JaBUTi
(Java Bytecode Understanding and Testing)

Dr. Durga Prasad Mohapatra

Associate Professor

Department of Computer Sc. & Engineering

National Institute of Technology, Rourkela

Outline

- Introduction
- Structural test requirements supported by JaBUTi
- Coverage Criteria
- Installing JaBuTi
- Settings for JaBuTi
- Example

Introduction

- The first version of this manual was compiled in 2002, by Vincenzi, Delamaro and Maldonado.
- JaBUTi is a set of tools designed for understanding and testing of Java programs.

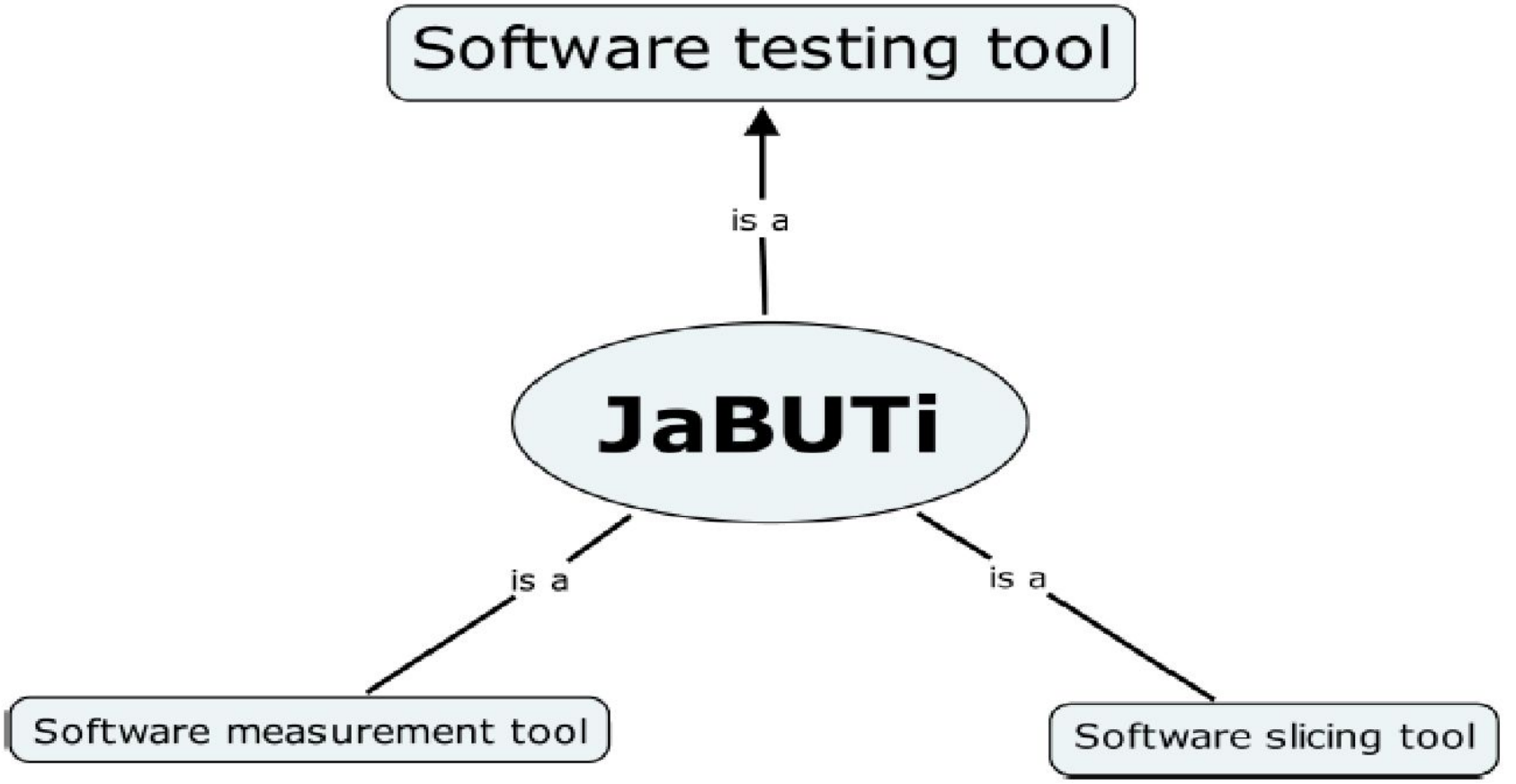
Introduction

- The main advantage of JaBUTi is that it does not require the Java source code to perform its activities.

- Such a characteristic allows, for instance, to use the tool for testing Java-based components (whose source code is usually not available) or any alien Java application.

- JaBuTi is designed to work with Java bytecode
  - such that no source code is required to perform its activities.
- It is composed of
  - a coverage analysis tool,
  - a slicing tool, and
  - a complexity metric's measure tool.
- The coverage tool can be used
  - to assess the quality of a given test set or
  - to generate test set based on different control-flow and data-flow testing criteria.

- The slicing tool can be used to
  - identify fault-prone regions in the code, being useful for debugging and also for program understanding.
- The complexity metric's measure tool can be used to
  - identify the complexity and
  - the size of each class under testing, based on static information.


Software testing tool
is a
JaBUTi
is a
is a
Software measurement tool
Software slicing tool

Structural test requirements supported by JaBUTi

- JaBUTi requires just Java bytecode to test an application.
- Java bytecode can be thought of as an assembly language -like,
  - based on a stack-based (virtual) machine (instead of accumulator or register based real machines).
- Programs described in such language
  - can be analysed, and
  - control-flow and data-flow information can be collected
  - and represented as a def/use graph.

Structural test requirements supported by JaBUTi

- Once collected such information for each method,
  - intra-method test criteria can be defined and applied.
- Basically, at method level,
  - traditional control-flow and data-flow test criteria can be applied to object-oriented software,
    - since they are based on the same underlying representation, i.e., the def/use graph.
- However, there is an important difference between procedural and object-oriented program at method level:
  - the presence of exceptions and exception-handlers.
- Exception-handling is handled by specializing the default edge of the graph into primary and secondary edges.

Structural test requirements supported by JaBUTi

- Primary edges represent the regular control-flow (i.e., when no exception is thrown).
- Secondary edges represent the exception-handling control-flow.
- For every statement that can throw an exception,
  - a secondary edge is defined to the exception handling code.
- The test criteria defined for JaBUTi also specializes the definition of the node.
- The default node of the graph is classified into primary and secondary nodes.

Structural test requirements supported by JaBUTi

- Primary nodes represent normal control-flow, i.e., which are reachable from primary edges.
- A secondary node represents a node which is reachable from secondary edges.
- For Java source code, this usually means that code within catch and finally blocks are secondary nodes; otherwise are primary nodes.

Coverage Criteria

There are six coverage criteria is used by the JaBUTi testing tool. This is described below.

- Data Flow Coverage Criteria:
  - all-uses criterion(All-Uses)
    - all-uses exception-independent (All–Uses-ei): Set of definition/use associations for which there exists a path of primary edges only.
    - all-uses exception-dependent (All–Uses-ed): Set of definition/use associations for which there exists secondary edges.
  - All-Potential-Uses
    - All-Potential-Used-ei: Set of definition/potential-use associations for which there exists a path of primary edges only.
    - All-Potential-Uses-ed: Set of definition/potential-use associations for which there exists secondary edges.

Coverage Criteria

- Control Flow Coverage Criteria:
  - all-nodes criterion(All-Nodes): Requires that each node be exercised at least once by a test case. This criterion ensures that every statement in the method have been executed at least once by a given test case.
  - all-nodes-exception-independent (All – Nodes-ei): Requires that each primary node that has been exercised at least once by a test case in the test set. This criterion requires that all statements not related with exception-handling mechanism were executed at least once.
  - all-nodes-exception-dependent (All – Nodes-ed): Requires that each secondary node has been exercised at least once by a test case. This criterion requires that all statements related with exception-handling mechanism were executed at least once.

