Assignment: State-Based Testing

This assignment is to be done in teams of TWO students. You are required to sign up to one of the assignment groups in myCourses. Keep in mind that two students who are on the same team for the assignment are not allowed to be on the same team for the project. Your team is required to hand in a ***single zip file*** via myCourses containing your report as well as your implementation as described below. If you realize that you need to make changes to your submission, do not resubmit only the file(s) that have changed, but rather resubmit another complete zip file. If you are using an application other than MSWord for your report, convert your report first to either a PDF file or a DOC(X) file.

Each team member must make contributions to the assignment. A team member who does not contribute to the assignment receives a mark of 0 for the assignment. A team member may optionally email a confidential statement of work to the instructor before the due date of the assignment. A statement of work first lists in point form the parts of the assignment to which the team member contributed. In addition, the statement of work also describes whether the work load was distributed fairly evenly among the team members. A statement of work may be used to adjust the mark of a team member who is not contributing sufficiently to the assignment. It is not necessary to send a statement of work, if a team distributed the work for the assignment fairly evenly and each team member contributed sufficiently.

Description

Automation of testing is a key factor for improved productivity in software development. You are asked to automate the first two steps of the N+ Test Strategy for state machines:

- Automatically derive the round-trip path tree from a state machine, and then
- Automatically generate a JUnit 4 test class that contains the conformance test cases for the state machine.

For each state machine, there exists an ***xml file*** describing the state machine and a ***class*** that implements the state machine. The goal is to test the implementation based on the state machine with the help of conformance test cases using the N+ Test Strategy.

To import the ***ConformanceTest.zip*** file into Eclipse, use the *Import* feature of Eclipse (Import – General – Existing Projects into Workspace – Select archive file).

XML FILE: The Eclipse project ***ConformanceTest*** (***ConformanceTest.zip***) provides the ability to read the xml file (see package ***persistence*** of ConformanceTest) and manipulate its content in memory (see package ***statemodel*** of ConformanceTest). When the xml file is read, instances of the classes in the statemodel package are created. You are required to handle only simple state machines that conform to the language definition (metamodel) in Figure 1. The classes in the statemodel package of ConformanceTest conform to this metamodel. You are not allowed to change these classes.

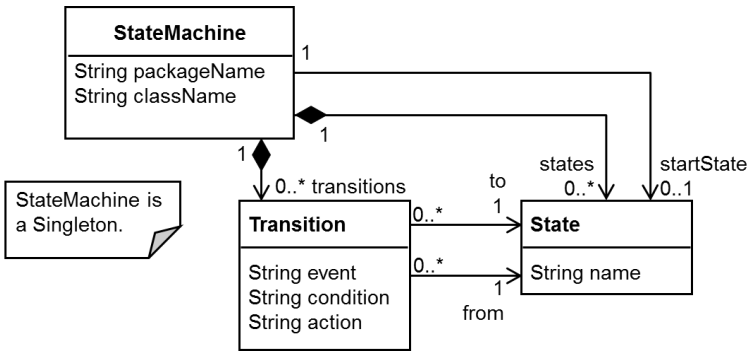

Figure 1. Metamodel of Simple State Machine

The state machine defined in ***ccoinbox.xml*** (see root folder of project ConformanceTest) is also shown in Figure 2. Note that *@ctor* identifies the constructor of the class.

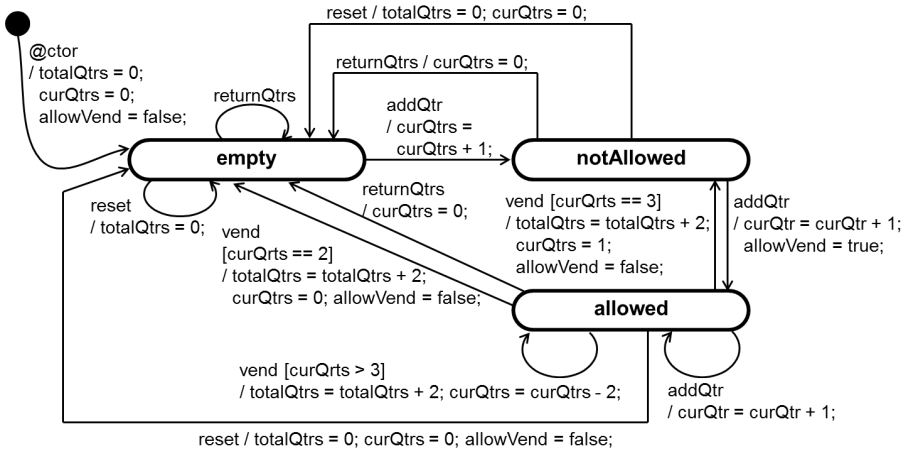

Figure 2. State Machine of CCoinBox

The state machine defined in ***legislation.xml*** (see root folder of project ConformanceTest) is also shown in Figure 3. Note that *@ctor* identifies the constructor of the class.

