Uvod

Vrlo je teško ustanoviti odakle početi s objašnjavanjem termina i suštine programiranja. Pretpostavljam da znate makar neke osnove matematike, i naravno koristiti računar...

Dosta vremena ćemo posvetiti objašnjavanju koncepata i terminologije. Terminologija je možda i najbitnija jer ćete kasnije lakše razgovarati o datim konceptima i brže ćete učiti ostale programske jezike, "povezivati kockice"...

Komunikacija

Mi ljudi za sebe tvrdimo da smo inteligentna bića. Jedan od dokaza za to je i mnoštvo naših načina komunikacije. Komunikacija podrazumijeva razmjenu informacija između dva ili više učesnika (ne moraju biti ljudi), koristeći zajedničke znakove i pravila.
U ovo se ubrajaju vizuelne metode (znakovni jezici, grimase, kolutanje očima), audio metode (govor, muzika), fizičke (kad vas neko udari jer ste nepristojni npr.).
Danas u svijetu postoji više od 5.000 jezika, od kojih većina nažalost lagano izumire.

Ljudi pričaju i sa robotima, ako ništa gledali ste futurističke filmove.
Ako se nas ljude pita, to je idealan način za komuniciranje.
Međutim, mašine su u suštini vrlo glupe, nemaju inteligenciju kao živa bića.

Mašine nemaju mozak, tijelo, uši, glasne žice... Sve im to moramo obezbijediti, da bi bile "inteligentne". Neki od tih organa su predstavljeni kompjuterom, mehaničkim dijelovima, mikrofonom, zvučnikom itd.

Kompjuter je mozak mašine, on upravlja svime. Na njega su povezani svi ostali uređaji (organi).
Nama ljudima misli dolaze same od sebe (većini, neki ljudi ne misle nikako), ali kompjuterima ne. I to im moramo "ručno" unijeti... Te kompjuterske misli su ustvari naše naredbe, koje se skupno nazivaju program.

Ovdje smo da naučimo neke od načina za komuniciranje s kompjuterima.
Naime, ljudi su razvili na stotine programskih jezika koje se koriste za pisanje kompjuterskih programa.
Ovi jezici su slični našim jezicima, imaju svoju sintaksu, gramatiku i pravopis.

Apstrakcije

Kroz život, svjesno ili nesvjesno, naučili smo koristiti apstrakcije. Apstrakcija je ustvari generalizovanje, uopćavanje nečeg konkretnog. U datom kontekstu bitna su nam samo neka svojstva datog objekta/stvari/koncepta. Naprimjer kada kažemo automobil, podrazumijevamo nešto sa 4 točka, motorom itd.
To je apstraktni automobil, njegova takoreći definicija, dok je konkretan automobil npr. komšijin Pežo(v) 307, iz 2001. godine.

U fizici i hemiji imamo apstrakciju atoma. Zatim se uvodi apstrakcija molekule, koja se sastoje od više atoma. Kada se govori o molekulama, znamo da su "ispod" atomi, ali nas u datom kontekstu to ne zanima! Kontekst predstavlja skup okolnosti i činjenica, još jedna vrlo bitna stvar kada se govori o apstrakcijama.

U matematici imamo apstrakciju broja. Kada kažemo broj obično mislimo na cijeli broj. Ali ne kažemo KOJI broj!?
Dakle, samo znamo da mislimo na neki broj.

Paradigme

Jedan od značajnijih problema koje p.j. pokušavaju riješiti je kompleksnost.
Kako programi postaju veći, to su komplikovaniji za razumjeti.
Zato danas imamo nekoliko paradigmi (pristupa, načina) za razvijanje programa, od kojih su najpoznatije:

- proceduralno
- funkcionalno
- objektno orijentisano
- logičko

Dosta jezika danas je miks dvije ili više ovih paradigmi.
Ove paradigme usmjeravaju naš način razmišljanja o problemima po svom nekom "šablonu" /fazonu.

Npr. kod proceduralnog/imperativnog načina razmišljanja mijenjamo varijable "u mjestu". Ovdje moramo misliti koja je vrijednost neke varijable u datom trenutku, što je većini ljudi izazovan zadatak.

Dok kod funkcionalnog/matematičkog načina razmišljanja pravimo nove varijable a stare ne diramo! To nam pomaže kod razumijevanja određenog dijela koda, ne moramo pamtiti u glavi stanje "cijelog svijeta" i na šta utiče promjena date varijable. Ali, otom potom...

Programi

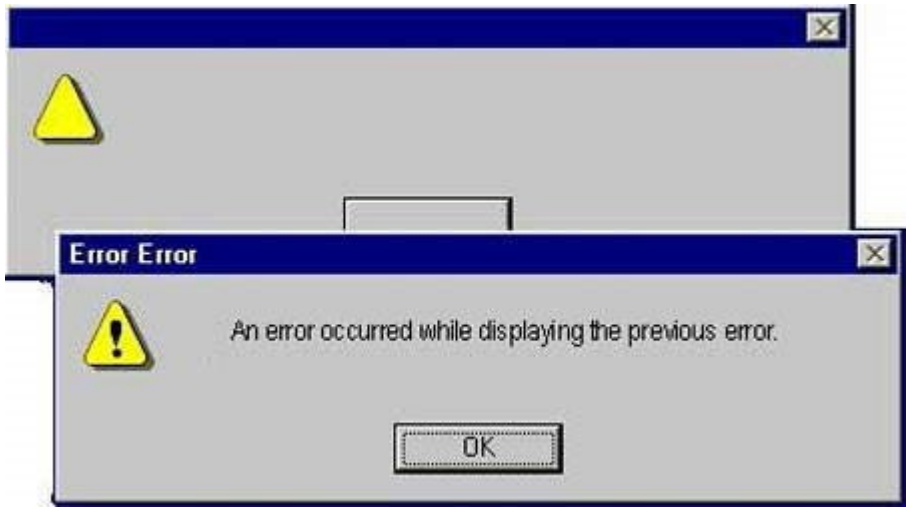
Rekli smo da je program niz naredbi koje računar treba izvršiti, u cilju dobijanja nekog rezultata.
Kako svi ljudi razmišljaju na sebi svojstven način, postoji teoretski beskonačno mnogo rješenja (programa) za jedan te isti problem!

Pored gore navedenih apstrakcija, programi često komuniciraju i sa "vanjskim svijetom".
Npr. ako trebamo učitati neki tekstualni fajl i prebrojati sve riječi, taj fajl nije dio programa već je ulaz (en. input) u naš program. Isto vrijedi i za klik miša, unos sa tastature, mikrofona i sl.

Analogno, kada želimo npr. ispisati nešto na ekran to se zove izlaz (en. output) iz našeg programa.

Naravno, da ne bismo "izmišljali toplu vodu" možemo se okoristiti znanjem stečenim kroz ljudsku historiju.
Ljudi su nadošli na koncept algoritma, koji predstavlja nedvosmislenu specifikaciju za rješavanje nekog problema.
Tj. Algoritam je niz precizno definisanih koraka za dobijanje nekog rezultata.
Laički rečeno, to je recept za implementaciju neke funkcije.

U toku pisanja koda nekog programa često će nam se pojavljivati logičke greške.
Tj. program se izvršava ali ne radi kako bismo željeli. Te greške se nazivaju bagovi (engl. bug - buba, insekt).
Za program koji ima takve greške kaže se da je bagovit.
Proces u kojem tražimo greške i ispravljamo ih naziva se debagovanje.

