Assignment 2: Source code modernisation

Due 10 Mar by 7:00 **Points** 10 **Submitting** a file upload **File types** zip

In this assignment you will design a tool to support a software modernisation step and apply it to the source code of a small software system. As before, this assignment should also be submitted in pairs. You should submit both the report and the Rascal source code.

Deadline: Mart 10, 2017 7:00, Eindhoven time.

This assignment in PDF.

Software Modernisation

Software modernisation is a process of evolving an existing software system to more modern platforms or technologies. Modernisation is usually carried out in order to reduce the maintenance costs, increase flexibility or improve performance. However, as any large scale software reengineering effort, modernisation tends to be error-prone and costly. In the following assignment you will design a tool that can support one specific kind of software modernisation effort: replacement of weakly typed containers with generic collections, e.g., *Map documents = new HashMap();* will be replaced by *Map<Integer, Document> documents = new HashMap<>();* To perform this task you will use the Object-Flow Graph discussed during the lecture.

Generic containers

Starting from Java 5 (2004), Java supports generics, i.e., classes that can be parameterised by other classes. For instance, in the following example [**Wikipedia** (https://en.wikipedia.org/wiki/Generics_in_Java)] the third line results in a run-time error: indeed, the compiler does not know that the list contains a string. However, during the execution, the attempt to interpret this value as an integer fails.

```
List v = new ArrayList();
v.add("test");
Integer i = (Integer)v.get(0);
```

If the developer can indicate that the list should contain strings, then the compiler can flag the last assignment as erroneous:

```
List<String> v = new ArrayList<>();
v.add("test");
Integer i = v.get(0);
```

Java 7 (2011) has further simplified the usage of Java generics by introducing the diamond operator <>. While earlier versions of Java required

```
Entry<String, String> grade = new Entry<String, String>("Mike", "A");
```

more recent ones also support

```
Entry<String, String> grade = new Entry<>("Mike", "A");
```

Many of the **java.util collections** (https://docs.oracle.com/javase/7/docs/api/java/util/package-summary.html) make use of generics.

Deriving type parameters from the OFG

Using the OFG you can infer that *Library.loans* contains information about objects of class *Loan*: *out(Library.loans) = {Loan}*. Hence, one should be able to replace *Collection loans = new LinkedList();* in the class *Library* with *Collection<Loan> loans = new LinkedList<>();*

Similarly, one can infer that *Library.users* contains information about objects of class *User*: *out(Library.users) = {User}*. However, *Library.users* is defined as a *HashMap*, and therefore it should have two type parameters. The first type parameter records the types of the keys in the *HashMap*, the second one - the type of the values to be added. In the eLib case line 12 of *Library.java* reads *users.put (new Integer(user.getCode()), user)* suggesting that the type parameters should be *Integer* and *User*, respectively. Hence, *Map users = new HashMap();* should be replaced

Finally, field and variable declarations are not the only situations where weekly typed container class can be used. You should identify additional case(s) when type parameters should be introduced and suggest appropriate replacements.

Tool

The tool should accept a Java program a produce a list of suggestions how should the program be corrected. Each suggestion is a pair: **Rascal location** (http://tutor.rascal-mpl.org/Rascal/Expressions/Values/Location/Location.html) and the modified code fragment. The list of suggestions should include all locations in the eLib source code where a weekly typed container class is used.

To develop the tool please use the **Rascal implementation** (https://github.com/cwi-swat/rascal-OFG), kindly provided by Davy Landman and Jurgen Vinju (CWI). Please consult the following page on **getting started with Rascal** (http://www.rascal-mpl.org/start/) (you need to use the *stable* version of Rascal unless you intend to port the Rascal implementation to the new API). Object-Flow-Graph is described in Chapter **2** (http://www.springerlink.com/content/g565610w346v15x1/fulltext.pdf) of **Reverse Engineering of Object Oriented Code** (http://link.springer.com/book/10.1007/b102522) by Paolo Tonella and Alessandra Potrich. The implementation provided by Davy Landman and Jurgen Vinju converts a regular Java program to the flow program; slides of Jurgen Vinju show how the conversion from the flow program to the Object Flow Graph can be done. You need to complete this first prior to implementing the modernisation step.

To evaluate the tool you should apply it to the **eLib example**, discussed during the lecture. To ensure that the suggestions generated by the tool are correct, you should apply them to the source code and check that the modified code compiles and runs as expected.

To be submitted

You should submit a **zip file** containing

- the **report** (PDF, see further), please ensure that the report meets the English level requirements stated in **Assignment 1**;
- the Rascal **source code of the tool** you have developed. Please note that I will look at your code and run it, if necessary, on additional software systems.

Report

1. Introduction. State the purpose of the assignment
2. Assumptions. Describe any assumptions made about the language of the input programs (do you support, e.g., inner classes or lambda expressions?)
3. Tool. Describe the tool you have designed:
   1. present an architecture overview, indicate responsibilities of each component as well as whether the component has been developed by you or obtained elsewhere,
   2. describe the size of your tool (number of files, number of lines of code)
   3. explain how the tool should be run: I should be able to reproduce your results on my machine. You can assume that I have Java, Eclipse and Rascal installed; all the rest should be explained.
4. Results
   1. list the suggestions produced by the tool
   2. discuss performance of the tool (measure the execution time: would it be plausible to use the same technology on a real world Java software?)
5. Discussion.
   1. Based on applying your tool to eLib identify strengths and weaknesses of the approach chosen.
   2. Further applications. You have seen two applications of the Object-Flow-Graph intended to assist developers in understanding and implementing software evolution, namely class/sequence diagram reconstruction and replacement of weekly typed containers with more modern generics. Propose at least one additional way how the Object-Flow-Graph can be used to help software developers, and discuss how would you implement it.
6. Conclusions.

Grading

The grading will be based on the report and the source code submitted. I will also test the source code on other software projects, should it be necessary.

- 1 - introduction, conclusion, general presentation, English
- 0.5 - discussion of the input language
- 6 - tool design, implementation and description
- 1 - application of the tool to eLib
- 1.5 - discussion

Feedback