UNIVERZA V LJUBLJANI
FAKULTETA ZA MATEMATIKO IN FIZIKO

Matematika – praktična matematika (VSŠ)

Nina Kerčmar

PRVI KORAKI V JAVI

DIPLOMSKA NALOGA

Ljubljana, 2006

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

IZJAVA

Študentka Nina Kerčmar izjavljam, da sem avtorica tega diplomskega dela, ki sem ga napisala pod mentorstvom mag. Matije Lokarja in dovoljujem uporabo diplomskega dela za izobraževalne namene.

Ljubljana, dne Podpis:

KAZALO

Program diplomske naloge

V diplomski nalogi predstavite programski jezik Java na način, primeren za prvo seznanjanje s tem jezikom. Predstavljene naj bodo osnovne značilnosti jezika – osnovni podatkovni tipi, osnovni stavki in kontrolne strukture. Omejite se na proceduralni del jezika, torej ne opisujte načina gradnje objektov in objektnega programiranja.

mentor:

mag. Matija Lokar

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

ZAHVALA

Zahvalila bi se najprej vsem predavateljem Fakultete za matematiko in fiziko, ki so nam omogočili pridobitev ustreznega znanja in veščin. Nadalje bi se rada zahvalila svojemu mentorju, gospodu mag. Matiji Lokarju , ki mi je svetoval in mi nudil strokovno pomoč.
Prav tako bi se zahvalila svojemu fantu, ki mi je stal ob strani ter me bodril v težkih trenutkih in se odrekal mnogim stvarem zaradi časa, ki je potreben za izdelavo diplomske naloge.

1 Uvod

Diplomska naloga govori o osnovnih značilnostih programskega jezika Java. Cilj diplomske naloge je srednješolcem (oziroma drugim začetnikom) na poljuden, domač način predstaviti programski jezik Java. Ker na to temo še ni diplomske naloge, sem se je lotila s toliko večjim veseljem in hkrati odgovornostjo.

Nina Kerčmar, avtorica tega diplomskega dela, sem se lotila pojasnjevanja oziroma razlage določenih programerskih pristopov, njihove predstavitve in uporabe v Javi.

Diplomsko delo sem razdelila v dva večja dela. Drugo poglavje predstavlja uvod v samo diplomsko delo ter uvod v Javo, od tretjega do osmega poglavja je vsebina posvečena podrobnejšemu pregledu osnovnih značilnosti programskega jezika.

Ključne besede: Java, programski jezik, objektno programiranje, razvoj Jave.

Key words: Java, programming language, object oriented programming, developmant of Java.

2 UVOD V JAVO

2.1 Zgodovina Jave

Leta 1991 so v podjetju Sun uvedli tako imenovani Green Project, projekt, katerega namen je bil računalniško podpreti različne množično uporabljane elektronske naprave, od kuhinjskih pripomočkov do izdelkov zabavne elektronike in drugih. V ta namen so združili skupino strokovnjakov, kasneje poznano pod imenom Green Team. Na čelu skupine so bili James Gosling, Mike Sheradin in Patrick Naughton. Njihova naloga naj bi bila ustvariti programsko opremo, ki bi omogočala krmiljenje teh elektronskih naprav. Ker so v naprave vgrajeni mikroprocesorji zelo različni, se je hitro pokazalo, da je potreben nov programski jezik, ki bi omogočal na enoten način programirati tako različne izdelke.

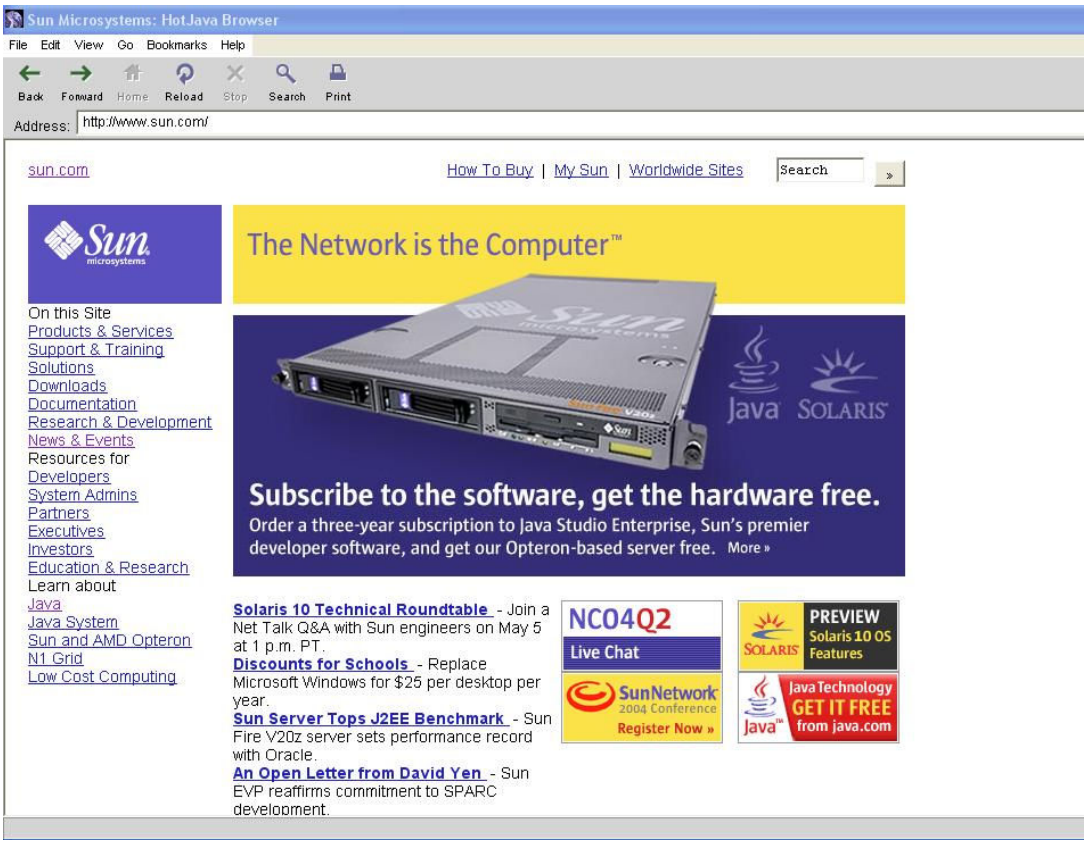
James Gosling je bil zadolžen za razvoj takega programskega jezika. Sprva je poskusil razširiti in posodobiti programski jezik C++. Vendar je kmalu ugotovil, da je bolje sestaviti nov jezik. Novonastali jezik je poimenoval Oak, vendar je bilo to ime že registrirano za nek drug programski jezik. Tako je prišlo do imena Java, povzetega po znamki kave, in naslednjega zaščitnega znaka.

Slika 1.1 .Zaščitni znak Jave

Da bi pokazali uporabnost svojih idej, so po 18–ih mesecih trdega dela v Green Teamu izdelali prototip naprave, velike kot roka, ki je nadzorovala različne elektronske naprave v gospodinjstvu. Poimenovali so jo Star7. Imela je na dotik občutljiv zaslon in aktivne slike.

Slika 1.2.Prototip Star7

Stvaritelji naprave in jezika so pričakovali uspeh na trgu. Vendar prvotna želja, da bi ju kot tehnologiji prodali izdelovalcem elektronskih naprav, ni bila izpolnjena.
Kazalo je že, da bo Java še eden v množici programskih jezikov, ki nastanejo in ne doživijo širše uveljavitve. A v tistem času je postajal vedno bolj popularen svetovni splet. Tudi tu se je pokazala potreba po programskem jeziku, ki bi preko spletnih brskalnikov omogočal izvajanje različnih programov. Ker je tudi strojna oprema računalnikov, priključenih v svetovni splet, zelo raznolika, je moral jezik na nek način vsemu temu ustrezati. To pa je bila ravno osnova, na kateri je bila že zasnovana Java. Bill Joy je opazil zmogljivost programskega jezika Java in možnost, da bi Java postala "skupni" jezik interneta. Tako je nastal projekt LiveOak. Paul Naughan je sprogramiral spletni brskalnik WebRunner, ki je bil kasneje preimenovan v HotJavo.



Slika 1.3. Brskalnik HotJava, ki je razumel Javo

Predstavitev programskega jezika Java in brskalnika HotJava je povzročila pravo internetno revolucijo. Nenadoma so bile spletne strani lahko opremljene s programi, ki so se izvajali na strani uporabnika. **Znotraj spletnega** brskalnika (ki je moral seveda razumeti Javo) je bilo moč izvajati programe, napisane v Javi, neodvisno od tega, na kakšni strojni opremi in v kakšnem operacijskem sistemu je bil program razvit. Udarni slogan je bil: "Write once, execute everywere" (Napiši enkrat, izvajaj povsod). Zato, da si na neki novi napravi, priključeni na splet, lahko izvedel poljuben Javanski program, ki se je nahajal na poljubni spletni strani, je bilo potrebno le, da je bil na tej napravi spletni brskalnik, ki je razumel Javo. In ker je podjetje Netscape, katerega brskalnik je takrat prevladoval med spletnimi brskalniki, v svoj izdelek hitro vgradilo podporo za Javo, se je govorilo le še o Javi in revoluciji, ki jo prinaša.

Kasneje je navdušenje nad Javo kot jezikom za razvoj spletnih aplikacij iz različnih vzrokov nekoliko upadlo, a se je Java kljub temu uveljavila tudi izven programiranja v spletu kot sodoben, objektno zasnovan jezik.

Naj zaključim uvod z mislijo Jamesa Goslinga, očeta Jave, ki je zapisal, da resnični izumi nastanejo, ko nori ljudje počnejo nore stvari. Ključi do uspeha so naslednji:

- preden se podaš v pustolovščino, se nauči vse, kar je možno,
- ne boj se delati napak,
- delaj le nove napake,
- imej odprte oči,
- ne glej samo naravnost, razširi obzorje,
- stvari, ki se nagibajo v najbolj nepričakovano smer, so ponavadi najpomembnejše.

3 OSNOVNI UKAZI

3.1 Prvi program

Pa začnimo! Najprej moramo imeti nameščeno Javo in urejevalnik. Ustreznih okolij, v katerih lahko pišemo programe v Javi, je več. V diplomski nalogi ne bomo govorili o nobenem posebnem.

Ko se učimo nov programski jezik, je po tradiciji prvi program, ki ga napišemo, program, ki na zaslon izpiše besedilo *"Pozdravljen, svet!"*. Zaženimo urejevalnik in vanj zapišimo[1]:

PRIMER 3.1 – 1: *Pozdravljeni.java*

```
1:   // Program, ki na zaslon izpiše niz "Pozdravljen, svet!".
2:
3:   public class Pozdravljeni {
4:      public static void main(String[] args) {
5:        System.out.println("Pozdravljen, svet!");
6:      } // main
7:    } // Pozdravljeni
```

Pri pisanju programa moramo paziti na velikost znakov, saj Java loči med velikimi in malimi črkami. Ko je program napisan, ga shranimo pod imenom *Pozdravljeni.java*. Tudi pri tem se moramo držati pravilnega poimenovanja. Program mora biti shranjen na datoteki *Pozdravljeni.java* in ne na primer na datoteki *pozdravljeni.java*.

Program na ustrezen način prevedemo, poženemo in na zaslon nam izpiše:

```
Pozdravljen, svet!
```

Poglejmo si kratko razlago programa *Pozdravljeni.java*.

V prvi vrstici je **komentar**, ki je namenjen kratkemu opisu programa. Na začetku izvorne kode običajno napišemo komentar, ki pove, čemu program služi.

Komentarji se uporabljajo za opis kode in za podajanje dodatnih informacij o kodi. Namenjen je programerju (oziroma bralcu kode). Ne vpliva na izvajanje programa, saj ga prevajalnik enostavno preskoči. Paziti moramo, da je izvorna koda s pomočjo komentarjev čim bolj čitljiva in razumljiva in da so komentarji smiselni.

Poznamo tri vrste komentarjev.

1) **Enovrstični komentar** je namenjen pisanju komentarjev, ki so v eni vrstici.
Zapišemo ga za znakoma *//*. Od tu naprej, pa do konca tekoče vrstice, je tisto, kar je napisano, komentar in ne spada v program.

```
// enovrstični komentar
```

2) **Več vrstični komentar** je namenjen pisanju komentarjev, ki se raztezajo čez več vrstic. Več vrstični komentar začnemo z znakoma */** in ga zaključimo z znakoma **/*.

```
/*
   Več vrstični komentar,
```

[1] Številke na levi so nam le v pomoč pri razlagi, zato jih ne vpisujemo v kodo programa.

```
   ki se razteza čez več vrstic
*/
```

Paziti moramo, da ga vedno zaključimo, sicer lahko nehote izključimo zajetne kose programske kode.

3) **Dokumentacijski komentar** je namenjen za avtomatsko pripravo dokumentacije, zapišemo ga med znaki `/**` in `*/`. Mi ga ne bomo uporabljali.

Znotraj programa ne pretiravamo s komentarji. Pri bolj zahtevnih postopkih napišemo več vrstični komentar, pri nezahtevnih pa krajšega. Priporočljivo pa je tudi napisati končni komentar, to je oznako za konec razreda, metode in bloka[2]. Tako za vsak zaviti zaklepaj, s katerim zaključimo tak del, vemo, kateri del kode zaključuje. Praviloma napišemo komentar tudi pri vsaki deklarirani spremenljivki. Komentarjev ne pišemo zato, da bi povedali, kaj pomenijo stavki, ampak da z njimi na kratko opišemo pomen celotnega sklopa stavkov. Vsebujejo naj samo informacije, ki so pomembne za branje in lažje razumevanje programa. Sicer bralcu ti stavki verjetno sedaj ne pomenijo kaj dosti, a velja si jih zapomniti in jih kasneje, ko bomo pisali "zaresne" programe, tudi upoštevati.

Komentarju sledi **najava razreda** (3. vrstica). Pri tem moramo paziti, da se ime datoteke (programa) in ime razreda (ang. class) ne razlikujeta (pazimo na velikost črk!). Razlago, kaj razred je, odložimo. Zaenkrat povejmo le, da je razred zaključen del programa, ki se začne in konča z zavitim oklepajem.

Naš razred se začne v tretji vrstici in konča v sedmi vrstici. To je:

```
public class Pozdravljeni {
   ...
}
```

Znotraj razreda se nahaja **metoda `main()`** (4. – 6. vrstica). Ta mora biti zapisana točno tako, kot je navedeno, torej

```
public static void main(String[] args) {
  ...
}
```

Pomen tega dela bomo razjasnili kasneje[3]. Metoda main vsebuje le en ukaz (stavek[4]), ki poskrbi, da se sporočilo izpiše na ekran.

```
System.out.println("Pozdravljen, svet!");
```

Če pri pisanju programa naredimo sintaktično napako, nam to sporoči prevajalnik. Te napake se pojavijo, kadar prevajalnik opazi, da naš program ne ustreza pravilom, kako naj bi bili programi v Javi napisani. Dokler program vsebuje sintaktične napake, ga prevajalnik ne zna prevesti – pretvoriti v obliko, ki jo lahko izvajamo na računalniku.

Oglejmo si primer sintaktične napake. Denimo, da smo pri pretipkavanju prejšnjega programa izpustili podpičje v peti vrstici.

---

[2] Blok je eden ali več stavkov, ki jih zapišemo med zavitima oklepajema {}.
[3] Za neučakane: metoda `main` ne vrača nobenega rezultata, zato je tipa `void`. Del `String[] args` pove, da metoda dobi en argument, ki se imenuje `args` in je tipa `String[]` (tabela nizov).
[4] Stavek je ukaz oziroma navodilo, ki "sili" računalnik v neko delo. V Javi moramo vsak stavek zaključiti s podpičjem.

PRIMER 3.1 – 2: *Pozdravljeni.java*

```
1:   // Program, ki na zaslon izpiše niz "Pozdravljen, svet!".
2:
3:    public class Pozdravljeni {
4:      public static void main(String[] args) {
5:        System.out.println("Pozdravljen, svet!")
6:      } // main
7:    } // Pozdravljeni
```

Ko pokusimo prevesti program, nam prevajalnik javi:

```
C:\Pozdravljeni.java:5: ';' expected
    System.out.println("Pozdravljen, svet!")
                                            ^
1 error
```

Prevajalnik je javil napako v peti vrstici datoteke *Pozdravljeni.java*, kar vidimo iz izpisa *"Pozdravljeni.java:5:"*. To ne pomeni, da je napaka res v peti vrstici, ampak je prevajalnik v peti vrstici opazil, da stvar ni povsem v redu! Ponavadi je napaka res v javljeni vrstici, ni pa nujno. Vsekakor jo iščemo od označenega mesta proti začetku datoteke. Prevajalnik je sporočil tudi vrsto napake (*';' expected*), ki se spet ne ujema nujno s pravim vzrokom za napako, in označil mesto vrstice *(^)*, v kateri je napako zaznal. V našem primeru je to za zaklepajem.

Poglejmo si še en primer. Tokrat v peti vrstici nismo zaključili niza.

PRIMER 3.1 – 3: *Pozdravljeni.java*

```
1:    // Program, ki na zaslon izpiše niz "Pozdravljen, svet!".
2:
3:    public class Pozdravljeni {
4:      public static void main(String[] args) {
5:        System.out.println("Pozdravljen, svet!);
6:      } // main
7:    } // Pozdravljeni
```

V tem primeru nam prevajalnik izpiše:

```
C:\Pozdravljeni.java:5: unclosed string literal
    System.out.println("Pozdravljen, svet!);
                       ^
C:\Pozdravljeni.java:5: ')' expected
    System.out.println("Pozdravljen, svet!);
                                           ^
2 errors
```

Prevajalnik je javil dve napaki. Pogosto se zgodi, da prva napaka povzroči ostale, zato vedno usmerimo pozornost le na prvo javljeno napako.
Če pogledamo obvestilo o napaki, prevajalnik javlja, da v peti vrstici nismo zaključili niza. Niz je poljubno zaporedje znakov med dvojnima narekovajema *(")*. Če popravimo to napako, tudi druga izgine.
Tretji zgled je povezan z napačno uporabljenimi komentarji.

PRIMER 3.1 – 4: *Pozdravljeni.java*

```
1:    /* Program,
2:    /* ki na zaslon izpiše niz */
3:    "Pozdravljen, svet!". */
```

```
4:
5:    public class Pozdravljeni {
6:      public static void main(String[] args) {
7:        System.out.println("Pozdravljen, svet!");
8:      } // main
9:    } // Pozdravljeni
```

Ko prevedemo program, nam prevajalnik javi:

```
C:\Pozdravljeni.java:3: 'class' or 'interface' expected
"Pozdravljen, svet!". */
^
1 error
```

V prvi vrstici je začetek komentarja. Ko prevajalnik na koncu druge vrstice vidi zaporedje znakov `*/` predpostavi, da smo s tem komentar zaključili. Zato prevajalnik del `"Pozdravljen, svet!". */` vzame za del kode. Tega ne razume in nam javi napako.

3.2 Struktura oziroma "okostje" programa

Program, napisan v Javi, sestoji iz enega ali več razredov. Mi bomo zaenkrat pisali programe, ki bodo vsebovali samo en razred. V Javi se program začne izvajati v metodi `main()`. Ukazi, oziroma stavki, znotraj te metode se izvajajo po vrsti od zgoraj navzdol.

Naši programi bodo videti takole:

```
public class ImePrograma {
  public static void main(String[] args) {
     // ukazi - stavki
  }
}
```

Shranili jih bomo na datoteke z imenom `ImePrograma.java`, seveda z ustreznim imenom programa.
Namesto komentarja `// ukazi - stavki` bomo napisali ustrezne stavke, ki bodo povedali, kaj želimo, da se v našem programu zgodi. Vse ostalo (`public, class, main ...`) pa mora ostati točno tako, kot je napisano.

Pazimo, da je koda pregledna, saj je tako nekomu, ki program proučuje, lažje ugotoviti, kaj program počne. Kar se prevajalnika tiče, bi naš prvi program lahko napisali tudi kot:

PRIMER 3.2 – 1: `Pozdravljeni.java`

```
1:    public class Pozdravljeni { public static void
2:    main(
3:    String[] args
4:    ) {System.out.println

5:    ("Pozdravljen, svet!"); }}
```

ali pa bi celo vse skupaj "stlačili" v eno samo vrstico. Prevajalnik bi vse te oblike prevedel enako, a nam bi bilo oblikovno grdo napisan program veliko težje razumeti.

Koda postane preglednejša, če jo razdelimo na odstavke. Za ločevanje odstavkov uporabljamo prazno vrstico. Tudi zamaknjena koda za dva do tri znake v desno znotraj bloka (torej znotraj para zavitih oklepajev `{}`) pripomore k boljši preglednosti. Tako je npr. zamaknjena metoda `main` znotraj razreda

in zamaknjen ukaz *System.out.println* znotraj metode *main*. Pomembno je, da je oblika znotraj programa povsod enaka (enaka uporaba presledkov, zamikanja, poimenovanja ...).

3.3 Izpisovanje na ekran

Osnovni način izpisovanja na zaslon je z metodo *println()* oziroma z metodo *print()*. Obe metodi izpišeta tisto, kar je podano med narekovaji (") znotraj okroglih oklepajev. V našem prvem programu se je tako izpisalo *Pozdravljen, svet!*. Edina razlika med metodama *println()* in *print()* je, da prva po izpisu prestavi izpisno mesto (položaj začetka naslednjega izpisa) na začetek nove vrstice, pri drugi pa ostane izpisno mesto v isti vrstici desno od zadnjega izpisanega znaka.

Zgledi

Spremenimo naš prvi program tako, da bo namesto *Pozdravljen, svet!* izpisal *Programski jezik Java*. Spremeniti moramo le tekst med " pri metodi *println()*. Zato, da ne bomo "povozili" prvega programa, bomo spremenjeni program shranili na datoteko *Izpis.java*. Seveda moramo potem spremeniti tudi ime razreda v *Izpis*.

PRIMER 3.3 – 1: *Izpis.java*

```
1:    // Program, ki na zaslon izpiše niz "Programski jezik Java".
2:
3:    public class Izpis {
4:      public static void main(String[] args) {
5:        System.out.println("Programski jezik Java");
6:      } // main
7:    } // Izpis
```

Prevedemo in poženemo:

```
Programski jezik Java
```

Kako pa bi napisali vsako besedo v svojo vrstico? Izpisati želimo:

```
Programski
jezik
Java
```

PRIMER 3.3 – 2: *Izpis1.java*

```
1:    public class Izpis1 {
2:      public static void main(String[] args) {
3:        System.out.println("Programski");
4:        System.out.println("jezik");
5:        System.out.println("Java");
6:      } // main
7:    } // Izpis1
```

Prevedemo in poženemo:

```
Programski
jezik
Java
```

Prvi ukaz (stavek), ki se izvede, je stavek v tretji vrstici. Tu metoda *println()* poskrbi, da se na zaslon izpiše beseda *Programski* in izpisno mesto prestavi na začetek naslednje vrstice. Potem se

izvede četrta vrstica, ki izpiše ustrezno besedo in izpisno mesto postavi v naslednjo vrstico. Nato je na vrsti peta vrstica, ki spet izpiše ustrezno besedilo in izpisno mesto prestavi v naslednjo vrstico.

Kaj pa se bo izpisalo v tem primeru?

PRIMER 3.3 – 3: *Izpis2.java*

```
1:    public class Izpis2 {
2:      public static void main(String[] args) {
3:        System.out.print("Programski ");
4:        System.out.print("jezik");
5:        System.out.print( "Java");
6:      } // main
7:    } // Izpis2
```

Če niste preskočili razlage o metodi *print()*, ne bi smelo biti dvomov, kaj se zgodi. Izpiše se:

```
Programski jezikJava
```

Program izpiše vse tri nize enega za drugim, saj smo uporabili metodo *print()*. Ta ne naredi nič drugega, kot da izpiše željen niz (izpisno mesto ostane za koncem niza). Enak učinek bi dobili, če bi uporabili stavek

```
System.out.print("Programski jezikJava");
```

Zakaj pa med besedami jezik in Java ni presledka? Rekli smo, da metoda *print()* (in *println()*) izpiše tisto, kar je med narekovaji. In ker v peti vrstici presledek ni znotraj narekovajev (je med (in "), ga prevajalnik pač ne upošteva in presledek se ne izpiše.

Poskusimo v metodi *print()* izpisati hkrati več stvari.

PRIMER 3.3 – 4: *Izpis3.java*

```
1:    public class Izpis3 {
2:      public static void main(String[] args) {
3:        System.out.print("Programski ", "jezik ", "Java");
4:      } // main
5:    } // Izpis3
```

Prevedemo in prevajalnik se pritoži!

```
C:\Izpis3.java:3: cannot resolve symbol
symbol: method print
(java.lang.String,java.lang.String,java.lang.String)
location: class java.io.PrintStream
    System.out.print("Programski ", "jezik ", "Java");
               ^
1 error
```

Metoda *print()* pričakuje, da bo med oklepaji dobila le en niz! Iz zadrege se rešimo takole:

PRIMER 3.3 – 5: *Izpis3.java*

```
1:    public class Izpis3 {
2:      public static void main(String[] args) {
3:        System.out.print("Programski " + "jezik " + "Java");
4:      } // main
5:    } // Izpis3
```

Prevedemo in poženemo:

```
Programski jezik Java
```

Tukaj vidimo, da lahko nize združujemo (stikamo) s tako imenovanim združitvenim operatorjem `'+'`. Na ta način pogosto razdelimo nize v bolj pregledne, krajše nize. Poglejmo primer.

PRIMER 3.3 – 6: *DolgNiz.java*

```
1:    public class DolgNiz {
2:      public static void main(String[] args) {
3:        System.out.println("Peter gre jutri na morje. " +
4:                           "Mislim, da gre z avtomobilom.");
5:      } // main
6:    } // DolgNiz
```

Prevedemo in poženemo:

```
Peter gre jutri na morje. Mislim, da gre z avtomobilom.
```

Ker prevajalnik pri prevajanju izloči tako imenovane "bele znake" (to so presledki, tabulatorji, prehodi v novo vrsto...), ki niso del nizov, obravnava tretjo in četrto vrstico tako, kot bi jo napisali v isti vrstici.

```
"Peter gre jutri na morje. " + "Mislim, da gre z avtomobilom."
```

Če znotraj oklepajev v metodi `println()` napišemo izraz (uporabimo torej operatorje – zaenkrat poznamo le operator stika *+*), se najprej izračuna vrednost tega izraza. V našem primeru je to sestavljeni niz. Rezultat izraza metoda `println()` izpiše na zaslon.

