DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

UNIVERZA V LJUBLJANI
FAKULTETA ZA MATEMATIKO IN FIZIKO

Matematika - praktična matematika (VSŠ)

Danica Petric

Spoznavanje osnov programskega jezika C#

Diplomska naloga

Ljubljana, 2008

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

Zahvala

Ob zaključku diplomske naloge se lepo zahvaljujem mag. Matiji Lokarju, ki me je vodil in nudil strokovno pomoč, ter za vso potrpežljivost in trud, ki mi ga je posvetil v času pripravljanja diplomske naloge.

Zahvaljujem se tudi vsem prijateljem za spodbudo, svetovanje in tehniško pomoč.

Posebno se bi zahvalila svoji družini za vso moralno podporo, spodbudo in razumevanje pri študiju in pisanju diplomske naloge.

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

Kazalo

1 Jezik C# 13
1.1 Zgodovina programskega jezika C# 13
1.2 Ogrodje.NET 15
2 Osnovni ukazi 17
2.1 Razvoj programske rešitve v okolju Microsoft Visual Studio .Net 17
2.2 "Okostje" programov 20
2.3 Prevajanje in zagon programa 21
2.4 Zapisovanje na zaslon 23
2.5 Spremenljivke in podatkovni tipi 24
2.5.1 Splošno o spremenljivkah v jezikih java in C# 24
2.5.2 Podatkovni tipi 26
2.5.3 Pretvarjanje med vgrajenimi podatkovnimi tipi 28
2.5.4 Branje iz konzole 31
2.6 Naloge za utrjevanje znanja iz osnovnih ukazov 32
3 Odločitve 34
3.1 Pogojni stavki 34
3.2 Stavek switch 36
3.2.1 Stavek switch v javi 36
3.2.2 Stavek switch v C# 43
3.3 Naloge za utrjevanje znanja iz odločitev 47
4 Zanke 49
4.1 While 49
4.2 For 52
4.3 Do 54
4.4 Naloge za utrjevanje znanja iz zank 57
5 Naključna števila 59
5.1 Naloge za utrjevanje znanja iz naključnih števil 61
6 Nizi 62
6.1 Naloge za utrjevanje znanja iz nizov 67
7 Tabele in zanka foreach 68
7.1 Tabele 68
7.2 Foreach 72
7.3 Naloge za utrjevanje znanja iz tabel 78
8 Metode 80
8.1 Naloge za utrjevanje znanja iz metod 83
9 Datoteke 85
9.1 Pisanje na tekstovno datoteko 86

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

9.2 Branje iz tekstovne datoteke *89*
9.3 Naloge za utrjevanje znanja iz datotek *93*
10 Zaključek **95**
11 Literatura in viri **96**

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

Program dela

V diplomski nalogi predstavite osnove programskega jezika C#. Pri tem izhajajte iz tega, da bralec pozna programski jezik java v obsegu, kot ga spozna študent Praktične matematike. Zato tudi uporabljajte izrazoslovje, obseg tematike in razlago na način, kot je bilo to pri spoznavanju jezika java.

Osnovna literatura:

- B. Bragnall, *C# for Java Programmers*

Mentor:

mag. Matija Lokar

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

Povzetek

V diplomski nalogi sem predstavila osnovne značilnosti programskega jezika C#. Pri tem sem predpostavila, da je bralec diplomskega dela absolvent praktične matematike, ki že pozna programski jezik java. Cilj diplomske naloge je torej absolventom praktične matematike na osnovi njihovega znanja jezika java predstaviti osnovne značilnosti jezika C#.

Diplomsko nalogo sem razdelila na devet poglavij. V uvodnem poglavju je navedena zgodovina jezika C# in ogrodja.NET. Vsebina od drugega do devetega poglavja je posvečena spoznavanju osnovnih značilnosti jezika C# in njegovi primerjavi s programskim jezikom java. Tako so v drugem poglavju predstavljeni osnovni ukazi, kot so osnovna zgradba programov, prevajanje in zagon programa, zapisovanje na zaslon ter spremenljivke in podatkovni tipi. Predstavljen pa je tudi razvoj programov v okolju Microsoft Visual C# Express Edition. V tretjem poglavju sta predstavljena pogojni stavek in stavek switch. Pri tem stavek switch natančneje spoznamo tudi v jeziku java. Temu poglavju sledi poglavje o zankah, v katerem obnovimo znanje o zanki while in zanki for ter spoznamo zanko do. Nato sledi še poglavje o naključnih številih ter poglavje o nizih.
Sedmo poglavje obravnava tabele in zanko foreach, ki jo natančneje spoznamo tudi za jezik java. Temu poglavju sledi poglavje o metodah. V zadnjem poglavju pa so obravnavane datoteke, s poudarkom na tekstovnih datotekah.

Math. Subj. Class. (2000): 68N15, 68Q32
Ključne besede: C#, C sharp, java, programski jezik, primerjava jave in C#,.NET
Keywords: C#, C sharp, java, programming language, comparison of Java and C#, .NET

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

1 Jezik C#

1.1 Zgodovina programskega jezika C#

Od izuma računalnika dalje se je pojavilo že veliko različnih programskih jezikov. Na koncu prejšnjega tisočletja so, če upoštevamo število programerjev v določenem jeziku, prevladovali jeziki C, C++, java in BASIC.

Razmah Interneta je med najpomembnejše programske jezike postavil programski jezik java, ki omogoča izdelavo programov, ki delujejo preko spletnih brskalnikov. To pomeni, da je java zasnovana tako, da koda napisana v tem jeziku, deluje na različnih računalnikih in operacijskih sistemih. Javo so razvili pri podjetju SUN, ki je njen nadaljnji razvoj (in ga načeloma še) povsem nadzoroval.

Pri Microsoftu so zaradi določenih težav, ki so jih imeli pri uveljavljanju svojih idej glede jezika java, odločili, da bodo razvili lasten programski jezik, namenjen spletnemu programiranju. Prvi koraki k novemu jeziku so temeljili na razširitvi jezika C++. Toda nov jezik, imenovan Cool, se ni uveljavil niti med novimi programerji, niti med tistimi, ki so do takrat delali bodisi s programskim jezikom C bodisi z jezikom C++. Zato so leta 2000 pri Microsoft jezik Cool opustili in se lotili razvoja povsem novega jezika, ki so ga poimenovali C#.

Avtorja novega jezika Scott Wilamuth in Andrers Hejilsberg[1] sta ideje za razvoj C# največ črpala iz že poznanih jezikov, kot so java, C++ in Visual Basic. Tako nekateri strokovnjaki menijo, da se C# ujema:

- z javo v 70%,
- s C++ v 10%,
- z Visual Basic-om 5%.

Ostalih nekaj odstotkov programa C# (15%) pripada novostim, ki sta jih avtorja uvedla. Tako visok odstotek ujemanja C# z javo ni presenetljiv. Izhaja iz tega, da je bila java v času pred nastankom C# že močno uveljavljena na svetovnem trgu. Poleg tega je za javo značilna univerzalnost, varnost, enostavnost, dinamičnost, itn. To so same lastnosti, ki si jih programerji želijo od programskega jezika. Kako sta si jezika podobna, kaže že spodnji zgled, ki na zaslon izpiše besedico Zdravo.

C#:
```
public class Pozdrav{
   public static void Main(string[] args){

     System.Console.WriteLine("Zdravo.");

   }
}
```

Java:
```
public class Pozdrav{
   public static void main(String[] args){

     System.out.println("Zdravo.");

   }
}
```

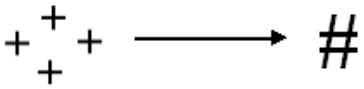
Tabela 1: Program Pozdrav zapisan v C# in Javi.

Okoli izbire imena jezika C# se širijo razne legende oziroma razlage, ki pojasnjujejo, zakaj so si pri Microsoftu za ime izbrali znaka C (velika tiskana črka) in # (številčni znak). Najpogostejši razlagi sta:

1. Microsoft se je za oznako C# odloči iz zgodovinskih razlogov. Najprej je obstajal jezik C. Za njim se je razvil C++. Ta je ime dobil tako, da so znaku C dodali še dva znaka + (C++), kar v samem jeziku pomeni "za eno povečan C". Ob razvoju novega jezika, ki izhaja iz C++, bi bilo nesmiselno in grdo dodati še nova dva znaka +, saj bi dobili C++++. Zato so se pri Microsoftu odločili, da zapis ++++, ki je predolg, zapišejo z enim

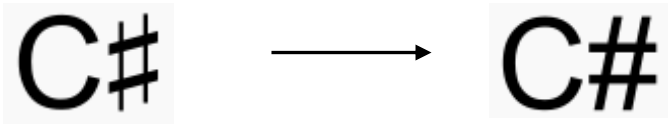
[1] Andres Hejilsberg je tudi avtor jezikov Delphi in Visual J++.

znakom. V ta namen je na tipkovnici najbolj primeren znak # (ali, v žargonu, lojtra). Izbira ja logična, saj s postavitvijo štirih znakov + lahko dobimo znak #.



2. Microsoft je želel izhajati iz imena C in povedati, da gre za "boljši (višji) C". V glasbi znak ♯ označuje povišano noto. Toda tu je nastopila težava, saj simbola ♯ na splošni tipkovnici ni. Zato so ga nadomestili z znakom #.

Glasbeni zapis: **Ime jezika:**



Microsoft je v začetku, ko se je C# šele začel dobro uvajati v javnost, trdil, da C# ni tekmec jeziku javi in da bodo ta jezik (torej javo) še naprej podpirali v svojih operacijskih sistemih in brskalnikih. Kasneje se je pokazalo, da se tega niso držali. Poleti leta 2001 je podjetje Microsoft javnosti sporočilo, da operacijski sistem Windows XP ne bo vsebovali lastnega javanskega navideznega stroja (JVM - Java Virtual Machine), ki operacijskemu sistemu omogoča poganjanje programov, napisanih v javi. Kritiki so takrat podjetje Microsoft obtožili, da z ukinjanjem jave želi prisiliti programerje k uporabi njihovega programskega jezika C#.

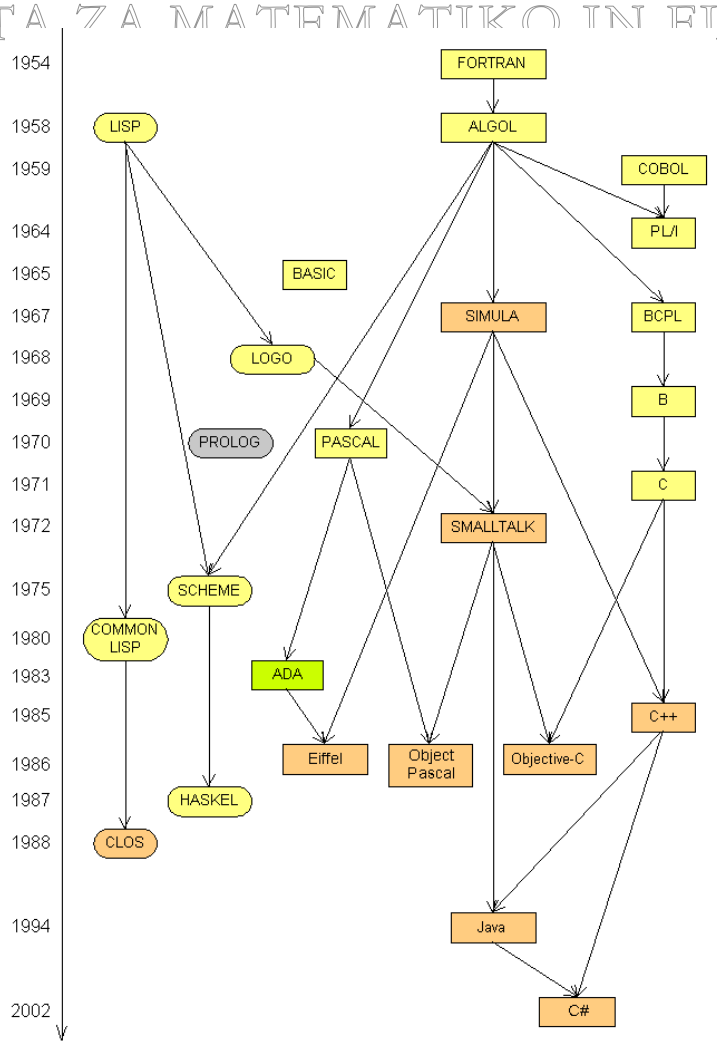
Leta 2002 je Microsoft z namenom, da bi jezik C# obstal na tržišču in bi si hkrati pridobil programerje, ki programirajo v javi, objavil brezplačno orodje, ki je omogočal pretvorbo izvorne kode iz jave v jezik C#, skladno s platformo .NET. Izdelek se je imenoval Java Language Conversion Assistant (JLCA). Kasneje je ta izdelek dobil še izboljšavi v obliki različic 2.0 in 3.0.

Leta 2005 je Microsoft objavil novo različico jezika C# z oznako 2.0, ki je bila izboljšava prve različice jezika C# z oznako 1.0. Različico C# 2.0 podpira platforma Microsoft .NET Framework 2.0, ki je bila izboljšava platforme Microsoft .NET Framework 1.1. Slednja podpira C# 1.0. Novosti, ki jih je predstavljal .NET 2.0 za različico C# 2.0, so bile anonimni tipi, iteratorji, generiki in delni razredi.

V začetku letošnjega leta (2008) je Microsoft objavil zadnjo različico jezika C#, ki nosi oznako 3.0. Novosti ni veliko, večina je potrebna za izvedbo novosti v platformi Microsoft .NET Framework 3.5. Te novosti so anonimni podatkovni tipi, delne metode, razširljive metode, izrazi lambda, inicializacija objektov in zbirk ter implicitni tip krajevne spremenljivke.

Microsoftovi arhitekti so, po objavi C# 3.0, že pričeli z razvojem naslednje različice, ki bo imela ob objavi oznako 4.0. Kdaj bo le ta dostopna za uporabo in katere novosti bo vsebovala, pri Microsoftu še molčijo.

Če bi morali programski jezik C# opisati z nekaj besedami, bi rekli da je preprost, varen, objektno orientiran, namenjen za pisanje programov v okolju .NET prilagojenih internetu in primeren za razvoj najzahtevnejše programske opreme. Poleg tega je zasnovan na tak način, da ga je mogoče preprosto izboljševati in razširjati, brez nevarnosti, da bi s tem izgubili združljivost z obstoječimi programi.



Slika 1: Razvoj programskih jezikov.

1.2 Ogrodje.NET

V prejšnjem razdelku smo omenjali ogrodje .NET, toda o njem nismo povedali ničesar. To nadoknadimo v tem poglavju, v katerem bomo povedali nekaj splošnih podatkov o tem ogrodju.

Podjetje Microsoft razvija in trži različne izdelke in tehnologije. Njihova skupna točka je odvisnost od ogrodja .NET. To ogrodje je sestavni del operacijskega sistema Windows, oziroma ga lahko temu operacijskemu sistemu dodamo. Ogrodje je sestavljeno iz velikega števila rešitev za različna programska področja kot so gradnja uporabniških vmesnikov, dostopanje do podatkov, kriptografija, razvoj mrežnih aplikacij, numerični algoritmi, omrežne komunikacije … Programerji pri razvoju programov metode, ki so uporabljene v teh rešitvah, kombinirajo z lastno kodo.
Ogrodje .NET določa arhitekturo razvoja programskih rešitev. Temelji na konceptih objektne tehnologije in komponentnega razvoja. Razvijalcem ponuja razvojno ogrodje, na osnovi katerega gradijo nove aplikacije.

Ogrodje .NET določajo naslednje osnovne komponente:

- izvajalnik kode skupnega jezika (CLR - *Common Language Runtime*),
- knjižnice razredov in
- storitve.

Programi, ki so napisani za ogrodje .NET, se izvršujejo v posebnem okolju. To upravlja z zahtevami programa med izvajanjem. Okolje je sestavni del ogrodja .NET in se imenuje izvajalnik kode skupnega jezika (*Common Language Runtime* - CLR).
Okolje CLR predstavlja navidezni računalnik, ki poskrbi, da se programerju ni potrebno ukvarjati z lastnostmi določenega procesorja in z ostalimi strojnimi deli računalnika. Prav tako poskrbi, da kode programov (izvorna koda) ni potrebno prevajati neposredno v strojno kodo, prilagojeno strojni opremi, temveč se prevede v tako imenovano vmesno kodo. Vmesna koda prevede v strojno kodo, prilagojeno strojni opremi, na kateri se izvaja, šele ob zagonu programa, ko se uporabi prevajalnik vmesne kode JIT (*Just in Time*). V ogrodju .NET se v vmesno kodo prevajajo vsi programski jeziki, prirejeni za okolje .NET. To pomeni, da lahko razvijalec pri pisanju kode uporablja kateregakoli izmed teh programskih jezikov (npr. C#, J#, Visual BASIC for .NET), saj so z vpeljavo vmesnega jezika zbrisane vse meje med temi jeziki.
Ogrodje .NET omogoča tudi sočasno uporabo več programskih jezikov v enem programu. To pomeni, da lahko razvijalec pri pisanju kode uporablja poljuben jezik. V programu napisanem v enem jeziku, lahko kličemo metodo, ki je napisana v drugem programskem jeziku.

V ogrodje .NET so vključeni številni razredi, ki olajšajo delo z grafiko, razredi za delo z bazo podatkov, razredi za delo s podatkovnimi strukturami, razredi, ki poenostavljajo ustvarjanje dokumentov v jeziku XML, razredi za delo z datotekami in podatkovnimi toki ter še mnogi drugi.
Razredi v ogrodju sestavljajo knjižnice, ki so organizirane po imenskih prostorih (*namespace*). V posamezen imenski prostor združujemo razrede, ki omogočajo izvajanje določenih funkcij (npr. grafično prikazovanje, branje in pisanje datotek). Imenske prostore zaradi večje preglednosti organiziramo smiselno in hierarhično. V programe jih vključujemo, saj omogočajo večjo preglednost in enostavnejše pisanje programske kode.

Primeri imenskih prostorov:

- `System.IO` - imenski prostor, namenjen za delo z datotekami, podatkovnimi tokovi in imeniki
- `System.Text` - imenski prostor, namenjen za formatiranje, obdelavo in odkodiranje teksta
- `System.Data` - imenski prostor, namenjen delu s strežnikom SQL in bazami podatkov
- `System.Windows.Forms` - imenski prostor, namenjen delu z namiznimi aplikacijami

Opis uporabe imenskih prostorov, npr. nekaj v stilu hočemo uporabiti metodo `WriteLine()`, ki se nahaja v razredu `Console`, torej v imenskem prostoru `System`. Če ne napovemo uporabe imenskega prostora, bi morali metodo klicati s `System.Console.WriteLine()`. Če pa uporabo imenskega prostora napovemo (s stavkom `using` - glej razdelek 2.2), jo lahko navajamo (kličemo) le s `Console.WriteLine()`.

2 Osnovni ukazi

2.1 Razvoj programske rešitve v okolju Microsoft Visual Studio .Net

Za pisanje programov v jeziku C# (in tudi ostalih jezikih v okolju .NET) je Microsoft razvil razvojno okolje, ki se imenuje Microsoft Visual Studio.NET. Združuje zmogljiv urejevalnik kode, prevajalnik, razhroščevalnik, orodja za dokumentacijo programov in druga orodja, ki pomagajo pri pisanju programskih aplikacij. Poleg tega okolje nudi tudi podporo različnim programskim jezikom, kot so na primer C#, C++ in Visual Basic.
Microsoft Visual Studio.Net obstaja v več različicah. Za spoznavanje z jezikom C# zadošča brezplačna različica Visual C# Express Edition. Ta podpira le razvoj programov v jeziku C#. Prenesemo jo lahko z Microsoftovih spletnih strani. Ker se točen naslov, na katerem to različico dobimo, občasno spreminja, naj naslov bralec poišče kar sam. Če v poljubni iskalnik vpišemo ključne besede "Visual Studio Express download", bomo različico zagotovo našli. V diplomski nalogi bomo opisovali v trenutku pisanja naloge najnovejšo različico tega okolja, Visual C# 2008 Express Edition. Vendar praktično vse povedano velja tudi za starejše različice tega okolja. Seveda obstajajo tudi druga razvojna okolja za pisanje programov v C#, na primer okolje SharpDevelop (http://www.icsharpcode.net/OpenSource/SD/).

Slika 2: Logotip okolja Visual Studio .NET.

Programske rešitve običajno ne sestavlja le ena datoteka z izvorno kodo, ampak je datotek več. Skupek datotek in nastavitev, ki rešujejo določen problem, imenujemo **projekt**.
Glede na to, za kakšno vrsto programske rešitve gre, imamo v okolju Visual C# Express vnaprej pripravljenih več različnih tipov projektov:

- **Console Application** (ali konzolne aplikacije) – namenjene gradnji aplikacij, ki ne potrebujejo grafičnega vmesnika. Izvajajo se preko ukaznega okna ali kot mu tudi rečemo, konzole (t.i. »DOS-ovskih« aplikacij).
- **Windows Forms Application** (ali namizne aplikacije) – namenjene za gradnjo namiznih aplikacij s podporo grafičnih gradnikov.
- **Windows Presentation Foundation (WPF) Application** – namenjen za gradnjo programov, ki tečejo v okolju Windows, zasnovanih na uporabi najnovejših gradnikov okolja WPF.
- **Windows Presentation Foundation (WPF) Browser Application** – namenjen za programiranje programov, ki tečejo v spletnih brskalnikih (npr. Internet Explorer, Firefox).
- **Class Library** (ali knjižnice) – namenjene gradnji knjižnic razredov.
- **Empty Project** (ali prazen projekt) – namenjen gradnji aplikacij brez vnaprej določenega vzorca.

Izbira projekta določa, kakšno bo vnaprej pripravljeno ogrodje programov, katere knjižnice se bodo naložile in podobno.
Za spoznavanje osnov programskega jezika C# bomo uporabljali več ali manj tip projekta Console Application. Opišimo, kako tak projekt ustvarimo.

Postopek izdelave novega projekta:

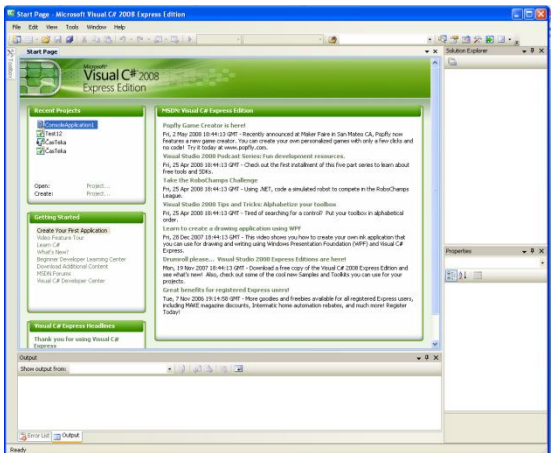
1. korak:

Najprej na znan način poženemo okolje MVCEE (Microsoft Visual C# 2008 Express Edition) (torej tako, kot poganjamo vse programe v okolju Windows). Kliknemo na gumb **Start**, se

premaknemo na izbiro **Vsi programi** oziroma **Programs** in poiščemo ikono **Microsoft Visual C# 2008 Express Edition.** Ko ikono **Microsoft Visual C# 2008 Express Edition** najdemo, jo kliknemo.

Slika 3: Zagon Microsoft Visual C# 2008 Express Edition.

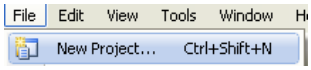
Po zagonu se odpre začetna stan okolja MVCEE. V zgornjem levem delu je okno, v katerem je seznam projektov, s katerimi smo nazadnje delali. Preostala okna v tem okolju nam služijo za pomoč pri delu in za informiranje o novostih, povezanih z okoljem MVCEE.



Slika 4: Začetna stran okolja Microsoft Visual C# 2008 Express Editon.

2. korak:

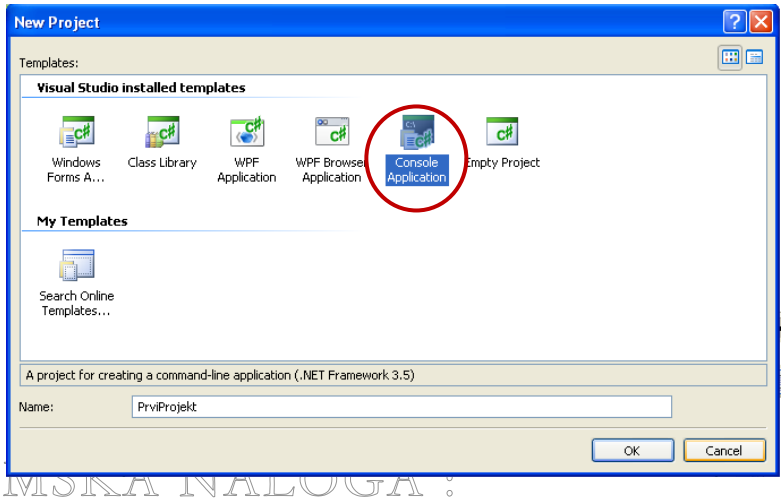
Naredimo nov projekt. To naredimo tako, da se postavimo na meni **File** in kliknemo na ikono **New Project**.



Slika 5: Zagon novega projekta.

3. korak

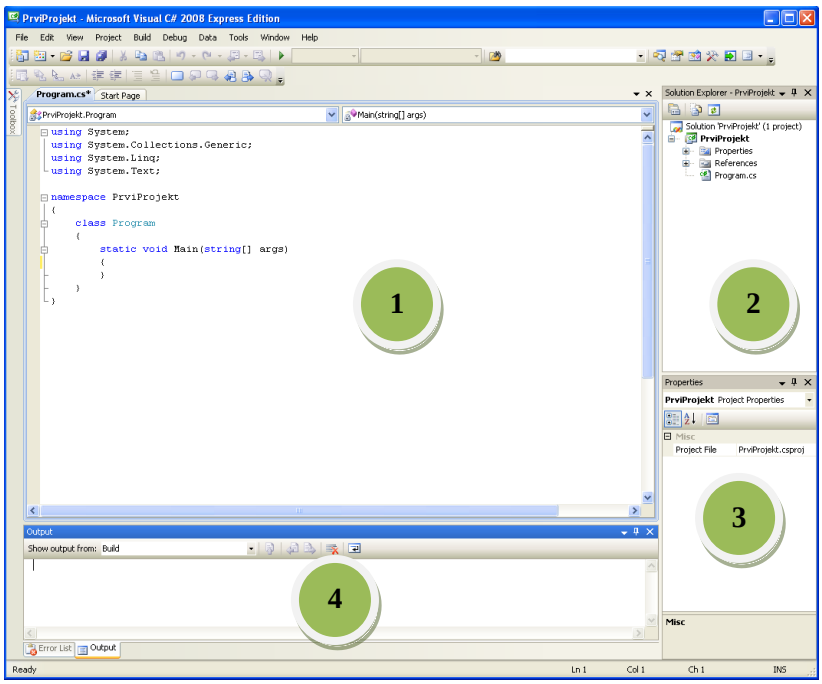
Po izvedbi drugega koraka se odpre okno *New Project.* V tem oknu med **Templates** (vzorci) izberemo tip projekta **Console Application** ter določimo ime (**Name**) projekta.



Slika 6: Okno za izbiro vrste projektov.

4. korak

Ko vnesemo ime projekta, kliknemo na gumb **OK**. Odpre se končno okno, ki je namenjeno razvoju aplikacije.



Slika 7: Osnovni deli okna za urejanje aplikacije.

Kratek opis osnovnih delov razvojnega okolja:

1) **Editor** ali **urejevalnik** je okno, namenjeno pisanju in urejanju programske kode. Pri pisanju daljših delov programov je zelo priročna načrtovalna črta (**outlining**). Črto sestavljajo + in – na levi strani urejevalnika. Kliki na te oznake omogočajo skrivanje in prikazovanje delov kode. Pri tem so sestavni deli logični deli kode, kot so posamezne metode, razredi in podobno. V primeru, da je del kode skrit (+ na levi), je naveden le njegov povzetek (začetek, ki mu sledijo …). Pregled vsebine lahko vidimo tako, da se z miško premaknemo na ta povzetek. Če na + kliknemo, se koda prikaže. Obratno s klikom na – kodo skrijemo.

2) **Soution Explorer** ali **raziskovalec rešitve** je okno, ki prikazuje in omogoča dostop do vseh datotek znotraj projekta. Datoteke lahko dodajamo, jih brišemo ali spreminjamo.

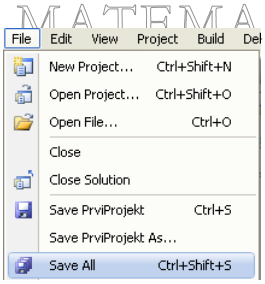
3) **Properties** ali **lastnosti objektov** je okno, v katerem lahko spreminjamo in nastavljamo lastnosti objektov. Pri izdelavi konzolne aplikacije je to okno prazno in ga ne potrebujemo.

4) **Output** ali **izpis** je okno, kjer dobimo sporočila o morebitnih napakah ali opozorilih, ki so se pojavili med preverjanjem kode.

Kot vidimo (del 1 na sliki 7), nam samo okolje v skladu z izbranim tipom projekta že zgradi osnovno "okostje" programa. V to "okostje" lahko začnemo zapisovati kodo programa.

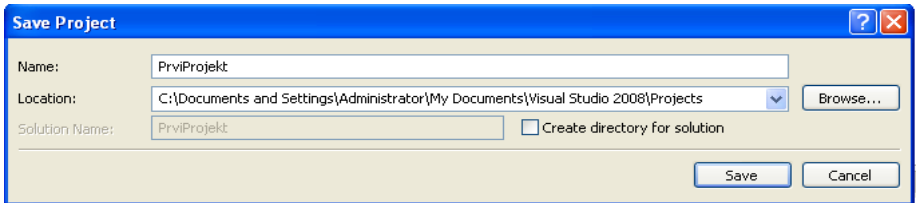
5. korak

Priporočljivo je, da nov projekt takoj shranimo. To storimo tako, da se postavimo na meni **File** in kliknemo izbiro **Save All**.



Slika 8: Shranitev novega projekta.

Prikaže se okno **Save Project**. V tem oknu določimo imenik (**Location)**, kjer bodo datoteke novega projekta. Poleg imenika lahko v tem oknu določimo tudi novo ime projekta. To naredimo tako, da v okence **Name** vnesemo novo ime.



Slika 9: Okno za nastavitev shranjevanja.

