Klase i objekti u Javi

Uvođenje vlastitih tipova podataka

- Tijekom programiranja je često potrebno kreirati vlastite tipove podataka, kao što je npr. struktura koja sadrži podatke o datumu (dan, mjesec i godinu)
- U programskom jeziku C to je moguće postići korištenjem struktura:

```
typedef struct {
        int day;
        char month[10];
        int year;
 } date;
```

- U programskom jeziku Java za uvođenje novih tipova podataka potrebno je kreirati klasu koja osim podataka može sadržavati i funkcije (metode)

Primjer jednostavne klase: Date

- Klasu "Date" moguće je definirati na sljedeći način:

```
class Date {
        int day;
        String month;
        int year;
}
```

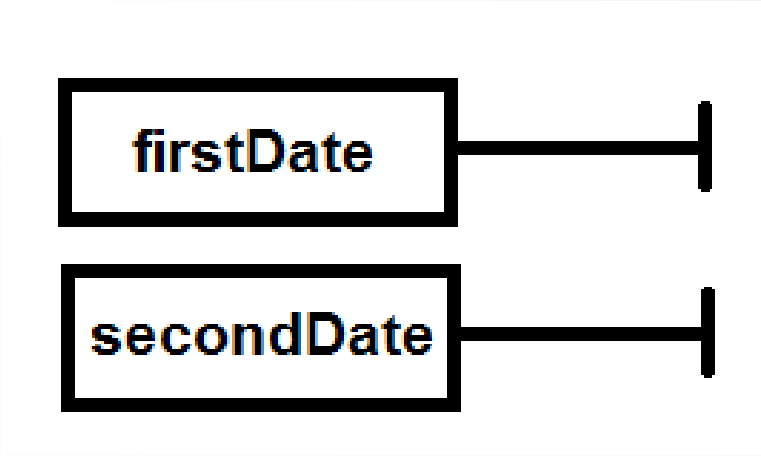
- Deklaracija klase "Date" započinje ključnom riječju "class", nakon čega slijedi naziv klase, te unutar vitičastih zagrada navode se tipovi i imena varijabli
- Varijable neke klase nazivaju se **polja** ili **atributi**
- Svaka klasa definira se unutar vlastite datoteke koja nosi identično ime kao i sama klasa, te ima ekstenziju ".java" – naziv datoteke za klasu "Date" bio bi "Date.java"

Korištenje klasa (1/2)

- Klasu koja definira novi tip podatka moguće je koristiti kao i svaki drugi tip, na primjer:

  Date firstDate, secondDate;

- Nakon te naredbe u memoriji se kreiraju reference (pokazivači) kojima je dodijeljen tip "Date", međutim, te reference ne pokazuju na određenu memoriju (imaju vrijednost "null"):

