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

Diplomová práce

Analýza popisů sémantického kontraktu v Java technologiích

Plzeň 2018 Václav Mareš

Místo této strany bude zadání práce.

Prohlášení

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

V Plzni dne 11. dubna 2018

Václav Mareš

Poděkování

Chtěl bych poděkovat vedoucímu mé diplomové práce Doc. Ing. Přemyslu Bradovi, MSc., Ph.D. za cenné rady a připomínky, které mi pomohly tuto práci dokončit.

Abstract

The text of the abstract (in English). It contains the English translation of the thesis title and a short description of the thesis.

Abstrakt

Text abstraktu (česky). Obsahuje krátkou anotaci (cca 10 řádek) v češtině. Budete ji potřebovat i při vyplňování údajů o bakalářské práci ve STAGu. Český i anglický abstrakt by měly být na stejné stránce a měly by si obsahem co možná nejvíce odpovídat (samozřejmě není možný doslovný překlad!).

Obsah

1 Úvod **8**

2 Zajištění kvality software **9**

3 Design by contract **10**
3.1 Koncept kontraktů softwarových modulů 10
3.2 Vliv použití kontraktů na kvalitu kódu a software 10
3.3 Design by contract 10
3.4 Způsoby popisu kontraktů v Java technologiích 10
3.5 Kontrakty v jiných technologiích 10

4 Reprezentace gramatiky a jazyky **11**
4.1 Jazyk Java a jeho gramatika 11
4.1.1 Možnosti parsování 11
4.2 Bytcode 11
4.2.1 Možnosti parsování 11
4.2.2 Limitace dekompilace 11

5 Datový model **12**
5.1 Rozbor vybraných DbC konstrukcí 12
5.1.1 Guava Preconditions 12
5.1.2 JSR305 12
5.2 Společné znaky 12
5.3 Vytvořený model 12
5.3.1 Popis 12
5.3.2 Diagram 12
5.3.3 Reprezentace modelu 12

6 Implementace nástroje pro analýzu kontraktů **13**
6.1 Implementace knihovny 13
6.1.1 Použité technologie 13
6.1.2 Dekompilace Bytcode 13
6.1.3 Parsování Java souborů 14
6.1.4 Extrakce kontraktů 14
6.1.5 Porovnávání kontraktů 14
6.1.6 Popis API 14

6.1.7 Přidání parseru pro nový typ kontraktu 14
6.2 Implementace uživatelské aplikace 14
6.2.1 Použité technologie 14
6.2.2 Rozdělení aplikace 15
6.2.3 Možnosti a limitace aplikace 15

7 Optimalizace **16**
7.1 Analýza a refaktoring kódu 16
7.2 Zjdenodušení modelu 16

8 Testování **17**
8.1 Prostředky použité k testování 17
8.2 Testovací data 17
8.3 Výsledky testů 17

9 Zhodnocení výsledků **18**
9.1 Úspěšnost detekce kontraktů 18
9.2 Úspěšnost porovnání kontraktů 18
9.3 Limitace 18
9.4 Prostor pro zlepšení 18

10 Závěr **19**

Literatura **21**

A Uživatelská příručka **23**

B Obsah CD **24**

1 Úvod

Cílem této práce bylo ... TODO

2 Zajištění kvality software

3 Design by contract

3.1 Koncept kontraktů softwarových modulů

3.2 Vliv použití kontraktů na kvalitu kódu a software

3.3 Design by contract

Pojem design by contract zavedl francouzský profesor Bertrand Meyer ve ... [3]

Hlavním cílem design by contract je zvýšení spolehlivosti a správnosti u rozsáhlých softwarových projektů. Principem DbC je zajištění formální dohody mezi vývojářem a uživatelem určitého softwarového modulu. Pomocí design by contract může vývojář specifikovat očekávané hodnoty. Za předpokladu, že vývojář odvede kvalitní práci a uživatel tento kontrakt dodrží, mělo by to zamezit množství chyb spojených s integrací. Desing by contract zajišťuje správnost pomocí "assertions", kterých jsou celkem tři typy. Prvním z nich jsou tzv. pre-condtions, které zajišťují správné hodnoty vstupních parametrů, tedy hodnoty před tím, než je provedena určitá operace. Jejich opakem jsou pak post-conditions, které naopak zajišťují správnost výstupu, tedy hodnot po provedení dané operace. Třetím typem jsou tzv invariants, nebo-li neměnné proměnné, které musejí platit po celou dobu.

