TESTING WITH JUNIT

Lab 3 : Testing

Overview

- Testing with JUnit
  - JUnit Basics
  - Sample Test Case
  - How To Write a Test Case
  - Running Tests with JUnit
- JUnit plug-in for NetBeans
  - Running Tests in NetBeans

Testing with JUnit

- JUnit is a simple testing framework for Java
- It can be used to define “test cases”, which can be grouped into “test suites”
- These tests can be run to get a pass/fail indication and a list of all failures
- Can be downloaded from:
  - http://www.junit.org

JUnit Basics

- To define test cases:
  - Create a new class *xxxTest* that
    - extends *TestCase*
    - and import *junit.framework.**
  - Define one or more testXXX() methods
  - Optionally define setUp() and tearDown() methods that are run before and after each test respectively
    - Can be used to initialize fields with test data common to all tests
  - Add static *suite()* and *main* methods

How to Write a Test Case

- Signature
  - Always start with *test*
  - Follow with name of method or functionality tested
  - No arguments or return value
- Body
  - Only test one point per method; keep it short
  - At the end, use one of the assert methods to check results:
    - assertEquals(exp, act) // checks equality
    - assertSame(exp, act) // checks identity
    - assertTrue(expr) // checks if condition is true
    - assertNull(obj) // checks if object is null
    - assertNotNull(obj) // checks if object is not null
    - fail() // fails (allows arbitrary check)
    - All these methods optionally take a failure message as the first argument.

JUnit Basics

- Consider the following class
-
```
public class Calculator{
    int sum(int num1,int num2){
        return num1+num2;
    }
}
```

JUnit Basics (cont…)

- To test that method sum() is working fine we need to check it.
- Create a new class *xxxTest* that
  - extends *TestCase*
  - and import *junit.framework.**
- So we create another class named CalculatorTest.

JUnit Basics (cont…)

- Coding Convention :
  - Name of the test class must end with "Test".
  - Name of the method must begin with "test".
  - Return type of a test method must be void.
  - Test method must not throw any exception.
  - Test method must not have any parameter.

Test Class

```
import junit.framework.TestCase;

  public class CalculatorTest extends TestCase {
     Calculator cal= new Calculator();

      public CalculatorTest(String name) {
            super(name);
      }

      public void testSum() {
            assertEquals(2,cal.sum(1,1));
     }
  }
```

Running Tests in NetBeans

- We will Use Linked List Example
  - Download from The lab website JLinkedList Project.
  - Open JlinkedList with NetBeans.

