MASARYKOVA UNIVERZITA
FAKULTA INFORMATIKY

Studie nástrojů pro trasování a testování programů v Javě

BAKALÁRSKA PRÁCA

Matej Majdiš

Brno, 2015

Prehlásenie

Prehlasujem, že táto bakalárska práca je mojím pôvodným autorským dielom, ktoré som vypracoval samostatne. Všetky zdroje, pramene a literatúru, ktoré som pri vypracovaní používal alebo z nich čerpal, v práci riadne citujem s uvedením úplného odkazu na príslušný zdroj.

Vedúci práce: RNDr. Adam Rambousek

Zhrnutie

TODO...

Kľúčové slová

TODO...

Poďakovanie

TODO...

Obsah

1 **Úvod** 2
1.1 *Cieľ práce* 3
1.2 *Členenie práce* 3
2 **Bajtkód** 4
2.1 *Štruktúra class súboru* 4
2.1.1 Reprezentácia dátových typov 6
2.1.2 Premenné tried a inštancií 7
2.1.3 Metódy 8
2.1.4 Atribúty 9
2.2 *Inštrukcie JVM* 9
2.2.1 Dátové typy 10
2.2.2 Architektúra a inštrukčná sada 10
3 **Classloadery** 13
3.0.3 Dynamické načítavanie tried 13
3.0.4 Znovunačítanie triedy 13
4 **Byteman** 15
4.1 *Byteman Agent* 16
4.2 *Jazyk* 17
5 **Nástroj Javasist** 18
Literatúra 19
A **Tabuľky** 20

Kapitola 1

Úvod

Java je dnes jedným z najpoužívanejších programovacích jazykov. Od syntakticky podobných programovacích jazykov ako napríklad C++, alebo C# sa líši prekladom zdrojových tried do medzikódu často označovaného ako bajtkód (*bytecode, p-code, portable code*).

Preklad a spustenie programu napísaných v programovacom jazyku Java prebieha v nasledujúcich fázach:

1. Preklad do medzikódu: Java compiler [1] preloží zdrojový kód do bajtkódu. V praxy to znamená, že každej triede, alebo rozhraniu je priradený súbor *class*, ktorý obsahuje inštrukcie popisujúce fungovanie danej triedy.

2. Načítanie a Interpretácia: Virtuálny stroj Javy (ďalej len JVM [2]) načíta inštrukcie *class* súboru potrebnej triedy, ktoré ďalej spracúva jedným z nasledujúcich spôsobov:

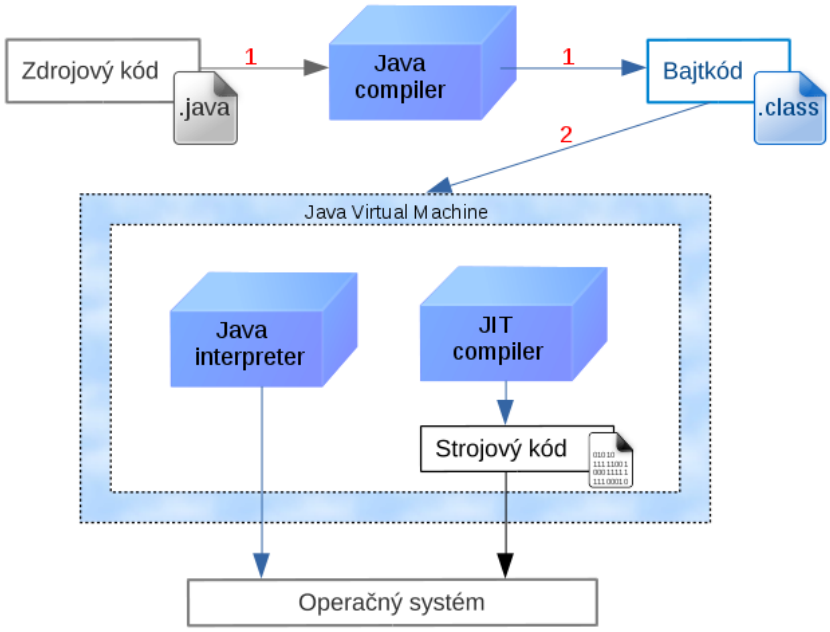
   - JIT prekladač (*Just In Time compiler*): Štandardne je z bajtkódu najskôr vygenerovaný strojový (*machine code*) konkrétneho zariadenia, ktorý je následne interpretovaný priamo vykonávaný procesorom.
   - Java interpreter: Ďalším spôsobom spracovania bajtkódu je využitie Java interpretru, ktorý bajtkódkód spracováva a sám interpretuje.

Výhodou prekladu do bajtkódu je jeho a prenositeľnosť. Samotný bajtkód je platformovo nezávislí. Program teda nieje nutné prispôsobovať jednotlivým operačným systémom, ktoré sa líšia len v implementácií JVM.

