Objekti

Objektno pogramiranje

Dogodkovno programiranje

Operacijski sistem / Uporabniški vmesnik

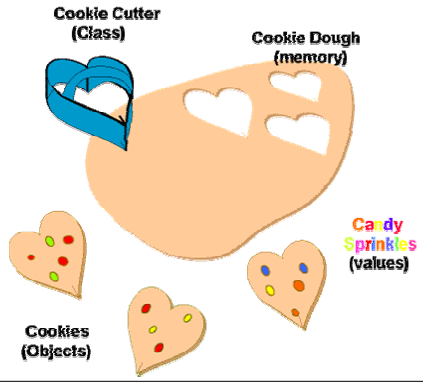
- Objekti:
  - Okna, orodna vrstica, gumbi, …
  - Hierarhija objektov
- "Program"
  - Ni zelo razvidnega toka izvajanja (zanke, vejitve, …)
  - Stvari so odvisne od tega, kaj se zgodi – od dogodkov
- Dogodki:
  - Klik z miško, premik miške, označenje dela besedila, klik na gumb, sprememba vnosnega polja, izbira v menuju, …
- Program: odziv objektov na dogodke

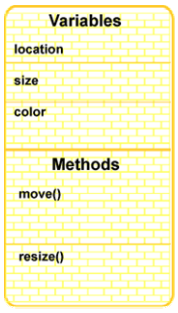
Objekti

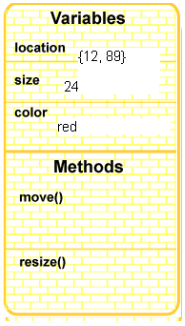
- stanja:
  - Podatki, komponente
- "znanje"
  - Odzivanje na dogodke
- Združeno v celoto
  - Podatki in metode, ki opisujejo neko stvar/objekt
  - Žoga:
    - Podatki: velikost, barva, položaj v prostoru, …
    - Metode: sprememba velikosti, sprememba položaja, …
- Razlika med žogo kot "idejo žoge" in "mojo žogo" (konkretni primerek – realizacija ideje žoge)

Java in objekti

- Razred (class) je opis vrste objekta (načrt, kako naj bo objekt videti) – opis ideje objekta
- Primerek razreda (instanca) – konkretni objekt








```
class Neki {…  new Neki(12, 89, 24, "red",
```

Programiranje v Javi

- Sestavljanje razredov
  - Opis lastnosti objektov
  - Opis metod (“znanja” objektov)
    - Ustvarjanje objektov in njihova uporaba
    - "Ukazovanje" objektom, kaj naj počnejo
    - Objekt za izpolnitev določene naloge potrebuje druge objekte in njihove metode
    - Proženje dogodkov
- Začetek
  - Glavni razred (ki ima metodo main)
  - Izvajanje metode main – ustvarjanje objektov, proženje dogodkov, odzivanje na dogodke, …
  - Applet – zgodba malo drugačna
    - Brskalnik ustvari osnovni objekt
    - Ta se odziva na dogodke kreacija objekta (metoda init), paint, …

Od kje razredi?

- Veliko vgrajenih (oziroma v standardnih knjižnicah) v Javo
  - Math, String, System, BufferedReader, Applet, JFrame, …
- Drugi viri
  - Naši “stari” razredi
  - Drugi programerji
- Potrebujemo ustrezno class datoteko (prevedena java datoteka)

Zgled – želvja grafika

- Razred Turtle
  - Kako je videti "splošna" želva
  - Datoteka `Turtle.class` (+ še dve pomožni datoteki 1, 2)
  - Dokumentacija
    - opis lastnosti objektov tega razreda, opis metod
    - Da znamo uporabljati tovrstne objekte
- Naredimo objekt Turtle (konkretno želvo)
  - `Turtle zelvak; // zelvak bo ime moje želve`
  - `zelvak = new Turtle(); // "ustvarimo" želvo`

Kaj je "zelvak" v programu?

- Ime objekta – njegov naslov (referenca)
- `Turtle zelvak;`
- Povemo, da bo v spremenljivki `zelvak` referenca (naslov) nekega objekta iz razreda `Turtle`
- Analogija:
  - povedali so vam, da vam bodo pošto dostavljali na naslov Janez Slovenec, Zgornji kašelj 7.
  - pisemskega nabiralnika za to pošto še ni! Ni še konkretne škatle, kamor bi poštar vrgel pismo!

Kaj je “zelvak” v programu?

