2

Uvod u Javu

- Programski jezik Java
- Izvršavanje Java programa
- BlueJ razvojno okruženje
- Elementi Jave
- Tipovi podataka
- Prvi programi

Programski jezik Java

Krajem 1990 godine skupina programera tvrtke SUN krenula je u kreiranje jedinstvenog programskog jezika čije će se aplikacije moći izvršavati ne samo na računalima već i na ostalim uređajima koji u sebi sadrže mikročipove (televizor, perilica rublja, toster, …).
Nakon nepunih 5 godina svjetlo dana je ugledao novi programski jezik poznat pod imenom Java.
Java je danas poznata kao programski jezik čije se aplikacije mogu izvršavati na Internetu, na računalima različitih proizvođača i različitih operativnih sustava. To znači da se ista aplikacija može izvršavati na PC-u, Mac-u, … odnosno ista se aplikacije može izvršavati pod operativnim sustavom Windows, Unix, Linux,…
Svjedoci smo da se danas Java aplikacije izvršavaju na Internetu, mobilnim telefonima ali i mnogim drugim uređajima.
Java je jednostavan, u potpunosti objektno-orijentirani programski jezik. Sintaksa Jave podsjeća na sintaksu C-a odnosno C++-a. No isto tako, kako zbog potpune objektno-orijentiranosti ali i niza drugih karakteristika dosta razlikuje od njih. Prema nekim autorima poznavanje jezika kao što su C odnosno C++ može biti čak otežavajuće za učenje Jave jer programer u početku koristi ono što već zna iz drugih jezika i teško usvaja mogućnosti koje mu nudi Java.

Kod Jave razlikujemo dvije vrste "programa":

- **aplikacije (application)**
- **apleti (applet)**

Aplikacija je program koji se nalazi na računalu korisnika i izvršavaju se kada ih korisnik pokrene. Klasični primjeri aplikacija bili bi npr. Warcraft III, Microsoft Word,…Da bi se Java aplikacije mogle izvršavati na računalu korisnika računalo mora imati instaliran Javin izvršni program (Java Virtual Machine), koji može pokrenuti aplikaciju pisanu u Javi. Većina današnjih operativnih sustava u sebi imaju ugrađen Java Virtual Machine.
Za razliku od aplikacije, **aplet** je svojevrsna Internet aplikacija koja se izvršava u pregledniku (Internet Explorer, Netscape Navigator,…). Apleti su u stvari aplikacije koje "žive" na Internetu i pokreću se učitavanjem stranice na kojoj se nalaze. Zbog činjenice da se izvršavaju na Internetu apleti su daleko siromašniji u svojim mogućnostima. Kao i ostale Internet aplikacije apleti najčešće nemaju mogućnost pisati po tvrdom disku korisnika. Ne mogu upotrebljavati sve resurse korisnikovog računala, ali imaju jednu veliku prednost a to je činjenica da se nalaze na Internetu i bez instaliranja su dostupne svim korisnicima Interneta.

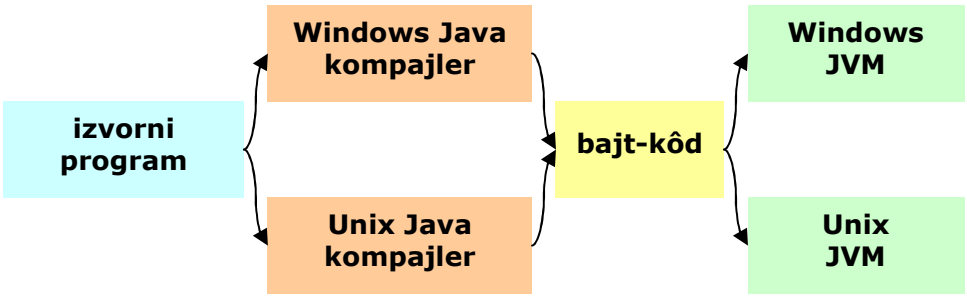
Aplikacija – program koji se samostalno izvršava na računalu korisnika
Applet – "mali program" koji se izvršava na Internetu, izvršava se unutar nekog od Web preglednika (Internet Explorer, Netscape Navigator,…)

Izvršavanje Java programa

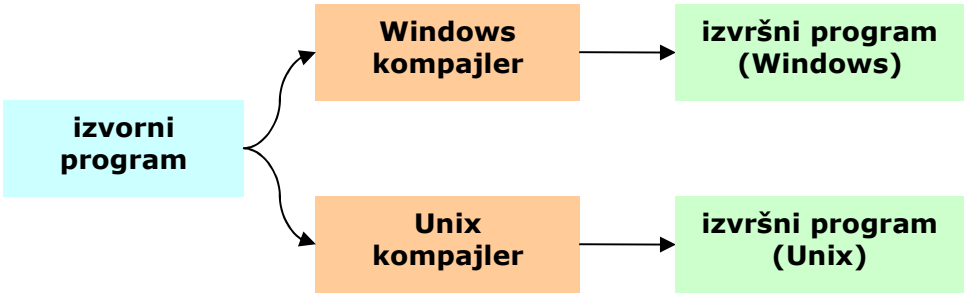
Kao što smo već napomenuli programi pisani u Javi mogu se izvršavati ne samo na različitim operativnim sustavima već i na različitim uređajima. Zbog toga se programi pisani u Javi izvršavaju malo drugačije nego što je to uobičajeno.

Kao i kod ostalih programskih jezika, programer u odabranom editoru piše kôd programa sljedeći pravila Jave. Takav program, naziva se **izvorni program** (**source program** ili **source code**). Svaki program napisan u Javi u stvari predstavlja jednu **klasu** (**class**) čije se ime navodi u izvornom programu. Tako napisani izvorni program potrebno je spremiti u tekstualni dokument pod imenom `ime_klase.java`, pri čemu je ime klase ime klase koje smo naveli u izvornom programu.

Nakon što je izvorni program napisan i spremljen slijedi druga faza a to je **prevođenje programa** (**kompajliranje**) u tzv. **bajt-kôd** (**bytecode)** oblik programa. Prilikom prevođenja programa prevoditelj analizira **sintaksnu** i **semantičku** ispravnost programa. U kôdu traži elemente Jave te ih interpretira i prevodi u univerzalni **bajt-kôd**, koji će se moći izvršavati neovisno o operativnom sustavu i u tome i jest osnovna razlika između Jave i ostalih programskih jezika. Kod ostalih se programskih jezika kompajliranjem stvara kôd specifičan za odgovarajući operativni sustav.



Slika 2-1 Kompajliranje Javi programa na različitim operativnim sustavima

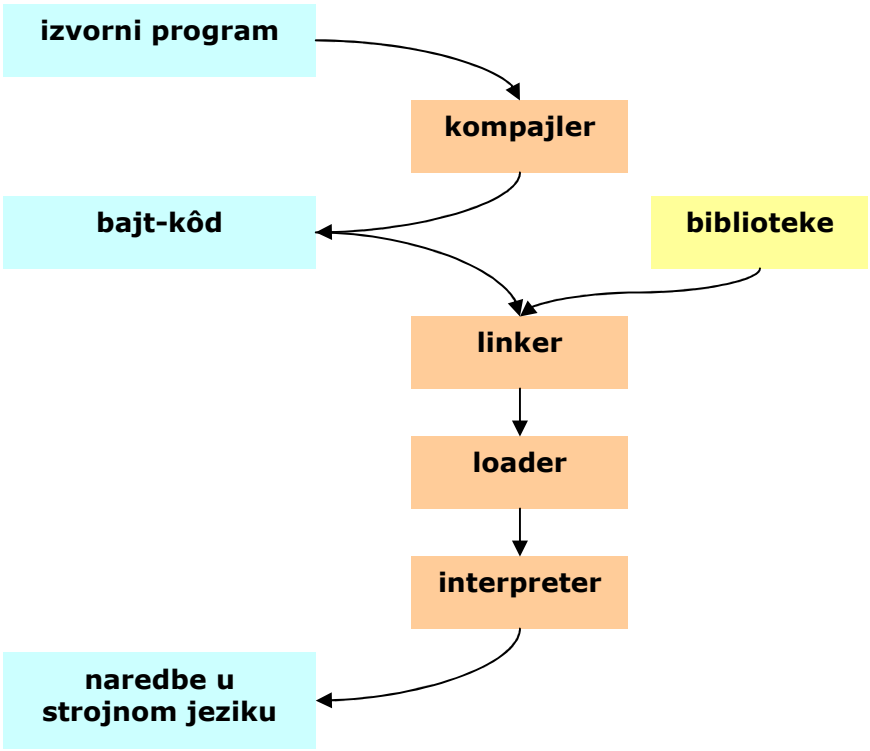


Slika 2-2 Standardno kompajliranje programa na različitim operativnim sustavima

Ukoliko prilikom prevođenja prevoditelj naiđe na izraze ili dijelove koje ne zna interpretirati prijavit će odgovarajuću grešku i kompajliranje neće biti uspješno završeno.
Krajnji rezultat kompajliranja je datoteka pod imenom `ime_klase.class`.

