Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Načítání informací o třídách z bytecode

Plzeň, 2012 Josef Vopalecký

Zadání

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 10.5.2012

Abstract

The aim this thesis is to develop loader that will retrieve data from a bytecode into a data model JavaTypes.
The first part of this thesis introduces data types and the structure of the bytecode. Knowledge of the structure is needed for loader operation.
The second part of this thesis introduces the project Java Class Compatibility Checker developed by Faculty of Applied Sciences. This project includes JavaTypes, the project is already a JClassLoader implemented.
The last part of this thesis describes a new loader, which loads the data. The retrieved data is stored in the structure JavaTypes.

Obsah

1. Úvod

Cílem této bakalářské práce je převedení informací z Java™ bytecodu do struktury `JavaTypes`, která je na Západočeské univerzitě v Plzni vyvíjená několik let. Tato struktura je použitá pro komparátor tříd v projektu Java Class Compatibility checker.

V současné době je již implementován jeden způsob čtení bytecodu, který užívá knihoven `BCEL` [1] a `ASM` [2]. Tento způsob načítání je pro současné použití zbytečně moc rozsáhlý, komparátor potřebuje pouze zpracovat data obsažená v Java™ bytecodu do reprezentace JavaTypes.

Motivací k zadání této práce je snaha vyvinout loader, který má snadno předvídatelné chování a je snadno upravitelný při případné změně JavaTypes. Výsledný loader by měl nahradit již dříve implementovaný loader a být jeho plnohodnotnou náhradou. Jako základ by měl sloužit již realizovaný program na čtení bytecodu [3].

2. Java™

Java™ je programovací jazyk [4] vyvinut firmou Sun Microsystems. V současné době tuto firmu vlastní firma Oracle. Java™ je objektový jazyk vycházející z jazykových konstrukcí C++. Přidělování paměti je zde automatické, objekty jsou automaticky odstraňovány, když už nebudou použity, pomocí garbage collectoru. Java™ má velmi rozsáhlé knihovny [4], obsahuje například knihovny pro práci s vlákny, s komunikací po síti a s různými formáty souborů.

V této kapitole se budu zabývat datovými typy v jazyku Java™ a způsobem uložení instrukcí.

2.1. Java datové typy

Každá proměnná ve zdrojovém kódu programu má svůj datový typ a název, Java™ se v tomto ohledu chová velmi striktně a nedovolí do proměnné uložit jinou hodnotu, než jaký je typ dané proměnné. Datový typ také definuje operace, které lze s proměnou provádět. Datové typy jsou děleny na primitivní a referenční.

2.1.1. Primitivní datové typy

Primitivních datových typů je osm. Patří mezi ně `byte`, `short`, `int`, `long`, `float`, `double`, `char` a `boolean`. Celočíselné typy `byte`, `short`, `int`, `long` jsou znamínkové. Primitivní datové typy jsou v jazyku Java™ výjimkou, protože nejsou objekty.

2.1.2. Referenční datové typy

Referenčními datovými typy jsou třídy a rozhraní. Instance datových typů mají svůj stav přiřazen referencí. Při změně hodnoty na dané referenci změníme stav všem proměnným, které na ní mají referenci.

2.1.3. Datové typy null a void

Hodnota `null` se využívá jako výchozí hodnota pro proměnnou, které ještě nebyla přiřazena reference.

Hodnota `void` se využívá jako návratový typ metody, která nevrací žádnou hodnotu.

2.2. Java bytecode

Bytecode jsou instrukce programovacího jazyka zapsané posloupností bytů srozumitelné Java Virtual Machine (dále JVM), který je zodpovědný za převedení bytecodu do strojového kódu. Java bytecode je obvykle uložen v souboru s příponou `.class`.

Tento přístup má výhodu pro programátora, protože mu například ulehčuje přenositelnost mezi systémy. Soubory `.class` mají jasně definovanou strukturu [5], která zaručuje správné načtení a průběh.

2.2.1. Soubor .class

Každý soubor `.class` obsahuje bytecode jedné třídy, jednoho rozhraní anebo jednoho výčtu. Jestliže zdrojový `.java` soubor obsahoval vnořené třídy, tak se vytvoří `.class` soubory pro každou zvlášť.

Vnořené třídy jsou třídy definované uvnitř jiné třídy. Vnořené třídy se dělí na statické a nestatické. Vnořené třídy jsou součástí třídy, do které jsou vnořené, tato třída je třídou vnější. Statické vnitřní třídy nemají přístup k jiným členům vnější třídy. Nestatické vnořené třídy jsou vnitřní třídy a mají přístup i k `private` atributům vnější třídy.

Anonymní třída je vnitřní třída, která nemá vlastní pojmenování. Anonymní třídy se vytvářejí za pomoci operátoru `new`.

Třída, která je definovaná uvnitř metody, se nazývá lokální třídou.

Za vnější se považuje ta, která se jmenuje stejně jako `.java` soubor a je veřejná nebo je přístupná ze stejného balíku. Názvy vnitřní třídy jsou složeny ze jména vnější třídy a znaku `´$´` a jména vnitřní třídy. Lokální třída má v názvu vnější třídu, znak `´$´`, generovaný index a jméno lokální třídy. Anonymní třída má v názvu vnější třídu, znak `´$´` a generovaný index.

Struktura

Základní struktura bytů uložených v .class souboru je uvedená ve specifikaci [5] a na její reprezentaci je využito značení z jazyka C:

```
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];
}
```

Typy položek jsou:

- `u4` – bezznaménková hodnota o délce 4 bytů
- `u2` – bezznaménková hodnota o délce 2 bytů
- `u1` – bezznaménková hodnota o délce jednoho bytu
- `Cp_info, field_info, method_info, attribute_info` – budou popsány dále v části, ke které patří

Položky kontrolované JVM

Následujíc položky `magic`, `minor_version` a `major_version` jsou načítány pro kontrolu JVM.

Magic

Pokud se jedná o `.class` soubor je hodnota `0xCAFEBABE`, při načítání provádí JVM kontrolu, zda je daná hodnota nastavená.

Minor _version a major_version

Hodnoty slouží k identifikaci verze bytecodu, JVM podle ní ověřuje kompatibilitu. Příslušná čísla `minorů` a `majorů` odpovídají jednotlivým verzím, lze je dohledat na webových stránkách java.sun.com. Současné verzi JVM 7.0 odpovídá hodnota 50.0., kde 50 je major verze a 0 je minor verze.

Položky uchovávající názvy a typy dat

Tato část bude popisovat vytvoření struktury, přes kterou jsou dostupné informace o členech třídy.

Constant_pool_count

Obsahuje hodnotu o jedna větší než je počet položek v `cp_info constant_pool[]`.

Cp_info constant_pool[constant_pool_count-1];

Metoda slouží k načtení konstant do vyhledávací tabulky. Tato tabulka je nejdůležitější v celém `.class` souboru, protože jsou v ní uloženy všechny údaje o datových typech a referencích. V bytecodu se dále používají jen indexy do této tabulky.

Cp_info

`Cp_info` se skládá z:

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

Tag

Tag udává jakého typu je `info[]`.

Info

Tabulka 1ukazuje závislost `infa` na `tagu`, podle vybraného typu se mění délka `infa`.

Tabulka 1

Tag	Info
1	`CONSTANT_Utf8`
3	`CONSTANT_Integer`
4	`CONSTANT_Float`
5	`CONSTANT_Long`
6	`CONSTANT_Double`
7	`CONSTANT_Class`
8	`CONSTANT_String`
9	`CONSTANT_Fieldref`
10	`CONSTANT_Methodref`
11	`CONSTANT_InterfaceMethodref`
12	`CONSTANT_NameAndType`

CONSTANT_Utf8_info

Struktura `CONSTANT_Utf8_infa` je:

```
CONSTANT_Utf8_info {
      u1 tag;
      u2 length;
      u1 bytes[length];
}
```

lenght

Udává délku sekvence bajtů.

bytes[lenght]

Zakódovaný řetězec znaků pomocí Utf8. Takto jsou zakódovány všechny názvy a `Stringy`.

CONSTANT_Integer, CONSTANT_Utf8_Float, CONSTANT_Long a CONSTANT_Double

U `integeru` a `floatu` následuje za `tagem` čtyřbytová hodnota daného typu. U `longu` a `doublu` následují dvě čtyřbytové hodnoty, `byty` s vyšší hodnotou jsou první.

CONSTANT_Class

Za `tagem` následuje dvoubytová hodnota, ukazující do tabulky `constant_pool` na `Utf8_info`, kde se nachází celé jméno třídy.