- `new Turtle();`
- Ustvari se objekt (instanca), ki pripada razredu `Turtle`
  - Ima lastnosti, značilnosti, kot je predpisano za objekte tega razreda
  - naredimo konkretni poštni nabiralnik – škatlo rumene barve, s počenim okencem levo spodaj, opraskanimi vratci, …
- `zelvak = new Turtle();`
- Povemo, da je ime (naslov) tega našega novoustvarjenega objekta `zelvak`
- Pošta, poslana na naslov Janez Slovenec, Zg. Kašelj 7, bo pristala v tistem konkretnem poštnem nabiralniku z opraskanimi vratci
- Rečemo: Spremenljivka `zelvak` “kaže” na objekt tipa `Turtle`

Kaj je “zelvak” v programu?

- Ali je v spremenljivki `zelvak` nek konkretni objekt tipa Turtle?
- NE!
- V spremenljivki `zelvak` je le NASLOV tega konkretnega objekta!
-
```
Turtle a, b;
…
a = b;
```
- Sta objekta `a` in `b` ista objekta?
  - Dejansko `a` in `b` sploh NISTA objekta, sta le naslova objektov!
  - `a` in `b` kažeta na isti objekt (ker sta to ista naslova)
- “običajno” govorenje: objekt a, objekt b
- Tudi mi bomo v nadaljevanju uporabljali slednje, a VEDETI zakaj dejansko gre!

Objekt in ime spremenljivke

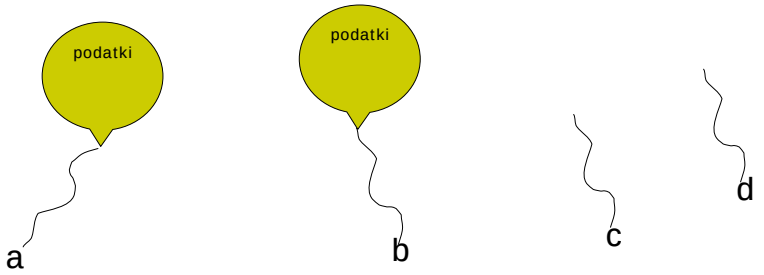
- NekiObjekt a;
  - a je naslov kjer bo objekt (referenca na objekt)
- new NekiObjekt();
  - V pomnilniku se je naredil objekt tipa NekiObjekt() / po pravilih, ki so določena z opisom razreda NekiObjekt
- a = new NekiObjekt();
  - a kaže na novo ustvarjeni objekt

Objekt in ime spremenljivke

- `NekiObjekt a = new NekiObjekt();`
- `NekiObjekt b = new NekiObjekt();`
- `NekiObjekt c, d;`
- Objekti
  - balončki s helijem
- Spremenljivke tipa `NekiObjekt`
  - vrvice za balončke

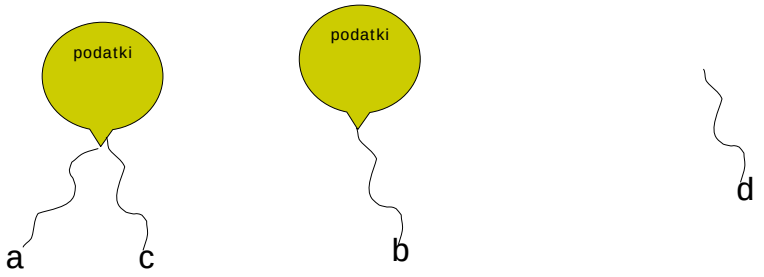
Objekt in ime spremenljivke

- `NekiObjekt a = new NekiObjekt();`
- `NekiObjekt b = new NekiObjekt();`
- `NekiObjekt c, d;`

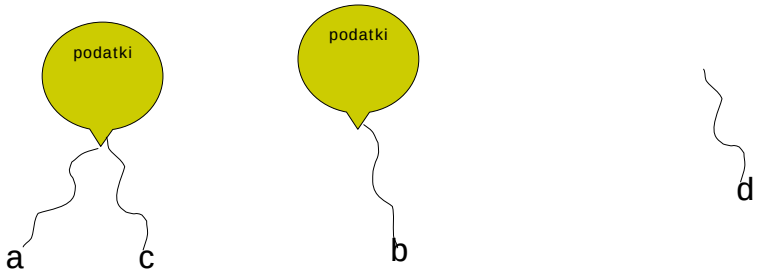


Objekt in ime spremenljivke

- c = a;