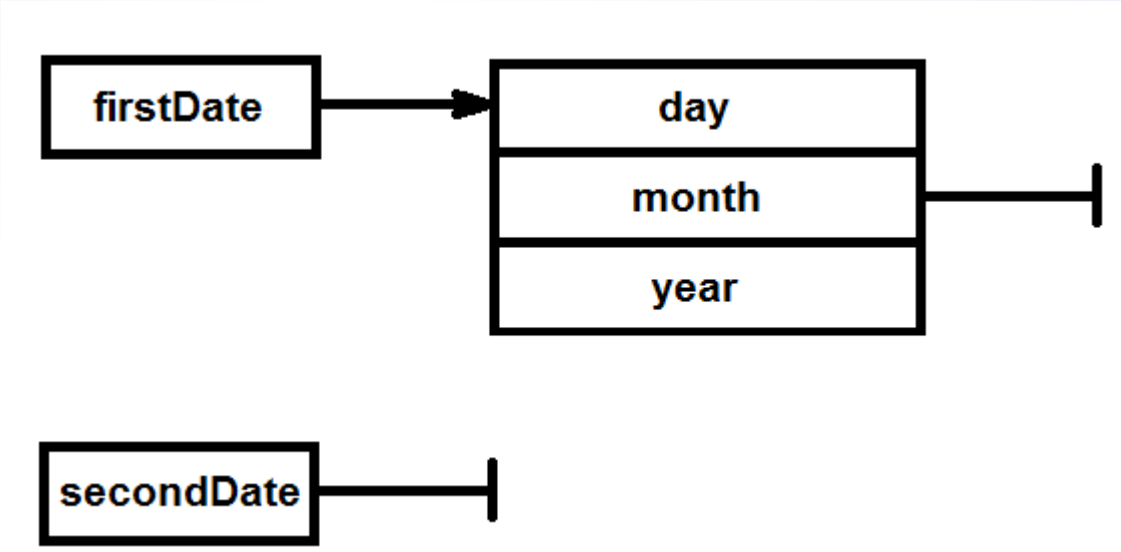


Korištenje klasa (2/2)

- Kako bi se mogle koristiti varijable tipa "Date", potrebno je obaviti njihovu inicijalizaciju korištenjem operatora "new":

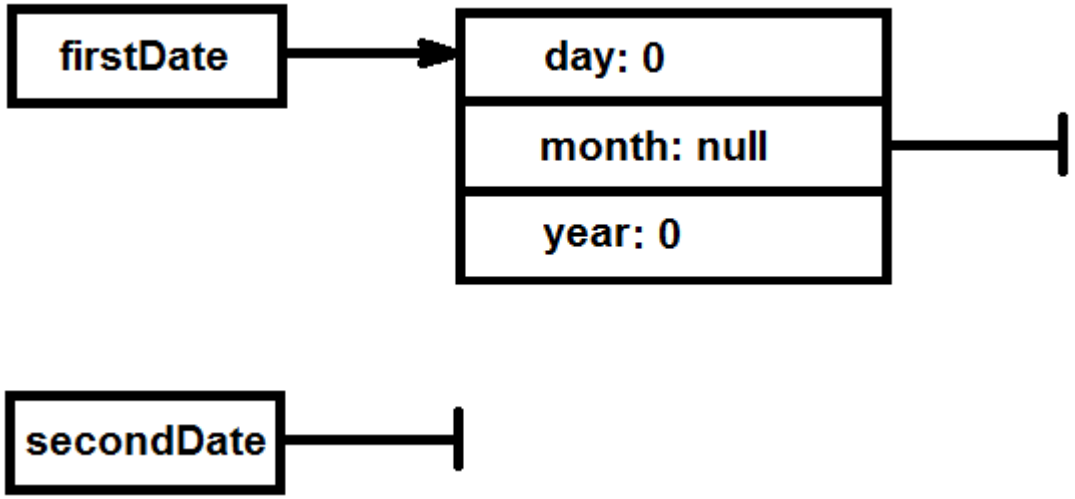
  Date firstDate = new Date();

- Nakon inicijalizacije varijabla "firstDate" ima svoj memorijski prostor u kojem se nalaze vrijednosti samih varijabli:



Operator "new" i objekti

- Služi za kreiranje varijabli kojima je dodijeljen tip koji predstavlja klasu: "Date firstDate = new Date();"
- Korištenjem operatora "**new**" kreira se **objekt klase**: varijabla "**firstDate**" predstavlja **objekt klase "Date"**
- Klasa predstavlja nacrt za kreiranje objekata i definira njihovu strukturu
- Nakon kreiranja objekta, polja unutar njega postavljaju se na inicijalne vrijednosti:



