UNIVERZA V LJUBLJANI
Fakulteta za računalništvo in informatiko

# PROGRAMIRANJE 2

TOMAŽ DOBRAVEC

## Objektno programiranje

# O OBJEKTNEM PROGRAMIRANJU

- Osnova objektnega programiranja so OBJEKTI: program sestavlja množica samostojnih objektov, ki med seboj lahko sodelujejo.

- Objekt združuje podatke in metode za delo z njimi.

- Objekt izdelamo s pomočjo NAČRTA za izdelavo objektov določenega tipa (za vsak tip objektov obstaja drug načrt).

# O OBJEKTNEM PROGRAMIRANJU

- V objektno usmerjenem programskem jeziku se načrt imenuje RAZRED.
- Na razred lahko gledamo kot na PREDLOGO (štampiljko) za izdelavo objektov.
- Objekt je primerek (instanca) razreda. V programu imamo pogosto več primerkov istega razreda.

# O OBJEKTNEM PROGRAMIRANJU

Primer: z ukazoma

```
String niz1 = new String("Prvi niz");
String niz2 = new String("Drugi niz");
```

ustvarimo dva objekta razreda `String`.

- Vsi objekti razreda `String` poznajo (med drugim) tudi metodo `charAt()`:

```
if (niz1.charAt(0) == niz2.charAt(0)) {
  System.out.prinltn("Niza se ujemata v prvi crki!");
}
```

- Pozor: za klic metode uporabljamo piko!


# KARAKTERISTIKE OBJEKTOV

- Osnovne karakteristike objektov so:
  - **stanje** in
  - **obnašanje**

## Stanje

- **Stanje** objekta določajo njegovi atributi (spremenljivke).
- Objekti istega razreda so lahko v različnih stanjih (različne vrednosti atributov).

# OBNAŠANJE OBJEKTOV

- **Obnašanje** objektov je določeno z metodami.
- Objekti istega razreda se lahko različno obnašajo.
- Obnašanje je odvisno od stanja, v katerem se objekt nahaja.
- Če sta dva objekta istega razreda v istem stanju, se bosta praviloma enako obnašala.

# PRIMER RAZREDA

Naloga: Napiši razred za “izdelavo” dreves.

- STANJA drevesa
  - ime (v drevesnici je več dreves, vsako ima svoje "ime")
  - starost (v letih)
  - višina (odvisna od starosti: prva tri leta drevo zraste za en meter, potem vsako leto 20 cm)

- OBNAŠANJE:
  - vsako pomlad se starost poveča (za ena), spremeni se tudi višina (odvisno od starosti)
  - drevo se zna narisati na zaslon (pozor: tega običajno drevesa ne počnejo ☺)

# STANJE OBJEKTA (ATRIBUTI)

- Stanje objekta je zajeto v **atributih** *(angl. attributes, instance variables, fields)*

- Primer (razred `Drevo`): `starost, visina, ime`

- Vsak objekt ima svoje atribute:
  - objekt atributov ne deli z drugimi objekti,
  - vsak objekt ima svoji kopijo posameznega atributa
  - spreminjanje vrednosti atributa nekega objekta NE vpliva na vrednost istega atributa drugega objekta

Prepričaj se , da imajo atributi `starost` in `visina` v različnih objektih različno vrednost.

# STATIČNE SPREMENLJIVKE

- Statične so tiste spremenljivke, ki jih najavimo z rezervirano besedo **`static`**.
- Statične spremenljivke == spremenljivke razreda.
- Vsi objekti nekega razreda imajo ISTO vrednost statičnih spremenljivk.
- Če eden od objektov spremeni vrednost, se spremeni vsem objektom tega razreda!
- Obstaja SAMO ena kopija spremenljivke za VSE objekte.
- Statično spremenljivko lahko uporabljamo tudi, če nimamo nobenega objekta tega razreda!

# STATIČNE SPREMENLJIVKE

- Primer: če je `ID` statična spremenljivka razreda `NekRazred`, ne potrebujem objekta, da bi spremenil njeno vrednost.

  Namesto NA OBJEKTU

```
NekRazred imeObjekta = new NekRazred();
imeObjekta.ID = 5;
```

  lahko spremenljivko uporabim NA RAZREDU

```
NekRazred.ID=5;
```

Razredu Drevo dodaj statično spremenljivko `ID`, ki šteje, koliko dreves je bilo izdelanih.

