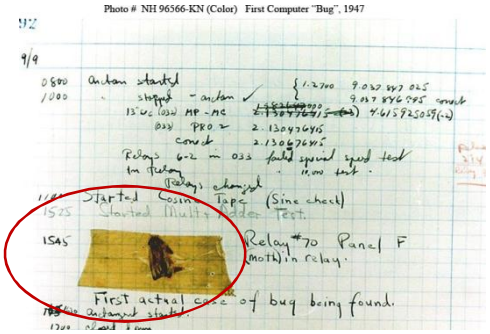
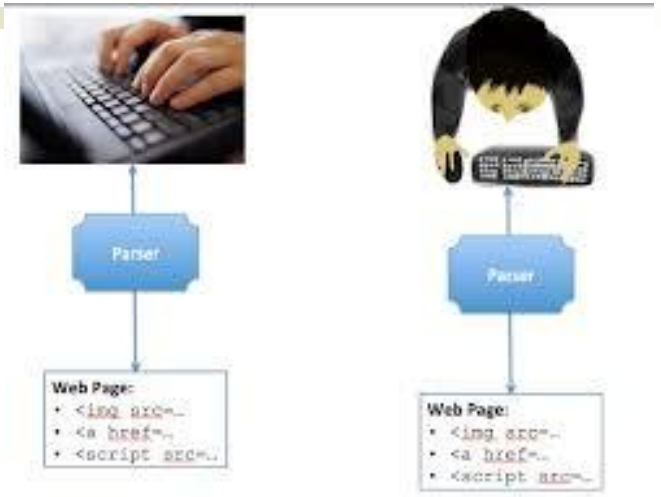
Programiranje





Nastava: prof.dr.sc. Dražena Gašpar

Vježbe: Jelena Zovko

Datum: 10.03.2015.

Osnovni plan nastave iz Programiranja

- Osnovni pojmovi
- Povijesni razvoj programskih jezika
- Programske paradigme
- Algoritamski pristup rješavanju problema
- Tipovi podataka
- Implementiranje imperativne paradigme
- Implementiranje objektne paradigme

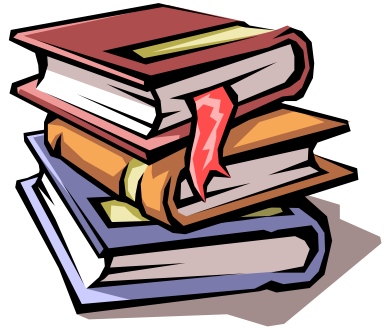
Osnovni plan nastave

- Programski jezik implementiranja programskih paradigmi

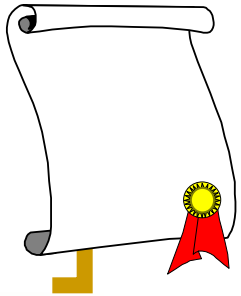
  JAVA

Bazna literatura

1. Dražena Tomić “Osnove programiranja”
2. Dario Sušanj “*Java Programiranje za Internet i World Wide Web*”, Znak Zg
3. Bruce Eckel “*Thinking in Java*”
4. **http://www.oracle.com/technetwork/java/index.html**
5. **http://www.java.com**



Način polaganja ispita

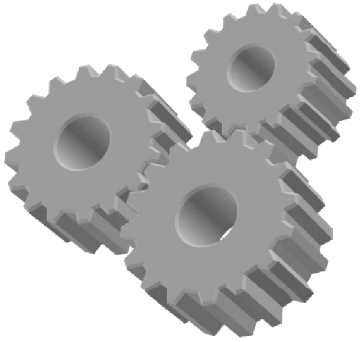


A. Kroz testove i projekt

   Ocjenjivanje Programiranje.doc

Definiranje programiranja

Što je to programiranje?



Što je sofware?

Je li programiranje=software ?

Definiranje programiranja

U širem smislu programiranje je procedura, postupak rješavanja nekog problema (problem solving procedure). Rezultat programiranja je program.

Program je niz naredbi koje se slijedno izvršavaju. Naredbama programa izvršava se neki zadatak. Podaci se transformiraju u informacije.

Programiranje i software

- **Program i softver nisu sinonimi.**
- **Pod softverom se podrazumijevaju programi, postupci, pripadajuća dokumentacija i podaci koji su povezani s radom nekog računalnog sustava.**
- **Softver je širi pojam od programa.**
- **Pisanje programa je oko 10-20 % ukupnog vremena razvoja softvera. Ako se to prevede na troškove softvera u jednom životnom ciklusu, onda je to oko 10% ukupnih troškova.**

Definicija programiranja

Programiranje je uži pojam od software-a.

Programiranje je procedura tj. postupak rješavanja nekog problema uz uporabu konkretnog programskog jezika.

Rezultat programiranja je program.

Definicija programiranja

Osnovni preduvjet za programiranje:

Da se kompleksni problem koji se želi isprogramirati može raščlaniti na konačan broj nedvosmislenih koraka koje stroj (računalo) može izvršiti.

Povijest Programiranja

- Korijeni u tekstilnoj industriji (1801)

Francuz Joseph Marie Charles Jacquard, po zanimanju tkalac

program za tkalački stroj, izrađen na drvenoj bušenoj kartici

2 bitne ideje programiranja:

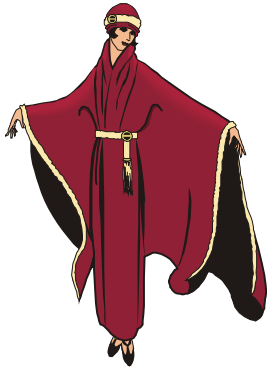
- raščlanjivanje kompleksnih zadaća na niz nedvosmislenih i konačnih koraka koje stroj može izvesti
- stroj na temelju programa može izvršavati ponavljajuće zadatke

Programski jezik

Govorni jezik: sustav glasova, gramatike, naglasaka, riječi i fraza kojima ljudi izražavaju svoje misli i osjećaje

Vezan pàs.

Vezan pâs.



- računala razumiju samo nizove brojeva

Programski jezik

Programski jezik -> formalni jezik u kojem je napisan računalni program

sastoji se od:

sintakse (načina na koji se različiti simboli jezika mogu kombinirati) i

semantike (značenja jezičnih konstrukcija)

Programski jezik

Principi dizajna jezika:

- Sintaksa

Sintaksa opisuje što čini strukturno ispravan program tj. što je gramatika jezika, koji je osnovni skup riječi i simbola dozvoljen za uporabu

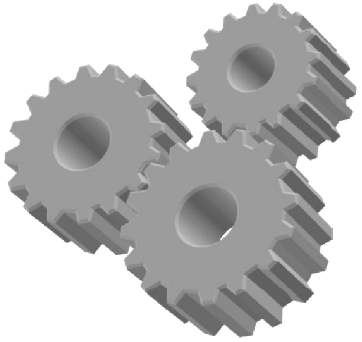
- Imenovanje i tipovi

Programski jezik sadrži potpuno izgrađen skup pravila za imenovanje varijabli, funkcija, klasa, parametara i sl., kao i njihovu “vidljivost” unutar programa. Tipovi podataka omogućavaju programerima bolje razumijevanje i uporabu operatora, kao i kvalitetniju kontrolu kompilatora.

- Semantika

Značenje programa, tj. programskih izraza definirano je semantikom programskog jezika.

Tko se smatra prvim programerom ?



Povijest programskih jezika

Prvi programer/ka :

1842. Ada Lovelace Byron

«Analitički Stroj tka algebarske uzorke na isti način kako Jacquard-ov tkalački stroj tka cvjetove i listove»

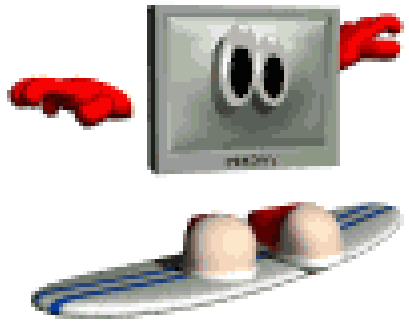
Ada je napisala skupove instrukcija koje bi se mogle izvršavati na Analitičkom Stroju.