Nakon kompajliranja programa, program je spreman za pokretanje. Program se pokreće pokretanjem datoteke `.class`. Java apleti se pokreću pomoću Web preglednika odnosno

pomoću odgovarajućeg programa koji omogućava pokretanje apleta – **AppletViewer**. Samo izvršavanje Java programa i nije tako jednostavno. Naime, prilikom pisanja programa programer najčešće koristi neke već gotove dijelove kôda. Npr. koristi već gotove naredbe za unos podataka s tipkovnice, ispis podataka na ekran, kreiranje korisničkog sučelja,… koje programeru znatno olakšavaju posao. Java u tu svrhu sadrži svoj **Java development kit** (**JDK**). **Java development kit** u osnovi sadrži niz gotovih klasa koje su vrlo korisne prilikom programiranja. Nakon što je program kompajliran, on se i dalje referencira na gotovi kôd iz **JDK**-a. Taj dio kôda nalazi se unutar tzv. **biblioteka** (**library**). Biblioteke su u Javi organizirane u **pakete** (**package**) koji su u stvari skupovi već gotovih klasa. Povezivanje bajt-kôda kreirane aplikacije s dijelovima odgovarajućih biblioteka zadaća je programa koji nazivamo **linker**.
Sada možemo detaljnije objasniti izvršavanje programa pisanih u Javi. Najprije se bajt-kôd napisanog programa povezuje s odgovarajućim bibliotekama a potom se učitava u radnu memoriju računala. Učitavanje programa u radnu memoriju računala automatski izvršava program poznat kao **loader**. Kao što smo već spomenuli program je u stvari niz instrukcija koje se izvršavaju. Isto smo tako rekli da se računalo razumije samo naredbe napisane u strojnom jeziku. Naš program se i dalje nalazi u bajt-kôdu. Prevođenje naredbi iz bajt-kôda u strojni jezik obavit će **interpreter**. **Interpreter** je još jedan u nizu programa čija je zadaća prevoditi naredbu po naredbu u strojni jezik računala i potom ju izvršiti.
Kao što smo već rekli bajt-kôd Java programa može se izvršavati na različitim operativnim sustavima. Jedino što je potrebno da bi se bajt-kôd izvršio na nekom računalu je **Java Virtual Machine** (**JVM**). **Java Virtual Machine** razumije bajt-kôd te omogućava prevođenje programa na strojni jezik računala.



Slika 2-3 Izvršavanje programa pisanih u Javi

Izvorni program (source program, source code) – program napisan u nekom višem programskom jeziku.
Bajt-kôd – optimizirani skup naredbi kreiranih tako da se mogu izvršavati na bilo kojem operativnom sustavu. Nastaje kompajliranjem izvornog programa.
AppletViewer – program koji omogućava pokretanje apleta izvan Web preglednika
Java development kit (JDK) – programska oprema za pisanje programa u Javi

Biblioteka (library) – skup svih klasa koje su već definirane i koje se mogu pozivati unutar programa
Paket – skup međusobno povezanih klasa
Linker – program koji povezuje bajt-kôd s već gotovim klasama iz JDK
Loader – program koji učitava izvršni program u radnu memoriju računala
Interpreter – program koji prevodi jednu po jednu naredbu iz bajt-kôda u strojni jezik računala te ju izvršava.
Java Virtual Machine – izvršni sustav za izvršavanje Java bajt-kôda na nekom računalu.

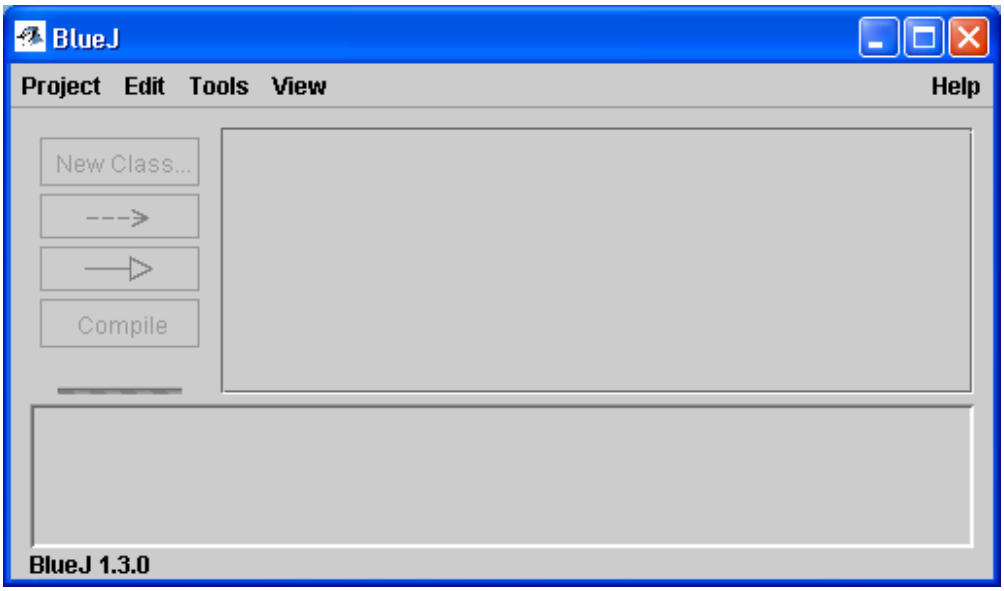
BlueJ razvojno okruženje

Da bismo započeli programirati u Javi potreban nam je u najmanju ruku Java interpreter koji će izvorni program prevesti u bajt-kôd. Izvorni program moguće je napisati u bilo kojem editoru. Programiranje u nekom običnom editoru nije baš ugodno. U običnom editoru nije vidljiva razlika između naredbi programskog jezika, komentara, ili teksta koji se ispisuje, što može znatno otežati programiranje. Stoga danas za većinu jezika postoje okruženja za pisanje programa tzv. **razvojna okruženja**. **Razvojna okruženja** čine programiranje "ugodnijim". Editori razvojnih okruženja najčešće različitim bojama označavaju ključne riječi, komentare, varijable,… Novija su razvojna okruženja još inteligentnija pa na osnovu prvih nekoliko znakova predviđaju unos, što znatno može olakšati posao programeru.
Većina današnjih velikih informatičkih tvrtki nudi razvojna okruženja za Javu. Borland je npr. napravio *JBuilder*, Microsoft je napravio *J++*, Sun je napravio *Java Workshop*,… Većina ovih okruženja je komercijala i cijena im je nerijetko dosta visoka.
Naše će razvojno okruženje u ovoj knjizi biti **BlueJ**. **BlueJ** je razvojno okruženje koje korisniku omogućava da "razmišlja na objektni način". Osim toga izuzetno je pogodan za kontrolu izvršavanja pojedinog dijela programa. No imajmo na umu i jedan bitan detalj a to je činjenica da je BlueJ besplatan i dostupan svima na Internetu.
BlueJ razvojno okruženje napisan je u Javi što omogućava njegovo izvršavanje na svim operativnim sustavima.

