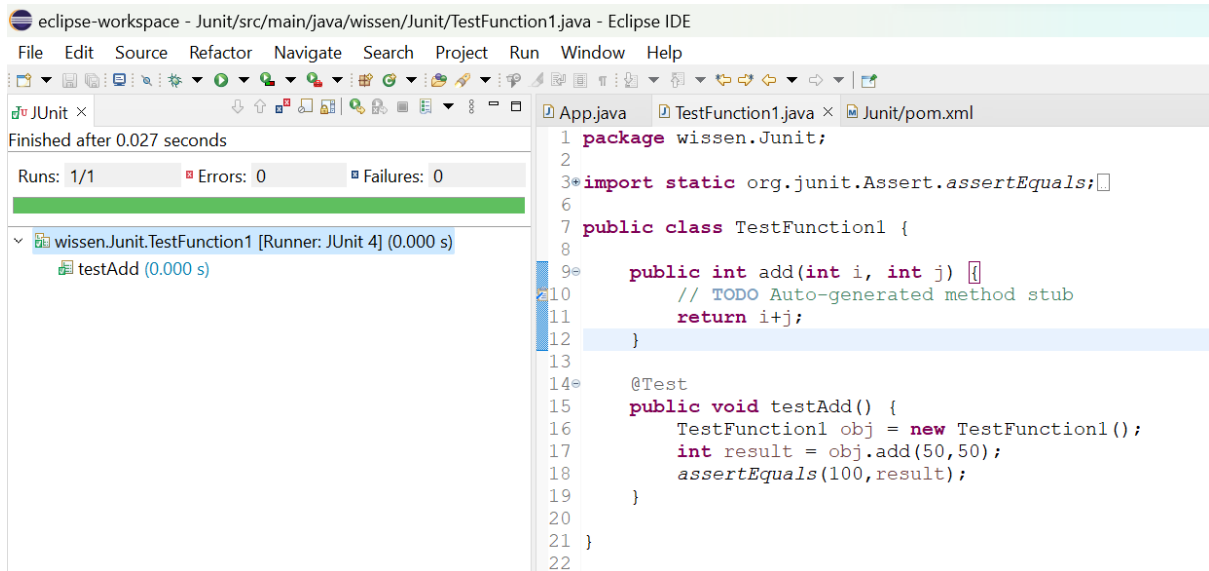


# Junit

Writing a function with JUnit.



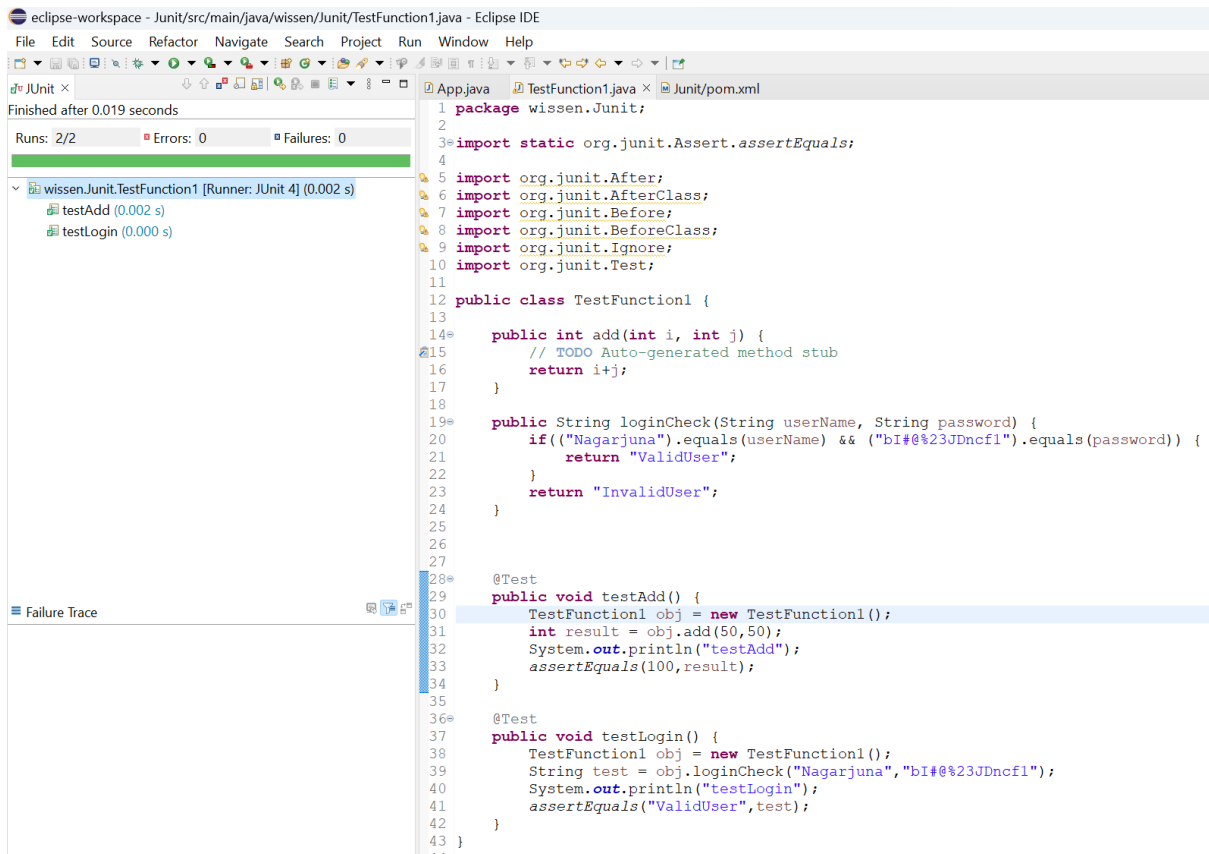
The screenshot shows the Eclipse IDE with a project named 'JUnit'. The main editor displays 'TestFunction1.java' with the following code:

```
1 package wissen.Junit;
2
3 import static org.junit.Assert.assertEquals;
4
5
6
7 public class TestFunction1 {
8
9     public int add(int i, int j) {
10         // TODO Auto-generated method stub
11         return i+j;
12     }
13
14     @Test
15     public void testAdd() {
16         TestFunction1 obj = new TestFunction1();
17         int result = obj.add(50,50);
18         assertEquals(100,result);
19     }
20
21 }
22
```