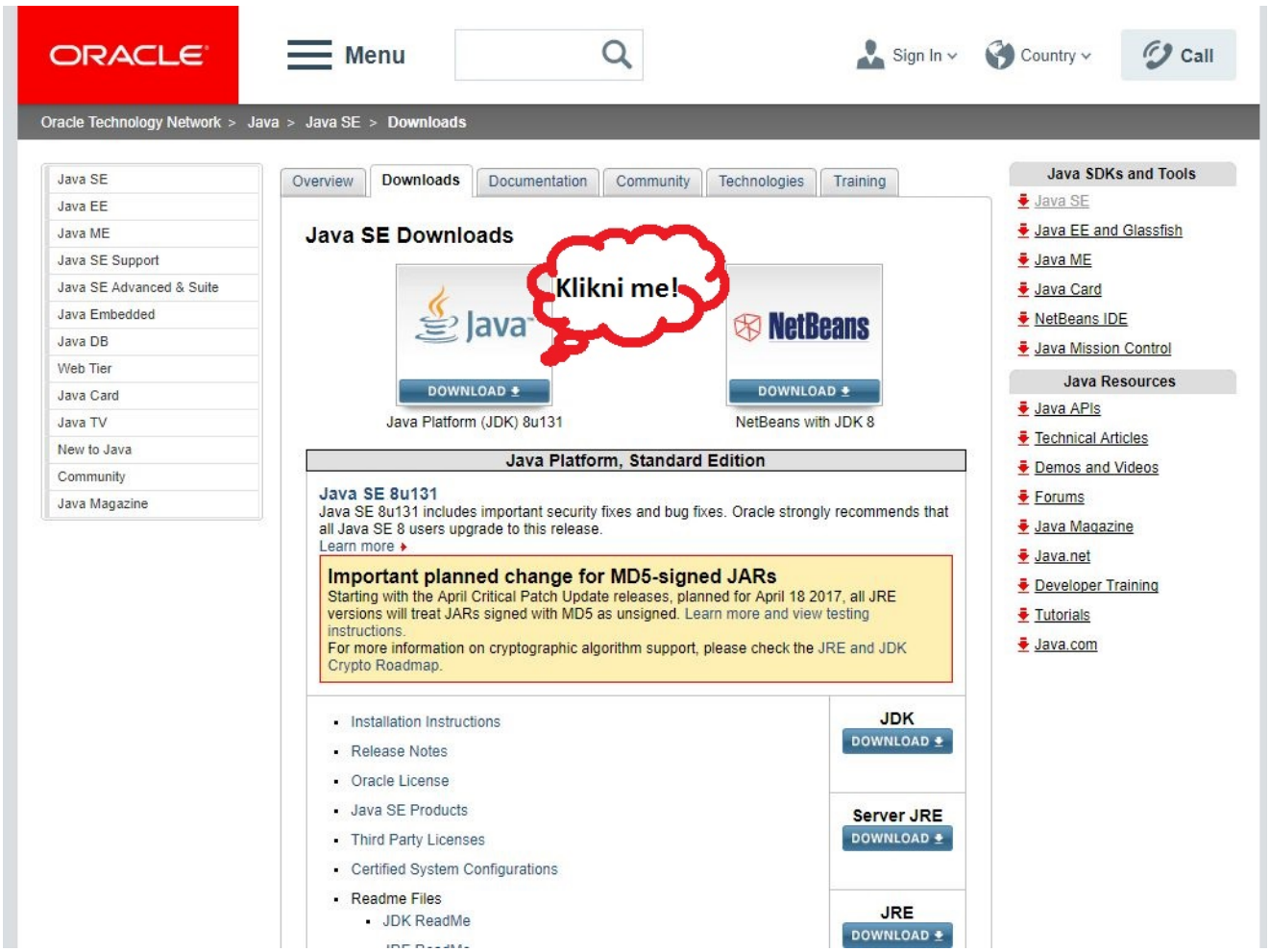


To je to.

Instalacija Jave

U nastavku slijedi primjer za instalaciju Jave 8.
Java 11 se samo raspakuje kao obični zip.
Nastavak za dodavanje PATH varijable je identičan.

Odaberite jednu od sljedećih verzija: Java 8 | Java 11.

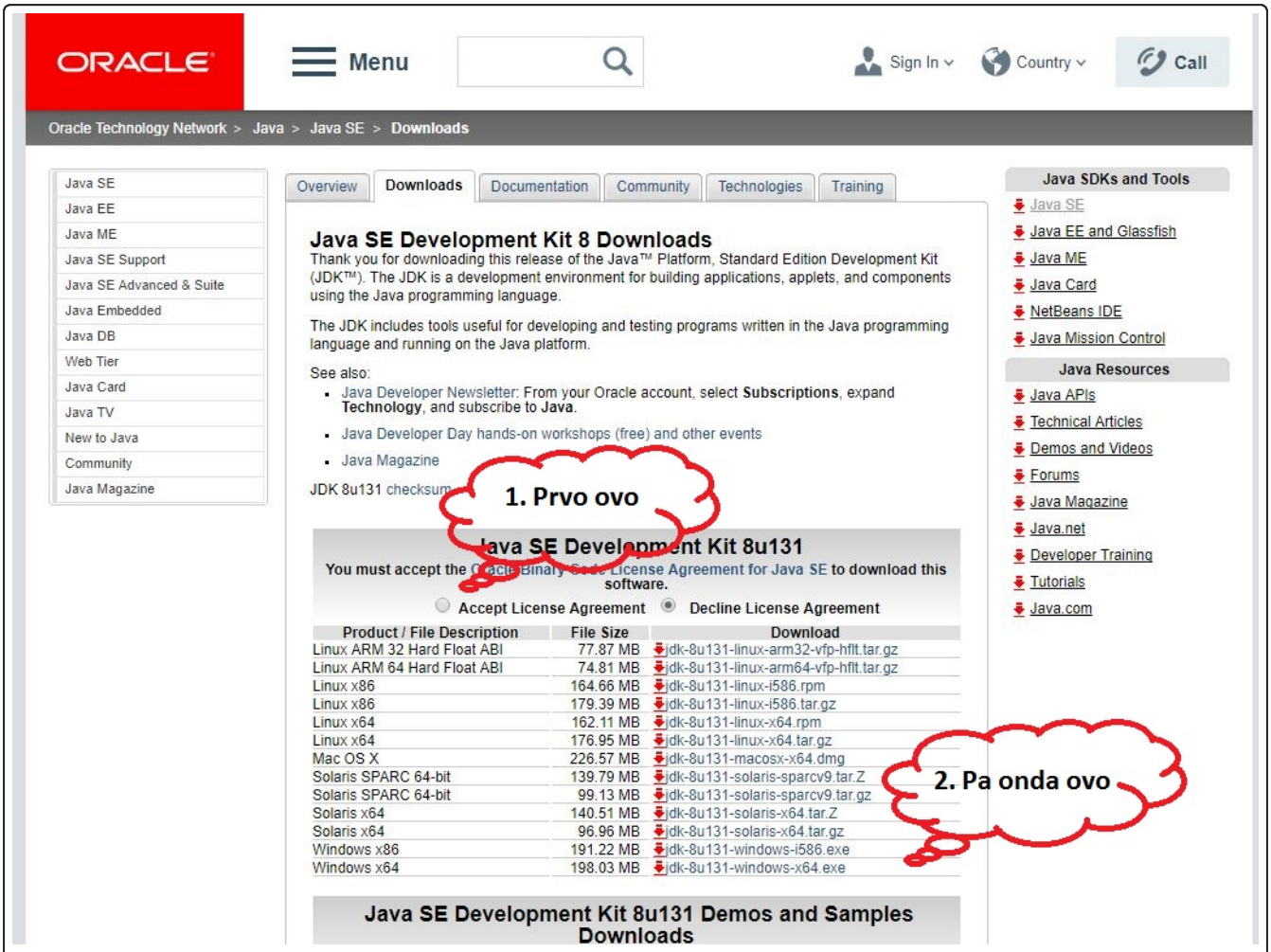


JDK download step 1

Kada se otvori stranica sa prethodne slike kliknite na dugme download JDK.

> JDK (Java Development Kit) su alati za razvijanje Java programa: kompajler, debager itd. JDK sadrži JRE (Java Runtime Environment) koji služi za pokretanje Java programa (JVM, Java API klase itd).

Na sljedećoj slici kliknite na "jdk-8u131-windows-x64.exe".
To je instalacija za 64-bitni Windows.
Ako znate da imate 32-bitni Windows onda skinite tu verziju...

