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

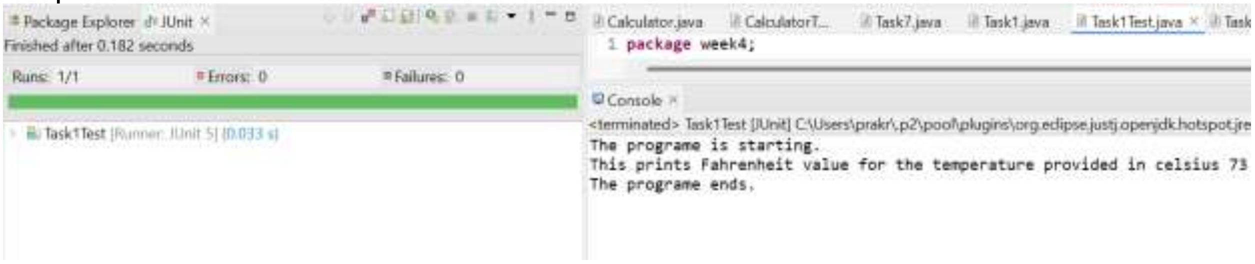
1. Write tests for converting temperatures from Celsius to Fahrenheit and vice versa.
   Hint: Use **assertEquals(expected, actual)** to compare the expected result with the actual result returned by the method.

Code/Implementation:

```
package week4;

public class Task1 {

	int celciusToFahrenheit(int temp) {
		return (int)((temp * 9.0 / 5) + 32);
	}
}

package week4;

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

import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Test;

class Task1Test {

	Task1 task1;

	@BeforeAll

	static void setUp() {
		System.out.println("The programe is starting.");
	}

	@AfterAll

	static void end() {
		System.out.println("The programe ends.");
	}

	@Test
	void testCalciusToFahrenheit() {

		Task1 task1 = new Task1();
		int result = task1.celciusToFahrenheit(23);
```

```
                assertEquals(73, result);
                System.out.println("This prints Fahrenheit value for the temperature provided in celsius "+ result);
        }

}
```

Output:



```
Finished after 0.182 seconds
Runs: 1/1        Errors: 0        Failures: 0
Task1Test [Runner: JUnit 5] (0.033 s)
```

```
1 package week4;
```

```
<terminated> Task1Test [JUnit] C:\Users\prakr\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre
The programe is starting.
This prints Fahrenheit value for the temperature provided in celsius 73
The programe ends.
```

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.

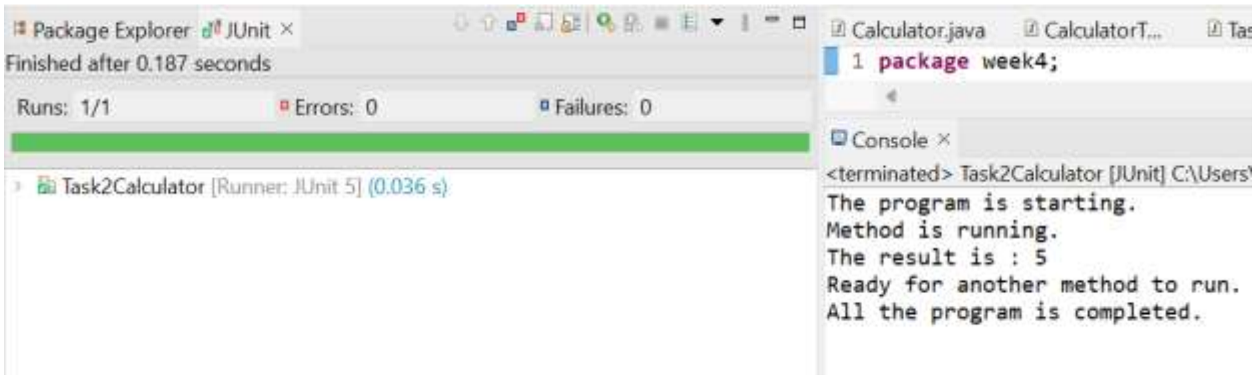
Code/Implementation:

```
package week4;

public class Calculator {

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

}
```

```
package week4;

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

import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
```

```
class Task2Calculator {

	static Calculator addObj;

	@BeforeAll

	static void setUp() {
		addObj=new Calculator();
		System.out.println("The program is starting.");
	}

	@AfterAll

	static void ends() {
		System.out.println("All the program is completed.");
	}

	@BeforeEach
	void runBeforeEach () {
		System.out.println("Method is running.");
	}

	@AfterEach
	 void runAfterEach () {
		System.out.println("Ready for another method to run.");
	}

	@Test
	void testCalculatorFunction() {
		int result=addObj.add(2,3);
		assertEquals(5,result);
		System.out.println("The result is : "+result);


	}
}
```

Output:



```
Finished after 0.187 seconds
Runs: 1/1        Errors: 0        Failures: 0
Task2Calculator [Runner: JUnit 5] (0.036 s)
```

```
package week4;
```

```
<terminated> Task2Calculator [JUnit] C:\Users\
The program is starting.
Method is running.
The result is : 5
Ready for another method to run.
All the program is completed.
```

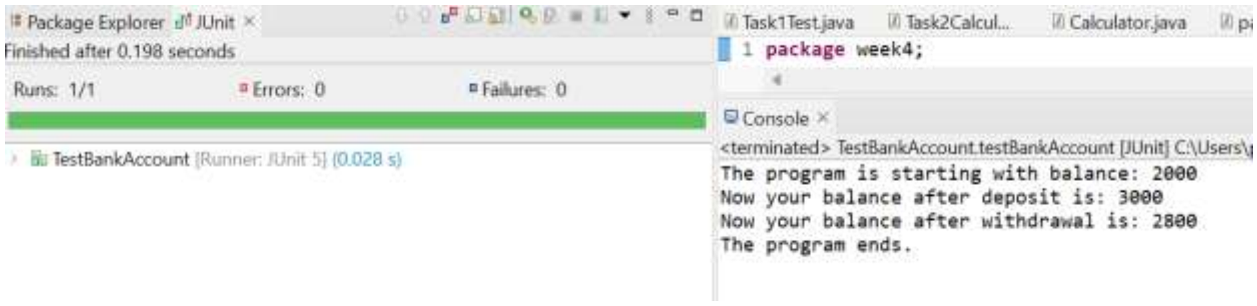
3. Write a class **BankAccount** with methods **deposit(double amount)** and withdraw(double amount). The account balance should start at 0.0, and the methods should update the balance accordingly.
   Write a JUnit test that:

- 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)

Code/Implementation :

```
package week4;

public class BankAccount {

        int balance;
        BankAccount(int balance){
                this.balance=balance;
        }

        int deposit(int amount) {
                balance+=amount;
        return balance;
        }

        int withdraw(int amount) {
           if(amount > 0) {
              if(balance >= amount) {
                balance-=amount;
                return balance;
              } else {
                 System.out.println("Insufficient Balance to withdraw.");
              }
           } else {
              System.out.println("Withdraw amount is not valid.");
           }
        return 0;
          }

}

package week4;

import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Test;

class TestBankAccount {
    static BankAccount acc;

    @BeforeAll
    static void setUp() {
        acc = new BankAccount(2000);
        System.out.println("The program is starting with balance: " + acc.balance);
    }

    @AfterAll
    static void end() {
        System.out.println("The program ends.");
    }

    @Test
    void testBankAccount() {
        int result1 = acc.deposit(1000);
        assertEquals(3000, result1);
        System.out.println("Now your balance after deposit is: " + result1);

        int result2 = acc.withdraw(200);
        assertEquals(2800, result2);
        System.out.println("Now your balance after withdrawal is: " + result2);

    }
}
```

Output:


Package Explorer JUnit

Finished after 0.198 seconds

Runs: 1/1 Errors: 0 Failures: 0

TestBankAccount [Runner: JUnit 5] (0.028 s)

Task1Test.java Task2Calcul... Calculator.java

```
package week4;
```

Console

```
<terminated> TestBankAccount.testBankAccount [JUnit] C:\Users\
The program is starting with balance: 2000
Now your balance after deposit is: 3000
Now your balance after withdrawal is: 2800
The program ends.
```

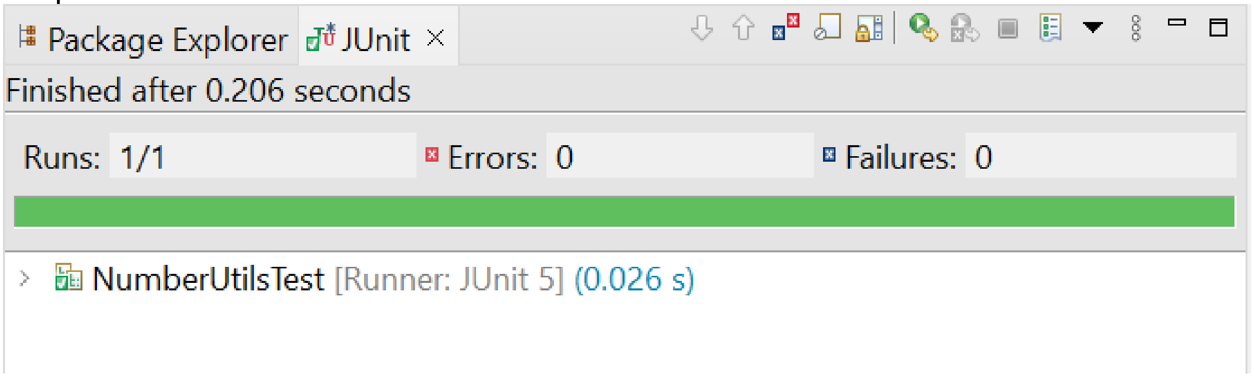
4. Create a method **getEvenNumbers(int[] numbers)** in a **NumberUtils** class that filters out and returns only the even numbers from a given array of integers. Write a JUnit test case to verify that the method correctly returns a list of even numbers.
   For example:
   Input: [1, 2, 3, 4, 5, 6]
   Expected Output: [2, 4, 6]

Code/Implementation:

```
package week4;

import java.util.ArrayList;

public class NumberUtils {

        int[] getEvenNumbers(int[] numbers) {
                int n=numbers.length;
                ArrayList<Integer>newArray=new ArrayList<>();
                for(int i=0;i<n;i++) {
                        if(numbers[i]%2==0) {
                        newArray.add(numbers[i]);
                        }
                      }
                 int[] result = new int[newArray.size()];
                        for (int i = 0; i < newArray.size(); i++) {
                           result[i] = newArray.get(i);
                }
                return result;
        }

}

package week4;

import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;
```

```
class NumberUtilsTest {

    @Test
    void testGetEvenNumbers() {
        NumberUtils utils = new NumberUtils();
        int[] input = {1, 2, 3, 4, 5, 6};
        int[] expected = {2, 4, 6};

        assertArrayEquals(expected, utils.getEvenNumbers(input));
    }
}
```

Outputs:



```
Finished after 0.206 seconds
Runs: 1/1    Errors: 0    Failures: 0
NumberUtilsTest [Runner: JUnit 5] (0.026 s)
```

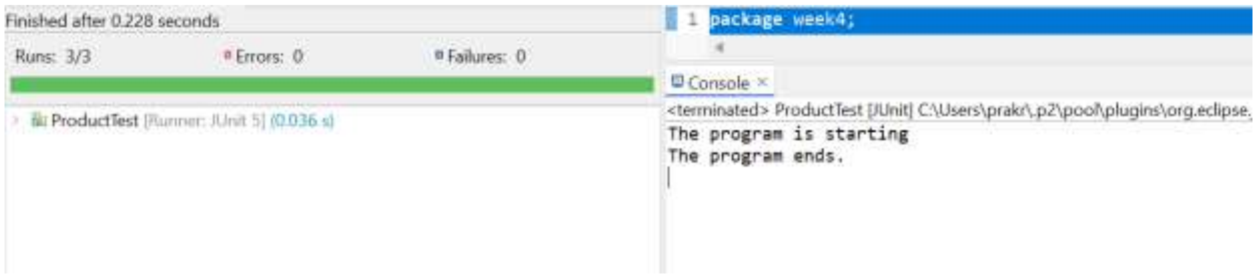
5. **Complex Assertion with assertAll**
   Write a class **Product** with fields **name** (String), **price** (double), and **quantity** (int). Write a method **isAffordable**(double budget) that returns true if the total price (price * quantity) is less than or equal to the given budget. Write a JUnit 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.

