Uvod u programski jezik *Java* i njegovo radno okruženje

Matej Mihelčić

Prirodoslovno-matematički fakultet, Sveučilište u Zagrebu

matmih@math.hr

5. listopada, 2022.

Povijest i razvoj

- Idejno razvijen 1991. godine od strane inženjera firme *Sun*. Grupa je predvođena **Patrickom Naughtonom** i **Jamesom Gauslingom**.
- Inicijalni cilj jezika: korištenje u korisničkim (elektroničkim) uređajima s **malo memorije** i potencijalno **različitim tipovima, karakteristikama i arhitekturama procesora**.
- Glavna ideja: razviti jezik koji se lako može prenositi **neovisno o arhitekturi**. Postiže se generiranjem među-koda za hipotetski stroj koji se zove *virtualni stroj*. Među-kod se može koristiti na svakom stroju koji ima odgovarajući **interpreter**.
- Prva uspješna primjena: za izgradnju web preglednika *HotJava*, demonstriranog 1995. godine.
- Prva javno dostupna verzija jezika *Java* objavljena 1996. godine.
- Od tada, *Java* bilježi veliki i brzi rast. $v1.0 - 1996.$: 211 klasa i sučelja, $v19.0 - 2022.$ 4000+ klasa i sučelja.

Glavne značajke jezika

Jednostavan

Dizajniran da bude **jednostavan za korištenje**, bez nepotrebnih jezičnih konstrukcija. Također, bitna karakteristika je da *interpreter*, podrška za klase, standardne biblioteke i podrška za dretve zauzimaju **jako malo memorije** (stoga je pogodan za korištenje u širokom spektru elektroničkih uređaja).

Objektno orijentiran

U tom pogledu znatno prati funkcionalnost jezika *C++*.

Distribuiran

Sadrži niz biblioteka za rad s *TCP/IP*, *HTTP* i *FTP* protokolima. Zbog jednostavnosti pristupa udaljenim resursima (npr. preko URL-a), pogodan za razvoj **mrežnih** aplikacija.

Glavne značajke jezika

Robustan

Vrši se **provjera tipova**, dostupni su **mehanizmi za detekciju i obradu iznimaka**, slijedi **strogi memorijski model** bez pokazivača koji onemogućava pristupe nedopuštenim memorijskim lokacijama.

Siguran

Zbog specifičnog memorijskog modela može **u potpunosti onemogućiti** programu (s niskom razinom povjerenja) pristup nedopuštenim memorijskim lokacijama, **separira klase** učitane s mrežnih lokacija od onih učitanih iz lokalne memorije, **provjerava dijelove koda** za nedopušten pristup resursima, **definira prava pristupa** resursima klasama u programu.

Glavne značajke jezika

Neovisan o platformi i arhitekturi

Jedna od najbitnijih značajki jezika. Zbog specifičnog dizajna, **stvaranja među-koda**, tzv. *oktetni kod* (eng. *bytecode*) koji razumije *virtualni stroj* može se lako izvršavati na **svakoj arhitekturi** i na **svakoj platformi** ukoliko je instaliran *Java runtime environment*. Taj softverski okvir sadrži skup pravila koja dopuštaju **lako i brzo prevođenje** *oktetnog koda* u strojne instrukcije od strane *interpretera*, tijekom izvođenja programa.

Prijenosan

Nema aspekata koji ovise o implementaciji. **Veličine tipova podataka i ponašanje odgovarajućih aritmetičkih operacija** su definirane *specifikacijom* jezika.

Glavne značajke jezika

Interpretiran

Java interpreter može prevoditi postojeći *Java oktetni kod* u izvorni kod, **tijekom izvršavanja programa**, na **svakom stroju** (bez obzira na arhitekturu procesora ili tip operacijskog sustava) ukoliko je **instaliran** odgovarajući *Java runtime environment*.

Visokih performansi

Brži od većine interpretiranih jezika zbog pretvorbe: izvorni kod $\rightarrow$ *oktetni kod* $\rightarrow$ strojni jezik. Taj postupak je još uvijek sporiji od izvršavanja programa koji su prevedeni (compilirani) direktno u strojni jezik (npr. *C, C++*). *Java* **smanjuje razliku** u izvođenju prema prevedenim programima tzv. *Just in time compiler*-om. **Često korišteni** dijelovi programa se **prevode u strojni jezik** i taj dio koda u strojnom jeziku se koristi kod svih daljnjih poziva odgovarajućeg dijela programa.

Glavne značajke jezika

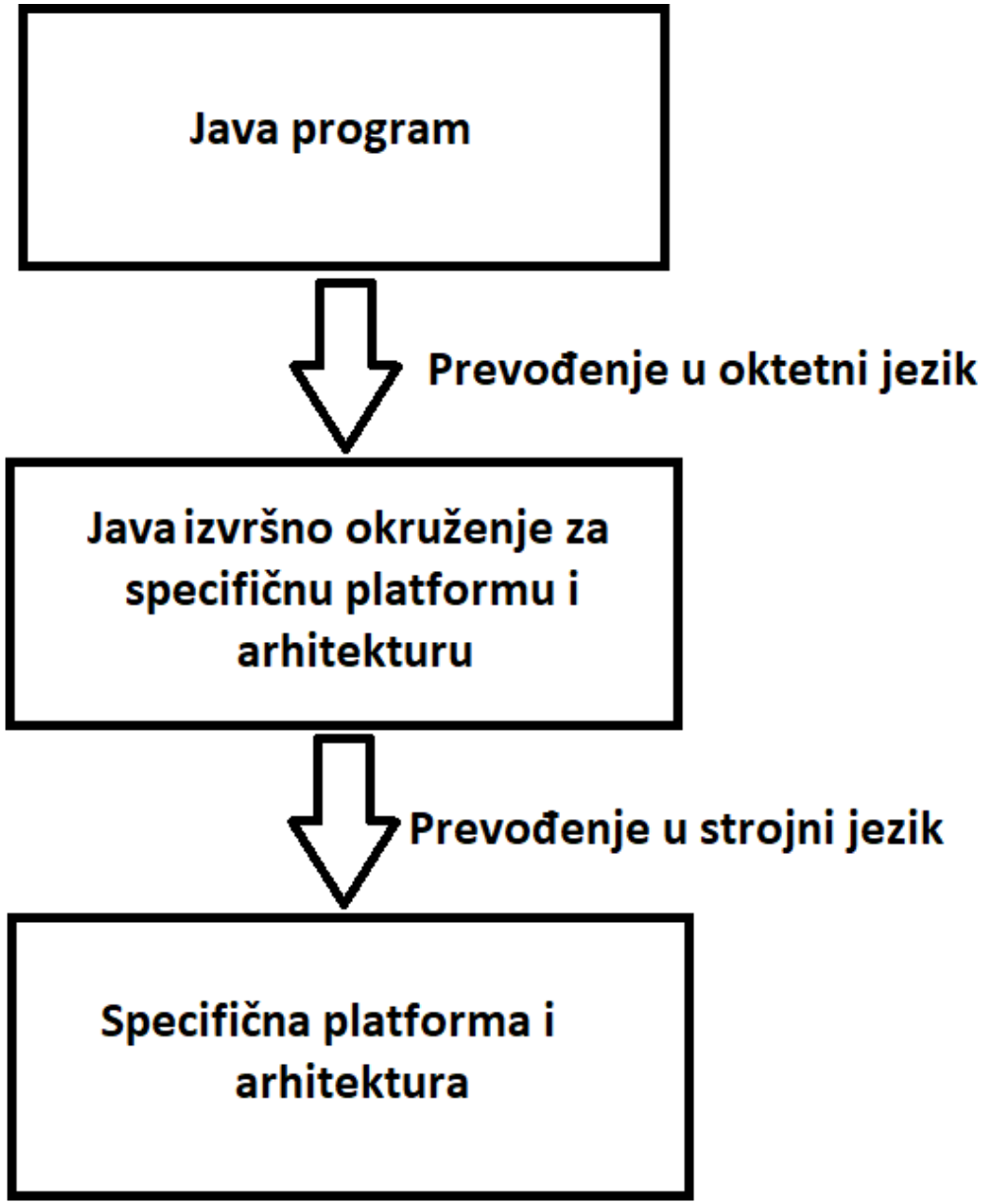
Više dretven

Znatno pojednostavlja programiranje višedretvenih programa.
Poboljšava interaktivnost i reakciju na događaje u stvarnom vremenu.