# OBNAŠANJE OBJEKTA

- Obnašanje objekta narekujejo **metode**
  *(angl. methods)*

  Primer: drevo1.pomlad() .... Spremeni se vrednost atributov
  drevo1.izrisiSe() .... Vpliv na okolico (izris na zaslon)

- Metode imajo pri objektnem programiranju podobno nalogo kot funkcije pri proceduralnem programiranju.

- “Logika" programa je razbita na eno ali več metod.

- Metoda lahko spreminja objekt (nastavlja atribute) ali vpliva na "okolico" (druge objekte, sistem, ...).

# PARAMETRI METODE

- Poznamo metode brez parametrov in take s parametri.
- Primer metode brez parametrov je metoda izpisiSe(), ki jo kličemo takole:

```
drevo1.izpisiSe()
```

- Primer metode s parametri:

```
void spremeniIme(String novoIme) {
  ime = novoIme;
}
```

- Klic metode `spremeniIme()` iz programa:

```
drevo1.spremeniIme("Hrast");
```

V razred Drevo dodaj metodo `spremeniIme(String novoIme)`

# REZULTAT METODE

- Metode so lahko tipa `void` ali pa vračajo rezultat.
- Primer metode tipa `void` je metoda `izrisiSe()`,
- Metodo tipa `void` kličemo na objektu `drevo1` takole:

```
drevo1.izrisiSe();
```

- Primer metode, ki vrne rezultat: `povprecnaRast()`, ki vrne, koliko je drevo v povprečju zraslo vsako leto:

```
double povprecnaRast() { … }
```

- Rezultat Metode `povprecnaRast()` je tipa `double`.

# REZULTAT METODE

```
double povprecnaRast() {
     if (starost==0)
        return 0;
     else
       return visina / starost;
}
```

- Metodo `povprecnaRast()` kličemo takole:

```
double rast = drevo1.povprecnaRast();
```

(rezultat, ki ga vrne metoda `povprecnaRast()` se shrani v spremenljivki `rast`).

V razred `Drevo` dodaj metodo double `povprecnaRast()`

# STATIČNE METODE

- Metode, ki jih najavimo z rezervirano besedo `static`, so *statične metode.*
- Statične metode so metode razreda -- ne potrebujemo objekta, kličemo jih direktno na razredu.

Primer statične metode: metoda `sin()` razreda `Math`.

Namesto

```
Math m = new Math();
double rezultat = m.sin(3.14);
```

pišemo kar

```
double rezultat = Math.sin(3.14);
```

V razred `Drevo` dodaj statično metodo za izpis navodil za obrezovanje in statično metodo, ki vrne število že narejenih dreves.

# REZERVIRANA BESEDA THIS

- Rezervirano besedo `this` uporabjamo za sklicevanje na trenutni objekt
- Besedo `this` lahko uporabljamo le v metodah, ki niso statične.
- Besedo `this` preberemo kot »jaz«.
- Besedo `this` običajno uporabljamo zato, da preprečimo konflikte z imeni

# REZERVIRANA BESEDA THIS

Primer:

```
void spremeniIme(String ime) {
      this.ime = ime;
}
```

- **Priporočilo:** rezervirano besedo `this` uporabljamo pred vsakim atributom (tudi, kadar ni nevarnosti, da bi prišlo do konflikta z imeni).

V razredu `Drevo` spremeni metodo `spremeniIme()`, kot je napisano zgoraj.

# KONSTRUKTOR

- Konstruktor je podoben metodi, vendar NI metoda.

  Primer: konstruktor za razred Drevo

```
Drevo() {
    … // koda, ki se izvrši
    … // ob izgradnji objekta
}
```

- Namen metode: združuje zaporedje javanskih ukazov
- Namen konstruktorja: »ustvari« objekt

- Konstruktor ima isto ime kot razred.

- Konstruktor ne vrača rezultata (niti `void` ne).

# KONSTRUKTOR

- Konstruktor ustvari razred, čas nastanka razreda pa je najprimernejši čas za nastavitev začetnih vrednosti atributom:

```
Drevo() {
     ID = ID + 1;
     mojID = ID;
     ime = "";
     starost=0;
     visina=0;
}
```

# KONSTRUKTOR

- Razred ima lahko več konstruktorjev; vsi imajo enako ime (ime razreda), vendar različno število in tip parametrov.

- Konstruktorji običajno kličejo eden drugega, v ta namen uporabijo rezervirano besedo this

- Beseda `this` znotraj konstruktorja ima drug pomen kot `this` v metodi:
  - v metodi pomeni 'trenutno izvedbo',
  - v konstruktorju pomeni klic drugega konstruktorja.