Coverage Criteria

- all-edges criterion(All-Edges): Requires that each edge be exercised at least once by a test case. This criterion ensures that every possible control transfer in the method has been executed at least once by a given test case.
  - all-edges-exception-independent (All – Edges-ei): Requires that each primary edge has been exercised at least once by a test case. This criterion requires that all conditional expressions have been evaluated as true and false at least once.
  - all-edges-exception-dependent (All –Edges-ed): Requires that each secondary edge has been exercised at least once by a test case. This criterion requires that each secondary node be executed at least once from each node where an exception might be raised.

Installing JaBuTi

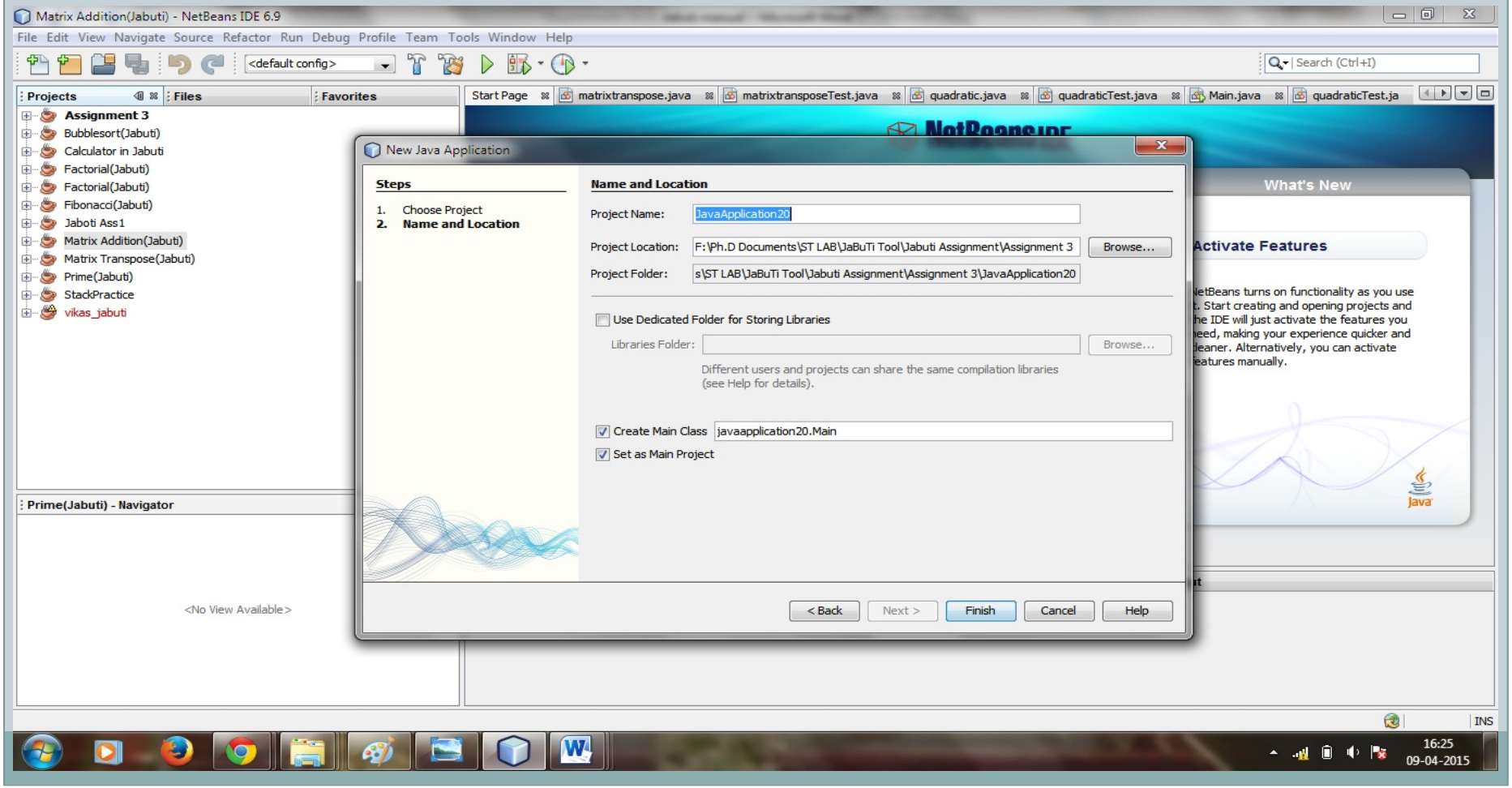
- Install Jdk 1.6 or higher
- Install NetBeans 6.9.1 or higher
- Install Graphviz 2.26
- Set PATH for Java: C:\Program Files\Java\jdk1.7.0_45\bin;
- Set CLASSPATH for Java: C:\Program Files\Java\jdk1.7.0_45\bin;
- Set PATH for Graphviz: C:\Program Files\Graphviz2.26\bin\Gvedit.exe;
- Set CLASSPATH for Graphviz: C:\Program Files\Graphviz2.26\bin\ Gvedit.exe;

Settings

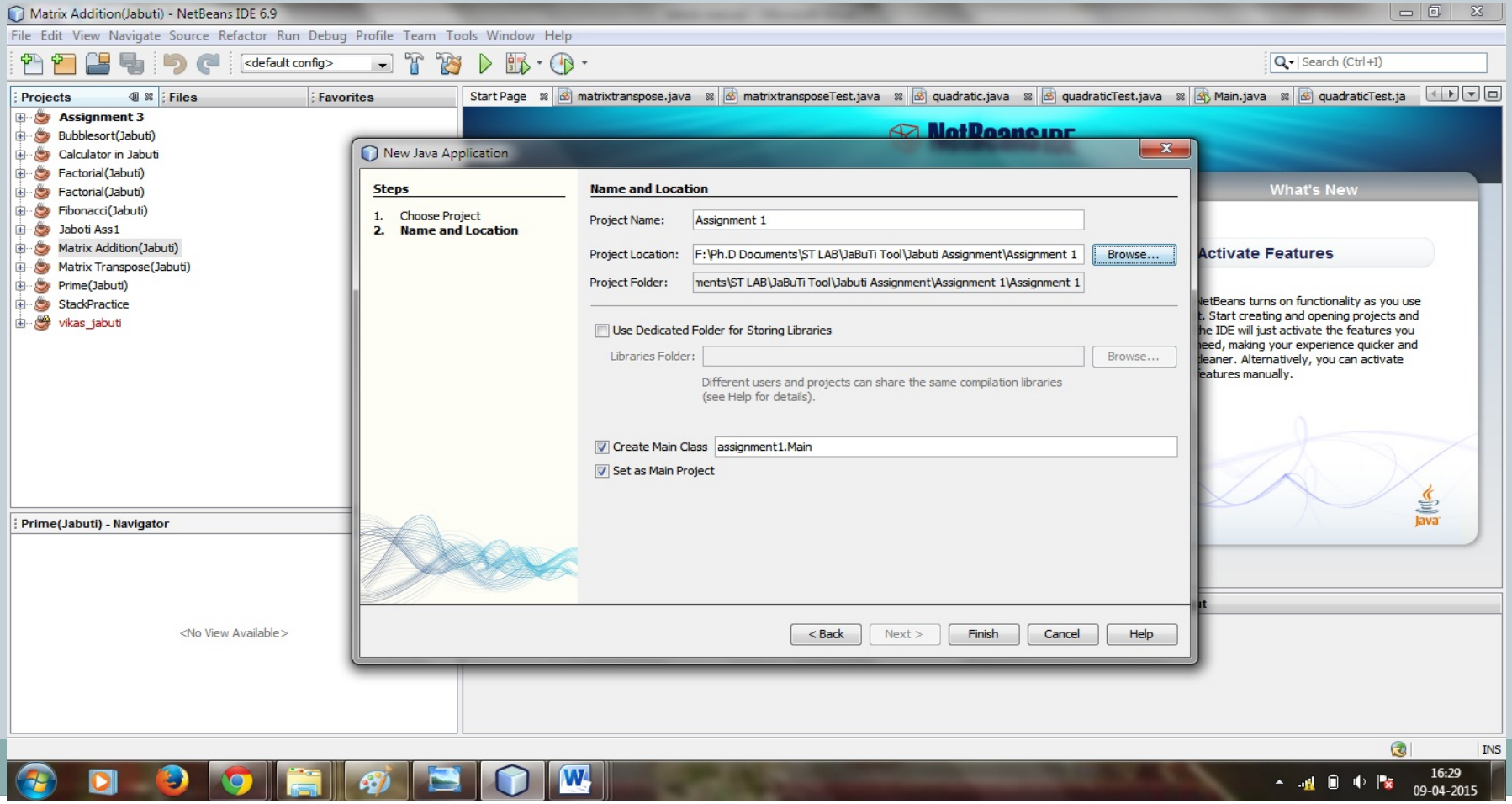
- Download Jabuti.jar
- Create a folder named “jab” in any drive, say E:
- E:\jab
- Copy Jabuti.jar to E:\jab and Extract Jabuti.jar to that folder.
- This will create two new folders: Tools and examples
- Create a folder named “lib” under E:\jab\Tools\jabuti.
- Put the following files under E:\jab\Tools\jabuti\lib:
  - bcel-5.2.jar,
  - crimson.jar,
  - junit-4.10.jar,
  - jabuti.jar
- Set PATH for Jabuti: E:\jab; E:\jab\Tools\jabuti\lib;
- Set CLASSPATH for Jabuti: E:\jab; E:\jab\Tools\jabuti\lib;