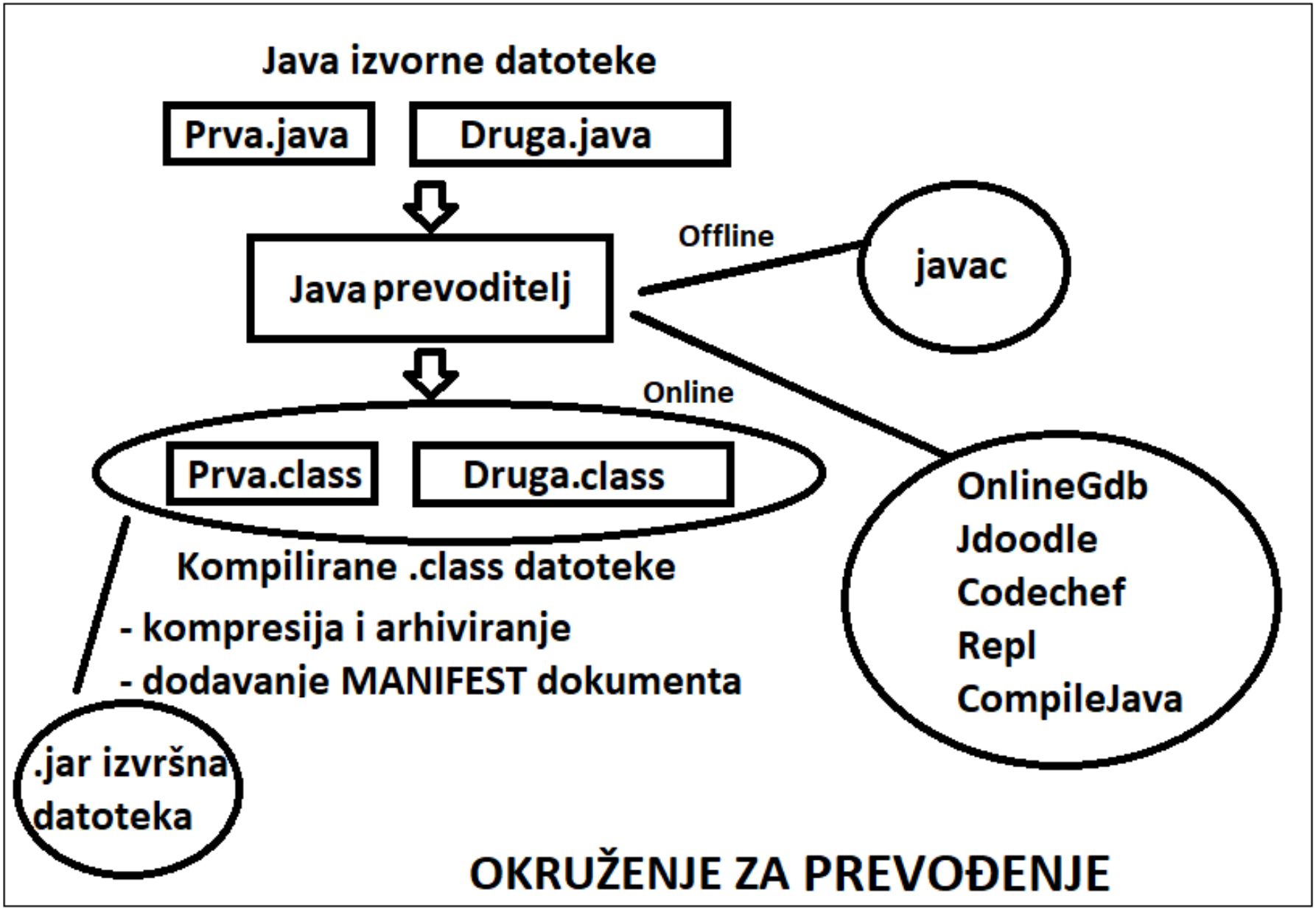
Dinamičan

Prilagođava se okruženju koje evoluira, biblioteke mogu dodavati nove metode ili instance variabli bez utjecaja na klijent dok se program izvršava (dinamičko učitavanje klasa). Ne učitava sve klase i ovisnosti već samo trenutno potrebne.

