University of Glasgow | School of Computing Science

Assessed Coursework

Course Name	**Object Oriented Software Engineering**			
Coursework Number	**2**			
Deadline	**Time:**	**4:30pm**	**Date:**	**19th March 2021**
% Contribution to final course mark	**10%**			
Solo or Group ✓	**Solo**	**solo**	**Group**	
Anticipated Hours				
Submission Instructions	**Submit via Moodle**			
Please Note: This Coursework cannot be Re-Assessed				

Code of Assessment Rules for Coursework Submission

Deadlines for the submission of coursework which is to be formally assessed will be published in course documentation, and work which is submitted later than the deadline will be subject to penalty as set out below.

The primary grade and secondary band awarded for coursework which is submitted after the published deadline will be calculated as follows:

(i) in respect of work submitted not more than five working days after the deadline
   a. the work will be assessed in the usual way;
   b. the primary grade and secondary band so determined will then be reduced by two secondary bands for each working day (or part of a working day) the work was submitted late.

(ii) work submitted more than five working days after the deadline will be awarded Grade H.

Penalties for late submission of coursework will not be imposed if good cause is established for the late submission. You should submit documents supporting good cause via MyCampus.

Penalty for non-adherence to Submission Instructions is 2 bands

You must complete an "Own Work" form via https://studentltc.dcs.gla.ac.uk/ for all coursework

Object Oriented Software Engineering
Assessed Exercise 2
Visitors Design Pattern with JavaParser

Objective

In this coursework, you will learn how to use the software framework called `JavaParser` and its documentation to build a static analyser. You will also understand how the visitor design pattern is applied in the implementation of such framework. This objective will be evaluated based on your ability to extend the `VoidVisitorAdapter` in `JavaParser`. This is to implement concrete detectors that traverses an Abstract Syntax Tree representation of a program code using the visitors design pattern. [**Total Marks: 100, Task 1: 60 marks, Task 2: 30 marks, Deliverable form: 10 marks**]

Setup

Create a simple maven project without archtype using eclipse IDE. For Eclipse 2020-12 you can navigate to File, New, Other..., Maven, Maven Project:

1.
   - Check create a simple project (skip archetype selection).
   - Enter Group Id = oose.coursework2
   - Artifact Id= coursework2
   - Name = coursework2
   - Click on pom.xml file generated and Include the following dependency:

```
<dependencies>
   <dependency>
        <groupId>com.github.javaparser</groupId>
        <artifactId>javaparser-core</artifactId>
        <version>3.12.0</version>
   </dependency>
</dependencies>
```

2. Create a package called `detectors` to contain the four java classes for Tasks 1 and 2 highlighted below.

Task 1: (60 Marks)

A control flow statement in a method is characterised as a **useless control flow** when the control flow continues onto the same place regardless of whether or not the branch is taken. For example, having an empty statement block for an `if` statement:

```
if (argv.length == 1);
System.out.println( argv[0]);

if (argv.length == 0) {
   // TODO: handle this case
}
```

Other forms of control flows should be also considered, namely `for`, `while`, `do... while` and `switch`.

Using the visitors design pattern, create the class `UselessControlFlowDetector.java` that detects useless control flow in java program code. Create a container class called `Breakpoints.java` to collect this observed behaviour as the `visit` method(s) implemented in detector transverses through the program structure. Finally, implement the class `Driver.java` that instantiates the detector and provides it with program code to operate on. `Driver.java` should print each detected pattern to command line in the format:

`Useless Control Flows:`
`className` =?, `methodName` =?, `startline` =?, `endline` =?
`className` =?, `methodName` =?, `startline` =?, `endline` =?
...

Task 2: (30 Marks)

Recursion is a programming technique of making a function calling itself. **Polymorphic recursion** is a property of a program code where the parameter types of a method may change with each recursive invocation. For example, the methods named `method1` in `Test.java` below implement polymorphic recursion. Here, both recursion and polymorphic recursion should be detected, independtly of the number of parameters and their types.

```
class Test{
   int x;
   public void method1(String args[]){
      method1(args);
```

```
    }
    public void method1(String args[],int x, Test t){
       this.x =1;
       method1(args);
    }
}
```

Using the visitors design pattern, create the class `RecursionDetector.java` that detects polymorphic recursion in java program code. Use the container `Breakpoints.java` to collect observed recursive behaviour as the `visit` method(s) implemented in detector transverses through the program structure. Also extend `Driver.java` to instantiate this detector and provide it with program code to operate on. Finally, the output of `Driver.java` for detected behaviour should be similar to Task 1, by printing the class and method name, and also start and end line of observed pattern to command line. in the format:

`Recursions:`

`className` =?, `methodName` =?, `startline` =?, `endline` =?

`className` =?, `methodName` =?, `startline` =?, `endline` =?

...

Hints

- You may need to reference JavaParser's documentation which is available at: `http://www.javadoc.io/doc/com.github.javaparser/javaparser-core/3.12.0`
- You may find the tool `AstVisualizer` - for visualising Abstract Syntax Trees of java programs useful for understanding the structure of that matches a desired behaviour. This tool can be downloaded from OOSE Moodle webpage.
- Finally, we will use `Calculator.java` that is available on Moodle to test your solutions. For example, we may refactor `Calculator.java` to introduce recursion or change the control flow to match a desired pattern. We assume there are no nested classes.
- We will also check the clarity of your solution, documentation and the extent it satisfies the SOLID software design principles.

Deliverables

Submission should be made electronically via Moodle for Assessed Coursework 2.

1. Attach your zipped maven project with your solutions to Task1 and Task2. State your name and registration number at the top of `UselessControlFlowDetector.java`, `RecursionDetector.java`, `Breakpoints.java` and `Driver.java`.

2. A runnable jar file of your project that is able to be executed from command line with the file path and name passed as argument (**10 marks**). For example:
   java -jar coursework2.jar "C:\...\calculator.java"

Assessment

Submission is due by **16:30** on **19th March 2021**. You should submit your solution through the class Moodle page.

Tutors and demonstrators will be in your allocated Lab to offer assistance. You should endeavour to attend each lab so you can get the necessary help.

As per the Code of Assessment policy regarding late submissions, submissions will be accepted for up to 5 working days beyond the due date. Any late submissions will be marked as if submitted on time, but reduced by 2 marks for each additional day. Submissions received more than 5 working days after the due date will receive an H (band value of 0).

Dr Fani Deligianni (fani.deligianni@glasgow.ac.uk), 321b Sir Alwyn Williams Building, School of Computing Science, University of Glasgow