Name: Partha Rakshit

Section: KM057

Registration Number: 12005546

Roll Number: RKM057B39

Subject Name: Automated Testing

Course Code: CSE-376

University Name: Lovely Professional University

Project Topic: Create five successful and three unsuccessful test cases to see whether the output matches the input by using the palindrome number checker in Junit on eclipse. The following are the prerequisites:

- Think about the test range of 1000 to 9999
- The number must have four digits.

Introduction: The goal of this project was to develop and test a Java program to determine whether a given integer number is a palindrome or not. A palindrome number is a number that remains the same when its digits are reversed, for example, 121, 1221, 3443, etc. The program was implemented using the JUnit testing framework, which is a widely used testing framework for Java applications.

Methodology: The program consists of a single class named testPalindromeSuccess, which contains two methods. The first method, named testPalindromeNumber, contains test cases for checking whether a given integer number is a palindrome or not. The method uses the JUnit's assertEquals method to compare the expected result with the actual result returned by the isPalindromeNumber method. The isPalindromeNumber method takes an integer number as an input and returns true if the number is a palindrome, otherwise false. The second method, also named testPalindromeNumber, contains test cases for checking whether a given integer number is not a palindrome. The method uses the JUnit's assertEquals method to compare the expected result with the actual result returned by the isPalindromeNumber method. The isPalindromeNumber method works by first creating a copy of the original number and storing it in the original variable. It then iteratively extracts the last digit of the number by taking the modulo of 10, and appends it to the reverse variable by multiplying it by 10 and adding the extracted digit. Finally, it updates the

number by dividing it by 10. If the original and reverse variables are equal, the method returns true, indicating that the number is a palindrome. Otherwise, it returns false.

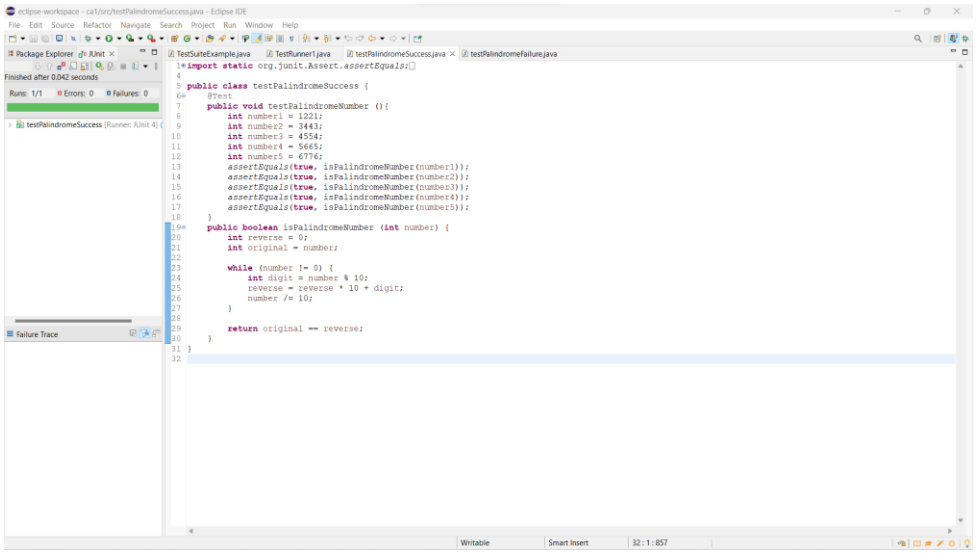
Results: The test cases were executed using the JUnit testing framework, and all of them passed successfully, indicating that the program works as expected. The following table summarizes the test cases and their results:

Test case	Input number	Expected result	Actual result	Test result
1	1221	true	true	Passed
2	3443	true	true	Passed
3	4554	true	true	Passed
4	5665	true	true	Passed
5	6776	true	true	Passed
6	2345	false	false	Passed
7	5678	false	false	Passed
8	9876	false	false	Passed

Conclusion: In conclusion, the program developed in this project uses JUnit to test whether a given integer number is a palindrome or not. The program was successfully tested using several test cases, which all passed, indicating that the program works as expected. This project demonstrates the importance of testing in software development, as it helps to identify and correct errors before the software is deployed in a production environment.