Razvojno okruženje – skup programa koji programiranje u nekom programskom jeziku čine ugodnijim. Najčešće u sebi imaju ugrađen editor za pisanje programa, omogućavaju kompajliranje programa, omogućavaju kontrolu izvršavanja programa,…
BlueJ – besplatno Java razvojno okruženje namijenjeno prvenstveno učenju objektno-orijentiranog programiranja u Javi.

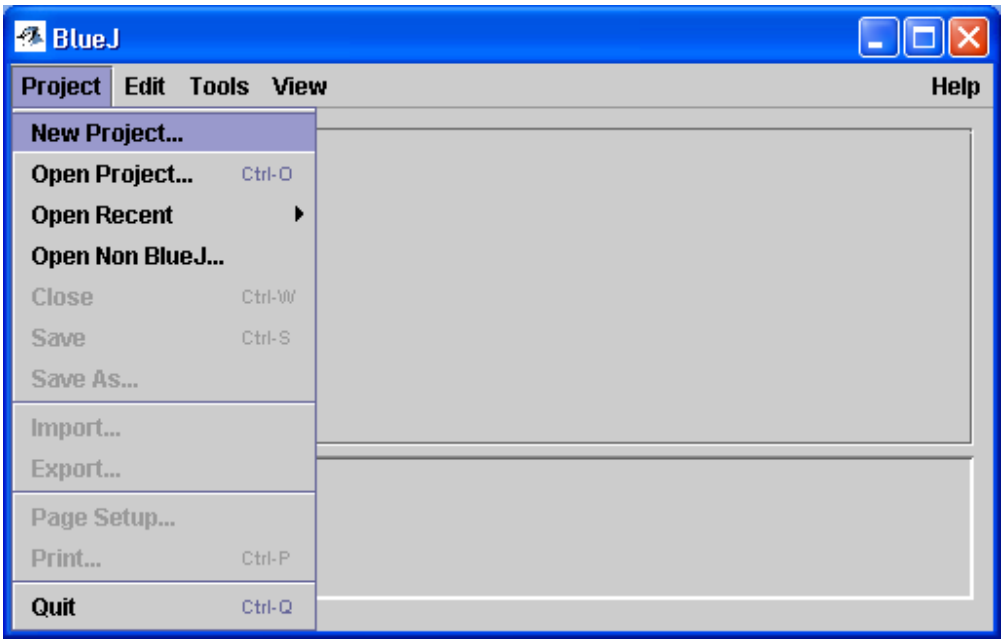
Prije instalacije BlueJa na računalo potrebno je instalirati Java 2 runtime (JDK). JDK i BlueJ nalaze se na CD-u u prilogu ove knjige.

Pokretanjem BlueJ aplikacije na ekranu će se pojaviti sučelje koje će nam omogućiti pisanje programa.



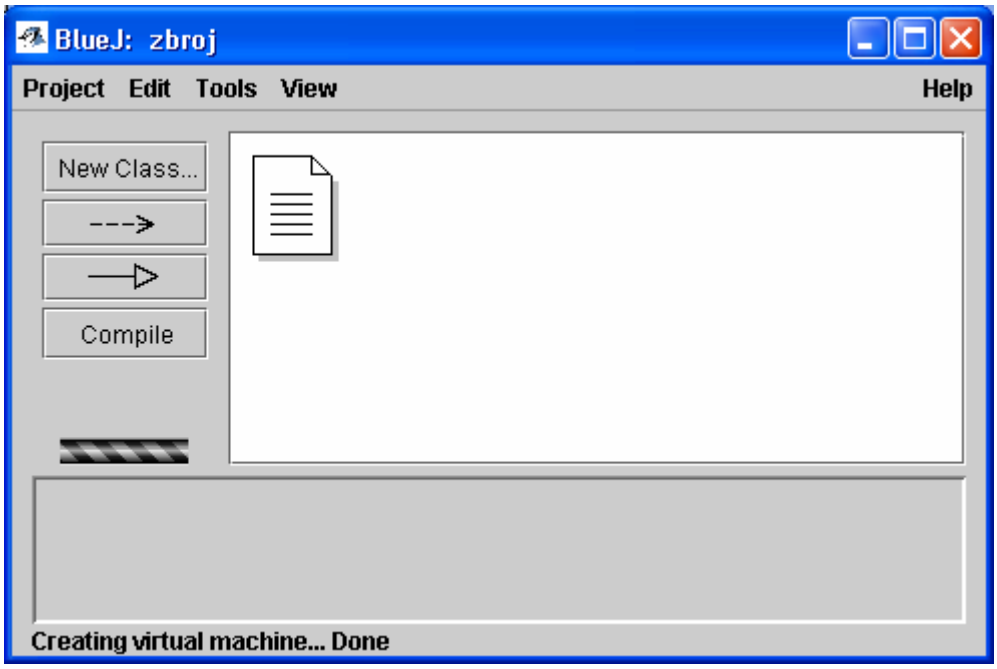
Slika 2-3 Sučelje BlueJ razvojnog alata

Nakon što smo pokrenuli program za početak ćemo otvoriti novi projekt. Novi projekt ćemo otvoriti tako da u izborniku **Project** odaberemo **New Project**.



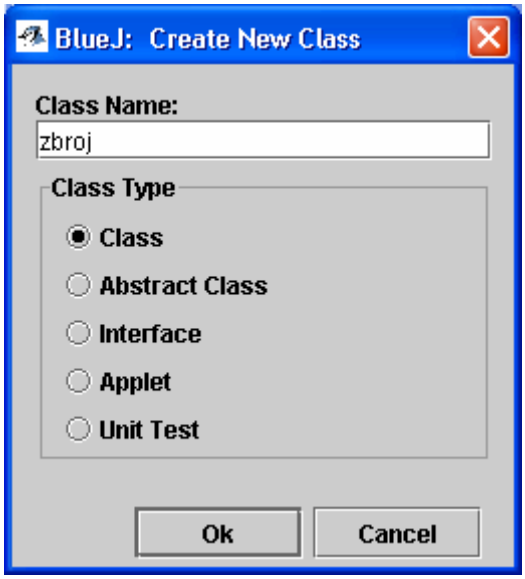
Slika 2-4 Kreiranje novog projekta

Nadalje ćemo u prozoru koji se otvori upisati ime projekta. BlueJ će potom kreirati poseban direktorij s upisanim imenom te će otvoriti novi prozor za upravo kreirani projekt.



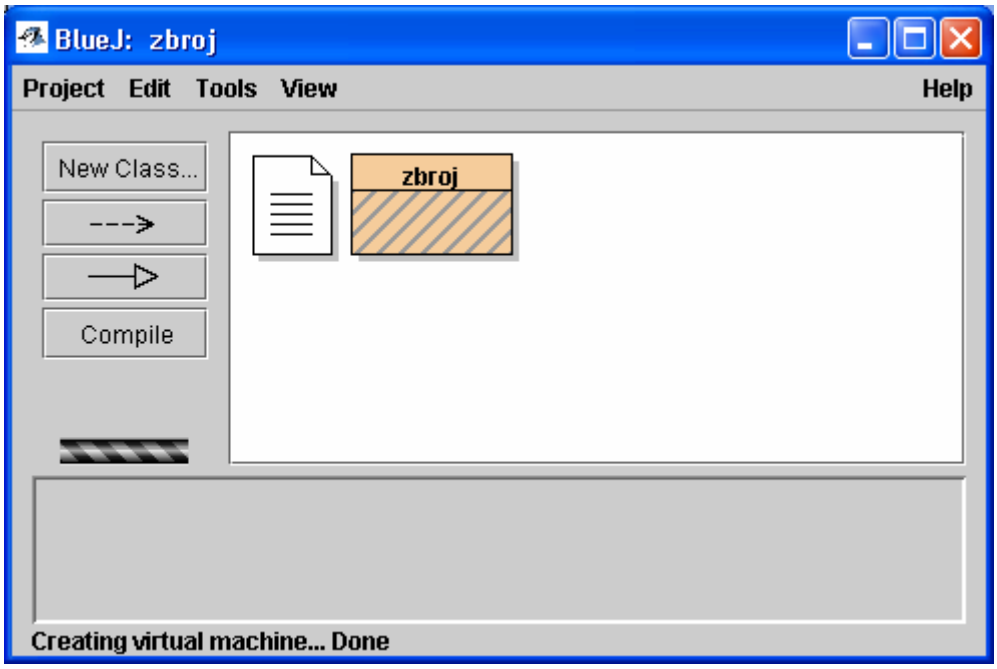
Slika 2-5 Prozor kreiranog projekta

Sljedeći korak je dodavanje klase u projekt. Klasu ćemo u ovom trenutku poistovjetiti s programom. Više riječi o klasama bit će u sljedećim poglavljima. Novu ćemo klasu kreirati tako da kliknemo na gumb **New Class…** koji se nalazi pri vrhu lijevog dijela prozora.
U prozoru koji se otvori upišemo ime klase.