CONSTANT_String

Za `tagem` následuje dvoubytová hodnota, ukazující do tabulky `constant_pool` na `Utf8_info`, kde je uložena hodnota Stringu.

CONSTANT_Fieldref_info, CONSTANT_Methodref_info a CONSTANT_InterfaceMethodref_info

Za `tagem` následuje dvoubytová hodnota, která odkazuje do tabulky `constant_pool` na hodnotu `CONSTANT_Class` další je dvoubytová hodnota, odkazující do tabulky na `CONSTANT_NameAndType`.

CONSTANT_NameAndType

Používá se pro uchování jmen a datových typů.

Struktura `CONSTANT_NameAndType` je:

```
CONSTANT_NameAndType_info {
     u1 tag;
     u2 name_index;
     u2 descriptor_index;
}
```

name_index

Dvoubytová hodnota odkazující na `CONSTANT_Utf8_info` v tabulce, kde se nalézá plné jméno.

deskriptor_index

Odkaz do `constant_pool` tabulky na `CONSTANT_Utf8_info`, v něm jsou uloženy hodnoty na další odkazování.

Přístupová práva ke třídě

Tato část definuje možnost přístupu ke třídě.

access_flags

Výsledná hodnota přístupu vzniká sečtením hodnot příznaků z Tabulka 2.

Tabulka 2

Název	Hodnota	Popis
`ACC_PUBLIC`	`0x0001`	Přístupná i mimo balík.
`ACC_FINAL`	`0x0010`	Bez podtříd.
`ACC_SUPER`	`0x0020`	Lze používat vlastnosti super class.
`ACC_INTERFACE`	`0x0200`	Není třída, je rozhraní.
`ACC_ABSTRACT`	`0x0400`	Abstraktní třída.

Položky související s objektově orientovaným programováním

Následující položky obsahují jméno současné třídy, jméno rodičovské třídy a jména rozhraních, které daná třída implementuje.

this_class
Dvoubytová hodnota odkazující do tabulky `constant_pool` na `CONSTANT_Utf8_info`, kde se nalézá plné jméno třídy, které bytecode patří.

super_class
Dvoubytová hodnota odkazující do tabulky `constant_pool` na `CONSTANT_Utf8_info`, kde se nalézá plné jméno třídy, která je nadřazená `this_class`. V případě, že třída nemá `super_class`, je hodnota rovna 0.

interface_count
Počet rozhraní, které daná třída implementuje.

interface[interface_count]
Obsahuje odkazy na `CONSTANT_Class` v tabulce `constant_pool`. Tyto třídy (rozhraní) jsou implementovány třídou, které bytecode patří.

Položky definující fieldy

Následující položky definují všechny fieldy obsažené ve třídě. U všech fieldů jsou známá jejich jména, přístupová práva a jejich atributy.

field_count
Obsahuje počet fieldů, které jsou deklarované v kódu.

field[field_count]
Definuje deklarované fieldy ve třídě.

Struktura je:

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

access_flags
Dvoubytová hodnota přístupu k položce je výsledkem součtu hodnot z Tabulka 3.

Tabulka 3

Název	Hodnota	Popis
ACC_PUBLIC	0x0001	Lze přistupovat i z jiných balíčků.
ACC_PRIVATE	0x0002	Přístupná jen z dané třídy.
ACC_PROTECTED	0x0004	Přístupná jen z daného balíčku.
ACC_STATIC	0x0008	Statická.
ACC_FINAL	0x0010	Nelze měnit.
ACC_VOLATILE	0x0040	Nelze cachovat.
ACC_TRANSIENT	0x0080	Přechodný.

Hodnoty `public`, `private` a `protected` se navzájem vylučují. Dále se vylučují `volatile` a `final`. Hodnota může být i 0, když nic nebylo nastaveno. Rozhraní musí mít všechny metody `public` a `abstract`.

name_index

Dvoubytová hodnota odkazující na `CONSTANT_Utf8_info` v tabulce, kde se nalézá plné jméno.

descriptor_index

Výsledný index se skládá z posloupnosti znaků z Tabulka 4:

Tabulka 4

BaseType	Type	Popis
B	byte	znamínkový byte
C	char	Unicode znak
D	double	double
F	float	float
I	int	integer
J	long	long
L<classname>;	reference	instance třídy
S	short	znamínkový short
Z	boolean	true nebo false
[	reference	jednodimenzionální pole

attributes_count

Attributes_count udává počet atributů v poli.

attribute_info attributes[attributes_count]

Atribut obsahuje daný počet atributů ve struktuře dané atributy.

Položky definující metody

Následující položky definují všechny metody obsažené ve třídě. U všech metod jsou známá jejich jména, přístupová práva, návratové typ, parametry a jejich atributy.

method_count

Udává počet metod uložených ve struktuře `method_info`.

method_info methods[method_count]

Obsahuje popis metod deklarovaných v dané třídě.

Struktura je:

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

acces_flags

Hodnota vzniká součtem hodnot příznaků z Tabulka 5.

Tabulka 5

Název	Hodnota	Popis
ACC_PUBLIC	0x0001	Lze přistupovat i z jiných balíčků.
ACC_PRIVATE	0x0002	Přístupná jen z dané třídy.
ACC_PROTECTED	0x0004	Přístupná jen z daného balíčku.
ACC_STATIC	0x0008	Statická.
ACC_FINAL	0x0010	Nelze přepsat.
ACC_SYNCHRONIZED	0x0020	Použití v monitorech.
ACC_NATIVE	0x0100	Implementovaná jinak než Javou.
ACC_ABSTRACT	0x0400	Není implementovaná
ACC_STRICT	0x0800	Floating point mod je FP-strict.

Při nenastavení příznaku před překladem je hodnota 0. Hodnoty `public`, `protected` a `private` se vylučují. `Abstract` vylučuje `final`, `native`, `private`, `static`, `strict` a `synchronized`. Rozhraní musí mít všechny metody `public` a `abstract`. `Acces_flags` určuje chování a přístupy k metodám v JVM.

name_index

Dvoubytová hodnota odkazující na `CONSTANT_Utf8_info` v `constant_pool`, kde se nalézá plné jméno metody.

methodDescriptor
Obsahuje dvoubytový odkaz na `CONSTANT_Utf8_info` v `constant_pool`, který je ve tvaru `(ParameterDescriptor*) ReturnDescriptor`. `ParametrDescriptory` jsou typu `FieldType`, `ReturnDescriptor` je typu `FieldType` nebo `V`, které značí `void`.

atribute_count
Počet parametrů v metodě.

attribute_info attributes[attributes_count]
Obsahuje daný počet atributů ve struktuře dané metody.

Položky definující atributy třídy

Tyto položky popisují atributy třídy.

attribute_count
Počet atributů v dané třídě.

attribute_info attributes[attributes_count];
Struktura uchovává informace o atributech třídy.

Struktura je:

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

attribute_name_index
Dvoubytová hodnota ukazující do `constant_pool` tabulky, kde se nachází `CONSTANT_Utf8_info` obsahující název.

atribute_length
Délka následujícího `infa`.

info[attribute_length]
Předdefinované typy `SourceFile`, `ConstantValue`, `Code`, `Exceptions`, `InnerClasses`, `Synthetic`, `LineNumberTable`, `LocalVariableTable` a `Deprecated`.

Anotace

Anotace lze naleznout ve specifikaci [5] v kapitolách 4.7.16 až 4.7.20, zde anotace nejsou uvažovány, protože ve výsledném projektu nejsou v současné době používány.

Generiky

Generiky jsou popisovány přes atribut Signature.

Tvar generického fieldu `public List<String> list;` bude:

```
Signature Ljava/util/List<Ljava/lang/String;>;
```

Tvar generické třídy `public class Trida<T extends Trida2>` bude:

```
Signature
<T:Lcz/zcu/kiv/jacc/loader/impl/bytecode/raw/Trida2;>Ljava/
lang/Object;
```

3. Java Class Compatibility checker

Java Class Compatibility checker (`JaCC`) je knihovna, která extrahuje reprezentaci jazyku Java z bytecodu (`.class`, `.jar`) nebo z reflexe. Pro uložení načtené třídy je využita struktura `JavaTypes`. Po načtení a uložení tříd se kontroluje komptabilita typů a na základě vzájemného porovnávání se detekují rozdíly mezi reprezentacemi tříd. Více o způsobech využití `JaCC` v [6].

3.1. JavaTypes

`JavaTypes` je objektový model, který reprezentuje načtené objekty pro porovnávání v `JaCC`. Názvy rozhraní vycházející z názvů v balíku `java.lang.reflect`. V této práci popíši jen rozhraní a implementace tříd, které se používají v současné implementaci `RawByteCodeLoaderu`. `RawByteCodeLoader` je loader, který má být výsledkem této práce.