---

1. Najčastejšie využívaným Javacompilerom je *javac*, ktorý je súčasťou JDK (Java Development Kit).
2. Java Virtual Machine, špecifikácia je dostupná na `http://docs.oracle.com/javase/specs/jvms/se7/html`

Class súbory obsahujúce bajtkód je možné za behu programu modifikovať. Jednotlivé triedy a rozhrania aplikácie uložené v týchto súboroch podľa potreby načítava JVM. Vkladanie nových metód a tried na úrovni bajtkódu, pred načítaním *class* súboru do JVM sa nazýva injekcia bajtkódu (*bytecode injection*, ďalej len BI). Pridávanie novej funkcionality pomocou BI bez nutnosti zastavenia behu programu je často využívané pri testovaní a trasovaní (*tracing*) programov.



Obr. 1.1: Grafické znázornenie prekladu a spustenia programu, zdroj: vlastné spracovanie

1.1 Cieľ práce

TODO...

1.2 Členenie práce

TODO...

Kapitola 2

Bajtkód

Po preklade zdrojových kódov prekladačom *javac* je každej triede, prípadne rozhraniu programu priradený jeden *class* súbor popisujúci jej funkcionalitu.

Pri načítavaní *class* súbru JVM dostane takzvaný prúd inštrukcií bajtkódu (*bytecode stream*) pre každú metódu triedy. V prípade volania konkrétnej metódy za behu programu sú inštrukcie danej metódy vykonávané. Každá z inštrukcí bajtkódu je reprezentovaná číselnou hodnotou nazývanou *opcode*. Zároveň má každá inštrukcia aj textovú podobu (*mnemonic*), ktorá je jej menom. V *class* súboorch sú inštrukcie uložené v numerickej podobe.

Táto kapitola popisuje formát *class* súboru a následne stručne charakterizuje inštrukčnú sadu bajtkódu. [1]

2.1 Štruktúra *class* súboru

Class súbor pozostáva z jednej *ClassFile* štruktúry. *ClassFile* štruktúra jednoznačne identifikuje konkrétnu triedu, prípadne rozhranie, definuje jej premenné a metódy.

Nasledujúci popis definuje sadu datových typov. Typy *u1*, *u2*, a *u4* reprezentujú neznamienkové jedno, dvoj, alebo štvorbajtové číslo. *ClassFile* je zobrazená ako pseudoštruktúra v notácií jazyka C. Obsah štruktúry je popísaný ako po sebe nasledujúce položky.

Formát *ClassFile* štruktúry

```
ClassFile {
  u4 magic;
  u2 minor_version;
  u2 major_version;
  u2 constant_pool_count;
  cp_info constant_pool[constant_pool_count-1];
```

1. Nasledujúci text vychádza zo 4. až 6. kapitoly špecifikácie JVM [LYBB13].

```
  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];
}
```

Konštanta *magic* identifikuje formát súboru *class*, jej hodnota je 0xCAFEBABE.

Položky *minor_version* a *major_version* určujú verziu *class* súboru. Napríklad *minor_version* s hodnotou *m* a *major_version* s hodnotou *M* indikujú verziu s hodnotou *M.m*.

Hodnota položky *constant_pool_count* je rovná počtu záznamov v *constant_pool[]* plus jeden.

Úložisko záznamov *constant_pool[]* (v podobe poľa štruktúr) zahŕňa rôzne konštanty: mená tried a rozhraní, mená premenných a iné. Každý záznam *constant_pool[]* sa skladá zo značky (*tag*) a indexu (*name index*). Značka určuje typ záznamu. Tabuľka značiek je uvedená v prílohe A.1. Pomocou unikátneho indexu, je možné odkazovať sa na záznamy v d'alších častiach bajtkódu. Existuje niekoľko typov štruktúr [2] reprezentujúcich rôzne druhy záznamov. Napríklad štruktúra *CONSTANT_String_info* reprezentuje objekty typu *String* zatiaľ čo štruktúry *CONSTANT_Methodref_info* a *CONSTANT_InterfaceMethodref_info* reprezentujú metódy danej triedy, alebo rozhrania.

Hodnota *access_flags* popisuje oprávnenia prístupu k informáciam a vlastnosti tejto triedy, respektíve rozhrania pomocou indikátorov. Napríklad nastavenie indikátora *ACC_INTERFACE* znamená, že *class* súbor popisuje rozhranie. Tabuľka indikátorov je uvedená v prílohe A.2.

Položka *this_class* obsahuje index *constant_pool[]* odkazujúci na štruktúru typu *CONSTANT_Class_info* [3]. Reprezentuje triedu, respektíve rozhranie, definované týmo class súborom.