Testing JLinkedList with JUnit

- The JlinkedList consist of two classes:
  - Node.java
  - List.java

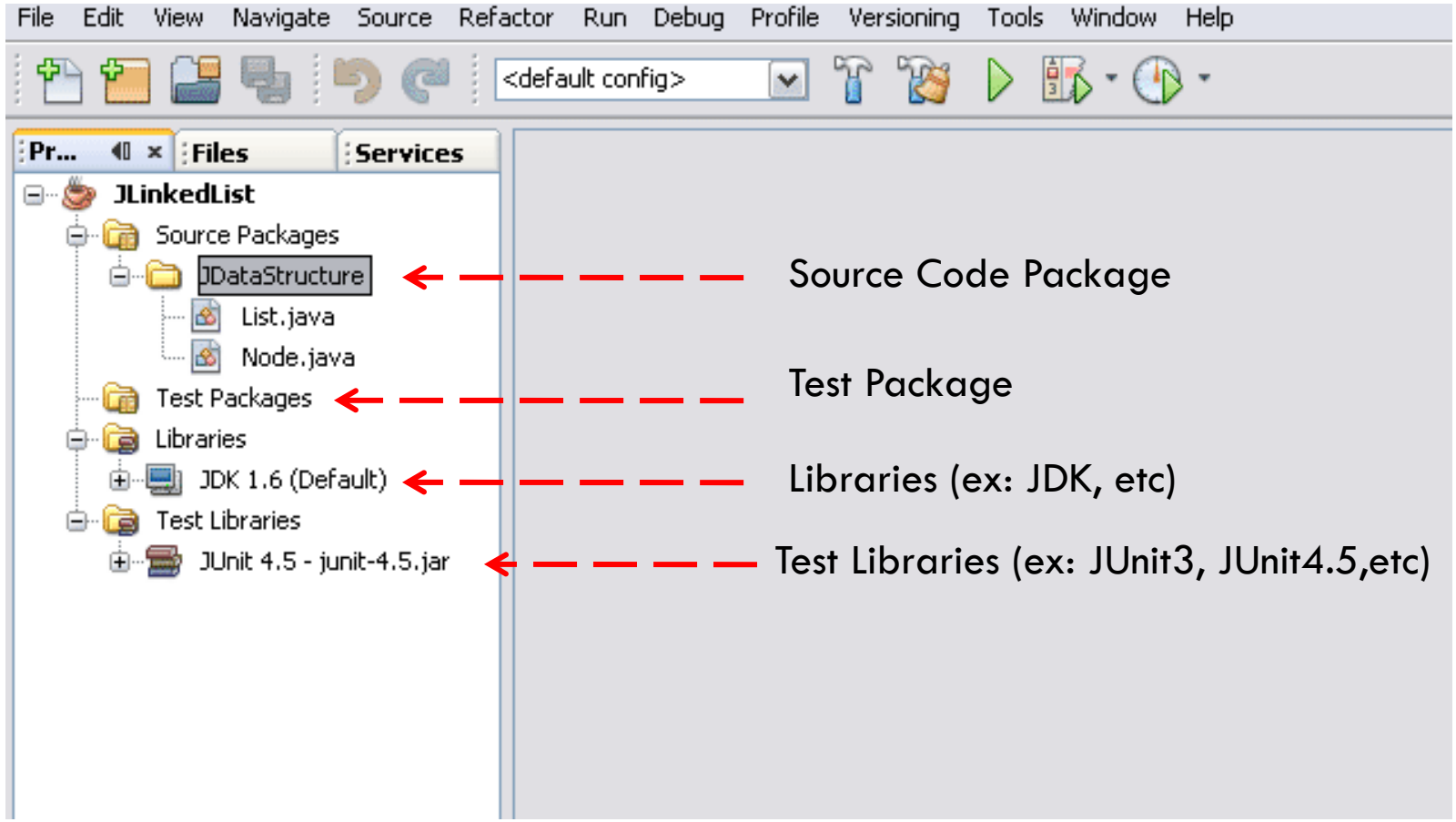
Node.java

```
1   package JDataStructure;
2
3   import java.io.Serializable;
4   /**
5    *
6    * @author Sherif Saad
7    */
8   public class Node implements Serializable{
9       private int data;
10      private Node next;
11
12      public Node() {...}
16
17      public void setX(int data) {...}
20
21      public int getX() {...}
24
25      public void setNext(Node next) {...}
28
29      public Node getNext() {...}
32
33  }
34
```

