Format `class` dokumenta

Matej Mihelčić

Prirodoslovno-matematički fakultet, Sveučilište u Zagrebu

matmih@math.hr

24. listopada, 2022.

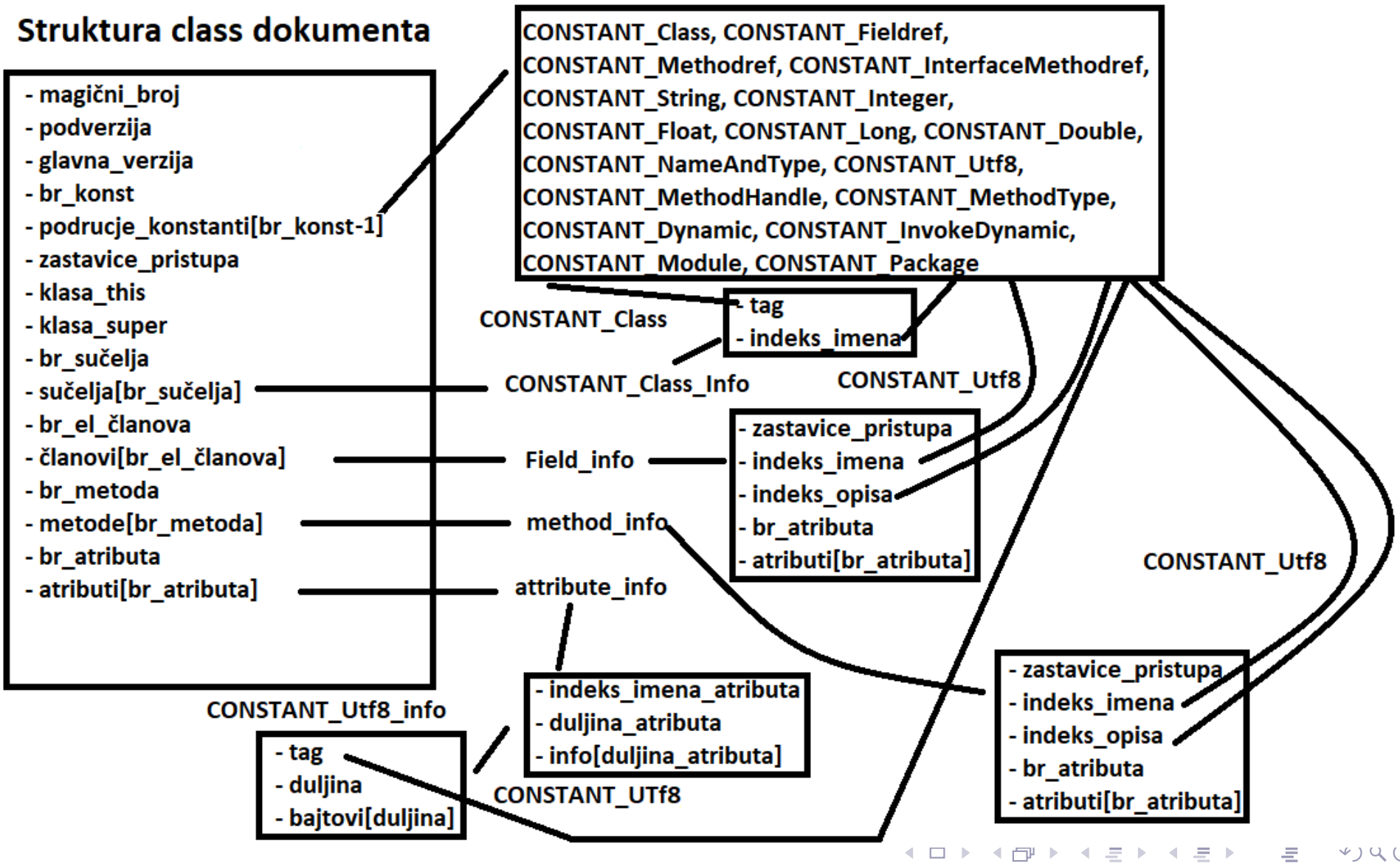
Struktura `class` dokumenta

`Class` dokument se sastoji od **niza bajtova** i u principu opisuje jednu klasu ili sučelje. 16 i 32-bitne veličine se konstruiraju **čitanjem odgovarajućeg broja bajtova**. Objekti koji sadrže više od jednog bajta se uvijek spremaju u *big-endian* poretku.

Unutar `Class` dokumenta su opisane strukture koje imaju oblik tablica s **elementima varijabilne duljine** (nije moguće jednostavno pretvaranje indeksa u pomak bajtova) i strukture koje sadrže **uzastopne elemente jednake veličine** (gdje je pretvorba lako moguća).

u_1, u_2 i u_4 predstavljaju veličinu bez predznaka od 1, 2 ili 4 bajta.