Hodnotou *super_class* je taktiež index *constant_pool[]* odkazujúci na štruktúru typu *CONSTANT_Class_info*. Reprezentuje priamu nadtriedu triedy

2. Všetky štruktúry *constant_pool[]* sú popísane v špecifikácií JVM [LYBB13].
3. *CONSTANT_Class_info* je štrukura *constant_pool*, ktorá reprezentuje triedu, alebo rozhranie.

definovanej týmto *class* súborom. V prípade, že tento *class* súbor popisuje rozhranie, index odkazuje na triedu *Object*. Trieda *Obejct* má ako jediná hodnotu *super_class* nulovú.

Počet rozhraní, ktoré trieda implementuje vyjadruje položka *interface_count*, v prípade rozhrania je táto položka rovná počtu priamych nadrozhraní.

Pole *interfaces[]* obsahuje indexy *constant_pool[]* odkazujúce na štruktúru typu *CONSTANT_Class_info*. Zahŕňa indexy všetkých rozhraní, ktoré sú implementované triedou, prípadne priamymi nadrozhraniami *class* súboru.

Položka *fields_count* je rovná počtu premenných triedy a premenných inštancí (*fields*) *class* súboru.

Štruktúry typu *field_info* sú združené v poli *fields[]*. Toto pole zahŕňa každú premennú danej triedy, respektíve rozhrania. Nezahŕňa zdedené atribúty. Podrobne sa štruktúrou *field_info* sa zaoberá kapitola 2.1.2.

Hodnata položky *methods_count* vyjadruje počet štruktúr *method_info* v poli *methods[]*.

Položka *methods[]* je pole štruktúr typu *method_info*. Každá štruktúra *method_info* popisuje metódu tejto triedy, respektíve rozhrania. Zahŕňa konštruktory, metódy triedy a metód inštancí. Neobsahuje však žiadne zdedené metódy. Štruktúru *method_info* popisuje kapitola 2.1.3.

Hodnota *attributes_count* je rovná počtu atribútov poľa *attributes[] class* súboru.

Pole *attributes[]* obsahuje štruktúry typu *attribute_info*. Atribútmi štruktúry *ClassFile* sú napríklad: *SourceFile*, *Deprecated*, *InnerClasses* a iné. Atribút *SourceFile* slúži na reprezentáciu mena *class* súboru. Pole *attributes[] class* súboru môže obsahovať maximálne jeden takýto atribút. Atribút *Depricated* môže byť použitý v prípade, že bola daná trieda nahradená (*depricated*). Pri volaní takejto triedy môže prekladač upozorniť užívateľa, že sa odkazuje na nahradenú triedu [4]. Vo všeobecnosti sa štruktúre *attribute_info* sa venuje kapitola 2.1.4.

2.1.1 Reprezentácia dátových typov

Dátové typy sú v *class* súboroch reprezentované vo formáte reťazcov s kódovaním *UTF-8*. Delíme ich na:

- dátové typy premenných
  - primitívne dátové typy
  - referenčné dátové typy

4. Rovnakým spôsobom je možné atribút *Depricated* aplikovať aj na premenné a metódy.

  - polia

- dátové typy metód

Primitívnym dátovým typom (*byte*, *integer*, ...) je priradený popis v podobe znaku (*B*, *I*, ...). Napríklad premenná typu *int* je reprezentovaná znakom: *I*.

Referenčné dátové typy reprezentuje popis v tvare: *L<classname>;*, kde *classname* je meno triedy, alebo rozhrania daného referenčného dátového typu. Premenná typu *Object* je interpretovaná ako *java/lang/Object;*.

Identifikačný reťazec jednorozmerného poľa typu *T* sa značí *[T*, pričom počet znakov *[* je rovný dimenzii poľa. Napríklad premenná typu: *double d[][][]* generuje reťazec: *[[[D*.

Reťazec dátového typu metódy sa skladá z reťazcov pre dátový typ parametrov, ohraničených v zátvorkách *(P*)* a reťazca pre dátový typ návratovej hodnoty *R*. Tvar reťazca dátového typu metódy je potom *(P*)R*. V prípade návratovej hodnoty *null* je reťazcom návratovej hodnoty znak *V*. Napríklad metódu *boolean long pow (int n, int k)* reprezentuje reťazec: *(II)J*, v prípade metódy *Object method(byte b)* by šlo o reťazec: *(B)Ljava/lang/Object;*. Komplexný prehľad reprezentácie datových typov je uvedený v prílohe A.3.

2.1.2 Premenné tried a inštancií

Premenné tried inštancií (*fields*) *class* súboru sú v poli *fields[]* reprezentované pomocou štruktúry *field_info*. Formát štruktúry *field_info* je nasledovný:

Štruktúra *field_info*

```
field_info {
  u2 access_flags;
  u2 name_index;
  u2 descriptor_index;
  u2 attributes_count;
  attribute_info attributes[attributes_count];
}
```

Položka *access_flags* je indikátorom oprávnenia prístupu k danej premennej. Mená indikátorov spolu s ich interpretáciou a hodnotou sú uvedené v prílohe A.4.

