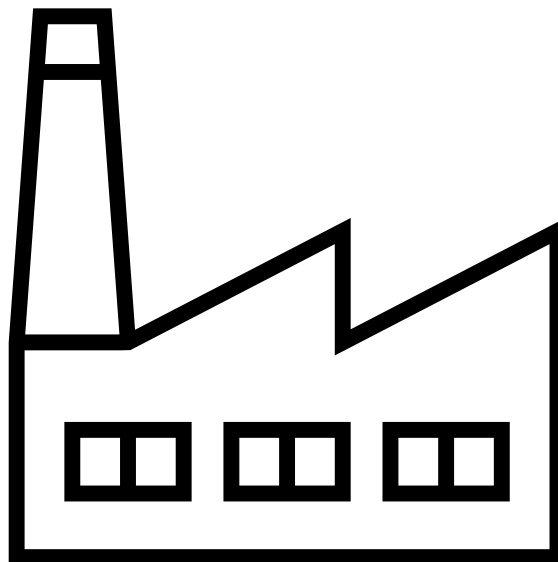


Summative Assessment 4

18-23 July 2024

Jamie Myburgh



Testing
Question 4

Contents

a) Assessing the Need for a Testing Program	3
Objectives	3
Maximum Values	3
b) Developing Test Strategy	4
Cycle 1	4
Cycle 2	5
c) Record of Test Results	8
Cycle 1: New Inputted Data	8
Hours	8
Shift	9
Retirement	10
Cycle 2: Existing Data	11
Regular Salary	11
Overtime Salary	11
Total Payroll	11
Not-Equals	12
Retirement with Overtime	13
Retirement without Overtime	13
Net Income for Basic Salary	14

a) Assessing the Need for a Testing Program

Considering that the program is in a very volatile development position, it was deduced that there is a need for testing. Testing is an extremely imperative part of this software development process as it will identify issues and defects within the written code to fix them before the software product is delivered.

Objectives

There are a multitude of objectives that should be covered by these test cases relating to input and calculation accuracy.

When users are to input their hours, the program should not accept any other data type except for that of an integer. When users are to input their hours, the program should not accept any other value or number except '1', '2', or '3'.

If the user inputs shift '2' or '3', they will be prompted to answer the question of whether they would like to apply for a retirement fund. The program should not accept any other answer except for 'yes' or 'no'.

Once all errors related to user input conditions are properly absolved, the calculations for these programs should also be tested accordingly.

The calculations that need to be tested include the regular payroll, the payroll for overtime hours, the total of the regular and overtime payroll, the deduction for any retirement funds as well as the net income.

Maximum Values

The maximum number of users that can make use of the program at a time is one person per runtime. The program as a whole, however, can accept an unlimited number of users.

According to the South African Labour Guide, an employee is under no obligation to work more than 45 hours per week. This particular company, UrbanFurn, however, considers any hours exceeding that of 40 to be overtime.

According to the Labour Guide, the maximum permissible overtime is 3 hours on any one day or 10 hours on any one week. This means that the maximum permissible overtime should be 55 hours, however, for this particular company, the maximum permissible overtime is 70 hours.

b) Developing Test Strategy

A test strategy has been created that is tailored to the program's specific needs. Two testing cycles were developed to test the program's functions with fresh data as well as predetermined data.

Cycle 1

The first test cycle tests new or 'fresh' data that is inputted into the system at each run time. This includes the hours-variable, shift-variable, as well as user input-variable to apply for a retirement fund.

To test the functionality of the hours-variable, incorrect data will have to be inputted. This variable was originally declared as an integer, therefore it can only store information that is an integer. To observe the program's behaviour, a double value, and a String value will be inputted into the 'hours' category.

To test whether the 'shift' values are behaving correctly, incorrect data types were introduced to the program. The shift-variable is an integer; therefore, it should only accept integer values. However, more importantly, it should also be noted that the shift-variable only accepts very specific integers, '1', '2', or '3'.

To test how this variable behaves under the incorrect inputs, other integers were inputted as well as different data types.

