

# ASSIGNMENT

B. LIKHITHA

## Example 1

### Test User Login Validation

#### Problem Statement:

Check whether login is successful when username and password are correct.

#### Method Logic:

Returns true if username is "admin" and password is "admin123".

#### Test Focus:

- Boolean result
- assertTrue, assertFalse

#### Expected Outcome:

Valid credentials → pass

Invalid credentials → fail

The screenshot shows the Eclipse IDE interface with two main panes. The left pane displays the Java code for the `Login` class, which contains a static method `validate` that returns `true` if the username is "admin" and the password is "admin123". The right pane shows the `JUnit` view in the `Package Explorer`. It displays the results of a JUnit run: 2 tests passed, 0 errors, and 0 failures. The test class `LoginTest` contains two test methods: `testValidLogin` and `testInvalidLogin`. The `testValidLogin` method creates a new `Login` object and calls its `validate` method with the arguments "admin" and "admin123", then asserts that the result is `true`. The `testInvalidLogin` method creates a new `Login` object and calls its `validate` method with the arguments "user" and "1234", then asserts that the result is `false`.

```
1 package com.example.code;
2
3 public class Login {
4     public boolean validate(String username, String password) {
5         return username.equals("admin") && password.equals("admin123");
6     }
7 }
8
```

```
Finished after 0.026 seconds
Runs: 2/2      Errors: 0      Failures: 0
```

```
1 package com.example.test;
2
3 import static org.junit.Assert.*;
4 import org.junit.Test;
5 import com.example.code.Login;
6
7 public class LoginTest {
8
9     @Test
10    public void testValidLogin() {
11        Login l = new Login();
12        assertTrue(l.validate("admin", "admin123"));
13    }
14
15    @Test
16    public void testInvalidLogin() {
17        Login l = new Login();
18        assertFalse(l.validate("user", "1234"));
19    }
20 }
```

## Example 2

### Test Age Eligibility

#### Problem Statement:

Verify whether a user is eligible to vote.

#### Method Logic:

Returns true if age  $\geq 18$ .

#### Test Focus:

- Conditional logic
- assertTrue, assertFalse

#### Expected Outcome:

Age  $\geq 18 \rightarrow$  eligible

Age  $< 18 \rightarrow$  not eligible

```
1 package com.example.test;
2
3 import static org.junit.Assert.*;
4 import org.junit.Test;
5 import com.example.code.ExampleFunctions;
6
7 public class ExampleTests {
8
9     @Test
10    public void testAgeEligibility() {
11        ExampleFunctions f = new ExampleFunctions();
12        assertTrue(f.isEligible(20));
13        assertFalse(f.isEligible(16));
14    }
15}
16
```

## Example 3

### Test String Uppercase Conversion

#### Problem Statement:

Convert a string to uppercase and test the output. **Method Logic:**

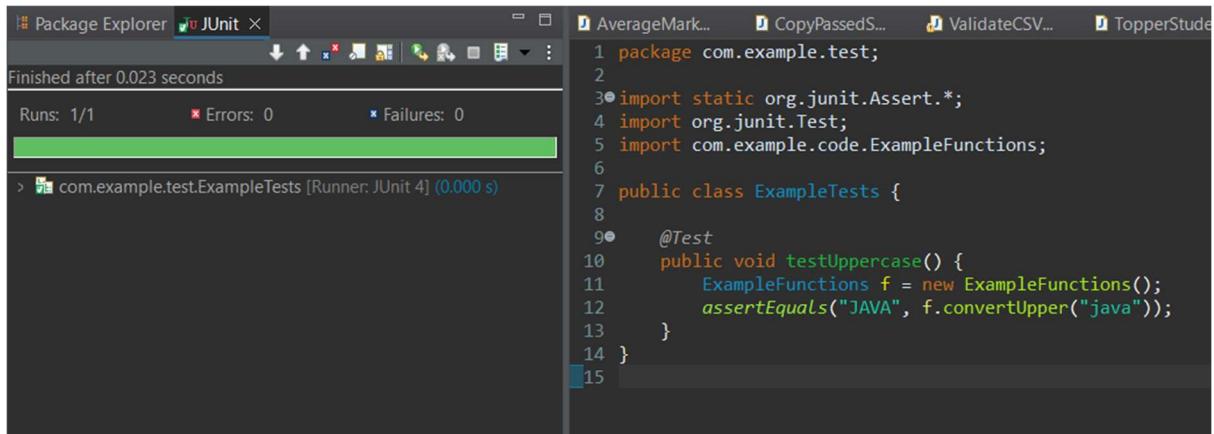
Returns input string in uppercase.