Dvojbajtová hodnota *name_index* je index *constant_pool[]* reprezentujúci meno premennej

Podobne ako *name_index* aj *descriptor_index* je dvojbajtová položka odkazujúca sa na štruktúru v *constant_pool*. Na rozdiel od mena premennej však

popisuje datový typ premennej. Reprezentáciou datových typov sa zaoberá kapitola 2.1.1.

Položka *attributes_count* vyjadruje počet atribútov v poli *attributes[]*.

Pole *attributes[]* môže obsahovať ľubovoľné množstvo atribútov popisujúcich premennú. Štruktúra reprezentujúca atribút je daná všeobecným predpisom *attributeq_info*. Atribúty premenných musia byť reprezentované jednou zo štruktúr *ConstantValue*, *Synthetic*, *Signature*, *Deprecated*, *RuntimeVisibleAnnotations*, alebo *RuntimeInvisibleAnnotations*. Atribút *ConstantValue* popisuje konštantné statické premenné, *Synthetic* je používaný u položiek, ktoré sa nevyskytujú v zdrojovom kóde. Štruktúrou *attribute_info* sa zaoberá kapitola 2.1.4.

2.1.3 Metódy

Každá metóda triedy, prípadne rozhrania je v poli *methods[]* uložená pomocou štruktúry *method_info*. Štruktúra *method_info* má nasledujúci formát:

Štruktúra *method_info*

```
method_info {
  u2 access_flags;
  u2 name_index;
  u2 descriptor_index;
  u2 attributes_count;
  attribute_info attributes[attributes_count];
}
```

Indikátor *access_flags* zahŕňa nastavenia prístupových práv a vlastností metódy. Tabuľaka indikátorov *access_flags* štruktúry *method_info* sa nachádza v prílohe A.5.

Položky *name_index* a *descriptor_index* sú podobne ako u štruktúry *field_info* indexmi do *constant_pool*. Tieto indexy v *constant_pool* odkazujú na štruktúry popisujúce meno a datový typ metódy. Reprezentácia dátových typov je popísaná v kapitole 2.1.1.

Hodnotou položky *attributes_count* je počet atirbútov poľa *attributes[*.

Pole *attributes[]* zahŕňa dodatočné atribúty (položky) danej metódy. Každá položka poľa je reprezentovaná všeobecným predpisom *attributes_info*. Počet štruktúr v poli nieje obmedzený, každá položka však musí byť jednou zo štruktúr: *Code*, *Exceptions*, *Synthetic*,*Signature*, *Deprecated*, *RuntimeVisibleAnnotations*, *RuntimeInvisibleAnnotations*, *RuntimeVisibleParameterAnnotations*, *RuntimeInvisibleParameterAnnotations*, alebo *AnnotationDefault*. Atribút *Code* je jedným z najdôležitejších. Obshauje inštrukcie bajtkódu popisujúce fungovanie metódy. Okrem metód deklarovaných ako abstraktná,

alebo natívna musí každá metóda obsahovať práve jeden atribút *Code*. Atribút *Exceptions* zahŕňa indexy výnimiek, ktoré metóda vyhadzuje. Popisom formátu štruktúry *attributes_info* sa zaoberá kapitola 2.1.4.

2.1.4 Atribúty

Pojem atribút v tomto texte vyjadruje atribúty používané v poli *attributes[]* štruktúr *field_info*, *method_info* a *Code_attributes*. Všeobecný predpis všetkých atribútov je vyjadrený štruktúrou *attribute_info*. Existuje niekoľko základných preddefinovaných atribútov: *SourceFile*, *ConstantValue*, *Code*, *Exceptions*, *InnerClasses*, *Synthetic*, *LineNumberTable*, *LocalVariableTable*, *Deprecated* a iné. Líšia sa funkcionalitou a využitím jednotlivými časťami *class* súboru. Všetky atribúty vychádzajú z už spomínaného všeobecného predpisu *attribute_info*, ktorý má nasledujúci formát:

Štruktúra *attribute_info*

```
attribute_info {
  u2 attribute_name_index;
  u4 attribute_length;
  u1 info[attribute_lenght];
}
```

Položka *attributes_name_index* je indexom do *constant_pool* odkazujúcim na meno atribútu. Tento proces sa nazýca kontrola formátu (*format checking*). Prvé štyri bajty musia obsahovať tzv. magickú konštantu *magic*. Všetky rozoznané atribúty musia mať správnu dĺžku

2.2 Inštrukcie JVM

Po načítaní *class* súboru JVM sa JVM nasjkôr uistí, že je tento súbor v správnom formáte popísanom v kapitole 2.1. Štvorbajtová položka *attribute_length* je rovná hodnote vyjadrujúcej dĺžku následných informácií uložených v *info[attribute_length]*. Informácie sa líšia na základe odlišnej funkcionality a využitá jednotlivých atribútov. *Class* súbor nesmie byť skrátený ani obsahovať nadbytočné bajty, takisto úložisko *constant_pool* nesmie obsahovať žiadne nerozoznateľné informácie.