Inicijalizacija vrijednosti objekata

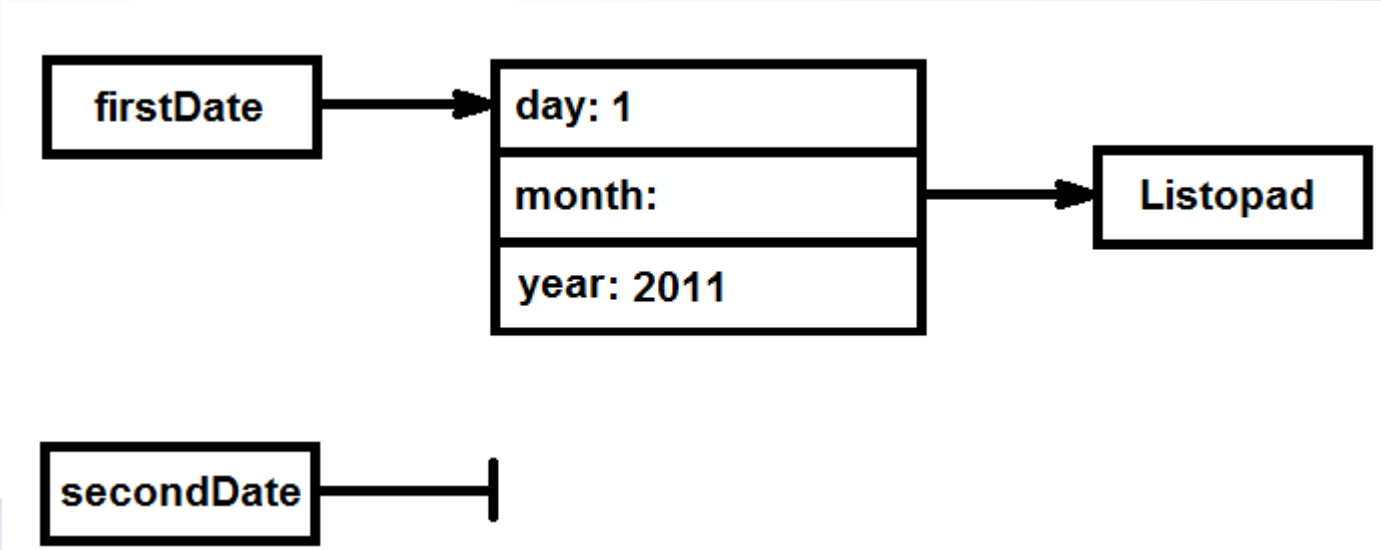
- Tek nakon što je kreiran objekt klase "Date" moguće je pristupati poljima unutar objekta (day, month i year)
- Pristupanje poljima unutar objekta moguće je na sljedeći način:

  firstDate.day = 1;

  firstDate.month = "Listopad";

  firstDate.year = 2011;

- Nakon navedene inicijalizacije stanje u memoriji je sljedeće:

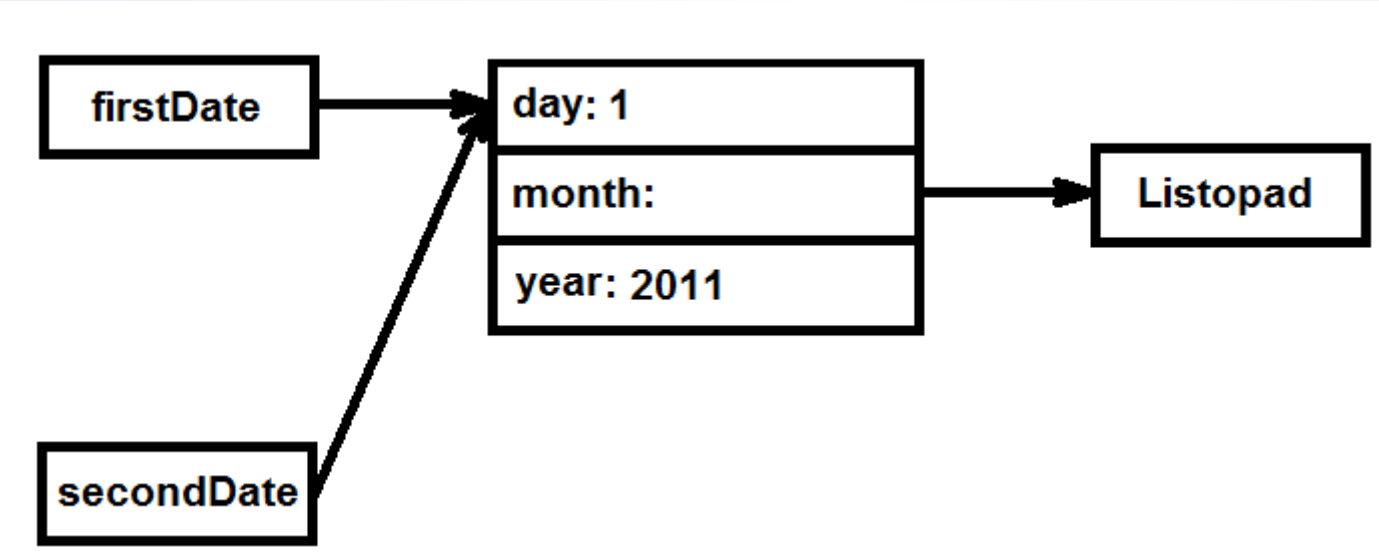


Objekti i reference (1/2)

- Objekti mogu međusobno dijeliti istu referencu, odnosno, dva objekta "pokazuju" na isti dio memorije
- Na primjer, ukoliko se izvrši naredba:

  secondDate = firstDate;

  u memoriji se događa sljedeće:



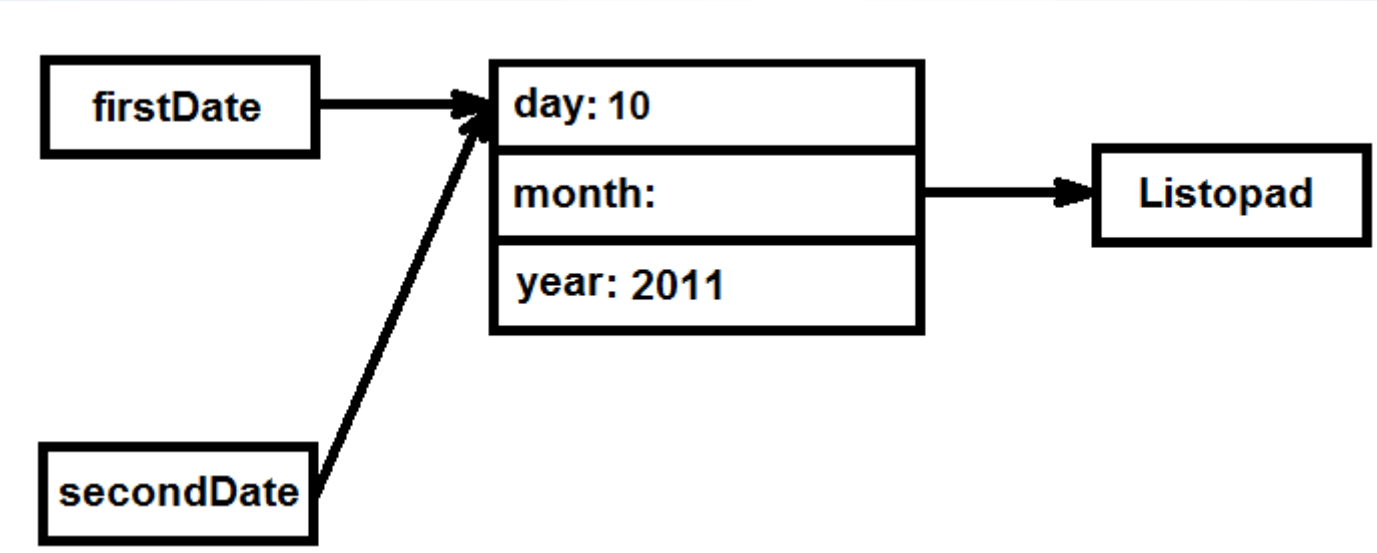
- Varijable "firstDate" i "secondDate" pokazuju na isti objekt

