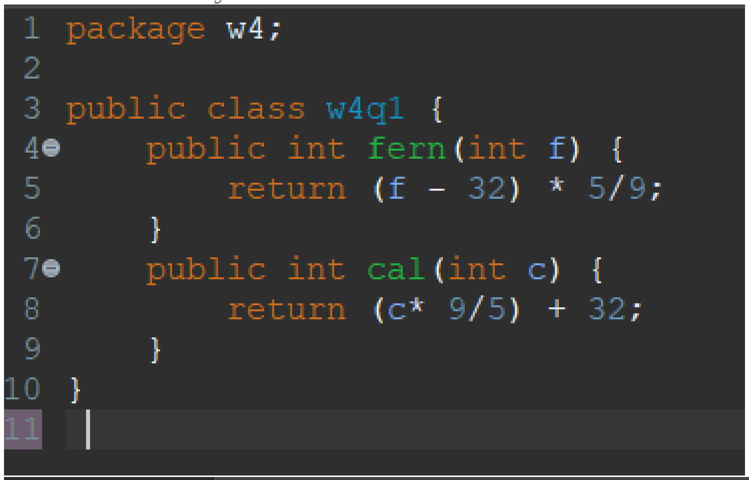
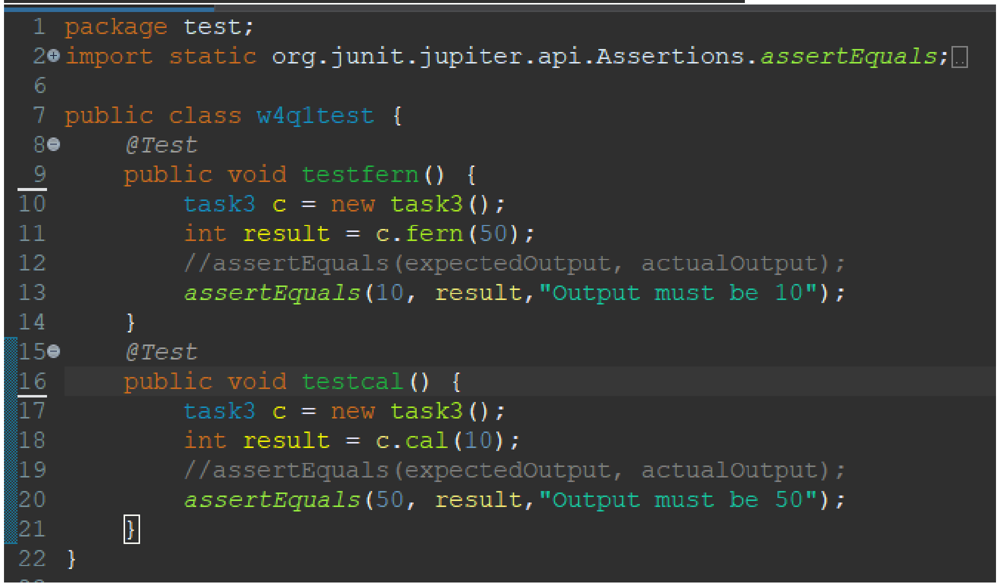
Chapter 4: Software Testing and Test-driven development (TDD)

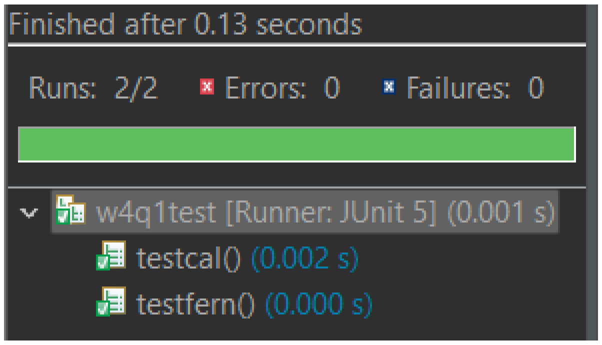
1. Write tests for converting temperatures from Celsius to Fahrenheit and vice versa.
   Hint: Use assertEquals(expected, actual) to compare the expected result with the actual result returned by the method.



```java
package w4;

public class w4q1 {
    public int fern(int f) {
        return (f - 32) * 5/9;
    }
    public int cal(int c) {
        return (c* 9/5) + 32;
    }
}

```

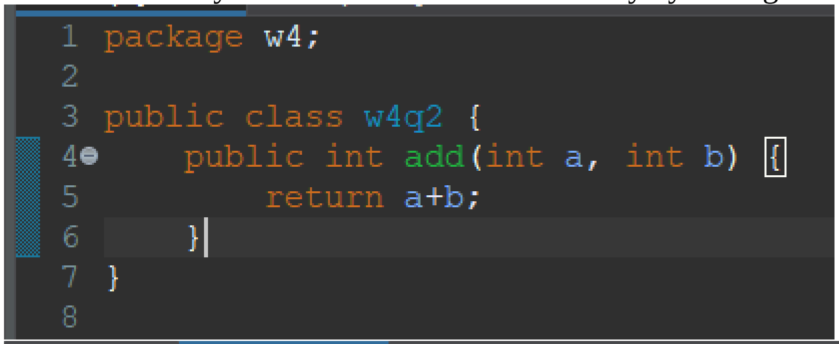


```java
package test;
import static org.junit.jupiter.api.Assertions.assertEquals;

public class w4q1test {
    @Test
    public void testfern() {
        task3 c = new task3();
        int result = c.fern(50);
        //assertEquals(expectedOutput, actualOutput);
        assertEquals(10, result,"Output must be 10");
    }
    @Test
    public void testcal() {
        task3 c = new task3();
        int result = c.cal(10);
        //assertEquals(expectedOutput, actualOutput);
        assertEquals(50, result,"Output must be 50");
    }
}
```

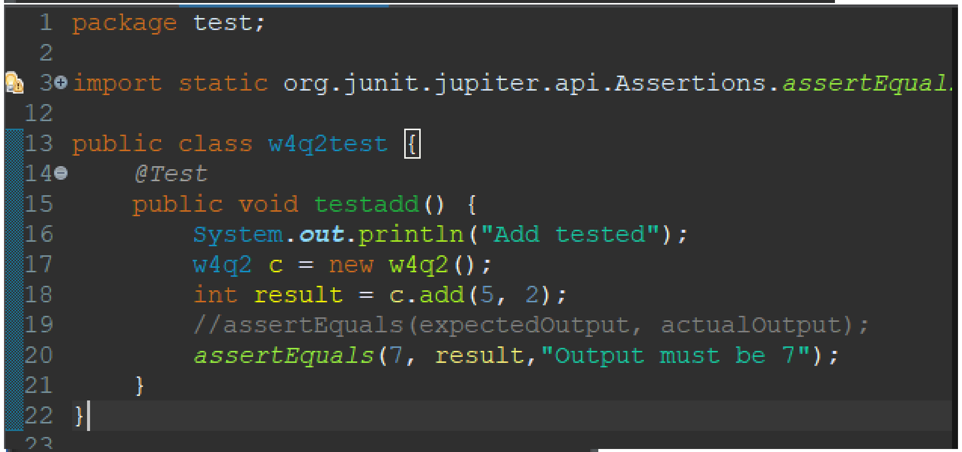


```
Finished after 0.13 seconds

Runs: 2/2    Errors: 0    Failures: 0

w4q1test [Runner: JUnit 5] (0.001 s)
    testcal() (0.002 s)
    testfern() (0.000 s)
```

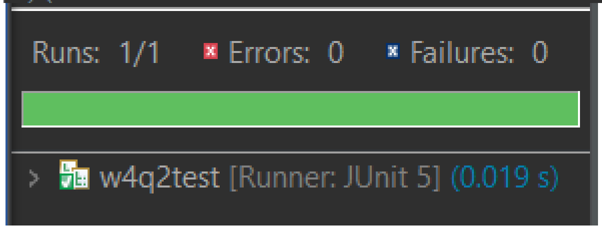
2. Write a simple method in a **Calculator** class that adds two integers. Then, create a JUnit test case to verify that the method works correctly by adding two numbers together.



```
package w4;

public class w4q2 {
    public int add(int a, int b) {
        return a+b;
    }
}

```

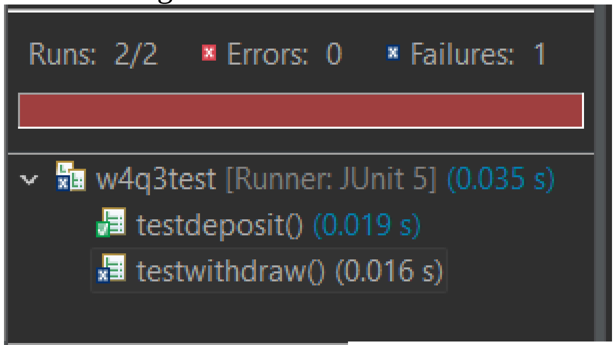


```
package test;

import static org.junit.jupiter.api.Assertions.assertEqual

public class w4q2test {
    @Test
    public void testadd() {
        System.out.println("Add tested");
        w4q2 c = new w4q2();
        int result = c.add(5, 2);
        //assertEquals(expectedOutput, actualOutput);
        assertEquals(7, result,"Output must be 7");
    }
}
```

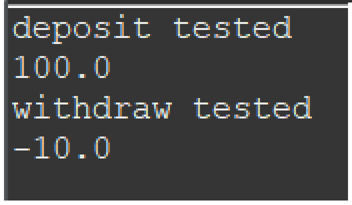


```
Runs: 1/1    Errors: 0    Failures: 0

> w4q2test [Runner: JUnit 5] (0.019 s)
```

Withdrawing 60:



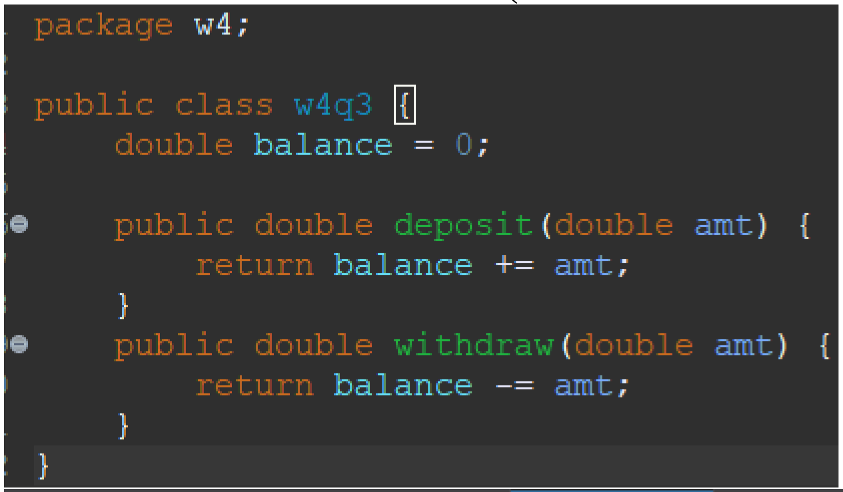
```
Runs: 2/2    Errors: 0    Failures: 1

v w4q3test [Runner: JUnit 5] (0.035 s)
    testdeposit() (0.019 s)
    testwithdraw() (0.016 s)
```



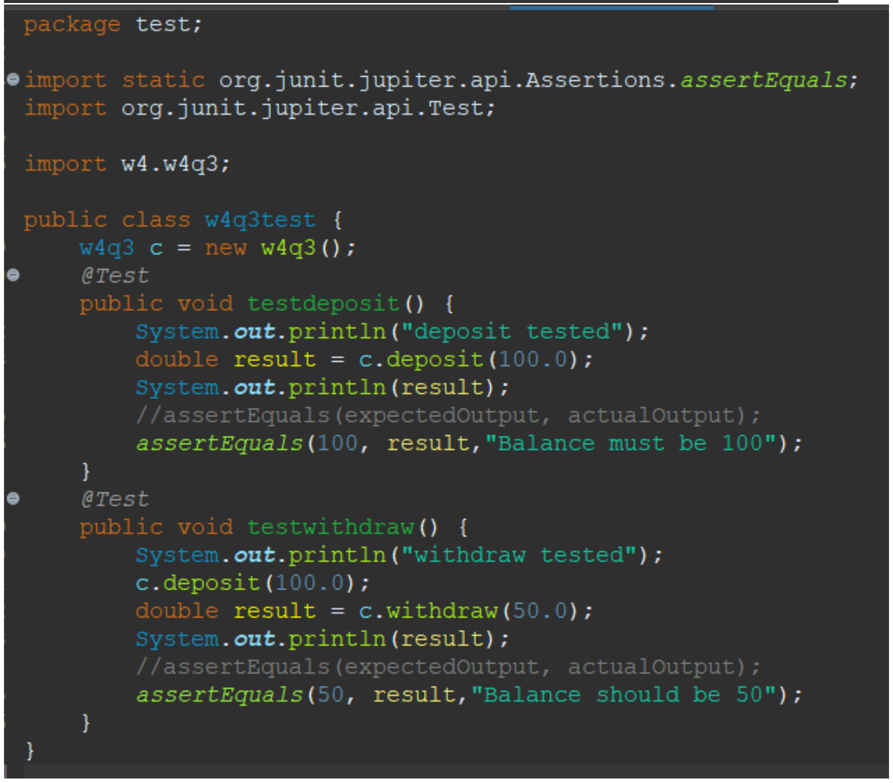
```
deposit tested
100.0
withdraw tested
-10.0
```

3. Write a class **BankAccount** with methods **deposit(double amount)** and withdraw(double amount). The account balance should start at 0.0, and the methods should update the balance accordingly.
   Write a JUnit test that:

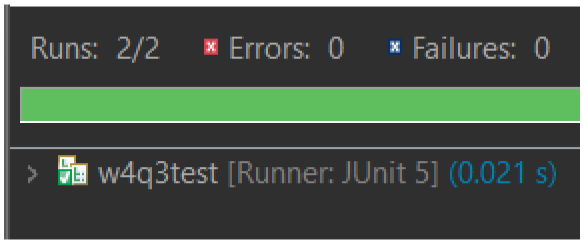
   - Ensures a deposit of 100.0 increases the balance to 100.0.
   - Ensures a withdrawal of 50.0 decreases the balance to 50.0.
   - Verifies that a withdrawal of 60.0 fails (balance should remain 50.0)



```java
package w4;

public class w4q3 {
    double balance = 0;

    public double deposit(double amt) {
        return balance += amt;
    }
    public double withdraw(double amt) {
        return balance -= amt;
    }
}
```



```java
package test;

import static org.junit.jupiter.api.Assertions.assertEquals;
import org.junit.jupiter.api.Test;

import w4.w4q3;

public class w4q3test {
    w4q3 c = new w4q3();
    @Test
    public void testdeposit() {
        System.out.println("deposit tested");
        double result = c.deposit(100.0);
        System.out.println(result);
        //assertEquals(expectedOutput, actualOutput);
        assertEquals(100, result,"Balance must be 100");
    }
    @Test
    public void testwithdraw() {
        System.out.println("withdraw tested");
        c.deposit(100.0);
        double result = c.withdraw(50.0);
        System.out.println(result);
        //assertEquals(expectedOutput, actualOutput);
        assertEquals(50, result,"Balance should be 50");
    }
}
```



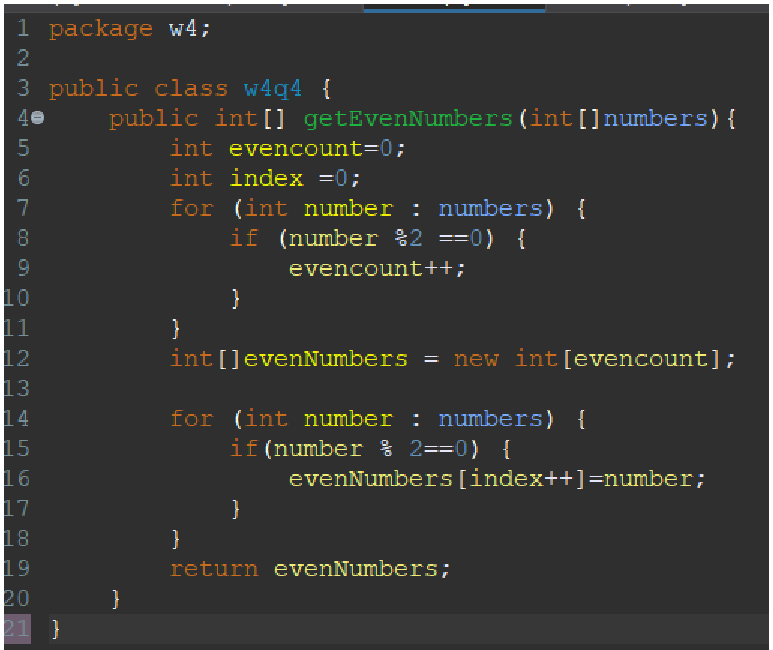
```
Runs: 2/2    Errors: 0    Failures: 0

> w4q3test [Runner: JUnit 5] (0.021 s)
```

4. Create a method **getEvenNumbers(int[] numbers)** in a **NumberUtils** class that filters out and returns only the even numbers from a given array of integers. Write a JUnit test case to verify that the method correctly returns a list of even numbers.

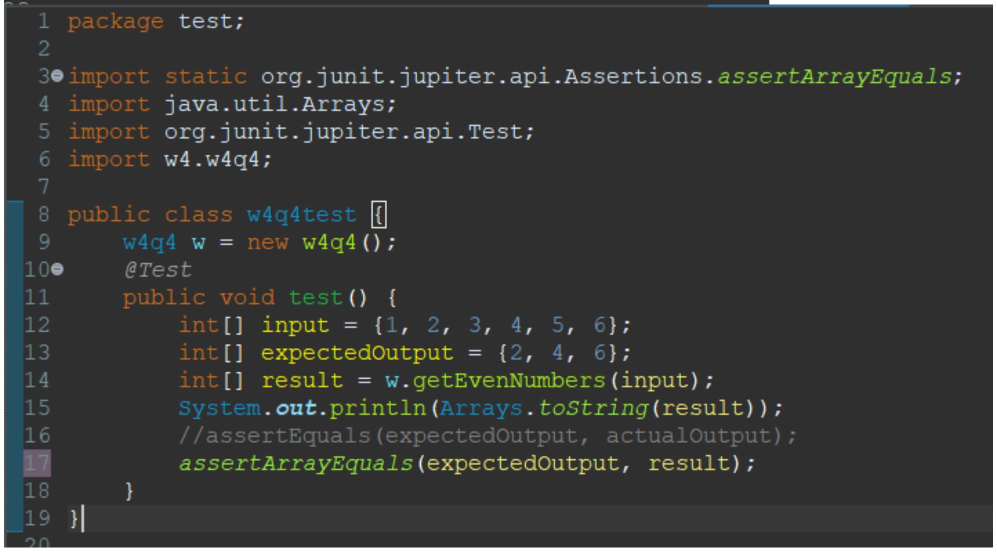
   For example:

   Input: [1, 2, 3, 4, 5, 6]

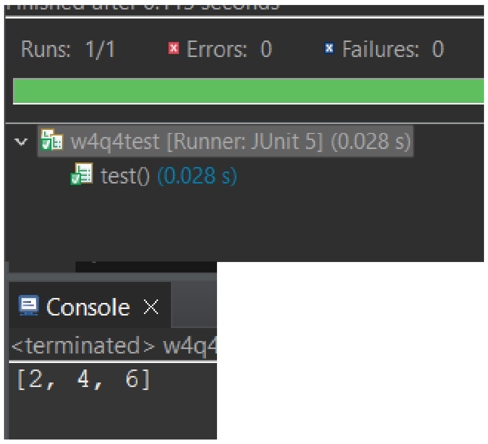
   Expected Output: [2, 4, 6]



```java
package w4;

public class w4q4 {
    public int[] getEvenNumbers(int[]numbers){
        int evencount=0;
        int index =0;
        for (int number : numbers) {
            if (number %2 ==0) {
                evencount++;
            }
        }
        int[]evenNumbers = new int[evencount];

        for (int number : numbers) {
            if(number % 2==0) {
                evenNumbers[index++]=number;
            }
        }
        return evenNumbers;
    }
}
```

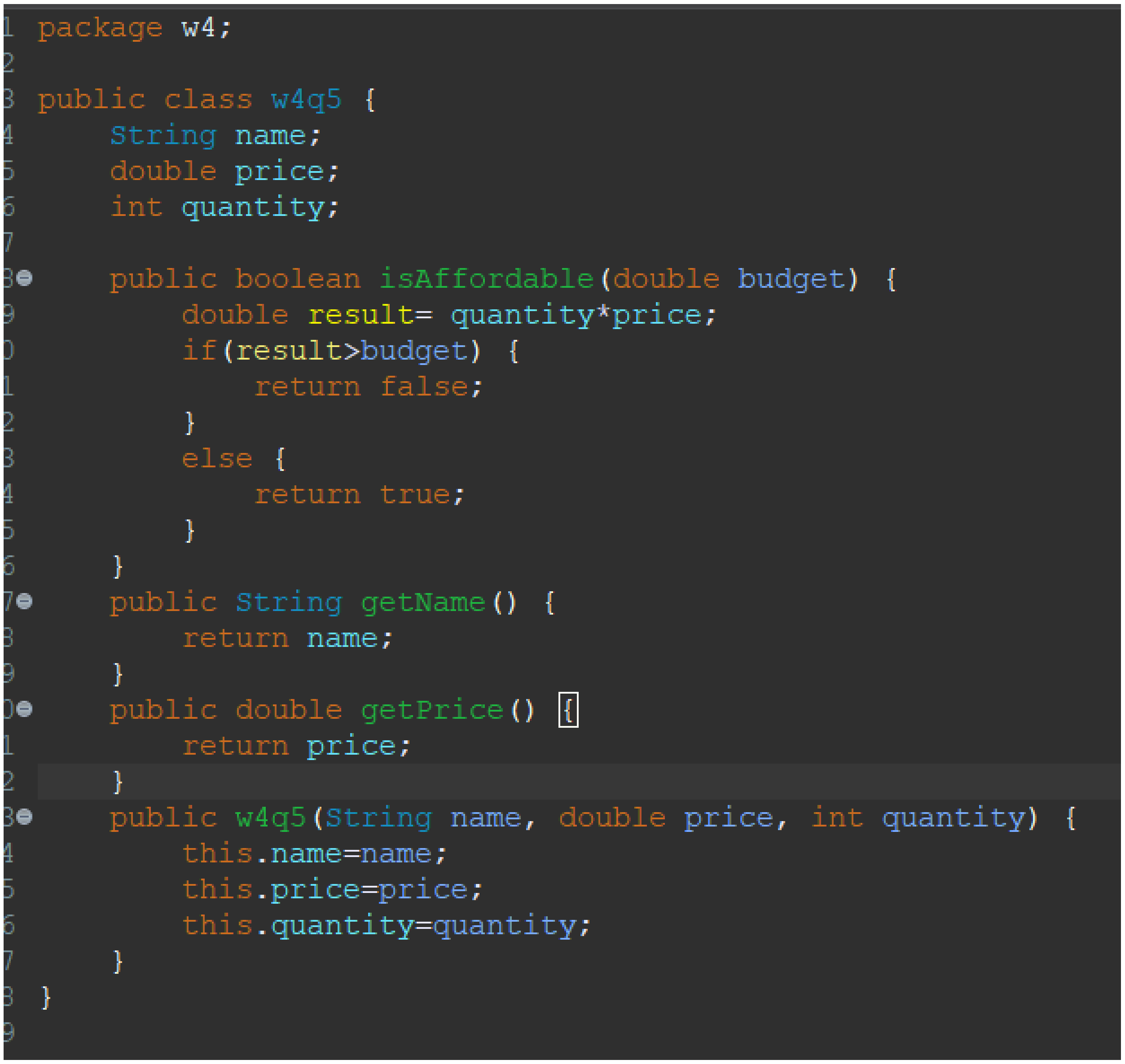


```java
package test;

import static org.junit.jupiter.api.Assertions.assertArrayEquals;
import java.util.Arrays;
import org.junit.jupiter.api.Test;
import w4.w4q4;

public class w4q4test {
    w4q4 w = new w4q4();
    @Test
    public void test() {
        int[] input = {1, 2, 3, 4, 5, 6};
        int[] expectedOutput = {2, 4, 6};
        int[] result = w.getEvenNumbers(input);
        System.out.println(Arrays.toString(result));
        //assertEquals(expectedOutput, actualOutput);
        assertArrayEquals(expectedOutput, result);
    }
}
```

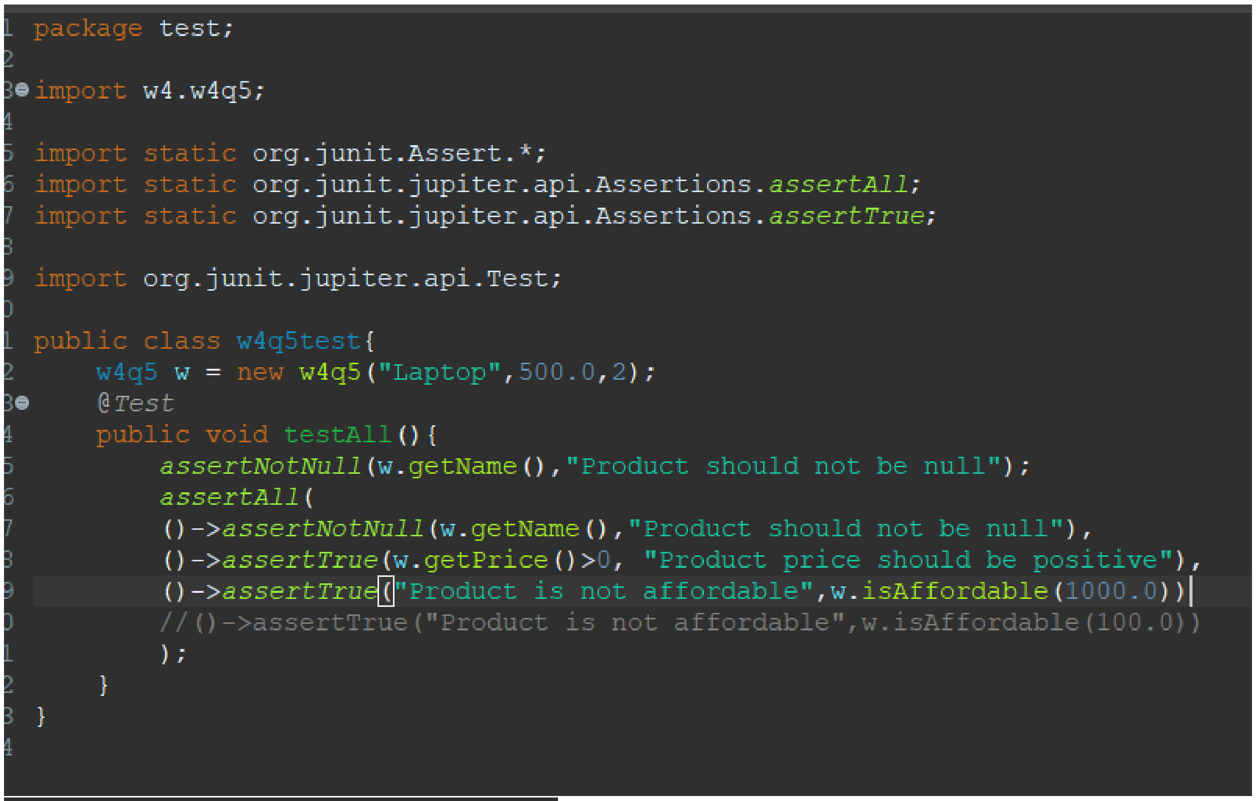


```
<terminated> w4q4
[2, 4, 6]
```

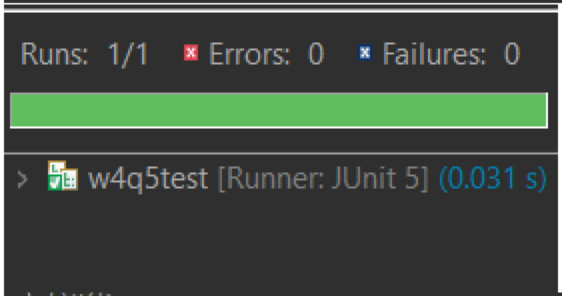
5. **Complex Assertion with assertAll**
   Write a class **Product** with fields **name** (String), **price** (double), and **quantity** (int). Write a method **isAffordable**(double budget) that returns true if the total price (price * quantity) is less than or equal to the given budget. Write a JUnit test that:
   - Verifies that the name is not null.
   - Verifies that the price is a positive value.
   - Verifies that the isAffordable() method works correctly with different budgets using assertAll.