Example Using Netbeans

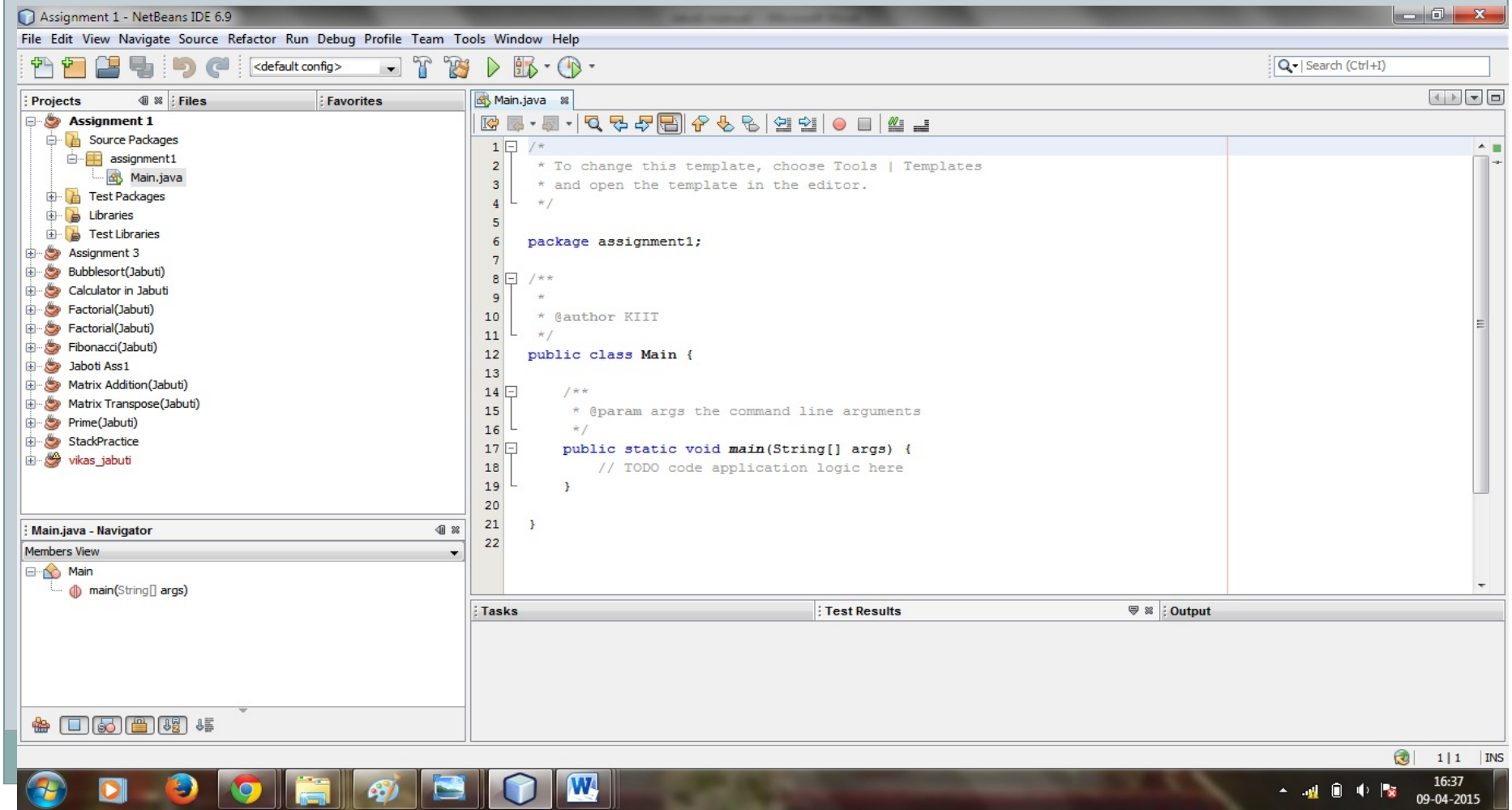
- File ☐ New Project ☐ Java ☐ Java Application ☐ Next.



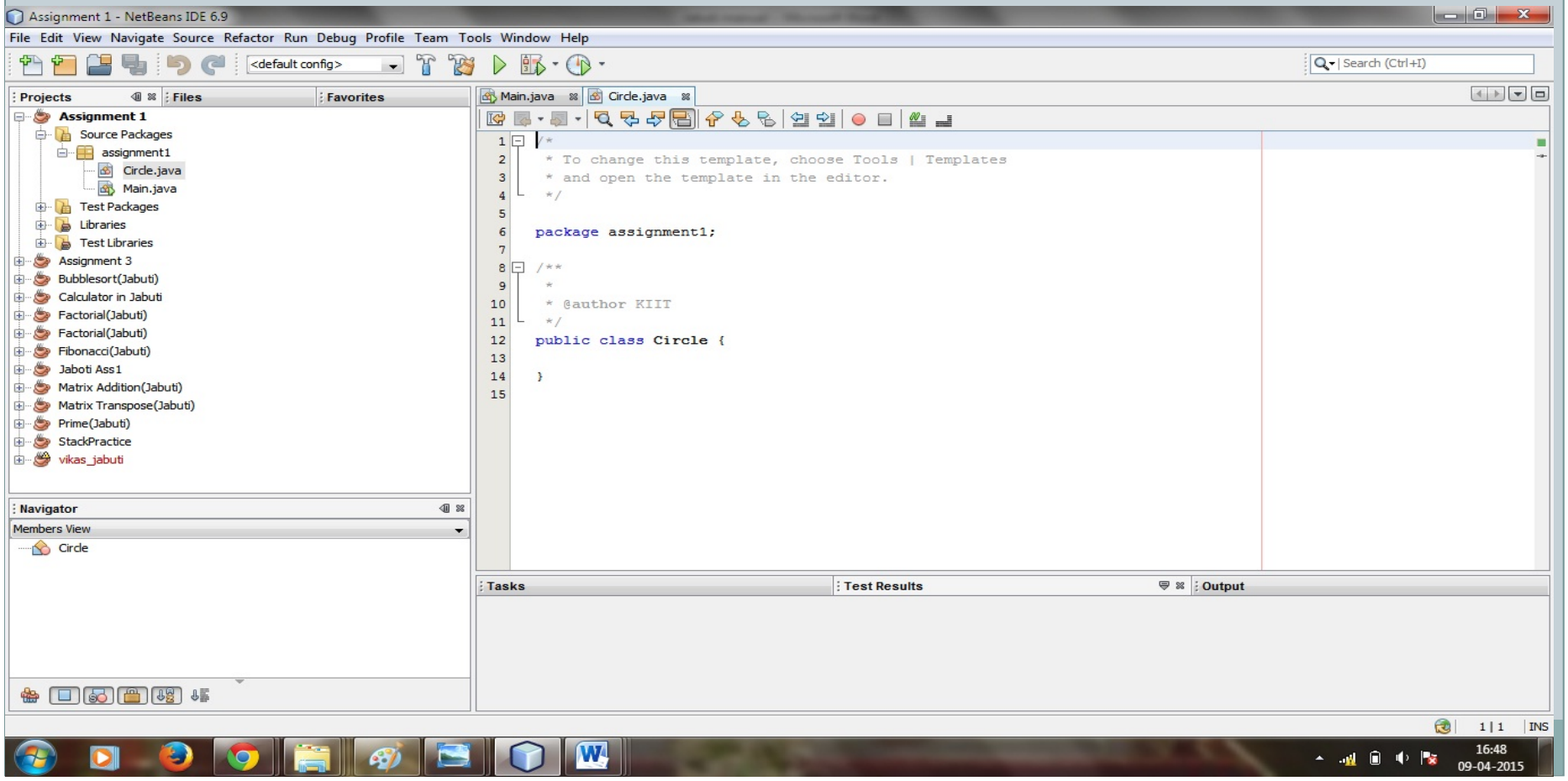
- Give the Project name (i.e. Assignment 1) and project location.