Objekti i reference (2/2)

- Međutim, ukoliko se pomoću korištenja varijable "secondDate" promijeni neka od vrijednosti objekta, ta promjena će se odraziti i na varijablu "firstDate", npr:

  secondDate.day=10;

  rezultira sljedećom promjenom:



Uspoređivanja objekata (1/2)

- Varijable koje predstavljaju objekte mogu se uspoređivati na sljedeće načine:

  if (firstDate == secondDate)... ili

  if (firstDate != secondDate)...

  pri čemu se uspoređuju samo **vrijednosti pokazivača** (reference, adrese u memoriji), a **ne vrijednosti varijabli unutar objekta** (day, month i year)
- Ako je potrebno uspoređivati vrijednosti varijabli, to je moguće na sljedeći način, unutar nove funkcije:

```
boolean isEqual(Date f, Date s) {
  if (f.day == s.day && f.month.equals(s.month) && f.year == s.year)
    return true;
  else
    return false;
}
```

Uspoređivanja objekata (2/2)

- Korištenjem nove metode "isEqual" može se provjeriti imaju li dva objekta iste vrijednosti dana, mjeseca i godine:

  if (isEqual(firstDate, secondDate)...

- Klasa "String" je dio Java API-a i već ima ugrađenu metodu "equals" koja uspoređuje dva Stringa, te je potrebno pozvati tu metodu za uspoređivanje dvaju Stringova (za detalje pogledati Javadoc dokumentaciju):

  public boolean equals(Object anObject)

  Compares this string to the specified object. The result is true if and only if the argument is not null and is a String object that represents the same sequence of characters as this object.

Polja u Javi

- Polja u Javi su također predstavljena pomoću objekata, odnosno, za kreiranje polja potrebno je koristiti operator "new":

  int[] array = new int[100];

- Moguće ih je inicijalizirati već kod same deklaracije:

  int[] primes = {2, 3, 5, 7, 11};

- Veličinu polja moguće je dohvatiti pomoću varijable "length" kojoj se pristupa na sljedeći način:

  array.length

- Korištenjem te vrijednosti moguće je pomoću "for" petlje proći kroz sve elemente polja:

```
for (int i = 0; i < array.length; i++) {
   System.out.println(array[i]);
}
```

Stringovi

- Klasa String predstavlja vrlo često korištenu strukturu podataka u Javi
- String vrijednosti definiraju se unutar dvostrukih navodnika, npr. "Hello world!"
- String objekte moguće je deklarirati na sljedeći način (nije potrebno koristiti operator "new"):

  String a = "Hello";

  b = a; // reference "a" i "b" pokazuju na istu memoriju
- Konkatenacija Stringova moguća je i pomoću operatora "+":

  a = a + " World!";
- Nakon konkatenacije stvara se novi objekt "Hello World!" koji se pridružuje referenci "a", a referenca "b" i dalje ima vrijednost "Hello"

Operacije sa Stringovima

String s1 = "Prvi String";
String s2 = "Drugi String";

- Usporedba Stringova:
  if (s1.equals(s2))...
- Dohvaćanje duljine Stringa:
  if (s1.length() > s2.length())...
- Dohvaćanje određenog znaka iz Stringa:
  char znak = s1.charAt(5); //vraća 'S'
- Pronalaženje pozicije podstringa unutar Stringa:
  int i = s1.indexOf("String"); //vraća '5'
- Dohvaćanje podstringa unutar Stringa:
  String s3 = s.substring(5); //s3 = "String!";

