Univerza v Ljubljani
Fakulteta za računalništvo in informatiko
Univerzitetni študijski program, 3. letnik

Sistemska programska oprema

predavatelj: doc. Tomaž Dobravec

Dinamično izvajanje programov

Osnovne ideje

D*inamično izvajanje porazdeljenih računalniških aplikacij* vključuje eno ali več "naprednih" komponent:

- **Globalne podatke**
- **Globalno kodo**
- **Povezovanje objektov**
- **Sistemsko neodvisnost**

Potrebujemo spremenjeno zasnovo sistemske programske opreme, ki bo podpirala sistemsko neodvisnost in prenosljivost aplikacij.

Vmesna koda

- Prevajalnik prevede program v vmesno kodo + optimizacija
- Vmesno kodo izvajamo na poljubnem sistemu z interpreterjem
- Vmesna koda se izvaja znotraj navideznih izvajalnih okolij

Prednosti:

- Varnost
- Majhno število prevajalnikov
- Preprosta prenosljivost

Slabosti:

- Počasnejše delovanje

Implementacije

- Prve ideje se pojavijo že leta 1970.

- ETH Pascal
  - zaradi slabih računalnikov in ponesrečene implementacije ideja ni zaživela

- Java (Sun, IBM, Oracle)
  - 1991 skupina "the Green Team" pri Sun-u začne z razvojem
  - prva prava verzija (Java 1.0) leta 1996.

- .NET in C# (Microsoft, HP, Intel)
  - 1997 Sun toži Microsoft zaradi nespoštovanja osnovne javanske ideje "napiši enkrat, uporabi povsod"
  - 2000 Microsoft odgovori z .NET in C#
  - 2001 Sun dobi tožbo, Microsoft preneha z distribucijo Jave

Javina arhitektura

Javino arhitekturo sestavljajo:

- programski jezik Java,
- programski vmesnik za aplikacije (API),
- format razrednih zbirk (class files) in
- navidezni stroj (JVM).

JVM je srce javine arhitekture

- omogoča izvajanje zložne kode
- zagotavlja sistemsko neodvisnost, varnost, omrežno prenosljivost
- sestavljajo ga: začetni razredni nalagalnik in izvajalnik

Java tolmač (interpreter)

Izvajalnik tolmači zložno kodo in jo izvede. Tolmačenje je izvedeno na enega od naslednjih načinov:

- **Tolmačenje kode vsakič znova** . Prednost: preprosta implementacija. Slabost: počasnost.
- **Sprotno prevajanje** (just-in-time compiler, JIT) : ob prvem klicu metode se njena koda prevede v strojni jezik, prevod se optimizira; JVM si optimiziran prevod zapomni in ga uporabi ob vsakem naslednjem klicu. Prednost: zelo hitro izvajanje. Slabost: velika poraba pomnilnika.
- **Prilagodljivo optimiranje** (adaptive optimizer): naložena koda se najprej sproti tolmači; JVM pri tem ugotavlja, kateri odseki kode se največkrat kličejo – te dele prevede v strojni jezik in jih optimizira.

Dinamično izvajanje programov pri Javi

Klasičen (statičen) življenjski cikel programske opreme, ki vsebuej faze **prevajanje/povezovanje/nalaganje/izvajanje** dobi v javanskem svetu drugačno podobo.

a) Prevajanje v objektno kodo

- nadomesti ga prevajanje v zložno kodo (bytecode)

b) Povezovanje in nalaganje

- spremeni se vrstni red: razredi se najprej naložijo, nato se med seboj povežejo

c) Nalaganje

- razredi se naložijo na poljubno lokacijo (ni absolutnega nalaganja);
- vsa sklicevanja na polja, metode in razrede potekajo preko simboličnih imen
- prenaslavljanje pri nalaganju ni potrebno.
- za nalaganje poskrbi t.i. **razredni nalagalnik**.

Dinamično izvajanje programov pri Javi

d) Povezovanje

- povezovanje v javanskem svetu pomeni razreševanje simboličnih imen;
- simbolično ime v DNK se zamenja z absolutnimi naslovi;
- razreševanje se izvede samo enkrat;
- pri kasnejših sklicevanjih (iz istega ali drugega mesta v programu) na vpis v DNK se uporabi absolutni naslov.

Dinamično izvajanje programov pri Javi
Razreševanje simboličnih imen

- Program mora biti pravilno povezan šele v času izvajanja. Natančneje: povezavo do polja, metode ali razreda potrebujemo šele takrat, ko se izvajani programski ukaz na omenjeni tip sklicuje.

- Različne izvedbe JVM se razreševanja lotijo na različne načine:

  - **Zgodnje razreševanje**: vnaprej se razreši celoten nabor konstant; morebitne napake (npr. odsotnost metode) se javi šele ob uporabi!
  - **Pozno razreševanje**: vsako sklicevanje na nabor konstant se opravi v trenutku, ko se med izvajanje programa pojavi.
  - **Vmesna možnost**: nekateri simboli se razrešijo ob nalaganju, nekateri ob izvajanju.

