Source Code Analysis

Munster Technological University

2023

1 Lab Assignment

Extend your project from Lab2 and create a new package and call it Lab3 and add all the required code for this lab in that package.

This lab has two exercises as follows:

1.1 1) Source Code Dependency

Strong or tight source code dependency is not good and it should be avoided as much as possible.

The purpose of this lab exercise is to discover the dependencies between any pair of classes based on their methods invocations. E.g., if *Class A* has *method 1* that is called 7 times by methods in *Class B*, then the dependency between Class A and Class B may be defined as degree 7.

1. Use Symbol Solver and visitor class for JHotDraw project in order to detect all the method invocations that both the caller and callee methods are *static*. Identify the class container of each method (Callee and Caller) and ignore the cases where the class containers of Callee and Caller are the same.

2. The number of method invocations for each pair of classes is counted and stored in an appropriate data structure e.g., HashMap, where the keys are the concatenation of classes' names (e.g., "Class1 -⟩ Class2") and values are the number of method invocation between them (see the skeleton code). Visualize the data using a Pie chart. See Figure 1.

A skeleton code with some code is provided for you. You may work on this code (or you can develop your code from scratch). You will only need to work on the parts that are marked as
TODO

What is your interpretation of the result. Note: For this week, you are required to share the visitor class (**MethodCalls**) once it is completed (see the skeleton code in Canvas) in the discussion forum and provide feedback for your peers as well.

1.2 2) Inheritance

Another measure to identify the class dependencies is the amount of class inheritance. This exercise is going to discover those classes that have more than 4 super-classes in total.

1. Use Symbol Solver on JHotDraw project in order to detect all the classes and their supper classes where all supper classes are part of the source code (AST).

2. For each class, count the number of super-classes and if it is greater than 4, store the name of the class and the number of super-classes in an appropriate data structure and visualize it using a Pie chart. See Figure 2

Farshad Ghassemi Toosi
farshad.toosi@mtu.ie

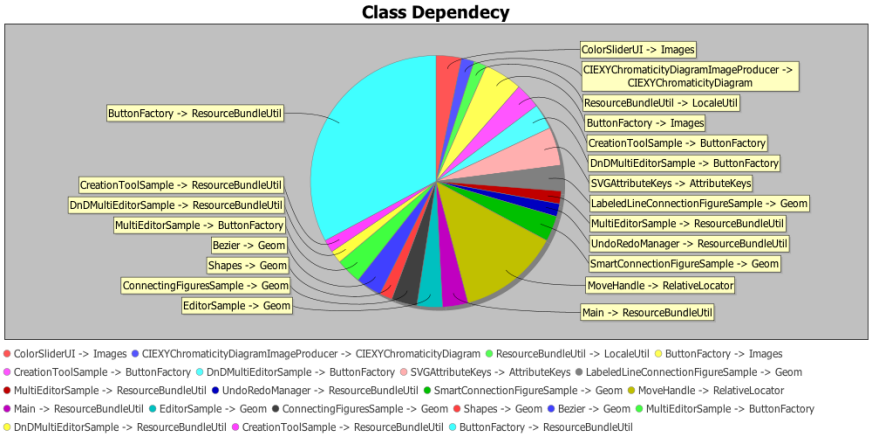

Figure 1: Visualization sample to illustrate the pairs of classes and their dependencies based on method invocations.

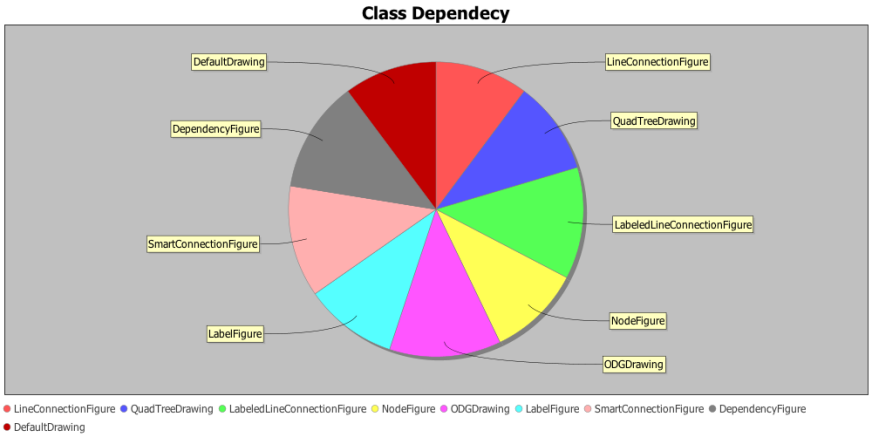

Figure 2: Visualization sample to illustrate the classes based on their number of superclasses. E.g., the bigger chunks are for those classes that have larger number of superclasses.

Farshad Ghassemi Toosi
farshad.toosi@mtu.ie