Editiranje Stringa nakon kreiranja objekta

- Klasa String ima svojstvo "nepromjenjivosti" objekata nakon što se kreiraju (engl. ***immutable objects***)
- Ponekad je u praksi potrebno mijenjati već stvoreni String objekt, za što se koristi klasa "StringBuilder"
- Objekti klase "StringBuilder" se ponašaju kao objekti klase "String", samo što se može mijenjati njihovu sadržaj, odnosno", oni su "promjenjivi" (engl. ***mutable objects***)
- Klasa "StringBuilder" ima mnoštvo metoda za manipuliranje Stringovima kao što su: ***length*** (za određivanje duljine Stringa), ***insert*** (za umetanje Stringova u String), ***delete*** (za brisanje dijelova Stringova), ***replace*** (za zamjenu dijelova Stringova), ***substring*** (za dohvaćanje dijelova Stringova), ***setCharAt*** (za postavljanje znaka na određeno mjesto u Stringu) itd.

Rad *Garbage Collectora*

- Nakon što se neki objekt tijekom rada Java programa prestane koristiti i više nije potreban, započinje proces oslobađanja memorije koju taj objekt zauzima
- Takvi objekti i varijable prosljeđuju se tzv. "*Garbage collector*" sustavu za upravljanje memorijom koja oslobađa memoriju objekta i omogućava ponovno iskorištavanje
- Radi se o automatskom procesu, što znači da određenu memoriju (npr. koja je zauzeta naredbom "***malloc***" u programskom jeziku C) nije potrebno "**ručno**" oslobađati pozivanjem određenih funkcija ("***free()***" u programskom jeziku C)
- *Garbage collector* je teoretski moguće (prijevremeno) pozvati "ručno" pozivom metode "finalize()" nad objektom

Učitavanje podataka s konzole

- Kako bi korisnik mogao definirati podatke koje koristi Java program, mora mu se omogućiti unos tih podataka preko konzole
- U Javi postoji klasa "Scanner" koja služi za tu namjenu
- Objekt klase "Scanner" potrebno je kreirati na sljedeći način:

  Scanner unos = new Scanner(System.in);
- Nakon kreiranja objekta "unos", od korisnika je moguće zatražiti unos cijelog broja na sljedeći način:

  int uneseniBroj = unos.nextInt();
- Slično kao što se unose cijeli brojevi, unose se i ostali tipovi podataka
- Ukoliko korisnik unese krivi tip podatka, događa se iznimka (engl. *exception*)

Ispisivanje podataka na konzolu

- Za razliku od unošenja podataka, gdje se kod kreiranja objekta koristi klasa koja predstavlja **standardni ulaz** ("System.in"), kod ispisivanja podataka je potrebno koristiti klasu koja koristi **standardni izlaz** ("System.out")
- Prilikom korištenja naredbe za ispisivanje podataka ("System.out.println"), potrebno je definirati String koji će se ispisivati na konzolu, npr:

  System.out.println("Hello World!");
- Osim metode "println" koja ispisuje zadani String i pomiče kursor u sljedeći redak, postoje i metode "printf" koja definira format ispisivanja Stringa (slično kao i u programskom jeziku C), te metoda "print" koja ispisuje zadani String bez pomicanja kursora u sljedeći redak

Vidljivost klasa unutar paketa

- Ukoliko se Java aplikacija sastoji od više klasa, često je potrebno organizirati ih u više različitih paketa
- Međutim, ako je potrebno unutar jedne klase koristiti klasu koja se nalazi u drugom paketu, tu drugu klasu potrebno je označiti modifikatorom "public"
- Na primjer:

```
package point;
public class Point {...
----------------------------------
package line;
import point.Point;
public class Line {
private Point p1;
```

Vidljivost varijabli unutar objekata

- Varijablama unutar objekata ne smije biti dozvoljen “izravan” pristup, kako ne bi došlo do nekontroliranih izmjena podataka
- Kako bi se to postiglo, varijablama je potrebno dodati modifikator “private”, koji onemogućava izravno korištenje varijable
- Na primjer: **private int x;**
- Međutim, kako bi bilo omogućeno dohvaćanje i postavljanje vrijednosti varijable, trebaju postojati "javne" metode za tu namjenu
- Javne metode moraju omogućiti funkcionalnost postavljanja i dohvaćanja vrijednosti varijabli, kako bi se pomoću njih rukovalo varijablom, bez izravnog pristupa
- Takve metode zovu se "getter" i "setter" metode