#### Test Focus:

- String comparison
- assertEquals

#### Expected Outcome:

Input "java" → Output "JAVA"



The screenshot shows the Eclipse IDE interface. On the left, the 'Package Explorer' view shows a single test run: 'com.example.test.ExampleTests [Runner: JUnit 4] (0.000 s)'. The status bar indicates 'Runs: 1/1', 'Errors: 0', and 'Failures: 0'. On the right, the code editor displays the following Java code:

```
1 package com.example.test;
2
3 import static org.junit.Assert.*;
4 import org.junit.Test;
5 import com.example.code.ExampleFunctions;
6
7 public class ExampleTests {
8
9     @Test
10    public void testUppercase() {
11        ExampleFunctions f = new ExampleFunctions();
12        assertEquals("JAVA", f.convertUpper("java"));
13    }
14 }
15
```

## Example 4

### Test Email Validation

#### Problem Statement:

Check whether an email contains @ symbol.

#### Method Logic:

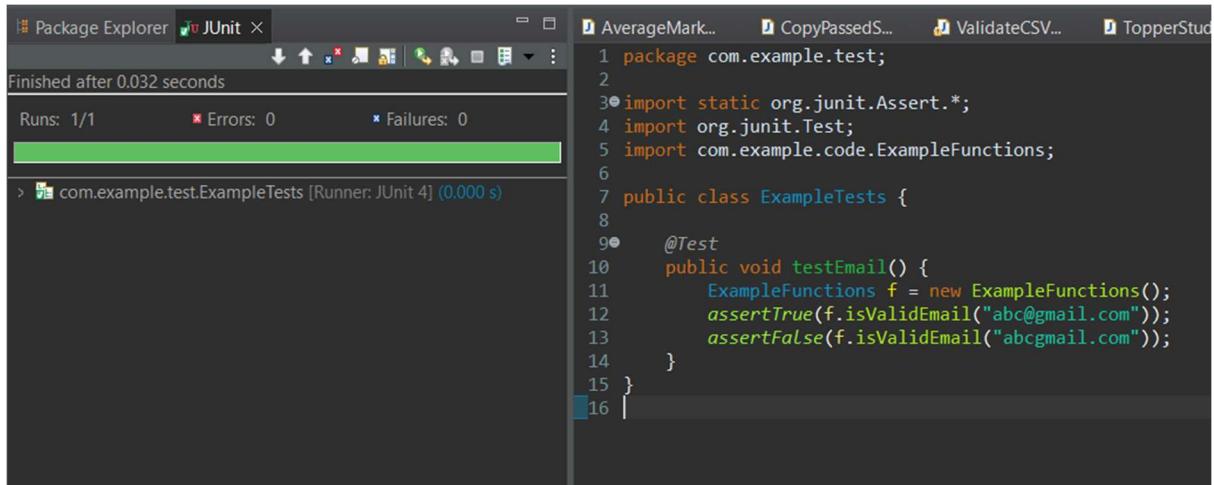
Returns true if email contains "@".

#### Test Focus:

- Basic string check
- assertTrue

#### Expected Outcome:

Valid email passes test.



The screenshot shows the Eclipse IDE interface. On the left, the 'Package Explorer' view shows a single test run: 'com.example.test.ExampleTests [Runner: JUnit 4] (0.000 s)'. The status bar indicates 'Runs: 1/1', 'Errors: 0', and 'Failures: 0'. On the right, the code editor displays the following Java code:

```
1 package com.example.test;
2
3 import static org.junit.Assert.*;
4 import org.junit.Test;
5 import com.example.code.ExampleFunctions;
6
7 public class ExampleTests {
8
9     @Test
10    public void testEmail() {
11        ExampleFunctions f = new ExampleFunctions();
12        assertTrue(f.isValidEmail("abc@gmail.com"));
13        assertFalse(f.isValidEmail("abcgmail.com"));
14    }
15 }
16
```

## Example 5

## Test Password Length Rule

### Problem Statement:

Check whether password length is at least 8 characters.