- Click on Finish. Then the project will be created and Main.java is displayed like



- Right click on Source package ☐ New ☐ Java class ☐ give Class name (Circle.java) and Package (assignment 1)



- Write the code for calculate for area and perimeter.

Circle.java


```
package assignment1;
public class Circle {

   public int area(int r)
   {
      int area = (int) (Math.PI * (r * r));

          return(area);
    }

   public double perimeter(double r)
   {
      double perimeter = 2 * Math.PI * r;

          return(perimeter);
    }
}
```

- Right click on Circle.java ☐ Tools ☐ Create Junit Tests ☐ Select ☐ Ok
- Write the code for test cases.

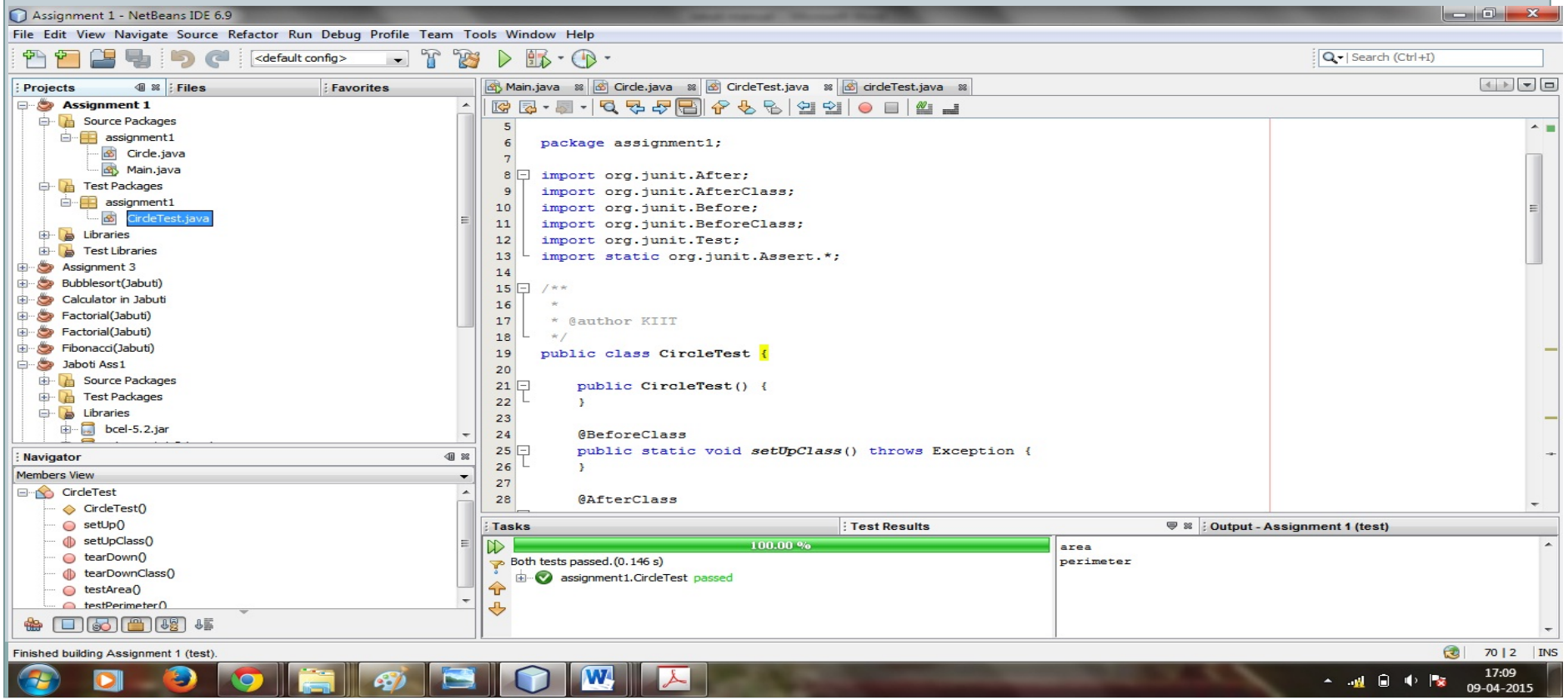
CircleTest.java


```
package assignment1;
import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import static org.junit.Assert.*;
public class CircleTest {
    public CircleTest() {    }
    @BeforeClass
public static void setUpClass() throws
Exception{}
@AfterClass
public static void tearDownClass() throws
Exception {}
```

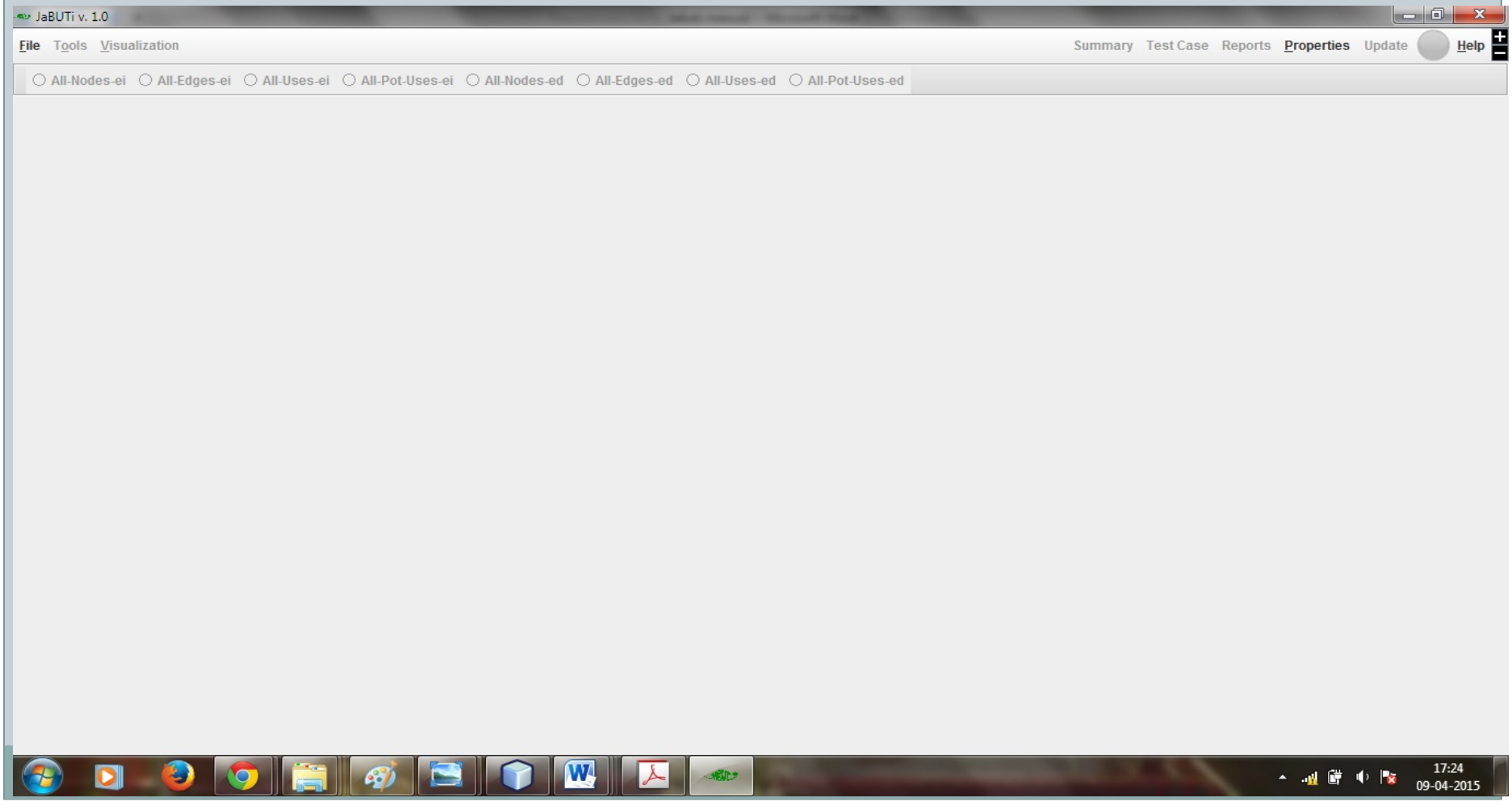
```
 @Before
    public void setUp() {
    }

    @After
    public void tearDown() {
    }

    /**
     * Test of area method, of class
Circle.
     */
    @Test
    public void testArea() {
        System.out.println("area");
        int r = 2;
        Circle instance = new Circle();
        int expResult = 12;
        int result = instance.area(r);
        assertEquals(expResult, result);
        // TODO review the generated
test code and remove the default call
to fail.
```

```
// fail("The test case is a
prototype.");
    }
/**
     * Test of perimeter method, of
class Circle.   */
    @Test
    public void testPerimeter() {

System.out.println("perimeter");
        double r = 2.5;
        Circle instance = new Circle();
        double expResult = 15.705;
        double result =
instance.perimeter(r);
        assertEquals(expResult,
result, 15.705);
        // TODO review the generated
test code and remove the default
call to fail.
     //   fail("The test case is a
prototype.");
    }
}
```