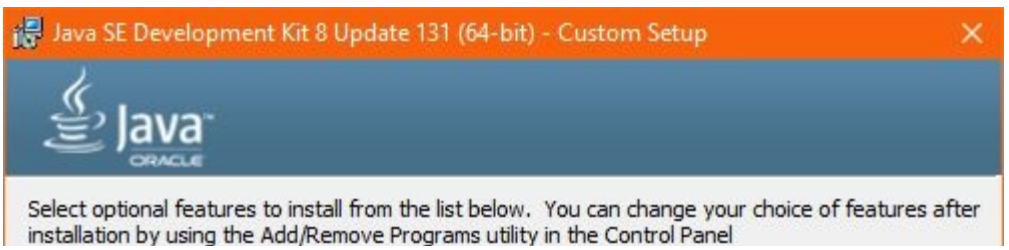


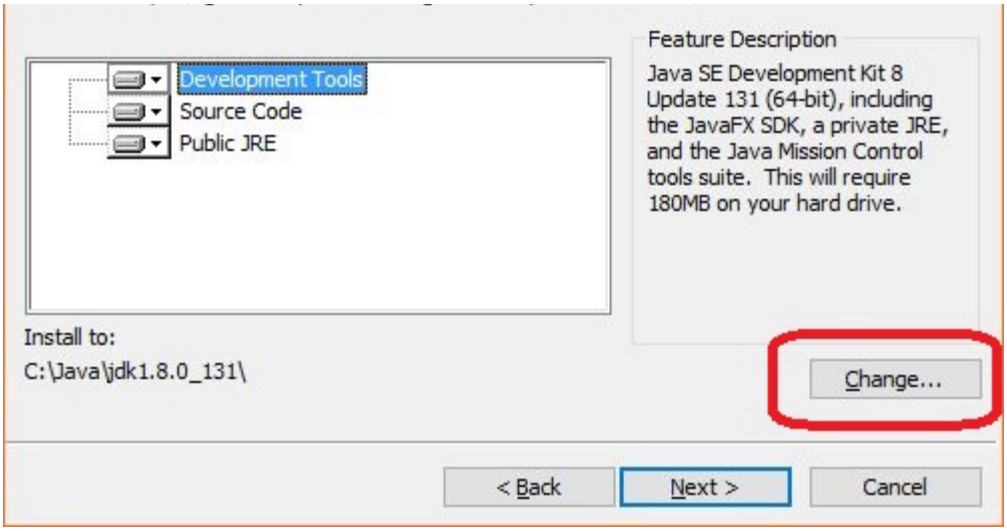
JDK download step 2

Preporučujem da Javu instalirate u folder pod nazivom "C:\Java".
Ovo ne morate raditi ali je korisno kada vam treba više verzija Jave.
Također, neki programi na Windowsu imaju problema kada putanja sadrži razmak, zato nećemo instalirati u "Program Files"...

> Napomena: Ne trebate instalirati i JRE!
> Kada vam dođe prozor za JRE kliknite Cancel! (iksić).
> Upravo sam vam uštedio 100-tinjak megabajta, nema na čemu... :D

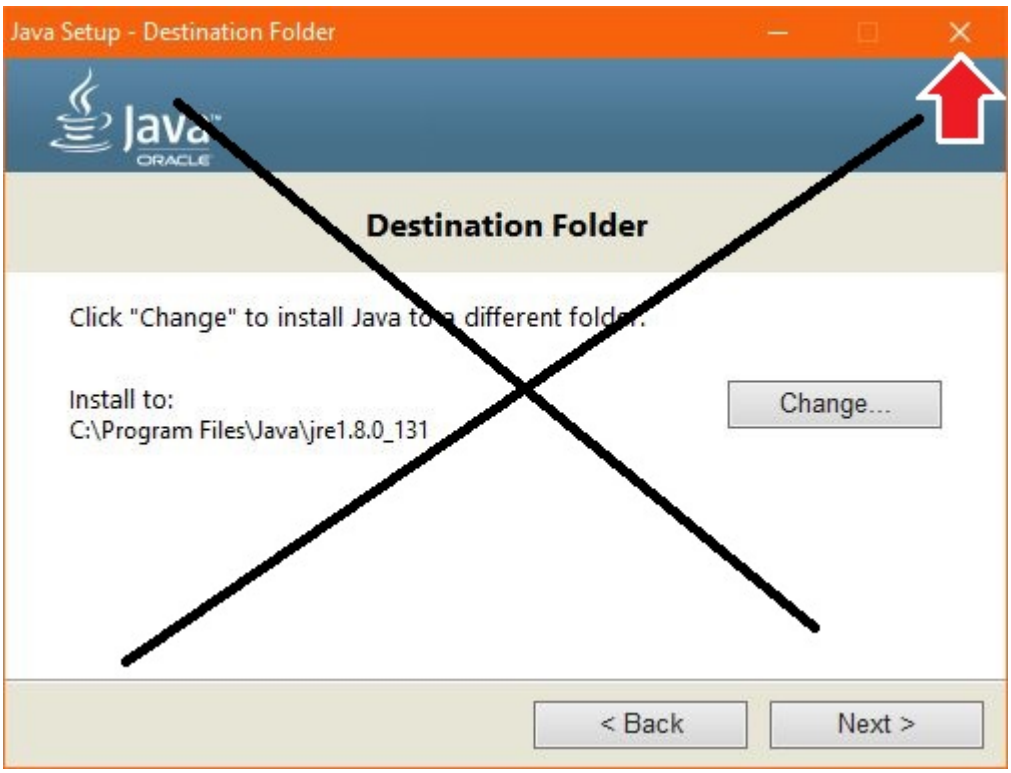
Dakle, kada se završi download, otvorite instalaciju i kliknite Next.





JDK folder

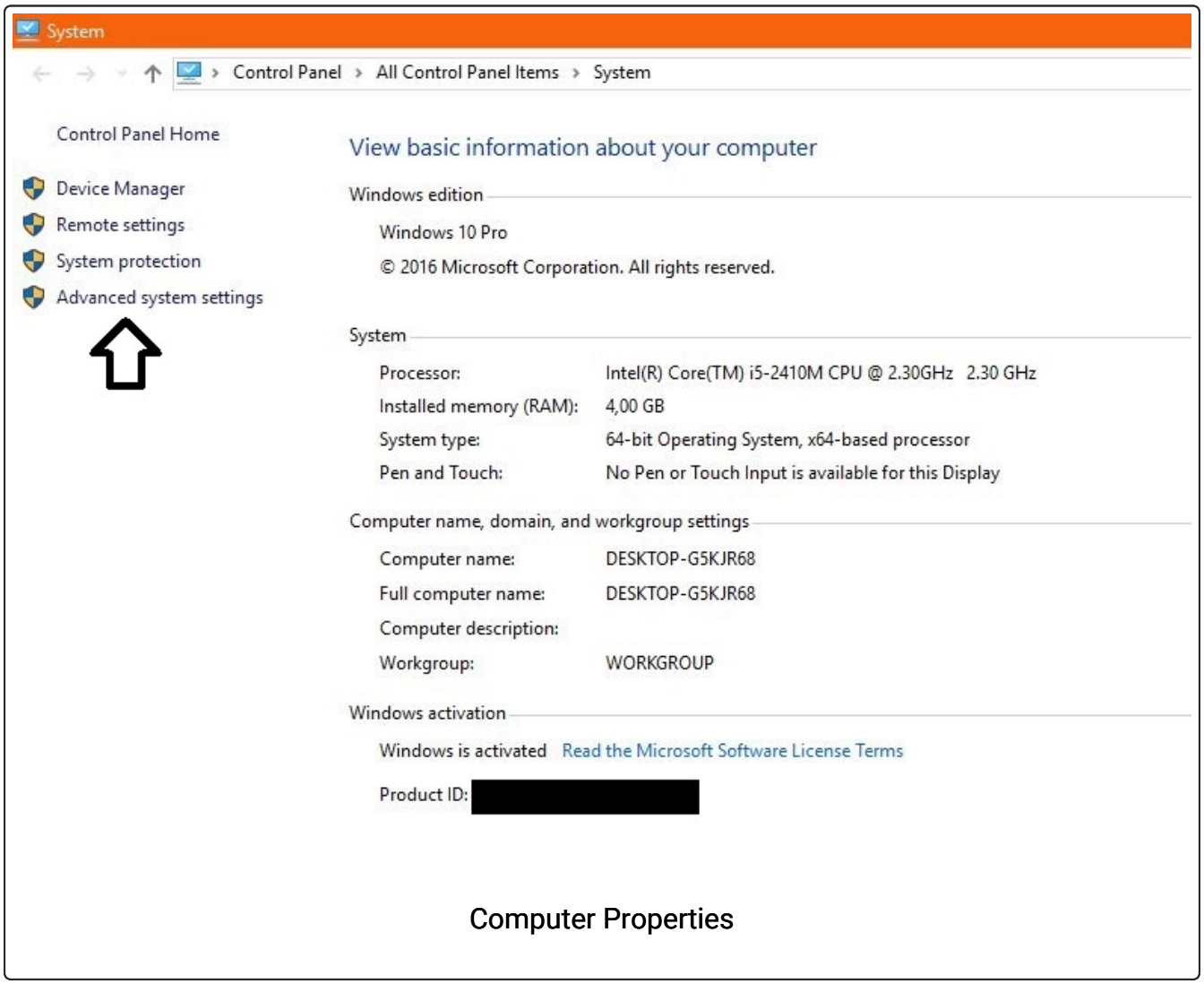
Zatim kliknite dugme "Change..." i prepravite putanju foldera na "C:\Java\jdk1.8.0_131\".
JRE će već biti instaliran zajedno sa JDK! Tako da nam ne treba još jedna instalacija...