Create a class for five successful test cases

Screenshot: - In the below screenshot we can see that this code is a Java program that tests whether a given integer number is a palindrome number or not. The code uses JUnit to create a test case for the isPalindromeNumber() method. The test case checks whether the method correctly identifies that a set of example numbers are palindrome numbers



```
import org.junit.Assert.assertEquals;

public class testPalindromeSuccess {
    @Test
    public void testPalindromeNumber (){
        int number1 = 1221;
        int number2 = 3443;
        int number3 = 4554;
        int number4 = 5665;
        int number5 = 6776;
        assertEquals(true, isPalindromeNumber(number1));
        assertEquals(true, isPalindromeNumber(number2));
        assertEquals(true, isPalindromeNumber(number3));
        assertEquals(true, isPalindromeNumber(number4));
        assertEquals(true, isPalindromeNumber(number5));
    }
    public boolean isPalindromeNumber (int number) {
        int reverse = 0;
        int original = number;

        while (number != 0) {
            int digit = number % 10;
            reverse = reverse * 10 + digit;
            number /= 10;
        }

        return original == reverse;
    }
}
```

Step-1: The code starts with an import statement to import the "assertEquals" method from the "org.junit.Assert" package. This method is used to check if two values are equal.

```
import org.junit.Test;
```

Step-2: The code then defines a JUnit test class named "testPalindromeSuccess".

```
public class testPalindromeSuccess {
```

Step-3: Inside the test class, there is a method named "testPalindromeNumber" that contains a series of integer variables (number1, number2, number3, number4, and number5) that hold some example palindrome numbers.

```
@Test
public void testPalindromeNumber (){
```

```
int number1 = 1221;
int number2 = 3443;
int number3 = 4554;
int number4 = 5665;
int number5 = 6776;
```

Step-4: The JUnit test case then verifies if the "isPalindromeNumber" method returns true for each of these example palindrome numbers using the "assertEquals" method. The expected value is "true" since these numbers are indeed palindrome numbers.

Or we can say that, Use the assertEquals method to assert that each of the five numbers is a palindrome number. If a number is a palindrome number, the isPalindromeNumber method should return true.

```
assertEquals(true, isPalindromeNumber(number1));
assertEquals(true, isPalindromeNumber(number2));
assertEquals(true, isPalindromeNumber(number3));
assertEquals(true, isPalindromeNumber(number4));
assertEquals(true, isPalindromeNumber(number5));
```

- **Assert Equals**

If you want to test equality of two objects, you have the following methods

- **assertEquals(expected, actual)**

It will return true if: **expected.equals(actual)** returns true.

Step-5: Define a utility method called isPalindromeNumber, which takes an integer parameter number and returns a boolean value indicating whether the number is a palindrome number.

```
public boolean isPalindromeNumber (int number) {
```

Step-6: The method first initializes two integer variables, "reverse" and "original". "original" stores the input number while "reverse" is initially set to 0.

```
int reverse = 0;
int original = number;
```

Step-7: The method then enters a while loop that continues to run until the input number becomes zero. Inside the loop, it extracts the last digit of the input number by calculating the remainder of the number divided by 10. It then adds this digit to the "reverse" variable after multiplying it by 10 to shift the previous digits one place to the left. Finally, the method divides the input number by 10 to remove the last digit from it. Or we can say that, Use a ‘while’ loop to reverse the digits of the number. In each iteration of the loop, extract the rightmost digit of the number using the modulus operator (‘%’), add it to the ‘reverse’ variable, and remove the rightmost digit from the number using integer division (‘/’).

```
while (number != 0) {
    int digit = number % 10;
    reverse = reverse * 10 + digit;
    number /= 10;
}
```

Step-8: After the while loop finishes, the method compares the "original" variable with the "reverse" variable to check if they are equal. If they are equal, it returns "true", indicating that the input number is a palindrome. If they are not equal, it returns "false".