V razredu `Drevo` dodaj konstruktor `Drevo(tIme String)`

# KONSTRUKTOR

- V vsakem razredu obstaja vsaj en konstruktor (če ga programer ne napiše, ga doda prevajalnik).

- Privzet konstruktor je konstruktor brez parametrov.

Primer: Razreda

```
class MojRazred {
}
```

in

```
class MojRazred{
  MojRazred() {
    super();
  }
}
```

se prevedeta v popolnoma enak razred.

# REZERVIRANA BESEDA INSTANCEOF

- če imamo objekt `x` in bi radi preverili, ali je `x` objekt razreda `X`, uporabimo rezervirano besedo `instanceof`:

`x instanceof X` (rezultat: `true` ali `false`)

Primer:

```
Drevo d1 = new Drevo();
boolean jeDrevo = d1 instanceof Drevo;
System.out.println(jeDrevo);                    ..... true

Object s = "Test";
System.out.println(s instanceof Drevo)          ..... false
System.out.println(s instanceof String)         ..... true
```

# RAZŠIRITVE RAZREDOV IN DEDOVANJE

- Razširitve razredov in dedovanje so najpomembnejši koncepti objektnega programiranja.

- Osnovni ideja:
  - za izhodišče izberem obstoječ razred in ustvarim njegovega »naslednika«;
  - naslednik ima vse lastnosti (metode in atribute) ENAKE kot izhodiščni razred (ker je od njega vse podedoval);
  - razreda se razlikujeta le po imenu;
  - če želim, lahko v nasledniku nekatere metode spremenim (redefiniram);
  - lahko dodam tudi povsem nove metode in atribute.

# RAZŠIRITVE RAZREDOV IN DEDOVANJE

- Primer razširitve: z rezervirano besedo `extends` ustvarim razširitev razreda.

  Primer: z deklaracijo

```
class Bonsai extends Drevo {

}
```

  ustvarim razred `Bonsai`, ki se ujema z razredom `Drevo` v vseh metodah in atributih.

# RAZŠIRITVE RAZREDOV IN DEDOVANJE

```
class Bonsai extends Drevo {
    …
}
```

- Če želim, lahko (znotraj oklepajev, namesto …) nekatere metode razreda `Bonsai` napišem na novo (redefiniram).
- Napišem lahko tudi nove metode (take, ki v razredu `Drevo` ne obstajajo).
- Metode, ki jih v novem razredu ne redefiniram, ostanejo enake kot v njegovem predniku (so od njega podedovane).

# RAZŠIRITVE RAZREDOV IN DEDOVANJE

- Izrazi:
  - Prvotni razred je *prednik* (oče, nadrazred, *angl. super class*)
  - Razširjeni razred je *potomec* (sin, podrazred, *angl. sub class*),

Razred `Bonsai` je potomec (podrazred) razreda `Drevo`.

Razred `Drevo` je prednik (nadrazred) razreda `Bonsai`

Napiši razred `Bonsai`, v katerem redeklariraš metodo `povecajVisino` (bonsai raste po 5 cm prvi 2 leti, potem pa se rast v višino ustavi).

# NOVE METODE IN ATRIBUTI

- Razred `Bonsai` je naslednik razreda `Drevo`, zato pozna vse metode razreda `Drevo`
- `Bonsai` lahko nekatere metode razreda `Drevo` redefinira (prejšnji primer: `povecajVisino()` in `izrisiSe()`)...
- ... in še več: `Bonsai` lahko uvede nove atribute in metode, ki jih razred `Drevo` ni poznal!

- `Bonsai` je torej `Drevo` s popravki in dodatki.

V razred Bonsai dodaj atribut `sirina` (koliko je bonsai širok v cm);vsako pomlad naj se širina poveca za 2 cm; širino lahko zmanjšamo, če bonsai ostrižemo (vsako striženje: -1cm); dodaj še metodo za striženje

# KATEREGA TIPA (RAZREDA) JE NEK OBJEKT?

- Naredimo objekt razreda `Drevo`:

```
Drevo d = new Drevo();
```

Ker je `d` primerek razreda `Drevo`, bo ukaz

```
System.out.println(d instanceof Drevo);
```

izpisal `true`.

- Podobno: naredimo objekt razreda `Bonsai`:

```
Bonsai b = new Bonsai();
```

Ker je b primerek razreda Bonsai, bo ukaz

```
System.out.println(b instanceof Bonsai);
```

izpisal `true`.

