Software Engineering & Software Development Project

Tutorials >

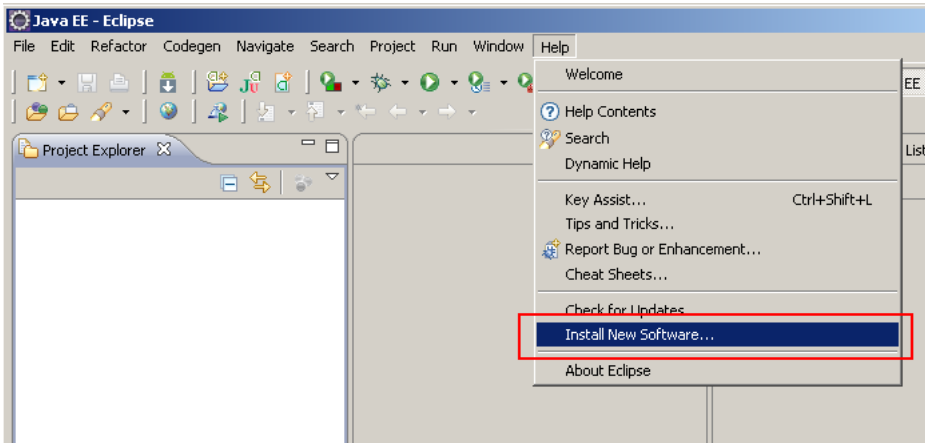
Code Coverage with CodeCover

In this tutorial, we explain step-by-step how to use the CodeCover plugin in Eclipse to measure various types of coverage of your code.

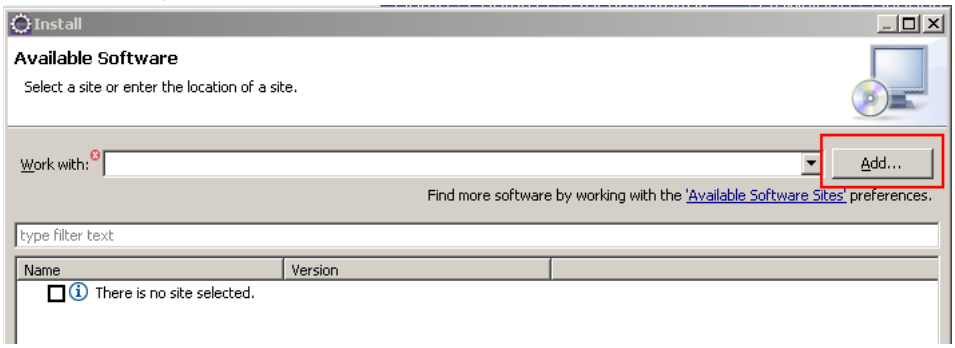
1. Installation Instructions

We briefly present the steps to install the plugin in Eclipse.
For more information, follow the instructions at http://codecover.org/. If the plugin is already installed, skip to section 2.

Click "Help/Install New Software".



In the Install window, click "Add".



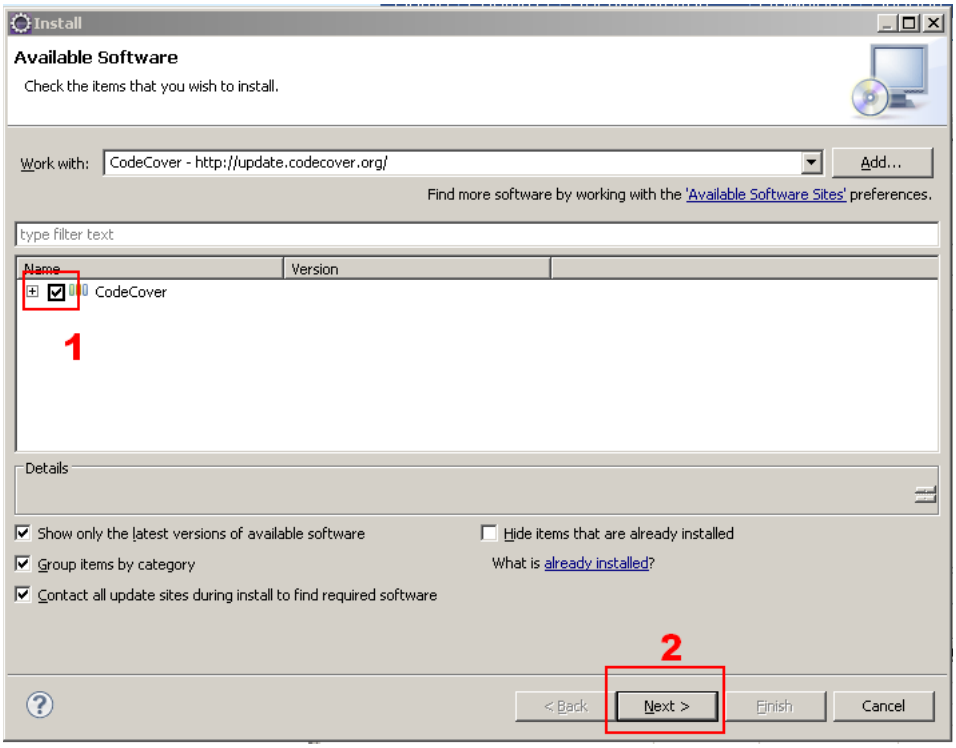
In the Add Site window, specify "http://update.codecover.org/ in the location field.
Also choose a name, then confirm.