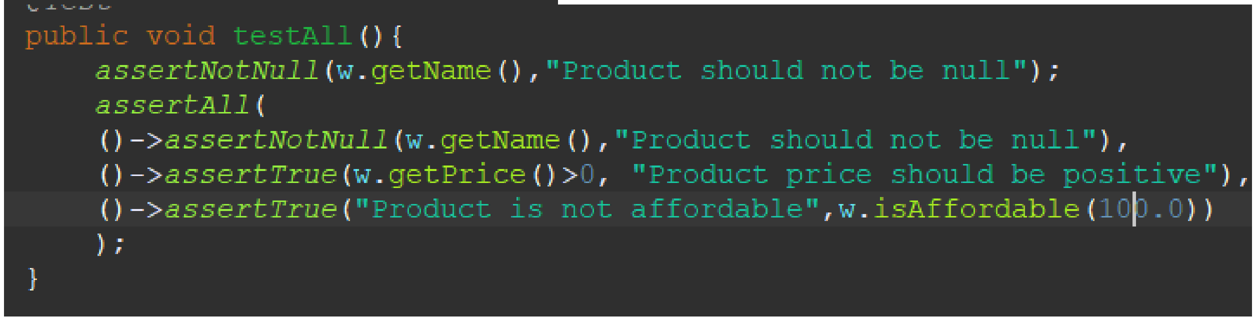
```
package w4;

public class w4q5 {
    String name;
    double price;
    int quantity;

    public boolean isAffordable(double budget) {
        double result= quantity*price;
        if(result>budget) {
            return false;
        }
        else {
            return true;
        }
    }
    public String getName() {
        return name;
    }
    public double getPrice() {
        return price;
    }
    public w4q5(String name, double price, int quantity) {
        this.name=name;
        this.price=price;
        this.quantity=quantity;
    }
}
```



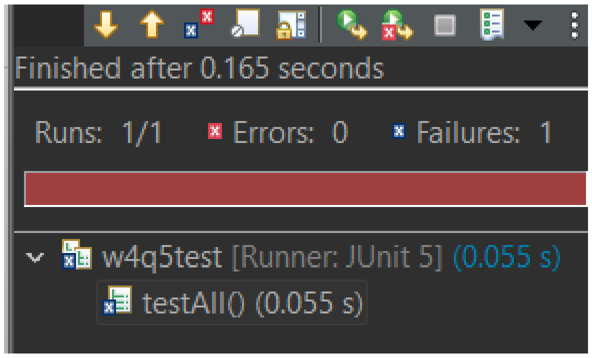
```
package test;

import w4.w4q5;

import static org.junit.Assert.*;
import static org.junit.jupiter.api.Assertions.assertAll;
import static org.junit.jupiter.api.Assertions.assertTrue;

import org.junit.jupiter.api.Test;

public class w4q5test{
    w4q5 w = new w4q5("Laptop",500.0,2);
    @Test
    public void testAll(){
        assertNotNull(w.getName(),"Product should not be null");
        assertAll(
        ()->assertNotNull(w.getName(),"Product should not be null"),
        ()->assertTrue(w.getPrice()>0, "Product price should be positive"),
        ()->assertTrue("Product is not affordable",w.isAffordable(1000.0))
        //()->assertTrue("Product is not affordable",w.isAffordable(100.0))
        );
    }
}
```



```
Runs: 1/1    Errors: 0    Failures: 0

> w4q5test [Runner: JUnit 5] (0.031 s)
```

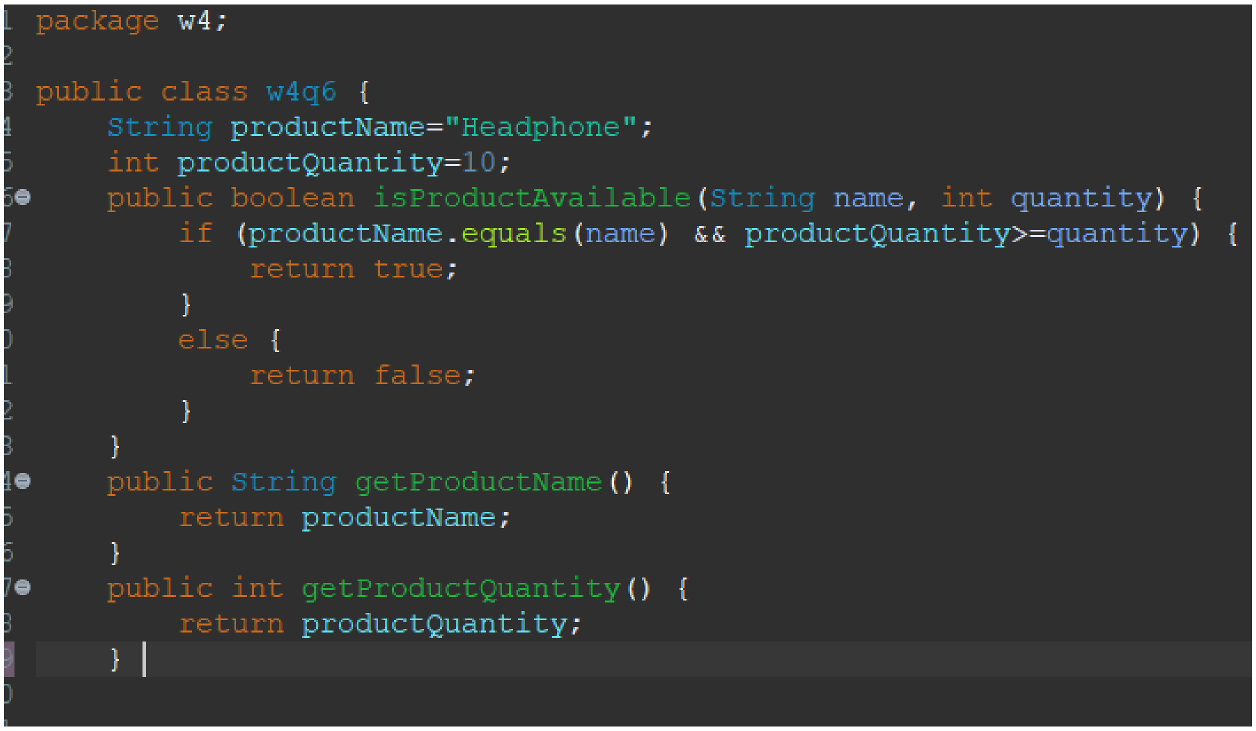


```
public void testAll(){
    assertNotNull(w.getName(),"Product should not be null");
    assertAll(
    ()->assertNotNull(w.getName(),"Product should not be null"),
    ()->assertTrue(w.getPrice()>0, "Product price should be positive"),
    ()->assertTrue("Product is not affordable",w.isAffordable(100.0))
    );
}
```

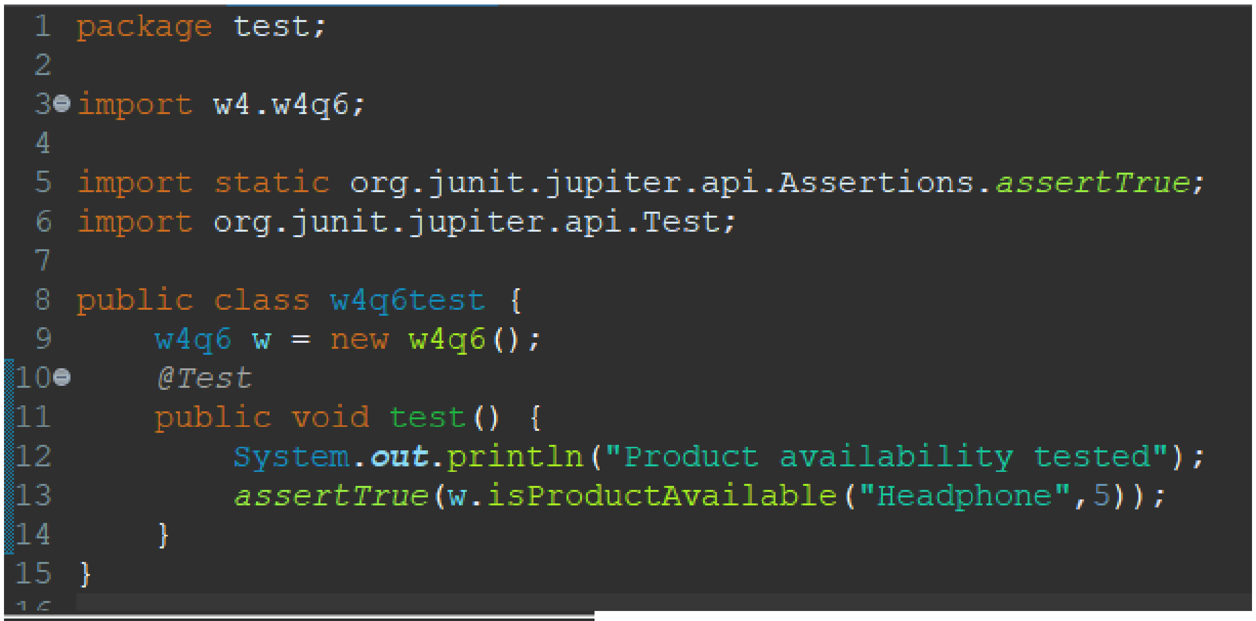


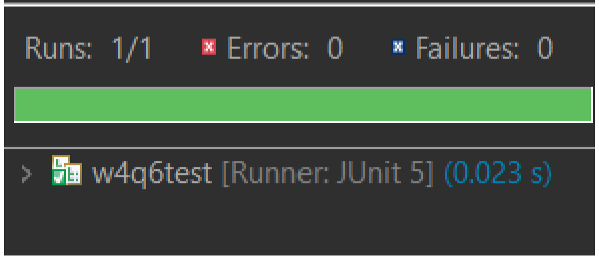
```
Finished after 0.165 seconds

Runs:  1/1     Errors:  0     Failures:  1

w4q5test [Runner: JUnit 5] (0.055 s)
    testAll() (0.055 s)
```

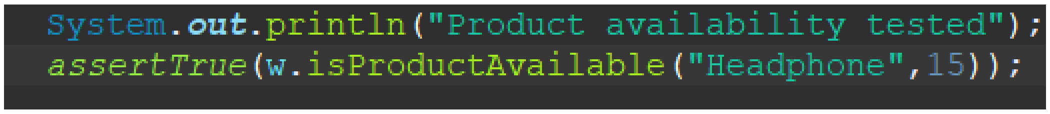
6. In an inventory management system, you need a method **isProductAvailable(String productName, int quantity)** to check if the given product is in stock. The method should return true if the requested quantity is available in stock and false if the requested quantity exceeds the available stock.

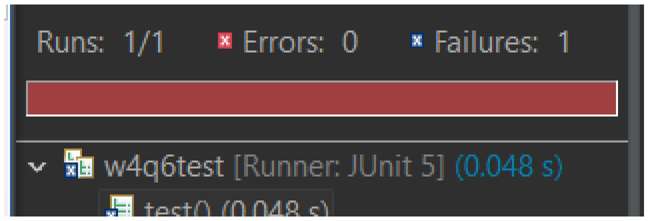


```java
package w4;

public class w4q6 {
	String productName="Headphone";
	int productQuantity=10;
	public boolean isProductAvailable(String name, int quantity) {
		if (productName.equals(name) && productQuantity>=quantity) {
			return true;
		}
		else {
			return false;
		}
	}
	public String getProductName() {
		return productName;
	}
	public int getProductQuantity() {
		return productQuantity;
	}
```

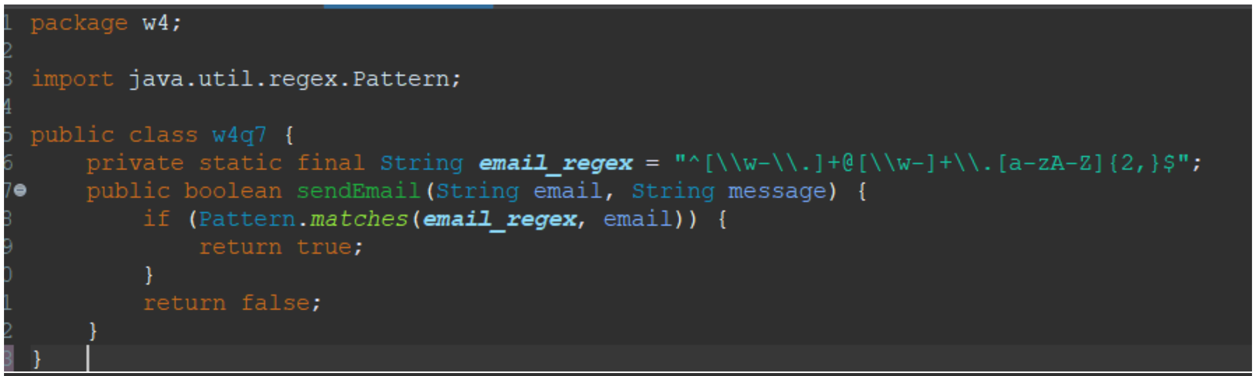

```java
package test;

import w4.w4q6;

import static org.junit.jupiter.api.Assertions.assertTrue;
import org.junit.jupiter.api.Test;

public class w4q6test {
    w4q6 w = new w4q6();
    @Test
    public void test() {
        System.out.println("Product availability tested");
        assertTrue(w.isProductAvailable("Headphone",5));
    }
}
```


```
Runs: 1/1    Errors: 0    Failures: 0

w4q6test [Runner: JUnit 5] (0.023 s)
```