The left sidebar shows the 'JUnit' test runner with the following output:

```
JUnit
Finished after 0.027 seconds
Runs: 1/1 Errors: 0 Failures: 0
wissen.Junit.TestFunction1 [Runner: JUnit 4] (0.000 s)
  testAdd (0.000 s)
```

TDD Approach for login.



The screenshot shows the Eclipse IDE with a project named 'JUnit'. The main editor displays 'TestFunction1.java' with the following code:

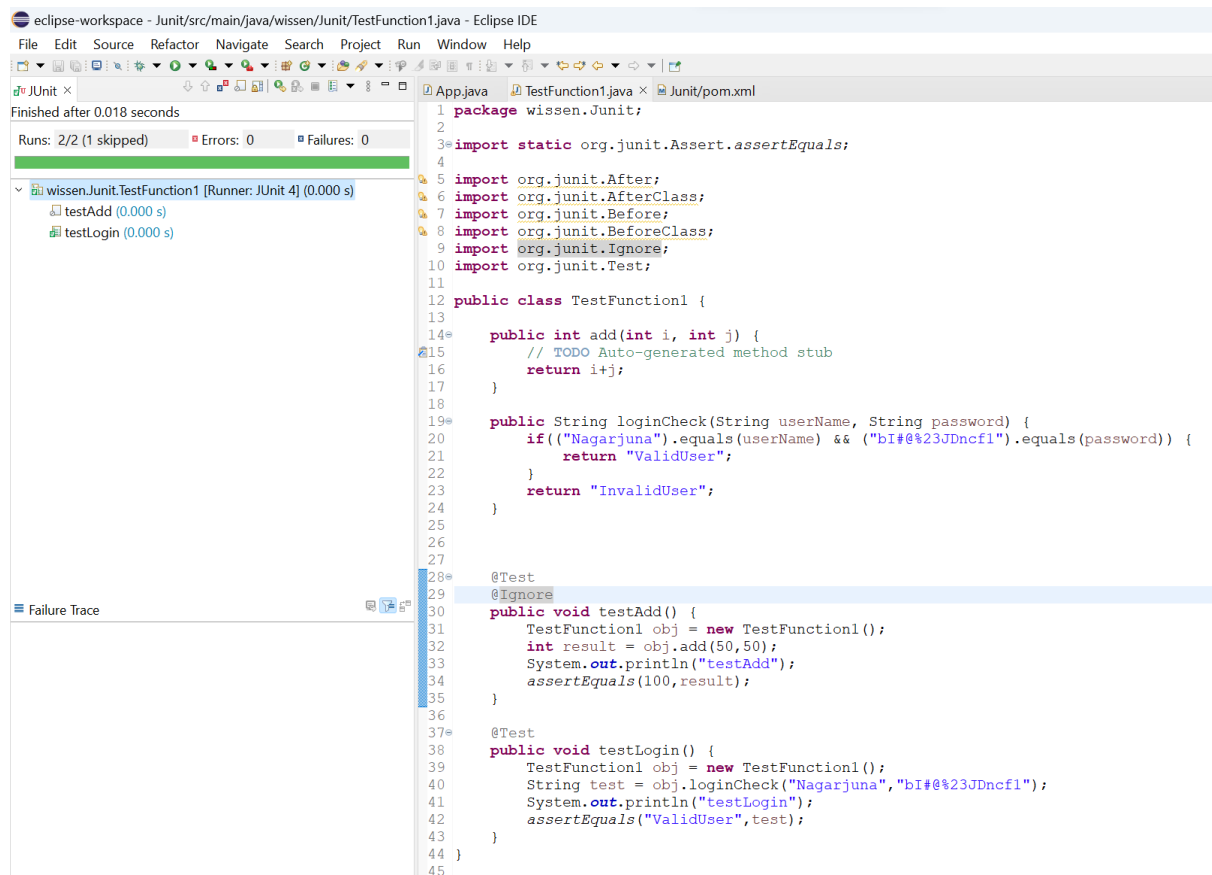
```
1 package wissen.Junit;
2
3 import static org.junit.Assert.assertEquals;
4
5 import org.junit.After;
6 import org.junit.AfterClass;
7 import org.junit.Before;
8 import org.junit.BeforeClass;
9 import org.junit.Ignore;
10 import org.junit.Test;
11
12 public class TestFunction1 {
13
14     public int add(int i, int j) {
15         // TODO Auto-generated method stub
16         return i+j;
17     }
18
19     public String loginCheck(String userName, String password) {
20         if(("Nagarjuna").equals(userName) && ("bI#@%23JDncf1").equals(password)) {
21             return "ValidUser";
22         }
23         return "InvalidUser";
24     }
25
26
27
28     @Test
29     public void testAdd() {
30         TestFunction1 obj = new TestFunction1();
31         int result = obj.add(50,50);
32         System.out.println("testAdd");
33         assertEquals(100,result);
34     }
35
36     @Test
37     public void testLogin() {
38         TestFunction1 obj = new TestFunction1();
39         String test = obj.loginCheck("Nagarjuna","bI#@%23JDncf1");
40         System.out.println("testLogin");
41         assertEquals("ValidUser",test);
42     }
43 }
44
```

The left sidebar shows the 'JUnit' test runner with the following output:

```
JUnit
Finished after 0.019 seconds
Runs: 2/2 Errors: 0 Failures: 0
wissen.Junit.TestFunction1 [Runner: JUnit 4] (0.002 s)
  testAdd (0.002 s)
  testLogin (0.000 s)
```

At the bottom left, there is a 'Failure Trace' section which is currently empty.

