1. Auditorne vježbe: Klase i objekti

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 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)

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();