Tick the checkbox next to the "CodeCover" entry, then click "Next".

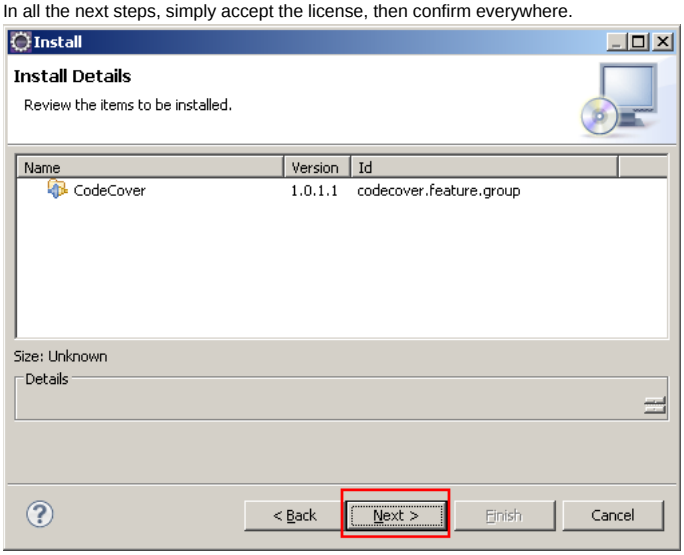
Overview
Schedule
Administrivia
Pocket Campus
Collaboration Policy
Tutorials
Archive
Quizzes

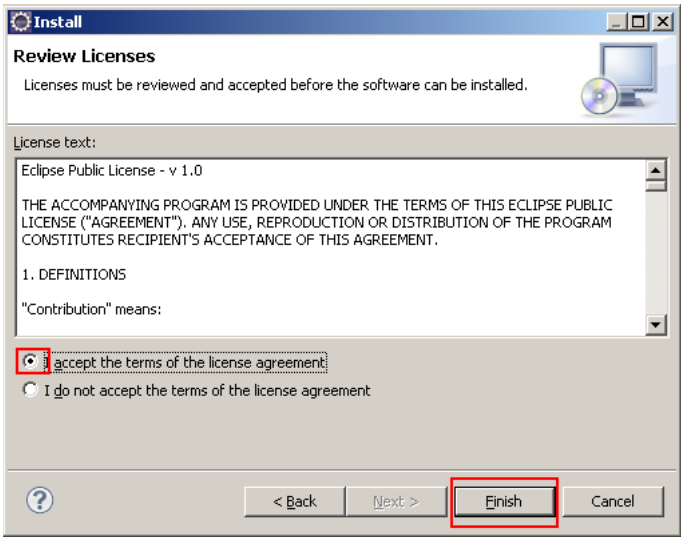
ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