3.1.1. Rozhraní JavaTypes

Tato rozhraní se nacházejí v balíku `cz.zcu.kiv.jacc.javatypes`. Tato rozhraní jsou základní kostrou `JavaTypes` a definují základní tvar všech objektů.

JType

`JType` je rozhraní, které definuje datový typ, který je podobný parametru `Type` v Javě.

Definuje veřejné metody:

- `String getName()` – vrací jméno datového typu
- `JPackage getPackage()` – vrací JPackage, do kterého datový typ patří

JParameterizedType

Rozhraní rozšiřuje `JType`, používá se pro práci s generickými parametry. `JParameterizedType` má atributy `ownerType, rawType a actualType.`

Vysvětlení těchto parametrů je jasné z příkladu:

```
class Cl implements List<String>
Cl – ownerType
List – rawType
String - actualType
```

Definuje veřejné metody:

- `List<JType> getActualTypeArguments()` – vrací seznam `JTypů` použitý jako aktuální generický typ
- `@Deprecated void setActualTypeArguments(List<JType> args)` – umožňuje vložit seznam JTypů jako aktuální generický typ
- `JType getOwnerType()` – vrací `JType`, který implementuje generickou třídu
- `JType getRawType()` – vrací `JType`, který má udává rawType
- `@Deprecated void setRawType(JType raw)` – umožňuje nastavit `rawType` pomocí `JTypu`

JAnnotation

Rozhraní definuje anotace používané v jazyce Java™. Anotace je definovaná jménem a množinou parametrů a jejich hodnot, které lze dané anotaci přiřadit.

Definuje veřejné metody:

- `String getName()` – vrací jméno anotace
- `Map<String, ?> getParameters()` – vrací mapu parametrů anotace, kde klíčem je `String` s názvem a hodnotou je datový typ.

JAnnotatedElement

Rozhraní definuje přístup k anotacím použitým na objektu.

Definuje veřejné metody:

- `List<JAnnotation> getAnnotations()` – vrací seznam anotací nad objektem
- `List<JAnnotation> getDeclaredAnnotations()` – vrací seznam deklarovaných anotací

JModifier

Rozhraní definuje práva přístupu k objektu a jeho vlastnosti. Možné vlastnosti objektu jsou zřejmé z následujících veřejných `boolean` metod. Hodnota přístupu je v podstatě daná jednou `int` hodnotou, ze které lze bitovými posuvy zjistit aktuální nastavení.

Definuje veřejné metody:

- `int getModifiers()` – vrací `int`, který svojí hodnotou reprezentuje daný přístup k objektu
- `boolean isAbstract()` – vrací `true`, když je objekt abstraktní, jinak je navrácena hodnota `false`
- `boolean isFinal()` – vrací hodnotu `true`, když objektu nelze vytvořit podtřídu, jinak je vrácena hodnota `false`
- `boolean isInterface()` – je vrácena hodnota `true`, jestliže se jedna o rozhraní, jinak je vrácena hodnota `false`
- `boolean isNative()` – vrací hodnotu `true`, jestliže je objekt nativní, jinak vrací hodnotu `false`
- `boolean isPrivate()` – vrací hodnotu `true`, jestliže je objekt mimo třídu nedostupný, jinak vrací `false`.
- `boolean isProtected()` – vrací hodnotu `true`, když je objekt dostupný jen v jeho balíku, když není, vrací `false`
- `boolean isPublic()` – vrací hodnotu `true`, když je objekt veřejně přístupný, jinak vrací hodnotu `false`
- `boolean isStatic()` – vrací hodnotu `true`, když je objekt statický, jinak vrací hodnotu `false`
- `boolean isStrict()` – vrací hodnotu `true`, když je `floating point` typu `FP-strict`, jinak vrací `false`
- `boolean isSynchronized()` – vrací hodnotu `true`, jestliže je přístup k objektu synchronizovaný, jinak vrací `false`
- `boolean isTransient()` – vrací hodnotu `true`, když se jedná o přechodný objekt, jinak vrací hodnotu `false`
- `boolean isVolatile()` – vrací hodnotu `true`, když objekt nelze cachovat jinak vrací hodnotu `false`

JAccessibleObject

Rozhraní rozšiřuje rozhraní `JAnnotatedElement` a definuje jakým způsobem lze přistupovat k objektu. Přístup lze zjistit pomocí `JModifier` parametru.

Definuje veřejné metody:

- `JModifier getModifiers()` – vrací přístup k objektu v podobě `JModifier`

JGenericDeclaration

Rozhraní definuje generickou deklaraci.

Definuje veřejné metody:

- `List<JTypeVariable<? extends JGenericDeclaration>> getTypeParameters()` – Vrací seznam objektů `JTypeVariable`

JTypeVariable

Rozhraní rozšiřuje `JType` a definuje tvar generických typových proměnných a hodnot, které mohou nabývat

Definuje veřejné metody:

- `List<JType> getBounds()` – vrací seznam `JType`, který odpovídá `Bounds`
- `D getGenericDeclaration()` – vrací `JGenericDeclaration`, která daný `JTypeVariable` deklaruje

JMember

Rozhraní rozšiřuje `JAccessibleObject` o informaci, která třída daný prvek deklaruje a jak se daný prvek jmenuje.

Definuje veřejné metody:

- `JClass getDeclaringClass()` – vrací třídu, která daný objekt deklaruje
- `String getName()` - vrací název objektu

JClass

Rozšiřuje `JType`, `JAccessibleObject`, `JGenericDeclaration` a přidává možnosti pro práci s metodami, položkami, konstruktory, které daná třída deklaruje a definuje dodatečné informace o třídě.

Definuje veřejné metody:

- `String getShortName()` – vrací název třídy, bez části označující balík, ze které daná třída je
- `JClass getSuperclass()` – vrací `JClass`, ve které je načtená třída, které je současná třída potomek
- `List<JMethod> getMethods()` – vrací seznam `JMethod`, které deklarují předci a které deklaruje daná třída
- `List<JMethod> getDeclaredMethods()`- vrací seznam `JMethod`, které daná třída deklaruje
- `List<JField> getFields()` – vrací seznam veřejných položek, které obsahují třídy předků a veřejné položky, které deklaruje současná `JClass`

- `List<JField> getDeclaredFields()`- vrací seznam veřejných položek, které daná třída deklaruje
- `List<JConstructor> getConstructors()` – vrací seznam `JConstructorů`, kterou lze danou třídu inicializovat
- `boolean isArray()` – vrací `true`, když je třída polem, když není, vrací `false`
- `JClass getComponentType()` - pokud je třída pole, vrací `JClass` typ daného pole, jinak vrací `null`
- `boolean isInterface()` – vrací hodnotu `true`, když je třída rozhraním, jinak vrací `false`
- `boolean isPrimitive()` – vrací hodnotu `true`, když je třída jedním z primitivních typů popsaných v části Primitivní datové typy, pokud není je vrácena hodnota `false`
- `boolean isEnum()` – vrací hodnotu `true` když je třída výčtem, pokud není, vrací `false`
- `@Deprecated JClassLoader getClassLoader()` – vrací `JClassLoader`, v současné době doporučeno nepoužívat, zůstává kvůli komptabilitě

JMethod

Rozhraní rozšiřuje `JMember` a `JGenericDeclaration` a definuje přístup k metodám. Metoda se v Javě skládá z návratového typu, jména, z 0 až N parametrů. Metoda může mít definované výjimky, které může vyhazovat, a můžou nad ní být definované anotace.

Definuje veřejné metody:

- `JType getReturnType()` – vrací `JType`, který popisuje návratový typ metody
- `List<JType> getParameterTypes()` – vrací seznam `JTypů`, které popisují typy parametrů metody
- `List<JType> getExceptionTypes()` – vrací seznam `JTypů`, které popisují výjimky, které může metoda vyhodit
- `boolean equals(Object arg0)` – vrací hodnotu `true`, když je metoda zadaná jako `Object arg0` stejná jako metoda, které třída patří, jinak vrací hodnotu `false`

JConstructor

Rozhraní rozšiřuje `JMethod` a definuje přístup ke konstruktorům tříd.

Definuje veřejné metody:

- `List<JType> getParameterTypes()` – vrací seznam `JTypů`, které popisují typy parametrů metody
- `List<JType> getExceptionTypes()` – vrací seznam `JTypů`, které popisují výjimky, které může metoda vyhodit

JField

Rozhraní rozšiřuje `JMember` a definuje přístup k fieldům ve třídě. Příklad fieldu je například `int number`.