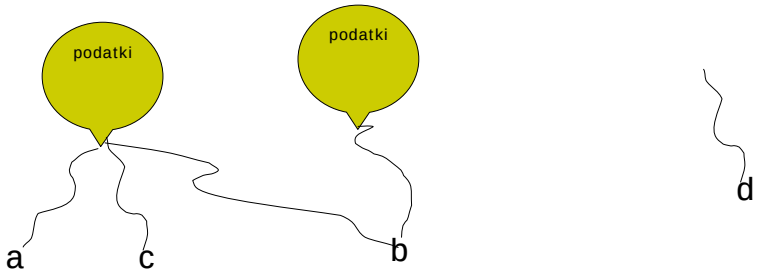


Objekt in ime spremenljivke

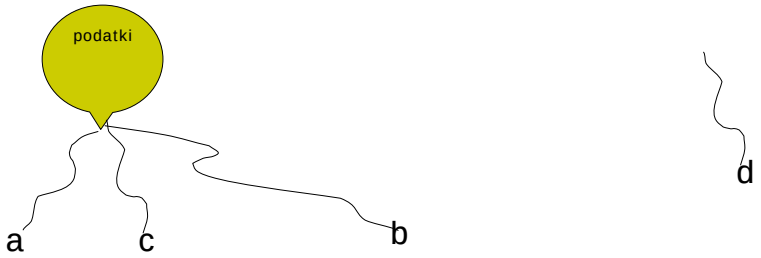


Objekt in ime spremenljivke

- b = a;

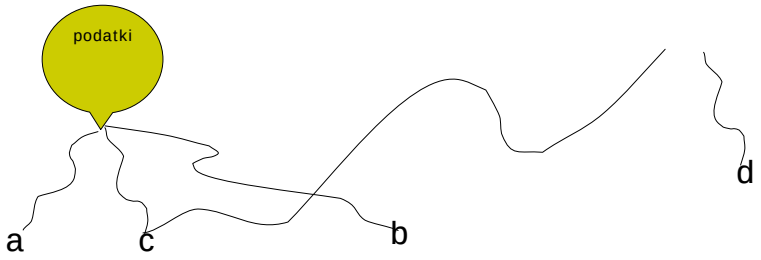


Objekt in ime spremenljivke

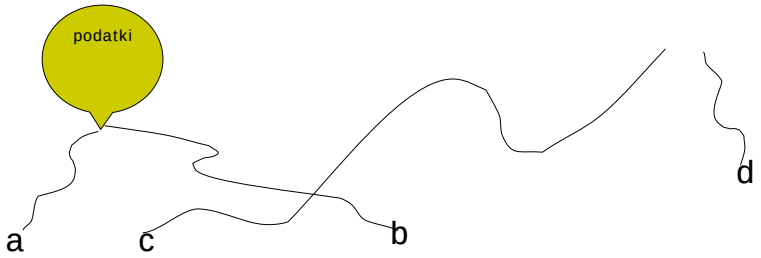


Objekt in ime spremenljivke

- c = d;