Osnovno pravilo: uporabniška izkušnja mora biti enaka kot pri poznem razreševanju!

Razredne zbirke

- Razredna zbirka = .class datoteka
- Razredna zbirka predstavlja standardiziran zapis **prevedene** kode, primerne za nalaganje, povezovanje in izvajanje v navideznem stroju.
- Logično je razredna zbirka v dinamičnem svetu ekvivalentna objektnemu modulu iz statičnega sveta.

Razredne zbirke

- Razredna zbirka hrani podatke v binarni obliki zapisane na standarden način, ki ga razume vsak javanski navidezni stroj.
- Ker je razredna zbirka namenjena tudi prenosu po omrežjih, morajo biti podatki v njej zakodirani na čim bolj učinkovit (kompakten) način.
- Osnovni podatek v zbirki je 8-bitni zlog. Večzložne vrednosti so zapisane v formatu big-endian.
- Posamezne komponente zbirke (atributi, metode, …) so zapisani z variabilnim formatom (brez vnaprej predpisane dolžine), dolžina zapisa je podana pred zapisom samim.

Struktura razredne zbirke - primer

```
public class Vsota {
  final static int ID=15;
  int zadnjaVsota;
  protected int vsota(int a, int b) {
    zadnjaVsota = a + b;
    return zadnjaVsota;
  }
  public static void main(String args[]) {
    System.out.println("ID=" + ID);

    Vsota v = new Vsota();
    int   r = v.vsota(4,9);
    System.out.println(r);
  }
}
```

- `hexdump –C Vsota.class`

Struktura razredne zbirke

```
ClassFile {
  u4 magic;
  u2 minor_version;
  u2 major_version;
  u2 constant_pool_count;
  cp_info constant_pool[constant_pool_count-1];
  u2 access_flags;
  u2 this_class;
  u2 super_class;
  u2 interfaces_count;
  u2 interfaces[interfaces_count];
  u2 fields_count;
  field_info fields[fields_count];
  u2 methods_count;
  method_info methods[methods_count];
  u2 attributes_count;
  attribute_info attributes[attributes_count];
}
```

Opomba: u1 pomeni en zlog, u2 2 zloga, u4 pa 4 zloge.

Struktura razredne zbirke

- **magic**: čarobna številka (0xCA FE BA BE)

- **minor_version, major_version**: glavna številka in podštevilka. JVM na podlagi teh dveh številk presodi, ali zna interpretirati dani razred (če je glavna številka razreda večja od največje glavne številke, ki jo še podpira JVM, pride do napake).

- Številčenje:

  J2SE 8 = 52 (0x34 hex)

  J2SE 7 = 51 (0x33 hex),

  ….

  JDK 1.4 = 48 (0x30 hex),
  JDK 1.3 = 47 (0x2F hex),
  JDK 1.2 = 46 (0x2E hex),
  JDK 1.1 = 45 (0x2D hex).

Struktura razredne zbirke
Nabor konstant

- **Nabor konstant** (**constant_pool_count, constant_pool**): poleg programske kode najpomembnejša struktura v datoteki, saj povezovanje brez nje ni možno!

- Nabor konstant je tabela različnih struktur, ki predstavljajo znakovne konstante, imena razredov, vmesnikov in atributov ter druge konstante.

- En vpis v naboru konstant je naslednje oblike:

```
cp_info {
    u1 tag;
    u1 info[];
 }
```

Struktura razredne zbirke
Nabor konstant

Tip posamezne konstante je podan s prvim zlogom (Tag) v stukturi, število in pomen preostalih podatkov je odvisen in tipa:

Tag	**Tip**	**Število dodatnih zlogov**
1	CONSTANT_Utf8	2(len) + len
3	CONSTANT_Integer	4
4	CONSTANT_Float	4
5	CONSTANT_Long	4+4
6	CONSTANT_Double	4+4
7	CONSTANT_Class	2
8	CONSTANT_String	2
9	CONSTANT_Fieldref	2+2
10	CONSTANT_Methodref	2+2
11	CONSTANT_InterfaceMethodref	2+2
12	CONSTANT_NameAndType	2+2

Struktura razredne zbirke
Nabor konstant

- **Utf8** … niz znakov; prva dva zloga predstavljata dolžino niza (len), sledi len znakov.

```
11 Utf8                 : ID
27 Utf8                 : a
28 Utf8                 : b
29 Utf8                 : main
30 Utf8                 : ([Ljava/lang/String;)V
```

- **Integer, Float:** konstanta tipa int ali float, ki je v kodi podana npr. s `static final` določili.

```
14 Integer              : bytes: 00 00 00 0F
```

Struktura razredne zbirke
Nabor konstant

- **Long, Double**: konstanta tipa long ali double. Posebnost: te konstante zasedejo 8 zlogov oziroma DVA zaporedna slota v tabeli! Torej: število vseh slotov NI enako številu konstant!!