Poglejmo, kako bi z metodo `print()` izpisali vsako besedo v svoji vrstici. V tem primeru si pomagamo s posebnim znakom[5] `\n`. Ko ta znak "izpisujemo", se to pozna tako, da se na tistem mestu izvede prehod v novo vrsto.

PRIMER 3.3 – 7: *IzpisNovaVrsta.java*

```
1:    public class IzpisNovaVrsta {
2:      public static void main(String[] args) {
3:        System.out.print("Programski\n");
4:        System.out.print("jezik\nJava\n");
5:      } // main
6:    } // IzpisNovaVrsta
```

Prevedemo in poženemo:

```
Programski
jezik
Java
```

Prvi ukaz (stavek), ki se izvede, je v tretji vrstici. Tu metoda `print()` poskrbi, da se na zaslon izpiše beseda `Programski`. Znak `\n` na koncu niza pomeni, da se izvede preskok v novo vrsto. Naslednja se izvede četrta vrstica. Na zaslon se izpiše beseda `jezik`, zaradi znaka `\n` pa skočimo v novo vrsto. Nato se na zaslon izpiše beseda `Java`, znak `\n` pa povzroči še preskok v novo vrsto.

[5] Posebni znaki so znaki, ki prek tipkovnice niso dostopni ali pa imajo v kodi poseben pomen. Prikažemo jih s simbolom \ (poševnica). Med pomembnejšimi so znak za prehod v novo vrsto `\n`, tabulator `\t`, poševnica `\\` in dvojni narekovaj `\"`.

V prejšnjih primerih smo si pogledali, kako izpisujemo nize. Povejmo še enkrat, da so to poljubna zaporedja znakov med dvojnimi narekovaji. Poglejmo še, kako bi izpisali število.

PRIMER 3.3 – 8: *IzpisStevila.java*

```
1:    public class IzpisStevila {
2:      public static void main(String[] args) {
3:        System.out.println("12");
4:        System.out.println(12);
5:      } // main
6:    } // IzpisStevila
```

Prevedemo in poženemo:

```
12
12
```

S tretjo vrstico izpišemo niz, s četrto pa število. Na zaslonu je izpis enak, a če želimo izpisati niz, ga zapišemo med narekovaji. Če pa izpisujemo število, ga navedemo brez narekovajev. Do razlike pride takrat, ko sta število oziroma niz, v katerem je zapisano število, sestavni del nekega izraza. Videli smo že, da je argument metode `println` (tisto, kar je med oklepaji), lahko tudi izraz. Če napišemo `"12" + "13"` bomo dobili niz `"1213"`. Če pa `+` uporabimo med števili, ima učinek, ki smo ga navajeni iz računstva – sešteje števili. Izračunajmo (in izpišimo) vsoto števil `12` in `13`.

PRIMER 3.3 – 9: *Racun.java*

```
1:    public class Racun {
2:      public static void main(String[] args) {
3:        System.out.println("12 + 13");
4:        System.out.println(12 + 13);
5:      } // main
6:    } // Racun
```

Prevedemo in poženemo:

```
12 + 13
25
```

V 3. vrstici izpišemo niz, ki je med oklepaji (`"12 + 13"`). V 4. vrstici pa imamo izraz. Zato se najprej izvede računska operacija (seštevanje), šele nato se izpiše dobljena vrednost (število 25).

Operator `'+'` torej nastopa v dveh vlogah – kot operator seštevanja in za stikanje (združevanje) nizov. Če sta oba operanda števili ali če sta oba operanda niza, je pomen jasen. Kaj pa, če je eden od operandov niz in drugi število? Takrat se število pretvori v niz (`12` v niz `"12"`) in `'+'` je operator združevanja nizov. Za lažje razumevanje uporabe si poglejmo naslednji primer.

PRIMER 3.3 – 10: *IzpisNizStevilo.java*

```
1:    public class IzpisNizStevilo {
2:      public static void main(String[] args) {
3:        System.out.println("Stevilo" + 1 + 2);
4:        System.out.println(1 + 2 + "Stevilo");
5:        System.out.println(1 + "Stevilo" + 2);
6:      } // main
7:    } // IzpisNizStevilo
```

Prevedemo in poženemo:

```
Stevilo12
3Stevilo
1Stevilo2
```

Poglejmo, zakaj je izpis tak:
3. vrstica. V izrazu nastopata dva operatorja `'+'`. Kot smo navajeni, izvedemo izračun od leve proti desni. Najprej se bo izračunala vrednost (pod)izraza `"Stevilo" + 1`. Ker je prvi operand niz (`"Stevilo"`), drugi pa število `1`, prevajalnik poskrbi, da se število `1` pretvori v niz `"1"`. Dobimo `"Stevilo" + "1"` . Zaradi operatorja za združevanje se ta dva niza stakneta. Rezultat je torej niz `"Stevilo1"`. Sedaj moramo izračunati še `"Stevilo1" + 2`. Ker "seštevamo" niz in število, prevajalnik ponovno poskrbi, da se število `2` pretvori v niz `"2"`. Dobimo `"Stevilo1" + "2"`. Niza se stakneta in rezultat celotnega izraza je `"Stevilo12"`, ki ga metoda `println` izpiše.
4. vrstica. Ker sta na začetku izraza števili `1` in `2`, operator `'+'` pomeni operacijo seštevanja. Izračuna se vsota `1 + 2`. Dobimo število `3`, ki mu "prištevamo" niz `"Stevilo"`. Prevajalnik poskrbi, da se število `3` pretvori v niz `"3"` in niza se stakneta. Dobimo niz `"3Stevilo"`, ki se izpiše.
5. vrstica. Ker je med številom (`1`) in nizom (`"Stevilo"`) operator `'+'`, prevajalnik število `1` pretvori v niz `"1"`. S stikanjem dobimo niz `"1Stevilo"`. Ker imamo med nizom (`"1Stevilo"`) in številom (`2`) zopet operator `'+'`, prevajalnik število `2` pretvori v niz `"2"` . Zaradi operatorja `'+'` se niza stakneta. Dobimo niz `"1Stevilo2"`, ki se nato izpiše.

Kaj torej velja za vrednost izraza `x + y?`
- **Če sta x in y niza,**
  se niza stakneta in dobimo niz.
- **Če je *x* število in *y* niz,**
  prevajalnik pretvori število x v niz. Nato se pretvorjeni niz in niz y stakneta in dobimo niz.
- **Če je *x* niz in *y* število,**
  prevajalnik najprej pretvori število `y` v niz. Niza se stakneta, dobimo niz.
- **Če sta *x* in *y* števili,**
  se števili seštejeta in dobimo število.

3.4 Spremenljivke in podatkovni tipi

3.4.1 Spremenljivke

Programi upravljajo s podatki, ki so shranjeni v pomnilniku. V strojnem jeziku se podatki lahko prikličejo samo s klicem na numerični naslov pomnilniške lokacije, kjer je podatek shranjen. V Javi se za naslove podatkov uporabljajo imena namesto številk. Programer si mora zapomniti ime, računalnik pa poskrbi za mesto, kje je podatek shranjen. Takšno ime (naslov), ki služi za klic podatkov shranjenih v spominu, se imenuje **spremenljivka**.
Spremenljivko si lahko predstavljamo kot tablo, na kateri je zapisan nek podatek. Spremenljivka se neposredno nanaša na tablo in posredno na podatek. Med izvajanjem programa se podatek na tabli lahko spreminja. Na tabli (v spremenljivki) bodo različne vrednosti (podatki), toda vedno bo to ista tabla. Vedno lahko pogledamo, kakšno vrednost ima podatek na tabli.
Vsako v programu uporabljeno spremenljivko moramo najaviti oziroma deklarirati. Navadno vse spremenljivke deklariramo skupaj, pred ostalimi stavki (takoj za začetkom metode `main`).

Spremenljivke najavimo z **deklaracijskim stavkom**. Ta je oblike

```
podatkovniTip imeSpremenljivke;
```

Primeri:

```
int stevilo;
double stranica;
String niz;
```

Ob najavi spremenljivke v računalniškem pomnilniku rezerviramo mesto za spremenljivko in povemo, kakšnega tipa je. Kaj je tip, si bomo ogledali v nadaljevanju. Zaenkrat povejmo le, da z njim določimo, kakšne vrednosti lahko hranimo v spremenljivki.

Spremenljivke istega tipa lahko najavimo z istim deklaracijskim stavkom.

```
podatkovniTip spremenljivka1, spremenljivka2, spremenljivka3;
```

Primeri:

```
int enice, desetice, stotice;
double stranica_a, stranica_b, stranica_c;
String niz1, niz2, niz3;
```

Spremenljivki ob najavi lahko priredimo začetno vrednost.

```
podatkovniTip spremenljivka1 = začetnaVrednost;
```

Primeri:

```
int stevilo = 0;
char znak = 'A';

String niz = "Vhodni niz";
```

Spremenljivko deklariramo samo enkrat. Kot smo omenili, to običajno naredimo na samem začetku, ni pa to nujno. Obvezno pa mora biti spremenljivka deklarirana pred prvo uporabo, drugače nam prevajalnik javi napako.

Imena spremenljivk

Vsaka spremenljivka mora imeti svoje ime, po katerem jo ločimo od ostalih. Pri poimenovanju spremenljivk se držimo naslednjih pravil.

Prva črka v imenu spremenljivke naj bo poljubna **mala** črka abecede[6]. Tej črki sledi poljubno zaporedje črk (malih in velikih), števk in podčrtajev (_). Tako so imena npr. `vhodni_niz`, `izhodniNiz`, `niz1`, `niz_2` ipd. Spremenljivkam damo **smiselna imena, ki povedo, kaj je tisto, kar hranimo v spremenljivki.**

Primeri:

```
stranica
znak
naslov
```

Če je ime sestavljeno iz več besed, ne smemo uporabljati presledkov. Zato ga zaradi boljše čitljivosti zapišemo bodisi tako, da vsako naslednjo besedo začnemo z veliko črko (`dolgoIme`), ali pa besede

[6] Pravila jezika Java dopuščajo, da je prvi znak tudi podčrtaj (_) ali dolar ($). Slednja sta v uporabi pri javanskih sistemskih knjižnicah, zato njuna uporaba ni priporočljiva, saj lahko precej oteži iskanje napak.

povežemo s podčrtajem *(dolgo_ime)*. Znotraj programa zaradi preglednosti uporabljajmo samo en način.

Primeri:

```
dolzinaNiza
barvaStranice
ploscina_trikotnika
```

Ne pozabimo, da Java loči med malimi in velikimi črkami. Zato so *ime*, *Ime*, *iMe*, *IME* štiri različna imena.

Prireditveni stavek

Vrednost, zapisano v spremenljivki, spreminjamo (ali nastavimo prvič, če tega ob deklaraciji nismo naredili) s pomočjo prireditvenega stavka. Prireditveni stavek vsebuje prireditveni operator (=). Ta priredi vrednost desne strani spremenljivki na levi strani. Prireditven stavek je oblike

```
spremenljivka = izraz;
```

Tu je *izraz* lahko konstanta, ki jo želimo prirediti spremenljivki ali pa predstavlja izraz z operatorji, metodami, ...

Primer:

```
1  x = 5;
2  y = x + 7;
3  z = x + 8 + y;
```

1. vrstica. Spremenljivki *x* priredimo vrednost *5*.
2. vrstica. Ta prireditveni stavek priredi spremenljivki *y* vrednost izraza na desni. Najprej izračunamo desni del. Ker smo v prejšnji vrstici priredili spremenljivki *x* vrednost *5*, se k *5* prišteje *7*. Na desni strani dobimo *12*. Ta vrednost se zapiše v spremenljivko *y* ali kot tudi rečemo, se priredi spremenljivki *y*.
3. vrstica. Tako kot v drugi vrstici se tudi tu najprej izračuna vrednost izraza na desni strani. Vrednosti *5* se prišteje *8* in tej vrednost spremenljivke *y*. Ta je 12. Vrednost izraza na desni strani prireditvenega stavka je torej *25* in spremenljivki *z* se priredi vrednost *25*.

Če torej uporabimo ime spremenljivke na desni strani (v izrazu), mislimo na vrednost, shranjeno v tej spremenljivki. Tako *a = b* ne pomeni, da je *a* enako *b*, ampak da vrednost shranjeno v *b* priredimo *a*–ju. V spremenljivko *a* torej shranimo vrednost, kot je v spremenljivki *b*.

Konstante

Če spremenljivki ne želimo spreminjati vrednosti, povemo, da je **konstanta**. To naredimo tako, da ob deklaraciji pred tipom spremenljivke dodamo besedo *final*. Imena konstant napišemo z velikimi tiskanimi črkami, zato da jih ločimo od spremenljivk.

```
final podatkovniTip IME_KONSTANTE = vrednostKonstante;
```

Primera:

```
final int POVECAJ_ZA = 4;
final double PI = 3.14;
```

Zgledi

Poglejmo si naslednji program.

PRIMER 3.4.1 – 1: *Spremenljivka.java*

```
1:    public class Spremenljivka {
2:      public static void main(String[] args) {
3:        System.out.println(x);
4:      } // main
5:    } // Spremenljivka
```

Ko ga prevedemo, nam prevajalnik javi:

```
C:\Spremenljivka.java:3: cannot resolve symbol
symbol  : variable x
location: class Spremenljivka
    System.out.println(x);
                       ^
1 error
```

Ker spremenljivke *x* pred uporabo nismo najavili, nam prevajalnik javi napako, saj je ne pozna.

Popravimo program.

PRIMER 3.4.1 – 2: *Spremenljivka.java*

```
1:    public class Spremenljivka {
2:      public static void main(String[] args) {
3:        int x;
4:
5:        System.out.println(x);
6:      } // main
7:    } // Spremenljivka
```

Ko program skušamo prevesti, nam prevajalnik še vedno javlja napako:

```
C:\Spremenljivka.java:5: variable x might not have been
 initialized
    System.out.println(x);
                       ^
1 error
```

Zakaj se program ne prevede? V 3. vrstici smo za spremenljivko *x* rezervirali prostor in določili, da je tipa *int*. Prevajalnik se pritožuje, ker ji pred uporabo (v 5. vrstici) nismo določili vrednosti. Program popravimo tako, da spremenljivki določimo vrednost.

PRIMER 3.4.1 – 3: *Spremenljivka.java*

```
1:    public class Spremenljivka {
2:      public static void main(String[] args) {
3:        int x;
4:        x = 5;
5:
6:        System.out.println(x);
7:      } // main
8:    } // Spremenljivka
```

Program prevedemo in poženemo:

```
5
```

Spremenljivki x iz prejšnjega programa priredimo novo vrednost 7.

PRIMER 3.4.1 – 4: *Spremenljivka1.java*

```
1:    public class Spremenljivka1 {
2:      public static void main(String[] args) {
3:        int x = 5;
4:        System.out.println(x);
5:
6:        int x = 7;
7:        System.out.println(x);
8:      } // main
9:    } // Spremenljivka1
```

Ko prevedemo ta program, nam prevajalnik javi:

```
C:\Spremenljivka1.java:6: x is already defined in
 main(java.lang.String[])
    int x = 7;
        ^
1 error
```

Ker smo v 6. vrstici ponovno definirali spremenljivko x, ki je že definirana v 3. vrstici, se prevajalnik pritoži. Program popravimo takole:

PRIMER 3.4.1 – 5: *Spremenljivka1.java*

```
1:    public class Spremenljivka1 {
2:      public static void main(String[] args) {
3:        int x = 5;
4:        System.out.println(x);
5:
6:        x = 7;
7:        System.out.println(x);
8:      } // main
9:    } // Spremenljivka1
```

Program prevedemo in poženemo:

```
5
7
```

V 3. vrstici deklariramo spremenljivko x in ji priredimo vrednost 5. Nato v 4. vrstici izpišemo trenutno vrednost spremenljivke x. V 6. vrstici priredimo spremenljivki x novo vrednost 7. V 7. vrstici se izpiše vrednost spremenljivke x, ki je sedaj 7.

Spremenimo programa tako, da bo spremenljivka konstanta.

PRIMER 3.4.1 – 6: *Konstanta.java*

```
1:    public class Konstanta {
2:      public static void main(String[] args) {
3:        final int X = 5;
4:
5:        System.out.print(X);
```

```
6:      } // main
7:    } // Konstanta
```

Program prevedemo in poženemo:

```
5
```

Tu je bilo vse v redu. Kaj pa drugi program?

PRIMER 3.4.1 – 7: *Konstanta1.java*

```
1:    public class Konstanta1 {
2:      public static void main(String[] args) {
3:        final int X = 5;
4:        System.out.println(X);
5:
6:        X = 7;
7:        System.out.println(X);
8:      } // main
9:    } // Konstanta1
```

Ko program prevedemo, nam prevajalnik javi:

```
C:\Konstanta1.java:6: cannot assign a value to final variable X
    X = 7;
    ^
1 error
```

Zakaj v tem primeru prevajalnik javi napako? Kot smo povedali na začetku poglavja, konstante uporabljamo, ko spremenljivkam ne želimo spreminjati vrednosti. Če to kljub temu storimo, nas prevajalnik na to opozori.

3.4.2 Podatkovni tipi

Podatkovni tip pove prevajalniku, koliko pomnilnika naj rezervira za spremenljivko in katere so dovoljene operacije, ki jih lahko s temi spremenljivkami izvajamo.

3.4.2.1 CELA ŠTEVILA

Cela števila (ang. integer) označimo s tipom **int**[7]. Seveda ne gre za poljubna cela števila, kot jih poznamo iz matematike, saj v končni pomnilnik računalnika ne moremo shraniti neskončno velikega števila. Spremenljivke tipa *int* v Javi tako lahko zavzamejo vrednosti med -2.147.483.648 in 2.147.488.647.

Primeri:

```
int x = 2;
int y = 3;
int z = 2 * x + y - 1;
```

Nad celimi števili lahko izvajamo vse osnovne računske operacije (seštevanje, odštevanje, množenje in deljenje). Vsota, razlika in produkt dveh celih števil so definirani na običajen način, deljenje pa je

[7] Poznamo še tri podskupine: byte, short in long. Vsi so celoštevilski podatkovni tipi, razlikujejo se po zalogi vrednosti in zasedenosti pomnilniškega prostora. Če nam obseg podatkovnega tipa *int* ne zadošča, uporabimo podatkovni tip *long*.

celoštevilsko. Pri deljenju dveh celih števil torej Java zanemari vsa decimalna mesta (tako je kvocient dveh celih števil vedno celo število). Poleg deljenja pozna Java še operator **%**, ki vrne ostanek pri deljenju (zopet celo število).

OPERATOR	RAZLAGA	UPORABA	REZULTAT
+	vsota	3 + 2	5
-	razlika	3 - 2	1
*	množenje	3 * 2	6
/	celoštevilsko deljenje	3 / 2	1
%	celoštevilski ostanek pri deljenju	3 % 2	1

Zgledi

Napišimo preprost program, ki bo ponazarjal delovanje osnovnih operacij med celimi števili.

PRIMER 3.4.2.1 – 1: *Operacije.java*

```
 1:    public class Operacije {
 2:      public static void main(String[] args) {
 3:        int x = 20;
 4:        int y = 3;
 5:
 6:        System.out.println("x = " + x);
 7:        System.out.println("y = " + y + "\n");
 8:        System.out.println("x + y = " + (x + y));
 9:        System.out.println("x - y = " + (x - y));
10:        System.out.println("x * y = " + (x * y));
11:        System.out.println("x / y = " + (x / y));
12:        System.out.println("x % y = " + (x % y));
13:     } // main
14:   } // Operacije
```

Program prevedemo in poženemo:

```
x = 20
y = 3

x + y = 23
x - y = 17
x * y = 60
x / y = 6
x % y = 2
```

V 3. vrstici najavimo celoštevilski spremenljivki *x* in *y* in jima priredimo vrednost 20 in 3. Zatem izpišemo vrednost, shranjeno v *x*, opremljeno z ustreznim tekstom. Poglejmo kako se to zgodi. Nizu *"x = "* želimo pridružiti vrednost, ki je shranjena v spremenljivki *x*. Ker je ta vrednost celo število, se pretvori v niz (*"20"*), niza (*"x = "* in *"20"*) se stakneta v niz *"x = 20"* in ta se izpiše.
V 7. vrstici izpišemo niz *"y = 3"*, sledi posebni znak (*\n*). Ta povzroči prehod v novo vrsto. Ker smo uporabili metodo *println()*, bomo torej izpustili eno prazno vrstico.
Ker je izraz *x + y* v oklepaju, se najprej izračuna njegova vrednost, to je *23*. Nato prevajalnik spremeni to vrednost v niz (*"23"*) in niza *"x + y = "* ter *"23"* stakne v niz *"x + y = 23"*, ki ga metoda *println* izpiše.
Na podoben način kot v 8. vrstici se izpišeta še niza *"x - y = 17"* in *"x * y = 60"*.
Ker sta tako *x*, kot tudi *y*, celoštevilski spremenljivki, je deljenje v izrazu *x / y* celoštevilsko. Pri celoštevilskemu deljenju števila *20* s *3* dobimo *6*. Prevajalnik to vrednost spremeni v niz (*"6"*). Nato se izvede operacija *"x / y = "+"6"*. Niza se stakneta in izpiše se *"x / y = 6"*.

Ostanek pri celoštevilskemu deljenju števila *20* s *3* je *2*. Prevajalnik to vrednost spremeni v niz *"2"*. Nato se izvede stikanje nizov: *"x % y = " + "2"* in metoda *println()* izpiše *"x % y = 2"*.

Paziti moramo, da v programu ne pride do deljenja z nič, saj se v tem primeru izvajanje programa prekine. Izvajalno okolje nam javi, da je prišlo do izjeme (ang. exception).

PRIMER 3.4.2.1 – 2: *DeljenjeInNic.java*

```
 1:    public class DeljenjeInNic {
 2:      public static void main(String[] args) {
 3:        int x = 20;
 4:        int y = 0;
 5:
 6:        System.out.println("x = " + x);
 7:        System.out.println("y = " + y + "\n");
 8:        System.out.println("x / y = " + (x / y));
 9:      } // main
10:   } // DeljenjeInNic
```

Program prevedemo in poženemo:

```
x = 20
y = 0

java.lang.ArithmeticException: / by zero
  at DeljenjeInNic.main(DeljenjeInNic.java:8)
Exception in thread "main"
```

Po vrsti se izvedejo vse vrstice do 8. V tej pride do deljenja z nič, zato izvajalno okolje prekine izvajanje in sporoči, da se je zgodila tako imenovana izjema.

Sedaj vemo že dovolj, da lahko rešimo bolj "zapleten" problem. Napišimo program, ki bo celo število 123 izpisal obrnjeno, torej kot 321. Želimo, da izpis zgleda tako:

```
Stevilo 123
Obrnjeno stevilo 321
```

Seveda bomo to naredili tako, da bo program deloval pravilno, če bomo uporabili poljubno trimestno število, in ne le za število 123.

PRIMER 3.4.2.1 – 3: *ObrnjenoStevilo.java*

```
 1:    public class ObrnjenoStevilo {
 2:      public static void main(String[] args) {
 3:        int trimestnoStevilo = 123;
 4:        int enice = trimestnoStevilo % 10;
 5:        int dvomestnoStevilo = trimestnoStevilo / 10;
 6:        int desetice = dvomestnoStevilo % 10;
 7:        int stotice = dvomestnoStevilo / 10;
 8:
 9:        System.out.println("Stevilo " + trimestnoStevilo);
10:        System.out.println("Obrnjeno stevilo "
11:                           + enice + desetice + stotice);
12:      } // main
13:    } // ObrnjenoStevilo
```

Kaj bo program izpisal? Pa poglejmo:

Spremenljivki *tromestnoStevilo* priredimo vrednost *123*, torej število, ki ga želimo izpisati v obratnem vrstnem redu.
Sedaj moramo priti do stotic, desetic in enic(vrstice 4. – 7.). Pomagamo si z operatorjema / in %. Najlažje pridemo do enic. Izračunamo ostanek pri deljenju števila z *10* (vrstica 4). Sedaj enic ne potrebujemo več. Zato jih "odrežemo" in dobljeno shranimo v spremenljivko *dvomestnoStevilo* (vrstica 5). Desetice prvotnega števila so v novem številu enice. Te pa že znamo "izluščiti" – zopet si pomagamo z ostankom pri deljenju z *10*. Stotice lahko izračunamo bodisi iz dvomestnega števila z deljenjem z *10* (vrstica 7) ali pa iz trimestnega števila z deljenjem s *100* (*trimestnoStevilo / 100*).
Na koncu še dobljene vrednosti izpišemo.

Seveda zapisani način ni edini. Lahko bi napisali tudi takle program:

PRIMER 3.4.2.1 – 4: *ObrnjenoStevilo1.java*

```
1:    public class ObrnjenoStevilo1 {
2:      public static void main(String[] args) {
3:        int trimestnoStevilo = 123;
4:        int stotice = trimestnoStevilo / 100;
5:        int dvomestnoStevilo = trimestnoStevilo % 100;
6:        int desetice = dvomestnoStevilo / 10;
7:        int enice = dvomestnoStevilo % 10;
8:
9:        System.out.println("Stevilo " + trimestnoStevilo);
10:       System.out.println("Obrnjeno stevilo "
11:                          + enice + desetice + stotice);
12:     } // main
13:   } // ObrnjenoStevilo1
```

Če želimo dobiti stotice, število delimo s *100* (vrstica 4). Uporabili smo celoštevilsko deljenje, saj bomo tako dobili *1*. Torej *123 / 100 = 1*. Nato smo poiskali ostanek števila *123* pri deljenju s *100* (vrstica 5). Dobimo *23*, ki ga delimo z *10* in dobimo *2* desetici (vrstica 6). Sedaj nam mankajo samo še enice. V spremenljivki *dvomestnoStevilo* je *23* , ostanek števila pri deljenju z *10* nam da vrednost *3*.