Struktura class dokumenta



Struktura class dokumenta

```
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];
}
```

Class dokument se sastoji od jedne ClassFile strukture.

Struktura `class` dokumenta

magic - služi za identifikaciju formata `class` dokumenta
major/minor version - glavna verzija i podverzija `class` dokumenta
constant_pool_count - broj elemenata u tablici konstanti
constant_pool[] - područje konstanti koje sadrži razne string konstante, imena klasa, sučelja, članova klasa i drugih konstanti koje su referirane unutar klase `ClassFile` i njezinih podstruktura. Format svakog elementa polja područja konstanti je naznačen prvim bajtom koji se naziva *tag*.
access_flags - maska bitova (zastavica) koje označavaju pravo pristupa i svojstva klase ili sučelja.
this_class - valjani indeks u tablici područja konstanti. Element na zadanom indeksu mora biti `CONSTANT_Class_Info` struktura koja reprezentira klasu ili sučelje definiranu zadanim elementom.
super_class - ili nula ili valjani indeks u tablici područja konstanti. Ukoliko je indeks različit od nula, element u području konstanti mora biti struktura `CONSTANT_Class_info` koja reprezentira direktnu nadklasu klase definirane u `class` dokumentu. Niti nadklasa niti njezine nadklase ne smiju biti `final`.

Struktura `class` dokumenta

super_class - Ukoliko je vrijednost indeksa nula, `class` dokument mora reprezentirati klasu *Object* (jedinu klasu bez nadklase). Kod sučelja, vrijednost mora uvijek biti valjan indeks u tablici područja konstanti koji je `CONSTANT_Class_Info` struktura koja reprezentira klasu *Object*.
interfaces_count - broj izravnih nadsučelja klase ili sučelja.
interfaces[] - mora biti valjan indeks u tablici područja konstanti. Svaki element na zadanom indeksu mora biti struktura `CONSTANT_Class_info` koja reprezentira sučelje koje je izravno nadsučelje klase ili sučelja opisanog u `Class` dokumentu (poredak s lijeva na desno definiran u izvornom kodu).
fields_count - broj `field_info` struktura u tablici članova. Struktura `field_info` reprezentira sve članove, varijable klasa, varijable instanci deklarirane tim tipom klase ili sučelja.
fields[] - svaki element u polju mora biti struktura `field_info` koja detaljno opisuje članice klase ili sučelja. Uključuje samo članice te klase ili sučelja (ne elemente naslijeđene od nadklasa ili nadsučelja).

Struktura `class` dokumenta

methods_count - broj `method_info` struktura u tablici metoda.
methods[] - svaki element u tablici mora biti struktura `method_info` koja sadrži potpuni opis metode u toj klasi ili sučelju. Ukoliko metoda nije *native* ili *abstract*, sadrži i instrukcije *Java virtualnog stroja* koje implementiraju funkcionalnost metode. Strukture `method_info` reprezentiraju sve metode deklarirane u klasi ili sučelju, uključujući statične metode i konstruktore. Ne sadrži metode naslijeđene od nadklasa ili nadsučelja.
attributes_count - broj atributa u tablici atributa klase.
attributes[] - Svaki element tablice atributa mora biti struktura `attribute_info`.

Predefinirani atributi u class dokumentu

Atribut	Lokacija
SourceFile	ClassFile
InnerClasses	ClassFile
EnclosingMethod	ClassFile
SourceDebugExtension	ClassFile
BootstrapMethods	ClassFile
Module, ModulePackages, ModuleMainClass	ClassFile
NestHost, NestMembers	ClassFile
ConstantValue	field_info
Code	method_info
Exceptions	method_info
RuntimeVisibleParameterAnnotations, RuntimeInvisibleParameterAnnotations	method_info
AnnotationDefault	method_info
MethodParameters	method_info

Predefinirani atributi u class dokumentu

Atribut	Lokacija
Synthetic	ClassFile, field_info, method_info
Deprecated	ClassFile, field_info, method_info
Signature	ClassFile, field_info, method_info
RuntimeVisibleAnnotations, RuntimeInvisibleAnnotations	ClassFile, field_info, method_info
LineNumberTable	Code
LocalVariableTable	Code
LocalVariableTypeTable	Code
StackMapTable	Code
RuntimeVisibleTypeAnnotations, RuntimeInvisibleTypeAnnotations	ClassFile, field_info, method_info, Code

Binarna imena

Binarne klase i sučelja se reprezentiraju `CONSTANT_Utf8_info` strukturom i sadrže puno ime (uključujući imena paketa), samo što se u binarnoj reprezentaciji znak "." zamijeni s "/". Npr. `java.lang.Thread` se zapisuje kao `java/lang/Thread`.