  Primer: če ID v kodi spremenim iz int v long, dobim:

```
13 Utf8          : ConstantValue
14 Long          : bytes: 00 00 00 00 00 00 00 0F
16 Utf8          : zadnjaVsota
```

- **Class** … ime razreda; parameter je indeks na Utf8 opis imena

```
6 Class          : 44
44 Utf8          : Vsota
```

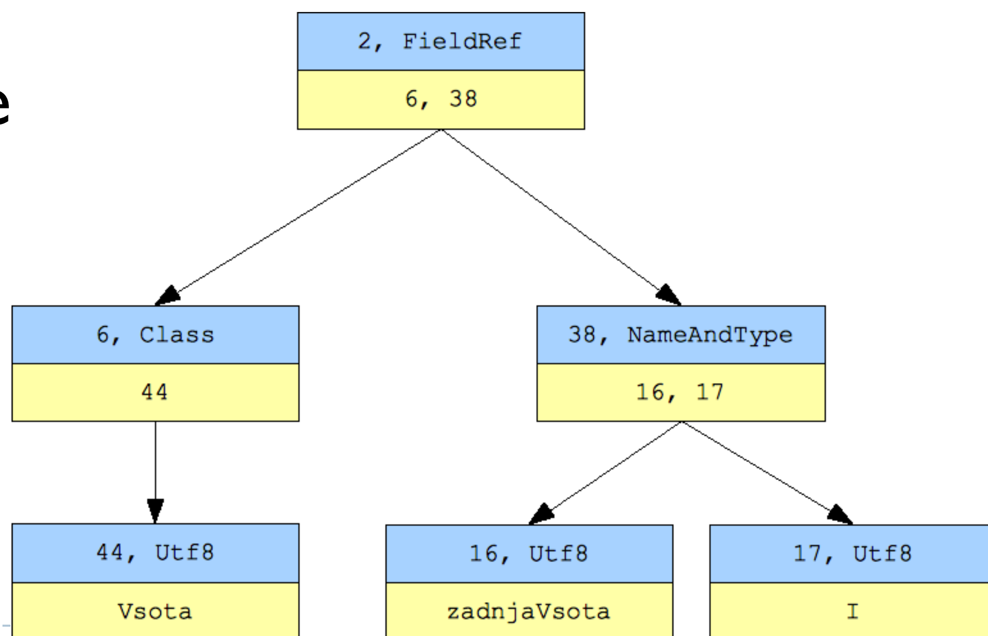
Struktura razredne zbirke
Nabor konstant

- **String …** znakovna konstanta; parameter je indeks na Utf8

```
4   String                   : 41
41  Utf8                     : ID=15
```

- **FieldRef , MethodRef, InterfaceMethodref**… podatek o atributu, metodi in metodi vmesnika ; parametra sta tipa Class in NameAndType

```
 2 Fieldref     : 6 38
 6 Class        : 44
38 NameAndType  : 16 17
44 Utf8         : Vsota
16 Utf8         : zadnjaVsota
17 Utf8         : I
```



Struktura razredne zbirke
Nabor konstant

- **NameAndType** … podaja ime in tip atributa ali metode; parametra sta kazalca na Utf8 opisa za ime in deskriptor.

Deskriptor – primeri

BaseType **Character**	**Type**	**Interpretation**
B	`byte`	signed byte
C	`char`	Unicode character
D	`double`	double-precision floating-point value
F	`float`	single-precision floating-point value
I	`int`	integer
J	`long`	long integer
L<classname>;	`reference`	an instance of class `<classname>`
S	`short`	signed short
Z	`boolean`	`true` or `false`
[	`reference`	one array dimension