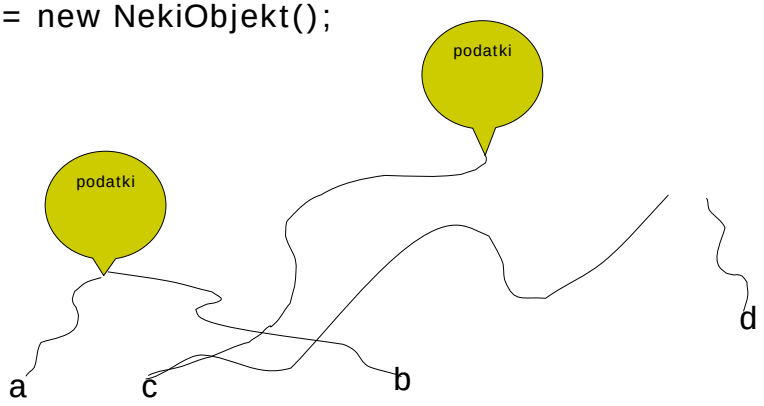


Objekt in ime spremenljivke

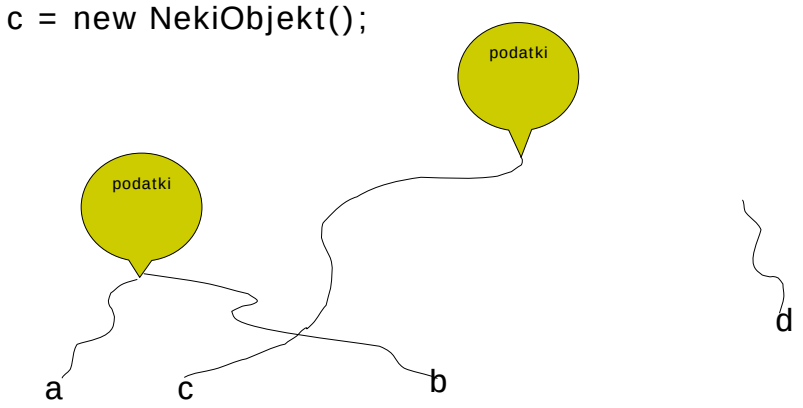


Objekt in ime spremenljivke

```
c = new NekiObjekt();
```



Objekt in ime spremenljivke



Zgled – želvja grafika

- Razred `Turtle`
  - Kako je videti "splošna" želva
  - Ko je narejen razred `Turtle` nimamo še NOBENE konkretne želve, le NAČRT, kako naj bodo želve videti
  - `new Turtle();` // ustvari se konkretna želva
    - instanca razreda `Turtle`
    - Objekt `zelvak` je primerek objekta iz razreda `Turtle`
- Kakšne lastnosti ima vsaka želva:
  - Smer
  - Položaj (x koor, y koor)
- "Stanje" te želve (v trenutku kreacije)
  - Smer (v desno (na vzhod))
  - Na sredi zaslona – v točki (380, 300)

Zgled – želvja grafika

- Kaj znajo vse želve (na katere metode se odzovejo):
  - `barva` // nastavi barvo sledi
  - `narisi` // obrni se za določen kot in pojdi naprej za določeno število korakov
  - `spi` // zaspi za nekaj trenutkov (in z zelvo tudi ves svet)
  - …
- "Ukazovanje"
  - ime_objekta.metoda(parametri)
  - `zelvak.barva(mojaBarva);`
  - `zelvak.spi(2000); // zaspi za 2 sekundi`
  - `zelvak.narisi(90, 30);` // obrni se za 90 st v levo in pojdi naprej za 30

Zelva1.java

```
public class Zelva1
{  // v istem imeniku mora biti datoteka
   // Turtle.class
   public static void main(String[] args)
   {
      Turtle zelvak; // zelvak bo ime moje želve
      // zelvak je naslov neke želve
      zelvak = new Turtle(); // "ustvarimo" želvo
      // na to želvo kaže zelvak
      // zelvak je naslov nove zelve
      // naj želva zelvak nariše kvadrat
      // dejansko: naj želva, ki je na naslovu
      // (na katerega kaze) zelvak narise kvadrat
      zelvak.narisi(90, 40);
      zelvak.spi(1000); // malo počij
      zelvak.narisi(90, 40);
      zelvak.narisi(90, 40);
      zelvak.spi(2000); // malo počij
      zelvak.narisi(90, 40);
   }
}
```

Brez Turtle.class

Obvestilo prevajalnika:

```
C:\WINDOWS\Namizje\P3\Zelva1.java:5: cannot resolve
   symbol
symbol  : class Turtle
location: class Zelva1
      Turtle zelvak; // zelvak bo ime moje želve
```


Dve želvi

- Kaj, če želimo imeti dve želvi?
- `Turtle zelvak; // zelvak bo ime moje želve`
- `zelvak = new Turtle(); // "ustvarimo" prvo želvo`
- `Turtle zelvica = new Turtle(); // zelvica bo ime moje druge želve`

- Zelva2.java

Pogosto uporabljani postopki

- Denimo, da z želvo zelo pogosto izvajamo določen postopek
- Npr. rišemo kvadrat

- Napišemo ustrezno metodo
- `Zelva3.java`

Dedovanje

- "razširjanje" objektov
- `extends`
- Naredimo načrt za razred pametnih želv
  - Znajo risati kvadrate
- `public class PametnaZelva extends Turtle`
- Vsaka pametna želva "zna" vse, kar zna razred `Turtle` in morda še kaj

Razred PametnaZelva

```
public class PametnaZelva extends Turtle
{  // naredimo nov razred "pametnih zelv"
   public void kvadrat(int velikost)
   { // Vsaka pametna zelva ve, kako se nariše
     // kvadrat s stranico velikost
      narisi(90, velikost);
       // metoda narisi je podedovana iz razreda
   Turtle
      narisi(90, velikost);
      narisi(90, velikost);
      narisi(90, velikost);
   }
}
```


Naredimo pametno želvo

```
public class Zelva4
{
   public static void main(String[] args)
   {
      PametnaZelva zelvak; // zelvak bo ime moje pametne želve
      zelvak = new PametnaZelva(); // "ustvarimo" "pametno" želvo

      // naj želva zelvak nariše kvadrat
      // ker je pametna, ji ni potrebno posredovati
      // podrobnih navodil, ampak le ukaz za risanje kvadrata

      zelvak.kvadrat(40);

   }
}
```