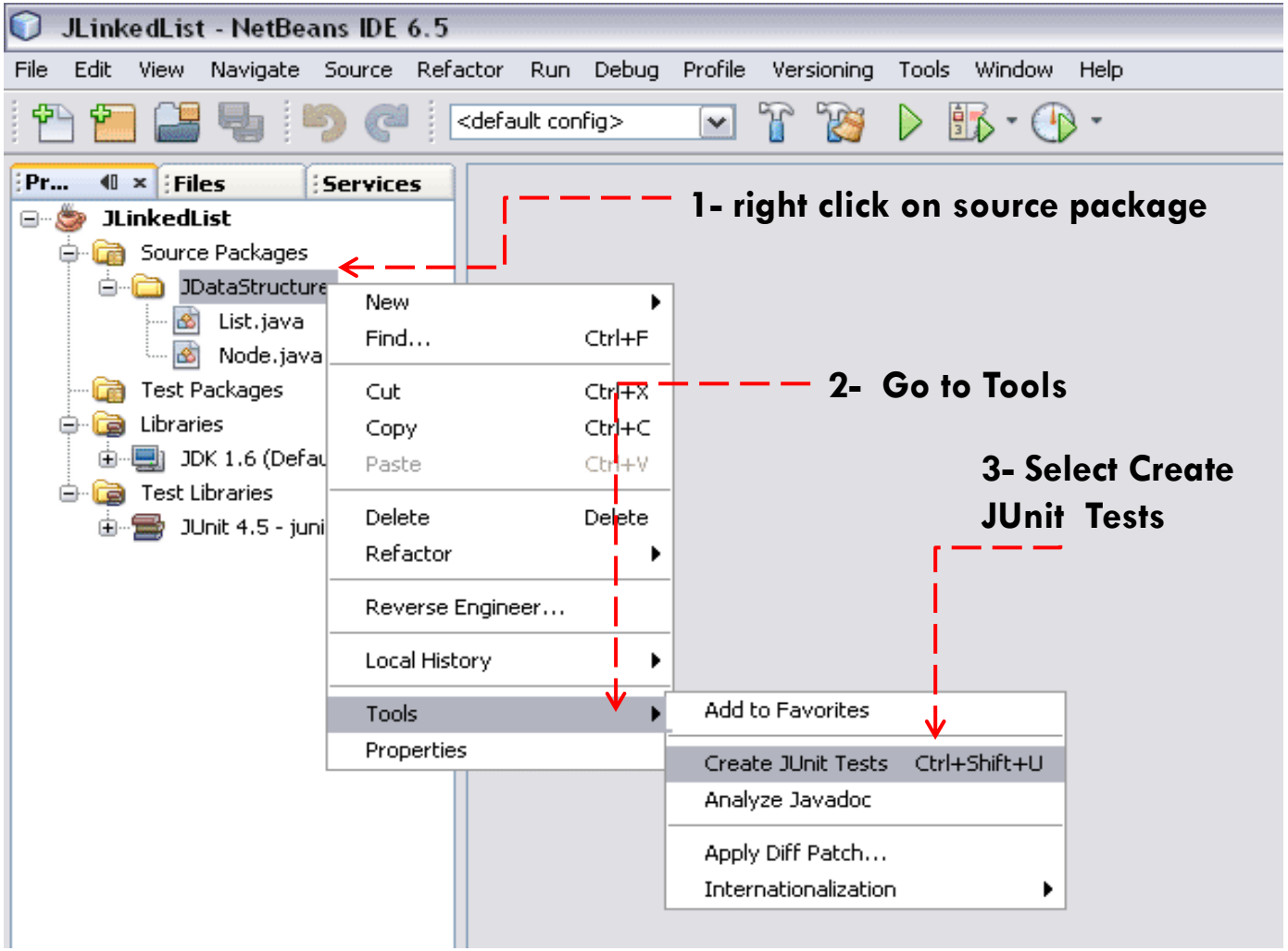
List.java

```
1    package JDataStructure;
2
3    import java.io.Serializable;
4    /**
5     *
6     * @author Sherif Saad
7     */
8    public class List implements Serializable {
9        private Node head;
10
11       public List() {...}
15
16       public boolean isEmpty() {...}
24
25       public void insertNode(int x) {...}
45
46       public void printList() {...}
57
58       public boolean deleteNode(int x) {...}
81
82       public Node search(int key) {...}
95
96       public int getSzie() {...}
108  }
109
```