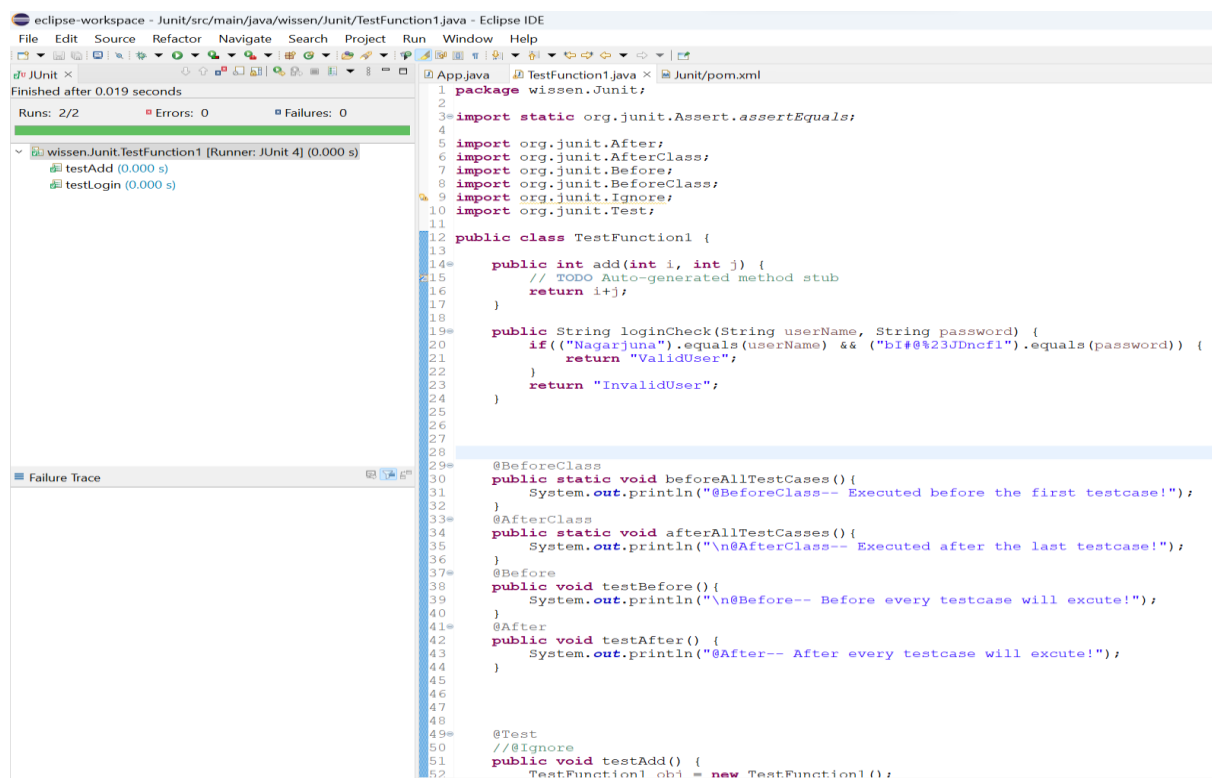
## Ignoring a test case



The screenshot shows the Eclipse IDE with a project named 'JUnit'. The main editor displays the file 'TestFunction1.java'. The code defines a class 'TestFunction1' with two methods: 'add' and 'loginCheck'. The 'add' method is annotated with '@Test' and '@Ignore', and the 'loginCheck' method is annotated with '@Test'. The 'add' method is currently selected, and its code is visible in the editor. The 'loginCheck' method is also visible. The 'JUnit' test runner is shown in the bottom-left corner, indicating that the test was executed successfully after 0.018 seconds, with 2/2 tests passed and 0 failures.

```
1 package wissen.Junit;
2
3 import static org.junit.Assert.assertEquals;
4
5 import org.junit.After;
6 import org.junit.AfterClass;
7 import org.junit.Before;
8 import org.junit.BeforeClass;
9 import org.junit.Ignore;
10 import org.junit.Test;
11
12 public class TestFunction1 {
13
14     public int add(int i, int j) {
15         // TODO Auto-generated method stub
16         return i+j;
17     }
18
19     public String loginCheck(String userName, String password) {
20         if(("Nagarjuna").equals(userName) && ("bI#@%23JDncf1").equals(password)) {
21             return "ValidUser";
22         }
23         return "InvalidUser";
24     }
25
26
27
28     @Test
29     @Ignore
30     public void testAdd() {
31         TestFunction1 obj = new TestFunction1();
32         int result = obj.add(50,50);
33         System.out.println("testAdd");
34         assertEquals(100,result);
35     }
36
37     @Test
38     public void testLogin() {
39         TestFunction1 obj = new TestFunction1();
40         String test = obj.loginCheck("Nagarjuna","bI#@%23JDncf1");
41         System.out.println("testLogin");
42         assertEquals("ValidUser",test);
43     }
44 }
45
```

