LAB REPORT

JUnit Testing

Department of Computer Science & Engineering

Name : K.M. Sakif Muhtasim

Id : 2022-2-60-025

Course : CSE430

Section : 01

Date : 23.10. 2025

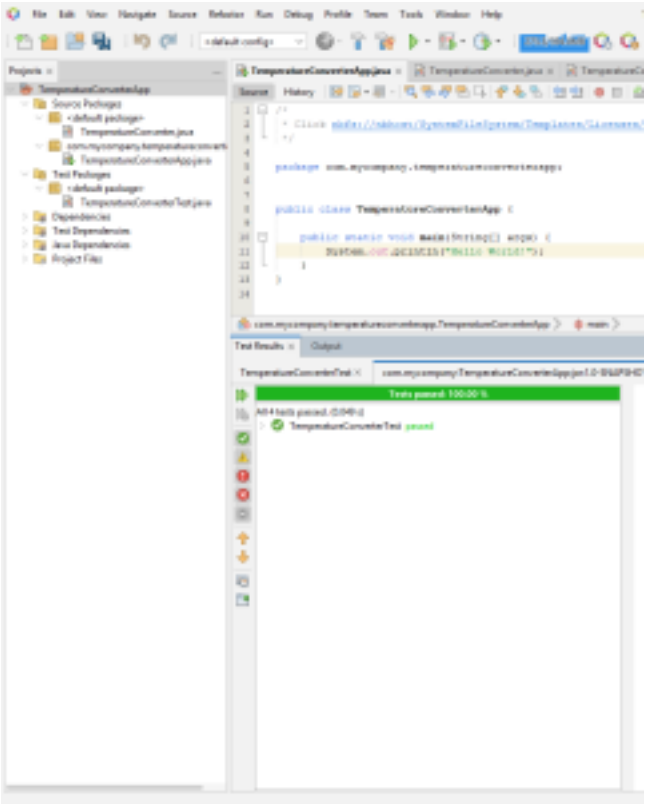
Task1

source code :

```
import org.junit.Test;
import static org.junit.Assert.*;

public class TemperatureConverterTest {

 TemperatureConverter converter = new TemperatureConverter();

 @Test
 public void testCelsiusToFahrenheit_KnownValues() {
assertEquals(32.0, converter.celsiusToFahrenheit(0), 0.01);
assertEquals(212.0, converter.celsiusToFahrenheit(100), 0.01);  }

 @Test
 public void testFahrenheitToCelsius_KnownValues() {
```

```
assertEquals(0.0, converter.fahrenheitToCelsius(32), 0.01);
assertEquals(100.0, converter.fahrenheitToCelsius(212), 0.01);  }

 @Test
 public void testCelsiusToKelvin_KnownValues() {
 assertEquals(273.15, converter.celsiusToKelvin(0), 0.01);
assertEquals(373.15, converter.celsiusToKelvin(100), 0.01);  }

 @Test
 public void testRoundTrip_CelsiusFahrenheit() {
 for (double c = -100; c <= 100; c += 10) {
 double f = converter.celsiusToFahrenheit(c);  double
cBack = converter.fahrenheitToCelsius(f);
assertEquals(c, cBack, 0.01);
 }
 }
}
```