Open JLinkedList Using NetBeans

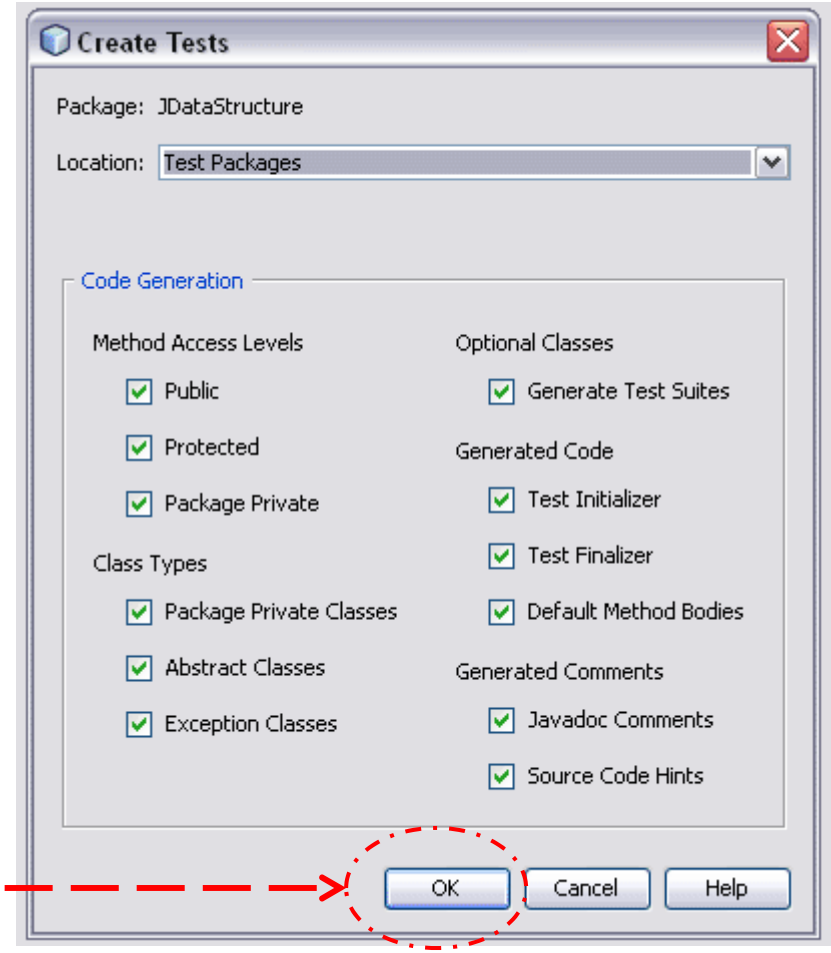


Create JUnit Test Classes

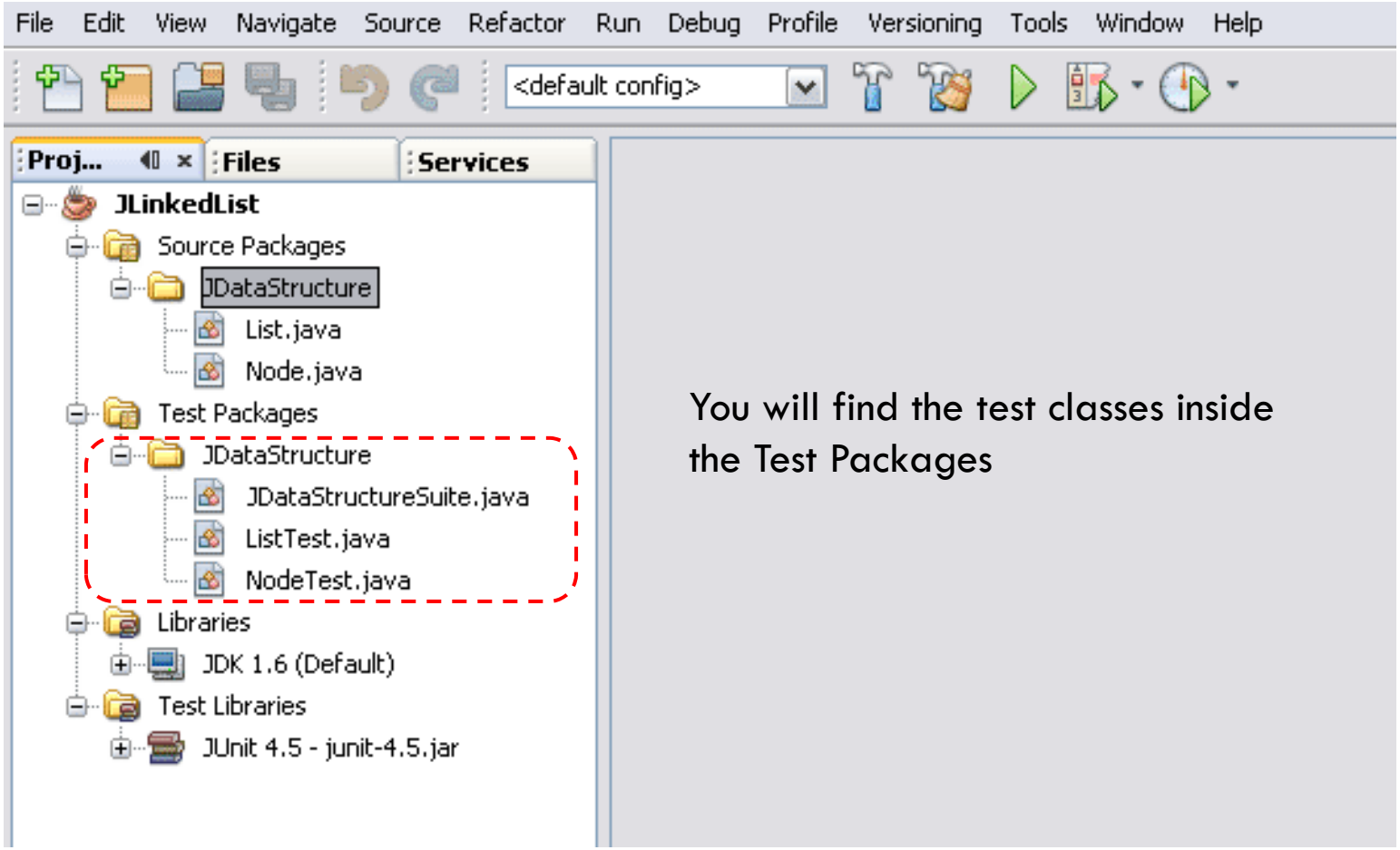


Choose Tests Properties

Keep Default Selections and click OK button



Generated Test Classes



You will find the test classes inside the Test Packages

Exploring the Test Classes

ListTest.java is the test class for List.java

```
1   package JDataStructure;
2   import org.junit.After;
3   import org.junit.AfterClass;
4   import org.junit.Before;
5   import org.junit.BeforeClass;
6   import org.junit.Test;
7   import static org.junit.Assert.*;
8
9   public class ListTest {
10      public ListTest() {...}
12      @BeforeClass
13      public static void setUpClass() throws Exception {...}
15      @AfterClass
16      public static void tearDownClass() throws Exception {...}
18      @Before
19      public void setUp() {...}
21      @After
22      public void tearDown() {...}
24      @Test
25      public void testIsEmpty() {...}
34      @Test
35      public void testInsertNode() {...}
43      @Test
44      public void testPrintList() {...}
51      @Test
52      public void testDeleteNode() {...}
62      @Test
63      public void testSearch() {...}
73      @Test
74      public void testGetSzie() {...}
83
84  }
```

Exploring the Test Classes

NodeTest.java is the test class for Node.java

```
1   package JDataStructure;
2
3   import org.junit.After;
4   import org.junit.AfterClass;
5   import org.junit.Before;
6   import org.junit.BeforeClass;
7   import org.junit.Test;
8   import static org.junit.Assert.*;
9
10  public class NodeTest {
11
12      public NodeTest() {...}
14
15      @BeforeClass
16      public static void setUpClass() throws Exception {...}
18      @AfterClass
19      public static void tearDownClass() throws Exception {...}
21      @Before
22      public void setUp() {...}
24      @After
25      public void tearDown() {...}
27      @Test
28      public void testSetX() {...}
36      @Test