Code/Implementation:

```
package week4;
public class Product {
        String name;
        double price;
        int quantity;

        public Product(String name,double price, int quantity) {
        if (name == null || name.isEmpty()) {
            throw new IllegalArgumentException("Name cannot be null or empty");
        }
        if (price <= 0) {
            throw new IllegalArgumentException("Price must be positive");
        }
        this.name= name;
        this.price=price;
        this.quantity=quantity;
        }

        boolean isAffordable(double budget) {
                    return (price * quantity) <= budget;
                }
}

package week4;

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

import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Test;

class ProductTest {

        static Product product;

    @BeforeAll
    static void setUp() {
        product = new Product("TestProduct", 10.0, 2);
        System.out.println("The program is starting ");
    }

    @AfterAll
    static void end() {
        System.out.println("The program ends.");
    }

    @Test
    void testProductNameNotNull() {
        Exception exception = assertThrows(IllegalArgumentException.class, () -> {
            new Product(null, 10.0, 1);
        });
        assertEquals("Name cannot be null or empty", exception.getMessage());

        exception = assertThrows(IllegalArgumentException.class, () -> {
            new Product("", 10.0, 1);
        });
        assertEquals("Name cannot be null or empty", exception.getMessage());
    }

    @Test
    void testPricePositive() {
        Exception exception = assertThrows(IllegalArgumentException.class, () -> {
            new Product("TestProduct", -5.0, 1);
        });
        assertEquals("Price must be positive", exception.getMessage());

        exception = assertThrows(IllegalArgumentException.class, () -> {
            new Product("TestProduct", 0, 1);
        });
        assertEquals("Price must be positive", exception.getMessage());
    }

    @Test
    void testIsAffordable() {
        assertTrue(product.isAffordable(20.0));
        assertFalse(product.isAffordable(15.0));
        assertTrue(product.isAffordable(25.0));
    }
}
```

OUTPUT:



```
Finished after 0.228 seconds
Runs: 3/3        Errors: 0        Failures: 0
ProductTest [Runner: JUnit 5] (0.036 s)

package week4;

Console
<terminated> ProductTest [JUnit] C:\Users\prakr\.p2\pool\plugins\org.eclipse.
The program is starting
The program ends.
```

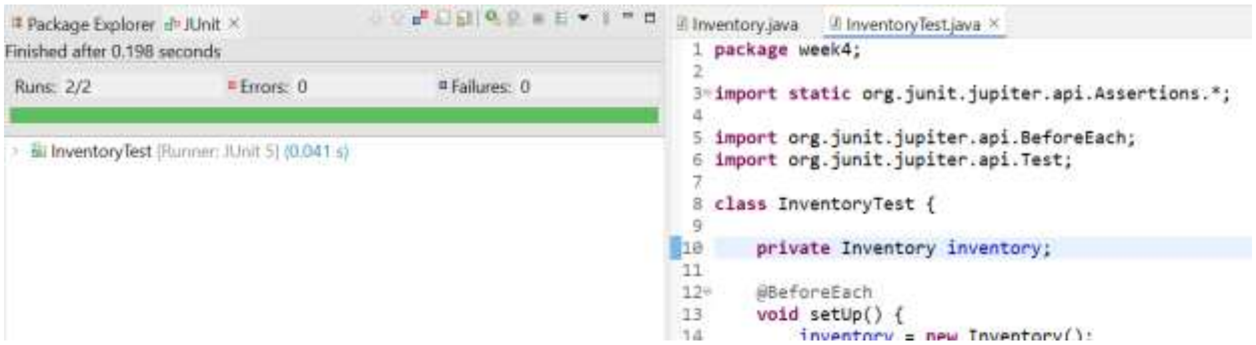
6. In an inventory management system, you need a method **isProductAvailable(String productName, int quantity)** to check if the given product is in stock. The method should return true if the requested quantity is available in stock and false if the requested quantity exceeds the available stock.

   Code/Implementation:

```
package week4;

import java.util.HashMap;
import java.util.Map;

public class Inventory {
    private Map<String, Product> products;
    public Inventory() {
        products = new HashMap<>();
    }

    public void addProduct(Product product) {
        products.put(product.name, product);
    }

    public boolean isProductAvailable(String productName, int requestedQuantity) {
        if (requestedQuantity < 0) {
            throw new IllegalArgumentException("Requested quantity must be non-
negative.");
        }
        Product product = products.get(productName);
        if (product == null) {
            return false; // Product does not exist
        }
        return product.isProductAvailable(requestedQuantity);
    }
}

class Product {
    String name;
    double price;
    int quantity;

    public Product(String name, double price, int quantity) {
        if (name == null || name.isEmpty()) {
            throw new IllegalArgumentException("Name cannot be null or empty");
        }
        if (price <= 0) {
            throw new IllegalArgumentException("Price must be positive");
        }
        if (quantity < 0) {
            throw new IllegalArgumentException("Quantity cannot be negative");
        }
        this.name = name;
        this.price = price;
        this.quantity = quantity;
    }

    public boolean isProductAvailable(int requestedQuantity) {
        if (requestedQuantity < 0) {
            throw new IllegalArgumentException("Requested quantity must be non-
negative.");
```

```java
        }
        return quantity >= requestedQuantity;
    }
}

package week4;

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

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

class InventoryTest {

    private Inventory inventory;

    @BeforeEach
    void setUp() {
        inventory = new Inventory();
        inventory.addProduct(new Product("Laptop", 1000, 10));
        inventory.addProduct(new Product("Phone", 500, 5));
    }

    @Test
    void testIsProductAvailable() {
        assertTrue(inventory.isProductAvailable("Laptop", 5));
        assertFalse(inventory.isProductAvailable("Laptop", 15));
        assertFalse(inventory.isProductAvailable("Tablet", 1));
        assertFalse(inventory.isProductAvailable("Phone", 6));
    }

    @Test
    void testNegativeRequestedQuantity() {
        Exception exception = assertThrows(IllegalArgumentException.class, () -> {
            inventory.isProductAvailable("Laptop", -1);
        });
        assertEquals("Requested quantity must be non-negative.",
exception.getMessage());
    }
}
```

