ECS 160 Assignment 2

Due by Monday, Nov 4 at 8 pm

There are 2 parts to the assignment. Credit as noted. When you submit, create a zip file of your entire maven project folder. It should include Java programs, with classes and methods with names as indicated. Each part of HW2 is a (partial) grader for HW1 (implemented as a VISITOR).

We will run your HW2 solutions on several (correct & incorrect) solutions to HW1, and check the output with a JUnit test case. You will be provided with an example JUnit test file (**TestHW2.java**) which tests if the grader correctly grades a correct implementation of HW1 (which we provide in a zipped Eclipse project: **ECS160HW1Fall2019.zip)**. Your method and class names for the visitors should exactly match those in the example test file. Additionally, the visitors called from the JUnit tests should have a static factory method that:

a) sets up the Eclipse AST (similar to the block of code in App.java from the example).
b) creates a new instance extending ASTVisitor, performs the visit on the file(s), and returns the objects after extracting whatever information is needed for grading.

Then, for each question part you will have a separate method to grade that part (see the JUnit file for details on the expected interface). Also, **DO NOT** use static member variables in your visitors. Your code may pass on the test cases provided, but it will fail on larger test sets. You should also write your own tests to check if the visitors correctly grade *incorrect* versions of HW1 (see the questions below for expected behavior).

Our tests will require a full, correct parse of Java: so you will HAVE to use the ASTNode and ASTVisitor framework provided by Eclipse, similar to the examples provided on Canvas and shown in discussion. Tricks like “grep” won’t work on our test cases.

To set up the provided JUnit tests, take the Maven project demonstrating visitors for the AST shown in the discussion (see **DiscussionNotes/Discussion3ASTVisitor/visitorexample.zip**) and place the JUnit test file in the **ecs160.visitor.visitorexample** package under **src/test/java**. Then you should take the provided HW1 reference solution, unzip it, and put it in your eclipse workspace. You may use the utility functions in the **ecs160.visitor.utilities** package for reading in the source files and printing out information about the AST, along with the ASTView plugin demonstrated during discussion.

You should do the homework by yourself. Make sure you understand the concepts. The assignment grade will be the geometric mean of the score on the homework, and the (possible) related pop quiz, which will be given in the lecture. If you do the homework yourself, you will find the related quiz easy, otherwise, it will be well-nigh impossible.

This homework will be evaluated using test cases only, with some occasional spot checks for conformance to VISITOR patterns. In addition, we will be running tools to check for plagiarism.

1. (SingletonCheckerVisitor.java, package ecs160.visitor.astvisitors) Implement a VISITOR that grades a SINGLETON Pattern instance as indicated. You will create a static factory method **setUpGrader(String sourceFile, String className)** which takes the source code file and the name of the singleton class as arguments to parse the AST and record any information you need using the VISITOR pattern. After performing the required action, the factory should return the Visitor. Then implement the following methods to return grading results **after** parsing the file with a visitor:
   a. (**boolean gradeA()**) has a private constructor (2 points)
   b. (**boolean gradeB()**) has a public static method that returns an instance of the class type. (2 points)
   c. (**boolean gradeC()**) a private static instance variable, of the type of the class (3 points)
   d. (**boolean gradeD()**) The public static method calls the private constructor exactly once. Furthermore, this call should be inside an if statement (3 points). (Hint: Look at the ASTNode type ClassInstanceCreation).

   You can assume field declarations come before any method declarations in any files you are testing.

2. (StateCheckerVisitor.java, package ecs160.visitor.astvisitors) Implement a VISITOR that grades STATE Pattern instances. Your static factory method **setUpGrader(String contextPath, String contextName, String abstractPath, String abstractName)** will take four arguments: a CONTEXT source file path and class name, and ABSTRACTSTATE source file path and class name. Find the set of method names of each class, take the intersection of this set. For each method in the intersection, ensure that a method with that name in the CONTEXT class calls a method of the same name in ABSTRACTSTATE. (similar JUnit tester below as above). Once again, you will use the VISITOR pattern in the static method and return stored grading results in two methods:
   a. (**boolean gradeA()**) checks that the CONTEXT has matching method names for each method in ABSTRACTSTATE. (5 points)
   b. (**int gradeB()**) checks that each of the CONTEXT methods calls the correct ABSTRACTSTATE method. It should return an integer that matches the number of methods correctly matched. (5 points)

Finally, the Javadoc documentation for the Eclipse AST can help you figure out what types of ASTNodes exist and methods they support. The following 2 links are a good starting point:
ASTNode
https://help.eclipse.org/neon/index.jsp?topic=%2Forg.eclipse.jdt.doc.isv%2Freference%2Fapi%2Forg%2Feclipse%2Fjdt%2Fcore%2Fdom%2FASTNode.html
ASTVisitor
https://help.eclipse.org/neon/index.jsp?topic=%2Forg.eclipse.jdt.doc.isv%2Freference%2Fapi%2Forg%2Feclipse%2Fjdt%2Fcore%2Fdom%2FASTVisitor.html