37      public void testGetX() {...}
46      @Test
47      public void testSetNext() {...}
55      @Test
56      public void testGetNext() {...}
65
66  }
```

Exploring Test Function

Check *testIsEmpty()*, and *testInsertNode()* in the TestList.java Class

```
@Test
public void testIsEmpty() {
    System.out.println("isEmpty");
    List instance = new List();
    boolean expResult = false;
    boolean result = instance.isEmpty();
    assertEquals(expResult, result);
    // TODO review the generated test code and remove the default call to fail.
    fail("The test case is a prototype.");
}


@Test
public void testInsertNode() {
    System.out.println("insertNode");
    int x = 0;
    List instance = new List();
    instance.insertNode(x);
    // TODO review the generated test code and remove the default call to fail.
    fail("The test case is a prototype.");
}
```

Create Test Cases

- What are the possible test cases for ***isEmpty()*** function?
  - The list is empty then the function should return true.
  - The list is not empty then the function should return false.
- What are the possible test cases for ***insertNode()*** function?
  - The list is Empty and the node is the first node in the list.
  - The list has one or more nodes and the new node will be added to the end of the list.
  - The new node is already exist in the list, and so the insert operation will be ignored.

Test Cases for isEmpty()

```
@Test
public void testIsEmpty() {
    System.out.println("isEmpty");
    // Test the case that the list is an empty list
    List instance = new List();
    boolean expResult = true;
    boolean result = instance.isEmpty();
    assertEquals(expResult, result);

    //test the case that the list contain one or more element.
    expResult = false;
    instance.insertNode(13);
    result = instance.isEmpty();
    assertTrue(expResult==result);

    // TODO review the generated test code and remove the default call to fail.
    //fail("The test case is a prototype.");

}
```