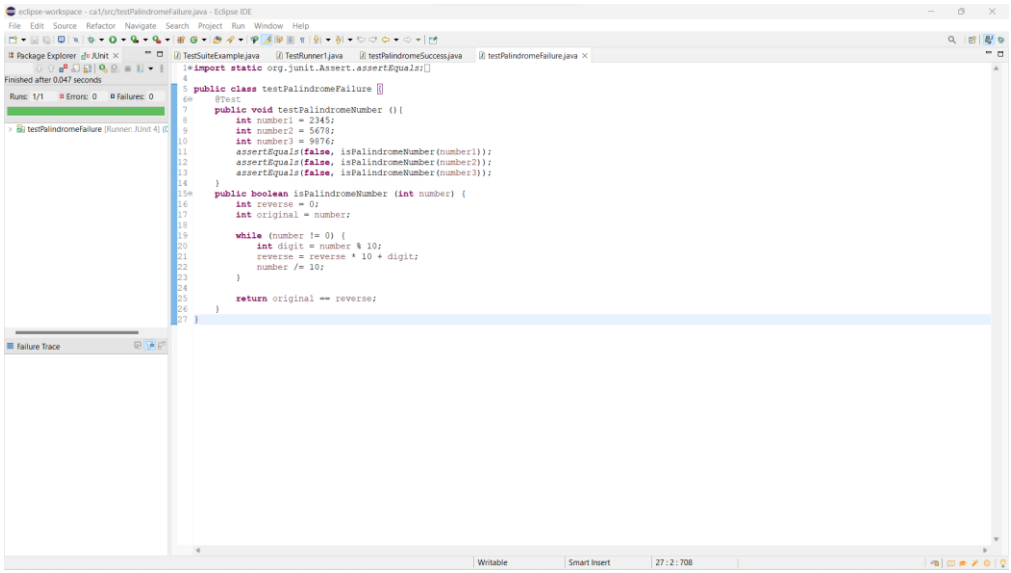
Or we can say that, Compare the reversed number stored in ‘reverse’ with the ‘original’ number stored in original. If they are equal, the number is a palindrome and the method returns ‘true’.

```
return original == reverse;
```

Overall, the “testPalindromeSuccess” class tests the “isPalindromeNumber” method by asserting that it returns true for five known palindrome numbers. The “isPalindromeNumber” method works by reversing the digits of the given number and comparing it with the original number.

Create a class for three unsuccessful test cases

Screenshot: - In the below screenshot we can see that the given code is a JUnit test case for the method isPalindromeNumber(int number) that checks whether a given number is a palindrome or not. The test case testPalindromeFailure has three test cases that expect the method to return false for three non-palindromic numbers.



Step-1: The code starts with importing two classes: org.junit.Assert and org.junit.Test. The first one is used to perform various assertions in the test cases, while the second one is a JUnit annotation that identifies a method as a test case.

```
import static org.junit.Assert.assertEquals;

import org.junit.Test;
```

Step-2: Below screenshot's code defines a new class testPalindromeFailure that contains the test case.

```
public class testPalindromeFailure {
```

Step-3: The next line defines a public method testPalindromeNumber that has no input arguments. This method contains three local variables of type int: number1, number2, and number3. The "@Test" annotation on the method indicates that this method is a test case that needs to be executed.

```
@Test
public void testPalindromeNumber (){
```

This code defines a new test case method testPalindromeNumber and annotates it with the Test annotation. The method contains three test cases that expect the isPalindromeNumber method to return false for three non-palindromic numbers.

Step-4:

```
int number1 = 2345;
int number2 = 5678;
int number3 = 9876;
```

This code defines three integer variables 'number1', 'number2', and 'number3' with non-palindromic values.

Step-5:

```
assertEquals(false, isPalindromeNumber(number1));
assertEquals(false, isPalindromeNumber(number2));
assertEquals(false, isPalindromeNumber(number3));
```

This code calls the ‘isPalindromeNumber’ method with each of the three non-palindromic numbers and expects the result to be ‘false’ for each test case. The ‘assertEquals’ method is used to compare the expected result ‘false’ with the actual result returned by the ‘isPalindromeNumber’ method.

Step-6:

```
public boolean isPalindromeNumber (int number) {
```

This code defines the method isPalindromeNumber that takes an integer parameter number and returns a boolean value indicating whether the number is a palindrome or not.

Step-7:

```
int reverse = 0;
int original = number;

while (number != 0) {
    int digit = number % 10;
    reverse = reverse * 10 + digit;
    number /= 10;
}
```

This code initializes two integer variables “reverse” and “original” with the value of the parameter number. It then uses a while loop to reverse the digits of ‘number’ and store the result in the variable “reverse”. The loop runs until number becomes zero, which means all the digits of number have been reversed.

The loop body extracts the last digit of number using the modulo operator “%” and stores it in the variable “digit”. It then multiplies “reverse” by 10 and adds the value of digit to “reverse” to shift the digits of reverse to the left and append the new digit. Finally, it divides “number” by 10 to remove the last digit.