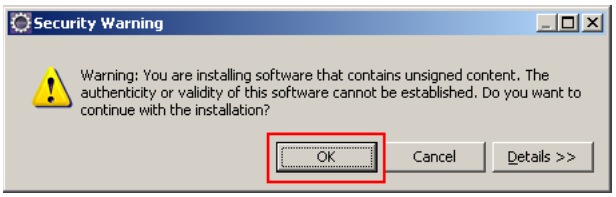
1232
days since
the courses start



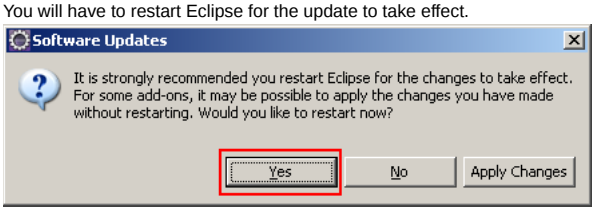
In all the next steps, simply accept the license, then confirm everywhere.







You will have to restart Eclipse for the update to take effect.



2. Creating a Test Project

In this section, we show how to measure various coverage metrics for a test project.

Create a new Java project in Eclipse, and add a new class called `CoverageTutorial` in the `epfl.sweng package`.

Paste the following code in your `CoverageTutorial.java`.

CoverageTutorial.java

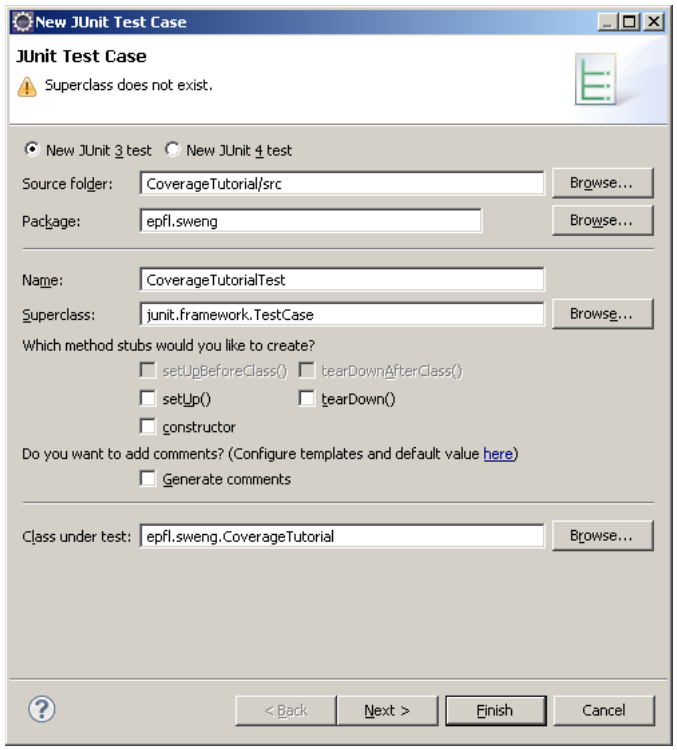
```
package epfl.sweng;

import java.util.ArrayList;
import java.util.Set;
import java.util.TreeSet;

public class CoverageTutorial {

    static boolean func1(int a1, int a2, int a3) {
        if (a1 > 0 && a2 < 4) {
            return true;
        }

        if (a1 == 7 && a3 < 8) {
            return false;
        }

        for (int i=a1; i<a2; ++i) {
```

```
            System.out.println(i + "\n");
        }
        return true;
    }

    public static void main(String[] args) {

    }
}
```