JRE, nope!

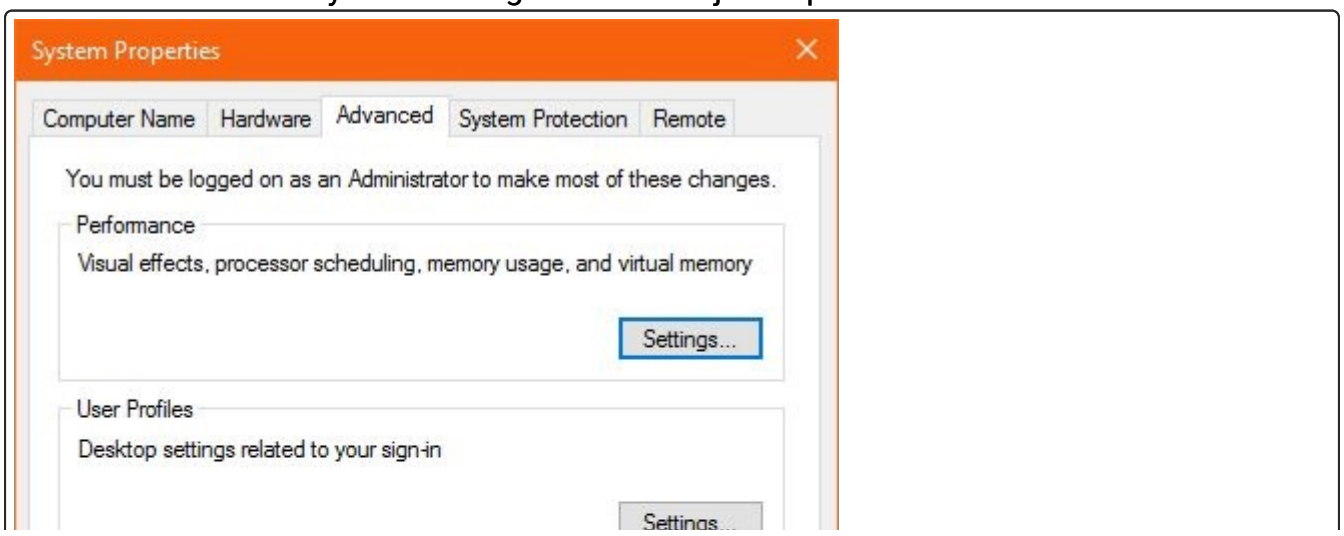
Dodavanje PATH varijabli

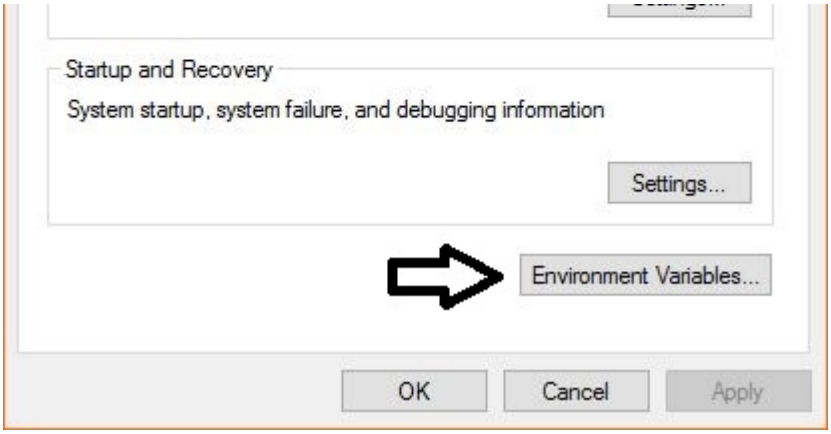
Dio koji slijedi vezan je više za sam rad operativnog sistema (Windows u našem slučaju) nego za Javu.
Otvorite `Computer Properties` na Windowsu (desni klik na My Computer -> Properties).
Otvoriće vam se sljedeći prozor:



Computer Properties

Kliknite na Advanced System Settings. Dobićete sljedeći prozor:





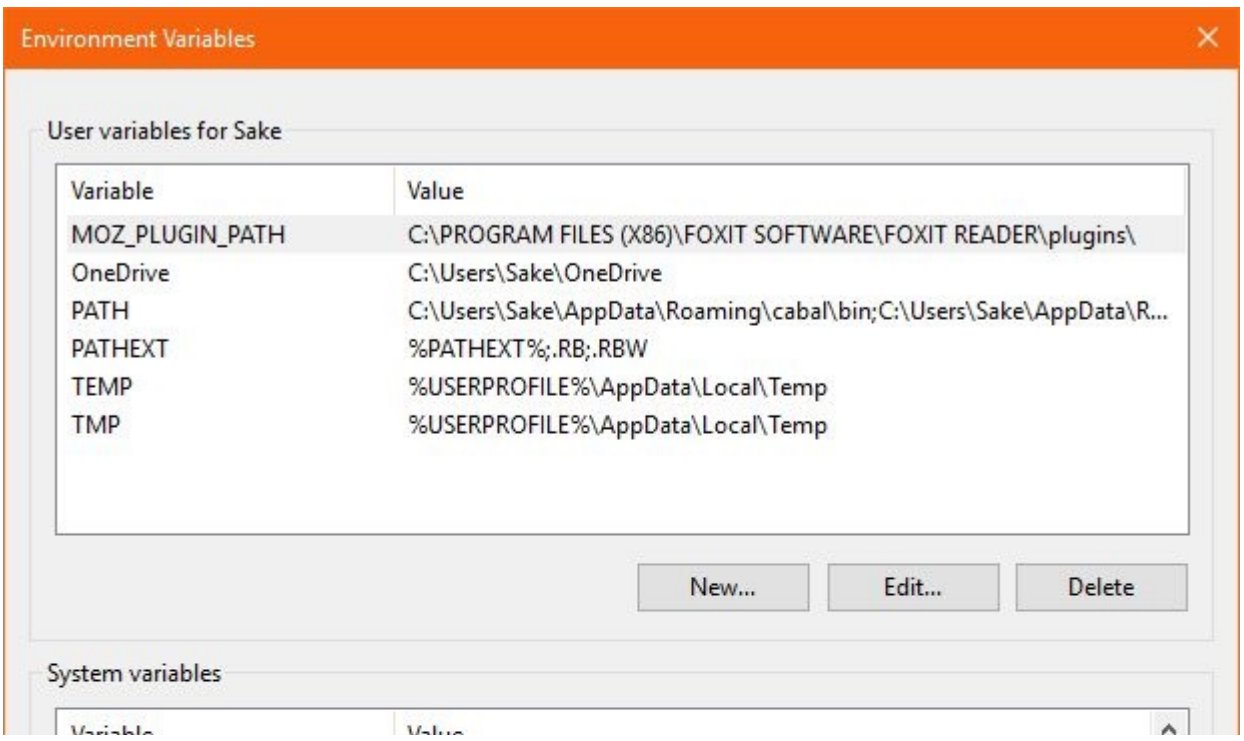
System Properties

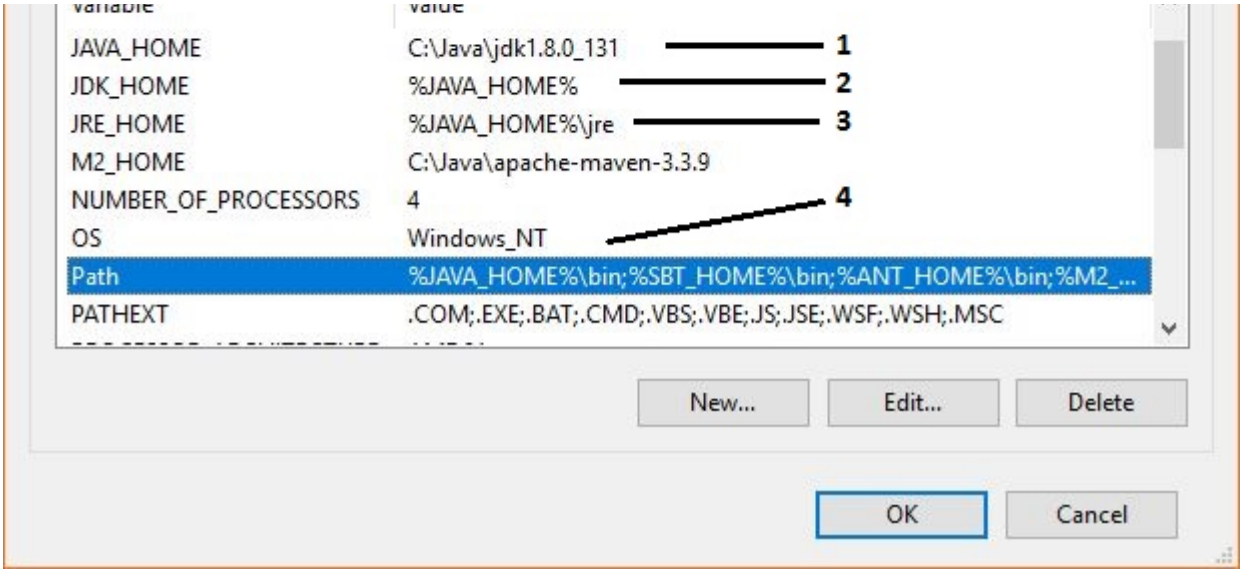
Otvorite `Advanced` tab i kliknite `Environment Variables...`.
Dodajte 3 sistemske varijable u donji prozor (u System Variables):

Naziv varijable	Vrijednost
JAVA_HOME	C:\Java\jdk1.8.0_131
JDK_HOME	%JAVA_HOME%
JRE_HOME	%JAVA_HOME%\jre

Glavno, otvorite "PATH" varijablu i dodajte ";%JAVA_HOME%\bin" na kraj.
Nemojte zaboravit tačkazarez kopirati! :D
Bez navodnika, naravno! Kliknite Ok, Ok...

To je to! Uspješno ste instalirali Javu! :)
Stanje bi trebalo biti otprilike ovako:



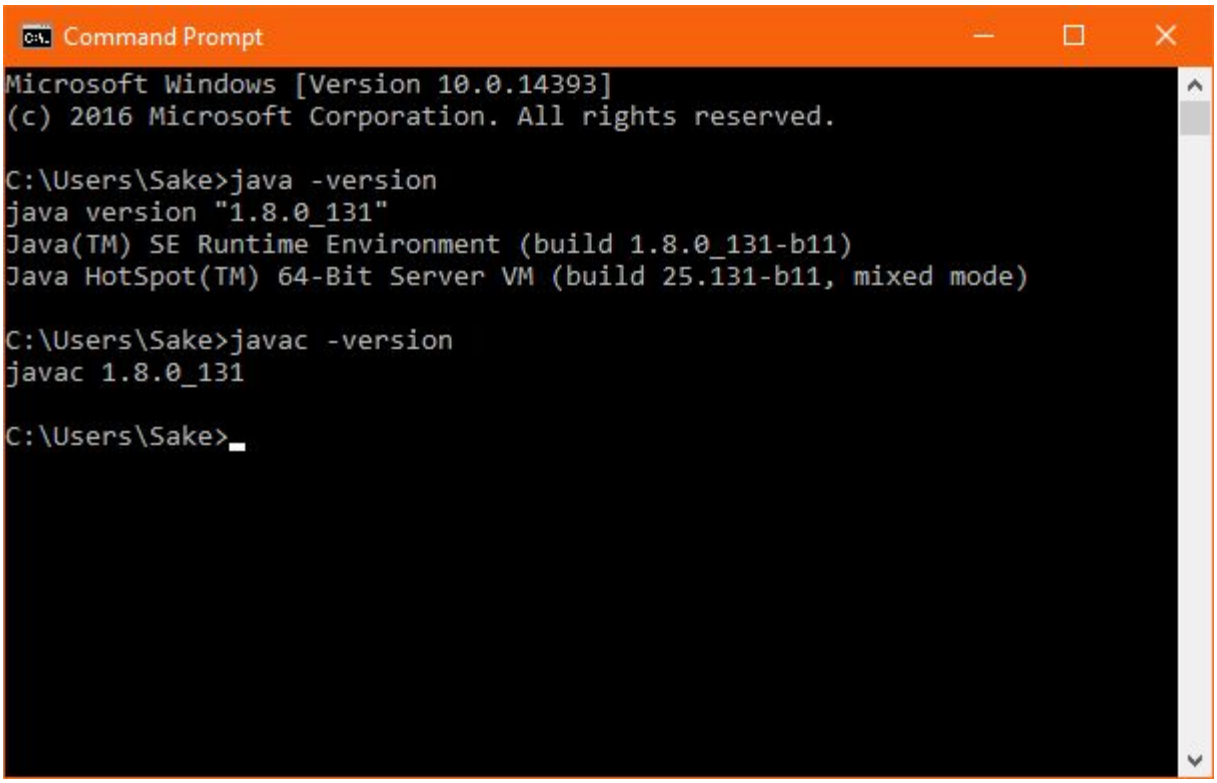


System Properties

Provjera

Da bi provjerili jesmo li uspješno obavili zadatak, otvorićemo Command Prompt (kod Linuxaša se rekne Shell).
Otvorite Start, ukucajte "cmd" i kliknite Enter. Ukucajte `java -version`.
Ako dobijete poruku `'java' is not recognized as an internal or external command, operable program or batch file.` nešto nije uredu, provjerite sve korake sekcije Dodavanje PATH varijabli ponovo!

Ako dobijete nešto slično sljedećoj slici, sve je OK:



Java verzija

JShell

Morate imati instaliranu Javu verzije 9 ili više. Ispratite sekciju za instalaciju.

Koristićemo JShell program za upoznavanje i igranje s Java jezikom.
Otvorite konzolu (bez ikakvog straha! :D), ukucajte `jshell` i udarite Enter.
Dobićete tzv. interaktivni shell u kojem možete kucati Javu.
Zovu ga još i REPL (Read Evaluate Print Loop), jer dobija unos od nas, evaluira ga, isprinta neki odgovor, i ponavlja ukrug.