Po nastavitvi imena (*Name*) in imenika (*Location*) kliknemo na gumb **Save**. Projekt se shrani in okno *Save Project* se zapre.

Po zaprtju okna *Save Project* se ponovno nahajamo v osnovnem oknu.

Sedaj v pripravljeno okostje programa dodamo našo kodo. Nato ponovimo peti korak, pri čemer seveda ponujenih nastavitev (ime, imenik …) ni smiselno spreminjati. Nato moramo kodo programa samo še prevesti in zagnati.

2.2 "Okostje" programov

Ko v okolju Microsoft Visual C# 2008 Express Edition ustvarimo nov projekt, se v oknu Editor prikaže osnovna zgradba programa v jeziku C# (Slika 10). Ob pogledu na to "okostje" opazimo, da se ne razlikuje veliko od osnovnega okostja v javi. Razlike so v uvozu želenih razredov, v imenskem prostoru in malenkostna sprememba v zapisu deklaracije "glavne" metode.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace PrviProjekt
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

Slika 10: Okostje programa v jeziku C#.

Na začetku okostja programa so navedeni stavki `using`. Stavek `using` v jeziku C# uporabljamo za vključitev imenskega prostora v naš program. Ta vključitev poenostavi pisanje kode, koda je krajša in lažje razumljiva. Če na primer želimo v programski kodi uporabiti metodo `X` iz imenskega prostora `System.IO`, katerega smo v program vključili z `using System.IO;`, lahko za klic metode uporabimo le `X()`. Če pa imenskega prostora `System.IO` ne vključimo v naš

program, potem moramo metodo klicati s `System.IO.X()`. V javi imenski prostor vključujemo s stavkom `import`, ki ima enako vlogo kot stavek `using` v jeziku C#.

Primer vpeljave:

Java	***C#***
import java.io.*;	using System.io;

Stavkom za vključitev imenskih prostorov v program sledi stavek za sestavo novega imenskega prostora. Ta stavek se prične z besedo `namespace`. Pri spoznavanju jezika java imenskih prostorov nismo omenjali, čeprav jih java, tako kot večina današnjih programskih jezikov, podpira. V javi je imenski prostor predstavljen kot paket, ki ga ustvarjamo z besedo `package`. Programi, ki jih bomo zapisali tekom diplomske naloge, imenskih prostorov ne bodo vsebovali. Torej bomo v okostju programov jezika C# ukaz `namespace` izpustili (Slika 11). Prav tako bomo v javi izpustili ukaz `package`.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

class Program
{
    static void Main(string[] args)
    {
    }
}
```

Slika 11: Okostje programa brez navedbe imenskega prostora (namespace).

Deklaracija "glavne" metode ima v jeziku java obliko

```
static void main(String[] args)
```

V jeziku C# je ta oblika nekoliko drugačna

```
static void Main(string[] args)
```

Metoda `Main` je v vsaki konzolni aplikaciji v jeziku C# obvezna. V primeru, da metode v konzolni aplikaciji ne navedemo, prevajalnik izpiše napako (Slika 12).

```
Program 'C:\Documents and Settings\Administrator\Desktop\VajeC#\Test12\Test12\obj\
Release\Test12.exe' does not contain a static 'Main' method suitable for an entry
point
```

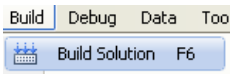
Slika 12: Izpis napake, če v konzolni aplikaciji izpustimo metodo Main.

Prevajalnik jezika java se ob izpustitvi metode `main` ne pritoži. A če jo ne navedemo, programa ne bomo mogli zagnati.
Ko smo se v javi odločali o imenu razreda (`class`), smo se hkrati odločali tudi o imenu datoteke, v kateri je napisani program shranjen, saj se morata ujemati. V jeziku C# med imenoma razreda in datoteke ni povezave.
Ko shranimo projekt, se izvorna koda programa v C# shrani na datoteko `Program.cs`. Če nam ime datoteke ne ustreza, se v oknu *Solution Explorer* (Slika 7, okno 2) postavimo na ikono *Program.cs*. Kliknemo na desno tipko miške in izberemo opcijo *Rename*. Spremenimo ime, pri tem pa ohranimo končnico `cs`. S ponovnim klikom miške se ime datoteke spremeni.

2.3 Prevajanje in zagon programa

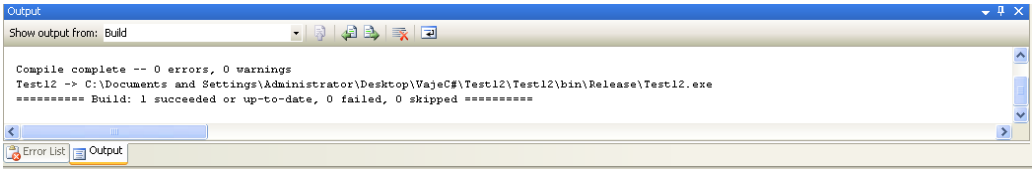
Pred zagonom programa moramo program prevesti. Poglejmo, kako to naredimo v okolju Microsoft Visual C# 2008 Express Edition.

Postopek prevajanja programov v C#
Program po zapisu prevedemo. To storimo tako, da se postavimo na meni **Build** in izberemo **Build**.



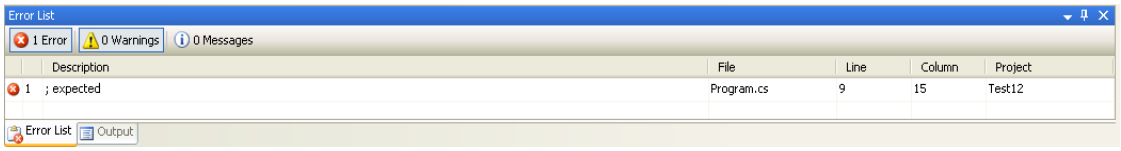
Slika 13: Prevajanje programa.

Po končanem prevajanju se v oknu **Output** izpiše povzetek.



Slika 14: Prikaz zapisanega povzetka.

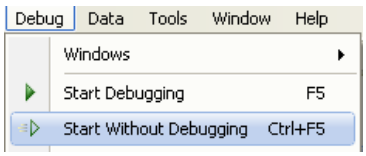
V povzetku je zapisan podatek o številu napak (*errors*). Če je število napak 0, je program sintaktično pravilen. V primeru, da so v programu napake, se prevajanje prekine in prevajalnik nam sporoči opis sintaktične nepravilnosti in številko vrstice, kjer se napaka nahaja.



Slika 15: Prikaz napake ob prevajanju.

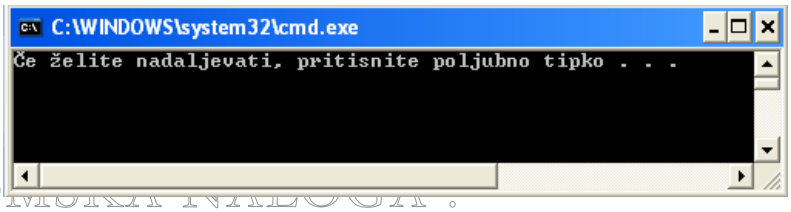
Sporočilo o napaki je vidno v oknih **Error List** in **Output**.

Postopek zagona programa v C#
Program, ki je preveden brez sintaktičnih napak, lahko zaženemo. To storimo tako, da se postavimo na meni **Debug** in izberemo opcijo **Start Without Debugging**.



Slika 16: Zagon programa.

S tem zaženemo program. Prikaže se konzolno okno, v katerem se prikazujejo rezultati napisanega programa. Ko se program izvede do konca, sistem izpiše še obvestilo "Če želite nadaljevati, pritisnite poljubno tipko ...". To obvestilo se izpiše v konzolnem oknu, ki se odpre ob zagonu programa. V preostalih razvojnih okoljih je izpis obvestila nekoliko drugačen, ali pa ga sploh ni.



Slika 17: Konzolno okno v katerem je prikazano obvestilo.

Konzolno okno zapremo s pritiskom poljubne tipke na tipkovnici.

2.4 Zapisovanje na zaslon

Za zapisovanje na zaslon smo v javi uporabljali metodi `println()` in `print()`. Obe metodi izvajamo nad objektom `out` iz razreda `System`. V jeziku C# za zapisovanje uporabljamo metodo `WriteLine()`, ki je ekvivalentna metodi `println()`. Poleg metode `WriteLine()` poznamo še metodo `Write()`, ki ustreza javanski metodi `print()`. Obe metodi jezika C# pripadata razredu `Console`.

Zapisovanje na zaslon se v jeziku C# razen po uporabi drugačnih imen metod praktično ne razlikuje od zapisovanja v javi. Tudi v C# v nizih lahko uporabljamo posebne znake, ki jih napovemo z znakom '\'. Da osvežimo svoj spomin (rekli smo, da predpostavljamo da programirati v javi znamo), si poglejmo naslednji zgled. Program bomo napisali kar v jeziku C#.

Simbol

Sestavimo program, ki bo na zaslon izpisal naslednji simbol

```
 xxxxxx
x  /\  x
x /  \ x
x/ xx \x
x\ xx /x
x \  / x
x  \/  x
 xxxxxx
```

Ker v konzoli lahko izpisujemo le od leve proti desni in od zgoraj navzdol, bomo simbol izpisali po vrsticah. Uporabili bomo metodo `WriteLine()`. Pomagali si bomo tudi s kombinacijo `\\`, s pomočjo katere bomo na zaslon izpisali znak '\'.

Zapis v C#:

```
C1:   using System;
C2:   public class Simbol{
C3:      public static void Main(string[] args){
C4:         Console.WriteLine(" xxxxxx");
C5:         Console.WriteLine("x  /\\  x");
C6:         Console.WriteLine("x /  \\ x");
C7:         Console.WriteLine("x/ xx \\x");
C8:         Console.WriteLine("x\\ xx /x");
C9:         Console.WriteLine("x \\  / x");
C10:        Console.WriteLine("x  \\/  x");
C11:        Console.WriteLine(" xxxxxx");
C12:     }
C13:  }
```

V kodi jezika C# lahko znak `'\'` zapišemo tudi kot običajen znak, če pred nizom uporabimo znak `'@'`. S tem znakom povemo, da se morajo vsi znaki v nizu jemati dobesedno. Zgornji program bi torej lahko napisali tudi na naslednji način.

Zapis v C#:

```
C1:   using System;
C2:   public class Simbol{
C3:      public static void Main(string[] args){
C4:         Console.WriteLine(" xxxxxx");
C5:         Console.WriteLine(@"x  /\  x");
C6:         Console.WriteLine(@"x /  \ x");
C7:         Console.WriteLine(@"x/ xx \x");
C8:         Console.WriteLine(@"x\ xx /x");
C9:         Console.WriteLine(@"x \  / x");
```

```
C10:         Console.WriteLine(@"x  \/  x");
C11:         Console.WriteLine(" xxxxxx");
C12:     }
C13: }
```

Razlaga. Ker smo v vrsticah C5 - C10 pred nizom uporabili znak '@', smo s tem povedali, da znak '\' ni posebni znak, ampak so vsi znaki v nizu "navadni". Seveda bi znak '@' lahko uporabili tudi pri izpisu v C4 in C11.

2.5 Spremenljivke in podatkovni tipi

2.5.1 Splošno o spremenljivkah v jezikih java in C#

Kot vemo, je spremenljivka v programskem jeziku mesto, kjer hranimo podatke, katerih vrednost se med izvajanjem programa lahko spreminja. V programskih jezikih java in C# spremenljivkam, ki dejansko pomenijo pomnilniške lokacije, dodelimo ime in podatkovni tip.

Ime spremenljivke

Vsem spremenljivkam moramo določiti ime, da se lahko kasneje nanj sklicujemo in tako pridemo do vrednosti, ki jo posamezna spremenljivka vsebuje. Sama imena spremenljivk morajo biti smiselna. To pomeni, da je iz imena razvidno, kaj pomeni podatek, ki ga hranimo v spremenljivki. Uporaba takih imen nam olajša razumevanje programske kode.

Pri imenovanju spremenljivk moramo upoštevati določena pravila. Tu bomo navedli poenostavljeno različico teh pravil, ki se pri jeziku C# ujemajo s tistimi, ki smo jih upoštevali pri jeziku java. Ta pravila za ime spremenljivke so naslednja:

- *prvi znak[2] mora biti črka,*
- *ime je lahko sestavljeno le iz črk angleške abecede[3], števk in znaka podčrtaj,*
- *ne sme biti enako drugemu imenu,*
- *ne sme biti enako rezerviranim besedam[4].*

V obeh jezikih velja, da se pri imenih upošteva (razlikuje), ali je črka mala ali velika. Navedimo še nekaj dodatnih nasvetov za poimenovanje spremenljivk v javi in C#:

- *Začetni znak imena spremenljivke naj bo mala črka (dogovor).*
- *Ime je sestavljeno iz ene besede. Če je besed več, jih zlepimo. Zlepimo jih tako, da naslednjo besedo začnemo z veliko začetnico, nap. vsotaSodihStevil.*
- *Podajanje tipa spremenljivk v imenu (t.i. madžarska notacija, na primer intVsota ali int _vsota) ni zaželeno.*
- *Ne uporabljamo imen, ki se med seboj razlikujejo le v uporabljenih velikih oz. malih črkah. Tako ni dobro, da bi imeli v programu tako spremenljivko* `Vsota` *kot tudi spremenljivko* `vsota`*.*

Podatkovni tip spremenljivke

Podatkovni tip spremenljivke nam pove, kakšna vrednost je lahko shranjena v spremenljivki. V javi smo spoznali tipe kot so `int`, `double`, `char` …. Te srečujemo tudi pri jeziku C#. Več o samih podatkovnih tipih si bomo ogledali v naslednjem poglavju.

[2] Prvi znak imena spremenljivke je lahko tudi podčrtaj, pa tudi nekateri drugi znaki. Toda uporabi drugih znakov se raje izogibamo.

[3] V imenu spremenljivk se po dogovoru ne uporabljajo šumniki, čeprav C# tako kot java podpira poljubne znake v kodi UNICODE.

[4] Rezervirane besede so besede, ki imajo v programskem jeziku vnaprej določen pomen, na primer `else`, `if`, `class`, `void` …

Deklaracija

Ko smo spoznavali spremenljivke v javi, smo povedali, da vsako spremenljivko pred prvo uporabo najavimo, ali, kot rečemo, deklariramo. S tem v pomnilniku rezerviramo prostor ustrezne velikosti, ki je določen glede na tip spremenljivke. Tudi v jeziku C# velja, da moramo vse spremenljivke pred prvo uporabo deklarirati. Kot v javi lahko to naredimo na začetku programa ali pa le vrstico pred njeno uporabo.

Deklaracija spremenljivke v C# in v javi:

tip imeSpremenljivke;

Deklaracija spremenljivke v C# in v javi, če poznamo začetno vrednost:

tip imeSpremenljivke = vrednost;

Ko pišemo programe, običajno na začetku deklariramo vse spremenljivke, za katere predvidevamo, da jih bomo potrebovali. Če določene deklarirane spremenljivke nikoli ne uporabimo, ob prevajanju prevajalnik za jezik C# izpiše opozorilo.

```
c:\documents and settings\administrator\my documents\visual studio projects\
consoleapplication1\avtomobili.cs(11,8): warning CS0168: The variable 'avto' is
declared but never used
```

Slika 18: Zapis opozorila za neuporabljeno spremenljivko avto.

Opozorila nam sicer ni potrebno upoštevati (program se je vseeno prevedel), vendar praviloma nima smisla, da imamo v programih nepotrebne spremenljivke.

Spremenljivke s konstantno vrednostjo

V določenih primerih potrebujemo spremenljivke, katerih vrednost se med izvajanjem programa ne spreminja. Gre za spremenljivke s konstantno vrednostjo ali krajše konstante. Namesto konstant bi lahko na tistem mestu v izrazu zapisali to vrednost. Toda tega se v C# in javi izogibamo. Na ta način namreč vemo, za kaj pri določeni vrednosti gre. Tako lahko ločimo med različnimi uporabami te konstante (npr. 20, ki pomeni starost in 20, ki pomeni ceno izdelka). Konstantne spremenljivke se od ostalih spremenljivk ločijo v deklaraciji. Obvezno uporabimo deklaracijo s prirejanjem začetne vrednosti ter pred tipom dodamo ustrezno rezervirano besedo (final v javi in const v C#).

Java:	**final** tip imeSpremenljivke = vrednost;
C#:	**const** tip imeSpremenljivke = vrednost;

Tabela 2: Deklaracija konstantnih spremenljivk.

V primeru, da želimo konstantni spremenljivki spremeniti njeno vrednost, prevajalnik javi napako.

Primer:

Poglejmo si preprost primer programa, v katerem nastopata dve spremenljivki a in b, pri čemer je spremenljivka b konstantna.

Zapis v javi:

```
J1:   public class Racunajmo{
J2:      public static void main(String[] args){
J3:         // Deklaracija spremenljivk
J4:         int a = 12;
J5:         final int b = 13;
J6:         int vsota, razlika;
J7:
J8:         // Uporaba
J9:         a = a + 2;
J10:        vsota = a + b;
```

```
J11:         razlika = a - b;
J12:
J13:         // Izpis
J14:         System.out.println("Spremenljivka a = " + a);
J15:         System.out.println("Spremenljivka b = " + b);
J16:         System.out.print("Vsota spremenljivk je " + vsota + ", razlika pa ");
J17:         System.out.println(razlika + ".");
J18:     }
J19: }
```

V jeziku C# sprememb praktično ni, razen tistih, ki smo jih opisali že prej (poimenovanje glavne metode in drugačna imena metod za izpisovanje). Razlika je le v deklaraciji konstante, ko moramo v C# uporabiti `const` namesto `final`.

Zapis v C#:

```
C1:  using System;
C2:  public class Racunajmo{
C3:     public static void Main(string[] args){
C4:        // Deklaracija spremenljivk
C5:        int a = 12;
C6:        const int b = 13;
C7:        int vsota, razlika;
C8:
C9:        // Uporaba
C10:       a = a + 2;
C11:       vsota = a + b;
C12:       razlika = a - b;
C13:
C14:       // Izpis
C15:       Console.WriteLine("Spremenljivka a = " + a);
C16:       Console.WriteLine("Spremenljivka b = " + b);
C17:       Console.Write("Vsota spremenljivk je " + vsota + ", razlika pa ");
C18:       Console.WriteLine(razlika + ".");
C19:    }
C20: }
```

Izpis na zaslonu:

```
Spremenljivka a = 14
Spremenljivka b = 13
Vsota spremenljivk je 27, razlika pa 1.
```

Razlaga. Najprej deklariramo spremenljivke `a`, `b`, `vsota` in `razlika` (J4 - J6, C5 – C7). Spremenljivka `a` ima vrednost 12, konstantna spremenljivka `b` pa ima vrednost 13. Nato vrednost spremenljivke `a` povečamo za 2 (J9, C10). Vrednost spremenljivke `a` je sedaj 14. Zatem določimo vrednost spremenljivke `vsota` (J10, C11) in spremenljivke `razlika` (J11, C12). Vrednost spremenljivke `vsota` je 27 in vrednost spremenljivke `razlika` je 1. Na koncu izvršimo stavke za izpis vrednosti (J14 – J17, C15 – C18).

2.5.2 Podatkovni tipi

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

V javi smo spoznali nekaj vgrajenih podatkovnih tipov, imenovanih osnovni podatkovni tipi. Tej množici ustreza množica vgrajenih podatkovnih tipov iz jezika C#, ki jih imenujemo tudi enostavni podatkovni tipi.

Spodnja tabela prikazuje najpogosteje uporabljene vgrajene podatkovne tipe. V prvem stolpcu so zapisani podatkovni tipi jezika java, medtem ko so v drugem stolpcu zapisani ustrezni podatkovni tipi jezika C#.

Java	**C#**	**Opis**
int	***int***	Celo število
long	***long***	Celo število (večje območje)

double	***double***	Realno število
char	***char***	Znak

boolean	***bool***	Logična vrednost
String	***string***	Niz znakov[5]

Tabela 3: Najpogosteje uporabljeni vgrajeni podatkovni tipi.

Jezik C# je z različico C# 3.0 pridobil še en podatkovni tip, ki ga napovemo z oznako `var`. Ta podatkovni tip so razvijalci C# uvedli zaradi potrebe po podpori anonimnim tipom, ki pa jih v tej diplomski nalogi ne bomo spoznali. Java tipa `var` ne pozna.

Osnovne značilnosti tipa `var`:

- `var` se uporablja za lokalne spremenljivke.
- Uporabljamo ga lahko le v deklaracijah, kjer spremenljivki priredimo še vrednost.
- Z oznako `var` prevajalniku naročimo, naj na podlagi tipa vrednosti izraza, ki jo spremenljivki priredimo, sam določi ustrezen tip spremenljivke.
- `var` v C# ni enakovreden tipu `var`, ki ga poznata JavaScript in Visual Basic.
- Vsak eksplicitno podan podatkovni tip (`int`, `double`, `string`, itn.) lahko pri deklaracijah, ki uporabljajo sočasno prirejanje vrednosti, zamenjamo z implicitnim podatkovnim tipom (`var`) in seveda obratno.

  Primeri:

  Eksplicitni tip:
  ```
  int a = 5;
  string b = "Balada.";
  bool c = true;
  ```

  Implicitni tip:
  ```
  var a = 5;
  var b = "Balada.";
  var c = true;
  ```

- Spremenljivki tipa `var` moramo ob deklaraciji obvezno navesti izraz na desni strani.

```
C:\Documents and Settings\Administrator\Desktop\VajeC#\Test12\Test12\
Program.cs(8,17): error CS0818: Implicitly-typed local variables must be
initialized
```

Slika 19: Izpis napake, če spremenljivki pri deklaraciji ne določimo izraza.

- Izraz, ki ga ob deklaraciji prirejamo spremenljivki tipa `var`, ne sme imeti vrednost `null`.

```
C:\Documents and Settings\Administrator\Desktop\VajeC#\Test12\Test12\
Program.cs(8,17): error CS0815: Cannot assign <null> to an implicitly-typed
local variable
```

Slika 20: Izpis napake, če za izraz zapišemo null.

- Spremenljivki, deklarirani s tipom `var`, lahko priredimo novo vrednost, če je le ta enakega tipa kot izraz pri deklaraciji. V primeru, ko spremenljivki poskušamo prirediti vrednost, katere tip se ne ujema s tipom izraza ob deklaraciji, prevajalnik izpiše napako.

  Primera:
  ```
  // Tip novega izraza se ujema s tipom izraza ob deklaraciji
     var a = 5;
     var b = 10;
     a = b;
  ```

[5] Podatkovna tipa Sting in string sta formalno referenčna podatkovna tipa. Ker pa java in C# (vsaj navidezno) omogočata, da z nizi delamo tako kot s števili, bomo nekaj časa tipu String v javi rekli kar osnovni tip in tipu string v C# enostavni tip, čeprav to formalno ni pravilno.

```
// Tip novega izraza se ne ujema s tipom izraza ob deklaraciji
var a = 5;
var b = "pet";
a = b;
```

```
C:\Documents and Settings\Administrator\Desktop\VajeC#\Test12\Test12\
Program.cs(10,17): error CS0029: Cannot implicitly convert type 'string' to
'int'
```

Slika 21: Izpis napake, ki opozarja, da tipa string ni mogoče implicitno pretvoriti v tip int.

Dejansko torej pri običajni uporabi s tipom `var` ne pridobimo praktično nič. Le ob deklaraciji namesto, da bi posebej napisali za kakšen tip gre, to povemo z vrednostjo izraza. Torej `var` pravzaprav sploh ni tip, ampak je le oznaka, da naj prevajalnik to nadomesti z ustreznim tipom.

Zgled

Dolžina poti

Prekolesarili smo dve petini poti, do cilja je še 21 kilometrov. Izračunajmo dolžino celotne poti.

Za izračun celotne poti uporabimo enačbo:

```
nePrevozeno = 3.0 / 5;
dolzinaPoti = 21 / nePrevozen;
```

Zapis v C# (eksplicitno):

```
C1:   using System;
C2:   public class DolzinaPotiEksplicitno{
C3:      public static void Main(string[] args){
C4:         double nePrevozeno = 3.0 / 5; // Neprevožena pot
C5:         int prevozeno = 21; // Prevožena pot
C6:
C7:         // Izračun dolžine prekolesarjene poti
C8:         double dolzinaPoti = prevozeno / nePrevozeno;
C9:
C10:        // Izpis celotne poti
C11:        Console.WriteLine(dolzinaPoti);
C12:     }
C13:  }
```

Zapis v C# (implicitno):

```
C1:   using System;
C2:   public class DolzinaPotiEksplicitno{
C3:      public static void Main(string[] args){
C4:         var nePrevozeno = 3.0 / 5; // Neprevožena pot
C5:         var prevozeno = 21; // Prevožena pot
C6:
C7:         // Izračun dolžine prekolesarjene poti
C8:         var dolzinaPoti = prevozeno / nePrevozeno;
C9:
C10:        // Izpis celotne poti
C11:        Console.WriteLine(dolzinaPoti);
C12:     }
C13:  }
```

Zapis na zaslonu:

```
35
```

2.5.3 Pretvarjanje med vgrajenimi podatkovnimi tipi

Pogosto tip vrednosti izraza ne ustreza želenemu tipu spremenljivke. Zato je potrebno vrednost ustrezno pretvoriti v drug tip.

V jeziku java smo spoznali tri načine pretvarjanja. In sicer:

- konverzija ali avtomatsko pretvarjanje,
- eksplicitna konverzija ali pretvarjanje z '`(tip)`' in
- pretvarjanje s metodami.

Tudi jezik C# pozna zgoraj naštete načine pretvarjanja med vgrajenimi podatkovnimi tipi.

Pretvarjanje iz niza v celo število

Če so v nizu zapisane le števke (in kot prvi znak morebiti – ali +), ga v C# lahko pretvorimo v število s metodo `int.Parse()`. V javi smo namesto omenjene metode uporabili metodo `Integer.parseInt()`.

Zgled

Zamenjava števk

Podan imamo niz "8304". Spremenimo ga tako, da bo njegova oblika "3084".

Pomagali si bomo s števkami števila 8304. Števke števila bomo pridobili s pomočjo aritmetičnih operatorjev / in %. Program zapišimo le v jeziku C#.

Zapis v C#:

```
C1:   using System;
C2:   public class SpremeniNiz{
C3:      public static void Main(string[] args){
C4:         // Podan niz
C5:         string niz = "8304";
C6:
C7:         // Niz pretvorimo v celo število
C8:         int stevilo = int.Parse(niz);
C9:
C10:        // Določimo števke števila
C11:        int e = stevilo % 10; // enice
C12:        int d = stevilo / 10 % 10; // destice
C13:        int s = stevilo / 100 % 10; // stotice
C14:        int t = stevilo / 1000; // tisočice
C15:
C16:        // V niz shranimo novo obliko
C17:        niz = "" + s + d + t + e;
C18:
C19:        // Izpišemo novo število
C20:        Console.WriteLine(niz);
C21:     }
C22:  }
```

Zapis na zaslon:

```
3084
```

Razlaga. Niz `niz` s klicem metode `int.Parse()` pretvorimo v celo število, ki ga priredimo spremenljivki `stevilo`. V vrsticah od C11 do C14 izračunamo števke. Po izračunu števk jih v ustreznem vrstnem redu shranimo v niz (C17). Na začetku smo navedli "" z namenom, da vrednosti navedenih spremenljivk (`s`, `d`, `t` in `e`) avtomatsko pretvorimo v nize in jih združimo v skupen niz. Ta niz predstavlja spremenjen niz `niz`, ki smo ga želeli dobiti. Pri tem si pomagamo z združevalnim operatorjem `+`.

Pretvarjanje iz niza v realno število

Če je v nizu zapisano realno število, ga v jeziku C# pretvorimo v število z metodo `double.Parse()`. Metodi `double.Parse()` v javi ustreza metoda `Double.parseDouble()`.

Zgled

Razlika števil

V dveh celoštevilskih spremenljivkah imamo shranjeni dve enomestni števili. Izračunajmo razliko med največjim in najmanjšim številom, ki ju lahko sestavimo iz teh dveh števk, če eno od teh pomeni enice, drugo pa desetinke.

Pri računanju razlike dveh realnih števil si pomagamo z metodo `Abs()`, ki se nahaja v razredu `Math`. Omenjeno metodo poznamo že iz jave. Program navedimo le v jeziku C#.

Zapis v C#:

```
C1:   using System;
C2:   public class RazlikaStevil{
C3:      public static void Main(string[] args){
C4:         // Števki števil
C5:         int prvaStevka = 9;
C6:         int drugaStevka = 5;
C7:
C8:         // Sestavimo niza
C9:         string prviNiz = prvaStevka + "," + drugaStevka;
C10:        string drugiNiz = drugaStevka + "," + prvaStevka;
C11:
C12:        // Niza pretvorimo v realni števili
C13:        double prvoStevilo = double.Parse(prviNiz);
C14:        double drugoStevilo = double.Parse(drugiNiz);
C15:
C16:        // Izračun razlike
C17:        double razlika = prvoStevilo - drugoStevilo;
C18:
C19:        // Izpis
C20:        Console.WriteLine("Razlika dveh sestavljenih stevil je " +
                              Math.Abs(razlika) + ".");
C21:     }
C22:  }
```

Zapis na zaslonu:

```
Razlika dveh sestavljenih stevil je 3,6.
```

Razlaga. Iz dveh števk in decimalne vejice sestavimo število tako, da jih ustrezno staknemo z združevalnim operatorjem `+`. S tem v vrstici C9 dobimo niz `prviNiz` in v vrstici C10 niz `drugiNiz`. V vrsticah C13 in C14 s pomočjo metode `double.Parse()` iz niza dobimo realno število. Sedaj lahko izračunamo razliko med realnima številoma. Izračunano razliko priredimo spremenljivki `razlika` (C17). Na koncu še izpišemo absolutno vrednost spremenljivke `razlika`, opremljeno z ustreznim besedilom.

Opomba: Metoda `double.Parse(niz)` v nizu ne prepozna decimalne pike (jo enostavno ignorira), zato v nizu uporabimo decimalno vejico. Tako obnašanje prevajalnika za C# je odvisno od nastavitev operacijskega sistema. Okolja .NET namreč ločilo decimalnega dela števila privzeto prevzeme iz operacijskega sistema. Zato pri običajnih nastavitvah operacijskega sistema (pri nas v Sloveniji), vejica (,) pomeni ločilo med celim in decimalnim delom števila. Znak . pa operacijski sistem uporablja le za vidno ločevanje skupin števk pri večjih številih. Zato se . v nizih, ki jih v jeziku C# pretvarjamo z metodo `Parse`, kar preskoči.
Obnašanje prevajalnika za javo pa načeloma ni odvisno od nastavitev operacijskega sistema. Zato tudi v nizih metoda `Double.parseDouble(niz)` pričakuje decimalno piko. Če uporabimo decimalno vejico, se program sesuje.

Primeri za jezik C#:

```
// V nizu uporabimo decimalno vejico
string prviNiz = "23,4";
double prvoStevilo = double.Parse(prviNiz); // Vrednost spremenljivke je 23,4

// V nizu uporabimo decimalno piko
string drugiNiz = "1.5";
double drugoStevilo = double.Parse(drugiNiz); // Vrednost spremenljivke je 15

// V nizu uporabimo decimalno piko in decimalno vejico
string tretjiNiz = "1.235,4";
double tretjeStevilo = double.Parse(tretjiNiz); // Vrednost spremenljivke je 1235,4
```

Primeri za jezik java:

```
// V nizu uporabimo decimalno vejico
```

```
String prviNiz = "23,4";
double prvoStevilo = Double.parseDouble(prviNiz); // Program se sesuje


// V nizu uporabimo decimalno piko
String drugiNiz = "1.5";
double drugoStevilo = Double.parseDouble(drugiNiz); // Vrednost spremenljivke je 1.5

// V nizu uporabimo decimalno piko in decimalno vejico
String tretjiNiz = "1.235,4";
double tretjeStevilo = Double.parseDouble(tretjiNiz); // Program se sesuje
```

2.5.4 Branje iz konzole

Za branje podatkov iz konzole smo v javi uporabljali metodo `readLine()`. Da smo lahko omenjeno metodo uporabili, smo morali predhodno ustvariti objekt tipa `BufferedReader`. To smo storili s stavkom

```
BufferedReader vhod = new BufferedReader(new InputStreamReader(System.in));
```

v katerem smo uporabili razreda iz knjižnice `java.io`. Podatek (niz) smo torej v javi prebrali z

```
vhod.readLine();
```

V jeziku C# za branje uporabimo metodo `ReadLine()`, ki se obnaša enako kot javanska metoda `readLine()`. Metoda je v jeziku C# statična in pripada razredu `Console`. V C# podatek torej preberemo z

```
Console.ReadLine();
```

Branje iz konzole v javi že dobro poznamo, zato si naslednji zgled oglejmo le v jeziku C#.

Uredimo tri števila od najmanjšega do največjega

Napišimo program, ki bo prebral tri cela števila in jih izpisal v padajočem vrstnem redu.

Pri razporeditvi števil si pomagajmo z metodama `Min()` in `Max()`, ki ju najdemo v razredu `Math`. Omenjeni metodi sta ekvivalentni metodama `min()` in `max()` iz jave, ki sta v javanskem razredu `Math`.

Zapis v C#:

```
C1:   using System;
C2:   public class PadajoceUrejanje{
C3:      public static void Main(string[] args){
C4:         // Vnos podatkov
C5:         Console.Write("Vnesi prvo stevilo: ");
C6:         int prvo = int.Parse(Console.ReadLine());
C7:         Console.Write("Vnesi drugo stevilo: ");
C8:         int drugo = int.Parse(Console.ReadLine());
C9:         Console.Write("Vnesi tretje stevilo: ");
C10:        int tretje = int.Parse(Console.ReadLine());
C11:
C12:        // Določimo razporeditev števil
C13:        int min = Math.Min(Math.Min(prvo, drugo), tretje);
C14:        int max = Math.Max(Math.Max(prvo, drugo), tretje);
C15:        int srednje = prvo + drugo + tretje - max - min;
C16:
C17:        // Izpis
C18:        Console.WriteLine();
C19:        Console.WriteLine("Urejena stevila od najmanjsega do najvecjega so: ");
C20:        Console.WriteLine(min + " " + srednje + " " + max);
C21:     }
C22:  }
```

Zapis na zaslon:

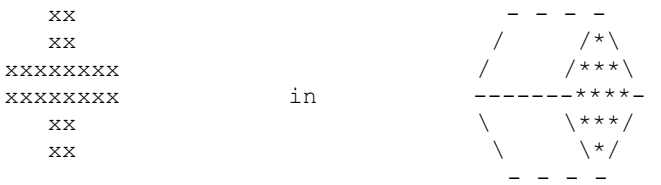
```
Vnesi prvo stevilo: 59
Vnesi drugo stevilo: -46
Vnesi tretje stevilo: 23

Urejena stevila od najmanjsega do najvecjega so:
-46 23 59
```

Razlaga. Preberemo podatke in jih pretvorimo v cela števila (C5 – C10). Nato določimo razporeditev števil s pomočjo metod `Math.Min()` in `Math.Max()` (C13 – C15). Po določitvi razporeditve števil izpišemo ustrezno besedilo (C19 – C20).

2.6 Naloge za utrjevanje znanja iz osnovnih ukazov

1. Sestavite program, ki bo na zaslon izpisal naslednja simbola:



2. Sestavite program, ki prebere osebne podatke določene osebe in jih na zaslon izpiše po naslednji obliki:

```
Ime in priimek
       Ulica in hišna številka
              Kraj in poštna številka
                     Država
```

   Namig: Pri oblikovanju izpisa osebnih podatkov si pomagajte s tabulatorskim znakom" `\t`.

3. Napišite program, ki prebere cela števila `a`, `b`, `c` in `x` ter izračuna vrednost polinoma $ax^2 + bx + c$. Poskrbite za lep izpis.

4. Za zrak se nam zdi, kot da nima teže. Vendar pa je količina zraka v nekem prostoru kar velika in to na koncu prinese tudi določeno težo. Napišite program, ki bo izračunal, kolikšna je masa zraka v sobi, ki ima obliko kvadra. Vse mere sobe vnašajte preko tipkovnice na cm natančno, rezultat izpišite na dag natančno. Gostota zraka je 1,3kg/m^3.

5. Trgovsko podjetje Trgovec s.p. je hotelo veliko zaslužiti, zato je podražilo izdelke za 30%. Ker blaga po tako visoki ceni ni moglo prodati, je ceno znižalo za 25%. Kolikšna je nova cena izdelka? Začetno ceno izdelka vnesite preko tipkovnice.

6. Napišite program, ki izračuna obseg in ploščino pravokotnega trikotnika. Dolžini katet vnesemo v program na začetku preko tipkovnice.
   Namig: Pomagajte si z metodo `Sqrt()` iz razreda `Math`.

7. Sestavite program, ki bo prebral podatke o času odhoda in prihoda vlaka (oba podatka sta podana v urah in minutah). Izračunajte čas potovanja. Predpostavite lahko, da ima vlak odhod in prihod na isti dan. Seveda je predpostavka tudi, da se pri vnosu ne zmotimo in so vsi vneseni podatki smiselni (torej, da so minute in ure v ustreznih mejah in da je odhod pred prihodom).

8. Napišite program, ki obrne dano trimestno celo število (npr. iz 415 naredi število 514) in izpiše razliko prvotnega in obrnjenega števila.

9. Napišite program, ki dolžino, izraženo v centimetrih, pretvori v dolžino v palcih. En palec je 2.54 cm. Rezultat naj bo izpisan na dve decimalki.

10. Pleskarji pleskajo plavalni bazen, ki ima obliko kvadra (zgornje ploskve bazen seveda nima). Pomagajte jim in napišite program, ki bo izračunal, kolikšno površino sten bazena morajo prebarvati. Dolžino, širino in globino bazena vnesimo preko tipkovnice na dm natančno. Globino bazena merimo od roba bazena do dna.

3 Odločitve

Stavki, ki smo jih spoznali do sedaj, so se izvajali zaporedoma. Velikokrat pa postopek zahteva, da nek stavek izvršimo le, če je izpolnjen določen pogoj. To nam jezika java in C# omogočata s pogojnim stavkom in s stavkom switch.

3.1 Pogojni stavki

Pogojne stavke uporabljamo, kadar želimo določene stavke izvesti samo v primeru, ko je izpolnjen nek pogoj. Za ponovitev znanja si še enkrat poglejmo sintakso pogojnega stavka v javi:

```
if(pogoj){
    stavek_a1;
    stavek_a2;
    …
    stavek_an;
}
else{
   stavek_b1;
   stavek_b2;
   …
   stavek_bn;
}
```

Sintaksa pogojnega stavka pomeni: "Če je vrednost izraza `pogoj` enaka `true`, potem izvedi $stavek_{a1}$ do $stavek_{an}$, sicer izvedi $stavek_{b1}$ do $stavek_{bn}$.".

Pogojni stavek se v jeziku C# ne razlikuje od pogojnega stavka v javi. To pomeni, da imajo pogojni stavki v obeh jezikih enako obliko in tudi enako izvajanje. To ilustrirajmo na naslednjih zgledih. Z zgledi bomo tudi poskrbeli, da se naše poznavanje pogojnih stavkov nekoliko osveži.

Zgledi

Absolutna vrednost

Napišimo preprost program, ki bo izpisal absolutno vrednost danega števila. V standardni knjižnici `Math` sicer obstaja metoda `Abs`, vendar rajši uporabimo postopek, kjer v primeru, da je dano število negativno, uporabimo nasprotno vrednost tega števila.

Zapis v javi:

```
J1:   import java.io.*;
J2:   public class Absolutno{
J3:      public static void main(String[] args)throws IOException{
J4:         BufferedReader vhod = new BufferedReader(
                                    new InputStreamReader(System.in));
J5:         // Prebrana vrednost
J6:         System.out.print("Vnesi stevilo: ");
J7:         double stevilo = Double.parseDouble(vhod.readLine());
J8:
J9:         // Preverimo predznak
J10:        if (stevilo < 0) stevilo = -stevilo;
J11:
J12:        // Izpis
J13:        System.out.println("Absolutna vrednost: " + stevilo);
J14:     }
J15:  }
```

V jeziku C# sprememb ni, razen tistih, ki so jih že spoznali v prejšnjih zgledih.

Zapis v C#:

```
C1:   using System;
C2:   public class Absolutno{
C3:      public static void Main(string[] args){
```

```
C4:          // Prebrana vrednost
C5:          Console.Write("Vnesi stevilo: ");
C6:          var stevilo = double.Parse(Console.ReadLine());
C7:
C8:          // Preverimo predznak
C9:          if (stevilo < 0) stevilo = -stevilo;
C10:
C11:         // Izpis
C12:         Console.WriteLine("Absolutna vrednost: " + stevilo);
C13:      }
C14:   }
```

Zapis na zaslonu:

```
Vnesi stevilo: -19.5
Absolutna vrednost: 19.5
```

Razlaga. Preberemo podatek in ga pretvorimo v realno število (J6 – J7, C5 – C6). Če je število negativno, določimo nasprotno vrednost tega števila (J10, C9). Na koncu izpišemo absolutno vrednost vnesenega števila.

Pravilni večkotnik

Sestavimo program, v katerega preko tipkovnice vnesemo število stranic pravilnega (enakostraničnega) večkotnika in dolžino stranice. Program izračuna in izpiše njegov obseg. Če je število stranic premajhno, da bi lahko tvorile lik, izpiše: "*Napaka! Stevilo stranic je premajhno.*". Program zapišimo le v jeziku C#.

Zapis v C#:

```
C1:   using System;
C2:   public class Veckotnik{
C3:      public static void Main(string[] args){
C4:         // Vneseno število vseh stranic
C5:         Console.Write("Vnesi stevilo stranic veckotnika: ");
C6:         int stranice = int.Parse(Console.ReadLine());
C7:
C8:         // Preverimo število stranic
C9:         if(stranice < 3){
C10:           Console.WriteLine("Napaka! Stevilo stranic je premajhno.");
C11:        }
C12:        else{
C13:           // Vnesena dolžina stranic
C14:           Console.Write("Vnesi dolzino stranice: ");
C15:           double dolzina = double.Parse(Console.ReadLine());
C16:
C17:           // Izračun in izpis obsega večkotnika
C18:           double obseg = stranice * dolzina;
C19:           Console.WriteLine("Obseg veckotnika je " + obseg + ".");
C20:        }
C21:     }
C22:  }
```

Zapis na zaslonu:

```
Vnesi stevilo stranic veckotnika: 5
Vnesi dolzino stranice: 5,3
Obseg veckotnika je 26,5.
```

Razlaga. V spremenljivko `stranice` se shrani vneseno število. Če je število manjše od `3`, se izpiše "`Napaka! Stevilo stranic je premajhno.`". Če pa je število večje ali enako `3`, se v spremenljivko `dolzina` shrani (novo) vneseno število. Nato se izračuna obseg večkotnika, ki se shrani v spremenljivko `obseg`. Vrednost spremenljivke `obseg` se na koncu izpiše.

Presek premic

Napišimo program, ki bo ugotovil, ali se dani dve premici v ravnini $y = k_1x + n_1$ in $y = k_2x + n_2$ sekata ali ne. Če se sekata, naj program določi še koordinate preseka.

Tu bomo uporabili gnezdene pogojne stavke. Stavki v telesu pogojnega stavka so namreč poljubni stavki. Torej lahko med njimi spet uporabimo pogojni stavek. Ker je ta znotraj drugega pogojnega stavka ("v gnezdu"), rečemo, da gre za gnezdeni pogojni stavek.

Kadar imata premici enaka smerna koeficienta, sta vzporedni. Če imata enaka tudi prosta člena, se premici pokrivata. Če pa sta smerna koeficienta različna, se premici sekata in sicer v natanko eni točki. Ker sta programa v jezikih java in C# praktično enaka, navedimo le tistega v C#.

Zapis v C#:

```
C1:   using System;
C2:   public class PresekPremic{
C3:      public static void Main(string[] args){
C4:         // Preberemo vrednosti prve premice
C5:         Console.WriteLine("Prva premica: ");
C6:         Console.Write("- smerni koeficient: ");
C7:         double k1 = double.Parse(Console.ReadLine());
C8:         Console.Write("- prosti clen: ");
C9:         double n1 = double.Parse(Console.ReadLine());
C10:
C11:        // Preberemo vrednosti druge premice
C12:        Console.WriteLine("Druga premica: ");
C13:        Console.Write("- smerni koeficient: ");
C14:        double k2 = double.Parse(Console.ReadLine());
C15:        Console.Write("- prosti clen: ");
C16:        double n2 = double.Parse(Console.ReadLine());
C17:
C18:        Console.WriteLine(); // Za lepši videz
C19:        // Glede na vrednosti izpišemo ustrezen tekst
C20:        if(k1 == k2){
C21:           if(n1 == n2){
C22:             Console.WriteLine("Premici se pokrivata.");
C23:           }
C24:           else{
C25:             Console.WriteLine("Premici sta vzporedni.");
C26:           }
C27:        }
C28:        else{
C29:           double x = (n2 - n1)/(k1 - k2);
C30:           double y = k2 * x + n2;
C31:
C32:           Console.Write("Premici se seceta v tocki ");
C33:           Console.WriteLine("(" + x + "," + y +").");
C34:        }
C35:     }
C36:  }
```

3.2 Stavek switch

Stavek `switch` je odločitveni stavek, ki ga uporabljamo v primerih, ko se je treba odločiti med več možnostmi samo na podlagi različnih vrednosti nekega izraza. Stavek je poseben primer pogojnega stavka.

Stavek `switch` poznata oba programska jezika. Pri spoznavanju odločitvenih stavkov v javi tega stavka nismo obravnavali. Zato stavek `switch` spoznajmo najprej v jeziku java.

3.2.1 Stavek switch v javi

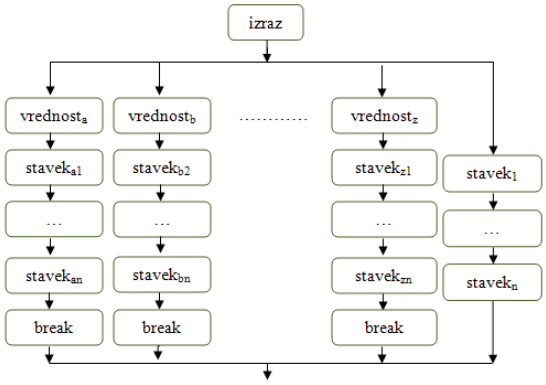
Sintaksa stavka `switch` v javi je naslednja:

```
switch(izraz){
   case vrednost_a:
      stavek_a1;
      …
      stavek_an;
      break;
   case vrednost_b:
      stavek_b1;
      …
```

```
            stavek_bn
            break;
        …
        case vrednost_z:
            stavek_z1;
            …
            stavek_zn;
            break;
        default:
            stavek_1;
            …
            stavek_n;
    }
```

Opomba: Stavek `break` je neobvezen del posamezne možnosti `case`. V začetku spoznavanja stavka `switch` predpostavimo, da je stavek `break` obvezen pri vsaki možnosti.

Vrednosti, ki so zapisane za besedo `case`, morajo biti konstantne. Program najprej izračuna vrednost izraza `izraz` ter poišče možnost `case` s to vrednostjo. Če program najde ustrezno možnost `case`, izvrši vse stavke, ki sledijo. Stavke izvršuje zaporedoma, dokler ne izvrši stavka `break`. Stavek `break` povzroči prekinitev izvajanja stavka `switch`. Če pa program ne najde ustrezne možnosti `case`, izvrši vse stavke, ki so zapisani za besedo `default`.



Slika 22: Shematičen prikaz stavka switch.

Izraz `izraz` mora v javi zavzemati vrednosti, ki so celoštevilskega ali znakovnega tipa. Če za izraz določimo vrednost, ki ni celoštevilskega ali znakovnega tipa (na primer `boolean`, `String`, `double`, itn.), prevajalnik javi napako.

```
C:\Documents and Settings\Administrator\Desktop\VajeC#\Test.java:4: incompatible types
found   : java.lang.String
required: int
                switch(sprem){
                       ^
```

Slika 23: Zapis napake, če za izraz določimo spremenljivko tipa String.

Stavek `switch`, pri katerem stavek `break` vsebujejo vse možnosti `case`, lahko zapišemo tudi kot naslednjo kombinacijo pogojnih stavkov:

```
if(izraz == vrednost_a){
    stavek_a1;
    …
    stavek_an;
}
else if(izraz == vrednost_b){
    stavek_b1;
    …
    stavek_bn;
}
```

```
…
else if(izraz == vrednostz){
   stavekz1;
   …
   stavekzn;
}
else{
   stavek1;
   …
   stavekn;
}
```

V stavku `switch` lahko del, ki se prične z besedo `default`, izpustimo. Če program ne bo našel ustrezne možnosti `case`, ne bo naredil ničesar.

Zgleda

Dnevi v tednu

Napišimo program, ki bo prebral številko dneva v tednu in izpisal njegovo ime. Pri tem predpostavimo, da se teden začne s ponedeljkom (torej številki 1 ustreza ponedeljek, številki 2 torek …). V primeru napačnega vnosa naj se izpiše ustrezno obvestilo.

Zapis v javi:

```
J1:   import java.io.*;
J2:   public class DneviTedna{
J3:      public static void main(String[] args)throws IOException{
J4:         BufferedReader vhod = new BufferedReader(
                                    new InputStreamReader(System.in));
J5:         // Preberemo številko dneva
J6:         System.out.print("Vnesi stevilko dneva v tednu: ");
J7:         int dan = Integer.parseInt(vhod.readLine());
J8:
J9:         // Številki določimo ime dneva
J10:        switch(dan){
J11:           case 1:
J12:              System.out.println("Ponedeljek.");
J13:              break;
J14:           case 2:
J15:              System.out.println("Torek.");
J16:              break;
J17:           case 3:
J18:              System.out.println("Sreda.");
J19:              break;
J20:           case 4:
J21:              System.out.println("Cetrtek.");
J22:              break;
J23:           case 5:
J24:              System.out.println("Petek.");
J25:              break;
J26:           case 6:
J27:              System.out.println("Sobota.");
J28:              break;
J29:           case 7:
J30:              System.out.println("Nedelja.");
J31:              break;
J32:           default:
J33:              System.out.println("Teden ima samo 7 dni.");
J34:        }
J35:     }
J36:  }
```

Zapis na zaslonu:

```
Vnesi stevilko dneva v tednu: 6
Sobota.
```

Razlaga. Preberemo številko dneva. Glede na vneseni podatek dobimo izpisan ustrezen dan tedna:

- Če je število enako `1`, izpišemo"`Ponedeljek.`" in s stavkom `break` prekinemo izvajanje stavka `switch`.

- Če je število enako `2`, izpišemo "`Torek.`". Nato s stavkom `break` izvajanje stavka `switch` prekinemo.
- ...
- Če je število enako `7`, izpišemo "`Nedelja.`". Iz stavka `switch` izstopimo s stavkom `break`.
- Če število ne obstaja, izpišemo "`Teden ima samo 7 dni.`".

Stara slovenska imena mesecev

V drugem razredu osnovne šole se otroci pri predmetu Spoznavanje narave in družbe učijo stara slovenska imena mesecev. Imen je veliko, zato si jih otroci težko zapomnijo. Sestavimo jim vadbeni program, ki bo sprejel številko meseca in na zaslon izpisal stavek "*Staro slovensko ime meseca je* ___ ustrezno pripadajoče ime___".

Stara slovenska imena mesecev so po vrsti: *prosinec, svečan, sušec, mali traven, veliki traven, rožnik, mali srpan, veliki srpan, kimavec, vinotok, listopad* in *gruden*.

Zapis v javi:

```
J1:   import java.io.*;
J2:   public class ImenaMesecev{
J3:      public static void main(String[] args)throws IOException{
J4:         // Preberemo številko meseca
J5:         BufferedReader vhod = new BufferedReader(
                                    new InputStreamReader(System.in));
J6:         System.out.print("Vnesite stevilko meseca: ");
J7:         int mesec = Integer.parseInt(vhod.readLine());
J8:
J9:         System.out.print("Staro slovensko ime meseca je ");
J10:        // Izpis imena meseca
J11:        switch (mesec){
J12:           case 1:
J13:              System.out.println("prosinec.");
J14:              break;
J15:           case 2:
J16:              System.out.println("svecan.");
J17:              break;
J18:           case 3:
J19:              System.out.println("susec.");
J20:              break;
J21:           case 4:
J22:              System.out.println("mali traven.");
J23:              break;
J24:           case 5:
J25:              System.out.println("veliki traven.");
J26:              break;
J27:           case 6:
J28:              System.out.println("roznik.");
J29:              break;
J30:           case 7:
J31:              System.out.println("mali srpan.");
J32:              break;
J34:           case 8:
J35:              System.out.println("veliki srpan.");
J36:              break;
J37:           case 9:
J38:              System.out.println("kimavec.");
J39:              break;
J40:           case 10:
J41:              System.out.println("vinotok.");
J42:              break;
J43:           case 11:
J44:              System.out.println("listopad.");
J45:              break;
J46:           case 12:
J47:              System.out.println("gruden.");
J48:              break;
J49:        }
J50:     }
J51:  }
```

Zapis na zaslonu:

```
Vnesite stevilko meseca: 7
Staro slovensko ime meseca je mali srpan.
```

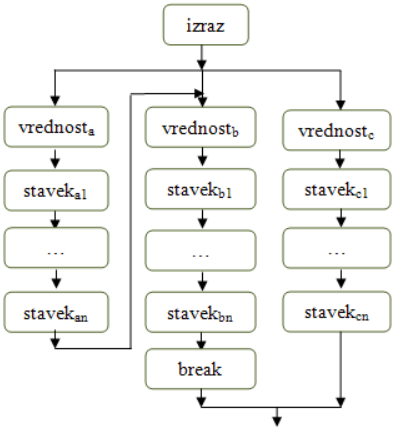
Razlaga. V konzolo vnašamo številke meseca. Glede na vneseno številko meseca se izpiše, kateremu staremu slovenskemu imenu pripada. Če vnesenega števila ni, se ne izpiše nič. V stavku `switch` smo uporabili stavek `break`, s katerim smo zaključili obravnavo posamezne možnosti in s tem tudi izvajanje stavka `switch`. Posamezno možnost smo s stavkom `break` zaključili, saj nismo želeli izvedbe stavkov naslednje možnosti (glej nadaljevanje).

Do sedaj smo stavek `break` obravnavali kot obvezen del vsake možnosti `case`. V nadaljevanju pa si poglejmo še stavek `switch`, v katerem posamezne možnosti `case` ne vsebujejo stavka `break`. Za lažje razumevanje predpostavimo, da ima stavek `switch` naslednjo obliko:

```
switch(izraz){
   case vrednost_a:
      stavek_a1;
      …
      stavek_an;
   case vrednost_b:
      stavek_b1;
      ..
      stavek_bn;
      break;
   case vrednost_c:
      stavek_c1;
      …
      stavek_cn;
}
```

Stavek se izvede na naslednji način:
Najprej se izračuna vrednost izraza `izraz`. Nato se poišče možnost `case` z vrednostjo, ki je enaka izračunani vrednosti izraza. Če ima iskana možnost `case`:

- $vrednost_a$, se izvršijo stavki od $stavek_{a1}$ do $stavek_{an}$. V najdeni možnosti `case` stavka `break` ni, zato se izvršijo še stavki naslednje možnosti `case`, torej še stavki od $stavek_{b1}$ do $stavke_{bn}$. Nato je zapisan stavek `break`, zato se izvajanje stavka `switch` prekine.
- $vrednost_b$, se izvršijo stavki od $stavek_{b1}$ do $stavek_{bn}$. V najdeni možnosti `case` tem stavkom sledi stavek `break`, zato se izvajanje stavka `switch` prekine.
- $vrednost_c$, se izvršijo stavki od $stavek_{c1}$ do $stavek_{cn}$. Stavka `break` tu ni. Toda ker ni naslednje možnosti `case`, se izvajanje stavka `switch` kljub temu konča. Izvršijo se torej le stavki od $stavek_{c1}$ do $stavek_{cn}$.



Slika 24: Shematičen prikaz zgornje oblike stavka switch.

Za splošno sintakso stavka `switch` bi izvajanje opisali tako: "Izračunaj izraz in poišči možnost `case`, katere vrednost je enaka vrednosti izraza. Za najdeno možnost `case` izvajaj vse stavke v stavku `switch` od tu dalje, dokler ne naletiš na stavek `break`, kjer prekini izvajanje stavka `switch`.

Zgleda

Dnevi v tednu

Popravimo program `DneviTedna.java` tako, da vrstice J13, J16 in J25 označimo kot komentar. Kako se bo sedaj izvajal program?

Zapis v javi:

```
J1:   import java.io.*;
J2:   public class DneviTedna1{
J3:      public static void main(String[] args)throws IOException{
J4:         BufferedReader vhod = new BufferedReader(
                                    new InputStreamReader(System.in));
J5:         // Preberemo številko dneva
J6:         System.out.print("Vnesi stevilko dneva: ");
J7:         int dan = Integer.parseInt(vhod.readLine());
J8:
J9:         // Številki določimo ime dneva
J10:        switch(dan){
J11:           case 1:
J12:              System.out.println("Ponedeljek.");
J13:              // break;
J14:           case 2:
J15:              System.out.println("Torek.");
J16:              // break;
J17:           case 3:
J18:              System.out.println("Sreda.");
J19:              break;
J20:           case 4:
J21:              System.out.println("Cetrtek.");
J22:              break;
J23:           case 5:
J24:              System.out.println("Petek.");
J25:              // break;
J26:           case 6:
J27:              System.out.println("Sobota.");
J28:              break;
J29:           case 7:
J30:              System.out.println("Nedelja.");
J31:              break;
J32:           default:
J34:              System.out.println("Teden ima samo 7 dni.");
J35:        }
J36:     }
J37:  }
```

Zapis na zaslonu:

```
Vnesi stevilko dneva: 5
Petek.
Sobota.
```

Razlaga. V konzolo vnašamo številke dneva. Glede na vneseno število dobimo izpisan ustrezen dan (oziroma dneve) v tednu:

- Če je število enako `1`, izpišemo "`Ponedeljek.`". Ker stavka `break` ni, nadaljujemo in izpišemo še "`Torek.`". Ker stavka `break` še vedno ni, nadaljujemo in izpišemo še "`Sreda.`". Sedaj nastopi stavek `break`, zato prekinemo izvajanje stavka `switch`.
- Če je število enako `2`, izpišemo "`Torek.`". Ker stavka `break` ni, nadaljujemo in izpišemo še "`Sreda.`". Sedaj nastopi stavek `break`, zato prekinemo izvajanje stavka `switch`.
- Če je število enako `3`, izpišemo "`Sreda.`". Ker sledi stavek `break`, prekinemo izvajanje stavka `switch`.
- Če je število enako `4`, izpišemo "`Cetrtek.`". S stavkom `break` prekinemo izvajanje stavka `switch`.

- Če je število enako `5`, izpišemo "`Petek.`". Ker stavka `break` ni, nadaljujemo in izpišemo še "`Sobota.`". Sedaj nastopi stavek `break`, zato prekinemo izvajanje stavka `switch`.
- Če je število enako `6`, izpišemo "`Sobota.`" in s stavkom `break` prekinemo izvajanje stavka `switch`.
- Če je število enako `7`, izpišemo "`Nedelja.`". Stavek `break` poskrbi, da se izvajanje stavka `switch` prekine.
- Če število ne obstaja, izpišemo "`Teden ima samo 7 dni.`".

Delavnik

Napišimo program, ki bo za prebran dan v tednu (6 - sobota, 7 - nedelja) izpisal, ali je delovni ali dela prost dan in ali so trgovine odprte ali zaprte (ob nedeljah morajo biti zaprte). Praznikov ne upoštevamo. Predpostaviti moramo, da je podatek o dnevu lahko tudi napačno vnesen.

Če je število dneva 1, 2, 3, 4 in 5 (ponedeljek, torek … petek), naj program torej izpiše "`Dan je delovni, trgovine so odprte.`". Če je število dneva 6 (sobota), naj izpiše "`Dan je dela prost, trgovine so odprte.`". V primeru, da je število dneva 7 (nedelja), naj se izpiše "`Dan je dela prost, trgovine so zaprte.`". Če število dneva ni število med 1 in 7, naj program izpiše "`Podatek je narobe vnesen.`".

Zapis v javi:

```
J1:   import java.io.*;
J2:   public class DelavnikTrgovin{
J3:      public static void main(String[] args)throws IOException{
J4:         BufferedReader vhod = new BufferedReader(
                                    new InputStreamReader(System.in));
J5:
J6:         // Vnos podatkov
J7:         System.out.print("Vnesi stevko dneva: ");
J8:         int stevka = Integer.parseInt(vhod.readLine());
J9:
J10:        // Določitev delovnika
J11:        switch(stevka){
J12:           case 1,2,3,4,5:
J13:              System.out.println("Dan je delovni, trgovine so odprte.");
J14:              break;
J15:           case 6:
J16:              System.out.println("Dan je dela prost, trgovine so odprte.");
J17:              break;
J18:           case 7:
J19:              System.out.println("Dan je dela prost, trgovine so zaprte.");
J20:              break;
J21:           default:
J22:              System.out.println("Podatek je narobe vnesen.");
J23:        }
J24:     }
J25:  }
```

Program prevedemo:

```
C:\Documents and Settings\Administrator\Desktop\VajeC#\naloge text3\
DelavnikTrgovin.java:10: : expected
       case 1,2,3,4,5:
              ^
```

Zakaj prevajalnik izpiše napako? Jezik java dovoljuje, da je za besedo `case` zapisana le ena konstantna vrednost. V primeru, ko v eni možnosti `case` navedemo več vrednosti, nas prevajalnik na to opozori z izpisom napake.

Popravimo program. Pri tem si pomagajmo z znanjem pridobljenim v prejšnjem zgledu, ko smo v izbrani možnosti `case` izpustili stavek `break`.

Zapis v javi:

```
J1:   import java.io.*;
J2:   public class DelavnikTrgovinPopravljen{
J3:      public static void main(String[] args)throws IOException{
J4:         BufferedReader vhod = new BufferedReader(
                                    new InputStreamReader(System.in));
J5:
J6:         // Vnos podatkov
J7:         System.out.print("Vnesi stevko dneva: ");
J8:         int stevka = Integer.parseInt(vhod.readLine());
J9:
J10:        // Določitev delovnika
J11:        switch(stevka){
J12:           case 1:
J13:           case 2:
J14:           case 3:
J15:           case 4:
J16:           case 5:
J17:              System.out.println("Dan je delovni, trgovine so odprte.");
J18:              break;
J19:           case 6:
J20:              System.out.println("Dan je dela prost, trgovine so odprte.");
J21:              break;
J22:           case 7:
J23:              System.out.println("Dan je dela prost, trgovine so zaprte.");
J24:              break;
J25:           default:
J26:              System.out.println("Podatek je narobe vnesen.");
J27:        }
J28:     }
J29:  }
```

Zapis na zaslonu:

```
Vnesi stevko dneva: 4
Dan je delovni, trgovine so odprte.
```

Razlaga. V konzolo vnašamo številke dneva. Glede na vneseno število izvedemo ustrezne stavke:
- Če je število `1`, izvedemo stavke možnosti `case1:`. Stavkov te možnosti ni, zato izvedemo stavke možnosti `case 2:`. Stavkov te možnosti prav tako ni, zato izvedemo stavke možnosti `case 3:`. Tudi stavkov te možnosti ni, zato izvedemo stavke možnosti `case 4:`. Stavkov omenjene možnosti spet ni, zato izvedemo stavke možnosti `case 5:`. Možnost `case 5:` ima stavke, zato le te izvedemo. Med izvedenimi stavki je tudi stavek `break`, s katerim prekinemo izvajanje stavka `switch`.
- Če je število `2`, izvedemo stavke možnosti `case 2:`. Stavkov te možnosti ni, zato izvedemo stavke možnosti `case 3:`. Stavkov te možnosti prav tako ni, zato izvedemo stavke možnosti `case 4:`. Stavkov omenjene možnosti ni, zato izvedemo stavke možnosti `case 5:`. Možnost `case 5:` ima stavke, zato jih izvedemo. Med izvedenimi stavki je tudi stavek `break`, s katerim prekinemo izvajanje stavka `switch`.
- ...
- Če je število `6`, izvedemo stavke možnosti `case 6:`. Med izvedenimi stavki je tudi stavek `break`, s katerim prekinemo izvajanje stavka `switch`.
- Če je število `7`, izvedemo stavke možnosti `case 7:`. Med izvedenimi stavki je tudi stavek `break`, s katerim prekinemo izvajanje stavka `switch`.
- Če število ne obstaja, izvedemo stavek možnosti `default`.

3.2.2 Stavek switch v C#

Stavek `switch` v jeziku C# se ne razlikuje veliko od istoimenskega stavka v javi. Razlika je v uporabi stavka `break` in v dovoljenem tipu vrednosti izraza `izraz`.
Stavek `break`, ki ima vlogo prekinitve stavka `switch`, je v jeziku C# obvezen del vsake možnosti `case`. Obvezen je tudi pri možnosti `default`, pa čeprav je ta zadnja. Če stavka `break` ne zapišemo, prevajalnik javi napako.