"Getter" i "Setter" metode

- Ako je definirana varijabla

  private int x;
- "getter" i "setter" javne metode izgledaju ovako:

```
public int getX() {
   return x;
}
public void setX(int x) {
   this.x = x;
}
```
- "Getter" i "setter" metode najčešće nije potrebno napisati "ručno", već ih je moguće automatski izgenerirati pomoću razvojnog okruženja (Eclipse nudi tu opciju)

Konstruktori (1/2)

- Konstruktori su posebne metode koje služe za kreiranje objekata
- Nazivaju se isto kao i same klase, mogu primati ulazne parametre te sadrže naredbe koje inicijaliziraju početno stanje objekta
- Na primjer:

```
public class Student {
        private String jmbag;

        public Student(String jmbag) {
             this.jmbag = jmbag;
        }
}
```

Konstruktori (2/2)

- Ukoliko se ne navede konstruktor, Java compiler će automatski izgenerirati podrazumijevani (engl. *default*) konstruktor koji ne prima nikakve parametre i ne sadrži nikakvu inicijalizacijsku logiku:

```
public Student() {}
```

- Objekti se instanciraju pomoću operatora "new":

```
Student student = new Student("0036274849");
```

- Svaka klasa može imati više različitih konstruktora koji se razlikuju po broju i tipu ulaznih parametara
- Tehnika uvođenja istoimenih metoda/konstruktora s različitim brojem i/ili tipom ulaznih parametara naziva se *method overloading*

Dodavanje metoda klasama

- Većina klasa osim varijabli (polja) sadrži i funkcije (metode) koje koriste podatke unutar objekata
- Svaka metoda ima svoj naziv, te može (ali i ne mora) primati ulazne parametre, te vraćati rezultate
- Ulazni parametri mogu biti primitivni tipovi ili referentni tipovi
- Na primjer, klasa "Line" može sadržavati funkciju koja računa duljinu linije na osnovi točaka u koordinatnom sustavu koji definiraju samu liniju:

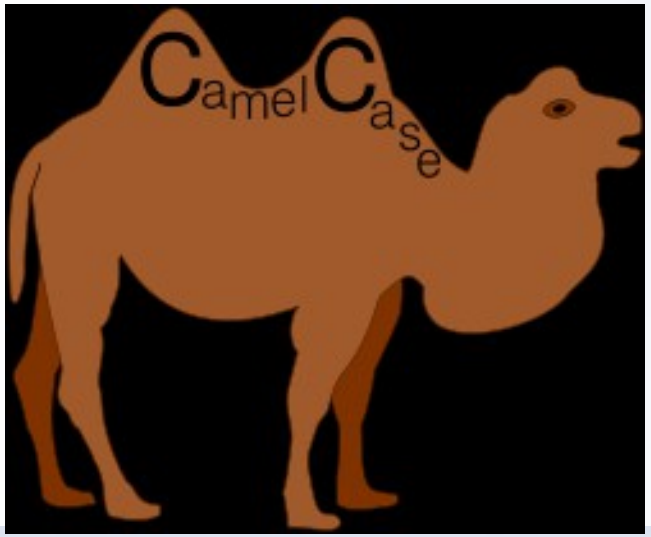
```
public double lineLength() {
    return Math.sqrt(
        Math.pow(p2.getX() - p1.getX(), 2) +
        Math.pow(p2.getY() - p1.getY(), 2));
}
```