## Setup and Teardown functions.



The screenshot shows the Eclipse IDE with a project named 'JUnit'. The main editor displays the file 'TestFunction1.java'. The code defines a class 'TestFunction1' with two methods: 'add' and 'loginCheck'. The 'add' method is annotated with '@Test' and '@Ignore', and the 'loginCheck' method is annotated with '@Test'. The 'add' method is currently selected, and its code is visible in the editor. The 'loginCheck' method is also visible. The 'JUnit' test runner is shown in the bottom-left corner, indicating that the test was executed successfully after 0.019 seconds, with 2/2 tests passed and 0 failures.

```
1 package wissen.Junit;
2
3 import static org.junit.Assert.assertEquals;
4
5 import org.junit.After;
6 import org.junit.AfterClass;
7 import org.junit.Before;
8 import org.junit.BeforeClass;
9 import org.junit.Ignore;
10 import org.junit.Test;
11
12 public class TestFunction1 {
13
14     public int add(int i, int j) {
15         // TODO Auto-generated method stub
16         return i+j;
17     }
18
19     public String loginCheck(String userName, String password) {
20         if(("Nagarjuna").equals(userName) && ("bI#@%23JDncf1").equals(password)) {
21             return "ValidUser";
22         }
23         return "InvalidUser";
24     }
25
26
27
28
29     @BeforeClass
30     public static void beforeAllTestCases() {
31         System.out.println("@BeforeClass-- Executed before the first testcase!");
32     }
33
34     @AfterClass
35     public static void afterAllTestCases() {
36         System.out.println("\n@AfterClass-- Executed after the last testcase!");
37     }
38
39     @Before
40     public void testBefore() {
41         System.out.println("\n@Before-- Before every testcase will excute!");
42     }
43
44     @After
45     public void testAfter() {
46         System.out.println("@After-- After every testcase will excute!");
47     }
48
49     @Test
50     // @Ignore
51     public void testAdd() {
52         TestFunction1 obj = new TestFunction1();
53     }
54 }
55
```

File Edit Source Refactor Navigate Search Project Run Window Help

```
<terminated> TestFunction1 [JUnit] C:\Users\akkis\.p2\pool\plugins\org.eclipse.jst.openide.hotspot.ireful
```

```
@Before-- Before every testcase will excute!
```

```
@After-- After every testcase will excute!
```

```
@Before-- Before every testcase will excute!
```

testLogin

```
@After-- After every testcase will excute!
```

```
@AfterClass-- Executed after the last testcase!
```

eclipse-workspace - Junit/src/main/java/wissen/Junit/ArraysComparision.java - Eclipse IDE

Project Explorer JUnit × App.java TestFunction1.java Junit/pom.xml ArraysComparision.java ×

```
1 package wissen.Junit;
```

Finished after 0.018 seconds

Runs: 1/1    Errors: 0    Failures: 0

```
2
3*import static org.junit.Assert.*
6
7 public class ArraysComparision {
```

```
8
9 @Test
10 public void arrayComparison() {
11     int[] array1= {1,2,3,4,5};
12     int[] array2= {1,2,3,4,5};
13     assertEquals(array1,array2);
14 }
15
16 }
17
```

✓ wissen.Junit.ArraysComparision [Runner: JUnit 4] (0.001 s)

arrayComparision (0.001 s)

## Checking exceptions in JUnit

The screenshot shows the Eclipse IDE interface. The top toolbar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, and Help. The Project Explorer on the left shows the project structure with 'JUnit' selected. The Run and Debug console shows the test results: 'Finished after 0.021 seconds', 'Runs: 3/3 (2 skipped)', 'Errors: 0', and 'Failures: 0'. The test suite 'wissen.Junit.TestFunction1 [Runner: JUnit 4] (0.000 s)' is expanded, showing three tests: 'testAdd (0.000 s)', 'testLogin (0.000 s)', and 'testCheckAgeForVoting (0.000 s)'. The main editor displays the source code of 'TestFunction1.java'. The code defines a package 'wissen.Junit', imports 'org.junit.Assert.assertEquals', and defines a class 'InvalidAgeForVoting' that extends 'Exception'. The 'TestFunction1' class contains methods for 'add', 'loginCheck', 'checkForAgeVoting' (which throws 'InvalidAgeForVoting' if age is less than 18), and test methods 'testBefore', 'testAfter', 'testAdd', 'testLogin', and 'testCheckAgeForVoting' (which asserts that 'checkForAgeVoting(16)' returns 'Invalid').

