Programski jezik Java

Interno gradivo za predmet
Algoritmi in programski jeziki (4. letnik)

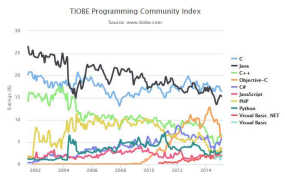
OOP

(neprečiščeno besedilo)

bostjan.vouk@tsc.si

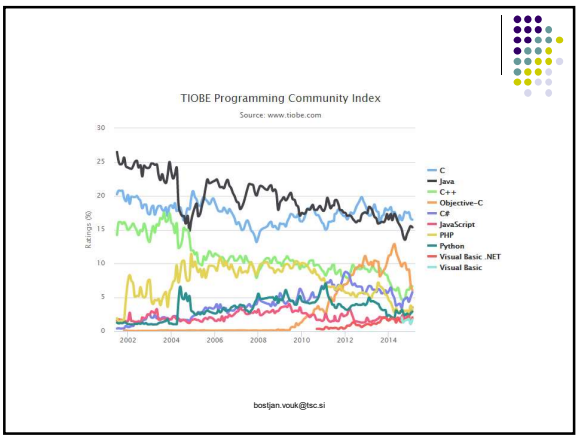
Spomnimo - Zakaj "še" vedno java?

- TIOBE Programming Community Index
- http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html



Feb 2015	Feb 2014	Change	Programming Language	Ratings	Change
1	1		C	16.488%	-1.85%
2	2		Java	15.345%	-1.97%
3	4	^	C++	6.612%	-0.28%
4	3	v	Objective-C	6.024%	-5.32%
5	5		C#	5.738%	-0.71%
6	9	^	JavaScript	3.514%	+1.58%
7	6	v	PHP	3.170%	-1.05%
8	8		Python	2.882%	+0.72%
9	10	^	Visual Basic .NET	2.026%	+0.23%
10	-	^^	Visual Basic	1.718%	+1.72%
11	20	^^	Delphi/Object Pascal	1.574%	+1.05%
12	13	^	Perl	1.390%	+0.50%
13	15	^	PL/SQL	1.263%	+0.66%
14	16	^	F#	1.179%	+0.59%
15	11	vv	Transact-SQL	1.124%	-0.54%
16	30	^^	ABAP	1.048%	+0.69%

bostjan.vouk@tsc.si



Memory allocation

- Uvod v OOP
- http://www.youtube.com/watch?v=W8nNdNZ40EQ

Sinonimi za računalniške programe

- V okviru razvoja IS med drugim nastanejo računalniški programi.
- Izraz “Računalniški program” ima številne sinonime:
  - Program
  - Aplikacija
  - Aplikativni sistem
  - Informacijska rešitev
  - Računalniška rešitev
  - …

Programski jezik java

- Je postopkoven in objektno usmerjen.
  - Postopkoven
    - opisati je potrebno postopek kako pridemo do rešitve
  - Objektno usmerjen
- Objekti so osnovni elementi objektno usmerjenega programiranja (OOP).
- Objekt vsebuje operacije in podatke. Tako vsak objekt pozna samega sebe in zna nad svojimi podatki izvajati operacije.
- Objekt navzven deluje kot "črna škatla", ki iz okolice sprejema in pošilja sporočila.

Osnovne značilnosti OOP

- Osnovni koncepti OOP: razred, objekt, atribut, metoda
- **Združevanje** (enkapsulacija) Podatki in metode so združeni v predmetu. Ožje ščitenje podatkov pred neposrednim dosegom.
  - Enkapsulacija - vmesnik (interface) pove kaj lahko delamo z objektom. Implementacija objekta (kako) je skrita znotraj objekta.
- **Dedovanje**(inheritance) Iz razreda izpeljemo nov razred. Prevzem lastnosti in metod.
  - Dedovanje (inheritance) - bolj specializiran objekt lahko izpeljemo iz bolj splošnega.
- **Večličnost** (polimorfizem) Objekti se lahko obnašajo v različnih vlogah.
  - Polimorfizem dovoljuje, da se dva ali več razredov odzivata na isto sporočilo na različne načine.

Izrazoslovje OOP

- **"Razred" pomeni kategorijo stvari**
  - Ime razreda lahko v Javi uporabimo kot tip polja ali lokalne spremenljivke ali kot povratni tip funkcije (metode)
- **"Objekt" pomeni konkretno stvar, ki pripada nekemu razredu**
  - Temu pravimo tudi "instanca"
- **Primer: poglejmo naslednjo vrstico:**
  - **String s1 = "Pozdrav";**
    - String je razred, spremenljivka s1 in vrednost "Pozdrav" sta objekta (ali "instanci iz razreda String")

Temeljni pojmi

- Temelji objektno orientiranega programiranja:
  - Vidnost podatkov ali enkapsulacija
    - vidnost razredov, njihovih polj in metod
  - Polimorfizem (angl. polymorphism)
    - različno obnašanje metod z enakim imenom
  - Dedovanje (angl. inheritance)
    - strukturiranje hierarhije razredov

bostjan.vouk@tsc.si

Dostopna določila

- Vidnost razredov:
  - public - viden povsod
  - brez - viden znotraj paketa
- Vidnost članov (polj in metod):

Ključna beseda	Razredu	Paketu	Podrazredu	Vsem
public	DA	DA	DA	DA
protected	DA	DA	DA	NE
brez	DA	DA	NE	NE
private	DA	NE	NE	NE

bostjan.vouk@tsc.si

Primitivni podatkovni tipi / objekti

- Primitivni podatkovni tipi
  - Osnovni podatkovni tipi: int, double, char, boolean,…
    - ne moremo jih narediti sami
- Objekti
  - dobimo jih z Javo(String, Integer, Color, …) ali pa jih naredimo sami.
- Vsi nadgradnja če ni objekt tipa “extends” potem extends Object

bostjan.vouk@tsc.si

Prednosti OOP

- Programi so krajši
- Čas razvoja novih programov je krajši
- Kodo lahko ponovno uporabimo (reusability)
- Vzdrževanje programov je lažje
- Razvoj grafičnih uporabniških vmesnikov (GUI) je lažji in hitrejši
- Podatkovne baze lahko vsebujejo kompleksne podatkovne tipe in multimedijske elemente