Klasa Math

- U sklopu Java API klasa postoji klasa **Math** koja sadrži niz metoda za matematičke izračune, kao što su **abs** (za računanje apsolutne vrijednosti), **sin** i **cos** (za računanje vrijednosti trigonometrijskih izračuna), **pow** (za računanje eksponenata), **sqrt** (za računanje kvadratnog korijena) itd.
- Metode iz klase **Math** mogu se koristiti bez potrebe kreiranja objekta klase **Math**, već "izravno" iz same klase:
- Na primjer: **Math.pow(2, 4);**
- Metode koje imaju svojstvo "izravnog" pozivanja iz same klase nazivaju se **statičke metode** i kod njihove definicije koristi se ključna riječi "static"
- Statičke metode su najčešće one metode koje su zajedničke za sve objekte neke klase, tj. svi objekti koriste istu formulu za neki izračun itd.

Nazivi klasa

- Često se u praksi nazivi imena tvore od više riječi, što zahtijeva korištenje posebno naznačavanje svake od riječi kako bi naziv bio čitljiviji
- Za tu namjenu koristi se konvencija "CamelCase" koja definira način imenovanja klasa kod koje svaka nova riječ mora biti napisana velikim slovom, što olakšava čitanje i razumijevanje
- Na primjer: **P**rogramiranje**UJ**ezikuJava



Primitivni i referentni tipovi

- Tipovi podataka u Javi dijele se na dvije kategorije: **primitivne tipove** i **referentne tipove**
- Primitivni tipovi podataka su tipovi uglavnom naslijeđeni iz programskog jezika C: **boolean, byte, char, short, int, long, float** i **double**
- Za kreiranje primitivnih tipova nije potrebno kreirati objekte klase, već je dovoljno napisati deklaraciju (npr. **int i;**) i varijabli će se dodijeliti inicijalna vrijednost (0)
- Referentni tipovi podataka su tipovi vezani uz klase, odnosno, potrebno je kreirati objekte koji predstavljaju reference
- Na primjer:

  Scanner unos = new Scanner(System.in);

  Line linija = new Line();

Operacije s primitivnim i referentnim tipovima

- S primitivnim tipovima u Javi je moguće obavljati operacije kao i u programskom jeziku C:

```
int a = 10;
int  b = 20;
int c = a + b;
```

- Kod referentnih tipova nije moguće obavljati operacije na taj način (jer objekti predstavljaju reference, a one se ne mogu zbrajati), već je potrebno koristiti određene metode:

```
BigDecimal a = new BigDecimal("3.14");
BigDecimal b = new BigDecimal("10.45");
BigDecimal c = a.add(b);
```

(nije moguće napisati **BigDecimal c = a + b;**)

Veza između primitivnih i referentnih tipova

- Za svaki primitivni tip u Javi postoji njegov ekvivalentni referentni tip
- Na primjer za primitivni tip "int" postoji referentni tip "Integer", za primitivni tip "boolean" postoji referentni tip "Boolean" itd.
- Iz primitivnih tipova moguće je kreirati referentne tipove, npr. **Boolean refBoolean = new Boolean(false);** ili

  Float refFloat = new Float(1.23f);
- Obrnuto također vrijedi, iz referentnih tipova moguće je dobiti primitivne tipove:

  boolean bool = refBoolean.booleanValue(); ili

  float f = refFloat.floatValue();

Nepromjenjivi ("Immutable") objekti

- Objekti čija svojstva se ne mogu mijenjati nakon kreiranja samih objekata (kod promjene se kreira novi objekt s promjenom)
- Primjer nepromjenjivih objekata su objekti klase String, Integer, **BigDecimal** itd.
- Ako se izvede sljedeći programski kod:

```
BigDecimal prvi = new BigDecimal("1.23");
BigDecimal drugi = new BigDecimal("2.34");
prvi.multiply(drugi);
System.out.println(prvi);
```

- ispisati će se "1.23"
- Ako se koristi novi objekt, rezultat će biti ispravan:

```
BigDecimal treci = prvi.multiply(drugi);
System.out.println(treci);
```