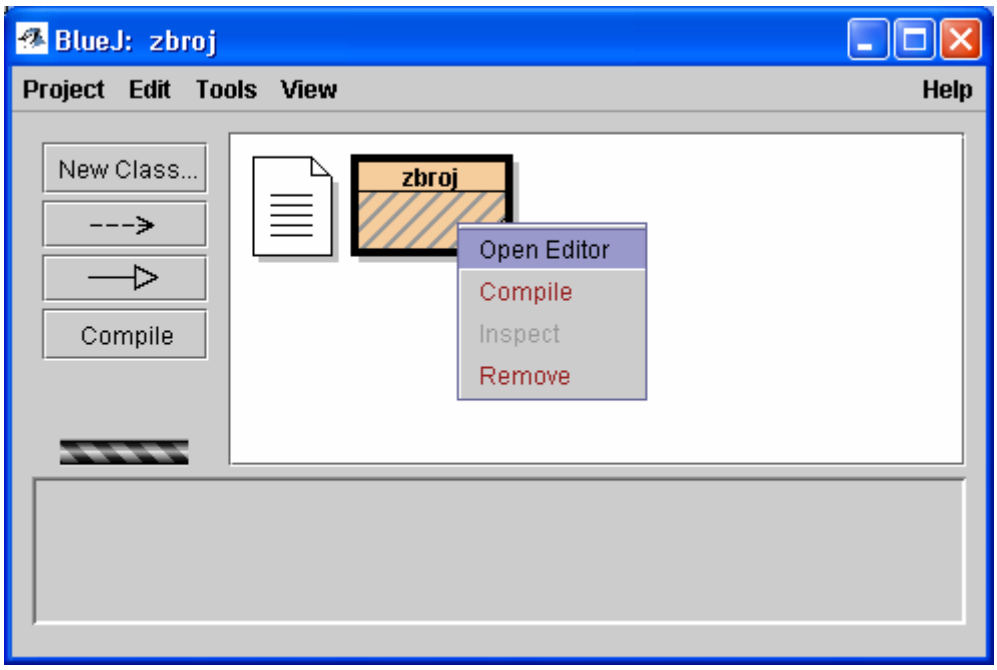
Slika 2-6 Prozor za kreiranje klase

Nakon što smo kliknuli na gumb OK prozor će se zatvoriti i u postojećem prozoru će se pojaviti još jedna sličica koja će predstavljati upravo kreiranu klasu, a pri vrhu sličice će se nalaziti ime klase (*zbroj*).



Slika 2-7 Kreirana klasa je dodana u projekt

Ovime definicija klase nije gotova. Štoviše definicija klase tek slijedi. Stoga ćemo kliknuti desnim gumbom miša na sličicu klase *zbroj* i u izborniku odabrati **Open Editor** ili jednostavno 2 puta kliknuti lijevim gumbom miša na ime klase.

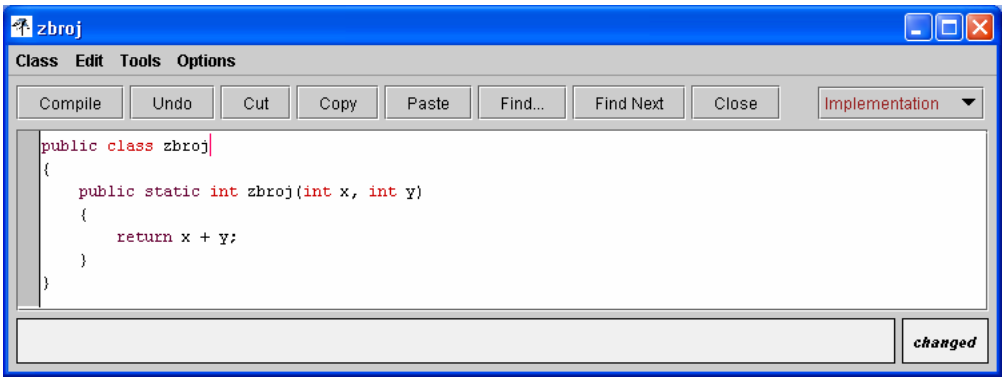


Slika 2-8 Pokretanje editora za definiranje klase

U editoru koji se je otvorio nalazi se već upisan dio klase koju želimo kreirati. Za početak ćemo izbrisati sadržaj prozora i upisati sljedeći kôd:

```
public class zbroj
{
     public static int zbroj(int x, int y)
     {
```

```
        return x + y;
    }
}
```

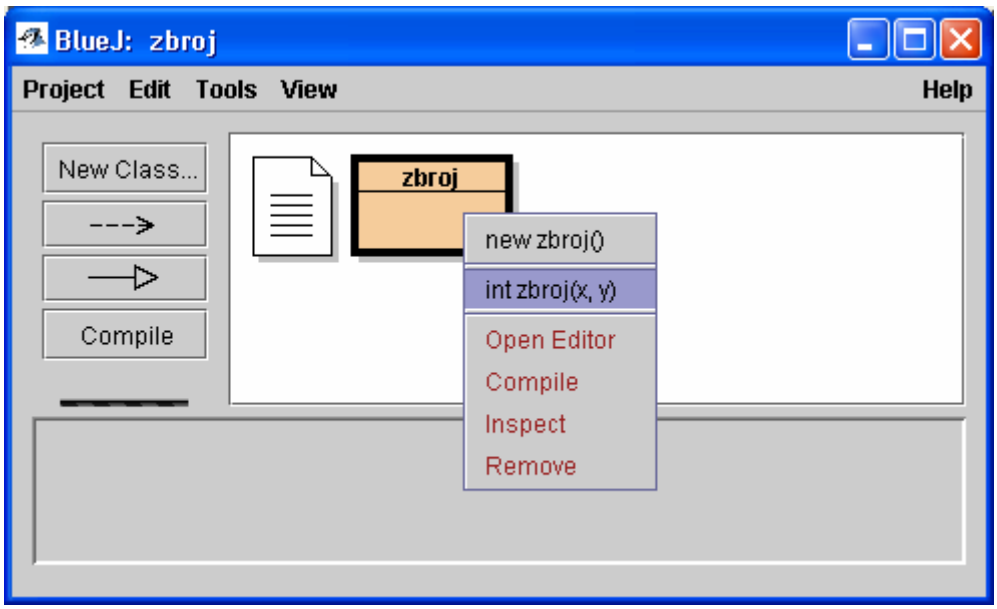


Slika 2-9 Izgled prozora nakon što upišemo kôd klase

Sljedeći je korak kompajliranje ovako programa. Da bismo program kompajlirali lijevim ćemo gumbom miša kliknuti na gumb **Compile** u gornjem izborniku prozora. Prije samog kompajliranja program će se spremiti. Ukoliko kompajler nije naišao na greške u kôdu, u donjem će se dijelu prozora pojaviti poruka:

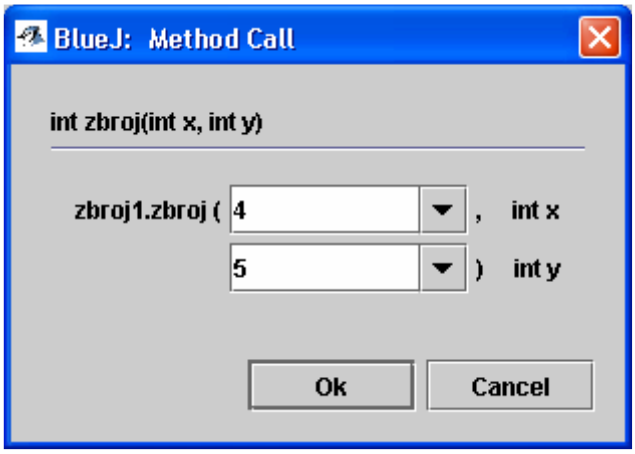
```
Class compiled - no syntaks errors
```

Ponovo ćemo se vratiti na prozor u kojem su prikazane sve klase našeg projekta (*Slika 2-8*) te ćemo desnim gumbom miša kliknuti na sličicu klase **`zbroj`** te ćemo iz izbornika odabrati **int zbroj(x, y)**. Time ćemo u stvari pokrenuti metodu **`zbroj()`** klase **`zbroj`**.