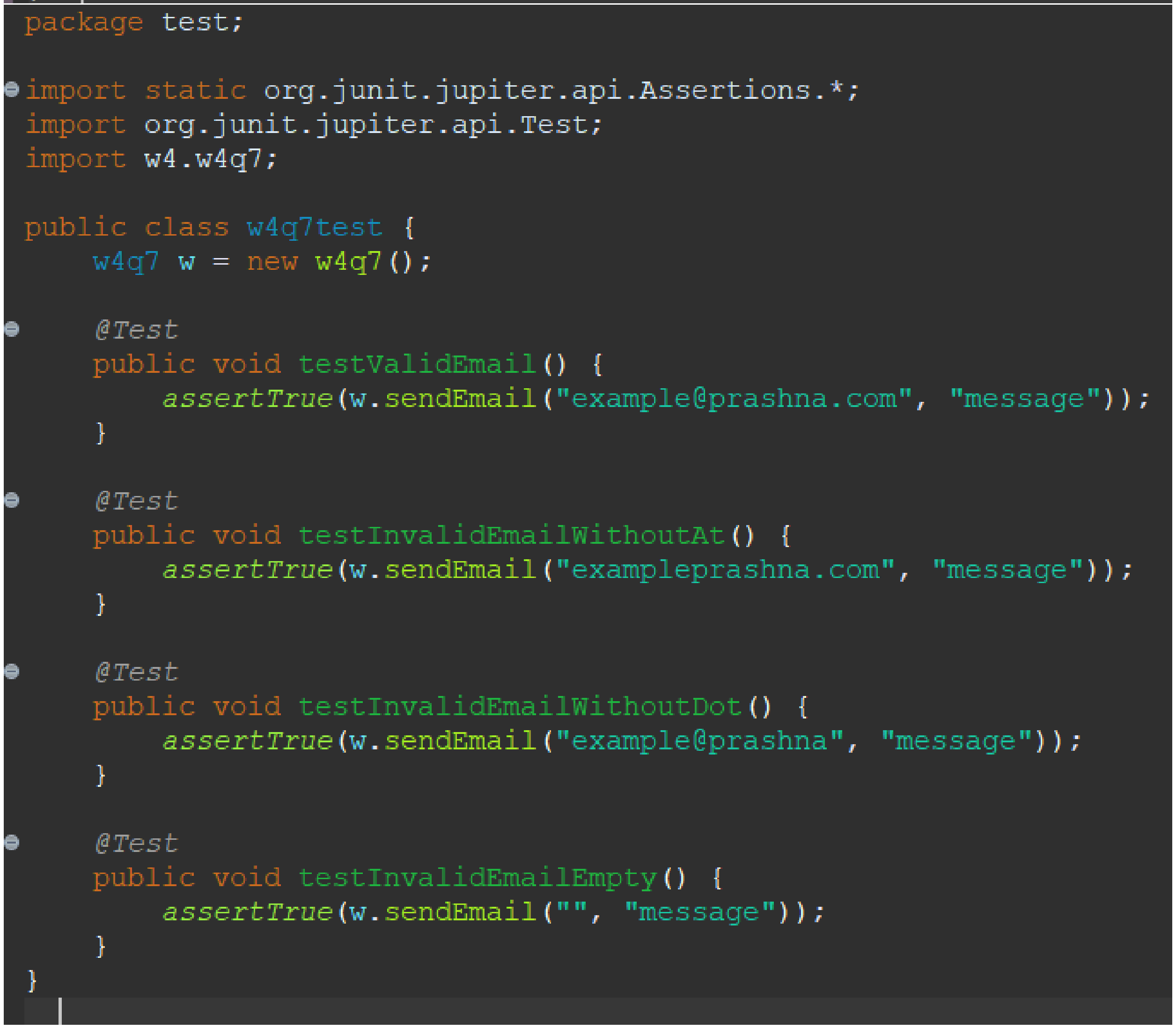

```java
System.out.println("Product availability tested");
assertTrue(w.isProductAvailable("Headphone",15));
```


```
Runs: 1/1    Errors: 0    Failures: 1

w4q6test [Runner: JUnit 5] (0.048 s)
    test() (0.048 s)
```

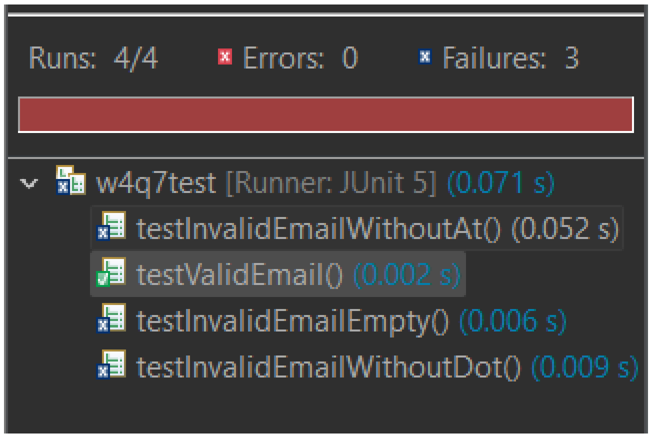
7. In a notification service, you need to implement a **sendEmail(String email, String message)** method to send an email. The method should return true if the email is sent successfully and false if the email address is invalid.



```java
package w4;

import java.util.regex.Pattern;

public class w4q7 {
    private static final String email_regex = "^[\\w-\\.]+@[\\w-]+\\.[a-zA-Z]{2,}$";
    public boolean sendEmail(String email, String message) {
        if (Pattern.matches(email_regex, email)) {
            return true;
        }
        return false;
    }
}
```

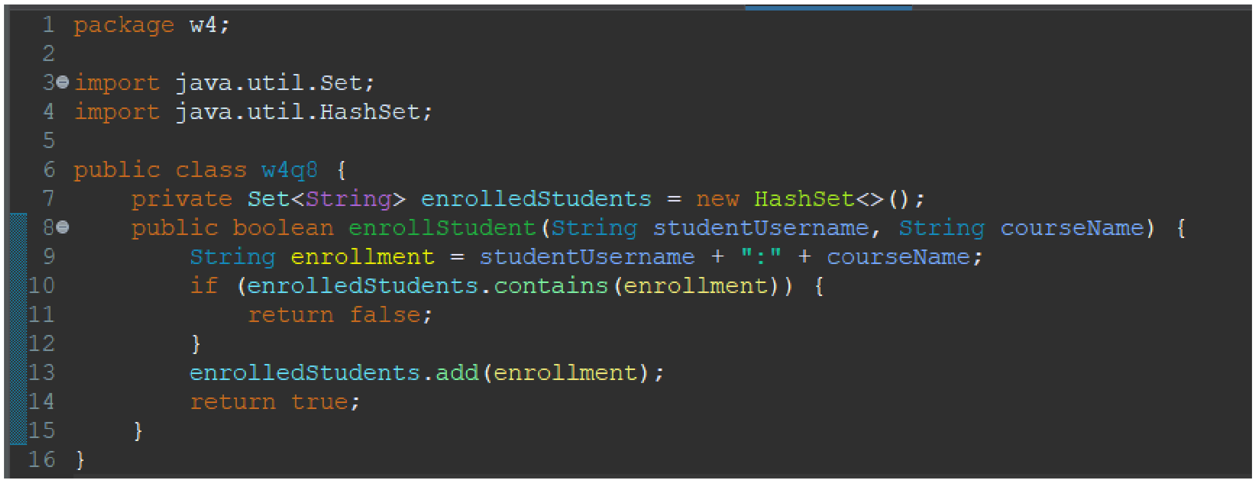


```java
package test;

import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;
import w4.w4q7;

public class w4q7test {
    w4q7 w = new w4q7();

    @Test
    public void testValidEmail() {
        assertTrue(w.sendEmail("example@prashna.com", "message"));
    }

    @Test
    public void testInvalidEmailWithoutAt() {
        assertTrue(w.sendEmail("exampleprashna.com", "message"));
    }

    @Test
    public void testInvalidEmailWithoutDot() {
        assertTrue(w.sendEmail("example@prashna", "message"));
    }

    @Test
    public void testInvalidEmailEmpty() {
        assertTrue(w.sendEmail("", "message"));
    }
}
```

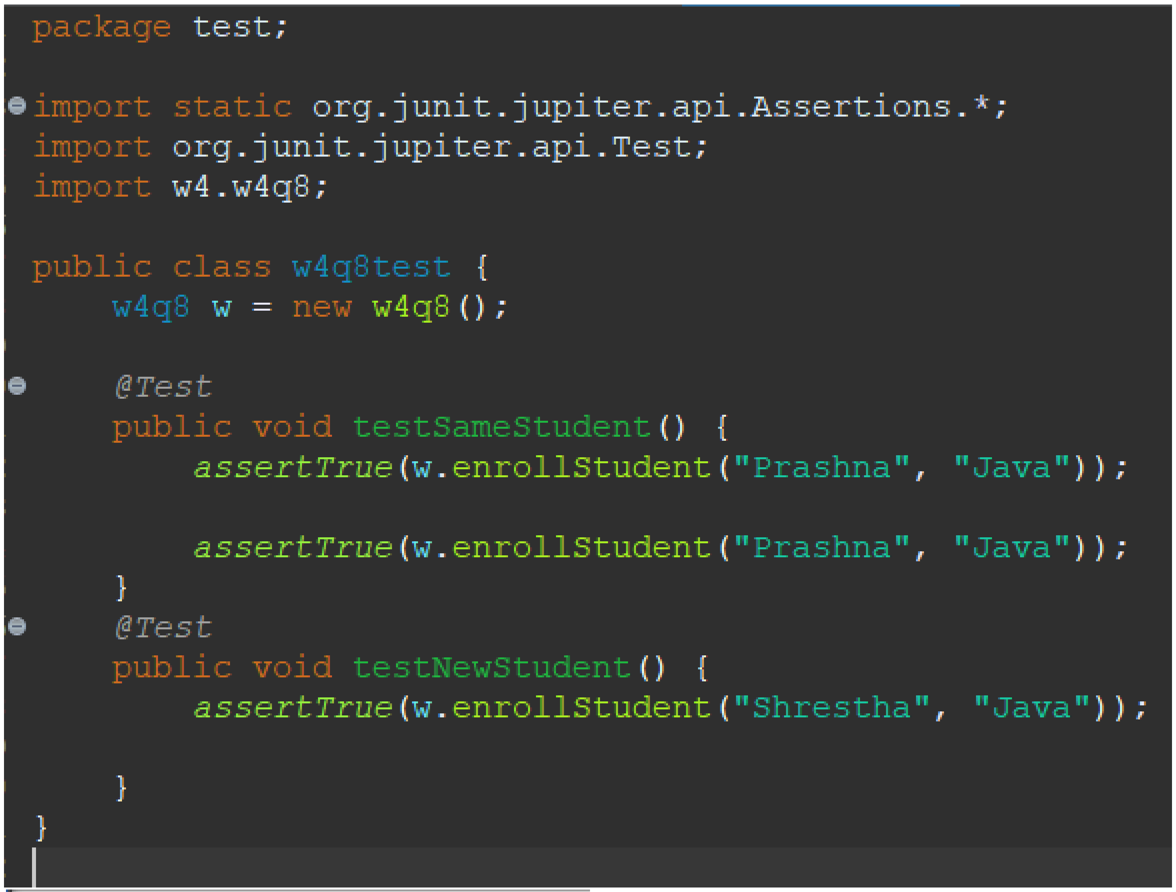


```
Runs: 4/4    Errors: 0    Failures: 3

w4q7test [Runner: JUnit 5] (0.071 s)
    testInvalidEmailWithoutAt() (0.052 s)
    testValidEmail() (0.002 s)
    testInvalidEmailEmpty() (0.006 s)
    testInvalidEmailWithoutDot() (0.009 s)
```

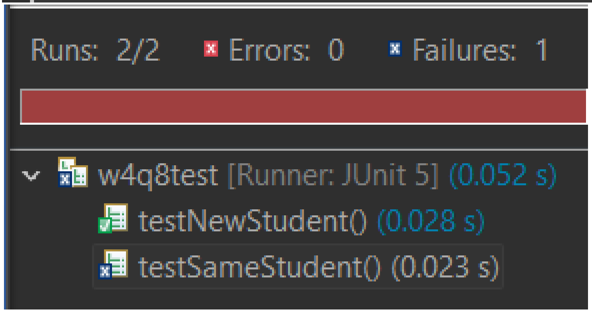
8. In an Learning management system, students can enroll in courses. The **EnrollmentService** class needs a method **enrollStudent(String studentUsername, String courseName)** to allow students to enroll in courses. The method should return true if the student is successfully enrolled, and false if the student is already enrolled in the course.



```java
package w4;

import java.util.Set;
import java.util.HashSet;

public class w4q8 {
    private Set<String> enrolledStudents = new HashSet<>();
    public boolean enrollStudent(String studentUsername, String courseName) {
        String enrollment = studentUsername + ":" + courseName;
        if (enrolledStudents.contains(enrollment)) {
            return false;
        }
        enrolledStudents.add(enrollment);
        return true;
    }
}
```