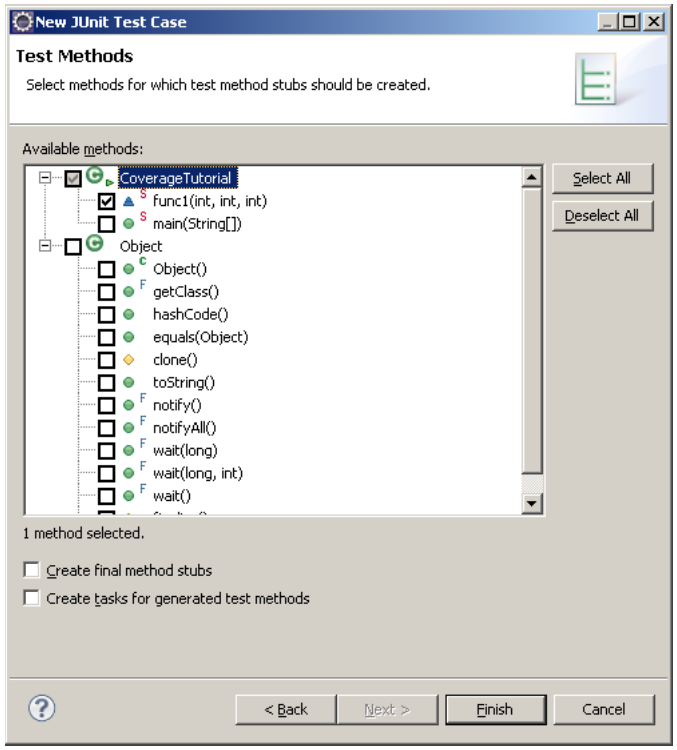
3. Creating a JUnit Test Case

This tutorial focuses on JUnit3. JUnit4 is similar except that it uses annotations.

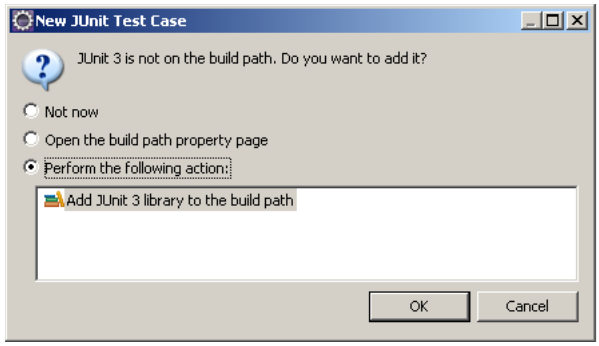
Create a JUnit test case using "File/New/JUnit Test Case".
Fill in the form as follows, then click **"Next"**.



Select "func1" as the function for which to create a test case.
When you are done, click "Finish".



Confirm that you want to add JUnit to the build path.



Now, replace the generated `CoverageTutorialTest.java` file with the following content.
It will call `func1` with some arbitrary arguments to exercise one of its code paths.

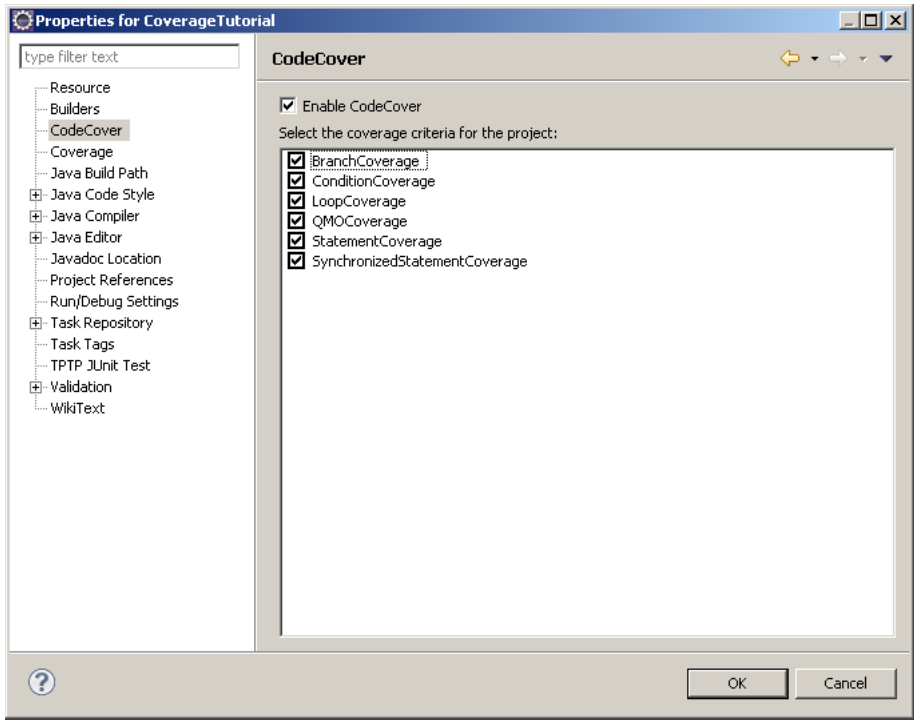
CoverageTutorialText.java

```
package epfl.sweng;

import junit.framework.TestCase;

public class CoverageTutorialTest extends TestCase {

   public void testFunc1() {
      CoverageTutorial.func1(1, 2, 3);
   }
}
```