Imena metoda, elemenata članova klasa i sučelja, lokalnih varijabli i formalnih argumenata se spremaju imenima koja moraju sadržavati barem jedan *Unicode* kod i ne smiju sadržavati znakove `.` `;` `[` `/` . Osim `<init>` (konstruktora), imena metoda ne smiju sadržavati niti znakove <>.

Imena modula se spremaju u strukturu `CONSTANT_Module_info` u prostoru konstanti. Ona sadrži strukturu `CONSTANT_Utf8_info` koja sadrži ime modula. Kod imena modula se znak "." ne mijenja s "/". Smiju se koristiti svi *Unicode* kodovi osim onih u rasponu `\u0000` do `\u001F`, `\` je rezerviran za označavanje posebnih znakova (eng. *escape*), znakovi `:` `,` `@` su rezervirani za buduću upotrebu u imenima modula (smiju se koristiti samo `\:` `,` `\@`. Imena paketa se spremaju u strukturu `CONSTANT_Package_info` koja sadrži `CONSTANT_Utf8_info`.

Označavanje elemenata i metoda članova klasa i sučelja

Elementi članovi klasa i sučelja mogu biti osnovnog tipa, objekti ili polja. Članovi osnovnog tipa se označavaju slovima B (`byte`), C (`char`), D (`double`), F (`float`), I (`int`), J (`long`), S (`short`) i Z (`boolean`). Reference klase se reprezentiraju oznakom `LImeKlase` a reference polja oznakom `[` (npr. jednodimenzionalna).

Oznaka varijable tipa `int` je I. Oznaka varijable tipa *Object* je `Ljava/lang/Object;`. Oznaka višedimenzionalnog polja `double[][][]` je `[[[D`.

Metode se označavaju specificiranjem liste formalnih argumenata i izlaznog tipa. Npr. `Object m(int i, double d, Thread t) {...}` se označava kao `(IDLjava/lang/Thread;)Ljava/lang/Object;`.

Prostor konstanti

Instrukcije *Java virtualnog stroja* koriste simboličke informacije u tablici `constant_pool` (prostor konstanti). To polje sadrži elemente koji su strukture `cp_info`.

```
1 cp_info {
2     u1 tag;
3     u1 info[];
4 }
```

Izgled cp_info strukture.

Svaki element unutar tablice `constant_pool` mora imati `tag` od 1-bajt koji označava tip konstante koju reprezentira. Nakon `tag`-a slijede 2 ili više bajtova koji pružaju informacije o danoj konstanti. Format informacija ovisi o tipu konstante.

Svaki element prostora konstanti mora biti odgovarajuće verzije (jednake ili manje verziji `class` dokumenta). Neki elementi prostora konstanti se mogu **učitavati** pošto ih je potrebno staviti na stog da bi se moglo izvršiti računanje.

Elementi članovi klasa

Svaki element je opisan strukturom `field_info`. Ne smiju postojati dva elementa u `class` dokumentu s istim imenom polja i opisom.

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

Izgled filed_info strukture.

access_flags - maska zastavica koje se koriste za definiranje dopuštenja pristupa elementima te svojstva elemenata članova klasa.
name_index - valjani indeks u tablici polja konstanti gdje mora biti element strukture `CONSTANT_Utf8_info`. Taj element predstavlja valjano ime elementa člana klase.
descriptor_index - valjani indeks u tablici polja konstanti gdje mora biti element strukture `CONSTANT_Utf8_info`. Taj element predstavlja opis elementa člana klase.

Elementi članovi klasa i metode

attributes_count - broj dodatnih atributa elementa člana klase.
attributes[] - dodatni atributi elementa člana klase. Atributi su `attribute_info` strukture.

Metode: Svaka metoda je opisana strukturom `method_info`. Ne smiju postojati dvije metode u `class` dokumentu koje imaju isto ime i opis.

```
1 method_info {
2     u2             access_flags;
3     u2             name_index;
4     u2             descriptor_index;
5     u2             attributes_count;
6     attribute_info attributes[attributes_count];
7 }
```

Izgled method_info strukture.

Elementi `method_info` strukture imaju isto značanje kao i kod strukture `field_info`.

Atributi

Atributi se koriste u strukturama `ClassFile`, `field_info`, `method_info` i `Code_attribute`.