Načinov, kako sestaviti program, ki reši dani problem, je seveda več. Želimo, da program deluje za poljubno trimestno število. Pomembno je , da program napišemo tako, da lahko spremenimo le vrstico 3 in v spremenljivko *trimestnoStevilo* shranimo kakšno drugo vrednost kot *123*. Na ta način bo naš program deloval, za poljubno trimestno število. Vendar nas moti, da moramo spremenjeni program in ga na novo prevesti. Kako bi vnesli vrednost med samim izvajanjem programa? To si bomo ogledali malo kasneje, v razdelku Branje.

Velikokrat želimo zamenjati vrednost dveh spremenljivk med sabo. Kako to izvedemo, pokažimo na primeru.

PRIMER 3.4.2.1 – 5: *ZamenjavaSpremenljivk.java*

```
1:    public class ZamenjavaSpremenljivk {
2:      public static void main(String[] args) {
3:        int x = 4;
4:        int y = 5;
5:
6:        int t = x;
7:        x = y;
8:        y = t;
9:
10:       System.out.println("Sedaj je x = " + x);
11:       System.out.println("Sedaj je y = " + y + "\n");
```

```
12:
13:         x = y;
14:         y = x;
15:
16:         System.out.println("Sedaj je x = " + x);
17:         System.out.println("Sedaj je y = " + y + "\n");
18:       } // main
19:     } // ZamenjavaSpremenljivk
```

Program prevedemo in poženemo:

```
Sedaj je x = 5
Sedaj je y = 4

Sedaj je x = 4
Sedaj je y = 4
```

Od 6. do 8. vrstice je zamenjava spremenljivk *x* in *y*. Najprej deklariramo novo začasno spremenljivko *t*. Tej priredimo vrednost spremenljivke *x*, tako da *t* dobi vrednost *4* (*x* in *y* ostaneta nespremenjeni). Vrednost spremenljivke *x* imamo shranjeno v spremenljivki *t*. Tako lahko spremenljivki *x* priredimo vrednost spremenljivke *y* in s tem "povozimo" staro vrednost, shranjeno v *x* (*t = 4*, *y = 5*). Spremenljivka *x* ima že ustrezno vrednost. Preostane nam še, da spremenljivki *y* priredimo vrednost *t* in s tem končamo zamenjavo.
V 13. in 14. vrstici smo poskusili zamenjati vrednosti spremenljivk "kar na hitro". Želeli smo, da bi v spremenljivki *x* shranjena vrednost *4* in v spremenljivki *y* vrednost *5*. A to ni rezultat naše zamenjave. Poglejmo, zakaj ne. Vrednost spremenljivke *y* priredimo spremenljivki *x*. V *x* s tem shranimo *4*. To smo tudi želeli. V 14. vrstici pa **trenutno** vrednost spremenljivke *x* priredimo *y*. Torej tudi *y* postane *4*, saj smo v 13. vrstici v *x* shranili *4*.

Kot smo omenili, lahko v spremenljivkah tipa *int* hranimo cela števila velikosti do približno 2 milijard. Kaj se zgodi, če to mejo prekoračimo?

PRIMER 3.4.2.1 – 6: *DolgoStevilo.java*

```
1:    public class DolgoStevilo {
2:      public static void main(String[] args) {
3:        int stevilo1 = 2147483647;
4:        int stevilo2 = 10;
5:        int vsotaStevil = stevilo1 + stevilo2;
6:
7:        System.out.println("2147483647 + 10 = " + vsotaStevil);
8:      } // main
9:    } // DolgoStevilo
```

Prevedemo in poženemo:

```
2147483647 + 10 = -2147483639
```

Dobimo napačen rezultat brez predhodnega opozorila. Izvajalno okolje ne preverja, če so rezultati izrazov v smiselnih mejah. Na to moramo paziti sami.

Kadar nam obseg tipa *int* ne zadošča, uporabimo podatkovni tip *long*, a se v podrobnosti ne bomo spuščali.

3.4.2.2 REALNA ŠTEVILA

Realna števila shranjujemo v spremenljivkah tipa **double**. Sestavljena so iz celega dela in decimalne vrednosti, ki ju loči **decimalna pika** (ne vejica!).

Primeri:

```
double dolzinaX = 3.4;
double cenaEnote = 1.9;
double faktorPodrazitve = 1.1;
double novaCenaEnote = cenaEnote * faktorPodrazitve;
```

Spremenljivke tipa *double* lahko zavzamejo vrednosti med $4.9*10^{-324}$ in $1.7976931348623157*10^{308}$. Zapišemo jih lahko tudi v eksponentni obliki z znakom E, npr. 12.16 E20, kar pomeni število $12.16 * 10^{20}$.

Tudi nad realnimi števili lahko izvajamo vse osnovne operacije (seštevanje, odštevanje, množenje in deljenje). Vsota, razlika, produkt in deljenje dveh realnih števil so definirani na običajen način. V realnih aritmetičnih izrazih nikoli ne pride do napake med izvajanjem (izjeme), saj način kodiranja omogoča zapis neskončnih in neizračunljivih vrednosti, ki se pripetijo pri deljenju z ničlo, neskončnost pa se uporabi tudi pri prekoračitvi obsega.
Če operator (+, -, /, *) povezuje celo in realno število, se celo število najprej pretvori v realno. Rezultat je realno število (tudi v primeru, ko je decimalni del enak nič!).

Primeri:

```
12.9 / 3 = 4.3
13.4 / 6.7 = 2.0
6.7 - 4.6 = 2.1000000000000005
1.4 * 0 = 0.0
3.6 / 0 = Infinity
```

Ker se realna števila hranijo v dvojiškem zapisu in zaradi omejenega prostora pride pri delu z njimi do zaokrožitvenih napak. Pri kakršni koli uporabi realnega tipa moramo te napake vzeti v zakup (kot se je v zgornjem primeru izraz 6.7 - 4.6 izračunal v 2.1000000000000005 namesto v 2.1).

OPERATOR	RAZLAGA	UPORABA	REZULTAT
+	vsota	6.7 + 2.3	9.0
-	razlika	6.7 - 2.3	4.4
*	množenje	6.7 * 2.3	15.409999999999998
/	deljenje	6.7 / 2.3	2.91304347826087

Uporabo operacij si bomo pogledali na nekaj zgledih.

Zgledi

Kolesarja

Zoran in Franci sta v treh dneh prikolesarila do kraja A. Zoran je prvi dan prekolesaril 32,1 kilometrov, drugi dan 21,8 in tretji dan 12,4 kilometrov. Franci pa je vsak dan prekolesaril tretjino celotne poti. Izračunajmo, koliko kilometrov je Franci prevozil vsak dan.

Potrebovali bomo dve realni spremenljivki. V eno bomo shranili dolžino poti do kraja A (vsoto Zoranovih dnevno prevoženih kilometrov), drugo pa potrebujemo za izračun Francijeve dnevno prevožene poti, ki je tretjina celotne poti.

PRIMER 3.4.2.2 – 1: *Kolesarja.java*

```
 1:    public class Kolesarja {
 2:      public static void main(String[] args) {
 3:        // razdalja do kraja A
 4:        double dolzinaPoti = 32.1 + 21.8 + 12.4;
 5:
 6:        // dolzina Francijeve enodnevne prekolesarjene poti
 7:        double tretjinaPoti = dolzinaPoti / 3;
 8:
 9:        System.out.println("Franci je dnevno prevozil " +
10:                           tretjinaPoti + " km.");
11:      } // main
12:    } // Kolesarja
```

Program prevedemo in poženemo:

```
Franci je dnevno prevozil 22.1 km.
```

Plačilo bencina

Bencin se je podražil za 0,2 evra. Pred podražitvijo, ko je bil bencin še 1,2 evra, smo za poln tank plačali 40,5 evrov. Izračunajmo, koliko bomo po podražitvi plačali za poln tank.

Novo ceno bomo izračunali tako, da bomo volumen tanka pomnožili z novo ceno za liter bencina. Volumen tanka dobimo tako, da staro ceno polnega tanka delimo s staro ceno za liter bencina.

PRIMER 3.4.2.2 – 2: *PlaciloBencina.java*

```
 1:    public class PlaciloBencina {
 2:      public static void main(String[] args) {
 3:        double staraCena = 1.2;
 4:        double novaCena = staraCena + 0.2;
 5:        double stariZnesek = 40.5;
 6:        double noviZnesek;
 7:        double volumenTanka = stariZnesek / staraCena;
 8:        // cena bencina po podrazitvi
 9:        noviZnesek = volumenTanka * novaCena;
10:
11:       System.out.println("Za poln tank bomo placali " +
12:                          noviZnesek + " evrov.");
13:      } // main
14:    } // PlaciloBencina
```

Program prevedemo in poženemo:

```
Za poln tank bomo placali 47.25 evrov.
```

Barvanje sob

Soba v obliki kvadra je dolga 5,6 m, široka 4,2 m in visoka 4,5 m. Pobarvati želimo stene in strop. Kilogram barve stane 1200 tolarjev in zadošča za 5 m^2. Izračunajmo, koliko denarja bomo potrebovali za barvanje sobe.

Za izračun površine ki jo moramo prebarvati uporabimo naslednjo enačbo:
*povrsina = dolzinaSobe * sirinaSobe + 2* (dolzinaSobe * visinaSobe) + 2*(sirinaSobe* visinaSobe)*
oz. če izpostavimo skupni faktor v drugem delu enačbe dobimo:
*povrsina = dolzinaSobe * sirinaSobe + 2 * visinaSobe * (dolzinaSobe + sirinaSobe).*

PRIMER 3.4.2.2 – 3: *BarvanjeSobe.java*

```
1:    public class BarvanjeSobe {
2:      public static void main(String[] args) {
3:        double dolzinaSobe = 5.6;
4:        double sirinaSobe = 4.2;
5:        double visinaSobe = 4.5;
6:        double povrsina, cenaBarve;
7:        final int CENAKGBARVE = 1200;
8:        final int IZDATNOST = 5;//za koliko k. metrov je kg  barve
9:        // povrsina sobe brez tal
10:       povrsina = dolzinaSobe * sirinaSobe + 2 * visinaSobe *
11:                  (dolzinaSobe + sirinaSobe);
12:
13:       // cena barve
14:       cenaBarve = CENAKGBARVE * povrsina / IZDATNOST;
15:
16:       System.out.println("Za barvanje sobe potrebujemo " +
17:                          cenaBarve + " tolarjev.");
18:     } // main
19:   } // BarvanjeSobe
```

Program prevedemo in poženemo:

```
Za barvanje sobe potrebujemo 26812.8 tolarjev.
```

Popravimo program tako, da bo znesek zaokrožili na tolar natančno. Pomagali si bomo z metodo *round()* iz razreda *Math*. Več o tem razredu bomo izvedeli v nadaljevanju, sedaj pa povejmo le, da metoda *round()* zaokroži realno število na najbližje celo število, pokličemo pa jo z *Math.round(steviloZaZaokrozanje)*.

PRIMER 3.4.2.2 – 4: *BarvanjeSobe1.java*

```
1:    public class BarvanjeSobe1 {
2:      ... (glej program BarvanjeSobe)

15:
16:       System.out.println("Za barvanje sobe potrebujemo " +
17:                          Math.round(cenaBarve) + " tolarjev.");
18:     } // main
19:   } // BarvanjeSobe1
```

Program prevedemo in poženemo:

```
Za barvanje sobe potrebujemo 26813 tolarjev.
```

Plačilo mesečne vozovnice

Mesečna vozovnica za avtobus stane 3000 tolarjev. Izračunajmo, koliko bo stala po 5–odstotni podražitvi. Ceno bomo ustrezno zaokrožili.

PRIMER 3.4.2.2 – 5: *MesecnaVozovnica.java*

```
1:    public class MesecnaVozovnica {
2:      public static void main(String[] args) {
3:        int cenaMesecne = 3000;
4:        double podrazitev = 5 / 100;
5:        double novaCena;
6:
7:        // cena mesecne po podrazitvi
```

```
 8:        novaCena = cenaMesecne + cenaMesecne * podrazitev;
 9:
10:        System.out.println("Mesecna bo stala " +
11:                           Math.round(novaCena) + " tolarjev.");
12:      } // main
13:    } // MesecnaVozovnica
```

Program prevedemo in poženemo:

```
Mesecna bo stala 3000 tolarjev.
```

Ker je rezultat enak prvotni ceni mesečne vozovnice pred podražitvijo, nekaj ni v redu. Kje smo se zmotili?

Razlog tiči v četrti vrstici. V prireditvenem stavku se najprej izračuna desna stran. Izračunamo *5 / 100.* Ker sta števili celi, je to *0*. Spremenljivka *podrazitev* je realno število, zato se celo število (*0*) pretvori v realno, to je v *0.0*. Ta vrednost se priredi spremenljivki *podrazitev*.

Kadar želimo dve celi števili deliti na običajen način (da dobimo realni kvocient), je pred deljenjem potrebno vsaj eno od števil pretvoriti v realno. To pa lahko naredimo na več načinov.

```
5 / 100.0 // celemu številu dodamo decimalno vrednost (.0)
(double)5 / 100 // pred število, ki ga želimo pretvoriti,
napišemo željen tip v oklepaju
```

Več o spremembah med tipi si bomo ogledali v posebnem razdelku, sedaj pa popravimo naš program.

PRIMER 3.4.2.2 – 6: *MesecnaVozovnica.java*

```
 1:    public class MesecnaVozovnica {
 2:      public static void main(String[] args) {
 3:        int cenaMesecne = 3000;
 4:        double podrazitev = 5. / 100;
 5:        double novaCena;
 6:
 7:        // cena mesecne po podrazitvi
 8:        novaCena = cenaMesecne + cenaMesecne * podrazitev;
 9:
10:        System.out.println("Mesecna bo stala " +
11:                           Math.round(novaCena) + " tolarjev.");
12:      } // main
13:    } // MesecnaVozovnica
```

Program prevedemo in poženemo:

```
Mesecna bo stala 3150 tolarjev.
```

Dolžina poti

Prehodili smo 3/5 poti, do cilja je še 16 kilometrov. Izračunajmo dolžino celotne poti.

Za izračun celotne poti uporabimo enačbo:
delezPoti = 2 / 5.0 (ne prevožene)
*delezPoti * dolzinaPoti = 16* oz.
dolzinaPoti= 16 / delezPoti

PRIMER 3.4.2.2 – 7: *DolzinaPoti.java*

```
1:    public class DolzinaPoti {
2:      public static void main(String[] args) {
3:        // delez poti do cilja
4:        double delezPoti = 2 / 5.0;
5:        int ostane = 16;
6:        double dolzinaPoti;
7:
8:        // dolzina prehojene poti
9:        dolzinaPoti = ostane / delezPoti;
10:
11:       System.out.println(dolzinaPoti);
12:     } // main
13:   } // DolzinaPoti
```

Program prevedemo in poženemo:

```
40.0
```

3.4.2.3 LOGIČNE VREDNOSTI

Logične vrednosti označimo s tipom **boolean**. V spremenljivki tipa *boolean* lahko shranjujemo le vrednosti pravilno **(true)** ali napačno **(false)**.

Primera:

```
boolean trditev = true;
boolean konec = false;
```

Poglejmo preprost primer uporabe podatkovnega tipa *boolean*.

PRIMER 3.4.2.3 – 1: *TipBoolean.java*

```
1:    public class TipBoolean {
2:      public static void main(String[] args) {
3:        boolean da = true;
4:        boolean ne = false;
5:
6:        System.out.println(da);
7:        System.out.println(ne);
8:      } // main
9:    } // TipBoolean
```

Program prevedemo in poženemo:

```
true
false
```

Osnovne logične operacije

Logična operacija je operacija med logičnimi vrednostmi, katere rezultat je logična vrednost true (pravilno) ali false (napačno). Logične vrednosti lahko združujemo z logičnimi operatorji: in, ali ter negacijo. Naj bosta a in b logična izraza. Poglejmo si logične operacije med njima

LOGIČNA OPERACIJA	POMEN	OPIS
!a	logična negacija	Vrne negacijo.
a && b	logični in	Vrne true, kadar sta oba pogoja resnična.
a || b	logični ali	Vrne false, kadar je vsaj en pogoj resničen.

POGOJ	REZULTAT
true && true	true
true && false	false
false && true	false
false && false	false

POGOJ	REZULTAT
true || true	true
true || false	true
false || true	true
false || false	false

Glede prioritete veljajo enaka pravila kot pri matematiki, zato se && izvede pred ||. Kadar nismo prepričani, kakšna je prioriteta operatorjev, uporabimo oklepaje, da zagotovimo vrednotenje v želenem vrstnem redu. Opozoriti velja, da se izrazi, v katerih uporabljamo operatorja && in ||, vrednotijo vedno od leve proti desni in sicer toliko časa, da je vrednost celega izraza določena.

Naj bosta **a** in **b** dva logična izraza. Če ima izraz **a** vrednost **false**, potem ima tudi izraz **a && b** vrednost **false**. Če ima izraz **a** vrednost **true**, potem ima tudi izraz **a || b** vrednost **true**.
Izraza **b** Java v teh dveh primerih sploh ne izračuna.

Primerjalne operacije

S primerjalnimi operatorji primerjamo dva izraza. Rezultat primerjalnih operatorjev je vedno tipa *boolean*. Najpogosteje se z njimi srečujemo v zankah in pogojnih stavkih. Naj bosta a in b izraza, katerih vrednosti sta števili. Poglejmo si primerjalne operacije med njima

PRIMERJALNA OPERACIJA	POMEN	OPIS
`a == b`	enako	Pogoj je resničen, če sta vrednosti obeh izrazov enaki.
`a != b`	različno	Pogoj je resničen, če sta vrednosti obeh izrazov različni.
`a > b`	večje	Pogoj je resničen, če je vrednost levega izraza večja od vrednosti desnega izraza.
`a >= b`	večje ali enako	Pogoj je resničen, če je vrednost levega izraza večja ali enaka od vrednosti desnega izraza.
`a < b`	manjše	Pogoj je resničen, če je vrednost levega izraza manjša od vrednosti desnega izraza.
`a <= b`	manjše ali enako	Pogoj je resničen, če je vrednost levega izraza manjša ali enaka od vrednosti desnega izraza.

3.4.2.4 ZNAKI

V spremenljivki lahko hranimo tudi en znak. V ta namen uporabljamo podatkovni tip **char** (ang. character). To so črke, številke, presledek, ločila, ... Znake pišemo med enojnimi narekovaji.

Primer:

```
char znak = 'n';
char oznakaVrat = 'N';
```

PRIMER 3.4.2.4 – 1: *IzpisiZnak.java*

```
1:    public class IzpisiZnak {
2:      public static void main(String[] args) {
3:        char znak_a = 'a';
4:
5:        System.out.println(znak_a);
6:      } // main
7:    } // IzpisiZnak
```

Program prevedemo in poženemo:

```
a
```

Java obravnava znake kot "majhna" števila. Zato smo v spremenljivko *znak_a* pravzaprav shranili kodo znaka mali a. Ta koda je neko naravno število. Na vsako spremenljivko tipa *char* lahko gledamo bodisi kot na znak, bodisi kot na število. Ker na znake lahko gledamo tudi kot na števila, smemo napisati tudi tak program.

PRIMER 3.4.2.4 – 2: *ZnakKotStevilo.java*

```
1:   public class ZnakKotStevilo {
2:       public static void main(String[] args) {
3:           char znak_a = 'a';
4:
5:           System.out.println(znak_a + znak_a);
6:       } // main
7:   } // ZnakKotStevilo
```

Program prevedemo in poženemo:

```
194
```

Kot smo že omenili, v spremenljivko tipa *char* pravzaprav shranimo kodo znaka (3. vrstica). Zato smo v spremenljivko *znak_a* shranili kodo znaka mali a, ki je neko naravno število (v našem primeru *97*). In ker je med dvema "številoma" operator *+*, se izvede operacija seštevanja, kot smo že navajeni. Torej se seštejeta dve številski vrednosti malega znaka a in vsota se izpiše.

Ker v spremenljivko tipa *char* shranimo kodo znaka, znake primerjamo tako kot števila z operatorjem enakosti (==). S tem se v bistvu primerja koda znaka. Ker ima vsak znak svojo kodo, lahko dobimo vrednost *true* samo, kadar primerjamo enaka znaka, drugače dobimo *false*. Poglejmo si primer.

PRIMER 3.4.2.4 – 3: *EnakaZnaka.java*

```
1:    public class EnakaZnaka {
2:      public static void main(String[] args) {
3:        char znak_a = 'a';
4:        char znak_A = 'A';
5:
6:        System.out.println(znak_a == znak_A);
7:        System.out.println(znak_a == 'a');
8:      } // main
9:    } // EnakaZnaka
```

Program prevedemo in poženemo:

```
false
true
```

Kot smo že povedali, Java primerja znake po njihovih celoštevilčnih kodah. Ker njuna koda ni enaka, se izpiše *false*. V 7. vrstici primerjamo vrednost spremenljivke *znak_a* in znakovne *konstanta* *'a'*. Ker je njuna celoštevilčna koda enaka, se izpiše *true*.

3.4.2.5 NIZI

Pogosteje kot posamezne znake uporabljamo nize znakov (ang. string). Za delo z nizi ima Java definiran tip *String*[8]. Željen niz navedemo med dvojnimi narekovaji.

Primera:

```
String niz = "Pozdravljeni!";
String presledek = " ";
```

Dva niza staknemo (združimo) z operatorjem za združevanje (*+*). To smo že večkrat srečali pri izpisovanju.

PRIMER 3.4.2.5 – 1: *Zdruzevanje.java*

```
1:    public class Zdruzevanje {
2:      public static void main(String[] args) {
3:        String niz1 = "Dobro ";
4:        String niz2 = "jutro.";
5:        String niz3 = niz1 + niz2;
6:
7:        System.out.println(niz3);
8:      } // main
9:    } // Zdruzevanje
```

Program prevedemo in poženemo:

```
Dobro jutro.
```

Oglejmo si, kako lahko ugotovimo, da sta dva niza enaka (ang. equals)? Nizov ne primerjamo z operatorjem ==, ampak s posebno metodo. V razredu *String* najdemo metodo *equals()*, ki vrača rezultat tipa *boolean*.

PRIMER 3.4.2.5 – 2: *EnakaNiza.java*

```
1:    public class EnakaNiza {
2:      public static void main(String[] args) {
3:        String niz1 = "Ana";
4:        String niz2 = "Zdravko";
5:        boolean trditev = niz1.equals(niz2);
6:
7:        System.out.println(trditev);
8:      } // main
9:    } // EnakaNiza Program prevedemo in poženemo:
```

```
false
```

V 5. vrstici smo uporabili metodo *equals()*. Primerjali smo *niz1* in *niz2* , ker sta različna nam metoda vrne *false*.

Večkrat se nam porodi ideja da bi niza primerjali (ang. compare) po abecednem redu. Na voljo imamo metodo *compareTo()*, ki vrača celoštevilčno vrednost. Poglejmo, kako deluje, oziroma, kako jo uporabljamo.

PRIMER 3.4.2.5 – 3: *PrimerjanjeNizov.java*

[8] Dejansko gre za razred String. To je tudi razlog, da se ime (String) začne z veliko začetnico. A Java nam (vsaj navidez) omogoča, da z nizi delamo tako kot s števili, zato bomo nekaj časa govorili kar o tipu String, čeprav to formalno ni povsem prav.

```
1:    public class PrimerjanjeNizov {
2:      public static void main(String[] args) {
3:        String niz1 = "Ana";
4:        String niz2 = "Zdravko";
5:        int vrednost = niz1.compareTo(niz2);
6:
7:        System.out.println(vrednost);
8:      } // main
9:    } // PrimerjanjeNizov
```

Program prevedemo in poženemo:

```
-25
```

Poglejmo, kaj nam pove metoda *compareTo()*. Denimo, da primerjamo *niz1* in *niz2* z *niz1.compareTo(niz2)*. Če je niz v spremenljivki *niz1* po abecedi pred nizom, ki je v spremenljivki *niz2*, vrne metoda negativno število. Če je *niz1* enak *niz2*, vrne metoda vrednost 0. Če je *niz1* po abecedi za *niz2*, vrne metoda pozitivno število.

3.4.3 Pretvarjanje med tipi

Večkrat tip vrednosti ne ustreza željnemu. Zato ga je potrebno pretvoriti v drugi tip. Včasih to stori prevajalnik, včasih pretvorbo zahteva programer. Avtomatične pretvorbe (tiste, ki jih opravi prevajalnik) iz tipa *double* in tipa *int* v tip *String* ter iz tipa *int* v tip *double* smo že srečali v več zgledih.

Pretvarjanje iz tipa int v tip double

Če želimo celo število pretvoriti v realno, napišemo pred celim številom v okroglih oklepajih *double*.

```
(double)celoStevilo
```

Število tipa *int* se pretvori v število tipa *double* tako, da se zadaj doda .0 (decimalni del).

Zgled

Plačilo sira

V trgovini stane 1 kilogram sira 1150 tolarjev. Majda kupi 20 dekagramov sira. Izračunajmo koliko bo Majda plačala za sir.

PRIMER 3.4.3 – 1: *CenaSira.java*

```
1:    public class CenaSira {
2:      public static void main(String[] args) {
3:        int teza1 = 100; // teza v dekagramih
4:        int cena1 = 1150; // cena za 1 kilogram
5:        int teza2 = 20; // teza v dekagramih
6:
7:        // cena sira
8:        double cena2 = teza2 * cena1 / (double)teza1;
9:
10:       System.out.println(teza2 + " dag sira stane " +
11:                          cena2 + " tolarjev.");
12:     } // main
13:   } // CenaSira
```

Program prevedemo in poženemo:

```
20 dag sira stane 230.0 tolarjev.
```

Ceno sira dobimo tako, da ceno za 1 dekagram (*cena1 / (double)teza1 = 11.5*) pomnožimo s težo kupljenega sira. Če želimo pri deljenju dveh celih števil dobiti realno število *(11.5)*, moramo vsaj eno od števil spremeniti v realno. Z *(double)teza1* smo celoštevilčno vrednost, ki je shranjena v spremenljivki *teza1*, spremenili v realno število. Ko se izračuna desni del, se priredi realni spremenljivki *cena2*.
Na koncu še izpišemo vrednosti spremenljivk *teza2* in *cena2* opremljeni z ustreznim besedilom.

Pretvarjanje iz tipa double v int

Če želimo realno število pretvoriti v celo, napišemo pred realnim številom v okroglih oklepajih *int*.

```
(int)realnoStevilo
```

Sprememba tipa iz realnega v celo število se opravi tako, da se preprosto odreže decimalni del.

