1. Programiranje in programski jeziki

Kazalo:

- Kaj je računalniški program
- Štiri generacije programskih jezikov
  - značilnosti visokonivojskih programskih jezikov 3. generacije
- Programski jezik Java
  - razvojno okolje, ki ga bomo uporabljali pri svojem delu
- Primer preprostega programa
  - vsak program napišemo kot razred
  - metoda `main()`
  - pravila za pisanje imen, lepopisna pravila, komentarji
  - primer služi kot ogrodje kateregakoli programa
  - postopek priprave in izvajanja programa

Kaj je računalniški program

- zaporedje ukazov, ki naj jih izvede računalnik
- ukazi morajo biti zapisani v nekem programskem jeziku

Štiri generacije programskih jezikov

- strojni jezik – jezik 1. generacije
  - vsi ukazi in podatki morajo biti zapisani z ustrezno kombinacijo ničel in enic
  - neprimeren za človeka
- zbirni jezik – jezik 2. generacije
  - vpeljava mnemonikov (imen) za ukaze in podatke
  - še vedno zahteva programiranje na nizkem nivoju
- visokonivojski programski jeziki 3. generacije
  - bližji človeku: uporaba posameznih besed iz naravnega jezika (npr. `if… else`; `do… while; print ipd`.)
  - postopkovni: opisati je treba postopek, kako pridemo do rešitve
  - algoritem: z nizom navodil podan postopek za reševanje problema
  - za opis postopka zadostujejo 3 osnovni programski konstrukti:
    - zaporedje ali sekvenca
    - izbira ali selekcija
    - ponavljanje ali iteracija
  - prevajalnik prevede izvorni program v obliko, ki se lahko izvede na računalniku
  - primeri visokonivojskih programskih jezikov: Java, Pascal, Oberon, C, C++, Fortan, Cobol, Basic,…
  - sintaksa: pravila za pisanje programov (slovnica)
  - sintaktično pravilnost preverja prevajalnik
  - logične napake odkrijemo šele med izvajanjem
- visokonivojski programski jezik 4. generacije
  - nepostopkovni: opisati je treba KAJ želimo dobiti kot rezultat, ne pa KAKO pridemo do njega
  - uporabni predvsem za reševanje specifičnih problemov
  - primeri: SQL, Prolog, Mantis

- namen predmeta je spoznati osnove programiranja v eni izmed visokonivojskih programskih jezikov 3. generacije

Programski jezik Java

- objektno usmerjen programski jezik 3. generacije
  - razvilo ga je podjetje Sun Microsystems
    - Java™ Platform Standard edition (J2SE™), verzija 5.0
    - http:// java.sun.com/
- splošno namenski programski jezik, uporaben za vse vrste aplikacij
- glavne prednosti Jave:
  - objektna usmerjenost
  - neodvisnost od platforme
  - primernost za razvoj spletnih aplikacij
  - varnost
  - relativna enostavnost (v primerjavi s C++)
- razvojno okolje
  - potrebujemo razvojni komplet (JDK – Java Development Kit) in preprost urejevalnik besedil, npr. Notepad (Beležnica)
  - po namestitvi JDK znotraj izbrane mape dobimo več podmap
    - `bin` orodja JDK (npr. prevajalnik)
    - `demo` demonstracijski programi
    - `include` datoteke za povezavo z izvorno strojno kodo
    - `jre` izvajalno okolje
    - `lib` knjižnice
  - nastavitev sistemske poti do podmape `bin` (sistemska spremenljivka PATH)
  - nastavitev sistemske poti do podmap s prevedenimi programi (sistemska spremenljivka CLASSPATH)
- kompleksna razvojna orodja
  - NetBeans, Eclipse, JBuilder, JDeveloper, JCreator itd.

Primer preprostega programa

```
public class Prvi
{
      public static void main (String [] args)
      {
            System.out.println ("Nas prvi program.");
      }
}
```

Program izpiše niz med obema dvojnima narekovajema.

- `public class Prvi`
  - deklarira razred z imenom `Prvi`
  - `public` je določilo za način dostopa do razreda (ang. access modifier)
    - dostop je možen iz vseh razredov
- `class` je rezervirana beseda, ki označuje začetek deklaracije razreda
- ime razreda določi programer v skladu z naslednjimi pravili:
  - pričeti se mora s črko, podčrtajem ali znakom za dolar
  - vsebuje lahko črke, številke, podčrtaje in znake za dolar
  - ne sme biti enako nobeni izmed rezerviranih besed
  - ne sme biti `true`, `false` ali `null`

- zgoraj našteta pravila veljajo za vsa imena v Javi
- Java razlikuje med velikimi in malimi črkami
    - `Prvi` in `prvi` sta različni imeni
    - priporočila za imena razredov:
        - ime razreda naj se začne z veliko začetnico (npr. `Avtomobil`)
        - če je ime sestavljeno iz več besed, se vsaka beseda začne z veliko začetnico (npr. `RedniStudent`)
        - ne uporabljamo podčrtajev (ime `Redni_Student` je nezaželeno)
- `public static void main (String [] args)`
    - glava metode `main`
        - izvajanje vsakega programa se začne z metodo `main`, ki je obvezna
        - poleg metode `main` lahko program vsebuje še druge metode
    - `public` in `static` sta določili za način dostopa
        - `static` pomeni, da metoda lahko obstaja samostojno, ne da bi bila vezana na posamezne objekte nekega razreda (metoda je enaka (nespremenljiva))
        - statične metode so v pomnilniku shranjene samo enkrat
        - vsi objekti uporabljajo isti »izvod« metode
    - `void` je rezervirana beseda, ki pove, da metoda `main` ne vrne nobene vrednosti
    - `(String [] args)` je deklaracija argumentov (parametrov)
        - argumenti so podatki, ki jih metoda potrebuje za svoje delo
        - navedemo ji ob klicu metode, npr. `sestej (5,10);` ali `sestej (a,b);`
        - tudi če metoda `main` ne potrebuje argumentov morajo biti deklarirani
        - `String` pove, kakšnega tipa so argumenti: nizi znakov
        - `args` je skupno ime za vse argumente
        - oglata oklepaja označujeta, da argumenti tvorijo tabelo nizov
- `System.out.println ("Nas prvi program.");`
    - stavek, ki izpiše niz `"Nas prvi program."`
    - na koncu vsakega stavka je podpičje
    - nižje zaporedje znakov med dvema dvojnima narekovajema
    - `println` je ime metode za izpis
    - metoda `println` pripada objektu `out`, ki spada v razred `System`
    - tipična notacija za klicanje metod: <razred>.<objekt>.<metoda>
- zaviti oklepaji `{}`
    - označujejo začetek in konec posameznih blokov
    - nastopajo vedno v parih
        - prvi par: začetek in konec deklaracije razreda `Prvi`
        - drugi par: začetek in konec deklaracije metode `main`
- lepopisna pravila
    - v Javi ni posebnih pravil glede presledkov, zamikov, skokov v novo vrstico ipd.
    - zaradi boljše čitljivosti upoštevamo naslednja pravila:
        - začetni in končni oklepaj sta eden pod drugim
        - vsak blok zamaknemo desno za nekaj presledkov
        - vse stavke, ki pripadajo istemu bloku, pišemo enega pod drugim
- komentarji
    - zaradi boljše dokumentacije programa
    - tri vrste komentarjev:
        - vrstični komentar: preostanek vrstice, ki sledi // je komentar

        - bločni komentar: tekst med /* in */
        - javadoc komentar: tekst med /** in */
- zaključna ugotovitev
    - čeprav še ne razumemo v celoti pomena vseh sestavnih delov, smo spoznali lupino, ki predstavlja ogrodje vsakega programa

```
public class <ime razreda>
{
      public static void main (String [] args)
      {
            //stavki ki se izvedejo
      }
}
```

- postopek poprave in izvajanja programa
    - program napišemo z urejevalnikom besedil, npr. Notepad
        - izvorni program moramo shraniti kot navadno besedilo
        - datoteka z izvornim programom mora imeti podaljšek `java` npr. `Prvi.java`
    - program prevedemo z ukazom `javac Prvi.java`
        - obstajati mora pot do prevajalnika; npr. `path=c:\java\bin`
        - če se datoteka z izvornim programom nahaja v drugi mapi (poddirektoriju), moramo pred imenom datoteke navesti še pot npr. `javac c:\mojiProgrami\Prvi.java`
        - program se prevede v vmesno kodo (ang. bytecode), ki je sestavljena iz množice virtualnih, od platforme neodvisnih ukazov
        - vmesna koda je shranjena na datoteki s podaljškom `class`, npr. `Prvi.class`
    - izvajanje programa sprožimo z ukazom `java Prvi`
        - virtualni ukazi se izvedejo na t.i. virtualnem računalniku (JVM- Java Virtual Machine)
        - uporaba vmesne kode omogoča platformno neodvisnost, vendar upočasni izvajanje, ker je treba med izvajanjem interpretirati virtualne ukaze
    - napake
        - sintaktične napake odkrije prevajalnik
            - vmesna koda se generira šele, ko odpravimo vse sintaktične napake
        - logične napake se pojavijo med izvajanjem
            - kljub temu, da je program sintaktično pravilen, ne dela tako, kot smo si zamislili
        - postopnost pri razvoju omogoča sprotno odpravo napak

2. Osnovni podatkovni tipi

Kazalo:

- Pojem podatkovnega tipa
  - zakaj je pomemben
- Osem osnovnih podatkovnih tipov
  - za tiste vrste podatkov, ki se najpogosteje uporabljajo
  - glavne značilnosti vsakega tipa
  - operacije, ki jih lahko izvajamo
- Deklaracije spremenljivk
  - podatki, ki jih obdelujemo, so shranjeni v spremenljivkah
  - uporabo vsake spremenljivke moramo prej napovedati: deklaracija

Pojem podatkovnega tipa

- vsak podatek pripada točno določenemu tipu
- tip podatka določa:
  - zalogo vrednosti, ki jih lahko podatek zavzame
  - operacije, ki jih lahko izvajamo med podatkom

Osem osnovnih podatkovnih tipov

- v Javi imamo 8 osnovnih podatkovnih tipov
  - numerični tipi (za predstavitev števil):
    - za cela števila: `byte, short, long, int`
      - vsako število je predstavljeno z ustrezno kombinacijo enic in ničel
      - če imamo na voljo n bitov, lahko zapišemo 2^n različnih števil
      - glavni problem: množica celih števil je omejena (čeprav je v matematiki neskončna)

tip	bitov	minimalna vrednost	maksimalna vrednost
`byte`	8	–128	127
`short`	16	–32768	32767
`int`	32	–2 147 483 648	2 147 483 647
`long`	64	–9 223 372 036 854 775 808	9 223 372 036 854 775 807

      - najpogostejše operacije

operator	opis	primer
+	seštevanje	32+3, rezultat je 35
-	odštevanje	32-3, rezultat je 29
*	množenje	32*3, rezultat je 96
/	deljenje	32/3, rezultat je 10
%	ostanek	32%3, rezultat je 2

      - pozor: rezultat celoštevilskih operacij je vedno tipa `int` ali `long`, četudi so operandi (podatki) samo tipa `byte` ali `short`
      - posebne aritmetične operacije: bitni operatorji

operator	opis	primer za števili 27_{10} in 2_{10}
&	bitni IN	00011011 & 00000010 = 00000010 = 2_{10}

|	bitni ALI	00011011 | 00000010 = 00011011 = 27_{10}
^	ekskluzivni ALI	00011011 ^ 00000010 = 00011001 = 25_{10}
~	bitna negacija	~00011011 = 11100100 = -28_{10}
<<	bitni pomik v levo	00011011<<00000010 = 01101100 = 108
>>	pomik v desno s predznakom	00011011>>00000010 = 00000110 = 6_{10} 11100100>>00000010 = 11111001 = -7_{10}
>>>	pomik v desno z vstavljanjem ničel	00011011>>>00000010 = 00000110 = 6_{10} 11100100>>>00000010 = 1073741817_{10}

- za realna števila: `float, double`
  - predstavitev s pomično vejico (ang. floating point)
  - zapis vsakega števila je sestavljen iz treh delov
    - predznak: prvi bit (0 – pozitivno število; 1 – negativno število)
    - mantisa: za zapis decimalnih mest
    - karakteristika: za zapis eksponenta
  - primer: $135,698435 = 0,135698 \times 10^3$
  - glavni problem: zaradi omenjenega števila bitov za predstavitev mantise ni moč predstaviti vseh decimalk (zaokrožitvena napaka)

tip	bitov	minimalna vrednost	maksimalna vrednost	število točnih decimalnih mest
`float`	32	$1,4 \times 10^{-45}$	$3,4 \times 10^{38}$	6 do 7
`double`	64	$4,9 \times 10^{-324}$	$1,8 \times 10^{308}$	14 do 15

  - računanje z realnimi števili
    - na voljo so operatorji +, -, * in /
  - pretvorbe med numeričnimi tipi
    - pri seštevanju (množenju, deljenju, odštevanju) dveh števil istega tipa je tip rezultata enak tipu obeh podatkov
    - pri seštevanju (množenju, deljenju, odštevanju) dveh števil različnega tipa je rezultat enak višjemu od obeh tipov; pretvorba se izvrši avtomatsko
    - primer: če seštejemo tipa `int` in `float`, je rezultat tipa `float`
    - pretvorba iz višjega tipa v nižji se lahko izvrši na zahtevo programerja (ang. type casting)
- za logične vrednosti: `boolean`
  - tip `boolean`
    - zavzame lahko samo 2 vrednosti
      - `true` (resnično, da, 1)
      - `false` (neresnično, ne, 0)
  - nad podatki tipa `boolean` lahko izvajamo logične operacije
    - konjunkcija (IN, AND): `&&`
      - rezultat je true, če imata oba operanda vrednost `true`
    - disjunkcija (ALI, OR): `||`
      - rezultat je true, če ima vsaj eden od operandov vrednost `true`
    - negacija (NOT): `!`

- ❖ če ima operand vrednost `true`, postane `false` in obratno
- kratkostično ovrednotenje konjunkcije in disjunkcije
  - recimo, da sta `p` in `q` dva pogoja, ki sta lahko izpolnjena ali ne
  - `p && q`: če ima `p` vrednost `false`, potem vrednosti `q` sploh ni treba izračunati, saj je rezultat v vsakem primeru `false`
  - `p || q`: če ima `p` vrednost `true`, potem vrednosti `q` ni treba izračunati, saj je rezultat v vsakem primeru `true`
  - prednosti kratkostičnega ovrednotenja
    - ❖ hitrejše izvajanje
    - ❖ enostavnejše programiranje: `if(x!=0 && y/x>10)`
- polno ovrednotenje konjunkcije in disjunkcije
  - namesto operatorjev `&&` in `||` uporabimo `&` in `|`
  - operatorja `&` in `|` imata dva pomena
    - ❖ pri celih številih pomenita bitni IN oziroma ALI
    - ❖ pri logičnih vrednostih pomenita polno ovrednoten logični IN oziroma ALI
- rezultat, ki ga dobimo pri primerjanju dveh vrednosti je `boolean`
- primerjalni operatorji so prikazani v spodnji tabeli:

operator	opis	primer za true	primer za false
<	manjše kot	5<7	7<5
>	večje kot	7>5	5>7
<=	manjše ali enako	5<=5	7<=5
>=	večje ali enako	7>=3	3>=7
==	enako	5==5	5==6
!=	ni enako	8!=6	5!=5

- za znake: `char`
  - tip `char`
    - služi za predstavitev posameznih znakov
    - znak zapišemo med dvojnima narekovajema, npr. `'a', 'A', '$', ':', '?'`
    - neizpisljive znake zapišemo s pomočjo ubežnih sekvenc (ang. escape sequence)

ubežna sekvenca	opis	ubežna sekvenca	opis
\t	tab	\r	carriage return
\n	new line	\"	dvojni narekovaj
\f	form feed	\'	enojni narekovaj

  - interna predstavitev znakov
    - Java uporablja kodno shemo Unicode
    - v pomnilniku je vsak znak predstavljen s 16-bitno kodo
    - kodo lahko interpretiramo tudi kot celo število, sestavljeno iz 16 ničel in enic
    - primer: koda znaka A je 0000 0000 0100 0001 ali 0041_{16}
    - vsak znak lahko tako zapišemo na 2 načina:
      - med dvema enojnima narekovajema: `'A'`
      - z njegovo šestnajstiško kodo: `'\u0041'`
      - `\u` je ubežna sekvenca, ki »napove« numerično kodo

        - znake lahko primerjamo med sabo s primerjalnimi operatorji
    - pozor: nizi znakov so tipa `String`, posamezni znaki so tipa `char`

Deklaracije spremenljivk

- spremenljivka: lokacija (celica) v pomnilniku, v kateri shranimo neko vrednost
- vse spremenljivke morajo biti deklarirane
- ob deklaraciji navedemo:
    - tip spremenljivke
    - ime spremenljivke
    - začetno vrednost (neobvezno)
    - podpičje
- primer:

```
int vsota;                  //brez inicializacije
double produkt = 234.5;     //z inicializacijo
```

*inicializacija=začetna vrednost (spremenljivke)

3. Stavki programskega jezika Java

Kazalo:

- Prireditveni stavek
  - v spremenljivko shranimo neko vrednost
  - računanje izrazov in prioriteta operatorjev
- Izbirni stavki (`if` in `switch`, pogojni operator)
  - izbiramo med več različnimi možnostmi za nadaljevanje programa
- Ponavljalni stavki (`do...while`, `while`, `for`)
  - omogočajo, da se del programa večkrat ponovi
- Stavka `break` in `continue`
- Krajši zapis prireditvenih stavkov
- Program za izpis možnih potez lovca na šahovnici

Prireditveni stavek

- splošna oblika: `<ime spremenljivke> = <izraz>;`
- izraz (ang. expression) je računska formula ali predpis, na podlagi katerega izračunamo neko vrednost. Ta vrednost se shrani v spremenljivko, ki je navedena na levi strani
- skladnost tipov: tip izraza se mora ujemati s tipom spremenljivke
- izrazi z numeričnimi operandi različnih tipov
  - avtomatična pretvorba v tip najvišjega operanda
  - pretvorbo v nižji tip mora eksplicitno zahtevati programer (ang. type casting)
- izrazi z numeričnimi operandi različnih tipov

```
int a=10;
short b=5;
int rez1;
float rez2;
short rez3;
byte rez4, rez5;

rez1 = a*b;                         // 50 tipa int
rez2 = a*b;                         // 50.0 tipa float
rez3 = (short) (a*b);               // 50 tipa short
rez4 = (byte) (a*b);                // 50 tipa byte

//napaka zaradi prekoračitve obsega
rez5 = (byte) (rez4*b);             //-6 tipa byte
```

- prioriteta operatorjev
  - operacije se izvajajo od leve proti desni v skladu s spodaj navedeno prioriteto operaterjev
  - vrstni red lahko spremenimo s pomočjo oklepajev

