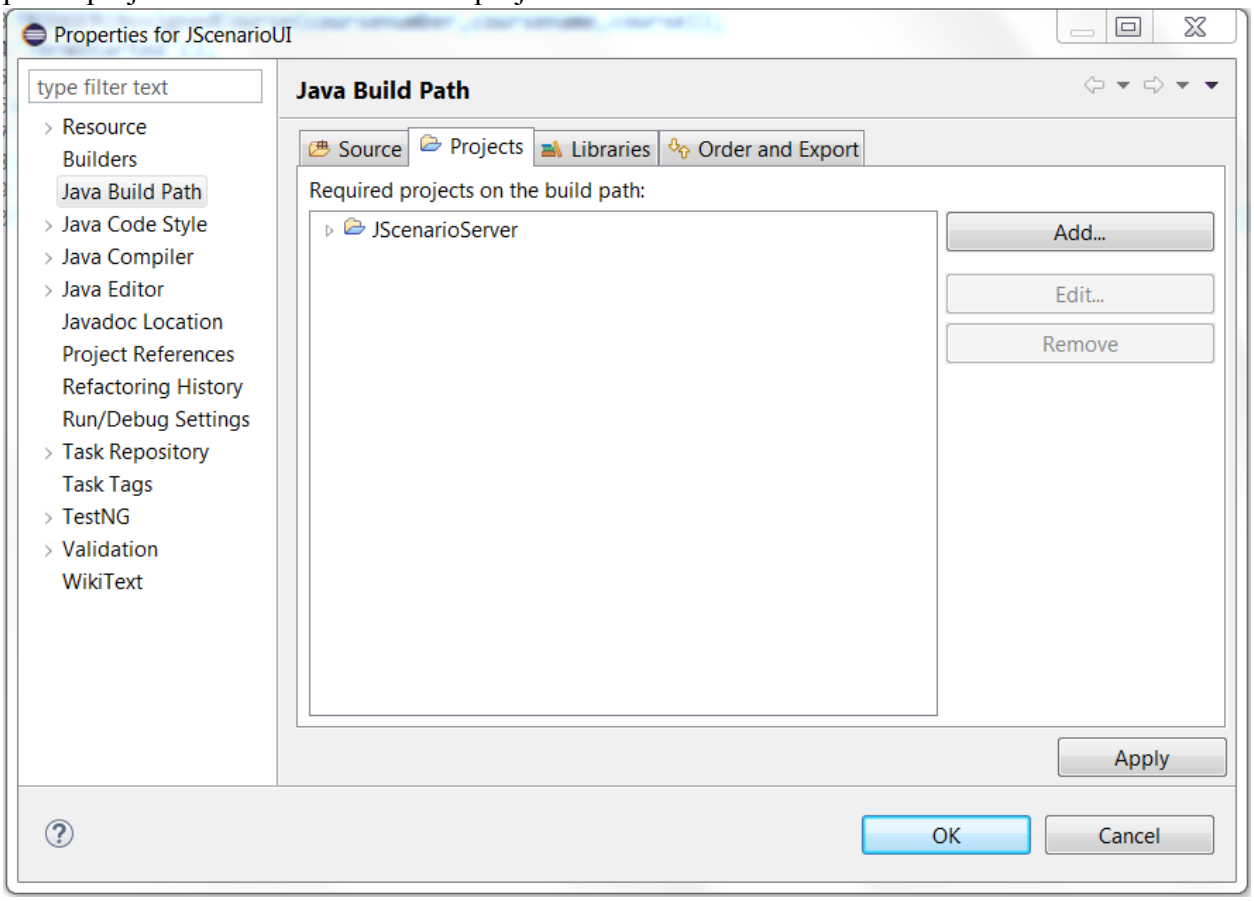
JScenario Guide

1. Installation

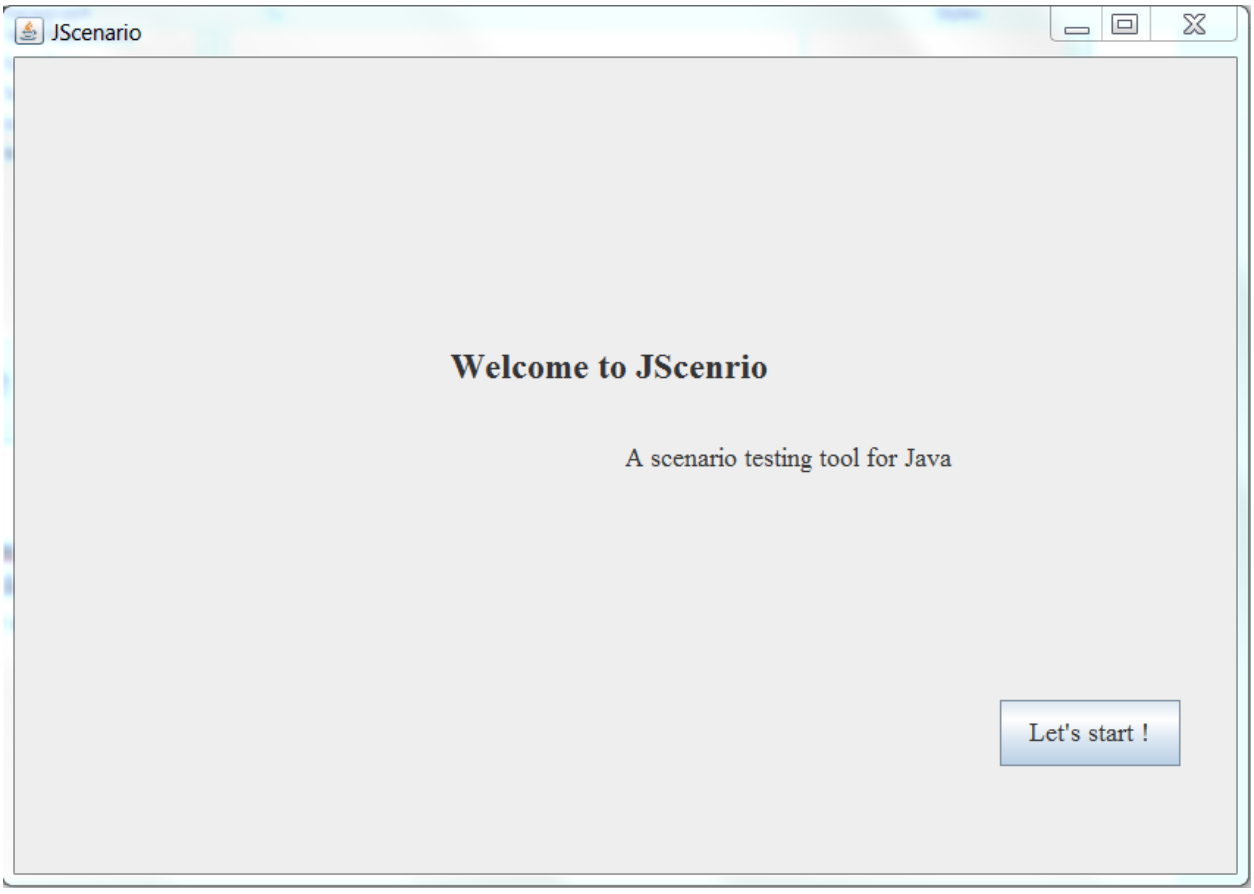
- Prerequires:
  1) Have JDK correctly installed
  2) Have eclipse installed