Zgleda

Višina vode

V valjast sod s premerom 100 centimetrov nalijemo 100 litrov vode. Izračunajmo, kolikšno višino doseže voda. Rezultat naj bo zaokrožen na eno decimalno mesto.

Pri računanju si pomagamo s konstanto *PI* ter metodo *pow()*, ki ju najdemo v razredu *Math*.
V konstanti *PI* je shranjena vrednost π, uporabimo jo z ukazom *Math.PI*. Metoda pow( ) vrne eksponent števila. Z ukazom *Math.pow(osnova, eksponent)* dobimo število $osnova^{eksponent}$.
Da bomo število zaokrožili na eno decimalko število najprej pomnožimo z 10, ga zaokrožimo ter ga spremenimo v tip int. S tem smo odrezali odvečne decimalne vrednosti. Nato število delimo z 10 in tako dobimo število, ki je zaokroženo na eno decimalno mesto.

PRIMER 3.4.3 – 2: *VisinaVode.java*

```
1:    public class VisinaVode {
2:      public static void main(String[] args) {
3:        double volumenVode = 100000; // volumen vode v cm3
4:        double polmerSoda = 50; // polmer soda v cm
5:        double visinaVode; // visina vode v cm
6:        double ploscinaKroga; //  ploscina sodovega dna v cm2
7:
8:        // visina vode
9:        ploscinaKroga = Math.PI * Math.pow(polmerSoda, 2);
10:       visinaVode = volumenVode / ploscinaKroga;
11:
12:       // zaokrozimo na eno decimalno mesto
13:       visinaVode = (int)(Math.round(visinaVode * 10));
14:       visinaVode = visinaVode / 10;
15:
16:       System.out.println("Voda stoji " + visinaVode +
17:                          " centimetrov visoko.");
18:     } // main
19:   } // VisinaVode
```

Program prevedemo in poženemo:

```
Voda stoji 12.7 centimetrov visoko.
```

Višino vode izračunamo s pomočjo formule za volumen valja. Volumen vode delimo s ploščino sodovega dna. Ploščino dna izračunamo po formuli za ploščino kroga z `Math.PI * Math.pow(polmerSoda, 2)`. `Math.PI` vrne vrednost π, `Math.pow(polmerSoda, 2)` vrne $polmerSoda^2$. Ko se izračuna desna stran, se vrednost priredi spremenljivki `visinaVode`.
Sedaj moramo še "odrezati" odvečne decimalke. To naredimo na naslednji način. Realno število `visinaVode` najprej pomnožimo s 10. S tem smo decimalno piko prestavili za eno mesto v desno. Dobljeno vrednost zaokrožimo z Math.round() in decimalni del "odrežemo" tako, da realno število pretvorimo v celo število z `(int)`. Sedaj moramo deliti še z 10, da dobimo nazaj prvotno vrednost, vendar zaokroženo na eno decimalko natančno.
Nato še izpišemo vrednost spremenljivke `visinaVode` z ustreznim tekstom.

Dolžina ograje

Ploščina kvadratnega vrta meri 119,40 metrov. Izračunajmo, koliko metrov ograje bomo potrebovali za ograditev vrta. Rezultat naj bo zaokrožen na dve decimalni mesti.

Tokrat bomo potrebovali metodo `sqrt()`, ki jo prav tako najdemo v razredu `Math`. Metoda `sqrt()` vrne kvadratni koren števila. Z ukazom `Math.sqrt(število)` torej dobimo kvadratni koren števila `število`.
Pri zaokroževanju na določeno število mest si pomagamo podobno kot v prejšnjem primeru. Število najprej pomnožimo s `100` ter ga celoštevilsko zaokrožimo. S tem smo odrezali odvečne decimalne vrednosti. Nato število delimo s `100` in dobimo število, ki je zaokroženo na dve decimalni mesti.

PRIMER 3.4.3 – 3: *DolzinaOgraje.java*

```
 1:    public class DolzinaOgraje {
 2:      public static void main(String[] args) {
 3:        double ploscinaVrta = 119.40;
 4:        double dolzinaOgraje;
 5:        double a; // dolzina (oziroma sirina) vrta
 6:
 7:        // izracunamo dolzino (oziroma sirino) vrta
 8:        a = Math.sqrt(ploscinaVrta);
 9:
10:        // izracunamo dolzino ograje
11:         dolzinaOgraje = 4 * a;
12:
13:        // zaokrozimo na dve decimalni mesti
14:        dolzinaOgraje = (int)(Math.round(dolzinaOgraje * 100));
15:        dolzinaOgraje = dolzinaOgraje / 100;
16:
17:        System.out.println("Potrebovali bomo " + dolzinaOgraje +
18:                           " metrov ograje.");
19:      } // main
20:    } // DolzinaOgraje
```

Program prevedemo in poženemo:

```
Potrebovali bomo 43.71 metrov ograje.
```

Pretvarjanje iz tipa String v tip double

Če je v nizu zapisano realno število, ga v število pretovorimo z metodo *parseDouble()*, ki je v razredu *Double*. Če izvedemo metodo *Double.parseDouble(niz)*, dobimo število, kot je zapisano v nizu.

Zgled

Niz pretvorimo v realno število

Kako bi iz niza *"8.70"* naredili realno število *8,7*?

PRIMER 3.4.3 – 4: *NizRealnoStevilo.java*

```
1:    public class NizRealnoStevilo {
2:      public static void main(String[] args) {
3:        String niz = "8.70";
4:        double stevilo = Double.parseDouble(niz);
5:
6:        System.out.println(stevilo);
7:      } // main
8:    } // NizRealnoStevilo
```

Program prevedemo in poženemo:

```
8.7
```

Če bi v niz napisali *"8,70"*, bi namesto izpisa rezultata dobili sporočilo o napaki. Namreč v nizu ni pravilno zapisano decimalno število, saj smo uporabili decimalno vejico in ne pike.

Pretvarjanje iz tipa String v tip int

Če so v nizu le števke (ter kot prvi znak morebiti + ali -), ga lahko pretvorimo v celo število. V ta namen imamo v razredu *Integer* metodo *parseInt()*

Zgled

Razlika števil

V dveh celoštevilskih spremenljivkah imamo shranjeni dve števki. Izračunajmo razliko med največjim in najmanjšim številom, ki ju lahko sestavimo iz teh dveh števk.

Pri računanju si pomagamo z metodo *abs()*, ki jo najdemo v razredu *Math*. Metoda *abs()* vrne absolutno vrednost števila.

PRIMER 3.4.3 – 5: *RazlikaStevil.java*

```
1:    public class RazlikaStevil {
2:      public static void main(String[] args) {
3:        int prvaStevka = 2;
4:        int drugaStevka = 4;
5:
6:        // sestavimo oz. staknemo stevili
7:        String prviNiz = "" + prvaStevka + drugaStevka;
8:        String drugiNiz = "" + drugaStevka + prvaStevka;
9:
10:       // niza pretvorimo v celi stevili
11:       int prvoStevilo = Integer.parseInt(prviNiz);
12:       int drugoStevilo = Integer.parseInt(drugiNiz);
13:
14:       // izracunamo razliko
```

```
15:         int razlika = prvoStevilo - drugoStevilo;
16:
17:         System.out.println("Razlika sestavljenih stevil je " +
18:                            Math.abs(razlika) + ".");
19:       } // main
20:     } // RazlikaStevil
```

Program prevedemo in poženemo:

```
Razlika sestavljenih stevil je 18.
```

Iz dveh števk sestavimo število tako, da ju staknemo, pri tem si pomagamo z združitvenim operatorjem *+*. S tem dobimo niz (prviNiz = "24" in drugiNiz = "42"). V 11. vrstici in 12. vrstici iz niza dobimo celo število s pomočjo metode *Integer.parseInt()*. Niz *prviNiz* se v vrstici 11 najprej pretvori v celo število, ki se priredi celoštevilski spremenljivki *prvoStevilo*. Niz *drugiNiz* se s klicem metode *Integer.parseInt(drugiNiz)* pretvori v celo število, ki se priredi celoštevilski spremenljivki na levi strani *drugoStevilo* (vrstica 12). Sedaj lahko izračunamo razliko med številoma. Razliko priredimo spremenljivki *razlika*. Sledi še izpis absolutne vrednosti spremenljivke *razlika*, opremljen z ustreznim tekstom.

Pri pretvarjanju iz številskih vrednosti v niz si pomagamo z združitvenim operatorjem *+* kot smo to že večkrat naredili. Če torej število seštejemo s praznim nizom (*""*), bomo dobili ustrezno predstavitev števila kot niz.

3.4.4 Branje

Če želimo, da uporabnik sam vnaša določene podatke v naš program, moramo spoznati način, kako lahko spremenljivki priredimo vrednost, ki jo uporabnik vnese s pomočjo tipkovnice. Najprej si bomo pogledali, kako bi prebrali in izpisali niz, ki ga vpiše uporabnik.

PRIMER 3.4.4 – 1: *BranjeNiza.java*

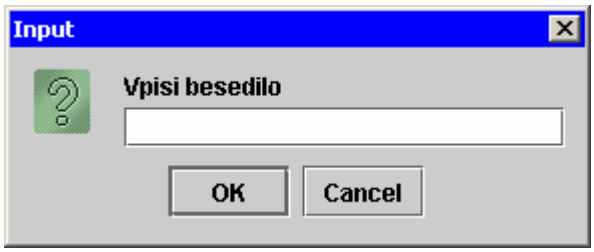
```
1:    import javax.swing.*;
2:
3:    public class BranjeNiza {
4:      public static void main(String[] args) {
5:        String beri;
6:
7:        beri = JOptionPane.showInputDialog("Vpisi besedilo");
8:        System.out.println(beri);
9:      } // main
10:   } // BranjeNiza
```

V 1. vrstici povemo, da bomo uporabljali določeno knjižnico, v kateri so metode, ki jih potrebujemo za branje. To naredimo z besedo *import*. Zaenkrat si glede knjižnic zapomnimo le, da moramo takrat, ko beremo na spodaj opisani način, v prvi vrstici dodati stavek *import javax.swing.*;*. Zatem deklariramo spremenljivko tipa String, ki jo bomo potrebovali za branje niza.
V 7. vrstici prikaže sporočilno okno s tekstom, ki je med narekovaji. V našem primeru se bo pokazalo sporočilno okno s tekstom *Vpisi besedilo*. To okno je namenjeno vnosu podatkov, ki so tipa *String*. Ko uporabnik zapiše niz in klikne na gumb OK, se vrednost niza prenese v niz *beri*.
Klic metode

JOptionPane.showInputDialog("Izpisano sporocilo")

torej prikaže sporočilo z ustreznim tekstom in vnosno vrstico, kamor lahko vnesemo niz.. Slednji je rezultat klica te metode.Za konec (vneseni) niz še izpišemo.

Ko program prevedemo in poženemo, se nam prikaže okno.



Tu vpišemo željen tekst. Natipkajmo *Preberi me*

In izpiše se (8. vrstica):

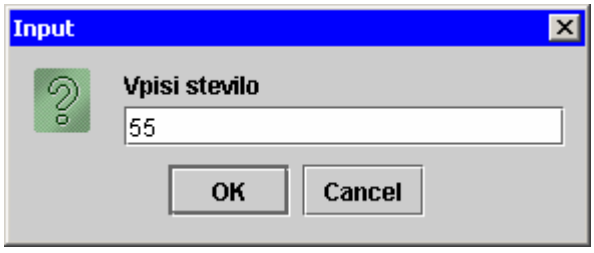
```
Preberi me
```

Omenjeno metodo lahko uporabimo tudi za branje števil. Poglejmo si, kako bi prebrali in izpisali celo število. Podatek bomo prebrali kot niz, saj ga drugače sploh še ne znamo. Nato bomo uporabili metodo *Integer.parseInt()*. Z njo bomo iz niza dobili število.

PRIMER 3.4.4 – 2: *BranjeStevil.java*

```
 1:    import javax.swing.*;
 2:
 3:    public class BranjeStevil {
 4:      public static void main(String[] args) {
 5:        String beri;
 6:        int stevilo;
 7:
 8:        beri = JOptionPane.showInputDialog("Vpisi celo stevilo");
 9:        stevilo = Integer.parseInt(beri);
10:        System.out.println(beri);
11:      } //main
12:    } // BranjeStevil
```

V okno vpišemo željno število. V našem primeru je *55*.



Izpiše se:

```
55
```

Sedaj že vemo dovolj, da lahko program, s katerim smo trimestno število izpisali obrnjeno, spremenimo tako, da bomo število prebrali. Na ta način bo naš program uporaben za obračanje poljubnega trimestnega števila, saj se bo šele ob zagonu programa "odločilo", katero trimestno število obračamo.

PRIMER 3.4.4 – 3: *ObrnjenoStevilo2.java*

```
 1:    import javax.swing.*;
 2:    public class ObrnjenoStevilo2 {
 3:      public static void main(String[] args) {
 4:        String beri =
 5:            JOptionPane.showInputDialog("Trimestno število: ");
 6:        int trimestnoStevilo = Integer.parseInt(beri);
 7:        int enice = trimestnoStevilo % 10;
 8:        int dvomestnoStevilo = trimestnoStevilo / 10;
 9:        int desetice = dvomestnoStevilo % 10;
10:        int stotice = dvomestnoStevilo / 10;
11:
12:        System.out.println("Stevilo " + trimestnoStevilo);
13:        System.out.println("Obrnjeno stevilo "
14:                           + enice + desetice + stotice);
15:      } // main
16:    } // ObrnjenoStevilo2
```

Izpiše se:

```
Stevilo 123

Obrnjeno stevilo 321
```

Vidimo, da smo le dodali vrstico 1 (napovedali uporabo knjižnice javax.swing) ter v vrstici 4 in 5 prebrali število kot niz. Sledi še popravek v 6. vrstici kjer namesto števila 123 v spremenljivko *trimestnoStevilo* shranili število, ki ga dobimo s pretvorbo niza *beri* v tip *int*. Vse ostalo je ostalo nespremenjeno. To pove, da smo prvotni program res napisali tako, da je deloval za poljubno trimestno število.

4 POGOJNI STAVKI

Vsi stavki, ki smo jih spoznali do sedaj, so se izvajali zaporedoma. Najprej se je izvedel prvi stavek v metodi *main*, nato naslednji in tako naprej. Pogosto pa postopek zahteva, da kak ukaz izvedemo le, če so izpolnjeni določeni pogoji. To nam omogoča pogojni stavek.

4.1 If

Pogojni stavek *if* uporabimo, kadar želimo izvesti določene stavke (oz. stavek) samo v primeru, ko je izpolnjen nek pogoj.
Za besedo *if* v oklepajih zapišemo *pogoj*. To je poljuben logični izraz. Sledi mu blok stavkov (oz. stavek), ki jih želimo izvesti v primeru, če je pogoj resničen.

```
if(pogoj) {
  stavek1;
  stavek2;
  ...;
  stavekn;
}
```

Če je blok sestavljen samo iz enega stavka, lahko oklepaje izpustimo.

```
if (pogoj) stavek;
```

Kljub temu je bolje, da zaradi boljše preglednosti zavite oklepaje `{}` ohranimo.

Zgledi

Ali je prvo število manjše ali enako drugemu številu?

Napišimo preprost program, ki bo primerjal celi števili, ki sta shranjeni v dveh spremenljivkah. V primeru, da je prvo število manjše ali enako drugemu, bo izpisal "vrednost prvega števila *je manjse ali enako* vrednosti drugega števila". Če je prva vrednost *5*, druga pa *7*, bo program izpisal *"5 je manjse ali enako 7."*.

PRIMER 4.1 – 1: *StavekIf1.java*

```
 1:    public class StavekIf1 {
 2:      public static void main(String[] args) {
 3:        int prvoStevilo = 5;
 4:        int drugoStevilo = 7;
 5:
 6:        if (prvoStevilo <= drugoStevilo) {
 7:          System.out.println(prvoStevilo + " je manjse ali " +
 8:                             "enako " + drugoStevilo + ".");
 9:        } // if
10:        System.out.println("Konec programa.");
11:      } // main
12:    } // StavekIf1
```

Program prevedemo in poženemo:

```
5 je manjse ali enako 7.
Konec programa.
```

V 6. vrstici se preveri pogoj *prvoStevilo <= drugoStevilo*. Ker je pogoj izpolnjen, saj je 5 manjše ali enako 7, se izpišeta vrednosti v spremenljivkah *prvoStevilo* in *drugoStevilo* z ustreznim tekstom. Program se nato nadaljuje v vrstici 10. Izpiše se besedilo *Konec programa.*.

Kaj pa se zgodi, če prvemu številu priredimo vrednost *7*, drugemu pa *5*?

PRIMER 4.1 – 2: *StavekIf2.java*

```
 1:    public class StavekIf2 {
 2:      public static void main(String[] args) {
 3:        int prvoStevilo = 7;
 4:        int drugoStevilo = 5;
 5:
 6:        if(prvoStevilo <= drugoStevilo) {
 7:          System.out.println(prvoStevilo + " je manjse ali " +
 8:                             "enako " + drugoStevilo + ".");
 9:        } // if
10:        System.out.println("Konec programa.");
11:      } // main
12:    } // StavekIf2
```

Program prevedemo in poženemo:

```
Konec programa.
```

Opis programa.
V 6. vrstici se preveri pogoj *prvoStevilo <= drugoStevilo.* Ker pogoj ni resničen (ima vrednost false), se stavek *System.out.println()* znotraj pogojnega stavka sploh ne izvede in program se nadaljuje za stavkom *if* v vrstici 10. Izpiše se besedilo *Konec programa.*.

Ukaz v vrstici za stavkom *if* (10. vrstica v našem primeru) se izvede vedno, ukaz v vrstici 7 in 8 (stavek znotraj stavka *if*) pa le, če je pogoj izpolnjen.

Spremenimo program tako, da bomo prebrali dve števili in izpisali v kakšnem razmerju je prvo število z drugim.

PRIMER 4.1 – 3: *StavekIf3.java*

```
 1:    import javax.swing.*;
 2:
 3:    public class StavekIf3 {
 4:      public static void main(String[] args) {
 5:        String beri =
 6:            JOptionPane.showInputDialog("Prvo stevilo:");
 7:        int prvoStevilo = Integer.parseInt(beri);
 8:        beri = JOptionPane.showInputDialog("Drugo stevilo:");
 9:        int drugoStevilo = Integer.parseInt(beri);
10:
11:        if(prvoStevilo <= drugoStevilo) {
12:          System.out.println(prvoStevilo + " je manjse ali " +
13:                             "enako " + drugoStevilo + ".");
14:        } // if
15:        if(prvoStevilo > drugoStevilo) {
16:          System.out.println(prvoStevilo + " je vecje od " +
17:                             drugoStevilo + ".");
18:        } // if
19:      } // main
20:    } // StavekIf3
```

Opis programa.
V 5. in 6. vrstici prikažemo sporočilno okno s tekstom, ki je med narekovaji in vnosno vrstico. Ko uporabnik zapiše niz in klikne gumb OK, se zapisani niz prenese v niz *beri*. S klicem metode *Integer.parseInt(beri)* niz *beri* pretvorimo v celoštevilčno vrednost, ki se nato priredi spremenljivki *prvoStevilo*. V 8. vrstici ponovno prikažemo sporočilno okno s tekstom, ki je med narekovaji. Ko uporabnik zapiše niz in klikne gumb OK, niz *beri* postane ta vpisani niz. Niz *beri* pretvorimo v celoštevilčno vrednost, ki jo shranimo v spremenljivko *drugoStevilo*. V 11. vrstici preverimo pogoj in če je resničen, se izpiše ustrezen tekst (vrstici 12 in 13). Isto ponovimo v 15. vrstici. Preveri se pogoj in če je resničen, se izpiše ustrezen tekst (vrstici 16 in 17).
Ker je v našem primeru lahko resničen samo en pogoj, bi bilo bolj smiselno uporabiti pogojni stavek v obliki *if-else*. Tega si bomo ogledali v naslednjem razdelku.

Dvig denarja na bankomatu

Na transakcijskem računu imamo dovoljen limit 10.000 tolarjev. Ker nam je zmanjkalo denarja, želimo dvigniti določen znesek. Če ne presegamo limita, nam bankomat izplača izbrani znesek. Napišimo program, ki bo v primeru opravljene transakcije izpisal dvignjen znesek, stanje na računu in razpoložljivo stanje na računu. Če transakcija ni dovoljena, bo izpisal le stanje in razpoložljivo stanje na računu. Stanje na računu in znesek dviga preberemo.

PRIMER 4.1 – 4: *DvigDenarja.java*

```
 1:    import javax.swing.*;
 2:
 3:    public class DvigDenarja {
```

```
 4:      public static void main(String[] args) {
 5:        final int LIMIT = -10000; // dovoljen limit
 6:
 7:        String beri =
 8:            JOptionPane.showInputDialog("Stanje na racunu:");
 9:        int stanje = Integer.parseInt(beri);
10:        beri = JOptionPane.showInputDialog("Znesek dviga:");
11:        int dvig = Integer.parseInt(beri);
12:
13:        if(stanje - dvig >= LIMIT) {
14:          System.out.println("Dvig: " + dvig + " SIT");
15:          stanje = stanje - dvig;
16:        } // if
17:
18:        int razpolozljivoStanje = stanje + Math.abs(LIMIT);
19:
20:        System.out.println("Stanje na racunu: " + stanje +
21:                           " SIT");
22:        System.out.println("Razpolozljivo stanje: " +
23:                           razpolozljivoStanje + " SIT");
24:      } // main
25:    } // DvigDenarja
```

Opis programa.
V 5. vrstici deklariramo konstanto in ji priredimo vrednost. Slediti izpis dveh sporočilnih oken za vnos podatkov. Ko podatke preberemo in jih pretvorimo v cela števila, se preveri pogoj v vrstici 13. Če je pogoj resničen *(stanje - dvig >= LIMIT)*, se izvedeta stavka znotraj *if* (vrstici 14 in 15). V 18. vrstici sledi izračun razpoložljivega stanja. Izračuna se vrednost izraza, ki se priredi spremenljivki *razpolozljivoStanje*. Pomagali smo si z metodo *abs()* iz razreda *Math*, ki vrne absolutno vrednost števila. V našem primeru je to *10000*. Na koncu še izpišemo stanje na računu ter razpoložljivo stanje.

Uredi tri števila od najmanjšega do največjega

Napišimo program, ki bo prebral tri števila in jih izpisal v naraščajočem vrstnem redu. Pri tem bomo večkrat morali zamenjati vrednost dveh spremenljivk. To smo se že naučili ob programu *ZamenjavaSpremenljivk.java*, ki smo ga napisali v razdelku o celih številih. Vrednosti v spremenljivkah *a*, *b* in *c* bomo zamenjali tako, da bo v *a* najmanjša vrednost, v *c* pa največja vrednost. Nato izpišemo števila v vrstnem redu *a*, *b*, *c*.

PRIMER 4.1 – 5: *UrediTriStevila.java*

```
 1:    import javax.swing.*;
 2:
 3:    public class UrediTriStevila {
 4:      public static void main(String[] args) {
 5:
 6:        // preberemo podatke (nize) in jih
 7:        // pretvorimo v cela stevila
 8:        String beri =
 9:            JOptionPane.showInputDialog("Prvo stevilo:");
10:        int a = Integer.parseInt(beri);
11:        beri = JOptionPane.showInputDialog("Drugo stevilo:");
12:        int b = Integer.parseInt(beri);
13:        beri = JOptionPane.showInputDialog("Tretje stevilo:");
14:        int c = Integer.parseInt(beri);
15:
16:        System.out.println("Vnesel si: " + a + " " + b + " " +c);
17:        if(b < a) {
18:          // zamenjaj a in b
```

```
19:           int t = b;
20:           b = a;
21:           a = t;
22:         } // if
23:        // sedaj zagotovo velja a <= b
24:         if(c < a) {
25:           // zamenjaj a in c
26:           int t = c;
27:           c = a;
28:           a = t;
29:         } // if
30:
31:         // sedaj je a najmanjse izmed treh stevil
32:
33:         if(c < b) {
34:           // zamenjaj b in c
35:           int t = b;
36:           b = c;
37:           c = t;
38:         } // if
39:
40:         // v c je sedaj najvecje izmed treh stevil
41:
42:         System.out.println("Urejena stevila po velikosti " +
43:                            "od najmanjšega do najvecjega:");
44:         System.out.print(a + " " + b + " " + c);
45:       } // main
46:     } // UrediTriStevila
```

Opis programa.
Preberemo podatke in jih pretvorimo v cela števila. Sledi izpis vrednosti, ki so shranjene v spremenljivkah *a*, *b* in *c*, *opremljene* z ustreznim tekstom. V 17. vrstici poskrbimo, da je *a <= b*. Preverimo, če to ne drži. Če je vrednost spremenljivke b manjša od vrednosti spremenljivke a, se njuni vrednosti zamenjata (19.- 21. vrstice) .
Sedaj zagotovimo, da je a najmanjše število. Od b je že manjše (ali kvečjemu enako). Zato preverimo razmerje do spremenljivke *c*. Če je v spremenljivki *c* slučajno manjša vrednost, jo z zamenjavo »preselimo« v *a*. Tako je potem v *a* zagotovo najmanjša vrednost.
V *a* je že najmanjše število, ne vemo pa, v kakšnem razmerju sta vrednosti v spremenljivkah *b* in *c*. S pogojem v vrstici 33 in morebitno zamenjavo smo zagotovili, da je v spremenljivki *c* večje od obeh števil in s tem tudi največje število. Vrstni red vrednosti je torej *a*, *b* in *c*.
Na koncu še izpišemo urejene vrednosti z ustreznim tekstom.

Če si izberemo števila v padajočem vrstnem redu (npr. *7, 4 in 1*), bodo resnični vsi trije stavki *if*. Ker je *b* (*4*) manjše od *a* (*7*), se izvedejo stavki znotraj stavka *if* in zamenjata se vrednosti spremenljivk *a* in *b* (vrstice 17 do 22). Sedaj so vrednosti spremenljivk *a = 4*, *b = 7*, *c = 1*. Ker je *c* (*1*) manjše od *a* (*4*), se zamenjata ti dve vrednosti (vrstice 24 do 29). Sedaj so vrednosti spremenljivk *a = 1*, *b = 7*, *c = 4*. Ker je *c* (*4*) manjše od *b* (*7*), se zamenjata ti dve vrednosti (vrstice 33 do 38). Sedaj so vrednosti spremenljivk *a = 1*, *b = 4*, *c = 7*. Te nato opremljene z ustreznim tekstom izpišemo (vrstice 42 do 44).