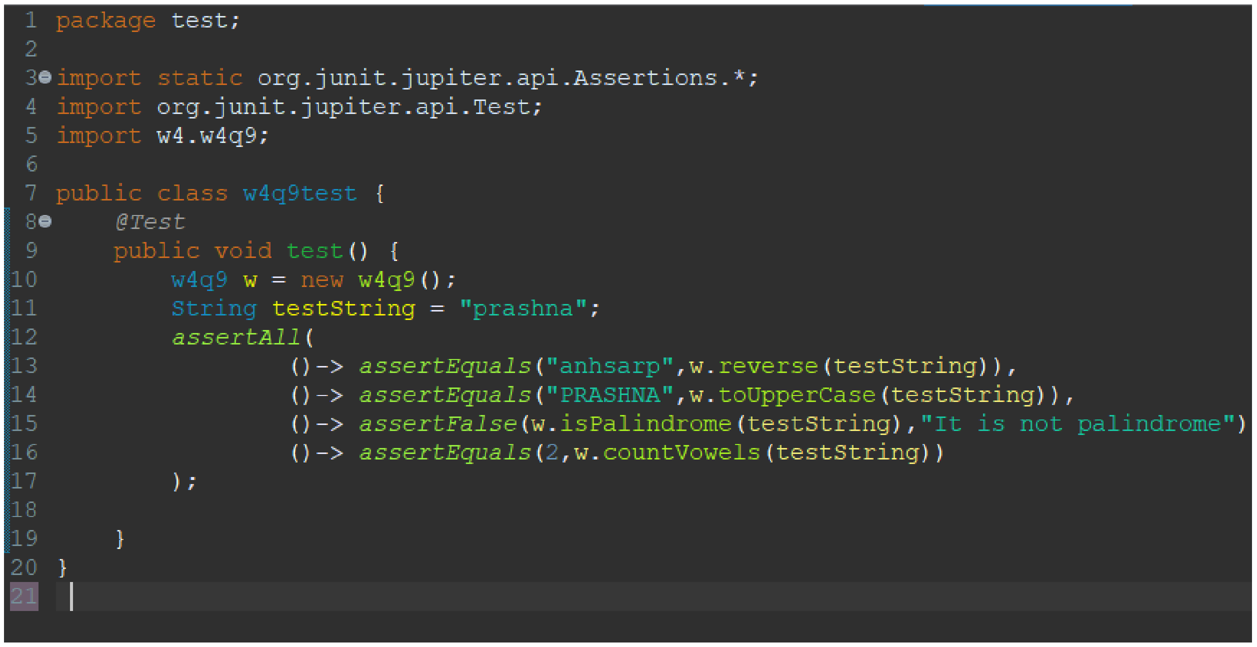
```
package test;

import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;
import w4.w4q8;

public class w4q8test {
    w4q8 w = new w4q8();

    @Test
    public void testSameStudent() {
        assertTrue(w.enrollStudent("Prashna", "Java"));

        assertTrue(w.enrollStudent("Prashna", "Java"));
    }
    @Test
    public void testNewStudent() {
        assertTrue(w.enrollStudent("Shrestha", "Java"));

    }
}
```



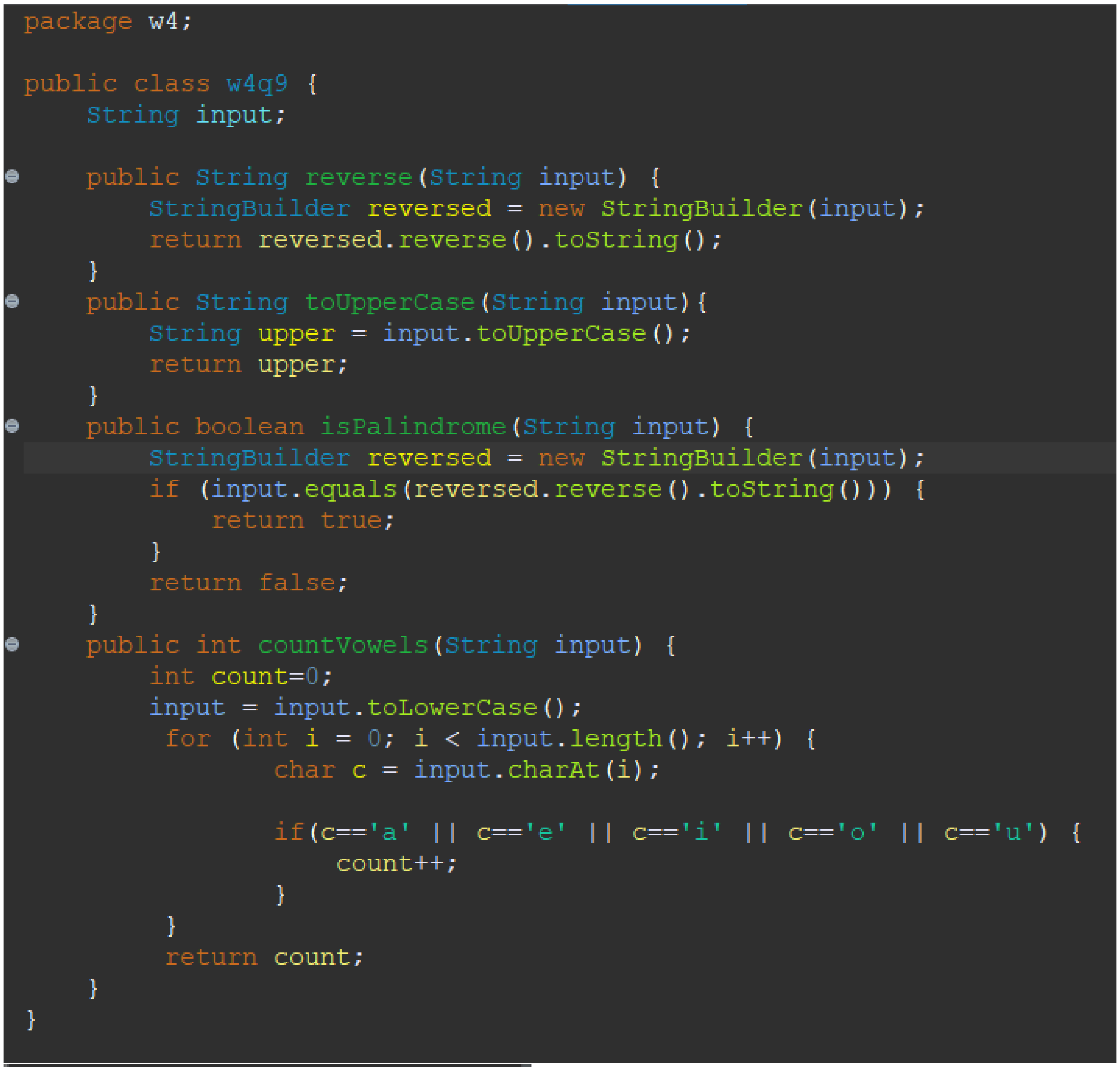
```
Runs: 2/2    Errors: 0    Failures: 1

w4q8test [Runner: JUnit 5] (0.052 s)
    testNewStudent() (0.028 s)
    testSameStudent() (0.023 s)
```

9. Create a class StringManipulator with the following methods:
    a. reverse(String input):
    This method should take a string and return the reversed version of the string.
    b. toUpperCase(String input):
    This method should convert all characters of the given string to uppercase.
    c. isPalindrome(String input):
    This method should return true if the input string is a palindrome (i.e., it reads the same forwards and backwards), and false otherwise.
    d. countVowels(String input):
    This method should count and return the number of vowels (a, e, i, o, u) in the input string.

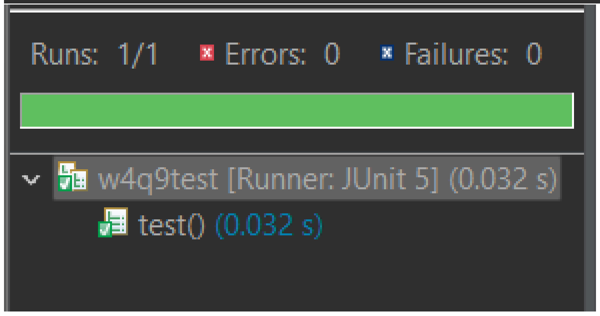
Write a single JUnit test case using **assertAll** to verify all the methods of the **StringManipulator** class.



```java
package test;

import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;
import w4.w4q9;

public class w4q9test {
    @Test
    public void test() {
        w4q9 w = new w4q9();
        String testString = "prashna";
        assertAll(
                ()-> assertEquals("anhsarp",w.reverse(testString)),
                ()-> assertEquals("PRASHNA",w.toUpperCase(testString)),
                ()-> assertFalse(w.isPalindrome(testString),"It is not palindrome")
                ()-> assertEquals(2,w.countVowels(testString))
        );

    }
}

```



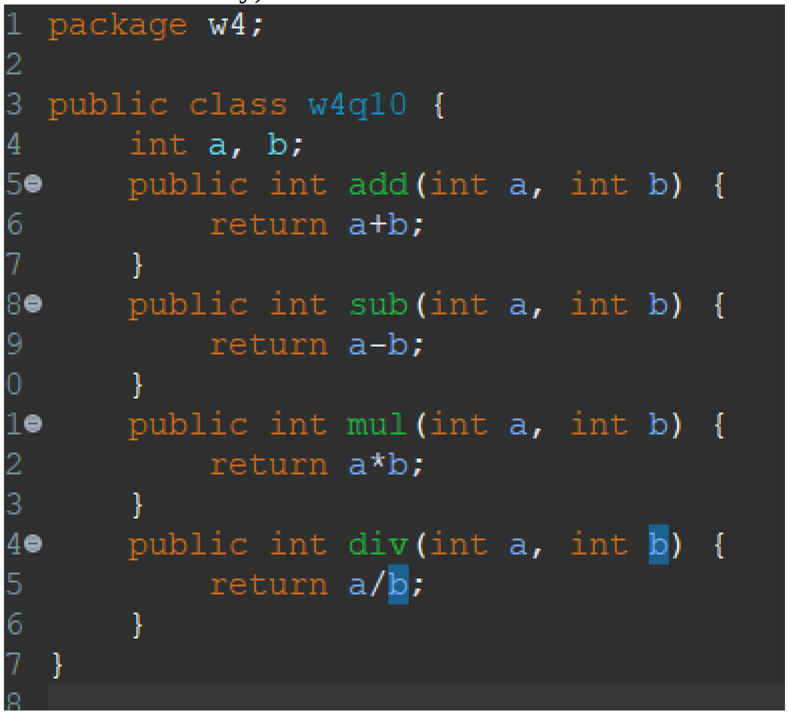
```java
package w4;

public class w4q9 {
    String input;

    public String reverse(String input) {
        StringBuilder reversed = new StringBuilder(input);
        return reversed.reverse().toString();
    }
    public String toUpperCase(String input){
        String upper = input.toUpperCase();
        return upper;
    }
    public boolean isPalindrome(String input) {
        StringBuilder reversed = new StringBuilder(input);
        if (input.equals(reversed.reverse().toString())) {
            return true;
        }
        return false;
    }
    public int countVowels(String input) {
        int count=0;
        input = input.toLowerCase();
         for (int i = 0; i < input.length(); i++) {
                char c = input.charAt(i);

                if(c=='a' || c=='e' || c=='i' || c=='o' || c=='u') {
                    count++;
                }
         }
         return count;
    }
}
```



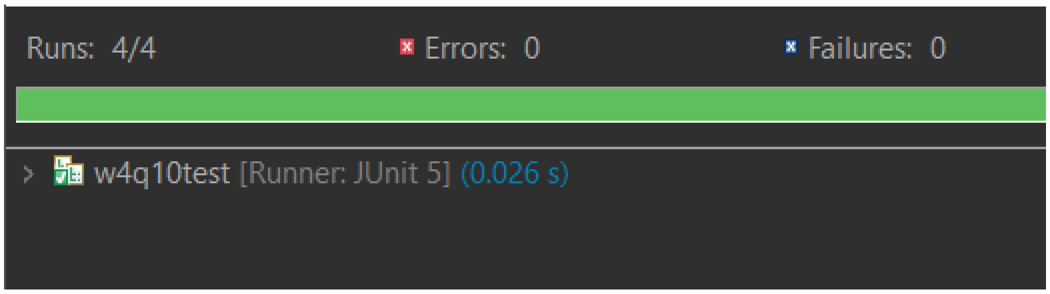
```
Runs: 1/1    Errors: 0    Failures: 0

w4q9test [Runner: JUnit 5] (0.032 s)
    test() (0.032 s)
```

10. You are developing a basic calculator application with operations like addition, subtraction, multiplication, and division. Each test case checks a specific operation.
Tasks:
Write a JUnit test using annotations that:

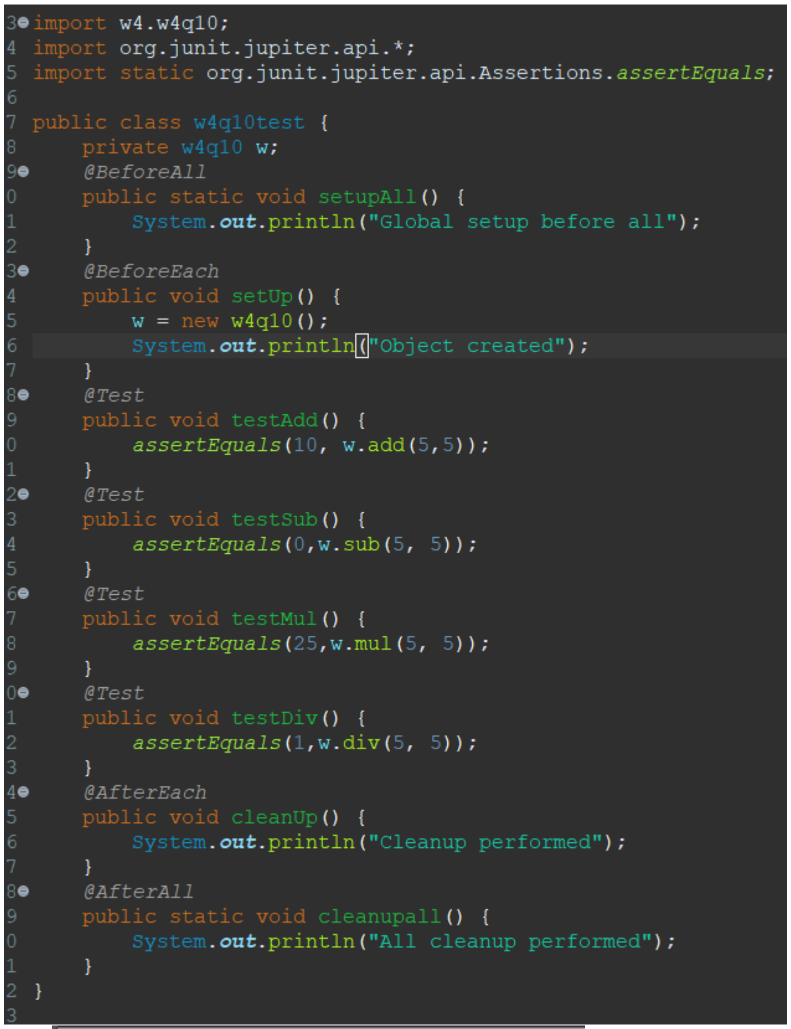
- **Before** each test, initializes a Calculator object.
- **After** each test, resets any necessary states or prints a message.
- **BeforeClass**: Set up any global configuration (if needed).
- **AfterClass**: Perform any clean-up after all tests are completed (e.g., release resources if any).