4. Setting up CodeCover

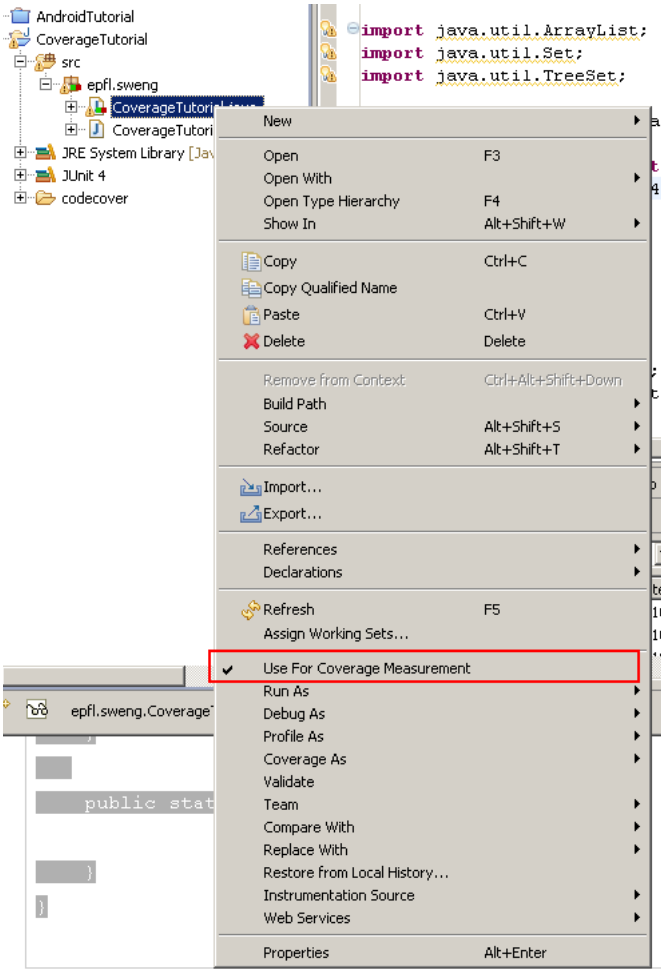
Right click on the CoverageTutorial project in the package explorer, then click "Properties".

In the window that appears, select "CodeCover", "Enable CodeCover", then tick all the checkboxes, as shown in the following screenshot.