- Steps:
  1) Import the two source code folders into your workspace (namely: JScenarioServer and JScenarioUI)
  2) Put the IUT that you want to test under folder JScenarioServer->src->IUT
  3) In eclipse, right click on the imported JScenarioUI project: build path->configure build path->project->add JScenarioServer project



2. Run

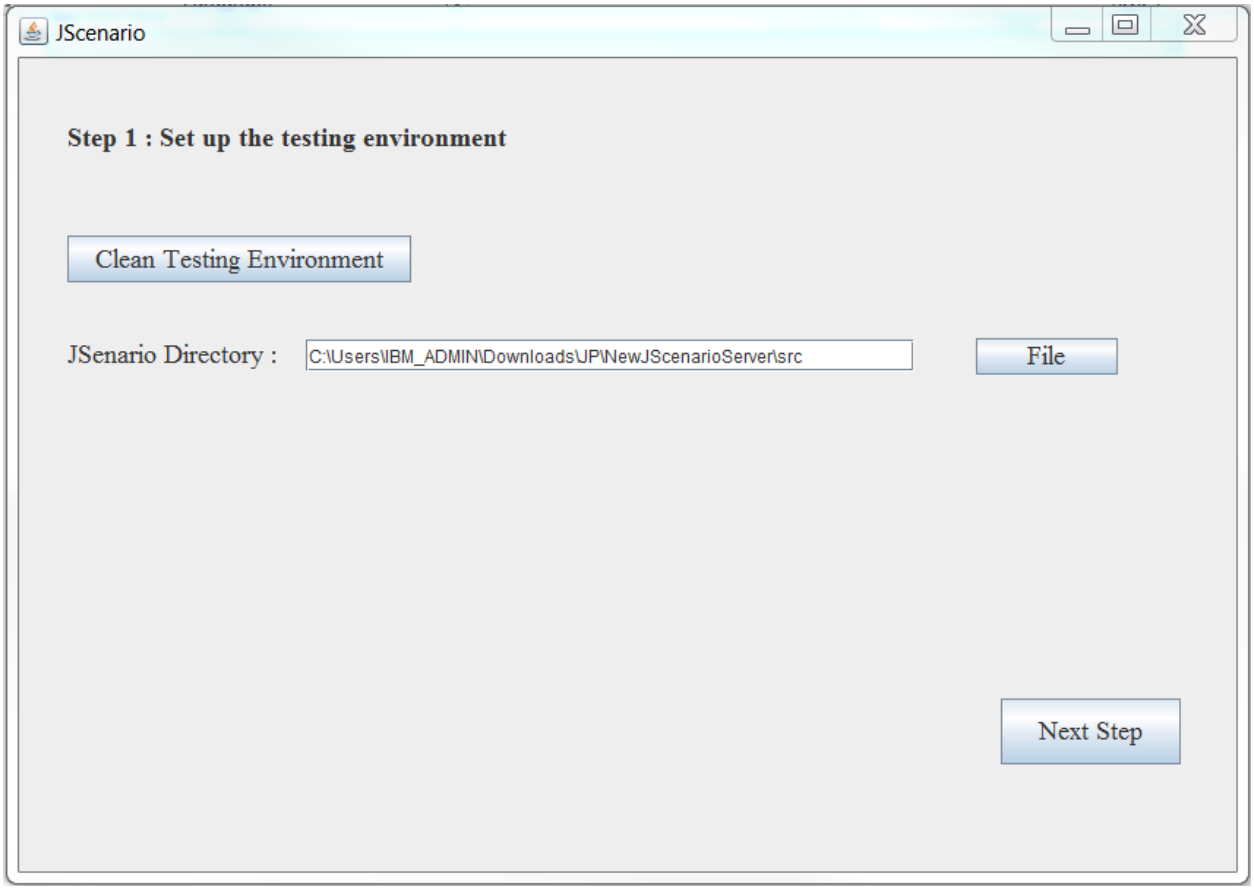
- Prerequisitess:
  In eclipse, go to JScenarioUI->src->layout->JScenario->run as Java Application
  (If any error dialog shows up, just click “continue” or “yes”)