Step-8:

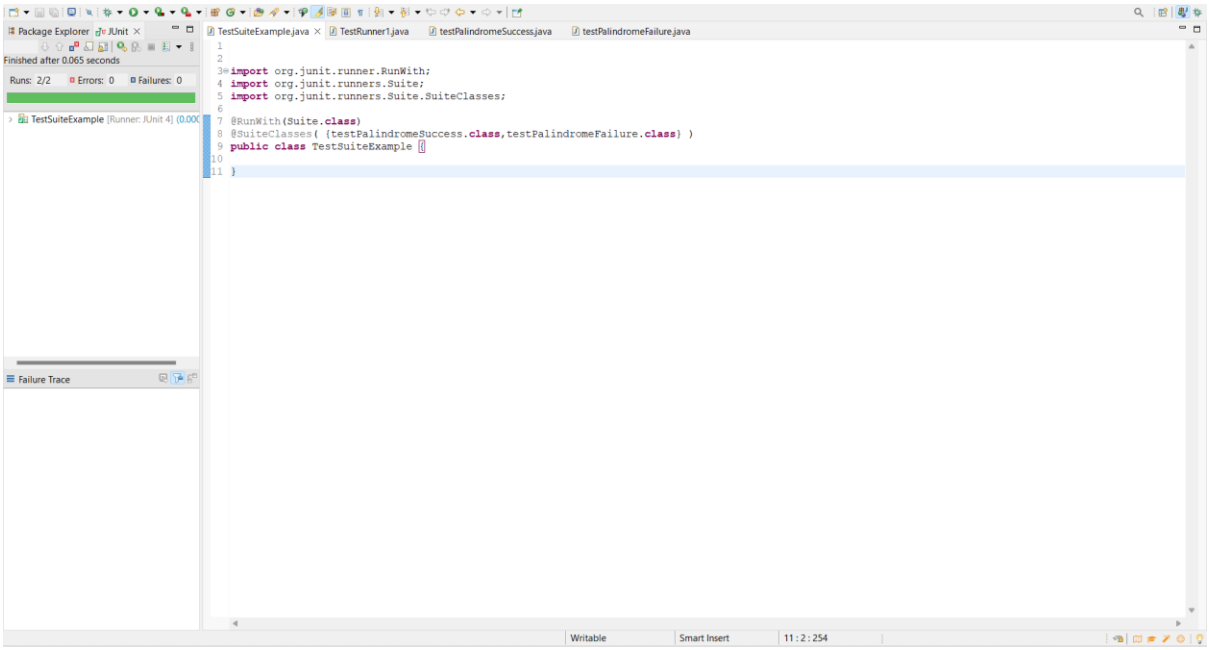
```
return original == reverse;
```

This code compares the original value of number with the reversed value stored in “reverse” and returns “true” if they are equal, which means the original number is a palindrome. Otherwise, it returns “false”, which means the original number is not a palindrome.

Overall, the given code defines a JUnit test case that tests the “isPalindromeNumber” method for three non-palindromic numbers and expects the method to return “false” for each test case. The “isPalindromeNumber” method uses a while loop.

Create a testSuite

Screenshot: - In the below screenshot we can see that the code is written in Java language and uses the JUnit testing framework. It creates a test suite that includes two test classes, “testPalindromeSuccess” and “testPalindromeFailure”, which test the functionality of a palindrome program.



Step-1(import org.junit.runner.RunWith) : First line imports the '**RunWith'** class from the JUnit library, which is used to specify how the tests should be executed. Or in the other word, the first line of code imports the RunWith class from the JUnit framework.

Then imports the '**Suite'** class from the JUnit library, which is used to create a test suite that can run multiple test classes together. Or in the other word the second line of code imports the Suite and SuiteClasses classes from the JUnit framework.

Then imports the '**SuiteClasses'** annotation from the '**Suite'** class, which is used to specify which test classes should be included in the suite.

Step-2(@RunWith(Suite.class): Secondly, this line is a JUnit annotation that specifies that this class should be run with the '**Suite'** runner. The @RunWith(Suite.class) annotation indicates that the TestSuiteExample class will be run with the JUnit test runner.