Definuje veřejné metody:

- `JType getType()` – vrací `JType`, který udává typ fieldu
- `boolean isEnumConstant()` – vrací `true`, když je položka součástí výčtu, jinak vrací `false`
- `@Override String toString()` – přepisuje metodu `toString()` na novou podobu

JPackage

Rozhraní definující přístup k balíkům v Javě.

Definuje veřejné metody:

- `String getName()` – vrací název balíku
- `List<JClass> getJClasses()` – vrací seznam všech veřejných `JClass`, které daný balík obsahuje

3.1.2. Standardní implementace rozhraní JavaTypes

Implementovaná rozhraní lze naleznout v balíku `cz.zcu.kiv.jacc.javatypes.impl`. Nejsou zde popsány celé zavislosti dědičnosti, ale jen části související s implementací mého loaderu.

JAnnotationImpl

Třída implementuje `JAnnotation`. Lze přes ni informace o anotaci dostat i nastavit jiné hodnoty.

Konstruktor:

- `public JAnnotationImpl(String name, Map<String, ?> parameters)` – umožňuje vytvořit anotaci se jménem a s parametry

Přidané veřejné metody:

- `public void setParameters(Map<String, ?> parameters)` – přidává možnost nastavit pomocí mapy parametry anotace
- `@Override public String toString()` – přepisuje metodu `toString()`

JClassImpl

Třída odvozená od `AbstractJClass`, pomocí této třídy lze nastavit všechny důležité parametry třídy.

Konstruktory:

- `public JClassImpl()`
- `public JClassImpl(JModifier modifiers, String name)` – nastavení s přístupem a se jménem
- `public JClassImpl(JModifier modifiers, String name, JPackage pkg, JClass superclass)` – inicializace s přístupem, se jménem, balíkem a s předkem

Konstruktory volají metodu `super()` se svými parametry.

Přidané veřejné metody:

- `public static JClassImpl createArrayRepresentation (JModifier modifiers, String name, JClass componentType)` – vytváří `JClass`, která reprezentuje třídu reprezentující pole, `JClass componentType` udává jakého typu je dané pole
- `public static JClassImpl createEnumRepresentation (JModifier modifiers, String name, JPackage pkg)` – vytváří `JClass`, která reprezentuje třídu reprezentující výčet
- `public static JClassImpl createClassRepresentation (JModifier modifiers, String name, JPackage pkg, JClass superclass)` – vytváří instanci `JCLassImpl` reprezentující třídu,
- `public static JClassImpl createClassRepresentation (JModifier modifiers,String name, JPackage pkg, JClass superclass,List<JField> DeclaredFields, List<JMethod> declaredMethods, List<JConstructor> constructors)` – vytváří instanci JClassImpl reprezentující třídu, nastavuje `List<JField> DeclaredFields` deklarované položky, `List<JMethod> declaredMethods` deklarované metody a `List<JConstructor> constructors` konstruktory

- `public void setDeclaredFields(List<JField> declaredFields)` – nastavuje třídě seznam položek, které deklaruje
- `public void setDeclaredMethods(List<JMethod> declaredMethods)` – nastavuje seznam deklarovaných metod
- `public void setConstructors(List<JConstructor> constructors)` – nastavuje seznam deklarovaných konstruktorů
- `public void setTypeParameters(List<JTypeVariable<? extends JGenericDeclaration>> typeParameters)` – nastavuje použité generiky v názvu třídy

JClassNotFoundException

Třída rozšiřuje třídu `Exception` z balíku `java.lang.Exception`, tato výjimka je vyvolána, když je třída nenalezena.

Konstruktory:

- `public JClassNotFoundException()` – konstruktor pouze volá metodu `super()`
- `public JClassNotFoundException(String message, Throwable cause)` – nastavuje příčinu a zprávu výjimky
- `public JClassNotFoundException(String message)` – nastavuje zprávu o výjimce
- `public JClassNotFoundException(Throwable cause)` – nastavuje příčinu výjimky
- `public String toString()` – přepisuje metodu `toString()`

JConstructorImpl

Třída rozšiřuje `AbstractJMethod` a implementuje `JConstructor`, používá se pro reprezentaci konstruktoru dané třídy.

Konstruktory:

Všechny konstruktory volají metodu `super()` se svými parametry.

- `public JConstructorImpl()` – konstruktor bez parametrů
- `@Deprecated public JConstructorImpl(JModifier modifiers, String name, JClass declaringClass, List<JType> exceptionTypes, List<JType> parameterTypes, JType returnType)` – konstruktor definovaný přístupem, jménem, třídou, která daný konstruktor deklaruje, seznamem `<JType>` parametrů a návratovým typem

- `public JConstructorImpl(JModifier modifiers, String name, JClass declaringClass)` – konstruktor definovaný přístupem, jménem a deklarující třídou

Přidané veřejné metody:

- `@Override public String toString()` – přepisuje metodu `toString()`

JFieldImpl

Třída rozšiřuje `AbstractJMember` a implementuje `JField`, používá se pro reprezentaci položky.

Konstruktory:

- `public JFieldImpl(JModifier modifiers, String name, JClass declaringClass, JType type)` – nastavuje přístup, jméno, deklarující třídu a typ položky
- `public JFieldImpl(JModifier modifiers, String name, JClass declaringClass, JType type, boolean isEnumConstant)` – nastavuje přístup, jméno, deklarující třídu a typ fieldu a zda je field součástí výčtu
- `public JFieldImpl()` – konstruktor bez parametrů

Přidané veřejné metody:

- `public void setType(JType type)` – nastaví typ položky pomocí `JType` typu
- `public void setEnumConstant(boolean isEnumConstant)` – nastaví, zda je položka částí výčtu
- `@Override public String toString()` – přepíše metodu `toString()`

JMethodImpl

Třída rozvíjí `AbstractJMethod` a implementuje `JMethod`, používá se pro metodu.

Konstruktory:

- `public JMethodImpl()` – prázdný konstruktor, zavolá pouze `super()`
- `public JMethodImpl(final JModifier modifiers, final String name, final JClass declaringClass, final List<JType> exceptionTypes, final List<JType> parameterTypes, final JType returnType)` – nastaví metodu

pomocí přístupových práv, jména, deklarující třídy, seznamu výjimek, seznamu parametrů metody a návratového typu

- `public JMethodImpl(final JModifier modifiers, final String name, final JClass declaringClass)` – nastaví metodu pomocí přístupových práv, jména a deklarující třídy

Přidané veřejné metody:

- `public void setReturnType(final JType returnType)` – metoda umožňující nastavit metodě návratový typ pomocí `JType returnType`

JModifierImpl

Třída implementuje `JModifier`, nastavuje přístupová práva k objektu.

Konstruktory:

- `public JModifierImpl(int modifiers)`- má jako parametr `int modifiers`, který nastaví přístupová práva k objektu
- `public JModifierImpl()`- konstruktor bez parametru

Přidané veřejné metody:

- `public void setModifiers(int modifiers)` – pomocí `int modifiers` nastavuje přístup k objektu
- `public boolean equals(Object obj)` – přepisuje metodu `equals()`
- `public int hashCode()` – přepisuje metodu `hashCode()`
- `public static String toString(int mod)` - slouží k výpisu přístupu vyjádřeného `int modem`
- `public String toString()`- přepisuje metodu `toString()`

JTypeVariableImpl

Třída implementuje rozhraní `JTypeVariable`, slouží k reprezentaci generik.

Konstruktory:

- `public JTypeVariableImpl(String name, JGenericDeclaration genericDeclaration)` – inicializace se jménem a generickou deklarací
- `public JTypeVariableImpl(String name, List<JType> bounds, JGenericDeclaration genericDeclaration)` –

nastavuje jméno, seznam `JType bounds` udávající typ generiky, generickou deklaraci

Přidané veřejné metody:

- `public final void setBounds(List<JType> bounds)` – nastaví seznam `JType generik`
- `public static String toString()` – přepisuje metodu `toString()`
- `public void setPackage(JPackage jPackage)` – pomocí `JPackage jPackage` nastavuje balík, do kterého `JTypeVariable` patří

3.2. Rozhraní loaderu

Loader bude sloužit k načítání informací z bytecodu do datového modelu JavaTypes. Rozhraní definuje přístupy k načtenému obsahu, neomezuje se na jeden způsob načítání, měl by umožnit stejné výstupy pro data načtená přes reflexi i bytecode. Rozhraní loaderu se nachází v balíku `cz.zcu.kiv.jacc.loader`.