- Rclick on Libraries ☐ Add JAR/Folder ☐ Select bcel-5.2.jar, crimson-1.1.3.jar, jabuti.jar and junit.jar ☐ click on Open.
- Rclick on CircleTest.java ☐ Run File

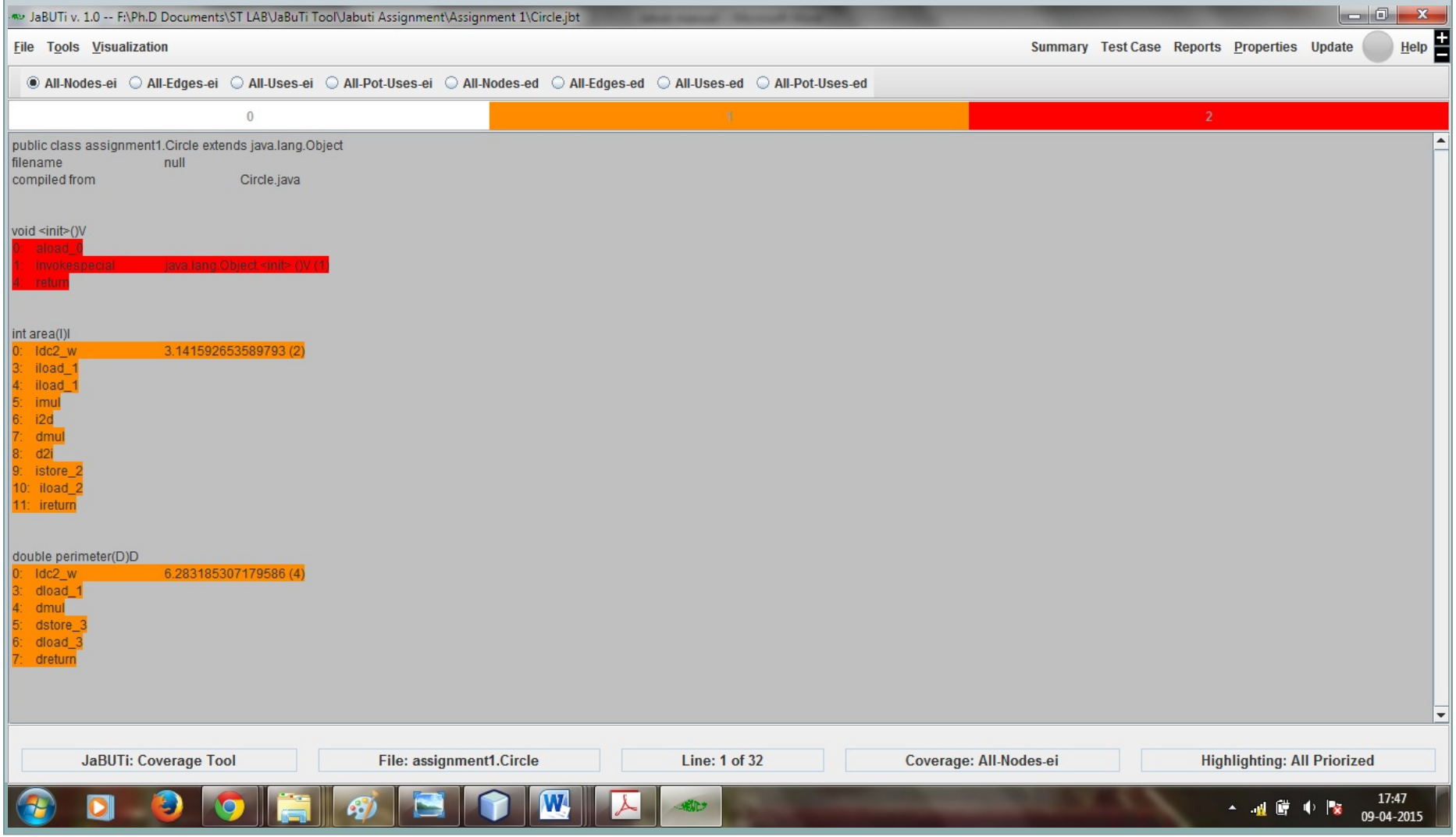


- Open Jabuti.jar by double clicking on it.