PRIORITETA	OPERATORJI	SIMBOLI
najvišja	unarni operatorji	- ! ~
	množenje, deljenje	* / %
	seštevanje, odštevanje	+ -
	pomik	<< >> >>>
	primerjanje	> < >= <=
	enakost, neenakost	== !=
	bitni IN	&

	bitni ekskluzivni ALI	^
	bitni ALI	|
	logični IN	&&
	logični ALI	||
	pogojni	? :
najnižja	prirejanje	=

Opomba: Manjkajo še nekateri operatorji, ki jih bomo spoznali kasneje…

Izbirni stavki

- omogočajo, da izmed več različnih zaporedij stavkov izberemo tisto, ki naj se izvede
  - stavek `if` oziroma `if … else`
    - stavek `if`: izbira na podlagi pogoja, ki ima lahko vrednost `true` ali `false`
    - ogledali si bomo tri primere (za `if` oziroma `if … else`):
      - izbira v primeru enega samega zaporedja stavkov

```
if (<pogoj>)
{
// stavki, ki se izvršijo, če je pogoj true
}
```

      - izbira med dvema različnima zaporedjema stavkov (z dodatkom `else`)

```
if (<pogoj>)
{
  // stavki, ki se izvršijo, če je <pogoj> true
}
else
{
  // stavki, ki se izvršijo, če je <pogoj> false
}
```

      - izbira med več različnimi zaporedji stavkov (gnezdenje stavkov `if`)

```
if (<pogoj>)
{
  // stavki, ki se izvršijo, če je <pogoj> true
}
else
{
  // stavki, ki se izvršijo, če je <pogoj> true
}
else
{
  // stavki, ki se izvršijo, če nobeden izmed prej naštetih pogojev ni
izpolnjen
}
```

  - stavek `switch`
    - izbira poteka na podlagi vrednosti nekega izraza (spremenljivke)
      - izraz (spremenljivka) mora biti celoštevilski ali tipa `char`
      - vsako zaporedje stavkov "označimo" z eno izmed možnih vrednosti
      - izvede se tisto zaporedje, pri katerem se "oznaka" ujema z vrednostjo izraza
      - primeren je takrat:
        - ❖ kadar poteka izbira na podlagi ene same celoštevilske ali znakovne spremenljivke
        - ❖ kadar je število različnih vrednosti omejeno

primer uporabe stavka `switch`:

```
switch (ocena) // vsebuje oceno od 1 do 10
{
  case 6:
      System.out.println ("zadostno");
      break;
  case 7:
      System.out.println ("dobro");
      break;
  case 8;
      System.out.println ("prav dobro");
      break;
  case 9;
      System.out.println ("prav dobro");
      break;
  case 10;
      System.out.println ("odlično");
      break;
  default:
      System.out.println ("nezadostno");
}
```

splošna oblika

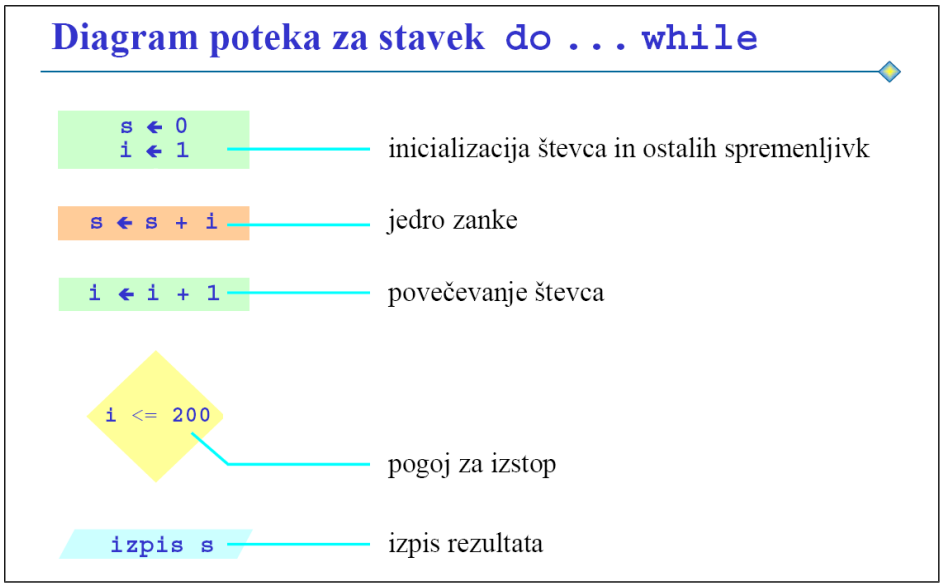
```
switch (<izraz>)
{
  case <k1>:
      // zaporedje stavkov, ki ustreza vrednosti <k1>
      break;
  case <k2>:
      // zaporedje stavkov, ki ustreza vrednosti <k2>
      break;
  …
  default:
      // zaporedje znakov, ki se izvrši, če vrednost izraza ne ustreza nobeni
konstanti
}
```

- pogojni operator (`? :`)
  - pogojni operator (ang. conditional operator) omogoča izbiro med dvema izrazoma
    - splošna oblika: `<pogoj> ? <izraz1> : <izraz2>`
    - če je pogoj izpolnjen, vrne vrednost, ki jo določa `izraz1`, v nasprotnem primeru pa vrne vrednost, ki jo določa `izraz2`
  - primer:

```
vecji = (a>b) ? a : b;

//je isto kot:

if(a>b)
večji=a;
else
večji=b;
```

Ponavljalni stavki

- s stavki za ponavljanje dosežemo, da se določeno zaporedje ukazov večkrat ponovi (zanka)
  - stavek `do … while`



rešitev za izračun vsote s=1+2+3+4+…+200

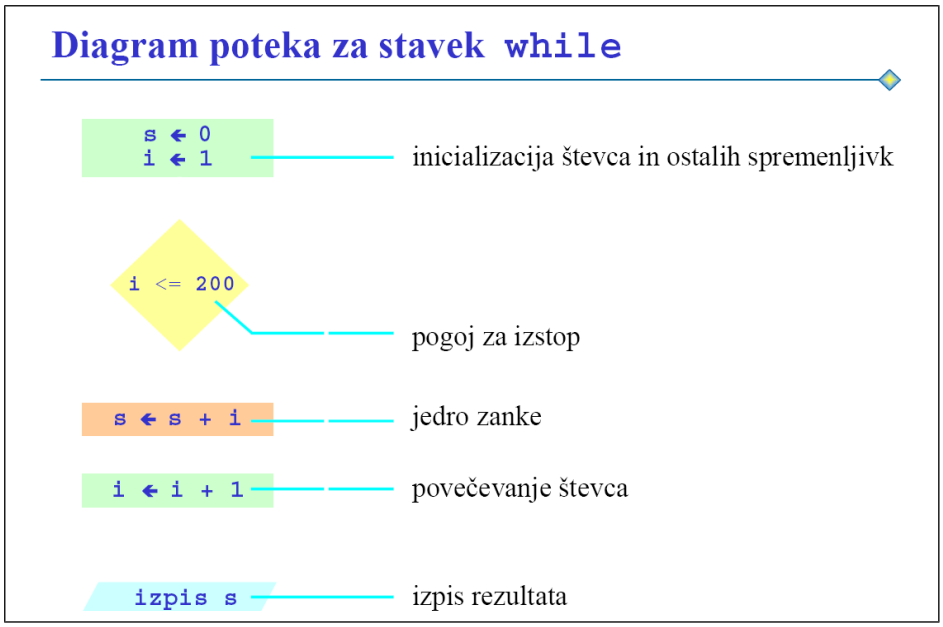
```
public class Vsota1
{
      public static void main (String args [])
      {
            int s=0; //začetna vrednost vsote
            int i=1; //prvi člen, začetna vrednost števca

do
            {
                  s=s+i; //prištevanje člena
                  i=i+1; //naslednji člen, povečanje števca
            } while (i<=200);

System.out.println ("Vsota je "+s);
      }
}
```

splošna oblika

```
do
{
    // stavki, ki se ponavljajo
} while (<pogoj>)
```

- pogoj za ponavljanje je na koncu
- ponavljanje traja toliko časa, dokler je `<pogoj>` `true`
- jedro zanke se izvrši vsaj enkrat

o stavek `while`



rešitev za izračun vsote s=1+2+3+4+…+200

```
public class Vsota2
{
  public static void main(String[] args)
  {
    int s=0;   // začetna vrednost vsote
    int i=1;   // prvi člen, začetna vrednost števca

while (i<=200)
      {
            s=s+i;   // prištevanje člena
            i=i+1;   // naslednji člen, povečanje števca
      }

System.out.println("Vsota je "+s);
  }
}
```

splošna oblika

```
while (<pogoj>)
{
    // stavki, ki se ponavljajo
}
```

- pogoj za ponavljanje je na začetku
- ponavljanje traja toliko časa, dokler je `<pogoj>` `true`
- možno je, da se jedro zanke ne izvrši niti enkrat

o stavek `for`

- večina zank je številnih (ang. counted loop)
  - število iteracij je vnaprej znano
  - s pomočjo števca lahko zapišemo pogoj za ponavljanje
- stavek `for` na enem mestu združuje
  - inicializacijo števca
  - pogoj za nadaljevanje/prekinitev ponavljanja
  - izraz, s katerim je določena nova vrednost števca po vsaki iteraciji
- potek izvajanja:
  - najprej se izvrši inicializacija števca

- nato se preveri pogoj za ponavljanje
- če je pogoj izpolnjen, se izvrši jedro zanke
- na koncu jedra se poveča/zmanjša števec
- sledi ponovno preverjanje pogoja za ponavljanje itd.

- stavek `for` se obnaša podobno kot stavek `while`
  - pogoj na začetku
  - možno je, da se jedro zanke ne izvrši niti enkrat
- kadar število iteracij ni vnaprej znano, ne moremo uporabljati stavka `for` v kombinaciji s stavkom `break`

rešitev za izračun vsote `s=1+2+3+4+…+200`

```
public class Vsota3
{
  public static void main(String[] args)
  {
    int s=0;
    for (int i=1; i<=200; i=i+1)
    {
      s=s+i;   // prištevanje člena
    }
    System.out.println("Vsota je "+s);
  }
}
```

splošna oblika

```
for (<inicializacija števca>;<pogoj>;<povečevanje števca>)
{
    //stavki, ki se ponavljajo
}
```

- primer: izračun vsote `s=1+2+3+4+…+200` (že opisan za vsak primer posebej zgoraj)
- vsoto računamo v zanki
  - začetna vrednost vsote je 0, prvi člen je 1
  - ob vsakem prehodu skozi zanko prištejemo naslednji člen
  - posebna spremenljivka (števec) šteje, koliko členov smo že prišteli
  - s pomočjo števca zapišemo pogoj za izstop iz zanke

Stavka `break` in `continue`

- stavek `break`
  - preskok izbir v stavku `switch`
  - predčasen izstop iz zanke

```
for(…)
{
      …// nekaj stavkov
      if(…) break;
      …// še nekaj stavkov
}
```

  - predčasen izstop iz večjega števila ugnezdenih zank: uporaba oznake `izstop:`

```
for(…)
{
      for(…)
      {
            … // nekaj stavkov
            if(<pogoj>) break;
            …// še nekaj stavkov
```

```
        }
}
```

- stavek `continue`
  - preskok preostalih stavkov v jedru zanke
  - še vedno ostanemo v zanki

```
for(…)
{
        …// nekaj stavkov
        if(<pogoj>) continue;
        …// še nekaj stavkov
}
```

Krajši zapis prireditvenih stavkov (uporaba bližnjic)

- pogosto srečamo prireditvene stavke naslednje oblike:
  - `s=s+i;`
  - `i=i+1;`
- za zgornja stavka obstaja krajši zapis:
  - `s+=i;`
  - `i+=1;`
  - operator `+=` združuje operaciji seštevanja in prirejanja vrednosti
- podobne bližnjice so definirane tudi za ostale aritmetične operacije
  - `s-=i;` je isto kot `s=s-i;`
  - `s*=i;` je isto kot `s=s*i;`
  - `s/=i;` je isto kot `s=s/i;`
- za prištevanje in odštevanje enice lahko uporabljamo unarna operatorja `++` in `--`
  - `i++;` je isto kot `i+=1;` ali `i=i+1;`
  - `i--;` je isto kot `i-=1;` ali `i=i-1;`
- operatorja `++` in `--` lahko uporabljamo v prefiksni ali postfiksni obliki
  - prefiksna oblika `++i`: vrednost `i` se najprej poveča, šele nato uporabi
    `b=4; c=++b;  // c dobi vrednost 5`
  - postfiksna oblika `i++`: vrednost `i` se najprej izvrši, šele nato poveča
    `b=4; c=b++;  // c dobi vrednost 4`
  - v obeh zgornjih primerih dobi `b` vrednost 5

Program za izpis možnih potez lovca na šahovnici

- recimo, da se lovec nahaja v tretji vrstici in drugi koloni šahovnice
- napisati želimo program, ki bo izpisal vse možne poteze v naslednji obliki

```
BCB*BCBC
*B*BCBCB
B*BCBCBC        B – belo polje
*B*BCBCB        C – črno polje
BCB*BCBC        * - možna poteza
CBCB*BCB
BCBCB*BC
CBCBCB*B
```

- program bomo razvijali postopoma
  - začetni položaj določata spremenljivki `zacVrstica` in `zacKolona`
  - izpis šahovnice poteka v zanki: ob vsakem prehodu izpišemo eno vrstico
  - tudi izpis vrstice poteka v zanki: ob vsakem prehodu izpišemo eno polje

```
int zacVrstica=3, zacKolona=2;
int v,k;
```

```
for (v=1; v<=8; ++v)
{
      for (k=1; k<=8; ++k)
            izpiši polje;
      System.out.println();
}
```

- izpis polja: stavek `if`, v katerem izbiramo med 3 možnostmi
  - če je poteza možna, izpišemo zvezdico
  - če je polje belo, izpišemo črko B
  - v ostalih primerih izpišemo črko C

```
if ((v-k==zacVrstica-zacKolona) || (v+k==zacVrstica+zacKolona))
{
      System.out.print("*");
}
else if ((v+k)%2==0)
{
      System.out.print("B");
}
else
{
      System.out.print("C");
}
```

*celotni program:

```
public class Lovec
{
  public static void main(String[] args)
  {
    int zacVrstica=3, zacKolona=2;
    int v, k;
    for (v=1; v<=8; ++v)
    {
      for (k=1; k<=8; ++k)
        if ((v-k==zacVrstica-zacKolona) || (v+k==zacVrstica+zacKolona))
          System.out.print("*");
        else if ((v+k)%2==0)
          System.out.print("B");
        else
          System.out.print("C");
      System.out.println();
    }
  }
}
```

4. Metode

Kazalo:

- Kaj so metode, zakaj jih uporabljamo, kako jih deklariramo
- Primeri metod
  - metoda brez argumentov
  - metoda z argumenti (parametri)
    - vloga parametrov, formalni in dejanski parametri
    - klicanje metod
    - prenos parametrov
  - metode, ki vračajo vrednost
- Rekurzija: metoda kliče samo sebe
  - izračun n-tega Fibonaccijevega števila
  - primerjava rekurzivne in iterativne rešitve
- Zgledi: perfektna števila, izračun vsote številske vrste
- Oblikovanje izpisa in branje podatkov

Kaj so metode, zakaj jih uporabljamo, kako jih deklariramo

- zaporedje stavkov, ki ima svoje ime
  - ime omogoča, da to zaporedje pokličemo z različnih mest v programu
- uporaba
  - ko se zaporedje stavkov ponovi na več različnih mestih
  - ko je treba sprogramirati nek zaokrožen (pod) problem
  - ko se treba sprogramirati operacijo, ki jo izvaja nek objekt

Primeri metod

- metoda brez argumentov
  - vrste metod
    - metode, ki ne vračajo vrednosti (procedure)
      - poudarek je na postopku
      - klic take metode je samostojen stavek
    - metode, ki vračajo vrednosti (funkcije)
      - pudarek je na vrednosti
      - klic take metode je sestavni del izraza
      - v Javi lahko klic funkcije tudi samostojen stavek
    - deklaracije metode
      - glava metode: prva vrstica, s katero damo metodi ime
      - odprt zaviti oklepaj
      - telo metode: stavki, vključno z deklaracijami lokalnih spremenljivk
      - zaprt zaviti oklepaj
    - glava metode
      - določila glede na način dostopa (ang. access modifiers)
        - ❖ zaenkrat `public static`
      - tip rezultata

- ime metode
- seznam argumentov (med dvema okroglima oklepajema)
  - ❖ lahko je prazen
  - ❖ za vsak argument navedemo njegov tip in ime
  - ❖ argumenti so med seboj ločeni z vejicami

▪ metoda za izpis naslova fakultete (metoda brez argumentov)

```
class NaslovFRI
{
  public static void main(String[] args)
  {
    naslov();
  }

  public static void naslov()
  {
    System.out.print("Fakulteta za računalništvo");
    System.out.println(" in informatiko");
    System.out.println();
    System.out.println("Tržaška 25");
    System.out.println("1000 Ljubljana");
  }
}
```

izpiše:

```
Fakulteta za računalništvo in informatiko

Trzaška 25
1000 Ljubljana
```

- metoda z argumenti
  - o vloga parametrov, formalni in dejanski parametri
    - ▪ z argumenti posredujemo metodam podatke, ki jih le-te potrebujejo za svoje delo
      - primer 1: metoda za rezervacijo mize v restavraciji
        - ❖ smiselni argumenti: datum, ura, število oseb
        - ❖ posledica: z isto metodo lahko izvršimo katerokoli rezervacijo
      - primer 2: izračun obresti pri enoletni vezavi
        - ❖ s miselna argumenta: glavnica, obrestna mera (v %)
        - ❖ posledica: z isto metodo lahko izračunamo obresti za poljubno glavnico in poljubno obrestno mero
    - ▪ formalni in dejanski parametri
      - formalni parametri: argumenti/parametri, ki jih navedemo od deklaraciji
      - služijo samo za opis postopka
      - postopek se izvrši nad dejanskimi parametri (argumenti)
      - dejanske parametre navedemo ob klicu metode
    - ▪ primer metode z dvema argumentoma:
      - izračun stanja po enoletni vezavi sredstev