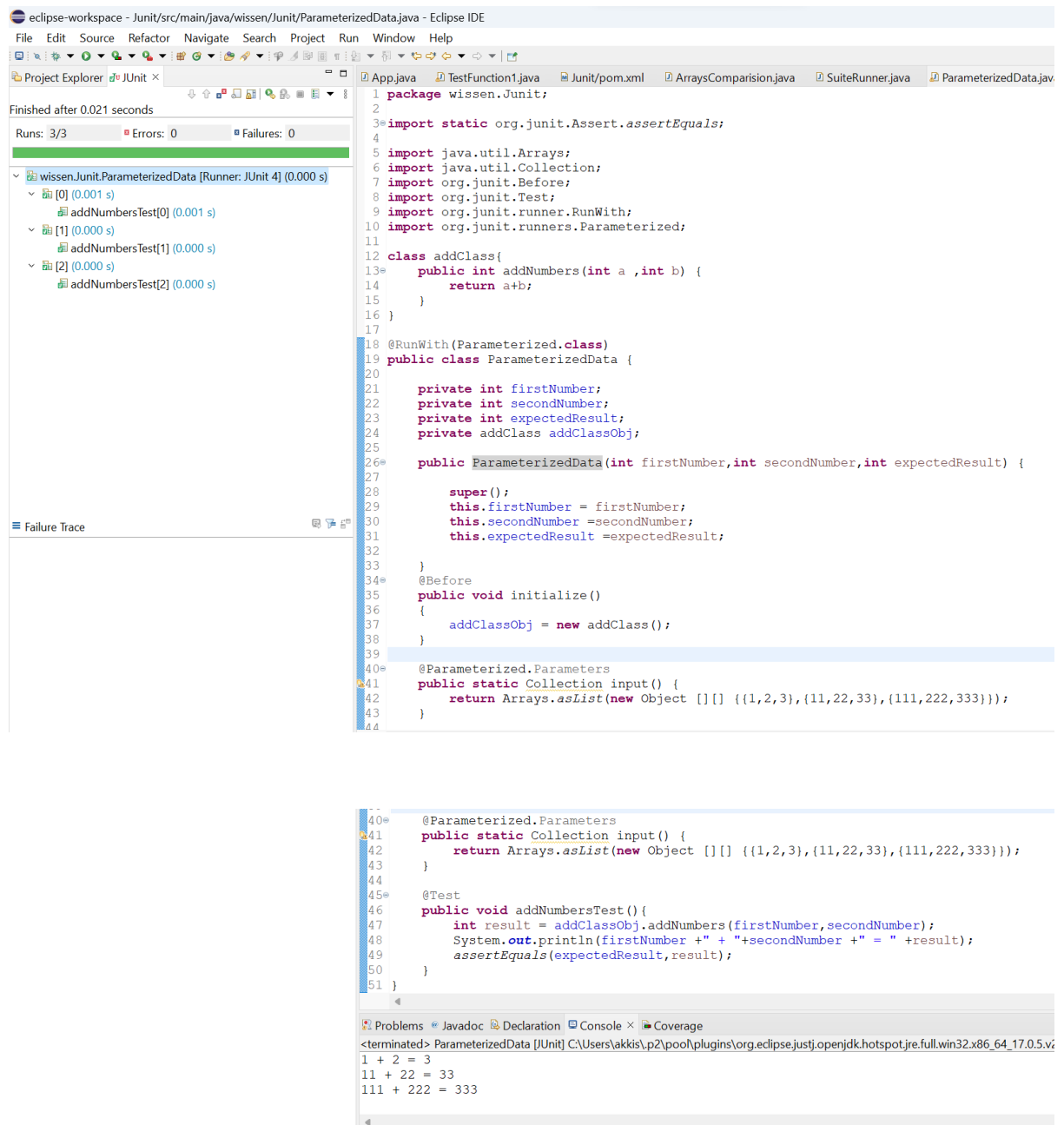
```
1 package wissen.Junit;
2
3 import static org.junit.Assert.assertEquals;
4
5
6
7
8
9
10
11
12 class InvalidAgeForVoting extends Exception{
13     public InvalidAgeForVoting(String msg) {
14         super(msg);
15     }
16 }
17
18 public class TestFunction1 {
19
20     public int add(int i, int j) {}
21
22     public String loginCheck(String userName, String password) {}
23
24
25     public static String checkForAgeVoting(int age) throws InvalidAgeForVoting{
26         if(age<18) {
27             throw new InvalidAgeForVoting("You can't vote as your age is less than 18");
28         }
29         return "valid";
30     }
31
32     public static void beforeAllTestCases() {}
33     public static void afterAllTestCases() {}
34     public void testBefore() {}
35     public void testAfter() {}
36
37
38     public void testAdd() {}
39
40     public void testLogin() {}
41
42     @Test(expected = InvalidAgeForVoting.class)
43     public void testCheckAgeForVoting() throws InvalidAgeForVoting {
44         System.out.println("Testing the function testCheckAgeForVoting");
45         assertEquals(TestFunction1.checkForAgeVoting(16), "Invalid");
46     }
47 }
```