Test Cases for InsertNode()

```java
46      public void testInsertNode() {
47          System.out.println("insertNode");
48          int x = 5;
49          int listSize =0;
50          List instance = new List();
51
52          //Case 1: test the case of inserting a new node in an empty list
53          if(instance.isEmpty()== true){
54              instance.insertNode(x);
55          }
56          else
57          {
58              fail(" @Case 1: The list is not empty");
59          }
60          assertTrue((instance.getSize()-listSize)==1);
61          listSize = instance.getSize();
62
63          //Case 2: test the case of inserting a new node in a non empty list
64          if(instance.isEmpty()== false){
65              instance.insertNode(7);
66          }
67          else
68          {
69              fail(" @Case 2: the list is empty");
70          }
71          assertTrue((instance.getSize()-listSize)==1);
72
73          //Case 3: test the case of inserting a node that already exist in the list
74          instance.insertNode(7);
75          assertTrue("Case 3: insert an item that already exist",(instance.getSize()-listSize)==0);
76          //TODO review the generated test code and remove the default call to fail.
77          //fail("The test case is a prototype.");
78      }
```

Test Cases for InsertNode()

```java
46      public void testInsertNode() {
47          System.out.println("insertNode");
48          int x = 5;
49          int listSize =0;
50          List instance = new List();
51
52          //Case 1: test the case of inserting a new node in an empty list
53          if(instance.isEmpty()== true){
54              instance.insertNode(x);
55          }
56          else
57          {
58              fail(" @Case 1: The list is not empty");
59          }
60          assertTrue((instance.getSize()-listSize)==1);
61          listSize = instance.getSize();
62
63          //Case 2: test the case of inserting a new node in a non empty list
64          if(instance.isEmpty()== false){
65              instance.insertNode(7);
66          }
67          else
68          {
69              fail(" @Case 2: the list is empty");
70          }
71          assertTrue((instance.getSize()-listSize)==1);
72          listSize = instance.getSize();
73
74          //Case 3: test the case of inserting a node that already exist in the list
75          instance.insertNode(7);
76          assertTrue("Case 3: insert an item that already exist",(instance.getSize()-listSize)==0);
77      }
```

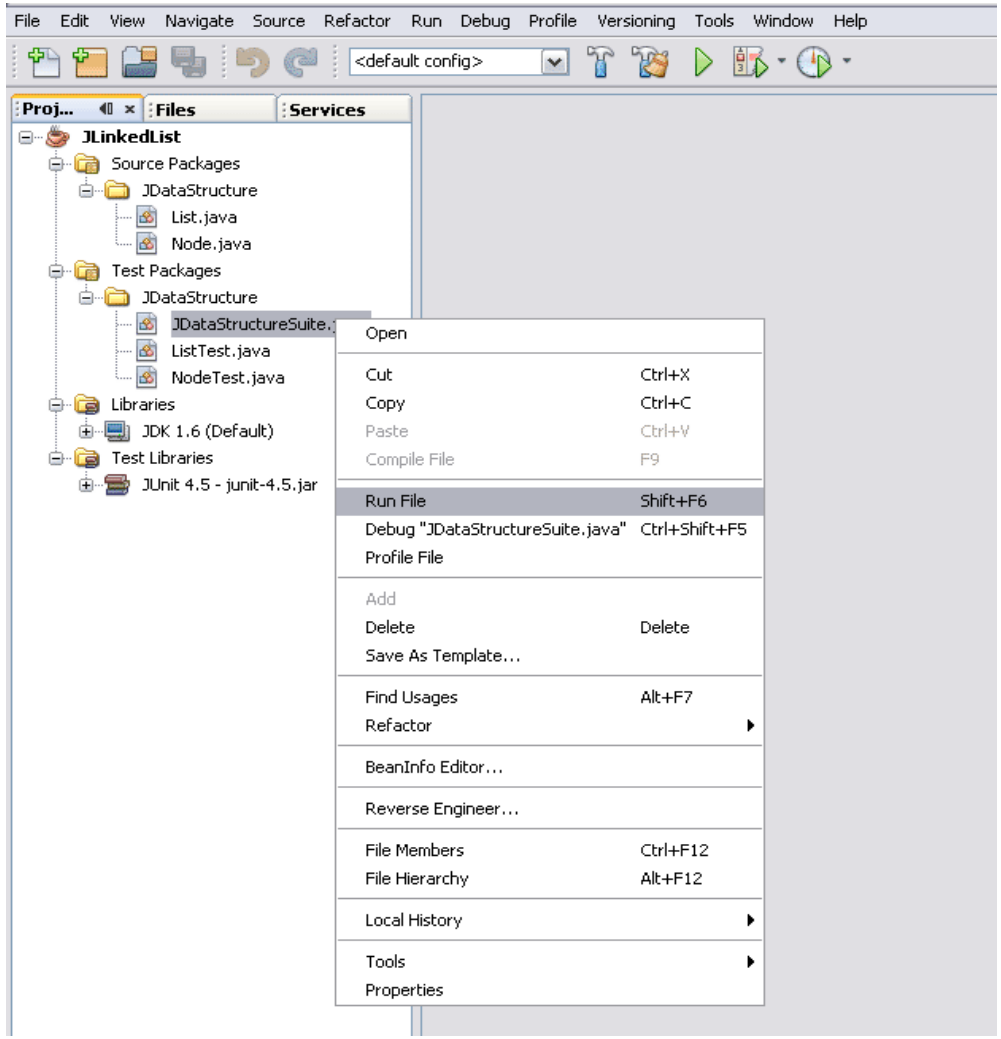
Watch your testing code you may inject more bugs