Click “Let’s start!” to start…

1) Set up the testing environment

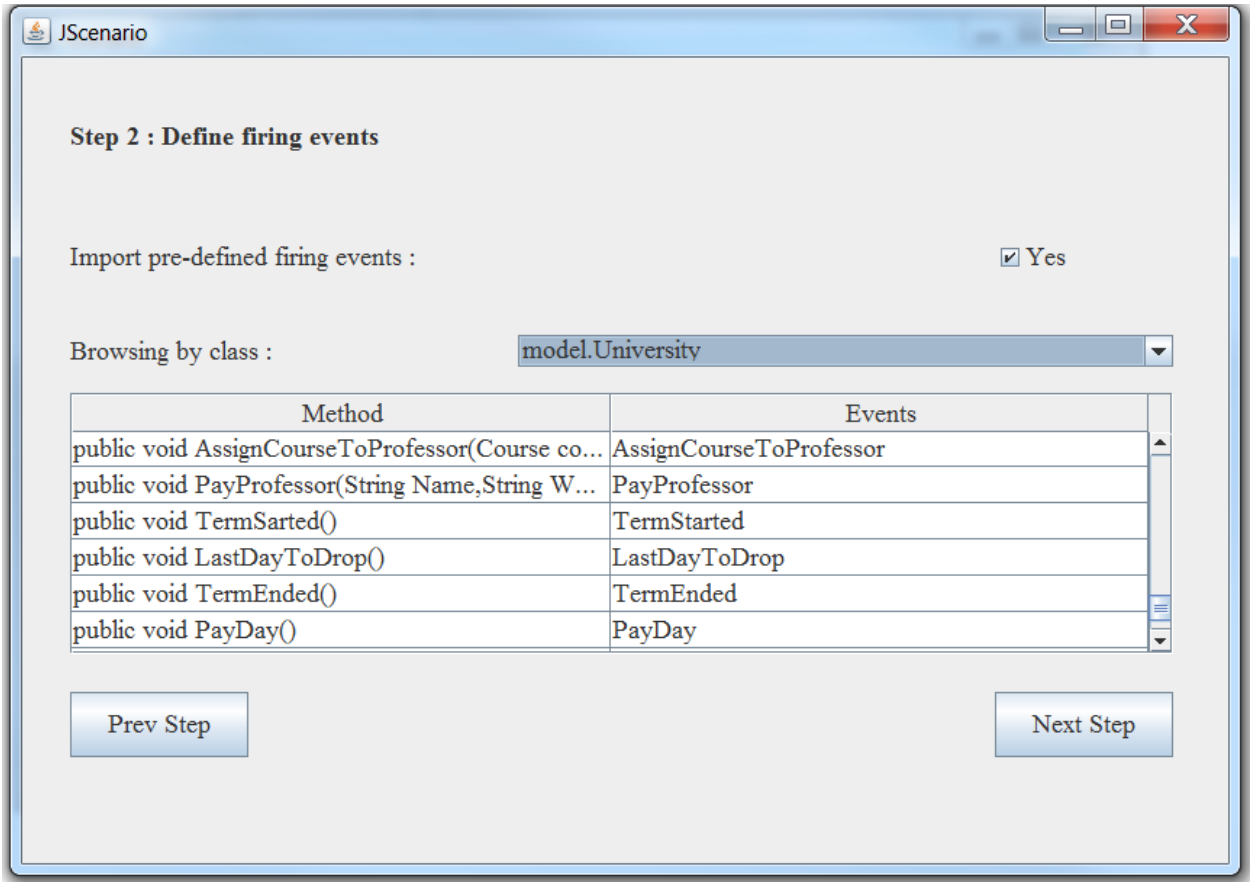
The first step is to set up the testing environment, as the image shows:



- Click “Clean Testing Environment” to clean all the metadata generated from previous uses. If you are testing a brand new IUT,you can click this to clear all the metadata. But if you have some scenarios already defined from previous usage and you want to continue using them, **do not click this**.
- Click “File”: this will open a document selector, and you need to use this to find the source code directory of your JScenarioServer project (where your JScenarioServer is). You can also manually input the directory in the text field.
- After these two steps, click “Next Step”.

2) Define firing events

The second step is to bind the events you want to fire with the actual java methods of the IUT. The checkbox for "Import pre-defined firing events" is by default on "Yes". If this box is checked, the tool will search through your IUT and find all pre-defined firing events. You can filter the event list by class, in which case the table displayed will present a list of defined firing events for that specific class (which is part of what we call the 'model', ie. the IUT). If you uncheck the "Yes" checkbox, the tool will open View 2 show later in this document.



View 1

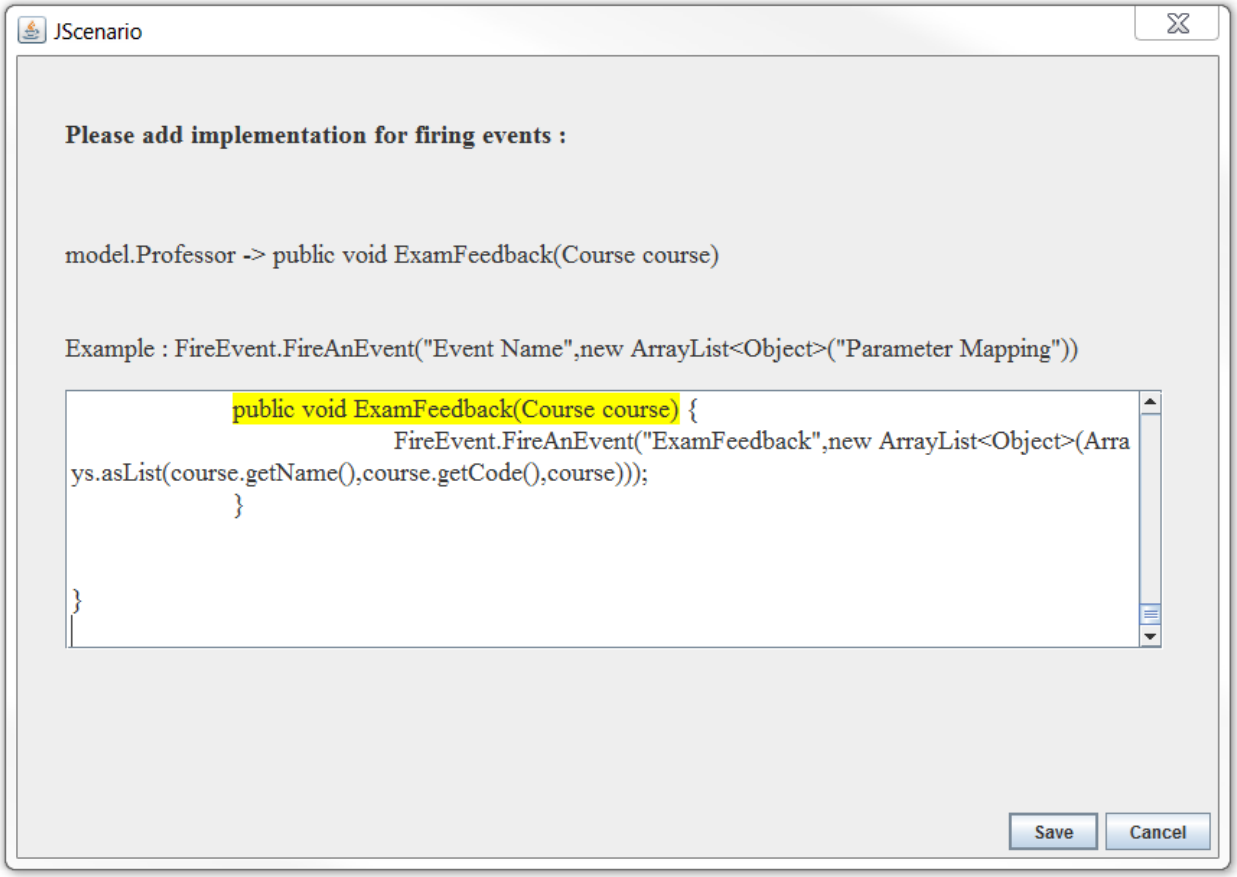
With View 1, you can:

- Uncheck the "Yes" to open view 2.
- Click "PrevStep" to go back to set up the testing environment.
- Use the dropbox to filter the firing events list by class name in your IUT.
- Click on any entry in the table at which point the tool will open an editor view (see below) for you to insert/modify the corresponding firing event statement in the selected method. The editor will highlight the signature (ie name and parameter list) of the method in which you want to insert or modify a firing statement. You then carry out your editing directly in that method's body. For the exact syntax rules for firing statements, please

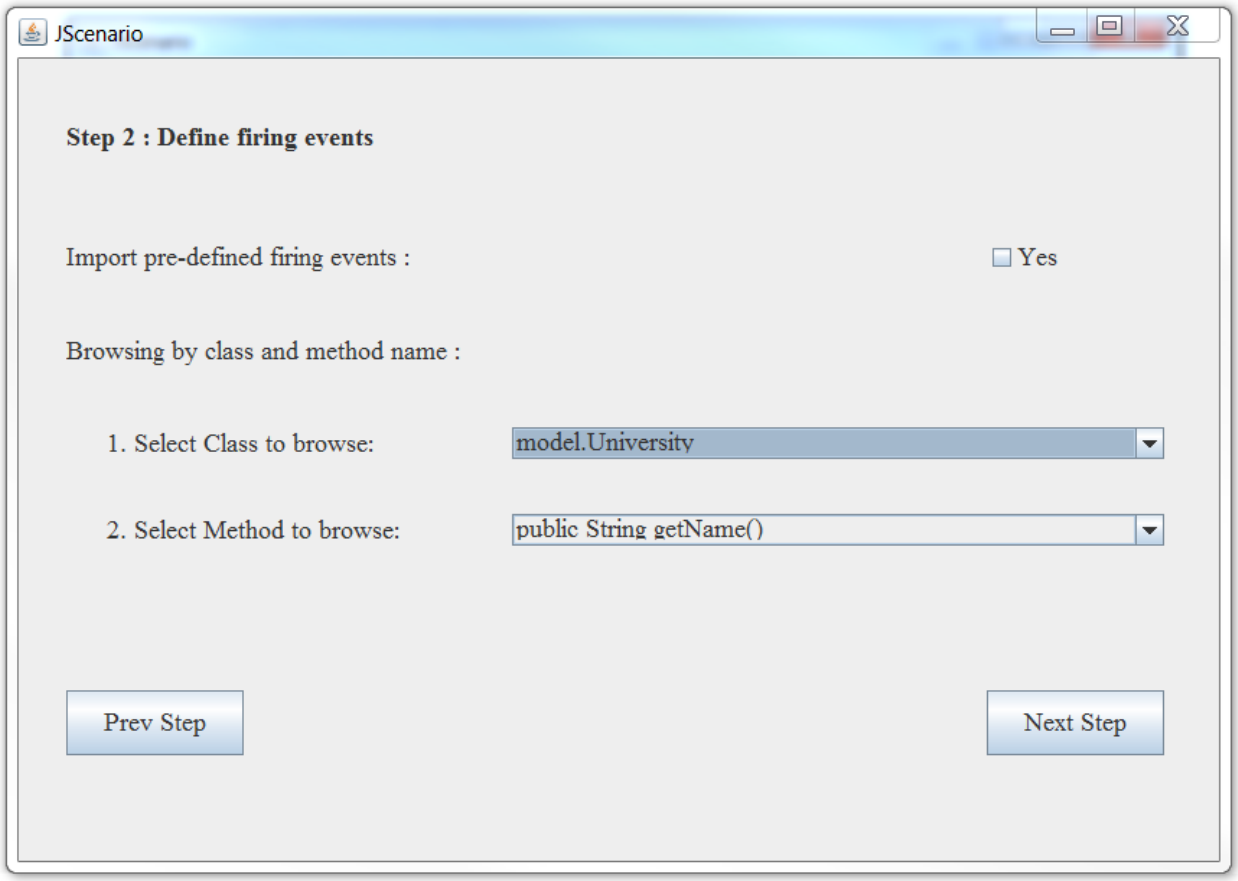
refer to part 3 of this guide. Click “Save” to save the updated method. If you then go to eclipse to check that class’s source code, you should see the most up-to-date version of that method and of its firing statement.

- In View 1, click “NextStep” once you have finished ALL event binding.

Terminology: The *model(s)* are the actual java classes of an IUT. The *contracts* are the java classes generated from (and corresponding to) the actual java classes of an IUT. Think of these contracts as the IUT classes *instrumented* (ie modified to include code) for the JScenario framework. The tool generates these contracts.