Razlika med razredi in objekti

- class Oseba {
  private String ime;
  private String priimek;
  public Oseba(String i, String p) {
  ime=i;
  priimek=p;
  }
  }
- main() {
  Oseba moja_oseba;
  moja_oseba=new Oseba("Janez","Novak");
  }

Rezervirani besedi *this in super*

- this
  - pomeni nanašanje na trenutni primerek tega razreda
- super
  - pomeni nanašanje na nadrazred
  - pogosto uporabljamo v konstruktorju

Ustvarjanje in brisanje objektov

- *Constructor*
  - ima isto ime kot razred
  - lahko jih je več in imajo različne parametre
  - ne vrača ničesar
  - v prvi vrstici lahko kličemo super()
  - v njem inicializiramo objekt za ta razred
- *Finalizer*
  - ustvarimo ga z finalize()
  - izvede se tik preden zbiralec odpadkov odstrani objekt
  - težko jih pišemo (čas)
  - z njim počistimo za objektom

Ustvarjanje primerkov razreda

```
public class Oseba {
    // koda razreda
}

public static void main (String[] args) {
    Oseba os1, os2;
    os1 = new Oseba();
    os2 = new Oseba("Janez");
}
```


Dodajanje funkcionalnosti

```
public class Oseba {
    private String imeOsebe;
    private Date datum_rojstva;
    public Oseba (String ime, Date datum) {
      imeOsebe = ime;
      datum_rojstva = datum;
    }
    public int izracunStarosti() {
      Date danes = new Date();
      return danes.getYear() - datum_rojstav.getYear();
    }
}
```

Dedovanje

```
class Student extends Oseba {
private Date datum_vpisa;
    public Student (String ime, Date datum) {
      super(ime);
      datum_vpisa = datum;
    }
    public boolean bruc () {
      if (datum_vpisa.getYear()==datum_rojstva.getYear()+18)
      return true;
      else return false;
    }
}
```

bostjan.vouk@tsc.si

Dedovanje primer

```
class B {
int x;
void setIt (int n) { x=n;}
void increase () { x=x+1;}
void triple () {x=x*3;};
int returnIt () {return x;} }
```

```
class C extends B {
void triple () {x=x+3;} // override existing method void
quadruple () {x=x*4;} // new method }
```

bostjan.vouk@tsc.si

Dedovanje primer

```
public class GetRich {
public static void main(String[] args) {
B b = new B();
b.setIt(2);
b.increase();
b.triple();
System.out.println( b.returnIt() ); // prints 9
C c = new C();
c.setIt(2);
c.increase();
c.triple();
System.out.println( c.returnIt() ); // prints 6 } }
```

bostjan.vouk@tsc.si

Razredne spremenljivke

```
class Student extends Oseba {
    private Date datum_vpisa;
    // razredna spremenljivka
    private static int st_studentov=0;
}
```

bostjan.vouk@tsc.si

Razredne metode

```
class Student extends Oseba {
    private Date datum_vpisa;
    private static int st_studentov=0;
    static int getSteviloStudentov() {
      return st_studentov;
    }
    public static void main( String[] args) {
      System.out.println( "Število študentov je " + Student.getSteviloStudentov());
    }
}
```

bostjan.vouk@tsc.si

Abstraktni razred

```
abstract class Vozilo {
    abstract String VrniBarvo();
}
```

```
class Avto extends Vozilo {
    private String barva;
    public String VrniBarvo() {
      return barva;
    }
}
```

bostjan.vouk@tsc.si

Preobteževanje (overloading)

- *Več metod z enakim imenom.*
  - Metode se morajo razlikovati ne v imenu ampak podpisu.
  - Podpis metode: ***ime + število in tipi parametrov***
  - return tip ni del podpisa
  - Tudi navadne metode ne le konstruktorji so lahko preobtežene. Možnost, da imamo enako poimenovano metodo, ki pa sprejema parametre različnega tipa. Metoda ploščina, ki kot parameter dobi objekt iz razreda Kvadrat, Krog ali Pravokotnik.

Preobteževanje (overloading)

- **Uporaba**: ploscina(bla), kjer je bla lahko objekt tipa Kvadrat,Krog ali Pravokotnik.
- Tri metode, z enakim imenom a različnimi podpisi. Enostavnejša uporaba kot tri metode z različnimi imeni ali ena metoda kjer preverimo kakšen objekt smo dobili.
- Enostavneje, če dobimo še četrti tip objekta - v glavno kodo ni potrebno posegati klic je še vedno ***ploscina(bla)***.
- Primer preobteženih metod:
  - Math.abs() lahko uporabimo na tipu double in tipu int.
    - Dve metodi

- **Prekrite metode**:
  - V predhodniku (neposrednem - extends … ali posrednem) definirana metoda nam ne ustreza.
- **Predefiniranje(overriding)**:
  - prekrite metode
    - Enak podpis metode kot pri predniku

Razlika statična/objektna metoda

- Statična metoda ima dostop do statičnih komponent.
- Objektna metoda ima dostop do objektnih komponent objekta, na katerem je klicana in do statičnih komponent.
- Statične komponente so vsebovane v razredu, objektne (nestatične) pa v objektih.
- Vsak objekt vsebuje svojo kopijo objektnih komponent, vsaka statična komponenta pa vedno obstaja v eni sami kopiji.
- ***Statične metode**: Vsaka metoda vrača rezultat in sprejme argumente. Ko metodo napišemo vedno povemo, kakšnega tipa je rezultat, ter kakšna so imena in tipi argumentov.*
- ***Objektne metode**: metodam, ki niso statične pravimo objektne metode.*

bostjan.vouk@fsc.si

Razlika statična/objektna metoda

- Primer:
  - s **statično metodo** bi sešteli dve kompleksni števili takole:
    Kompleksno **c = vsota_static(a, b);**
  - z **objektno metodo** pa:
    Kompleksno **c = a.vsota(b);**
- Oba načina imata enak učinek. Drugi način izraža osnovno načelo objektnega programiranja.
  - Objekt ***a*** vsebuje metodo vsota. Objekt vsebuje komponente in objektne metode. Komponento ***re*** v objektu ***a*** pokličemo z ***a.re***.