Struktura razredne zbirke
Dostopnostna določila razreda

- `access_flags` v naboru konstatnt

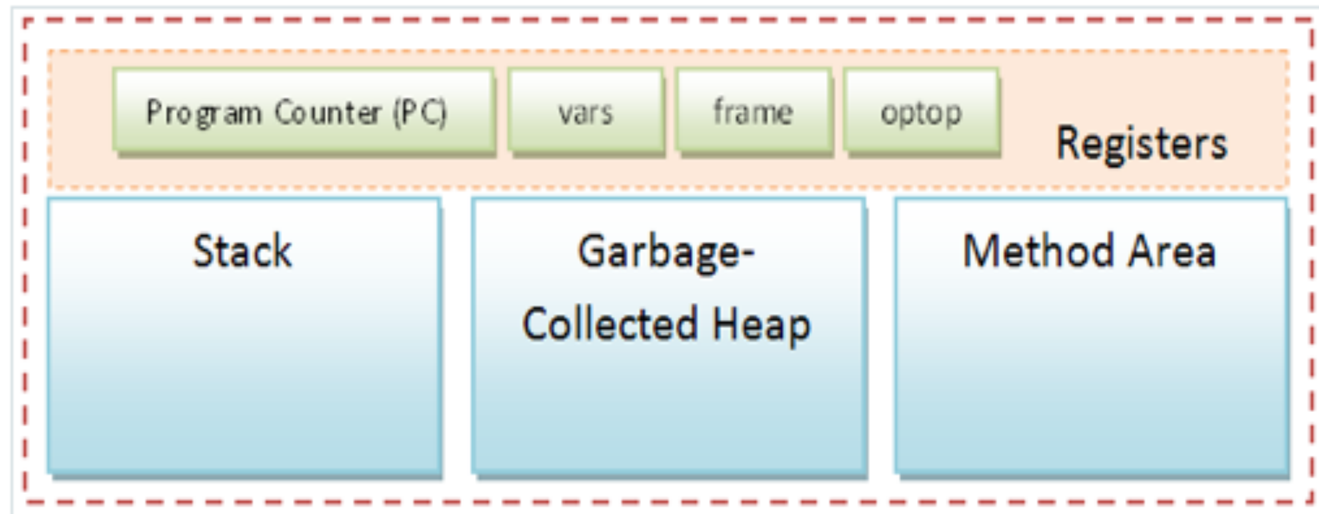
Flag Name	Value	Interpretation
ACC_PUBLIC	0x0001	Declared `public`; may be accessed from outside its package.
ACC_FINAL	0x0010	Declared `final`; no subclasses allowed.
ACC_SUPER	0x0020	Treat superclass methods specially when invoked by the *invokespecial* instruction.
ACC_INTERFACE	0x0200	Is an interface, not a class.
ACC_ABSTRACT	0x0400	Declared `abstract`; must not be instantiated.
ACC_SYNTHETIC	0x1000	Declared synthetic; not present in the source code.
ACC_ANNOTATION	0x2000	Declared as an annotation type.
ACC_ENUM	0x4000	Declared as an `enum` type.

JVM 32

- **Naslovni prostor** JVM je 32 biten (naslovljivih je 4GB pomnilnika)
- **Beseda** (word) v JVM je 32 bitna.
- Interni javini JVM registri so 32 bitni.
- JVM razpolaga z naslednjimi **primitivnimi tipi**:
  - byte (8 bits), short (16 bits), int (32 bits), long (64 bits), float (32 bits), double (64 bits) in char (16 bits).
  - Char je nepredznačen Unicode znak, ostali primitivni tipi so predznačeni.
  - Referenca na object (object handle) – 32 bitni naslov na objekt na kopici

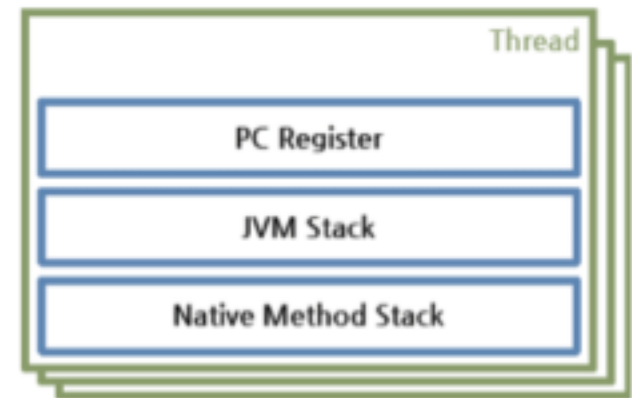
Elementi JVM

- Java navidezni stroj sestavljajo:
  - sklad,
  - interni registri (pc, vars, frame, optop)
  - kopica,
  - področje za metode.