Če si izberemo števila, ki so že urejena v naraščajočem vrstnem redu (npr.*1, 2 in 3*), bodo neresnični vsi trije pogoji pri stavkih *if*. Zato se ne bo izvedel noben stavek znotraj stavkov *if*. V primeru omenjenem prej (podatki so npr. 1, 2, 3) se bo po vrstici 16 preveril pogoj v vrstici 17 in ker je njegova vrednost false (*3* ni manjše od *1*), se stavki znotraj stavka *if* ne izvedejo. Prav tako se zgodi v vrstici 24 in 33. V tem primeru izpišemo vrednosti v takšnem vrstnem redu, kot smo jih vnesli (vrstice 42 do 44).

4.2 If – else

Prejšnja oblika pogojnega stavka je poskrbela, da se je določen ukaz izvedel le, če je bil pogoj izpolnjen. Nato smo v vsakem primeru nadaljevali z istim ukazom. Včasih pa želimo, da se takrat, ko pogoj ni izpolnjen, zgodi nekaj drugega. Takrat uporabimo stavek `if` v kombinaciji z `else`.

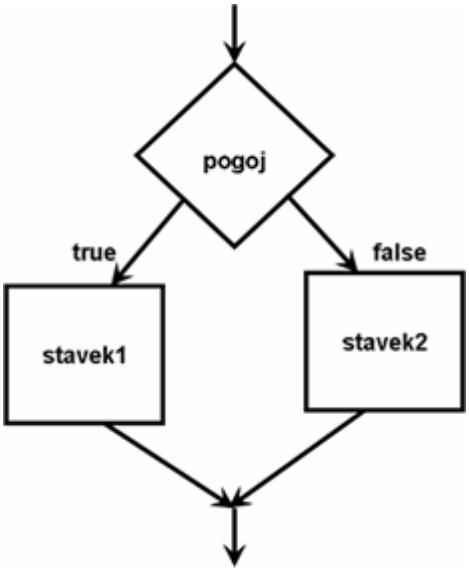
```
if(pogoj) {
  stavekp1;
  stavekp2;
  ...;
  stavekpn;
}
else {
  stavekr1;
  stavekr2;
  ...;
  stavekrk;
}
```

Če bo pogoj izpolnjen, se bodo izvedli $stavekp_1$ do $stavekp_n$ (oz. `stavek1` v spodnjem zgledu), sicer $stavekr_1$ do $stavekr_k$ (oz. `stavek2` v spodnjem zgledu).Če sta bloka sestavljena samo iz enega stavka, lahko oklepaje izpustimo

```
if(pogoj) stavek1;
else stavek2;
```

vendar jih zaradi boljše preglednosti praviloma vseeno zapišemo.

Shematično prikažemo stavek `if-else` takole:



preveri se pogoj,

če je resničen, izvedi **stavek1**,
če ni resničen, izvedi **stavek2**

Zgledi

Spomnimo se programa `StavekIf3.java`, kjer smo prebrali dve števili in izpisali, v kakšnem razmerju je prvo število z drugim. Popravimo program tako, da bomo namesto dveh zaporednih pogojnih stavkov uporabili stavek `if-else` .

PRIMER 4.2 – 1: `StavekIfElse.java`

```
 1:    import javax.swing.*;
 2:
 3:    public class StavekIfElse {
 4:      public static void main(String[] args) {
 5:        String beri =
 6:            JOptionPane.showInputDialog("Prvo stevilo:");
 7:        int prvoStevilo = Integer.parseInt(beri);
 8:        beri = JOptionPane.showInputDialog("Drugo stevilo:");
 9:        int drugoStevilo = Integer.parseInt(beri);
10:
11:        if(prvoStevilo <= drugoStevilo) {
12:          System.out.println(prvoStevilo + " je manjse ali " +
13:                             "enako " + drugoStevilo + ".");
14:        } else {
15:          System.out.println(prvoStevilo + " je vecje od " +
16:                             drugoStevilo + ".");
17:        } // else
18:        System.out.println("Konec programa.");
19:      } // main
20:    } // StavekIfElse
```

Popravili smo vrstico 14. Če je pogoj v 11. vrstici resničen (kadar je prvo prebrano število manjše ali enako drugemu prebranemu številu), se izvedeta stavka v vrsticah 12 in 13. Nato nadaljujemo v vrstici 19. V primeru, če pogoj v 11. vrstici ni resničen, je prvo prebrano število večje od drugega prebranega števila. Takrat se izvede stavek v vrstici 16 in stavek v vrstici 17. Nato nadaljujemo v vrstici 19.

Dostopnost spremenljivk

Poglejmo si naslednji program.

PRIMER 4.2 – 2: *DostopnostSpremenljivk.java*

```
 1:    public class DostopnostSpremenljivk {
 2:      public static void main(String[] args) {
 3:        if(27 * 38 > 650) {
 4:          int t = 10;
 5:          System.out.println(t);
 6:        } // if
 7:        else {
 8:          t = 20;
 9:          System.out.println(t);
10:        } // else
11:      } // main
12:    } // DostopnostSpremenljivk
```

Prevedemo in poženemo:

```
C:\DostopnostSpremenljivk.java:8: cannot resolve symbol
symbol  : variable t
location: class DostopnostSpremenljivk
      t = 20;
      ^
C:\DostopnostSpremenljivk.java:9: cannot resolve symbol
symbol  : variable t
location: class DostopnostSpremenljivk
      System.out.println(t);
                         ^
2 errors
```

Zakaj nam prevajalnik pravi, da ne pozna spremenljivke `t`, čeprav smo jo v 4. vrstici?

Razlog je ta, da so deklaracije spremenljivk lokalne. To pomeni, da deklaracija spremenljivke velja le znotraj bloka. Naša spremenljivka `t` je veljavna od 4. pa do vključno 6. vrstice. Torej je spremenljivka `t` v 8. vrstici neka nova spremenljivka, ki ima slučajno enako ime in jo je potrebno ponovno deklarirati. Ta spremenljivka nima nobene povezave s tisto, ki je "v igri" od 4. do 6. vrstice.

Popravimo program.

PRIMER 4.2 – 3: *DostopnostSpremenljivk.java*

```
 1:    public class DostopnostSpremenljivk {
 2:      public static void main(String[] args) {
 3:        if(27 * 38 > 650) {
 4:          int t = 10;
 5:          System.out.println(t);
 6:        } // if
 7:        else {
 8:          int t = 20;
 9:          System.out.println(t);
10:        } // else
11:      } // main
12:    } // DostopnostSpremenljivk
```

Prevedemo in poženemo:

```
10
```

Seveda bi lahko spremenljivko deklarirali tudi izven notranjega bloka – na nivoju celotne metode main,

PRIMER 4.2 – 3a: *DostopnostSpremenljivk1.java*

```
 1:    public class DostopnostSpremenljivk {
 2:      public static void main(String[] args) {
 3:        int t;
 4:        if(27 * 38 > 650) {
 5:          t = 10;
 6:          System.out.println(t);
 7:        } // if
 8:        else {
 9:          t = 20;
10:          System.out.println(t);
11:        } // else
12:      } // main
13:    } // DostopnostSpremenljivk1
```

Ko prevajalnik naleti na ime spremenljivke, preveri, če je deklarirana v trenutnem bloku (znotraj { }) . Če najde tam njeno deklaracijo, jo upošteva. Če deklaracije v tem bloku ni, jo prevajalnik išče v bloku, ki trenutni blok vsebuje (prvem zunanjem bloku) in tako naprej,. ... Zato je npr. v vrstici 9 prevajalnik najprej preveril, če je spremenljivka `t` deklarirana v bloku, ki sega od 8. do 11. vrstice. Ker je tam ni, je njeno deklaracijo iskal v zunanjem bloku, torej izven pogojnega stavka do vrstice 12.

Ali sta niza enaka?

Napišimo program, ki bo ugotovil, ali sta vnesena niza enaka. Če sta, naj izpiše *"Niza sta enaka."*, drugače pa *"Niza sta različna."*.
Spomnimo se na **PRIMER 3.4.2.5 – 2,** kjer smo uporabili dejstvo, da enakost dveh nizov preverjamo z metodo `equals`.

PRIMER 4.2 – 4: *EnakaNiza.java*

```
 1:    import javax.swing.*;
 2:
 3:    public class EnakaNiza {
 4:      public static void main(String[] args) {
 5:        String prviNiz =
 6:            JOptionPane.showInputDialog("Vpisi prvi niz:");
 7:        String drugiNiz =
 8:            JOptionPane.showInputDialog("Vpisi drugi niz:");
 9:
10:        if(prviNiz.equals(drugiNiz)) {
11:          System.out.println("Niza sta enaka.");
12:        } // if
13:        else {
14:          System.out.println("Niza sta razlicna.");
15:        } // else
16:      } // main
17:    } // EnakaNiza
```

Praznovanje rojstnega dne

Janja ima rojstni dan 29. februarja. Če leto ni prestopno, ga praznuje 1. marca. Napišimo program, ki bo Janji pomagal ugotoviti, ali bo na določeno leto praznovala rojstni dan 29. februarja ali 1. marca.

Leto je prestopno, kadar je deljivo s 4 in ne s 100 ali kadar je deljivo s 400.

PRIMER 4.2 – 5: *Praznovanje.java*

```
 1:    import javax.swing.*;
 2:
 3:    public class Praznovanje {
 4:      public static void main(String[] args) {
 5:        String beri =
 6:            JOptionPane.showInputDialog("Vpisi leto: ");
 7:        int leto = Integer.parseInt(beri);
 8:
 9:        if (((leto % 4 == 0) && (leto % 100 != 0))
10:            || (leto % 400 == 0)) {
11:          System.out.println("Praznovanje bo 29. februarja.");
12:        } // if
13:        else {
14:          System.out.println("Praznovanje bo 1. marca.");
15:        } //else
16:      } // main
17:    } // Praznovanje
```

Prebrani niz pretvorimo v celo število(vrstice 5 do 7). Nato preverimo pogoj v vrstici 9 in 10. Pogojni stavek je sestavljen iz dveh delov. Prvi del *((leto % 4 == 0) && (leto % 100 != 0))* je sestavljen iz dveh pod pogojev. Če sta pod pogoja resnična, se drugi del sploh ne preverja in izvede se vrstica 11, ki izpiše *"Praznovanje bo 29. februarja."*. V primeru, da prvi del ni resničen, se preveri drugi del. Če je resničen drugi del, se prav tako izpiše *"Praznovanje bo 29. februarja."*. Če pa sta oba neresnična, se izvede else del (vrstica 13). Izpiše se *"Praznovanje bo 1. marca."* (vrstica 14).

Telesna teža

Jelka je v lekarni dobila brošuro, v kateri je formula za izračun indeksa telesne mase. Indeks telesne mase je razmerje med telesno maso (v kg) in kvadratom višine (v m). S tem indeksom ugotovimo, če imamo čezmerno telesno maso. Napišimo program, ki bo prebral podatke o teži in višini in glede na izračunani indeks telesne mase (ITM) izpisal ustrezen tekst.

Če je indeks telesne mase manj kot 18.5, naj program izpiše *"Premajhna telesna masa."*. V primeru, da je indeks telesne mase med 18.5 in 24.9, naj se izpiše *"Normalna telesna masa."*. Če pa je indeks telesne mase več kot 25, naj program izpiše *"Cezmerna telesna masa."*. Na koncu v oklepaju še izpišimo izračunani ITM.

PRIMER 4.2 – 6: *TelesnaTeza.java*

```
 1:    import javax.swing.*;
 2:
 3:    public class TelesnaTeza {
 4:      public static void main(String[] args) {
 5:        double itm;
 6:        String beri =
 7:            JOptionPane.showInputDialog("Vnesi maso (v kg):");
 8:        int masa = Integer.parseInt(beri);
 9:        beri =JOptionPane.showInputDialog("Vnesi visino (v m):");
10:        double visina = Double.parseDouble(beri);
11:
12:        // izracunamo itm
13:        itm = masa / Math.pow(visina, 2);
14:
15:        if(itm < 18.5) {
16:          System.out.println("Premajhna telesna teza." +
17:                             " (itm = " + itm +")");
18:        } // if
19:        else if(itm <= 24.9) { // vemo da velja itm >= 18.5
20:          System.out.println("Normalna telesna teza." +
21:                             " (itm = " + itm +")");
22:        } // else if
23:        else {
24:          System.out.println("Cezmerna telesna teza." +
25:                             " (itm = " + itm +")");
26:        } // else
27:      } // main
28:    } // TelesnaTeza
```

Opis programa.

Prvič smo uporabili kombinacijo več pogojnih stavkov *if..else*. Znotraj pogojnega stavka imamo spet pogojni stavek. Kombinacijo smo zapisali v obliki

```
if (pogoj1){
   stavek1;
}
else if(pogoj2){
   stavek2;
}
…
else {
   stavekn;
}
```

Če velja *pogoj1*, se izvede *stavek1*, sicer če velja *pogoj2*, se izvede *stavek2*, sicer …, sicer se izvede *stavekn*. V navedenem primeru se vedno izvede natanko en od stavkov *stavek1, stavek2, ..., stavekn*. Če bi uporabljali pisanje pogojnega tsavka na "klasičen način", bi napisali

```
if (pogoj1){
   stavek1;
```

```
}
else {
  if(pogoj2) {
   stavek2;
  }
  else {
    if(pogoj3){
      stavek3;
    }
    …
    else {
      stavekn;
    }
   …
  }
}
```

Ampak pri takem pisanju nam hitro zmanjka prostora, pa še bolj nepregledno je.

V našem primeru imamo tak kostrukt v vrsticah 15 – 26. Najprej se preveri prvi pogoj. Če je ITM manjši od *18.5* se izpiše *Premajhna telesna teža* in program se zaključi. Če to ni res (ITM je večji ali enak 18.5, preverimo drugi pogoj. Če je naš ITM manjši ali enak *24.9* (večji ali enak 18.5 je že!), se izpiše *Normalna telesna teža*, če pa ta pogoj ni izpolnjen, se izpiše *Cezmerna telesna teža*.

Kockanje

Peter in Gorazd kockata za denar. Na začetku vsak stavi nekaj denarja, nato vržeta kocki. Tisti, ki vrže več pik, zmaga in dobi stavo. V primeru, da oba vržeta enako število pik, je igra izenačena in nihče ne dobi denarja. Napišimo program, ki bo simuliral to igro.

Metanje kocke bomo simulirali s pomočjo metode *random()* iz razreda *Math*. Ta metoda vrne naključno realno število večje ali enako 0.0 in manjše od 1.0.

PRIMER 4.2 – 7: *Kockanje.java*

```
 1:    import javax.swing.*;
 2:
 3:    public class Kockanje {
 4:      public static void main(String[] args) {
 5:        String beri =
 6:            JOptionPane.showInputDialog("Vnesi Petrovo stavo:");
 7:        int stavaPeter = Integer.parseInt(beri);
 8:        beri =
 9:           JOptionPane.showInputDialog("Vnesi Gorazdovo stavo:");
10:        int stavaGorazd = Integer.parseInt(beri);
11:
12:        int pikePeter = (int)(6 * Math.random()) + 1;
13:        int pikeGorazd = (int)(6 * Math.random()) + 1;
14:
15:        System.out.println("Peter je stavil " + stavaPeter +
16:                           " in vrgel " + pikePeter + ".");
17:        System.out.println("Gorazd je stavil " + stavaGorazd +
18:                           " in vrgel " + pikeGorazd + ".");
19:
20:        if(pikePeter == pikeGorazd) {
21:          System.out.println("Igra je izenacena.");
```

```
22:         } // if
23:         else if(pikePeter > pikeGorazd) {
24:           System.out.println("\nPeter dobi " + stavaGorazd +
25:                              " od Gorazda.");
26:         } // else if
27:         else {
28:           System.out.println("\nGorazd dobi " + stavaPeter +
29:                              " od Petra.");
30:         } // else
31:       } // main
32:     }// Kockanje
```

Ko preberemo podatka o višini stave, ju pretvorimo v celi števili. Nato se izvedeta vrstici 12 in 13. V teh dveh vrsticah generiramo naključni celi števili od 1 do vključno 6, ki se shranita v spremenljivki *pikePeter* in *pikeGorazd*. Ker nam metoda *random()* vrača nenegativno realno število, ki je manjše od 1.0, to vrednost pomnožimo s *6*. S tem dobimo realne vrednosti, ki so med 0.0 in 5.999999.... Ker lahko na kocki vržemo le celo število od 1 do vključno 6, z operatorjem spremembe tipa *(int)* pretvorimo realno vrednost v celoštevilčno vrednost. Vendar s tem še nismo dobili željnih vrednosti, saj odrežemo decimalni del in lahko dobimo le vrednosti od 0 do vključno 5. Če vse skupaj povečamo še za ena, bomo lahko dobili vrednosti od 1 do vključno 6.
Enake vrednosti bi dobili, če bi namesto *(int)(6 * Math.random()) + 1* napisali *(int)(6 * Math.random() + 1)*. V tem primeru bi se najprej izvedlo množenje. Tako bi lahko dobili vrednosti od 0.0 do manj kot 6.0. Ko temu prištejemo 1, dobimo vrednosti od 1.0 do manj kot 7.0. Če odrežemo decimalni del, dobimo vrednosti od 1 do vključno 6.
Ostane nam še, da izpišemo rezultat igre. Če imata enako pik (pogoj *pikePeter = pikeGorazd* je resničen), izpišemo *Igra je izenacena,* sicer preverimo, če je imel Peter več pik od Gorazda in izpišemo *Peter dobi+ stavaGorazd + od Gorazda.* Če ni resničen noben od prejšnih pogojev, izpišemo *Gorazd dobi + stavaPeter + od Petra.*

Nakup stanovanja

Janez se je odločil, da gre živeti na svoje in si začel iskati stanovanje. V časopisu je zasledil nekaj primernih stanovanj. Na banki ima že nekaj privarčevanega denarja. Na banki je zaprosil za informativni izračun kredita. Pomagajmo Janezu pri nakupu. Poznamo ceno stanovanja, znesek privarčevanega denarja ter znesek odobrenega kredita. Napišimo torej program, ki bo glede na vnesene zneske izpisal ustrezen tekst:

Če bo potrebno vzeti kredit, izpišemo, koliko kredita moramo vzeti.
Če ima dovolj privarčevanega denarja (kredita ne bomo vzeli), izpišemo koliko denarja nam ostane po nakupu stanovanja.
Če kljub kreditu nima dovolj denarja, izpišemo, koliko kredita lahko vzame ter koliko mu zmanjka za nakup stanovanja.
V vseh primerih izpišemo, koliko stane stanovanje, koliko denarja ji privarčevali ter ali lahko kupi stanovanje.

PRIMER 4.2 – 8: *NakupStanovanja.java*

```
1:    import javax.swing.*;
2:
3:    public class NakupStanovanja {
4:      public static void main(String[] args) {
5:        int privarcevano, cena, kredit, razlika;
6:        String stanje = "Stanovanje lahko kupimo.";
7:
8:        // preberemo zneske
9:        String beri =
10:           JOptionPane.showInputDialog("Vnesi ceno stanovanja:");
11:       cena = Integer.parseInt(beri);
```

```
12:         beri = JOptionPane.showInputDialog("Vnesi " +
13:                                            "privarcevani znesek:");
14:         privarcevano = Integer.parseInt(beri);
15:         beri = JOptionPane.showInputDialog("Koliko " +
16:                                            "kredita lahko dobis:");
17:         kredit = Integer.parseInt(beri);
18:
19:         // izracunamo razliko
20:         razlika = cena - privarcevano;
21:
22:         System.out.println("Stanovanje stane " + cena + " SIT," +
23:                            " privarcevano imamo "+  privarcevano +
24:                            " SIT.");
25:
26:         // glede na vrednosti izpisemo ustrezen tekst
27:         if(razlika <= 0) {
28:           System.out.println("Za nakup stanovanja imamo dovolj" +
29:                              " privarcevanega denarja.");
30:           if(razlika < 0) {
31:             System.out.println("Ostane nam se " +
32:                                Math.abs(razlika) + " SIT.");
33:           } // if
34:           else {
35:           System.out.println("Porabimo ves privarcevani denar.");
36:           } // else
37:         } // if
38:         else {
39:           System.out.println("Za nakup stanovanja nimamo dovolj"+
40:                              " privarcevanega denarja.");
41:           if(razlika <= kredit) {
42:              System.out.println("Kredita moramo vzeti " + razlika
43:                                 + " SIT.");
44:            } // if
45:           else {
46:             System.out.print("Ker lahko kredita vzamemo le " +
47:                              kredit + " SIT, nam za nakup " +
48:                              "stanovanja zmanjka " +
49:                              (razlika - kredit) + " SIT.");
50:             stanje = "Stanovanja si ne moremo privosciti.";
51:           } // else
52:         } // else
53:         System.out.println(stanje);
54:       } // main
55:     } // NakupStanovanja
```

V zgledu vidimo, kako lahko tudi znotraj pogojnega stavka spet uporabimo pogojni stavek. Pri pisanju ustreznih pogojev moramo paziti pri kakšnih pogojih se znajdemo znotraj pogojnega stavka – bodisi v pritrdilnem bodisi v else delu.

Kvadratna enačba $ax^2 + bx + c = 0$

Napišimo program, ki bo poiskal rešitve kvadratne enačbe $ax^2 + bx + c = 0$. Pri tem upoštevajmo vse možnosti, ki lahko nastopijo. Po branju koeficientov a, b in c najprej preverimo, ali je koeficient pri kvadratnem členu enak 0. Če je, imamo opraviti z linearno enačbo. Takrat preverimo še b. Če je tudi ta 0, sta zopet dve možnosti. Prva, ko je še c enak 0, pomeni, da imamo opraviti z identiteto, druga pa, da je enačba protislovna. Kadar je b od 0 različen, imamo eno rešitev, in sicer -c / b.
Če je a od 0 različen, imamo opraviti s pravo kvadratno enačbo. Izračunamo njeno diskriminanto $b^2 - 4ac$. Če je pozitivna, imamo realne rešitve, do katerih si pomagamo z znanim obrazcem in Vietovo formulo. Če je negativna, imamo kompleksni rešitvi. Pri teh izračunamo realni del in nato še imaginarnega.

PRIMER 4.2 – 9: *KvadratnaEnacba.java*

```
 1:    import javax.swing.*;
 2:
 3:    public class KvadratnaEnacba {
 4:      public static void main(String[] args) {
 5:        double d, x1, x2;
 6:
 7:        // preberemo vrednosti
 8:        String beri =
 9:            JOptionPane.showInputDialog("Vnesi a:");
10:        double a = Double.parseDouble(beri);
11:        beri = JOptionPane.showInputDialog("Vnesi b:");
12:        double b = Double.parseDouble(beri);
13:        beri = JOptionPane.showInputDialog("Vnesi c:");
14:        double c = Double.parseDouble(beri);
15:
16:        // glede na vrednosti izpisemo ustrezen tekst
17:        if(a == 0) {
18:          if(b == 0) {
19:            if(c == 0) {
20:              System.out.println("Enacbo resi vsako " +
21:                                 "(kompleksno) stevilo.");
22:            }  // if
23:            else {
24:              System.out.println("Enacba nima resitve.");
25:            } // else
26:          } // if
27:          else {
28:            System.out.println("Edina resitev enacbe je " +
29:                               (-c / b) + ".");
30:          } // else
31:        } // if
32:        else {
33:          d = b * b - 4 * a * c;
34:          if(d == 0) {
35:            System.out.println("Enacba ima dvojno niclo " +
36:                               (-b / (2 * a)) + ".");
37:          } // if
38:          else if(d > 0) {
39:            if(b > 0) {
40:              x1 = (-b - Math.sqrt(d)) / (2 * a);
41:            } // if
42:            else {
43:              x1 = (-b + Math.sqrt(d)) / (2 * a);
44:            } // else
45:            System.out.println("Enacba ima dve realni resitvi.");
46:            System.out.println("x1 = " + x1 + ".");
47:            System.out.println("x2 = " + (c / a / x1) + ".");
48:          } // else if
49:          else {
50:            x1 = -b / (2 * a);
51:            x2 = (Math.sqrt(-d)) / (2 * a);
52:            System.out.println("Enacba ima kompleksni resitvi.");
53:            System.out.println("x1 = " + x1 + " + " +
54:                               x2 + " * i.");
55:            System.out.println("x2 = " + x1 + " - " +
56:                               x2 + " * i.");
57:          } // else
58:        } // else
```

```
59:         } // main
60:     } // KvadratnaEnacba
```

5 ZANKE

Zanke omogočajo ponavljanje istih stavkov, dokler je določen pogoj izpolnjen. Ogledali si bomo zanki `while` in `for`.

5.1 While

Zanko `while` uporabimo, kadar želimo, da se izvajanje določenih stavkov (oz. stavek) ponavlja, dokler je določen pogoj izpolnjen.
Tako kot pri pogojnem stavku za besedo `if`, tukaj za besedo `while` v oklepajih zapišemo pogoj. Sledi mu blok stavkov (oz. stavek), ki jih želimo izvajati večkrat – toliko časa, dokler je pogoj izpolnjen.
Zanka `while` se izvaja, dokler je pogoj resničen (ima vrednost true). Njegovo resničnost se preveri tudi takoj na začetku, zato ni nujno, da se stavki (oz. stavek), ki mu sledijo, sploh izvedejo.

```
while(pogoj) {
  stavek1;
  stavek2;
  ...
  Stavekn;
}
```

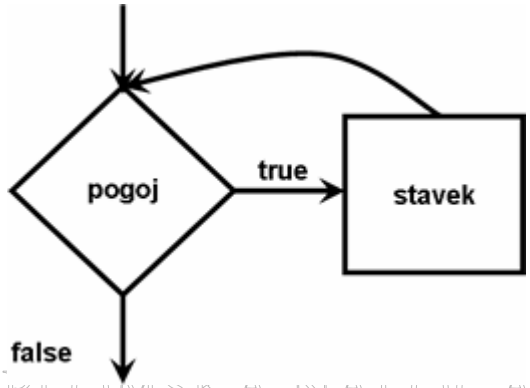
Če je blok sestavljen samo iz enega stavka, lahko oklepaje izpustimo.

```
while(pogoj) stavek;
```

Tako kot pri pogojnem stavku, tudi tu zaradi boljše preglednosti oklepaje ohranimo.