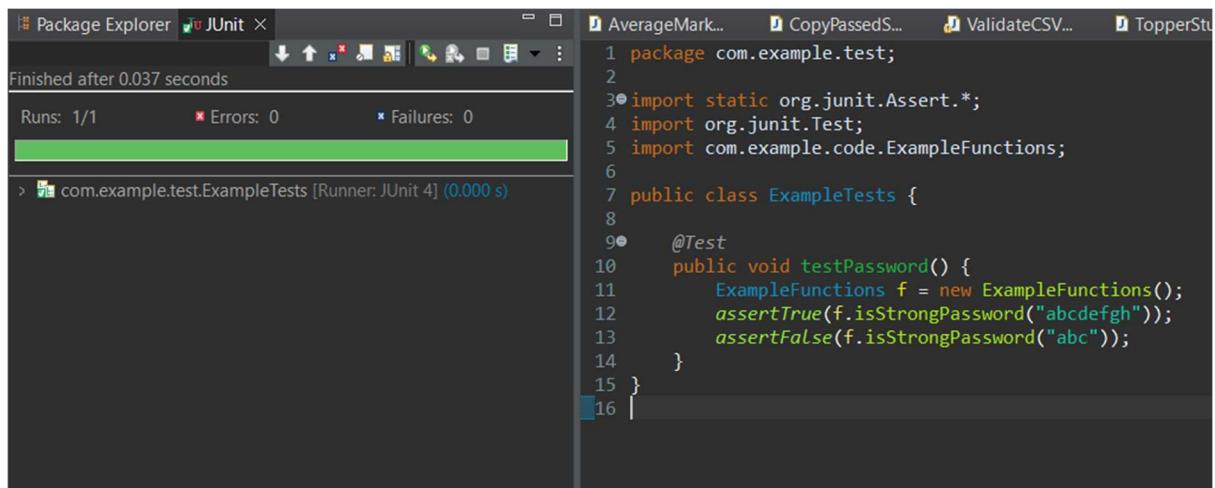
### Method Logic:

Returns true if password length  $\geq 8$ .

### Test Focus:

- Validation logic
  - `assertTrue`, `assertFalse`
- Expected Outcome:**

Short password fails.



The screenshot shows the Eclipse IDE interface. On the left, the 'Package Explorer' view is visible. In the center, the 'JUnit' view displays the test results: 'Finished after 0.037 seconds', 'Runs: 1/1', 'Errors: 0', and 'Failures: 0'. A green progress bar indicates the test has passed. On the right, the code editor shows the Java test class 'ExampleTests'. The code defines a single test method 'testPassword' that creates an instance of 'ExampleFunctions', calls its 'isStrongPassword' method with the argument 'abcdefg', and asserts that the result is true. It also calls the same method with the argument 'abc' and asserts that the result is false.

```
1 package com.example.test;
2
3 import static org.junit.Assert.*;
4 import org.junit.Test;
5 import com.example.code.ExampleFunctions;
6
7 public class ExampleTests {
8
9     @Test
10    public void testPassword() {
11        ExampleFunctions f = new ExampleFunctions();
12        assertTrue(f.isStrongPassword("abcdefg"));
13        assertFalse(f.isStrongPassword("abc"));
14    }
15 }
16 }
```

## Example 6

### Test File Name Extension

### Problem Statement:

Check whether file name ends with .csv.

### Method Logic:

Returns true if file name ends with .csv.

### Test Focus:

- String method testing
- `assertTrue`

### Expected Outcome:

CSV file passes test.

The screenshot shows the Eclipse IDE interface. On the left, the 'JUnit' view displays a summary: 'Finished after 0.037 seconds', 'Runs: 1/1', 'Errors: 0', and 'Failures: 0'. Below this, it lists a single test: 'com.example.test.ExampleTests [Runner: JUnit 4] (0.000 s)'. On the right, the code editor shows the following Java code:

```
1 package com.example.test;
2
3 import static org.junit.Assert.*;
4 import org.junit.Test;
5 import com.example.code.ExampleFunctions;
6
7 public class ExampleTests {
8
9     @Test
10    public void testCSV() {
11        ExampleFunctions f = new ExampleFunctions();
12        assertTrue(f.isCSV("data.csv"));
13        assertFalse(f.isCSV("data.txt"));
14    }
15 }
16
```

## Example 7

### Test Null Check

#### Problem Statement:

Return a username and ensure it is not null.