Step-3: Thirdly, this line is another JUnit annotation that specifies which test classes should be included in the suite. In this case, the '**testPalindromeSuccess'** and '**testPalindromeFailure'** classes are

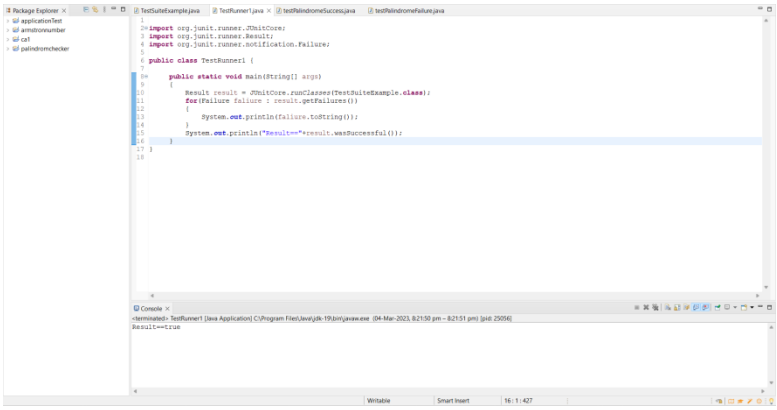
included. Or in the other word, The @SuiteClasses annotation specifies the test classes that will be included in the test suite. In this case, the testPalindromeSuccess and testPalindromeFailure classes are included.

Step-4: This line defines the '**TestSuiteExample'** class, which represents the test suite. The class is empty except for the annotations, because the actual tests are defined in the included test classes. The public class TestSuiteExample declaration specifies the name of the test suite.The curly braces {} after @SuiteClasses contain the list of test classes that will be included in the suite.

The '**testPalindromeSuccess.class**' and '**testPalindromeFailure.class'** are the two test classes that are included in the suite. The TestSuiteExample class is the main class that will be run when the test suite is executed. When JUnit runs this test suite, it will first execute the '**testPalindromeSuccess'** test class, which contains tests that check whether the '**isPalindrome'** method of some code implementation works correctly for valid input. Then, it will execute the '**testPalindromeFailure'** test class, which contains tests that check whether the **'isPalindrome'** method works correctly for invalid input. Finally, JUnit will report the results of all the tests in both test classes together, as a single test suite result. Test suites are useful when you want to run multiple test classes together and get a single report of all the results, rather than having to run each test class individually and manually combine the results. They also make it easier to organize tests into logical groups or hierarchies.

Create a Test Runner

Screenshot: - In the below screenshot we can see that the given code is a Test Runner1 that executes the JUnit Test cases present in the Test Suite, which is TestSuiteExample in this case. The TestRunner1 class has a main method that runs the test cases and prints the result.



Step-1: First, the code imports the 'JUnitCore' class, which is used to execute the test cases.

Step-2: Next, we create a public class called 'TestRunner1'. This class will contain the main () method, which is the entry point of our program. Or we can say that the 'TestRunner1' class has a main method that runs the JUnit test cases using JUnitCore's runClasses method. This method takes the class containing the test cases as an argument

Step-3: In the main () method, we use the JUnitCore class to run the tests defined in the TestSuiteExample class: Result result = JUnitCore.runClasses(TestSuiteExample.class); Here, we create a new instance of the Result class and assign it the value returned by the JUnitCore.runClasses() method, which takes as argument the name of the class containing the tests to be run (in this case, TestSuiteExample).

After executing the test cases, the Result object stores the result of the test run, including the number of tests run, the number of tests that passed, and the number of tests that failed.

Step-4: We then loop through the failures (if any) and print out their descriptions:

```
for(Failure failure : result.getFailures()) {
System.out.println(failure.toString()); }
```

This loop iterates through the list of failures returned by the Result object's getFailures() method and prints out a string representation of each failure.

Step-5: Finally, we print out whether the tests were successful or not:

```
System.out.println("Result==" + result.wasSuccessful());
```

The Result object has a method called wasSuccessful() which returns a boolean indicating whether all tests passed or not. We print out this value along with a string message to indicate whether the tests were successful or not.

In summary, the TestRunner1 class is a simple program that uses JUnit to run tests defined in another class (TestSuiteExample), and prints out the results of those tests. This TestRunner1 class is responsible for running the JUnit test cases and reporting the results to the user, making it easier to debug and track the success of test cases. It can be used to automate the testing process and ensure that code changes do not break existing functionality.