Definované veřejné metody jsou:

- `Set<String> getPackageNames()` – vrací množinu `Stringů`, která obsahuje jména všech balíků
- `Set<String> getImportedPackageNames()` – vrací množinu `Stringů`, která obsahuje jména všech importovaných balíků
- `Set<JClass> getClasses(String pcg)` – vrací množinu `JClass`, které obsahuje balík s názvem `String pcg`
- `JClass getClass(String className)` – načítá konkrétní `JClass` podle jména `String className`
- `Set<JClass> getImportedClasses(String className)` – vrací množinu `JClass`, která jsou importované třídou, která je definovaná pomocí `String name`

3.3. Komparátor

Komparátor slouží k porovnávání reprezentací tříd. Přes načtené parametry se zkoumají souvislosti mezi třídami a podle toho se vyhodnocuje, jaké mají k sobě dané třídy vztahy. Objekt může být přidán nebo odebrán, může mít přidané nebo odebrané vlastnosti anebo nemusí být mezi objekty žádná souvislost. Informace o komparátoru jsou zde uvedeny jen pro komplexnost práce, samotný komparátor nebyl součástí mé práce.

4. Načítání Java tříd z bytecode

Pro načtení dat z bytecodu jsem implementoval vlastní loader. Loader slouží k načtení struktury třídy z bytecodu uloženého v `.class` souboru nebo `.jar` souboru do datového modelu `JavaTypes`. Vzhledem k použití přímého čtení bytecode struktur se loader jmenuje RawByteCodeClassLoader.

`RawByteCodeClassLoader` je rozdělen do tří tříd, které mají svojí implementaci umístěnou v balíku `cz.zcu.kiv.jacc.loader.impl.bytecode.raw`. Pro správné načítání dat z bytecodu do `JavaTypes` musí uživatel vědět, kde na disku se daný bytecode nalézá, nebo ve kterém `.jar` souboru je bytecode archivován.

4.1. RawByteCodeClassDumper

RawByteCodeClassDumper je nejnižší úrovní loaderu. Tato třída obsluhuje čtení bytecodu. Po zavolání konstruktoru dojde k automatickému načtení celého bytecodu. Jediným vstupním parametrem pro celou třídu je soubor `.class`, který je zadaný i s cestou.

`RawByteCodeClassDumper` používá pro zpracování konstant odkazujících do `constant_poolu` vnitřní třídu `RawBytecodeAssoc`. Tato třída má jediný úkol, podle druhu `tagu`, dát jeden nebo dva indexy do tabulky `pool`, která je definovaná jako `constant_pool` v kapitole 2.2.1. Pojem tabulka `pool` bude hodně využíván, protože struktura `.class` souboru je postavena na indexech do této tabulky.

4.1.1. Inicializace

Konstruktory slouží k nastavení cesty k danému `.class` souboru.

- `public RawByteCodeClassDumper(String path) throws Exception` – jediným parametrem je `String path`, který udává místo uložení `.class` souboru na disku, ze kterého vytváří `InputStream`.
- `public RawByteCodeClassDumper(InputStream in) throws Exception` – jediným parametrem je `InputStream in`, který bude dále zpracováván.

Konstruktory volají privátní metodu `dump()`, která zajišťuje další čtení.

4.1.2. Načítaní dat

Načítání probíhá podle stejného schématu, protože struktura `.class` souboru je jasně definovaná.

private void dump() throws Exception

Vytváří `DataInputStream din` z `InputStream in`, potom volá `parseClassFile(din)`, nakonec uzavírá `DataInputStream` pomocí

din.close(). Metoda vyhazuje výjimku, pokud se nepovedl vytvořit DataInputStream a nebo došlo k chybě při čtení.

private void parseClassFile(DataInputStream in) throws IOException,Exception

Vlastní načtení bytecodu je prováděno následující posloupností operací, které mají ze struktury jasně definovaný tvar.

Kontrola vstupu

Kontroluje pomocí int magic, zda se jedná o .class soubor. V případě, že se o class soubor nejedná, vyhazuje výjimku IOException("Not a valid class file (no CAFEBABE header)").

Tvorba Constant_poolu

Metoda vytváří tabulku Object pool[]. Tuto tabulku naplní hodnotami, které jsou popsány v kapitole 2.2.1. V případě, že je vložen tag, který není definován je vyhozena výjimka IllegalArgumentException("Unknown tag: " + tag).

Informace o třídě

Metoda načte hodnotu unsigned short classAccess, která udává přístup ke třídě.

Načítá unsigned short hodnoty, které ukazují na místa v poolu, kde je uloženo plné jméno třídy a plné jméno předka. Hodnoty z poolu na daných indexech uloží jako String this_class a String superclass.

Načítá unsigned short hodnotu, která říká, kolik rozhraní daná třída implementuje. Potom pro každé načte unsigned short odkaz do poolu, kde je název daného rozhraní. Tato rozhraní jsou uložena do List<String> implementedInterfaces.

Načtení fieldů

Metoda načte unsigned short hodnotu, která udává, kolik fieldů daná třída obsahuje. Pro každý field načte přístupnost položky int fieldAccess, int name_index, který odkazuje do poolu na jméno dané položky.

Jméno fieldu je uloženo ve Stringu fieldName. Dále načte int descriptor_index, který odkazuje do poolu na popisek dané položky. Pak se načte int descriptor_index, podle něj se v poolu vyhledá String fieldDescriptor.

Poté se rozhodne podle fieldAccess, zde je položka veřejná, když je, tak se zkontroluje, zda je přítomná signatura. Test na signaturu se provádí z důvodu, že položka může být generikou. V případě, že je signatura přítomná dojde k přepsání fieldDescriptoru signaturou.

Výsledné Stringy mají tvar fieldAccess + " " + fieldName + " " + fieldDescriptor + atribut. Generiky jsou přidány do seznamu publicGenerics, když jsou veřejné nebo do seznamu otherGenerics. Negenerické položky jsou přidány do seznamu publicFields nebo do seznamu otherFields.

Příklad výsledného tvaru:

public int number – bude přidán do publicFields ve tvaru 1 number I

public List<String> list – bude přidán do publicGenerics ve tvaru 1 list Ljava/util/List<Ljava/lang/String;>; Signature Ljava/util/List<Ljava/lang/String;>;

Načtení metod

Metoda načte int method_count, který udává, kolik metod daná třída deklaruje. Pro každou metodu se načte int access_flags, udávající přístup k metodě, int name_index, odkazující do poolu na jméno, int descriptor_index odkazující na popis třídy.

Po načtení těchto základních vlastností se volá metoda doAttributes(in), jejíž výstup je uložen do Stringu attributes.

Dále se porovnává, zda je jméno <init>, když je, je metoda konstruktorem a do seznamu constructors je vložen řetězec access_flags + " " + methodName + " " + methodDescriptor + attributes.

Příklad ukazující tvar načteného konstruktoru:

Konstruktor public Trida(List<String> list, int number) bude do seznamu vložen jako 1 <init> (Ljava/util/List;I)V Signature (Ljava/util/List<Ljava/lang/String;>;I)V LineNumberTable 2 LocalVariableTable 3 this Lcz/zcu/kiv/jacc/loader/impl/bytecode/raw/Trida; list Ljava/util/List; number I.

V případě, že metoda není konstruktorem, rozhoduje se, zda je veřejná nebo ne. Podle rozhodnutí je řetězec access_flags + " " + methodName + " " + methodDescriptor + attributes vložen buď do seznamu publicMethods nebo do otherMethods.

Příklad ukazující tvar načtené metody:

Metoda public int get(List<String> list, int number) bude do seznamu publicMethods vložena jako 1 get (Ljava/util/List;I)I Signature (Ljava/util/List<Ljava/lang/String;>;I)I

```
LineNumberTable 1 LocalVariableTable 3 this
Lcz/zcu/kiv/jacc/loader/impl/bytecode/raw/Trida; list
Ljava/util/List; number I
```

Načtení atributů třídy

Čtení se provádí pomocí metody `doAttributes(in)`, která načte rozšiřující atributy třídy do `classAttributes`. Jestliže za atributy následují ještě nějaké byty, vypíše se `zpráva („Extra byte follow…")`.

private String doAttributes(DataInputStream in) throws IOException

Metoda načte počet atributů a pak pro každý zavolá `private String doAttribute (DataInputStream in)throws IOException`. Výsledkem je `String`, který obsahuje všechny načtené parametry.

private String doAttribute(DataInputStream in) throws IOException

Metoda načítá index do poolu, kde se nalézá jméno daného atributu a velikost atributu. Po té se podle jména provede načtení daného atributu.