Izrazi i operatori

Za početak, JShell možemo koristiti kao primitivni kalkulator.
Tako ćemo steći osjećaj kako se ponašaju brojevi u Javi, i usput objasniti neke osnovne koncepte i pojmove.
Ako ukucamo u REPL broj 5 i pritisnemo Enter, vidjećemo sljedeće:

```
jshell> 5
$1 ==> 5
```

Batch Copy

REPL je razumio i odgovorio nam da je to vrijednost `5`, jer je vrlo inteligentan.
U programiranju se kaže da je broj izraz. Izraz u programiranju ima isto značenje kao i u matematici.
Dakle, misli se na nešto što ima vrijednost. Nešto što možemo prikazati, s čim možemo baratati.

Ako se opet prisjetimo (dosadne) matematike, sjetićemo se da se simboli `+`, `-`, `*`, itd. nazivaju operatori.
Pošto smo rekli da je REPL kao kalkulator, da vidimo šta sve možemo uraditi:

```
jshell> 2 + 2
$2 ==> 4
jshell> 2 - 5
$3 ==> -3
jshell> 7 * 3
$4 ==> 21
jshell> 9 / 4
$5 ==> 2
```

Batch Copy

Izgleda da dijeljenje ne radi kako treba... Našli smo grešku! :D
Ali, nije tako. Java podrazumijeva da želimo podijeliti dva cijela broja!
Ako se sjetimo osnovne škole, `5 / 2` je ustvari `2` cijelih i ostatak `1`. :)
Dakle, kada dijelimo dva cijela broja, ustvari radimo "cjelobrojno dijeljenje".

Također, možemo dobiti i ostatak od dijeljenja, pomoću operatora `%`.
Npr. `5 % 2` je `1`.

Kada želimo koristiti realne brojeve (sa zarezom), onda moramo poslije broja napisati tačku.
Tačka se koristi jer se znak zarez koristi za neke druge stvari... Otom potom.
Uglavnom, da vidimo kako rade realni brojevi (en. floating point):

```
jshell> 3.0 + 5.1
$6 ==> 8.1
jshell> 7.3 - 2.9
$7 ==> 4.4
jshell> 5.5 * -2.0
$8 ==> -11.0
jshell> 9.0 / 4.0
$9 ==> 2.25
```

Batch Copy

Sada se sve čini onako kako treba. :)

Redoslijed operatora (en. precedence) radi očekivano.
Npr. množenje se radi prije sabiranja.
Ako želimo baš, baš, biti sigurni da će program raditi kako treba, možemo koristiti zagrade.

```
jshell> 2 + 2 * 2
$10 ==> 6
jshell> (2 + 2) * 2
$11 ==> 8
```

Batch Copy

Varijable

Primijetite šta REPL ispisuje kao odgovor, poslije svake unesene linije, `$1 ==> 5`.
Ako ukucamo `$1` u REPL vidjećemo sljedeće:

```
jshell> $1
$1 ==> 5
```

Batch Copy

Dakle, REPL nam kaže da je vrijednost varijable `$1` jednaka `5`.
Izraz `5` je spremljen u varijablu `$1`, koju možemo koristiti umjesto broja `5`.

> Šta je to varijabla?
> Zamislite da imate kutije raznih oblika i dimenzija.
> Varijabla je baš kao kutija sa natpisom koji vi želite. Npr. kutija u kojoj čuvamo neki broj.

Kreiraćemo novu varijablu pod nazivom `brojJabuka`, za čuvanje `int` egera (en. "integer" je cijeli broj).
Ovo `int` je tip varijable (veličina i oblik kutije)! Kasnije ćemo govoriti malo detaljnije o tipovima.

```
jshell> int brojJabuka;
brojJabuka ==> 0
|  created variable brojJabuka : int
```

Batch Copy

Pošto nismo dali vrijednost našoj varijabli, Java nam pomaže tako što postavlja `brojJabuka` na vrijednost `0`.
Obično je to ono što želimo, ali je najbolje postaviti početnu vrijednost eksplicitno, čim uvodimo novu varijablu. Tako odmah znamo koja joj je početna vrijednost i ne moramo razmišljati o tome... :)
To radimo ovako:

```
jshell> int brojJabuka = 0;
brojJabuka ==> 0
|  created variable brojJabuka : int
```

Batch Copy

Vrlo jednostavno, nakon naziva varijable slijedi znak jednako i vrijednost, ovdje je to `0`.
Primijetite da ovaj znak jednakosti nije matematička jednakost već operacija dodjele!
Tj. kao da kažemo "postavi varijablu x na nulu!".
U nekim jezicima postoji manje zbunjujuća sintaksa za ovo, znak `:=` (dvotačka-jednako), kao npr. Pascal.

Varijablu možemo izmijeniti, tj. postaviti na novu vrijednost:

```
jshell> brojJabuka = 5;
brojJabuka ==> 5
|  assigned to brojJabuka : int
```

Batch Copy

Sada je vrijednost promijenjena na `5`. Nakon ove linije, gdje god koristimo `brojJabuka` to je kao da napišemo `5`.

Također, možemo iskoristiti prethodnu vrijednost varijable da bismo dobili novu:

```
brojJabuka = brojJabuka + 1;
brojJabuka ==> 6
```

Batch Copy

```
jshell> brojJabuka += 1;
$23 ==> 7
jshell>
```

U izrazu `brojJabuka + 1`, vrijednost `brojJabuka` odnosi se na zadnju vrijednost varijable, dakle vrijednost prije ove linije, tj. `5`.

Postoji i skraćena verzija za ovo, prikazana na liniji 3, s operatorom `+=`.
Također imaju skraćene verzije i za oduzimanje, množenje, dijeljenje i modulo: `-=`, `*=`, `/=`, `%=` respektivno.
Ovo "respektivno" je fancy izraz za "u navedenom redoslijedu". :D

Nazivi varijabli

Nazivi varijabli i tipova (en. identifiers) ne mogu imati razmake u sebi, ne mogu biti nijedna od rezervisanih riječi kao što je `class`, `int`.
Ne mogu biti neki rezervisani znakovi, kao npr. `,`, `&` itd.
Ne mogu počinjati brojem. Primjeri nevalidnih naziva varijabli (isprobajte u JShellu):

- int my variable = 5
- int int = 5
- int , = 5
- int 5something = 5

Primjeri validnih naziva varijabli:

- int _abc = 5
- int $xyz = 5
- int Z = 5
- int t$df_x = 5

Mogu se koristiti simboli `_` i `$` ali ih treba izbjegavati, pogotovo kao prvi karakter u nazivu varijable!

Preporučeno je (konvencija, dogovor, dobra praksa) da varijable počinju malim slovom, da su camelCase (svaka iduća riječ počinje velikim slovom), npr. `myVariable`.
Konstante (varijable koje se neće mijenjati) se pišu velikim slovima i riječi su razdvojene donjom crticom, npr. `MY_CONST`.

Tipovi

Većina ozbiljnih jezika ima tipove. One koji nemaju slobodno izbjegavajte! ;)
Tip je apstrakcija koju dati p.j. razumije. Npr. kada napišemo `int broj`, ovo "int" je tip varijable "broj".
Tako će kompjuter znati, između ostalog, koliko memorije da rezerviše za tu varijablu.
Pošto smo mu dali tu informaciju, onda nam može i ukazati na greške u programu koji pišemo. Npr. da ne možemo sabirati broj i slovo, kruške i jabuke...
Također nas može i usmjeriti, npr. reći nam koje su operacije dostupne nad datim tipom, da ne moramo nagađati.