Test Result



Task 2

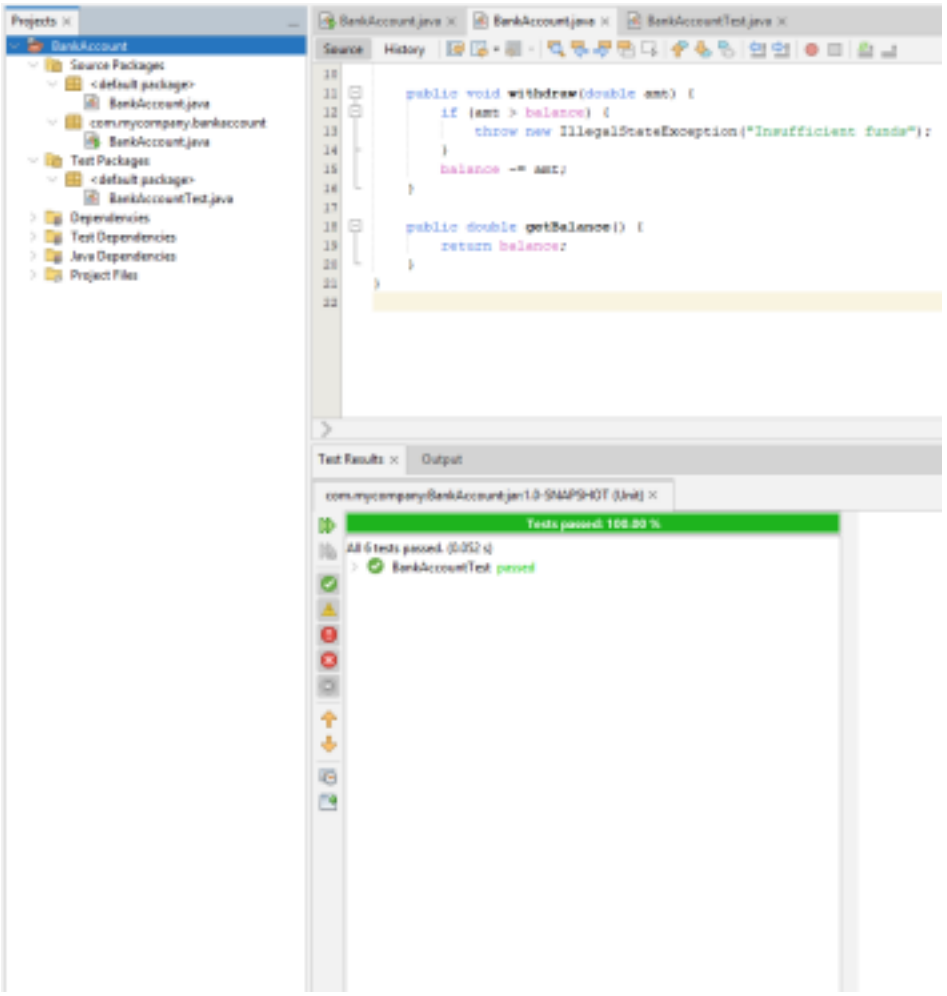
source code

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

import org.junit.jupiter.api.Test;


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


class BankAccountTest {


 private BankAccount account;


 @BeforeEach

 void setUp() {

 account = new BankAccount();

 }
```

```
@Test
void depositPositive_increasesBalance() {
account.deposit(200);
assertEquals(200, account.getBalance(), 0.0001);

account.deposit(50.5);
assertEquals(250.5, account.getBalance(), 0.0001);
}

@Test
void depositNegative_throwsIllegalArgumentException() {
assertThrows(IllegalArgumentException.class, () -> account.deposit(-10));  }

@Test
void withdrawValid_decreasesBalance() {
account.deposit(500);
account.withdraw(120);
assertEquals(380, account.getBalance(), 0.0001);
}

@Test
void withdrawBeyondBalance_throwsIllegalStateException() {
account.deposit(100);
assertThrows(IllegalStateException.class, () -> account.withdraw(150));  }

@Test
void withdrawExactBalance_resultsInZeroBalance() {
account.deposit(100);
account.withdraw(100);
```

```
assertEquals(0, account.getBalance(), 0.0001);
}

@Test
void balanceNeverGoesNegative_afterFailedWithdraw() {
account.deposit(100);
try {
account.withdraw(200);
} catch (IllegalStateException ignored) { }
assertEquals(100, account.getBalance(), 0.0001);
}
}
```

Test Result :

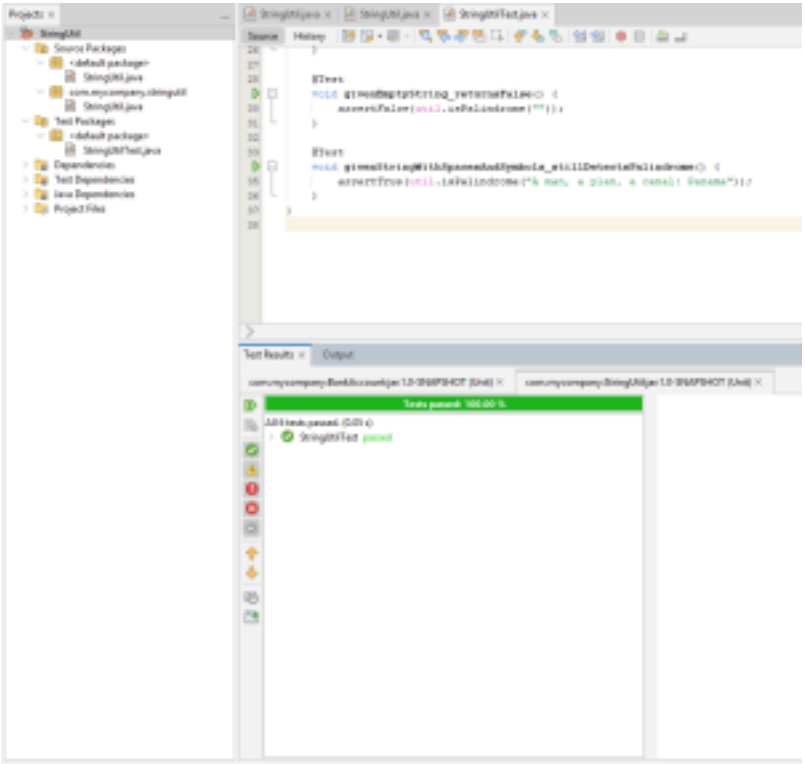


Task 3
source code

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

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


class StringUtilTest {


 private final StringUtil util = new StringUtil();


 @Test

 void givenMadam_returnsTrue() {
 assertTrue(util.isPalindrome("madam"));

 }


 @Test

 void givenRaceCar_caseInsensitive_returnsTrue() {

assertTrue(util.isPalindrome("RaceCar"));

 }


 @Test

 void givenHello_returnsFalse() {

 assertFalse(util.isPalindrome("hello"));

 }


 @Test

 void givenNull_returnsFalse() {

 assertFalse(util.isPalindrome(null));

 }


 @Test

 void givenEmptyString_returnsFalse() {
```

```
 assertFalse(util.isPalindrome(""));
 }

 @Test
  void  givenStringWithSpacesAndSymbols_stillDetectsPalindrome()  {
assertTrue(util.isPalindrome("A man, a plan, a canal: Panama"));  }
}
```