Používané atributy:

- `RuntimeVisibleAnnotations` – volá metodu `doAnnotations(in)`, která načítá anotace viditelné za běhu programu
- `RuntimeInvisibleAnnotations` – volá metodu `doAnnotations(in)`, která načítá anotace neviditelné za běhu
- `SourceFile` - volá metodu `doSourceFile(in)`, která načte zdrojové místo
- `Code` - pomocí metody `doCode(in)` načte byty, které vyjadřují kód, který je vykonáván
- `LineNumberTable` – obsahuje proměnné pro kód z `Code`, vytváří se pomocí `doLineNumberTable(in)`
- `LocalVariableTable` – obsahuje lokální proměnné pro metody, sestavuje jí `doLocalVariableTable(in)`
- `InnerClasses` – vnitřní třídy načtené pomocí `doInnerClasses(in)`
- `Exceptions` – výjimky se načítají pomocí `doExceptions(in)`
- `EnclosingMethod` – vyskytuje se jen v lokálních a v anonymních třídách načítá se pomocí `doEnclosingMethod(in)`
- `Signature` – důležitý atribut pro zjišťování generik, protože obsahuje plný tvar, načítá se pomocí `doSignature(in)`
- `Synthetic` – atribut je v současné době přeskakován
- `Deprecated` – patří mezi `RuntimeVisibleAnnotations`, mezi atributy je nyní duplicitní, proto se nic neprovádí

- `Override` – patří mezi `RuntimeVisibleAnnotations`, mezi atributy je nyní duplicitní, proto se nic neprovádí
- Když atribut nepatří mezi výše zmíněné atributy, tak se načte délka atributu a dané byty se přeskočí, když je délka větší než 2Gb, je vyhozena výjimka `IllegalArgumentException("Attribute > 2Gb")`

Výsledný `String` obsahuje mezerou oddělené názvy atributů následované obsahem atributy.

private String doAnnotations(DataInputStream in) throws IOException

Metoda načte počet anotací a pro každou zavolá metodu `doAnotation(in)`. Vrací „`Annotations `" + počet anotací + výsledky z `doAnotation(in)`.

private String doAnnotation(DataInputStream in) throws IOException

Metoda načte hodnotu, na které je v `poolu` uložen typ anotace, poté se načte kolik párů proměnných a hodnot daná anotace má. Pro každý pár zavolá `doElementValue(in)`. Vrací typ anotace a hodnotu indexu v `private List<String> annotation`, do `private List<String> annotation` ukládá `Stringy` ve tvaru `@počet(@typ@jméno@)`$^{\text{počet}}$, když je `počet` roven 0 vrátí jen „0".

private String doElementValue(DataInputStream in) throws IOException

Metoda načte hodnotu `tagu`, podle `tagu` rozhodne, zda se jedná o primitivní datový typ, třídu, pole, anotaci anebo výčet. V případě, že se jedná o primitivní datový typ, je vráceno jméno datového typu, např. pro `I` je vrácen „`int`". Pro pole se zavolá znovu metoda `doElementValue(in)`. Když je `tagem` anotace zavolá se metoda `doAnnotation(in)`. Pro výčet se vrací datový typ a jeho hodnota. Pro třídu je vráceno `CONSTANT_ClassInfo` z `poolu`. Když má `tag` jinou hodnotu je vyhozena výjimka `IllegalArgumentException("Invalid value for Annotation ElementValue tag " + tag)`.

private String doSourceFile(DataInputStream in) throws IOException

Metoda načte hodnotu, podle které najde v poolu název zdrojového souboru, Vrací „`SourceFile `" + název souboru.

Ukázka výstupu:

```
public class Trida{
}
```

Bude vracet `SourceFile Trida.java`.

private String doCode(DataInputStream in) throws IOException

Metoda načítá kód, vrací atributy kódu zjištěné přes `doCodeAttributes(in)`. Samotný kód není v současné době vracen, protože se obsahy metod, tříd nezpracovávají.

private String doCodeAttributes(DataInputStream in) throws IOException

Metoda funguje stejně jako `doAttributes(in)`, jediný rozdíl je, že nepracuje nad hlavičkami ale nad samotným kódem. Volá metodu `doAttribute(in)`.

private String doLineNumberTable(DataInputStream in) throws IOException

Metoda vrací „`LineNumberTable ‟` + velikost tabulky. Za každý řádek načte dvakrát `in.readUnsignedShort()`.

private String doLocalVariableTable(DataInputStream in) throws IOException

Metoda načte počet prvků v tabulce, pro každý načte začátek příkazů, délku, index na jméno do `poolu` a index na deskriptor do `poolu` a index. Vrací `" LocalVariableTable "` + počet parametrů a pak tolikrát kolik je parametrů „ “ + jméno z `poolu` + „ “ + deskriptor z `poolu`.

Výstup pro `public Trida(List<String> list, int number)` je `„LocalVariableTable 3 this Lcz/zcu/kiv/jacc/loader/impl/bytecode/raw/Trida; list Ljava/util/List; number I‟.`

private String doInnerClasses(DataInputStream in) throws IOException

Metoda načítá hodnotu, která udává, kolik vnitřních tříd obsahuje. Pro každou třídu jsou načteny indexy, na kterých lze v `poolu` nalézt vnitřní a vnější info o třídě. Dále se načítá index, na kterém lze najít jméno třídy. Poté se načte přístup k třídě. Když je vnitřní info nenulové, načte se vnitřní jméno, když je vnější jméno nenulové, načte se vnější jméno.

Příklad výstupu pro:

```
public class Trida{
     private class VnitrniTrida {
     }
}
```

Výstupem bude:

```
InnerClasses 1 2
cz/zcu/kiv/jacc/loader/impl/bytecode/raw/Trida$VnitrniTrida
cz/zcu/kiv/jacc/loader/impl/bytecode/raw/TridaVnitrniTrida.
```

Hodnota 1 udává počet vnitřních tříd, hodnota 2 udává přístupová práva ke třídě.

private String doExceptions(DataInputStream in) throws IOException

Napřed metoda načte počet výjimek, pro každou načítá index pro nalezení názvu výjimky z `poolu`.

Výsledný vrácený tvar pro metodu s `throws IOException, IOError` bude `Exceptions 2 java/io/IOException java/io/IOError.`

private String doSignature(DataInputStream in) throws IOException

Metoda načte index, na kterém je signatura uložena v poolu. Vrací „Signature “ + String uložený na načteném indexu.

4.1.3. Zpracování načtených dat pro výstup

Tyto metody transformují načtená data do formátu, který je lépe zpracovatelný vyššími vrstvami.

private List<String> getFormatedMethods(List<String> methods)

Metoda provádí úpravu načtených dat do srozumitelnější struktury. Odstraňuje počáteční ´L´ u objektů a ´;´ na konci názvu. V případě, že je metoda konstruktor je název z `<init>` změněn na jméno třídy. U anotací přidává do výsledného `Stringu` informaci z `private List<String> annotation`. Nakonec mění všechny ´/´ na ´.´.

private List<String> getFormatedField(List<String> fields)

Metoda upravuje vzhled položek. Přidává informace z `private List<String> annotation.` Nakonec mění všechny ´/´ na ´. ´.

4.1.4. Výstup dat

Zde budou popsány veřejné metody, přes které se z `RawByteCodeClassDumperu` získávají informace o načtené třídě. V současné implementaci nelze získat bytecode samotných metod.

Rozhodl jsem se, že návratovými datovými typy budou primitivní datové typy, a to `String` a `List<String>`. Tyto datové typy byly zvoleny, protože `RawByteCodeClassLoader` bude pouze transformovat data

z `RawByteCodeClassDumperu` do `JavaTypes`, proto není potřeba vytvářet další datovou strukturu.

public int getAccess()

Metoda vrací hodnotu `int`, která říká, jaká přístupová práva k dané třídě jsou nastavena.

public String getClassAttributes()

Metoda vrací `String`, který obsahuje atributy třídy.

public String getClassName()

Metoda vrací `String`, který obsahuje jméno třídy.

public List<String> getConstructors()

Metoda volá pro `private List<String> constructors` metodu `private List<String> getFormatedMethods(List<String> methods`.

Ukázka výstupu:

```
public class Trida

{
     public Trida(int Number, String name) {}
     public Trida(int number){}
     public Trida(double d){}
}
```

Vrátí `List<String>`, který bude mít `String` na indexu 0 „`1 Trida ( int , java.lang.String )`“, na indexu 1 „`1 Trida ( int )`“ „a na indexu 2 “ `1 Trida ( double )`“.

public String getFullClassName()

Metoda vrací plné jméno třídy včetně části udávající balík.

public String getFullSuperClassName()

Metoda vrací plné jméno nadřazené třídy.

public List<String> getImplementedInterfaces()

Metoda vrací seznam rozhraní, které daná třída implementuje.

public String getPackageName()

Metoda vrací jméno balíku, do kterého třída patří.

public List<String> getPublicFields()

Metoda vrací seznam veřejných položek zformátovaných pomocí `getFormatedFields(publicFields)`.

public List<String> getPublicGenerics()

Metoda vrací seznam veřejných generických položek zformátovaných pomocí `getFormatedFields(publicGenerics)`.

public List<String> getPublicMethods()

Metoda vrací seznam veřejných metod zformátovaných pomocí `getFormatedMethods(publicMethods)`.

public String getSuperClassName()

Metoda vrací jméno nadřazené třídy bez části package.

public boolean isEnum()

Metoda vrací, zda je třída výčtem.

public boolean isInterface()

Metoda vrací, zda je třída rozhraním.

public boolean isPrimitive()

Metoda vrací vždy `false`, protože nelze zavolat `RawByteCodeClassDumper` na primitivní třídu.