Ada - prvi programer za računala Programski jezik Ada je u njenu čast dobio to ime.

Povijest programskih jezika

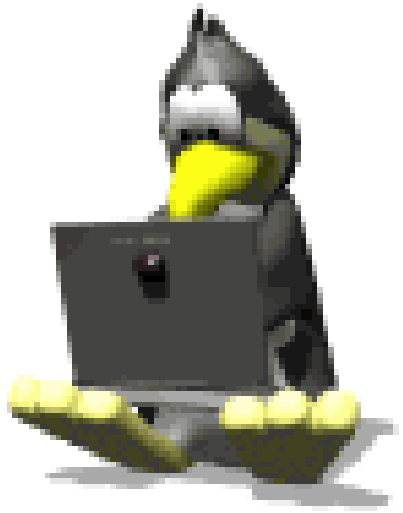
Pet generacija programskih jezika:

- strojni
- asembler
- proceduralni
- problemski orijentiran
- prirodni



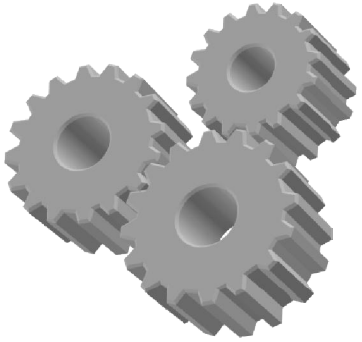
Postojeće stanje

- **C ++; C#**
- **JAVA**
- **VisualBasic**
- **Generatori**
- **Razvojna okruženja**

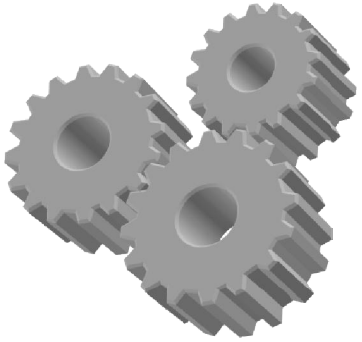


Tema: Implementiranje programskog jezika

Što se podrazumijeva pod implementiranjem programskog jezika?



Što je to
IZVORNI KOD?



Implementiranje programskog jezika

- Osnova za implementiranje je IZVORNI KOD

Izvorni kod je bilo koji niz izraza napisan u nekom, od strane čovjeka čitljivom programskom jeziku.

Izvorni kod je obično jedna ili više tekstualnih datoteka.

Transformiranje izvornog koda (čitljivog za čovjeka) u kod čitljiv od strane računala ostvaruje se ili pomoću kompilatora (prevoditelja) ili interpretera.

Implementiranje programskog jezika

BITNO:

promjene programa mogu se raditi ISKLJUČIVO na izvornom programu.

Da bi računalo moglo izvršiti zadaće napisane u izvornom programu, neophodno je taj program prevesti na jedini jezik koji računala razumiju, a to je jezik jedinica i nula (strojni programski jezik).

Implementiranje programskog jezika

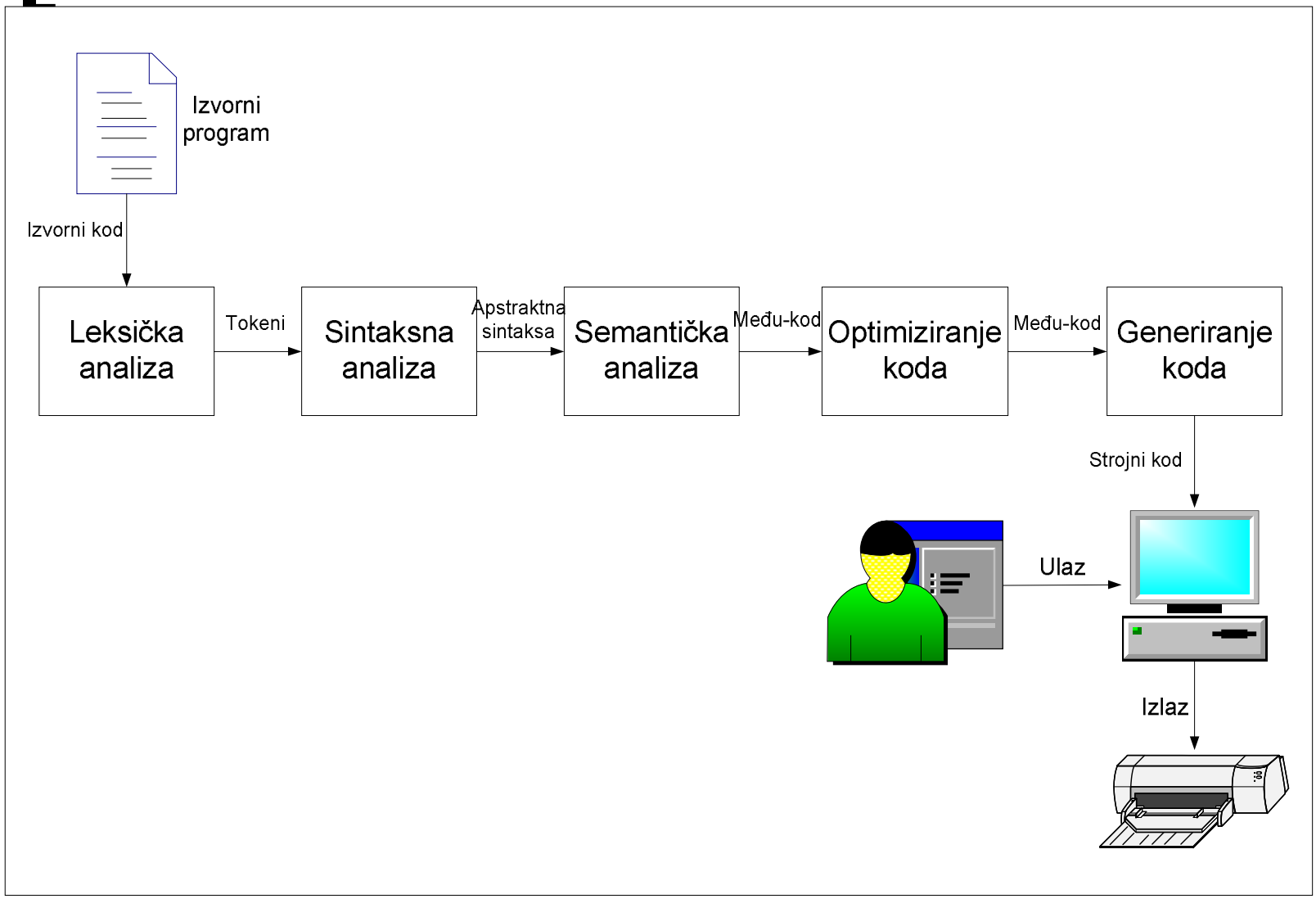
- Način izvršavanja konkretnog programa na jednoj ili više konfiguracija hardware-a i software-a
- 2 osnovna pristupa:
  - Kompiliranje tj. prevođenje
  - Interpretiranje

Implementiranje programskog jezika

Kompilator ili prevoditelj (engl. compiler) je računalni program (ili skup programa) koji prevodi tekst napisan u konkretnom programskom jeziku (izvorni kod) u drugi računalni jezik – objektni kod i/ili izvršni kod.

Osnovni cilj prevođenja (kompiliranja) izvornog koda je kreiranje tzv. izvršnog (engl. executable) programa tj. programa koji se može samostalno pozvati i izvršiti na odgovarajućoj računalnoj platformi.

Implementiranje programskog jezika



Interpreter

Interpreter je u biti program koji izvršava korake apstraktnog programa (međukoda) dok se izvršava na realnom stroju (računalu), odnosno to je program koji zamjenjuje dvije posljednje faze kompiliranja na način da izravno izvršava međukod.