Na dolgo bi delovanje `while` zanke opisali takole:

- najprej se preveri pogoj,
- če ima pogoj vrednost `true` (je resničen), se izvede stavek,
- nato se ponovno preveri pogoj,
- če je še vedno izpolnjen, se znova izvede stavek,
- ponovno se preveri pogoj,
- ...
- če pogoj ob preverjanju nima več vrednosti `true` (ni več resničen), se konča izvajanje zanke `while`.



Ponavljaj:

če je **pogoj true,**
    izvedi **stavek**,
sicer prekini izvajanje,

Zgledi

Izpis števil

Izpišimo števila od *1* do *10*. Vsako naj bo v svoji vrstici.

PRIMER 5.1 – 1: *IzpisStevil.java*

```
 1:    public class IzpisStevil {
 2:      public static void main(String[] args) {
 3:        int stevilo = 1;
 4:
 5:        while(stevilo <= 10) {
 6:          System.out.println(stevilo);
 7:          stevilo = stevilo + 1;
 8:        } // while
 9:        System.out.println("Konec programa.");
10:      } // main
11:    } // IzpisStevil
```

Program prevedemo in poženemo:

```
1
2
3
4
5
6
7
8
9
10
Konec programa.
```

Na začetku ob deklaraciji spremenljivki *stevilo* priredimo začetno vrednost *1* (vrstica 3). Nato se preveri pogoj (vrstica 5). Ker je vrednost v spremenljivki manjša ali enaka od *10* (*1 <= 10*), se izpiše vrednost spremenljivke *stevilo* (vrstica 6), torej *1*. Nato se izračuna desni del, ki se priredi spremenljivki na levi strani (*stevilo*). Vrednost v spremenljivki je sedaj *2*. Nato se zopet preveri pogoj v vrstici 5 in ker je resničen (*2 <= 10*), se izvede blok zanke. Izpiše se trenutna vrednost v spremenljivki *stevilo*, tokrat *2*. Povečamo vrednost spremenljivke *stevilo* na *3*. Ponovno preverimo pogoj v vrstici 5. Ker je izpolnjen, ...
Zanka se torej ponavlja, dokler je vrednost v spremenljivki *stevilo* manjša ali enaka *10*. Ko je vrednost v spremenljivki enaka *10*, pogoj še vedno velja. Zato se izvede ukaz v vrstici 6 in na zaslon se izpiše *10*. Nato se vrednost spremenljivke *stevilo* poveča za *1*. Sedaj je vrednost v spremenljivki *stevilo* enaka *11*. Zopet se preveri pogoj v peti vrstici in ker ni več resničen (*11 <= 10*), se zanka zaključi. Program se nadaljuje v vrstici 9. Izpiše se besedilo *Konec programa.*.

Stavek v sedmi vrstici *stevilo = stevilo + 1;* lahko krajše zapišemo kot *stevilo++;*.

Program bi lahko napisali tudi malce drugače

PRIMER 5.1 – 2: *IzpisStevil1.java*

```
 1:    public class IzpisStevil1 {
 2:      public static void main(String[] args) {
 3:        int stevilo = 0;
```

```
4:
5:        while(stevilo < 10) {
6:          stevilo ++;
7:          System.out.println(stevilo);
8:        } // while
9:        System.out.println("Konec programa.");
10:     } // main
11:   } // IzpisStevil1
```

Če program izvedemo, vidimo, da nam da enak izpis kot prejšnji program.

Izpis lihih števil

Izpišimo liha števila od *1* do *18*. Tokrat izpišimo vsa v isti vrstici in števila ločimo z presledkom.

PRIMER 5.1 – 3: *LihaStevila.java*

```
1:    public class LihaStevila {
2:      public static void main(String[] args) {
3:        int lihoStevilo = 1;
4:
5:        while(lihoStevilo <= 18) {
6:          System.out.print(lihoStevilo + " ");
7:          lihoStevilo = lihoStevilo + 2;
8:        } // while
9:      } // main
10:   } // LihaStevila
```

Program prevedemo in poženemo:

```
1 3 5 7 9 11 13 15 17
```

Opis programa.
Program deluje podobno kot prejšnji, le da tu vrednost, ki je shranjena v spremenljivki *lihoStevilo*, povečujemo za 2.
Stavek v sedmi vrstici *lihoStevilo = lihoStevilo + 2;* lahko krajše zapišemo *lihoStevilo += 2;*. Podobno okrajšano pisanje lahko uporabimo tudi za ostale operacije (*-, *, /, %*).

Pri zanki *while* moramo paziti, da se zanka konča. V primeru nepravilne uporabe lahko zanka teče v neskončnost (se zacikla). Pravilno zapisana zanka bo torej taka, kjer pogoj nekoč le dobi vrednost *false*.

PRIMER 5.1 – 4: *Ciklanje.java*

```
1:    public class Ciklanje {
2:      public static void main(String[] args) {
3:        int stevilo = 1;
4:
5:        while(stevilo > 0) {
6:          System.out.println(stevilo);
7:          stevilo++;
8:        } // while
9:      } // main
10:   } // Ciklanje
```

Program prevedemo in poženemo:

```
1
2
```

```
3
4
5
6
.
.
.
```

Program izpisuje števila v neskončnost in se sploh ne ustavi. Pogoj je vedno resničen, saj je vrednost spremenljivke *stevilo* vedno večja od *0*. No, če bi bili potrpežljivi, bi videli, da se bo program vseeno ustavil. Namreč ko bi prišlo do prekoračitve obsega celih števil, bi v spremenljivki *stevilo* dobili negativno število, ki bi ustavila zanko.

No, pri naslednjem zgledu pa se zanka zagotovo nikoli ne ustavi.

PRIMER 5.1 – 5: *Ciklanje1.java*

```
1:    public class Ciklanje1 {
2:      public static void main(String[] args) {
3:        int stevilo = 1;
4:
5:        while(stevilo < 10)
6:          System.out.println(stevilo);
7:          stevilo++;
8:      } // main
9:   } // Ciklanje1
```

Program prevedemo in poženemo:

```
1
1
1
1
1
1
.
.
.
```

Tudi v tem primeru je pogoj vedno resničen, saj je vrednost spremenljivke *stevilo* vedno *1* in je vedno večja od *0*. Ker ni zavitih oklepajev, spada v *while* zanko samo stavek *System.out.println(stevilo);* (vrstica 6) in ker je pogoj vedno resničen, se vrstica 7 sploh ne izvede. Vrednost spremenljivke *stevilo* se torej nikoli ne spremeni in je pogoj vedno izpolnjen, iz česar sledi, da se program zacikla.

Izpis vsote števil

Preberimo število in izračunajmo vsoto celih števil od 1 do prebranega števila. Sicer bi lahko uporabili kar znamenito Gaussovo formulo, s katero bi takoj izračunali to vsoto. A ker se je morda ne spomnimo, bomo s pomočjo zanke res sešteli vsa števila.

PRIMER 5.1 – 6: *VsotaStevil.java*

```
1:    import javax.swing.*;
2:
3:    public class VsotaStevil {
4:      public static void main(String[] args) {
5:        int stevilo1 = 1;
```

```
 6:         int vsota = 0;
 7:         String beri =
 8:             beri = JOptionPane.showInputDialog("Vnesi stevilo:");
 9:         int doKam = Integer.parseInt(beri);
10:
11:         while(stevilo1 <= doKam) {
12:           vsota = vsota + stevilo1;
13:           stevilo1++;
14:         } // while
15:         System.out.println(vsota);
16:       } // main
17:     } // VsotaStevil
```

Opis programa.
Zanimivi del programa se začne v 11. vrstici. Tam se začne zanka, ki ponavlja stavka v vrsticah 12 in 13 toliko časa, dokler je spremenljivka *stevilo1* manjša ali enaka vnesenemu številu. V 12. vrstici seštevamo števila in njihovo vsoto hranimo v spremenljivki *vsota*.
Denimo, da smo vnesli število *2*. Preveri se pogoj (vrstica 11) in ker je *1* manjše ali enako *2*, se izračuna *0 + 1* (*vsota + stevilo1*) in ta vsota se shrani v spremenljivko *vsota*. Sedaj je v spremenljivki *vsota* vrednost *1*. Vrednost v spremenljivki *stevilo1* se poveča za *1* (vrstica 13) in spremeni se vrednost v spremenljivki *stevilo1* na *2*. Zopet se preveri pogoj (vrstica 11) in ker je *2* manjše ali enako *2*, se izračuna *1 + 2* (*vsota + stevilo1*) in ta vsota se shrani v spremenljivko *vsota*. V spremenljivki *vsota* je sedaj vrednost *3*. Vrednost v spremenljivki *stevilo1* se poveča za *1* (vrstica 13) na *3*. Zopet se preveri pogoj (vrstica 11) in ker *3* ni manjše ali enako *2*, se zanka konča.

Rešitev enačbe

Poiščimo celoštevilske rešitve enačbe *x * x - 4 = 0* na intervalu *[-5, 5]*. V primeru, da na izbranem intervalu ni take rešitve, to izpišimo. Rešitev bomo poiskali enostavno tako, da bomo za vsa cela števila na izbranem intervalu preverili, če ustrezajo enačbi.

PRIMER 5.1 – 7: *ResitevEnacbe.java*

```
 1:   public class ResitevEnacbe {
 2:     public static void main(String[] args) {
 3:       int x = -5;
 4:       boolean jeResitev = false;
 5:
 6:       while(x <= 5) {
 7:         if(x * x - 4 == 0) {
 8:           System.out.println("Resitev: x = " + x);
 9:           jeResitev = true;
10:         } // if
11:         x = x + 1;
12:       } // while
13:
14:       if(!jeResitev) {
15:         System.out.println("Ni resitve!");
16:       } // if
17:     } // main
18:   } // ResitevEnacbe
```

Program prevedemo in poženemo:

```
Resitev: x = -2
Resitev: x = 2
```

Opis programa.

V 3. vrstici deklariramo spremenljivko *x* in ji določimo začetno vrednost *-5*. Ob deklaraciji spremenljivko *jeResitev* nastavimo na false. S pomočjo te spremenljivke bomo ugotovili, ali enačba ima rešitve.
Zanka *while* se bo končala, ko bo *x* večji od *5*, torej *x* teče od *-5* do *5*. Če *x* ustreza enačbi, se izpiše vrednost v spremenljivki *x* (vrstica 8). Pri tem si v logično spremenljivko zapomnimo, da smo našli vsaj eno rešitev. V vsakem primeru se *x* poveča za *1* (vrstica 11).
V vrstici 14 preverimo, če smo našli rešitev. Če je spremenljivka *jeResitev* ostala false, enačba nima celoštevilske rešitve in to dejstvo izpišemo.

Vsota vnesenih števil

Berimo števila preko sporočilnega okna in izračunajmo njihovo vsoto. Števila beremo, dokler ne vpišemo k za konec.

Pomagali si bomo z ukazom *break*, ki poskrbi, da lahko zanko predčasno zaključimo. Ko se izvede stavek *break;*, zapustimo zanko, v kateri smo.
Uporabili bomo neskončno zanko, tako da bo pogoj vedno resničen. Znotraj zanke bomo spraševali po številu toliko časa, dokler ne vnesemo *k*. Takrat bomo zanko prekinili s stavkom *break*.

PRIMER 5.1 – 8: *VsotaVnesenihStevil.java*

```
 1:    import javax.swing.*;
 2:
 3:    public class VsotaVnesenihStevil {
 4:      public static void main(String[] args) {
 5:        int vsota = 0;
 6:        int stevec = 1;
 7:
 8:        System.out.println("Za konec vnesi k.\n");
 9:
10:        // sestevamo vnesena stevila dokler ne vnesemo k
11:        while (true) {
12:          String beri =
13:              JOptionPane.showInputDialog("Vnesi " + stevec +
14:                                          ". stevilo: ");
15:          if (beri.equals("k")) {
16:            break;
17:          } // if
18:          else {
19:            vsota = vsota + Integer.parseInt(beri);
20:            stevec++;
21:            System.out.print(beri + " ");
22:          } // else
23:        } // while
24:        System.out.println("\n");
25:        System.out.println("Vsota teh stevil je " + vsota + ".");
26:      } // main
27:    } // VsotaVnesenihStevil
```

Opis programa.
Najprej deklariramo spremenljivko *vsota* in ji priredimo začetno vrednost *0*. V tej spremenljivki bomo hranili trenutno vsoto števil. Zatem deklariramo spremenljivko *stevec* in ji priredimo začetno vrednost, ki je *1*. To spremenljivko potrebujemo za štetje števil.
V 11. vrstici smo uporabili neskončno zanko, saj je pogoj vedno *true* (resničen). Znotraj zanke definiramo pogojni stavek *if* in če je ta resničen s stavkom, prekinemo izvajanje zanke `while`. Če pogoj ni resničen, se izvedejo stavki od 19. do 20. vrstice. Zatem se ponovno vrnemo na začetek *while* zanke in ponovimo postopek, natanko toliko časa, da je pogoj *(beri.equals("k"))* resničen in se zanka zaradi tega, ker se izvede stavek *break;* konča.

Povprečje ocen

Denimo da imamo n predmetov. Iz ocen predmetov izračunajmo povprečje ocen, najmanjšo in največjo oceno. Ocene so cela števila med 1 in 10. Pomagali si bomo z metodama *min()* in *max()* iz razreda *Math*. Prva vrne manjše, druga pa večje število izmed dveh.

PRIMER 5.1 – 9: *Povprecje.java*

```
 1:    import javax.swing.*;
 2:
 3:    public class Povprecje {
 4:      public static void main(String[] args) {
 5:        int vsota = 0; // sestevek ocen
 6:        int stevec = 1; // steje predmete
 7:        int najnizja = 11; // najnizja ocena
 8:        int najvisja = 0; // najvisja ocena
 9:        String beri =
10:            JOptionPane.showInputDialog("Stevilo predmetov:");
11:        int steviloPredmetov = Integer.parseInt(beri);
12:
13:        // sestevamo vnesene predmete ter
14:        // iscemo najnizjo in najvisjo oceno
15:        while(stevec <= steviloPredmetov) {
16:          beri = JOptionPane.showInputDialog("Vnesi " + stevec +
17:                                             ". oceno:");
18:          int ocena = Integer.parseInt(beri);
19:          vsota = vsota + ocena;
20:          najnizja = Math.min(najnizja, ocena);
21:          najvisja = Math.max(najvisja, ocena);
22:          stevec++;
23:        } // while
24:
25:        // povprecje ocen
26:        double povprecje = (double)vsota / steviloPredmetov;
27:
28:        System.out.println("Tvoje povprecje je " +
29:                           povprecje + ".");
30:        System.out.println("Zaokrozeno povprecje je " +
31:                           (int)(povprecje + 0.5) + ".");
32:        System.out.println("Najnizja ocena je " + najnizja);
33:        System.out.println("Najvisja ocena je " + najvisja);
34:      } // main
35:    } // Povprecje
```

Opis programa.
Najprej deklariramo ustrezne spremenljivke in jim priredimo ustrezne začetne vrednosti. Preko okna za vnos podatkov smo napolnili spremenljivko `steviloPredmetov`.
S pomočjo zanke *while* beremo ocene predmetov. Če je novo število (nova ocena) večje od dosedaj največjega, je med vsemi do sedanjimi števili največje prav to *(Math.max(najvisja, ocena))* , če pa je manjše ali enako, pa doslej največje število ostane enako. Podobno velja tudi za najmanjše število! Na vsakem koraku seštevamo ocene. Njihovo vsoto bomo potrebovali za izračun povprečja. Na koncu se izpišemo želene vrednosti.

Ugibanje števila

Napišimo program, ki si bo "izbral" naključno število med 1 in 100, mi pa ga moramo v čim manj poskusih uganiti. Program naj glede na naše ugibanje pove, ali je povedano (vneseno) število premajhno ali preveliko. Ko število uganemo, naj izpiše število poskusov.

V zanki bomo uporabniku ponudili vnos števila in nato glede na vneseno število izpisali ustrezno obvestilo.

PRIMER 5.1 – 10: *UgibanjeStevila.java*

```
 1:    import javax.swing.*;
 2:
 3:    public class UgibanjeStevila {
 4:      public static void main(String[] args) {
 5:        int stevec = 0; // steje stevilo poskusov
 6:        int stevilo = -1; // stevilo, ki ga vnesemo
 7:        // stevilo, ki ga ugibamo
 8:        int nakljucnoStevilo = (int)(100 * Math.random()) + 1;
 9:
10:        while(nakljucnoStevilo != stevilo) {
11:          String beri =
12:              JOptionPane.showInputDialog("Vnesi stevilo:");
13:          stevilo = Integer.parseInt(beri);
14:          stevec++; // nov poskus
15:
16:          if(stevilo < nakljucnoStevilo) {
17:            System.out.println("Stevilo " + stevilo +
18:                               " je premajhno!");
19:          } // if
20:          else if(stevilo > nakljucnoStevilo) {
21:            System.out.println("Stevilo " + stevilo +
22:                               " je preveliko!");
23:          } // else
24:        } // while
25:
26:        System.out.println("\n");
27:        System.out.println("Bravo! Uganil si!");
28:        System.out.println("Stevilo je " +nakljucnoStevilo+ ".");
29:        System.out.println("Stevilo poskusov: " + stevec);
30:      } // main
31:    } // UgibanjeStevila
```

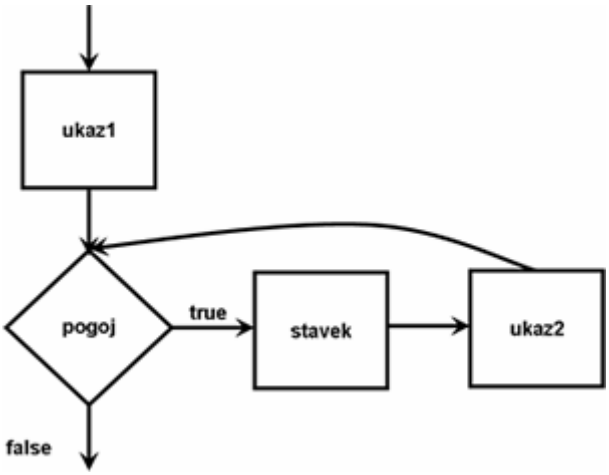
Opis programa:
Najprej definiramo *stevec* s katerim bomo šteli število poskusov in *stevilo,* v katerem bomo hranili vneseno število. Slednjemu dodelimo vrednost -1, zakaj? Odgovor dobimo v 10. vrstici, kjer si podrobneje oglejmo pogoj zanke *while*. V pogoju primerjamo *nakljucnoStevilo*, v katerem hranimo naključno število med 1 in 100 in *stevilo*. Če sta vrednosti spremenljivk različni, vstopimo v zanko. Da je delovanje programa pravilno, moramo v zanko vstopiti vsaj enkrat . To bomo zagotovili le če, pogoj na začetku ne bo izpolnjen. Ker v spremenljivki *nakljucnoStevlo* hranimo cela števila med 1 in 100, moramo začetno vrednost spremenljivke *stevilo* nastaviti na katerokoli število, ki ni med 1 in 100, na primer na -1.
Znotraj zanke *while* sprašujemo po tej naključni številki. Če jo uganemo, se ob naslednjem poskusu izvajanja zanke, le-ta konča. Če število ni enako izbranemu, program izpiše, ali je naše število premajhno ali preveliko in nam tako pomaga čim hitreje uganiti število. Ko uganemo, se izpiše koliko poskusov smo potrebovali, da smo prišli do želenega števila.

5.2 For

For je verjetno najpogosteje uporabljena zanka. Je zelo podobna zanki *while*. Za besedo *for* v oklepajih zapišemo *ukaz1*, *pogoj* in *ukaz2,* ločene z podpičjem. Sledi mu blok stavkov (oz. stavek), ki jih želimo izvajati večkrat – toliko časa, dokler je *pogoj* izpolnjen.

Preden se zanka sploh začne izvajati, se izvede *ukaz1*. Nato se preveri *pogoj*. Če je ta izpolnjen, se izvedejo stavki (oz. stavek), nato še *ukaz2*. Ponovno se preveri *pogoj*. Če je še vedno izpolnjen, se spet izvedejo stavki v telesu zanke in za njimi še *ukaz2*. Če pogoj ni izpolnjen, se zanka konča.

```
for(ukaz1; pogoj; ukaz2) {
  stavki;
}
```



Najprej izvedi **ukaz1**

Dokler **pogoj resničen,** ponavljaj:

izvedi **stavek** in nato še **ukaz2**

Čeprav je v zanki *for ukaz2* napisan pred *stavki*, se najprej izvedejo *stavki* in šele nato *ukaz2*.
Vsako *for* zanko lahko zapišemo z enakovredno *while* zanko:

```
ukaz1
while(pogoj) {
  stavki;
  ukaz2;
}
```

Katero zanko bomo uporabili, je naša izbira. Velja pa nenapisano pravilo, da zanko *for* uporabimo takrat, ko štejemo ponovitve (kolikokrat se zanka ponovi), sicer pa uporabimo zanko *while*.

Zgledi

Izpis števil

Popravimo program *IzpisStevil.java* iz prejšnjega poglavja tako, da bomo uporabili zanko *for*. Izpisati želimo števila od *1* do *10* vsakega v svoji vrstici.

PRIMER 5.2 – 1: *IzpisStevil2.java*

```
1:    public class IzpisStevil2 {
2:      public static void main(String[] args) {
3:        for(int stevilo = 1; stevilo <= 10; stevilo++) {
4:          System.out.println(stevilo);
5:        } // for
6:        System.out.println("Konec zanke");
7:      } // main
8:    } // IzpisStevil2
```

Spremenljivki *stevilo* se najprej priredi začetna vrednost *1(int stevilo = 1)*. Nato se preveri pogoj (*stevilo <= 10*). Če je resničen, se izvedejo stavki v telesu zanke, sicer ne. V

našem primeru je pogoj izpolnjen (*1 <= 10*), zato se izvede stavek v 4. vrstici, ki izpiše vrednost spremenljivke (*1*). Nato se izvede *ukaz2*, ki vrednost spremenljivke *stevilo* poveča za ena. Preveri se pogoj in ker še vedno velja (*2 <= 10*), se s stavkom v 4. vrstici zopet izpiše vrednost spremenljivke (*2*). Na enak način se *for* zanka izvaja, dokler je vrednost v spremenljivki *stevilo* manjša ali enaka *10*. Ko je v spremenljivki vrednost 10, je pogoj še vedno izpolnjen. Ponovno se izvede stavek *System.out.println(stevilo);*, ki izpiše 10. Nato se z *ukazom2* (*stevilo++*) vrednost v spremenljivki *stevilo* zopet poveča za ena in postane *11*. Preveri se pogoj. Ker 11 ni manjše od 10, pogoj ni več izpolnjen, zato se zanka zaključi in program se nadaljuje v vrstici 6. Izpiše se besedilo *Konec zanke*.

Izpis lihih števil

Popravimo program *LihaStevila.java* iz prejšnjega poglavja tako, da bomo uporabili zanko *for*. Izpisati želimo liha števila od *1* do *10*, v isti vrstici in s presledkom kot ločilnim elementom.

PRIMER 5.2 – 2: *LihaStevila1.java*

```
1:    public class LihaStevila1 {
2:      public static void main(String[] args) {
3:        for(int lihoStevilo = 1; lihoStevilo <= 10;
4:            lihoStevilo = lihoStevilo + 2) {
5:          System.out.print(lihoStevilo + " ");
6:        } // for
7:      } // main
8:    } // LihaStevila1
```

Spremenljivki *lihoStevilo* se najprej priredi začetna vrednost *1*. Nato se preveri pogoj. Ker je resničen *(1 <= 10)*, vstopimo v zanko in izpiše se vrednost spremenljivke (*1*). Nato se izvede ukaz2 (*lihoStevilo = lihoStevilo + 2*), ki vrednost spremenljivke *lihoStevilo* poveča za dva. Preveri se pogoj in ker še vedno velja *(3 <= 10)*, se zopet izpiše vrednost spremenljivke (*3*). Na ta način se zanka *for* izvaja, dokler je vrednost v spremenljivki *lihoStevilo* manjša ali enaka *10*. Ko je v spremenljivki vrednost *9*, je pogoj izpolnjen in 5. vrstica izpiše to število. Sedaj spremenljivka *lihoStevilo* dobi vrednost *11*. Preveri se pogoj. Ker *11* ni manjše od *10*, pogoj ni več izpolnjen, zato se zanka konča.

Tudi pri zanki for moramo paziti, da se zanka konča. V primeru nepravilne uporabe tudi zanka *for* teče v neskončnost (se zacikla).

PRIMER 5.2 – 3: *Ciklanje.java*

```
1:    public class Ciklanje {
2:      public static void main(String[] args) {
3:        for(int stevilo = 1; stevilo > 0; stevilo++) {
4:          System.out.println(stevilo);
5:        } // for
6:      } // main
7:    } // Ciklanje
```

Program prevedemo in poženemo:

```
1
2
3
4
5
6
.
.
```

Velja enaka opazka, kot smo jo naredili pri zanki *while*. Tudi tu bi zaradi prekoračitve obsega prišli do negativnih števil in dejansko bi se zanka ustavila. A programirati na način, ko izrabljamo, da za *1* povečano največje možno število tipa *int* postane negativno, je zelo nevarno in ga ne smemo uporabljati.

Obratni izpis

Napišimo program, ki bo vneseni niz izpisal v obratnem vrstnem redu.

Tu si bomo pomagali z metodo *charAt(i)*, ki vrne znak z indeksom *i* ter z metodo *length()*, ki vrne dolžino niza. Ker v nizih znake indeksiramo od 0 dalje, nam *niz.charAt(i)* vrne i+1-ti znak niza *niz*.