#### Method Logic:

Returns hardcoded username.

#### Test Focus:

- Null validation
- assertNotNull

#### Expected Outcome:

Username should not be null.

The screenshot shows the Eclipse IDE interface. On the left, the 'JUnit' view displays a summary: 'Finished after 0.032 seconds', 'Runs: 1/1', 'Errors: 0', and 'Failures: 0'. Below this, it lists a single test: 'com.example.test.ExampleTests [Runner: JUnit 4] (0.001 s)'. On the right, the code editor shows the following Java code:

```
1 package com.example.test;
2
3 import static org.junit.Assert.*;
4 import org.junit.Test;
5 import com.example.code.ExampleFunctions;
6
7 public class ExampleTests {
8
9     @Test
10    public void testNotNull() {
11        ExampleFunctions f = new ExampleFunctions();
12        assertNotNull(f.getUsername());
13    }
14 }
15
```

## Example 8

### Test List SizeProblem Statement:

Add 3 items to a list and verify its size.

**Method Logic:**

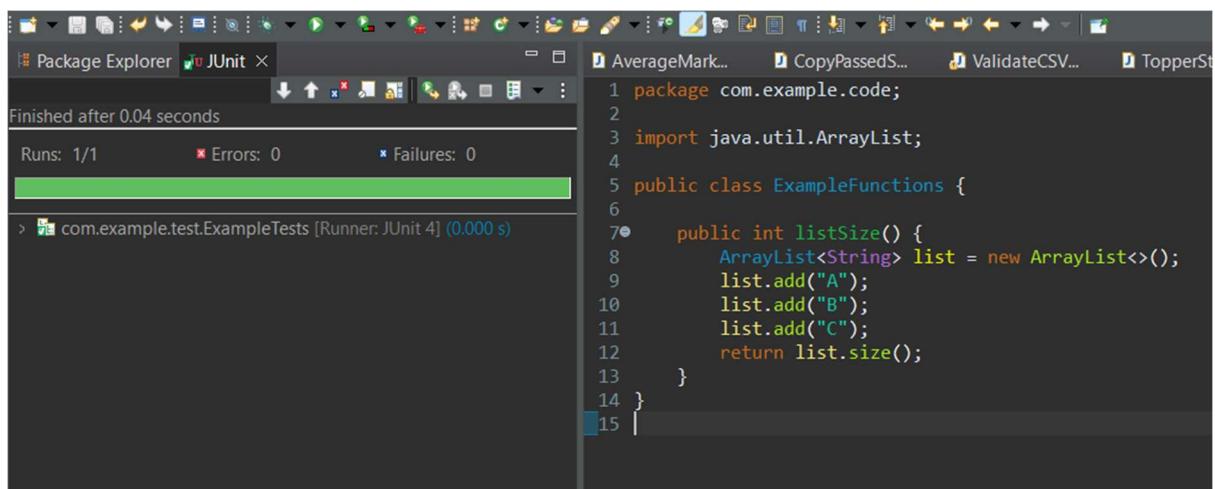
Returns list size.

**Test Focus:**

- Collection size
- assertEquals

**Expected Outcome:**

List size should be 3.



The screenshot shows the Eclipse IDE interface. In the top left, the 'Package Explorer' and 'JUnit' perspectives are visible. The 'JUnit' perspective is active, showing a green bar indicating 'Finished after 0.04 seconds' with 'Runs: 1/1', 'Errors: 0', and 'Failures: 0'. Below this, a list of tests is shown, with one entry for 'com.example.test.ExampleTests [Runner: JUnit 4] (0.000 s)'. On the right side of the interface, the Java code for 'ExampleFunctions' is displayed:

```
1 package com.example.code;
2
3 import java.util.ArrayList;
4
5 public class ExampleFunctions {
6
7     public int listSize() {
8         ArrayList<String> list = new ArrayList<>();
9         list.add("A");
10        list.add("B");
11        list.add("C");
12        return list.size();
13    }
14 }
15 }
```

**Example 9**

**Test Array Contains Value**

**Problem Statement:**

Check whether an array contains a specific value.