Inštrukcie bajtkódu načítanej metódy sú uložené v poli *code[]* atribútu *Code*, štruktúry *method_info* daného *class* súboru. Štruktúra *Code_attribute* reprezentujúca atribút *Code* musí spĺňať obmedzenia definované JVM. Tieto obmedzenie rozdeľujeme na dve základné kategórie:

- Statické obmedzenia: Stanovujú rozloženie inštrukcií v poli *code* a priradenie operandov jednotlivým inštrukciám. Niektorými z nich sú napríklad:
  - prvá inštrukcia musí začínať na indexe 0,
  - pole *code* nesmie byť prázdne.
- Štrukturované obmedzenia: Špecifikujú vzťahy medzi inštrukciami JVM. Ide o podmienky ako napríklad:
  - žiadna lokálna premenná nemôže byť volaná predtým ako jej bola priradená hodnota,
  - pred volaním (nestatickej) metódy respektíve premennej musí byť inicializovaná inštancia triedy ktorá ju obsahuje.

Prekladače jazyka Java generujú *class* súbory, ktoré spĺňajú požiadavnky popísané v predchádzajúcom odseku. JVM však nemá žiadnu záruku, že *class* súbor, ktorý požaduje bol generovaný prekladačom. Metódou verifikácie [5] *class* súboru môže JVM určiť či daný súbor pochádza z dôveryhodného zdroja.

2.2.1 Dátové typy

Dátové typy JVM delíme do troch základných kategórií:

- Primitívne dátové typy: byte, short, int, long, boolean, float, double.
- Referenčné dátové typy: pole, inštancia triedy, rozhranie.
- Typ *returnAddress*: používaný výhradne ištrukciami *jsr*, *ret* a *jsr_w*.

Väčšina uvedených typov má veľkosť 32 bitov, typy *long* a *double* sú však 64 bitové, preto zaberajú dva sloty v zásobníku.

2.2.2 Architektúra a inštrukčná sada

Architektúra JVM je založená na datovej štruktúre zásobník [6], ktorej základnými operáciami sú *push* - vloženie prvku do zásobníka a *pop* - výber

5. Ďalšie príklady obmedzení a podorbný popis verifikácie *class* súborov je dostupný v špecifikácií JVM [LYBB13]
6. Dátová štruktúra zásobník (*stack*) funguje na princípe FIFO (*first in first out*), kde posledný vložený prvok je prvým vybraným.

prvku z vrcholu zásobníka. JVM nemá registre na ukladanie hodnôt, preto musia byť pred použitím všetky uložené na zásobník.

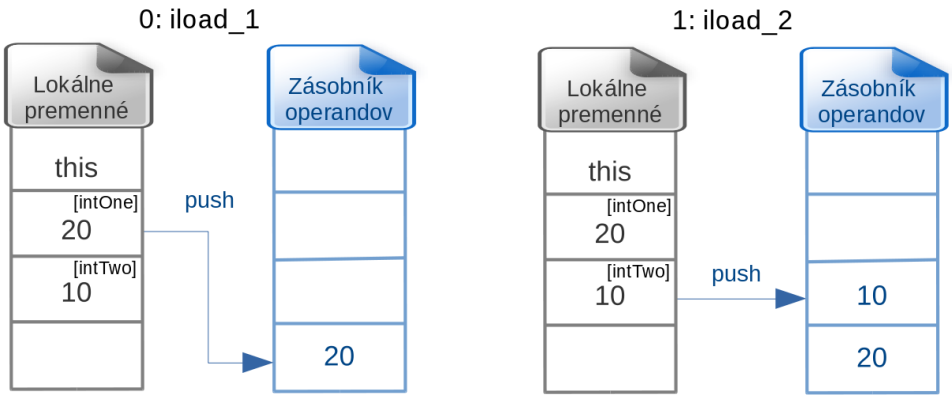
Na nasledujúcom jednoduchom príklade sú popísané základné inštrukcie bajtkódu pre prácu s premennými a konštantami.

Metóda *greatherThen* pred a po kompilácií

```
public int greaterThen(int intOne, int intTwo) {
  if (intOne > intTwo) {
    return 0;
  } else {
    return 1;
  }
}

0: iload_1
1: iload_2
2: if_icmple  7
5: iconst_0
6: ireturn
7: iconst_1
8: ireturn
```

Inštrukcie *iload_1* a *iload_2* pridajú do zásobníka operandov (ďalej len zásobník) hodnoty lokálnych premenných na indexoch 1 a 2. V tomto prípade ide o parametre *intOne* a *intTwo*.



Obr. 2.1: Znázornenie funkcionality inštrukcií *iload_1* a *iload_2*.