Output:



```
Finished after 0.198 seconds
Runs: 2/2        Errors: 0        Failures: 0
InventoryTest [Runner: JUnit 5] (0.041 s)
```

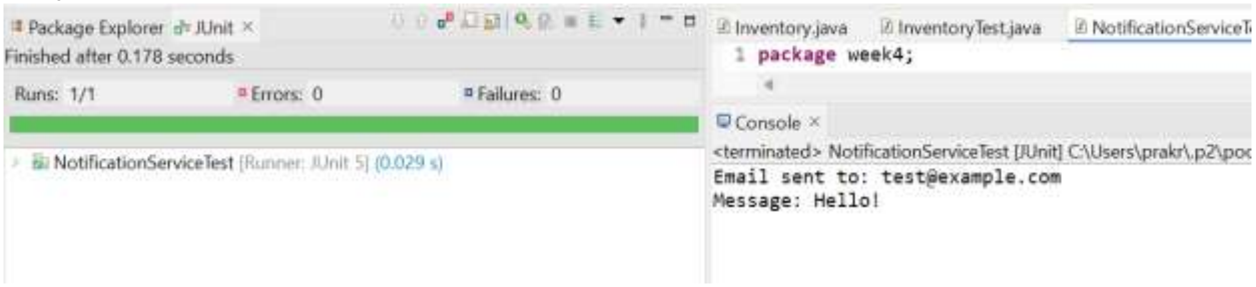
```java
package week4;

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

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

class InventoryTest {

    private Inventory inventory;

    @BeforeEach
    void setUp() {
        inventory = new Inventory();
```

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.

Code/Implementation:

```java
package week4;

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

import org.junit.jupiter.api.Test;

class NotificationServiceTest {

    @Test
    void testSendEmail() {
        NotificationService service = new NotificationService();
        assertTrue(service.sendEmail("test@example.com", "Hello!"));
        assertFalse(service.sendEmail("invalid-email", "Hello!"));
        assertFalse(service.sendEmail(null, "Hello!"));
        assertFalse(service.sendEmail("", "Hello!"));
        assertFalse(service.sendEmail("test@example.com", ""));
    }
}

package week4;
import java.util.regex.Pattern;

class NotificationService {

    private static final String EMAIL_REGEX = "^[A-Za-z0-9+_.-]+@[A-Za-z0-9.-]+$";
    private boolean isValidEmail(String email) {
        return Pattern.matches(EMAIL_REGEX, email);
    }

    public boolean sendEmail(String email, String message) {
        if (email == null || message == null || email.isEmpty() || message.isEmpty()) {
            return false;
        }
        else if (!isValidEmail(email)){
            return false;
        }
        System.out.println("Email sent to: " + email);
        System.out.println("Message: " + message);
        return true;
    }
}
```

Output:



Package Explorer JUnit ×

Finished after 0.178 seconds

Runs: 1/1 Errors: 0 Failures: 0

NotificationServiceTest [Runner: JUnit 5] (0.029 s)

Inventory.java InventoryTest.java NotificationServiceT

```
package week4;
```

Console ×

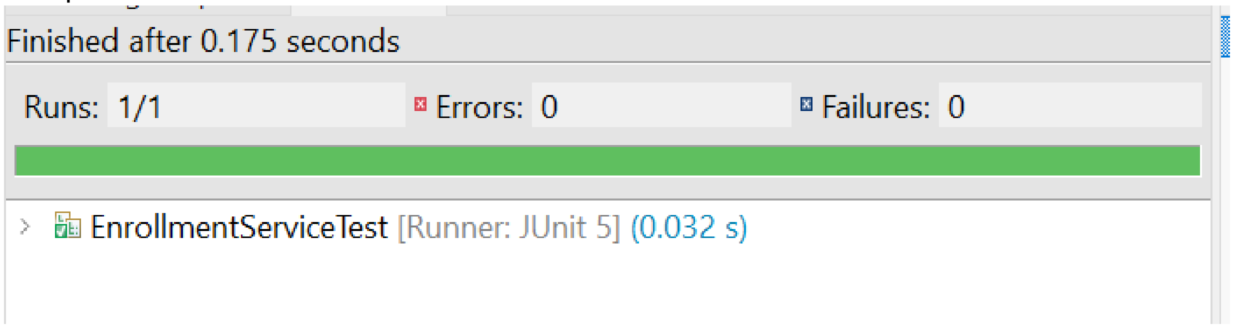
<terminated> NotificationServiceTest [JUnit] C:\Users\prakr\.p2\poc

```
Email sent to: test@example.com
Message: Hello!
```