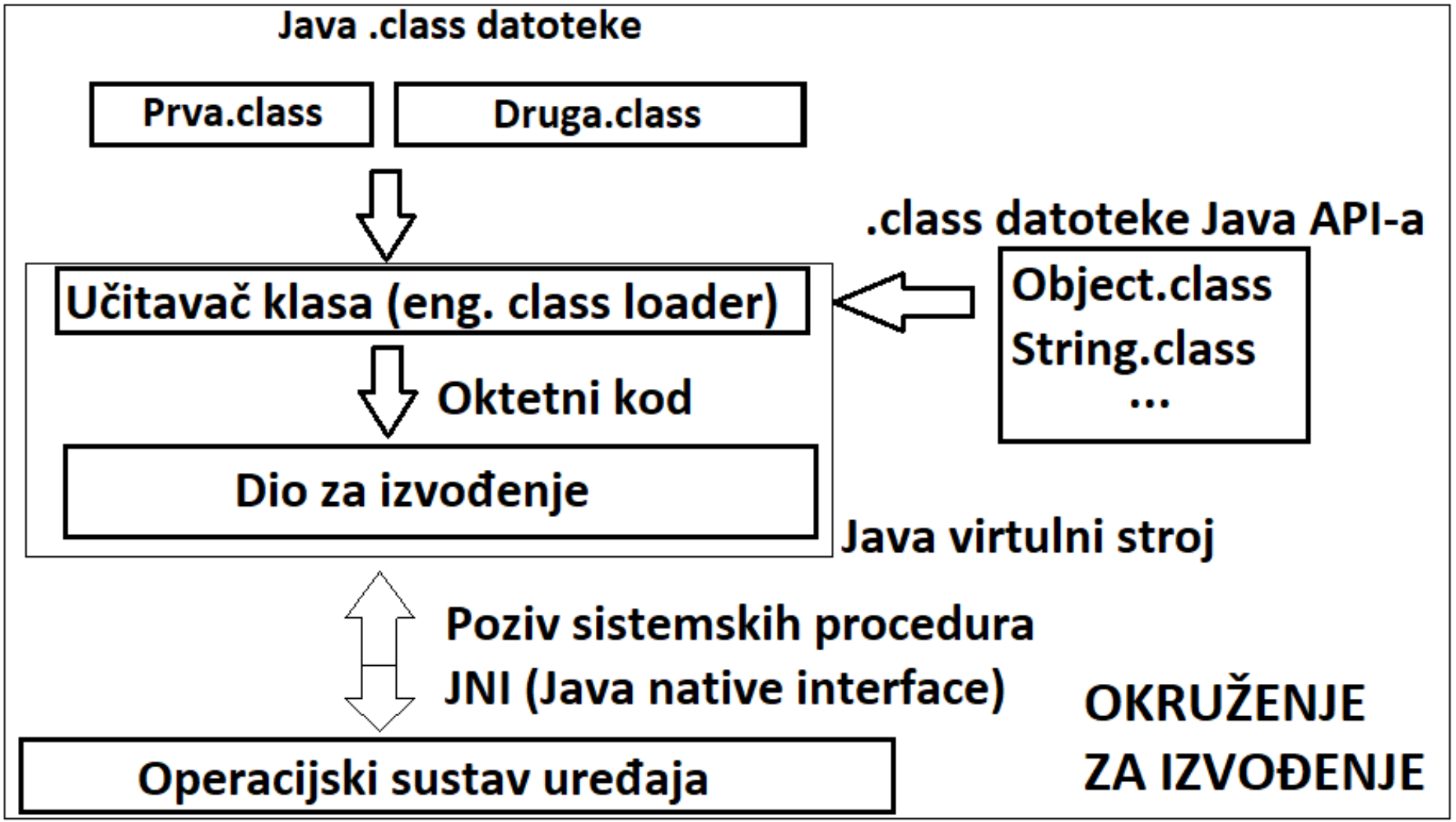
Java radno okruženje



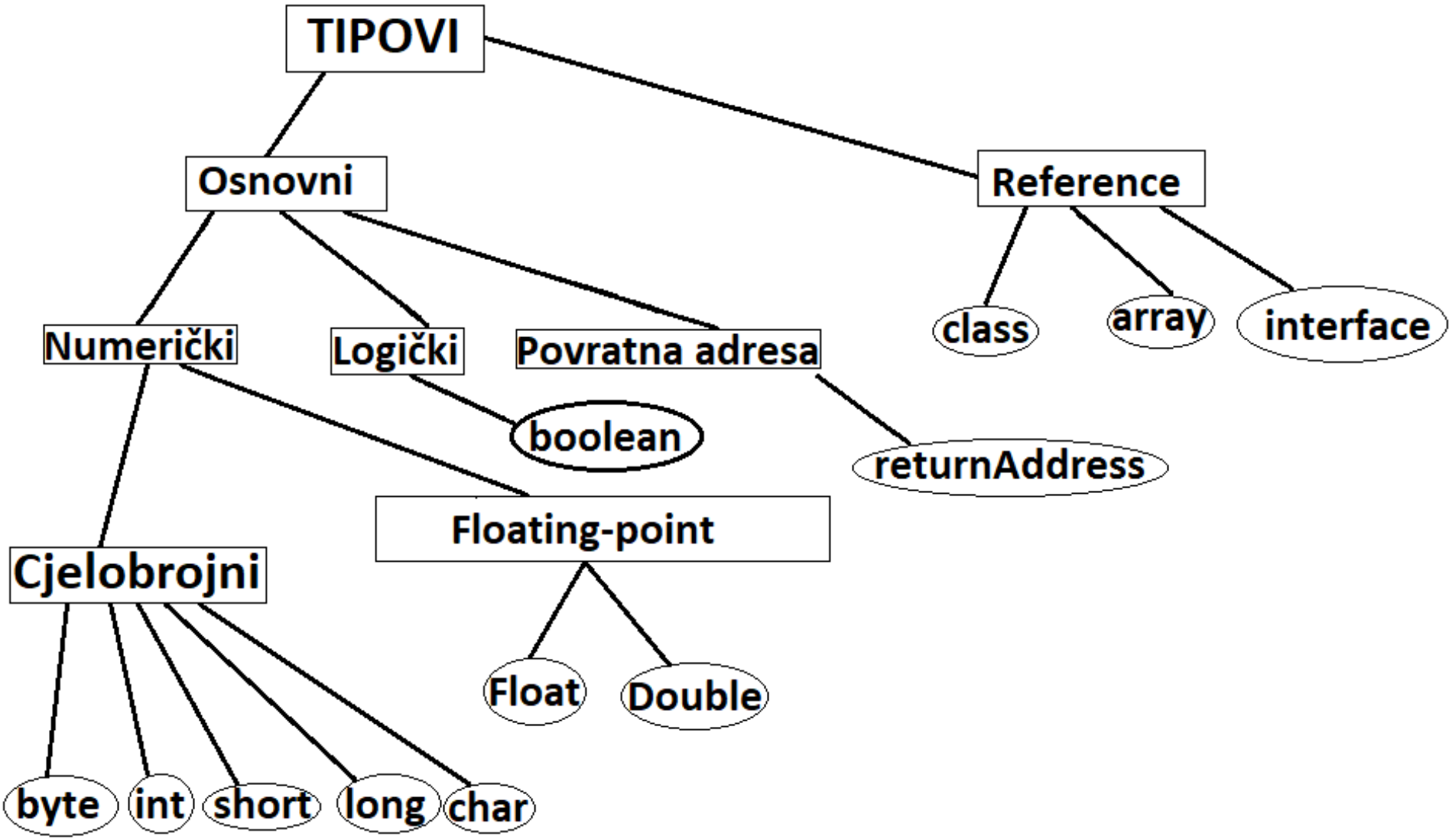
Java radno okruženje



Java radno okruženje



Java virtualni stroj - tipovi



Cjelobrojni tipovi:

Tip	Veličina (bita)	Raspon	Inicijalna vrijednost
byte	8	$[-2^7, 2^7 - 1]$	0
short	16	$[-2^{15}, 2^{15} - 1]$	0
int	32	$[-2^{31}, 2^{31} - 1]$	0
long	64	$[-2^{63}, 2^{63} - 1]$	$0L$
char	16	$[0, 2^{16} - 1]$	'\u0000'

Floating-point tipovi[1] **(reprezentacija realnih brojeva):**
Brojevi oblika $s \cdot m \cdot 2^{e-N+1}$, gdje $s \in \{-1, 1\}$, $m \in < 0, 2^N >$, $e \in [E_{min}, E_{max}]$, N i K parametri koji ovise o skupu vrijednosti tipa, $E_{min} = -(2^{K-1} - 2)$, $E_{max} = 2^{K-1} - 1$.

Tip	N	K	E_{min}	E_{max}	**Inicijalna vrijednost**
float	24	8	-126	127	+0.0f
float ext.[2]	24	≥ 11	≤ -1022	≥ 1023	+0.0f
double	53	11	-1022	1023	+0.0d
double ext.	53	≥ 15	≤ -16382	≥ 16383	+0.0d

Java virtualni stroj **mora standardno** podržavati tipove *float* i *double* a **opcionalno** i *float extended exponent* i *double extended exponent*.

[1] *float* i *double* prate standard 32-bitni *binary32* i 64-bitni *binary64* iz standarda IEEE 754 definiranog 2019. godine (Java SE 15). Extended tipovi nisu definirani tim standardom.

[2] float ext. i double ext. predstavljaju tipove *float extended exponent* i *double extended exponent*.

Java virtualni stroj - tipovi

Floating-point tipovi:

Pet posebnih vrijednosti: **pozitivna** i **negativna nula** ($+0.0$, -0.0), **pozitivna** i **negativna beskonačnost** ($+\infty, -\infty$), nije broj (eng. not a number) (`NaN`).

Postoji strogi mod računanja kod floating-point tipa (osigurava da se rezultati i međurezultati računaju jednako na svim platformama i verzijama virtualnog stroja). Potrebno dodati ključnu riječ `strictfp` (opcionalno od Java SE 17, **sve** operacije su strict).