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

Izgled attribute_info strukture.

attribute_name_index - 16-bitni indeks u polje konstanti klase. Odgovarajući element je `CONSTANT_Utf8_info` struktura koja sadrži ime atributa.
attribute_length - označava duljinu polja koje sadrži informacije o atributu u bajtovima (ne uključujući početnih šest bajtova korištenih za spremanje strukture koja sadrži ime atributa i duljinu polja informacija o atributu).
info - polje koje sadrži dodatne informacije o atributu.

Atributi

Postoji ukupno 28 predefiniranih atributa od kojih je 6 **ključno** za korektnu interpretaciju `class` dokumenta.

6 ključnih atributa je:

- `ConstantValue` - sprema vrijednosti konstantnih izraza
- `Code` - sadrži instrukcije *Java virtualnog stroja* i pomoćne informacije metoda.
- `StackMapTable` - sprema informacije potrebne za verifikaciju dokumenta provjerom tipova (eng. *type checking*).
- `BootstrapMethods` - sprema *bootstrap metode* koje se koriste za dinamičko računanje konstanti i dinamički određene točke poziva metoda.
- `NestHost` - atribut fiksne duljine koji sadrži informaciju o klasi (eng. *nest host*) koja sadrži spisak klasa i sučelja koja spadaju u skup klasa koje međusobno dijele pristup privatnim članovima. Tom skupu spada i trenutna klasa.

Atributi

Tablica atributa `class` dokumenta smije sadržavati najviše jedan `NestHost` dokument.

- `NestMembers` - atribut promijenjive duljine koji sadrži klase i sučelja koja imaju pravo pristupa zaštičenim članovima klasa i sučelja iz skupa specificiranog od strane *nest host*-a (koji je trenutna klasa).

Tablica atributa `class` dokumenta smije sadržavati najviše jedan `NestMembers` atribut. Također, tablica atributa ne smije istovremeno sadržavati i `NestMembers` i `NestHost` atribut.

Atribut code

Atribut koda (eng. code) je atribut promijenjive veličine u tablici atributa strukture `method_info`.

```
Code_attribute {
    u2 attribute_name_index;
    u4 attribute_length;
    u2 max_stack;
    u2 max_locals;
    u4 code_length;
    u1 code[code_length];
    u2 exception_table_length;
    {   u2 start_pc;
        u2 end_pc;
        u2 handler_pc;
        u2 catch_type;
    } exception_table[exception_table_length];
    u2 attributes_count;
    attribute_info attributes[attributes_count];
}
```

Izgled code_attribute strukture.

Atribut code

Sadrži instrukcije *Java virtualnog stroja* i pomoćne informacije metoda.

Apstraktne metode i sistemske metode ne smiju sadržavati atribut `code`. Sve ostale metode sadrže točno jedan `code` atribut u tablici atributa.

attribute_name_index - valjani indeks u području konstanti. Odgovarajući element je struktura `CONSTANT_Utf8_info` koja reprezentira string *Code*.
attribute_length - označava duljinu atributa bez početnih 6 bajtova.
max_stack - sadrži maksimalnu dubinu stoga operanada metode u bilo kojem trenutku izvođenja.
max_locals - sadrži broj lokalnih varijabli u polju lokalnih varijabli koje su alocirane tijekom poziva metode (uključuje lokalne varijable korištene za prijenos parametara metodi). Najveći indeks lokalne varijable tipa *long* ili *double* je `max_locals`-2 a varijable bilo kojeg drugog tipa `max_locals`-1.
code_length - sadrži broj bajtova ($0 < n < 65536$) u polju koje sadrži kod metode.

Atribut code

code[] - sadrži oktenti kod instrukcija *Java virtualnog stroja* koje čine metodu.
exception_table_length - sadrži broj elemenata u tablici `exception_table`.
exception_table[] - svaki element opisuje jedan kod za obradu iznimke u polju `code`. Poredak elemenata u polju je bitan. Svaka tablica iznimke sadrži četiri elementa:

- ***start_pc*** - početak koda (indeks je uključiv) u polju koda u kojemu je aktivan kod za obradu iznimke.
- ***end_pc*** - kraj koda (indeks je isključiv i veći od `start_pc`) u polju koda u kojem je aktivan kod za obradu iznimke.
- ***handler_pc*** - označava početak koda za obradu iznimke.

Atribut code

- ***catch_type*** - različit od nule i valjan indeks u polju konstanti. Element na odgovarajućoj lokaciji je struktura `CONSTANT_Class_info` koji reprezentira klasu iznimaka koje zadani kod za obradu iznimaka može obraditi. Kod za obradu iznimke će se pozvati samo ako je izbačena iznimka instance dane klase ili neke njezine podklase. Ukoliko je vrijednost `catch_type` jednaka nula, tada se kod za obradu iznimaka poziva za sve vrste iznimaka.

attributes_count - označava broj atributa od atributa `code`.
attributes[] - elementi su strukture `attribute_info`.