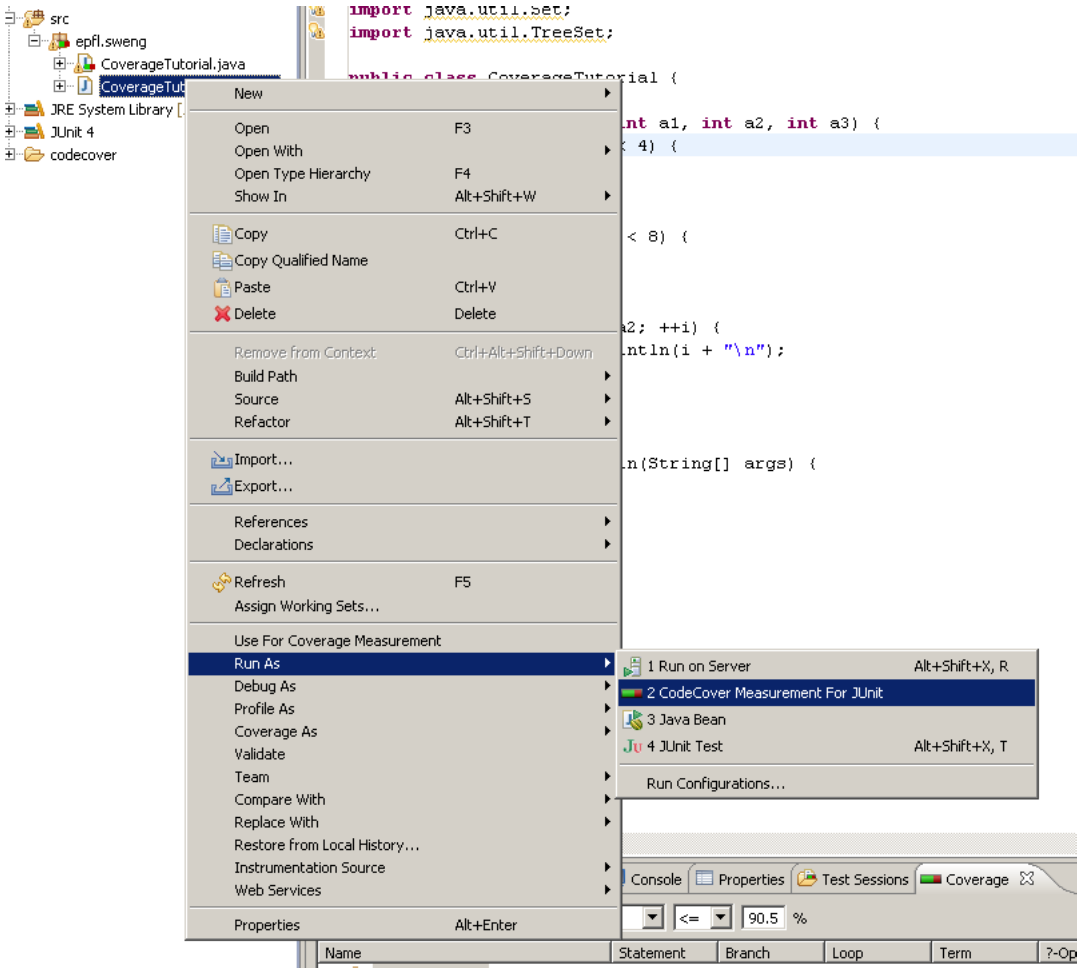


5. Running the Program with Coverage

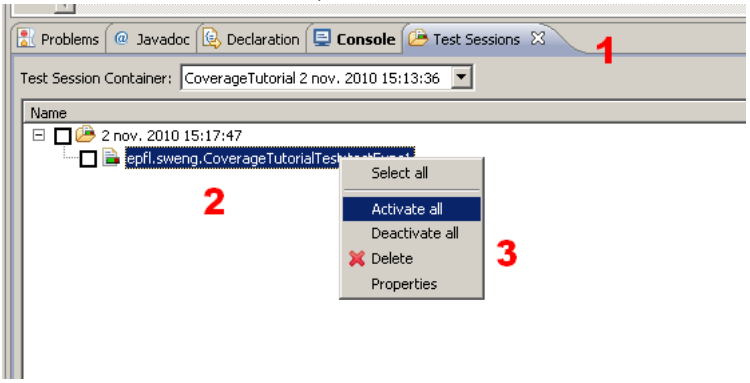
First, select "Use for Coverage Measurement" for every file you want to measure.