```
C:\documents and settings\administrator\desktop\vajec#\test\class1.cs(9,4): error
CS0163: Control cannot fall through from one case label ('case 1:') to another
```

Slika 25: Zapis napake, če v možnosti 'case 1:' ne zapišemo stavka break.

V javi mora izraz `izraz` imeti celoštevilsko ali znakovno vrednost tipa. Jezik C# dovoljuje, da ima izraz vrednosti tudi tipa `string`.

```
C:\Documents and Settings\Administrator\Local Settings\Application Data\Temporary
Projects\Test\Program.cs(12,21): error CS0151: A value of an integral type expected
```

Slika 26: Zapis napake, če za vrednost izraza izberemo nedovoljen tip.

Zgled

Dnevi v tednu

Spomnimo se programa `DneviTedna.java`, kjer smo prebrali številko dneva in izpisali dan po imenu. Popravimo program tako, da bo deloval v jeziku C#.
V stavku `switch` sprememb praktično ni. Razlika je le v možnosti `default`, ko moramo v jeziku C# dodati še stavek `break`.

Zapis v C#:

```
C1:   using System;
C2:   public class DneviTedna{
C3:      public static void Main(string[] args){
C4:         // Preberemo številko dneva
C5:         Console.Write("Vnesi stevilko dneva: ");
C6:         int dan = int.Parse(Console.ReadLine());
C7:
C8:         // Številki določimo ime dneva
C9:         switch(dan){
C10:           case 1:
C11:              Console.WriteLine("Ponedeljek.");
C12:              break;
C13:           case 2:
C14:              Console.WriteLine("Torek.");
C15:              break;
C16:           case 3:
C17:              Console.WriteLine("Sreda.");
C18:              break;
C19:           case 4:
C20:              Console.WriteLine("Cetrtek.");
C21:              break;
C22:           case 5:
C23:              Console.WriteLine("Petek.");
C24:              break;
C25:           case 6:
C26:              Console.WriteLine("Sobota.");
C27:                break;
C28:           case 7:
C29:              Console.WriteLine("Nedelja.");
C30:              break;
C31:           default:
C32:              Console.WriteLine("Teden ima samo 7 dni.");
C33:              break;
C34:        }
C35:     }
C36:  }
```

Razlaga. V konzolo vnašamo številke dneva. Glede na vneseno številko dneva dobimo izpisan ustrezen dan tedna:

- Če je število enako `1`, izpišemo"`Ponedeljek.`" in s stavkom `break` prekinemo izvajanje stavka `switch`.
- …
- Če je število enako `7`, izpišemo "`Nedelja.`" in s stavkom `break` prekinemo izvajanje stavka `switch`.

- Če število ne obstaja, izpišemo "`Teden ima samo 7 dni.`" in s stavkom `break` prekinemo izvajanje stavka `switch`.

Delovnik

Spomnimo se še programa `DelovnikTrgovin.java`, kjer smo za prebrano številko dneva za trgovine izpisali, ali je delovni ali dela prost dan in ali je trgovina odprta ali zaprta. V primeru napačnega vnosa smo izpisali ustrezno obvestilo. Spremenimo program `DelovnikTrgovin.java` v program `DelovnikTrgovin.cs`. Ali morda jezik C# dopušča, da za besedo `case` navedemo več konstantnih vrednosti?

Zapis v C#:

```
C1:   using System;
C2:   public class DelavnikTrgovin{
C3:      public static void Main(string[] args) {
C4:         // Vnos podatkov
C5:         Console.Write("Vnesi stevko dneva: ");
C6:         int stevka = int.Parse(Console.ReadLine());
C7:
C8:         // Določitev delovnika
C9:         switch(stevka){
C10:           case 1,2,3,4,5:
C11:              Console.WriteLine("Dan je delovni, trgovine so odprte.");
C12:              break;
C13:           case 6:
C14:              Console.WriteLine("Dan je dela prost, trgovine so odprte.");
C15:              break;
C16:           case 7:
C17:              Console.WriteLine("Dan je dela prost, trgovine so zaprte.");
C18:              break;
C19:           default:
C20:              Console.WriteLine("Podatek je narobe vnesen.");
C21:              break;
C22:        }
C23:     }
C24:  }
```

Program prevedemo:

```
C:\Documents and Settings\Administrator\Local Settings\Application Data\Temporary
Projects\Delovnik trgovin\Program.cs(11,8): error CS1003: Syntax error, ':' expected
C:\Documents and Settings\Administrator\Local Settings\Application Data\Temporary
Projects\Delovnik trgovin\Program.cs(11,8): error CS1525: Invalid expression term ','
C:\Documents and Settings\Administrator\Local Settings\Application Data\Temporary
Projects\Delovnik trgovin\Program.cs(11,9): error CS1002: ; expected
C:\Documents and Settings\Administrator\Local Settings\Application Data\Temporary
Projects\Delovnik trgovin\Program.cs(11,10): error CS1002: ; expected
C:\Documents and Settings\Administrator\Local Settings\Application Data\Temporary
Projects\Delovnik trgovin\Program.cs(11,10): error CS1525: Invalid expression term ','
…
```

Kot vidimo, tudi jezik C# zahteva, da je za besedo `case` zapisana le ena konstantna vrednost.

Program popravimo na enak način kot v javi.

Zapis v C#:

```
C1:   using System;
C2:   public class DelavnikTrgovinPopravljen{
C3:      public static void Main(string[] args) {
C4:         // Vnos podatkov
C5:         Console.Write("Vnesi stevko dneva: ");
C6:         int stevka = int.Parse(Console.ReadLine());
C7:
C8:         // Določitev delovnika
C9:         switch(stevka){
C10:           case 1:
C11:           case 2:
C12:           case 3:
C13:           case 4:
C14:           case 5:
C15:              Console.WriteLine("Dan je delovni, trgovine so odprte.");
```

```
C16:              break;
C17:           case 6:
C18:              Console.WriteLine("Dan je dela prost, trgovine so odprte.");
C19:              break;
C20:           case 7:
C21:              Console.WriteLine("Dan je dela prost, trgovine so zaprte.");
C22:              break;
C23:           default:
C24:              Console.WriteLine("Podatek je narobe vnesen.");
C25:              break;
C26:        }
C27:     }
C28:  }
```

Tu smo kršiti pravilo, da mora vsaka možnost obvezno vsebovati stavek `break`. Kljub temu se takšen program prevede in izvede. Izvede se po enakem postopku kot v javi (stran 43).
Jezik C# torej dopušča, da v posamezni možnosti `case` stavek `break` izpustimo. A to velja le, če možnost ne vsebuje nobenih stavkov (torej le v delih oblike kot je del `case 1:`).

Poglejmo si še en primer, v katerem stavek `break` izpustimo.

Letni časi

Leto delimo na štiri letne čase. Napišimo program, ki bo za prebrano ime meseca izpisal ustrezen letni čas.

Če je mesec december, januar ali februar, naj program izpiše "`Zima.`". Če je mesec marec, april ali maj, naj izpiše "`Pomlad.`". V primeru, da je mesec junij, julij ali avgust, naj program izpiše "`Poletje.`". Če pa je mesec september, oktober ali november, naj izpiše "`Jesen.`".
Pri pisanju kode bomo uporabili tako možnost, da uporabimo več vrednosti k enemu sklopu stavkov, kot tudi to, da je vrednost izraza lahko tipa `string`. V javi takega programa ne bi mogli napisati, ampak bi morali uporabiti stavek `if` (oziroma bi si pomagali kako drugače).

Zapis v C#:

```
C1:   using System;
C2:   public class LetniCas{
C3:      public static void Main(string[] args){
C4:         // Preberimo ime meseca
C5:         Console.Write("Vnesi ime meseca: ");
C6:         string mesec = Console.ReadLine();
C7:
C8:         // Določitev letnega časa
C9:         switch (mesec){
C10:          case "december":
C11:          case "januar":
C12:          case "februar":
C13:              Console.WriteLine("Zima");
C14:              break;
C15:          case "marec":
C16:          case "april":
C17:          case "maj":
C18:              Console.WriteLine("Pomlad");
C19:              break;
C20:          case "junij":
C21:          case "julij":
C22:          case "avgust":
C23:              Console.WriteLine("Poletje");
C24:              break;
C25:          case "september":
C26:          case "oktober":
C27:          case "november":
C28:               Console.WriteLine("Jesen");
C29:               break;
C30:        }
C31:     }
C32:  }
```

Zapis na zaslonu:

```
Vnesi ime meseca: julij
Poletje
```

Razlaga. V konzolo vnašamo imena mesecev. Glede na vnesen mesec se izpiše, kateremu letnemu času pripada. Če meseca ni, se ne izpiše nič. Po izpisu letnega časa se stavek `switch` zaključi s pomočjo stavka `break`.

3.3 Naloge za utrjevanje znanja iz odločitev

1. Napišite program, ki zahteva vnos stranic trikotnika in ugotovi, ali tak trikotnik obstaja. Če obstaja, ugotovite, če:
   - je trikotnik pravokoten
   - je trikotnik enakokrak
   - je trikotnik enakostraničen.

   Namig: Pogoj, da lahko tri števila določajo stranice trikotnika je ta, da so stranice pozitivna števila, posamezna stranica pa mora biti manjša kot vsota drugih dveh.

2. Sestavite program *KvadratnaEnacba*, ki bo prebral realni števila *a, b* in *c* ter poiskal rešitve kvadratne enačbe $ax^2 + bx + c = 0$. Pri tem upoštevajte vse možnosti, ki lahko nastopijo.

3. Sestavite program, ki bo čim lepše izpisal vsoto dveh danih kompleksnih števil s celimi komponentami, na primer:

   ```
   5     + -2+ i = 3+i
   1+3i + -1-2i =   i
   2+ i +   - i = 2
   ```

4. Smolčki praznujejo rojstni dan 29. februarja. Če leto ni prestopno, ga praznujejo 1. marca. Napišite program, ki bo smolčkom sporočil, ali bodo na določeno leto praznovali rojstni dan 29. februarja ali 1. marca.
   Namig: Leto je prestopno, kadar je deljivo s 4 in ni deljivo s 100 ali kadar je deljivo s 400.

5. Z uporabo stavka `switch` sestavite program, ki bo prebrano oceno po ameriški lestvici (torej znak A, B, …) pretvoril v slovensko. Pri tem upoštevajte, da velja:
   - *A odlično,*
   - *B prav dobro,*
   - *C in D dobro,*
   - *E zadostno in*
   - *F nezadostno.*

   Oceno izpišite na zaslon z besedo.

6. V spremenljivki *karta* je število od 0 do 51, ki predstavlja eno od igralnih kart za poker. Karte so predstavljene s celoštevilskimi vrednostmi od 0 do 51 kot prikazuje tabela. Napišite program, ki bo na podlagi števila izpisal vrednost in barvo karte.
   Namig: 52 = 13 x 4

Barva/karta	2	3	4	5	6	7	8	9	10	J	Q	K	A
križ	0	1	2	3	4	5	6	7	8	9	10	11	12
kara	13	14	15	16	17	18	19	20	21	22	23	24	25
srce	26	27	28	29	30	31	32	33	34	35	36	37	38
pik	39	40	41	42	43	44	45	46	47	48	49	50	51

7. Napišite program, ki zna izračunati obseg in ploščino geometrijskih likov. Uporabniku ponudite nabor geometrijskih likov (1 – kvadrat, 2 – pravokotnik, 3 – pravokotni trikotnik in 4 - krog). S pomočjo stavka `switch` ugotovite uporabnikovo izbiro, nato pa v odvisnosti od izbranega lika zahtevajte vnos ustreznih podatkov. Nato izpišite obseg in ploščino lika.

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

4 Zanke

Zanke v programiranju uporabljamo takrat, kadar želimo enega ali več stavkov ponoviti večkrat zaporedoma. V jezikih java in C# poznamo naslednje zanke: `while`, `for`, `do` in `foreach`.

V tem poglavju si bomo ogledali zanke `while`, `for` in `do`, medtem ko bomo zanko `foreach` spoznali v poglavju Tabele.

4.1 While

Zanka `while` je zanka, ki jo verjetno uporabljamo najbolj pogosto. V javi smo zanko `while` dodobra spoznali. Vseeno pa si za osvežitev znanja še enkrat poglejmo njeno obliko:

```
while(pogoj){
   stavek1;
   stavek2;
   …
   stavekn;
}
```

Oblika zanke pomeni: "Dokler je izpolnjen pogoj `pogoj` (torej dokler ima vrednost `true`), ponavljaj stavke $stavek_1$, $stavek_2$ … $stavek_n$.".

V jeziku C# se zanka `while` od istoimenske zanke v javi ne razlikuje. To pomeni, da ima zanka `while` v obeh programskih jezikih enako obliko in tudi enako izvajanje. To ilustrirajmo na nekaj zgledih. Z njimi bomo tudi poskrbeli, da se naše poznavanje zank, pridobljeno pri spoznavanju jezika java, nekoliko osveži. Zato je vsak zgled tudi podrobno razložen.

Zgledi

Izpis sodih števil

Izpišimo vsa soda števila od -10 do 10. Vsako število izpišimo v svojo vrstico.

Zapis v javi:

```
J1:   public class SodaStevila{
J2:      public static void main(String[] args){
J3:        // Deklaracija spremenljivke
J4:        int a = -10;
J5:        // Izpisujemo vsa soda števila
J6:        while (a <= 10){
J7:           System.out.println(a);
J8:           a = a + 2;
J9:        }
J10:       System.out.println("Konec izpisa sodih stevil.");
J11:    }
J12:  }
```

V jeziku C# je zanka povsem enaka. Da ne bomo čisto pozabili na novost v jeziku C#, tip `var` (glej stran 27), ga uporabimo pri deklaraciji spremenljivke `a`.

Zapis v C#:

```
C1:   using System;
C2:   public class SodaStevila{
C3:      public static void Main(string[] args){
C4:         // Deklaracija spremenljivke
C5:         var a = -10;
C6:         // Izpisujemo vsa soda števila
C7:         while (a <= 10){
C8:           Console.WriteLine(a);
C9:           a = a + 2;
C10:        }
C11:        Console.WriteLine("Konec izpisa sodih stevil.");
C12:     }
C13:  }
```

Zapis na zaslonu:

Razlaga. Spremenljivko `a` na začetku nastavimo na `-10` (J4, C5). Nato preverimo pogoj (J6, C7). Ker je vrednost v spremenljivki manjša ali enaka `10`, se izpiše vrednost spremenljivke `a` (J7, C8), torej `-10`. Vrednost spremenljivke `a` se poveča za `2` in je sedaj `-8`. Nato se ponovno preveri veljavnost pogoja (J6, C7). Ker je resničen (ima vrednost `true`), se izvede telo zanke. Izpiše se trenutna vrednost spremenljivke `a`. V našem primeru je to `-8`. Vrednost spremenljivke se poveča na `-6`. Ponovno preverimo pogoj. Ker je resničen …
Zanka se torej ponavlja, dokler je vrednost spremenljivke `a` manjša ali enaka `10`. Ko je vrednost v spremenljivki enaka `10`, se telo zanke izvede zadnjič. Vrednost spremenljivke `a` je po izvedbi `12`. Pogoj se ponovno preveri in ker ni več resničen, se zanka konča. Program se nadaljuje z naslednjo vrstico (J10, C11). Izpiše se stavek `Konec izpisa sodih stevil.`.

Buuum

"Buuum" je igra z naslednjim pravilom: Sodelujoči zaporedoma govorijo števila, vendar morajo namesto večkratnikov števila pet in večkratnikov števila sedem reči *buuum*. Sestavimo program, ki bo po tem pravilu izpisal cela števila od `a` do `b`.

Pomagali si bomo s pogojnim stavkom, ki preveri, ali je število večkratnik števila 5 ali števila 7. Ko bo pogoj pogojnega stavka resničen, bomo izpisali besedo *buuum*.

Razlik v kodi med jezikoma java in C# praktično ni, razen tistih, ki smo jih do sedaj že dobro spoznali. Zato navedimo le zapis v C#.

Zapis v C#:

```
C1:   using System;
C2:   public class Buuum{
C3:      public static void Main(string[] args){
C4:         // Preberemo interval
C5:         Console.Write("Vnesite a: ");
C6:         int a = int.Parse(Console.ReadLine());
C7:         Console.Write("Vnesite b: ");
C8:         int b = int.Parse(Console.ReadLine());
C9:         // Drugi vnos je manjši od prvega
C10:        if(b < a){
C11:            int t = a;
C12:            a = b;
C13:            b = t;
C14:        }
C15:        // Izpis besede buuum ali števil
C16:        while(a <= b){
C17:           if((a % 5 == 0) || (a % 7 == 0)){
C18:              Console.Write("buuum");
C19:           }
C20:           else{
C21:              Console.Write(a);
C22:           }
C23:           Console.Write(" ");
C24:           a = a + 1;
C25:        }
C26:     }
C27:  }
```

Zapis na zaslonu:

```
Vnesite a: 1
Vnesite b: 20
1 2 3 4 buuum 6 buuum 8 9 buuum 11 12 13 buuum buuum 16 17 18 19 buuum
```

Razlaga. Z branjem napolnimo spremenljivki `a` in `b`. Nato s pomočjo pogojnega stavka preverimo, če je vrednost spremenljivke `b` manjše od vrednosti spremenljivke `a` (C10). Če je pogoj resničen, zamenjamo vrednost spremenljivke `a` z vrednostjo spremenljivke `b` in vrednost spremenljivke `b` z vrednostjo spremenljivke `a` (C11 – C13). Pri zamenjavi vrednosti si pomagamo s pomožno spremenljivko `t`.
S pomočjo zanke `while` izpisujemo števila od `a` do `b`. Če je število večkratnik števila `5` ali števila `7` (C17), izpišemo besedo `buuum`, sicer izpišemo preverjeno število. Na koncu vsakega izpisa izpišemo presledek in se pomaknemo na naslednje število.

Trgovec

Z novim letom se je povečala stopnja davka na dodano vrednost (DDV) z 19% na 20%. Podjetje Trgovec d.o.o. mora v kratkem času spremeniti cene izdelkov tako, da se cena brez davka ne spremeni. To bi bilo sicer nepotrebno, a kaj, ko imajo v svojih podatkih le končne cene (torej cene z že upoštevanim davkom). Zato napišimo program, s katerim jim bomo pomagali. V program preko tipkovnice vnašamo cene, program pa izpisuje nove vrednosti. To počnemo, dokler ne vnesemo 0. Če je vnesena cena negativna, naj program izpiše "*Cena izdelka je narobe vnesena. Vnesi znova.*". Novo ceno zaokrožimo na dve mesti natančno.

Pomagali si bomo s stavkom `break`, ki poskrbi, da se zanka predčasno zaključi. Ko se namreč izvede stavek `break`, tako kot pri stavku `switch`, zapustimo stavek (zanko), v katerem se nahajamo.
Uporabili bomo neskončno zanko, v kateri je pogoj vedno `true` (resničen). Znotraj zanke bomo spraševali po ceni toliko časa, dokler ne vnesemo števila 0. Takrat bomo zanko prekinili s stavkom `break`.

Zapis v javi:

```
J1:   import java.io.*;
J2:   public class SpremembaCen{
J3:      public static void main(String[] args)throws IOException{
J4:         BufferedReader vhod = new BufferedReader(
                                    new InputStreamReader(System.in));
J5:         // Deklaracija spremenljivk
J6:         final double ddvPrvi = 1.19; // DDV 19%
J7:         final double ddvDrugi = 1.20; // DDV 20%
J8:         double cena;
J9:
J10:        // Preračunavanje cene iz 19% DDV v 20% DDV
J11:        while(true){
J12:           System.out.print("Vnesi ceno izdelka z 19% DDV: ");
J13:           cena = Double.parseDouble(vhod.readLine());
J14:
J15:           if(cena == 0){ // Konec vnosov
J16:              break;
J17:           }
J18:           if(cena < 0){
J19:              System.out.println(
                     "Cena izdelka je vnesena narobe. Vnesi znova.\n");
J20:           }
J21:           else{
J22:              cena = ddvDrugi * cena / ddvPrvi; // Nova cena
J23:              cena = Math.round(cena * 100) / 100.0;
J24:              System.out.println("Cena izdelka z 20% DDV je " + cena + ".\n");
J25:           }
J26:        }
J27:     }
J28:  }
```

V kodi jezika C# sprememb praktično ni, razen tistih, ki smo jih spoznali že pri prejšnjih zgledih. Tudi v jeziku C# za prekinitev zank uporabljamo stavek `break;`.

Zapis v C#:

```
C1:   using System;
C2:   public class SpremembaCen{
C3:      public static void Main(string[] args){
C4:         // Deklaracija spremenljivk
C5:         const double ddvPrvi = 1.19; // DDV 19%
C6:         const double ddvDrugi = 1.20; // DDV 20%
C7:         double cena;
C8:
C9:         // Preračunavanje cene iz 19% DDV v 20% DDV
C10:        while(true){
C11:           Console.Write("Vnesi ceno izdelka z 19% DDV: ");
C12:           cena = double.Parse(Console.ReadLine());
C13:
C14:           if(cena == 0){ // Konec vnosov
C15:              break;
C16:           }
C17:           if(cena < 0){
C18:              Console.WriteLine(
                     "Cena izdelka je vnesena narobe. Vnesi znova.\n");
C19:           }
C20:           else{
C21:              cena = ddvDrugi * cena / ddvPrvi; // Nova cena
C22:              cena = Math.Round(cena * 100) / 100.0;
C23:              Console.WriteLine("Cena izdelka z 20% DDV je " + cena + ".\n");
C24:           }
C25:        }
C26:     }
C27:  }
```

Zapis na zaslonu:

```
Vnesi ceno izdelka z 19% DDV: 200
Cena izdelka z 20% DDV je 201.68.

Vnesi ceno izdelka z 19% DDV: -65.5
Cena izdelka je vnesena narobe. Vnesi znova.