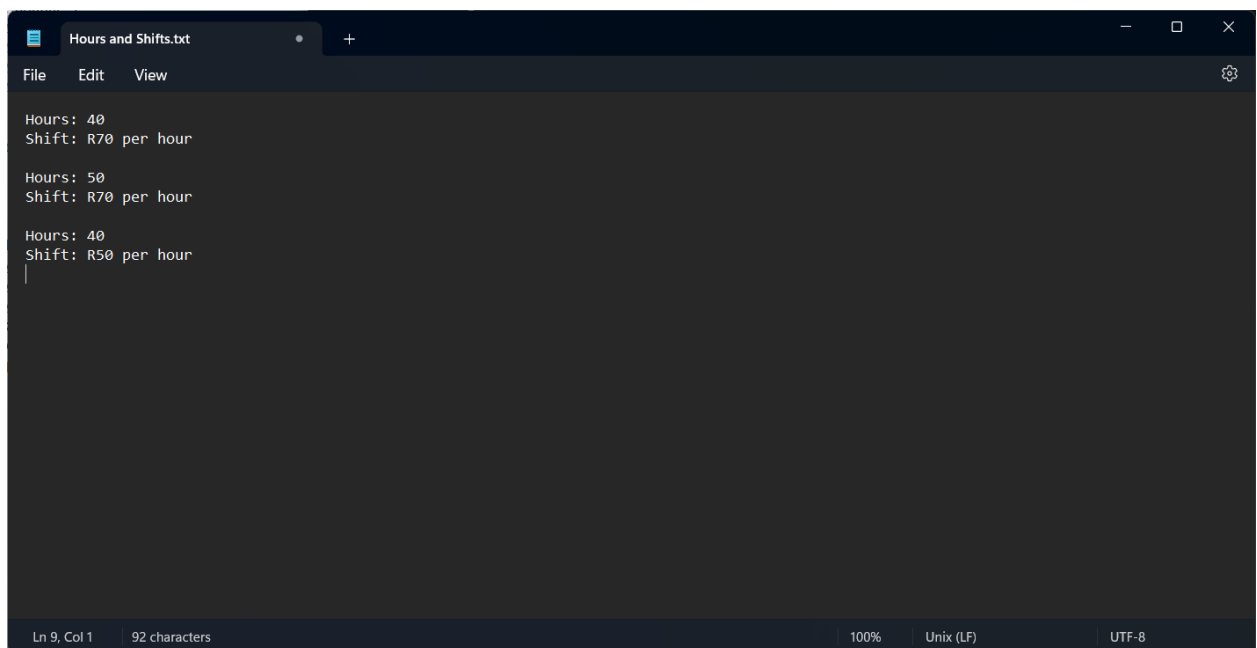
The input for the retirement-variable is a String. It is a simple 'yes or no' answer, meaning that if the user enters 'yes', the deduction will go through, and if they enter 'no', the program will automatically assume that they have rejected the offer.

Cycle 2

The second cycle tests existing data, to ensure that the program can handle data that has been previously processed or is already stored in the system. This cycle tests the calculations present in the 'calculations' class and the 'retirement' class.

To commence these tests, JUnit tests have to be developed, in which two test classes were created in the test package of the project. This was done to test the methods that are in these classes.

The text file to which hours and shifts were stored was opened for observation, and the values within this file were used to test the calculations.



```
Hours: 40  
Shift: R70 per hour  
  
Hours: 50  
Shift: R70 per hour  
  
Hours: 40  
Shift: R50 per hour
```

The screenshot shows a text editor window with the title 'Hours and Shifts.txt'. The editor has a menu bar with 'File', 'Edit', and 'View'. The content of the file is as follows:

```
Hours: 40  
Shift: R70 per hour  
  
Hours: 50  
Shift: R70 per hour  
  
Hours: 40  
Shift: R50 per hour
```

The status bar at the bottom indicates 'Ln 9, Col 1', '92 characters', '100%', 'Unix (LF)', and 'UTF-8'.

```

11 public void perfCalc() {
12
13     //IF HOURS ARE GREATER THAN 40
14     if (hours > 40) {
15
16         total = 40 * shift;
17         overtime = 1.5 * (hours - 40) * shift;
18         payroll = total + overtime;
19
20     //IF HOURS ARE LESS THAN 40
21     } else {
22
23         payroll = hours * shift;
24     }
25 }
26
27 }

```

Above is the method to calculate the values for the regular payroll and the payroll for an employee who is working overtime.

Should an employee input a value that is greater than 40 hours, this indicates that the employee is working overtime. Their regular salary is calculated and stored in the variable 'total'.

Their overtime pay is calculated by taking the hours exceeding 40 and multiplying it by the shift's pay rate, produced by one and one-half.

Their payroll will then be the sum of the total and their overtime pay. The method does not multiply the user's collective hours by the shift rate. Meaning, should the user work 70 hours, the overtime pay is only applied to the hours exceeding 40 (overtime by 30).

```

10  public void perfRetire() {
11
12
13  if (input.equalsIgnoreCase("yes")) {
14      perfCalc();
15      retirement = payroll * 0.05;
16      net = payroll - retirement;
17      System.out.println("Retirement deduction: R" + retirement);
18      System.out.println("Net: R" + net);
19  } else {
20      perfCalc();
21      net = payroll;
22      System.out.println("Net: R" + net);
23  }
24  }
25
26  }
27
28  public void regularNet() {
29      perfCalc();
30      net = payroll;
31      System.out.println("Net: R" + net);
32  }
33
34  }

```

Above is a method for the retirement and regular net income calculation. Should an employee input the values '2' or '3', they will be given the option to apply for a retirement fund.