Podmínky jsou definovány pomocí konstrukcí v kódu programu a jsou pak ověřovány při každém běhu. V případě, že byla některá z nich porušena, je vyvolána výjimka. Tímto chováním je zajištěno, že kontrakt bude dodržen. [2]

3.4 Způsoby popisu kontraktů v Java technologiích

3.5 Kontrakty v jiných technologiích

4 Reprezentace gramatiky a jazyky

4.1 Jazyk Java a jeho gramatika

4.1.1 Možnosti parsování

4.2 Bytcode

4.2.1 Možnosti parsování

4.2.2 Limitace dekompilace

5 Datový model

5.1 Rozbor vybraných DbC konstrukcí

5.1.1 Guava Preconditions

5.1.2 JSR305

5.2 Společné znaky

Při rozboru jednotlivých nástrojů pro reprezentaci design by contract snadno zjistíme, že sdílejí mnoho podobných aspektů, které jsou klíčové pro vytvoření obecného modelu, který je schopen zachytit libovolnou konstrukci tohoto kontraktu.

5.3 Vytvořený model

5.3.1 Popis

Vytvořil jsem model...

5.3.2 Diagram

Obrázek

5.3.3 Reprezentace modelu

Data jsou reprezentována pomocí JSON...

6 Implementace nástroje pro analýzu kontraktů

shrnout zadání (kontext, proč se ta aplikace dělá, pak návrh řešení)

6.1 Implementace knihovny

6.1.1 Použité technologie

Knihovna byla implementována v jazyce Java verze 1.8 ve vývojovém prostředí IDEA IntelliJ Ultimate 2017.3.3. Pro zajištění snadného získání závislostí a následného zjednodušení použití knihovny byla pro vytvoření projektu použita technologie Apache Maven.

Knihovny třetích stran

Při vývoji knihovny pro analýzu kontraktů byly použity následující knihovny třetích stran:

Apache Log4j Tato knihovna umožňuje pokročilé možnosti pro logování. Byla použita zejména pro zaznamenávání chyb a různých informačních záznamů [4].

Procyon Knihovna Procyon byla použita pro dekompilaci přeložených Java souborů `*.class` [5].

JavaParser JavaParser byl použit tokenizaci zdrojových souborů [6].

Google Gson Tato knihovna poskytuje prostředky pro uložení objektů jazyka Java do reprezentace pomocí formátu JSON [7].

jUnit 5 Poskytuje možnosti testování pomocí jednotkových testů [8].

6.1.2 Dekompilace Bytcode

Pro dekompilaci Java `*.class` souborů byla použita knihovna Procyon. Ta umožňuje použití metody `decompile()`, která přečte vstupní soubor s přeloženým kódem a do jiného souboru uloží jeho dekompilovanou verzi. Tento

dočasný soubor s dekompilovaným kódem je poté předán pro zpracování třídě `JavaFileParser`, která jej zpracuje stejně jako běžný zdrojový soubor. V knihovně dekompilaci obstarává metoda `decompileClassFile()`, která se nachází v třídě `io.IOServices`.

6.1.3 Parsování Java souborů

Pro zpracování zdrojových souborů jazyku Java byla použita knihovna JavaParser. Ta poskytuje metodu `parse()`, která vytvoří komplexní strukturu daného zdrojového souboru. V prvním kroku se tato struktura projde a vyhledá všechny třídy (*class*) a také rozhraní *interface* a výčtové typy *enum*. Pro účely modelu jsou si tyto tři prvky rovny. Každý nalezený prvek je následně uložen do modelu. V případě třídy a rozhraní je se struktura prochází dále a do modelu jsou uloženy všechny konstruktory, které se z hlediska modelu považují za metody (viz níže). Následně jsou uloženy všechny anotace dané „třídy“.