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.
   Code/Implementation:

```
package week4;

import java.util.HashMap;
import java.util.Map;

public class EnrollmentService {
    private Map<String, String> enrolledCourses;

    public EnrollmentService() {
        enrolledCourses = new HashMap<>();
    }

    public boolean enrollStudent(String studentUsername, String courseName) {
        if (enrolledCourses.containsKey(studentUsername)) {
            return false;
        }
        enrolledCourses.put(studentUsername, courseName);
        return true;
    }
}

package week4;

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

class EnrollmentServiceTest {

    @Test
    void testEnrollStudent() {
        EnrollmentService enrollmentService = new EnrollmentService();
        assertTrue(enrollmentService.enrollStudent("student1", "Math101"));
        assertFalse(enrollmentService.enrollStudent("student1", "Math101"));
    }
}
```

Output:



```
Finished after 0.175 seconds
Runs: 1/1        Errors: 0        Failures: 0
> EnrollmentServiceTest [Runner: JUnit 5] (0.032 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.

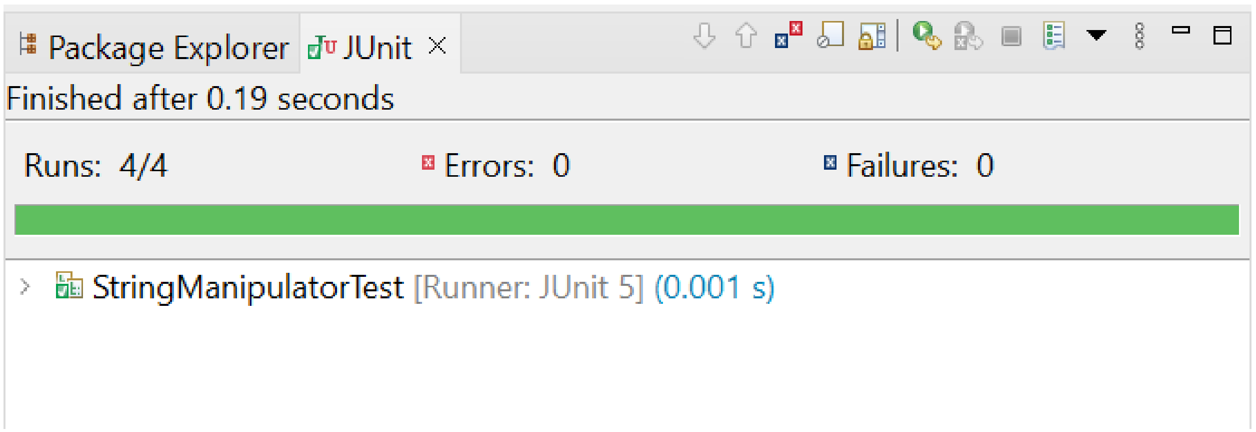
Code/Implamantation:

```
package week4;
```

```
public class StringManipulator {

    public String reverse(String input) {
        if (input == null) return null;
        return new StringBuilder(input).reverse().toString();
    }

    public String toUpperCase(String input) {
        if (input == null) return null;
        return input.toUpperCase();
    }

    public boolean isPalindrome(String input) {
        if (input == null) return false;
        String reversed = reverse(input);
        return input.equalsIgnoreCase(reversed);
    }

    public int countVowels(String input) {
        if (input == null) return 0;
        int count = 0;
        String vowels = "aeiouAEIOU";
        for (char c : input.toCharArray()) {
            if (vowels.indexOf(c) != -1) {
                count++;
            }
        }
        return count;
    }
}
```

```
package week4;

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

import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.TestInstance;

@TestInstance(TestInstance.Lifecycle.PER_CLASS)
class StringManipulatorTest {
    StringManipulator manipulator;

    @BeforeAll
    static void setup() {
        System.out.println("The program is starting");
    }

    @BeforeAll
    void setUp() {
        manipulator = new StringManipulator();
    }

    @Test
    void testReverse() {
        assertEquals("tac", manipulator.reverse("cat"));
        assertEquals("", manipulator.reverse(""));
        assertNull(manipulator.reverse(null));
    }

    @Test
    void testToUpperCase() {
        assertEquals("HELLO", manipulator.toUpperCase("hello"));
        assertEquals("", manipulator.toUpperCase(""));
        assertNull(manipulator.toUpperCase(null));
    }

    @Test
    void testIsPalindrome() {
        assertTrue(manipulator.isPalindrome("madam"));
        assertTrue(manipulator.isPalindrome("Madam"));
        assertFalse(manipulator.isPalindrome("hello"));
        assertFalse(manipulator.isPalindrome(null));
    }

    @Test
    void testCountVowels() {
        assertEquals(5, manipulator.countVowels("education"));
        assertEquals(0, manipulator.countVowels("rhythm"));
        assertEquals(0, manipulator.countVowels(""));
        assertEquals(0, manipulator.countVowels(null));
    }
}
```

Output:


Package Explorer JUnit

Finished after 0.19 seconds

Runs: 4/4 Errors: 0 Failures: 0

StringManipulatorTest [Runner: JUnit 5] (0.001 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:

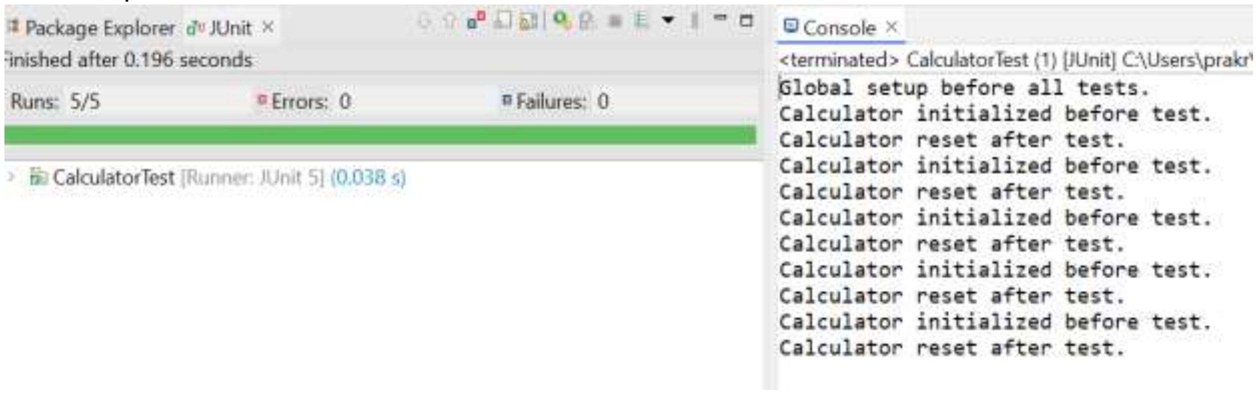
- **Before** each test, initializes a Calculator object.
- **After** each test, resets any necessary states or prints a message.
- **BeforeClass**: Set up any global configuration (if needed).
- **AfterClass**: Perform any clean-up after all tests are completed (e.g., release resources if any).

Code/Implamentation:

```java
package week4;
public class Calculator {
   public int add(int a, int b) {
      return a + b;
   }

   public int subtract(int a, int b) {
      return a - b;
   }

   public int multiply(int a, int b) {
      return a * b;
   }

   public double divide(int a, int b) {
      if (b == 0) {
         throw new ArithmeticException("Division by zero");
      }
      return (double) a / b;
   }
}

package week4;
```

```java
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.Test;

class CalculatorTest {
    private Calculator calculator;

    @BeforeAll
    static void beforeClass() {
        System.out.println("Global setup before all tests.");
    }

    @BeforeEach
    void setUp() {
        calculator = new Calculator();
        System.out.println("Calculator initialized before test.");
    }

    @Test
    void testAdd() {
        int result = calculator.add(5, 3);
        assertEquals(8, result);
    }

    @Test
    void testSubtract() {
        int result = calculator.subtract(9, 4);
        assertEquals(5, result);
    }

    @Test
    void testMultiply() {
        int result = calculator.multiply(7, 6);
        assertEquals(42, result);
    }

    @Test
    void testDivide() {
        double result = calculator.divide(20, 4);
        assertEquals(5.0, result, 0.001);
    }

    @Test
    void testDivideByZero() {
        assertThrows(ArithmeticException.class, () -> {
            calculator.divide(10, 0);
        }, "Division by zero should throw an exception.");
    }

    @AfterEach
    void tearDown() {
        System.out.println("Calculator reset after test.");
    }

    @AfterClass
    static void afterClass() {
        System.out.println("Clean-up after all tests.");
    }
}
```

Output:



```
Finished after 0.196 seconds
Runs: 5/5        Errors: 0        Failures: 0
CalculatorTest [Runner: JUnit 5] (0.038 s)
```

```
<terminated> CalculatorTest (1) [JUnit] C:\Users\prakr
Global setup before all tests.
Calculator initialized before test.
Calculator reset after test.
Calculator initialized before test.
Calculator reset after test.
Calculator initialized before test.
Calculator reset after test.
Calculator initialized before test.
Calculator reset after test.
Calculator initialized before test.
Calculator reset after test.
```

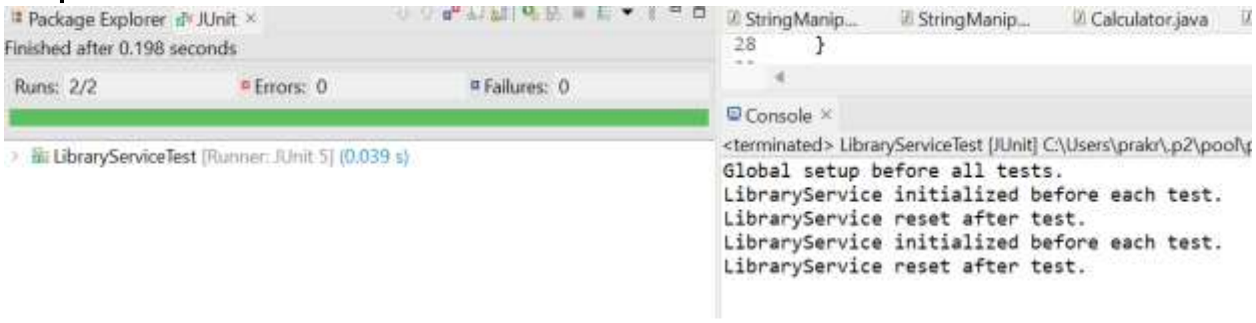
11. You are given a **LibraryService** class that manages books in a library. The **LibraryService** allows adding books to the library and searching for books by title. The class uses an internal **ArrayList** to store the books.
    Your task is to write unit tests for the **LibraryService** class. 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.

Code/Implementation:

```
package week4;

import java.util.ArrayList;
import java.util.List;

public class LibraryService {
    private List<String> books;
```

```java
    public LibraryService() {
        books = new ArrayList<>();
    }

    public void addBook(String title) {
        if (title != null && !title.trim().isEmpty()) {
            books.add(title);
        }
    }

    public boolean searchBook(String title) {
        return books.contains(title);
    }
}

package week4;

import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.Test;

class LibraryServiceTest {
    private LibraryService libraryService;

    @BeforeAll
    static void beforeClass() {
        System.out.println("Global setup before all tests.");
    }

    @BeforeEach
    void setUp() {
        libraryService = new LibraryService();
        System.out.println("LibraryService initialized before each test.");
    }

    @Test
    void testAddBook() {
        libraryService.addBook("Book 1");
        assertTrue(libraryService.searchBook("Book 1"));
    }

    @Test
    void testSearchBook() {
        libraryService.addBook("Book 2");
        assertTrue(libraryService.searchBook("Book 2"));
        assertFalse(libraryService.searchBook("Nonexistent Book"));
    }

    @AfterEach
    void tearDown() {
        System.out.println("LibraryService reset after test.");
    }

    @AfterClass
    static void afterClass() {
        System.out.println("Clean-up after all tests.");
    }
}
```

Output:



```
Package Explorer  JUnit
Finished after 0.198 seconds
Runs: 2/2        Errors: 0        Failures: 0
LibraryServiceTest [Runner: JUnit 5] (0.039 s)
```

```
Console
<terminated> LibraryServiceTest [JUnit] C:\Users\prakr\.p2\pool\
Global setup before all tests.
LibraryService initialized before each test.
LibraryService reset after test.
LibraryService initialized before each test.
LibraryService reset after test.
```

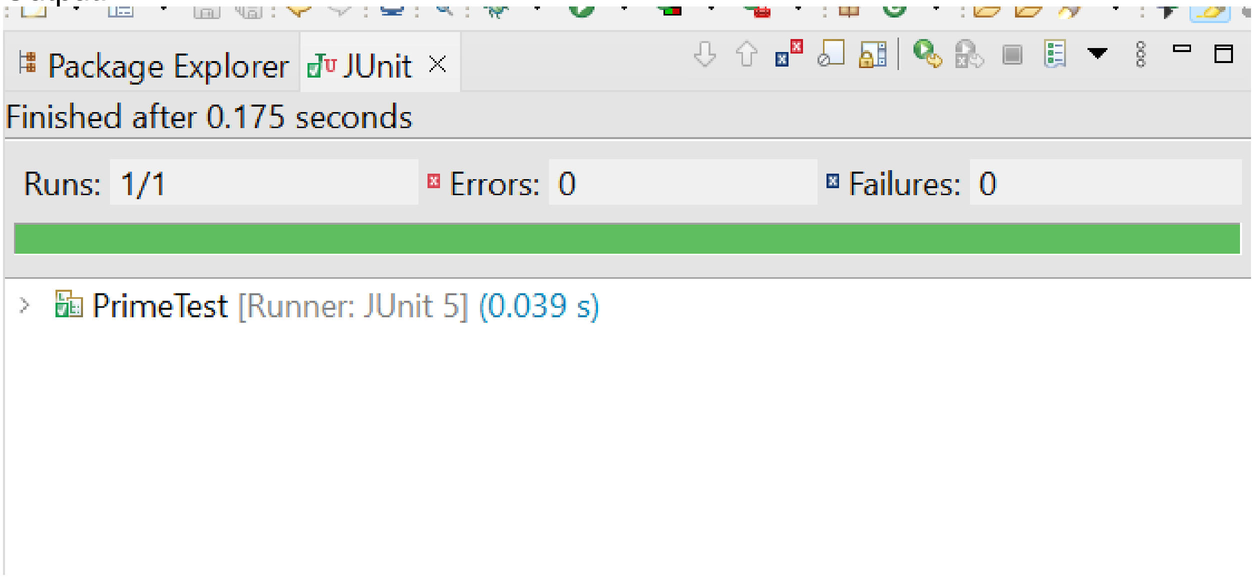
Follow the TDD Approach

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

Code/Implementation

```
package week4;


class Prime {

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

package week4;

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

import org.junit.jupiter.api.Test;

class PrimeTest {

    @Test
    void testPrime() {
        Prime prime = new Prime();
        assertTrue(prime.isPrime(2));
        assertFalse(prime.isPrime(1));
    }
}
```

Output:



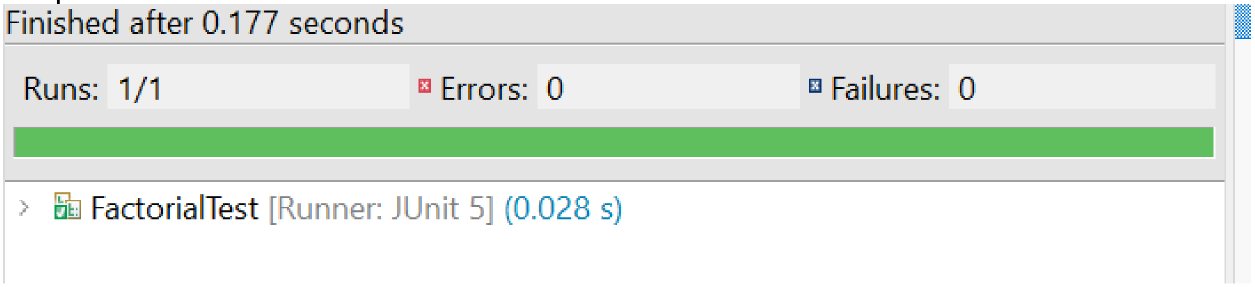
```
Finished after 0.175 seconds
Runs: 1/1    Errors: 0    Failures: 0
> PrimeTest [Runner: JUnit 5] (0.039 s)
```

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

Code/Implementation:

```
package week4;

public class Factorial {

      int calculateFactorial(int number) {
    if (number < 0) {
            throw new IllegalArgumentException("Number must be non-negative.");
        }
        int factorial = 1;
        for (int i = 1; i <= number; i++) {
            factorial *= i;
        }
        return factorial;
    }

}
package week4;

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

import org.junit.jupiter.api.Test;

class FactorialTest {
            @Test
    void testFactorial() {
        Factorial factorial = new Factorial();
        assertEquals(1, factorial.calculateFactorial(0));
        assertEquals(2, factorial.calculateFactorial(2));
}
}
```

Output:



```
Finished after 0.177 seconds
Runs: 1/1    Errors: 0    Failures: 0
> FactorialTest [Runner: JUnit 5] (0.028 s)
```

14. Create a class `Rectangle` with the following:

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

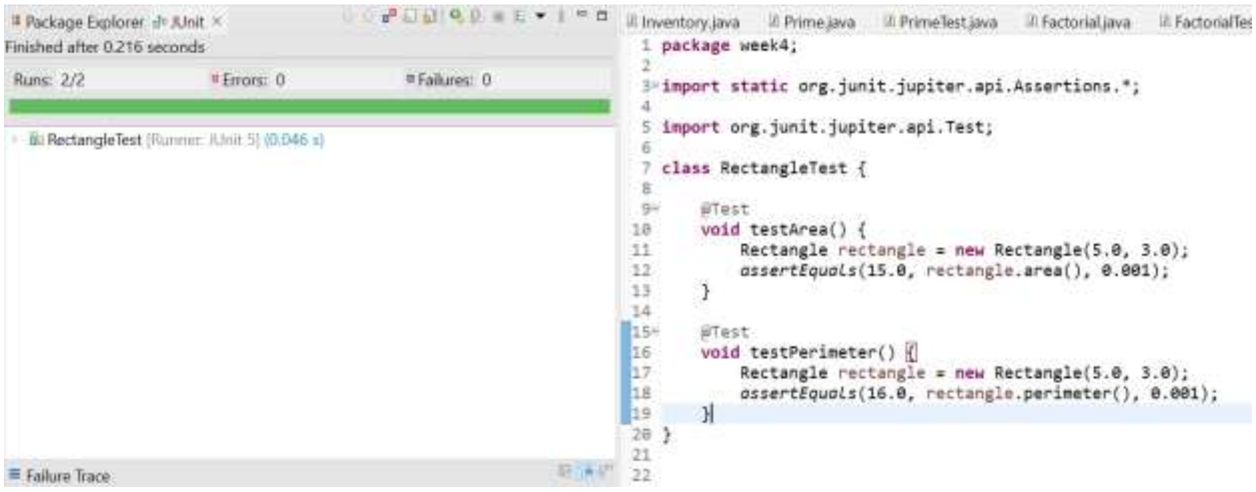
  `perimeter()` to calculate the perimeter of the rectangle.

- Create a test cases

Code/Implementation:

```
package week4;
```

```java
public class Rectangle {

    private double length;
    private double width;

    public Rectangle(double length, double width) {
        if (length <= 0 || width <= 0) {
            throw new IllegalArgumentException("Length and width must be positive.");
        }
        this.length = length;
        this.width = width;
    }

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

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

}
```



```
Package Explorer  JUnit
Finished after 0.216 seconds
Runs: 2/2        Errors: 0        Failures: 0
RectangleTest [Runner: JUnit 5] (0.046 s)
Failure Trace
```

```
Inventory.java  Prime.java  PrimeTest.java  Factorial.java  FactorialTes
package week4;

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

import org.junit.jupiter.api.Test;

class RectangleTest {

    @Test
    void testArea() {
        Rectangle rectangle = new Rectangle(5.0, 3.0);
        assertEquals(15.0, rectangle.area(), 0.001);
    }

    @Test
    void testPerimeter() {
        Rectangle rectangle = new Rectangle(5.0, 3.0);
        assertEquals(16.0, rectangle.perimeter(), 0.001);
    }
}
```

- 15.Create a base class Shape with a method area() that returns 0. Create two derived classes:

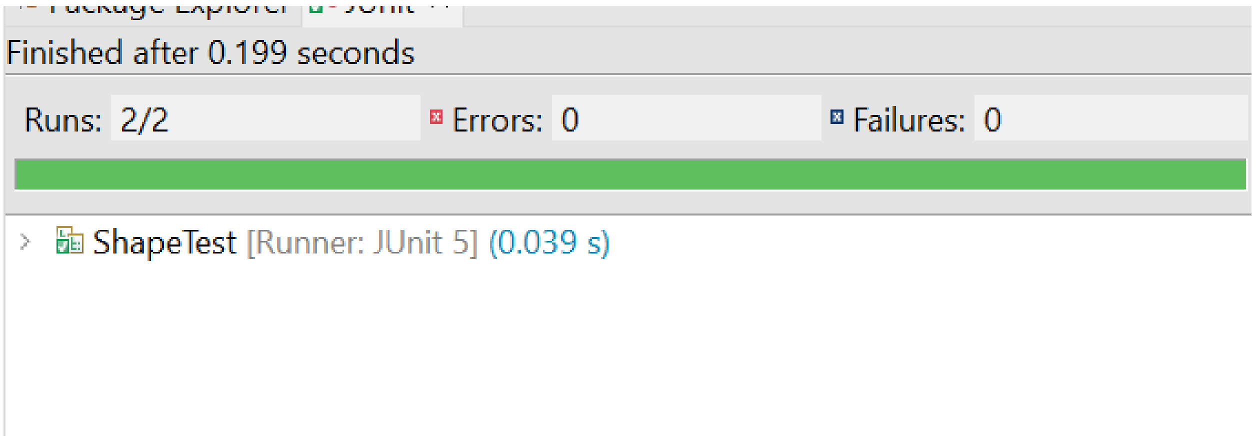
- Circle with attribute radius and area() method to calculate the area
- Rectangle with attributes length and width and area() method to calculate the area.

Code/Implementation:

```java
package week4;

public class Shape {
    public double area() {
        return 0.0;
    }
}

class Circle extends Shape {
    private double radius;

    public Circle(double radius) {
        if (radius <= 0) {
            throw new IllegalArgumentException("Radius must be positive.");
        }
        this.radius = radius;
    }

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

    public double getRadius() {
        return radius;
    }
}

class Rectangle extends Shape {
    private double length;
    private double width;

    public Rectangle(double length, double width) {
        if (length <= 0 || width <= 0) {
            throw new IllegalArgumentException("Length and width must be positive.");
        }
        this.length = length;
        this.width = width;
    }

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

```
package week4;

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

class ShapeTest {

    @Test
    void testCircleArea() {
        Circle circle = new Circle(3.0);
        assertEquals(Math.PI * 3.0 * 3.0, circle.area(), 0.001);
    }

    @Test
    void testRectangleArea() {
        Rectangle rectangle = new Rectangle(4.0, 5.0);
        assertEquals(20.0, rectangle.area(), 0.001);
    }
}
```

Output:



```
Finished after 0.199 seconds

Runs: 2/2    Errors: 0    Failures: 0

> ShapeTest [Runner: JUnit 5] (0.039 s)
```