Odstupanja od standarda `IEEE 754`:
a) floating-point operacije **ne izazivaju iznimke niti na bilo koji način signaliziraju iznimna stanja** propisana `IEEE 754` standardom, b) pravila zaokruživanja *Java virtualnog stroja* **ne slijede smjernice** `IEEE 754` standarda, c) instrukcije za ostatak pri dijeljenju floating-point brojeva su bazirane na dijeljenju koje koristi **zaokruživanje prema nuli**. `IEEE 754` standard propisuje zaokruživanje prema **najbližem cijelom broju**,

Floating-point tipovi:

Odstupanja od standarda `IEEE 754`:
d) *Java virtualni stroj* **ne podržava** tipove *binary32 extended* i *binary64 extended* definirane standardom `IEEE 754`. Skup vrijednosti tipova *double* i *double extended* podržavaju format *binary32* extended.

Tip povratne adrese:

- Koriste ga instrukcije *Java virtualnog stroja* `jsr` (jump subroutine), `ret` (return from subroutine) i `jsr_w` (jump subroutine wide index).
- Vrijednosti toga tipa su **pokazivači na strojne instrukcije** *Java virtualnog stroja*.
- **Ne pripada niti jednom tipu** programskog jezika *Java* i *Java* program **ne može mijenjati vrijednost** tog tipa.

Java virtualni stroj - tipovi

Logički tip:

- **Prevodi se** u tip *int* stoga ne postoje instrukcije virtualnog stroja za rad s logičkim tipom.
- 1 reprezentira logičku istinu a 0 logičku laž.

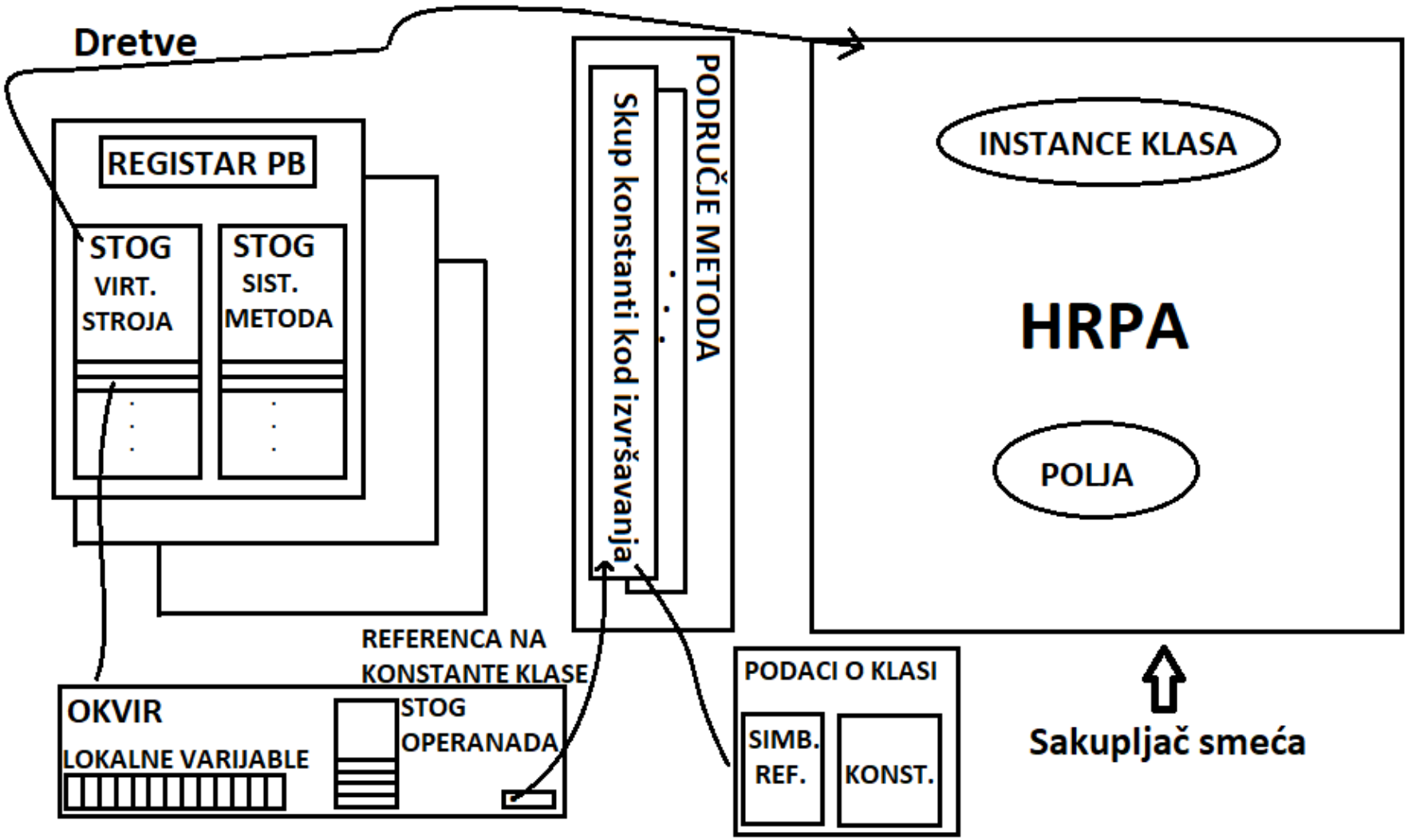
Tip reference:

Vrijednosti ovoga tipa su **reference na dinamički kreirane** instance klasa i polja ili instance klasa i polja koji **implementiraju sučelje**.

Tip polja se sastoji od **tipa komponente s jednom dimenzijom**. Tip komponente **može ponovo biti tip polja**, u tom slučaju opet gledamo tip komponente tog tipa dok ne dođemo do **osnovnog tipa, klase ili sučelja**.

Vrijednost reference može biti i posebna vrijednost `null`. `null` referenca **nema inicijalni tip kod izvođenja** ali se **može pretvoriti u proizvoljni tip korištenjem operatora pretvaranja tipova (eng. cast)**. **Standardna vrijednost** tipa reference je `null`.

Java virtualni stroj - podatkovna područja kod izvršavanja



Java virtualni stroj - podatkovna područja kod izvršavanja

Programsko brojilo

Sadrži **adresu naredbe** *Java virtualnog stroja* **koja se trenutno izvršava** (ukoliko naredba nije sistemska). Ukoliko je naredba sistemska, vrijednost programskog brojila je **nedefinirana**. Registar programskog brojila je dovoljno velik za spremanje tipa *povratne adrese* ili sistemski pokazivač na specifičnoj platformi. Svaka dretva *Java virtualnog stroja* ima **svoje programsko brojilo**. U svakom trenutku, svaka dretva izvodi kod **točno jedne metode** koja se zove *trenutna metoda*.

Stogovi (eng. stacks) *Java virtualnog stroja*

Ekvivalentan po namjeni stogu kod tradicionalnih programskih jezika (npr. *C, C++*). Sadrži **lokalne varijable**, **parcijalne rezultate**, koristi se kod **poziva funkcija** i **vraćanja vrijednosti iz funkcija**. Svaka dretva ima svoj privatni stog. U slučaju nedostatka memorije *Java virtualni stroj* vraća `StackOverflowError`.

Java virtualni stroj - podatkovna područja kod izvršavanja

Hrpa (eng. heap)

Dijele ga **sve dretve** *Java virtualnog stroja*. Sadrži **instance svih klasa** i **sva alocirana polja**. Kreira se pri **pokretanju** virtualnog stroja. Memorijske lokacije koje se ne koriste **preuzima automatizirani sustav za baratanje memorijom** koji se zove *sakupljač smeća* (eng. garbage collector). U slučaju nedostatka memorije *Java virtualni stroj* vraća `OutOfMemoryError`.