**Method Logic:**

Returns true if value exists.

**Test Focus:**

- Loop-based logic
- assertTrue

**Expected Outcome:**

Value exists in array.

The screenshot shows the Eclipse IDE interface. On the left, the 'Package Explorer' view displays a green bar indicating successful execution. In the center, the 'JUnit' view shows 'Runs: 1/1', 'Errors: 0', and 'Failures: 0'. On the right, the code editor displays the following Java code:

```
1 package com.example.code;
2
3 public class ExampleFunctions {
4
5     public boolean containsValue(int[] arr, int value) {
6         for (int i : arr) {
7             if (i == value) return true;
8         }
9     }
10    }
11
12 }
```

## Example 10

### Test Exception for Invalid Age

#### Problem Statement:

Throw exception if age is negative.

#### Method Logic:

Throws `IllegalArgumentException`.

#### Test Focus:

- Exception testing • `@Test(expected = IllegalArgumentException.class)`

#### Expected Outcome:

Exception should be thrown.

The screenshot shows the Eclipse IDE interface. On the left, the 'Package Explorer' view displays a green bar indicating successful execution. In the center, the 'JUnit' view shows 'Runs: 1/1', 'Errors: 0', and 'Failures: 0'. On the right, the code editor displays the following Java code:

```
1 package com.example.code;
2
3 public class ExampleFunctions {
4
5     public void validateAge(int age) {
6         if (age < 0)
7             throw new IllegalArgumentException("Invalid age");
8     }
9 }
10
```

## Example 11

### Test Greeting Message

#### Problem Statement:

Return greeting message based on username.

#### Method Logic:

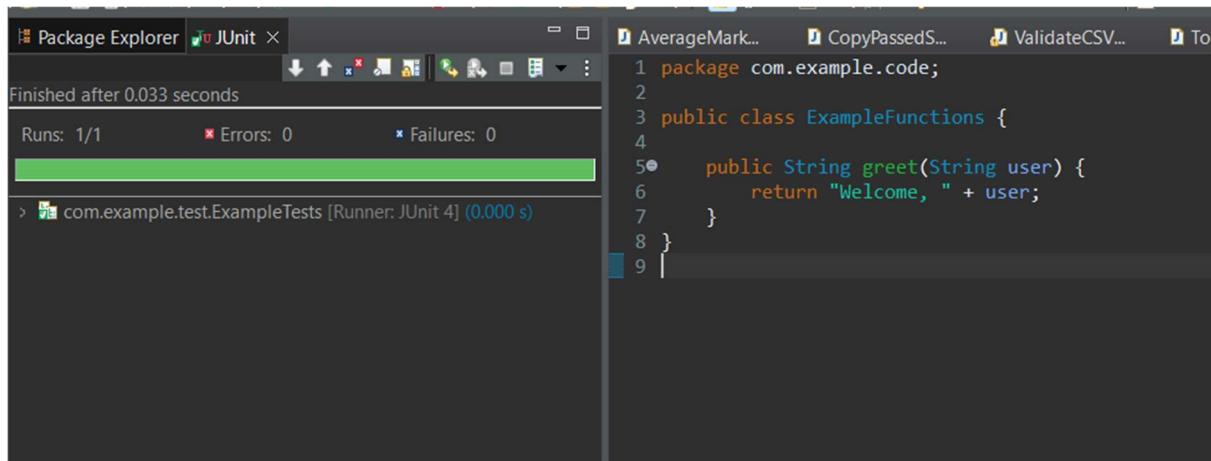
Returns "Welcome, User".

### **Test Focus:**

- String comparison
- assertEquals

### **Expected Outcome:**

Message should match exactly.



The screenshot shows a Java development environment with two panes. The left pane is the 'JUnit' view, which displays the results of a test run. It shows 'Runs: 1/1', 'Errors: 0', and 'Failures: 0', indicating a successful test. The right pane shows the source code for a class named 'ExampleFunctions'. The code contains a single method 'greet' that takes a string parameter 'user' and returns a welcome message.

```
1 package com.example.code;
2
3 public class ExampleFunctions {
4
5     public String greet(String user) {
6         return "Welcome, " + user;
7     }
8 }
9 }
```

## **Example 12**

### **Test Boolean Flag**

#### **Problem Statement:**

Return true if user is active.