Postoje dva opća tipa interpretera:

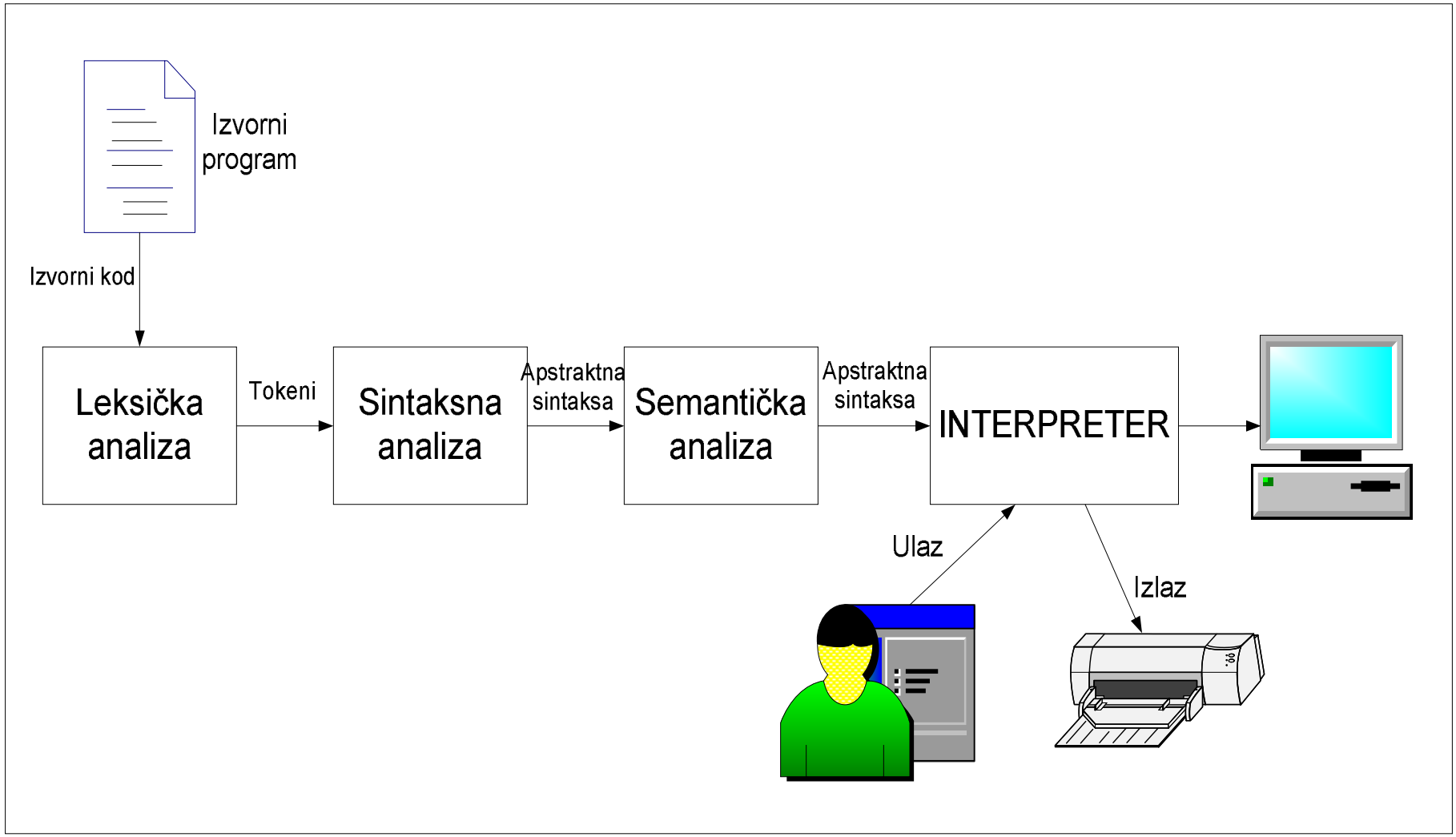
- Čisti
- Miješani.

Čisti interpreter svaki izraz tokenizira[1], raščlanjuje, semantički provjerava i interpretira pri svkom izvršavanju. Klasični primjer čistog interpretera je interpreter za Basic programski jezik.

Miješani interpreter prvo prevodi čitav program u među-kod, ali samo jedanput po izvršavanju, a onda dalje ponavlja interpretiranje međukoda bez daljnjeg prevođenja. Većina skript jezika (npr. Perl) koristi miješani interpreter.

[1] Pojam tokenizira preuzet je iz engleskog jezika a odnosi se na grupiranje programskih znakova u tokene tj. najmanje dijelove programa koji imaju neko značenje

Interpreter



Interpreter

Poseban slučaj miješanog interpretera je virtualni stroj (engl. *virtual machine*).

Kompilacija izvršava samo jedanput, s ciljem dobivanja koda za apstraktni virtualni stroj, a zatim se taj virtualni stroj implementira pomoću interpretera za svaki različiti realni tip stroja (računala).

Najpoznatiji predstavnik ovog pristupa je programski jezik Java čiji se apstraktni stroj naziva Java virtualni stroj - JVM. Osnovna namjera Java dizajnera pri kreiranju JVM bila je osigurati što bolju fleksibilnost i portabilnost Java programa, po cijenu nešto manje efikasnsti. Naime, JVM omogućava izvršavanje svih promjene u Java programu pomoću samo jednog kompilatora, umjesto nekoliko različitih kompilatora. Java kompilator prevodi izvorni kod u vanjski oblik međukoda poznat kao bajt kod koji se interpretira pomoću JVM. Novije verzije Java programskog jezika smanjuju gubitak efikasnosti ugradnjom JIT kompilatora u JVM. Ova nadogradnja omogućuje da se JVM bajt kod prevodi „trenutno“ (engl. *on-the-fly*) u izvorni strojni kod na host stroju prije samog izvršavanja.