Dedovanje

- Izpeljava novih razredov iz obstoječih
- Uvajanje dodatnih metod
- Lahko tudi dodatne lastnosti (podatki)
- Primer:
  - matrike
    - Seštevanje, odštevanje, množenje
  - Kvadratne matrike / `class KvMatrike extends Matrike`
    - Ker je razred izpeljan – ni potrebno na novo pisati metod za seštevanje, odštevanje, množenje
    - Možne dodatne operacije
      - Inverz, deljenje, …

Dedovanje

- Hierarična zgradba razredov
- Vrhnji objekt
  - Object
  - Iz njega izpeljani vsi drugi rezredi
  - Če ne napišemo `extends` …
  - `extends Object`
- "specializacija" objektov

Zakaj dedovanje

- Prilagoditev pripravljenih razredov našim potrebam
- Razred Turtle je sicer povsem v redu, a želvja grafika je pogosto izpeljana drugače
- left, rigth, forward, back
- Kako narediti objekt, ki bo ponazarjal Logo_želvo
  - Na novo
    - Zamudno
    - Možne napake
    - …
  - Vzamemo razred Turtle in ga prilagodimo

Razred Turtle

- Konstante (razredne):
  - RDECA, MODRA, CRNA, SIVA, RUMENA, ROZA, ORANZNA, ZELENA, VIJOLICNA, BELA
- Turtle.BELA
- /** Nastavimo barvo pisanja na dano barvo */
- `public void barva (Color danaBarva)`
- /** Na mestu želve napiši sporočilo */
- `public void napisi (String sporocilo)`
- /** Pocakaj za cakaj milisekund. */
- `public void spi (int cakaj)`
- /** Obrni se levo za levo stopinj; premakni se za naprej korakov.*/
- `public Turtle premakniSe (double levo, double naprej)`
- /** Obrni se levo za levo stopinj; premakni se za naprej korakov in puscaj sled.*/
- `public Turtle narisi (double levo, double naprej)`

Logo želva

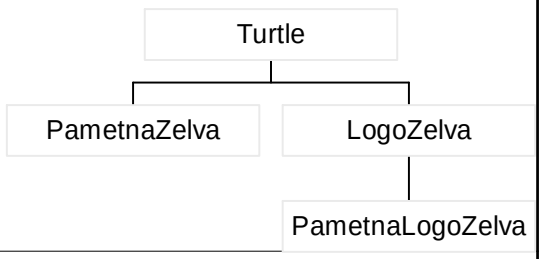
- Ukazi (metode):
  - fd, forward
    - Kako?
    - Naša želva že pozna (preko dedovanja) ukaz narisi (iz razreda Turtle)
    - fd(a) <==> narisi(0,a)
  - bk, back
    - narisi(180,0); // zasukamo se v nasprotno smer
    - narisi(0, a); // ali pa kar fd(a)
    - narisi(180,0); // zasukamo se nazaj

Logo želva

```
public class LogoZelva extends Turtle
{
  public void fd(int a)
  /**  naprej za a */
  {  narisi(0, a); }
  public void bk(int a)
  /** nazaj za a */
  {  narisi(180, 0); // zasukamo se v nasprotno smer
     narisi(0, a); // sled dolzina a;
     narisi(180, 0); // zasukamo se v prvotno smer
  }
  public void right(int a) {
  /** zaskuk smeri v desno za a */
    narisi(-a, 0);
  }
  public void left(int a) {
  /** zaskuk smeri v levo za a */
    narisi(a, 0);
  }
}
```

“Pametna” Logo želva

- `public class PametnaLogoZelva extends LogoZelva`
- Dodatne metode
- Hierarhija objektov:



“Prava” Logo želva

- Logo želve dejansko pri premikanju ne puščajo vedno sledi za sabo
- Pisalo: dvignjeno/spuščeno
- Naredimo tovrstno logo želvo
- Kam jo uvrstiti v našo hierarhijo objektov?
- Ker nam metode iz LogoŽelva ne ustrezajo, izpeljimo kar iz razreda Turtle!

“Prava” Logo želva

- `public class PravaLogoZelva extends Turtle`
- Potrebujemo dodatno stanje!
- `private boolean pisalo = true;`
  `// pisalo je spusceno`
- Metode, ki dvigajo/spuščajo pisalo
- `PravaLogozelva.java`