#### **Method Logic:**

Returns boolean status.

### **Test Focus:**

- Boolean assertion
- assertTrue

### **Expected Outcome:**

Active user returns true.

The screenshot shows the Eclipse IDE interface. On the left, the 'Package Explorer' view shows a project structure. In the center, the 'JUnit' view displays the test results: 'Finished after 0.045 seconds', 'Runs: 1/1', 'Errors: 0', and 'Failures: 0'. Below this, it lists the test class 'com.example.test.ExampleTests [Runner: JUnit 4] (0.000 s)'. On the right, the code editor displays the source code for the 'ExampleFunctions' class:

```
1 package com.example.code;
2
3 public class ExampleFunctions {
4
5     public boolean isActive() {
6         return true;
7     }
8 }
9
```

## Example 13

### Test Character Count Problem Statement:

Count characters in a string.

### Method Logic:

Returns string length.

### Test Focus:

- Value comparison
- assertEquals

### Expected Outcome:

Correct character count returned.

The screenshot shows the Eclipse IDE interface. On the left, the 'Package Explorer' view shows a project structure. In the center, the 'JUnit' view displays the test results: 'Finished after 0.039 seconds', 'Runs: 1/1', 'Errors: 0', and 'Failures: 0'. Below this, it lists the test class 'com.example.test.ExampleTests [Runner: JUnit 4] (0.001 s)'. On the right, the code editor displays the source code for the 'ExampleTests' class:

```
1 package com.example.test;
2
3 import static org.junit.Assert.*;
4 import org.junit.Test;
5 import com.example.code.ExampleFunctions;
6
7 public class ExampleTests {
8
9     @Test
10    public void testCharCount() {
11        ExampleFunctions f = new ExampleFunctions();
12        assertEquals(4, f.countChars("Java"));
13    }
14 }
15
```

## Example 14

### Test Default Value

### Problem Statement:

Return default country code.

### **Method Logic:**

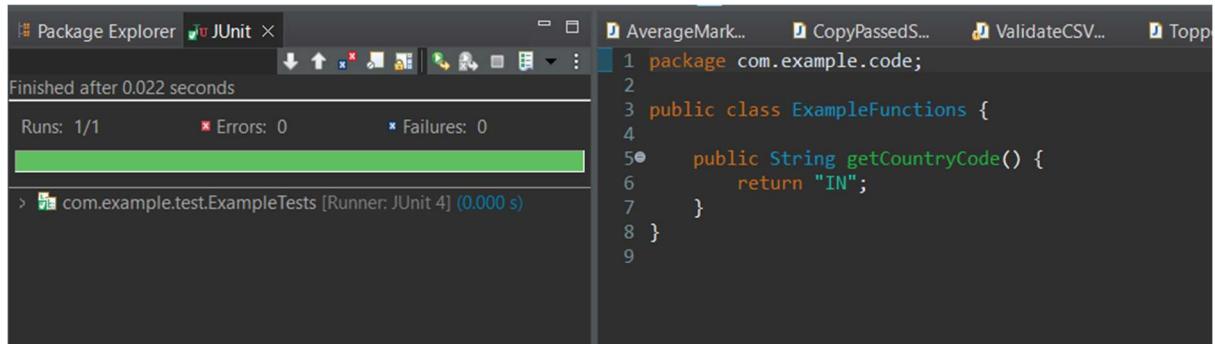
Returns "IN".

### **Test Focus:**

- Default value testing
- assertEquals

### **Expected Outcome:**

Country code matches.



The screenshot shows the Eclipse IDE interface. On the left, the Package Explorer view is open, showing a JUnit test named 'ExampleTests' has run successfully with 1/1 run, 0 errors, and 0 failures. On the right, the code editor displays the source code for a class named 'ExampleFunctions'. The code contains a single method 'getCountryCode' which returns the string 'IN'.

```
1 package com.example.code;
2
3 public class ExampleFunctions {
4
5     public String getCountryCode() {
6         return "IN";
7     }
8 }
```

## **Example 15**

### **Test Data Present Flag**

#### **Problem Statement:**

Check whether data exists.