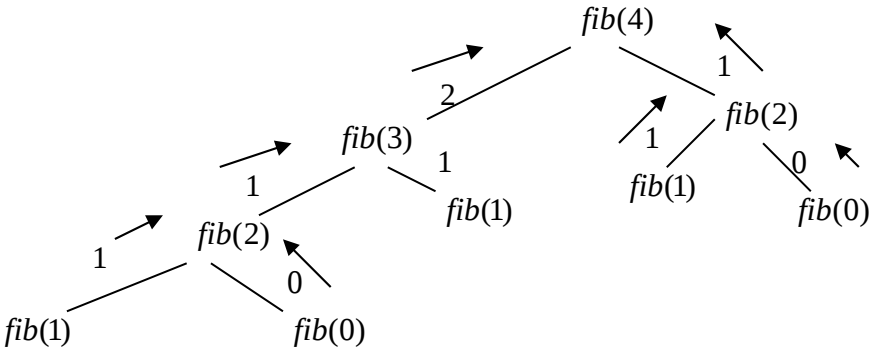
```
public static void poEnemLetu
  (double glavnica, double obrMera)
{
  double novoStanje;
```

```
  novoStanje = glavnica + glavnica * obrMera/100;
  System.out.println ("Po enem letu dobimo " + novoStanje);
}
```

- klic metode `poEnemLetu`
  - `poEnemLetu (100000, 9.5);`
  - ```
    double gl = 100000.0, obr = 9.5;
    poEnemLetu (gl, obr);
    ```

- klicanje metod
  - ob klicu metode navedemo
    - ime metode
      - za metode iz drugih razredov je treba navesti tudi ime razreda
    - dejanske parametre (v oklepaju)
  - formalni in dejanski parametri se morajo ujemati v
    - tipu
    - številu
    - vrstnem redu (istoležni parametri se zamenjajo)
  - dejanski parameter je lahko
    - konstanta
    - spremenljivka
    - izraz
  - navedene spremenljivke se prenašajo po vrednosti, objekti pa po referenci
- prenos parametrov po vrednosti
  - metoda dobi kopijo dejanskega parametra

```
public class PrenosPoVrednosti
{
 public static void main(String[] args)
  {
  int x=10;
  spremeniX(x);
  System.out.println(x); // izpiše 10
  }
  public static void spremeniX(int x)
  {
  x+=2;
  System.out.println(x); // izpiše 12
  }
}
```

- metode, ki vračajo vrednosti (funkcije)
  - metoda lahko vrne vrednost kakršnegakoli tipa
    - enostavnega tipa
    - tipa razred
  - deklaracija metode, ki vrača vrednost
    - v glavi mora biti naveden tip rezultata
    - v telesu mora biti prisoten stavek `return`
  - stavek `return`
    - določa vrednost, ki naj jo metoda vrne
    - običajno zadnji stavek v metodi
```

- splošna oblika: `return<izraz>`

o primer metode, ki vrača vrednost

- izračun stanja po enoletni vezavi sredstev

```
public static double poEnemLetu(double glavnica, double obrMera)
{
  double novoStanje;
  novoStanje = glavnica + glavnica * obrMera / 100;
  return novoStanje;  // namesto izpisa vrne vrednost
}
```

- klic metode `poEnemLetu`
  - `ns=poEnemLetu(100000,9.5);`
  - `double gl=100000,obr=9.5;`
    `System.out.print(»Novo stanje: «+poEnemLetu(gl,obr));`

- rekurzija: metoda kliče samo sebe

o o rekurziji govorimo takrat, kadar metoda kliče samo sebe

- uporaba rekurzije je smiselna takrat, kadar je problem definiran rekurzivno
- primer: izračun n-tega Fibonaccijevega števila

$fib_n = fib_{n-1} + fib_{n-2} \qquad n>1$
$fib_0 = 0, fib_1 = 1$

```
public static int fib(int n)
{
    if (n<=1)
      return n;
    else
      return fib(n-1)+fib(n-2);
}
```

risani primer Fibonaccijevega števila



o primerjava rekurzivne in iterativne rešitve

- vsako rekurzivno metodo lahko pretvorimo v iterativno
- če je iterativna rešitev preprosta, damo prednost iteraciji pred rekurzijo
  - iterativna rešitev zagotavlja boljše performanse; je hitrejša
  - vsak rekurziven klic povzroči nekaj dodatnega dela zaradi dodeljevanja pomnilniškega prostora za lokalne spremenljivke in zamenjave formalnih parametrov z dejanskimi
- če je iterativna rešitev kompliciranа, lahko algoritme, ki so po svoji naravi rekurzivni, izrazimo z rekurzivnimi metodami
  - hanojski stolpiči
  - operacije nad drevesnimi strukturami
- iterativna rešitev za Fibonaccijeva števila

```
public static int fibIte(int n)
```

```
{
    int x=1,y=0;        // zadnje in predzadnje Fibonaccijevo število
    int z;              //začasna pomožna spremenljivka (izračun x-sa)
    for (int i=2; i<=n; ++i)
    {
      z=x;              // zacasno shranimo zadnje Fibonaccijevo število
      x=x+y;            // naslednje Fibonaccijevo število
      y=z;              // prejšnje Fibonaccijevo število
    }
    return x;
}
```

primer za Fibonaccijevo število (oba načina):

```
public class Fibonacci
{
  public static void main(String[] args)
  {
    System.out.print("Vpiši katero fibonaccijevo število po vrsti želiš
izpisano: ");
    int n=BranjePodatkov.preberiInt();

    System.out.println(n+". Fibonaccijevo stevilo (rekurzivno) je "+fib(n));
    System.out.println(n+". Fibonaccijevo stevilo (iterativno) je "+fibIte(n));
  }

  public static int fib(int n)
  {
    if (n<=1)
      return n;
    else
      return fib(n-1)+fib(n-2);
  }

  public static int fibIte(int n)
  {
    int x=1,y=0;        // zadnje in predzadnje Fibonaccijevo število
    int z;
    for (int i=2; i<=n; ++i)
    {
      z=x;              // zacasno shranimo zadnje Fibonaccijevo število
      x=x+y;                  // naslednje Fibonaccijevo število
      y=z;              // prejšnje Fibonaccijevo število
    }
    return x;
  }
}
```

- zgledi: perfektna števila, izračun vsote številske vrste
  - funkcijo sinus lahko aproksimiramo z vsoto številske vrste

$$s = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \ldots + \frac{(-1)^i * x^{2i+1}}{(2i+1)!}$$

- vsoto izračunamo v zanki, ob vsakem prehodu prištejemo en člen

```
določi začetno vrednost člena in vsote;
while (ni še dosežena predpisana natančnost)
{
      izračunaj naslednji člen;
      prištej vsoti naslednji člen;
}
return vsota;
```

- naslednji člen izračunamo iz prejšnjega člena

$$k_{j+1} = k_j + 2$$

$$clen_{j+1} = \frac{-clen_j * x^2}{(k_{j+1} * (k_{j+1} - 1))}$$

*iz prosojnic:

```
k+=2;
clen=-clen*x*x/(k*(k-1));
```

- začetna vrednost člena in pomožne spremenljivke k

```
clen0 = x          k0 = 1
```

- določitev začetnih vrednosti: prvi člen upoštevamo v inicializaciji

```
double clen=x, vsota=clen;
int k=1;
```

- preverjanje dosežene natančnosti
  - s prištevanjem členov prenehamo, ko absolutna vrednost člena postane manjša od 10^{-5}
  - funkcijo, ki vrne absolutno vrednost, dobimo v razredu `Math`

```
double clen=x, vsota=clen;
int k=1;
while (Math.abs(clen)>=1e-5)
{
      k+=2;
      clen=-clen*x*x/(k*(k-1));
      vsota+=clen;
}
return vsota;
```

primer, ki ga je pokazal na predavanjih

```
public class MatFunkcije
{
  static double eps=0.00001;

  public static double sinus(double x)
  {
    double vsota, clen;
    int k;
    clen=x;
    k=1;
    vsota=clen;
    while (Math.abs(clen)/Math.abs(vsota)>eps)
    {
      k+=2;                          // k=k+2;
      clen=-clen*x*x/(k*(k-1));
      vsota+=clen;                   // vsota=vsota+clen;
    }
    return vsota;
  }
}
```

vaja: Napišite program, ki izpiše vrednosti kotne funkcije sinus za vse kote od 0 do 360 stopinj z intervalom 30 stopinj

Kot	sinus
0	xx
30	xx
60	xx
…	…

rešitev:

```
public class Tabela
{
  public static void main(String[] args)
  {
    int kot;
    System.out.println("Kot          Sinus");
    System.out.println("--------------------------");
    for (kot=0; kot<=360; kot+=30)
    {
      System.out.println(kot+"        "+MatFunkcije.sinus(Math.PI/180*kot));
    }
  }
}
```

- oblikovanje izpisa in branje podatkov
  - formatiranje izpisa
    - metoda `printf`, dodana v verziji 1.5 (razred `PrintWriter`)
      - splošna oblika: `printf(format,args)`
        - `format` je niz, v katerem je za vsak argument posebej določena oblika izpisa
        - `args` je seznam vrednosti, ki naj se izpišejo
      - specifikacija izpisa za numerične vrednosti in znake:
        - `%[argument_index$][flags][width]`
          `[.precision]conversion`
        - `argument_index:` zaporedna številka argumenta
        - `flags:` znaki, ki podrobneje določajo posebnosti izbranega formata (npr. leva poravnava, obvezen izpis predznaka, izpis negativnih števil v oklepaju ipd.)
        - `width`: skupno število mest, vključno z decimalkami in eksponentom
        - `precision`: število decimalk
        - `conversion`: znak, ki dejansko določa obliko izpisa
      - najpogosteje uporabljene oblike izpisa za števila
        - `d` : izpis celih števil v desetiškem sistemu
        - `f` : izpis realnih števil v obliki s fiksno vejico
        - `e` ali `E` : izpis realnih števil v eksponentni obliki

--- manjka Tabela1 (novi »programčki profesorja«)---

  - branje podatkov
    - metode, v katerih beremo podatke, morajo imeti v glavi `throws Exception`

```
public static void main (String args []) throws Exception
```

    - metoda `System.in.read ()`
      - vrne vrednost tipa `int`, ki predstavlja kodo vtipkanega znaka
        - npr. črki A ustreza koda 41_{16} oziroma 65_{10}
      - to vrednost lahko pretvorimo v tip `char`
      - primer:

```
char vtipkanZnak;
vtipkanZnak = (char) System.in.read ()
```

      - števila in nize dobimo tako, da v zanki beremo znake in jih sestavimo v število oziroma niz
    - razred `Branje Podatkov`
      - metoda za branje celih števil: `preberiInt ()`

- metoda za branje realnih števil: `preberiDouble ()`
- metoda za branje nizov: `preberiString ()`

- primer uporabe

```
int st;
String bes;
double re;

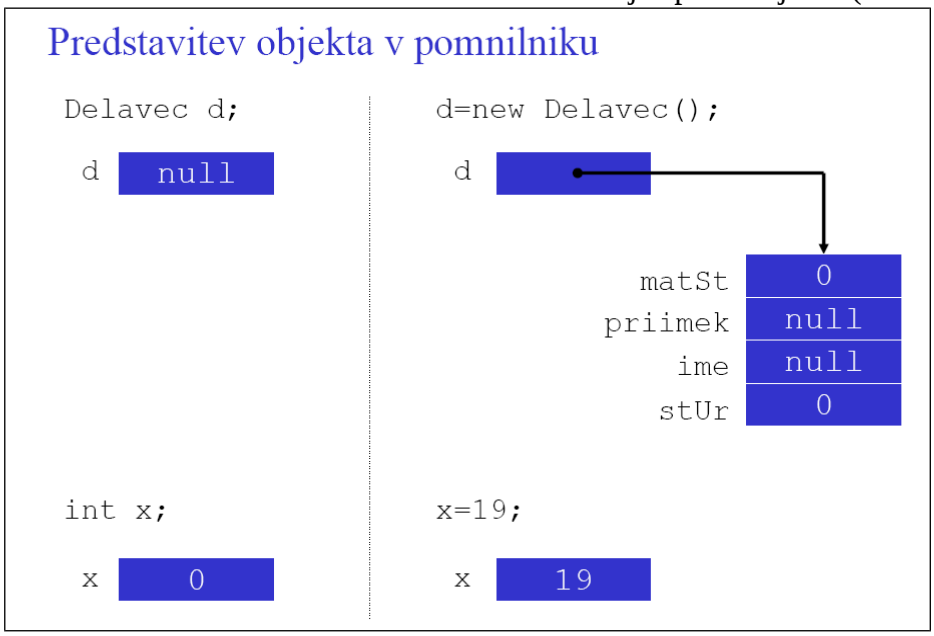
st = BranjePodatkov.preberiInt ();
bes = BranjePodatkov.preberiString ();
re = BranjePodatkov.preberiDouble ();
```

vaja:
Napišite program, ki prebere 5 primerov krogov in izpiše koliko krogov je v celoti znotraj kvadrata, koliko krogov seka kvadrat in koliko krogov je izven kvadrata s stranico a=10. Središče vseh krogov je v težišču kvadrata.

1. primer:

```
public class prva
{
  static final int N=5;    // število krogov    ti dve sta konstanti (z
veliko!!!!!!!!!!!!!!) ;-)
  static final int A=10;   // stranica kvadrata

  public static void main(String[] args)
  {
    double premer;
    int v=0,izven=0,seka=0;
    for (int i=1;i<=N;++i)
    {
      System.out.print("Vtipkaj premer:");
      premer=BranjePodatkov.preberiDouble();
      if (premer<A)
        ++v;
      else if (premer>A*Math.sqrt(2))
        ++izven;
      else
        ++ seka;
    }
    System.out.println("V kvadratu je "+v+" krogov.");
    System.out.println("Kvadrat seka "+seka+" krogov.");
    System.out.println("Izven kvadrata je "+izven+" krogov.");
  }
}
```

2. primer:

```
public class prva
{
      public static void main(String[] args)
      {
            int v=0;
            int seka=0;
            int izven=0;

            double d;

            for(int i=1; i<=5; ++i)
            {
                  System.out.print("Vtipkaj premer: ");
                  d=BranjePodatkov.preberiDouble();

            if(d<10)
```

```
            {
                  v++;
            }
            else if(d<10*Math.sqrt(2))
            {
                  izven++;
            }
            else
            {
                  seka++;
            }

            System.out.println("V kvadratu je "+v+" krogov");
            System.out.println("Kvadrat seka "+seka+" krogov");
            System.out.println("Izven kvadrata je "+izven+" krogov");
             }
      }
}
```

vaja: Napišite program, ki simulira 300 metov kocke in izpiše število koliko je bilo 1 (enic), 2, 3, 4, 5, 6.

```
public class prva
{
      static final int N=300;
      public static void main(String[] args)
      {
            int st1=0;
            int st2=0;
            int st3=0;
            int st4=0;
            int st5=0;
            int st6=0;
            for(int i=1; i<=N; i++)
            {
                  double ns=Math.random();
                  int met=(int)Math.floor(6*ns)+1;
                  switch(met)
                  {
                        case 1:
                        {
                              st1++;
                              break;
                        }
                        case 2:
                        {
                              st2++;
                              break;
                        }
                        case 3:
                        {
                              st3++;
                              break;
                        }
                        case 4:
                        {
                              st4++;
                              break;
                        }
                        case 5:
                        {
                              st5++;
```

```
                                    break;
                              }
                              case 6:
                              {
                                    st6++;
                              }
                        }
                  }
                  System.out.println(st1+" "+st2+" "+st3+" "+st4+" "+st5+" "+st6);
            }
}
```

5. Razredi in objekti

Kazalo:

- Osnovni koncepti OOP: razred, objekt, atribut, metoda
- Programiranje razredov
  - deklaracije atributov
  - deklaracije metod
  - deklaracije konstruktorjev
- Kreiranje objektov
  - operator new: generira objekt med izvajanjem programa
  - predstavitev objekta v pomnilniku
- Pisanje konstruktorjev
  - večkratno definirane metode

Osnovni koncepti OOP:razred, objekt, atribut, metoda

- programske komponente so objekti (ang. objects), ki so podobni objektom iz realnega sveta.
- vsi objekti iste vrste tvorijo razred (ang. class)
- primer 1:
  - razred `Avtomobil` združuje vse avtomobile
  - moj avto je objekt, ki spada v razred `Avtomobil`
- primer 2:
  - razred `Pes` predstavlja vse pse
  - psica `Lajka` je eden izmed objektov, ki sestavljajo razred `Pes`
- vsak objekt je konkreten primerek (ang. instance) nekega razreda
- vsak objekt vsebuje atribute (ang. attributes) in metode (ang. methods)
  - z atributi so opisane lastnosti in trenutno stanje nekega objekta
  - z metodami so določene operacije, ki jih ta objekt lahko izvaja
- primer za objekt, ki spada v razred `Avtomobil`
  - atributi: znamka, model, leto izdelave, barva, prostornina motorja, ali trenutno vozi, s kakšno hitrostjo, v kateri prestavi
  - metode: vožnja naprej, vožnja nazaj, polnjenje rezervoarja, pranje avtomobila, ugotavljanje trenutnega stanja (npr. trenutna hitrost, količina goriva v rezervoarju ipd.)
- vsi objekti nekega razreda imajo iste atribute, toda drugačne vrednosti atributov
- razlika med razredom in objektom je kot razlika med abstraktnim in konkretnim
  - razred je abstrakten opis atributov in metod
  - objekt je konkreten primerek nekega razreda s točno določenimi vrednostmi atributov in točno določenimi metodami
- enkapsulacija (ang. encapsulation)
  - atributi in metode so vgrajeni v objekt tako, da se le-ta obnaša navzven kot “črna škatla”
  - za uporabnika ni pomembno, kako so realizirane posamezne metode, ampak zadostuje, da pozna vmesnik (ang. interface)
  - primer: za natakanje goriva ni treba vedeti, kje je rezervoar; zadostuje, da vemo, kje je odprtina

Programiranje/kreiranje razredov

- deklaracija razreda
  - glava razreda
  - deklaracije atributov
  - deklaracije metod
  - deklaracija konstruktorjev
- glava razreda
  - določilo za način dostopa (ang. access modifier)
    - neobvezno
    - največkrat `public`, možno še `private`, `final` ali `abstract`
  - rezervirana beseda `class`
  - ime razreda (določi programer)
- deklaracija atributov
  - atribute običajno deklariramo kot spremenljivke, ki so lokalne razredu, zato uporabimo dostopno določilo `private`
  - z določilom `private` dosežemo, da se razred oziroma njegovi objekti obnašajo kot "črne škatle"
    - atributi niso vidni navzven, ampak so dostopni samo preko metod, ki so deklarirane v razredu
    - posledica: potrebujemo metode, ki vpisujejo vrednosti atributov v posamezne objekte oziroma vračajo vrednosti atributov določenega objekta
  - koncept skrivanja informacij (ang. information hiding) je zelo pomemben koncept objektno usmerjenega programiranja
    - onemogoča, da bi kdo od zunaj neposredno posegal v stanje objekta
  - primer deklaracije atributov za razred `Delavec`

```
public class Delavec
{
   private int matStev;
   private String priimek;
   private String ime;
   private int stUr;
}
```