bostjan.vouk@fsc.si

Dedovanje

- Koncept dedovanja
  - podrazred podeduje atribute in metode nadrazreda
- Redefinicija metod
  - v podrazredu lahko ponovno deklariramo podedovano metodo
  - pri tem si lahko pomagamo z metodo nadrazreda (super)
- Uporaba konstruktorjev
  - konstruktor podrazreda mora poskrbeti za parametre, ki jih zahteva konstruktor nadrazreda
- Dostopno določilo protected
  - podrazredom je omogočen neposreden dostop do podedovanih atributov
- Metode, ki jih ni moč redefinirati
  - **4 tipi: private, static, final, metode v razredih final**

bostjan.vouk@fsc.si

Metode, ki jih ni moč redefinirati

- Metode private
  - V podrazredu niso dostopne (se ne podedujejo)
  - V podrazredu lahko še enkrat deklariramo metodo z enakim imenom in parametri, vendar to ni redifinicija
- Metode static
  - So metode razreda in niso vezane na posamezne objekte
  - Kličemo jih z imenom razreda, ne z imenom objekta: **<ime razreda>.<ime metode>**
  - Metoda je ena sama za osnovni razred in vse naslednike

Metode, ki jih ni moč redefinirati

- Metode final
  - final pomeni, da je neka komponenta (npr. spremenljivka, metoda ali razred) dokončna
  - Razlika med **static** in **final**
    - **static** se uporablja, kadar želimo preprečiti redefinicijo metod razreda
    - **final** se uporablja, kadar želimo preprečiti redefinicijo metod objekta
- Metode znotraj razredov final
  - Če je razred deklariran kot **final**, potem so avtomatično vse njegove metode **final**
  - Razred **final** ne more biti uporabljen kot osnova za dedovanje
  - Primer razreda **final**: *razred Math*

Dedovanje

- Abstraktni razred in abstraktne metode
  - v abstraktnem razredu specificiramo metodo, ki je še ne moremo sprogramirati
  - vsak podrazred mora to metodo redefinirati
    - s tem določimo obnašanje podrazredov
- Dinamično povezovanje metod
  - med izvajanjem se izbere metoda, ki pripada dejanskemu tipu objekta
- Razred Object in njegove metode
  - univerzalni nadrazred, iz katerega so izpeljani vsi ostali
  - v vseh razredih so na voljo metode, deklarirane v razredu Object
- Koncept vmesnika kot nadomestek za večkratno dedovanje
  - vsak podrazred lahko deduje samo od enega nadrazreda, implementira pa lahko več vmesnikov

Dedovanje

- Mehanizem, ki omogoča, da nek razred podeduje atribute in metode nekega drugega razreda
  - osnovni razred (ang. base class): razred, ki služi kot osnova za dedovanje
  - izpeljan razred (ang. derived class): razred, ki je bil izpeljan iz osnovnega razreda (tj. razred, ki deduje)
  - drugi izrazi:
    - nadrazred (ang. superclass) – podrazred (ang. subclass)
    - starš (ang. parent class) – otrok (ang. child class)
  - izpeljan razred je poseben primer bolj splošnega osnovnega razreda
- Primer: razreda Student in IzredniStudent
  - razred Student je osnovni razred (nadrazred)
  - razred IzredniStudent je izpeljan razred (podrazred)

Dedovanje

- Razred Student ima
  - 3 atribute: vpisna številka, priimek, ime
  - 6 metod: 3 za vpis vrednosti, 3 za branje vrednosti vsakega atributa
- Razred IzredniStudent
  - potrebuje vse atribute in metode razreda Student
  - zahteva še dodatni atribut znesekSolnine
  - koncept dedovanja omogoča, da atributov in metod razreda Student ni treba še enkrat deklarirati
  - na novo je treba deklarirati le atribut znesekSolnine ter metodi za vpis in branje zneska šolnine
- Prednosti dedovanja
  - krajši čas razvoja (uporabimo atribute in metode, ki že obstajajo)
  - manj napak (podedovane metode so že preizkušene in stestirane)
  - večja razumljivost (programer že razume delovanje podedovanih metod)

Dedovanje

- Deklaracija izpeljanega razreda
  - osnovni razred mora že obstajati
  - uporabimo rezervirano besedo extends
    - ***public class IzredniStudent extends Student***
  - deklariramo samo dodatne atribute
  - deklariramo samo dodatne metode
- Dedovanje poteka samo v eni smeri: otroci vedno podedujejo od staršev
  - nadrazred ima dostop do atributov in metod, ki so bili deklarirani v nadrazredu
  - podrazred ima dostop do atributov in metod, deklariranih v nadrazredu
- Primer dedovanja
  - http://www.roseindia.net/java/language/inheritance.shtml

Redefinicija podedovanih metod

- Redefinicija podedovanih metod (ang. overriding)
- Če neka podedovana metoda ne ustreza zahtevam podrazreda, jo lahko v podrazredu ponovno definiramo
  - ob redefiniciji moramo metodo deklarirati z enakim imenom in enakim seznamom parametrov
  - objektom nadrazreda pripada metoda, deklarirana v nadrazredu
  - objektom podrazreda pripada redefinirana metoda, deklarirana v podrazredu
- Razlika med redefinicijo metod (ang. overriding) in večkratnim definiranjem metod (ang. overloading)
  - overriding: nadrazred in podrazred imata metodo z enakim imenom in enakim seznamom parametrov
  - overloading: v istem razredu obstaja več metod z enakim imenom, a različnim seznamom parametrov

Redefinicija podedovanih metod

- Preprost primer redefinicije
  - recimo, da razred Student vsebuje metodo izpisTipa, ki izpiše ime razreda

```
public void izpisTipa()
{
System.out.println("Student");
}
```

  - razred IzredniStudent to metodo podeduje, vendar ni uporabna, ker bi morala izpisati drugačno ime razreda
  - rešitev: v razredu IzredniStudent to metodo redefiniramo
- Uporaba metode nadrazreda v podrazredu
  - kljub temu, da smo metodo, ki je deklarirana v nadrazredu, redefinirali, jo lahko še vedno pokličemo
  - uporabimo rezervirano besedo super: ***super.<ime metode>***