### **Method Logic:**

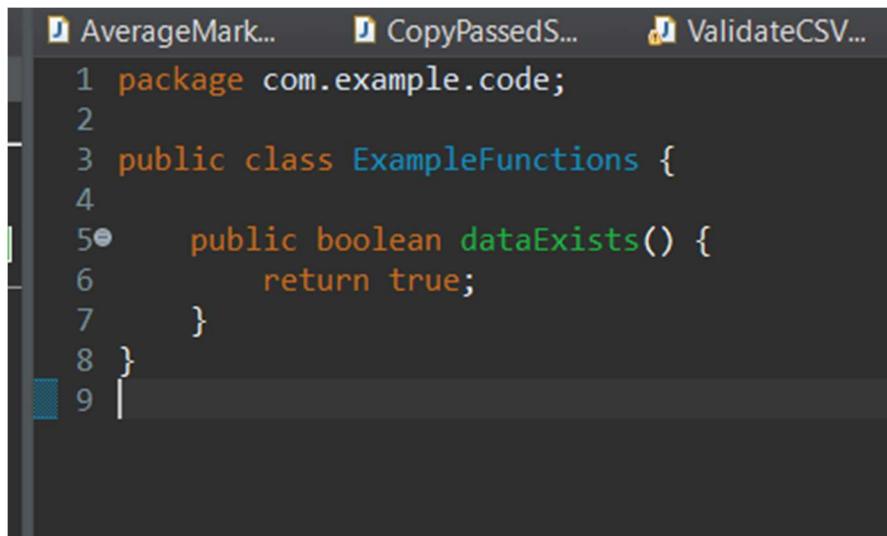
Returns true if data is present.

### **Test Focus:**

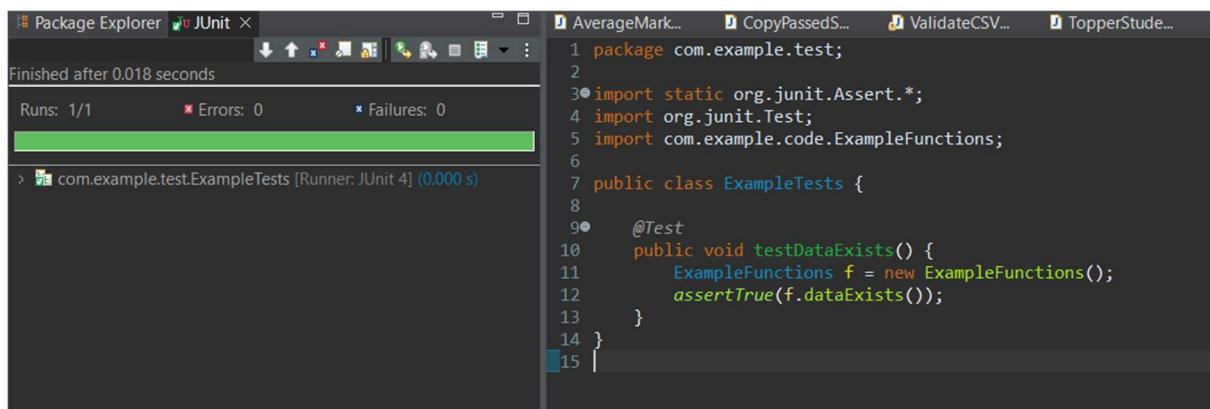
- Simple flag testing • assertTrue

### **Expected Outcome:**

Data presence validated.



```
1 package com.example.code;
2
3 public class ExampleFunctions {
4
5     public boolean dataExists() {
6         return true;
7     }
8 }
9
```



Package Explorer JUnit X |

Finished after 0.018 seconds

Runs: 1/1 Errors: 0 Failures: 0

> com.example.test.ExampleTests [Runner: JUnit 4] (0.000 s)

```
1 package com.example.test;
2
3 import static org.junit.Assert.*;
4 import org.junit.Test;
5 import com.example.code.ExampleFunctions;
6
7 public class ExampleTests {
8
9     @Test
10    public void testDataExists() {
11        ExampleFunctions f = new ExampleFunctions();
12        assertTrue(f.dataExists());
13    }
14 }
15
```