- deklaracije metod
  - metode običajno deklariramo kot `public`
  - dostopno določilo `static` izpustimo
    - statične metode so v pomnilniku shranjene samo enkrat
    - statične metode ne morejo posegati v atribute posameznih objektov
    - lahko se kličejo tudi takrat, ko ne obstaja noben objekt
- primer deklaracije metod za razred `Delavec`
  - zaradi skrivanja informacij potrebujemo metode za vpis in branje vrednosti posameznih atributov:

```
vpisiMatSt (int st)          vrniMatSt ()
vpisiPriimek (String  p)     vrniPriimek ()
vpisiIme (String i)          vrniIme ()
vpisiStUr (int u)            vrniStUr ()
```

  - dodamo še druge metode, odvisno od problema, ki da želimo sprogramirati, npr.

```
izracunajBrutoOD (int cenaUre)
izpisiVse ()
```

Kreiranje objektov

- deklaracija razreda ne kreira nobenega objekta, ampak predstavlja samo abstrakten opis strukture objektov
- kreiranje objektov poteka v dveh korakih
  - najprej moramo objekt deklarirati (podobno kot spremenljivko)
  - nato objekt generiramo s pomočjo operatorja `new` in posebne metode, ki ji rečemo konstruktor (ang. constructor method)

primer:

```
public class Delavec1
{
  // atributi
  private int matStev;
  private String priimek;
  private String ime;
  private int stUr;
  // konstruktor
  Delavec1(int ms, String p, String i)
  {
    matStev=ms;
    priimek=p;
    ime=i;
  }
  // metode
  public void vpisiMatSt(int st)
  {
    matStev=st;
  }
  public void vpisiPriimek(String p)
  {
    priimek=p;
  }
  public void vpisiIme(String i)
  {
    ime=i;
  }
  public void vpisiStUr(int u)
  {
    stUr=u;
  }
  public int vrniMatSt()
  {
    return matStev;
  }
  public String vrniPriimek()
  {
    return priimek;
  }
  public String vrniIme()
  {
    return ime;
  }
  public int vrniStUr()
  {
    return stUr;
  }
  public int izracunajBrutoOD(int cenaUre)
  {
    return stUr*cenaUre;
  }
```

```
  public void izpisiVse()
  {
    System.out.println("Maticna stevilka: "+matStev);
    System.out.println("Priimek in ime: "+priimek+' '+ime);
    System.out.println("Stevilo ur: "+stUr);
  }
}
```

- primer: kreiranje objekta `d`, ki spada v razred `Delavec`

```
Delavec d;              // deklaracija objekta
d = new Delavec ();     // generiranje objekta
```

- oba koraka lahko združimo:

```
Delavec d = new Delavec ();
```

- operator `new`
  - dodeli prostor za objekt
  - spremenljivka `d` dobi naslov tiste lokacije v pomnilniku, kjer se nahaja objekt
  - `d = new Delavec ();` je prireditveni stavek, ki vpiše naslov objekta v spremenljivko `d`
- razlika v primerjavi z navadnimi spremenljivkami
  - za navadne spremenljivke se prostor dodeli že med prevajanjem (ob deklaraciji), za objekte pa šele ob klicu konstruktorja
  - navadne spremenljivke hranijo vrednost (tj. nek podatek), imena objektov pa vsebujejo naslove
  - razlikovati moramo med vrednostjo spremenljivke (naslovom) in vrednostjo objekta



- sklicevanje na atribute objekta
  - navedemo ime objekta, piko in ime atributa
  - primeri za objekt `d` tipa `Delavec`:

```
d.matSt
d.priimek
d.ime
d.stUr
```

- uporaba metod, ki pripadajo objektu (ang. instance methods)
  - metode nekega objekta pokličemo tako, da navedemo
    - ime objekta
    - piko

        - ime metode
        - dejanske parametre (v oklepaju)
    - primer za objekt `d` tipa `Delavec`: `d.vpisiIme("Janez”);`

Pisanje konstruktorjev

- konstruktor
    - metoda, ki se izvede ob kreiranju objekta
    - ime konstruktorja je enako imenu razreda
    - v Javi je konstruktor avtomatsko na razpolago
    - programer lahko napiše svoj lasten konstruktor (ali celo več konstruktorjev)
    - konstruktor lahko pokličemo samo z operatorjem `new` (ne moremo ga klicati samostojno, kot druge metode)
- standarden konstruktor nastavi začetne vrednosti atributov
    - numerični atributi dobijo vrednost 0
    - atributi tipa `char` dobijo vrednost Unicode `'\u0000'`
    - atributi tipa `boolean` dobijo vrednost `false`
    - atributi objektnega tipa dobijo vrednost `null`

*opomba(program Delavec.java mora biti v isti mapi!!!)

```
public class PreizkusDelavec
{
      public static void main (String[] args)
    {
        // kreiranje objekta
            Delavec d=new Delavec();

            // vpis vrednosti atributov
            d.vpisiMatSt(234);
            d.vpisiPriimek("Novak");
            d.vpisiIme("Janez");
            d.vpisiStUr(182);

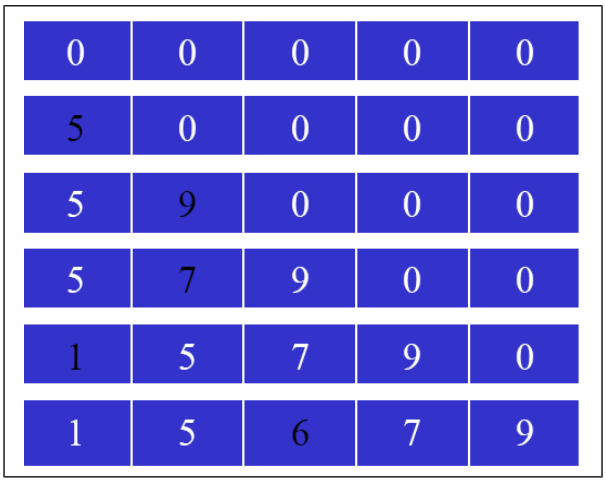
            //izpis vsebine objekta s pomočjo posameznih metod
            System.out.println("Matična številka: "+d.vrniMatSt());
            System.out.println("Priimek in ime: "+d.vrniPriimek()+"
"+d.vrniIme());
            System.out.println("Število ur: "+d.vrniStUr());


            //izpis vsebine objekta s pomočjo metode izpisivse()
            d.izpisivse();

            //izpis bruto osebnega dohodka
            System.out.println("Bruto OD: "+d.izracunajBrutoOD(1500));
    }
}
```

- namesto standardnega konstruktorja lahko napišemo svojega
    - vzpostavimo lahko drugačne začetne vrednosti atributov
    - ob kreiranju objekta lahko izvršimo še druge naloge
    - konstruktorju lahko dodamo svoje argumente
    - standarden konstruktor ni več na razpolago
- primer lastnega konstruktorja za objekte tipa `Delavec`
    - ob generiranju objekta se vzpostavijo prave vrednosti atributov matična številka, priimek in ime

- število ur se vnaša naknadno s pomočjo metode `vpisiStUr`

```
Delavec (int ms, String p, String i)
{
   matStev = ms;
   priimek = p;
   ime = i;
}
```

- večkratno definiranje metod (ang. overloading)
  - deklariramo več metod z enakim imenom, a drugačnimi parametri
    - iz klica metode je razvidno, katera metoda se bo v resnici izvedla
    - izvede se tista metoda, pri kateri se dejanski parametri ujemajo s formaln
  - primer: metoda za izračun obresti

```
public static double obresti (double g, double om)
{
      return g*om;
}
public static double obresti (double g, int om)
{
      return g*om/100;
}
```

    - ob klicu `obresti (1000.,0.08)` se izvede prva, ob klicu `obresti (1000.,8)` pa druga metoda
  - večkratno definiranje konstruktorjev
    - omogoča, da inicializiramo objekt na različne načine
  - primer: kreiranje objektov tipa `Delavec`
    - konstruktor `Delavec ()` inicializira objekt tako, da vsebuje matično številko 9999
    - konstruktor `Delavec (int ms)` inicializira objekt tako, da vsebuje številko ms
    - konstruktor `Delavec (int ms, String p, String i)` inicializira objekt tako, da vsebuje matično številko, priimek in ime, ki so posredovani z argumenti `ms, p in i`

```
public class Delavec2
{
// zgled z vec konstruktorji
  // atributi
  private int matStev;
  private String priimek;
  private String ime;
  private int stUr;
  // konstruktorji
  Delavec2()
  {
    matStev=9999;
  }
  Delavec2(int ms)
  {
    matStev=ms;
  }
  Delavec2(int ms, String p, String i)
  {
    matStev=ms;
    priimek=p;
    ime=i;
  }
```

```
// metode
public void vpisiMatSt(int st)
{
  matStev=st;
}
public void vpisiPriimek(String p)
{
  priimek=p;
}
public void vpisiIme(String i)
{
  ime=i;
}
public void vpisiStUr(int u)
{
  stUr=u;
}
public int vrniMatSt()
{
  return matStev;
}
public String vrniPriimek()
{
  return priimek;
}
public String vrniIme()
{
  return ime;
}
public int vrniStUr()
{
  return stUr;
}
public int izracunajBrutoOD(int cenaUre)
{
  return stUr*cenaUre;
}
public void izpisiVse()
{
  System.out.println("Maticna stevilka: "+matStev);
  System.out.println("Priimek in ime: "+priimek+' '+ime);
  System.out.println("Stevilo ur: "+stUr);
}
```

6. Organizacija programa in dostopnost deklaracij

Kazalo:

- Območje veljavnosti deklaracij spremenljivk
  - lokalne in globalne spremenljivke
- Dostopno določilo `static`
  - spremenljivke objekta in spremenljivke razreda
  - metode objekta in metode razreda
  - referenca `this` v metodah objekta
- Dostopno določilo `final`
- Dostopna določila `public`, `private` in `protected`
- Uporaba vnaprej deklariranih razredov
  - `Math`
  - `Character`

Območje veljavnosti deklaracij spremenljivk

- deklaracija spremenljivke velja:
  - od mesta, kjer je bila spremenljivka deklarirana
  - do konca bloka, v katerem smo jo deklarirali
- primer: gnezdenje blokov

```
{ // zunanji blok
  int spr1 = 10;
  …
  { // notranji blok
    int spr2 = 20;
    … // obstajata spr1 in spr2
    int spr1 = 34; //napaka
  }
  // spr1 še vedno obstaja, spr2 pa ne
  …
}
```

- lokalne in globalne spremenljivke
  - globalne spremenljivke
    - deklarirane so na ravni razreda
    - dostopne so vsem metodam v razredu
  - lokalne spremenljivke
    - deklarirane so znotraj posameznih metod
    - dostopne so samo v metodi, kjer so deklarirane
  - globalne spremenljivke so lahko statične ali vezane na posamezne objekte, odvisno od dostopnega določila `static`
    - statične spremenljivke: obstaja ena sama spremenljivka za celoten razred, zato jim rečemo tudi spremenljivke razreda (ang. Class variables)
    - spremenljivke objekta: vsakemu objektu pripada svoja spremenljivka, ki vsebuje vrednost atributa tistega objekta (ang. instance variables)
  - primer: Razred, ki opisuje zgradbo objektov

```
public class Primer1
{
  private double atr1;        // vsak objekt ima 2 atributa, ki sta globalni
  private int atr2;           // spremenljivki, dostopni v celem razredu
```

```
  public void metoda1()
  {
    int spr3=1;               // lokalni spremenljivki,
    int spr4=25;              // dostopni v metodi metoda1()
    // obstajajo atr1, atr2, spr3 in spr4
   }

   public int metoda2()
  {
    double spr5=500.0;        // lokalna spremenljivka
   // obstajajo atr1, atr2=10 in spr5
   }
}
```

- primer: Razred, ki predstavlja aplikacijo

```
public class Primer2
{
      static double spr1=2.5;  // globalni spremenljivki
      static int spr2=10;      // dostopni v celem razredu
      public static void main (String args [])
      {
            int spr3=1;             // lokalni spremenljivki
            int spr2=25;            // dostopni v metodi main ()
                                          // obstajajo spr1=2.5, spr2=25 in
spr3=1
      }
      public static void xy ()
      {
            double spr4=500.0;  //lokalna spremenljivka
                                          // obstajajo spr1=2.5, spr2=10 in
spr4=500.0
      }
}
```

Dostopno določilo `static`

- statične spremenljivke in spremenljivke objekta
  - statične spremenljivke (ang. class variables)
    - tipične so za razrede, ki predstavljajo aplikacije, vendar lahko nastopajo tudi v razredih, ki opisujejo zgradbo objektov
    - deklarirane so z rezervirano besedo `static`
    - obstaja samo ena kopija spremenljivke
    - to kopijo uporabljajo vse metode in vsi objekti
    - spremenljivka obstaja tudi v primeru, ko nismo kreirali nobenega objekta
    - sprememba vrednosti je dostopna vsem objektom in metodam v razredu
  - spremenljivke objekta (ang. instance variables)
    - nastopajo v razredih, ki opisujejo zgradbo objektov
    - pri deklaraciji ne smemo uporabiti dostopnega določila `static`
    - vsak objekt ima svojo kopijo spremenljivke (tj. svojo vrednost atributa)
    - sprememba vrednosti se odraža samo znotraj objekta
- statične metode in metode objekta
  - dostopno določilo `static` pri metodah
    - statične metode ali metode razreda (ang. class methods)
    - metode objekta (ang. instance methods)
  - statične metode / metode razreda

        - tipične so za razrede, ki predstavljajo aplikacije, in razrede, ki služijo kot knjižnice podprogramov (npr. razred Math)
        - deklariramo jih z dostopnim določilom `static`
        - v pomnilniku so shranjene samo enkrat
        - niso vezane na posamezne objekte, ampak so skupne za celoten razred
        - uporabljamo jih tudi takrat, ko ne obstaja noben objekt
        - ne morejo posegati v atribute posameznih objektov
        - metoda `main()` mora biti obvezno statična
        - če deklariramo statično metodo v razredu, ki je namenjen generiranju objektov, potem je ta skupna (tj. enaka) za vse objekte
- metode razreda in metode objekta
    - metode objekta
        - vezane so na posamezne objekte (tj. primerke nekega razreda)
            - predstavljamo si lahko, da ima vsak objekt svoje metode
            - v resnici so tudi metode, ki pripadajo objektom istega tipa, shranjene v pomnilniku samo enkrat; s posebnim mehanizmom dosežemo, da se izvede prava metoda (dinamično povezovanje)
        - tipične so za razrede, ki so namenjeni generiranju objektov
        - omogočajo dostop do posameznih atributov v objektu
        - pri njihovi deklaraciji ne smemo uporabiti dostopnega določila `static`
        - ob klicu metode objekta je treba obvezno navesti tudi ime objekta, ki mu metoda pripada

```
<ime objekta> . <ime metode> (<dejanski parametri>)
<ime razreda> . <ime objekta> . <ime metode> (<dejanski parametri>)
```

- referenca `this`
    - implicitno prisotna v vsaki metodi objekta
    - predstavlja naslov objekta, na katerega se nanaša klic metode
    - ta naslov omogoča, da dostopamo do spremenljivk pravega objekta
    - pred vsakim takim sklicevanjem na spremenljivko objekta dejansko stoji referenca `this`
    - referenco `this` avtomatsko vstavi prevajalnik, lahko pa jo vključi že programer

```
public void izpisiVse ()
{
      System.out.println ("Maticna stevilka: " +this.matStev);
      System.out.println ("Priimek in ime: " +this.priimek+ ' '+this.ime);
      System.out.println ("Stevilo ur: "+this.stUr);
}
```

Dostopno določilo `final`

- kadar želimo preprečiti spremembo neke deklaracije
    - pri spremenljivkah: vrednost spremenljivke se ne more več spremeniti
    - pri metodah: metode ni moč redefinirati (je dokončna)
    - pri razredih: razreda ni moč razširiti (vse metode so dokončne)
- deklaracije konstant
    - konstante deklariramo na enak način kot spremenljivke, le da dodamo rezervirano besedo `final`
    - velja dogovor, da imena konstant pišemo z velikimi črkami (`static final double PI=3.14159;`)
    - konstanti moramo prirediti vrednost ob deklaraciji; kasneje to ni več mogoče

- prednost uporabe konstant
  - boljša čitljivost programa
  - enostavnejše vzdrževanje

Dostopna določila `public, private` in `protected`

- določajo način dostopa do posameznih atributov in metod v razredu

dostopno določilo	dovoljen dostop
brez dostopnega določila	iz kateregakoli razreda v istem paketu
public	iz kateregakoli razreda glede na paket
private	samo znotraj razreda
protected	iz kateregakoli razreda v istem paketu in iz kateregakoli podrazreda ne glede na paket

Vnaprej deklarirani razredi

- v Javi obstaja približno 500 vnaprej deklariranih razredov
  - ti razredi so shranjeni v obliki paketov
  - paket si lahko predstavljamo kot skupino sorodnih razredov, ki so shranjeni v isti mapi (poddirektoriju)
  - paket `java.lang` vsebuje osnovne razrede, ki se največ uporabljajo
- uporaba vnaprej deklariranih razredov
  - nekateri paketi, kot npr. java.lang, so na razpolago avtomatsko
  - uporabo ostalih paketov ali razredov je treba napovedati s stavkom import, ki mora biti naveden na začetku programa
    - `import java.util.*` napove uporabo vseh razredov iz paketa `java.util`
    - `import java.util.Date` napove uporabo razreda `Date` iz paketa `java.util`
- razred `Math`
  - vsebuje matematične konstante in metode
    - deklariran je v paketu `java.lang`
    - klicanje konstant (`PI, E`): `Math.<ime konstante>`
    - klicanje metod: `Math.<ime metode> (<argumenti>)`

`abs (x)`	absolutna vrednost x
`sin (x), cos (x), tan (x)`	trigonometrične funkcije sinus, kosinus in tangens
`asin (x), acos (x); atan (x)`	obratne trigonometrične funkcije
`round (x)`	zaokroževanje na najbližje celo število
`sqrt (x)`	kvadratni koren iz x
`random ()`	naključno število med 0.0 in 1.0
`exp (x), log (x)`	eksponentna in logaritemska funkcija

- razred `Character`
  - vsebuje koristne metode za delo z znaki
    - deklariran je v paketu `java.lang`
    - klicanje metod: `Character.<ime metode> (<argChar>)`

`isUpperCase ()`	Preveri, ali je znak velika črka
`toUpperCase ()`	Pretvori malo črko v veliko
`isLowerCase ()`	Preveri, ali je znak mala črka
`toLowerCase ()`	Pretvori veliko črko v malo

isDigit ()	Preveri,ali je znak številka ('0' – '9')
isLetter ()	Preveri, ali je znak črka
isLetterOrDigit ()	Preveri, ali je znak črka ali številka
isWhiteSpace ()	Preveri, ali je znak presledek, tab, newline, carriage return ali form feed

```
public class DemoCharacter
{
  public static void main(String[] args) throws Exception
  {
    char znak;

    do
    {
      System.out.println("Vtipkaj znak: ");
      znak=(char)System.in.read();
      System.in.read(); System.in.read();    // tipka Enter

      if (Character.isLetterOrDigit(znak))
      {
        System.out.println(znak+" je crka ali stevilka.");
        if (Character.isLetter(znak))
        {
          if (Character.isUpperCase(znak))
            System.out.println(znak+" je velika crka.");
          else
            System.out.println(znak+" je mala crka.");
        }
        else
          System.out.println(znak+" je stevilka.");
      }
      else if (Character.isWhitespace(znak))
        System.out.println(znak+" je \"whitespace\".");
      else
        System.out.println(znak+" je locilo ali poseben znak.");
    } while (znak!='#');
  }
}
```