# KATEREGA TIPA (RAZREDA) JE NEK OBJEKT?

- Toda pozor: `b` je tudi primerek razreda `Drevo`, saj je `Bonsai` naslednik razreda `Drevo` (`b` ima vse kot `Drevo`, morda celo kaj več). Ukaz

```
System.out.println(b instanceof Drevo);
```

bo izpisal `true`, ukaz

```
System.out.println(d instanceof Bonsai);
```

pa seveda `false`.

# KATEREGA TIPA (RAZREDA) JE NEK OBJEKT?

- Ker je `Bonsai` nadgradnja razreda `Drevo`, lahko napišemo tudi tole:

```
Drevo drevo1 = new Bonsai();
```

vendar potem lahko nad objektu `d` kličemo le metode, ki jih pozna `Drevo`, na pa tudi tistih, ki jih je uvedel `Bonsai`.

# PRA-OČE OBJECT

- Vsi razredi v Javi so potomci razreda `Object`.
- Ob deklaraciji razreda prevajalnik sam doda besedi `extends Object`.
- Na vseh javanskih objektih lahko kličemo metode, ki so jih podedovali od praočeta.
- Ena od teh metod je tudi metoda `toString()`, ki jo poznajo vsi objekti.

Ustvari `Drevo drevo1` in pokaži njegove metode, ki so podedovane iz razreda `Object`.

# PRA-OČE OBJECT

- Zanimiva posledica: v Javi lahko z ukazom `System.out.println()` izpisujemo vsak objekt.

Izpiši `drevo1` z ukazom `println()`

- Kaj se je izpisalo? Njegova `toString()` vrednost!

- Če želimo, lahko redefiniramo metodo `toString()` in s tem med drugim dosežemo tudi to, da bomo objekt lahko lepo izpisovali.

V `Drevo` dodaj metodo `toString()` in `drevo1` ponovno izpiši.

# SKRIVANJE ATRIBUTOV

Ustvari primerek razreda Drevo in spremeni atribut starost

- Da bi onemogočili možnost spreminjanja atributov, atribut najavimo z določilom `private`.

  Namesto

  ```
  int starost;
  ```

  pišemo

  ```
  private int starost;
  ```

- S tem smo atribut `starost` skrili in do njega lahko dostopamo samo v razredu samem.

# GETTER/SETTER

- Ko smo skrili atribut `starost`, smo preprečili, da bi ga uporabnik nekontrolirano spremnijal.
- Toda s tem smo preprečili vsakršen dostop do tega atributa – uporabnik ga ne more niti brati niti spreminjati.
- Nastalo težavo rešimo z uporabo **getter**-jev in **setter**-jev (t.j. metod, ki uporabniku omogočajo dostop do skritih atributov).

# GETTER/SETTER

- **getter** je metoda, ki omogoča branje atributa; ime metode: get**ImeAtributa**

  Primer: `getStarost()`

- **setter** je metoda, s katero lahko kontrolirano nastavim vrednost atributa; ime metode: set**ImeAtributa**

  Primer: `setStarost()`

Napiši getter  in setter za starost.

# ABSTRAKTNE METODE IN RAZREDI

- V Java kontekstu beseda **abstraktno** pomeni, da nekaj ni (v celoti) definirano.
- Metoda je abstraktna, če poznamo samo njen podpis, telo metode pa ni na voljo.
- Abstraktni razred je razred, v katerem je vsaj ena od metod abstraktna.
- Iz abstraktnega razreda ne moremo narediti objektov!

# ABSTRAKTNE METODE IN RAZREDI

Primer:

```
abstract class Funkcija {
  abstract double vrednost(double x);
}
```

- Metoda vrednost je v Funkcija samo najavljena (brez telesa)

`new Funkcija();` ... napaka

# ABSTRAKTNE METODE IN RAZREDI - PRIMER

- Ničlo funkcije f(x) lahko iščemo z Newtnovo metodo:
  - začnemo s približkom za ničlo (x0)
  - na vsakem koraku iz prejšnjega približka (x0) izračunamo naslednji približek (x1) po formuli

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)},$$

# ABSTRAKTNE METODE IN RAZREDI – PRIMER

Napiši program za računanje ničel funkcij s pomočjo Newtnove metode.

Delovanje programa za računanje ničel funkcije preveri na naslednjih primerih:

- $f(x) = 2\,x^2 - 5\,x - 3$ (ničle: x1 = -0.5, x2 = 3)
- $f(x) = \sin(x)$ (ničle: 0, π, ….)