Područje metoda

Dijele ga **sve dretve** *Java virtualnog stroja*. Sadrži informacije o klasama kao što su **informacije o skupu konstanti**, **elementima klase** i **podacima metoda**. Sadrži i **kodove metoda**, **konstruktora**, **posebnih metoda** korištenih kod **inicijalizacija klasa i sučelja** te **inicijalizacija instanci**. Kreira se pri **pokretanju** virtualnog stroja. U slučaju nedostatka memorije *Java virtualni stroj* vraća `OutOfMemoryError`.

Java virtualni stroj - podatkovna područja kod izvršavanja

Skup konstanti kod izvršavanja

Sadrži razne vrste konstanti (od numeričkih do referenca metoda i polja). U slučaju nedostatka memorije *Java virtualni stroj* vraća `OutOfMemoryError`.

Stogovi sistemskih metoda

Koristi se kod poziva funkcija i vraćanja vrijednosti **sistemskih metoda**. Svaka dretva ima svoj stog sistemskih metoda. U slučaju nedostatka memorije *Java virtualni stroj* vraća `StackOverflowError`.

Okviri

Okvir se koristi za **spremanje podataka** i **djelomičnih rezultata**, za **dinamičko povezivanje**, **vraćanje rezultata metoda** i **odašiljanje iznimaka**.

Java virtualni stroj - podatkovna područja kod izvršavanja

Okviri

Okvir se stvara kod svakog **novog poziva** metode i uništava kod **prestanka izvođenja poziva** (neovisno o tome je li metoda završila normalno ili zbog iznimke).
Okviri se **alociraju** na stogu *Java virtualnog stroja* svake dretve.
Svaki okvir ima **polje lokalnih varijabli**, **stog operanada** i **referencu na skup konstanti kod izvršavanja** klase one metode koja se trenutno izvodi.
Veličina polja lokalnih varijabli i stoga operanada su **određeni za vrijeme prevođenja** te se **isporučuju okviru uz kod metode** koja je povezana s okvirom.
Samo **jedan okvir je aktivan** u bilo kojem trenuku na nekoj dretvi. Taj okvir se zove *trenutni okvir* a odgovarajuća metoda/klasa *trenutna metoda/klasa*. Ukoliko metoda povezana s okvirom pozove drugu metodu, povezani okvir prestaje biti trenutni.

Java virtualni stroj - podatkovna područja kod izvršavanja

Lokalne varijable

Koriste se za **prosljeđivanje parametara** metodama klasa ili metodama instanci klasa. Svaka lokalna varijabla u polju može spremiti vrijednost tipa `boolean`, `byte`, `char`, `short`, `int`, `float`, `reference` ili `returnAddress`. Vrijednosti tipa `double` i s oznakom duljine tipa `long` se spremaju u dvije uzastopne lokalne varijable.

Stog operanada

Inicijalno je **prazan**. Instrukcije *Java virtualnog stoga* **preuzimaju operande** sa stoga, **izvršavaju operacije** i **rezultat ponovo stavljaju na stog**. Stog operanada se koristi i za **spremanje parametara** koji se prosljeđuju metodama i za **spremanje povratnih vrijednosti** metoda. Stog operanada ima **predefiniranu dubinu** gdje `long` ili `double` doprinose dubini s **dvije** jedinice dok svi ostali tipovi doprinose s **jednom** jedinicom.

Dinamičko povezivanje

Okvir vrši dinamičko povezivanje **koda metode** preko **reference** na skup konstanti kod izvršavanja. Kod iz *class* dokumenta pokazuje na **metode** koje treba pozivati i **varijable** kojima treba pristupati kroz simboličke reference. Dinamičko povezivanje **prevodi simboličke reference u reference metode**, **učitava klase** po potrebi za **definiranje trenutno nedefiniranih simbola** i **prevodi pristupe varijablama u odgovarajuće pomake** u spremišnim strukturama asociranim s lokacijom tih varijabli tijekom izvršavanja. **Kasno povezivanje** metoda i varijabli čini **kod metode** otpornijim na promjene u drugim klasama.

Kraj poziva metode

Metode mogu završiti poziv **normalno** ili **neočekivano**.

- Metoda završava poziv normalno **ukoliko nije došlo do iznimke** niti od strane *Java virtualnog stroja* niti zbog eksplicitnog poziva iznimke. Ukoliko poziv normalno završi metoda **potencijalno vraća vrijednost metodi koja ju je pozvala**. **Trenutni okvir** se koristi za **obnovu stanja metode pozivatelja, lokalnih varijabli i stoga operanada uz odgovarajući inkrement programskog brojila**. Izvršavanje se nastavlja u **metodi pozivatelju** i **povratna vrijednost pozvane metode se potencijalno sprema na stogu operanada**.
- Metoda završava poziv neočekivano **kada izvršavanje prouzrokuje iznimku** *Java virtualnog stroja* koja **nije obrađena unutar metode**. Instrukcija `athrow` također **uzrokuje iznimku**. Metoda koja završi izvođenje neočekivano **nikada ne vraća vrijednost pozivatelju**.

Java virtualni stroj - skup instrukcija

```
do {
    automatski izracunaj PB i dohvati operacijski kod opcode
     na adresi PB;
    if (operandi) dohvati operandi;
    izvrsi akciju za kod opcode;
} while (nesto treba raditi);
```

Glavna petlja Java virtualnog stroja

Operandi se spremaju u *Big-endian* redoslijedu. **Operacijski kod** određuje broj i veličinu operanada.

Java virtualni stroj - primjer operacijskih kodova

Tip	Taload	Tastore	Tadd	Tsub	Tmul	Tdiv
byte	`baload`	`bastore`				
short	`saload`	`sastore`				
int	`iaload`	`iastore`	`iadd`	`isub`	`imul`	`idiv`
long	`laload`	`lastore`	`ladd`	`lsub`	`lmul`	`ldiv`
float	`faload`	`fastore`	`fadd`	`fsub`	`fmul`	`fdiv`
double	`daload`	`dastore`	`dadd`	`dsub`	`dmul`	`ddiv`
char	`caload`	`castore`				
reference	`aaload`	`aastore`				

Neki tipovi **ne podržavaju instrukciju** (npr. *reference* instrukciju `add`). Međutim, mnogi tipovi se **konvertiraju u tip** *int* stoga se u daljnjem računanju koriste instrukcije toga tipa.

Java virtualni stroj - stvarni tipovi i tipovi pri računanju

Pravi tip	Tip tijekom izvođenja
boolean	*int*
byte	*int*
char	*int*
short	*int*
int	*int*
float	*float*
reference	*reference*
returnAddress	*returnAddress*
long	*long*
double	*double*

Sakupljač smeća

Locira objekte na hrpi (eng. heap) koji više **nisu referencirani unutar programa**, **poziva destruktore na takvim objektima** i **čini memoriju zauzetu takvim objektima slobodnom za korištenje** od strane programa (memorija se reciklira). Sakupljač smeća također **može poslužiti za defragmentaciju hrpe**.

Prednosti sakupljača smeća:

- **Automatizira proces recikliranja memorije** čime povećava produktivnost programera.
- **Osigurava integritet programa**. Onemogućava programera da slučajno ili namjerno sruši *Java virtualni stroj* nepravilnim dealokacijama memorije.

Mane sakupljača smeća:

- **Troši procesorsko vrijeme** stoga može utjecati na performanse programa.
- Programeri imaju **manje kontrole glede korištenja procesorskog vremena** za oslobađanje nepotrebnih objekata.