7. Tabele

Kazalo:

- Koncept tabele
- Kreiranje tabele
  - deklaracija in dodelitev prostora
  - vpis vrednosti
  - atribut `length`
- Nekatere operacije nad tabelo
  - iskanje elementa
  - dodajanje v urejeno tabelo
  - urejanje (sortiranje)
- Dvodimenzionalne
- Večdimenzionalne tabele
- Primeri
  - razdalje med točkami
  - množenje matrik
- Tabele objektov

Koncept tabele

- tabela je sestavljena podatkovna struktura, ki združuje več elementov istega tipa
  - elementi so lahko enostavnega tipa, npr. `int, double, char`
  - elementi so lahko objekti, ki pripadajo istemu razredu
  - tabela kot celota ima svoje ime
  - vsak element tabele ima svoj indeks (zaporedno številko)
    - v javi tečejo indeksi od 0 dalje
- primer: osebni dohodki delavca v preteklem letu
  - namesto 12 samostojnih spremenljivk `od1, od2, … od12` vpeljemo tabelo
  - tabeli damo ime `od`, od posameznih elementov pa dostopamo s pomočjo indeksov
  - `od [0]` je prvi element, `od [1]` je drugi element itd.
- ponazoritev tabele `od`

od [0]	od [1]	od [2]	od [3]	…	od [11]

- uporaba tabel olajša nekatere operacije, npr. izračun vsote osebnih dohodkov
  - če bi imeli 12 samostojnih spremenljivk:

```
vsota = od1 + od2 + od3 + od4 + od5 + … +od10 + od11 + od12
```

  - z uporabo tabele od:

```
vsota = 0;
for (int i=0; i<=11; ++i)
vsota = vsota + od [i]
```

  - z indeksom `i` se pomikamo preko vseh elementov in jih prištevamo vsoti

Kreiranje tabele

- kreiranje tabele poteka v dveh korakih
  - najprej moramo tabelo deklarirati
    - navedemo tip tabele in ime tabele
    - tip tabele je enak tipu elementov, le da dodamo par oglatih oklepajev

  - nato dodelimo prostor z operaterjem `new`
    - namesto konstruktorja navedemo tip elementov in v oglatem oklepaju njihovo število

- primer: kreiranje tabele `od`

```
double [] od;              //tip in ime tabele
od = new double [12];     // navedemo število elementov
```

- oba koraka združimo:

```
double [] od = new double [12];
```

- vpisovanje vrednosti v tabelo
  - ob kreiranju tabela ne vsebuje pravih vrednosti
    - po izvršitvi `double [] od`; dobi `od` vrednost `null`
    - po dodelitvi prostora z `od = new double [12];` dobijo vsi elementi vrednost `0`
    - elementi tipa `char` dobijo ob inicializaciji vrednost `'\0000'`, elementi tipa `boolean` pa vrednost `false`
  - za inicializacijo lahko uporabimo seznam vrednosti

```
double [] od = {220815.80, 201234.50, 199410.80, …};
```

    - v zavitem oklepaju naštejemo vrednosti, ločene z vejico
    - operatorja `new` in števila elementov ni treba navesti
    - število elementov, za katere se dodeli prostor, je enako številu vrednosti, ki jih navedemo v seznamu
  - vrednost lahko priredimo vsakemu elementu posebej
    - primer: `od [4] = 215876.40;`
    - stavek `double [] od = {220815.80, 201234.50, 199410.80, …};` ima enak učinek kot stavki

```
od [0] = 220815.80;
od [1] = 201234.50;
od [2] = 199410.80;
…
```

  - največkrat uporabimo zanko

```
for (int i=0; i<=11; ++i)
    od [i] = BranjePodatkov.preberiDouble ();
```

- atribut `length`
  - vsaki tabeli avtomatsko pripada atribut `length`, ki vsebuje dolžino tabele (število elementov v tabeli)
    - ko deklariramo tabelo `double [] od = new double [12];` dobi `od.length` vrednost `12`
    - pri uporabi indeksov moramo vedno paziti, da so v mejah od `0` do `od.length – 1`
    - namesto eksplicitnega navajanja indeksa zadnjega elementa raje uporabljamo atribut `length`

```
for (int i = 0; i <= 11; ++i)
   od [i] = BranjePodatkov.preberiDouble ();
for (int i = 0; i <= od.length – 1; ++i)
   od [i] = BranjePodatkov.preberiDouble ();
```

    - če se spremeni dolžina tabele, ni treba popravljati programske kode

Nekatere operacije nad tabelo

- iskanje elementa v tabeli
  - podana je tabela `t`, ugotoviti želimo, ali se v njej nahaja element `x`

- zaporedno pregledovanje elementov
- binarno iskanje (bisekcija)
  - binarno iskanje lahko uporabimo samo, če je tabela urejena

o zaporedno pregledovanje elementov

```
public static int poisci(int[] t, int x)
{
	int i=0;
	while (i<t.length && t[i]!=x) i++;
	return (i<t.length ? i : -1);
}
```

o binarno iskanje

- poiščemo element, ki se nahaja sredi tabele
- če je element na sredini enak x, je iskanje končano
- če je element na sredini manjši od x, nadaljujemo v gornjem delu tabele
- če je element na sredini večji od x, nadaljujemo v spodnjem delu tabele
- postopek ponavljamo dokler
  - ne najdemo elementa x
  - leva meja tistega dela tabele, ki ga moramo pregledati, ne preseže desne

o časovna kompleksnost

- zaporedno pregledovanje: $T_{\max} = o(n), T_{\exp} = o(n/2)$
- binarno iskanje: $T_{\max} = o(\log_2 n)$

```
public static int poisci(int[] t, int x)
{
	int l=0, d=t.length-1, s;
	while (l<=d)
	{
		s=(l+d)/2;
		if (t[s]==x)
			return s;
		else if (t[s]<x)
			l=s+1;
		else
			d=s-1;
	}
	return -1;
}
```

o

- dodajanje v urejeno tabelo

o napisati želimo program, ki zgradi urejeno tabelo

- program v zanki bere števila in jih sproti vstavlja na ustrezna mesta v tabeli
- primer za vhodne podatke 5, 9, 7, 1, 6



- postopen razvoj programa

```
int[] t=new int[10];
int stevilo;
for (int i=0; i<t.length; ++i)
{
      preberi število;
      poišči mesto, kamor naj se vstavi;
      pripravi prostor;
      vpiši število
}
```

- preberi število

```
System.out.print("Vpisi stevilo:");
stevilo=BranjePodatkov.preberiInt();
```

- poišči mesto, kamor naj se vstavi

```
int j=0;
while ((j<i) && (stevilo>t[j]))
++j;
```

- pripravi prostor

```
for (int k=i-1; k>=j; --k)
t[k+1]=t[k];
```

- vpiši število

```
t[j]=stevilo;
```

- urejanje (sortiranje)
  - sortiranje števil
    - algoritem za sortiranje z navadnim izbiranjem

```
for (i=0; i<=a.length-2; ++i)
{
      poišči najmanjše število izmed a[i]..a[a.length-1];
      zamenjaj a[i] in najmanjše število;
}
```

- prikaz postopka

```
i = 0 23  78  36  12  92  46  15  65
i = 1 12  78  36  23  92  46  15  65
i = 2 12  15  36  23  92  46  78  65
i = 3 12  15  23  36  92  46  78  65
i = 4 12  15  23  36  92  46  78  65
i = 5 12  15  23  36  46  92  78  65
i = 6 12  15  23  36  46  65  78  92
```

- poišči najmanjše število izmed `a[i]..a[a.length-1];`
  - iskanje najmanjšega števila poteka v zanki

- pred vstopom v zanko predpostavimo, da je najmanjše število `a[i]`
  - ❖ zapomnimo si njegovo vrednost (`vMin`) in indeks (`iMin`)
- v zanki pregledamo preostala števila do konca tabele
  - ❖ če naletimo na manjše število, popravimo vrednost trenutnega minimuma in si zapomnimo njegov indeks
- po izvršitvi zanke vsebuje `vMin` vrednost, `iMin` pa indeks najmanjšega števila

```
iMin=i; vMin=a[i];
for (j=i+1; j<=a.length-1; ++j)
if (a[j]<vMin)
{
      iMin=j; vMin=a[j];
}
```

- zamenjaj `a[i]` in najmanjše število
  - najmanjše število se nahaja v elementu z indeksom `iMin`, njegova vrednost pa je shranjena v spremenljivki `vMin`
    - ❖ vrednost `a[i]` prepišemo v `a[iMin]`
    - ❖ vrednost `vMin` shranimo v `a[i]`

```
a[iMin]=a[i];   a[i]=vMin;
```

- sortiranje v padajočem zaporedju
  - namesto najmanjšega iščemo največje število
  - v stavku `if` nadomestimo operator <z operatorjem>

Dvodimenzionalne tabele

- v dvodimenzionalni tabeli so elementi urejeni v vrstice in stolpce
  - vsak element ima dva indeksa: prvi indeks je indeks vrstice, drugi indeks je indeks stolpca

a [0,0]	a [0,1]	a [0,2]	a [0,3]	a [0,4]	a [0,5]
a [1,0]	a [1,1]	a [1,2]	a [1,3]	a [1,4]	a [1,5]
a [2,0]	a [2,1]	a [2,2]	a [2,3]	a [2,4]	a [2,5]

- deklaracija

```
int [] [] a = new int [3] [6];
```

- inicializacija s seznamom vrednosti
  - dvodimenzionalno tabelo si predstavljamo kot enodimenzionalno tabelo, katere elementi (vrstice) so spet enodimenzionalne tabele

```
int [] []  a = {{4, 2, 3, 7, 5, 6},
                {5, 3, 6, 8, 2, 1},
                {6, 9, 0, 3, 1, 2}};
```

- branje podatkov v tabelo
  - potrebujemo 2 zanki
    - zunanja zanka teče po vrsticah
    - notranja zanka teče po stolpcih znotraj ene vrstice

```
for (int vr = 0; vr < a.length; ++vr)
     for (int st = 0; st < a [vr].length; ++st)
        a [vr, st] = BranjePodatkov.preberiInt ();
```

Večdimenzionalne tabele

- število dimenzij v Javi ni omejeno
  - od deklaraciji navedemo toliko dimenzij (toliko parov oglatih oklepajev), kot želimo

- za vsako dimenzijo posebej specificiramo število elementov
- vsak element ima toliko indeksov, kolikor je dimenzij
- za obdelavo vseh elementov potrebujemo toliko zank, kolikor je dimenzij

- primer: tridimenzionalna tabela celih števil

```
int [] [] [] a = new int [4] [6] [3];
for (int i = 0; i < a.length; ++i)
   for (int j = 0; j < a [i].length; ++j)
       for (int k = 0; k < a [i] [j].length; ++k)
           a [i] [j] [k] = BranjePodatkov.preberiInt ();
```

Primeri

- razdalje med točkami
  - izračunati želimo razdalje med n točkami v dvodimenzionalnem prostoru
    - vsaka točka je predstavljena z dvema koordinatama
      $T_i(x_i, y_i)$ $T_j(x_j, y_j)$
    - razdalje računamo po Pitagorovem izreku
      $dx = x_i - x_j, dy = y_i - y_j$
      $r_{ij}^2 = dx^2 + dy^2$
    - predstavitev podatkov v računalniku
      - uporabimo globalne spremenljivke

```
static final int ST_TOCK=5;
static double[] x=new double[ST_TOCK];
static double[] y=new double[ST_TOCK];
static double[][] r=new double[ST_TOCK][ST_TOCK];
```

  - postopen razvoj programa

```
public class Razdalje1
{
	static final int ST_TOCK=5;
	static double[] x=new double[ST_TOCK];
	static double[] y=new double[ST_TOCK];
	static double[][] r=new double[ST_TOCK][ST_TOCK];
	public static void main(String[] args)
	{
		PreberiKoordinate();
		IzracunajRazdalje();
		IzpisiRazdalje();
	}
		// tu deklariramo metode PreberiKoordinate,
		// IzracunajRazdalje in IzpisiRazdalje
}
```

  - branje koordinat

```
public static void PreberiKoordinate()
{
	for (int i=0; i<ST_TOCK; ++i)
	{
		System.out.print("Koordinata x "+(i+1)+". tocke:");
		x[i]=BranjePodatkov.preberiDouble();
		System.out.print("Koordinata y "+(i+1)+". tocke:");
		y[i]=BranjePodatkov.preberiDouble();
	}
}
```

  - izračun razdalj
    - vrednosti elementovna diagonali so enake 0

- tabela je simetrična: vrednost `r[i][j]` je enaka vrednosti `r[j][i]`
- zadostuje, da izračunamo samo vrednosti elementov pod diagonalo

```
public static void IzracunajRazdalje()
{
	double dx,dy;
	for (int i=0; i<ST_TOCK; ++i)
		for (int j=0; j<=i; ++j)
			if (i==j)
				r[i][j]=0;
			else
			{
				dx=x[i]-x[j];
				dy=y[i]-y[j];
				r[i][j]=Math.sqrt(dx*dx+dy*dy);
				r[j][i]=r[i][j];
			}
}
```

- izpis razdalj

```
public static void IzpisiRazdalje()
{
	for (int i=0; i<ST_TOCK; ++i)
	{
		for (int j=0; j<ST_TOCK; ++j)
			System.out.print(r[i][j]);
		System.out.println();
	}
}
```

- rešitev z izmenjavo podatkov preko parametrov
- rešitev s formatiranjem izpisa

- množenje matrik
  - pojem matrike
    - dvodimenzionalna ureditev podatkov
    - `m` vrstic, `n` stolpcev

$$A = \begin{matrix} a_{11} & a_{12} & a_{13} & \dots & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & \dots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \dots & \dots & a_{3n} \\ \dots & & & \dots & \dots & \dots \\ a_{m1} & a_{m2} & a_{m3} & & & a_{mn} \end{matrix}$$

  - za produkt matrik `C = A * B` velja
    - `A` je dimenzije `m x n`
    - `B` je dimenzije `n x p`
    - `C` je dimenzije `m x p`
    - upoštevati moramo, da je v Javi prvi indeks 0

$$\begin{matrix} a_{00} & a_{01} & \dots & a_{0,n-1} \\ a_{10} & a_{11} & \dots & a_{1,n-1} \\ a_{20} & a_{21} & \dots & a_{2,n-1} \\ \dots & & & \dots \\ a_{m-1,0} & a_{m-1,1} & \dots & a_{m-1,n-1} \end{matrix} \qquad \begin{matrix} b_{00} & b_{01} & \dots & b_{0,p-1} \\ b_{10} & b_{11} & \dots & b_{1,p-1} \\ b_{20} & b_{21} & \dots & b_{2,p-1} \\ \dots & & & \dots \\ b_{n-1,0} & b_{n-1,1} & \dots & b_{n-1,p-1} \end{matrix}$$

$$\begin{matrix} c_{00} & c_{01} & \dots & c_{0,p-1} \\ c_{10} & c_{11} & \dots & c_{1,p-1} \\ c_{20} & c_{21} & \dots & c_{2,p-1} \\ \dots & & & \dots \\ c_{m-1,0} & c_{m-1,1} & \dots & c_{m-1,p-1} \end{matrix}$$

formula za izračun enega elementa: $c_{ij} = \sum a_{ik} * b_{kj}$ $k = 0, 1, 2, \ldots, n - 1$

$c_{00} = a_{00} * b_{00} + a_{01} * b_{10} + a_{02} * b_{20} + \ldots + a_{0,n-1} * b_{n-1,0}$
$c_{01} = a_{00} * b_{01} + a_{01} * b_{11} + a_{02} * b_{21} + \ldots + a_{0,n-1} * b_{n-1,1}$
$c_{02} = a_{00} * b_{02} + a_{01} * b_{12} + a_{02} * b_{22} + \ldots + a_{0,n-1} * b_{n-1,2}$ itd

- podatkovne strukture

```
int [] [] a={{2,6,5,5}, {3,8,6,3}, {1,5,0,2}};
int [] [] b={{3,2}, {2,4}, {1,5}, {2,2}};
int [] [] c=new int [a.length] [b [0].length];
```

- izračunati moramo vse elemente matrike C

```
for (i = 0; i <= c.length – 1; ++i)
    for (j = 0; j <= c [0].length – 1; ++j)
    {
      izračunaj c [i] [j];
    }
```

- vsak element izračunamo kot vsoto

```
c [i] [j] = 0;
for (k = 0; k <= vb – 1; ++k)      //vb = b.length;
    c [i] [j] += a [i] [k] * b [k] [j];
```

Tabele objektov

- elementi tabele so lahko tudi objekti
  - primer: tabela objektov tipa `Delavec`

```
Delavec [] td = new Delavec {100};
for (int = 0; i < td.length; ++i)
emp [i] = new Delavec (1000 + i);
```

  - stavek `Delavec [] td = new Delavec [100];` samo rezervira prostor za tabelo, ne kreira pa objektov tipa `Delavec`
  - objekte kreiramo tako, da v zanki kličemo ustrezen konstruktor
    - v našem primeru smo uporabili konstruktor, ki zahteva en argument: matično številko
    - stavek `for` kreira 100 objektov tipa `Delavec` z matičnimi številkami od 1000 do 1099
  - metode, ki pripadajo posameznim objektom, kličemo tako, da navedemo ime tabele, indeks elementa in ime metode, npr.
    `td [4]. vpisiPriimek ("Novak");`
- program za obdelavo tabele delavcev
  - izračunati želimo vrstni red delavcev glede na zaslužek v preteklem letu
    - za vsakega delavca hranimo:
      - matično številko
      - priimek in ime
      - osebne dohodke za 12 mesecev preteklega leta
    - rešitev obsega 2 razreda:
      - razred `Delavec4` opisuje strukturo objektov (atribute, metode, konstruktor)
      - razred `Delavec4Glavni` vsebuje opis postopka
    - celoten problem razdelimo na 4 podprobleme:
      - branje podatkov (kreiranje tabele objektov tipa `Delavec4`)
      - izpis prebranih vrednosti (samo zaradi kontrole vnosa)

- sortiranje tabele glede na seštevek osebnih dohodkov
- izpis vrstnega reda glede na zaslužek v preteklem letu

- razred `Delavec4`
  - deklaracije atributov

```
private int matStev;
private String priimek;
private String ime;
private double[] od;
```