Vnesi ceno izdelka z 19% DDV: 0
```

Razlaga. Najprej deklariramo konstantni spremenljivki `ddvPrvi` in `ddvDrugi` in jima priredimo začetni vrednosti. V spremenljivki `ddvPrvi` hranimo 19% DDV, medtem, ko v spremenljivki `ddvDrugi` hranimo 20% DDV. Zatem najavimo spremenljivko `cena`. To spremenljivko potrebujemo za ceno posameznega izdelka. Nato vstopimo v zanko, katere pogoj je vedno `true`. Torej se načeloma zanka izvaja neskončno dolgo, saj bo pogoj ves čas izpolnjen. Znotraj zanke sprašujemo po ceni izdelka. Če je vnesena cena enaka `0`, prekinemo izvajanje zanke `while`. Če je vnesena cena negativna, izpišemo opozorilno besedilo `Cena izdelka je vnesena narobe. Vnesi znova.`. Če pa je vnesena cena izdelka pozitivna, izračunamo novo ceno. Zaokrožimo jo na dve decimalni mesti ter jo izpišemo. Po izpisu opozorila ali nove cene se vrnemo na začetek zanke `while` in ponovimo postopek. Postopek ponavljamo tako dolgo, dokler ni vnesena cena enaka `0`. Takrat se izvede stavek `break` in izvajanje zanke se konča.

4.2 For

Zanko `for` ima v javi naslednjo obliko:

```
for (predZanko; pogoj; poKoraku){
   stavek1;
   stavek2;
   …
   stavekn;
}
```

Oblika zanke pomeni: "Izvedi stavek `predZanko`. Nato preveri `pogoj`. Če je izpolnjen, izvedi stavke $stavek_1$, $stavek_2$ … $stavek_n$ in nato še stavek `poKoraku`. Ponovno preveri `pogoj`. Če je še vedno izpolnjen, ponovno izvedi stavke $stavek_1$, $stavek_2$ … $stavek_n$ in še stavek `poKoraku`. Ponovno preveri `pogoj`. Ko `pogoj` ni izpolnjen, se zanka konča."

Zanka **`for`** v jeziku C# se od istoimenske zanke v javi ne razlikuje.

Pri spoznavanju zanke `for` v javi smo povedali, da je zanka `for` ekvivalentna zanki `while`. Ker pa sta zanki `while` in `for` v jezikih java in C# enaki, velja ekvivalentnost tudi v C#.

Katero zanko bomo uporabili, je odvisno od naše odločitve. Zanko `for` običajno uporabljamo, ko moramo šteti ponovitve. Stavek `predZanko` takrat običajno uporabimo za nastavitev števca, v pogoju preverjamo, če je števec že dosegel določeno mejo, v stavku `poKoraku` pa povečujemo (ali zmanjšujemo) števec.

Zgleda

Izpis sodih števil

Popravimo programa `SodaStevila.java` in `SodaStevila.cs` iz prejšnjega poglavja tako, da bomo uporabili zanko `for`. Izpisati želimo soda števila od -10 do 10, vsakega v svojo vrstico.

Zapis v javi:

```
J1:   public class SodaStevila1{
J2:      public static void main(String[] args){
J3:         // Izpisujemo vsa soda števila
J4:         for(int a = -10; a <= 10; a = a + 2){
J5:            System.out.println(a);
J6:         }
J7:         System.out.println("Konec izpisa sodih stevil.");
J8:      }
J9:   }
```

V jeziku C# glede na javanski program sprememb v zanki ni.

Zapis v C#:

```
C1:   using System;
C2:   public class SodaStevila1{
C3:      public static void Main(string[] args){
C4:         // Izpisujemo vsa soda števila
C5:         for(int a = -10; a <= 10; a = a + 2){
C6:            Console.WriteLine(a);
C7:         }
C8:         Console.WriteLine("Konec izpisa sodih stevil.");
C9:      }
C10:  }
```

Zapis na zaslonu:

```
-10
-8
-6
-4
-2
0
2
4
6
8
10
Konec izpisa sodih stevil.
```

Razlaga. V vrsticama J4 in C5 se spremenljivki `a` najprej priredi začetna vrednost `-10`. Nato se preveri pogoj. Ker je resničen, vstopimo v zanko (J5, C6) in izpiše se vrednost spremenljivke (`-10`). Nato se vrednost spremenljivke `a` poveča za `2`. Preveri se pogoj in ker še vedno velja (ima vrednost `true`), se ponovno izpiše vrednost spremenljivke (`-8`). Na ta način se zanka `for` izvaja, dokler je vrednost spremenljivke `a` manjša ali enaka `10`. Ko vrednost spremenljivke doseže `12`,

se zanka zaključi. Program se nadaljuje z vrstico za zanko `for` (J7, C8). Izpiše se besedilo `Konec izpisa sodih stevil..`

Pravokotnik

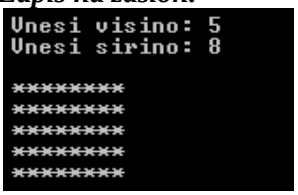
Napišimo program, ki vpraša po širini in višini pravokotnika, nato pa izriše pravokotnik sestavljen s samih zvezdic.

Tu bomo uporabili gnezdeno zanko `for`. Tako kot pri pogojnem stavku, so tudi stavki v telesu zanke `for` poljubni stavki. In če uporabimo spet zanko `for` (ali tudi katero drugo zanko), je ta znotraj druge zanke `for` ("v gnezdu"). Zato rečemo, da gre za gnezdeno zanko `for`.
Program zapišimo le v jeziku C#.

Zapis v C#:

```
C1:   using System;
C2:   public class Pravokotnik{
C3:      public static void Main(string[] args){
C4:         // Vnos dimenzij pravokotnika
C5:         Console.Write("Vnesi visino: ");
C6:         int visina = int.Parse(Console.ReadLine());
C7:         Console.Write("Vnesi sirino: ");
C8:         int sirina = int.Parse(Console.ReadLine());
C9:
C10:        // Preverimo, obstoj pravokotnika
C11:        if(0 < visina && 0 < sirina){
C12:           Console.WriteLine();
C13:
C14:           // Izpisovanje vrstic
C15:           for(int i = 1; i <= visina; i++){
C16:              // Izpis stolpcev
C17:              for(int j = 1; j <= sirina; j++){
C18:                 Console.Write("*");
C19:              }
C20:              // Po izpisu stolpcev, se pomaknemo v novo vrstico
C21:              Console.WriteLine();
C22:           }
C23:        }
C24:     }
C25:  }
```

Zapis na zaslon:



4.3 Do

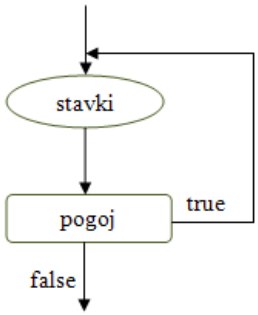
Zanka `do` je manj pogosto uporabljena oblika zanke. Na voljo je v obeh programskih jezikih. Zanko nekateri imenujejo tudi zanka `do-while`, saj nas po obliki spominja na zanko `while`.

Pri spoznavanju jezika java smo zanko `do` zgolj omenili, zato si jo poglejmo podrobneje.

Zanka `do` ima naslednjo obliko:

```
do{
   stavek1;
   stavek2;
   …
   stavekn;
}while(pogoj);
```

Zanka se izvede na naslednji način:
Najprej se izvede telo zanke (stavki zapisani za besedo do v zavitih oklepajih). Nato se preveri veljavnost pogoja. Če je izpolnjen, se ponovno izvede telo zanke. Nato se spet preveri veljavnost pogoja. Če je še vedno izpolnjen, se zanka ponovi, sicer se zanka konča.
Iz opisa je razvidno, da je zanka `do` v primerjavi s prejšnjima dvema zankama malo drugačna. Pri zankah `while` in `for` se veljavnost pogoja preverja pred vsako ponovitvijo telesa zanke, pri zanki `do` pa se najprej izvede telo zanke in šele nato se preveri pogoj. Števki v telesu zanke `do` se torej v vsakem primeru izvedejo vsaj enkrat.



Slika 27: Shematičen prikaz zanke do.

Zgled
Štirimestno število
Napišimo program, ki bo bral števila toliko časa, dokler ne bo vneseno štirimestno pozitivno celo število.

Zapis v javi:

```
J1:   import java.io.*;
J2:   public class StirimestnoStevilo{
J3:      public static void main(String[] args)throws IOException{
J4:         // Deklaracija spremenljivk
J5:         BufferedReader vhod = new BufferedReader(
                                    new InputStreamReader(System.in));
J6:         int vnos;
J7:
J8:         // Branje števil
J9:         do{
J10:           System.out.print("Vnesite stirimestno pozitivno stevilo: ");
J11:           vnos = Integer.parseInt(vhod.readLine());
J12:        }while (vnos < 1000 || vnos > 9999);
J13:
J14:        System.out.println("KONEC PROGRAMA");
J15:     }
J16:  }
```

Zapis na zaslonu:

```
Vnesite stirimestno pozitivno stevilo: 134
Vnesite stirimestno pozitivno stevilo: 1
Vnesite stirimestno pozitivno stevilo: 666666
Vnesite stirimestno pozitivno stevilo: 1234
KONEC PROGRAMA
```

Razlaga. Najavimo spremenljivko `vnos`. S pomočjo zanke `do` beremo števila, dokler je pogoj zanke (J12) izpolnjen. Napisani pogoj pravi, da je vneseno število bodisi premajhno (je negativno ali pa tri ali manj mestno), bodisi preveliko (je pet ali večmestno). Ko pogoj ni izpolnjen (ko je torej število ustrezno), se delovanje zanke zaključi in izpiše se stavek `KONEC PROGRAMA` (J14).

Tako kot pri večini do sedaj spoznanih stavkov, se tudi zanka `do` v jeziku C# ne razlikuje od istoimenske zanke v javi. Vseeno pa popravimo javanski program `StirimestnoStevilo` tako, da bo deloval v jeziku C#. V sami zanki spremembe ne bodo potrebne.

Zapis v C#:

```
C1:   using System;
C2:   public class StirimestnoStevilo{
C3:      public static void Main(string[] args){
C4:         // Deklaracija spremenljivke
C5:         int vnos;
C6:
C7:         // Branje števil
C8:         do{
C9:            Console.Write("Vnesite stirimestno pozitivno stevilo: ");
C10:           vnos = int.Parse(Console.ReadLine());
C11:        }while(vnos < 1000 || vnos > 9999);
C12:
C13:        Console.WriteLine("KONEC PROGRAMA");
C14:     }
C15:  }
```

Zapis na zaslonu:

```
Vnesite stirimestno pozitivno stevilo: 14387
Vnesite stirimestno pozitivno stevilo: 13
Vnesite stirimestno pozitivno stevilo: 1355
KONEC PROGRAMA
```

Oglejmo si uporabo zanke `do` še na dveh zgledih.

Število e

Določimo vrednost števila `e` (osnove naravnih logaritmov) tako, da seštevamo vrsto `1 + 1/1 + 1/(1·2) + 1/(1·2·3) + …` dokler ne bo prišteti člen manjši od `1/100000`.

Vrednost števila `e` bomo določili tako, da bomo ponavljali postopek, kjer bomo najprej izračunali posamezen člen vrste, nato pa ta člen prišteli preostalim že izračunanim členom. Program zapišimo le v jeziku C#.

Zapis v C#:

```
C1:   using System;
C2:   public class SteviloE{
C3:      public static void Main(string[] args){
C4:         // Deklaracija spremenljivk
C5:         int i = 0;
C6:         double clen = 1.0;
C7:         double e = 0;
C8:
C9:         // Izračun števila e
C10:        do{
C11:           e = e + clen;
C12:           i = i + 1;
C13:           clen = clen / i;
C14:        }while(0.00001 <= clen);
C15:
C16:        // Izpis
C17:        Console.WriteLine("Stevilu e smo izracunali vrednost " + e + ".");
C18:     }
C19:  }
```

Zapis na zaslonu:

```
Stevilu e smo izracunali vrednost 2,71827876984127.
```

Razlaga. Najprej deklariramo spremenljivko `i` in ji priredimo začetno vrednost `0`. S to spremenljivko povemo, kateri člen po vrsti računamo. Zatem deklariramo spremenljivko `clen` in ji priredimo začetno vrednost, ki je `1.0`. To spremenljivko potrebujemo za vrednost posameznega člena. Zatem deklariramo spremenljivko `e` in ji priredimo začetno rednost `0`. V tej spremenljivki bomo hranili trenutno vrednost števila `e`.

Vrednost prvega člena že poznamo. Zato to vrednost prištejemo vrednosti spremenljivke `e`. Nato povečamo vrednost spremenljivke `i`. S tem povemo, da bomo računali drugi člen. Potem izračunamo vrednost drugega člena. Ko vrednost izračunamo, jo preverimo. Če je pogoj zanke resničen (če torej člen še ni dovolj majhen), ponovno ponovimo telo zanke in prištejemo vrednost drugega člena vrednosti spremenljivke `e`. Nato ponovno povečamo vrednost spremenljivke `i`. Izračunamo vrednost tretjega člena. Vrednost ponovno preverimo. Če je pogoj zanke resničen, se telo zanke ponovi. Če pa je zaračunani člen že dovolj majhen, se izvajanje zanke konča. Izvrši se stavek za zanko, s katerim izpišemo dobljeno vrednost števila `e`.

Števila deljiva s tri

Izpišimo števila od 1 do 30, ki so deljiva s številom tri. Števila naj bodo izpisana v isti vrstici in ločena s presledkom.

Pomagali si bomo z ukazom `continue`, ki poskrbi, da se posamezna ponovitev zanke predčasno prekine in se zanka nadaljuje z naslednjo ponovitvijo (če je pogoj za ponovitev zanke še izpolnjen).
Uporabili bomo pogojni stavek, s katerim bomo preverjali, če število ni deljivo s 3. Če bo pogoj resničen, bomo s pomočjo stavka `continue` takoj nadaljevali z naslednjo ponovitvijo stavkov telesa zanke.

Zapis v C#:

```
C1:   using System;
C2:   public class Deljivo3{
C3:      public static void Main(string[] args){
C4:         // Deklaracija spremenljivke
C5:         int stevilo = 0;
C6:         const int zgMeja = 30; // Zgornja meja števil
C7:
C8:         // Izpis števil deljivih s tri
C9:         do{
C10:           stevilo++;
C11:           if(stevilo % 3 != 0) continue;
C12:           Console.Write(stevilo + " "); // Izpis le, če je število deljivo s 3
C13:        }while(stevilo <= zgMeja);
C14:
C15:        // Premik v novo vrstico
C16:        Console.WriteLine();
C17:     }
C18:  }
```

Razlaga. Spremenljivko `stevilo` nastavimo na vrednost `0` in konstantno spremenljivko `zgMeja` na vrednost `30`. Potem povečamo vrednost spremenljivke `stevilo` za `1`. Če je vrednost spremenljivke deljivo s številom `3` (C11), se izpiše vrednost spremenljivke in presledek. Če vrednost spremenljivke ni deljivo s `3`, pa se samo poveča vrednost spremenljivke `stevilo` za `1`. V tem primeru s stavkom `continue` skočimo na pogoj v vrstici C13.

Zapis na zaslonu:

```
3 6 9 12 15 18 21 24 27 30
```

Opomba: Stavek `continue` uporabljamo tudi v telesu zank `while` in `for`, poznamo pa ga seveda tudi v javi.

4.4 Naloge za utrjevanje znanja iz zank

1. Tabelirajte funkcijo $y = tan(x)$ na intervalu od 0 do π s korakom $\pi/10$. Če vrednost y vmes preseže 10, prekinite izvajanje zanke.
   Namig: Pomagajte si z metodo *Math.Tan()*.

2. Napišite program, ki bere posamezne znake in jih kodira v števila na naslednji način:

- Ne loči velike in male črke,
- črke od a do e (A do E) kodira po vrsti s števili od 1 do 5,
- številke 0 do 9 kodira po vrsti s števili od 6 do 15,
- črke od f do z (F do Z) kodira s števili od 16 do 36 v obratnem vrstnem redu,
- vse ostale znake zakodira s številom 0.

Program naj za vsak posamezen znak izpiše na zaslon `znak = koda`. Branje se konča, če je prebran znak `*`.
Namig: Za pretvarjanje iz tipa `string` v tip `char` si pomagajte z metodo `char.Parse(niz)`.

Primer kodiranja:

```
Vnesi znak: a
a = 1

Vnesi znak: Z
Z = 16

Vnesi znak: *
```

3. Pitagora je imenoval dve celi števili prijateljski, če je vsota deliteljev prvega števila enaka drugemu številu in obratno. Napišite program, ki bo prebral dve celi števili in preveril, ali sta števili prijateljski.

   *Primer***:**
   Števili 220 in 284 sta si prijateljski.
   220 = 1 + 2 + 4 + 5 + 10 + 11 + 20 + 22 + 44 + 55 + 110 = 284
   284 = 1 + 2 + 4 + 71 + 142 = 220

4. Podjetje Parkiranje d.o.o. upravlja parkirno hišo in hoče voditi evidenco njene zasedenosti. Zato vas pokliče, da bi jim napisali program, ki bi jim pri tem pomagal. Parkirna hiša ima 100 parkirnih mest in je ob zagonu programa prazna. Za vsak avto, ki v njej parkira, operater vtipka **+**, za vsak avto, ki odpelje iz nje pa, **-**. Za izhod iz programa pritisne malo črko **e** (kot Exit). Program vsakič tudi izpiše, koliko parkirnih mest je zasedenih. Program naj ustrezno reagira, če hoče avto parkirati v polni parkirni hiši ali če avto pripelje iz "prazne" parkirne hiše.

5. Gosenulje so posebne živali. Ko se izvalijo iz jajčeca, so videti natanko tako kot gosenice. Za razliko od gosenic, ki se zabubijo in iz katerih nato nastane metulj, odraščajo gosenulje malo drugače.

   Prvo zimo svojega življenja se zabubijo in zaspijo za toliko let, kolikor je vsota števk števila njihovih nog. V času, ko spijo, jim vsako leto zrase dodaten par nog. Primer: gosenulja, ki bi šla spat s 173 pari nog, bi se zbudila čez 1+7+3=11 let, hkrati pa bi imela tudi 11 parov nog več, torej 184. Naslednjič zaspijo v istem letu, kot se zbudijo.

   Napiši program, ki najprej vpraša, koliko parov nog je imela gosenulja, ko je šla prvič spat, nato pa izpiše, ali se bo kdaj zgodilo, da se bo gosenulja zbudila s toliko pari nog, da bo njihovo število deljivo s 199. Gosenulje umrejo, ko dopolnijo 1234 let.

5 Naključna števila

V programiranju večkrat potrebujemo naključna števila. V jeziku java smo za generiranje naključnih števil uporabljali predvsem metodo `random()` iz razreda `Math`. Omenjeni razred najdemo tudi v jeziku C#, vendar v njem te metode ali njej podobne ni.

Poleg metode `random()` smo v javi za generiranje naključnih števil uporabljali tudi razred `Random`, ki pripada paketu `java.util`. Temu razredu v jeziku C# ustreza istoimenski razred.

V jeziku java smo metode razreda `Random` uporabljali tako, da smo najprej ustvarili objekt tipa `Random`. Ta objekt si lahko predstavljamo kot nekakšen boben, iz katerega potem z različnimi metodami "žrebamo" naključna števila. Enak način dela velja tudi v jeziku C#.

Najpogostejše metode razreda `Random` so prikazane v spodnji tabeli. V prvem stolpcu tabele so zapisane metode jezika java, v drugem stolpcu pa ekvivalentne metode jezika C#.

Java	C#	Razlaga
nextInt()	***Next()***	Naključno celo število tipa `int`.
nextInt(int n)	***Next(int n)***	Naključno število tipa `int` iz intervala [`0`, `n`).
-	***Next(int n, int m)***	Naključno število tipa `int` iz intervala [`n`, `m`).
nextDouble()	***NextDouble()***	Naključno število tipa `double` iz intervala [`0`, `1`).

Tabela 4: Najpogosteje uporabljena metode razreda Random.

Ilustrirajmo uporabo teh metod z dvema zgledoma.

Kocka

Srečko bi rad zvedel, kolikokrat mora vreči kocko, da bo skupaj vrgel 100 pik ali več. A ve, da je metanje kocke naporno opravilo. Zato mu napišimo program, ki bo simuliral metanje kocke tako, da bo "metal" kocko toliko časa, dokler ne bo vsota vseh pik presegla 100. Program naj po vsakem metu izpiše vrženo število pik in skupno vsoto. Na koncu naj izpiše število metov kocke, potrebnih, da je skupna vsota presegla 100.

V zanki bomo z ustrezno metodo razreda `Random` določili število pik pri posameznem metu. Dobljene pike bomo nato prišteli k skupni vsoti vseh pik.

Zapis v javi:

```
J1:   import java.util.*;
J2:   public class Kocka{
J3:      public static void main(String[] args){
J4:         // Ustvarimo generator naključnih števil
J5:         Random nakljucno = new Random();
J6:
J7:         // Zgornja meja vseh pik
J8:         final int zgMeja = 100;
J9:         // Šteje število pik
J10:        int vsota = 0;
J11:        // Šteje število metov
J12:        int stevec = 0;
J13:
J14:        // Simulacija meta kocke
J15:        while(vsota <= zgMeja){
J16:           // Število pik ob posameznem metu kocke
J17:           int met = nakljucno.nextInt(6) + 1;
J18:           vsota = vsota + met;
J19:           System.out.println("Zadnji met: " + met + "\tVsota: " + vsota);
J20:           stevec = stevec + 1;
J21:        }
J22:        // Izpis števila metov kocke
J23:        System.out.println("\nStevilo metov kocke je " + stevec + ".");
J24:     }
J25:  }
```

Pri programu v jeziku C# moramo spremeniti klic metode za generiranje števil. Namesto `nextInt(…)` uporabimo `Next(…)`.

Zapis v C#:

```
C1:   using System;
C2:   public class Kocka{
C3:      public static void Main(string[] args){
C4:         // Ustvarimo generator naključnih števil
C5:         Random nakljucno = new Random();
C6:
C7:         // Zgornja meja vseh pik
C8:         const int zgMeja = 100;
C9:         // Šteje število pik
C10:        int vsota = 0;
C11:        // Šteje število metov
C12:        int stevec = 0;
C13:
C14:        // Simulacija meta kocke
C15:        while(vsota <= zgMeja){
C16:           // Število pik ob posameznem metu kocke
C17:           int met = nakljucno.Next(6) + 1;
C18:           vsota = vsota + met;
C19:           Console.WriteLine("Zadnji met: " + met + "\tVsota: " + vsota);
C20:           stevec = stevec + 1;
C21:        }
C22:        // Izpis števila metov kocke
C23:        Console.WriteLine("\nStevilo metov kocke je " + stevec + ".");
C24:     }
C25:  }
```

Razlaga. Najprej ustvarimo generator naključnih števil (J5, C5). Nato deklariramo ustrezne spremenljivke in jim priredimo ustrezne začetne vrednosti.
S pomočjo zanke `while` simuliramo metanje kocke. Pri vsakem "metu" kocke določimo število dobljenih pik (J17, C17). Te pike prištejemo skupni vsoti pik (J18, C18). Dobljene pike in skupno vsoto pik na vsakem koraku izpišemo (J19, C19). Na vsakem koraku tudi povečujemo števec (J20, C20), s katerim štejemo število metov kocke oz. korake zanke. Na koncu še izpišemo število vseh metov kocke.

Geslo

Iz oddelka za varovanje podatkov smo dobili nalogo, da napišemo program za samodejno generiranje gesel. Sodelavci tega oddelka želijo gesla naslednje oblike:

- najprej dve naključni števki,
- potem tri naključne velike črke,
- nato še ena naključna števka in
- na koncu naključno trimestno število.

Pri generiranju naključnih črk si bomo pomagali z dejstvom, da je znak predstavljen kar s številom, ki predstavlja njegovo kodo. Z njim torej lahko računamo. Izraz `geslo.Next('A', 'Z'+1)` je torej koda neke velike črke. Za pretvarjanje iz celega števila v znak bomo uporabili operator `(char)`. Ker sta programa po logiki praktično enaka, navedimo le tistega v C#.

Zapis v C#:

```
C1:   using System;
C2:   public class Geslo{
C3:      public static void Main(string[] args){
C4:         // Ustvarimo generator naključnih števil
C5:         Random geslo = new Random();
C6:
C7:         // Določimo dve števki
C8:         int stev1 = geslo.Next(10);
C9:         int stev2 = geslo.Next(10);
C10:
C11:        // Določimo tri naključne črke (velike tiskane)
C12:        char c1 = (char)geslo.Next('A','Z'+1);
C13:        char c2 = (char)geslo.Next('A','Z'+1);
C14:        char c3 = (char)geslo.Next('A','Z'+1);
```

```
C15:        // Določimo eno naključno številko
C16:        int stev3 = geslo.Next(10);
C17:
C18:        // Določimo naključno tromestno število
C19:        int s = geslo.Next(100, 1000);
C20:
C21:        // Izpis gesla
C22:        Console.WriteLine(
                    "Geslo: " + stev1 + stev2 + c1 + c2 + c3 + stev3 + s);
C23:    }
C24: }
```

Zapis na zaslonu:

```
Geslo: 99OSF8654
```

Razlaga. Najprej ustvarimo generator naključnih števil. Nato generiramo posamezne elemente gesla (C8 - C19). Za generiranje naključne črke uporabimo izraz `geslo.Next('A', 'Z' + 1)`. Z omenjenim izrazom določimo kodo črke. Ker se koda črke (tipa `int`) ne spremeni v črko (tip `char`) samodejno, pred omenjenim izrazom zapišemo operator `(char)`. S tem operatorjem iz kode dobimo naključno črko, ki jo nato priredimo spremenljivki. Po generiranju združimo vse elemente in jih izpišemo na zaslon (C22).

5.1 Naloge za utrjevanje znanja iz naključnih števil

1. Napišite program, ki bo prebral število *n* ter izpisal 5 naključnih realnih števil med 0 in *n-1*. Realna števila naj imajo največ dve decimalki.

2. V poglavju Odločitve smo med nalogami za utrjevanje znanja naredili nalogo, ki za vnesene stranice preveri, če lahko določajo trikotnik. V tej nalogi pa naj bodo stranice naključna izbrana števila iz intervala [-6, 20). Sestavimo program, ki bo za tri naključno izbrane stranice preveril, ali lahko tvorijo trikotnik. V primeru, da je trikotnik mogoče sestaviti, naj program izračuna še njegovo ploščino in obseg.
   Namig: Kot smo povedali pri omenjeni nalogi, je pogoj, da lahko tri števila določajo stranice trikotnika ta, da so stranice pozitivna števila, posamezna stranica pa mora biti manjša kot vsota drugih dveh.

3. Generirajte naključno štirimestno pozitivno število in izračunajte vsoto njegovih števk.

4. Računalnik si je "izmislil" število med 1 in 100. Koliko korakov boste potrebovali, da število uganete? Sestavite ustrezen program. Računalnik vam odgovarja:
   - s "premajhno", če bo vaše število manjše od računalnikovega in
   - s "preveliko", če bo vaše število večje.

   Seveda vam bo povedal tudi, ko boste število uganili.

5. Napišite program, ki bo pomagal generirati števila za igro Loto. Računalnik naj naključno generira samo prvo in drugo število, medtem ko ostalih 5 vnese uporabnik. Pri tem mora uporabnik pri vnosu sam poskrbeti, da kombinacija ne vsebuje dveh enakih števil. Števila izpišite.
   Namig: Pri generiranju prvega in drugega števila z ustrezno kodo poskrbite, da se prvi števili ne ujemata.

6. Napišite program, ki generira naključno celo število med 1 in 100000 in nato izpiše, koliko je lukenj v tem številu. Z luknjami so mišljene "luknje" v grafični podobi posameznih števk:
   - števke 0, 4, 6, 9 imajo po eno luknjo,
   - števka 8 ima dve luknji in
   - preostale števke nimajo nobene luknje.

   Primer:

   ```
   Stevilo 6854 ima 4 luknje.
   ```

6 Nizi

Nizi so referenčni[6] podatkovni tipi, ki jih poznamo v obeh programskih jezikih. Z njimi smo se v predhodnih poglavjih že nekoliko seznanili.

Niz je lahko sestavljen iz enega ali več znakov, poznamo pa tudi t.i. prazen niz, ki ne vsebuje nobenega znaka. V javi do posameznega znaka v nizu dostopamo z metodo `charAt()`. V jeziku C# pa v ta namen uporabimo kar oglate oklepaje (`[]`). Uporaba oklepajev je ekvivalentna uporabi metode `charAt()`. To pomeni, da z uporabo oglatih oklepajev ne moremo spreminjati znakov v nizu, temveč jih lahko le beremo. Izraz `niz[indeks]` se torej ne sme pojaviti na levi strani prireditvenega stavka.

Java	niz.**charAt(**i**)**;
C#	niz**[**i**]**;

Tabela 5: Dostop do znaka z indeksom i v nizu *niz*.

Vsak znak, ki je vsebovan v nizu, ima svoj indeks. Indeksiranje znakov se v obeh programskih jezikih prične z `0` in se konča z `dolžina niza - 1`.

Primer:

Niz	"D	o	b	e	r		d	a	n	."
Indeks	0	1	2	3	4	5	6	7	8	9

Da smo zvedeli dolžino niza, smo v javi uporabljali metodo `length()`. V jeziku C# v ta namen uporabljamo lastnost `Length`.

Primer:

Java:
```
String n = "Rock Otocec";
int dolzina = n.length();
```

C#:
```
string n = "Rock Otocec";
int dolzina = n.Length;
```

Vrednost spremenljivke `dolzina` je 11.

Za delo z nizi poleg omenjenih metod, lastnosti in operatorja [] pogosteje uporabljamo še metode, ki so prikazane v spodnji tabeli. V prvem stolpcu so zapisane metode jezika java, v drugem stolpcu pa ekvivalentne metode jezika C#.

Java	**C#**	**Opis**
equals(niz)	***Equals(niz)***	S postopkom preverjamo enakost niza *niz* in niza, nad katerim uporabljamo metodo. Ukaz vrne vrednost *true*, če sta niza enaka.
compareTo(niz)	***CompareTo(niz)***	S postopkom primerjamo dva niza po abecednem položaju. Če je vrnjena vrednost: - *pozitivna*, je *niz* po abecedi pred nizom, nad katerim je uporabljena metoda. - *negativna*, je *niz* po abecedi za uporabljenim nizom. - *nič*, sta *niz* in uporabljeni niz enaka (ju sestavljajo isti znaki).
toUpperCase()	***ToUpper()***	Metoda spremeni v nizu, nad katerim je uporabljena,vse male črke v velike.
toLowerCase()	***ToLower()***	Metoda spremeni v nizu, nad katerim je uporabljena, vse velike črke v male.

[6] Poleg že obravnavanih podatkovnih tipov poznamo v javi in v C# še referenčne ali tako imenovane sklicne podatkovne tipe. V spremenljivki referenčnega podatkovnega tipa hranimo referenco na ustrezen podatek in ne podatka samega. Med referenčne podatkovne tipe spadajo tabele, objekti …

substring(int,int)	***Substring(int,int)***	Metoda vrne podniz danega niza. Prvi argument pomeni začetni položaj (indeks) v nizu. Drugi argument v javi pomeni položaj (indeks) za zadnjim znakom podniza, medtem pa v C# dolžino podniza. Če drugega argumenta ni, metoda vrne vse znake do konca niza.
indexOf(niz)	***IndexOf(niz)***	S postopkom preverimo, če je *niz* podniz v nizu, nad katerim metodo uporabljamo. Metoda vrne celo število, ki predstavlja na katerem indeksu se v nizu prvič kot podniz prične niz *niz*. Če *niz* ni podniz uporabljenega niza, je vrnjena vrednost -1.

Tabela 6: Metode razreda String.

Ilustrirajmo uporabo omenjenih metod (Tabela 6) z naslednjimi zgledi.

Samoglasniki

Sestavimo program, ki prebere niz in ga izpiše tako, da vse samoglasnike nadomesti z zvezdico (*).

Pomagali si bomo z metodo `IndexOf()`, s katero bomo preverili, če je posamezen znak prebranega niza podniz niza "`aAeEiIoOuU`".
Posamezne znake prebranega niza, ki jih bomo bodisi spremenili bodisi ohranili, bomo dodali v pomožni niz. Pomožni niz bo na začetku programa prazen (`""`).

Zapis v javi:

```
J1:   import java.io.*;
J2:   public class Samoglasnik{
J3:      public static void main(String[] args)throws IOException{
J4:         BufferedReader vhod = new BufferedReader(
                                    new InputStreamReader(System.in));
J5:         // Vnos niza
J6:         System.out.print("Vnesi niz: ");
J7:         String niz = vhod.readLine();
J8:
J9:         // Pomožne spremenljivke
J10:        String samoglasniki = "aAeEiIoOuU"; // Niz samoglasnikov
J11:        String novNiz = ""; // Nov niz
J12:        int dolzina = niz.length(); // Dolžina prebranega niza
J13:
J14:        // Zamenjava samoglasnikov z zvezdico
J15:        for(int i = 0; i < dolzina; i++){
J16:           char znak = niz.charAt(i);
J17:           if(samoglasniki.indexOf(znak) != -1){ // Znak je samoglasnik
J18:              novNiz = novNiz + '*'; // Samoglasnik zamenjamo z *
J19:           }
J20:           else{
J21:              novNiz = novNiz + znak; // Ostali znaki ostanejo nespremenjeni
J22:           }
J23:        }
J24:
J25:        // Izpis novega niza
J26:        System.out.println("Spremenjen niz: " + novNiz);
J27:     }
J28:  }
```

V kodi v jeziku C# so (poleg že obravnavanih razlik) razlike v ukazu za:

- določitev dolžine niza,
- določitev posameznega znaka niza in
- preverjanje obstoja podniza v nizu,

ko moramo v jeziku C# uporabiti:

- `Length` namesto `length()`,
- oglate oklepaje namesto `charAt()` in
- `IndexOf()` namesto `indexOf()`.

Zapis v C#:

```
C1:   using System;
C2:   public class Samoglasnik{
C3:      public static void Main(string[] args){
C4:         // Vnos niza
C5:         Console.Write("Vnesi niz: ");
C6:         string niz = Console.ReadLine();
C7:
C8:         // Pomožne spremenljivke
C9:         string samoglasniki = "aAeEiIoOuU"; // Niz samoglasnikov
C10:        string novNiz = ""; // Nov niz
C11:        int dolzina = niz.Length; // Dolžina prebranega niza
C12:
C13:        // Zamenjava samoglasnikov z zvezdico
C14:        for(int i = 0; i < dolzina; i++){
C15:           char znak = niz[i];
C16:           if(samoglasniki.IndexOf(znak) != -1){ // Znak je samoglasnik
C17:              novNiz = novNiz + '*'; // Samoglasnik zamenjamo z *
C18:           }
C19:           else{
C20:              novNiz = novNiz + znak; // Ostali znaki ostanejo nespremenjeni
C21:           }
C22:        }
C23:
C24:        // Izpis novega niza
C25:        Console.WriteLine("Spremenjen niz: " + novNiz);
C26:     }
C27:  }
```

Zapis na zaslonu:

```
Vnesi niz: Lepa Vida kolo vodi.
Spremenjen niz: L*p* V*d* k*l* v*d*.
```

Razlaga. Najprej preberemo niz (J7, C6), nato deklariramo niz samoglasnikov (J10, C9) in prazen niz (J11, C10). Potem določimo dolžino niza `niz` (J12, C11). Z zanko se sprehodimo preko vseh znakov v nizu. Znotraj zanke deklariramo spremenljivko `znak` in ji priredimo trenutni znak niza `niz`. V vrsticama J17 in C16 preverimo, če je znak, katerega koda je shranjena v spremenljivki `znak`, podniz niza `samoglasniki`. Če je pogoj izpolnjen, se nizu `novNiz` doda znak '`*`' (J18, C17), sicer pa se mu doda trenutno preverjeni znak (J21, C20). Na koncu izvedemo izpis niza `novNiz`.

Galci in Rimljani

Verjetno je vsak med nami že kdaj prebral kak strip o Asterixu in Obelixu. Zato vemo, da se imena vseh Galcev zaključijo z "ix", na primer Asterix, Filix, Obelix, Dogmatix, itn. Napišimo program, ki bo prebral ime, nato pa bo izpisal "*Galec!*", če gre za galsko ime, v nasprotnem primeru pa bo izpisal "*Rimljan!*". Predpostavimo, da so vnesena imena dolga vsaj tri znake. Program zapišimo le v jeziku C#.

Zapis v C#:

```
C1:   using System;
C2:   public class Galec{
C3:      public static void Main(string[] args){
C4:         // Preberemo ime
C5:         Console.Write("Vnesi ime: ");
C6:         string ime = Console.ReadLine();
C7:
C8:         // Podniz
C9:         string podniz = ime.Substring(ime.Length - 2);
C10:        podniz = podniz.ToLower(); // Da ne bo težav z velikimi črkami
C11:
C12:        // Glede na ime izpišimo ustrezen tekst
C13:        if(podniz.Equals("ix")){
C14:           Console.WriteLine("Galec!");
C15:        }
C16:        else {
C17:           Console.WriteLine("Rimljan!");
C18:        }
C19:     }
C20:  }
```

Metodo `Equals()` iz razreda `String` lahko v jeziku C# nadomestimo z operatorjem enakosti (==). Tega v javi ne moremo narediti, saj v tem jeziku operator == ne preverja enakost vsebine dveh nizov, temveč preverja enakost dveh naslovov, kjer sta niza shranjena.
Zgornji program bi torej lahko zapisali tudi na naslednji način.

Zapis v C#:

```
C1:   using System;
C2:   public class Galec{
C3:      public static void Main(string[] args){
C4:         // Preberemo ime
C5:         Console.Write("Vnesi ime: ");
C6:         string ime = Console.ReadLine();
C7:
C8:         // Podniz
C9:         string podniz = ime.Substring(ime.Length - 2);
C10:        podniz = podniz.ToLower(); // Da ne bo težav z velikimi črkami
C11:
C12:        // Glede na ime izpišimo ustrezen tekst
C13:        if(podniz == "ix"){
C14:           Console.WriteLine("Galec!");
C15:        }
C16:        else {
C17:           Console.WriteLine("Rimljan!");
C18:        }
C19:     }
C20:  }
```

Zapis na zaslonu:

```
Vnesi ime: Filix
Galec!
```

Razlaga. Na začetku preko konzolnega okna preberemo ime osebe in ga shranimo v spremenljivki `niz` (C6). Potem določimo podniz iz zadnjih dveh znakov niza `niz` (C9). V vrstici C10 pretvorimo podniz `podniz` v male tiskane črke. Nato preverimo, če je podniz `podniz` enak nizu "`ix`" (C13). Če to drži, izpišemo "`Galec!`", sicer izpišemo "`Rimljan!`".

Razporeditev besed po abecedi

Napišimo program, ki bo prebral tri besede in jih izpisal po abecednem vrstnem redu. Na primer za prebrane besede *buda*, *vescine* in *tempelj*, naj program izpiše *buda tempelj vescine*. Predpostavimo, da so prebrane besede zapisane z malimi tiskanimi črkami.

Program sestavimo po korakih:
Najprej preko konzolnega okna določimo tri besede.

```
Console.Write("Vnesi prvo besedo: ");
string beseda1 = Console.ReadLine();
Console.Write("Vnesi drugo besedo: ");
string beseda2 = Console.ReadLine();
Console.Write("Vnesi tretjo besedo: ");
string beseda3 = Console.ReadLine();
```

Nato preverimo, če je beseda `beseda3` abecedno pred besedo `beseda1`.

- Če je pogoj resničen, zamenjamo vrstni red besed `beseda3` in `beseda1`.

```
if(beseda1.CompareTo(beseda3) > 0) {
   string pom = beseda1;
   beseda1 = beseda3;
   beseda3 = pom;
}
```

V naslednjem koraku preverimo, če je beseda `beseda2` abecedno pred besedo `beseda1`.

- Če je pogoj resničen, zamenjamo vrstni red besed `beseda2` in `beseda1`.

```
if(beseda1.CompareTo(beseda2) > 0) {
   string pom = beseda1;
   beseda1 = beseda2;
   beseda2 = pom;
}
```

Za konec preverimo še, če je beseda `beseda3` abecedno pred besedo `beseda2`.

- Če je pogoj resničen, zamenjamo vrstni red besed `beseda3` in `beseda2`.

```
if(beseda2.CompareTo(beseda3) > 0) {
   string pom = beseda2;
   beseda2 = beseda3;
   beseda3 = pom;
}
```

Besede so abecedno urejene, zato jih izpišemo. Ločimo jih s presledkom.

```
Console.WriteLine("\n" + beseda1 + " " + beseda2 + " " + beseda3);
```

Združimo posamezne korake v celoten program.

Zapis v C#:

```
C1:   using System;
C2:   public class SkritZaklad{
C3:      public static void Main(string[] args){
C4:         // Vnesene besede
C5:         Console.Write("Vnesi prvo besedo: ");
C6:         string beseda1 = Console.ReadLine();
C7:         Console.Write("Vnesi drugo besedo: ");
C8:         string beseda2 = Console.ReadLine();
C9:         Console.Write("Vnesi tretjo besedo: ");
C10:        string beseda3 = Console.ReadLine();
C11:
C12:        // beseda3 je abecedno pred beseda1
C13:        if(beseda1.CompareTo(beseda3) > 0) {
C14:           string pom = beseda1;
C15:           beseda1 = beseda3;
C16:           beseda3 = pom;
C17:        }
C18:
C19:        // beseda2 je abecedno pred beseda1
C20:        if(beseda1.CompareTo(beseda2) > 0) {
C21:           string pom = beseda1;
C22:           beseda1 = beseda2;
C23:           beseda2 = pom;
C24:        }
C25:
C26:        // beseda3 je abecedno pred beseda2
C27:        if(beseda2.CompareTo(beseda3) > 0) {
C28:           string pom = beseda2;
C29:           beseda2 = beseda3;
C30:           beseda3 = pom;
C31:        }
C32:
C33:        // Izpis po abecedi
C34:        Console.WriteLine("\n" + beseda1 + " " + beseda2 + " " + beseda3);
C35:     }
C36:  }
```

Zapis na zaslonu:

```
Vnesi prvo besedo: buda
Vnesi drugo besedo: vescina
Vnesi tretjo besedo: tempelj

buda tempelj vescina
```

6.1 Naloge za utrjevanje znanja iz nizov

1. Na svetovnem spletu uporabljamo več kodnih tabel znakov, od katerih samo nekatere vsebujejo šumevce (č, š, ž). Da se šumevci ne pretvorijo v nerazumljive znake, jih pretvorimo v ustrezne sičnike (c, s, z) in jih take tudi izpišimo. Napišite program, ki prebere vrstico besedila s šumevci in na zaslon izpiše spremenjeno besedilo.

2. Sestavite program, ki prebere niz in ga izpiše tako, da med znake besede vrine presledke.
   Primer: Prebran niz je `Lepa Anka kolo vodi.`
   `L e p a  A n k a  k o l o  v o d i .`

3. Napišite program Okvir, ki bo zahteval vnos niza. Nato ga bo izpisal na zaslon v "okvirju".
   Primer:
   Vnesi niz: `Mladost je norost, skače čez jarke, kjer je most.`
   ```
   *-------------------------------------------------*
   |Mladost je norost, skače čez jarke, kjer je most.|
   *-------------------------------------------------*
   ```

4. Sestavite program, ki bo zahteval vnos besede. Nato to besedo izpiše na zaslon tako, da bo beseda izpisana v obliki trikotnika.
   Primer: Vnesena beseda je `JEZIK`

   ```
   JEZIK
   JEZI
   JEZ
   JE
   J
   ```

5. Napiši program, ki bo prebral niz in preveril prvi znak. Če prvi znak ni črka, naj izpiše "*Niz se ne začne s črko.*" Če je prvi znak velika črka, naj izpiše "*Niz* niz *ima veliko začetnico*.". Če je prvi znak mala črka, naj niz spremeni, tako da bo prvi znak velika začetnica in izpiše: "*Vneseni niz* niz *ni imel velike začetnice. Po popravku zgleda tako: popravljen niz* niz.".

7 Tabele in zanka foreach

7.1 Tabele

Tabelo ali polje (ang. *array*) v jeziku java in C# uporabljamo v primerih, ko moramo v programu na enak način obdelati skupino vrednosti istega tipa.

V jeziku java smo spoznali naslednje načine ustvarjanja nove tabele:

1. *način.* Najprej najavimo ime tabele. Nato z operatorjem `new` dejansko ustvarimo tabelo.

Primer:

```
int[] tab1; // Vemo, da je tab1 ime tabele celih števil,
            // tabela kot taka pa še ne obstaja
tab1 = new int[20]; // tab1 kaže na tabelo 20 celih števil
```

2. *način.* Oba zgornja koraka združimo v enega.

Primera:

```
int[] tab1 = new int[2]; // Vsi elementi so samodejno nastavljeni na 0
double[] tab2 = new double[5]; // Vsi elementi so samodejno nastavljeni na
0.0
```

Tako v 1. in 2. načinu velja, da se, ko z `new` ustvarimo tabelo, vsi elementi samodejno nastavijo na začetno vrednost, ki jo določa tip tabele (npr. `int` - `0`, `double` - `0.0`, `boolean` - `false`).

3. *način.* Tabelo najprej najavimo, z operatorjem `new` zasedemo pomnilniško lokacijo, nato pa elementom določimo začetno vrednost.

Primer:

```
int[] tab3 = new int[]{-7, 5, 65}; // Elementi so -7, 5 in 65
```

4. *način.* Tabelo najprej najavimo, nato pa elementom določimo začetno vrednost. Velikost tabele ni potrebno podati, saj jo prevajalnik izračuna sam.

Primer:

```
char[] tab4 = {'a', 'b', 'c'}; // Elementi so 'a', 'b', in 'c'
```

V jeziku C# tabelo ustvarimo na enake načine.

Dolžino tabele določamo v jeziku C# z lastnostjo `Length`, ki ustreza javanski lastnosti `length`.

Vsak element, ki nastopa v tabeli, ima v javi in v C# svoj indeks. Indeksiranje se pri obeh jezikih prične z `0` in konča z `dolzina tabele - 1`.

Do `i`-tega elementa tabele v jeziku java dostopamo z izrazom `imeTabele[i-1]`. Ta način dostopanja uporabljamo tudi v C#.

Primer:

```
string[] tab = new string[]{"Mojca", "Peter", "Katarina"};
string element = tab[2];
tab[1] = "Ana";
```

Izraz, s katerim dostopamo do `i`-tega elementa tabele (`imeTabele[i]`), se v jeziku C# po videzu ujema z izrazom za dostopanje do `i`-tega znaka v nizu (`imeNiza[i]`). Izraza pa se ne ujemata v uporabi. Pri tabelah uporabljamo izraz za pridobitev in spremembo elementa tabele, medtem, ko ga pri nizih uporabljamo le za pridobitev znaka. V primeru, da izraz uporabimo za spremembo določenega znaka v nizu, nam prevajalnik vrne napako.

```
C:\Documents and Settings\Administrator\Desktop\VajeC#\Test12\Test12\
Program.cs(10,13): error CS0200: Property or indexer 'string.this[int]' cannot be
assigned to -- it is read only
```

Slika 28: Izpis napake, če želimo znak v nizu spremeniti z izrazom imeNiza[indeks].

Primer:

```
// Deklaracija tabele in niza
int [] tab = new int[]{1, 6, 40, 15};
string niz = "Trubarjevo leto 2008.";

// Pridobivanje
int el = tab[2]; // Vrednost spremenljivke je 6.
char z = niz[2]; // Vrednost spremenljivke je koda zanka 'u'.

// Spreminjanje
tab[1] = 5; // Spremenjena tabela je [1, 5, 40, 15].
niz[18] = '9'; // Izpiše se napaka.
```

Zgleda

Najdaljša beseda v nizu

Napišimo program, ki bo našel in izpisal najdaljšo besedo prebranega niza. Predpostavimo, da so besede ločene z enim presledkom.

Da bomo iz niza izluščili besede, si bomo pomagali z metodo `Split`, ki jo najdemo v razredu `String`. Metoda vrne tabelo nizov, v kateri vsak element predstavlja podniz niza, nad katerim smo metodo uporabili. V jeziku C# niz razmejimo na besede z razmejitvenim znakom oziroma znaki, ki jih ločimo z vejico (npr. `Split('/')`). Posamezen razmejitveni znak pišemo v enojnih narekovajih (`'`). V javi za razmejitev niza ne uporabljamo razmejitvenih znakov, temveč uporabimo razmejitveni niz (npr. `split("/")`), ki ga pišemo v navednicah (`"`). Metodo `Split` kot vedno v javi pišemo z malo začetnico.

Primer:
Denimo, da imamo

```
string stavek = "a/b/c.";
```

Če uporabimo

```
string[] tab = stavek.Split('/');
```

smo s tem ustvarili novo tabelo velikosti `3`. V tabeli so shranjeni podnizi niza `stavek`, kot jih loči razmejitveni znak `/`. Prvi element tabele `tab` je niz "`a`", drugi element je niz "`b`" in tretji element niz "`c.`".

Zapis v javi:

```
J1:   import java.io.*;
J2:   public class NajdaljsaBeseda{
J3:      public static void main(String[] args) throws IOException{
J4:         BufferedReader vhod = new BufferedReader(
                                    new InputStreamReader(System.in));
J5:         // Vnos niza
J6:         System.out.print("Vnesi niz: ");
J7:         String niz = vhod.readLine();
J8:
J9:         // Pretvorba niza v tabelo nizov
J10:        String[] tab = niz.split(" ");
J11:
J12:        // Iskanje najdaljše besede
J13:        String pomozni = ""; // Najdaljša beseda (oz. podniz)
J14:        for (int i = 0; i < tab.length; i++){
J15:           if (pomozni.length() < tab[i].length()) {
J16:              pomozni = tab[i];
J17:           }
J18:        }
```

```
J19:         // Izpis najdaljše besede
J20:         System.out.println("Najdaljsa beseda: " + pomozni );
J21:      }
J22:   }
```

Koda v jeziku C# sprememb praktično ne zahteva, razen tistih, ki smo jih že opisali v prejšnjih zgledih. "Nova" razlika je v ukazu za razmejitev niza in v metodi za določitev dolžine tabele, ko moramo v jeziku C# uporabiti `Split(' ')` namesto `split(" ")` in zapis `Length` namesto `length`.

Zapis v C#:

```
C1:   using System;
C2:   public class NajdaljsaBeseda{
C3:      public static void Main(string[] args){
C4:         // Vnos niza
C5:         Console.Write("Vnesi niz: ");
C6:         string niz = Console.ReadLine();
C7:
C8:         // Pretvorba niza v tabelo nizov
C9:         string[] tab = niz.Split(' ');
C10:
C11:        // Iskanje najdaljše besede
C12:        string pomozni = ""; // Najdaljša beseda (oz. podniz)
C13:        for (int i = 0; i < tab.Length; i++){
C14:           if (pomozni.Length < tab[i].Length){
C15:              pomozni = tab[i];
C16:           }
C17:        }
C18:
C19:        // Izpis najdaljše besede
C20:        Console.WriteLine("Najdaljsa beseda: " + pomozni );
C21:     }
C22:  }
```

Zapis na zaslonu:

```
Vnesi niz: Lep poletni dan.
Najdaljsa beseda: poletni
```

Razlaga. Najprej določimo niz `niz` (J7, C6). Nato s pomočjo klica metode `Split()` določimo tabelo `tab` (J10, C9). V metodi za razmejitveni znak uporabimo presledek (`' '`). Če je med besedami niza `niz` več presledkov, se vsi presledki obravnavajo kot razmejitveni znaki. Nato določimo pomožni niz `pomozni`, ki predstavlja trenutno najdaljšo besedo (J13, C12). Z zanko se sprehodimo po tabeli `tab` in poiščemo najdaljšo besedo. V vrstici J20 oziroma v vrstici C20 izpišemo najdaljšo besedo niza `niz` oziroma tabele `tab`.
Če za spremenljivko `niz` določimo prazen niz (v konzoli pri vnosu niza stisnemo le tipko Enter), se program izvede in za najdaljši niz izpiše prazen niz. Tabela `tab` je ostala prazna, zato je niz v spremenljivki `pomozni` ostal "".

Uredi elemente po velikosti

Sestavimo program, ki bo uredil vrednosti v tabeli celih števil po naslednjem postopku: Poiščimo najmanjši element in ga zamenjajmo s prvim. Nato poiščimo najmanjši element od drugega dalje in ga zamenjajmo z drugim, itn. Tabelo velikosti `n` preberemo in jo po urejanju izpišemo na ekran.

Sestavimo program po korakih:
Najprej deklariramo spremenljivko `n`, katere prebrano vrednost uporabimo za velikost tabele.

```
Console.Write("Velikost tabele: ");
int n = int.Parse(Console.ReadLine());
```

Nato deklariramo tabelo velikosti `n`.

```
int[] tab = new int[n];
```

Z zanko, ki se bo izvedla `n`-krat, napolnimo tabelo s števili, dobljenimi pri branju iz konzole.

```
Console.WriteLine();
for(int i = 0; i < n; i++){
   Console.Write("Vnesi " + (i+1)+ ". element: ");
   tab[i] = int.Parse(Console.ReadLine());
}
```

Nato naredimo zanko, ki se izvede `(n-1)`-krat. Ko nam bo ostal le zadnji element, bo že urejen. V zanki:

- Deklariramo pomožni spremenljivki. V prvi spremenljivki hranimo kandidata za najmanjši element med tistimi z indeksi od indeksa določenega z zanko do konca tabele. V drugi spremenljivki pa hranimo indeks kandidata, shranjenega v prvi spremenljivki.
- Naredimo zanko `for`, ki se sprehodi po tabeli od elementa, katerega indeks je določen s prvo zanko, do zadnjega elementa tabele.
  - Če najdemo manjši element, si zapomnimo njegovo vrednost in indeks.
- Izvedemo zamenjavo med trenutnim elementom in najdenim najmanjšim elementom.

```
for(int i = 0; i < n-1; i++){
   int min = tab[i]; // Kandidat za najmanjši element od indeksa i
                     // do konca tabele
   int indeks = i; // Indeks najmanjšega kandidata
   for(int k = i+1; k < n; k++){
      if (min > tab[k]){ // Poiščimo najmanjši element v tem delu
         min = tab[k];
         indeks = k;
      }
   }
   // Zamenjava elementov
   int t = tab[i];
   tab[i] = tab[indeks];
   tab[indeks] = t;
}
```

Na koncu z zanko `for`, ki se sprehodi po urejeni tabeli `tab`, izpišemo elemente in jih pri izpisu ločimo s presledkom.

```
Console.WriteLine();
for (int i = 0; i < n; i++){
   Console.Write(tab[i] + " ");
}
// Prehod v novo vrstico
Console.WriteLine();
```

Poglejmo še zapis celotnega programa.

Zapis v C#:

```
C1:   using System;
C2:   public class UrediTabelo{
C3:      public static void Main(string[] args){
C4:         // Vnos vrelikosti tabele
C5:         Console.Write("Velikost tabele: ");
C6:         int n = int.Parse(Console.ReadLine());
C7:
C8:         // Deklaracija tabele
C9:         int[] tab = new int[n];
C10:
C11:        // Napolnitev tabele
C12:        Console.WriteLine();
C13:        for(int i = 0; i < n; i++){
C14:           Console.Write("Vnesi " + (i+1)+ ". element: ");
C15:           tab[i] = int.Parse(Console.ReadLine());
C16:        }
C17:
C18:        // Uredimo tabelo
C19:        for(int i = 0; i < n-1; i++){
C20:           int min = tab[i]; // Kandidat za najmanjši element od indeksa i
```

```
C21:                                    // do konca tabele
C22:            int indeks = i; // Indeks najmanjšega kandidata
C23:            for(int k = i+1; k < n; k++){// Poiščimo najmanjši element
C24:                                         // v tem delu
C25:               if (min > tab[k]){
C26:                  min = tab[k];
C27:                  indeks = k;
C28:               }
C29:            }
C30:            // Zamenjava elementov
C31:            int t = tab[i];
C32:            tab[i] = tab[indeks];
C33:            tab[indeks] = t;
C34:         }
C35:
C36:         // Izpis urejene tabele
C37:         Console.WriteLine();
C38:         for (int i = 0; i < n; i++){
C39:            Console.Write(tab[i] + " ");
C40:         }
C41:         // Prehod v novo vrstico
C42:         Console.WriteLine();
C43:      }
C44:   }
```

Zapis na zaslonu:

```
Velikost tabele: 5

Vnesi 1. element: 6
Vnesi 2. element: 7
Vnesi 3. element: 3
Vnesi 4. element: 9
Vnesi 5. element: 1

1 3 6 7 9
```

7.2 Foreach

Zanka `foreach` je nam manj znana oblika zanke, ki pa jo poznata oba programska jezika. Zanko v nasprotju z že omenjenimi zankami uporabljamo le pri delu s tabelami in z zbirkami (Collections).

Pri spoznavanju jezika java smo zanko `foreach` zgolj omenili, zato si jo poglejmo podrobneje. Pri tem se bomo omejili le na uporabo skupaj s tabelami.

Splošna oblika zanke `foreach`:

```
for(tip spremenljivka : tabela){
   stavki;
}
```

Oblika zanke pomeni: "Ponavljaj stavke `stavki`, dokler za vrednost spremenljivke `spremenljivka` niso uporabljeni vsi elementi tabele `tabela`. Pri prvem prehodu zanke za vrednost spremenljivke `spremenljivka` uporabljaj prvi element tabele, drugi element tabele …"

Opomba: Podatkovni tip (`tip`) spremenljivke se mora ujemati s tipom elementa tabele. V primeru, da se tip spremenljivke in tip elementa ne ujemata, prevajalnik izpiše napako.

```
C:\Documents and Settings\Administrator\Desktop\VajeC#\Test.java:6: incompatible types
found   : char
required: java.lang.String
               for(String s : a){
                           ^
```

Slika 29: Izpis napake, če je spremenljivka s tipa String in element tabele a tipa char.

Glede na ujemanje tipov sta izjemi tega pravila sledeči:

1. primer. Podatkovni tip spremenljivke je `double`, tip elementa je `int`.

2. primer. Podatkovni tip spremenljivke je `int`, tip elementa je `char`.

Podrobneje bi delovanje `foreach` zanke opisali takole:

- Najprej se najavi spremenljivka `spremenljivka`.
- Zatem se najavi pomožna spremenljivka z začetno vrednostjo `0`. Določi se dolžina tabele.
- Nato se preveri, ali je vrednost pomožne spremenljivke manjša od dolžine tabele.
- Če je vrednost pomožne spremenljivke manjša od dolžine, se spremenljivki `spremenljivka` priredi element z indeksom, ki je enak vrednosti pomožne spremenljivke.
- Izvede se telo zanke. Vrednost pomožne spremenljivke se poveča za `1`.
- Nato se ponovno preveri, ali je vrednost pomožne spremenljivke manjša od dolžine tabele.
- Če je vrednost pomožne spremenljivke ponovno manjša od dolžine, se spremenljivki `spremenljivka` znova priredi element z indeksom, ki je enak vrednosti pomožne spremenljivke.
- Ponovno se izvede telo zanke. Vrednost pomožne spremenljivke se znova poveča za `1`.
- …
- Postopek se ponavlja toliko časa, dokler vrednost pomožne spremenljivke ni enaka dolžini tabele. Porabili smo vse elemente tabele, zato se zanka konča.

Vsako zanko `foreach` bi torej lahko nadomestili z naslednjim zapisom zanke `for`:

```
for(tip spremenljivka, int i = 0; i < tabela.length; i++){
   spremenljivka = tabela[i];
   stavki;
}
```

Oglejmo si značilnosti zanke `foreach`:

- V telesu zanke ne smemo spremeniti elementov tabele, po kateri se sprehajamo. Torej ne moremo na primer napisati

  ```
  tabela[2] = …;
  ```

  kot tudi ne

  ```
  spremenljivka = …;
  ```

- Zanko lahko uporabimo le za obdelavo cele tabele, ne pa tudi za obdelavo dela tabele.
- Iteracija zanke poteka vedno od indeksa `0` do indeksa `length - 1`. Pri tem si indeksi sledijo zapovrstjo (indeks `0`, indeks `1`, indeks `2`, … indeks `length - 1`).
- Znotraj zanke lahko uporabimo stavka `break` in `continue`.
- Zanko lahko uporabimo le na eni tabeli.
- V glavi zanke `foreach` lahko deklariramo le eno spremenljivko, katere vrednost je trenutni element tabele.

Zgled

Vsota elementov

Predpostavimo, da imamo tabelo z naslednjimi elementi: `2`, `5`, `7`, `1`, `6`, `10`, `3`, `8`, `0` in `11`. Napišimo program, ki bo določil in izpisal vsoto vseh elementov. Vsoto elementov izračunajmo s pomočjo zanke `foreach`.

Zapis v javi:

```
J1:   public class VsotaElementov{
J2:      public static void main(String[] args){
J3:         // Deklariramo tabelo in jo napolnimo z elementi
J4:         int[] tab = {2, 5, 7, 1, 6, 10, 3, 8, 0, 11};
J5:
J6:         // Vsota elementov
J7:         int vsota = 0;
J8:         // Ugotovitev vsote elementov tabele
J9:         for(int element : tab){
J10:           vsota = vsota + element;
J11:        }
J12:
J13:        // Izpis vsote elementov tabele
J14:        System.out.println("Vsota elementov je " + vsota + ".");
J15:     }
J16:  }
```

Zapis na zaslon:

```
Vsota elementov je 53.
```

Razlaga. Najprej deklarirajmo tabelo `tab` in jo napolnimo z elementi. Zatem deklariramo spremenljivko `vsota` in ji priredimo začetno vrednost `0`.
V J9. vrstici uporabimo zanko `foreach`, ki ponavlja stavek v vrstici J10 toliko časa, dokler v spremenljivki `element` niso uporabljeni vsi elementi tabele `tab`. V J10. vrstici seštevamo elemente tabele in njihovo vsoto hranimo v spremenljivki `vsota`. Po koncu zanke izpišemo skupno vsoto vseh elementov (J14).

Zanka `foreach` se v jeziku C# ne razlikuje veliko od enako imenovane zanke v javi. Razlika je več ali manj le v zapisu glave zanke.
Glava zanke ima v javi obliko `for(tip spremenljivka : tabela)`, medtem, ko ima v jeziku C# obliko `foreach(tip spremenljivka in tabela)`.

Popravimo program `VsotaElementov.java` tako, da bo deloval v jeziku C#. Spremembe bodo potrebne v glavi zanke, kjer moramo besedo `for` nadomestiti z besedo `foreach` in znak `:` z besedo `in`.

Zapis v C#:

```
C1:   using System;
C2:   public class VsotaElementov{
C3:      public static void Main(string[] args){
C4:         // Deklariramo tabelo in jo napolnimo z elementi
C5:         int[] tab = {2, 5, 7, 1, 6, 10, 3, 8, 0, 11};
C6:
C7:         // Vsota elementov
C8:         int vsota = 0;
C9:
C10:        // Ugotovitev vsote elementov tabele
C11:        foreach(int element in tab){
C12:           vsota = vsota + element;
C13:        }
C14:
C15:        // Izpis vsote elementov tabele
C16:        Console.WriteLine("Vsota elementov je " + vsota + ".");
C17:     }
C18:  }
```

Zapis na zaslonu:

```
Vsota elementov je 53.
```

Razlaga. Najprej deklarirajmo tabelo `tab` in jo napolnimo z elementi. Zatem deklariramo spremenljivko `vsota` in ji priredimo začetno vrednost `0`.

V vrstici C11 uporabimo zanko `foreach`, ki ponavlja stavek v vrstici C12 toliko časa, dokler v spremenljivki `element` niso uporabljeni vsi elementi tabele `tab`. V vrstici C12 seštevamo elemente tabele in njihovo vsoto hranimo v spremenljivki `vsota`. Po koncu zanke izpišemo skupno vsoto vseh elementov (C16).

Oglejmo si uporabo zanke `foreach` še na enem zgledu.

Manjkajoča števila

Generirajmo tabelo 50 naključnih števil med 0 in 100 (obe meji štejemo zraven). S pomočjo zanke `foreach` ugotovimo in izpišimo, katerih števil med 0 in 100 ni v tej tabeli. Za kontrolo naredimo izpis elementov tabele.

Sestavimo program po korakih:
Najprej deklariramo dve konstanti. Prvo uporabimo za velikost tabele, drugo pa za zgornjo mejo naključnih števil.

```
const int vel = 50; // Velikost tabele
const int mejaStevil = 100; // Zgornja meja naključnih števil
```

Nato deklariramo tabelo, ki bo ima velikost *vel*.

```
int[] tab = new int[vel];
```

Ustvarimo še generator naključnih števil.