4.2. RawByteCodeGenericDeclaration

Třída implementuje `JGenericDeclaration`, přes tuto třídu se nastavují `private List<JTypeVariable<? extends JGenericDeclaration>> parameters`.

Veřejné metody:

- `public void setTypeParameters(List<JTypeVariable<? extends JGenericDeclaration>> par)` – metoda nastavuje generické parametry
- `public void setTypeParameters(String signature, RawByteCodeClassLoader rbccl, JClass cl) throws JClassNotFoundException` – metoda nastavuje generické parametry, parametr `signature` obsahuje signaturu třídy, `rbccl` udává, kterým loaderem se má načítat, `cl` je třída, která generické parametry deklaruje

4.3. RawByteCodeClassLoader

Třída implementuje `JClassLoader` z balíku `cz.zcu.kiv.jacc.loader`. Třída zajišťuje převedení dat načtených pomocí `RawByteCodeClassDumperu` do datového modelu `JavaTypes`.

4.3.1. Inicializace

Konstruktory v sobě obsahují cesty na všechny místa uložení bytecodů, které budou potřeba. V případě, že nejsou uvedeny všechny zdrojové cesty, budou generovány prázdné třídy obsahující jen název.

- `public RawByteCodeClassLoader(String[] path) throws IOException` – konstruktor pro více složkových vstupů, kontroluje se, zda zadané cesty jsou složkami, jinak je vyhozena výjimka `IOException`.
- `public RawByteCodeClassLoader(JarFile jar) throws Exception` – vstupem je `.jar` soubor, ze kterého budou dané bytecody čteny
- `public RawByteCodeClassLoader(String path) throws IOException` – konstruktor pro složkový vstup, kontroluje se, zda je zadaná cesta složkou, jinak je vyhozena výjimka `IOException`

4.3.2. Převod dat z RawByteCodeClassDumperu do JavaTypes

Jak již bylo popsáno výše, `RawByteCodeClassDumper` vrací jako výsledek načtených bytecodů pouze primitivní datové typy, `String` a `List<String>`. Tyto výsledky se musí převést do struktury `JavaTypes`. Převedení je provedeno pomocí několika metod.

private JClass loadClass(String className)

Metoda načítá `JClass` podle zadaného plného jména.

Prohledávání již načtených tříd

`RawByteCodeClassLoader` načítá třídu podle plného jména, jestliže již třída byla jednou načtena, tak je uložena v `private HashMap<String, JClass> classes` . Když je zde třída nalezena je rovnou vrácena.

Hledání bytecodu dané třídy

Když nebyla třída mezi načtenými, prochází se cesty, které byly nastaveny a na nich se hledá daný bytecode. Když je nalezen bytecode třídy daného jména a je přečten `RawByteCodeClassDumperem`, ověří se, zda se shoduje balík, do kterého daná třída patří s balíkem hledané třídy.

Nastavení přístupových práv

Pomocí metody `getAccess()` se zjistí hodnota přístupových práv, která se využije v `new JModifierImpl(int hodnota)`. Jméno třídy dostaneme pomocí metody `getClassName()`.

Vytvoření instance JClassImpl

Se získanými údaji se vytvoří instance `JClassImpl(modifikátor, jméno)`. Tato třída je v tuto dobu vytvářena, protože by mohlo dojít k zacyklení, když by třída jako položku obsahovala sama sebe, nebo kdyby třída byla typem parametru v metodě.

Když je vytvořena instance čtené třídy, tak dojde k uložení do `classes`, to zamezí výše zmíněnému zacyklení (třída se již nebude číst `RawByteCodeClassDumperem`, bude pouze předána reference na již načtenou třídu).

Nastavení rodičovské třídy

Z výstupu `getSuperClassName()` zjistíme jméno rodičovské třídy, jestliže rodičem není třída `Object`, zavolá se metoda `getJClassForClass(superClassName)` s parametrem `superClassName`, který se získá z `RawByteCodeClassDumperu` pomocí `getFullSuperClassName()`. Třídě se tato rodičovská třída přidá do seznamu importovaných tříd.

Nastavení balíku

Metoda `getPackageName()` nám dá název balíku, který se vyhledá v `private HashMap<String, JPackageImpl> packages`; v případě, že je balík nalezen, je do něj daná třída přidána, když není nalezen, vytvoří se podle jména nový `JPackageImpl` a do mapy se přidá.

Nastavení importovaných tříd

Třídě je nastaven seznam importovaných tříd pomocí `setImports(List<JClass>)`, v současné době je seznam prázdný nebo obsahuje `JClass superClass`.

Nastavení základních informací o třídě

Poté se třídě nastaví základní vlastnosti pomocí `setEnum(boolean parametr)`, kde se jako `parametr` použije hodnota získána z `RawByteCodeClassDumperu` z metody `isEnum()`, `setInterface(boolean parametr)`, kde se jako `parametr` použije hodnota získána z `RawByteCodeClassDumperu` z metody `isInterface()`, `setPrimitive(boolean parametr)`, kde se jako `parametr` použije hodnota získána z `RawByteCodeClassDumperu` z metody `isPrimitive()`.

Nastavení členů třídy a anotací

V následujících metodách budou použity parametry `RawByteCodeClassDumper rbccd`, který obsahuje načtený bytecode a `JClass c2`, která obsahuje referenci na `JClass`, pro kterou se daná část třídy načítá.

Třídě se nastaví, jaké anotace obsahuje pomocí metody `getDeclaredAnnotations(List<JAnnotation> anotace)`, která získá

daný seznam z `getDeclaredAnnotations(RawByteCodeClassDumper rbccd)`.

Nastavení konstruktorů třídě se provede pomocí `setConstructors(List<JConstructor> konstruktory)`, kde se seznam konstruktorů se získá pomocí privátní metody loaderu `loadConstructors(RawByteCodeClassDumper rbccd, JClass c2)`.

Přiřazení fieldů se provádí přes `setDeclaredFields(List<JField> fields)`, pomocí privátních metod se získá seznam z loaderu `loadFields(RawByteCodeClassDumper rbccd, JClass c2)`.

K nastavení metod třídy `setDeclaredMethods(List<JMethod> metody)`, seznam metod je načten pomocí `loadMethods(RawByteCodeClassDumper rbccd, JClass c2)`.

Nastavení implementovaných rozhraní

K nastavení rozhraní, která daná třída implementuje je použita metoda `setInterfaces(List<JClass> rozhraní)`, seznam tříd je získán z loaderu pomocí `loadInterfaces(RawByteCodeClassDumper rbccd, JClass c2)`.

Nastavení použité generiky ve třídě

K nastavení generické části třídy je použita metoda `setTypeParameters(List<JTypeVariable<? extends JGenericDeclaration>> parametr)`, kde generický parametr je získán z metody loaderu `loadTypeParameters(RawByteCodeClassDumper rbccd, JClass c2)`.

private List<JConstructor> loadConstructors(RawByteCodeClassDumper dumperSource, JClass cl)

Metoda načte `List<String>` konstruktory pomocí `dumperSource.getConstructors()`. Postupně se projde celý seznam, každý `String` je rozdělen podle mezer. Struktura vráceného řetězce je popsána v části věnované `RawByteCodeClassDumperu`.

Z první části se zjistí přístupová práva a podle nich se vytvoří `JModifierImpl`. Jméno konstruktoru je uloženo vlevo od indexu `´(´`. a parametry a výjimky konstruktorem vyhazované a z nich se vytvoří instance `JConstructorImpl` a pomocí setterů se nastaví požadované vlastnosti. Výsledný konstruktor se přidá do seznamu vracených konstruktorů.

private List<JAnnotation> loadDeclaredAnnotations(RawByteCodeClassDumper dumperSource)

Metoda načítá anotace, které jsou deklarované na třídě. Výsledkem je seznam složený z `JAnnotationImpl`, které mají název a vloženou `HashMap<String, String>`, ve které jsou uloženy jména a typy parametrů.

private List<JField> loadDeclaredFields(RawByteCodeClassDumper dumperSource, JClass cl)

Metoda z `RawByteCodeClassDumperu` získá seznam `Stringů`, které obsahují informace o položce, každý `String` je postupně rozdělen podle mezer. Vytvořenému `JFieldImpl` se jako deklarující třída nastavuje `cl`. `JFieldImpl` se pomocí setterů nastaví anotace, jméno a modifikátor, které lze získat z rozparsovaného `Stringu`.