Algoritmi za skupljanje smeća

Dvije bitne zadaće:

- Otkriti nereferencirane objekte
- Osloboditi memoriju za ponovno korištenje

Skupljanje smeća se vrši **definiranjem skupa korjenskih čvorova** te **traženjem povezanosti referenci s tim korjenima**. Svaki objekt se smatra **živim (potrebnim)** ukoliko postoji niz referenci pomoću kojih je moguće pristupiti objektu krenuvši iz jednog od korjena.

Vrste sakupljača smeća

- **Sakupljač koji broji reference**: kod **kreiranja objekta** postavi brojač reference tog objekta na 1. Kod **dodijeljivanja reference** tog objekta nekoj varijabli, brojač se inkrementira a kod dodijeljivanja varijabli koja sadrži referencu na objekt neke druge reference (**odjeljivanje**) ili kada **prestane životni vijek varijable** koja sadrži referencu na objekt, brojač se dekrementira za 1. Svi objekti kojima je brojač na 0 su podložni kolekciji od strane sakupljača smeća.
- **Sakupljač tragač**: **pretražuje graf referenci objekata** krečući od čvorova korjena. **Objekti koji se nalaze tijekom pretraživanja su označeni** ili postavljanjem zastavice u objektu ili postavljanjem zastavice u posebnoj bitmapi. Nakon završetka pretraživanja svi **objekti koji nisu označeni su podložni kolekciji** od strane sakupljača smeća. Osnovna verzija algoritma za traganje se zove *označi i počisti* (eng. mark and sweep).

Vrste sakupljača smeća

- **Zbijajući (kompaktirajući) sakupljači**: **skupljaju sve živuće objekte na jedan dio hrpe**. Zbog toga drugi dio hrpe postaje **neprekinuto slobodno područje**. Tim procesom se **smanjuje fragmentiranost hrpe**, međutim **potrebno je ažurirati reference** svih objekata koji su preseljeni na drugu memorijsku lokaciju.
- **Kopirajući sakupljači**: **pomiče sve živuće objekte u novo područje na hrpi jedan do drugoga** (razmaci između objekata se eliminiraju). Stara područja su **potpuno slobodna za korištenje**. Prednost ovoga pristupa je da se **objekti mogu kopirati na novu lokaciju čim su otkriveni tijekom pretrage**, na staroj lokaciji se ostavljaju usmjeravajući pokazivači koji daju do znanja sakupljaču da je objekt pomaknut. Ovim postupkom **nema potrebe za odvojenom fazom traženja i čišćenja** (eliminiranja).

Vrste sakupljača smeća

- **Generacijski sakupljači**: smanjuju količinu kopiranja koristeći dvije činjenice: a) većina objekata kreirana od većine programa ima kratak životni vijek, b) većina programa kreira nekolicinu objekata koji imaju dugi životni vijek. **Objekti se grupiraju po starosti, mlađi objekti se češće čiste** (time se izbjegava konstantno kopiranje objekata koji imaju dugi životni vijek). Objekt koji preživi nekoliko ciklusa čišćenja se prebacuje u stariju generaciju.
- **Adaptivni sakupljači**: **koriste informaciju o trenutnom stanju na hrpi da bi modificirali strategiju čišćenja**. Adaptivni sakupljač može tijekom izvršavanja **modificirati parametre algoritma čišćenja**, može **izmijeniti algoritam čišćenja**, **podijeliti hrpu u pod-hrpe** i **primjenjivati različite algoritme na svakom dijelu**.

Primjer pretvaranja izvorne datoteke u *class* datoteku

Pretpostavimo da imamo izvornu datoteku `HelloWorld.java` sadržaja:

```
1  public class HelloWorld {
2
3      public static void main(String [] args){
4          int i=1,j=2, k=0;
5
6          System.out.println("Hello world!");
7
8          k=i+j;
9          System.out.println(i+" + "+j+" = "+k);
10         k=i-j;
11         System.out.println(i+" - "+j+" = "+k);
12         k=i*j;
13         System.out.println(i+" * "+j+" = "+k);
14         k=i/j;
15         System.out.println(i+" / "+j+" = "+k);
16     }
17 }
```

Jednostavni Java program

Primjer pretvaranja izvorne datoteke u *class* datoteku

Datoteku prevodimo korištenjem naredbe komandne linije:
`javac HelloWorld.java`
Rezultat je *class* datoteka imena `HelloWorld.class`. Navedena datoteka je zapisana u **binarnom formatu** i ne možemo ju čitati korištenjem tekstualnih editora.

Da bi dobili informacije o **paketu**, **zaštičenim i javnim poljima** te **metodama klase** koristimo naredbu:
`javap HelloWorld.class`.

Za detaljan ispis *class* dokumenta koristimo naredbu:
`javap -c HelloWorld.class`.

Primjer pretvaranja izvorne datoteke u *class* datoteku

```
public class proba2.HelloWorld {
  public proba2.HelloWorld();
    Code:
       0: aload_0
       1: invokespecial #1                  // Method java/
    lang/Object."<init>":()V
       4: return

  public static void main(java.lang.String[]);
    Code:
       0: getstatic     #7                  // Field java/
    lang/System.out:Ljava/io/PrintStream;
       3: ldc           #13                 // String Hello
    world!
       5: invokevirtual #15                 // Method java/
    io/PrintStream.println:(Ljava/lang/String;)V
       8: iconst_1
       9: istore_1
```

Prevedeni Java program

Primjer pretvaranja izvorne datoteke u *class* datoteku

```
    10: iconst_2
    11: istore_2
    12: iconst_0
    13: istore_3
    14: iload_1
    15: iload_2
    16: iadd
    17: istore_3
    18: getstatic     #7                  // Field java/
lang/System.out:Ljava/io/PrintStream;
    21: iload_1
    22: iload_2
    23: iload_3
    24: invokedynamic #21,  0             // InvokeDynamic
 #0:makeConcatWithConstants:(III)Ljava/lang/String;
    29: invokevirtual #15                 // Method java/
io/PrintStream.println:(Ljava/lang/String;)V
    32: iload_1
```

Prevedeni Java program

Primjer pretvaranja izvorne datoteke u *class* datoteku

```
   33: iload_2
   34: isub
   35: istore_3
   36: getstatic     #7                  // Field java/
lang/System.out:Ljava/io/PrintStream;
   39: iload_1
   40: iload_2
   41: iload_3
   42: invokedynamic #25,  0             // InvokeDynamic
 #1:makeConcatWithConstants:(III)Ljava/lang/String;
   47: invokevirtual #15                 // Method java/
io/PrintStream.println:(Ljava/lang/String;)V
   50: iload_1
   51: iload_2
   52: imul
   53: istore_3
```

Prevedeni Java program

Primjer pretvaranja izvorne datoteke u *class* datoteku

```
   54: getstatic     #7                  // Field java/
lang/System.out:Ljava/io/PrintStream;
   57: iload_1
   58: iload_2
   59: iload_3
   60: invokedynamic #26,  0             // InvokeDynamic
 #2:makeConcatWithConstants:(III)Ljava/lang/String;
   65: invokevirtual #15                 // Method java/
io/PrintStream.println:(Ljava/lang/String;)V
   68: iload_1
   69: iload_2
   70: idiv
   71: istore_3
   72: getstatic     #7                  // Field java/
lang/System.out:Ljava/io/PrintStream;
   75: iload_1
   76: iload_2
```

Prevedeni Java program

Primjer pretvaranja izvorne datoteke u *class* datoteku

```
      77: iload_3
      78: invokedynamic #27,  0              // InvokeDynamic
    #3:makeConcatWithConstants:(III)Ljava/lang/String;
      83: invokevirtual #15                  // Method java/
   io/PrintStream.println:(Ljava/lang/String;)V
      86: return
}
```

Prevedeni Java program

Leksička struktura *Java* programa

Za opisivanje mogućih riječi (eng. token) koje se smiju koristiti u programskom jeziku koristimo **Leksičku gramatiku**. Način na koji se te riječi smiju kombinirati, njihov redoslijed je definiran **sintaksnom gramatikom**.
Obije gramatike spadaju u grupu **Kontekstno - slobodnih gramatika** (za detalje pogledati materijale kolegija *Interpretacija programa*).