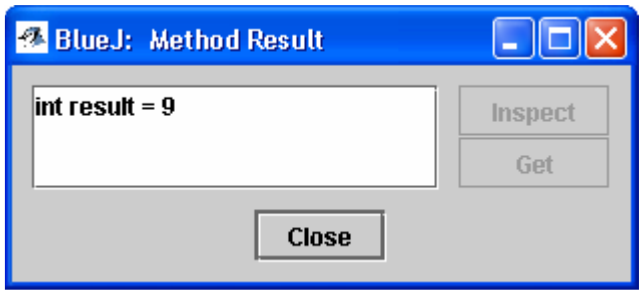
Slika 2-10 Pokretanje metode *zbroj()*

Budući da metoda **`zbroj()`** ima dva ulazna parametra (varijable) (**`x`** i **`y`**) te vraća njihov zbroj, otvorit će se novi prozor unutar kojeg ćemo upisati vrijednosti za parametre **`x`** i **`y`**.



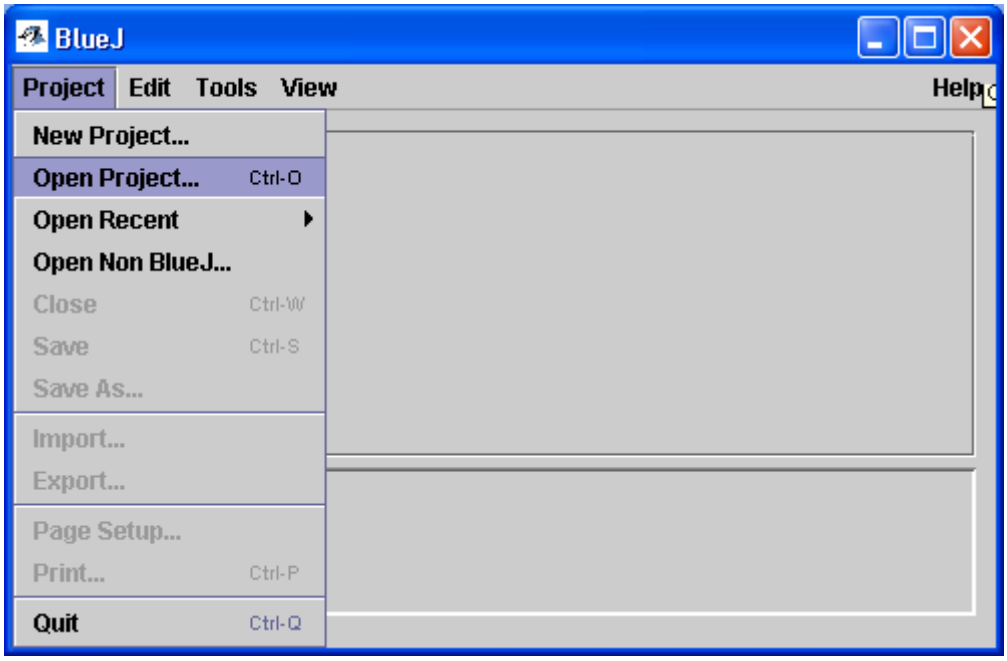
Slika 2-11 Okvir za definiranje vrijednosti parametara metode

Klikom na gumb **OK** otvorit će se novi prozor u kojem će se nalaziti vrijednost koju metoda vraća. Budući da naša metoda računa zbroj brojeva **x** i **y** rezultat će, za unesene vrijednosti 4 i 5 biti **9**.



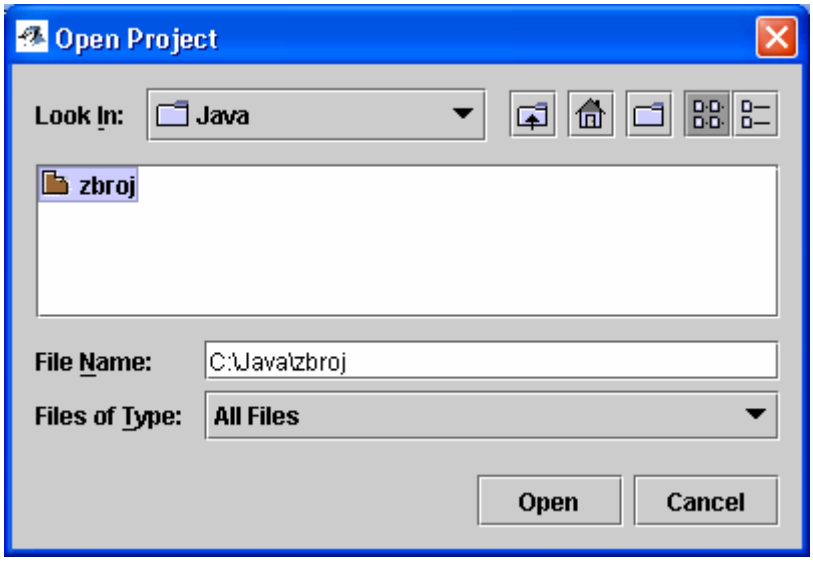
Slika 2-12 Rezultat izvršavanja metode

Jednom kreiranu klasu moguće je ponovno otvarati i po potrebi mijenjati te izvršavati. Klasu ćemo otvoriti tako da u izborniku **Project** odaberemo **Open Project**.



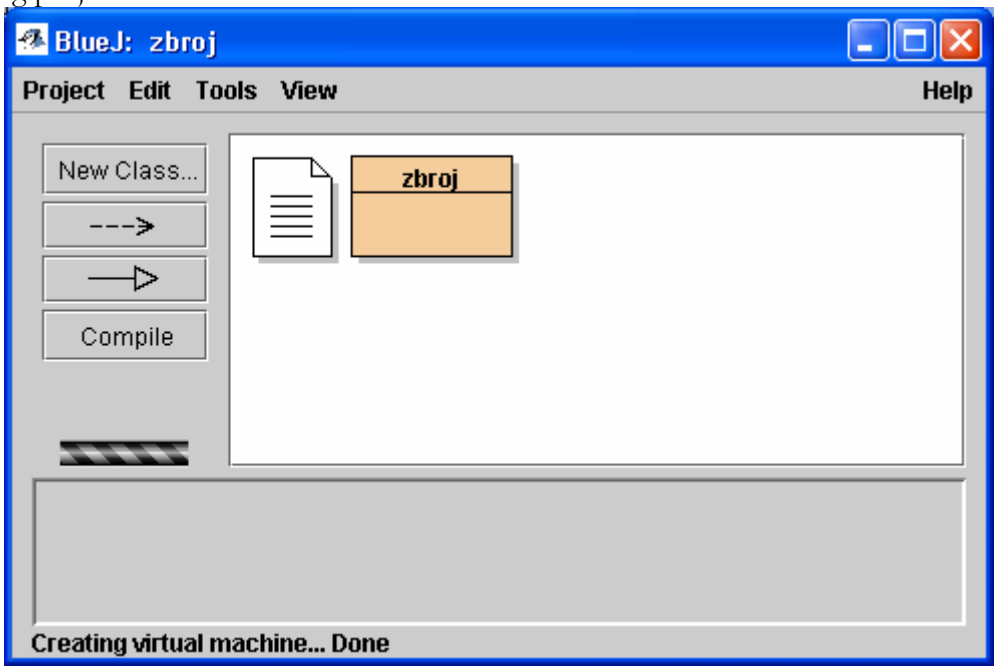
Slika 2-13 Otvaranje postojećeg projekta

U prozoru koji se otvori odaberemo ime projekta koji želimo otvoriti.