Redefinicija podedovanih metod

- **Primer uporabe super**
  - recimo, da razred Student vsebuje metodo izpisiVse(), ki izpiše vrednosti vseh atributov
  - v razredu IzredniStudent moramo to metodo redefinirati tako, da bo izpisala tudi atribut znesekSolnine
  - redefinirana metoda je sestavljena iz dveh delov:
    - iz klica super.izpisiVse()
    - iz stavka za izpis zneska šolnine

```
public void izpisiVse(){
super.izpisiVse();
System.out.println(znesekSolnine);
}
```

Redefinicija podedovanih metod

- Zaključek: uporaba super nam pogosto olajša pisanje redefiniranih metod
  - metoda nadrazreda, ki jo pokličemo s super, opravi tisti del postopka, ki je skupen nadrazredu in podrazredu
  - sprogramirati moramo samo preostali del postopka, ki je specifičen za podrazred
- **Primerjava med this in super v podrazredu**
  - super se nanaša na metodo nadrazreda
    - super.izpisiVse() pokliče metodo razreda Student
  - this se nanaša na metodo podrazreda
    - this.izpisiVse() pokliče metodo razreda IzredniStudent
    - referenco this običajno izpustimo

Spremenljivke instance (podatkovni člani objekta)

```
class Ship1 {
    public double x, y, speed, direction;
    public String name;
}
public class Test1 {
    Ship1 s1, s2;
    public static void main(String[] args) {
        s1 = new Ship1();
        s1.x = 0.0;
        s1.y = 0.0;
        s1.speed = 1.0;
        s1.direction = 0.0; // East
        s1.name = "Ship1";
        s2 = new Ship1();
        s2.x = 0.0;
        s2.y = 0.0;
        s2.speed = 2.0;
        s2.direction = 135.0; // Northwest
        s2.name = "Ship2";
...  // glej nadaljevanje
```

Tvorba objekta

nadaljevanje

```
...
        s1.x = s1.x + s1.speed * Math.cos(s1.direction * Math.PI / 180.0);
        s1.y = s1.y + s1.speed * Math.sin(s1.direction * Math.PI / 180.0);
        s2.x = s2.x + s2.speed * Math.cos(s2.direction * Math.PI / 180.0);
        s2.y = s2.y + s2.speed * Math.sin(s2.direction * Math.PI / 180.0);
        System.out.println(s1.name + " is at ("+ s1.x + "," + s1.y + ").");
        System.out.println(s2.name + " is at ("+ s2.x + "," + s2.y + ").");
    }
}
```

Ship1 is at (1,0).
Ship2 is at (-1.41421,1.41421).

Objekti in reference

- Potem, ko definiramo razred, lahko zlahka deklariramo spremenljivko (referenco na objekt) za ta razred
  - Ship s1, s2;
  - Point start;
  - Color blue;
- Reference objektov so v začetku "null".
- Vrednost "null" ne smemo enačiti z vrednostjo "nič".
- Primitivnih podatkovnih tipov ne moremo "kastati" na nek objekt.
  - int a=10;
  - s1=(Ship) a;
- Za eksplicitno tvorbo objekta, ki je naslovljen, potrebujemo operator "new".
  - ClassName variableName = new ClassName();

bostjan.vouk@fsc.si

Metode

```
public class Ship2 {
    public double x=0.0, y=0.0, speed=1.0, direction=0.0;
    public String name = "UnnamedShip";

    private double degreesToRadians(double degrees) {
        return(degrees * Math.PI / 180.0);
    }
    public void move() {
        double angle = degreesToRadians(direction);
        x = x + speed * Math.cos(angle);
        y = y + speed * Math.sin(angle);
    }
    public void printLocation() {
        System.out.println(name + " is at (" + x + "," + y + ").");
    }
}
```

bostjan.vouk@fsc.si

Metode - nadaljevanje

```
public class Test2 {
    public static void main(String[] args) {
        Ship2 s1 = new Ship2();
        s1.name = "Ship1";
        Ship2 s2 = new Ship2();
        s2.direction = 135.0; // Northwest
        s2.speed = 2.0;
        s2.name = "Ship2";
        s1.move();
        s2.move();
        s1.printLocation();
        s2.printLocation();
    }
}
```

Izpis

Ship1 is at (1,0).
Ship2 is at (-1.41421,1.41421).

bostjan.vouk@fsc.si

Statične metode

- Statične metode kličemo preko imena razreda:
  - ClassName.functionName(arguments);
  - Primer: razred **Math** ima *statično metodo* **cos**, ki pričakuje kot argument podatek tipa double.
    - Tako lahko pokličemo Math.cos(3.5) ne da bi imeli kakšen objekt (instanco) iz razreda Math.
- Opomba za metodo "main"
  - Sistem pokliče metodo main, ne da bi prej tvoril objekt. Zato lahko ta metoda direktno (brez predhodne tvorbe objekta) kliče le statične metode.

bostjan.vouk@tsc.si

Enkapsulacija

- Enkapsulacija pomeni, da damo skupaj tako podatke, kot kodo, ki nad temi podatki operira. Tako dobimo zaključeno celoto - objekt.
  - Del enkapsulacije je tudi skrivanje notranje izvedbe objekta.
- Navzven naj bi objekt kazal le nek vmesnik (interface - skupek metod), ki naj služi za delo z objektom.
- Enkapsulacija prinese lažje vzdrževanje ter ponovno uporabljivost kode.
- *Enkapsulacija je "skrivanje":*
  - notranjih podatkov pred uporabo od zunaj
  - podrobnosti realizacije - večja prilagodljivost
- Vidnost podatkov ali enkapsulacija
  - vidnost razredov, njihovih polj in metod

bostjan.vouk@tsc.si

Enkapsulacija

- **Enkapsulacija** preprečuje, da bi drugi programerji spreminjali kodo, ki ste jo napisali. Npr. funkcija printf() v programskem jeziku C. Ko to funkcijo uporabimo, nikoli ne vemo, kakšno je njeno telo, uporabljamo jo zgolj za izpisovanje besedila na zaslon.
- Z enkapsulacijo se pojavi nova vrsta funkcij **razreda**, pogosto imenovanih **članske funkcije**. Razredi zaokrožajo **objekte** oziroma **primerke** in določajo, kakšen dostop imamo kot programer do posameznih **članov razreda**.
- **Obseg** članov, ki jih smemo uporabljati se deli v tri skupine:
  - **Privatni (privat):** Dostop je dovoljen drugim funkcijam istega razreda.
  - **Zaščiteni (protected):** Dostop je dovoljen drugim funkcijam istega razreda in izpeljanim razredom iz originalnega razreda.
  - **Javni (public):** Dostop je dovoljen vsem objektom v obsegu, kjer je razred definiran.