Editor View



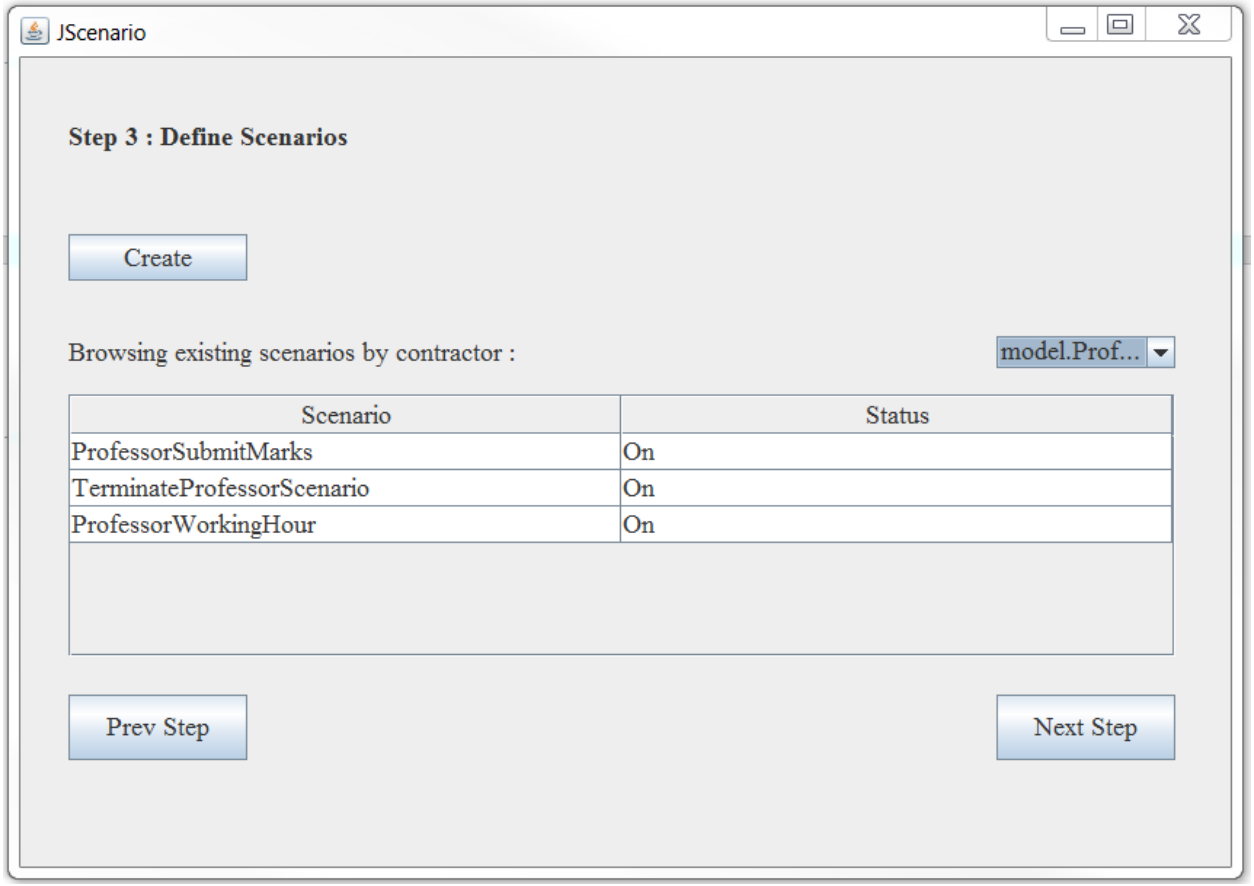
View 2

With View 2, you can:

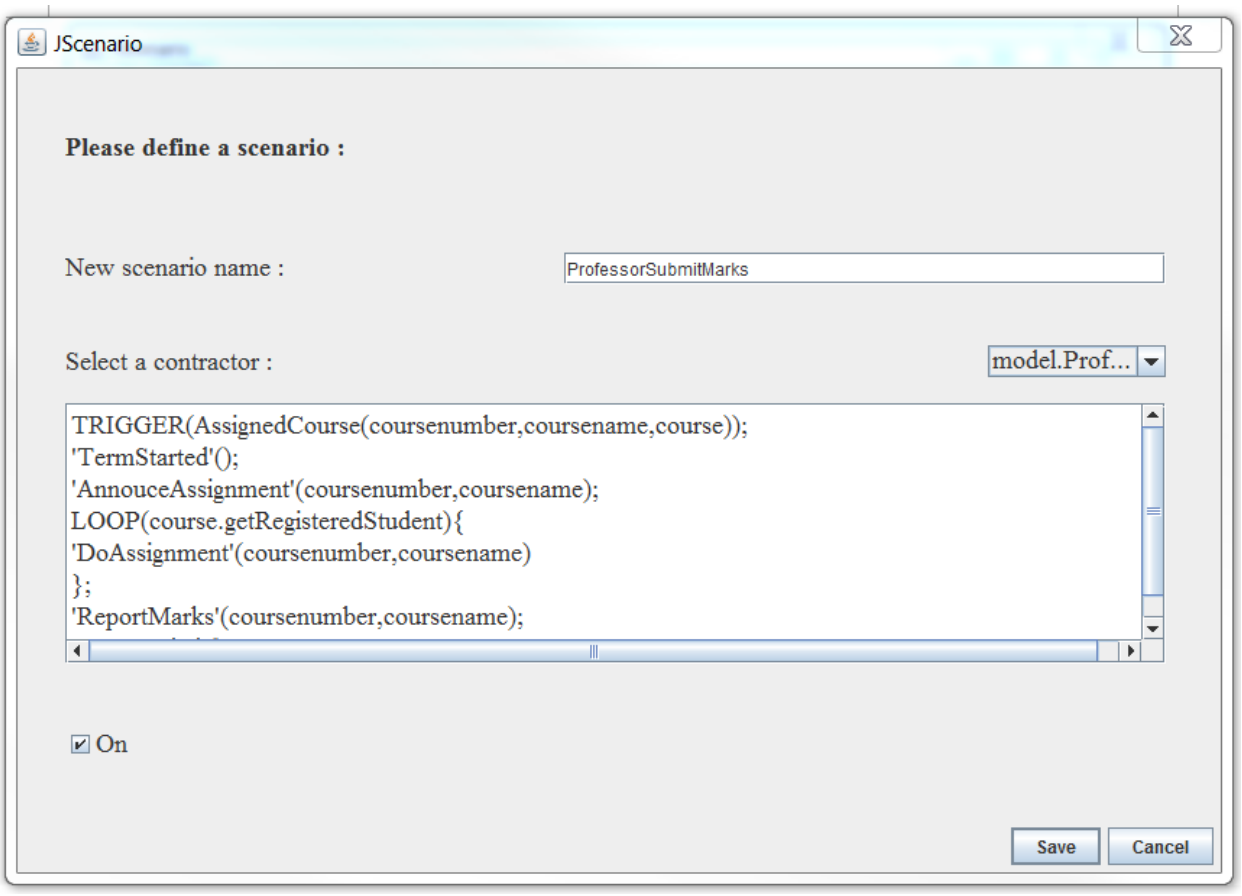
- Check the "Yes" to open View 1.
- Click "PrevStep" to go back to set up the testing environment.
- Choose the class you want to browse from the first drop box, and then choose the method in which you want to insert firing statement via the second drop box. The tool will then open the previous Editor view in which you will able to insert your firing statement.
- Click "NextStep" when you finished ALL event binding.

3) Define scenarios

The third step is to define the scenarios you want to test.