  - deklaracije metod
    - metode za vpis vrednosti posameznih atributov
    - metode, ki vračajo vrednosti posameznih atributov
    - metoda za izračun vsote osebnih dohodkov

```
public double vrniVsotoOd()
{
double vsota=0;
for (int i=0; i<od.length; ++i)
vsota+=od[i];
}
```

  - metoda za izpis vseh atributov objekta

```
public void izpisiVse()
{
      System.out.println("Maticna stevilka: "+matStev);
      System.out.println("Priimek in ime: "+priimek+' '+ime);
      for (int i=0; i<od.length; ++i)
            System.out.println("Osebni dohodek za"+Integer.toString(i+1)+".
mesec:"+od[i]);
}
```

  - konstruktor

```
Delavec4()
{
      System.out.print("Maticna stevilka:");
      matStev=BranjePodatkov.preberiInt();
      System.out.print("Priimek:");
      priimek=BranjePodatkov.preberiString();
      System.out.print("Ime:");
      ime=BranjePodatkov.preberiString();
      od=new double[12];
      for(int i=0; i<od.length; ++i)
      {
            System.out.print("Osebni dohodek za "
                        +Integer.toString(i+1)+". mesec:");
            od[i]=BranjePodatkov.preberiDouble();
      }
}
```

- razred `Delavec4Glavni`
  - branje podatkov (kreiranje tabele objektov)
    - dodelitev prostora za tabelo

```
Delavec4[] td=new Delavec4[ST_DEL];
```

    - kreiranje objektov

```
for (int i=0; i<ST_DEL; ++i)
td[i]=new Delavec4();
```

  - izpis podatkov

```
static void izpisi(Delavec4[] td)
{
      for (int i=0; i<ST_DEL; ++i)
```

```
	{
		System.out.print(td[i].vrniMatSt());
		System.out.print(" "+td[i].vrniPriimek()+" "+
				td[i].vrniIme());
		System.out.println(" "+td[i].vrniVsotoOd());
	}
}
```

- sortiranje tabele

```
static void sortiraj(Delavec4[] a)
{
	int i,j,iMin;
	Delavec4 vMin;
	for (i=0; i<=a.length-2; ++i)
	{
		iMin=i; vMin=a[i];
		for (j=i+1; j<=a.length-1; ++j)
			if (a[j].vrniVsotoOd()>vMin.vrniVsotoOd())
			{
				iMin=j; vMin=a[j];
			}
		a[iMin]=a[i]; a[i]=vMin;
	}
}
```

- metoda `main()` odraža razdelitev na podprobleme

```
public static void main(String[] args)
{
	Delavec4[] td=new Delavec4[ST_DEL];
	for (int i=0; i<ST_DEL; ++i)
		td[i]=new Delavec4();
	System.out.println();
	System.out.println("Pred sortiranjem:");
	izpisi(td);
	sortiraj(td);
	System.out.println();
	System.out.println("Po sortiranju:");
	izpisi(td);
}
```

8. Nizi

Kazalo:

- Koncept niza
  - tabela, katere elementi so znaki
- Razred `String`
  - nespremenljivost nizov
  - primerjanje nizov: metodi `equals()` in `compareTo()`
  - pregled ostalih metod
  - primer: program za ugibanje pregovora
  - pretvorba nizov, ki predstavljajo števila, v numerične vrednosti
- Razred `StringBuffer`
  - omogoča spreminjanje vsebine nizov
  - pregled metod
- Argumenti metode `main()`

Koncept niza

- niz je zaporedje znakov (npr. beseda ali celo več besed skupaj)
- v Javi lahko nize predstavimo kot objekte razreda `String`
- razred `String` je definiran v paketu `Java.lang`

Razred String

- nespremenljivost nizov
  - deklaracija niza
    - `String niz;`
      `niz = new String ("Dober dan");`
    - `String niz = new String ("Dober dan");`
    - `String niz = "Dober dan";`
    - v vseh treh primerih se generira niz z vrednostjo `"Dober dan"`
    - spremenljivka `niz` vsebuje naslov tega niza
  - spreminjanje vrednosti
    - upoštevati je treba, da spremenljivke tipa `String` vsebujejo naslove, ne pa vrednosti objektov
    - primer:

```
String pozdrav = "Dober dan";
…
pozdrav = "Hello";
```

    - stavek `pozdrav = "Hello";` generira nov niz (tj. nov objekt), katerega naslov se shrani v `pozdrav`
      - niz `"Dober dan"` ostane v pomnilniku, vendar ni več dostopen
      - prostor, ki ga zaseda, se sprosti avtomatsko s pomočjo programa za čiščenje pomnilnika (ang. garbagge collector)

- nizi torej nikoli ne spremenijo vrednosti; namesto tega se kreirajo novi objekti, katerih naslovi se shranijo v spremenljivke tipa `String`

- primerjanje nizov: metodi `equals()` in `compareTo()`
  - primerjanje dveh spremenljivk tipa String pomeni dejansko primerjanje naslovov dveh nizov

```
String niz1 = "Dober dan";
String niz2 = "Dober dan";
//rezultat primerjave (niz1 == niz2) je false
```

  - primerjanje je možno z metodo `equals ()`

```
if (niz1.equals (niz2))
  System.out.println ("Niza sta enaka");
if (niz2.equals (niz1))
  System.out.println ("Niza sta enaka");
if (niz1.equals ("Dober dan"))
  System.out.println ("Niza sta enaka");
```

    - v vseh treh stavkih `if` je vrednost pogoja `true`
  - ostale metode za primerjanje nizov
    - metoda `equalsIgnoreCase ()` je podobna `equals ()`, le da ne razlikuje med velikimi in malimi črkami

```
String niz1 ="Dober dan";
String niz2 ="DOBER dan";
```

      rezultat primerjave `(niz1.equals (niz2))` je `false`
      rezultat primerjave `(niz1.equalsIgnoreCase (niz2))` je `true`
    - metoda `compareTo ()` vrne celo število
      - če sta oba niza enaka, vrne 0
      - če je niz, iz katerega je bila metoda poklicana, večji, vrne pozitivno vrednost
      - če je niz, iz katerega je bila metoda poklicana, manjši, vrne negativno vrednost
      - metoda `compareTo ()` deluje tako, da primerja kode posameznih znakov
  - primer uporabe metode `compareTo ()`

```
String niz1= "Marko";
String niz2 = "Martin";
System.out.println (niz1.compareTo (niz2));          // -9
System.out.println (niz2.compareTo (niz1));          // 9
System.out.println (niz1.compareTo ("Marko");        // 0
```

    - metoda primerja istoležne pare znakov, dokler ne naleti na dva različna znaka
      - na prvih treh mestih (znaki `'M'`, `'a'` in `'r'`) se oba niza ujemata
      - razlika nastopi na četrtem mestu, kjer je koda znak `'k'` za 9 manjša od kode znaka `'t'`
    - v praksi (npr. pri urejanju nizov po abecedi) nas ne zanima točna vrednost, ki jo vrne metoda `compareTo ()`
      - zadostuje, da vemo, ali je vrednost negativna ali pozitivna

- pregled ostalih metod (prikazali jih bomo ob predpostavki, da smo deklarirali spremenljivko `String niz = "Dobro jutro";`)
  - `toUpperCase ()`
    - pretvori vse male črke v nizu v velike

        - klicu `niz.toUpperCase ()` vrne vrednost `"DOBRO JUTRO"`
    - `toLowerCase ()`
    - `indexOf (zn)`
        - vrne pozicijo znaka `zn` znotraj nekega niza (indeksi tečejo od 0 dalje)
        - če niz ne vsebuje znaka `zn`, vrne -1
        - klic `niz.indexOf ('b')` vrne vrednost 2
        - klic `niz.indexOf ('v')` vrne vrednost -1
    - `indexOf (zn, poz)`
        - išče znak `zn` od pozicije `poz` dalje
        - klic `niz.indexOf ('o')` vrne vrednost 1
        - klic `niz.indexOf ('o', 2)` vrne vrednost 4
    - `charAt (n)`
        - vrne znak, ki se nahaja na n-tem mestu v nizu
        - klic `niz.charAt (2)` vrne vrednost `'b'`
    - `startsWith (niz1)`
        - vrne `true`, če se niz začne s podnizom `niz1`
        - klic `niz.startsWith ("Dob")` vrne `true`, `niz.startsWith ("Sl")` pa `false`
    - `endsWith (niz1)`
        - vrne `true`, če se niz konča s podnizom `niz1`
        - klic `niz.endsWith ("tro")` vrne `true`, `niz.endsWith ("ber")` pa `false`
    - `replace ()`
        - zamenja vsa nastopanja znaka `zn1` z znakom `zn2`
        - klic `niz.replace ('o', 'x');` vrne vrednost `"Dxbrx jutrx"`
    - `length ()`
        - vrne dolžino niza (število znakov)
        - klic `niz.length ()` vrne vrednost 11
    - `substring (zac, kon)`
        - vrne podniz, ki se prične na mestu `zac` in konča na mestu `kon -1`
        - klic `niz.substring (2, 7)` vrne vrednost `"bro j"`
    - `toString (arg)`
        - pretvori v niz argument `arg`, ki je lahko kateregakoli osnovnega tipa
        - ni metoda razreda `String`
        - ```
          int a = 500;
          String niz = Integer.toString (a);   //"500"
          ```
        - ```
          double x = 32.45;
          String niz = Double.toString (x);    //"32.45"
          ```
        - ```
          boolean b = false;
          String niz3 = Boolean.toString (b);   //"false"
          ```
    - metoda `toString()` je implicitno prisotna pri konkatenaciji nizov: `System.out.print ("Vrednost x je " +x);`
        - vrednost spremenljivke `x` se najprej pretvori v niz in nato konkatenira

- primer: program za ugibanje pregovora
    - napisati želimo program, ki bo v zanki spraševal za posamezne črke, dokler uporabnik ne ugane vseh črk
        - potrebujemo dva niza:
          ```

- prvi vsebuje besedilo pregovora: `iskaniNiz`
- drugi vsebuje zvezdice, presledke in ločila: `prikazaniNiz`
  - ❖ zvezdice so na tistih mestih, kjer sicer nastopijo črke

▪ po vsaki vtipkani črki
- preverimo, ali `iskaniNiz` vsebuje vtipkano črko
- če črke ni, samo izpišemo ustrezno sporočilo
- če črko najdemo, nadomestimo zvezdice pri tistih mestih, kjer nastopa vtipkana črka, in ponovno prikažemo `prikazaniNiz`

▪ postopek zaključimo, ko `prikazaniNiz` ne vsebuje več nobene zvezdice

o prva verzija programa

```
String iskaniNiz="Rana ura, zlata ura.";
String prikazaniNiz="**** ***, ***** ***.";
char crka;      // vtipkana crka
int poz;        // pozicija vtipkane crke v iskaniNiz
System.out.println(prikazaniNiz);
while (prikazaniNiz.indexOf('*') != -1)
{
      System.out.print("Vtipkaj crko:");
      crka=(char)System.in.read();
      System.in.read(); System.in.read(); // Enter
      poz=iskaniNiz.indexOf(crka);
      if (poz == -1)
            System.out.println("Te crke ni, ugibaj ponovno!");
      else
      {
            //nadomesti ustrezne zvezdice z vtipkano črko;
            System.out.println(prikazaniNiz);
      }
}
```

o nadomesti ustrezne zvezdice z vtipkano črko

▪ vtipkana črka lahko nastopa večkrat, zato je potrebna zanka

▪ vtipkano črko iščemo v `iskaniNiz`, zvezdico pa zamenjamo na isti poziciji v `prikazaniNiz`

▪ novo vrednost `prikazaniNiz` sestavimo iz 3 delov
- prvi del: znaki pred vtipkano črko
- drugi del: vtipkana črka
- tretji del: znaki, ki sledijo vtipkani črki

▪ naslednjo črko iščemo od `poz+1` dalje

```
do
{
      prikazaniNiz=prikazaniNiz.substring(0,poz)+crka+
      prikazaniNiz.substring(poz+1,prikazaniNiz.length());
      poz=iskaniNiz.indexOf(crka,poz+1);
} while (poz != -1);
```

- pretvorba nizov, ki predstavljajo števila, v numerične vrednosti

o nize, ki so sestavljeni iz samih cifer (decimalne pike, predznaka), lahko pretvorimo v števila

▪ pretvorbo niza v celo število omogoča metoda `parseInt ()` iz razreda `Integer`

```
String niz = "54872";
 int celoStevilo = Integer.parseInt (niz);
```

- pretvorno niza v realno število omogoča metoda `parseDouble ()` iz razreda `Double`

```
String niz = "54.872";
 double realnoStevilo = Double.parseDouble (niz);
```

- podobne metode so na razpolago v ostalih razredih
  - `Byte.pasreByte (niz)`
  - `Short.parseShort (niz)`
  - `Long.parseLong (niz)`
  - `Float.parseFloat (niz)`

Razred `StringBuffer`

- podoben razredu `String`, le da omogoča spreminjanje nizov in njihove dolžine
  - niz tipa `String` je nespremenljiv; interna predstavitev obsega toliko znakov, kot znaša dolžina niza
  - niz tipa `StringBuffer` je shranjen v ti. vmesniku (ang. buffer), katerega dolžina (ang. capacity) se dinamično spreminja
    - metoda `capacity ()` vrne trenutno kapaciteto
    - metoda `length ()` vrne trenutno dolžino niza
    - metoda `setLength ()` omogoča nastavljanje dolžine niza
- deklaracija niza: uporabimo lahko 3 konstruktorje
  - `StringBuffer ()` generira niz brez znakov s kapaciteto 16
  - `StringBuffer (String s)` generira niz, ki vsebuje znake niza `s`, kapaciteta pa je za 16 večja od dolžine niza `s`
  - `StringBuffer (int k)` generira prazen niz s kapaciteto `k`
- spreminjanje vsebine niza
  - metoda `append ()` omogoča dodajanje na koncu niza
    - `append (niz2)` doda na konec `niz2` tipa `String`
    - `append (zn)` doda na konec znak `zn` tipa `char`
    - metoda je večkratno definirana in sprejme tudi parametre drugih tipov, npr. `int`
  - metoda `insert ()` omogoča vrivanje sredi niza
    - `insert (poz, niz2)` vrine `niz2` na pozicijo `poz`
    - `insert (poz, zn)` vrine znak `zn` na pozicijo `poz`
  - pri dodajanju z metodo `append ()` in vrivanju z metodo `insert ()` se kapaciteta avtomatsko poveča
  - metoda `delete (zac, kon)` briše znake med pozicijama `zac` in `kon-1`
  - metoda `deleteCharAt (poz)` briše znak na poziciji `poz`
  - metoda `setCharAt (poz, zn)` vpiše znak `zn` na pozicijo `poz`
  - metoda `replace (zac, kon, niz2)` briše znake med pozicijama `zac` in `kon-1` ter jih nadomesti z nizom `niz2`
- branje podatkov iz niza
  - metoda `charAt (poz)` vrne znak, ki se nahaja na poziciji `poz`
  - metoda `substring (zac, kon)` vrne podniz, ki vsebuje znake na pozicijah od `zac` do `kon -1`
- iskanje podatkov v nizu
  - metoda `indexOf (niz2)` vrne pozicijo, kjer se prične `niz2`

        - parameter `niz2` je tipa `String`
        - če niza `niz2` ne najde, vrne `-1`
    - metoda `indexOf (niz2, poz)` išče `niz2` od pozicije `poz` dalje

Argumenti metode `main ()`

- `(String [] args)` je deklaracija argumentov
    - argumenti so podatki, ki jih metoda potrebuje za svoje delo
    - tudi če metoda `main` ne potrebuje argumentov, morajo biti deklarirani
    - `String` pove, kakšnega tipa so argumenti: nizi znakov
    - `args` je skupno ime za vse argumente
    - oglata oklepaja označujeta, da argumenti tvorijo tabelo nizov
- argumente navedemo ob zagonu programa iz ukazne vrstice

```
java <ime razreda> <arg1> <arg2> … <argn>
```

    - dodamo tabelo `args`, v kateri so shranjene vrednosti posameznih argumentov
    - `args [0]` vsebuje `<arg1>`, `args [1]` vsebuje `<arg2>` itd.

9. Dedovanje

Kazalo:

- 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`
- 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

Koncept dedovanja

- 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)
    - 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)
- 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)
- 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 in podrazredu

Redefinicija 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 (overriding) in večkratnim definiranjem metod (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
- 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>`
- 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);
}
```

- 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

Uporaba konstruktorjev

- ob kreiranju objekta, ki pripada nekemu podrazredu, se dejansko kličeta dva konstruktorja
    - konstruktor osnovnega razreda
    - konstruktor podrazreda
    - vedno se najprej izvede konstruktor nadrazreda, nato konstruktor podrazreda
    - v našem primeru: najprej konstruktor `Student ()`, nato konstruktor `IzredniStudent ()`
- splošno pravilo pri pisanju lastnih konstruktorjev
    - konstruktor nadrazreda naj poskrbi za inicializacijo atributov, ki so deklarirani v nadrazredu
    - konstruktor podrazreda naj poskrbi za inicializacijo atributov, ki so deklarirani v podrazredu
- konstruktorji z argumenti
    - izhodišče: kreirati želimo objekt podrazreda, konstruktor nadrazreda pa zahteva argumente
    - podrazred mora poskrbeti, da dobi konstruktor nadrazreda ustrezne argumente
    - obvezno je treba napisati konstruktor podrazreda, ki pokliče konstruktor nadrazreda z ustreznimi argumenti

        - klic konstruktorja nadrazreda se izvede z rezervirano besedo `super` in ne z imenom konstruktorja
        - stavek `super (<seznam argumentov>)`; mora biti prvi stavek v konstruktorju podrazreda (niti deklaracije spremenljivk ne smejo biti pred njim)
    - primer: če konstruktor nadrazreda `Student` zahteva tri argumente, ga pokličemo s `super (63020888, "Novak", "Janez");`

Dostopno določilo `protected`

- koncept skrivanja informacij (doslej)
    - atributi razreda so deklarirani z dostopnim določilom `private`
    - metode razreda so deklarirane z dostopnim določilom `public`
    - ostali razredi lahko dostopajo do atributov samo preko metod
- posledice pri dedovanju
    - podrazredi nimajo direktnega dostopa do podedovanih atributov
    - do njih lahko dostopajo samo preko metod
- rešitev: uporaba dostopnega določila `protected`
    - vmesna stopnja zaščite med `public` in `private`
    - podrazredom dovoljuje neposreden dostop do atributov (in metod) nadrazreda
    - ostalim razredom (ki niso izpeljani iz nadrazreda) dostop do `protected` atributov in metod ni dovoljen

Metode, ki jih ni moč redefinirati

- obstajajo 4 tipi metod, 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 `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`

Abstraktni razred in abstraktne metode

- abstraktni razred je nek splošen nadrazred, ki predstavlja osnovo za izpeljavo različnih podrazredov
    - deklariramo ga izključno z namenom, da bomo iz njega izpeljali različne metode
    - ne moremo generirati objektov tega razreda, ampak samo objekte podrazredov
    - po zgradbi je podoben ostalim razredom (vsebuje atribute in metode) s tem da je ena ali več metod abstraktnih
    - abstraktne razrede in abstraktne metode deklariramo z rezervirano besedo `abstract`, npr.