bostjan.vouk@tsc.si

Polimorfizem

- **Polimorfizem** pomeni: eno ime, več oblik. Pri javi poznamo 3 vrste polimorfizma:
  - Method overloading
    - Polimorfizem v času prevajanja
  - Method overriding through inheritance
    - Polimorfizem v času izvajanja
  - Method overriding through the Java interface
    - Polimorfizem v času izvajanja
- Polimorfizem omogoča naslavljanje objektov različnih tipov v različnih časih izvajanja.
  - Polimorfično obnašanje dobimo z naslavljanjem "super-tipa", saj tak "nadtip" označuje objekte svojih podtipov.
- To poznamo tudi kot povezovanje v času izvajanja (*late binding, run-time binding*)
- *V praksi uporabljamo polimorfizem skupaj z naslavljanjem polj pri preletavanju kolekcij objektov in dostopu do polimorfičnih metod posameznih objektov*

bostjan.vouk@fsc.si

Polimorfizem: primer

```
public class PolymorphismTest {
   public static void main(String[] args) {
      Ship[] ships = new Ship[3];
      ships[0] = new Ship(0.0, 0.0, 2.0, 135.0, "Ship1");
      ships[1] = new Speedboat("Speedboat1");
      ships[2] = new Speedboat(0.0, 0.0, 2.0, 135.0, "Speedboat2",
          "blue");
      for(int i=0; i<ships.length ; i++) {
         ships[i].move();
         ships[i].printLocation();
      }
   }
}
```

```
RED Speedboat1 is at (20,0).
BLUE Speedboat2 is at (-1.41421,1.41421).
Ship1 is at (-1.41421,1.41421).
```

bostjan.vouk@fsc.si

Še en primer polimorfizma

```
public class PolymorphismTest {
   public static void main(String args[]) {
      Shape[ ] s = { new Cube(), new Circle(), new Cube() };
      for (int i = 0; i < s.length; i++) {
         System.out.println(s[i].getClass()
            + " ima povrsino " + s[i].area()
               + "  in prostornino " + s[i].volume());
      }
   }
}
```

bostjan.vouk@fsc.si

Večkratno definiranje metod

```
public class Ship4 {
   public double x=0.0, y=0.0, speed=1.0, direction=0.0;
   public String name;
   public Ship4(double x, double y,  double speed, double direction, String
    name) {
       this.x = x;
       this.y = y;
       this.speed = speed;
       this.direction = direction;
       this.name = name;
   }
   public Ship4(String name) {
       this.name = name;
   }
   private double degreesToRadians(double degrees) {
      return(degrees * Math.PI / 180.0);
   }
```

Konstruktorja z enakim imenom

(se nadaljuje)

Večkratno definiranje (nadaljevanje)

```
...
   public void move() {
      move(1);
   }
   public void move(int steps) {
      double angle = degreesToRadians(direction);
      x = x + (double)steps * speed * Math.cos(angle);
      y = y + (double)steps * speed * Math.sin(angle);
   }
   public void printLocation() {
       System.out.println(name + " is at (" + x + "," + y + ").");
   }
}
```

Metodi z enakim imenom

Večkratno prekrivanje metod (uporaba)

```
public class Test4 {
   public static void main(String[] args) {
      Ship4 s1 = new Ship4("Ship1");
      Ship4 s2 = new Ship4(0.0, 0.0, 2.0, 135.0, "Ship2");
      s1.move();
      s2.move(3);
      s1.printLocation();
      s2.printLocation();
   }
}
```

Ship1 is at (1.0,0.0).
Ship2 is at (-4.24264...,4.24264...).

"getters – setters"

```
public class Ship {
    private double x=0.0, y=0.0, speed=1.0, direction=0.0;
    private String name;

    /** Get current X location. */
    public double getX() {
        return(x);
    }
    /** Set current X location. */
     public void setX(double x) {
        this.x = x;
    }
```

- getter je metoda, ki vrne neko lastnost.
- setter je metoda, ki postavi neko lastnost na podano vrednost.

bostjan.vouk@tsc.si

Večkratna deklaracija konstruktorjev

```
public Ship (String n) {
        name = n;
    }
    public Ship (double x1; double  y1,  double s, double d;
      String n){
      x = x1;
      y = y1;
      speed = s;
      direction = d;
      name = n;
    }

}
```

Konstruktor Ship () brez parametrov ni potrebno posebej definirati

bostjan.vouk@tsc.si

Dedovanje

```
public class Speedboat extends Ship {
    private String color = "red";
    public Speedboat(String name) {
        super(name);
        setSpeed(20);
    }
    public Speedboat(double x, double y, double speed, double direction,
          String name, String color) {
        super(x, y, speed, direction, name);
        setColor(color);
    }
    public void printLocation() {
        System.out.print(getColor().toUpperCase() + " ");
        super.printLocation();
    }
}
```

bostjan.vouk@tsc.si

Večkratno dedovanje

- V Javi ni dovoljeno!
- dovoljeno v C++, Smalltalk, …

Uporaba izpeljanega in dedovanega razreda

```
public class SpeedboatTest {
    public static void main(String[] args) {
        Speedboat s1 = new Speedboat("Speedboat1");
        Speedboat s2 = new Speedboat(0.0, 0.0, 2.0, 135.0,
            "Speedboat2", "blue");
        Ship s3 = new Ship (0.0, 0.0, 2.0, 135.0, "Ship1");
        s1.move();
        s2.move();
        s3.move();
        s1.printLocation();
        s2.printLocation();
        s3.printLocation();
    }
}
```

```
RED Speedboat1 is at (20,0).
BLUE Speedboat2 is at (-1.41421,1.41421).
Ship1 is at (-1.41421,1.41421).
```


Kaj kaže prejšnji primer?