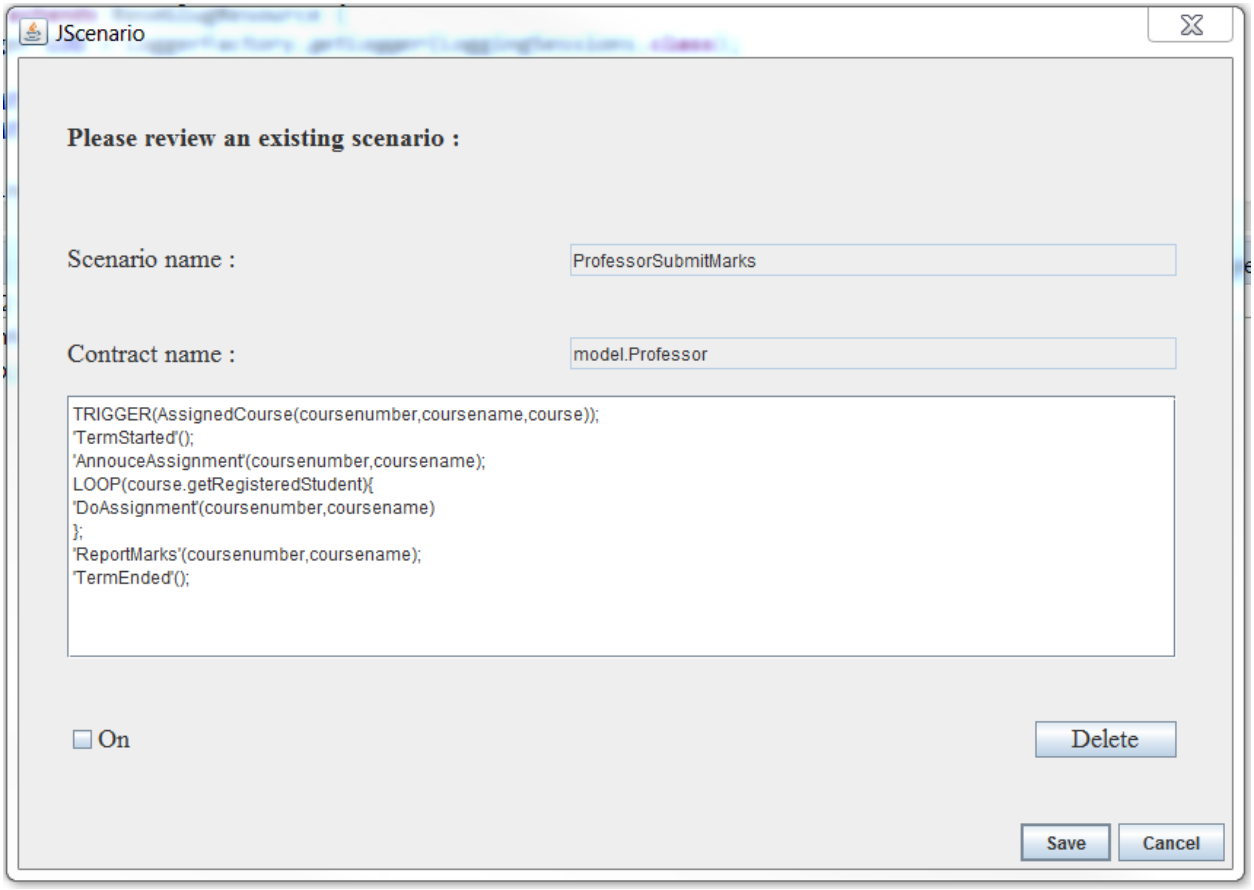
- Click “PrevStep” to go back to binding firing events.
- Click “Create” to open the create scenario view (see below).
- Browse the defined scenarios by contract name, and the table will present the list of scenarios.
- Click on any entry of the table and the tool will open the update scenario view (see below).
- Click “Next Step” when you are finished with scenario creation.



Create Scenario View

In the Create Scenario View, you can:

- Give the new scenario a name.
- Choose a contract for this scenario from the dropbox.
- Define this scenario in the text area. For syntax, please refer to part 3 of this guide.
- Check “On” to add this scenario to current test suite.
- Click “Save” to save the scenario.
- Your saved scenario can be found in JScenarioUI->resource->scenarios



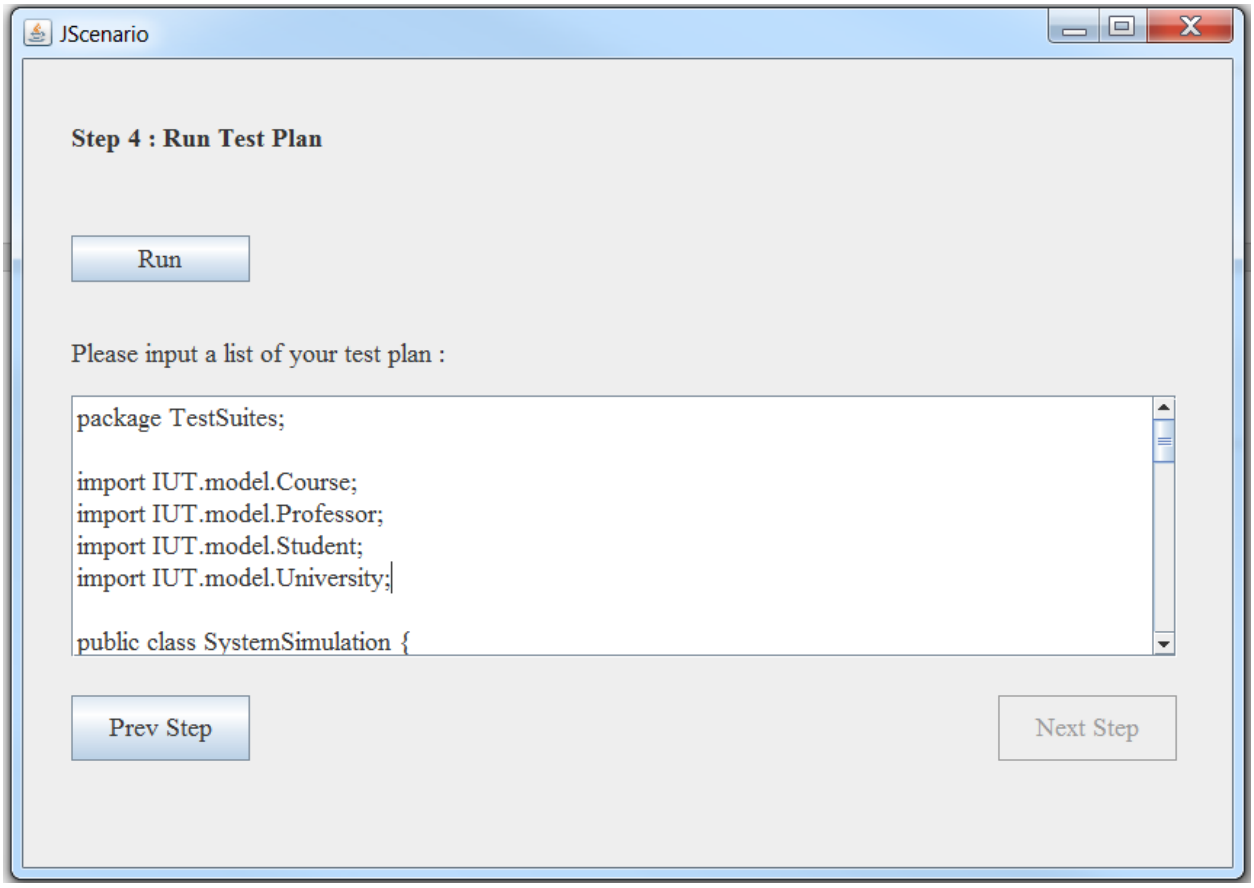
Update Scenario View

In the Update Scenario View, you can:

- View the current definition for this scenario.
- Update the scenario definition.
- Check “On” to add this scenario to current test suite, or uncheck “On” to remove this scenario from current testsuite.
- Delete this scenario.
- Click “Save” to update the scenario.
- Your updated scenario can be found in JScenarioUI->resource->scenarios

4) Run test plan

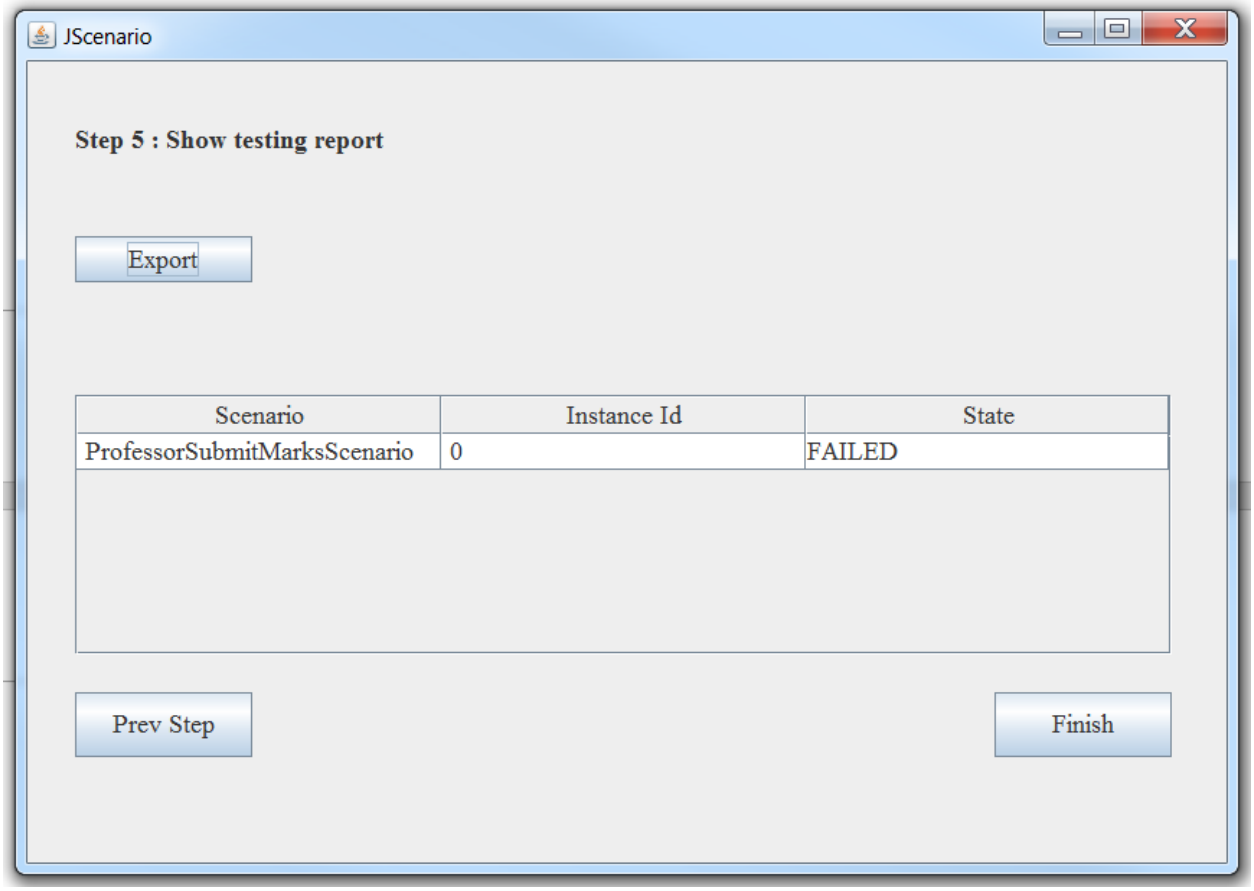
The next step is to give the tool your test plan and run it.



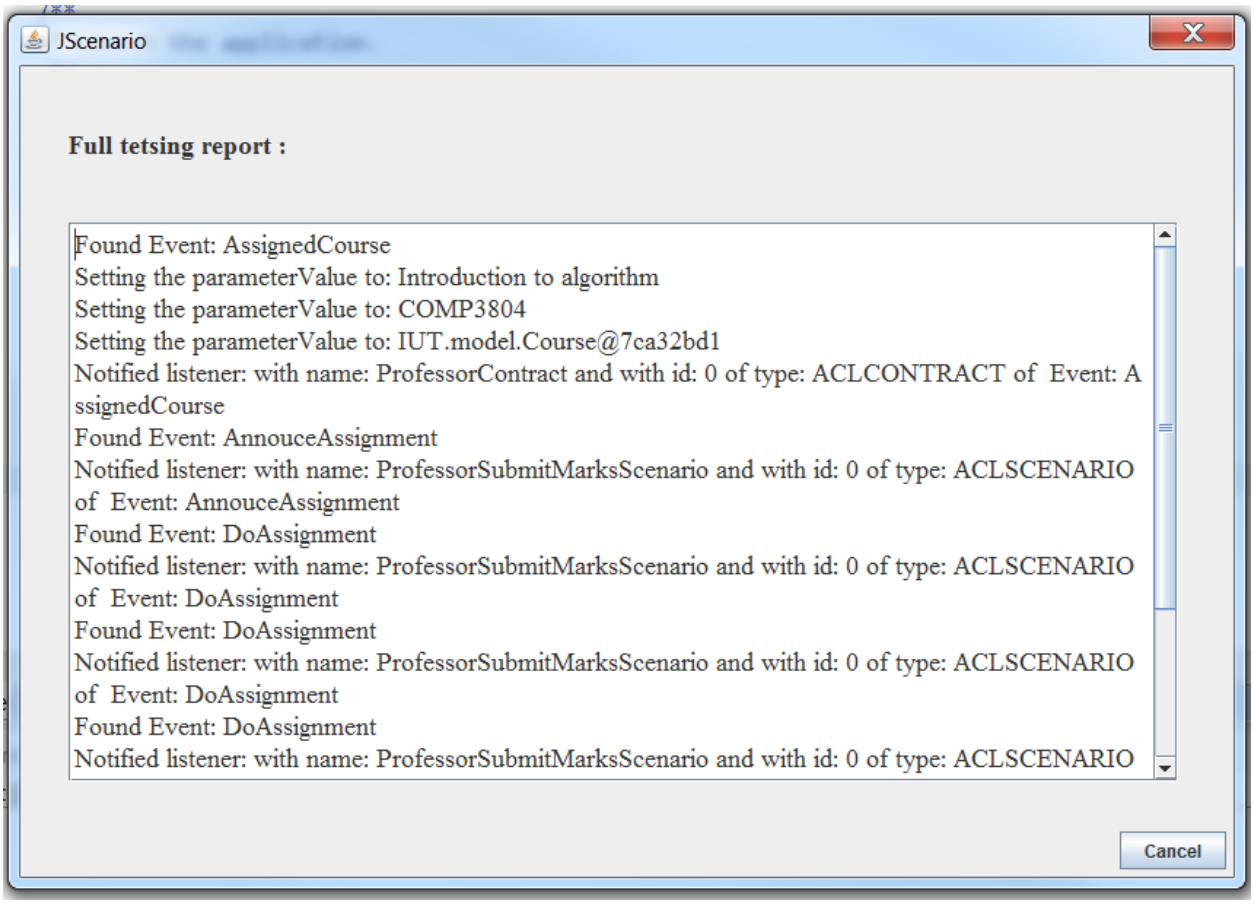
- Click “PrevStep” to go back to defining scenarios.
- Put your test plan in the text area first. See part 3 of this guide for syntax.
- Click “Run” to have JScenario start executing this test plan.
- When the test plan completes execution, the “Next Step” button will be enabled. Click “Next Step” to see the test report.