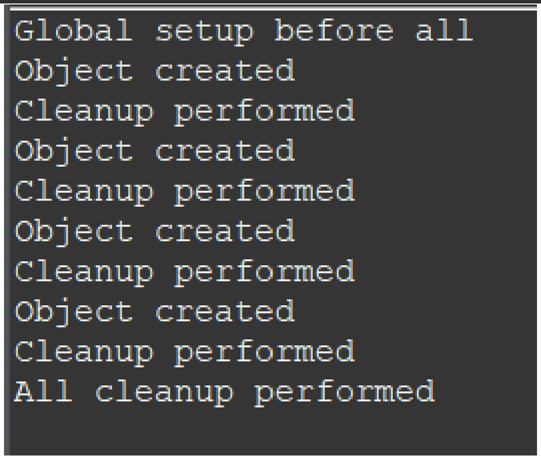
```
package w4;

public class w4q10 {
    int a, b;
    public int add(int a, int b) {
        return a+b;
    }
    public int sub(int a, int b) {
        return a-b;
    }
    public int mul(int a, int b) {
        return a*b;
    }
    public int div(int a, int b) {
        return a/b;
    }
}
```



```
Runs: 4/4          Errors: 0          Failures: 0

> w4q10test [Runner: JUnit 5] (0.026 s)
```



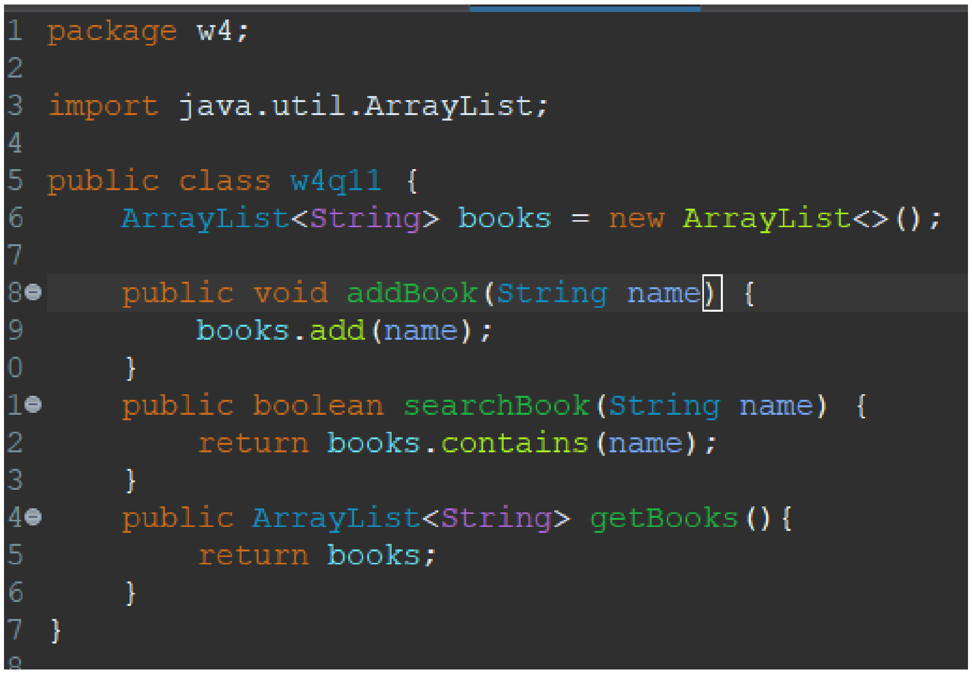
```java
import w4.w4q10;
import org.junit.jupiter.api.*;
import static org.junit.jupiter.api.Assertions.assertEquals;

public class w4q10test {
    private w4q10 w;
    @BeforeAll
    public static void setupAll() {
        System.out.println("Global setup before all");
    }
    @BeforeEach
    public void setUp() {
        w = new w4q10();
        System.out.println("Object created");
    }
    @Test
    public void testAdd() {
        assertEquals(10, w.add(5,5));
    }
    @Test
    public void testSub() {
        assertEquals(0,w.sub(5, 5));
    }
    @Test
    public void testMul() {
        assertEquals(25,w.mul(5, 5));
    }
    @Test
    public void testDiv() {
        assertEquals(1,w.div(5, 5));
    }
    @AfterEach
    public void cleanUp() {
        System.out.println("Cleanup performed");
    }
    @AfterAll
    public static void cleanupall() {
        System.out.println("All cleanup performed");
    }
}
```



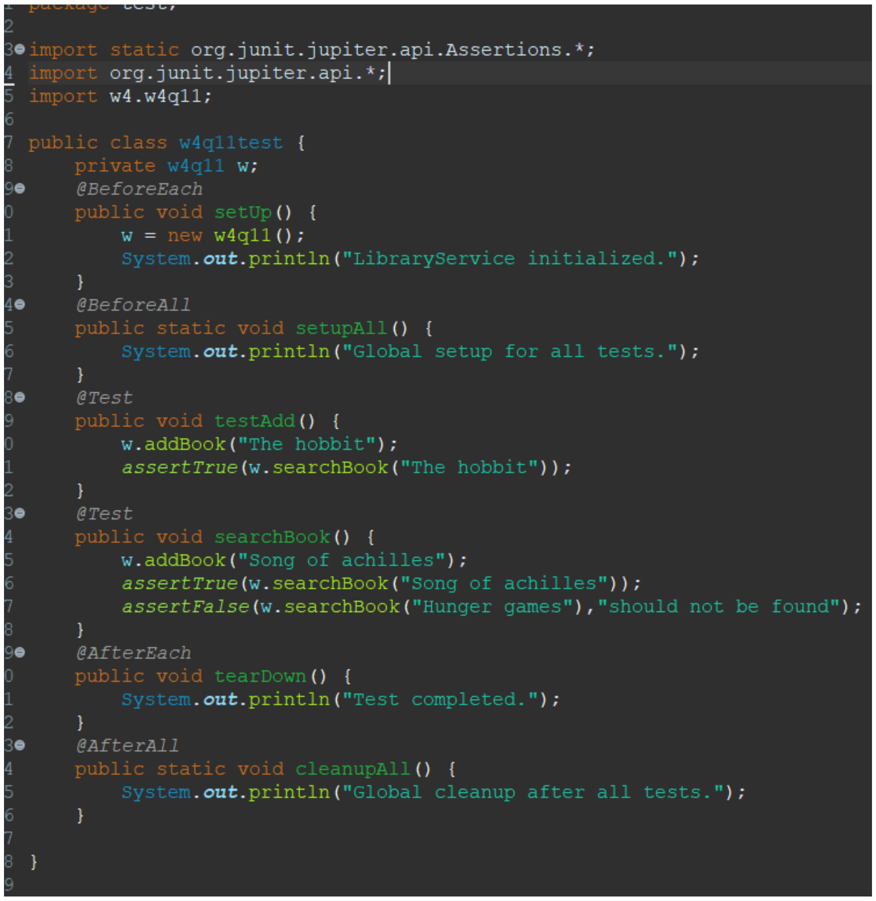
```
Global setup before all
Object created
Cleanup performed
Object created
Cleanup performed
Object created
Cleanup performed
Object created
Cleanup performed
All cleanup performed
```

11. You are given a **LibraryService** class that manages books in a library. The **LibraryService** allows adding books to the library and searching for books by title. The class uses an internal **ArrayList** to store the books.
    Your task is to write unit tests for the **LibraryService** class. You will need to test the methods for adding and searching for books using JUnit. Additionally, you must use the

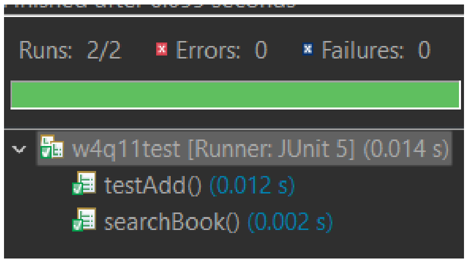
JUnit annotations (**@Before, @BeforeClass**, **@After**, **@AfterClass**) to manage setup and cleanup of resources during the tests.



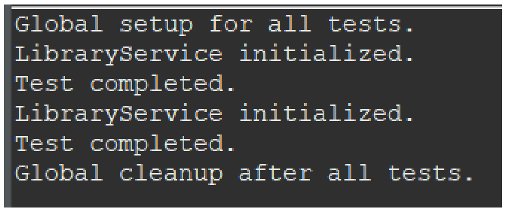
```java
package w4;

import java.util.ArrayList;

public class w4q11 {
    ArrayList<String> books = new ArrayList<>();

    public void addBook(String name) {
        books.add(name);
    }
    public boolean searchBook(String name) {
        return books.contains(name);
    }
    public ArrayList<String> getBooks(){
        return books;
    }
}
```



```java
package test;

import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.*;
import w4.w4q11;

public class w4q11test {
    private w4q11 w;
    @BeforeEach
    public void setUp() {
        w = new w4q11();
        System.out.println("LibraryService initialized.");
    }
    @BeforeAll
    public static void setupAll() {
        System.out.println("Global setup for all tests.");
    }
    @Test
    public void testAdd() {
        w.addBook("The hobbit");
        assertTrue(w.searchBook("The hobbit"));
    }
    @Test
    public void searchBook() {
        w.addBook("Song of achilles");
        assertTrue(w.searchBook("Song of achilles"));
        assertFalse(w.searchBook("Hunger games"),"should not be found");
    }
    @AfterEach
    public void tearDown() {
        System.out.println("Test completed.");
    }
    @AfterAll
    public static void cleanupAll() {
        System.out.println("Global cleanup after all tests.");
    }

}
```



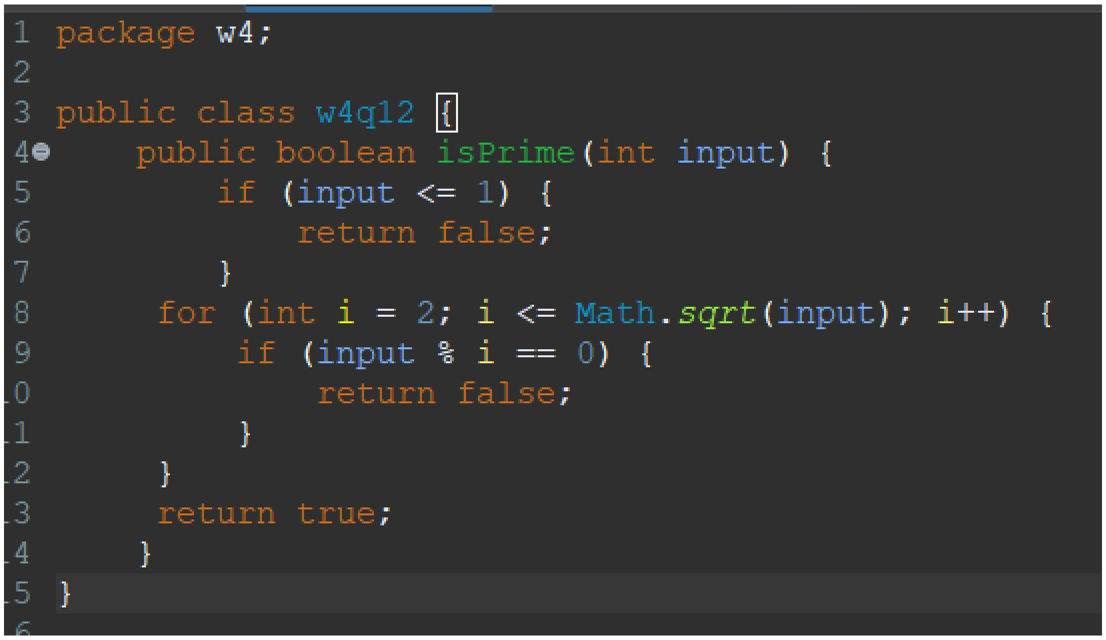
```
Runs: 2/2    Errors: 0    Failures: 0

w4q11test [Runner: JUnit 5] (0.014 s)
    testAdd() (0.012 s)
    searchBook() (0.002 s)
```