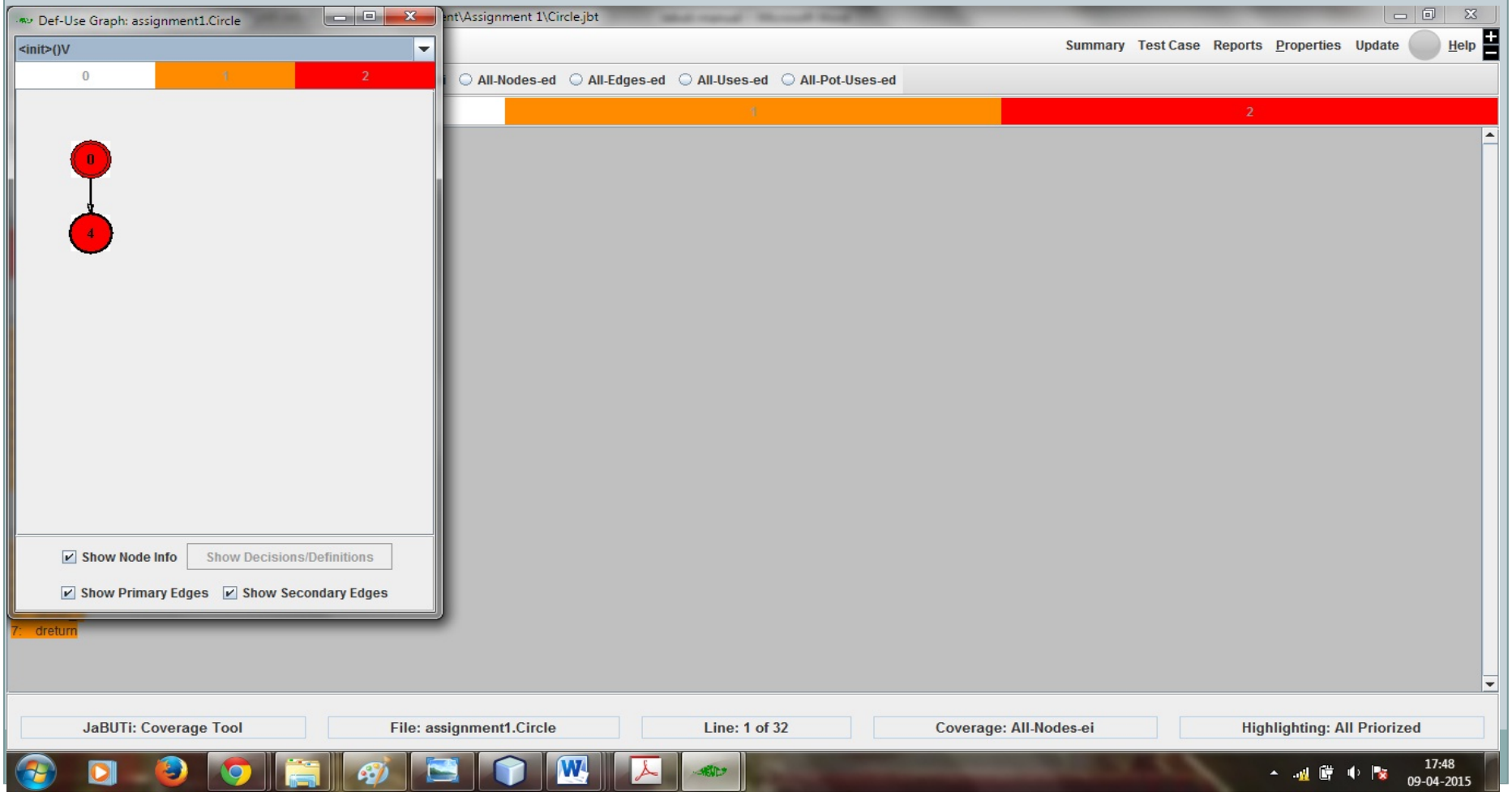


- Copy CircleTest.java to assignment 1 package.
- In Jabuti File ☐ Open Class ☐ select Circle.class in file name and assignment 1 as package name. Then give the path of class.
- E:\jab\examples\Assignment1\build\classes
- Click on Open.
- Click and extract the package (i.e assignment 1) and select java file (Circle).
- Give the project name and class path of project. Click on OK.

Bytecode of circle example program



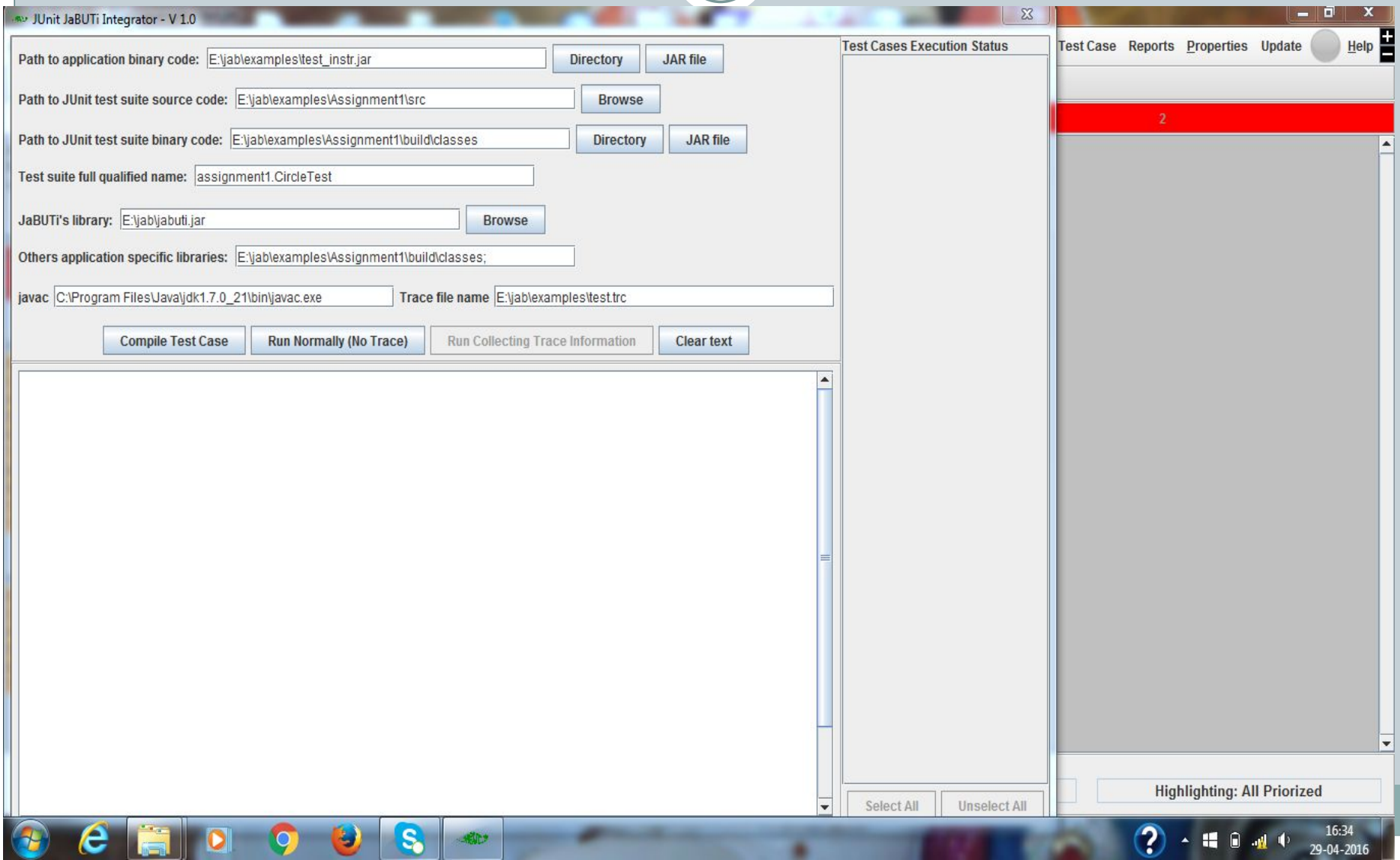
- Visualization (from menubar) ☐ Def – Use Graph