Učitavanje klasa i sučelja

Java virutalni stroj **dinamički učitava**, **povezuje** i **inicijalizira** klase i sučelja.

- **Učitavanje** - proces traženja binarne reprezentacije tipa klase ili sučelja zadanog imena i stvaranja te klase ili sučelja iz binarne reprezentacije.
- **Povezivanje** - proces povezivanja klase ili sučelja s trenutnim stanjem *Java virtualnog stroja* tako da se omogući izvršavanje.
- **Inicijalizacija** - sastoji se od izvršavanja inicijalizacijskih metoda `clinit` klase ili sučelja.

Java virtualni stroj se pokreće stvaranjem **inicijalne klase ili sučelja** koristeći *bootstrap* učitavač klase ili učitavač definiran od strane korisnika. Nakon toga povezuje početnu klasu ili sučelje, inicijalizira i poziva metodu `public static method void main(String[])`. Poziv te metode pokreće sva nadolazeća izvršavanja. To može rezultirati **povezivanjem** i **stvaranjem dodatnih klasa**, **sučelja** i **pozivom dodatnih metoda**.

Učitavanje i stvaranje klasa i sučelja

Stvaranje klasa ili sučelja se sastoji od **stvaranja interne reprezentacije** (ovisne o implementaciji) u **području metoda** *Java virtualnog stroja*. Stvaranje se pokreće od strane neke druge **klase** ili **sučelja** koje referencira novo kreiranu klasu kroz svoj **skup konstanti pri izvršavanju**. Stvaranje se može pokrenuti i **pozivom nekih metoda** iz *Java* biblioteka.

Polja se stvaraju od strane *Java virtualnog stroja* i **nemaju binarnu reprezentaciju**.

Vrste učitavača klase:
- **Bootstrap učitavač klase** - standardni učitavač *Java virtualnog stroja*.
- **učitavač definiran od strane korisnika** - instanca klase `ClassLoader`.

Koristi se za **kostumiziranje načina dinamičkog učitavanja i kreiranja klasa** (npr. stvaranje klasa iz izvornih datoteka definiranih od strane korisnika, klasa preuzetih preko mreže, generiranih tijekom izvršavanja ili dobivenih iz kriptiranih dokumenata).

Kod izvršavanja *Java* programa se može koristiti i **više različitih učitavača klasa**.

Učitavanje i stvaranje klasa i sučelja

Učitavač može **sam učitati klasu** ili **zadati učitavanje klase** nekom **drugom učitavaču**.

Klase i sučelja su tijekom izvršavanja određena svojim **binarnim imenom** i **učitavačem klase**. Takve klase i sučelja pripadaju najviše jednom paketu koji je definiran **imenom paketa** i **odgovarajućim učitavačem** klasa i sučelja.

Dobro definirani učitavač klasa ima tri glavna svojstva: a) za **dano ime klase**, treba uvijek vratiti **isti** `Class` objekt, b) ako učitavač klase L_1 doznači učitavanje klase C drugom učitavaču L_2, tada **za svaki tip** T koji je nadklasa, direktno nadsučelje klase C, T se javlja kao tip elementa člana klase C, tip formalnog parametra metode, konstruktora ili kao povratna vrijednost metode iz C, L_1 i L_2 trebaju vratiti **isti** `Class` objekt, c) ukoliko učitavač klasa definiran od strane korisnika **unaprijed** učitava binarnu reprezentaciju niza klasa i sučelja, tada potencijalne greške pri učitavanju treba odaslati **tek u trenutku** kada bi do njih došlo **bez učitavanja unaprijed**.

Učitavanje klasa i sučelja

Učitavanje korištenjem *Bootstrap* učitavača klasa:
- *Java virtualni stroj* odredi je li *Bootstrap* učitavač klasa već **zabilježen** kao učitavač zadane klase ili sučelja. Ukoliko je, nije potrebno kreiranje klase.
- Inače, *Java virtualni stroj* šalje klasu kao argument metodi *Bootstrap* učitavača klase koja **traži reprezentaciju klase ili sučelja** (obično datoteku u datotečnom sustavu) na način ovisan o platformi. Ukoliko reprezentaciju nije moguće pronaći, izbacuje se iznimka `ClassNotFoundException`. Ukoliko je dokument pronađen, *Java virtualni stroj* pokušava rekreirati klasu iz dane reprezentacije korištenjem *Bootstrap* učitavača klase.