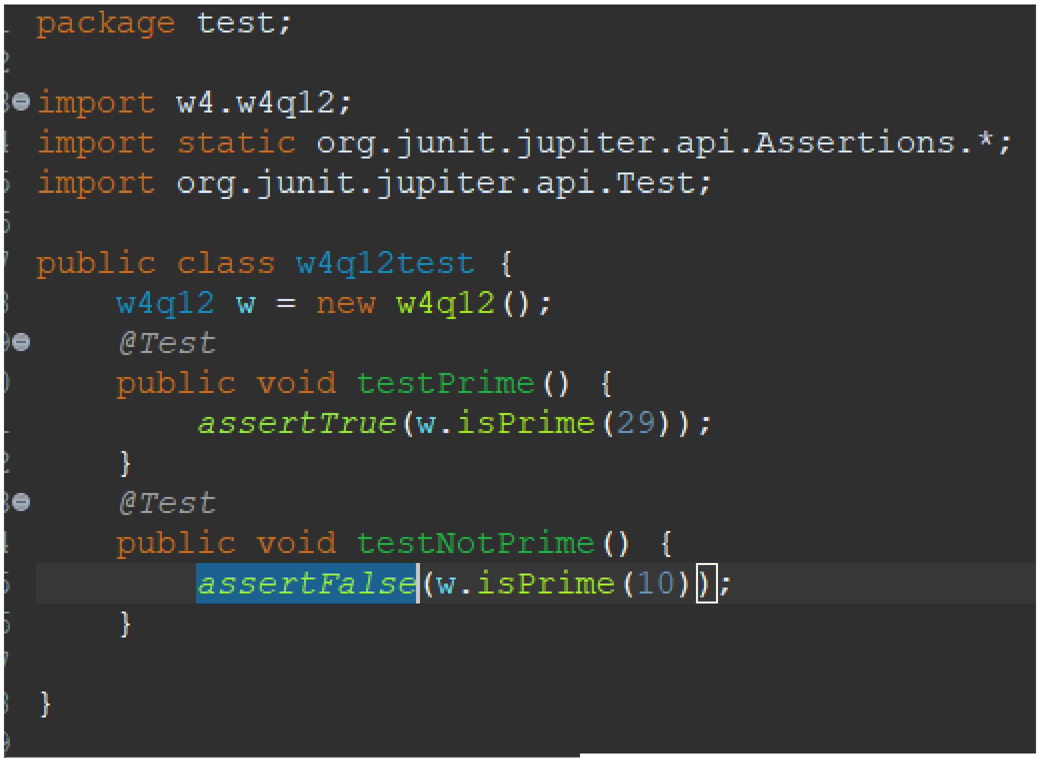
```
Global setup for all tests.
LibraryService initialized.
Test completed.
LibraryService initialized.
Test completed.
Global cleanup after all tests.
```

Follow the TDD Approach

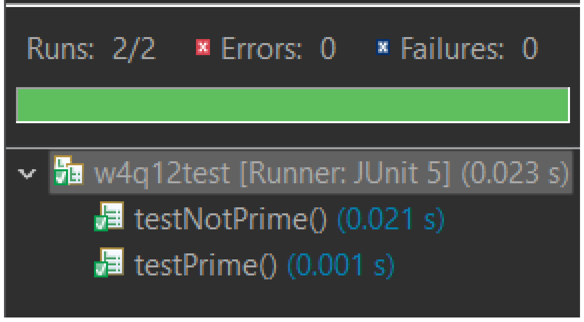
12. Write a function that takes an integer as input and returns True if it is a prime number, otherwise returns False.



```java
package w4;

public class w4q12 {
    public boolean isPrime(int input) {
        if (input <= 1) {
            return false;
        }
       for (int i = 2; i <= Math.sqrt(input); i++) {
           if (input % i == 0) {
               return false;
           }
       }
       return true;
    }
}
```

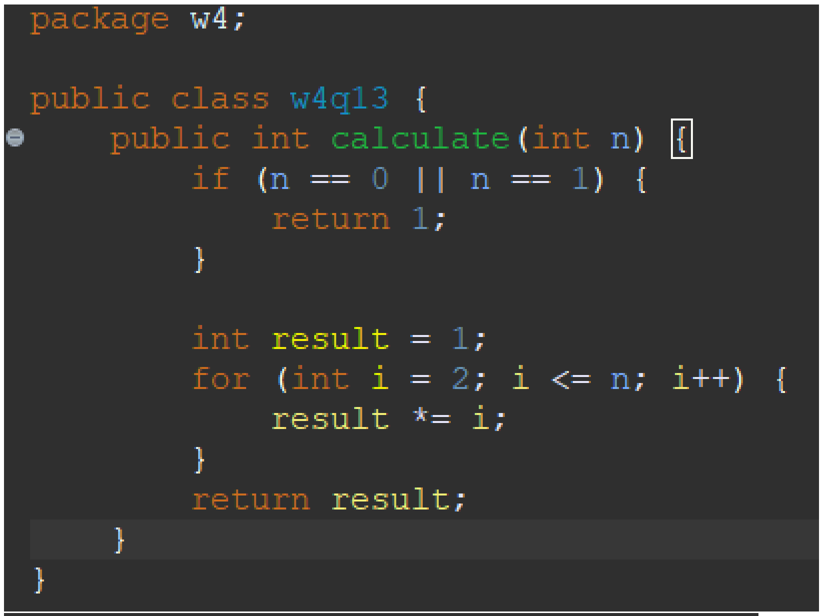


```
package test;

import w4.w4q12;
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;

public class w4q12test {
    w4q12 w = new w4q12();
    @Test
    public void testPrime() {
        assertTrue(w.isPrime(29));
    }
    @Test
    public void testNotPrime() {
        assertFalse(w.isPrime(10));
    }

}
```

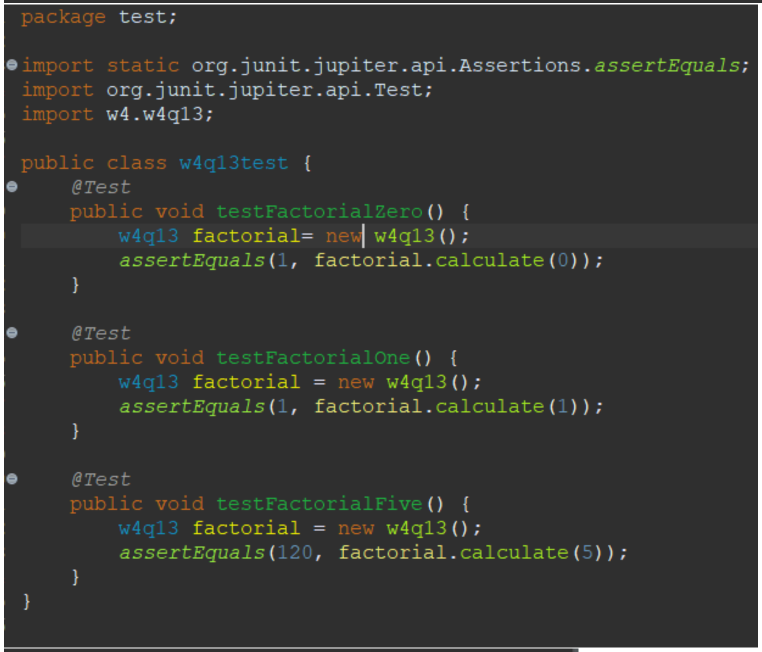


```
Runs: 2/2    Errors: 0    Failures: 0

w4q12test [Runner: JUnit 5] (0.023 s)
    testNotPrime() (0.021 s)
    testPrime() (0.001 s)
```

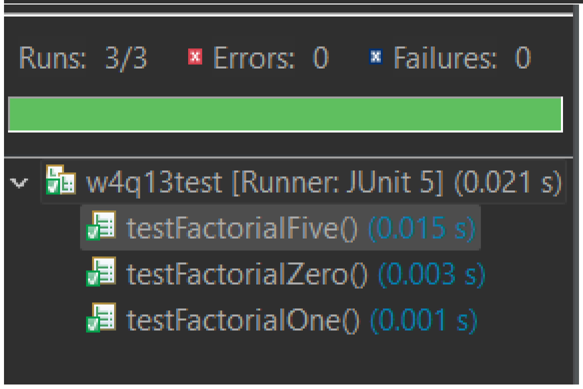
12. Write a function to calculate the factorial of a given non-negative integer.



```
package w4;

public class w4q13 {
    public int calculate(int n) {
        if (n == 0 || n == 1) {
            return 1;
        }

        int result = 1;
        for (int i = 2; i <= n; i++) {
            result *= i;
        }
        return result;
    }
}
```



```
package test;

import static org.junit.jupiter.api.Assertions.assertEquals;
import org.junit.jupiter.api.Test;
import w4.w4q13;

public class w4q13test {
    @Test
    public void testFactorialZero() {
        w4q13 factorial= new w4q13();
        assertEquals(1, factorial.calculate(0));
    }

    @Test
    public void testFactorialOne() {
        w4q13 factorial = new w4q13();
        assertEquals(1, factorial.calculate(1));
    }

    @Test
    public void testFactorialFive() {
        w4q13 factorial = new w4q13();
        assertEquals(120, factorial.calculate(5));
    }
}
```



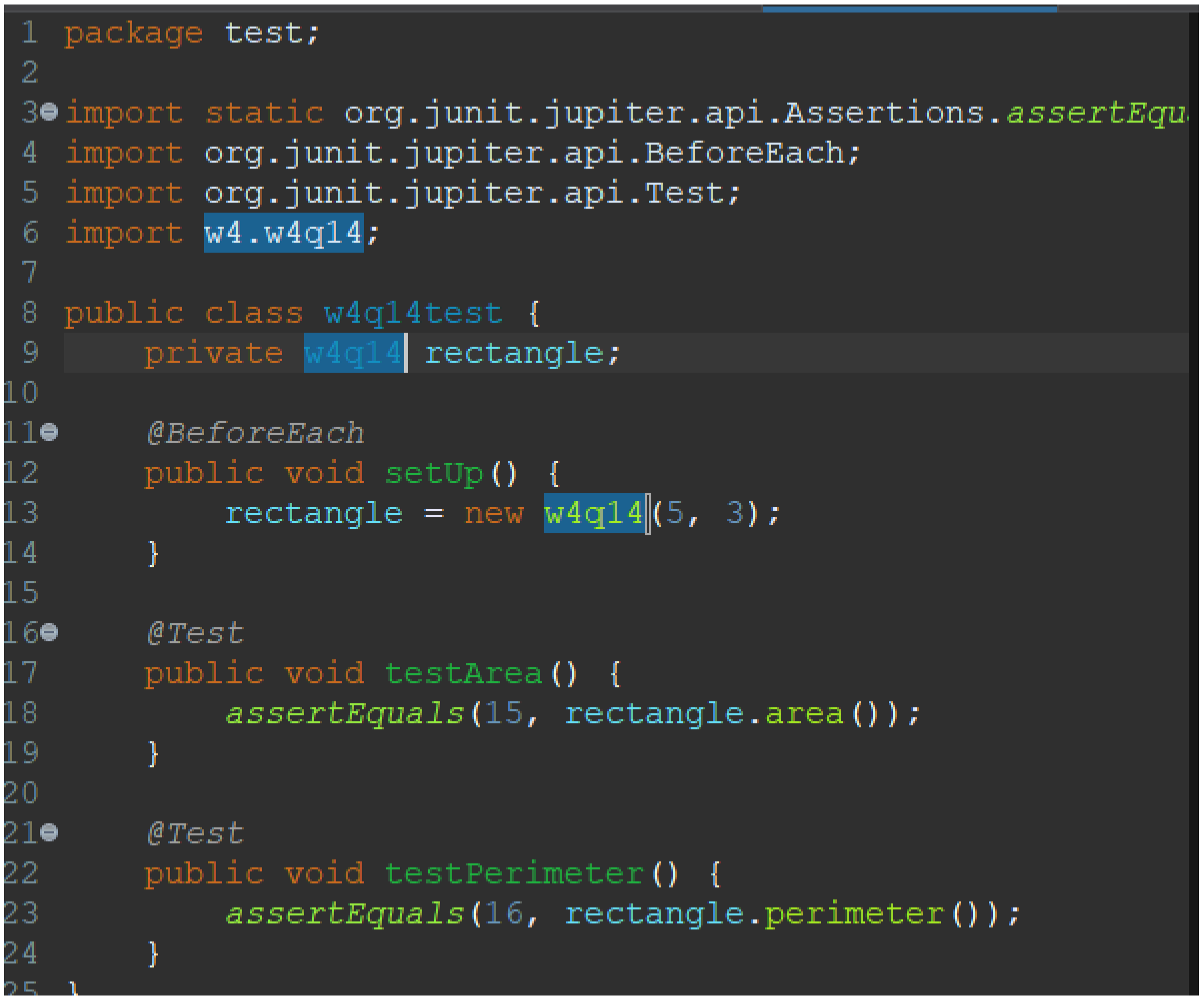
```
Runs: 3/3    Errors: 0    Failures: 0

w4q13test [Runner: JUnit 5] (0.021 s)
    testFactorialFive() (0.015 s)
    testFactorialZero() (0.003 s)
    testFactorialOne() (0.001 s)
```

14. Create a class Rectangle with the following:

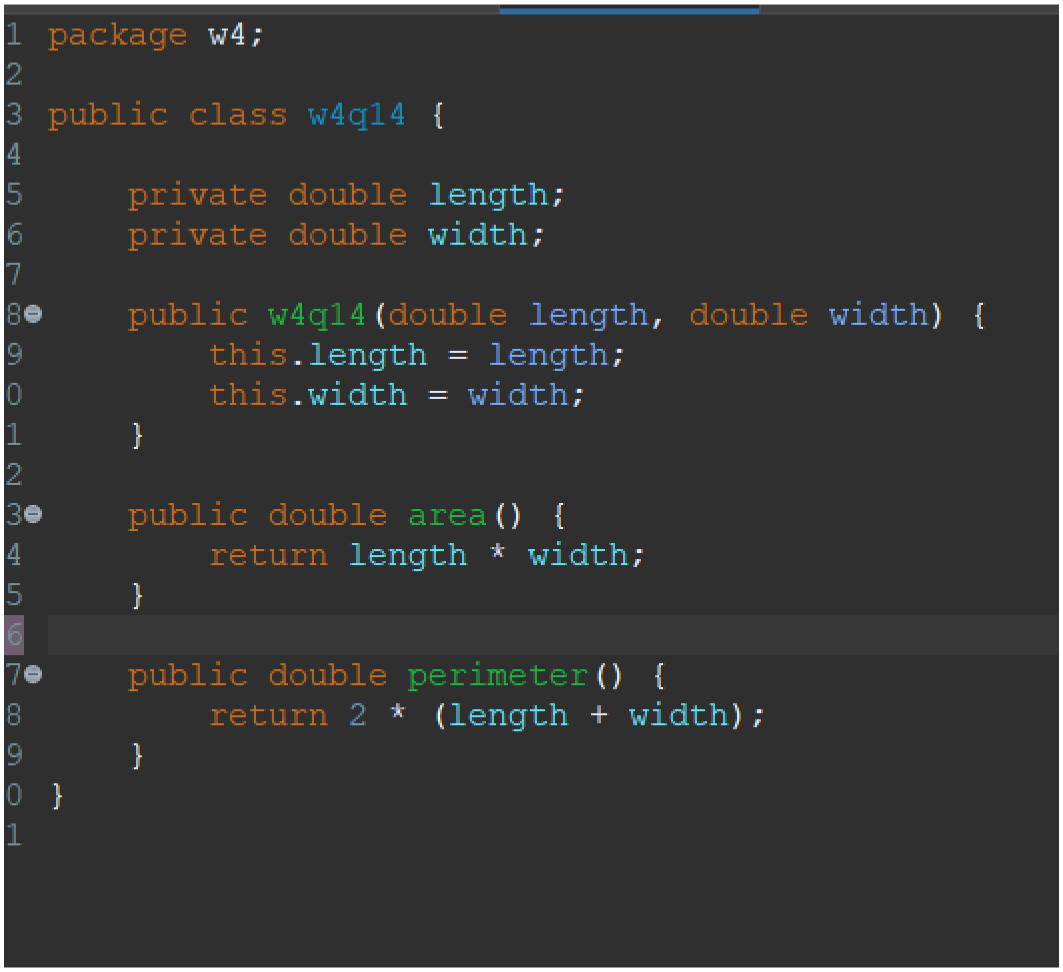
- Attributes: length and width.
- Methods: area() to calculate the area of the rectangle.

  perimeter() to calculate the perimeter of the rectangle.