PRIMER 5.2 – 4: *ObratniIzpis.java*

```
 1:    import javax.swing.*;
 2:
 3:    public class ObratniIzpis {
 4:      public static void main(String[] args) {
 5:        String beri =
 6:            JOptionPane.showInputDialog("Vnesi niz:");
 7:        System.out.println(beri);
 8:
 9:        for(int i = beri.length() - 1; i >= 0; i--) {
10:          System.out.print(beri.charAt(i));
11:        } // for
12:        System.out.println("\n");
13:      } // main
14:    } // ObratniIzpis
```

Opis programa.
Če želimo niz izpisati v obratnem vrstnem redu, bomo izpisovali znake od zadnjega proti prvemu. Recimo, da smo vnesli niz *"abeceda"*. Spremenljivki *i* se najprej priredi začetna vrednost *beri.length() - 1*. Dolžina niza je enaka številu vnesenih znakov, kar je v našem primeru 7. Ker se znaki štejejo od 0 dalje, ima naš zadnji znak indeks *length() - 1*. V našem primeru se spremenljivki *i* najprej priredi začetna vrednost 6. Nato se preveri pogoj. Ker je resničen *(6 >= 0)*, vstopimo v zanko in izpiše se znak na mestu *i (a)*. Nato se izvede ukaz2 *(i--)*, ki vrednost spremenljivke *i* zmanjša za ena. Preveri se pogoj in ker še vedno velja *(5 >= 0)*, se zopet izpiše znak na mestu *5* (*d*). Na ta način se zanka *for* izvaja, dokler je vrednost v spremenljivki *i* večja ali enaka *0*. Ko je v spremenljivki vrednost *0*, je pogoj še vedno izpolnjen in 10. vrstica izpiše znak na mestu *0,* torej prvi znak. Sedaj spremenljivka *i* dobi vrednost *-1*. Preveri se pogoj. Ker *-1* ni večje od *0*, pogoj ni več izpolnjen. Zanka se konča in program se nadaljuje v vrstici 12. Izpiše se prazna vrsta.

Abeceda

Napišimo program, ki bo izpisal angleško abecedo. Prvič naprej, drugič pa v obratnem vrstnem redu.

PRIMER 5.2 – 5: *Abeceda.java*

```
 1:    public class Abeceda {
 2:      public static void main(String[] args) {
 3:        for(char i = 'a'; i <= 'z'; i++) {
 4:          System.out.print(i + " ");
 5:        } // for
 6:
```

```
 7:          System.out.println("\n");
 8:
 9:          for(char j = 'z'; j >= 'a'; j--) {
10:            System.out.print(j + " ");
11:          } // for
12:        } // main
13:      } // Abeceda
```

Program prevedemo in poženemo:

```
a b c d e f g h i j k l m n o p q r s t u v w x y z
z y x w v u t s r q p o n m l k j i h g f e d c b a
```

Opis programa.
Kot smo že rekli, lahko na znake gledamo kot na števila. Zato smo v spremenljivki *i* pravzaprav shranili kodo znaka mali *a*. Ta koda je neko naravno število. Če so znaki kodirani po ASCII standardu, se v *a* shrani število *97*. Preveri se pogoj in ker je *97 <= 122* (koda znaka *'z'*) se izpiše znak s kodo *97*, torej *a*. Vrednost v spremenljivki *i* se poveča za ena (*i++*). Zopet se preveri pogoj in ker je resničen, se izpiše znak s kodo *98*. Na ta način se zanka *for* izvaja, dokler je vrednost v spremenljivki *i* manjša ali enaka *'z'* (*122*). Ko je v spremenljivki *i* vrednost *122*, je pogoj izpolnjen in 4. vrstica izpiše znak s kodo 122, torej *z*. Sedaj spremenljivka *i* dobi vrednost *123*. Preveri se pogoj. Ker *123* ni manjše ali enako *122*, pogoj ni več izpolnjen. Zato se zanka konča in program se nadaljuje v vrstici 7. Izpiše se prazna vrsta. Program se nadaljuje v vrstici 9. Če želimo izpisati abecedo v obratnem vrstnem redu, začnemo izpisovati znake od zadnjega proti prvemu. V spremenljivki j shranimo kodo znaka mali *z* (*122*). Preveri se pogoj in ker je *122 >= 97* (koda znaka mali *a*), se izpiše znak s kodo *122*, torej *z*. Vrednost v spremenljivki *j* se zmanjša za ena (*j--*) in zopet se preveri pogoj. Ker je resničen, se izpiše znak s kodo *121*. Na ta način se zanka *for* izvaja, dokler je vrednost v spremenljivki *j* večja ali enaka *'a'* (*97*). Ko je v spremenljivki vrednost *97*, je pogoj izpolnjen in 10. vrstica izpiše znak s kodo *97*, torej *a*. Sedaj spremenljivka *i* dobi vrednost *96*. Preveri se pogoj. Ker *96* ni večje ali enako *97*, pogoj ni več izpolnjen, zato se zanka konča.

MonteCarlo

Izračunajmo število π po metodi MonteCarlo. Izberemo koordinate točke v kvadratu s stranico 1. Če to naredimo velikokrat, je razmerje med številom tistih, ki ležijo v krogu z radijem 1 in vsemi, je pi/4.

Denimo, da bomo izbrali 10000000 točk. Izberimo naključno točko v enotskem kvadratu. Nato pogledamo ali se nahaja v enotskem krogu, če se povečamo število zadetkov za ena, sicer pa ne. Število π izračunamo po naslednji enačbi π = 4.0 * zadetki / poskusi.

PRIMER 5.2 – 6: *MonteCarlo.java*

```
 1:   public class MonteCarlo {
 2:        public static void main(String[] args)
 3:          int n = 10000000;
 4:          int k = 0;
 5:          for (int i = 0; i < n; i = i + 1) {
 6:              double x = Math.random();
 7:              double y = Math.random();
 8:              if ( x*x + y*y <= 1)
 9:                  k = k + 1;
10:          }
11:          System.out.println("V "+n+" metih dobimo, da je pi="
12:                                                  + (4.0 * k / n));
13:      }
14:   }
```

```
V 10000000 metih dobimo, da je pi=3.1410292
```

6 TABELE

Pogosto se srečamo z večjo količino podatkov istega tipa, nad katerimi želimo izvajati podobne (ali enake) operacije. Takrat si pomagamo s tabelami. Pri tabelah imamo opravka z indeksiranimi spremenljivkami.

6.1 Deklaracija tabele

Prvi korak pri ustvarjanju tabele je najava le te. To storimo tako, da za tipom tabele navedemo oglate oklepaje in ime tabele.

```
podatkovniTip[] imeTabele;
```

Z zgornjim stavkom zgolj napovemo spremenljivko, ki bo hranila naslov bodoče tabele, ne zasedemo pa še nobene pomnilniške lokacije, kjer bodo elementi tabele dejansko shranjeni. Potrebno količino zagotovimo in s tem tabelo dokončno pripravimo z ukazom `new`:

```
imeTabele = new podatkovniTip[dolzina];
```

Ukaz `new` zasede dovolj pomnilnika, da vanj lahko shranimo `dolzina` spremenljivk ustreznega tipa in vrne njegov naslov. Velikost tabele je enaka vrednosti spremenljivke `dolzina` in je enaka številu elementov, ki jih lahko hranimo v tabeli, le to lahko izvemo z ukazom `imetabele.length`.

Napoved in zasedanje pomnilniške lokacije lahko združimo v en stavek:

```
podatkovniTip[] imeTabele = new podatkovniTip[dolzina];
```

Primeri najave tabele:

```
// tabela 10 celih števil
int[] stevila = new int[10];

// tabela 3 realnih števil
double[] cena = new double[3];

// tabela 4 znakov
char[] tabelaZnakov = new char[4];

// tabela 500 nizov
String[] ime = new String[500];
```

Sočasno z najavo lahko elementom določimo tudi začetne vrednosti. Te zapišemo med zavita oklepaja. Posamezne vrednosti so ločene z vejico.

Primer:

```
int[] stevila = {1, 2, 3, 6, 8, 12, 14, 4, 7, 9};
```

Ob naštevanju vrednosti ne moremo navesti dolžine tabele, saj jo prevajalnik izračuna sam. Paziti moramo, da vsi elementi ustrezajo izbranemu tipu.

Omenimo še enkrat, da dolžino tabele *stevila* (število elementov, ki jih lahko shranimo v tabelo) dobimo z izrazom *stevila.length*. V našem primeru je dolžina tabele enaka 10.

Znotraj tabele z določeno velikostjo lahko vsak posamezen element dosežemo z zaporedno številko – njegovim indeksom. V Javi se indeksi tabele štejejo od *0* dalje. Če je v spremenljivki *dolzina* shranjena velikost tabele, je zadnji element označen z indeksom *dolzina - 1*. Indeksi elementov tabele torej tečejo *0* do *dolzina - 1*. Do *i*–tega elementa dostopamo z izrazom *imeTabele[i - 1]*.

S posameznimi elementov tabele lahko operiramo enako kot s spremenljivkami tega tipa. Tako je v spodnjem zgledu npr. *tabela[4]* povsem običajna spremenljivka tipa *int*.

Primer:

```
int[] tabela = new int[10]; // ustvarimo tabelo dolžine 10
// tabelo napolnimo z vrednostmi
tabela[0] = 10;
tabela[1] = 20;
tabela[2] = 31;
.
.
.
tabela[8] = 74;
tabela[9] = 97;
```

Poglejmo si preprost primer tabele imen. Napolnimo tabelo z imeni in izpišimo njeno dolžino.

PRIMER 6.1 – 1: *TabelaImen.java*

```
 1:    public class TabelaImen {
 2:      public static void main(String[] args) {
 3:        String[] imena={"Jan", "Matej", "Ana", "Grega", "Sanja"};
 4:
 5:        System.out.println("Dolzina tabele je " +
 6:                           imena.length + ".");
 7:        System.out.println("Ime na sredini tabele je  " +
 8:                           imena[imena.length / 2] + ".");
 9:      } // main
10:    } // TabelaImen
```

Program prevedemo in poženemo:

```
Dolzina tabele je 5.
Ime na sredini tabele je Ana.
```

Opis programa.
V 3. vrstici smo najavili tabelo *imena* tipa *String[]* in jo napolnili z elementi. Zatem smo izpisali njeno dolžino z ukazom *imena.length*. V 7. vrstici želimo izpisati element, ki se nahaja na sredini naše tabele. Izraz *imena.length / 2* nam vrne *2*, element, ki se nahaja na mestu *imena[2]* je *Ana*.

Podrobneje si poglejmo, kako indeksiramo podatke v tabeli *imena*.

Prvi element tabele ima indeks *0* in vsebuje niz *"Jan"*, drugi ima indeks *1* in vsebuje niz *"Matej"*, tretji ima indeks *2* in vsebuje niz *"Ana"*, četrti ima indeks *3* in vsebuje niz *"Grega"* in peti ima indeks *4* in vsebuje niz *"Sanja"*. Tabelo *imena* si lahko predstavljamo takole:

indeks	*0*	*1*	*2*	*3*	*4*
vrednost	Jan	Matej	Ana	Grega	Sanja

Kaj se bo zgodilo, če napišemo naslednji program.

PRIMER 6.1 – 2: *TabelaStevil.java*

```
1:    public class TabelaStevil {
2:      public static void main(String[] args) {
3:        String[] stevila = {1, 2, 3};
4:
5:        System.out.println("Dolzina tabele je " +
6:                           stevila.length + ".");
7:      } // main
8:    } // TabelaStevil
```

Program se ne prevede. Če pogledamo vrstico 3, smo deklarirali tabelo nizov, tej tabeli pa smo priredili celoštevilske vrednosti. Ker se tipa ne ujemata, bo prevajalnik javil:

```
C:\TabelaStevil.java:3: incompatible types
found   : int
required: java.lang.String
    String[] stevila = {1, 2, 3};
                       ^
C:\TabelaStevil.java:3: incompatible types
found   : int
required: java.lang.String
    String[] stevila = {1, 2, 3};
                          ^
C:\TabelaStevil.java:3: incompatible types
found   : int
required: java.lang.String
    String[] stevila = {1, 2, 3};
                             ^
3 errors
```

Napako popravimo tako, da napišemo ustrezen (pravilen) tip.

PRIMER 6.1 – 3: *TabelaStevil.java*

```
1:    public class TabelaStevil {
2:      public static void main(String[] args) {
3:        int[] stevila = {1, 2, 3};
4:
5:        System.out.println("Dolzina tabele je " +
6:                           stevila.length + ".");
7:      } // main
8:    } // TabelaStevil
```

Program prevedemo in poženemo:

```
Dolzina tabele je 3.
```

6.2 Izpis tabele

Poskusimo izpisati vse elemente tabele *imena* na naslednji način.

PRIMER 6.2 – 1: *IzpisImen.java*

```
1:    public class IzpisImen {
2:      public static void main(String[] args) {
3:        String[] imena={"Jan", "Matej", "Ana", "Grega", "Sanja"};
4:
5:        System.out.println(imena);
6:      } // main
7:    } // IzpisImen
```

Program prevedemo in poženemo:

```
[Ljava.lang.String;@45a877
```

Dobimo čuden izpis. V spremenljivki *imena* namreč ne hranimo tabele imen, ampak le naslov, kje se tabela nahaja. Z ukazom *System.out.println(imena)* izpišemo naslov, kjer se nahaja naša tabela, in ne posameznih elementov. Če želimo izpisati vrednosti slednjih, jih moramo obravnavati vsakega posebej. Omenili smo že, da do *i*–tega elementa (natančneje, do elementa z indeksom i) dostopamo z izrazom *imeTabele[i]*. Popravimo program. Za izpis vseh elementov tabele bomo uporabili zanko *for*,

PRIMER 6.2 – 2: *IzpisTabeleFor.java*

```
1:    public class IzpisTabeleFor {
2:      public static void main(String[] args) {
3:       String[] imena={"Jaka", "Matej", "Ana", "Grega", "Sanja"};
4:
5:       for(int i = 0; i < imena.length; i++)
6:          System.out.println(imena[i]);
7:      } // main
8:    } // IzpisTabeleFor
```

Program prevedemo in poženemo:

```
Jaka
Matej
Ana
Grega
Sanja
```

Opis programa:
Najprej definiramo tabelo tipa *String* in jo napolnimo s podatki. Njeno vsebino bomo izpisali na ekran s pomočjo zanke *for*. Definiramo spremenljivko *i* in ji priredimo vrednost *0* , zatem preverimo če je *i* manjše od dolžine tabele, ki je v našem primeru *5* (pogoj je izpolnjen), izpišemo spremenljivko, ki se nahaja na mestu *imena[0]* to je *Jaka*. Vrednost *i* povečamo za ena in ponovno preverimo resničnost pogoja (*i < imena.length*), ker je pogoj ponovno resničen izpišemo vrednost spremenljivke *imena[1] Matej* , …postopek nadaljujemo toliko časa, da izpišemo še preostal del tabele, ko pogoj ni več izpolnjen (izpisali smo vse vrednosti tabele) končamo.

Poskusimo izpisati peti element tabele *imena*.

PRIMER 6.2 – 3: *IzpisImena.java*

```
1:    public class IzpisImena {
2:      public static void main(String[] args) {
3:       String[] imena={"Jan", "Matej", "Ana", "Grega", "Sanja"};
```

```
4:
5:          System.out.println(imena[5]);
6:        } // main
7:     } // IzpisImena
```

Program prevedemo in poženemo:

```
java.lang.ArrayIndexOutOfBoundsException
  at IzpisImena.main(IzpisImena.java:5)
Exception in thread "main"
```

Prevajalnik javi napako, saj smo poskušali poklicati element, ki je izven definirane meje. Kot smo že povedali, se veljavni indeksi vedno gibljejo v meji od *0* do *dolzina - 1*, v našem primeru od *0* do *4*.

Oglejmo si še en primer pogoste napake. Denimo, da želimo primerjati dve tabeli. Zanima nas, če so vsi istoležni elementi enaki. "Naivna" rešitev z *t1 == t2* nam ne vrne pričakovan rezultat.

PRIMER 6.2 – 4: *PrimerjavaTabel.java*

```
1:   public class PrimerjanjeTabel{
2:         public static void main(String[] args) {
3:          int[] t1 = new int[] {1, 2, 3};
4:          int[] t2 = new int[] {1, 2, 3};
5:          System.out.println(t1 == t2);
6:       } //main
7:   }
```

```
false
```

Opis programa.
Čeprav obe tabeli vsebujeta enake elemente, nam izpis pove, da tabeli nista enaki. Zakaj? Spomnimo se, da *t1* in *t2* vsebujeta samo naslov, kjer se nahajata tabeli. Ker sta to dve različni tabeli (sicer z enakimi elementi, a vseeno gre za dve tabeli), zato tudi naslova nista enaka. In to nam pove primerjava *t1 == t2*. Oglejmo si, kako bi pravilno primerjali dve tabeli.

PRIMER 6.2 – 3:*PrimerjavaTabel1.java*

```
1:   public class PrimerjavaTabel1{
2:       public static void main(String[] args) {
3:           int[] t1 = new int[] {1, 2, 3};
4:           int[] t2 = new int[] {1, 2, 3};
5:
6:           boolean je_enaka = true;
7:
8:           if (t1.length == t2.length) {
9:               for(int i = 0; i < t1.length; i++) {
10:                  if (t1[i] != t2[i])
11:                      je_enaka = false;
12:              }
13:          }
14:          else je_enaka = false;
15:
16:          if (je_enaka == true)
17:              System.out.println("Tabeli sta enaki.");
18:          else System.out.println("Tabeli nista enaki.");
19:      } //main
20:  }
```

Opis programa.
V 8. vrstici najprej preverimo če sta tabeli enako veliki. Če je ta pogoj izpolnjen, nadaljujemo s primerjavo elementov znotraj tabele. Z zanko for se sprehodimo po vseh elementih in na vsakem koraku primerjamo `t1[0] != t2[0], t1[1] != t2[1], …` Če bi naleteli vsaj na en par neenakih števil, bi spremenljivko `je_enaka` nastavili na `false,` saj tabeli potem nista enaki. V našem primeru pa je ta pogoj vedno neresničen, saj so `1 != 1, 2 != 2, …` neresnične izjave,. zato izpišemo `Tabeli sta enaki.` Če pa naletimo na tabeli, ki nista enako veliki ali pa je pogoj v 10. vrstici resničen vsaj enkrat, se izpiše `Tabeli nista enaki.`

6.3 Zgled

Začeli smo zbirati sličice. V album moramo nalepiti 250 sličic. Zanima nas, koliko sličic bomo morali kupiti, da bomo napolnili album. Seveda ob nakupu ne vemo, katero sličico bomo dobili. Ker bi to radi vedeli še preden se bomo zares spustili po nakupih, bi radi polnjenje albuma simulirali s pomočjo računalniškega programa. In če bomo to simulacijo ponovili dovolj mnogokrat, bomo že dobili občutek o številu potrebnih nakupov. Pri tem seveda naredimo nekaj predpostavk:

- hkrati vedno kupimo le eno sličico.
- Vse sličice so enako pogoste. Torej je verjetnost, da bomo kupili i-to sličico ravno 1/250.
- Podvojenih sličic ne menjamo.

Poglejmo, kako bi sestavili program.
Naš album bo tabela. Če bo vrednost ustreznega elementa `false`, to pomeni da sličice še nimamo. Da nam ne bo po vsakem nakupu treba prečesati vsega albuma, da bi ugotovili, ali kakšna sličica še manjka, bomo vodili evidenco o tem, koliko sličic v albumu še manjka. V zanki, ki se bo odvijala toliko časa, dokler število manjkajočih sličic ne bo padlo na nič, bomo kupili sličico (naključno generirali število med 0 in 249). Če je še nimamo, bomo zmanjšali število manjkajočih sličic in si v albumu označili, da sličico imamo.

PRIMER 6.3 – 2: *Album.java*

```
 1:   public class Album {
 2:       public static void main(String[] args) {
 3:       int st_slicivalbumu = 250;
 4:       int st_zapolnjenih = 0;
 5:       int st_kupljnihslic = 0;
 6:       // album
 7:       boolean[] album = new boolean[st_slicivalbumu];
 8:       // polnimo album
 9:       while (st_zapolnjenih < st_slicivalbumu) {
10:           st_kupljnihslic = st_kupljnihslic + 1;
11:           int st_slikic=(int)(Math.random() * st_slicivalbumu);
12:           // jo damo v album
13:           if (!album[st_slikic]) {
14:               album[st_slikic] = true;
15:               st_zapolnjenih = st_zapolnjenih + 1;
16:               }
17:           }
18:       System.out.println("Da smo napolnili album, smo kupili "+
19:                          + st_kupljnihslic + " slikic.");
20:       }
21:   }
```

Program prevedemo in poženemo:

```
Da smo napolnili album, smo kupili 1661 slikic.
```


Opis programa:

V uvodu definiramo tri spremenljivke s katerimi nadzorujemo stanje album. V 9. vrstici začnemo polniti album, za to uporabili zanko `while`. Ob vsakem vstopu v zanko kupimo sličico , ki ji dodelimo naključno mesto v albumu (naključno generirali število med 0 in 249). Preverimo če že imamo to mesto zapolnjeno (`!album[st_slikic]`). Če ne, jo vstavimo v album in povečamo število zapolnjenih mest v albumu. To ponavljamo toliko časa, da je pogoj v zanki resničen. Na koncu še izpišemo, koliko sličic smo morali kupiti, da smo napolnili album.

7 METODE

Programi, ki smo jih napisali do sedaj, so bili precej preprosti in kratki, zato smo jih vse pisali v enem kosu znotraj metode main. Programi napisani v obliki ene same metode so lahko preobsežni in težko razumljivi. Zato program razdelimo v več manjših "neodvisnih" postopkov in nato obravnavamo vsak postopek posebej. V ta namen uporabljamo metode, ki jim v drugih programskih jezikih rečemo tudi funkcije, procedure ali podprogrami. Prednost uporabe metod je tudi v tem, da lahko enkrat zapisano metodo izvajamo večkrat. Naslednja zelo pomembna prednost metod je, da jih lahko uporabimo tudi v drugih programih. Nekaj vgrajenih metod smo že srečali (`abs, pow, min, max, print, println, ...`), v tem poglavju pa si bomo pogledali kako pišemo in uporabljamo lastne metode.

V Javi obstajata dve vrsti metod, **statične** in nestatične oziroma **objektne** metode. Zaenkrat povejmo le, da statične metode kličemo nad razredom, medtem, ko objektne nad objektom. V tej diplomski nalogi si bomo ogledali le statične metode. Metode ločimo tudi po načinu dostopa na **javne** (public), **privatne** (private) in **zaščitene** (protected). Ker bomo uporabljali le javne metode, se v način dostopa ne bomo spuščali.

7.1 Metode

Definicija metode

Metodo v program vpeljemo z definicijo. Metodo definiramo na naslednji način:

```
vrsta tipRezultata imeMetode(argumenti) {
  // deklaracije spremenljivk
  // stavki
}
```

Definicija metode je sestavljena iz glave oziroma deklaracije metode in telesa metode.
V deklaracijo zapišemo najprej **dostop** do metode (torej ali je metoda javna (`public`), privatna (`private`) ali zaščitena (`protected`)), nato **vrsto** metode s katero povemo ali je metoda statična (`static`) ali objektna,. V tej diplomski nalogi bodo vse naše metode vrste `public static`. Sledi ji **tip** rezultata metode. Tip rezultata metode je poljuben podatkovni tip (npr. `int`, `double[]`, `String`) in pomeni tip vrednosti, ki ga metoda vrača. Nato z **imenom metode** povemo kako se imenuje metoda. Velja dogovor, da se imena metod začnejo z malo črko. Metodam damo smiselna imena, ki povedo, kaj metoda dela. Za imenom metode zapišemo okrogle oklepaje, ki so obvezni del deklaracije. Znotraj oklepajev zapišemo **argumente metode**, ki jih metoda sprejme (ime argumentov) in kakšnega tipa so. Več argumentov med sabo ločimo z vejico. Če metoda ne sprejme nobenih argumentov, napišemo le prazne okrogle oklepaje `()`. Sledi **telo metode**, to je del, ki je znotraj bloka `{}`. V telo metode zapišemo stavke, ki izvršijo nalogo metode.
Posebne so metode, ki ne vračajo nobenega rezultata, ampak le opravijo določene delo (na primer izpišejo kaj, narišejo sliko, pošljejo datoteko na tiskalnik, ...). Pri takih metodah kot tip rezultata navedemo tip `void`.

Poglejmo si nekaj primerov deklaracij metod.

Primeri:

```
public static int beri()
```

Metoda `beri` je javna (`public`), statična (`static`), vrača rezultat tipa `int` in ne sprejme nobenega argumenta (znotraj oklepajev ni argumentov).

```
public static void nekaj(String[] tabNiz)
```

Metoda `nekaj` je javna (`public`), statična (`static`), ne vrača rezultata (tip `void`) in sprejme en argument, ki se imenuje `tabNiz` in je tipa `String[]` (tabela nizov).

```
public static int max(int a, int b)
```

Metoda `max` je javna (`public`), statična (`static`), vrača rezultat tipa `int` in sprejme dva argumenta, ki sta celi števili (tip `int`), prvi argument se imenuje `a`, drugi `b`.

```
public double abs()
```

Metoda `abs` je javna (`public`), objektna (ni določila `static`), vrača rezultat tipa `double` in ne sprejme nobenega argumenta (znotraj oklepajev ni argumentov).

Vrednost metode

Če je tip metode različen od `void`, moramo vrednost, ki jo vrne metoda, določiti s stavkom

```
return izraz;
```

Vrednost metode je vrednost izraza, ki je naveden za ključno besedo `return`. V opisu postopka je stavek `return` lahko na več mestih. Kakor hitro se eden izvede, se izvajanje metode konča z vrednostjo, kot jo določa izraz, naveden pri stavku `return`. Če metoda kaj izpisuje ali riše na ekran, to ni rezultat te metode, ampak samo stranski učinek, ki ga ima metoda. Take metode običajno ne vračajo nobenih rezultatov. V tem primeru je rezultat metode tipa `void`.

Primer:

```
public static double povprecje(double x, double y) {
  return (x + y) / 2.0;
}
```

Metoda povprečje je javna, statična, vrne rezultat tipa `double` in sprejme dva argumenta, ki sta tipa `double`, prvi se imenuje `x` in drugi `y`.