Učitavanje korištenjem učitavača klase definiranog od strane korisnika:
- *Java virtualni stroj* odredi je li dani učitavač klasa već **zabilježen** kao učitavač zadane klase ili sučelja. Ukoliko je, nije potrebno kreiranje klase.

Učitavanje klasa i sučelja

Učitavanje korištenjem učitavača klase definiranog od strane korisnika:
- Inače, *Java virtualni stroj* pozove metodu `loadClass(arg)` na **zadanom** učitavaču klasa. Povratna vrijednost je **kreirana klasa ili sučelje**. Nakon toga *Java virtualni stroj* **zabilježi** da je učitavač dane klase ili sučelja upravo korišteni učitavač klasa.

Nakon poziva `loadClass` metode zadanog učitavača klase, događa se jedan od dva scenarija:
- Učitavač klase **stvara polje bajtova** koji reprezentiraju bajtove `ClassFile` strukture tražene klase, nakon čega **poziva metodu** `defineClass` klase `ClassLoader`. Pozivom `defineClass` metode *Java virtualni stroj* **stvara klasu ili sučelje** iz polja bajtova.
- Učitavač klase može **doznačiti** učitavanje klase **nekom drugom učitavaču** pozivom `loadClass` metode tog učitavača. Tada se klasa ili sučelje **stvara iz polja bajtova** koje stvara pozvani (drugi) učitavač.

Ukoliko nije moguće učitati klasu ili sučelje, izbacuje se iznimka `ClassNotFoundException`.

Stvaranje klasa polja

Učitavač klase korišten u ovom kontekstu može biti ili *Bootstrap* učitavač klase ili učitavač klase definiran od strane korisnika.

- Ukoliko je učitavač klase **već zabilježen** kao učitavač klase polja s tipom komponente klase C, tada **nije potrebno stvaranje klase polja**.
- Inače, **ukoliko je tip komponente polja referencirani tip**, koristi se **algoritam za učitavanje klase ili sučelja** (definiran ranije), koristeći učitavač klase **rekurzivno na tip komponente polja**.
-*Java virtualni stroj* **stvara novu klasu polja zadanog tipa komponenti i broja dimenzija**. Ukoliko je tip komponente referencirani tip, označava se da je tip komponente definiran predefiniranim učitavačem klase ili *Bootstrap* učitavačem klase (što *Java virtualni stroj* i zabilježi kao definirani učitavač).

Ukoliko je tip komponete polja referencirani tip, dopuštenja pristupa (dohvatljivost) se **određuju prema dopuštenjima pristupa tipa komponente**. U suprotnom je klasa polja **dohvatljiva svim klasama i sučeljima**.

Stvaranje klase iz `class` dokumenta

Java virtualni stroj kontinuirano provjerava je li došlo do narušavanja uvjeta pri učitavanju klasa. Odnosno, je li došlo do slučaja u kojem **dva različita učitavača klase** učitaju klasu **istog binarnog imena** međutim radi se o **različitim klasama**. Ukoliko dođe do nedopuštenog učitavanja, *Java virtualni stroj* prekida stvaranje klase ili sučelja i vraća `LinkageError`.

Klasa ili sučelje se stvara koristeći neki učitavač klase iz `class` dokumenta na sljedeći način:
- *Java virtualni stroj* prvo utvrdi je li **već zabilježeno** da je učitavač klase *L* učitavač dane klase ili sučelja (u tom slučaju izbacuje `LinkageError`).
- Inače, učitavač **analizira i pokuša učitati** reprezentaciju klase. U ovom koraku može doći do sljedećih pogrešaka:

  - Ukoliko reprezentacija ne predstavlja strukturu `Class` dokumenta, izbacuje se iznimka `ClassFormatError`.
  - Ukoliko je reprezentacija stvorena od strane nepodržane verzije *Jave* (*JDK* koji odgovara višem broju *JRE* od onog koji pokušava pročitati reprezentaciju), izbacuje se `UnsupportedClassVersionError`.

Stvaranje klase iz `class` dokumenta

- Ukoliko reprezentacija ne reprezentira klasu pravog imena, izbacuje se iznimka `NoClassDefFoundError` ili neka njezina podklasa.

- Potencijalna direktna nadklasa neke klase se dohvača preko simboličke reference. Ukoliko se radi o sučelju, direktna nadklasa mora biti `Object`. Sve iznimke vezane uz dohvaćanje nadklase se izbacuju u ovoj fazi.

  - Direktna nadklasa klase ne smije biti sučelje inače dolazi do `IncompatibleClassChangeError` iznimke.
  - Ako je bilo koja nadklasa klase ta ista klasa, dolazi do `ClassCircularityError`.

- Potencijalno direktno nadsučelje se dohvača preko simboličke reference. Sve iznimke vezane uz dohvaćanje nadsučelja se izbacuju u ovoj fazi.