Značajke JAVE

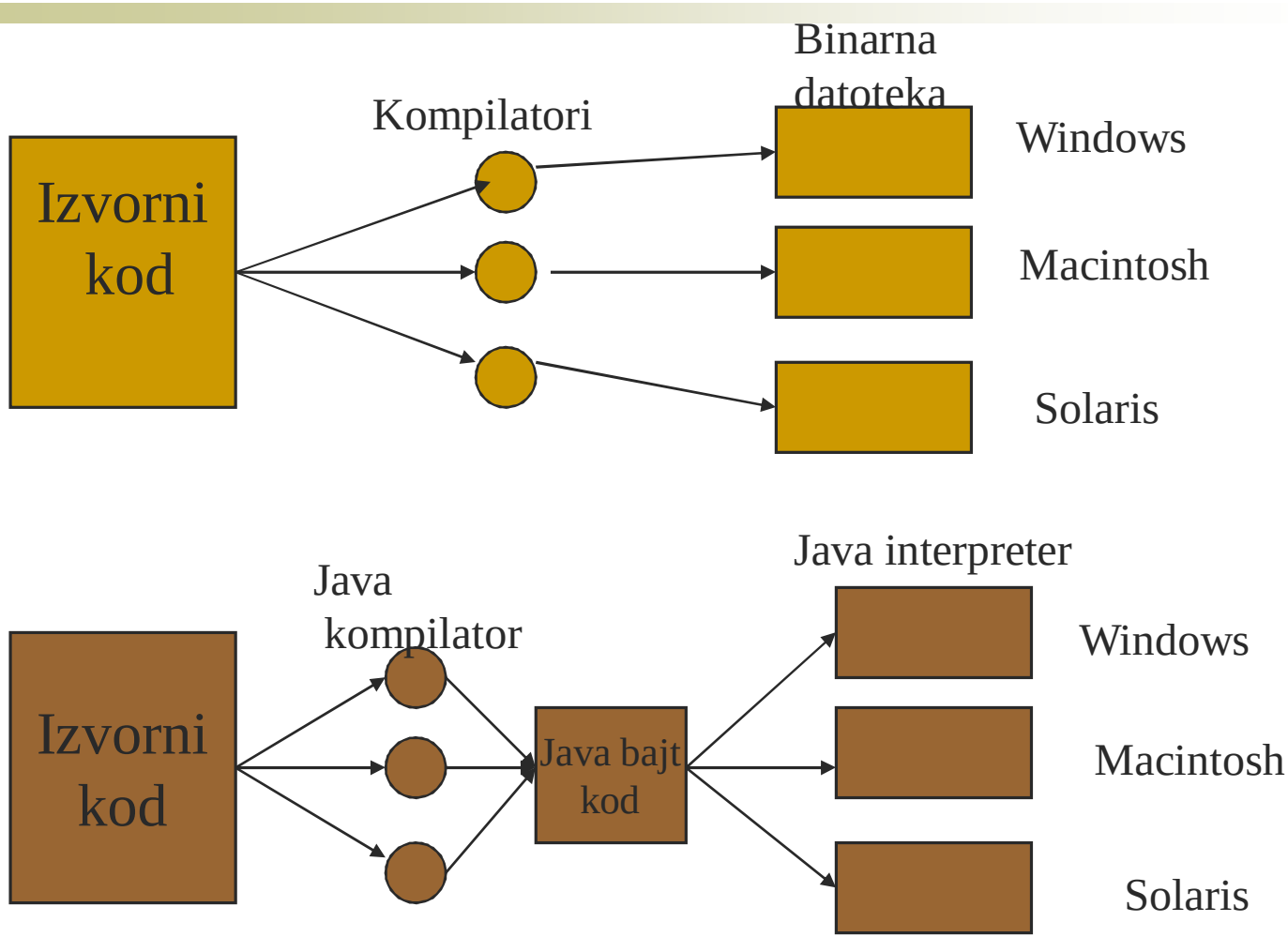
Interpretiranje

- **Prevođenje i izvršavanje Java aplikacije podijeljeno je u dvije faze:**

  1. Java kompilator iz izvornog koda programa stvara bajt-kod (tzv. J-kod)

  2. Java izvršni sustav koji interpretira bajt-kod i prevodi njegove naredbe u naredbe specifične za računalo na kojemu se izvršava aplikacija.

Interpretiranje … nastavak



Programske paradigme

- imperativno programiranje

- objektno-orijentirano programiranje

- funkcionalno programiranje

- logičko programiranje

Programske paradigme – Objektno orijentirano programiranje

Objektno orijentirano programiranje polazi od toga da se računalni program može promatrati kao skup pojedinačnih dijelova, ili objekata, koji surađuju (rade) jedan s drugim, dok se pak prema tradicionalnom shvaćanju program promatra kao skup funkcija, ili pojednostavljeno kao lista računalnih instrukcija.

Svaki objekt ima sposobnost primanja poruke, obrade podataka i slanja poruke drugim objektima, iz čega proizlazi da se svaki objekt može promatrati kao mali neovisni stroj ili glumac s različitom ulogom, odnosno odgovornošću.

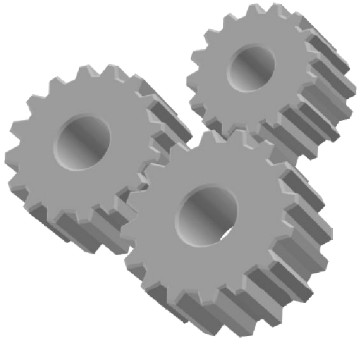
Programske paradigme – Objektno orijentirano programiranje

Osnovni koncepti:

- Objekt
- Klasa
- Nasljeđivanje
- Enkapsulacija (omatanje)
- Polimorfizam

Što je OBJEKT?

Koje su osnovne značajke OBJEKTA?



Programske paradigme – Objektno orijentirano programiranje

Objekt može biti bilo što sposobno za omogućavanje (pribavljanje) ograničenog skupa korisnih servisa (usluga). Svaki objekt ima stanje, ponašanje i identitet.

Svaki objekt ima pristup bilo kojem znanju potrebnom za izvršavanje njegovih usluga, što ne znači da objekt sadrži tu informaciju (znanje), već da joj može prići pitajući neki drugi objekt.

Uz objekt se vežu slijedeći pojmovi:

- odgovornost
- poruke
- protokol.

Programske paradigme – Objektno orijentirano programiranje

Odgovornost objekta - servis za koji se objekt “složio” ili mu je dodijeljen za izvršavanje. Objekti su zaduženi za izvršavanje specifičnih zadataka, a odgovornost se koristi kako bi se otkrilo tko (koji objekt) je zadužen za zadatak, bez potrebe da se razmišlja o strukturi objekta

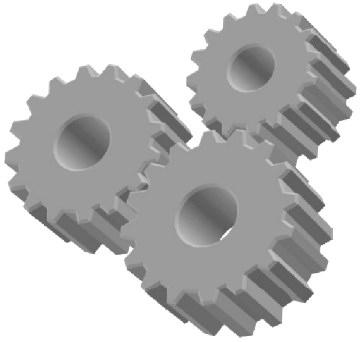
Poruka je formalna komunikacija poslana od strane jednog objekta drugom a koja zahtjeva neki servis (uslugu).

Pojam protokol se koristi za onaj dio sučelja koji prikazuje poruke na koje je objekt spreman odgovoriti.

Skup poruka na koje objekt odgovara i mijenjanje stanja objekta koje se bilježi – naziva se sučelje (engl. interface).

Što je KLASA?

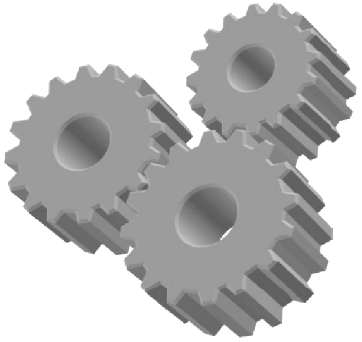
Razlika između KLASE
I objekta?



Klasa

je apstrakcija (generički opis) za skup objekata s istim atributima i operacijama, odnosno to je predložak (*engl. template*) za kreiranje objekata. Zbog toga se kaže da je objekt instanca (pojavni – konkretni oblik) klase

Što se podrazumijeva pod NASLJEĐIVANJEM?



Što se nasljeđuje?

Programske paradigme – Objektno orijentirano programiranje

Nasljeđivanje

mehanizam koji omogućuje da klasa (podklasa) redefinira (promijeni) ponašanje i osobine neke druge klase (nadklase). Podklasa ima sve što i nadklasa, ali i još nešto što je specifično samo za nju.

Nasljeđivanje se može definirati i kao nadređena-podređena relacija između klasa u kojoj podređena (dijete) klasa ima isto ponašanje (odgovornosti) kao i nadređena (roditelj) klasa i plus najmanje jedno dodatno

Programske paradigme – Objektno orijentirano programiranje

Enkapsulacija (omatanje)

osigurava da kod izvan klase (tj. Druga klasa) vidi samo funkcionalne, ali ne i implementacijske detalje klase.

“skriva” ponašanje objekta od njegove implementacije, odnosno odvaja kako objekt izgleda od toga kako implementira svoje ponašanje.

jamči da nitko osim objekta ne može znati kako “iznutra” taj objekt izgleda, tj. na koji način izvršava svoj zadatak. Rezultat enkapsulacije je da svaki objekt prema „vanjskom svijetu“, tj ostalim klasama pokazuje svoje sučelje tj. Skup poruka (metoda) na koje odgovara.

Programske paradigme – Objektno orijentirano programiranje

Polimorfizam

potječe od grčkih riječi

poly – mnogo i morph-oblik,

što znači mnogo oblika.

U objektno orijentiranom programiranju to znači da jedna poruka može uzrokovati različite oblike odgovora. Polimorfizam je ponašanje koje se mijenja ovisno od toga koja klasa ga uzrokuje, što znači da dvije ili više klasa mogu reagirati potpuno različito na istu poruku, odnosno da je **primatelj poruke odgovoran za njenu interpretaciju.**

Tema: Tipovi podataka

Pojam podatka

- Svojstva objekata i njihovih odnosa u prostoru i vremenu izražavamo pomoću podataka.
- Podatak je nematerijalne prirode
- Primarno postoji u našim mislima
- Pridružen je nekom konceptu, odnosno značenju kojim opisujemo svojstva
- Svojstva (objekt, odnosi i koncept) su promjenjivi u vremenu.