Sklad

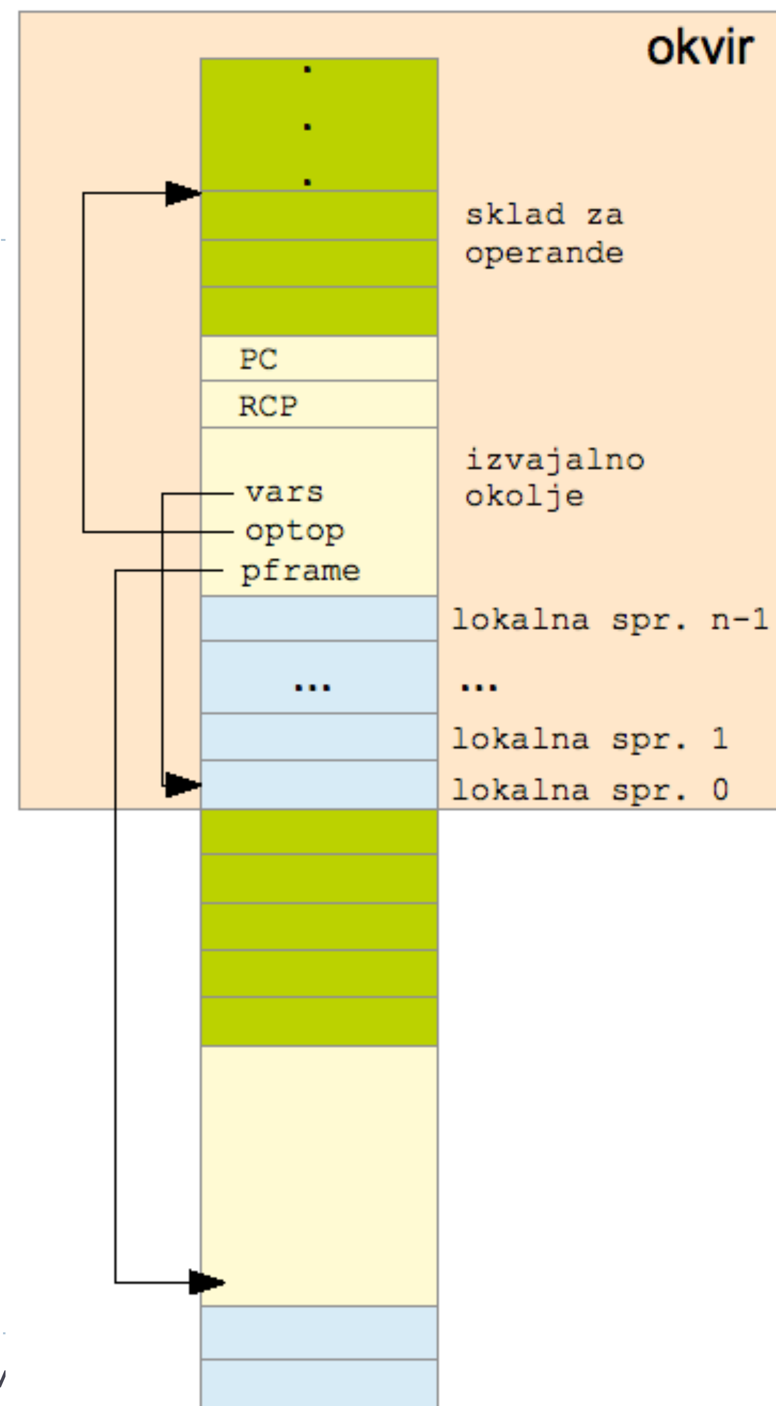
- Osnovna podatkovna struktura za hranjenje parametrov, naslovov in vmesnih rezultatov je sklad (*JVM stack*).
- Vsaka izvajalna nit dobi svoj **sklad**, PC register in sklad za domorodne (`native`) metode.
- Vsaki metodi se ob klicu dodeli del sklada – **okvir** (frame) - na katerem so shranjene lokalne spremenljivke in vmesni računski rezultati.



Okvir

Okvir vsebuje naslednje dele

- operandski sklad
  (operand stack)
- tabela lokalnih spremenljivk
  (local variables table)
- izvajalno okolje
  (execution environment)



Operandski sklad

- Na **operandskem skladu** se hranijo parametri za bytecode operacije in rezultati teh operacij;
- ta sklad je nujno potreben, saj JVM ne pozna registrov in se zato vse operacijo izvajajo preko sklada.
- Operandski sklad trenutne metode je vedno najvišje na skladu; register `optop`, ki kaže ne vrh tega sklada, posledično kaže na vrh celotnega sklada za to nit.

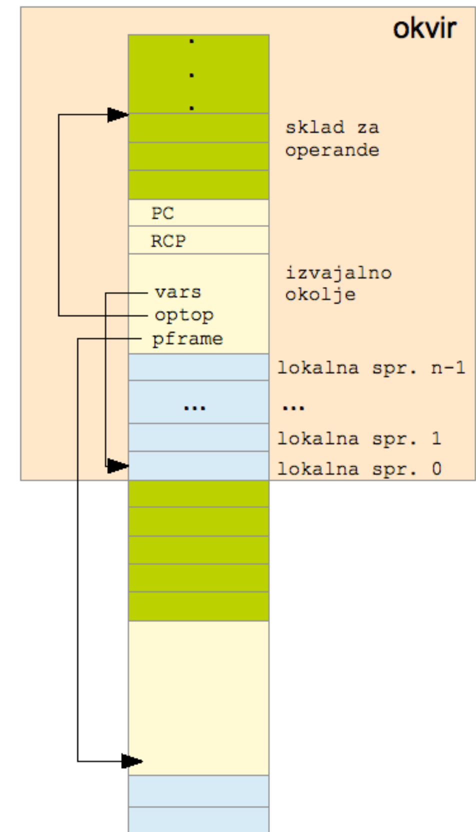
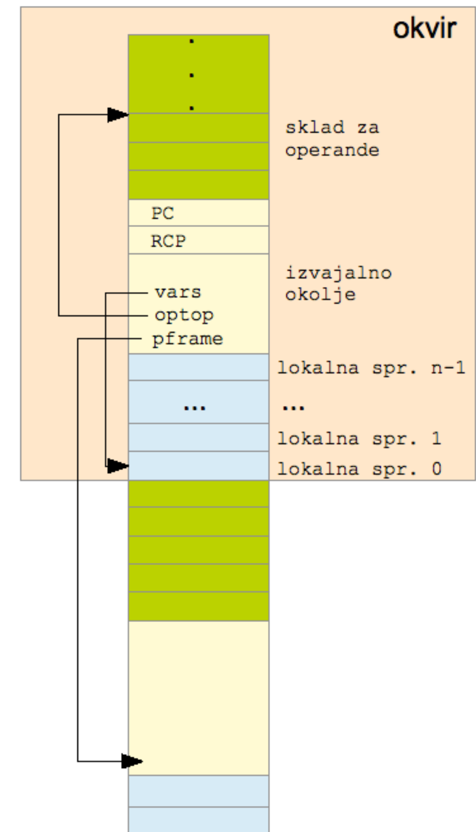