Vo všeobecnosti môžeme túto inštrukciu chápať ako *xload* s predponou extitx označujúcou ľubovoľný primitívny datový typ(napríklad: *lload* pre long, *fload* pre float). Existujú dva tvary, volania tejto inštrukcie:

- *load_<n>*, kde n označuje index (celé číslo) lokálnej premennej, zároveň musí platiť: $0 \leq n \leq 4$,
- *load vindex*, kde pozíciou lokalnej premennej je hodnota *vindex*.

Ďalšou inštrukciou je *if_icmple* s parametrom 7, ktorá porovná dva objekty na vrchole zásobníka a prejde na siedmu inštrukciu v prípade, že je hodnota položky na vrchole zásobníka väčšia ako hodnota druhej položky. V príklade sú na zásobníku len položky vložené predchádzajúcimi inštrukciami. Podmienka teda platí v prípade, že hodnota parametra *intOne* je menšia hodnota *intTwo*. Vo všeobecnosti je možné podmienené výrazy vyjadriť pomocou inštrukcií: *if_acmp<cond>*, *if_icmp<cond>*, *if<cond>*, *ifnonnull*, *ifnull*.

Inštrukcie *iconst_0* a *iconst_1* vložia na zásobník hodnotu 0 respektíve 1 v závislosti od vyhodnotenia podmienky *if_icmple*. Táto hodnota je následne vrátená inštrukciou *ireturn*. Inštrukcie *iconst_<n>*, *a ireturn* sú taktiež dostupné vo variantách s predponou ľubovoľného primitívneho dátového typu.

Dôkladný popis všetkých inštrukcií vrátene ich parametrov možno nájsť v špecifikácií JVM [LYBB13].

Kapitola 3

Classloadery

Class loader je objekt zodpovedný za načítavanie tried. Trieda *ClassLoader* je abstraktná. Pomocou mena *class* súboru by mal *class loader* nájsť a generovať obsah definujúci danú triedu. Každá trieda obsahuje referenciu na *ClassLoader*, ktorý ju definoval. [Ora11]

Zvyčajne je trieda do JVM načítaná len v prípade, že je potrebná. Načítané sú zároveň všetky triedy na ktoré sa odkazuje. Pomocou *class loaderov* je možné za behu programu dynamicky načítať d'alšie triedy, prípadne načítať nové inštancie pôvodných tried.

Pri štandardnom načítaní triedy niektorá z implementácií *ClassLoader* vykoná nasledujúce tri kroky:

1. Skontroluje či trieda už nebola načítaná
2. Ak nebola, požiada nadtriedu o načítanie danej triedy
3. V prípade, že nuspeje pokúsi sa načítať triedu pomocou vlastného *class loaderu*

3.0.3 Dynamické načítavanie tried

K načítaniu novej triedy za behu programu je potrebný *class loader*. Získať ho je možné pomocou príkazu *MyClass.class.getClassLoader();*. Novú triedu reprezentovanú súborom *class* následne vráti metóda *class loaderu*: *loadClass(class)*.

3.0.4 Znovunačítanie triedy

Dynamické znovunačítanie triedy je komplkovanejšie. Vstavané implementácie triedy *ClassLoader* vždy kontrolujú, či trieda už nebola do JVM načítaná. Preto nieje možné žiadnu triedu načítať dvakrát pomocou vstavaných *class loaderov*. Je nutné navrhnúť vlastnú implementáciu.

-

! PRÍKLAD VLASTNÉHO CLASSLOADERA - TODO ... !

-

Ďalšou komplikáciou je trieda *ClassLoader.resolve()*, ktorá zabezbečuje linkovanie. Táto trieda je *final*, z čoho vyplíva, že ju nieje možné prepísať, nepovolí však žiadnemu *class loaderu* linkovať dva-krát tú istú triedu. Preto je nutné pri kaďom ďalšom znovunačítaní triedy vytvoriť novú inštanciu *class loaderu*.

-

! PRÍKLAD POUŽITIA CLASSLOADERA - TODO ... !

Kapitola 4

Byteman

Byteman je nástroj manipulujúci s bajtkódom určný na zásah do bajtkódu Java aplikácií počas načítavania JVM, alebo za behu programu. Používa sa najmä na zjednodušenie trasovania a testovania aplikácií. Umožňuje používateľovi pridávať novú funkcionalitu do ktorejkoľvek časti programu. Funguje bez nutnosti prepisovania a opätovnej kompilácie pôvodnej aplikácie.

Byteman modifikuje bajtkód aplikácie za behu programu. Preto môže zmeniť Java kód, popisujúci časť treid JVM ako napríklad *String*, *Thread* a podobne. Vďaka tejto funkcionalite je taktiež možné napríklad trasovanie správania sa JVM.

Byteman používa jednoduchý jazyk ECA pravdiel [1] založený na Jave. Tieto ECA pravidlá používa na špecifikáciu kde, kedy a ako má byť pôvodný Java kód transformovaný aby modifikoval operáciu [Red].