Java programi mogu sadržavati znakove iz `Unicode` skupa znakova[3]. Pošto `Unicode` sadrži puno znakova koji se ne mogu direktno napisati s tipkovnice, u praksi se na ulazu koriste `ASCII` znakovi[4]. `Unicode` znakovi koji se ne mogu napisati s tipkovnice se kodiraju `ASCII` znakovima na način `\uxxxx`. `\u` označava početak koda a xxxx jedan heksadekadski broj. Tekst Java programa je reprezentiran (u binarnom formatu) nizovima 16-bitnih kodova koristeći `UTF-16` kodiranje.

[3] `https://www.unicode.org/`
[4] `http://www.asciitable.com/`

Leksička prevođenja

Inicijalni Java izvorni kod se pretvara u nizove riječi u tri koraka:

1. Prvo se `Unicode` kodovi oblika `\uxxxx` (ASCII znakovi) **pretvaraju u odgovarajuće** `Unicode` **znakove** (u UTF-16 kod odgovarajućeg znaka).
2. Rezultantni niz `Unicode` znakova se **pretvara u niz ulaznih znakova i terminatore linija**.
3. Nizovi ulaznih znakova i terminatora linija se **pretvaraju u niz ulaznih elemenata**, koji **nakon odbacivanja praznina i komentara čine riječi**. Te riječi su **terminalni simboli** Sintaksne gramatike.

Unicode kodovi

Formalno definiramo:

UnicodeUlazniZnak:
UnicodeKod
NeobrađeniUlazniZnak

UnicodeKod:
\ *UnicodeOznaka HexZnam. HexZnam. HexZnam. HexZnam.*

UnicodeOznaka:
u {u}

HexZnamenka:
(nešto od)
0 1 2 3 4 5 6 7 8 9 a b c d e f A B C D E F

NeobrađeniUlazniZnak:
Bilo koji `Unicode` znak

Terminatori linije, ulazni elementi i riječi

TerminatorLinije:

`ASCII` znak `LF` ili *nova linija* (eng. newline)
`ASCII` znak `CR` ili *povratak* (eng. return)
`ASCII` znak `CR` nakon kojeg slijedi `ASCII` znak `LF`

UlazniZnak:

UnicodeUlazniZnak bez `CR` i `LF`

Ulaz:

UlazniElement [Sub]

UlazniElement:

Praznina
Komentar
Riječ

Terminatori linije, ulazni elementi i riječi

Riječ:

- *Identifikator*
- *Ključna riječ*
- *Literal*
- *Separator*
- *Operator*

Sub: `ASCII SUB` znak, *control-Z* (zamjenski znak, ispiše se na mjestu znaka koji je nevežaći, koruptiran ili se ne može prikazati na danom uređaju).

Bjeline, komentari i identifikatori

Bjelina:

`ASCII SP` znak, *razmaknica*
`ASCII HT` znak, *horizontalni tab*
`ASCII FF` znak, *prijelaz na novu stranicu* (eng. form feed)
TerminatorLinije

Komentar:

TradicionalanKomentar
KomentarKrajLinije

TradicionalanKomentar:

* *ZavršetakKomentara*

ZavršetakKomentara:

* *ZavršetakKomentaraZvijezdica*
NeZvijezdica ZavršetakKomentara

Bjeline, komentari i identifikatori

ZavršetakKomentaraZvijezdica:
/
* *ZavršetakKomentaraZvijezdica*
NeZvijezdicaNeKosaCrta ZavršetakKomentara

NeZvijezdica:
UlazniZnak ali ne *
TerminatorLinije

NeZvijezdicaNeKosaCrta:
UlazniZnak ali ne * ili /
TerminatorLinije

KomentarKrajLinije:
/ / UlazniZnak

Bjeline, komentari i identifikatori

Identifikator:

IdentifikatorskiZnakovi ali ne *KljučnaRiječ* ili *LogičkiLiteral* ili *NullLiteral*

IdentifikatorskiZnakovi:

JavaSlovo JavaSlovoIliZnamenka

JavaSlovo: bilo koji `Unicode` znak koji je *Java slovo*

JavaSlovoIliZnamenka:

Proizvoljni `Unicode` znak koji je *Java slovo ili znamenka*

Ključne riječi

Ne mogu se koristiti kao identifikatori.

`abstract`	`continue`	`for`	`new`	`switch`
`assert`	`default`	`if`	`package`	`synchronized`
`boolean`	`do`	`goto`	`private`	`this`
`break`	`double`	`implements`	`protected`	`throw`
`byte`	`else`	`import`	`public`	`throws`
`case`	`enum`	`instanceof`	`return`	`transient`
`catch`	`extends`	`int`	`short`	`try`
`char`	`final`	`interface`	`static`	`void`
`class`	`finally`	`long`	`strictfp`	`volatile`
`const`	`float`	`native`	`super`	`while`

`_` (underscore)

Kontekstualne ključne riječi

Mogu se interpretirati kao ključne riječi ili riječi ovisno o kontekstu u kojem se javljaju.

`exports`	`opens`	`requires`	`uses`
`module`	`permits`	`sealed`	`var`
`non-sealed`	`provides`	`to`	`with`
`open`	`record`	`transitive`	`yield`

Literali

Literal:

- *CjelobrojniLiteral*
- *RealniLiteral*
- *LogičkiLiteral*
- *ZnakovniLiteral*
- *StringLiteral*
- *TekstBlok*
- *NullLiteral*

CjelobrojniLiteral:

- *DekadskiCjelobrojniLiteral*
- *HeksadekadskiCjelobrojniLiteral*
- *OktalniCjelobrojniLiteral*
- *BinarniCjelobrojniLiteral*

DekadskiCjelobrojniLiteral:

- *DekadskiBroj [SufiksCjelobrojnogTipa]*

Literali

HeksadekadskiCjelobrojniLiteral:
    HeksadekadskiBroj [SufiksCjelobrojnogTipa]

OktalniCjelobrojniLiteral:
    OktalniBroj [SufiksCjelobrojnogTipa]

BinarniCjelobrojniLiteral:
    BinarniBroj [SufiksCjelobrojnogTipa]

SufiksCjelobrojnogTipa:
    (nešto od)
    l L

DekadskiBroj:
    0
    NeNulaZnamenka [Znamenke]
    NeNulaZnamenka Podvlaka Znamenke

Literali

NeNulaZnamenka:

(nešto od)

1 2 3 4 5 6 7 8 9

Znamenke:

Znamenka

Znamenka [ZnamenkeIPodvlake] Znamenka

Znamenka:

0

NeNulaZnamenka

ZnamenkeIPodvlake:

ZnamenkaIliPodvlaka {ZnamenkaIliPodvlaka}

ZnamenkaIliPodvlaka:

Znamenka

_

Literali

Podvlake:

_ {_}

HeksadekadskiBroj:

0 x HeksadekadskeZnamenke

0 X HeksadekadskeZnamenke

HeksadekadskeZnamenke:

HeksadekadskaZnamenka

HeksadekadskaZnamenka [HeksadekadskeZnamenkeIPodvlake] HeksadekadskaZnamenka

HeksadekadskaZnamenka:

(jedna od)

0 1 2 3 4 5 6 7 8 9 a b c d e f A B C D E F

HeksadekadskeZnamenkeIPodvlake:

HeksadekadskaZnamenkaIliPodvlaka {HeksadekadskaZnamenkaIliPodvlaka}

Literali

HeksadekadskaZnamenkaIliPodvlaka:
    HeksadekadskaZnamenka

    _

OktalniBroj:
    0 OktalneZnamenke
    0 Podvlake OktalneZnamenke

OktalneZnamenke:
    OktalneZnamenke
    OktalnaZnamenka [OktalneZnamenkeIPodvlake] OktalnaZnamenka

OktalnaZnamenka: (jedna od)
    0 1 2 3 4 5 6 7

OktalneZnamenkeIPodvlake:
    OktalneZnamenkeIliPodvlake {OktalneZnamenkeIliPodvlake}

Literali

OktalneZnamenkeIliPodvlake:
    OktalnaZnamenka

    _

BinarniBroj:
    0 b *BinarneZnamenke*
    0 B *BinarneZnamenke*

BinarneZnamenke:
    BinarneZnamenke
    BinarnaZnamenka [BinarneZnamenkeIPodvlake] BinarnaZnamenka

BinarnaZnamenka:
    (jedna od)
    0 1

BinarneZnamenkeIPodvlake:
    BinarnaZnamenkaIliPodvlaka {BinarnaZnamenkaIliPodvlaka}

Literali

BinarnaZnamenkaIliPodvlaka:
    BinarnaZnamenka
    _

RealniLiteral:
    DekadskiRealniLiteral
    HeksadekadskiRealniLiteral

DekadskiRealniLiteral:
    Znamenke . [*Znamenke*] [*Eksponent*] [*SufiksRealnogTipa*]
    . *Znamenke* [*Eksponent*] [*SufiksRealnogTipa*]
    Znamenke Eksponent [*SufiksRealnogTipa*]
    Znamenke [*Eksponent*] *SufiksRealnogTipa*

Eksponent:
    IndikatorEksponenta BrojSPredznakom

Literali

IndikatorEksponenta:
(jedan od)
e E

BrojSPredznakom:
[*Predznak*] *Znamenke*

Predznak:
(jedan od)
+ -

SufiksRealnogTipa
(jedan od)
f F d D

HeksadekadskiRealniLiteral:
HeksadekadskiSignifikand BinarniEksponent [SufiksRealnogTipa]

Literali

HeksadekadskiSignifikand:

HeksadekadskiBroj [.]

0 x *[HeksadekadskeZnamenke]* . *HeksadekadskeZnamenke*

0 X [*HeksadekadskeZnamenke*] . *HeksadekadskeZnamenke*

BinarniEksponent:

IndikatorBinarnogEksponenta BrojSPredznakom

IndikatorBinarnogEksponenta:

(jedno od)

p P

LogičkiLiteral:

(jedan od)

true false

ZnakovniLiteral:

' *JedanZnak* '

' PosebanKod '

Literal

JedanZnak:
    UlazniZnak ali ne ' ili \

StringLiteral:
    " {*ZnakStringa*} "

ZnakStringa:
    Ulazni znak ali ne " ili \
    PosebanKod

NullLiteral: null

Blokovi teksta

BlokTeksta: " " " {BjelinaBlokaTeksta} TerminatorLinije
{ZnakBlokaTeksta} " " "

BjelinaBlokaTeksta:
    Bjelina ali ne *TerminatorLinije*

ZnakBlokaTeksta:
    UlazniZnak ali ne \
    PosebniKod
    TerminatorLinije

Posebni kodovi

\ b (pomak unatrag (eng. backspace) BS, `Unicode` \u0008)
\ s (razmaknica (eng. space) SP, `Unicode` \u0020)
\ t (horizontalni tab HT, `Unicode` \u0009)
\ n (prijelaz u novi red LF, `Unicode` \u000a)
\ f (prijelaz na novu stranicu FF, `Unicode` \u000c)
\ r (povratak na početak linije (eng. carriage return) CR, `Unicode` \u000d)
\ *TerminatorLinije* (nastavak linije, nema `Unicode` reprezentacija)
\ " (dvostruki navodnik ", `Unicode` \u0022)
\ ' (jednostruki navodnik ', `Unicode` \u0027)
\ (obrnuta kosa crta (eng. backslash) `Unicode`\u005c)
OktalniKod (oktalna vrijednost, `Unicode` \u0000 to \u00ff)

Oktalni kod: \ *OktalnaZnamenka*
\ *OktalnaZnamenka OktalnaZnamenka*
\ *NulDoTri OktalnaZnamenka OktalnaZnamenka*

Posebni kodovi, separtori i operatori

OktalnaZnamenka:

(jedan od)

0 1 2 3 4 5 6 7

NulDoTri:

(jedan od)

0 1 2 3

Separatori:

```
(   )   {   }   [   ]   ;   ,   .   ...   @   ::
```

Operatori:

```
=   >   <   !   ~   ?   :   ->
==  >=  <=  !=  &&  ||  ++  --
+   -   *   /   &   |   ^   %   <<   >>   >>>
+=  -=  *=  /=  &=  |=  ^=  %=  <<=  >>=  >>>=
```

Primjer - Java izvorni kod zapisan Unicode kodom

```
\u0070\u0075\u0062\u006c\u0069\u0063\u0020\u0020\u0020\u0020
\u0063\u006c\u0061\u0073\u0073\u0020\u0055\u0067\u006c\u0079
\u007b\u0070\u0075\u0062\u006c\u0069\u0063\u0020\u0020\u0020
\u0020\u0020\u0020\u0020\u0073\u0074\u0061\u0074\u0069\u0063
\u0076\u006f\u0069\u0064\u0020\u006d\u0061\u0069\u006e\u0028
\u0053\u0074\u0072\u0069\u006e\u0067\u005b\u005d\u0020\u0020
\u0020\u0020\u0020\u0020\u0061\u0072\u0067\u0073\u0029\u007b
\u0053\u0079\u0073\u0074\u0065\u006d\u002e\u006f\u0075\u0074
\u002e\u0070\u0072\u0069\u006e\u0074\u006c\u006e\u0028\u0020
\u0022\u0048\u0065\u006c\u006c\u006f\u0020\u0077\u0022\u002b
\u0022\u006f\u0072\u006c\u0064\u0022\u0029\u003b\u007d\u007d
```

Klasa se zove *Ugly*, stoga se odgovarajuća java datoteka mora zvati *Ugly.java*. Prevodimo naredbom `javac Ugly.java` te pokrenemo naredbom `java Ugly`.

Izlaz programa???

Hello World

Primjer - Leksički elementi *Java* izvornog koda

```
1  public class HelloWorld {
2  //KR   S   KR S     I       S
3      public static void main(String [] args){
4      //KR   S   KR   S KR S I   S I     SSSS   I SS
5          int i=1,j=2, k=0;
6        //KR   IOLSIOLSSIOLS
7          System.out.println("Hello world!\n");
8          //I    S I S   I     S       StrL    PK SS
9         int f=2_1;
10       //KR SIOLPLS
11        int g=0x12, h=012;
12     //KR SIO HB SSIO OBS
13       String s = null;
14     // I     SISOS NL S
15      double d = 0.25d;
16    // KR   S ISOSZ.ZZSRTS
17    //              ___RL__
18     }
19   //S
20   }
21 //S
```