Pojam podatka

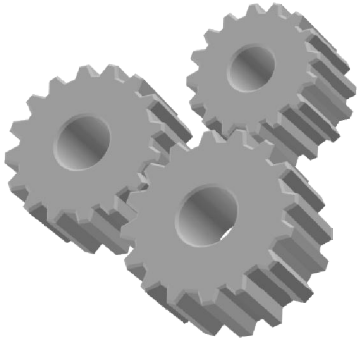
PODATAK

Apstraktna struktura sastavljena od:

- značenja (naziv i opis značenja određenog svojstva
- Vrijednosti (mjera i iznos)
- Vremena

Što je
VARIJABLA?

Čemu služi?



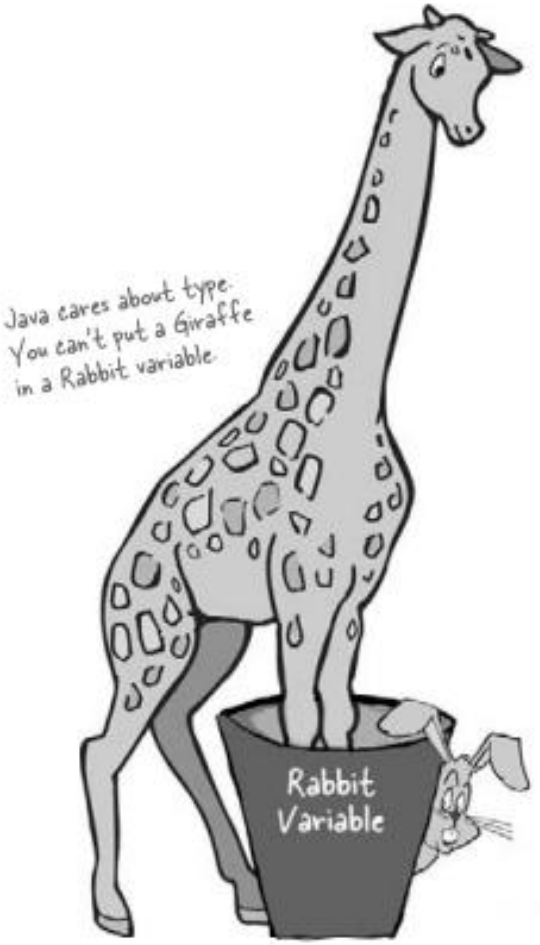
Domena: tip varijable

Pored imena i lokacije, svaka varijabla je definirana i tipom varijable. Tip varijable je imenovani skup (npr. cjelobrojni, znakovni i sl.) svih vrijednosti koje varijabla može poprimiti i za koji su dozvoljene odgovarajuće operacije.

Svaki kompilator mora znati:

- koji izrazi su valjani (validni)
- Kojeg je tipa rezultat svakog valjanog izraza
- Ukupan broj tipova je ZATVOREN SKUP

Što se podrazumijeva pod TIPOM PODATAKA?



Definiranje

Procesor rezerviranje memorije realizira preko tipova podataka i varijabli.

Tip podataka označava potrebnu količinu memorije za pohranu podatka, kao i vrstu podatka koja će biti pohranjena na toj memorijskoj lokaciji.

Broj byte-ova rezerviranih za tip podataka ovisi o programskom jeziku koji se koristi za pisanje programa i vrsti računala na kojoj je program kompiliran

Definicija: Tip podataka

Neka je zadan skup T sačinjen od n proizvoljnih apstraktnih objekata:

$$T:=\{v_1,\ldots,v_n\},\ n>1$$

Ukoliko su svi objekti istorodni u smislu da se u okviru nekog programskog jezika na njih može primjenjivati jedan određeni skup operatora, onda se T naziva tip podataka.

Definicija

Dodjeljivanje tipova podataka ima osnovnu namjenu dati određeno semantičko značenje skupu bitova koji inače nemaju nikakvo značenje. Tip podataka je obično povezan bilo sa vrijednostima u memoriji ili s objektima kao što su varijable.

Kako se bilo koja vrijednost jednostavno sastoji od skupa bitova u računalu, hardver ne pravi razliku čak ni između memorijskih adresa, koda instrukcija, znakova, cjelobrojnih i brojeva s pomičnim zarezom.

Tipovi podataka informiraju programe i programere kako bi trebali tretirati te puke bitove.

Tipovi podataka

Sustav tipova podataka je specifičan za svaki programski jezik i određuje načine na koji se programi pisani u tom programskom jeziku mogu ponašati, čineći istovremeno ponašanje izvan tih pravila ilegalnim i označavajući ga kao programsku pogrešku.

Tipovi podataka – osnovna podjela

1. Elementarni ili primitivni
2. Složeni ili korisnički definirani

Elementarni ili primitivni tipovi podataka su osnovni tipovi koji su već «ugrađeni» u, odnosno predstavljaju neodvojivi dio određenog programskog jezika, a koriste se u kao temelj za formiranje složenih ili korisnički definiranih tipova podataka.

Složeni ili korisnički definirani tipovi podataka su takvi tipovi koji mogu biti konstruirani u konkretnom programskom jeziku na bazi elementarnih tipova podataka tog jezika i drugih složenih tipova podataka.

Elementarni tipovi podataka

Danas se najviše koriste 32-bitna računala i memorijski zahtjevi u odnosu na elementarne tipove podataka uobičajeno se izražavaju u slijedećim memorijskim jedinicama:

- byte – 8 uzastopnih bitova memorije
- polu-riječ (engl. *half-word*) – 16 uzastopnih bitova = 2 byte-a
- riječ (engl. *word*) – 32 uzastopna bita = 4 byte-a
- dvostruka riječ (engl. *double word*) – 64 uzastopna bita = 8 byte-ova
- četverostruka riječ (engl. *quad word*) – 128 uzastopnih bitova = 16 byte-ova.

Elementarni tipovi podataka

Osnovna podjela:

a. Znakovni tip podataka
b. Numerički tipovi podataka
c. Logički tip podataka

Znakovni (character) tip podataka

Osnovni znakovni tip podataka jeste znak (engl. character). U računalnoj terminologiji znak je jedinica informacije koja odgovara slovu ili simbolu pisanog oblika prirodnog jezika.

Primjer za znak je slovo, broj ili oznaka interpukcije. Koncept znaka također uključuje i kontrolne znakove koji ne odgovaraju simbolima prirodnog jezika već drugim bitovima informacija koji se koriste za obradu teksta, kao što su instrukcije za pisače i druge uređaje koji prikazuju takav tekst.