Po této přípravě je využita třída `MethodVisitor`, která dědí od třídy `VoidVisitorAdapter` a umožňuje procházet všechny metody v daném souboru. V metodě pak máme k dispozici objekt typu `MethodDeclaration`, který obsahuje všechny potřebné údaje a také rodičovský `ExtendedJavaFile`. Pro každou metodu je nalezena její rodičovská třída. Hledá se nejvyšší rodič a tudíž vnořené metody nemají jako rodiče vyšší metodu ale nejvyšší dostupnou třídu. Pro danou metodu jsou následně uloženy všechny anotace a i její parametry. Následně je uloženo celé tělo metody jako seznam objektů typu `Node`, které umožňují další zpracování. Z těchto získaných dat je vytvořena instance objektu `ExtendedJavaMethod`, která je následně uložena do

6.1.4 Extrakce kontraktů

Obecně

Guava Precondtions

JSR305

6.1.5 Porovnávání kontraktů

6.1.6 Popis API

6.1.7 Přidání parseru pro nový typ kontraktu

6.2 Implementace uživatelské aplikace

6.2.1 Použité technologie

Java, JavaFX, Maven, knihovny, ...

6.2.2 Rozdělení aplikace

Grafická část

Konzolová část

6.2.3 Možnosti a limitace aplikace

7 Optimalizace

obecný kecy, snažil jsem se zajistit co nejlepší...

7.1 Analýza a refaktoring kódu

- ručně, nástroje IDE, -> snížení cyklomatičnosti, zpřehlednění kódu

7.2 Zjdenodušení modelu

- vyhnutí se použití rozsáhlých obejktů knihovny - pozitivní dopad na paměťovou náročnost, zpřehlednění modelu

- při batch soubory průběžně ukládat, aby nezatěžovalo paměť - přeparsování souborů se nevyplatí - stačí udělat vše a pak jen filtrovat - rozebrat nároky na paměť v aplikaci

8 Testování

8.1 Prostředky použité k testování

- Unit testy - integrační testy - malá aplikace nemusí se tolik řešit - funkční testy - testovat na úrovni GUI (ošetření vstupů) - PMD

8.2 Testovací data

- popis testovacích dat (syntetická, skutečná - výsledky testů)

8.3 Výsledky testů

9 Zhodnocení výsledků

9.1 Úspěšnost detekce kontraktů

9.2 Úspěšnost porovnání kontraktů

9.3 Limitace

9.4 Prostor pro zlepšení

- co by šlo zlepšit doplnit - lepší parsování kontrakt expression - zhruba jak

10 Závěr

TODO

Přehled zkratek

DbC Design By Contract (...)

Literatura

[1] Jens Dietrich, David J. Pearce, Kamil Jezek, and Premek Brada: *Contracts in the Wild: A Study of Java Programs*. In LIPIcs-Leibniz International Proceedings in Informatics (Vol. 74), ECOOP 2017. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2017

[2] Andreas Rausch: *"Design by Contract"+ "Componentware-"Design by Signed Contract"*. Journal of Object Technology 2002, Vol. 1 Pages 20-36

[3] *Bertrand Meyer's technology + blog* [online]. [cit. 2018-04-11]. <`bertrandmeyer.com/bio/`>

[4] *Apache Log4j* [online]. [cit. 2018-04-11]. <`logging.apache.org/log4j/2.x/download.html`>

[5] *Procyon* [online]. [cit. 2018-04-11]. <`bitbucket.org/mstrobel/procyon`>

[6] *JavaParser* [online]. [cit. 2018-04-11]. <`github.com/javaparser/javaparser`>

[7] *Gson* [online]. [cit. 2018-04-11]. <`github.com/google/gson`>

[8] *jUnit* [online]. [cit. 2018-04-11]. <`junit.org/junit5/`>

Seznam příloh

A Uživatelská příručka B Obsah CD

A Uživatelská příručka

B Obsah CD