```
Random nak = new Random(); // Generator naključnih števil
```

Z zanko, ki se izvede *vel*-krat, napolnimo tabelo z naključnimi števili med 0 in 100 (obe meji štejemo zraven):

```
for (int i = 0; i < vel; i++){
   tab[i] = nak.Next(mejaStevil + 1); // Določitev naključnega števila
}
```

Nato z zanko `foreach`, ki se sprehodi po elementih tabele *tab*, izpišemo elemente tabele. Elemente tabele ločimo s presledkom.

```
Console.WriteLine("Izpis vsebine tabele nakljucnih stevil: ");
// Izpis elementov tabele
foreach (int el in tab){
   Console.Write(el + " "); // Izpis elementa, ločenega s presledkom
}
```

Nato še enkrat naredimo zanko, ki se izvede 101-krat. V njej

- Ustvarimo logično spremenljivko za iskanje števila. Začetna vrednost spremenljivke je `false`.
- Naredimo zanko `foreach`, ki se sprehodi po elementih tabele `tab`. V zanki
  - Če število najdemo
    - vrednost logične spremenljivke nastavimo na `true` in
    - s stavkom `break` prekinemo iskanje.
- Če števila ne najdemo, ga izpišemo. Izpisana števila ločimo s presledkom.

```
Console.WriteLine("\n\nStevila med 0 in " + mejaStevil +
                        ", ki niso v tej tabeli, so: ");
// Iskanje števil, ki se ne nahajajo v tabeli
for (int i = 0; i < mejaStevil + 1; i++){
   bool nasli = false; // Spremenljivka za iskanje števil
   foreach (int el in tab){
      if(el == i){
         nasli = true; // Število smo našli
```

```
            break; // Prekinemo iskanje števila
        }
      }
      // Ali smo število našli, nam pove spremenljivka nasli
      if (!nasli) Console.Write(i + " ");
   }
```

Na koncu gremo še v novo vrstico.

```
   Console.WriteLine();
```

Poglejmo sedaj še zapis celotnega programa.

Zapis v C#:

```
C1:   using System;
C2:   public class Stevila{
C3:      public static void Main(string[] args){
C4:         const int vel = 50; // Velikost tabele
C5:         const int mejaStevil = 100; // Zgornja meja naključnih števil
C6:         int[] tab = new int[vel]; // Dekleracija tabele
C7:         Random nak = new Random(); // Generator naključnih števil
C8:
C9:         // Polnjenje tabele
C10:        for (int i = 0; i < vel; i++){
C11:           tab[i] = nak.Next(mejaStevil + 1); // Določitev naključnega števila
C12:        }
C13:
C14:        Console.WriteLine("Izpis vsebine tabele nakljucnih stevil: ");
C15:        // Izpis elementov tabele
C16:        foreach (int el in tab){
C17:           Console.Write(el + " "); // Izpis elementa, ločenega s presledkom
C18:        }
C19:
C20:        Console.WriteLine("\n\nStevila med 0 in " + mejaStevil +
                                 ", ki niso v tej tabeli, so: ");
C21:        // Iskanje števil, ki se ne nahajajo v tabeli
C22:        for (int i = 0; i < mejaStevil + 1; i++) {
C23:           bool nasli = false; // Spremenljivka za iskanje števil
C24:           foreach (int el in tab){
C25:              if (el == i){
C26:                 nasli = true; // Število smo našli
C27:                 break; // Prekinemo iskanje števila
C28:              }
C29:           }
C30:           // Ali smo število našli, nam pove spremenljivka nasli
C31:           if (!nasli) Console.Write(i + " ");
C32:        }
C33:
C34:        // Nova vrstica
C35:        Console.WriteLine();
C36:     }
C37:  }
```

V napisanem programu smo uporabili algoritem, katerega časovna zahtevnost je `O(velikost tabele x število vseh števil)`. Če pa si lahko "privoščimo" uporabo dodatne tabele take velikosti, kot je vseh možnih števil (v našem primeru 101), lahko razvijemo boljši (hitrejši) algoritem.

Za izboljšavo napisanega programa bomo uporabili pomožno tabelo, ki bo nadomestila tisti del "starega" programa, kjer smo uporabili gnezdeno zanko. V tabeli bomo vsem elementom, katerih indeksi so enaki izbranim številom, nastavili vrednosti na `true`. Po nastavitvi vrednosti bomo poiskali vse tiste elemente, ki so ohranili vrednost `false`. Kadar bomo našli tak element, bomo izpisali njegov indeks. Le ta bo neizbrano število.

Najprej iz programa `Stevila.cs` uporabimo vrstice od C4 do C18.

```
   const int vel = 50; // Velikost tabele
   const int mejaStevil = 100; // Zgornja meja naključnih števil
```

```
int[] tab = new int[vel]; // Dekleracija tabele
Random nak = new Random(); // Generator naključnih števil

// Polnjenje tabele
for (int i = 0; i < vel; i++){
   tab[i] = nak.Next(mejaStevil + 1); // Določitev naključnega števila
}

Console.WriteLine("Izpis vsebine tabele nakljucnih stevil: ");
// Izpis elementov tabele
foreach (int el in tab){
   Console.Write(el + " "); // Izpis elementa, ločenega s presledkom
}
```

Določimo tabelo logičnih vrednosti, katere velikost je `mejaStevil + 1` (vsa števila med 0 in 100, vključno z mejama).

```
bool[] pom = new bool[mejaStevil + 1]; // Pomožna tabela
```

Ob deklaraciji se vsi elementi tabele avtomatsko postavijo na `false`. S tem smo predpostavili, da podatek `i` ni v tabeli `tab`.

Nato naredimo zanko `foreach`, s katero v tabeli `pom` na `true` nastavimo tiste elemente, ki imajo indekse, ki so v tabeli `tab`.

```
// Označimo elemente, ki so v tabeli tab
foreach (int el in tab){
  pom[el] = true; // Vrednost elementa nastavimo na true
}
```

Sedaj moramo poskrbeti le še za izpis.
Zato naredimo zanko `for`, ki se sprehodi po tabeli `pom`.
Če je vrednost trenutnega elementa `false`, izpišemo njegov indeks. Indekse ločimo s presledkom.

```
Console.WriteLine("\n\nStevila med 0 in " + mejaStevil +
                        ", ki niso v tej tabeli, so: ");
// Iskanje števil, ki se ne nahajajo v tabeli
for(int i = 0; i < pom.Length; i++){
   if(pom[i] == false)
      Console.Write(i + " ");
}
// Nova vrstica
Console.WriteLine();
```

Poglejmo sedaj še zapis izboljšanega programa.

Zapis v C#:

```
C1:   using System;
C2:   public class Stevila1{
C3:      public static void Main(string[] args){
C4:         const int vel = 50; // Velikost tabele
C5:         const int mejaStevil = 100; // Zgornja meja naključnih števil
C6:         int[] tab = new int[vel]; // Dekleracija tabele
C7:         Random nak = new Random(); // Generator naključnih števil
C8:
C9:         // Polnjenje tabele
C10:        for (int i = 0; i < vel; i++){
C11:           tab[i] = nak.Next(mejaStevil + 1); // Določitev naključnega števila
C12:        }
C13:
C14:        Console.WriteLine("Izpis vsebine tabele nakljucnih stevil: ");
C15:         // Izpis elementov tabele tab
C16:         foreach (int el in tab){
C17:           Console.Write(el + " "); // Izpis elementa, ločenega s presledkom
C18:         }
C19:
C20:        bool[] pom = new bool[mejaStevil + 1]; // Pomožna tabela
```

```
C21:
C22:          // Označimo elemente, ki so v tabeli tab
C23:          foreach (int el in tab){
C24:            pom[el] = true; // Vrednost elementa nastavimo na true
C25:          }
C26:
C27:         Console.WriteLine("\n\nStevila med 0 in " + mejaStevil +
                                    ", ki niso v tej tabeli, so: ");
C28:         // Iskanje števil, ki se ne nahajajo v tabeli
C29:         for(int i = 0; i < pom.Length; i++){
C30:            if(pom[i] == false)
C31:               Console.Write(i + " ");
C32:         }
C33:         // Nova vrstica
C34:         Console.WriteLine();
C35:      }
C36:   }
```

Zapis na zaslonu:

```
Izpis vsebine tabele nakljucnih stevil:
2 43 5 85 17 32 97 72 72 23 18 95 4 86 42 51 1 9 14 57 79 99 37 85 72 63 17 67 43 92 47 57
 9 79 21 77 8 62 6 36 18 75 50 6 67 81 24 54 11 72

Stevila med 0 in 100, ki niso v tej tabeli, so:
0 3 7 10 12 13 15 16 19 20 22 25 26 27 28 29 30 31 33 34 35 38 39 40 41 44 45 46 48 49 52
53 55 56 58 59 60 61 64 65 66 68 69 70 71 73 74 76 78 80 82 83 84 87 88 89 90 91 93 94 96
98 100
```

Časovna zahtevnost programa `Stevila1.cs` je `O(max(dolžina tabele tab, dolžina tabele pom))`.

7.3 Naloge za utrjevanje znanja iz tabel

1. Napišite program, ki prebere tri cela števila n, a in b ter tabelo velikosti n napolni z naključnimi števili med a in b. Tabelo naj potem program izpiše tako, da v vrstici izpiše po 5 števil.

2. Sestavite program, ki bo prebral podatke v tabelo celih števil, nato pa preveril, ali tabela predstavlja permutacijo števil od 1 do n, kjer je n dolžina tabele.
   Namig: Tabela dolžine n predstavlja permutacijo, če v njej vsako naravno število od 1 do n nastopa natanko enkrat.

3. Dana je tabela [1, 6, 4 ,2, 8, 3, 7]. Uredite jo po naslednjem postopku: Poiščite največji element in ga zamenjajte z zadnjim. Nato poiščite največji element od prvega do predzadnjega in ga zamenjajte s predzadnjim …

4. Generirajte 1000 naključnih števil med 0 in 10 in preštejete, kolikokrat se vsak element pojavi.

5. V tabelo dolžine n zapišite naključne male črke angleške abecede. Sestavite novo tabelo, v kateri se vsak element prvotne tabele ponovi natanko dvakrat. Novo tabelo izpišite na zaslon.
   Primer:
   Če je prvotna tabela ['a', 'n', 'c'], je nova tabela ['a', 'a', 'a', 'n', 'n', 'n', 'c', 'c','c'].

6. V tabelo dolžine 30 zapišite naključna števila od 0 do 20 in jih izpišite kot množico: vsako samo enkrat.

7. Sestavite program, ki napolni tabelo velikost n z naključnimi nenegativnimi celimi števili. Prepišite te elemente v novo tabelo tako, da je i-ti element razlika med vsoto vseh elementov prvotne tabele in i-tim elementom prvotne tabele. Izpišite obe tabeli tako, da v vrsti izpišete po 10 elementov.

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

8 Metode

Programi, ki smo jih napisali do sedaj, so bili precej preprosti in kratki. Zato smo jih pisali v enem kosu znotraj glavne metode `Main` (C#) oziroma `main` (java). Programi so običajno večji, zato jih ne moramo kar v celoti napisati znotraj ene metode, saj bi bila taka metoda preobsežna, nepregledna in tudi manj razumljiva. Zato večje programe razdelimo na manjše kose in nato obravnavamo vsak kos posebej. V ta namen v javi in v C# uporabljamo metode, ki jim rečemo tudi funkcije. Prednosti uporab metod so še naslednje:

- uporabimo jih lahko večkrat,
- uporabimo jih lahko tudi v drugih programih.

V javi smo metode že spoznali. Vseeno pa si za ponovitev znanja zapišimo njihovo splošno definicijo:

```
dostop vrsta tip_rezultata ime_metode(parametri_metode){
   // Telo metode, ki se izvede,
   // ko metodo pokličemo
}
```

Pri spoznavanju metod v javi smo povedali, da obstajata dve vrsti metod, statične (*static*), ki jih kličemo na razredu in objektne metode, ki jih kličemo na objektu. Obe vrsti metod obstajata tudi v jeziku C#. V tej diplomski nalogi si bomo ogledali le statične metode.

V javi smo metode ločevali tudi po načinu dostopa na:

- javne (*public*) - dostop do metode imajo vsi razredi.
- zasebne (*private*) - dostop do metode ima samo razred, v katerem je metoda definirana.
- zaščitene (*protected*) - dostop do metode ima razred, v katerem je metoda definirana, njegovi podrazredi ter razredi iz istega paketa.

V jeziku C# se metode ločujejo po dostopu na:

- javne (*public*) - dostop je ekvivalenten istoimenskemu dostopu v javi.
- zasebne (*private*) - dostop je ekvivalenten istoimenskemu dostopu v javi.
- zaščitene (*protected*) - dostop do metode imajo poleg razreda, v katerem je metoda definirana, tisti razredi, ki dedujejo od razreda, v katerem so metode definirane.
- notranje (*internal*) - dostop do metode imajo razredi znotraj istega imenskega prostora.
- zaščiteno notranje (*protected internal*) - kombinacija zaščitenega (*protected*) in notranjega (*internal*) dostopa.

Ker bomo v nadaljnjih zgledih uporabljali le javen (public) dostop, se v natančnejšo razlago dostopov ne bomo spuščali.

Metode v jeziku C# se od metod v javi razlikujejo več ali manj le po dogovoru o njihovem imenu. Imena metod se v javi po dogovoru začnejo z malo tiskano črko, medtem, ko se v jeziku C# začnejo z veliko tiskano črko. Če se dogovora ne držimo, se ob prevajanju prevajalnik ne pritoži. Izjema je le metoda `Main` v C# oziroma `main` v javi, kjer je zahtevano, da se ta dogovor upošteva. V primeru, da za to metodo prekršimo dogovor, prevajalnik za jezik C# javi napako:

```
error CS5001: Program 'C:\Documents and Settings\Administrator\Desktop\VajeC#\Test12\
Test12\obj\Release\Test12.exe' does not contain a static 'Main' method suitable for
an entry point
```

Slika 30: Izpis napake v jeziku C#, če metodo 'Main' zapišemo kot 'main'.

Vsi parametri se v javi prenašajo po vrednosti. V jeziku C# pa imamo poleg tega še t.i. sklicne in izhodne parametre. V diplomski nalogi jih ne bomo obravnavali, navedimo pa le osnovno definicijo teh parametrov. Parametre metod v jeziku C# torej glede na način prenosa podatkov ob klicu delimo na:

a) vrednostne parametre

Vrednostni parametri so običajni ali klasični parametri. V metodo prenesejo le kopijo vrednosti originalnega argumenta, zato spreminjanje parametra v metodi ne vpliva na originalen argument.

b) sklicne (oziroma referenčne) parametre

Sklicni parametri omogočajo, da se v metode prenese sklic na vrednost originalnega argumenta. Zato spreminjanje parametra v metodi vpliva na vrednost originalnega argumenta. Ob klicu metode mora biti sklicni parameter inicializiran (mora imeti vrednost). Sklicni parameter najavimo s ključno besedo `ref`, ki jo uporabimo tudi pri samem klicu metode.

c) izhodne parametre

Izhodni parametri so podobni sklicnim parametrom. Razlikujejo se po tem, da je njihova začetna vrednost ob klicu metode nepomembna. Zato zanje ne velja, da morajo ob klicu že imeti vrednost. S temi parametri lahko prenašamo vrednosti le ven iz metode. Izhodni parametri mora biti najavljen s ključno besedo `out`. To navedemo ob ustrezni spremenljivki tudi pri samem klicu metode.

Sedaj pa si oglejmo nekaj zgledov, s katerimi bomo ilustrirali uporabo metod.

Dopolnjevanje niza

Napišimo metodo `DopolniNiz`, ki sprejme niz `s` in naravno število `n` ter vrne niz dolžine `n`. V primeru, da je dolžina niza `s` večja od `n`, spustimo zadnjih nekaj znakov. Drugače pa na konec niza `s` dodamo še ustrezno število znakov '`+`'. Preverimo delovanje metode.

Zapis v C#:

```
C1:     using System;
C2:     class Razred{
C3:        public static string DopolniNiz(string niz, int n) { // Metoda DopolniNiz
C4:           // Vrni nov niz dolžine n, ki ga dobimo tako, da bodisi skrajšamo
C5:           // niz niz, ali pa ga dopolnimo z znaki '+'.
C6:           int dolzina = niz.Length; // Določimo dolžino niza
C7:           string novNiz = ""; // Nov niz
C8:
C9:           if (dolzina > n){ // Dolžina niza je večja od n
C10:             novNiz = niz.Substring(0, n);
C11:          }
C12:          else { // Dolžina niza je manjša ali enaka n
C13:             novNiz += niz;
C14:             for(int i = dolzina; i < n; i++){
C15:             novNiz += '+'; // Dodamo mankajoče '+'
C16:             }
C17:          }
C18:
C19:          return novNiz; // Vrnemo nov niz
C20:       }
C21:       public static void Main(string[] args){ // Glavna metoda
C22:          Console.Write("Vnesi niz: ");
C23:          string niz = Console.ReadLine();
C24:          Console.Write("Vnesi n: ");
C25:          int n = int.Parse(Console.ReadLine());
C26:          Console.WriteLine("Nov niz: " + DopolniNiz(niz,n)); // Klic metode
C27:       }
C28:    }
```

Zapis na zaslonu:

```
Vnesi niz: Konec.
Vnesi n: 9
Nov niz: Konec.+++
```

Razlaga. Program začnemo z metodo `DopolniNiz`, ki ji v deklaraciji podamo parametra `niz` in `n` (C3). V metodi določimo dolžino niza `niz`, ki jo shranimo v spremenljivko `dolzina` (C6). Določimo nov niz `novNiz`, ki je na začetku prazen (C7). Preverimo pogoj v pogojnem stavku (C9). Če je izpolnjen (resničen), kličemo metodo `Substring()`, s katero izluščimo podniz dolžine `n` (C10). Rezultat te metode shranimo v spremenljivki `novNiz`. Če pogoj ni izpolnjen (neresničen), potem nizu `novNiz` dodamo niz `niz` (C13) in ustrezno število znakov '+' (C15). Te nizu dodamo s pomočjo zanke `for`. Na koncu metode s ključno besedo `return` vrnemo niz `novNiz` (C19).
Delovanje metode preverimo v glavni metodi `Main` (C21 - C27). V tej metodi preberemo podatka iz konzole, ju shranimo v spremenljivki `niz` in `n` ter s pomočjo klicane metode `DopolniNiz` (C26) izpišemo spremenjeni niz.

Poglejmo si še zgled rekurzivne metode, o kateri govorimo takrat, ko metoda kliče samo sebe. Kot vemo iz programskega jezika java, so rekurzivne metode sestavljene iz zaustavitvenega pogoja, ki ga uporabimo za prekinitev izvajanja metode in iz rekurzivnega dela, v katerem ponovimo klic metode. V jeziku C# se rekurzivne metode ne po obliki in ne po izvajanju razlikujejo od metod v javi. Zato si poglejmo zgled le v jeziku C#.

Fibonaccijevo zaporedje

Fibonaccijevo zaporedje je zaporedje, ki je sestavljeno tako, da je naslednji člen zaporedja enak vsoti prejšnjih dveh členov, prvi in drugi člen sta enaka 1. Zaporedje formalno zapišemo takole:

$$a_1 = 1,$$
$$a_2 = 1,$$
$$a_n = a_{n-1} + a_{n-2}, n \geq 3.$$

Sestavimo metodo `Fibonacci`, ki izračuna in vrne rezultat n-tega člena Fibonaccijevega zaporedja. Metodo napišimo rekurzivno.

Ideja.

- Pravili za člena a_1 in a_2 uporabimo za zaustavitveni pogoj.
- Pravilo a_n pa uporabimo za rekurzivni del metode.

Zapis v C#:

```
C1:    using System;
C2:    public class MetodaRekurzivna{
C3:       // Glavna metoda
C4:       public static void Main(string[] args){
C5:          Console.Write("Vnesi kateri clen zaporedja te zanima: ");
C6:          int clen = int.Parse(Console.ReadLine());
C7:          Console.WriteLine(Fibonacci(clen));
C8:       }
C9:       // Metoda za izračun Fibonaccijevega zaporedja
C10:      public static int Fibonacci(int n) {
C11:         // Zaustavitvena pogoja
C12:         if ((n == 1)|| (n == 2)) return 1;
C13:         // Rekurzivni del metode
C14:         return Fibonacci(n - 1) + Fibonacci(n - 2); // Klica metode
C15:      }
C16:   }
```

Zapis na zaslonu:

```
Vnesi kateri clen zaporedja te zanima: 8
21
```

Razlaga. Program je sestavljen iz dveh metod, metode `Main` (C4 - C8) in rekurzivne metode `Fibonacci` (C10 - C15). V metodi `Main` želimo izračunati vrednost določenega člena zaporedja. Kateri člen zaporedja nas zanima, shranimo v spremenljivki `clen`. Vrednost tega člena zaporedja se izračuna s pomočjo klica metode `Fibonacci` (C7). Da vrednost člena izračunamo, se v metodi `Fibonacci` izvrši več rekurzivnih klicev. Ti se izvršujejo toliko časa, dokler ni izpolnjen ustavitveni pogoj, oziroma dokler ne pridemo do računanja prvega ali drugega člena (C12). Podan člen zmanjšujemo znotraj rekurzivnega klica `Fibonaci(n - 1)` in znotraj klica `Fibonaci(n - 2)` (C14).

8.1 Naloge za utrjevanje znanja iz metod

1. Napišite metodo, ki bo izpisala vsa števila med 0 in 1000, katerih vsota števk je enaka številu, ki nastopa kot parameter te metode.

2. Denimo, da ste stari 16 let in imate zelo sitne starše, ki želijo brati vaša sporočila. Zato se s prijateljem dogovorita, da si bosta sporočila pošiljala zakodirana. Napisati morate metodo, ki bo sporočilo zakodirala. Zakodirano sporočilo naj bo sestavljeno najprej iz znakov, ki so bili v originalnem sporočilu na sodih mestih in nato iz znakov, ki so bili v originalnem sporočilu na lihih mestih.
   Primer:
   Originalno sporočilo `Greva na pijaco?`, se zakodira v sporočilo `Gean iaorv apjc?`.

3. Za lažje sporazumevanje s prijateljem sestavite še metodo, ki dekodira sporočilo, kodiramo po metodi prejšnje naloge.

4. Sestavite metodo `Razlika`, ki sprejme tabelo celih števil in vrne razliko med največjim in najmanjšim elementom v tabeli.
   Primer:
   Ko se izvedejo ukazi
   ```
   int[] tab = new []{5,1,2,4,6,8};
   int raz = Razlika(tab);
   ```
   ima `raz` vrednost `7`.

5. Sestavite metodo, ki kot parameter dobi tabelo in permutacijo (permutacija `1 3 2 4` pove, da prvi element ostane tam, kjer je, novi drugi element je stari tretji, tretji je stari drugi, četrti pa ostane tam kjer je). Vrnemo novo tabelo, ki ima elemente premešane tako, kot zahteva permutacija.

6. Napišite metodo, ki kot parameter dobi dve tabeli celih števil. Vrne naj novo tabelo, v kateri so tiste vrednosti, ki se pojavljajo v prvi tabeli in ne v drugi! Vemo, da so vse vrednosti v obeh tabelah med seboj različne (torej se v isti tabeli število ne ponovi).
   Primera:
   - Če so v prvi tabeli podatki `4`, `5`, `6` in v drugi `5`, `6`, `4`, naj metoda vrne prazno tabelo.
   - Če so v prvi tabeli podatki `4`, `5`, `6` in v drugi `1`, `2`, `4`, naj metoda vrne tabelo `5` in `6`.

7. Sestavite metodo `public static string Papajscina(string s)`, ki dani niz `s` pretvori v "`papajščino`". To pomeni, da za vsak samoglasnikom, ki se pojavi v nizu, postavi črko `p` in samoglasnik ponovi.
   Primer:
   Ko se izvedejo ukazi
   ```
   string s = "Danes je konec sole.";
   string papaj = Papajscina(s);
   ```
   je niz `papaj` enak "`Dapanepes jepe koponepec sopolepe.`".

8. Sestavite rekurzivno metodo, ki izračuna največji skupni delitelj dveh pozitivnih števil, če veste, da velja:

```
gcd(n,n) = n
gcd(n,k) = gcd(n-k,k) za n > k
gcd(n,k) = gcd(k,n) za n < k
```

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

9 Datoteke

V Slovarju slovenskega knjižnega jezika (SSKJ) piše:
datotéka -e ž (ẹ) elektr. urejena skupina podatkov pri (elektronskem) računalniku, podatkovna zbirka: vnesti nove podatke v datoteko / datoteka s programom.

Datoteka (ang. file) je zaporedje podatkov na računalniku, ki so shranjeni na disku ali kakem drugem pomnilniškem mediju (npr. CD-ROM, disketa, ključek). V datotekah hranimo različne podatke: besedila, preglednice, programe, ipd.

Glede na tip zapisa jih delimo na:

a) ***tekstovne*** datoteke
   Tekstovne datoteke vsebujejo nize znakov oziroma zlogov, ki so ljudem berljivi. Poleg teh znakov so v tekstovnih datotekah zapisani določeni t.i. kontrolni znaki. Najpomembnejša tovrstna znaka sta znaka za konec vrstice (end of line) in konec datoteke. Tekstovne datoteke so razdeljene na vrstice. Odpremo jih lahko v vsakem urejevalniku (npr. WordPad) ali jih izpišemo na zaslon.

   V tekstovnih datotekah so tudi vsi numerični podatki shranjeni kot zaporedje znakov. Zato jih moramo, če jih želimo uporabiti v aritmetičnih operacijah, spremeniti v numerične podatke.

b) ***binarne*** datoteke
   Binarne datoteke vsebujejo podatke zapisane v binarnem zapisu. Zato je vsebina le teh datotek ljudem neberljiva (urejevalniki besedil običajno ne znajo prikazati posebnih znakov, namesto njih prikazujejo "kvadratke"). Binarne datoteke ne vsebujejo vrstic.

   V binarne datoteke lahko shranimo vse, v programske jezike vgrajene numerične podatkovne tipe v ustrezni obliki. Zato so binarne datoteke bolj primerne za aplikacije, ki operirajo predvsem z numeričnimi podatki.

V tej diplomski nalogi se bomo ukvarjali le s tekstovnimi datotekami.

Imena datotek
Poimenovanje datoteke je odvisno od operacijskega sistema. V tej nalogi se bomo omejili na operacijske sisteme družine Windows.
Vsaka datoteka ima ime in je v neki mapi (imeniku). Polno ime datoteke dobimo tako, da zložimo skupaj njeno ime (t.i. kratko ime) in mapo.

Primer:
`D:\olimpijada\Peking\atletika.txt`
Če povemo z besedami: Ime datoteke je `atletika.txt` na enoti `D:`, v imeniku `Peking`, ki je v podimeniku imenika `olimpijada`.

Imena imenikov so v operacijskem sistemu Windows ločena z znakom \. Spomnimo se, da ima znak \, če ga v jeziku java in C# zapišemo kot sestavni del niza, poseben pomen. Šele v kombinaciji z naslednjim znakom predstavljajo nek znak (npr. \n - nova vrstica, \t - tabulator, \r - return). Zato, kadar želimo, da je znak \ sestavni del niza, moramo napisati kombinacijo \\. Spomnimo se, da v jeziku C# znak \ v nizu lahko izgubi posebni pomen, tako da pred nizom uporabimo znak @.

Knjižnica (imenski prostor)
Za delo z datotekami v jeziku C# so zadolžene metode iz razredov v imenskem prostoru `System.IO`. V javi smo za delo z datotekami potrebovali knjižnico `java.io`.

Kratica `io` oziroma `IO` predstavlja vhodne (*input*) in izhodne (*output*) tokove[7] (*streams*). S pomočjo njih lahko opravljamo želene operacije nad datotekami (npr. branje, pisanje). Zato na začetku vsake datoteke, v kateri je koda programa, ki dela z datotekami, navedemo uporabo omenjenega imenskega prostora. S tem poenostavimo klice teh metod (glej razdelek 2.2).

V podrobnosti glede tokov se ne bomo spuščali in bomo predstavili poenostavljeno zgodbo. Pisalni ali izhodni tok predstavimo s spremenljivko, ki je v javi tipa `PrintWriter` in v jeziku C# tipa `StreamWriter`. Bralni ali vhodni tok predstavimo s spremenljivko tipa `BufferedReader` (java) oziroma tipa `StreamReader` (C#).

9.1 Pisanje na tekstovno datoteko

Datoteke so nosilci različnih podatkov. Da lahko na tekstovno datoteko shranimo podatke, jo moramo najprej ustvariti. To storimo tako, da naredimo nov pisalni tok in ga povežemo z imenom, kot ga ima datoteka v operacijskem sistemu. To smo v javi naredili s stavkom

```
PrintWriter izhodniTok = new PrintWriter(new FileWriter(ime_datoteke));
```

V jeziku C# vlogo tega stavka prevzeme stavek

```
StreamWriter izhodniTok = File.CreateText(ime_datoteke);
```

Ko ustvarimo nov podatkovni tok, se pogosto zgodi, da pri tem izberemo ime že obstoječe datoteke. S tem povzročimo, da izgubimo že obstoječo vsebino. Če tega nočemo, preden odpremo pisalni tok, preverimo, ali datoteka z izbranim imenom že obstaja. To v javi storimo tako, da preverimo vrednost izraza `(new File(ime_datoteke).exists())`. V jeziku C# temu ustreza izraz `File.Exists(ime_datoteke)`.

Po končanem zapisu podatkov na datoteko podatkovni tok zapremo. To v javi storimo z metodo `close()`. V jeziku C# temu ustreza metoda `Close()`. Če podatkovnega toka po pisanju podatkov ne zapremo, je možno, da v nastali datoteki manjka del vsebine, oziroma vsebine sploh ni. Namreč, da napisan program pospeši operacije s diskom, se vsebina ne zapiše takoj v datoteko na disku, ampak v vmesni polnilnik. Šele ko je ta poln, se celotna vsebina medpomnilnika zapiše na datoteko. Metodi `close` v javi in `Close` v C# poskrbita, da je medpomnilnik izpraznjen tudi, če še ni povsem poln.

Za zapis podatkov na datoteko uporabljamo metodi, ki sta prikazani v naslednji tabeli. V prvem stolpcu tabele sta zapisani metodi jezika java, ki pripadata razredu `PrintWriter`. V drugem stolpcu sta zapisani ekvivalentni metodi jezika C#, ki pripadata razredu `StreamWriter`.

Java	C#	Razlaga
print()	***Write()***	Zapiše podatek na datoteko.
println()	***WriteLine()***	Zapiše podatek na datoteko in na koncu doda še znak za konec vrstice.

Tabela 7: Metode za zapisovanje podatkov na datoteko.

Oglejmo si sedaj nekaj zgledov programov, kjer pišemo na datoteke.

Osebni podatki

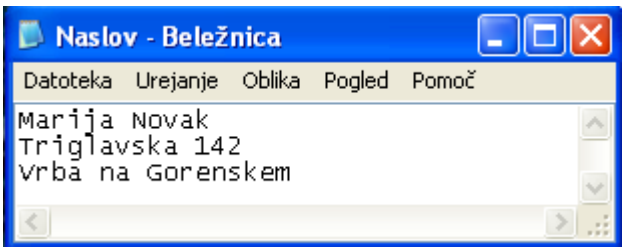
Na enoti `C` v korenskem imeniku ustvarimo mapo `Tekstovna`. V tej podmapi ustvarimo tekstovno datoteko `Naslov.txt` in vanjo zapišimo svoj naslov. To naredimo le, če datoteke še ni.

[7] Tok ponazarja potek informacij od enega objekta k drugemu objektu.

Zapis v C#:

```
C1:   using System;
C2:   using System.IO;
C3:   public class OsebniPodatki{
C4:      public static void Main(string[] args){
C5:         // Pot in ime
C6:         string ime = @"C:\Tekstovna\Naslov.txt";
C7:
C8:         // Preverimo obstoj datoteke
C9:         if(File.Exists(ime)){
C10:           Console.WriteLine("Datoteka ze obstaja.");
C11:           return;
C12:        }
C13:
C14:        // Ustvarimo novo datoteko
C15:        StreamWriter dat = File.CreateText(ime);
C16:
C17:        // Zapišemo osebne podatke
C18:        dat.WriteLine("Marija Novak");
C19:        dat.WriteLine("Triglavska 142");
C20:        dat.WriteLine("Vrba na Gorenskem");
C21:
C22:        // Zapremo datoteko za pisanje
C23:        dat.Close();
C24:     }
C25:  }
```

Zapis na datoteki Naslov.txt:



Razlaga. Najprej določimo niz, ki predstavlja polno ime datoteke (vrstica C6). Če bi za niz določili le kratko ime, bi se datoteka ustvarila tam, kjer bi izvedli program `OsebniPodatki.cs`. V vrstici C9 preverimo obstoj datoteke. Če datoteka obstaja, izpišemo obvestilo (vrstica C10) in končamo izvajanje programa (vrstica C11). Nato v vrstici C15 odprem datoteko za pisanje. S klicem metode `dat.WriteLine()` podatke zapišemo na datoteko (vrstice C18 - C20). Po končanem zapisu datoteko zapremo (vrstica C23). Če tega ne bi storili, datoteka ne bi imela vsebine.

Števila

V tekstovno datoteko `Stevila.txt` zapišimo prvih `n` sodih števil, ki niso deljiva s šest. Števila izpišimo ločena z znakom '`/`', torej na primer kot:

```
1/2/3/8/14/16/20/22/
```

Ideja.
Napišemo metodo, ki ima dva parametra:

- *ime* - ime datoteke
- *n* - koliko števil bo v datoteki

Rezultat metode bo *void*, saj ima metoda le učinek ustvarila bo datoteko.