```
public abstract class Zival
public abstract void oglasanje ()
```

- abstraktna metoda ima samo glavo brez stavkov
    - jo, kadar na nivoju nadrazreda ne moremo opisati operacije, ki jo ta metoda izvede, vendar želimo doseči, da to metodo vsebujejo vsi podrazredi
    - v podrazredu je treba abstraktno metodo redefinirati
    - če podrazred ne redefinira abstraktne metode, je tudi podrazred abstrakten
    - če je razred abstrakten, je prazna metoda vedno abstraktna, četudi izpustimo rezervirano besedo `abstract`
- primer: abstraktni razred `Zival`
    - služi kot osnova za izpeljavo podrazredov `Pes, Krava` in `Kaca`
    - vsebuje abstraktno metodo `oglasanje (),` ki pove kako se žival oglaša
- vsak objekt podrazreda je istočasno tudi objekt nadrazreda
    - `Pes` je `Zival`, `Avto` je `Vozilo` ("is a")
    - obratno ni možno
- posledica: spremenljivki tipa nadrazred lahko priredimo naslov objekta, ki pripada kateremukoli podrazredu
    - imamo spremenljivko z, ki je deklarirana kot `Zival z`;
    - imamo 3 objekte, ki pripadajo razredom `Pes, Krava` in `Kaca`

```
Pes p=new Pes ("Luks");
Krava kr=new Krava ("Liska");
Kaca ka=new Kaca ("Klopotaca");
```

    - potem so dovoljeni naslednji prireditveni stavki `z=p; z=kr; z=ka;`
    - vprašanje: katera metoda se izvede ob klicu `z.oglasanje ()`
        - odgovor: izvede se metoda tistega podrazreda, katerega objekt je trenutno shranjen v spremenljivki `z`

Dinamično povezovanje metod

- je sposobnost programa, da izbere metodo, ki pripada pravemu podrazredu
- metoda se izbere med izvajanjem programa, odvisno od dejanskega tipa objekta
- pojem statičnega in dinamičnega tipa
    - statični tip je naveden ob deklaraciji (spremenljivka z je tipa `Zival`)
    - dinamični tip je določen med izvajanjem programa (spremenljivka `z` je lahko dinamičnega tipa `Pes, Krava` ali `Kaca`)
- dinamični tip spremenljivke z določa, katera metoda `oglasanje ()` se bo izvedla
- tabele objektov, ki pripadajo različnim podrazredom

- dedovanje omogoča, da v tabeli hranimo objekte, ki pripadajo različnim podrazredom
  - zahteva: vsi objekti v tabeli morajo biti istega tipa
  - tabelo deklariramo kot tabelo objektov, ki pripadajo osnovnemu razredu
  - v resnici generiramo objekte, ki pripadajo posameznim podrazredom
  - ko so objekti vstavljeni v tabelo, jih lahko obdelujemo v zanki, kot da bi bili vsi istega tipa
  - edina omejitev: vsi podrazredi morajo biti izpeljani iz istega osnovnega razreda
- primer: tabela podatkov o živalih `Zival []z=new Zival [10];`

Razred `Object` in njegove metode

- razred `Object` – univerzalni nadrazred
  - je osnovni razred, deklariran v paketu `java.lang`, iz katerega so (neposredno ali posredno) izpeljani vsi drugi razredi
    - vsak razred v Javi (razen razreda `Object`) je v resnici podrazred
    - če razred ni deklariran kot razširitev nekega nadrazreda, se avtomatsko privzame, da je razširitev razreda `Object`
  - vsebuje metode, ki jih avtomatsko podedujejo vsi podrazredi
- metoda `toString ()`
  - pretvori vsebino objekta v niz tipa `String`, ki vsebuje
    - ime razreda, kateremu pripada objekt
    - naslov, kjer je ta objekt shranjen v pomnilnik, npr. `@ba34f2`
  - pri delu z objekti je priporočljivo to metodo redefinirati
    - vrne naj vrednost atributov kot en sam niz znakov
    - ta niz lahko kasneje izpišemo
- metoda `equals ()`
  - zahteva en argument, ki mora biti istega tipa kot objekt, iz katerega je bila metoda poklicana: `<ime objekta>.equals(<imeDrugegaObjekta>)`
  - če metode ne redefiniramo, velja, da sta objekta enaka, če imata enak naslov (preverja se enakost naslovov, ne enakost vsebine)
  - primer 1:

```
Pes pes1=new Pes ("Fifi");
Pes pes2=new Pes ("Fifi");
pes1.equals (pes2) vrne false
```

  - primer 2:

```
StringBuffer niz1=new StringBuffer ("Fifi");
StringBuffer niz2=new StringBuffer ("Fifi");
niz1.equals (niz2) vrne false
```

- redefinicija metode `equals ()`
  - če s podedovano metodo `equals ()` nismo zadovoljni, napišemo svojo metodo z enakim seznamom argumentov in istim tipom rezultata
  - za primer 1:

```
public boolean equals (Pes p2)
{
     if (vrniIme().equals (p2.vrniIme()))
         return true;
```

```
      else
          return false;
}
```

  - krajše: `return vrniIme ().equals (p2.vrniIme())`
  - uporaba `this`: `this.vrniIme().equals (p2.vrniIme())`
- kam vkomponirati redefinirano metodo `equals ()`
  - v razred `Pes`: redefinirana metoda velja samo za pse
    - dva psa sta enaka, če imata enako ime
    - za ostale živali velja, da sta dva objekta enaka, če imata enak naslov
  - v razred `Zival`: redefinirana metoda velja za vse živali
    - katerikoli dve živali sta enaki, če imata enako ime
- druge metode razreda `Object`
  - `getClass ()`: vrne objekt tipa `Class`, ki vsebuje ime razreda
  - `hashCode ()`: izračuna hash kodo (za shranjevanje objektov v razpršenih tabelah)
  - `nitify (), notifyAll (), wait ()`: pri delu z nitmi (ang. threads)
  - `clone ()`: za kopiranje objekta
  - `finalize ()`: metoda, ki se izvede ob uničenju objekta

Koncept vmesnika kot nadomestek za večkratno dedovanje (vmesnik – ang. interface)

- 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 eneka 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
  - vse metode morajo biti abstraktne
  - vsi atributi (če jih ima), morajo biti `static final`
- z vmesnikom predpišemo metode, ki jih mora implementirati podrazred (določimo obnašanje podrazreda)
- 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
- 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
- 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

10. Uvod v delo z grafiko

Kazalo:

- Knjižnici AWT in Swing
- Kreiranje okna
  - okno kot podrazred razreda `JFrame`
- Zapiranje okna
  - dogodkovni model: razredi dogodkov, vmesniki, adapterji, poslušalci
- Upoštevanje karakteristik uporabnikovega računalnika
- Risanje
- Pregled metod za risanje

Knjižnici AWT in Swing

- dve knjižnici (paketa) razredov za programiranje grafičnega uporabniškega vmesnika (GUI)
  - ATW (Abstract Window Toolkit)
    - starejša (prvotna) knjižnica
    - realizacija komponent prepuščena ciljni plaformi
    - problem: različno obnašanje na različnih platformah
  - Swing
    - novejša knjižnica
    - večji nabor komponent GUI
    - enostavnejša uporaba
    - manjša odvisnost od ciljne platforme
    - enak videz na vseh platformah
  - Swing še vedno uporablja dogodkovno voden način programiranja, ki ga definira ATW
    - v programih kombiniramo uporabo obeh knjižnic

Kreiranje okna

- osnovno okno, v katerem rišemo, je objekt razreda `JFrame`
  - v paketu AWT obstaja razred `Frame`
  - razred `JFrame` je razširitev razreda `Frame`
  - osnovno okno je ena redkih komponent, ki jih ne nariše Swing, ampak okenski sistem ciljne platforme
  - osnovno okno he kontejner (ang, container), ki lahko vsebuje druge komponente grafičnega uporabniškega vmesnika (gumbi, tekstovna polja…)
- program za kreiranje okna mora vsebovati
  - napoved uporabe paketa `Swing:` `import javax.swing.*;`
  - kreiranje objekta tipa `JFrame:` `JFrame okno=new JFrame();`
  - prikaz okna na zaslonu: `okno.show();`
- boljša rešitev
  - iz razreda `JFrame` izpeljemo podrazred, ki natančneje definira lastnosti okna
  - v podrazredu deklariramo konstruktor, ki vsebuje ukaze, kot so npr.

```
setTitle (<naslov>);                              //ime
setSize (<širina>, <višina>);                     //velikost
setLocation (<x>, <y>);                           //lega
setBounds (<x>, <y>, <širina>, <višina>);         //velikost in lega
setResizable (<vrednostBoolean>);                 //spreminjanje velikosti
```

- problem
  - ko okno zapremo, se izvajane programa ne prekine
  - za prekinitev je potrebno vtipkati CTRL+C

Zapiranje okna (prekinitev izvajanja programa ob zapiranju okna)

- upoštevati je treba ATW dogodkovni model
  - dogodek je sprememba v stanju določene komponente zaradi interakcije s strani uporabnika (npr. klik z miško, pritisk tipke na tipkovnici, vpis podatka v polje za vnos ipd.)
  - vsaka komponenta GUI, s katero je uporabnik v interakciji, mora vsebovati poseben objekt, ki nastopa v vlogi poslušalca
  - naloga poslušalca je, da prestreže določen dogodek, in izvrši primerno akcijo
- v javi so dogodki razvrščeni v več razredov
  - primeri: `KeyEvent, MouseEvent, ActionEvent, WindowEvent`
  - za vsak tip dogodka (razred) lahko dodamo ustreznega poslušalca
  - v našem primeru potrebujemo poslušalca za dogodke tipa `WindowEvent`
- pregled dogodkov, vmesnikov in odzivnih metod

tip dogodka	**vmesnik**	**odzivne metode**
`ActionEvent`	`ActionListener`	`actionPerformed (ActionEvent e)`
`ItemEvent`	`ItemListener`	`itemStateChanged (ItemEvent e)`
`TextEvent`	`TextListener`	`textValueChanged (ActionEvent e)`
`AdjustmentEvent`	`AdjustmentListener`	`adjustmentValueChanged (AdjustmentEvent e)`
`ContainerEvent`	`ContainerListener`	`componentAdded (ContainerEvent e)` `componentRemoved (ContainerEvent e)`
`ComponentEvent`	`ComponentListener`	`componentMoved (ComponentEvent e)` `componentHidden (ComponentEvent e)`
`FocusEvent`	`FocusListener`	`focusGained (FocusEvent e)` `focusLost (FocusEvent e)`

`MouseEvent`	`MouseListener`
	
	`MouseMotionListener`
		`mousePressed (MoiseEvent e)`
		`mouseReleased (MoiseEvent e)`
		`mouseEntered (MoiseEvent e)`
		`mouseExited (MoiseEvent e)`
		`mouseClicked (MoiseEvent e)`
		`mouseDragged (MoiseEvent e)`
		`mouseMoved (MoiseEvent e)`

`KeyEvent`	`KeyListener`	`keyPressed (KeyEvent e)`
		`keyTyped (KeyEvent e)`
		`keyReleased (KeyEvent e)`

`WindowEvent`	`WindowListener`	`windowActivated (WindowEvent e)`
		`windowClosing (WindowEvent e)`
		`windowClosed (WindowEvent e)`
		`windowDeactivated (WindowEvent e)`
		`windowDeiconified (WindowEvent e)`
		`windowIconified (WindowEvent e)`
		`windowOpened (WindowEvent e)`

- pregled metod za dodajanje poslušalcev

komponente (izvori dogodkov)	**metode za dodajanje poslušalcev**

JButton, JCheckBox, JComboBox, JToolBar, JTextField, JRadioButton	addActionListener()
JScrollBar	addAdjustmentListener()
Vse Swing komponente	addFocusListener(), addKeyListener(), addMouseListener(), addMouseMotionListener()
JButton, JCheckBox, JComboBox, JRadioButton	addItemListener()
JWindow, JFrame	addWindowListener()
JSlider	addChangeEvent()

- deklaracija poslušalca
  - poslušalec mora pripadati razredu, v katerem so implementirane metode za določeno vrsto dogodkov
  - za dogodke tipa `WindowEvent` se ta razred imenuje `WindowAdapter` in je implementacija vmesnika `WindowListener`
  - vmesnik `WindowListener` predpisuje 7 metod, ki ustrezajo posameznim dogodkom tipa `WindowEvent`

```
public void windowActivated (WindowEvent e)
public void windowClosed (WindowEvent e)
public void windowClosing (WindowEvent e)
public void windowDeactivated (WindowEvent e)
public void windowDeiconified (WindowEvent e)
public void windowIconified (WindowEvent e)
public void windowOpened (WindowEvent e)
```

  - v razredu `WindowAdapter` so te metode implementirane tako, da ob dogodku ne izvršijo nobene akcije
  - iz razreda `WindowAdapter` izpeljemo podrazred, ki redefinira metodo `WindowClosing`

```
class Poslusalec extends WindowAdapter
{
       Public void windowClosing (WindowEvent e)
       {
              System.exit (0);
       }
}
```

  - v konstruktorju za okno deklariramo poslušalca

```
WindowListener p=new Poslusalec ();
add WindowListener (p);
```

- uporaba metode `setDefaultCloseOperation()`
  - s pomočjo metode `setDefaultCloseOperation()` lahko določimo operacijo, ki se izvede ob zapiranju okna
  - operacijo podamo kot argument ob klicu metode
  - obstajajo vnaprej deklarirane konstante, ki določajo posamezne operacije
    - `DO_NOTHING_ON_CLOSE`: ne naredi nič
    - `HIDE_ON_CLOSE`: zapre okno, program teče naprej
    - `DISPOSE_ON_CLOSE`: zapre okno in uniči objekt, ki predstavlja okno; program teče naprej
    - `EXIT_ON_CLOSE`: prekine izvajanje programa
  - če ne določimo operacije ob zapiranju, se avtomatsko privzame `HIDE_ON_CLOSE`

Upoštevanje karakteristik uporabnikovega računalnika

- razred `Toolkit` iz paketa `java.awt.*`
  - metoda `getDefaultToolkit ()` vrne objekt tipa `Toolkit`
  - ugotovimo lahko velikost zaslona

```
Toolkit tk=Toolkit.getDefaultToolkit();
Dimension d=tk.getScreenSize();
int sirina=d.width;
int visina=d.height;
```

  - ob odpiranju okna upoštevamo dejansko širino in višino zaslona
  - primer za centrirano okno, katerega širina in višina je enaka polovici širine in višine zaslona

```
setSize (sirina/2, visina/2);
setLocation (sirina/4, visina/4);
```

Risanje

- vsebina okna je določena s ploščo *content pane*
  - v konstruktorju za okna je treba
    - deklarirati ploščo *content pane*
    - deklarirati komponento, ki jo želimo narisati na plošči
    - dodati komponento na ploščo
  - če hočemo v oknu risati različne like (črke, pravokotnike, kroge ipd.), je treba deklarirati komponento tipa `Jpanel`

```
Container vsebina = getContentPane ();
JPanel panel1=new Panel1();
vsebina.add (panel1);
```

  - rišemo dejansko pa panel
    - panel ima risalno površino, na katero rišemo
    - je kontejner (vsebuje lahko druge komponente)
    - podobno kot ostale komponente vsebuje metodo `paintComponent`
- risanje na panel
  - deklarirati je treba podrazred razreda `JPanel`
  - redefinirati je treba metodo `paintComponent (g Graphics)`
    - risanje realiziramo s klici ustreznih metod razreda `Graphics`
    - obvezen je klic istoimenske metode nadrazreda
    - metode `paintComponent` ni treba klicati, ampak se izvede avtomatsko
- primer deklaracije panela, v katerem se nariše črta

```
class Panel1 extends JPanel
{
    public void paintComponent (Graphics g)
     {
          super.paintComponent (g);
          g.drawLine (10, 10, 100, 70);
     }
}
```

Pregled metod za risanje

- "Risanje" nizov
  - `drawString (niz, x, y)`
    - izpiše `niz` tako, da je prvi znak niza odmaknjen od gornjega levega oglišča panela za `x` točk v desno in `y` točk navzdol
  - `setFont (f)`

        - nastavi obliko črk v skladu s parametrom `f`
        - `f` objekt je tipa `Font` in mora biti prej določen, npr.