# ANONIMNI NOTRANJI RAZRED

- Razred za enkratno uporabo.
- V fazi “izdelave” objekta lahko razred spremenimo (prilagodimo) - nastane anonimni notranji razred.
- Nadomestimo lahko vse ne-statične metode.
- Spremenimo lahko vrednost statičnim spremenljivkam.

Napiši razred TestANR, v katerem boš z uporabo Newtnove metode in anonimnega notranjega razreda izračunal ničle funkcije

$f(x) = x^3 - 3x^2 - x + 3$ (ničle: x1=1, x2=-1, x3=3)

# VMESNIKI

- Vmesnik *(angl. interface)* vsebuje podpise metod.
- Podobna vloga kot abstraktni razred (podaja samo podpise metod, telo ni na voljo).
- Vmesnik lahko vsebuje le konstante (`static final` spremenljivke).
- Vmesnik implementiramo z rezervirano besedo `implements`

# VMESNIKI

Primer:

```
interface Funkcija {
    double vrednost(double x);
}

class Sinus implements Funkcija {
    double vrednost(double x) {
        …
    }
}
```

# VMESNIKI IN ABSTRAKTNI RAZREDI - PRIMERJAVA

## Abstraktni razred

### Prednosti:

- nekatere metode lahko implementiramo,
- imamo lahko atribute

### Slabosti:

- pri razširitvi (`extends`) lahko navedemo samo en razred (Java ne pozna večkratnega dedovanja)

## Vmesnik

### Prednosti:

- v nekem razredu lahko implementiramo več vmesnikov

### Slabosti:

- ni delne implementacije in atributov
-

Primer z Newtnovo metodo napiši še z uporabo vmesnika.

# OP – PRIMER S HOTELOM 1/7

- Problem: investitor želi zgraditi verigo hotelov

- Pot k rešitvi:

  - Investitor najame projektante.

  - Projektanti izdelajo dokument z naslovom
    'Osnovne zahteve za gradnjo hotelov'
    in ga uskladijo z investitorjem.

# OP – PRIMER S HOTELOM



## Osnovne zahteve za gradnjo hotelov

Investitor, ki želi pridobiti dovoljenje za gradnjo hotelov, mora pripraviti projektno dokumentacijo, sestavljeno iz naslednjih delov.

- **Gradbeni načrt**, ki vsebuje natančna navodila za izgradnjo hotela.
- **Načrt dela z obiskovalci**, ki vsebuje opis postopkov za
  - registracijo obiskovalcev ter
  - odjavo obiskovalcev.
- **Računovodski načrt**, ki vsebuje opis postopka za
  - računanje dnevnega prometa.

# OP – PRIMER S HOTELOM 3/7

- Upravna enota potrdi dokument "Osnovne zahteve ..."
- Projektanti izdelajo projektno dokumentacijo, skladno z "Osnovnimi zahtevami za gradnjo hotelov":
  - gradbeni načrt,
  - delo z gosti: knjiga gostov (po sobah),
  - računanje prometa: `število gostov * cena nočitve`,
  - dodatno: možnost pretvorbe med valutami (EUR/SIT).


# OP – PRIMER S HOTELOM 4/7

- Upravna enota preveri, ali je projektna dokumentacija skladna z izdanim dokumentom in s splošno znanimi pravili (gradbeni uzanci)

- Ko investitor prejme potrjeno dokumentacijo, lahko začne graditi.

- Pred gradnjo lahko uporabi le tiste komponente potrjenega projekta, ki niso vezane na konkretno izvedbo:
  - cena nočitve in
  - postopek za pretvorbo SIT/EUR.

# OP – PRIMER S HOTELOM 5/7

- Dokument “Osnovne zahteve” ….. vmesnik (interface)
- Projektanti ….. programerji
- Projektna dokumentacija ….. razred

- Navodilo za izgradnjo hotela ….. konstruktor
- Podatki
  - Cena nočitve ….. statična spremenljivka
  - Knjiga zasedenosti sob ….. ne-statična spremenljivka

# OP – PRIMER S HOTELOM 6/7

- Postopki
  - Pretvorba iz EUR v SIT ….. statična metoda
  - Prijava/odjava obiskovalca ….. ne-statična metoda
- Upravna enota ….. prevajalnik
- Potrjena projektna dokumentacija ….. preveden razred
- Postavljen hotel ….. objekt

# OP – PRIMER S HOTELOM



Napiši program za "gradnjo" hotelov.

1. Napiši vmesnik "Osnovna zahteve"
2. Napiši razred Hotel
3. Napiši program, s katerim "zgradiš" nekaj hotelov
4. Napiši še razred za izgradnjo hotelov z bazenom