```
private static void Stevila(string ime, int n)
```

Najprej preverimo, če datoteka *ime* obstaja. V primeru obstoja izpišimo opozorilo in prekinimo izvajanje metode. Lepše bi bilo, če bi namesto izpisa obvestila metoda vrgla izjemo. A ker se z izjemami v jeziku C# ne bomo ukvarjali, bomo napako javili z izpisom obvestila.

```
if (File.Exists(ime)){
   Console.WriteLine("Napaka.");
   return;
}
```

Nato ustvarimo datoteko in jo povežemo s tokom za pisanje.

```
StreamWriter dat;
dat = File.CreateText(ime);
```

Ustvarimo še števec za štetje v datoteko zapisanih števil in spremenljivko, ki bo hranila tekoče sodo število.

```
int stevec = 1;
int soda = 0;
```

Nato naredimo zanko, ki se bo izvajala toliko časa, dokler ne bo *stevec* postal večji kot *n*. V zanki:

- Preverimo, če vrednost spremenljivke *soda* ni deljiva s *6*.
  - Če je pogoj izpolnjen, vrednost spremenljivke *soda* zapišemo na datoteko.
  - Povečamo spremenljivko *stevec* za 1.
- Spremenljivko *soda* nastavimo na naslednje sodo število..

```
while (stevec <= n){
   if (soda % 6 != 0){
      dat.Write(soda + "/");
      stevec++;
   }
   soda = soda + 2;
}
```

Zatem zapremo podatkovni tok.

```
dat.Close();
```

Na koncu še napišemo metodo *Main*. V metodi:

- napišemo ime datoteke,
- preberemo število *n* in
- pokličemo metodo *Stevila*.

```
public static void Main(string[] args){
   // Ime datoteke
   string datoteka = "Stevilo.txt";
   // Število števil v datoteki
   Console.Write("Vnesi n: ");
   int n = int.Parse(Console.ReadLine());
   Stevila(datoteka,n); // Klic metode
}
```

Sestavimo zapisane dele programa v celoto.

Zapis v C#:

```
C1:   using System;
C2:   using System.IO;
C3:   public class Program{
C4:      private static void Stevila(string ime, int n){
C5:         // Preverjanje obstoja datoteke
C6:         if (File.Exists(ime)){
C7:            Console.WriteLine("Napaka.");
C8:            return;
C9:         }
C10:
```

```
C11:         StreamWriter dat; // Ustvarimo tok za pisanje na datoteko
C12:         dat = File.CreateText(ime);
C13:
C14:         // Pomožni spremenljivki
C15:         int stevec = 1;
C16:         int soda = 0;
C17:
C18:         // Zapisujemo števila na datoteko
C19:         while (stevec <= n){
C20:            if (soda % 6 != 0){
C21:               dat.Write(soda + "/");
C22:               stevec++;
C23:            }
C24:            soda = soda + 2;
C25:         }
C26:
C27:         // Zapremo datoteko za pisanje
C28:         dat.Close();
C29:      }
C30:
C31:      public static void Main(string[] args){
C32:         // Ime datoteke
C33:         string datoteka = "Stevilo.txt";
C34:         // Število števil v datoteki
C35:         Console.Write("Vnesi n: ");
C36:         int n = int.Parse(Console.ReadLine());
C37:         Stevila(datoteka,n); // Klic metode
C38:      }
C39:   }
```

9.2 Branje iz tekstovne datoteke

Podatke na datoteke shranjujemo za kasnejšo uporabo. Če želimo podatke iz tekstovne datoteke prebirati, moramo datoteko odpreti v načinu za branje. To v javi storimo s stavkom

```
BufferedReader vhodTok = new BufferedReader(new FileReader(ime_datoteke));
```

Vlogo navedenega javanskega stavka v jeziku C# prevzeme stavek

```
StreamReader vhodTok = File.OpenText(ime_datoteke);
```

Pri odpiranju datotek za branje pogosto navedemo ime neobstoječe datoteke. To povzroči napako med izvajanjem programa. Napako znamo preprečiti tako v javi kot tudi v C#. Spomnimo se izraza `File.Exists(ime_datoteke)` v C# in ekvivalentnega izraza `(new File(ime_datoteke)).exists()` v javi. V razdelku o pisanju smo jih uporabili zato, da nismo ustvarili datoteke z enakim imenom, kot jo ima že obstoječa datoteka. Torej je tudi pred odpiranjem datoteke za branje smiselno, da prej preverimo, ali datoteka z določenim imenom obstaja.

Da iz datoteke preberemo podatke, so na voljo metode, ki so prikazane v spodnji tabeli. V prvem stolpcu tabele so zapisane metode v jeziku java. Najdemo jih v razredu `BufferedReader`. V drugem stolpcu tabele so zapisane ustrezne metode jezika C#, ki jih najdemo v razredu `StreamReader`.

Java	C#	Razlaga
readLine()	***ReadLine()***	Prebere naslednjo vrstico podatkov z datoteke in jo vrne kot niz.
read()	***Read()***	Prebere naslednji razpoložljiv znak z datoteke. ***Pozor:*** Metoda vrne kodo prebranega znaka.
-	***ReadToEnd()***	Prebere podatke v datoteki od trenutnega mesta v datoteki pa do konca. Podatke vrne kot niz. Navadno to metodo uporabljamo, da preberemo celotno vsebino datoteke.

ready()	-	Vrne logično vrednost, ki pove, ali je v datoteki še kakšna vrstica za branje ali ne.

-	***Peek()***	Vrne naslednji znak v datoteki, brez premika na naslednji znak. Če ni na voljo nobenega znaka več, metoda vrne vrednost `-1`.

Tabela 8: Metode pridobitev podatkov iz datoteke.

Odstranimo prazne vrstice

Na datoteki je pesem zapisana po kiticah. Med kiticami so prazne vrstice. Napišimo metodo, ki za dano ime datoteke vse kitice izpiše na zaslon. Pri tem prazne vrstice izpusti.

Vsebina datoteke `Dekle.txt`:	Izpis na zaslon:
Dekle je po vodo šlo *na visoke planine.* *Vodo je zajemala,* *je ribico zajela.* *Ribica jo je prosila:* *oj, pusti me živeti.* *Dekle b'la je usmiljena,* *je ribico spustila.* *Ribica je zaplavala,* *je dekle poškropila.*	*Dekle je po vodo šlo* *na visoke planine.* *Vodo je zajemala,* *je ribico zajela.* *Ribica jo je prosila:* *oj, pusti me živeti.* *Dekle b'la je usmiljena,* *je ribico spustila.* *Ribica je zaplavala,* *je dekle poškropila.*

Zapis v C#:

```
C1:   public static void Dekle(string ime){
C2:      // Preverimo obstoj datoteke
C3:      if(!File.Exists(ime)){
C4:         Console.WriteLine("Datoteka " + ime + "ne obstaja.");
C5:         return;
C6:      }
C7:
C8:      // Odpremo datoteko za branje
C9:      StreamReader dat = File.OpenText(ime);
C10:
C11:     // Beremo in izpisujemo (neprazne) vrstice datoteke
C12:     while(dat.Peek() != -1){ // Beremo, dokler ne preberemo kode oznake EOF
C13:        string vrstica = dat.ReadLine();
C14:        if(vrstica.Length != 0){ // Prebrana vrstica ni prazna
C15:          Console.WriteLine(vrstica);
C16:        }
C17:     }
C18:
C19:     // Zapremo datoteko za branje
C20:     dat.Close();
C21:  }
```

Razlaga. Najprej preverimo obstoj datoteke (C3). Če datoteka ne obstaja, izpišemo obvestilo (C4) in prekinemo izvajanje metode (C5). Odpremo datoteko za branje (C9). V vrstici C12 naredimo zanko, ki ponavlja vrstice od C13 do C15 toliko časa, dokler ne pridemo do konca datoteke. To prepoznamo po tem, da je na vrsti za branje znak EOF[8]. Ta znak ima kodo -1. V

[8] Kratica EOF pomeni End-Of-File oziroma konec datoteke.

C13. vrstici beremo posamezno vrstico datoteke. Če je dolžina vrstice različna od 0 (če torej vrstica ni prazna), vrstico izpišemo (C15). Po končanem branju datoteko zapremo (C20).
Zadnji znak datoteke je oznaka EOF, katerega koda je -1. Ko je na vrsti znak s to kodo, vemo, da smo na koncu besedila, zapisanega na datoteki. Oznake EOF ob odprtju datoteke ne vidimo.

Opomba: Datoteke, s katerih v javi in v C# beremo, se spodobi zapreti. Sicer neposrednih posledic (kot pri pisanju) ne bo. A če bomo odprto datoteko poskusili odpreti ponovno, se bo program sesul. Prav tako odprte datoteke trošijo sistemske vire računalnika. Zato je dobro, da se navadimo datoteke, potem ko jih ne potrebujemo več, vedno zapreti.

Odstranitev števk

Sestavimo metodo `PrepisiBrez`, ki sprejme imeni vhodne in izhodne datoteke. Metoda na izhodno datoteko prepiše tiste znake iz vhodne datoteke, ki niso števke. Predpostavimo, da sta imeni datotek ustrezni (torej, da datoteka za branje obstaja, datoteka za pisanje pa ne (oziroma da njeno morebitno prejšnjo vsebino lahko izgubimo)).

Koraki programa:

- Najprej bomo odprli ustrezni datoteki. Prvo datoteko odpremo za branje in drugo za pisanje.
- Brali bomo posamezne znake iz vhodne datoteke.
  - Če prebran znak ne bo števka, ga bomo prepisali v izhodno datoteko.
- To bomo počeli toliko časa, dokler ne bomo prebrali znaka s kodo -1 (torej znak EOF).

Zapis v C#:

```
C1:   public static void PrepisiBrez(string imeVhod, string imeIzhod){
C2:      StreamReader datZaBranje; // Ustvari podatkovni tok za branje na datoteki
C3:      datZaBranje = File.OpenText(imeVhod);
C4:      StreamWriter datZaPisanje; // Ustvari tok za pisanje na datoteko
C5:      datZaPisanje = File.CreateText(imeIzhod);
C6:
C7:      // Prepis znakov iz ene datoteke na drugo datoteko
C8:      int znak = datZaBranje.Read(); // Prebere en znak
C9:      while (znak != -1){
C10:        // Primerjamo ali je prebrani znak števka
C11:        if (!('0' <= znak && znak <= '9'))
C12:           datZaPisanje.Write("" + (char)znak); // Če ni števka, zapišemo znak
C13:        znak = datZaBranje.Read();
C14:     }
C15:
C16:     // Zapremo tokova datotek
C17:     datZaBranje.Close();
C18:     datZaPisanje.Close();
C19:  }
```

Primerjava vsebine dveh datotek

Napišimo metodo `Primerjaj`, ki sprejme imeni dveh datotek in primerja njuno vsebino. Če sta vsebini datotek enaki, metoda vrne vrednost `true`, sicer vrne `false`. Predpostavimo, da sta imeni datotek ustrezni (torej, da datoteki za branje obstajata).

Zapis v C#:

```
C1:   public static bool Primerjava(string ime1, string ime2){
C2:      // Odpremo obe datoteki za branje
C3:      StreamReader dat1 = File.OpenText(ime1);
C4:      StreamReader dat2 = File.OpenText(ime2);
C5:
C6:      // Preberemo vsebino prve in druge datoteke
C7:      string beri1 = dat1.ReadToEnd();
C8:      string beri2 = dat2.ReadToEnd();
C9:
C10:     // Zapremo obe datoteki za branje
C11:     dat1.Close();
C12:     dat2.Close();
C13:
C14:     // Primerjamo vsebini
```

```
C15:     return (beri1.Equals(beri2));
C16:  }
```

Razlaga. Najprej odpremo datoteki (C3 – C4). V niz `beri1` s pomočjo metode `ReadToEnd()` preberemo celotno vsebino datoteke `dat1` (C7). V niz `beri2` shranimo celotno vsebino datoteke `dat2` (C8). Zapremo datoteki (C11 – C12). Na koncu vrnemo rezultat, ki ga dobimo pri primerjavi niza `niz1` z nizom `niz2`.

V javi metode `ReadToEnd()` ne poznamo. Zato bi morali vrstici C7 in C8 v javi nadomestiti z deli, kjer bi spremenljivki `beri1` in `beri2` napolnili z ustrezno zapisanimi zanki `while`.

Zamenjava

Napišimo program, ki preko tipkovnice prebere ime vhodne datoteke in ime izhodne datoteke. Nato vhodno datoteko prepiše na izhodno, pri čemer vse znake 'a' nadomesti z nizom "`apa`".

Ideja programa:

- preberemo ime vhodne in izhodne datoteke,
- preverimo obstoj datotek,
- odpremo vhodno datoteko za branje in izhodno datoteko za pisanje,
- v zanki izpisujemo znake iz vhodne datoteko v izhodno datoteko. Če je znak '`a`', ga nadomestimo z "`apa`",
- vhodno in izhodno datoteko zapremo.

Zapis v C#:

```
C1:   using System;
C2:   using System.IO;
C3:   public class Zamenjava{
C4:      public static void Main(string[] args){
C5:         // Vnos imen datotek
C6:         Console.Write("Vnesi ime vhodne datoteke: ");
C7:         string vhod = Console.ReadLine();
C8:         Console.Write("Vnesi ime izhodne datoteke: ");
C9:         string izhod = Console.ReadLine();
C10:
C11:        // Preverjanje obstoja: vhodna mora obstajati, izhodna pa ne
C12:        if (!File.Exists(vhod) || File.Exists(izhod)) {
C13:           Console.WriteLine("Napaka v imenu datotek.");
C14:           return;
C15:        }
C16:
C17:        // Odprtje datotek
C18:        StreamReader branje = File.OpenText(vhod);
C19:        StreamWriter pisanje = File.CreateText(izhod);
C20:
C21:        // Prenos podatkov
C22:        while (branje.Peek() != -1) { // Do konca datoteke
C23:           char znak = (char)branje.Read();
C24:           if (znak == 'a') {
C25:              pisanje.Write("ap"); // Zadnji 'a' bo dodal naslednji stavek
C26:           }
C27:           pisanje.Write(znak);
C28:        }
C29:
C30:        // Zapremo za datoteke
C31:        branje.Close();
C32:        pisanje.Close();
C33:     }
C34:  }
```

Razlaga. Preberemo ime vhodne in izhodne datoteke (C6 - C9). Nato preverimo obstoj datotek (C12). Če vhodna datoteka ne obstaja ali izhodna datoteka obstaja, izpišemo obvestilo (C13) ter prekinemo izvajanje metode (C14). Datoteki odpremo za branje oziroma pisanje (C18 - C19). Nato z zanko `while` beremo posamezne znake iz vhodne datoteke in jih prepisujemo na izhodno datoteko (C22 - C28). Če je prebran znak '`a`' (C24), pred njim na izhodno datoteko zapišemo še niz "`ap`" (C25). Znak '`a`' iz vhodne datoteke na ta način zapišemo na izhodno datotako z nizom

"apa". Zanko končamo, ko preberemo kodo oznake EOF. Po koncu prepisovanja znakov datoteki zapremo (C31 - C32).

Kopiranje datotek

Sestavimo metodo `public static void Kopiranje(string vhod, string izhod)`, ki vsebino ene datoteke prepiše na drugo datoteko. Ime prve datoteke naj predstavlja prvi argument, ime druge datoteke pa naj predstavlja drugi argument. Predpostavimo, da je obstoj datotek že preverjen. Kopiranje izvedemo tako, da prepisujemo vrstico po vrstico.

Da preverimo, če smo že na koncu datoteke, lahko uporabimo tudi rezultat metode `ReadLine()`. Če omenjeno metodo uporabimo na neobstoječi vrstici (če smo torej že na koncu datoteke), nam metoda vrne vrednost `null`.

Zapis v C#:

```
C1:   public static void Kopija(string vhod, string izhod){
C2:      // Odpremo datoteko za branje in pisanje
C3:      StreamReader beri = File.OpenText(vhod);
C4:      StreamWriter pisi = File.CreateText(izhod);
C5:
C6:      // Beremo in kopiramo posamezne vrstice
C7:      string vrsta = beri.ReadLine();
C8:      while(vrsta != null){// Beremo toliko časa, dokler ne preberemo
C9:                           // vseh vrstic
C10:        pisi.WriteLine(vrsta);
C11:        vrsta = beri.ReadLine();
C12:     }
C13:
C14:     //Zapremo datoteko za branje in pisanje
C15:     beri.Close();
C16:     pisi.Close();
C17:  }
```

Razlaga. Najprej datoteki odpremo za branje oziroma pisanje (C3 - C4). Nato z zanko `while` beremo posamezne vrstice vhodne datoteke in jih prepisujemo v izhodno datoteko (C7 - C12). Zanko končamo, ko prebrane vrstice ni (ustrezni niz dobi vrednost `null`). Po končani zanki datoteki zapremo (C15 - C16).
Če naloga ne bi zahtevala prepisovanje po vrsticah, bi seveda lahko naenkrat prebrali kar celotno vsebino datoteke. Prepisovanje pa bi lahko izvedli tudi znak po znak.

9.3 Naloge za utrjevanje znanja iz datotek

1. Napišite program, v katerem tekstovno datoteko napolnite s 100 poljubnimi celimi števili. Ta števila potem preberite iz datoteke in izpišite povprečje vseh lihih.

2. Napišite program `Sestej`, ki prebere ime datoteke, v kateri so zapisana cela števila, vsako v svoji vrsti. Program naj na zaslon izpiše vsoto števil iz datoteke. Denimo, da je v datoteki `Stevila.txt` zapisano

   ```
   10
   4
   100
   12
   ```

   Spodnji okvir ponazarja izvajanje programa:

   ```
   Ime datoteke: Sestej.txt
   Vsota: 126
   ```

3. Napišite program, ki prebere imeni vhodne in izhodne datoteke ter vhodno datoteko prepiše v izhodno. Pri tem naj vse pojavitve znaka 'e' zamenja z znakom 'E'.

4. Napišite program `VrednostPolinoma`, ki iz datoteke "`polinom.txt`" prebere celo število `t` in celoštevilske vrednosti polinoma $p(x) = a_0 + a_1x + \ldots + a_nx^n$ ter izpiše na zaslon vrednost `p(t)`. Datoteka vsebuje dve vrstici: v prvi vrstici je zapisano celo število `t`, v drugi vrstici pa so koeficienti polinoma `p`, ločeni s presledki.

   Primer:
   Zapis v datoteki
   ```
   2
   1 2 0 3 4
   ```
   Za ta primer mora program izpisati na zaslon število `29`, ker podatki predstavljajo `t=2` in polinom $p(x)=1+2x+4x^3+3x^4$, od koder dobimo $p(t)=1+2\cdot2+4\cdot2^3+3\cdot2^4=29$.

5. Tekstovna datoteka naj bo sestavljena iz vrst, napolnjenih s celimi števili, ločenimi s presledki. Sestavite statično metodo `MinMax(imeDatoteke)`, ki vrne maksimum minimumov števil v vsaki vrstici v datoteki.

   Primer:
   Če je v datoteki `Stevila.txt` zapisano

   ```
   2 3 4
   5 -1
   8 9 6 3
   ```

   je rezultat metode `MinMax("Stevila.txt")` enak `3`.

6. Ustvarite tekstovno datoteko `APJ.txt`. V to datoteko zapišite poljubno število stavkov (berete jih preko tipkovnice, znak za konec vnosa je `*`). Vsak stavek naj bo na datoteki v svoji vrstici, med vrsticami pa naj bo po ena prazna vrstica. Napišite metodo, ki dobi za parameter ime te datoteke. Vrne naj, koliko znakov vsebuje celotna datoteka! Prazna vrstica seveda ne vsebuje nobenega znaka.

7. Dana je tekstovna datoteka `Naloga.txt`. V vsaki vrstici te datoteke so po tri cela števila, med seboj ločena s presledkom. Datoteko obdelajte tako, da za vsako vrstico na zaslon izpišete vsoto vseh treh števil, na koncu pa še skupno vsoto vseh števil!

8. V tekstovni datoteki so zapisani podatki o porabi bencina za posamezne tipe vozila. V vsaki vrstici je zapisan tip vozila, nato pa podatek o porabi goriva na 100 km (realno število). Koliko vozil je v datoteki? Ugotovite in izpišite tip vozila z najmanjšo porabo goriva. Podatka o tipu vozila in porabi goriva sta razmejena z ločilnim znakom |.

9. Napišite program, ki v poljubni tekstovni datoteki prešteje vse števke in na koncu izpiše, kolikokrat se vsaka števka pojavi v tej datoteki. Ime datoteke programu podamo preko konzolnega okna.

10. V datoteki `realna.txt` so zapisana realna števila (vsako v svoji vrstici). Napišite program, ki ugotovi in izpiše, koliko je vseh števil v datoteki in kakšen je procent števil 0.

    Primer izpisa:
    ```
    V datoteki je 20 stevil, od tega 25 procentov nicel.
    ```

10 Zaključek

Jezik C# sem sama spoznala med praktičnem usposabljanjem. Zdel se mi je zanimiv in hkrati vsaj tako uporaben kot jezik java, katerega znanje sem si pridobila že v času študija. Zaradi tega sem se odločila, da ga predstavim tudi preostalim absolventom praktične matematike.

Ker absolventi praktične matematike že imajo pridobljeno znanje jave, sem jezik C# predstavila tako, da sem primerjala značilnosti obeh jezikov med seboj in jih ponazorila tudi z zgledi. V predstavitvi sem zajela le tiste značilnosti, ki bi absolventa zanimale ob prvem srečanju z novim jezikom. Mednje spadajo predvsem osnovne značilnosti, kot so zanke, odločitve, nizi, tabele … Nisem opisovala ustvarjanja razredov kot tudi ne številnih že pripravljenih metod, knjižnic … in prijemov, kot so na primer izjeme in podobno. V diplomski nalogi tudi nisem omenila novih značilnosti jezika C#, predvsem tistih ne, ki so bile uvedene v začetku tega leta, v različici C# 3.0.

Najpomembnejša novost različice C# 3.0 je LINQ (Language Integrated Query), ki je namenjena za delo s podatkovnimi zbirkami. LINQ je prenos idej jezika SQL (Structured Query Language) v sam programski jezik. To omogoča preverjanje pravilnosti sintakse dela z bazami podatkov že na nivoju prevajanja. Poleg večje hitrosti to preprečuje tudi določene varnostne probleme. Poleg vpeljave LINQ so v novi različici vpeljane še novosti, kot so anonimni podatkovni tipi, delne metode, razširljive metode, izrazi lambda, inicializacija objektov in zbirk ter implicitni tip krajevne spremenljivke.

Z diplomsko nalogo sem želela absolventu praktične matematike zadovoljiti željo po spoznavanju programskega jezika C#. Za nadalnje spoznavanje tega jezika pa bo moral poskrbeti sam.

11 Literatura in viri

1. Bauer A., *Računalništvo1 (PRA)*. Dostopno na naslovu
http://andrej.com/java/lekcije.html
(zadnji obisk: 3.9.2008)

2. Bragnall B., *C# for Java Programmers*, Rockland (MA), Syngress, 2002

3. C# Online.NET. Dostopno na naslovu
http://en.csharp-online.net/CSharp_Certification%2C_Development%2C_and_Training
(zadnji obisk: 3.9.2008)

4. *Download the Visual Studio 2008 Express Edition*. Dostopno na strani
http://www.microsoft.com/Express/Download/
(zadnji obisk: 3.9.2008)

5. Harding University, *Java and C# Comparison*. Dostopno na naslovu
http://www.harding.edu/fmccown/java_csharp_comparison.html
(zadnji obisk: 3.9.2008)

6. Jagodic S., LAV - laboratorijske vaje, Srednja elektro in strojna šola Kranj, 2000
http://www.s-sess.kr.edus.si/jagodic/rai/index_lav.htm
(zadnji obisk: 3.9.2008)

7. JUGSI, *Dedovanje*. Dostopno na naslovu
http://www.jugsi.net/dokumentacija/Knjiga/2/24.html
(zadnji obisk: 3.9.2008)

8. JUGSI, *Razredi*. Dostopno na naslovu
http://www.jugsi.net/dokumentacija/Knjiga/2/23.html
(zadnji obisk: 3.9.2008)

9. Jurič M. B., *Ogrodje Microsoft.NET*. Dostopno na naslovu
http://lisa.uni-mb.si/~juric/pikanet.pdf
(zadnji obisk: 3.9.2008)

10. Kerčmar N., *Prvi koraki v Javi*, diplomska naloga, 2006, FMF. Dostopno tudi na naslovu
http://rc.fmf.uni-lj.si/matija/OpravljeneDiplome/diplomska_naloga_NinaKercmar.pdf
(zadnji obisk: 3.9.2008)

11. Knjižnica za javo. Dostopna na naslovu
http://java.sun.com/j2se/1.3/docs/api/
(zadnji obisk: 3.9.2008)

12. Knjižnica MSDN. Dostopno na naslovu
http://msdn.microsoft.com/sl-si/library/default(en-us).aspx
(zadnji obisk: 3.9.2008)

13. Kodirnica.net, *Visual Studio Express za začetnike*. Dostopno na naslovu
http://www.kodirnica.net/dotnetnuke/%C4%8Clanki/tabid/55/articleType/ArticleView/articleId/44/Visual-Studio-Express-za-zaetnike.aspx
(zadnji obisk: 3.9.2008)

14. Kodirnica.net, *Imenski prostori v ogrodju .NET*. Dostopno na naslovu
http://www.kodirnica.net/dotnetnuke/Članki/tabid/55/articleType/ArticleView/articleId/6/Imenski-prostori-v-ogrodju-NET.aspx
(zadnji obisk: 3.9.2008)

15. P. Mrhar, *Java 2 - Prvi koraki*, Šempeter pri Gorici, Flamingo založba, 2002

16. Privošnik M., *Osnove C#*. Dostopno na naslovu
http://www.guru.si/guru.php?l=0&AID=15
(zadnji obisk: 3.9.2008)

17. Sharp J., Jagger J., *Microsoft Visual C# .NET Step by Step*, Redmond (Washington), Microsoft Press, 2002

18. SLODUG, *Forum*. Dostopno na naslovu
http://slodug.si/forums/12.aspx
(zadnji obisk: 3.9.2008)

19. Uranič S., *C#.NET*. Dostopno na naslovu
http://uranic.tsckr.si/C%23/APJ%203.letnikOLD.pdf in
http://uranic.tsckr.si/VISUAL%20C%23/VISUAL%20C%23OLD.pdf
(zadnji obisk: 3.9.2008)

20. Vidmar D., *Kaj je novega v C#3.0*, MONITOR, **12**(2007), str. 72-74. Dostopno tudi na naslovu
http://www.monitor.si/listalnik.php?datum=200712#
(zadnji obisk: 3.9.2008)

21. Wikipedia, *.NET Framework*. Dostopno na naslovu
http://en.wikipedia.org/wiki/Microsoft_.NET#Microsoft_.NET
(zadnji obisk: 3.9.2008)

22. Zgledi in vaje iz programskega jezika Pascal. Dostopno na naslovu
http://www.educa.fmf.uni-lj.si/www375/2001/prj/vajepas/zgledi_vaje_pascal.htm
(zadnji obisk: 3.9.2008)

23. Zaveršnik M., *Java*. Dostopno na naslovu
http://zaversnik.fmf.uni-lj.si/Gradiva/Java/index.php
(zadnji obisk: 3.9.2008)