```
Font f = new Font ("SansSerif", Font.BOLD, 15);
```

        - logična imena fontov v AWT: `SansSerif, Serif, Monospaced, Dialog, DialogInput`
        - način izpisa (ang. style): `Font.PLAIN, Font.BOLD, Font.ITALIC, Font.BOLD+Font.ITALIC`
    - `stringWidth (niz)`
        - metoda razreda `FontMetrics`, ki za font `f` vrne dolžino niza

```
FontMetrics fm=getFontMetrics (f);
int dolzina=fm.stringWidth(niz);
```

- risanje črt, lokov in mnogokotnikov
    - `drawLine (x1, y1, x2, y2);`
        - nariše črto med točkama `(x1, y1)` in `(x2, y2)`
    - `drawArc (x, y, sirina, visina, zacKot, kot)`
        - nariše lok, ki se nahaja znotraj navideznega pravokotnika z levim zgornjim ogliščem v točki `(x, y)` in stranicama `a=sirina, b=visina`
        - lok se prične pri kotu `zacKot` in oklepa kot `kot` (tj. se konča pri `zacKot+kot`); koti so podani v stopinjah
    - `drawPolygon (p)`
        - nariše mnogokotnik, katerega stranice so določene s točkami objekta `p`
        - primer za risanje trikotnika

```
Polygon p=new Polygon();
p.addPoint (10, 10);
p.addPoint (10, 30);
p.addPoint (20, 20);
g.drawPolygon (p);
```

    - `drawPolygon (x, y, n)`
        - nariše mnogokotnik, katerega oglišča so podana s koordinatami točk v tabelah `x` in `y`
        - parameter `n` določa število oglišč
    - `drawPolyline (x, y, n)`
        - nariše lomljeno črto, ki povezuje točke, katerih koordinate so v tabelah `x` in `y`
        - `n` je število točk, ki jih je treba povezati
        - če sta prva in zadnja točka identični, je črta zaključena
- risanje pravokotnikov, krogov in elips
    - `drawRect (x, y, sirina, visina)`
        - nariše pravokotnik z levim zgornjim ogliščem v točki `(x, y)` in stranicama `a=sirina, b=visina`
    - `drawRoundRect (x, y, sirina, visina, rH, rV)`
        - nariše pravokotnik z zaobljenimi oglišči
        - `rH` in `rV` določata horizontalni in vertikalni polmer loka
    - `draw3DRect (x, y, sirina, visina, dvig)`
        - nariše pravokotnik, ki daje vtis gumba
        - če je parameter `dvig` enak `true`, je pravokotnik "dvignjen" nad površino okna, sicer pa "ugreznjen"

        - učinek postane viden, če narišemo več pravokotnikov, katerih stranice se povečujejo za eno piko, koordinate levega oglišča pa zmanjšujejo za eno piko
    - `drawOval (x, y, sirina, visina)`
        - nariše elipso, ki se nahaja znotraj navideznega pravokotnika z levim zgornjim ogliščem v točki `(x, y)` in stranicama `a=sirina, b=visina`
        - če sta parametra `sirina` in `visina` enaka, dobimo krog
- določanje barv
    - `setColor (barva)`
        - nastavi barvo, s katero rišemo od tega trenutka dalje
        - parameter `barva` je objekt tipa `Color`
        - v razredu `Color` so definirane konstante za 13 standardnih barv
          `black`
          `blue`
          `cyan`
          `darkGray`
          `gray`
          `green`
          `lightGray`
          `magenta`
          `orange`
          `pink`
          `red`
          `white`
          `yellow`
        - primer klica: `setColor(Color.green);`
        - druga možnost: objekte tipa `Color` lahko generiramo sami kot mešanico rdeče, zelene in modre barve z naslednjim konstruktorjem `Color (int rdeca, int zelena, int modra)`
        - delež vsake barve lahko zavzame vrednost od 0 do 255
        - primer klica: `g.setColor(new Color(0,128,128));`
    - `setBackground (barva)`
        - nastavi barvo podlage
        - metodo je treba poklicati, preden prikažemo okno na zaslonu
    - `setForeground (barva)`
        - nastavi barvo, s katero rišemo
- risanje polnjenih likov
    - imena metod so enaka kot za risanje likov, le predpona `draw` se nadomesti s

```
fill
fillRect (x, y, sirina, visina)
fillRoundRect (x, y, sirina, visina, rH, rV)
fill3DRect (x, y, sirina, visina, dvig)
fillOval (x, y, sirina, visina)
fillPolygon (p)
fillPolygon (x, y, n)
fillArc (x, y, sirina, visina, zacKot, kot)
```

11. Apleti

Kazalo:

- Koncept apleta
- Postopek izdelave
- Metode apleta
  - `init(), start(), stop(), destroy()`
- Osnovne komponente grafičnega vmesnika
  - labela, gumb, vnosno polje
  - razporejanje komponent po risalni plošči
- Primeri
  - preprost aplet s tremi komponentami
  - aplet za zajem podatkov o delavcih

Koncept apleta

- program, ki teče znotraj spletnega brskalnika (npr. Internet Explorer ali Netscape)
- je sestavni del neke spletne strani
- pokličemo ga iz dokumenta, napisanega v HTML (Hypertext Markup Language)

Postopek izdelave

- postopek izdelave apleta
  - aplet napišemo podobno kot druge samostojne programe (aplikacije)
    - shranimo ga v datoteki s podaljškom `.java`
    - prevedemo ga v vmesno kodo (datoteka s podaljškom `.class`)
  - kreiramo HTML dokument, ki mora vsebovati ukaz za klic apleta
  - poženemo spletni brskalnik in naložimo HTML dokument
- izdelava HTML dokumenta
  - vsak HTML dokument se prične z ukazom `<html>` in konča z ukazom `</html>`
    - vsi ukazi so v lomljenih oklepajih
    - HTML ni občutljiv na velike in male črke
  - izvajanje apleta sprožimo z ukazom `<applet>`
    - ob ukazu `<applet>` navedemo tri argumente: `code, width` in `heigth`
      `code` podaja ime datoteke z vmesno kodo (prevod apleta)
      `width` in `heigth` določata širino in višino apleta
    - ukaz `<applet>` zaključimo z `</applet>`
  - primer:

```
<html>
<applet code="Aplet1.class" width=450 height=200>
</applet>
</html>
```

- orodje `appletviewer`
  - omogoča izvajanje apletov brez uporabe brskalnika
  - uporabno je predvsem med razvojem in testiranjem
  - aplet poženemo iz ukazne vrstice z ukazom `appletviewer <ime HTML dokumenta>`
- razred `JApplet`

- vsak applet napišemo kot razširitev osnovnega paketa `JApplet`
- razred `JApplet` se nahaja v paketu `javax.swing` in je izpeljan iz razredov `java.awt.Component` in `java.awt.Container`
- na začetku vsakega apleta so zato prisotni ukazi

```
import javax.swing.*;
import java.awt.*;
public class Aplet1 extends JApplet
```

Metode apleta

- apleti nimajo metode `main ()`
- vsebujejo 4 metode, ki jih brskalnik kliče avtomatsko
  - `public void init()`
    - se izvede, ko se aplet prvič naloži in požene v brskalniku
    - služi za inicializacijo spremenljivk, razmestitev komponent na zaslonu ipd.
  - `public void start()`
    - se izvede takoj za metodo `init()` in nato vsakokrat, ko postane aplet aktiven (ko se uporabnik vrne na stran z apletom, ki jo je prej zapustil ali minimiziral)
    - primer uporabe: nadaljevanje animacije, ki je bila prekinjena, ko je uporabnik zapustil stran
  - `public void stop()`
    - se izvede vsakokrat, ko uporabnik zapusti stran z apletom
  - `public void destroy()`
    - se izvede, ko uporabnik zapre brskalnik ali appletviewer
    - če je potrebno sprostiti vire, ki jih je zasedal aplet
- java avtomatsko kreira prazne metode
- v praksi moramo napisati vsaj eno izmed njih, praviloma `init()`

Osnovne komponente grafičnega vmesnika

- aplet lahko vsebuje različne komponente GUI
  - labela (ang. label): izpis besedila (razred `JLabel`)
  - vnosno polje: polje za vnos nekega podatka (razred `JTextField`)
  - gumb: za sprožitev neke akcije (razred `JButton`)
  - ostale komponente: `JCheckBox, JRadioButton, JComboBox, JToolBar, JScrollBar`
  - komponente dodajamo na risalno ploščo (ang. content pane) z metodo `add()`
  - primer: dodajanje labele, vnosnega polja in gumba

```
JLabel zi=new JLabel ("Ime:");
JTextField i=new JTextField();
JButton potrdi=new JButton ("Potrdi");
Container rp=getContentPane();
rp.add(zi); rp.add(i);
rp.add(potrdi);
```

- razred `JLabel`
  - za izpis teksta in slik
  - več različnih konstruktorjev
    - `JLabel()`: labela brez teksta in slike
    - `JLabel (String text)`: labela s tekstom

        - `JLabel (String text, int horizontalAlignment)`: labela s tekstom, poravnanim v skladu s parametrom `horizontalAlignment`
        - `JLabel (Icon image)`: labela s sliko
        - `JLabel (Icon image, int horizontalAlignment)`: labela s sliko, ki je poravnana v skladu s parametrom `horizontalAlignment`
        - `JLabel (String text, Icon image, int horizontalAlignment)`: labela s tekstom in sliko ter predpisano poravnanostjo
    - metoda `l.setText(niz)`: vpiše `niz` v že kreirano labelo `l`
    - metoda `l.getText()`: vrne niz, ki ga vsebuje labela `l`
- razred `JTextField`
    - za kreiranje vnosnih polj
    - konstruktorji
        - `JtextField ()`: prazno vnosno polje dolžine 0
        - `JTextField (int numColumns)`: prazno vnosno polje dolžine `numColumns`
        - `JTextField (String text)`: vnosno polje z vnaprej vpisanim tekstom
        - `JTextField (String text, int numColumns)`: vnosno polje s tekstom text in dolžino `numColumns`
    - metoda `vp.setText (niz)`: vpiše `niz` v vnosno polje `vp`
    - metoda `vp.getText ()`: vrne niz, ki ga vsebuje vnosno polje `vp`
    - metoda `vp.setEditable (bool)`: določi, ali je možno vpisovanje (`bool` ima vrednost `true`) ali ne (`bool` je `false`)
- razred `JButton`
    - za kreiranje gumbov
    - konstruktorji
        - `JButton ()`: gumb brez napisa
        - `JButton (String text)`: gumb z napisom `text`
        - `JButton (String text, Icon icon)`: gumb z napisom in sliko
        - `JButton (Icon icon)`: gumb s sliko
    - metoda `g.setLabel (niz)`: nastavi napis na gumbu `g`
    - metoda `g.getText ()`: vrne napis, ki ga vsebuje gumb `g`
- nastavljanje fokusa
    - metoda `requestFocus()` omogoča, da vnaprej nastavimo kurzor tipkovnice v izbrano vnosno polje ali izpostavimo določen gumb
    - primer uporabe: `vp.requestFocus();` ali `g.requestFocus();`
- razporejanje komponent po risalni plošči
    - uporabimo enega izmed razporejevalnikov (ang. layout managers)
        - razporejevalnik avtomatsko razporeja komponente znotraj kontejnerja
    - lego določamo sami z ukazi `setLocation()`, `setSize()` in `setBounds()`
- pregled razporejevalnikov
    - `FlowLayout`
        - komponente razporedi po vrsticah
        - ko v neki vrstici zmanjka prostora, nadaljuje v naslednji vrstici
    - `BorderLayout`
        - se uporabi, če ne specificiramo nobenega razporejevalnika

        - površino kontejnerja razdeli na 5 con, ki se imenujejo `"North", "West", "Center", "East"` in `"South"`
        - ob dodajanju vsake komponente je treba navesti tudi cono, kamor naj se komponenta doda
    - `GridLayout`
        - komponente razvrsti v celice, ki tvorijo matriko, sestavljeno iz m vrstic in n stolpcev
        - število vrstic in stolpcev določimo ob inicializaciji razporejevalnika
        - vsaka nova komponenta se doda v naslednjo celico
        - preskakovanje celic ni možno
    - `GridBagLayout`
        - omogoča dodajanje komponent točno v določene celice
        - posamezne komponente lahko zasedajo več celic
    - `CardLayout`
        - komponente se nalagajo ena na drugo
        - primeren, ko želimo, da je naenkrat vidna samo ena komponenta
    - `BoxLayout`
        - vse komponente razporedi v eno vrstico ali v en stolpec
- določitev razporejevalnika
    - najprej generiramo objekt, ki pripada ustreznemu razporejevalniku, npr.

```
FlowLayout flow=new FlowLayout ();
```

ali

```
GridLayout grid=new GridLayout (4, 7);
```

    - nato nastavimo razporejevalnik z metodo `setLayout ()`

```
Container rp=getContentPane();
rp.setLayout(flow);
```

ali

```
rp.setLayout (grid);
```

    - oba koraka združimo

```
rp.setLayout (new FlowLayout() );
```

- brez razporejevalnika
    - razporejevalnik nastavimo na `null`

```
rp.setLayout (null);
```

Primeri

- aplet s tremi komponentami: labelo, vnosnim poljem in gumbom
    - napovemo uporabo paketov `javax.swing in java.awt`
    - aplet deklariramo kot razširitev razreda `JApplet`
    - generiramo vse tri komponente
    - izberemo razporejevalnik
    - pripravimo risalno ploščo
    - dodamo vse tri komponente na risalno ploščo
    - nastavimo fokus (kurzor tipkovnice)
- določanje barve
    - za vsako komponento lahko določimo barvo podlage in barvo črk

```
gumb.setForeground (Color.red);
gumb.setBackground (Color.yellow);
```

- določanje pisave
    - uporabimo že znano metodo `setFont(f)`
    - argument f je objekt tipa `Font` in mora biti prej določen, npr.

```
Font pisava1=new Font ("TimesRoman", Font.ITALIC, 24);
Font pisava2=new Font ("Helvetica", Font.BOLD, 20);
```

- pisavo določimo za vsako komponento posebej, npr.

```
labela.setFont (pisava1);
vnPolje.setFont (pisava2);
```

- odstranjevanje komponent
  - komponente odstranjujemo z metodo `remove (<ime komponente>)`, npr.

```
remove (gumb); ali remove (labela);
```

  - po odstranitvi komponente je treba vsebino risalne plošče ponovno narisati z metodo `repaint ()`
  - aplet se mora odzivati na dogodke
  - v našem apletu lahko uporabnik zaključi vnos na dva načina
    - s klikom na gumb
    - s tipko Enter
  - oba dogodka sta tipa `ActionEvent`
  - določiti je treba poslušalca, ki bo zaznal omenjena dogodka
    - poslušalec je aplet (ni treba vpeljati posebnega objekta)
    - poslušalec mora implementirati metode, ki so določene z vmesnikom (ang. interface) `ActionListener`
  - vmesnik `ActionListener` zahteva samo eno metodo
    `public void actionPerformed (ActionEvent d)`
    - za vmesnike, ki zahtevajo samo eno metodo, ne obstaja ustrezen adapter, v katerem bi bila ta metoda realizirana kot prazna metoda
    - zato mora poslušalec obvezno implementirati vmesnik
      `public class Pozdrav2 extends JApplet implements ActionListener`
- potrebne spremenljivke v našem apletu
  - dodaten stavek `import java.awt.event.*;`
  - sprememba glave

```
public class Pozdrav2 extends JApplet implements
   ActionListener
```

  - registracija poslušalcev: obema komponentama, ki nastopata kot možna izvora dogodkov, dodamo poslušalca

```
gumb.addActionListener(this);
vnPolje.addActionListener(this);
```

  - metoda `actionPerformed (ActionEvent d)`
    - ugotoviti mora izvor dogodka in izvršiti ustrezno akcijo
    - izvor dogodka določimo z metodo `getSource()`

```
Object izvor==d.getSource();
```

    - akcija, ki sledi dogodku, je odvisna od izvora

```
if (izvor==gumb) …
```

    - če nas zanima samo tip izvora uporabimo rezervirano besedo

```
instanceof
if (izvor instanceof JTextField) …
```

- aplet za zajem podatkov o delavcih
  - opis problema
    - zajeti želimo naslednje podatke:
      - matično številko
      - priimek

        - ime
        - število ur
    - zajeti podatki se vpišejo v tabelo, v kateri vsak element predstavlja enega delavca (objekt tipa `Delavec6`)
    - uporabnik vnaša podatke v 4 vnosna polja
        - ko vnese podatke za enega delavca, s pritiskom na gumb "Vnesi" sproži vpis v tabelo
        - če želi, lahko pobriše vsebino vseh vnosnih polj in ponovi vnos (gumb "Briši")
        - vnos poteka v zanki: ko se podatki za enega delavca vpišejo v tabelo, se vnosna polja izpraznijo, da je možen vnos podatkov za naslednjega delavca
        - ko so zajeti podatki za vse delavce, se izpiše ustrezno obvestilo
- potrebne komponente
    - štiri vnosna polja

```
JTextField ms=new JTextField();    // matična številka
JTextField p=new JTextField();     // priimek
JTextField i=new JTextField();     // ime
JTextField u=new JTextField();     // število ur
```

    - pred vsakim vnosnim poljem izpišemo ustrezen zahtevek: štiri labele

```
JLabel zms=new JLabel("Matična številka:");
JLabel zp=new JLabel("Priimek:");
JLabel zi=new JLabel("Ime:");
JLabel zu=new JLabel("Število ur:");
```

    - dva gumba

```
JButton vnesi=new JButton("Vnesi");
JButton brisi=new JButton("Briši");
```

    - obvestilo o zaključku vnosa: labela, ki je na začetku prazna

```
JLabel obv=new JLabel("");
```

- razporejanje komponent
    - ne bomo uporabili razporejevalnika

```
Container rp=getContentPane();
rp.setLayout(null);
```

    - komponente razporejamo s pomočjo metode `setBounds()`
    - razmestitev zahtevkov za vnos in vnosnih polj

```
zms.setBounds(30,30,110,20);  ms.setBounds(150,30,60,20);
zp.setBounds(30,50,110,20);   p.setBounds(150,50,100,20);
zi.setBounds(30,70,110,20);   i.setBounds(150,70,100,20);
zu.setBounds(30,90,110,20);   u.setBounds(150,90,60,20);
```

    - razmestitev gumbov

```
brisi.setBounds(30,120,70,20);
vnesi.setBounds(150,120,70,20);
```

    - položaj obvestila

```
obv.setBounds(30,180,150,20);
```

- dodajanje komponent na risalno ploščo

```
rp.add(zms);      rp.add(ms);
rp.add(zp);       rp.add(p);
rp.add(zi);       rp.add(i);
rp.add(zu);       rp.add(u);
rp.add(vnesi);    rp.add(brisi);
rp.add(obv);
```

- namestitev kurzorja tipkovnice v prvo vnosno polje

```
ms.requestFocus();
```

- deklaracija tabele delavcev

```
static final int ST_DEL=5;
static Delavec6[] td=new Delavec6[ST_DEL];
int j=-1;   // indeks v tabeli delavcev
//vrednost -1 označuje, da je na začetku tabela
//prazna; ob dodajanju vsakega delavca se j poveča za 1
```

- deklaracija razreda `Delavec6`
  - atributi

```
private int matStev;
private int String priimek;
private String ime;
private int stUr;
```

  - konstruktor
  - metode
    - za vpis vrednosti posameznih atributov
    - za vračane vrednosti posameznih atributov
    - za izpis vrednosti vseh atributov: `izpisiVse()`
- dodajanje poslušalca
  - izvora dogodkov sta gumba "Vnesi" in "Briši", poslušalec je aplet

```
vnesi.addActionListener(this);
brisi.addActionListener(this);
```

- metoda `actionPerformed (ActionEvent d)`
  - ugotavljanje izvora dogodka

```
Object izvor=d.getSource();
   if (izvor==vnesi)
   {
   }
    else if (izvor==brisi)
   {
   }
```

  - pritisk na gumb "Vnesi"
    - branje podatkov iz vnosnih polj in generiranje novega elementa tabele
    - če je tabela polna, odstranimo oba gumba in izpišemo obvestilo
    - če tabela še ni polna, izpraznimo vnosna polja in jih pripravimo za vnos naslednjega delavca
  - pritisk na gumb "Briši"
    - izpraznimo vnosna polja in s tem omogočimo uporabniku ponoven vnos podatkov
- branje podatkov iz vnosnih polj v nov element tabele

```
int matSt=Integer.parseInt(ms.getText());
String priimek=p.getText();
String ime=i.getText();
int stUr=Integer.parseInt(u.getText());
++j;
td[j]=new Delavec6(matSt,priimek,ime,stUr);
```

- odstranitev obeh gumbov in izpis obvestila

```
remove(vnesi); remove(brisi);
obv.setText("Tabela je polna.");
repaint();
```

- izpraznitev vnosnih polj

```
ms.setText("");
p.setText("");
i.setText("");
u.setText("");
ms.requestFocus();
```