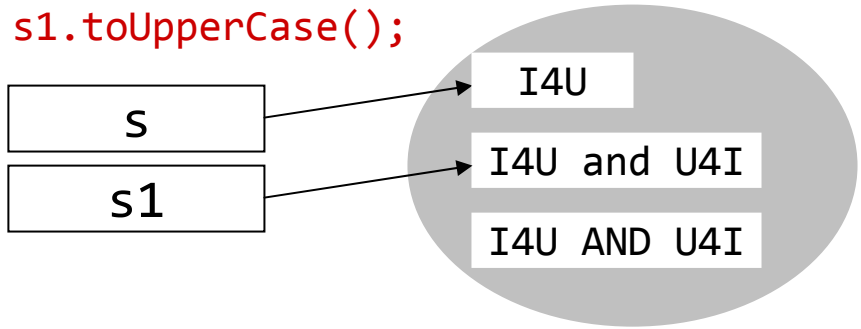

String s = "I4U";



String s1 = s.concat("I4U and U4I");



s1.toUpperCase();



Stringovi

- You should understand how the compiler treats String literals:
  - When the compiler encounters a string literal, it adds an appropriate string to the pool.
  - If the same literal string appears again in the class, the compiler does not create the duplicate. The one already in the pool would be used.
  - If a string literal appears in an expression after the keyword new (such as new ("abc")), the string is created at runtime even if it already exists in the pool.
- Primer

```
String str1 = "Hello Dear!";
String str2 = "Hello Dear!";
String str3 = new String ("Hello Dear!");

str1.equals(str2) // isti sadržaj
str1 == str2 // ista referenca
str1.equals(str3)// isti sadržaj
str1 == str3 //različite reference
```

Stringovi <-> Niz znakova

- ->

```
String text = "To be or not to be";
char[] textArray = text.toCharArray();
// kopira znakove iz text u textArray
   text.getChars(9,12,textArray,0);
```

- <-

```
char[] textArray = {'T','o',' ','b','e',' ','o','r',' ',
                    'n','o','t',' ','t','o',' ','b','e',};
String text1 = String.copyValueOf(textArray);
```

StringBuffer - mutable strings

- U slučaju da su potrene česte promene stringova – dodavanje, brisanje i zamena podstringova
- Kreiranje

```
StringBuffer sb = new StringBuffer("Hello Dear!");
ili
String str = "Hello Dear!";
StringBuffer sb = new StringBuffer(str);
```

StringBuffer - mutable strings

`StringBuffer append(String str)`

`StringBuffer insert(int offset, String str)`

`StringBuffer reverse()`

`StringBuffer setCharAt(int offset, char ch)`

`StringBuffer setLength(int nlength)` Sets the new `length` of the current string buffer to `nlength`. If `nlength` is smaller than the length of the current string buffer, it is truncated. If `nlength` is greater than the current length, the `null` characters are added to the end of the string buffer.

`String toString()`

```
StringBuffer sb = new StringBuffer("Hello");
sb.reverse(); // olleH
sb.insert(1, "My"); // oMylleH
sb.append("ello"); // oMylleHello
```

Dozvoljeno je i

```
String saying = new StringBuffer().append("Many").append(" hands");
```

KAKO RADI?

StringBuffer - mutable strings

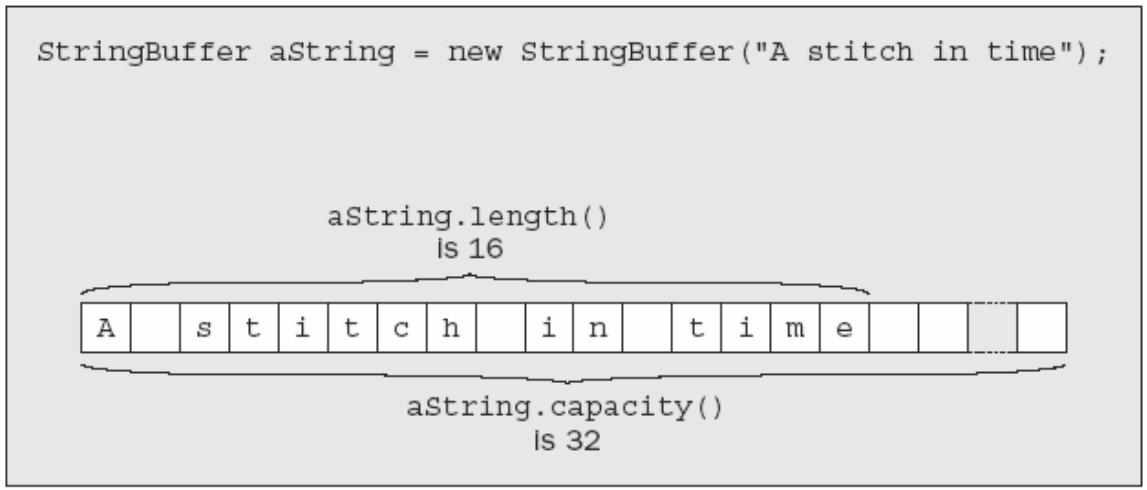
Zadatak 2.

Napisati program kojim se proverava broj parametara koje je main f-ja preuzela u argumentu:

- Ako je broj parametara 0 štampati komentar da ih nema
- Ako ih ima onda ih popakovati u `StringBuffer` objekat i nakon toga ih štampati

StringBuffer - mutable strings

A StringBuffer object contains a block of memory called a buffer, which may or may not contain a string, and if it does, the string need not occupy the entire buffer. **Thus, the length of a string in a StringBuffer object can be different from the length of the buffer that the object contains.** The length of the buffer is referred to as the capacity of the StringBuffer object **(sb.capacity())**.



When you create a StringBuffer object from an existing string, the capacity will be the length of the string plus 16. Both the capacity and the length are in units of Unicode characters, so twice as many bytes will be occupied in memory.

StringBuffer - mutable strings

```
public class Test {
 public static void main(String args[]){
      StringBuffer prazan=new StringBuffer();
      StringBuffer napunjen=new StringBuffer("ja");
      System.out.println("napunjen.length()="+napunjen.length());
      System.out.println("prazan.length()="+prazan.length());
      System.out.println("napunjen.capacity()="+napunjen.capacity());
      System.out.println("prazan.capacity()="+prazan.capacity());
      }
}
```

- provera kapaciteta int `theCapacity = aString.capacity();`
- eksplicitno povećanje kapaciteta `aString.ensureCapacity(40);`