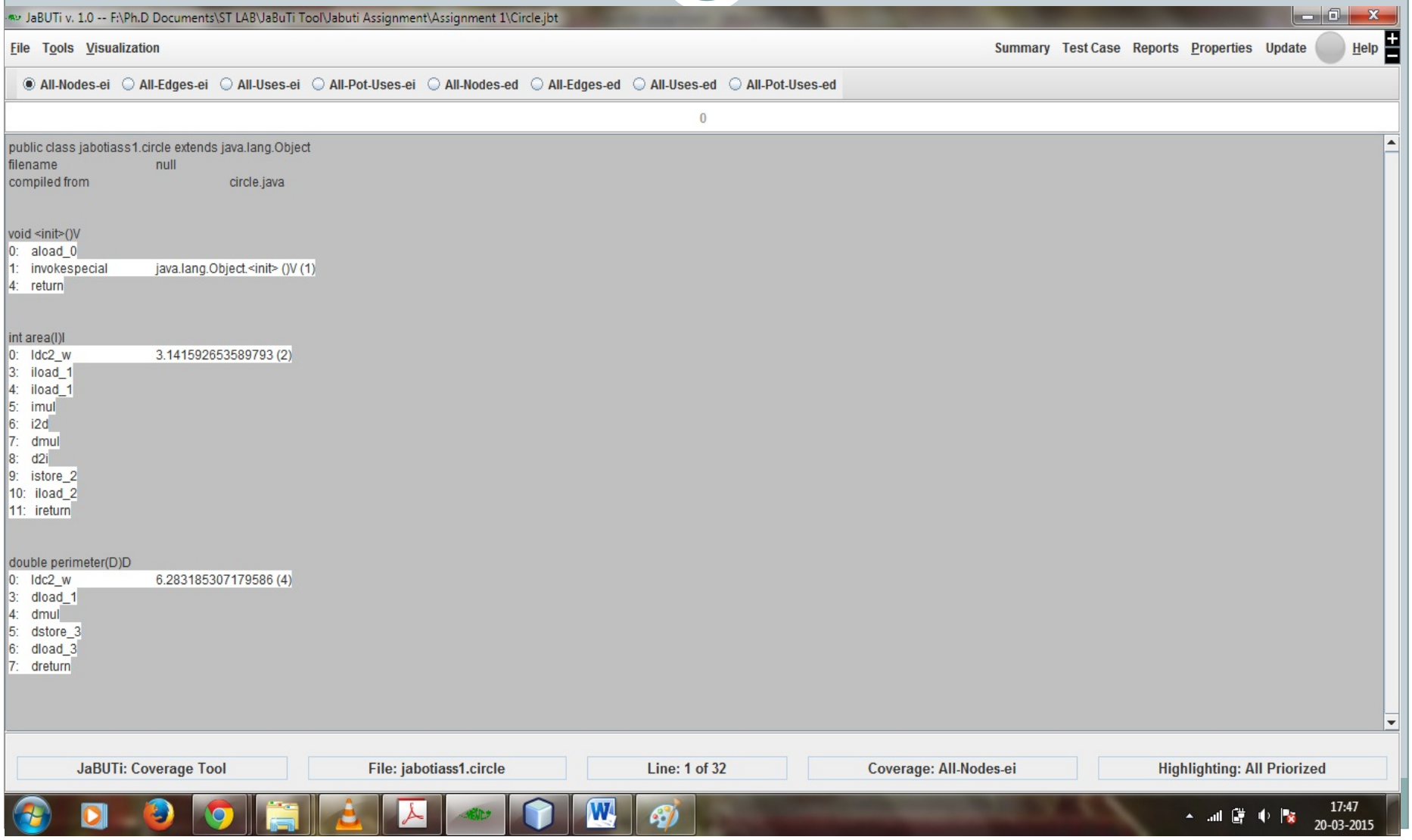
- Click on Test Cases ☐ Executing Junit Test Cases
- Path to application binary code – E:\jab\examples\test_instr.jar
- Path to Junit test suite source code – E:\jab\examples\Assignment 1\src
- Path to Junit test suite binary code – E:\jab\examples\Assignment 1\build\classes
- Test suite full qualified name – assignment1.CircleTest
- JaBUTi's Library – E:\jab\jabuti.jar
- Javac - C:\Program Files\Java\jdk1.7.0_21\bin\javac.exe

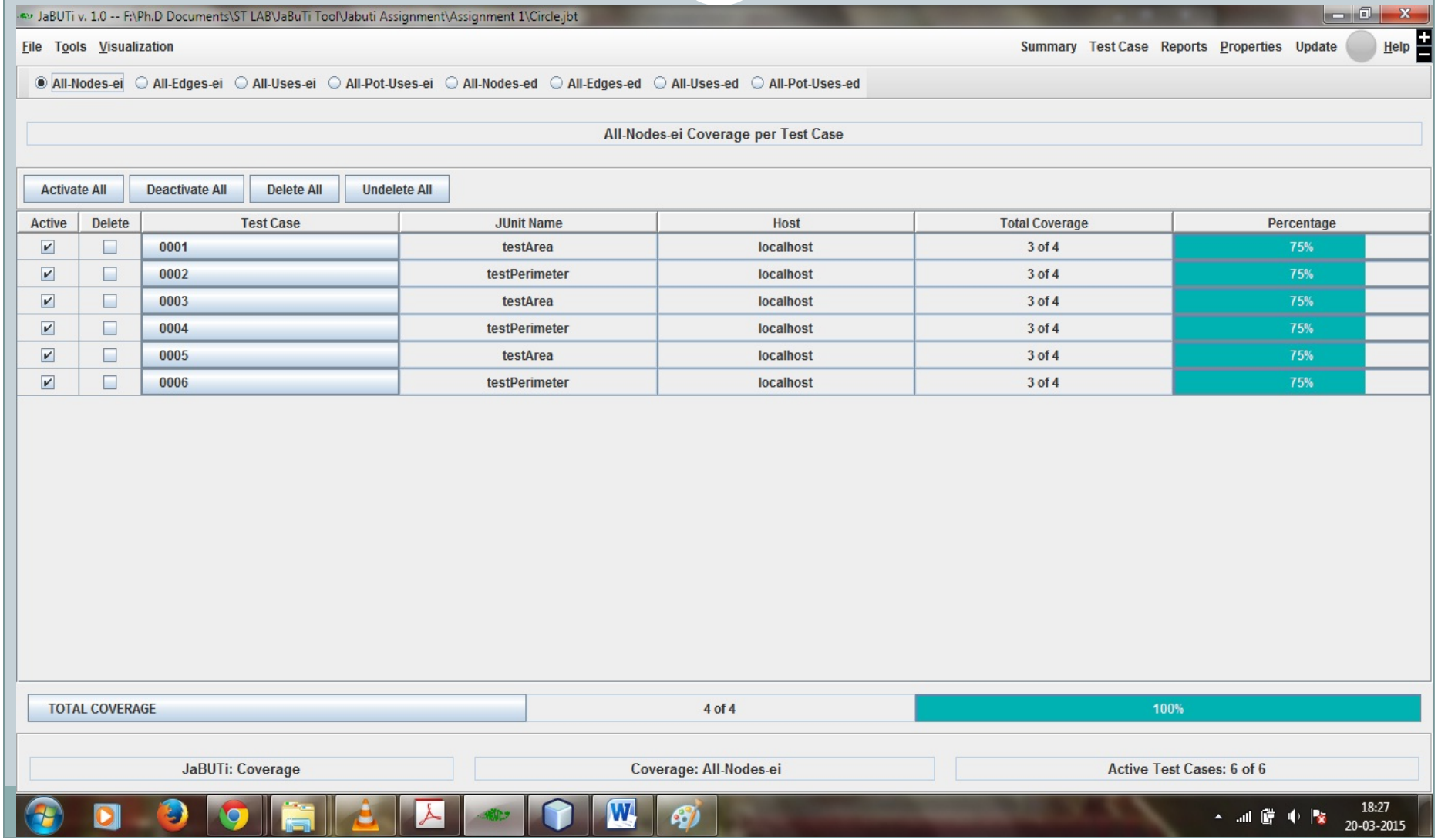
- Click on Compile Test Cases.

- Click on Run Collecting Trace Information.