  - Ukoliko bilo koja klasa ili sučelje koji su definirani kao nadsučelja nije sučelje izbacuje se `IncompatibleClassChangeError`.
  - Ukoliko je bilo koje nadsučelje to isto sučelje, izbacuje se iznimka `ClassCircularityError`.

- *Java virtualni stroj* označava učitavača klase.

Moduli i slojevi

Java virtualni stroj podržava **organizaciju** klasa i sučelja u **module**. Glavna svrha takve organizacije je **ograničiti pristup klasama** i **sučeljima** jednog modula iz **drugih modula**.

Pripadnost modulima se definira s obzirom na pripadnost **definiranim paketima tijekom izvođenja**. Program odredi **imena paketa u svakom modulu** i **učitavače klasa** koji će stvoriti klase i sučelja u tim imenovanim paketima. Pakete i učitavače klasa proslijedi metodi `defineModules` klase `ModuleLayer`. Nakon tog poziva, *Java virtualni stroj* **stvara novi modul definiran za potrebe izvođenja programa** koji je povezan s paketima definiranim za potrebe izvođenja programa određenih učitavača klasa.

Svaki modul definiran za potrebe izvođenja programa **navodi pakete čije korištenje dozvoljava drugim modulima** (*public* klase i sučelja tih paketa se smiju koristiti). Moduli također **definiraju koje pakete drugih modula trebaju čitati**.

Moduli i slojevi

Klasa kreirana od strane učitavača klase se nalazi u **točno jednom paketu** i **točno jednom modulu** (s obzirom da su paketi definirani za potrebe izvođenja uvijek asocirani s točno jednim modulom). Modul definiran za potrebe izvođenja je povezan s **točno jednim učitavačem klasa**. Učitavač klasa **može kreirati klase iz raznih modula**.

Svaki modul definiran u svrhu izvršavanja programa je dio **sloja** (skupine učitavača klasa koji zajedno stvaraju klasu za skup definiranih modula). Postoje dvije vrste slojeva: a) `boot` sloj definiran od strane *Java virtualnog stroja* i b) sloj definiran od strane korisnika.

Nakon stvaranja sloja, skup učitavača klasa pridružen sloju i skup modula koji su dio sloja se **ne mogu mijenjati**. Dopuštena je određena doza fleksibilnosti glede **veza između modula** unutar sloja **definiranog od strane korisnika**.

Povezivanje klase ili sučelja

Povezivanje klase ili sučelja se sastoji od sljedećih koraka:

- **Verifikacija** klase ili sučelja, direktne nadklase ili nadsučelja i tipa elemenata ukoliko se radi o polju.
- **Pripremanje** klase ili sučelja, direktne nadklase ili nadsučelja i tipa elemenata ukoliko se radi o polju.
- **Određivanje** elemenata klase ili sučelja povezanih simboličkim referencama.

Strategije povezivanja:

- **Strategija odgođenog (eng. lazy) povezivanja** - simboličke reference klasa i sučelja se određuju tek kada se koriste.
- **Strategija ranog (eng. eager) povezivanja** - sve simboličke reference se određuju odjednom pri verifikaciji klase ili sučelja.

Greške pri određivanju simboličkih referenci se prijavljuju **u dijelu programa** gdje se **koristi** simbolička referenca koja se ne može odrediti.

Pošto povezivanje alocira nove strukture podataka, može doći do `OutOfMemoryError`.

Verifikacija, pripremanje i određivanje

Verifikacija osigurava da je **binarna reprezentacija** klase ili sučelja **strukturalno točna**. **Može uzrokovati učitavanje dodatnih klasa** ili **sučelja** koja **ne trebaju biti verificirana** ili **pripremljena**.
Vraća `VerifyError` ukoliko dođe do greške pri verifikaciji. Može doći i do greške `LinkageError`.

Pripremanje uključuje **stvaranje statičkih elemenata klase** ili **sučelja** i **inicijaliziranje** takvih elemenata na **predefinirane standardne početne vrijednosti**. U ovom koraku se **ne pozivaju konstruktori**, međutim **provjeravaju se uvjeti pri učitavanju klase** ili **sučelja**. Pripremanje se može izvršiti u **bilo kojem trenutku nakon stvaranja klase** ili **sučelja** ali **mora završiti prije inicijalizacije** (gdje se pozivaju konstruktori).

Određivanje simboličkih referenci elemenata klasa i struktura je postupak **dinamičkog određivanja jedne ili više vrijednosti** iz **simboličkih referenci** smještenih u **skup konstanti kod izvršavanja**. Na početku su sve simboličke reference **neodređene**.