Test Result



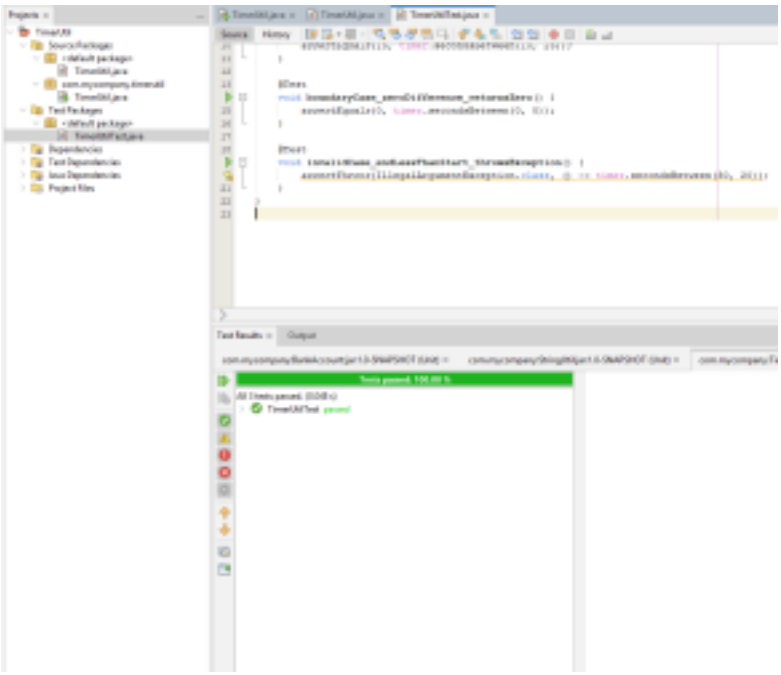
Task 4

Source code

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

class TimerUtilTest {

 private final TimerUtil timer = new TimerUtil();
```

```
@Test
void normalCase_returnsCorrectDifference() {
assertEquals(15, timer.secondsBetween(10, 25));  }

@Test
void boundaryCase_zeroDifference_returnsZero() {
assertEquals(0, timer.secondsBetween(0, 0));
}

@Test
void invalidCase_endLessThanStart_throwsException() {
assertThrows(IllegalArgumentException.class, () -> timer.secondsBetween(30, 20));  }
}
```

Test result



task 5

Source code

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

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

class ShoppingCartTest {
 private ShoppingCart cart;

 @BeforeEach
 void setUp() {
 cart = new ShoppingCart();
 }

 @Test
 void addItems_increasesCount() {
 cart.addItem("Apple");
 cart.addItem("Banana");
 cart.addItem("Orange");
 assertEquals(3, cart.getItemCount(), "Cart should have 3 items after adding");  }

 @Test
 void removeItem_decreasesCount() {
 cart.addItem("Apple");
 cart.addItem("Banana");
 cart.addItem("Orange");

 cart.removeItem("Banana");
 assertEquals(2, cart.getItemCount(), "Cart should have 2 items after removing one");  }

 @Test
 void clearCart_removesAllItems() {
 cart.addItem("Apple");
```

```
cart.addItem("Banana");

cart.clear();

assertEquals(0, cart.getItemCount(), "Cart should be empty after clearing");
}


@Test

void removeNonExistentItem_doesNotThrowError() {

cart.addItem("Apple");

cart.removeItem("Banana"); // not in cart

assertEquals(1, cart.getItemCount(), "Removing non-existent item should not affect count");  }

}
```

Test result