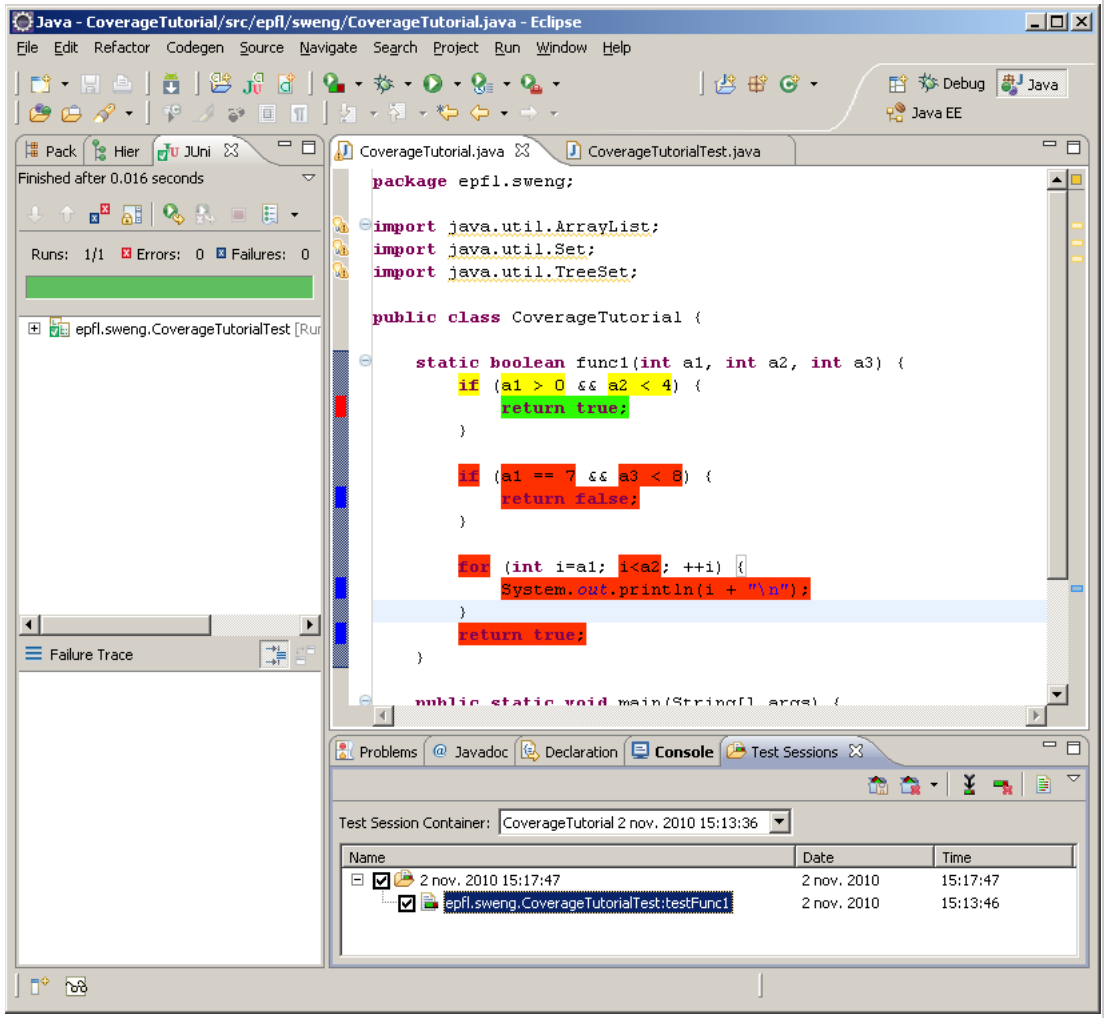


Right click on `CoverageTutorialTest.java`, then select "Run As/CodeCover Measurement for JUnit".



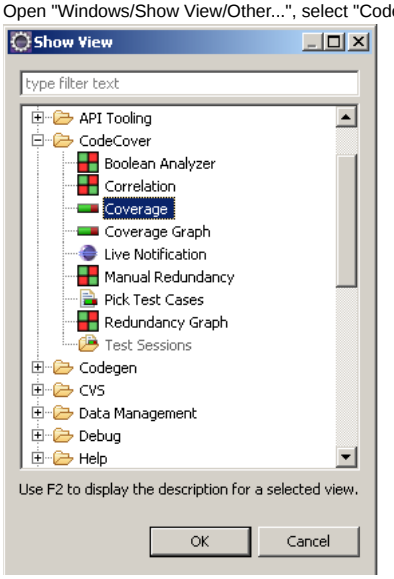
Once the program has completed, CodeCover creates a test session that needs to be activated to view the results.
If you do not see the "Test Sessions" tab, open "Windows/Show View/Other...", select "CodeCover/TestSessions".