Da vidimo JShell u akciji:

```
jshell> int broj = 6;
broj ==> 6
jshell> broj = 5.5;
|  Error:
|  incompatible types: possible lossy conversion from double to int
|  broj = 5.5;
|         ^-^
```

Dakle, Java nam ne da da dodijelimo broj `5.5` varijabli koja je tipa `int` (cijeli broj).
To ima smisla itekako, jer bismo izgubili ovo "zarez pet"... :)
Ako želimo da budemo pametniji od Jave, i da na silu "utrpamo" `5.5` u `int` to ide ovako:

```
jshell> broj = (int) 5.5;
broj ==> 5
```

I šta smo dobili? Samo cijeli dio... Ali, nekad nam ovo baš i treba!
Uglavnom, pouka je da trebamo slušati šta nam Java poručuje!
To radi za naše dobro, nju nije briga ako je rezultat netačan, ako je to što želimo... :D

Naravno, imamo još puno toga reći o tipovima, tek smo "zagrebali površinu"!
Npr. kako da grupišemo podatke? Kompleksni brojevi, podaci o ljudima, proizvodima itd?
O tom ćemo pričati u poglavlju o klasama. :)

Stringovi i karakteri

Prije nego nastavimo dalje, moramo objasniti i stringove, jer se vrlo često koriste.
String (iz engl. niz, kanafa) koristi se za predstavljanje teksta, tj. niza karaktera.
Karakter (en. character je simbol) označava jedan simbol, bio to broj, slovo ili neki simbol, emoji...

Stringovi se pišu s dvostrukim navodnicima:

```
jshell> "Poyy sviete!"
$1 ==> "Poyy sviete!"
```

Batch Copy

Stringovi se tretiraju specijalno u Javi, pa imamo i operator "sabiranja" nad njima.
Npr. možemo "sabrati" 2 stringa, string s brojem i sl:

```
jshell> "Ime" + "Prezime"
$2 ==> "ImePrezime"
jshell> "x je " + 5
$3 ==> "x je 5"
jshell> "x je " + 5 + ", i kad se pomnoži s " + 3 + " daje " + (5*3)
$4 ==> "x je 5, i kad se pomnoži s 3 daje 15"
jshell> "strin" + 'g'
$5 ==> "string"
jshell> String s = "ABC"
s ==> "ABC"
```

Batch Copy

Ovo sabiranje se naziva konkatenacija (engl. concatenate je nadovezati).
Radi dvosmjerno, možemo sabrati i broj sa stringom, npr. `5 + "hepek.abc"`.
String se deklariše s velikim početnim slovom!

Implementacija `String`a u Javi interno koristi `char` actere.
Kao što smo već rekli, `char` je samo jedan karakter.
Piše se s jednostrukim navodnicima:

```
'a'
$9 ==> 'a'
''
|  Error:
|  empty character literal
|  ''
|  ^
'ab'
|  Error:
|  unclosed character literal
|  'ab'
|  ^
```

Batch Copy

```
jshell> char slovoA = 'A'
slovoA ==> 'A'

jshell>
```

Dakle, karakter ne može biti prazan, niti može sadržati više karaktera.
Za ove namjene koristi se `String` ! :)

```
jshell>
```

Vrlo važne napomene:

- string "1" nije isto kao broj 1
- karakter '1' nije isto kao broj 1
- postoje posebne funkcije za pretvaranje iz jednog tipa u drugi, ali otom potom...

Naredbe

Naredbe (en. statements) su kao rečenice u govornom jeziku, i obično su u imperativnom obliku, kao da se obraćamo kompjuteru: uradi ovo, izbriši ono, upiši u fajl itd.
Deklaracije varijabli su naredbe, npr. `int i = 5;` , može se čitati kao "napravi varijablu s imenom 'i' i dodijeli joj vrijednost 5! ODMAH!".
Iako još nismo objasnili procedure, i one su također naredbe, npr:

```
jshell> System.out.println("Hello!")
        Hello!
```

Batch Copy

Vrlo duga naredba `System.out.println` je procedura za printanje u konzolu (ono gdje tipkamo naredbe, JShell ekran).
Do sada je JShell to automatski radio za nas, ispisivao vrijednost zadnjeg izraza, u svom nekom predefinisanom formatu... Ovako mi možemo ispisati šta god želimo. Primijetite da ispisani `String` nema navodnika!
Navodnici se samo koriste za pisanje stringova!!! Nisu uključeni u samu vrijednost stringa. Isto je i sa karakterima.

Da vidimo još par primjera:

```
jshell> System.out.println("x je " + 5);
        x je 5

jshell> String rezultat = "x je " + 5;
        rezultat ==> "x je 5"
jshell> System.out.println(rezultat);
        x je 5
```

Batch Copy

Dakle, možemo proslijediti string direktno, ili varijablu koja je string, svejedno. :)
Kraj naredbe se označava s `;` (tačkazarez, en. semicolon).

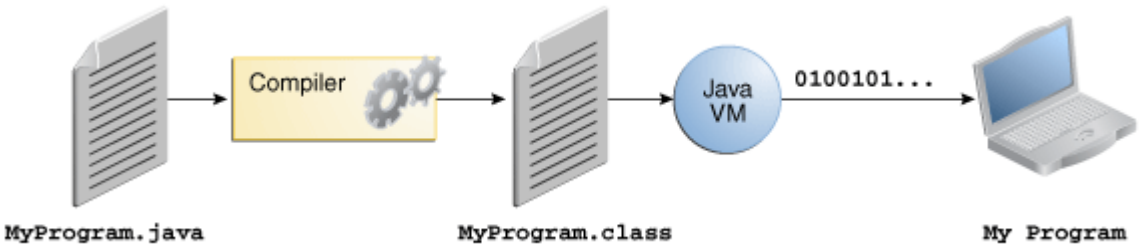
Poyy sviete!

Vjerovatno vam je do sada dodijao JShell i kucanje u konzoli... :D
Sada ćemo vidjeti kako se ustvari pokreću Java programi.
Sastoji se iz 2 dijela: kompajliranje i pokretanje (en. run).
Neki jezici su interpretirani, ne kompajliraju se nikako (Javascript, Ruby, Python).

Kompajler (en. compiler) je program koji provjerava sintaksu našeg programa, da li se poklapaju tipovi itd. te nam prijavljuje greške ako ih pronađe.
I to sve prije nego smo i pokrenuli naš program! Predobro! :D
Tako smo sigurni da se bar neke greške neće pojaviti u programu.
Javin kompajler se zove `javac` (skraćeno od Java compiler).

Ulaz u kompajler je izvorni tekst programa (en. source code) koji smo napisali (`.java` fajlovi, obični tekstualni fajlovi), a izlaz su fajlovi koji sadrže izvršni kod, koji se može pokrenuti nekako, izvršiti (`.class` fajlovi).

Idući korak je pokretanje programa.
Java programi se sastoje od `bytecode` naredbi (kod nezavisan od procesora i operativnog sistema), koje su vrlo slične mašinskom kodu (specifičnom za svaki procesor).
Izlaz iz kompajlera, `.class` fajlovi sadrže te `bytecode` naredbe, i njih izvršava tzv. JVM (Java Virtuelna Mašina).
JVM je ustvari `java` program.



Kompajliranje i pokretanje Java programa

Da vidimo (napokon) kako izgleda obavezni "Hello World" primjer:

Java Copy

```java
package primjer;

// komentar1
/*
 * komentar2
 */

class PoyySvijeteApp {
    public static void main(String[] argumenti) {
        String poruka = "Poyy sviete!";
        System.out.println(poruka);
    }
}
```

Kada se pokrene prethodno prikazani program dobićemo sljedeći rezultat u konzoli:

```
Poyy sviete!
```

Kompajliranje i pokretanje programa

Sadržaj programa sačuvajte u fajl "PoyySvijeteApp.java". Primijetite da se mora zvati isto kao i klasa, s nastavkom "java"!
Ovaj fajl se mora nalaziti u folderu "primjer" jer je to "package" od naše klase!
Kod mene se ovo sve nalazi na C: particiji, u folderu "programiranje":

```batch
PS C:\programiranje>
```

```batch
PS C:\programiranje> tree /f
C:.
####primjer
        PoyySvijeteApp.class
        PoyySvijeteApp.java
```

Dakle, imamo otvoren CMD i nalazimo se u folderu "programiranje".
Kucamo sljedeće:

```batch
PS C:\programiranje>
PS C:\programiranje>
```

```batch
javac .\primjer\PoyySvijeteApp.java
java primjer.PoyySvijeteApp
Poyy sviete!
```

To je to, sada možemo nastaviti dalje.
Svaki primjer koji budemo radili možete isprobati, nemojte mi slijepo vjerovati na riječ da fercera. :)

Komentari

Komentari su tekst koji ne utiče na rezultat programa.
Koriste se samo radi objašnjavanja koda, nekom drugom ili nama kad budemo čitali kasnije.
Prva vrsta komentara se piše sa 2 kose crte `//` (en. slash) i pišemo do kraja linije.
Druga vrsta je višelinijska (en. multiline) i počinje s `/*` a završava s `*/`.