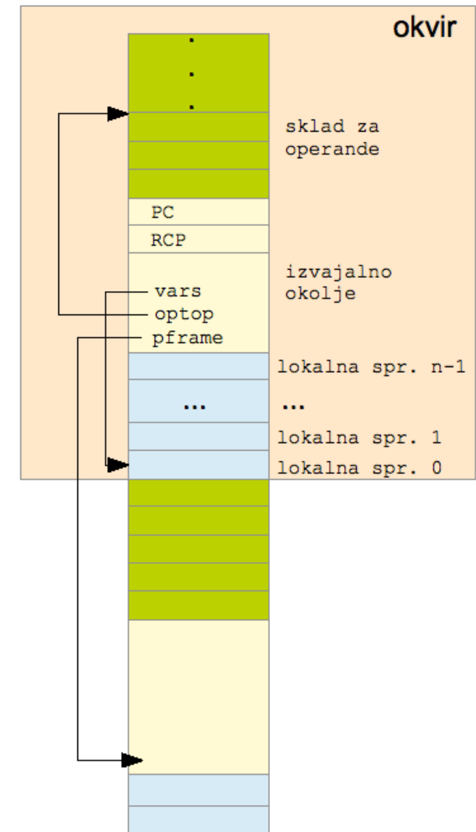
Tabela lokalnih spremenljivk

- **Tabela lokalnih spremenljivk** hrani formalne parameter in lokalne spremenljivke.
- Velikost te tabele se določi v času prevajanja.
- Pri konstruktorjih in virtualnih metodah je na prvem mestu zapisana referenca `this`;
- Prvi formalni parameter ima indeks 0 (za statične metode) in 1 za ostale (konstruktorji in virtualne metode);
- Lokalne spremenljivke sledijo formalnim parametrom (če je `m` indeks zadnjega formalnega parametra, potem je indeks prve lokalne spremenljivke `m+1`)



Registri

- JVM ima samo 4 registre
  - PC … naslov trenutno izvajanega ukaza
  - optop … vrh op. sklada
  - vars … lokalne spremenljivke
  - frame … kazalec na trenutni okvir



- JVM je "stack-oriented machine" (pri izvajanju operacij uporablja izključno sklad), zato omenjeni štirje registri zadoščajo

Kopica (garbage collected heap)

- Tu "živijo" vsi objekti (samo podatkovni del) trenutnega programa.
- Ukaz `new` rezervira del kopice in vrne referenco na rezerviran prostor.
- Programer v Javi ne more sam sprostiti rezerviranega pomnilnika; namesto njega to nalogo opravlja JVM čistilec (garbage collector).

Področje za metode

- V področju za metode se nehaja koda (bytecodes) vseh razredov in objektov.

- Podatek o trenutno aktivnem ukazu (njegov naslov) se hrani v programskem števcu.

- Ko se ukaz v JVM izvrši, se programski števec popravi
  - pri skočnih ukazih dobi vrednost parametra,
  - pri ostalih ukazih pa vrednost naslednjega ukaza

- Vsaka metoda se v tem področju pojavi samo enkrat.

Osnovno o izvajanju zložne kode

- Javanski program je zaporedje **enozložnih** operacijskih kod
- JVM izvaja ukaz za ukazom → izvajalna nit
- Vsaka nit dobi svoj **javin sklad** (Java stack), ki je razdeljen na **okvire** (frames)
- Del sklada, kamor se shranijo vmesni rezultati, se imenuje **operandski sklad**
- JVM s posameznim ukazom zložne kode dosega podatke, ki so shranjeni
  - kot takojšnje vrednosti v sami zložni kodi,
  - kot lokalne spremenljivke ali
  - kot vpisi na operacijskem skladu
- Način dostopa do podatkov je opredeljen z operacijsko kodo ukaza.