Slika 2-14 Odabiranje projekta koji želimo otvoriti

Na kraju ćemo kliknuti na gumb **Open** i pojavit će se prozor u kojem će se nalaziti sve klase odabranog projekta.



Slika 2-15 Prozor s klasama odabranog projekta

Pojmove kao što su *klasa*, *metoda*,… s kojima smo se ovdje susreli detaljno ćemo objasniti u nastavku knjige. Za sada će za nas *klasa* biti program a *metodu* možemo zamisliti kao manji program unutar velikog programa. Metoda ima svoje ulazne parametre, to su u stvari podaci koji su metodi potrebni da bi riješila neki problem. U opisanom primjeru imali smo jednu metodu **`zbroj()`** koja je računala zbroj dvaju prirodnih brojeva. Da bi metoda mogla izračunati zbroj dva broja treba znati koji su to brojevi, stoga je imala dva ulazna parametra, brojeve **`x`** i **`y`**.

Elementi Jave

Da bismo uspješno programirali u nekom programskom jeziku trebamo dobro vladati njegovim **sintaksnim i semantičkim pravilima**, **ključnim riječima** te **specijalnim znakovima**. **Sintaksa** jezika odnosi se na naredbe programskog jezika, objašnjava kako izgledaju ispravno napisani izrazi (naredbe),... **Semantika** se odnosi na značenje naredbi, tj. što ta naredba radi.
Specijalni znakovi su znakovi koji imaju svoje značenje unutar programskog jezika. Neki od specijalnih znakova dani su sljedećom tablicom:

Kategorija	Specijalni znakovi
matematički	`+ - * /`
interpunkcijski	`. , ? ;`
relacijski	`< <= > >= == !=`

Tablica 2-1 Najčešće korišteni specijalni znakovi u Javi

Kao što postoje simboli koji imaju svoje specijalno značenje, tako postoje i riječi koje imaju specijalno značenje, koje nazivamo rezervirane riječi (keywords). Neke od njih su: **`class, public, private, static, int, float, boolean,...`**

Kao što smo mogli primijetiti u primjeru iz zadnjeg poglavlja, pri samom kreiranju klase, morali smo joj dodijeliti ime (*zbroj*), nadalje smo metodi isto tako morali dodijeliti ime (*zbroj*), a dodatno smo imenovali i parametre (varijable) u metodi (*x*, *y*). Općenito ćemo sva takva i slična imena zvati **identifikatori**. **Identifikatori** u Javi mogu se sastojati od slova engleske abecede, znamenaka i znakova "_" i "$", pri čemu znamenka ne može biti na prvom mjestu. Duljina identifikatora (broj znakova) u Javi nije ograničena. Ovdje je isto tako bitno napomenuti da je Java **case sensitive** jezik, što znači da se velika i mala slova razlikuju (npr. *Zbroj* i *zbroj* su dva različita identifikatora).

Primjer 2 – 1:
Sljedeći identifikatori su korektno napisani:
`prvi`
`$2`
`_treci$a`

Primjer 2 – 2:
Primjeri neregularnih identifikatora:
`1a` – *na prvom mjestu se ne smije nalaziti znamenka*
`zbroj brojeva` – *sadrži razmak*
`kuća` – *nedozvoljeni znak* ć
Kako bismo se u vlastitom kodu uspješno snalazili i nakon nekog vremena, odnosno kako bismo olakšali razumijevanje koda svima onima koji će ga kasnije čitati, koristit ćemo komentare. Općenito je namjena komentara da se unutar njih stavljaju objašnjenja koda ili neke napomene za izmjene programa.
Želimo li u komentar staviti samo jednu liniju koda, na početku te linije ćemo staviti **`//`**.

```
//komentar
```

Više linija koda ćemo staviti u komentar tako da na početku prve linije koda stavimo **`/*`**, te iza posljednjeg znaka stavimo **`*/`**.

```
/* komentar1
komentar2
komentar3 */
```

Npr.

```
//ovo je komentar
```

ili

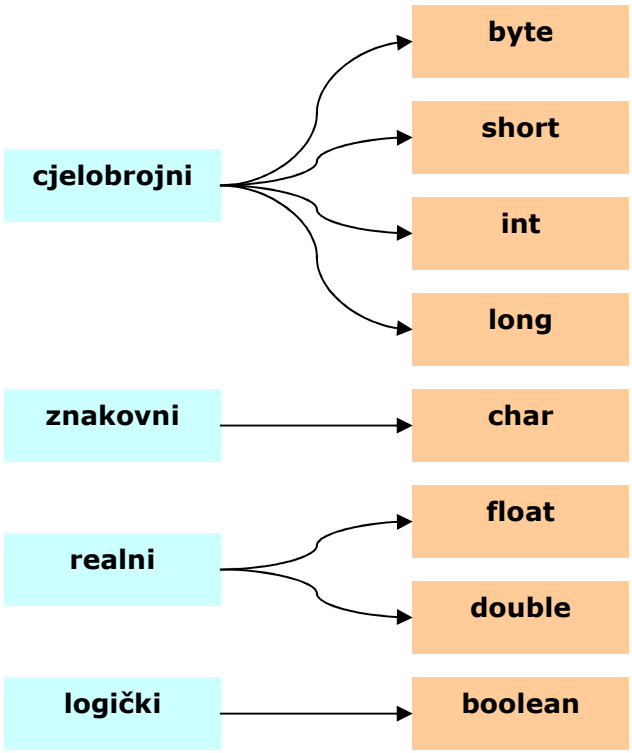
```
/*
ovo je komentar
*/
```

Tipovi podataka

U osnovi kod Jave razlikujemo dvije kategorije tipova podataka:

- **objektni tipovi podataka**
- **neobjektni tipovi podataka**

Objektni tipovi podataka definirani su klasama i o njima će biti govora nešto kasnije.
Neobjektni (**jednostavni**) tipovi podataka i shematski su prikazani na sljedećoj slici:



Slika 2-16 Jednostavni tipovi podataka

Jednostavni tipovi podataka

Cjelobrojni tip podataka

Unutar cjelobrojnog tipa podataka postoji nekoliko podtipova. Podtipovi se razlikuju s obzirom na raspon vrijednosti za koje su definirani, što neposredno utječe na količinu memorije koju zauzimaju.

Raspon vrijednosti i količina memorije potrebne za spremanje jedne varijable dani su sljedećom tablicom:

Tip podataka	Raspon vrijednosti	Zauzeće memorije (bitovi)
`byte`	-128 do 127	8
`short`	-32768 do 32767	16
`int`	-2147483648 do 2147483647	32
`long`	-9223372036854775808 do 9223372036854775807	64

Tablica 2-2 Cjelobrojni tipovi podataka

Znakovni tip podataka

Tip podataka `char` namijenjen je radu s znakovima. Vrijednosti koje ovaj tip podataka može primiti su bilo koji znak iz tzv. **Unicode** skupa znakova. Znakovi se navode unutar jednostrukih navodnika npr. `'A', '0', '+', '%',...`
Osim ovakvih znakova, tip podataka `char` sadrži i neke znakove koji postoje na tipkovnici, ali za njih ne postoji posebna oznaka (npr. tipka *tab*). Takvi znakovi sastoje se od oznake "\" i odgovarajućeg znaka. Neki od takvih oznaka dani su u sljedećoj tablici:

Znak	Opis
\'	jednostruki navodnik
\"	dvostruki navodnik
\\	kosa crta "\"
\r	prelazak u novi red (carriage return)
\n	prelazak u novi red (line feed)
\t	horizontalni tabulator
\b	backspace

Tablica 2-3 Neki od specijalnih znakova

Realni tip podataka

Kao što mu i ime govori, ovaj ćemo tip podataka koristiti pri radu s realnim vrijednostima. Java brojeve prikazuje u tzv. eksponencijalnom obliku. Kod takvog prikaza broj se sastoji od sljedećih elemenata:

- jednoznamenkasti cijeli broj
- decimalna točka
- 6 decimala
- znak "E"
- eksponent

Primjer 2 – 3:
Primjer zapisa realnih brojeva u eksponencijalnom obliku

broj	**eksponencijalni zapis**	**objašnjenje**
315.46	3.154600E2	$315.46 = 3.1546 \cdot 10^{2}$
0.02	2.000000E-2	$0.02 = 2 \cdot 10^{-2}$
-3.14	-3.140000E0	$-3.14 = -3.14 \cdot 10^{0}$

Osnovni podtipovi realnog tipa podataka su dani su sljedećom tablicom:

Tip podataka	Raspon vrijednosti	Zauzeće memorije (bitovi)
`float`	$-3.4 \cdot 10^{38}$ do $3.4 \cdot 10^{38}$	32
`double`	$-1.7 \cdot 10^{308}$ do $1.7 \cdot 10^{308}$	64

Tablica 2-4 Realni tipovi podataka

Osim po rasponu vrijednosti i količini memorije koju zauzimaju, ovi se tipovi podataka razlikuju i s obzirom na preciznost, tj. s obzirom na broj bitnih decimalnih znamenaka. Broj decimalnih znamenaka koje će se uzimati u obzir kod **`float`** tipa podataka je **6** ili **7**, odnosno **15** kod **`double`**.

Logički tip podataka

Logički tip podataka (**`boolean`**) može sadržavati samo dvije vrijednosti **`true`** ili **`false`** (istina ili laž). Memorija potrebna za pohranjivanje vrijednosti tipa **`boolean`** je **1 bit**.

Deklaracija varijabli

Kao što smo već rekli, varijable su u stvari "oznake" za memorijske lokacije na koje ćemo privremeno pohranjivati podatke tijekom izvršavanja programa. Prije pohranjivanja vrijednosti trebamo odrediti koji tip vrijednosti ćemo pohraniti na odgovarajuću lokaciju, te rezervirati (alocirati) prostor. To sve ćemo napraviti deklaracijom varijabli.

Opći oblik deklaracije varijabli u Javi je:

```
tip ime_varijable;
```

gdje je **tip** bilo koji tip podataka (**int**, **float**, **boolean**, **char**,...). Ukoliko se istovremeno deklarira više varijabli istog tipa, one se međusobno odvajaju zarezom.

Primjer 2 – 4:
Primjeri deklaracija varijabli:

```
int n;
float a, b;
char c;
boolean t;
```

Neki programski jezici deklariranjem varijable automatski postavljaju na neku početnu (inicijalnu) vrijednost, tj. **inicijaliziraju** ih na neku vrijednost. Npr. u Pascalu će varijabla tipa **byte** deklariranjem biti inicijalizirana na vrijednost 0. Kod Jave to nije slučaj. Kod Jave vrijednosti varijable nakon deklaracije ne moraju biti inicijalizirane, i ukoliko to ne napravimo eksplicitno a u programu koristimo takve varijable prilikom kompajliranja će se pojaviti greška.

Varijabli ćemo pridružiti vrijednost operatorom "=".

Primjer 2 – 5:
Primjeri pridruživanja vrijednosti varijablama:

```
n = 2;
x = 3.14;
c = 'c';
t = false;
```

Prilikom pridruživanja vrijednosti varijablama valja voditi računa da vrijednost koja se pridružuje varijabli bude istog tipa kao i varijabla.

Vrijednost varijable moguće je inicijalizirati na neku vrijednost i prilikom deklaracije varijabli.

U takvim slučajevima će deklaracija varijable imati sljedeći oblik:

```
tip ime_varijable = vrijednost;
```

Primjer 2 – 6:
Primjer deklaracije varijabli gdje se varijable inicijaliziraju neposredno iza deklaracije

```
int i = 3;
float x = -8.17;
```

```
char c = 'y', d = 'n';
```

Isto tako je prilikom deklariranja varijabli i njihovog inicijaliziranja, moguće koristiti i izraze koji će varijabli inicijalno pridružiti vrijednost tog izraza. Kod ovakve inicijalizacije trebamo voditi računa da su sve varijable, koje koristimo s desne strane znaka jednakosti inicijalizirane.

Primjer 2 – 7:
Primjer deklaracije varijabli gdje se varijable inicijaliziraju neposredno iza deklaracije i to dodjeljivanjem vrijednosti izraza.

```
int i = 3;
int j = i*2;
```

Operatori

Osnovni cilj programiranja je na osnovu ulaznih podataka dobiti potrebne rezultate. Da bismo iz ulaznih podataka dobili rezultate trebamo nad podacima izvršiti odgovarajuće manipulacije. Manipulacije nad podacima izvršit ćemo pomoću operatora. Operatore u Javi dijelimo u nekoliko osnovnih kategorija:

- aritmetički operatori;
- relacijski operatori;
- logički operatori;
- bitovni operatori.

S obzirom na koliko se podataka operator primjenjuje razlikujemo dvije vrste operatora:

- **unarni operatori** – primjenjuju se samo na jednu vrijednost (npr. operator ++, koji povećava vrijednost varijable za 1);
- **binarni operatori** – primjenjuju se na dvije vrijednosti (npr. operator + za zbrajanje dvaju brojeva).

Aritmetički operatori

U Javi je moguće koristiti sljedeće aritmetičke operatore:

Operator	Značenje
+	zbrajanje
-	oduzimanje
*	množenje
/	dijeljenje
%	ostatak cjelobrojnog dijeljenja
++	povećava vrijednost varijable za 1
--	smanjuje vrijednost varijable za 1
+=	povećavanje vrijednosti varijable za određeni broj
-=	analogno kao +=
*=	analogno kao +=
/=	analogno kao +=
%=	analogno kao +=

Tablica 2 – 5 Aritmetički operatori

Primjer 2 – 8:
Koja će biti vrijednost varijable `x` nakon djelovanja odgovarajućih operatora, ako je `x` cjelobrojna varijabla čija je vrijednost postavljena na 5?

a) `x++;`
b) `x = x + 3;`
c) `x = x / 2;`
d) `x = x % 3;`
e) `x *= 3;`

Rješenja:
a) **6**
b) **8**
c) **2** (budući da je `x` cjelobrojna varijabla operator / se tumači kao operator cjelobrojnog dijeljenja)
d) **2** (5 cjelobrojno podijeljeno s 3 je 1 i 2 je ostatak)
e) **15**

Relacijski operatori

Relacijski operatori koristi se za određivanje odnosa između dviju vrijednosti. Relacijski operatori dani su sljedećom tablicom:

Operator	Značenje
==	jednako
!=	različito
>	veće
>=	veće ili jednako
<	manje
<=	manje ili jednako

Tablica 2 – 6 Relacijski operatori

Relacijski operatori == i != mogu biti upotrijebljeni nad svim tipovima podataka (neobjektnim i objektnim), dok operatori >, <, >=, <= mogu biti upotrijebljeni samo nad onim tipovima podataka koji imaju poredak, tzv. **ordinalni** ili **redni** tipovi podataka.

Vrijednosti relacijskih operatora mogu biti **true** ili **false**.