Kontrola toka (programa)

Kompjuteri obično izvršavaju naredbe jednu nakon druge tj. sekvencijalno.
Međutim, potrebne su nam i naredbe za:

- grananje (`if` , `switch` , `match` i sl.) - kada želimo izvršiti nešto samo pod datim uslovima
- ponavljanje (`while` , `for` , rekurzija) - kada želimo ponavljati neki dio koda, da ne bi kopirali kod stalno...

Grananje

Grananje pomoću `if` naredbe je vrlo jednostavno i lahko razumljivo.
Ako je uslov ispunjen onda će se taj dio koda izvršiti.
U suprotnom program nastavlja dalje.

Ovo "uslov ispunjen" može biti DA ili NE, tj. ISTINA (en. true) ili NEISTINA (en. false).
Ovaj tip podatka se naziva Boolean po matematičaru George Boole-u.

Ako se sjećate logike iz matematike, tablica istine, "I", "ILI", "NE" i ostalih, TO JE TO! :D
Logička operacija "I" (en. and) se u Javi piše kao `&&` .
Logička operacija "ILI" (en. or) se u Javi piše kao `||` .
Logička operacija "NE" (en. negate, not) se u Javi piše kao `!` , i piše se prije uslova koji negira, npr. `!small` .

Primjer:

```java
double temperatura = 21.33;
boolean uslovToplo = temperatura > 25;
boolean uslovTaman = temperatura > 19 && temperatura <= 25;
if (uslovToplo) // mogli smo napisati i "if(temperatura > 25)"
    System.out.println("Toplo je");
else if (uslovTaman)
    System.out.println("Taman je");
else
    System.out.println("Hladno je");
```

Prethodni kod će ispisati "Taman je", jer je uslov `uslovTaman` ispunjen, tj. istina je da je `21.33 > 19 && 21.33 <=25` .

Niz uslova `if else-if ... else` se tretira kao jedna naredba!
Ovi uslovi bi trebali biti međusobno isključivi tj. smisleni.

> Ako napišete `if (broj>5) {/* prvi */} else if (broj>7) {/* drugi */}` to nema puno smisla jer ako broj nije veći od 5, ne može biti veći od 7, nema šansone. :D
> Ako jeste veći od 5, biće izvršen prvi blok, ali drugi blok koda NIKAD NEĆE BITI IZVRŠEN!

Ako postoji grana `else`, ona će biti izvršena ako nijedan uslov nije ispunjen.
Ako ne postoji `else` i nijedan uslov nije ispunjen, ništa neće biti izvršeno od te cijele if naredbe.

Grananje pomoću "switch-case" naredbe može učiniti kod dosta preglednijim.
Ova naredba se koristi umjesto mnoštva if-else grana, od koje svaka grana provjerava je li varijabla jednaka nekoj vrijednosti.
U Javi možete "switchati" cijele brojeve, karaktere, stringove i enumeracije.

Java Copy

```
1  int i = 5;
2  switch (i) {
3      case 3:
4          System.out.println("Tri");
5          break;
6      case 5:
7          System.out.println("Pet");
8          break;
9      default:
10         System.out.println("Ne znam...");
11         break;
12 }
```

Na liniji 2 ispituje se čemu je jednaka vrijednost varijable `i`, redom odozgo naravno.
Prvi slučaj (en. case) koji bude ispunjen će biti izvršen.
Ako je vrijednost varijable `i` broj 5 ispisaće se na ekran "Pet".

Često ne znamo koliko ima mogućih slučajeva, pa onda trebamo odlučiti šta da uradimo po tom pitanju.
Za to nam služi ključna riječ "default". To je ustvari ona `else` grana u `if` naredbi! ;)

Ponavljanje

U sljedećem primjeru želimo izvršavati kod sve dok je neki uslov ispunjen (en. while znači "dok").
Naravno, unutar tog bloka uslov se mora mijenjati, inače se program neće nikad zaustaviti.
To se naziva beskonačna petlja i obično nije poželjna... :)

Java Copy

```
1 int i = 0;
2 while(i < 10) {
3     System.out.println("Cifra i je: " + i);
4     i = i + 1;
5 }
```

Uslov se ispituje u svakoj iteraciji petlje. Dakle, svaki put kada se dođe do linije 5 uslov će se ponovo ispitati. Ako uslov nije zadovoljen, petlja se zaustavlja i program se nastavlja izvršavati na liniji 6.

Java ima i tzv. `for` petlje. To su haman-ha `while` petlje s finijom sintaksom.
Sljedeći primjer radi isto kao i prethodni sa `while` petljom.

Java Copy

```
1 for (int i = 0; i < 10; i = i + 1) {
2     System.out.println("Cifra i je: " + i);
3 }
```

Uopćena sintaksa for petlje je sljedeća: `for(inicijalizacije; uslov; koraci)`.
Vidimo da su dijelovi `for` petlje razdvojeni tačkazarezom.

Umjesto da deklarišemo varijable prije petlje, zgodno je deklarisati ih u sklopu petlje, jer nam poslije petlje one ne trebaju.
Za to nam služi dio inicijalizacije.

Dio uslov je isti kao i kod while petlje.

Dio koraci dolazi na kraju petlje, isto kao linija 4 u našoj `while` petlji.
Koraci obično utiču na uslov za izlaz iz petlje.

Funkcije

Prvo malo terminologije:

Potprogram (en. subprogram, subroutine) je dio programa koji možemo pozvati po imenu.
U nekim jezicima pravi se razlika između funkcije (potprogram koji ima rezultat), i procedure (nema rezultat).
Java ih zove... khm... metode... :D
Ja ću najčešće koristiti termin "funkcija" ili "metoda", kako kad.

Funkcijama se mogu proslijediti argumenti, tj. ulazni podaci koje funkcija može koristiti.

Slijedi primjer funkcije za sabiranje dva cijela broja:

```java
int suma(int a, int b) {
    return a + b;
}
```

Java Copy

Definicija funkcije ima sljedeće dijelove:

- tip vrijednosti koju vraća funkcija. Kod nas je to `int`, jer je rezultat sabiranja dva `int`a također `int`.
- naziv funkcije, kod nas je to "suma"
- lista parametara u zagradama, ovdje su to brojevi `a` i `b`: `int a, int b`.
  Ako nema parametara onda idu samo zagrade, npr. `mojaFunkcija()`
- tijelo funkcije u vitičastim zagradama
- ključna riječ `return` i rezultat funkcije

Da vidimo u JShellu kako da deklarišemo i pozovemo prethodnu funkciju:

```
jshell> int suma(int a, int b) {
   ...>     return a + b;
   ...> }
|  created method suma(int,int)
jshell> suma(1, 4)
$6 ==> 5
```

Batch Copy

Funkcija se poziva navođenjem njenog imena, i parametara unutar zagrada.
Npr. `suma(1, 4)`.
Pošto poziv funkcije vraća vrijednost `int`, možemo je spasiti u varijablu,
npr. `int rezultat = suma(1, 4)`.
Dakle, i poziv funkcije je izraz! Čim ga možemo dodijeliti varijabli. :)