Osnovno o izvajanju zložne kode Mnemoniki

- Mnemoniki so okrajšava za angleško besedo, ki predstavlja operacijo ukaza.
- Mnemoniki, ki se nanašajo na ukaze za delo s podatki, s prvo črko nakazujejo, za kakšen tip podatkov so namenjeni

Pomen predpone:

Predpona	Tip	Primer
b	byte	bastore
s	short	saload
i	int	iconst_0
l	long	lload
c	char	castore
f	float	fcmpl
d	double	dadd
a	referenca	areturn

Osnovno o izvajanju zložne kode
Delo s podatki

- Podatki so na kopici shranjeni v taki obliki, da zasedejo čim manj prostora:
  - byte porabi 8 bitov,
  - char in short porabita 16 bitov,
  - int, float in reference 32 bitov,
  - long in double porabita 64 bitov
- Podatki so na skladu shranjeni tako, da je z njimi najlažje delati in da je manj možnosti za napake:
  - long in double zasedeta 8 zlogov,
  - vsi ostali zasedejo 4 zloge.

Osnovno o izvajanju zložne kode
Izvajanje programa

- JVM nima registrov za hranjenje vmesnih rezultatov

- Ves prenos parametrov poteka preko operandskega sklada
  - primer: pred klicom ukaza `iadd` morata biti na skladu zapisani dve celi števili, po izvršitvi ukaza je na skladu rezultat.

- Lokalne spremenljivke posamezne metode so shranjene v tabeli lokalnih spremenljivk; do posamezne lokalne spremenljivke dostopamo z indeksom

Osnovno o izvajanju zložne kode
Izvajanje programa

- Razen ukaza `iinc`, ki poveča vrednost podane lokalne spremenljivke, noben ukaz ne dela z lokalnimi spremenljivkami (uporaba sklada).

- Tudi za prenos med lokalnimi spremenljivkami in kopico potrebujemo posredovanje sklada.

  - Primer: če želimo vrednost lokalne `int` spremenljivke shraniti v tabelo, moramo na sklad odložiti vrednost, referenco na tabelo in indeks v tabeli; prenos potem opravi ukaz `iastore`, klican brez parametrov.

Operacijske kode in ukazi

Javanski ukazni nabor vsebuje tri skupine ukazov:

- **200 “splošnonamenskih” ukazov**, ki se lahko pojavijo v razredni zbirki
- **3 rezervirani ukazi** za razhroščevanje (`breakpoint`) in ustvarjanje programskih klicev in prestreznikov (`impdep1` in `impdep2`).
- **25 hitrih ukazov**; te ukazi nadomestijo “počasne” ukaze po povezovanju (`invokevirtual` ⇨ `invokevirtual_quick`)

Operacijske kode in ukazi
Skupine ukazov

Splošnonamenski ukazi se delijo v naslednje skupine:

- operacije s skladom in lokalnimi spremenljivkami
- pretvorba tipov
- celoštevilska aritmetika
- logične operacije
- aritmetika s plavajočo vejico
- operacije z objekti in polji
- operacije za nadzor izvajalnega toka
- klici metod in vrnitve
- zaključni stavki (klic podprogramov in vrnitve iz njih)
- sinhronizacija niti

Operacijske kode in ukazi
Skupine ukazov

Vstavljanje konstant na sklad

- Za čimhitrejše izvajanje javanske kode so na voljo ukazi, ki zapišejo na sklad nekatere najpogosteje uporabljane konstante
  - integer 0 … 5, long 0 … 1, float 0 … 2, double 0 … 1

Primer: `iconst_0` zapiše integer konstanto 0 na sklad
`fconst_2` zapiše float konstanto 2 na sklad

- Poljubno konstanto vstavimo z ukazom

```
ldc <številka_konstante>
```

pri čemer je `številka_konstante` številka v naboru konstant.

Operacijske kode in ukazi
Skupine ukazov

Prenos med lokalnih spremenljivk in skladom

- Za prve tri lokalne spremenljivke obstajajo ukazi tipa

`iload_0` ... iz prve lokalne spr. prenesi integer vrednost na sklad

`fstore_2` ... v tretjo lokalno spr. shrani float vrednost iz sklada

za ostale pa

```
iload  <lokalna_spremenljivka>
astore <lokalna_spremenljivka>
```

Operacijske kode in ukazi
Skupine ukazov

Delo s skladom

`pop`	zbriše (zavrže) besedo z vrha sklada
`dup`	podvoji besedo na vrhu sklada
`pop2`	zbriše dve besedi s sklada
`dup2`	podvoji prvi dve besedi na skladu
`swap`	zamenja dve besedi na skladu

Operacijske kode in ukazi
Skupine ukazov

Pretvorba tipov

- Ukazi za pretvorbo med tipi se uporabljajo za pravilno uporabo aritmetičnih in logičnih operacij med različnimi tipi ter pri prenosu med skladom in kopico.