Verifikacija, pripremanje i određivanje

Određivanje simboličkih referenci klasa, sučelja, elemenata i metoda članica, dinamički izračunatih konstanti slijedi **predefinirani niz koraka** (detalje povezivanja ispuštamo[1]).

Ukoliko ne dođe do grešaka prilikom određivanja simboličkih referenci, određivanje je **uspješno** te će svaki sljedeći pristup tim referencama **uvijek uspjeti** i **rezultirati istim elementima** koji su dohvaćeni pri **prvom određivanju**. Ukoliko se radi o dinamičkim konstantama, `bootstrap` metode se **ne izvršavaju ponovo** u naknadnim dohvaćanjima.

Moguće greške su `IncompatibleClassChangeError`, `Error` ili `LinkageError` koje se izbacuju **na mjestu u programu gdje se koriste te simboličke reference**. Svi naredni pokušaji određivanja tih referenci **su neuspješni** i `bootstrap` metode se **ne izvršavaju ponovo** u slučaju dinamičkih konstanti.

[1] `https://docs.oracle.com/javase/specs/jvms/se17/html/jvms-5.html`

Kontrola pristupa

Kontrola pristupa se vrši tijekom **određivanja simboličkih referenci** da bi se osiguralo da je **dopušten pristup referenci klase, sučelja, elementa člana** ili **metode**.

Klasa ili sučelje C je **dostupno** klasi ili sučelju D ako i samo ako je istina jedna od tvrdnji:
- C je *public* i **član istog modula** kao D.
- C je *public* i član različitog modula od D ali modul čiji je D član **čita modul** čiji je C član i **taj modul pruža pristup paketu** kojemu je C član modulu od D.
- C nije *public* ali su C i D **članovi istog paketa**.

Ukoliko C nije dohvatljiv klasi ili sučelju D, dolazi do `IllegalAccessError`.

Gnijezdo (eng. *nest*) je skup klasa i sučelja koji **međusobno omogućuju pristup svojim privatnim elementima članovima**.

Kontrola pristupa

Element član klase ili sučelja ili metoda R su dohvatljivi iz klase ili sučelja D ako i samo ako vrijedi barem jedno od:
- R je *public*
- R je *protected* i deklariran u klasi C, D je **podklasa** od C ili **sam** C. Ukoliko R nije *static*, tada njegova simbolička referenca mora sadržavati simboličku referencu na podklasu, nadklasu od D ili na D.
- R je *protected* ili nije specificiran identifikatora pristupa i deklariran je u klasi koja je u **istom paketu** kao D.
- R je *private* i deklariran u klasi ili sučelju C koje **pripada istom gnijezdu** kao D.

Ukoliko R nije dohvatljiv iz D, kontrola pristupa vraća `IllegalAccessError`.

Nadjačavanje metoda

Metoda m_1 može nadjačati metodu m_2 ako i samo je istina sve od navedenog:
- m_1 ima **isto ime i opis** kao m_2.
- m_1 **nije označen** kao `ACC_PRIVATE`.
- **Jedno od navedenog** je istina:
  - m_2 **je označen** `ACC_PUBLIC`.
  - m_2 **je označen** `ACC_PROTECTED`.
  - m_2 **nije označen** niti `ACC_PUBLIC` niti `ACC_PROTECTED` niti `ACC_PRIVATE` i a) deklaracija od m_2 se nalazi u **istom paketu** kao deklaracija od m_1 ili b) ako je m_2 deklariran u klasi A i m_1 u klasi C, tada postoji metoda m_3 deklarirana u klasi B takva da je C podklasa od B a B je podklasa od A i m_1 **može nadjačati** m_3 i m_3 **može nadjačati** m_2.

Izabir metoda

Tijekom izvođenja naredbi `invokeinterface` ili `invokevirtual`, metoda se izabire s obzirom na **tip objekta na stogu** tijekom izvršavanja i **metodu koja je prije određena od strane instrukcije**.

Pravila za odabir metode s obzirom na klasu ili sučelje C i metodu m su:
- ako je m `ACC_PRIVATE`, onda je to izabrana metoda.
- inače, metoda se izabire **procedurom traženja**:

  - Ako C sadrži deklaraciju metode instance klase m_1 koja može nadjačati m, tada je m_1 tražena metoda.
  - Inače, ako C ima nadklasu, potraga za metodom koja može nadjačati m počinje s njom i nastavlja se s nadklasama te klase dok se ne pronađe odgovarajuća metoda ili ne ispitaju sve nadklase.
  - Inače, određuju se najspecifičnije metode nadsučelja od C. Ukoliko jedna odgovara m i nije *abstract*, to je tražena metoda.