Primera:

```
public static void main(String[] args) {
  System.out.println("Danes je lep dan.");
}
```

Metoda `main` je javna, statična, ne vrne rezultata (tip `void`), sprejme argument tipa `String[]`, ki se imenuje `args`.


```
public static void izpis(String ime, String priimek) {
```

```
    System.out.println(ime + " " + priimek);
  }
```

Metoda izpis je javna, statična, ne vrne rezultata (tip `void`), sprejme dva argumenta tipa `String`, ki se imenujeta `ime` in `priimek`.

Pogosta napaka, ki jo naredimo, ko sestavljamo metodo, je ta da rezultat, ki ga metoda vrne, izpišemo že znotraj nje. Vendar to ni naloga metode. Ta naj le izračuna določen rezultat. Kaj se bo s tem rezultatom zgodilo, odloča tisti del programa, ki je metodo klical. Morda bo rezultat izpisal ali pa ga le uporabil v naslednjem izračunu.
Podobno je z branjem. Ni nujno, da bomo določene podatke, od katerih je odvisna metoda, vedno prej prebrali. Morda bo podatki odvisni od prejšnjih izračunov, drugič pa ne. Metodi vse potrebne vrednosti posredujemo prek prenosa argumentov. V metodah torej praviloma ne beremo podatkov in tudi izpisujemo jih ne. Seveda so izjeme metode, ki so namenjene branju in izpisovanju.

Klic metode

Metodo uporabimo tako, da jo pokličemo. Ob klicu metode se začnejo izvajati stavki, ki so zapisani v metodi. Metodo pokličemo tako, da navedemo njeno ime, ki mu v okroglih oklepajih sledijo dejanski argumenti. Dejanski argumenti so izrazi, ki določajo začetno vrednost argumentov, napisanih v deklaraciji metode. Navedeni argumenti se morajo ujemati s tistimi v deklaraciji metode tako po številu kot tudi po tipu. Če je argumentov več, jih ločimo z vejico. Če metoda ne sprejme nobenih argumentov, napišemo `()`. Znotraj metode lahko kličemo že vgrajene Javine metode kot tudi naše lastne metode.

Metodo pokličemo na naslednji način:

```
imeMetode(argument1, argument2, .., argumentN);
```

Primera:

```
povprecje(7.8, 8.3);

izpis("Maja", "Jelen");
```

Metoda main

Metoda `main` se od ostalih metod razlikuje po tem, da jo kliče operacijski sistem in ne uporabnik. Definirana je na naslednji način

```
public static void main(String[] args) {
  // deklaracije spremenljivk
  // stavki
}
```

Metoda `main` je javna (`public`), statična (`static`), ne vrača rezultata (tip `void`) in sprejme en argument, ki se imenuje `args` in je tipa `String[]` (tabela nizov).

Zgledi

Povprečje dveh števil

Napišimo preprost program, ki bo iz vrednosti shranjenih v dveh celoštevilskih spremenljivkah izračunal povprečje teh dveh števil. Izračun povprečja bomo opravili z metodo `public static double povprecje(int a, int b)`.

Preden se lotimo pisanja programa, poglejmo kaj metoda *povprecje* sprejme in kaj vrača. Ker je metoda *povprecje* odvisna od dveh celoštevilskih argumentov, sprejme dva celoštevilska argumenta *a* in *b*. Ker je povprečje dveh celih števil lahko realna vrednost, metoda *povprecje* vrne rezultat tipa *double*.

PRIMER 7.1 – 1: *PovprecjeDvehStevil.java*

```
 1:   public class PovprecjeDvehStevil {
 2:       public static void main(String[] args) {
 3:           int stevilo1 = 4;
 4:           int stevilo2 = 6;
 5:
 6:           System.out.println(povprecje(stevilo1, stevilo2));
 7:           System.out.println("Konec programa.");
 8:       } // main
 9:
10:       public static double povprecje(int a, int b) {
11:           return (a + b) / 2.;
12:       } // povprecje
13:   } // PovprecjeDvehStevil
```

Program prevedemo in poženemo:

```
5.0
Konec programa.
```

Opis programa.
Program začnemo z deklaracijo dveh spremenljivk. V 6 . vrstici kličemo metodo *povprecje(int a, int b)*. V našem primeru kličemo *povprecje(4 , 6)*, program se nadaljuje v 10. vrstici kjer vrnemo vrednost (4 + 6) / 2. = 5.0. Ponovno se vrnemo v 6. vrstico kjer dobljeno vrednost 5.0 izpišemo.

Popravimo program tako, da bomo preko sporočilnega okna prebrali celoštevilski vrednosti. Podatka bomo prebrali z metodo, ki jo bomo poimenovali *preberiCeloStevilo*. Metoda bo brez argumentov, ker ni odvisna od nobenega argumenta. Vrnila pa bo celoštevilsko vrednost.

PRIMER 7.1 – 2: *PovprecjeDvehStevil1.java*

```
 1:   import javax.swing.*;
 2:
 3:   public class PovprecjeDvehStevil1 {
 4:       public static void main(String[] args) {
 5:           int stevilo1 = preberiCeloStevilo();
 6:           int stevilo2 = preberiCeloStevilo();
 7:
 8:           System.out.println(povprecje(stevilo1, stevilo2));
 9:           System.out.println("Konec programa.");
10:           } // main
11:
12:       public static int preberiCeloStevilo() {
13:           int stevilo;
14:           String beri =
15:           JOptionPane.showInputDialog("Vnesi celo stevilo: ");
16:           stevilo = Integer.parseInt(beri);
17:           return stevilo;
18:       } // preberiCeloStevilo
19:
20:       public static double povprecje(int a, int b) {
21:           return (a + b) / 2.;
```

```
22:     } // povprecje
23: } // PovprecjeDvehStevil1
```

Programu smo dodali metodo *preberiCeloStevilo* in jo uporabili v vrsticah 5 in 6. Spremenljivki *stevilo1* priredimo ustrezna vrednost tako, da pokličemo metoda *preberiCeloStevilo* (vrstice 12 do 18). S stavkoma v vrsticah 13 do 16 preberemo podatek in ga pretvorimo v celo število. Z ukazom *return* vrnemo vrednost, ki je shranjena v spremenljivki *stevilo*. Torej spremenljivki *stevilo1* se priredi vrednost, ki jo vrne metoda *preberiCeloStevilo*.
v 6. vrstici spremenljivki *stevilo2* priredimo ustrezno vrednost tako, da ponovno pokličemo metodo *preberiCeloStevilo* (vrstice 12 do 18). S stavkoma v vrsticah 13 do 16 preberemo podatek in ga pretvorimo v celo število. Z ukazom *return* vrnemo vrednost, ki je shranjena v spremenljivki *stevilo*. Torej spremenljivki *stevilo2* se priredi vrednost, ki jo vrne metoda *preberiCeloStevilo*.

Parametri metode main

Parametre metodi main vnašamo preko ukazne vrstice. Ob zagonu programa dodamo niz parametrov, ki se shranijo kot vrednosti tabele, ki je parameter metode *main* . Oglejmo si primer:

PRIMER 7.1 – 3: *BranjeIzUkazneVrstice.java*

```
1:  public class BranjeIzUkazneVrstice {
2:      public static void main(String[] args) {
3:          System.out.println("Stevilo posredovanih parametrov: "
4:                  + args.length);
5:
6:          for(int i = 0; i < args.length; i++)
7:          System.out.println("args[" + i + "]: " + args[i]);
8:      } // main
9:  } // BranjeIzUkazneVrstice
```

Program prevedemo in poženemo:

Število posredovanih parametrov: 0

Takšen rezultat dobimo, ker nismo preko ukazne vrstice vnesli nobenega parametra. Program bi morali pognati z ustrezno ukazno vrstico. To lahko storimo, če na primer program poženemo z ukaznim nizom:

```
java BranjeIzUkazneVrstice prvi drugi tretji

Stevilo posredovanih parametrov: 3
args[0]: prvi
args[1]: drugi
args[2]: tretji
```

Zgled

Povprečje ocen

Napišimo program, ki bo preko ukazne vrstice sprejel niz naših ocen in nam vrnil povprečno, najnižjo in najvišjo oceno.

PRIMER 7.1 – 4: *PovprecjeOcen.java*


```
1:  public class PovprecjeOcen {
```

```
 2:      public static void main(String[] args) {
 3:          int[] ocene = new int[args.length];
 4:
 5:          for(int stevec = 0; stevec < args.length; stevec++) {
 6:              ocene[stevec] = Integer.parseInt(args[stevec]);
 7:          } // for
 8:
 9:          double povprecje = povprecjeOcen(ocene);
10:
11:          System.out.println("Tvoje povprecje je " +
12:                             povprecje + ".");
13:          System.out.println("Zaokrozeno povprecje je " +
14:                            (int)(povprecje + 0.5) + ".");
15:          System.out.println("Najnizja ocena je " +
16:                            minOcena( ocene));
17:          System.out.println("Najvisja ocena je " +
18:                            maxOcena( ocene));
19:      } // main
20:
21:      public static int minOcena(int[] ocene) {
22:          int najnizja = ocene[0];
23:
24:          for(int i = 1; i < ocene.length; i++) {
25:              najnizja = Math.min(najnizja, ocene[i]);
26:          } // for
27:          return najnizja;
28:      } // minOcena
29:
30:      public static int maxOcena(int[] ocene) {
31:          int najvisja = ocene[0];
32:
33:          for(int i = 1; i < ocene.length; i++) {
34:              najvisja = Math.max(najvisja, ocene[i]);
35:          } // for
36:          return najvisja;
37:      } // maxOcena
38:
39:      public static int vsotaOcen(int[] ocene) {
40:          int vsota = 0;
41:
42:          for(int i = 0; i < ocene.length; i++) {
43:              vsota = vsota + ocene[i];
44:          } // for
45:          return vsota;
46:      } // vsotaOcen
47:
48:      public static double povprecjeOcen(int[] ocene) {
49:          return (double)vsotaOcen(ocene) / ocene.length;
50:      } // povprecjeOcen
51:  } // PovprecjeOcen
```

```
C:\>java PovprecjeOcen 2 3 4 2 3 3
```

```
Tvoje povprecje je 2.8333333333333335.
Zaokrozeno povprecje je 3.
Najnizja ocena je 2
Najvisja ocena je 4
```

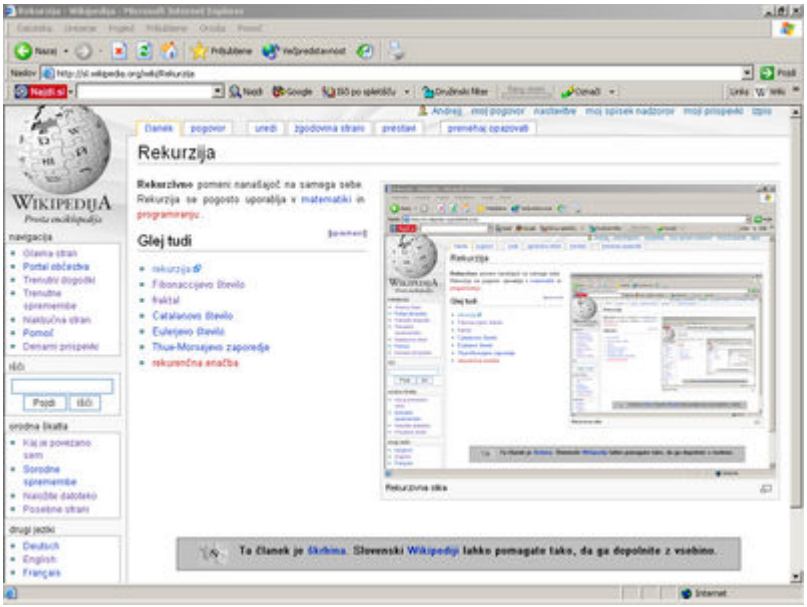
Program je sestavljen iz petih metod. Začnimo pri metodi *minOcena*, prvo oceno v naši tabeli označimo za najnižjo, postopoma se sprehodimo čez preostal del tabele in na vsakem koraku v

spremenljivko `najnizja` shranimo trenutno najnižjo vrednost . Za primerjavo uporabimo metodo `min()` iz razreda `Math`, ki nam vrne manjše število od obeh, ker je metoda tipa int nam vrne vrednost in to je mimimalna ocena. Sledi ji metoda `maxOcena`, ki je podobna prejšnji metodi le da ta vrne maksimalno oceno. Naslednja metoda je `vsotaOcen`, ki vrne vsoto vseh ocen. S pomočjo zanke `for` se sprehodimo čez tabelo in seštevamo vrednosti. Metoda `povprecjeOcen` pa vrne povprečje, ki ga dobimo z preprosto enačbo vsoto ocen deljeno z dolžino tabele ocene.
Še zadnja je metoda `main` v kateri preberemo parametre iz ukazne vrstice, jih zložimo v tabelo in s pomočjo klicev metod, ki smo jih ustvarili znotraj tega programa izpišemo statistiko naših ocen.

7.2 Rekurzivne metode

V Wikipediji, prosti enciklopediji, je beseda *rekurzivno* definirana takole:

Rekurzívno (recurrere v latinščini pomeni teči nazaj) pomeni nanašajoč na samega sebe. V matematiki označuje zaporedje, katerega n-ti člen je določen z enim ali več predhodnimi členi.



Rekurzivna slika, na kateri je rekurzivna slika, na kateri je rekurzivna slika, na kateri ...

Pri programskih jezikih se z rekurzijo srečamo takrat, ko napišemo metodo, ki v opisu postopka (v telesu metode) spet kliče samo sebe.
Bistvo rekurzije je, da problem razdelimo na več podproblemov enake narave oz. da rešitev problema podamo s samim istim problemom, le na manjši količini podatkov. Vsaka rekurzija je sestavljena iz rekurzivnega dela, kjer funkcija kliče samo sebe in ustavitvenega pogoja. Slednji določi, kdaj pri reševanju rekurzivnega problema ne bomo uporabili rekurzije oz. kdaj je problem tako majhen, da se delitev na manjše podrprobleme ne izplača več.

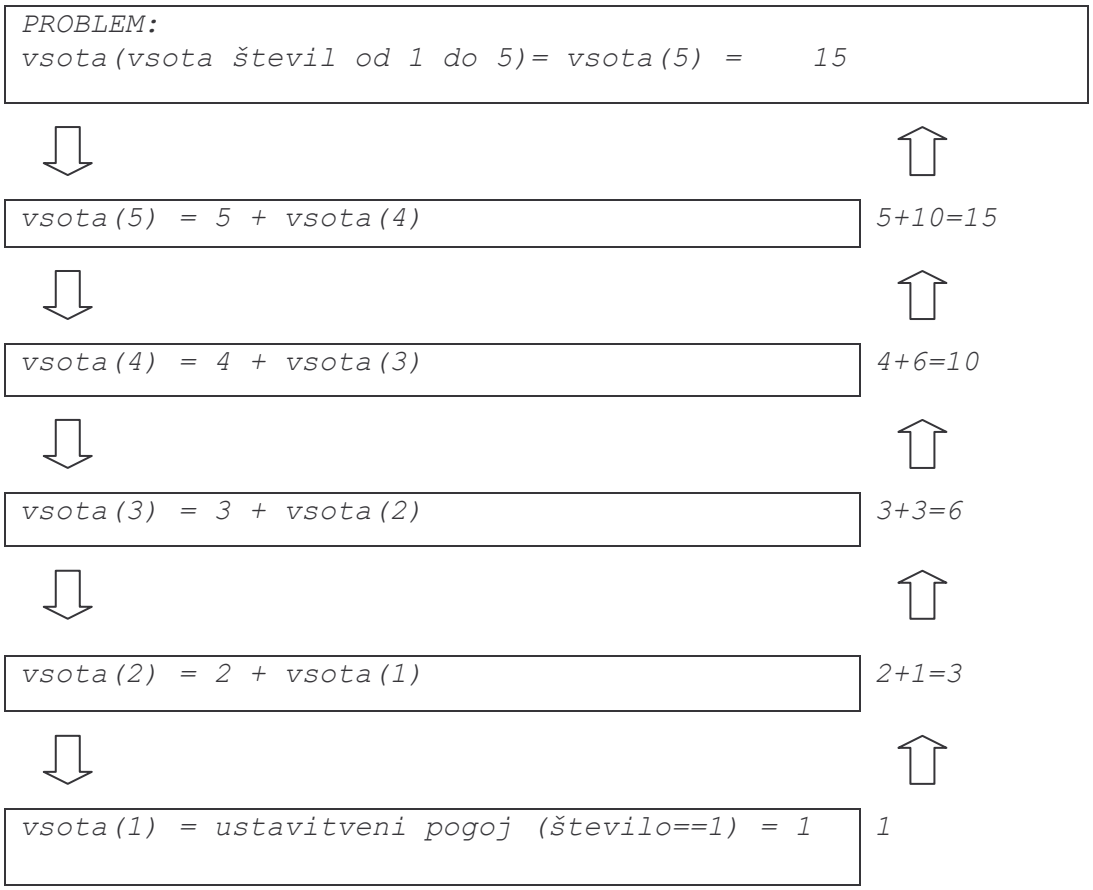
Rekurzivna metoda

```
rekurzivna_metoda (problem){
 if (problem majhen)
   return resitev;
else
 razdeli problem na enega ali več manjših podproblemov enake vrste
 za vse podprobleme
    rekurzivna_metoda (manjši problem);
 kombiniraj rešitev podproblemov v rešitev celotnea problema
}
```

Zgled

Vsota števil

Denimo, da želimo izračunati vsoto celih števil do n, kjer je n poljubno celo število. Recimo, da je n = 5. Potem moramo izračunati vsoto od 1 do 5. Sicer problem že znamo rešiti s pomočjo zanke, a tokrat se problema lotimo z rekurzijo. Vsoto števil do 5 namreč lahko izračunamo tudi tako, da k 5 prištejemo vsoto števil od 1 do 4. Slednje je spet problem iste vrste kot prvotni problem.



Problem `vsota(5)` smo razdelili na dva podproblema. Prvi sploh ni problem in vemo, da je njegova rešitev vrednost `5`. *Drugi je* `vsota (4)`. Ker ne vemo koliko je vsota števil od 1 do 4, ponovno uporabimo rekurzijo in problem razdelimo na dva podproblema, `4` in `vsota(3)`, … Postopek izvajamo, dokler ne naletimo na `ustavitveni pogoj`, ki poskrbi, da se ustavimo. Sedaj sledi postopno vračanje. `Ustavitveni pogoj` nam vrne vrednost `1`, tej vrednosti prištejemo `2,` … Ko se vrnemo nazaj k prvotnemu problemu, dobimo rešitev našega problema, ki je `15`.
Če naše razmišljanje zapišemo v obliki algoritma, vidimo, da smo rekli sledeče

$$\sum_{i=1}^{n} i = n + \sum_{i=1}^{n-1} i$$

Ali če uporabimo zapis v obliki metode

```
vsota(n) = n + vsota(n-1)
```

Obema zapisoma manjka še ustavitveni pogoj. Dodajmo še tega

$$\sum_{i=1}^{1} i = 1$$

oziroma

vsota(1) = 1

Zapišimo ustrezno metodo

```
1:     public static int vsota(int stevilo) {
2:       if(stevilo == 1) {
3:         return 1;
4:       } // if
5:       else {
6:         int rezultat = stevilo + vsota(stevilo - 1);
7:         return rezultat;
8:       } // else
9:     } // vsota
```

Vidimo, da v metodi sami spet kličemo to metodo (vrstica 6). Zaradi tega klica je metoda rekurzivna, saj je za opis postopka spet uporabljena metoda sama.

Zapišimo vse skupaj še v obliki programa. Tokrat našo metodo dopolnimo s klici izpisa, ki povedo, kako kličemo metodo in kakšen rezultat vrača.

PRIMER 7.2 – 1: *VsotaStevil.java*

```
 1:   public class VsotaStevil {
 2:     public static void main(String[] args) {
 3:       int stevilo = Integer.parseInt(args[0]);
 4:
 5:       vsota(stevilo);
 6:     } // main
 7:
 8:     public static int vsota(int stevilo) {
 9:       System.out.println("Racunam vsota(" + stevilo + ")");
10:       if(stevilo == 1) {
11:         System.out.println("vsota(" + stevilo + ") vrne " +
12:                            "rezultat 1");
13:         return 1;
14:       } // if
15:       else {
16:         int rezultat = stevilo + vsota(stevilo - 1);
17:         System.out.println("vsota(" + stevilo + ") vrne " +
18:                            "rezultat " + rezultat);
19:
20:         return rezultat;
21:       } // else
22:     } // vsota
23:   } // VsotaStevil2
```

Program prevedemo in poženemo:

```
Racunam vsota(5)
Racunam vsota(4)
```

```
Racunam vsota(3)
Racunam vsota(2)
Racunam vsota(1)
vsota(1) vrne rezultat 1
vsota(2) vrne rezultat 3
vsota(3) vrne rezultat 6
vsota(4) vrne rezultat 10
vsota(5) vrne rezultat 15
```

Opis programa.
Oglejmo si metodo *vsota*. Na začetku se nahaja ustavitveni pogoj (vrstice 10-14). Če ta ni izpolnjen, sledi rekurzivni klic (vrstica 16). Seveda je izpis znotraj programa namenjen le programerju za lažje razumevanje delovanja programa in v sam program ne spada (metodo bi zares napisali tako, kot smo navedli pšrej), nam pa potrdi, da je zgornja razlaga metode *vsota* prava.

V primeru, če kot parameter pri klicu uporabimo negativno število, se program zacikla. Premislimo zakaj! No, če le zaženemo naš program z *VsotaStevil - 5* , vidimo da začne z izpisovanjem

```
Racunam vsota(-5)
Racunam vsota(-6)
Racunam vsota(-7)
Racunam vsota(-8)
Racunam vsota(-9)
Racunam vsota(-10)
Racunam vsota(-11)
Racunam vsota(-12)
...
```

Popravimo program tako, da bo delal tudi za negativna števila. Dogovoriti se moramo, kaj pomeni vsota števil do negativnega števila n. Recimo, da je n -4. Želimo, da vsota(-4) pomeni – 4 + -3 + -2 + -1. Vemo sicer, da bi zadevo lahko enostavno rešili takole

vsota(-n) = - vsota(n)

A da se bomo bolj utrdili v spoznavanju rekurzije, bomo sestavili dve rekurzivni metodi. Prva, imenovana *pozitivnaVsota*, bo obstoječa metoda *vsota* z drugim imenom, druga, *negativnaVsota*, pa metoda, ki z rekurzijo izračuna vsoto negativnih števil, torej *vsota(-4) = -4 + vsota(-3)*.

PRIMER 7.2 – 3: *VsotaStevil3.java*

```
 1:   public class VsotaStevil3 {
 2:     public static void main(String[] args) {
 3:       int stevilo = Integer.parseInt(args[0]);
 4:       if(stevilo >= 0) {
 5:         pozitivnaVsota(stevilo);
 6:       } // if
 7:       else {
 8:         negativnaVsota(stevilo);
 9:       } // else
10:     } // main
11:
12:     public static int pozitivnaVsota(int stevilo) {
13:       System.out.println("Racunam vsota(" + stevilo + ")");
14:       if(stevilo == 0) {
```

```
15:           System.out.println("vsota(" + stevilo + ") vrne " +
16:                              "rezultat 1");
17:           return 0;
18:         } // if
19:         else {
20:           int rezultat = stevilo + pozitivnaVsota(stevilo - 1);
21:           System.out.println("vsota(" + stevilo + ") vrne " +
22:                              "rezultat " + rezultat);
23:
24:           return rezultat;
25:         } // else
26:       } // pozitivnaVsota
27:
28:       public static int negativnaVsota(int stevilo) {
29:         System.out.println("Racunam vsota(" + stevilo + ")");
30:         if(stevilo == -1) {
31:           System.out.println("vsota(" + stevilo + ") vrne " +
32:                              "rezultat -1");
33:           return -1;
34:         } // if
35:         else {
36:           int rezultat = stevilo + negativnaVsota(stevilo + 1);
37:           System.out.println("vsota(" + stevilo + ") vrne " +
38:                              "rezultat " + rezultat);
39:
40:           return rezultat;
41:         } // else
42:       } // negativnaVsota
43:     } // VsotaStevil3
```

Opis programa.
Program je sestavljen iz treh metod, metode *main* in dveh rekurzivnih metod *pozitivnaVsota* in *negativnaVsota*. V metodi *main* preverimo, če je dano število pozitivno in če je, kličemo metodo *pozitivnaVsota*. Če pa je število negativno, kličemo metodo *negativnaVsota*. Ker smo si metodo *pozitivnaVsota* že podrobneje pogledali v prejšnjem zgledu, se bomo osredotočili na metodo *negativnaVsota*. Kot vsaka rekurzivna metoda se tudi ta začne z ustavitvenim pogojem (ko pridemo do vrednosti *-1*, končamo z rekurzijo) in rekurzivnim delom. Recimo, da je prebrano število *-7*. Metoda *main* kliče metodo *negativnaVsota (-7)*. Ker ustavitveni pogoj ni izpolnjen, nadaljujemo z rekurzijo:

```
-7 + negativnaVsota(-6);
-7 + (-6) + negativnaVsota (-5)
-7 - 6 + (-5) + negativnaVsota (-4)
-7 - 6 - 5 + (- 4) + negativnaVsota (-3)
-7 - 6 - 5 - 4 + (- 3) + negativnaVsota (-2)
-7 - 6 - 5 - 4 - 3 + (- 2) + ustavitveni pogoj
-7 - 6 - 5 - 4 - 3 - 2 - 1
-7 - 6 - 5 - 4 - 3 - 3
-7 - 6 - 5 - 4 - 6
-7 - 6 - 5 - 10
-7 - 6 - 15
-7 - 21
-28.
```

11 Literatura

[1] Bratkovič Franc, *Uvod v C*

[2] Lokar Matija, *Prvi koraki v programski jezik C*

[3] Lokar Matija, Spletna izobraževalna televizija,
http://www.sitv.tv/vsebine.asp?drop=SiTV&fname=rekurzija#

[4] Mesojedec Uroš, Fabjan Borut, *Java2*

[5] Mesojedec Uroš, *Java programiranje za internet*

[6] Mrhar Peter, *Java 2*

[7] Prosta enciklopedija, *Wikipedija*, http://sl.wikipedia.org/wiki/Rekurzija

[8] Prtenjak Matjaž, *C++ za velike in male*

[9] Žumer Viljem, Brest Janez , *Uvod v programiranje in programski jezik C++*