5) Show testing report

The last step is to view the testing result.



- Click “PrevStep” to go back to run the test plan.
- View the testing results as a table of which scenarios have been tested and what is the outcome of each one.
- Click “Export” to see the full testing report (see below).
- Click “Finish” to exit JScenario.



Full Testing Report View

3. Syntax Rules

1) Firing Event Statement

This statement can be inserted anywhere inside a method of a class.

Also, one method may have multiple firing event statements (eg in the branches of an IF statement).

The basic Fire statement is defined as follows: (yes… it's quite verbose…)

FireEvent.FireAnEvent("EventName", new ArrayList<Object>(Arrays.asList(parameterList)));

Examples:

- Event with no parameters:
  - FireEvent.FireAnEvent("ReportMarks", new ArrayList<Object>());
- Event with parameters:
  - FireEvent.FireAnEvent("ReportMarks",

    newArrayList<Object>(Arrays.asList(name, code)));

2) Scenario definition

- Keywords currently supported: ***TRIGGER, CHOICE, ALTERNATIVE, LOOP, PARALLEL.***
  TRIGGER: Trigger addresses the triggering event of a scenario.
  CHOICE: Choice marks the path the scenario should take when the statement's condition is satisfied. (ie THEN branch)
  ALTERNATIVE: ELSE branch of a choice statement
  LOOP: self-explanatory.
  PARALLEL: parallel (ie concurrent) events of a scenario.
  Equalto, NotEqualTo, GreaterThan, GreaterThanOrEqualTo, LessThan, LessThanOrEqualTo: Operators.

- Syntax with examples
  - Simple scenario:

  TRIGGER(NewProfessor(professor)); //each scenario starts with a trigger event
  'PayDay'();//an event has to be marked with '', except for the trigger event
  'PayProfessor'(professor);//each event needs to start on a new line and end with a ;
  *'TerminateProfessor'(professor); //within the ()s you find the values you want to use as the actual parameters passed with this event. This parameter list must be consistent with the fire statement you defined in the IUT, which means the same number **and** order as what the fire statement defines.*

- Scenario with a loop:

TRIGGER(AssignedCourse(coursenumber,coursename,course));
'TermStarted'();
'AnnounceAssignment'(coursenumber,coursename);
LOOP(course.getRegisteredStudents){// a loop's condition is usually a call to one of a class's observability methods. In this case, getRegisteredStudent is an observability method of class Course, because course is an instance of class Course. An observability method is a method that does not have any parameters but has a return value. Here course.getRegisteredStudents returns a list of student instances over which to iterate.
'DoAssignment'(coursenumber,coursename) // events inside the loop's scope (ie {})) apply to each value iterated over, like the statements inside a for loop statement in java. While it appears the DoAssignment has only two parameters, it actually requires 3: the third one is implicit and is conceptually after the two explicit ones. It consists in the iterator. In this example:
DoAssignment(coursenumber,coursename, studentInstance[i])
Consequently, when defining the corresponding fire statement, the parameter list should have three elements including the student instance as last one.
};
// Conceptually, this loop statement would translate in Java as follows:
// Student[] studentList= course.getRegisteredStudent();
// for(int i=0;i< studentList.length;i++) {
// DoAssignment(coursenumber,coursename, studentInstance[i]); }
'ReportMarks'(coursenumber,coursename);
'TermEnded'();

- Scenario with choice/alternative:

TRIGGER(AssignedCourse(coursenumber,coursename,course));
'GiveLecture'(coursenumber,coursename);
'MonthElapsed'();
CHOICE(course.HasProject Equalto true){ // a choice condition has three parts, the observability method, the operator and the comparison result. You must use space as the delimiter to distinguish the three parts
'AnnouceProject'(coursenumber,coursename),
'MarkProject'(coursenumber,coursename),
'HoldOfficeHours()'
}
ALTERNATIVE{
'AnnouceFinal'(coursenumber,coursename),
'MarkFinal'(coursenumber,coursename),
'ExamFeedback'()
};
'TermEnded'();

- Scenario with parallel:

```
TRIGGER(RegisterStudentForCourse(student,course));
PARALLEL{
        'DoAssignment'(student,course),
        }
        {
        'DoProject'(student,course,
        };
```

- Restriction:
  - Inside one scenario, **all** parameters of all events have to be specified as parameters of the triggering event. If the triggering event of scenario s has only parameters a and b, and one of the events in that scenario s refers to parameter d (as in eventA(d)), then the scenario will be deemed invalid.
  - And remember that the definition of a list of actual parameters must correspond to the list of formal parameters (in number and order), except in the case of events inside loops.

3) Test plan

The test plan is simply a set of method calls. Make sure to import all the classes you are using.

4. Troubleshooting

1) Running a test plan

The thread-pool used in this application may cause some trouble. The most common bug occurs when you are trying to run the test plan: it ends up very soon and the testing report may be very short. This may be because you started running your test plan before the backend finished generating all the scenarios. There are several ways to avoid or fix this problem:

- Before clicking the “Run Test Plan” button, wait a few seconds and go to eclipse to refresh the project. Also make sure all your scenarios have been created under JScenarioServer->src->Scenario. Then and only then click on that button and let the test plan run.
- If the problem does occur, you can kill the application and run the JScenario.java again. But you do not need to reset anything: just click ‘next step’ until you can run your test plan.
- If the problem is still occurring, go directly to JScenarioUI->src->layout->RunTestPlan.java, and run as application.