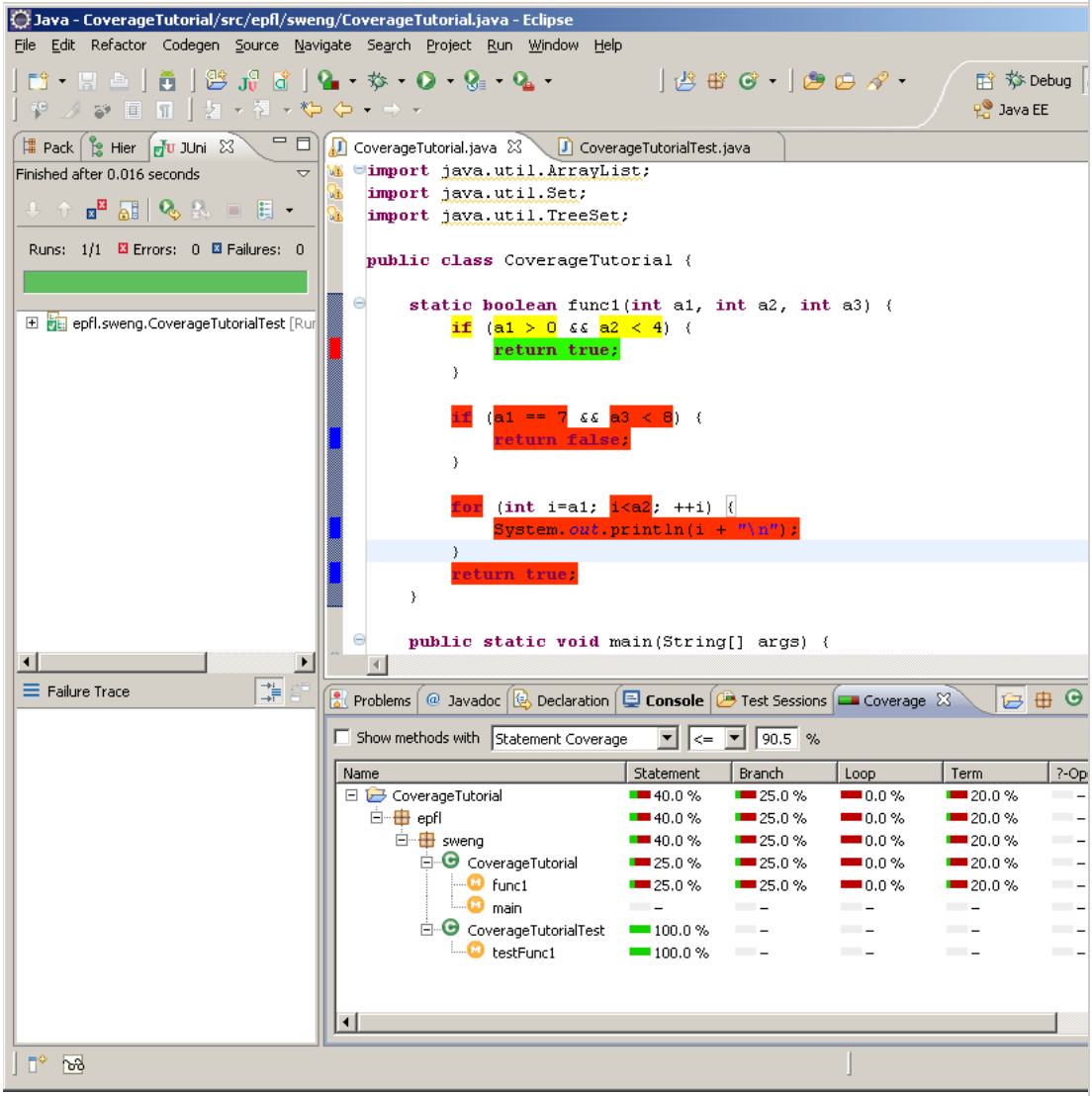




To view the detailed coverage, open the "Coverage" tab from the list of available views.

Open "Windows/Show View/Other...", select "CodeCover/Coverage".





5. CodeCover for JUnit4

In this part, we explain how to write test cases for JUnit4.

- You must select JUnit4 when you create a test case (New/Junit TestCase)
- Alternatively, you can add the JUnit library to your build path in your project properties

CoverageTutorialJUnit4.java

```
package epfl.sweng;

import static org.junit.Assert.*;

import org.junit.After;
import org.junit.Before;
import org.junit.Test;

public class CoverageTutorialTestJUnit4 {

   @Before
   public void initHere() {
      //Put your initialization code here, e.g., to setup
      //private variables and other state
```

```
    System.out.println("Initing");
  }

  @After
  public void tearDownHere() {
    //Put your cleanup code here
    System.out.println("tearDown");
  }

  @Test
  public void testFunc1() {
    CoverageTutorial.func1(1, 2, 3);
  }

  @Test
  public void testFunc2() {
    CoverageTutorial.func1(7, 5 ,7);
  }

  @Test
  public void testFunc3() {
    CoverageTutorial.func1(5, 7 ,8);
  }
}
```

The `@Before` and `@After` annotations run the initialization code for your test class. It is useful when you want to create multiple test cases that operate on the same data. It is bad practice to initialize such data in the test case itself.

Comentarios

No tienes permiso para añadir comentarios.