You need to update the listSize after each insert or your end up adding bug into your test code

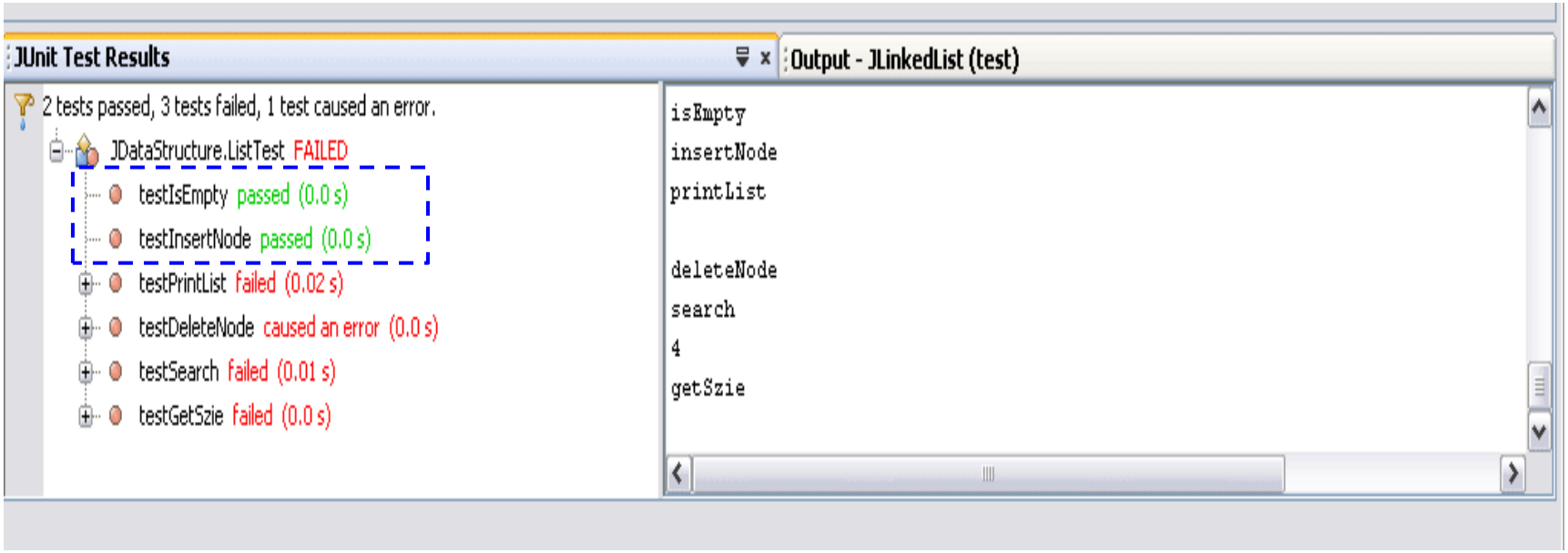
Execute Test Cases

You can execute your test cases by:

1. Right click on test Suite class and select Run File.
2. Right click on the test class and select Run File.
3. Right click on the Project Name and select Test.
4. Press Alt+F6 or Shift+F6



Test Result


JUnit Test Results
Output - JLinkedList (test)
2 tests passed, 3 tests failed, 1 test caused an error.
JDataStructure.ListTest FAILED
testIsEmpty passed (0.0 s)
testInsertNode passed (0.0 s)
testPrintList failed (0.02 s)
testDeleteNode caused an error (0.0 s)
testSearch failed (0.01 s)
testGetSzie failed (0.0 s)
isEmpty
insertNode
printList
deleteNode
search
4
getSzie

Questions?