Znakovni tip podataka

Tip CHARACTER

T:= { ‘0’,’1’,’2’,’3’,’4’,’5’,’6’,’7’,’8’,’9’,
‘A’,’B’,’C’,’D’,’E’,’F’,’G’,’H’,’I’,’J’,’K’,’L’,
‘M’,’N’,’O’,’P’,’Q’,’R’,’S’,’T’,’U’,’V’,’W’,
‘X’,’Y’,’Z’,’a’,’b’,’c’,’d’,’e’,’f’,’g’,’h’,’i’,’j’,
‘k’,’l’,’m’,’n’,’o’,’p’,’q’,’r’,’s’,’t’,’u’,’v’,’w’,
‘x’,’y’,’z’,’+’,’-’,’=‘,’/’,…kontrolni znaci}

Znakovni tip podataka

Različite metode za binarno kodiranje

ASCII (American Standard Code for Information Interchange)

Kontrolni znaci:

BEL (Bell) – zvučni signal (007 oktalno)

LF (Line Feed) – prijelaz u slijedeći red (012 oktalno)

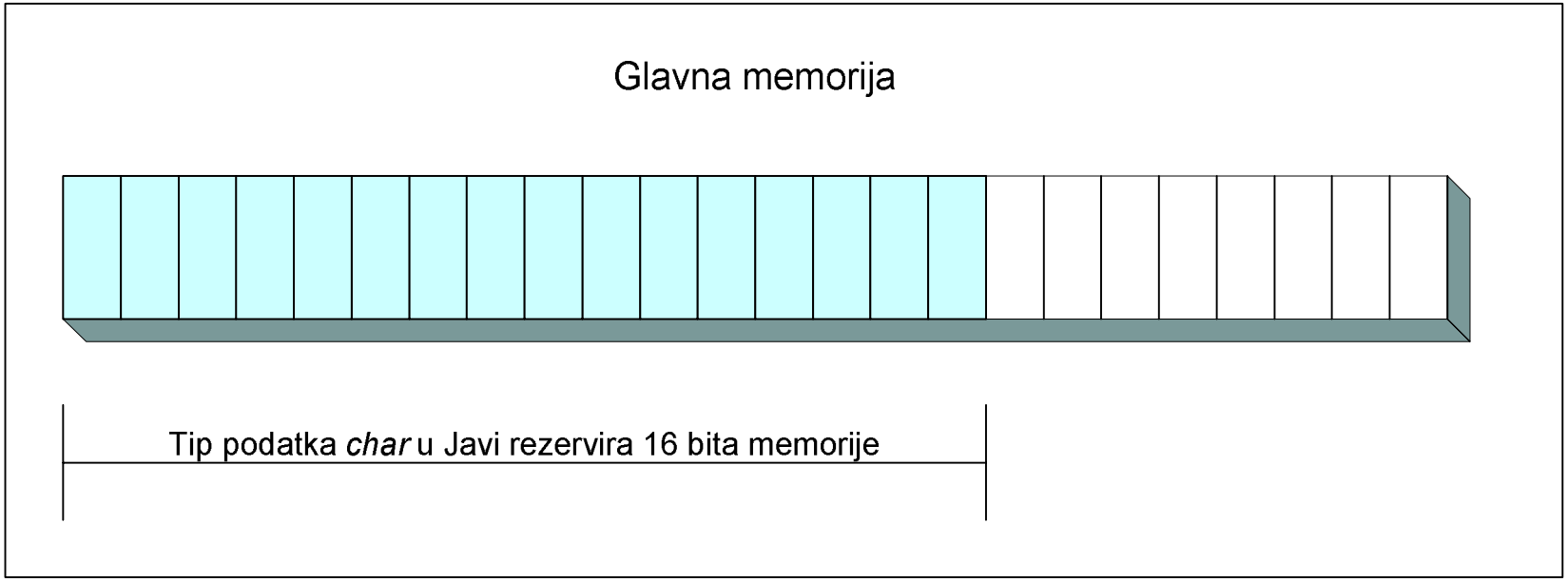
FF (Form Fed) – prijelaz na narednu stranu (014 okt.)

CR (Carriage Return) – povratak na početak reda (015)

ESC (Escape) – prelazak u naredbeni rad (033 okt.)

Znakovni tip podataka

Rezerviranje memorije:



Numerički tipovi podataka

Numerički tipovi podataka mogu se podijeliti na dvije opće skupine:

- cjelobrojni tipovi podataka
- tip podataka s pomičnim zarezom

Numerički tipovi podataka

Pojam cjelobrojni (engl. Integer) tip podataka odnosi se na bilo koji tip podatka koji može predstaviti neki podskup matematičkih cjelobrojnih vrijednosti.

Vrijednost podatka koji je cjelobrojnog tipa jeste matematička cjelobrojna vrijednost koja odgovara tom tipu. Predstavljanje ovakvog podatka je u biti način na koji se vrijednost pohranjuje u računalnoj memoriji. Cjelobrojni tip podatka može biti “neoznačeni” (engl. Unsigned) tj. Sposoban za predstavljanje samo pozitivnih cjelobrojnih vrijednosti, ili “označen” tj. sposoban za predstavljanje i negativnih i pozitivnih cjelobrojnih vrijednosti.

Numerički tipovi podataka

Uobičajeno predstavljanje pozitivnih cjelobrojnih vrijednosti je niz bitova, tj. uporaba binarnog numeričkog sustava.

Duljina ili preciznost cjelobrojnog tipa podataka je broj bitova koji se koristi u njegovom predstavljanju. Cjelobrojni tip podatka s n bitova može kodirati 2^n brojeva, tako na primjer neoznačeni cjelobrojni tip obično predstavlja pozitivne vrijednosti od 0 do 2^n-1.

Numerički tipovi podataka

Tip podatka s pomičnim zarezom (engl. *floating-point type*) je digitalno predstavljanje broja koji pripada nekom podskupu racionalnih brojeva i često se koristi za aproksimaciju proizvoljnih realnih brojeva u računalu.

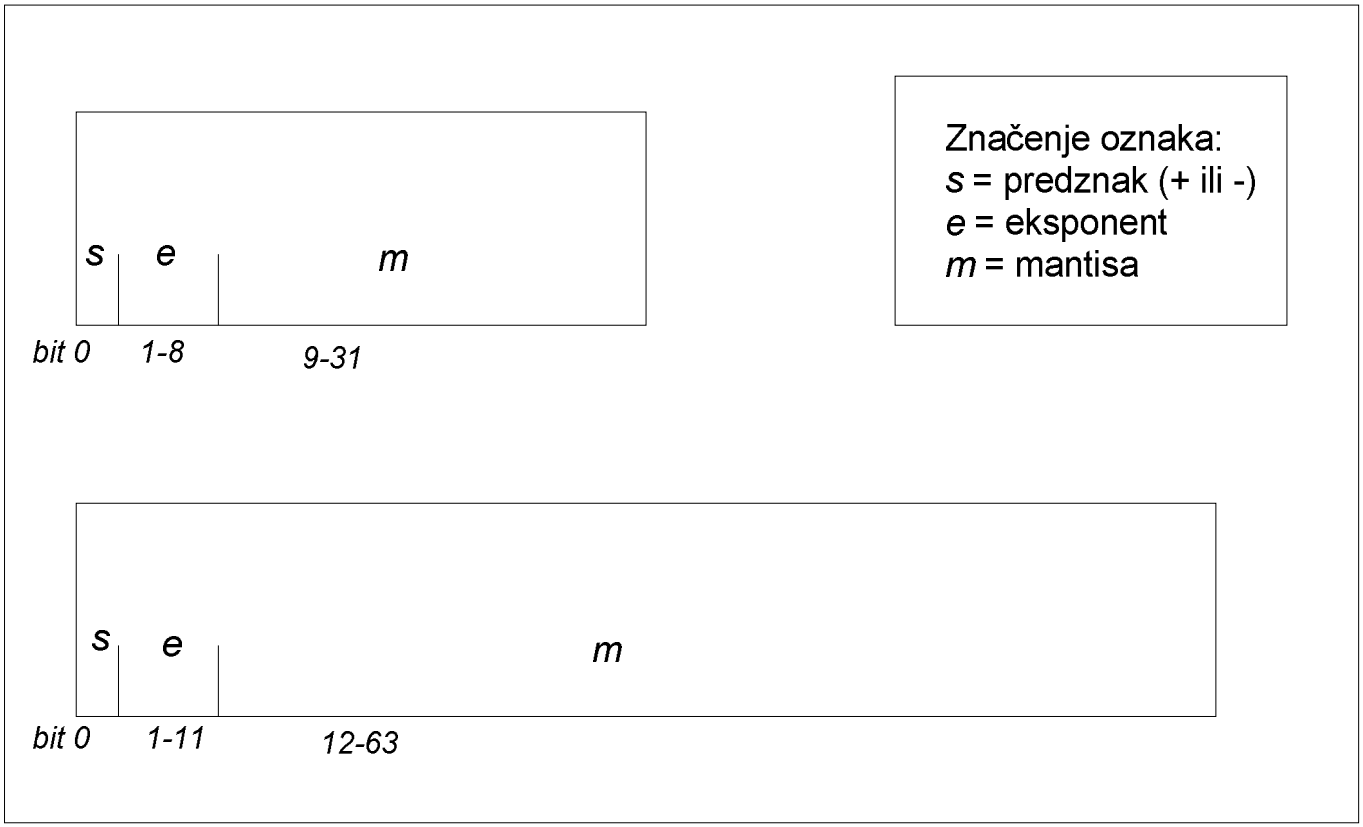
Predstavlja realni broj kao proizvod binarnog broja s jednim brojem koji nije nula lijevo od decimalne točke ili zareza, i odgovarajućeg eksponenta od 2.