Primárne bol *Byteman* určený na podoru testovania multivláknových a multi-JVM aplikácií za použitia techniky nazývanej *fault injection* [2]. Zahŕňa preto funkcionalitu, ktorá bola navrhnutá na riešenie problémov súvisiacich s týmto typom testovania. *Byteman* poskytuje podporu pre automatizáciu v štiroch hlavných oblastiach:

- trasovanie špecifických väzieb kódu a zobrazovanie stavu aplikácie, prípadne JVM,
- narúšanie normálneho priebehu zmenou stavov, volanie nenaplánovaných metód, vynucovanie návratových volaní, prípadne vyhadzovanie neočakávaných výnimiek,
- organizácia časovania aktivít vykonaných nezávislými vláknami aplikácie,

---

1. ECA (*event-condition-action*) pravidlá pozostávajú z udalosti, podmienky a akcie. Význam pravidla znamená: Ak nastane udalosť, skontrolu podmienku a v prípade, že platí, vykonaj akciu [Sel95].
2. TODO...

- monitorovanie a zhromažďovanie štatistík, sumarizujúcich aplikáciu a operácie JVM.

V súčasnosti je *Byteman* využívaný oveľa širšie ako nástroj na testovanie [Red].

Najjednoduchším použitím *Bytemana* je vkladanie kódu, ktorý trasuje správanie sa aplikácie. Táto metóda môže byť využitá na monitorovanie, alebo ladenie, ako aj na úpravu kódu pri testovaní a overenie, správneho fungovania aplikácie. Pri vkladaní kódu na veľmi špecifické miesta je možné vyhnúť sa režijným nákaldom, ktoré často rastú pri ladení, alebo trasovaní proguktu [Byt].

4.1 Byteman Agent

Aby mohol *Byteman* manipulovať s programom, musí na ňom bežať *Byteman Agent*, ktorý konfiguruje JVM pre prácu s pravidlami jeho jazyka.

Pri inštalácií agenta s prekladom programu je riešením použitie argumentu príkazu *java -javaagent*, ktorý zadáva cestu k *JAR* súboru popisujúcemu pravidlá jazyka. Agentovi je možné pomocou argumentov nakonfigurovať dve základné možnosti funkcionality:

- Základnou možnosťou je použiť argument *script:[PATH]*, ktorý načíta do programu skript definovaný pravidlami v súbore s cestou *PATH*

- V prípade potreby načítavania pravidiel do programu aj po spustení je nutné nastaviť argument *listener* na hodnotu *true*. Do takto spusteného programu je možné následne pomocou skriptu *bmsubmit.sh* pridávať a odoberať ľubovoľné pravidlá.

Byteman je nastavený aby neinjektoval kód do tried JVM. Pri zmene tried ako napríklad *String a Thread* je preto nutné zmeniť túto vlastnosť nastavením *system property org.jboss.byteman.transform.all*. Zároveň je nutné zaistiť, aby bol *Byteman Agent* načítaný (rovnako ako tieto triedy) defaultným (*bootstrap*) *classloaderom*.

Agenta je možné inštalovať taktiež do už bežaicich aplikácií [3] bez nutnosti ich opätovného spustenia. Slúži na to skipt *bminstall.sh*. *Byteman* je následne možné využiť ako nástroj na kontrolu správania sa programu [Red].

3. typicky ide o dlho bežiace aplikácie ako napríklad aplikačný server JBoss

4.2 Jazyk

TODO...

Kapitola 5

Nástroj Javasist

TODO...

Literatúra

[Byt] Byteman project description. `http://byteman.jboss.org`. Accessed: 2015-03-09.

[LYBB13] Tim Lindholm, Frank Yellin, Gilad Bracha, and Alex Buckley. *The Java Virtual Machine Specification, Java SE 7 Edition*. Addison-Wesley Professional, 1st edition, 2013. `http://docs.oracle.com/javase/specs/jvms/se7/html/`.

[Ora11] Oracle. *Class ClassLoader documentation*, 7th edition, 2011. `http://docs.oracle.com/javase/7/docs/api/java/lang/ClassLoader.html`.

[Red] Red Hat and individual contributors. *Byteman Programmer's Guide*. `http://downloads.jboss.org/byteman/2.2.1/ProgrammersGuide.pdf`.

[Sel95] Timos Sellis. *Rules in Database Systems: Second International Workshop, RIDS '95, Glyfada, Athens, Greece, September 25 - 27, 1995. Proceedings*. Lecture Notes in Artificial Intelligence. Springer, 1995.

Dodatok A

Tabuľky

Zdrojom nasledujúcich tabuliek je špecifikácia JVM [LYBB13].