## Suite Runner

The screenshot shows the Eclipse IDE interface. The top toolbar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, and Help. The Project Explorer on the left shows the project structure with 'JUnit' selected. The Run and Debug console shows the test results: 'Finished after 0.025 seconds', 'Runs: 4/4 (2 skipped)', 'Errors: 0', and 'Failures: 0'. The test suite 'wissen.Junit.SuiteRunner [Runner: JUnit 4] (0.000 s)' is expanded, showing four tests: 'wissen.Junit.TestFunction1 (0.000 s)', 'wissen.Junit.ArraysComparision (0.000 s)', 'wissen.Junit.TestFunction1 (0.000 s)', and 'wissen.Junit.ArraysComparision (0.000 s)'. The main editor displays the source code of 'SuiteRunner.java'. The code defines a package 'wissen.Junit', imports 'org.junit.runner.RunWith', 'org.junit.runners.Suite', and 'org.junit.runners.Suite.SuiteClasses', and defines a class 'SuiteRunner' that runs 'TestFunction1.class' and 'ArraysComparision.class'.

```
1 package wissen.Junit;
2
3 import org.junit.runner.RunWith;
4 import org.junit.runners.Suite;
5 import org.junit.runners.Suite.SuiteClasses;
6
7 @RunWith (Suite.class)
8 @SuiteClasses ({TestFunction1.class, ArraysComparision.class})
9 public class SuiteRunner {
10
11 }
12 }
```

## Parameterized Data



The screenshot displays the Eclipse IDE interface during a JUnit test run. The top toolbar includes standard IDE actions like File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, and Help. The Project Explorer on the left shows the test run results for `wissen.JUnit.ParameterizedData` [Runner: JUnit 4] (0.000 s). The test run is successful, with 3/3 runs, 0 errors, and 0 failures. The main editor shows the source code of `ParameterizedData.java`. The code defines a `ParameterizedData` class that implements `Parameterized` and `Test` interfaces. It includes a `addNumbers` method and a `ParameterizedData` constructor. The `input` method returns a collection of test data. The `addNumbersTest` method is annotated with `@Test` and `@Parameterized.Parameters`. The bottom console shows the output of the test run, including the test results and the output of the `addNumbersTest` method.

```
1 package wissen.JUnit;
2
3 import static org.junit.Assert.assertEquals;
4
5 import java.util.Arrays;
6 import java.util.Collection;
7 import org.junit.Before;
8 import org.junit.Test;
9 import org.junit.runner.RunWith;
10 import org.junit.runners.Parameterized;
11
12 class addClass{
13     public int addNumbers(int a ,int b) {
14         return a+b;
15     }
16 }
17
18 @RunWith(Parameterized.class)
19 public class ParameterizedData {
20
21     private int firstNumber;
22     private int secondNumber;
23     private int expectedResult;
24     private addClass addClassObj;
25
26     public ParameterizedData(int firstNumber,int secondNumber,int expectedResult) {
27
28         super();
29         this.firstNumber = firstNumber;
30         this.secondNumber =secondNumber;
31         this.expectedResult =expectedResult;
32
33     }
34     @Before
35     public void initialize()
36     {
37         addClassObj = new addClass();
38     }
39
40     @Parameterized.Parameters
41     public static Collection input() {
42         return Arrays.asList(new Object [][] { {1,2,3}, {11,22,33}, {111,222,333}});
43     }
44
45     @Test
46     public void addNumbersTest(){
47         int result = addClassObj.addNumbers(firstNumber,secondNumber);
48         System.out.println(firstNumber + " + " +secondNumber + " = " +result);
49         assertEquals(expectedResult,result);
50     }
51 }
```

Failure Trace

Problems Javadoc Declaration Console Coverage

<terminated> ParameterizedData [JUnit] C:\Users\lakkis\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86\_64\_17.0.5.v

1 + 2 = 3  
11 + 22 = 33  
111 + 222 = 333

\*\*\*  
tddf Test driven development.

test a little, Code a little.

First will write a test case & afterwards  
write a code. It improves code strength  
(according to that test case)

JUnit

It is a testing framework for Java programs.  
It plays a crucial role test-driven develop-  
ment, and is a family of unit testing  
frameworks collectively known as JUnit.  
First testing then Coding.

It reduces the time spent on debugging.