- Kako definiramo podrazrede (izpeljane razrede)
- Kako uporabljamo dedovane metode
- Uporaba super(…) za podedovane konstruktorje (le, če konstruktor brez argumentov ni uporaben)
- Uporaba super.nekaMetoda(…) za podedovane metode (le, če je konflikt v imenu)

Dedovani konstruktorji in super

- **Ko tvorimo instanco (objekt) nekega podrazreda, bo sistem avtomatično najprej poklical konstruktor nadrazreda.**
- Privzeto bo poklical nadrazredov konstruktor brez argumentov.
- Če hočemo poklicati kakšen drug starševski konstruktor, ga pokličemo s **super(*args)***
- Če v konstruktorju podrazreda kličemo **super(…)** , mora to biti prvi stavek v konstruktorju.

Redeklarirane metode in super.metoda()

- **Če razred definira metodo z enakim imenom, tipom povratka in argumentov kot jo ima nadrazred, tedaj ta metoda prekrije metodo nadrazreda.**
- Prekrijemo (redeklaracija) lahko le metode, ki niso statične
- **Če imamo lokalno definirano metodo in podedovano metodo z istim imenom in argumenti, lahko pokličemo podedovano metodo tako:**
  - **super.imeMetode(...)**
- Zaporedna uporaba predpon super ni dovoljena
  - **super.super.imemetode**

Uporaba this in super

```
class X {
  public static void main (String [] args){
    (new Y()).izpis();
    // Na daljši način => Y pr= new Y();  pr.izpis();
  }
  public void izpisiTest(){
    System.out.println("Vsebina iz X-a.");
  }
}
class Y extends X {
  public void izpisiTest(){
    System.out.println("Vsebina iz Y-a.");
  }
  public void izpis(){
    this.izpisiTest(); //lahko samo izpisiTest();
    super.izpisiTest();
  }
}
```

Izpis

Vsebina iz Y-a.

Vsebina iz X-a.

Abstraktni razredi

- Za abstraktne razrede ne moremo tvoriti instanc (ne moremo klicati "new").
- Abstraktni razredi dovolijo deklaracijo razredov, ki definirajo le del implementacije, podrobnosti pa morajo podati podrazredi
- **Razred je abstrakten, če vsaj eni metodi razreda manjka implementacija**
- Abstraktna metoda nima implementacije (pri C++ to imenujemo čista virtualna funkcija)
- Vsak razred, ki ima vsaj eno abstraktno metodo, moramo deklarirati kot abstrakten
- Če podrazred prekrije vse abstraktne metode nadrazreda, lahko tvorimo instance tega podrazreda
- Abstrakten razred ima lahko spremenljivke in polno implementirane metode– Vsak podrazred lahko prekrije podedovane metode
- **Abstraktne razrede lahko naslavljamo, čeprav ne moremo tvoriti instanc iz njih**

Abstraktni razredi

- Abstrakcija vnaša večjo jasnost v načrtovanje razredov. Hierarhija razredov lahko temelji na osnovnem abstraktnem razredu, ki vsebuje le najbolj bistvene značilnosti tipa.
- Konkretni podrazredi imajo lahko instance. Imajo lahko tudi specifične atribute, po katerih se podrazredi med seboj razlikujejo.
- Abstraktni razredi morajo imeti vsaj eno abstraktno metodo.
- Abstraktna metoda nima telesa: nima { }
- Koncept abstraktnega razreda si bomo ogledali na primeru na naslednji prosojnici.

Primer: splošen razred in podrazredi

```
public class Shape {
  double getArea ( )
  { return 0.0;}
}
```

Metoda getArea ( ) je redeklarirana

```
public class Rectangle extends Shape {
  double ht = 0.0;
  double wd = 0.0;

  public double getArea ( )
  { return (ht*wd);}

  public void setHeight(double ht)
  { this.ht = ht; }

  public void setWidth (double wd)
  { this.wd = wd; }
}
```

```
public class Circle extends Shape {
  double r =0.0;

  public double getArea ( )
  { return (3.14 * r*r); }

  public void setRadius (double r)
  { this.r = r; }
}
```

Primer z abstraktnim razredom

```
abstract class Shape {
  abstract int getArea();
}
```

- Izpeljani razredi **morajo** definirati metodo getArea( )

```
public class Rectangle extends Shape {
  double ht = 0.0;
  double wd = 0.0;

  public double getArea ( )
  { return (ht*wd);

  public void setHeight(double ht)
  { this.ht = ht; }

  public void setWidth (double wd)
  { this.wd = wd; }
}
```

```
public class Circle extends Shape {
  double r =0.0;

  public double getArea ( )
  { return (3.14 * r*r); }

  public void setRadius (double r)
  { this.r = r; }
}
```

bostjan.vouk@tsc.si

Primer

- V programskem jeziku Java definirajte razrede Kvadrat, Krog in Lik. Razreda Kvadrat in Krog naj bosta izpeljana iz osnovnega razreda Lik. Poleg konstruktorja naj razreda definirata tudi metodo obseg, ki izračuna in vrne obseg lika.
- Napišite program, ki ustvari nekaj predmetov tipa Krog in Kvadrat in jih shrani v polje, nato pa izpiše vsoto obsegov vseh likov v polju.
- Pri reševanju naloge uporabite princip polimorfizma.

bostjan.vouk@tsc.si

Rešitev

```
class Liki {
    public static void main(String[] args){
        Lik [] poljeLikov = new Lik [5];
        double vsotaObsegov = 0.0;
        poljeLikov[0] =  new Kvadrat(3);
        poljeLikov[1] =  new Krog(4);
        poljeLikov[2] =  new Kvadrat(5);
        poljeLikov[3] =  new Krog(2);
        poljeLikov[4] =  new Kvadrat(5);

        for(int i=0; i< poljeLikov.length;i++) {
            Lik lik = poljeLikov[i];
            vsotaObsegov = vsotaObsegov + lik.getObseg();
        }
        System.out.println ("Vsota obsegov = " + vsotaObsegov);
        System.out.printf("Vsota  obsegov = %7.2f \n", vsotaObsegov);
    }
}
```

bostjan.vouk@tsc.si

Rešitev nadaljevanje

```
//*******************************************************
abstract class Lik {
   double x,y;                  // lokacija lika
   abstract double getObseg();
}
//*******************************************************
class Kvadrat extends Lik {
   double a;                    // stranica kvadrata
   public Kvadrat ( double stranica){
      a = stranica;
   }
   public double getObseg () {
      return (4*a);
   }
}
```