Druh typu se zjišťuje `getJClassForClass(String názevTypu, JClass cl)`, když položka není generikou, nebo pomocí `getJPar(String názevTypu, JClass cl)` pro generiky.

private List<JMethod> LoadDeclaredMethods(RawByteCodeClassDumper dumperSource, JClass cl)

Metoda z `RawByteCodeClassDumperu` získá seznam `Stringů`, které obsahují informace o metodách, `String` je rozdělen podle mezer. Vytvořenému `JMethodImpl` se jako deklarující třída nastavuje `cl`.

`JMethodImpl` se pomocí setterů nastaví anotace, jméno a modifikátor zjištěné z `dumperSource`. O jaký návratový typ se jedná, se zjišťuje `getJClassForClass(String názevTypu, JClass cl)`, parametry se zjišťují pomocí `getClassTypes(String parametersNames, JClass cl)`. Výjimky se načítají přes `getClassTypes(exceptionTypes,cl)`.

private List<JClass> loadInterfaces(RawByteCodeClassDumper dumperSource, JClass cl)

Pro každý `String` ze seznamu získaného přes `dumperSource.getImplementedInterfaces()` metoda volá metodu `getJClassForClass(String, JClass)`. Načtenou `JClass` přidává do seznamu, který bude vrácen.

private List<JTypeVariable<? extends JGenericDeclaration>> loadTypeParameters(RawByteCodeClassDumper dumperSource, JClass cl)

Metoda přes `dumperSource.getClassAttributes()` načítá atributy třídy, v atributech se vyhledá řetězec „`Signature`", když je nalezen vytvoří se instance `RawByteCodeGenericDeclaration` a přes ni se zavolá `setTypeParameters(String typeParameters, RawByteCodeClassLoader loader,JClass cl)`, kde jako `typeParameters` bude část mezi ´<´ a ´>´ v signatuře.

private JParameterizedType getJPar(String s, JClass c)

Metoda použitá pro načtení generických typů. Nastavuje `rawType` pomocí `getJClassForClass(String rawName, JClass c)`, kde `rawName` obsahuje začátek `Stringu s` do znaku ´<´, `JClass c` je třída, která daný `JParameterizedType` deklaruje.

Balík, do kterého daný `JParameterizedType` patří, se zjistí vyhledáním jména v již načtených `packages`. Pro `setActualTypeArguments(List<JType> args)` `args` zjistíme pomocí `getClassTypes(actualType, cl)`, kde `actualType` odpovídá části mezi ´<´ a ´>´ ve `Stringu s`. `JClass cl` je třída, která `JParamerizetedTypu` deklaruje.

private List<JType> getClassTypes(String[] parameterTypes, JClass c)

Metoda vytváří seznam `JTypů` pro každý `String` v poli, když `String` obsahuje ´<´ volá se metoda `getJPar(String s,JClass c)`, kde `s` je daný `String` z pole a `c` je `JClass`, pro kterou bude `JType` importem.

protected JClass getJClassForClass(String s, JClass c)

Metoda načte `JClass` pomocí `loadClass(String className)`. Když má vrácená `JClass` referenci na `null`, tak se vytvoří prázdná `JClassImpl` obsahující jen jméno třídy a odkaz na balík, do kterého patří. Balík se v případě, že ještě neexistuje, vytvoří a uloží pod svým jménem do `HashMap<String, JPackageImpl> packages`.

4.3.3. Veřejné metody

Veřejné metody jsou definované v rozhraní `JClassLoader` z balíku `cz.zcu.kiv.jacc.loader`. Tyto metody slouží pro práci se samotným loaderem.

- `public JClass getClass(String className)`- vrací načtenou `JClass` podle jejího plného jména přes metodu `JClass`

`loadClass(String plneJmeno)`. Plné jméno je vyžadováno kvůli jednoznačnosti, protože lze mít v různých balících stejně pojmenované třídy.

- `public Set<JClass> getClasses(String pcg)` – vrací `Set<JClass>`, který obsahuje `JClassy` v současné době načtené, z balíku zadaného `String pcg`.
- `public Set<JClass> getImportedClasses(String className)` – vrací reprezentaci tříd, které byly načteny během načítání třídy se jménem `String className`, ve formátu `Set<JClass>`.
- `public Set<String> getImportedPackageNames()` – vrací `Set<String>` obsahující jména importovaných balíků, prakticky vrací všechny načtené balíky kromě balíku, do kterého patřila `JClass`, pro kterou byla volána metoda `getClass(String className)`.
- `public Set<String> getPackageNames()` – vrací jména všech balíků načtených loaderem, jména jsou vrácena v `Set<String>`.

4.4. Ověření implementace pomocí jednotkových testů

Současná implementace je ověřená pomocí JUnit testů. JUnit testy jsou napsány jak na úrovni `RawByteCodeClassDumperu`, tak úrovni `RawByteCodeClassLoaderu`. Jedinou chybnou částí jsou anotace, které ale nejsou implementované ani v současném `JClassLoaderu`. Pro knihovní funkce se generují prázdné třídy jako pro třídy, které nejsou nalezeny.

4.5. Možnosti budoucího rozšíření

Budoucí upravený loader by mohl lépe pracovat s anotacemi a vytvářet pro ně `JClass` třídy. Dalším možným rozšířením je pracovat s bytecodem metod a upravovat již kompilovaný `.class` ať už ve smyslu přidávání kódu, úprav kódu, přejmenovávání metod, fieldů anebo jiných úprav.

4.6. Získané zkušenosti

Důležitou zkušeností byla nutnost rychle se zorientovat v již nastaveném datovém modelu, který byl rozsáhlý a značně využívá vlastností Java™, jako jsou dědičnost a mnohotvárnost.

Mohl jsem nahlédnout, jak se převádějí data z jednoho datového modelu do jiného datového modelu, který se navíc za dobu práce dynamicky měnil. Tato dynamičnost byla často příčinou mnoha problémů, ale jsem za tuto zkušenost rád, protože jsem zjistil, jak vypadá práce v týmu.

Velice podnětné bylo zkoumání, jakým způsobem jsou jednotlivé informace reprezentované v bytecodu. Čtení specifikací k jazyku Java™ není jednoduché, protože se jedná o velmi rozsáhlou strukturu, u které je složitost zvyšována snahou o zpětnou kompatibilitu, proto v současném bytecodu existuje plno slepých uliček, které nemají momentálně žádné využití.

5. Závěr

Výsledkem mé práce je RawByteCodeClassLoader, který převádí data ze zdrojových bytecodů do datového modelu JavaTypes.

Současná implementace loaderu splňuje požadavek na jednoduchost implementace a plné nahrazení současného loaderu. Loader obsahuje všechny funkce implementované v jeho předchůdci a navíc je oproti původnímu loaderu implementována práce s anotacemi, která by pro lepší fungování potřebovala přesnější specifikaci v datovém modelu `JavaTypes`.

V současné implementaci nelze načítat knihovní funkce, protože nelze získat pro loader jejich bytecode. Toto je jediná velká nevýhoda současné implementace, která je mi známa, ale tento problém by byl řešitelný jen za předpokladu, že by se loaderu předložil bytecode knihovních funkcí v `JAR` souboru nebo ve složkách obsahující dané třídy.

Přehled zkratek

BCEL	The Byte Code Engineering Library
JAR	Java Archive
JaCC	Java Class Compatibility checker
JVM	Java Virtual Machine
pool	Constant_pool table

Literatura

[1] Apache Software Foundation, *Byte Code Engineering Library*, 2006
http://jakarta.apache.org/bcel/manual.html

[2] ASM, *Project ASM*,
http://asm.ow2.org/index.html

[3] Class Dump Utility
http://www.aqute.biz/Code/Clsd

[4] Java™ Platform, Standard Edition 7 API Specification
http://docs.oracle.com/javase/7/docs/api/

[5] The Java™ Virtual Machine Specification Java SE 7 Edition
http://docs.oracle.com/javase/specs/jvms/se7/html/index.html

[6] Bauml, J., Brada, P. Reconstruction of Type Information from Java Bytecode for Component Compatibility. Electronic Notes in Theoretical Computer Science, 2011, svazek 264, strany 3-18011, svazek 264, strany 3-18