Na primjer, realni broj 123.45 se predstavlja kao

$$1.2345 \times 10^2$$

Numerički tipovi podataka

Suvremena računala koriste standard IEEE 754 za predstavljanje vrijednosti s pomičnim zarezom.



Numerički tipovi podataka

Izračun s pomičnim zarezom je aritmetički izračun urađen s brojevima u pomičnom zarezu i često uključuje neku vrstu aproksimacije ili zaokruživanja pošto rezultat operacije možda nije moguće egzaktno predstaviti.

Bitno je primijetiti da nemogućnost predstavljanja postoji zbog činjenice da sustav pomičnog zareza definira diskretni brojni sustav čija je preciznost određena veličinom signifikanta i mantise. Čak i izračuni kod kojih su operandi potpuno predstavljivi često dovodi do rezultata koji nisu predstavljivi. U takvim slučajevima, sustav pomičnog zareza se referira na svoje postojeće diskretne brojevi i strateški odabire odgovarajuće predstavljanje kako bi osigurao najbolju moguću razinu točnosti.

Logički tip podataka

- **Tip podataka koji obuhvaća samo dvije vrijednosti T:={v_1,v_2}**

v_1 = false (ili v_1=F), v_2=true(ili v_2=T)

ili

v_1 =0, v_2=1

Logički tip podataka

Logički ili boolean tip podataka koriste operatori Boolean algebre kao što su:

- **konjukcija (AND)**
- **disjunkcija (OR)**
- **jednakost (=)**
- **negacija (NOT)**
- **i dr.**

JAVA TIPOVI PODATAKA

Java tipovi podataka

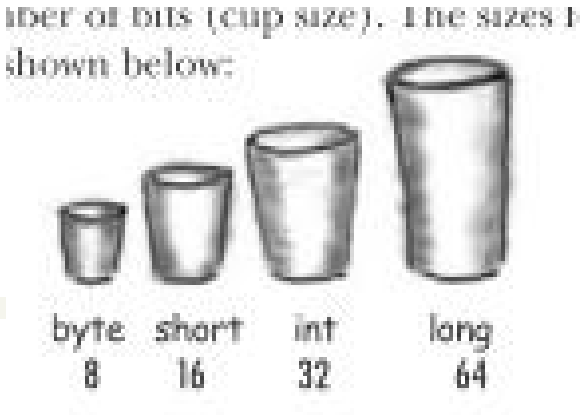
2 osnovne skupine:

- **jednostavni**
  - **cjelobrojni (byte, short, int, long)**
  - **za brojeve s pomičnim zarezom (float, double)**
  - **znakovni (char)**
  - **logički (boolean)**
- **složeni**
  - **polja**
  - **klase**
  - **Sučelja**

Osnovna osobina JAVA tipova: fiksna domena (područje)

Zbog osiguranja portabilnosti.

Java tipovi podataka



Cjelobrojni tipovi

Koriste se za prikaz predznačenih cijelih brojeva (i pozitivnih i negativnih)

Tip	Duljina	Najmanja vrijednost	Najveća vrijednost
long	**64**	**-9223372036854775808**	**9223372036854775807**
int	**32**	**-2147483648**	**2147483647**
short	**16**	**-32768**	**32767**
byte	**8**	**-128**	**127**

Java tipovi podataka

Tip byte

uporaba:

- za čitanje iz datoteka,

- za komuniciranje preko mreže ili

- **za rad sa sirovim binarnim podacima koji možda nisu izravno kompatibilni s drugim Java tipovima podataka.**

Deklariranje:

```
byte b, c;
```

Java tipovi podataka

Tip short

uporaba:

- vrlo rijetka, danas su računala 32-bitna

Deklariranje:

short s;

Java tipovi podataka

Tip int

uporaba:

- **najčešće korišten tip podataka (za brojače petlji, indekse nizova i sl.)**
- **važno: ukoliko se u izrazu miješaju tipovi byte, short i int, sve se vrijednosti prvo pretvaraju u int prije konačnog izračunavanja izraza**

Deklariranje:

```
int i, j, k;
```

Java tipovi podataka

Tip long

uporaba:

- **za prikaz dugačkih cijelih brojeva**

Deklariranje:

long l;

Java tipovi podataka

Brojevi s pomičnim zarezom (realni brojevi)

Koriste se za sva izračunavanja kod kojih treba povećati preciznost tako da se prikaže i decimalni dio.

Tip	Duljina	Najmanja vrijednost	Najveća vrijednost
double	**64**	**1.7E-308**	**1.7E+308**
float	**32**	**3.4E-038**	**3.4E+038**

Java tipovi podataka

Tip float

- **osigurava tzv, jednostruku preciznost**
- **pogodan za 32 bitna računala**
- **nedovoljno precizan kada je riječ o vrlo velikim ili vrlo malim vrijednostima**

Deklariranje:

float visokatemperatura;

Brojevi s pomičnim zarezom:

154.88

1.5488e2

Java tipovi podataka

Tip double

- **osigurava tzv, dvostruku preciznost**
- **pogodan za 64 bitna računala**
- **precizan kada je riječ o vrlo velikim ili vrlo malim vrijednostima**

Deklariranje:

double pi;

Java tipovi podataka

Tip char (znakovni tip)

- **baziran na Unicode standardu (16 bita – 2 byte)**
- **www.unicode.org**

Deklariranje:

char znak;

Java tipovi podataka

Tip boolean (logički tip)

- **true ili false**
- **ovaj tip vraćaju svi operatori usporedbe i to je obvezni tip podataka u uvjetnim izrazima (if, for)**

Deklariranje:

```
boolean b;
```

Izrazi i operatori

Izraz (engl. *expression*) je kombinacija literala, varijabli, operatora, poziva funkcijama koji kao rezultat daju određenu vrijednost koja se onda može koristiti u bilo kojem drugom kontekstu unutar programa.

Izlazni rezultat izraza je vrijednost.

Primjer:

a + b

a, b → operandi

\+ → operator

Izrazi i operatori

Operator - simbol koji označava operaciju koja će se izvršiti na jednoj ili više vrijednosti, a rezultat te operacije je neka druga vrijednost.

OSNOVNO grupiranje:

- Aritmetički operatori
- Operatori usporedbe
- Logički operatori

Aritmetički operatori

Operator	Namjena	Primjer
+	Zbrajanje	x=x+12
-	Oduzimanje	x=x-12
*	Množenje	x=x*12
/	Dijeljenje	x=x/12
%	Modulo (ostatak cjelobrojnog dijeljenja)	x=x%12
++	Inkrementiranje	x++
+=	Zbrajanje i dodjela vrijednosti	x += 12
-=	Oduzimanje i dodjela vrijednosti	x -= 12

Aritmetički operatori

Operator	Namjena	Primjer
*=	Množenje i dodjela vrijednosti	x *= 12
/=	Dijeljenje i dodjela vrijednosti	x /= 12
%=	Modulo i dodjela vrijednosti	x %= 12
--	Dekrementiranje	x--

Aritmetički operatori

i = i + 8; ili i += 8;

i = i % 2; ili i %= 2;

i = i * 7; ili i *= 7;

i = i – 4; ili i -= 4;

i = i / 5; ili i /= 5;