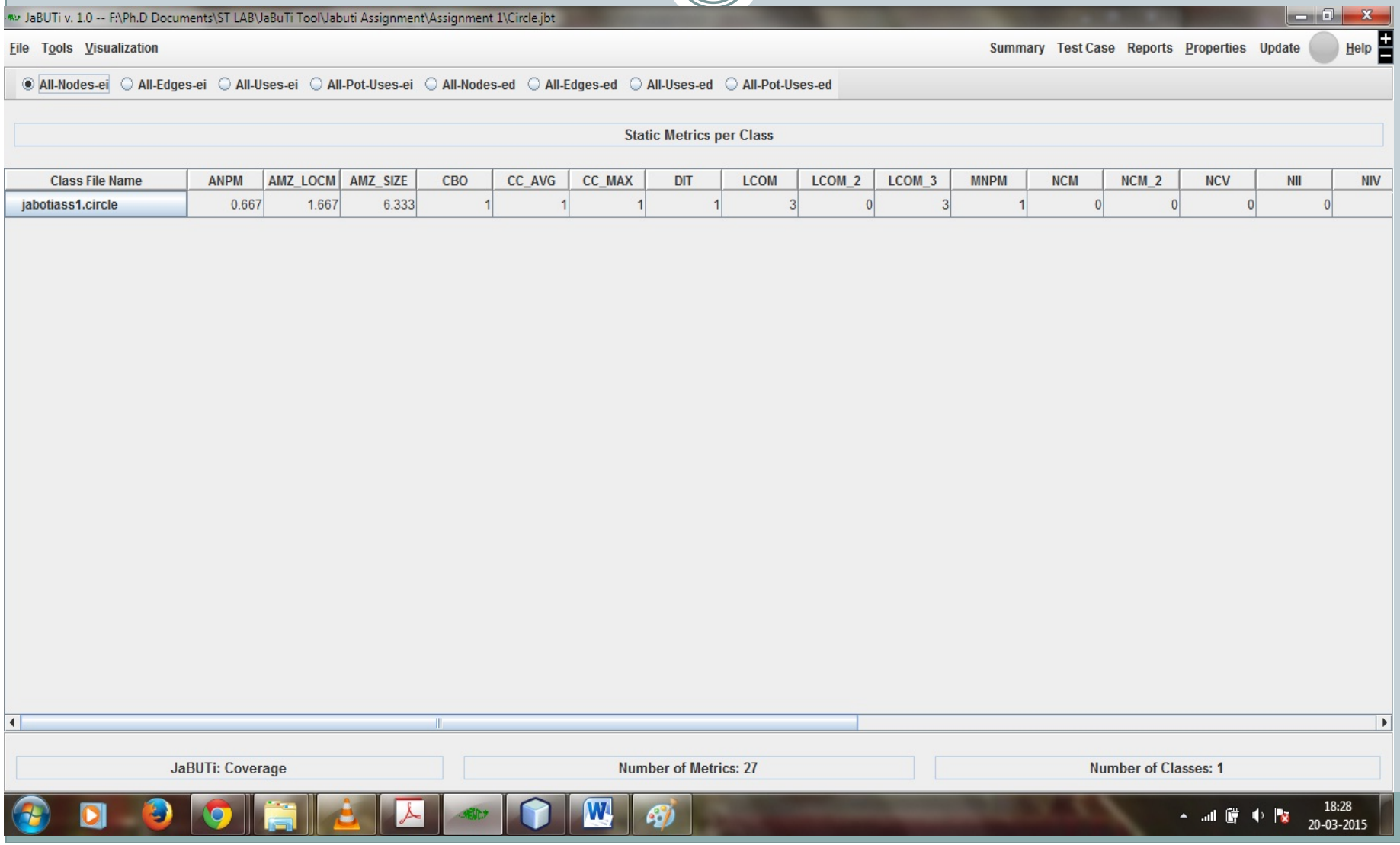


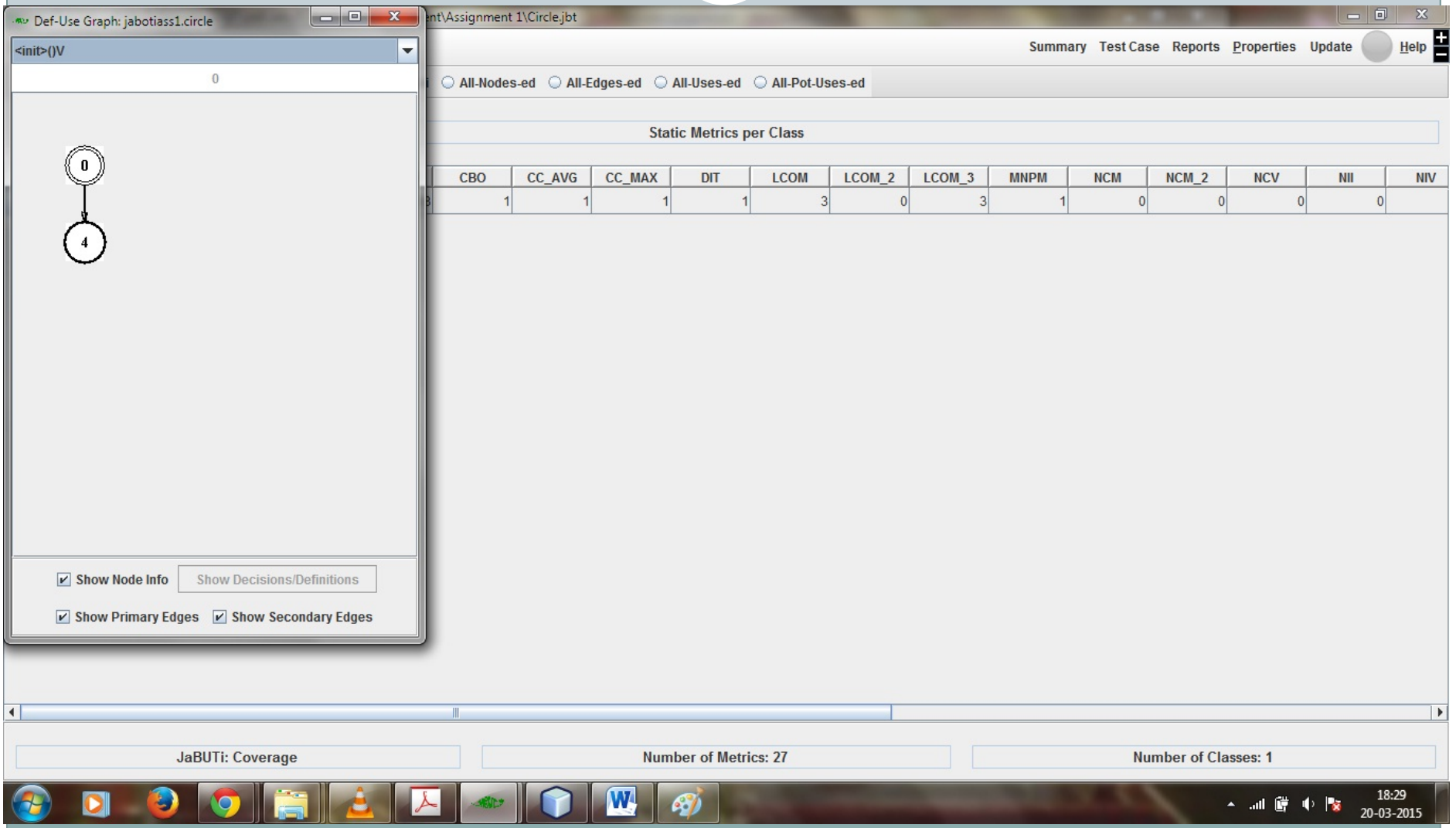
- Close the window and click on Update.




JaBUTi v. 1.0 -- F:\Ph.D Documents\ST LAB\JaBuTi Tool\Jabuti Assignment\Assignment 1\Circle.jbt
File Tools Visualization
Summary Test Case Reports Properties Update Help
All-Nodes-ei All-Edges-ei All-Uses-ei All-Pot-Uses-ei All-Nodes-ed All-Edges-ed All-Uses-ed All-Pot-Uses-ed
All-Nodes-ei Coverage per Test Case
Activate All
Deactivate All
Delete All
Undelete All
Active
Delete
Test Case
JUnit Name
Host
Total Coverage
Percentage
0001 testArea localhost 3 of 4 75%
0002 testPerimeter localhost 3 of 4 75%
0003 testArea localhost 3 of 4 75%
0004 testPerimeter localhost 3 of 4 75%
0005 testArea localhost 3 of 4 75%
0006 testPerimeter localhost 3 of 4 75%
TOTAL COVERAGE
4 of 4
100%
JaBUTi: Coverage
Coverage: All-Nodes-ei
Active Test Cases: 6 of 6
18:27
20-03-2015

Complexity metrics of circle example program.




Def-Use Graph: jabotiass1.circle
<init>()V
0
0
4
Show Node Info
Show Decisions/Definitions
Show Primary Edges
Show Secondary Edges
nt\Assignment 1\Circle.jbt
Summary
Test Case
Reports
Properties
Update
Help
All-Nodes-ed
All-Edges-ed
All-Uses-ed
All-Pot-Uses-ed
Static Metrics per Class
CBO
CC_AVG
CC_MAX
DIT
LCOM
LCOM_2
LCOM_3
MNPM
NCM
NCM_2
NCV
NII
NIV
1 1 1 1 3 0 3 1 0 0 0 0
JaBUTi: Coverage
Number of Metrics: 27
Number of Classes: 1
18:29
20-03-2015

THANK YOU