- Primer ukazov: `i2l, i2f, i2d, …,`
  `l2d, ….., d2f`

Operacijske kode in ukazi
Skupine ukazov

Aritmetične in logične operacije

- Delajo nad operandi na skladu.
- Poznamo operacije za celoštevilske in realne vrednosti.
- Logične operacije se izvajajo samo nad celoštevilskimi tipi.

Nekateri ukazi:

- iadd, ladd, isub, lmul, lrem, …
- fadd, frem, dmul, …
- iand, ior, ishr, ishl, …

Operacijske kode in ukazi
Skupine ukazov

Nadzor toka

- Java pozna dva tipa skočnih ukazov:
  - primerjava vrha sklada z 0 (`ifeq`, `ifne`, ...),
  - primerjava prvih dveh elementov na skladu (`if_cmpeq`, `if_cmplt`, ...).
- Oba ukaza imata po en parameter, ki predstavlja `<odmik>` (kam naj se preusmeri izvajanje, če je pogoj izpolnjen).
- Obstaja tudi ukaz `goto <odmik>`.

Operacijske kode in ukazi
Skupine ukazov

Nekateri ostali ukazi:

`new`	Ustvari nov objekt
`getfield`	Vzame polje iz objekta
`putfield`	Napolni polje iz objekta
`newarray`	Ustvari novo tabelo
`araylength`	Ugotovi dolžino tabele
`invokevirtual`	Klic virtualne metode
`invokestatic`	Klic statične metode

Podrobneje o naboru ukazov:

http://en.wikipedia.org/wiki/Java_bytecode_instruction_listings

Primer preprostega programa

```
4  void cmpadd(int a, int b)
5    byte bajt = 5;
6    long c; int d = bajt;
7    c = a + b;
8    if (c>10)
9      d=1;
10   else
11     d=2;
12 }
```

```
void cmpadd(int b, int bajt) {
       /* L5 */
       0 iconst_5;
       1 istore_3;                 /* bajt */
       /* L6 */
       2 iload_3;                  /* bajt */
       3 istore 6;                 /* d */
       /* L7 */
       5 iload_1;                  /* a */
       6 iload_2;                  /* b */
       7 iadd;
       8 i2l;
       9 lstore 4;                 /* c */
       /* L8 */
       11 lload 4;                 /* c */
       13 ldc2_w 16;               /* 10 */
       16 lcmp;
       17 ifle 9;
       /* L9 */
       20 iconst_1;
       21 istore 6;                /* d */
       23 goto 6;
       /* L11 */
       26 iconst_2;
       27 istore 6;                /* d */
       /* L12 */
       29 return;
   }
```

Razredni nalagalnik

- Razredni nalagalnik poskrbi, da se po potrebi naloži (prebere iz medija) iskana razredna zbirka.

- Osnovne naloge:
  - nalaganje razreda (branje iz medija) v tabelo zlogov,
  - pretvorba zlogov v JVM predstavitev razreda (Class)
  - (opcijsko) razreševanje simbolov

Razredni nalagalnik

Začetni razredni nalagalnik (*bootstrap class loader*)

- To je osnovna različica razrednega nalagalnika, in je del JVM.
- Pisan je v istem jeziku kot JVM (recimo v C).
- Dostavljati zna razrede, ki so shranjeni lokalno na disku.
- Začetni razredni nalagalnik je nujno potreben, da se celoten postopek nalaganja sploh lahko začne.

Razredni nalagalnik

Uporabniški razredni nalagalnik

- To je razred, ki ga napiše uporabnik
- Izveden je kot naslednik razreda `ClassLoader`; uporabnik nadomesti samo tiste metode, v katerih implementira različno obnašanje;
- Najpreprostejši uporabniški razredni nalagalnik:

```
public class SimpleClassLoader extends ClassLoader {

}
```

- Z njim lahko nalagamo tudi razrede iz oddaljenih lokacij (na primer preko interneta)

Razredni nalagalnik

Imensko področje (namespace)

- Vsi razredi, ki so naloženi z istim nalagalnikom, tvorijo eno imensko področje,
- Medsebojna povezovanja med razredi so mogoča le znotraj istega imenskega področja,
- **Osnovno pravilo nalaganja**: razred se vedno naloži z istim nalagalnikom, s katerim je bil naložen razred, ki je sprožil njegovo nalaganje.
- Pravilna uporaba razrednih nalagalnikov preprečuje interakcije med neodvisnimi programi.
  - Primer: če dva preko mreže naložena programa naložim vsakega s svojim nalagalnikom, se med sabo ne vidita.
  - Tak mehanizem uporablja na primer spletni brskalnik pri nalaganju apletov.