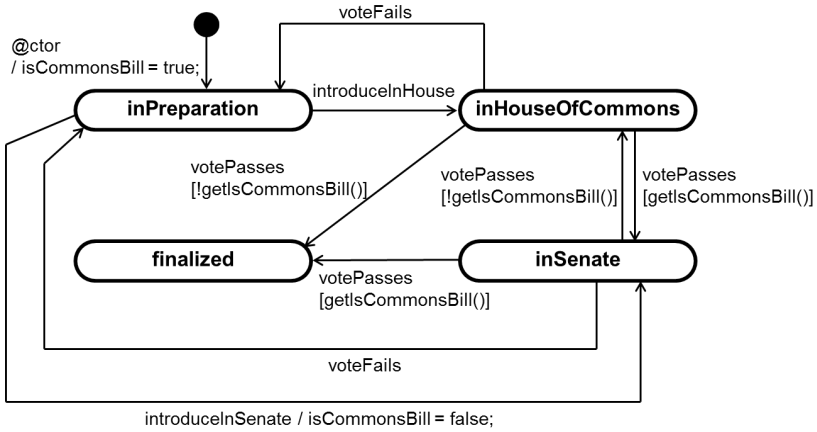

Figure 3. State Machine of Legislation

CLASS: You may assume that any class that implements a state machine conforming to the above metamodel provides an API as defined for the CCoinBox and Legislation examples (see ***CCoinBox.java*** in package ***ccoinbox*** and ***Legislation.java*** in package ***legislation*** of ConformanceTest). Note that *state reporter methods* are already available for the examples. You may use the existing state reporter methods or you may add your own state reporter methods, but you are not allowed to change anything else in a class that implements a state machine.

Evaluation

Your submission will be assessed by how well you are handling the CCoinBox example and another unknown state machine that conforms to the metamodel in Figure 1 (the class implementing the unknown state machine provides a similar API as the two provided examples). The Legislation example is only provided to give you further input for testing your test generation code. There are no defects in the class implementing the Legislation state machine. Your implementation will be judged by the number of compilation errors in your generated test class (the fewer, the better) and by how much of the test class is actually automatically generated (the more, the better). You are required to automatically generate the package definition, the imports, the class definition, and ***all*** conformance test cases including the code corresponding to the events and the checking of the resulting state. For bonus points, you may opt to also generate the setUp and tearDown methods, and the code corresponding to checking of conditions and results of actions. You are ***not*** required to generate sneak path test cases.

Submission

You are required to hand in a zip file of the original ***ConformanceTest*** Eclipse project with your implementation and report added to it as described below. To create the zip file, use the *Export* feature of Eclipse to create an *Archive File* (Export – General – Archive File) of the project.

Part 1 – Source Code

You have to hand in the following:

- The source code that generates the test class for the CCoinBox example given the state machine definition and implementation of the state machine.
- The result of the test class generation for the CCoinBox example without any manual changes after generation. This class must be called ***GeneratedTestCCoinBox.java*** and saved in the same package as the implementation of the state machine.

Part 2 – Complete Test Class

You have to hand in the following:

- The complete test class for the CCoinBox example with additional code added manually as needed to fully test the CCoinBox state machine based on the N+ Test Strategy (conformance tests only). This class must be called ***TestCCoinBox.java*** and saved in the same package as the implementation of the state machine. Any manual changes have to be clearly identified in the

complete test class. Any complete test class that cannot be executed as a JUnit test will result in a mark of 0 for this part.

Part 3 – Report (1-2 pages at the most)

The report must be called ***ConformanceTestReport.pdf/doc(x)*** and added to the root folder of the ConformanceTest project. Clearly state the course, term, group number, and team members on the title page of your report. You have to hand in a report containing the following sections:

- Describe how to run your source code to generate the test class for a given state machine (xml file) and corresponding implementation of the state machine. This description should work for the CCoinBox example but also for the unknown state machine and its implementation.
- In a few paragraphs, discuss which manual changes you had to make to the generated test class to get the complete test class and why you had to make those manual changes instead of automatically generating the test code.
- In one paragraph, describe whether you found any defects in the implementation of the CCoinBox example. For each found defect, describe how you fixed it.
- In one paragraph, discuss what would be the main challenge to automate the generation of sneak path test cases from a given state machine conforming to the metamodel in Figure 1.

Marking Scheme

Part of Assignment	***Marks***
Part 1 – Source code that generates the test class for CCoinBox (no manual changes)	35
a) compilation errors	5/35
b) package definition	1/35
c) imports	2/35
d) class definition	1/35
e) required number of conformance test cases (covering all paths through the round-trip path tree)	10/35
f) conformance test cases (event / check of resulting state)	16/35
Part 2 – Complete test class for CCoinBox (including manual changes)	25
Part 3 – Report	20
a) How to run your source code	2/20
b) Discussion of manual changes	8/20
c) Discussion of any defects in CCoinBox	6/20
d) Discussion of generation of sneak path test cases	4/20
Generation of test class for unknown state machine	20
Total Marks:	100
Bonus Point:	Up to +15
The total mark may be adjusted based on the actual contributions of a team member to the assignment.	