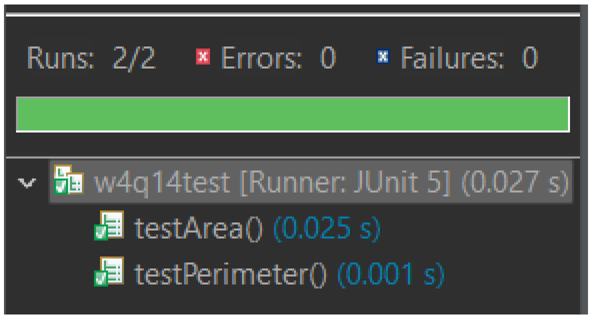
- Create a test cases



```java
package test;

import static org.junit.jupiter.api.Assertions.assertEqu
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import w4.w4q14;

public class w4q14test {
    private w4q14 rectangle;

    @BeforeEach
    public void setUp() {
        rectangle = new w4q14(5, 3);
    }

    @Test
    public void testArea() {
        assertEquals(15, rectangle.area());
    }

    @Test
    public void testPerimeter() {
        assertEquals(16, rectangle.perimeter());
    }
}
```



```java
package w4;

public class w4q14 {

    private double length;
    private double width;

    public w4q14(double length, double width) {
        this.length = length;
        this.width = width;
    }

    public double area() {
        return length * width;
    }

    public double perimeter() {
        return 2 * (length + width);
    }
}

```

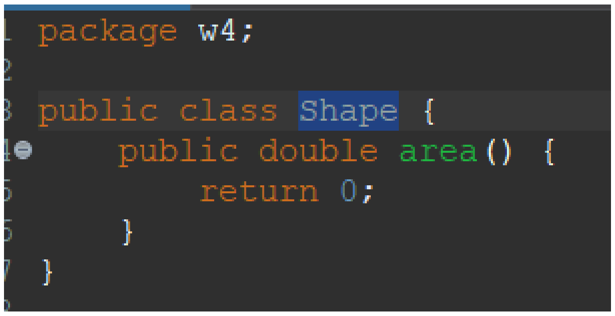


```
Runs: 2/2    Errors: 0    Failures: 0

w4q14test [Runner: JUnit 5] (0.027 s)
    testArea() (0.025 s)
    testPerimeter() (0.001 s)
```

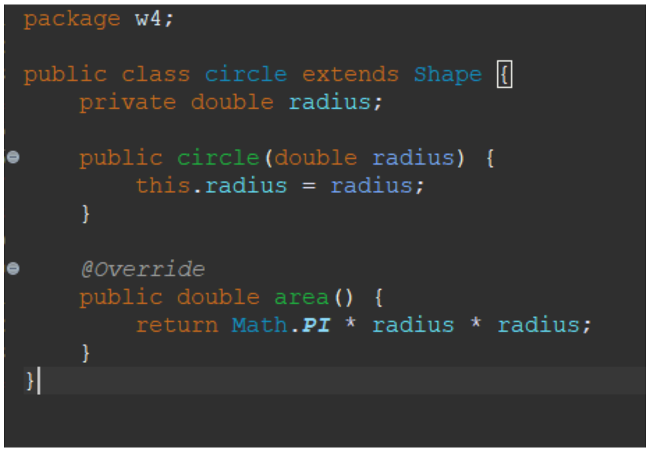
15.Create a base class Shape with a method area() that returns 0.

Create two derived classes:

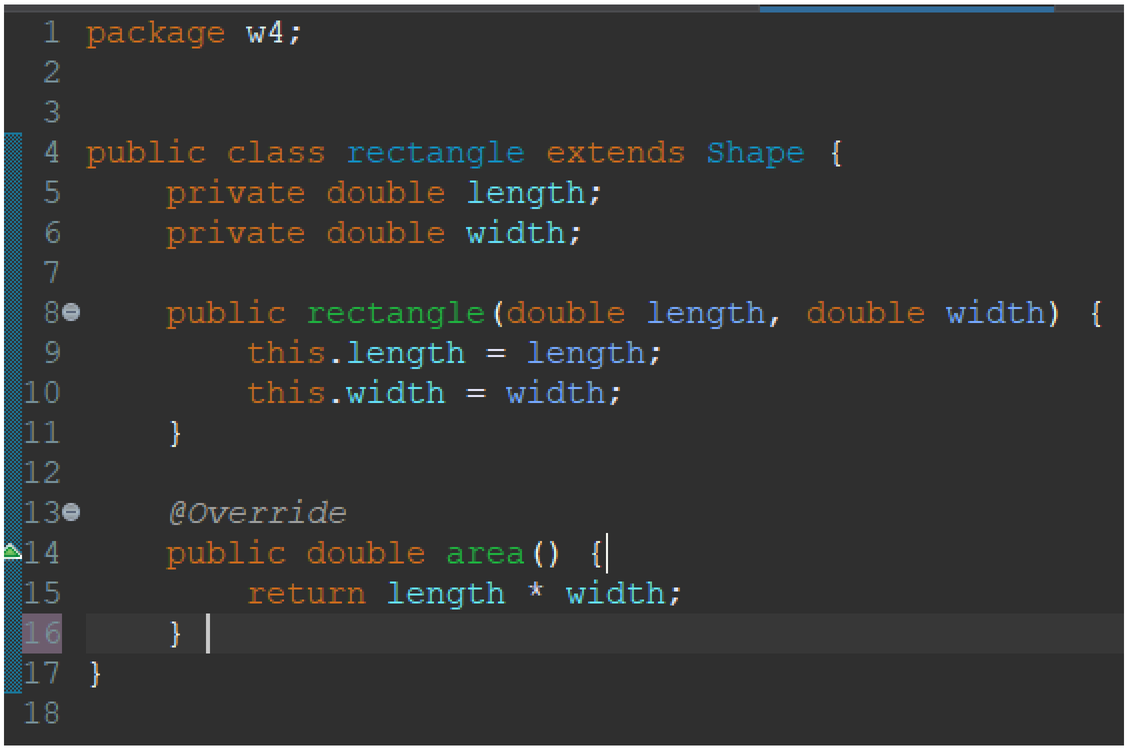
- Circle with attribute radius and area() method to calculate the area
- Rectangle with attributes length and width and area() method to calculate the area.



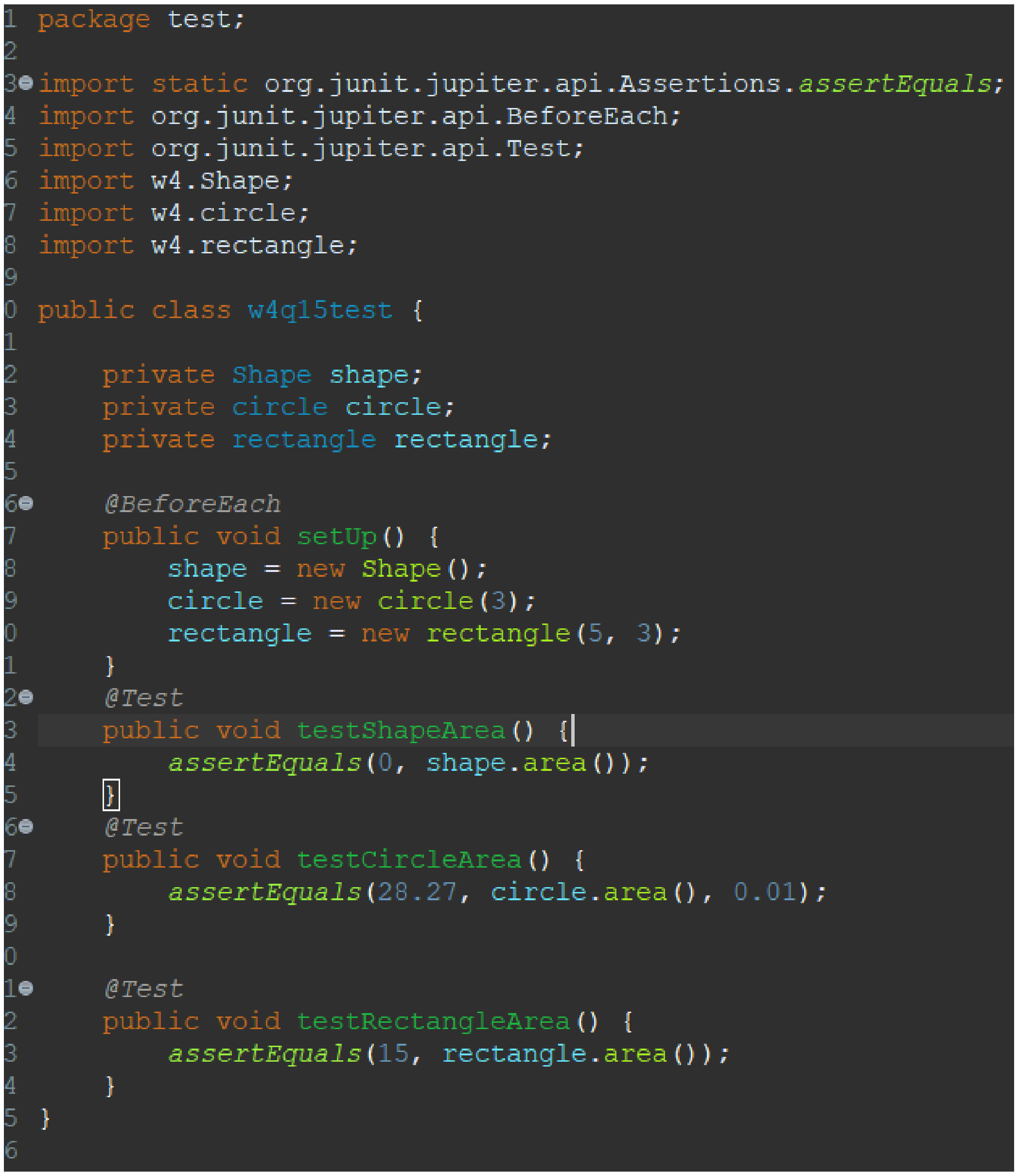
```java
package w4;

public class Shape {
    public double area() {
        return 0;
    }
}
```



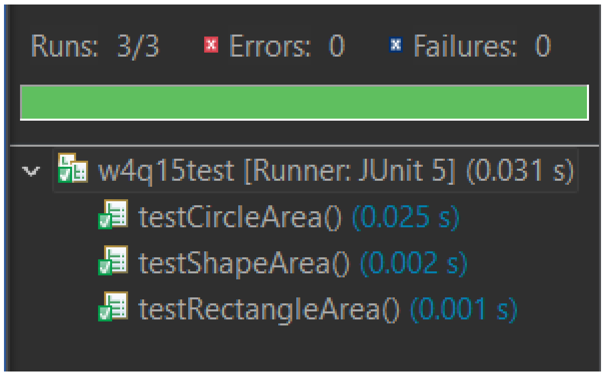
```java
package w4;

public class circle extends Shape {
    private double radius;

    public circle(double radius) {
        this.radius = radius;
    }

    @Override
    public double area() {
        return Math.PI * radius * radius;
    }
}
```



```java
package w4;


public class rectangle extends Shape {
    private double length;
    private double width;

    public rectangle(double length, double width) {
        this.length = length;
        this.width = width;
    }

    @Override
    public double area() {
        return length * width;
    }
}

```



```java
package test;

import static org.junit.jupiter.api.Assertions.assertEquals;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import w4.Shape;
import w4.circle;
import w4.rectangle;

public class w4q15test {

    private Shape shape;
    private circle circle;
    private rectangle rectangle;

    @BeforeEach
    public void setUp() {
        shape = new Shape();
        circle = new circle(3);
        rectangle = new rectangle(5, 3);
    }
    @Test
    public void testShapeArea() {
        assertEquals(0, shape.area());
    }
    @Test
    public void testCircleArea() {
        assertEquals(28.27, circle.area(), 0.01);
    }

    @Test
    public void testRectangleArea() {
        assertEquals(15, rectangle.area());
    }
}
```



```
Runs:  3/3    Errors:  0    Failures:  0

w4q15test [Runner: JUnit 5] (0.031 s)
    testCircleArea() (0.025 s)
    testShapeArea() (0.002 s)
    testRectangleArea() (0.001 s)
```