Rešitev nadaljevanje

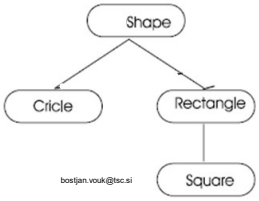
```
//*******************************************************
class Krog  extends Lik {
   double r;                    // polmer kroga
   public Krog ( double polmer){
      r = polmer;
   }
   public double getObseg () {
      return (2.0*r*Math.PI);
   }
}
```


Vmesniki (interfaces)

- Nadgradnja abstraktnih razredov
  - Namenjeni so za komuniciranje med razredi
- V vmesnikih so zapisane konstante in le ‘najavljene’ metode, ki jih kasneje napišemo
- Vmesnik povežemo v nov razred z ukazom *implements*
  - Za vse ‘najavljene’ metode je potrebno napisati kodo (lahko tudi prazno)

Vmesniki (interfaces)

- Vmesniki definirajo javanski tip, ki vsebuje le konstante in abstraktne metode
- Vmesnik **ne implementira nobene metode**, pač pa vsili to vsakemu razredu, ki uporablja tak vmesnik
- Razred, ki uporabi (*implements*) vmesnik, mora podati definicije **vseh metod**, ali pa mora tudi sam biti deklariran kot abstrakten



Vmesniki (interfaces)

- Večkratno dedovanje
  - podrazred podeduje atribute in metode več kot enega nadrazreda
  - C++ omogoča večkratno dedovanje, Java pa ne
  - problemi pri večkratnem dedovanju
    - kaj narediti, če imajo atributi in metode v nadrazredih enaka imena
    - konstruktor katerega nadrazreda naj se kliče pri klicu super()
- Rešitev, ki jo ponuja Java: koncept vmesnika
- Vmesnik je zelo podoben razredu, s to razliko, da
  - morajo biti vse metode abstraktne
  - vsi atributi (če jih ima), morajo biti static final
- Z vmesnikom predpišemo metode, ki jih mora implementirati podrazred (določimo obnašanje podrazreda)

Vmesniki (interfaces)

- Splošno pravilo
  - podrazred lahko deduje samo od enega nadrazreda
  - implementira lahko več vmesnikov
- Primer deklaracije
  - **public class Podrazred extends Nadrazred implements Vmesnik1, Vmesnik2**
- S stališča dedovanja Podrazred
  - podeduje atribute in metode razreda Nadrazred
  - dodatno lahko deklarira nove atribute in metode
  - redefinira lahko podedovane metode

Vmesniki (interfaces)

- S stališča implementacije vmesnika
  - v podrazredu moramo deklarirati vse metode, specificirane v vmesnikih Vmesnik1 in Vmesnik2
  - podrazred lahko uporablja statične spremenljivke iz obeh vmesnikov
- Primerjava: abstraktni razred – vmesnik
  - **podobnost:** ne moremo generirati objektov, ki bi pripadali abstraktnemu razredu ali vmesniku
  - **razlika:** v abstraktnem razredu so lahko samo nekatere metode abstraktne, v vmesniku pa morajo biti abstraktne vse metode
  - **razlika:** atributi abstraktnega razreda so vezani na posamezne objekte, atributi vmesnika pa so statični (vezani na razred) in nespremenljivi

Vmesniki (interfaces)

- *Kdaj uporabimo abstraktni razred?*
  - kadar lahko že na najvišjem nivoju sprogramiramo metode, ki so skupne različnim podrazredom
  - Primer: igre s kartami
    - metoda mešaj() je enaka za vse podrazrede, zato jo lahko sprogramiramo v nadrazredu
    - metoda deli() je za vsak podrazred drugačna, zato je v nadrazredu abstraktna
- *Kdaj uporabimo vmesnik?*
  - ko vemo, katere operacije mora izvajati podrazred, vendar dovolimo, da jih vsak podrazred sprogramira po svoje
  - Primer: glasbeni instrumenti
    - v vmesniku predpišemo metodo zaigrajTon(), ki jo lahko vsak instrument realizira po svoje

Vmesniki

- Vse metode v vmesniku so implicitno abstraktne in zato *besedice abstract* tu ne potrebujemo
- Podatkovna polja v vmesniku so implicitno konstante (*static final*)
- Vsa podatkovna polja in metode vmesnika so implicitno **public**

-
```
public interface Interface1 {
    DataType CONSTANT1 = value1;
    DataType CONSTANT2 = value2;
    ReturnType1 method1(ArgType1 arg);
    ReturnType2 method2(ArgType2 arg);
}
```

Primer vmesnika in njegove uporabe

```
interface Shape {
   public double area();
   public double volume();
}
```

Morali smo definirati obe metodi, napovedani v vmesniku

```
public class Point implements Shape {
        static int x, y;
        public Point() {
           x = 0;  y = 0;
        }
        public double area() {
                return 0;
        }
        public double volume() {
                return 0;
        }
        public static void print() {
          System.out.println("point: " + x + "," + y);
        }
        public static void main(String args[]) {
                Point p = new Point();
                p.print();
        }
}
```


Uporaba več vmesnikov

- Vmesnike lahko izpeljemo (extends) tudi iz drugih vmesnikov (dedovanje), kar pripelje do pojmov podvmesniki in nadvmesniki.
- Z razliko od razredov lahko vmesnike izpeljemo (iz več kot enega vmesnika)
- Uporaba več vmesnikov v razredih je nekakšen ekvivalent večkratnega dedovanja
- Primer:
  - **public class Circle extends Shape implements Drawable, Printable {**
    ...
    }

Razlike med vmesniki in razredi

- Vmesnik si lahko predstavljamo kot "seznam" metod in kostant, ki morajo biti v enem ali več razredih.
- Dovoljeno je torej zapisati vanj npr
  - public void izpisi(String s); lahko pa dodamo tudi konstanto v obliki npr public static int a = 5; in
- Drži tudi, da je osnovna uporaba vmesnikov namenjena temu, da določimo nabor metod, katere mora nek razred implementirat.
- Če želimo ustvarjati primerke tega razreda (torej razred ki implementira vmesnik A, mora imeti implementirane vse metode iz tega razreda, sicer moramo razred označiti kot abstract).