Primjer 2 – 9:
Koje će biti vrijednosti sljedećih izraza?

a) `3 <= 5`
b) `'a' < 'A'`
c) `3 != 5`

Rješenja:
a) **true** (3 je manje od 5)
b) **false** (slovo '**a**' se u Unicode tablici nalazi iza slova '**A**')
c) **true** (3 je različito od 5)

Logički operatori

Logički operatori definirani su nad logičkim tipom podataka (**boolean**), vrijednosti na koje se ovi operatori primjenjuju i rezultati mogu biti samo vrijednosti *true* ili *false*.
Najčešće korišteni logički operatori dani su sljedećom tablicom:

Operator	Značenje
`!`	negacija
`&&`	logički I
`||`	logički ili

Tablica 2 – 7 Logički operatori

Djelovanje logičkih operatora prikazano je u sljedećom tablicom:

a	b	!a	a && b	a || b
false	false	true	false	false
false	true	true	false	true
true	false	false	false	true
true	true	false	true	true

Tablica 2 – 8 Djelovanje logičkih operatora

Primjer 2 – 10:
Koje će biti vrijednost varijable `c` nakon odgovarajućih operacija, ako je vrijednost varijable `a = true` i `b = false`?

a) `!a`
b) `b && a`
c) `!(a && b)`
d) `a && (b || (!b))`

Rješenja:
a) **false**
b) **false**
c) **false**
d) **true**
e) **true**

Prioritet operatora

Promotrimo sljedeći izraz:

```
c = 2 + 3 * 5;
```

nije sasvim jasno koju će vrijednost imati varijabla `c` nakon ovakvog pridruživanja. Hoće li se najprije izvršiti zbrajanje pa nakon toga množenje ili će se najprije izvršiti množenje pa nakon toga zbrajanje?
Kao što postoji prioritet operatora u matematici, tako postoji i u Javi i štoviše dosta su slični. Prioritet operatora dan je sljedećom tablicom:

Prioritet	Operatori
1	`()`
2	`++, --, !`
3	`*, /, %`
4	`+, -`
5	`<, <=, >, >=`
6	`==, !=`
7	`&&`
8	`||`

Tablica 2 – 9 Prioritet operatora

Općenito se izrazi izvršavaju od lijeva na desno.

Primjer 2 – 11:
Vrijednost izraza 2 + 3 * 4 će biti 14, najprije se izvršava množenje (3 * 4), a zatim zbrajanje (2 + 12)

Izračunavanje mješovitih izraza

Mješoviti izrazi su oni koji u sebi sadrže cjelobrojne i realne vrijednosti. Primjer takvog izraza je:

```
8 / 3 + 2.7
```

Primijetimo da nije svejedno hoćemo li brojeva 8 i 3 podijeliti kao cijele brojeve ili kao realne.

Pravila za izvršavanje mješovitih izraza:

- ako su oba operanda, na koje se odnosi operator istog tipa (oba su cijeli ili realni brojevi), operator će se izvršiti kao operator nad tim tipom podataka
- ako operandi nisu istog tipa (jedan je cijeli a drugi realni broj), prilikom izračunavanja će se cijeli broj pretvoriti u realni

Primjer 2 – 12:

Kolika će biti vrijednost izraza `5 / 3 + 3.5`

Budući da su brojevi 5 i 3 oba cijeli brojevi, vrijednost operatora 5 /3 bit će 1, te je izraz poprimio oblik 1 + 3.5. Kako je 3.5 realan broj a 1 cijeli, broj jedan će isto postati realan i tada će izraz imati oblik 3.5 + 1.0 = **4.5**

Konverzija tipova (casting)

U prethodnom smo odjeljku naučili kako se izračunavaju mješoviti izrazi. Međutim često puta ćemo trebati neki tip podataka konvertirati u neki drugi (realni broj u cijeli ili obrnuto).
Java omogućava konverziju tipova podataka pomoću posebnog operatora i to na sljedeći način:

```
(tip) (izraz);
```

Ukoliko realni broj pretvaramo u cijeli, realnom će se broju jednostavno odbaciti decimale.

Primjer 2 – 13:

Izraz	Vrijednost
`(int) (3.14);`	`3`
`(float) (10);`	`10.0`
`(float) (10) / 4;`	`= 10.0 / 4 = 2.5`
`(float) (10 / 4);`	`= float (2) = 2.0`

Osim konverzije cijelih brojeva u realne i obrnuto, moguće je konvertirati i znakove u cijele brojeve i obrnuto. Pretvaranje znakova u cijele brojeve i obrnuto temelji se na Unicode tablici gdje je svaki broj predstavljen odgovarajućim brojem (kodom). Tako je npr. `(int) ('b')` 98 ('b' je na 98. mjestu u Unicode tablici), `(int ) ('8')` je 56,... Analogno tome je `(char) (98)` 'b'.

Prvi programi

Sada, nakon što smo naučili osnovno identifikatorima, tipovima podataka, inicijalizaciji te operatorima, možemo započeti s pisanjem jednostavnih programa u Javi. Za nas će u ovom trenutku program biti jedna metoda unutar neke klase, koju ćemo pokretati direktno i prosljeđivati joj vrijednosti preko dijaloškog prozora (*Slika 2 - 11*).

Svaki naš "program" će u ovom trenutku imati sljedeći oblik:

```
public class ime_klase
{
      public static tip ime_metode(parametri)
      {
            kod metode;
            return vrijednost;
      }
}
```

Primjer 2 – 14:
Napišimo metodu kojoj ćemo prosljeđivati duljine stranica pravokutnika (cijeli brojevi), a koja će vraćati površinu tog pravokutnika.

Rješenje:
Budući da se radi o pravokutniku čije su stranice cijeli brojevi, površina tog pravokutnika bit će isto tako cijeli broj. Budući da naša metoda vraća površinu (cijeli broj), zaglavlje metode (prvi red) će biti:

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

public i **static** ćemo za sada pisati u svim primjerima, a detaljno ćemo ih objasniti kasnije.
int je tip podataka koji metoda vraća (cijeli broj)
povrsina je ime metode
int a, int b su popisi parametara koji se prosljeđuju metodi (duljine stranica pravokutnika)
Budući da je površina pravokutnika jednaka umnošku duljina stranica, deklarirat ćemo varijablu P (cjelobrojnog tipa), u koju ćemo zapisati površinu pravokutnika.

```
int P;
P = a * b;
```

Na kraju će metoda vratiti površinu, naredbom: **return** P;

Dakle naš program će imati sljedeći oblik:

```
public class pravokutnik
{
      public static int povrsina(int a, int b)
      {
            int P;
            P = a * b;
            return P;
      }
}
```

Metodu ćemo nadalje pokrenuti na način kao što je to opisano na početku poglavlja.

Primjer 2 – 15:
Napišimo metodu koja će unositi duljine stranica trokuta i vraćati površinu tog trokuta.

Površinu trokuta, čije su stranice a, b i c računat ćemo Heronovom formulom:

$P = \sqrt{s(s-a)(s-b)(s-c)}$, pri čemu je $s = \frac{a+b+c}{2}$.

Drugi korijen iz realnog broja računat ćemo naredbom Math.Sqrt ().
Dakle, rješenje će biti:

```
public static double povrsina(int a, int b, int c)
{
      double P, s;
      s = (a + b + c) / 2;
      P = Math.sqrt(s * (s - a) * (s - b) * ( s - c));
      return P;
}
```

U prošlom smo primjeru koristili naredbu **`Math.sqrt ()`**. Ova naredba računa drugi korijen iz realnog broja. Vrijednost koju ova naredba vraća je tipa **`double`**. Općenito **`Math`** je gotova Javina klasa koja sadrži metode za rad s matematičkim funkcijama. Neke od metoda klase **`Math`** dane su u sljedećoj tablici:

Metoda	Opis
`abs (x)`	apsolutna vrijednost broja `x`
`ceil (x)`	najmanji cijeli broj koji je veći ili jednak od realnog broja `x` (vrijednost koju vraća je tipa **`double`**)
`exp (x)`	vraća vrijednost funkcije e^x
`floor (x)`	najveći cijeli broj koji je manji ili jednak od realnog broja `x` (vrijednost koju vraća je tipa **`double`**)
`log (x)`	prirodni logaritam broja `x`
`max (x, y)`	veći od brojeva `x` i `y`
`min (x, y)`	manji od brojeva `x` i `y`
`random ()`	vraća slučajan broj između 0.0 i 1.0
`round (x)`	cijeli broj koji je najbliži realnom broju `x` (vrijednost koju vraća je tipa **`double`**)
`sqrt (x)`	drugi korijen iz broja `x`
`cos (x)`	kosinus kuta `x` (`x` je u radijanima)
`sin (x)`	sinus kuta `x` (`x` je u radijanima)
`tan (x)`	tangens kuta `x` (`x` je u radijanima)

Tablica 2 – 10 Neke od metoda klase **`Math`**.