Aritmetički operatori

Operatori povećanja/smanjenja ++ / --

Ako je a = 40, slijedi

1.) a = a +1 → rezultat je 41 , a to se može napisati i pomoću operatora uvećavanja kao

a++ → rezultat je isto 41 ….. kao postfiks

++a → rezultat je isto 41 …. kao prefiks

2.) a = a -1 → rezultat je 39, a to se može pomoću operatora umanjenja napisati kao

a- - → rezultat je isto 39 … kao postfiks

--a → rezultat je isto 39 … kao prefiks

Aritmetički operatori

Operatori povećanja/smanjenja

++ / --

Dvojaki način pisanja: kao prefiks i kao sufiks

→ x=50; y=++x; => y=51,x=51

ili

x=x+1; y=x;

→x=50; y=x++; => y=50, x=51

ili

y=x; x=x+1;

Program za ++/-- operatore

```
class operatori1 {
    // Primjer za prefiks i sufiks

      public static void main (String args[]) {

      int x=50;
      int y=++x;

    System.out.println("Rezultat za prefiks -> y je "+y);
    System.out.println("Rezultat za prefiks -> x je "+x);

    x = 50;
    y = x++;

    System.out.println("Rezultat za sufiks -> y je "+y);
    System.out.println("Rezultat za sufiks -> x je "+x);
    }
}
```

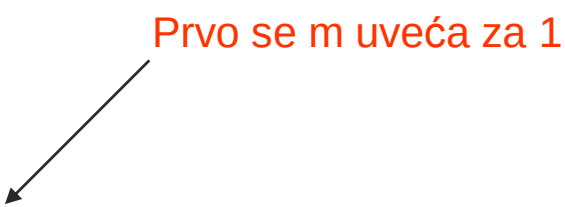
Aritmetički operatori

- Razlika u izrazima
- int m=7
- int a = 2 * ++m rezultat je ???
- int b = 2 * m++ rezultat je ???
- Napraviti program !!

DOMAĆA ZADAĆA

Aritmetički operatori

- Razlika u izrazima
- int m=7
- int a = 2 * ++m rezultat je 16
- int b = 2 * m++ rezultat je 14



Aritmetički operatori - dodjeljivanja

varijabla = varijabla operator izraz;

Ili

Varijabla operator=izraz;

Operatori usporedbe

Operator	Namjena	Primjer
==	Jednako	A==0
!=	Različito	A != 0
>	Veće	A > 0
<	Manje	A < 0
>=	Veće od ili jednako	A >=0
<=	Manje od ili jednako	A <= 0

Logički operatori

Operator	Namjena
&	Logička konjunkcija (AND – i)
|	Logička disjunkcija (OR – ili)
^	Logička isključiva disjunkcija (isključivo OR tj.XOR – isključivo ili)
||	Uvjetna disjunkcija
&&	Uvjetna konjunkcija
!	Logička unarna negacija (NOT – ne)
&=	Dodjeljivanje uz logičku konjunkciju
|=	Dodjeljivanje uz logičku disjunkciju

Logički operatori

Operator	Namjena
^=	Dodjeljivanje uz logičku isključivu disjunkciju
==	Jednako
!=	Različito
?:	Trojni operator uvjetne dodjele ili ternarni operator

Logički operatori

		OR	AND	XOR	
A	B	A|B	A&B	A^B	!A
False	False	False	False	False	True
True	False	True	False	True	False
False	True	True	False	True	True
True	True	True	True	False	False

Uvjetni logički operatori

- Uporaba && i || znači da Java neće izračunavati desnu vrijednost ako se rezultat može dobiti samo iz lijeve (operator OR daje kao rezultat true kada A ima vrijednost true, bez obzira na vrijednost B, odnosno operator AND daje kao rezultat false kada A ima vrijednost false bez obzira na vrijednost B.

Primjer:

```
if (djeljitelj !=0 && broj/djeljitelj >10)
```

Prioriteti izvršavanja operatora

```
()          []
++ --       !
*           /           %
+           -
>           >=          <           >=
== !=
&
^
|
&&
||
?:
=
```

Pridruživanje

- Opći oblik:

variabla = vrijednost

Na lijevoj strani - jednostavna varijabla ili element polja, odnosno bilo koja *l-vrijednost* (engl. *l-value*, gdje *l* dolazi od engl. *left tj. lijevo*). L-vrijednost mora označavati memorijsku lokaciju određenog tipa upisivu u vrijeme izvođenja (engl. *writable at run-time*). S desne strane operatora pridruživanja nalazi se bilo koji izraz odgovarajućeg tipa, ili tipa svodivog na tip l-vrijednosti. Taj izraz se, suprotno od l-vrijednosti naziva *r-vrijednost* (engl. *r-value*, *r* dolazi od engl. *right tj. desno*). Svaka l-vrijednost može imati ulogu r-vrijednosti, ali obrnuto ne vrijedi.

Pridruživanje

- Java operator pridruživanja je simbol =
- Vraća vrijednost sa svoje lijeve strane, tj. vrijednost koja se pridružuje u izrazu pridruživanja, npr.:

$$z = a + b/c ;$$

- Pridruživanje se obavlja s desna na lijevo.
- Najprije se izračuna r-vrijednost a + b/c , potom se ona pridruži l-vrijednosti z (varijabla).
- Primjer prikazuje model pridruživanja po vrijednosti koji se uglavnom koristi za elementarne tipove podataka (Java, C#), dok se za složene tipove podataka koriti model pridruživanja po referenci.

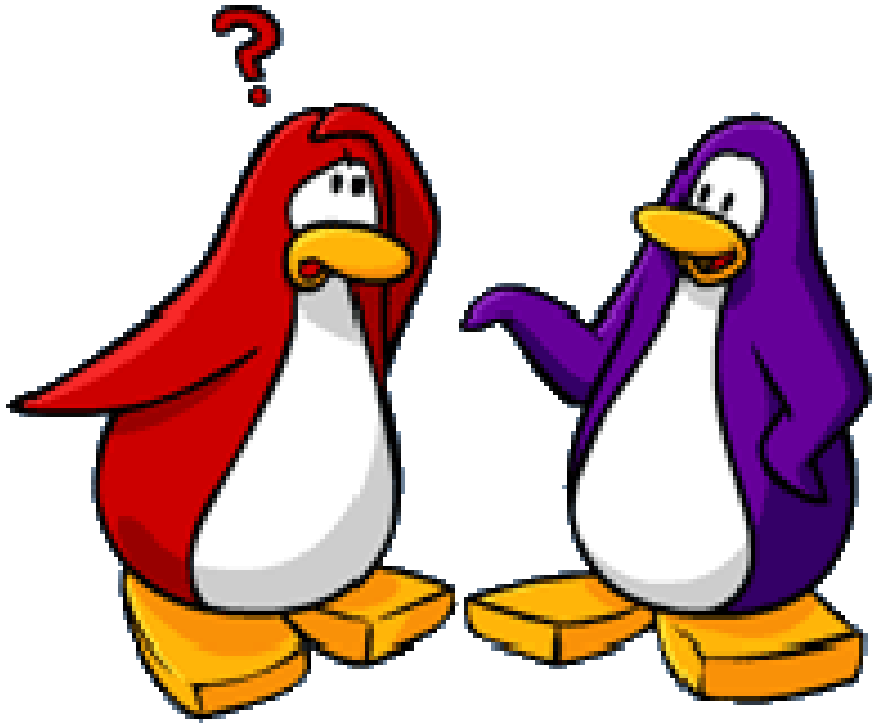
Pridruživanje

Za lokalne varijable metoda, Java (i C#) definira notaciju ***definitivnog pridruživanja*** (engl. *definite assignment*) koje onemogućava uporabu neinicijaliziranih varijabli.

Temelji se na kontrolnom toku programa i može biti statički provjeravana od strane kompilatora.

➔Svaki mogući put do izraza mora dodijeliti vrijednost svakoj varijabli i tom izrazu.

➔Konzervativno pravilo i može ponekad proglasiti pogrešnim neke programe, iako oni nikada ne bi stvarno koristili neinicijalizirane varijable



Pitanja..