Constant Type	Value
CONSTANT_Class	7
CONSTANT_Fieldref	9
CONSTANT_Methodref	10
CONSTANT_InterfaceMethodref	11
CONSTANT_String	8
CONSTANT_Integer	3
CONSTANT_Float	4
CONSTANT_Long	5
CONSTANT_Double	6
CONSTANT_NameAndType	12
CONSTANT_Utf8	1
CONSTANT_MethodHandle	15
CONSTANT_MethodType	16
CONSTANT_InvokeDynamic	18

Tabuľka A.1: Tabuľka značiek určujúcich typ záznamu v *constant_pool*. Stĺpec *Constant Type* označuje názov typu, stĺpec *value* priraďuje každému typu číselnú hodnotu.

Meno Indikátora	Hodnota	Interpretácia
ACC_PUBLIC	0x0001	Deklarovaná ako verejná; prístupná aj mimo balíka.
ACC_FINAL	0x0010	Deklarovaná ako final; žiadne podtriedy po inicializácií.
ACC_SUPER	0x0020	Volá metódu nadtriedy, hlavne inštrukcia invokespecial.
ACC_INTERFACE	0x0200	Je rozhranie, nie trieda.
ACC_ABSTRACT	0x0400	Deklarovaná ako abstraktná, nemôže byť inštanciovaná.
ACC_SYNTHETIC	0x1000	Deklarovaná ako synthetic, nieje prítomná v zdrojovom kóde.
ACC_ANNOTATION	0x2000	Deklarovaná ako typ annotation.
ACC_ENUM	0x4000	Deklarovaná ako typ enum.

Tabuľka A.2: Tabuľka indikátorov prístupových práv *ClassFile* štruktúry.

Reprezentácia pomocou reťazca	**Typ**	**Interpretácia**
B	byte	znamienkové celé číslo veľkosti jedného bajtu
C	char	Znak s kódovaním UTF-16
D	double	číselná hodnota s dvojitou presnosťou a plávajúcou desatinnou čiarkou
F	float	číselná hodnota s plávajúcou desatinnou čiarkou
I	int	celé číslo
J	long	celé číslo väčšieho rozsahu
L ClassName ;	referencia	inštancia triedy ClassName
S	short	znamienkové celé číslo krátkeho rozsahu
Z	boolean	pravda alebo nepravda
[	reference	jednorozmerné pole

Tabuľka A.3: Tabuľka reprezentácie datových typov pre premenné.

Meno Indikátora	Hodnota	Interpretácia
ACC_PUBLIC	0x0001	Deklarovaná ako verejná; prístupná aj mimo balíka.
ACC_PRIVATE	0x0002	Deklarovaná ako privátna; použiteľná len vrámci triedy, v ktorej bola definovaná.
ACC_PROTECTED	0x0004	Deklarovaná ako protected; prístupná aj podtriedam.
ACC_STATIC	0x0008	Deklarovaná ako statická.
ACC_FINAL	0x0010	Deklarovaná ako final; žiadne ďalšie priradenia po inicializácií.
ACC_VOLATILE	0x0040	Deklarovaná ako volatile; nemôže byť uložená do medzipamäte.
ACC_TRANSIENT	0x0080	Deklarovaná ako transient; nieje čítaná ani modifikovaná objektovým manažérom.
ACC_SYNTHETIC	0x1000	Deklarovaná ako synthetic, nieje prítomná v zdrojovom kóde.
ACC_ENUM	0x4000	Deklarovaná ako prvok objektu enum

Tabuľka A.4: Tabuľka indikátorov prístupových práv a vlastností štruktúry *field_info*.

Meno Indikátora	Hodnota	Interpretácia
ACC_PUBLIC	0x0001	Deklarovaná ako verejná; prístupná aj mimo balíka.
ACC_PRIVATE	0x0002	Deklarovaná ako privátna; použiteľná len vrámci triedy, v ktorej bola definovaná.
ACC_PROTECTED	0x0004	Deklarovaná ako protected; prístupná aj podtriedam.
ACC_STATIC	0x0008	Deklarovaná ako statická.
ACC_FINAL	0x0010	Deklarovaná ako final; nemôže byť prepísaná.
ACC_SYNCHRONIZED	0x0020	Deklarovaná ako synchronized; pri volaní je zabalená za použitia monitora.
ACC_BRIDGE	0x0040	Bridge metóda; je generovaná prekladačom.
ACC_VARARGS	0x0080	Deklarovaná s dynamickým počtom argumentov.
ACC_NATIVE	0x0100	Deklarovaná ako natívna; implementovaná v inom jazyku ako Java.
ACC_ABSTRACT	0x0400	Deklarovaná ako abstraktná, nieje implementovaná.
ACC_STRICKT	0x0800	Deklarovaná ako stricktfp, výpočty s plávajúcou čiarkou sú FP - strict.
ACC_SYNTHETIC	0x1000	Deklarovaná ako synthetic, nieje prítomná v zdrojovom kóde.

Tabuľka A.5: Tabuľka indikátorov prístupových práv a vlastností štruktúry *method_info*.