Razlike med vmesniki in razredi

- V javi je dovoljeno n-kratno dedovanje v globino a na istem nivoju samo enkratno
  - ne moremo napisat:
    - public class A extends B, C
- Pri vmesnikih je mogoče oboje, tako da nekateri profesorji pravijo, da vmesniki v javo prinesejo večkratno dedovanje na istem nivoju.
- Pri vmesniku ne moremo narediti primerka, razredu pa lahko.

Primer uporabe vmesnika

```
public interface Vmesnik{
   public void izpis();
}
public class Test implements Vmesnik{
   Test(){}
   public void izpis(){
      System.out.println("Burek");
   }
   public void nedostopnaMetoda(){
      System.out.println("Ni ni");
   }
   public static void main(String[] args)
   {
      Vmesnik objekt = new Test();
      objekt.izpis();
      //objekt.nedostopnaMetoda();
   }
}
```

V tem primeru lahko dostopamo na objektu "objekt" samo do metod, ki so definirane v vmesniku "Vmesnik", ostale pa niso dostopne.

Npr.: če bi želel poklicati metodo "nedostopnaMetoda()". Če napišemo

Test objekt = new Test() je omenjena metoda jasno dostopna.

Ponavljanje

Programski jezik Java je

a) postopkoven
b) postopkoven in funkcijski
c) postopkoven in objektno usmerjen
d) objektno usmerjen
e) objektno usmerjen in funkcijski

Ponavljanje

Po vrnitvi iz klicane metode v programskem jeziku Java se kontrola izvajanja prenese

a) na naslednji stavek za klicem te metode,
b) na naslednji stavek za klicem te metode ali v jedro stavka try,
c) na naslednji stavek za klicem te metode ali v jedro stavka catch,
d) na naslednji stavek za klicem te metode ali v jedro stavka try ali catch,
e) na naslednji stavek izven jedra stavka try ali catch.

bostjan.vouk@fsc.si

Ponavljanje

- Rekurzivne funkcije so napisane tako, da pri svojem izvajanju ponovno kličejo same sebe. Katera od naslednjih trditev drži?
- Izberite en odgovor.

a) vsak rekurziven program lahko spremenimo v iterativnega
b) rekurzija je časovno vedno učinkovitejša od iteracije
c) rekurzija je prostorsko vedno učinkovitejša od iteracije
d) globina rekurzije je omejena z velikostjo kopice

bostjan.vouk@fsc.si

Ponavljanje

Če govorimo o enkapsulaciji, dedovanju in rekurziji, potem med prvine objektno orientiranega programiranja uvrščamo

a) samo enkapsulacijo
b) enkapsulacijo in dedovanje
c) samo dedovanje
d) enkapsulacijo in rekurzijo
e) rekurzijo

bostjan.vouk@fsc.si

Ponavljanje

Programski jezik Java ne podpira večkratnega dedovanja, zato mora programer

a) uporabiti tovarne objektov,
b) uporabiti vmesnike,
c) uporabiti nekaj vnaprej znanih trikov,
d) uporabiti proceduralni pristop,
e) uporabiti model problema, ki ne zahteva večkratnega dedovanja.

bostjan.vouk@fsc.si

Ponavljanje

- Denimo, da imamo v Javi podana dva razreda (angl. class) z imenom Base1 in Base2 ter dva vmesnika (angl. interface) z imenom Line1 in Line2. Kateri od naslednjih vzorcev predstavljajo pravilno deklaracijo novega razreda, ki pravilno deduje oziroma uporablja kombinacijo navedenih razredov in vmesnikov?

1. class Dete extends Base1, Base2 overrides Line1 {...}
2. class Dete extends Base2 implements Line1, Line2 {...}
3. class Dete extends Base1, Base2, Line1, Line {...}
4. class Dete implements Line1, Line2 {...}
5. class Dete extends Base1, Base2 {...}
6. class Dete implements Base1, Base2 {...}

- Izberite en odgovor.
  - a. 2 in 4
  - b. samo 3
  - c. 4 in 5
  - d. vsi razen 1

bostjan.vouk@fsc.si

Ponavljanje

- V Javi je podana naslednja definicija razreda:
- **public class NamedPoint extends Point {**
  private String myName;
  public NamedPoint(int d1, int d2, String name) {
  super(d1, d2);
  myName = name;
  }
  }
- Stavek **super(d1, d2)** pomeni (*izberite en odgovor*)

a) klic konstruktorja nadrejenega razreda Point s parametroma d1 in d2
b) klic metode super, ki je dosegljiva v uvoženi knjižnici, s parametroma d1 in d2
c) spremembo dosega spremenljivk d1 in d2 iz lokalnih v globalne
d) klic ne deluje, prevajalnik javi sintaktično napako; deloval bi le v primeru, če bi razred imel metodo z imenom super

bostjan.vouk@fsc.si

Ponavljanje

- V Programskem jeziku Java je podan spodnji program, ki pa ne deluje zaradi napake v programu.
- public class Izpis {
  public static void main(String[] args) {
  double stevilo;
  stevilo = nakljucno();
  System.out.println("Nakljucno stevilo od 1 do 100 je: " + stevilo);
  }
  private final double nakljucno() {
  return (Math.random()*100)+1;
  }
  }
- **Kje je napaka v programu?**
- *Izberite en odgovor.*

a) metoda nakljucno() ni dostopna, ker ni klicana na primerku razreda Izpis
b) težava je pri pretvorbi tipa double v tip string v stavku System.out.println
c) metoda nakljucno() ni dostopna ker je označena kot zasebna (private)
d) težava je prisotna zaradi pretvorbe tipov pri seštevanju vrednosti različnih tipov: rezultata funkcije Math.random(), ki je tipa double, in celoštevilske vrednosti 1

bostjan.vouk@fsc.si