Should the user input 'yes', their retirement will be calculated. Their payroll is multiplied by 5%, and the value is set to the variable 'retirement'. Their net income is then calculated, by subtracting the retirement from their payroll. The net income is then printed out into the console.

Otherwise, their net income is set to whatever value their payroll is, and it is printed out accordingly.

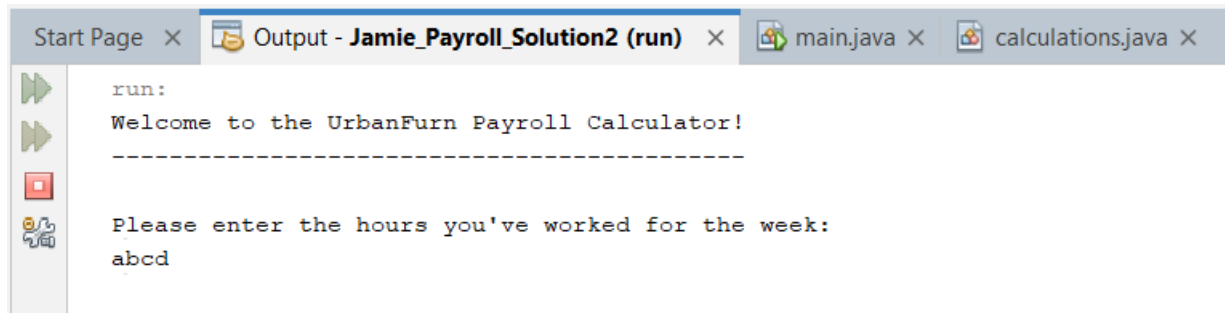
Below is a method to calculate the regular net income, for employees who are not working the second or third shift. It sets the net value to whatever value the payroll is set to and prints it out accordingly.

c) Record of Test Results

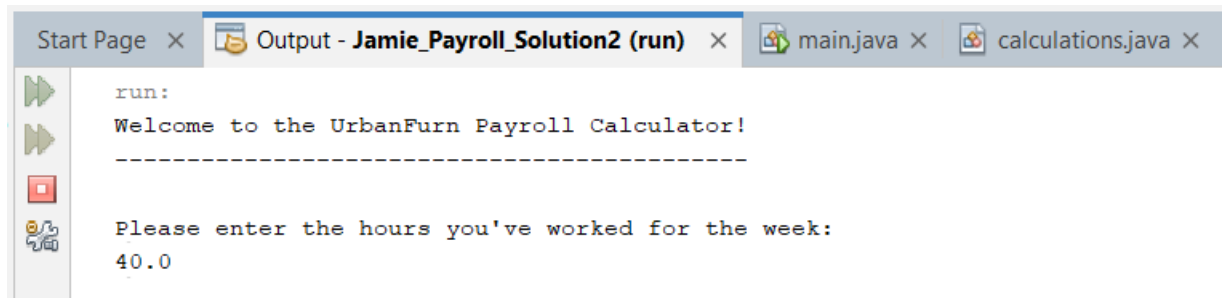
Cycle 1: New Inputted Data

Hours

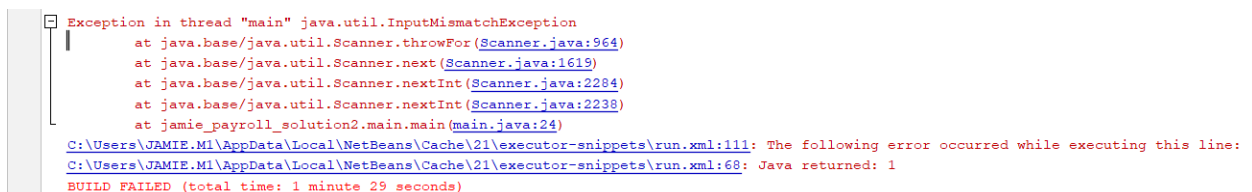
A String value and double values were introduced to the system to observe how it behaves. This resulted in two type-mismatch errors.



```
Start Page × Output - Jamie_Payroll_Solution2 (run) × main.java × calculations.java ×  
run:  
Welcome to the UrbanFurn Payroll Calculator!  
-----  
Please enter the hours you've worked for the week:  
abcd
```



```
Start Page × Output - Jamie_Payroll_Solution2 (run) × main.java × calculations.java ×  
run:  
Welcome to the UrbanFurn Payroll Calculator!  
-----  
Please enter the hours you've worked for the week:  
40.0
```



```
Exception in thread "main" java.util.InputMismatchException  
    at java.base/java.util.Scanner.throwFor(Scanner.java:964)  
    at java.base/java.util.Scanner.next(Scanner.java:1619)  
    at java.base/java.util.Scanner.nextInt(Scanner.java:2284)  
    at java.base/java.util.Scanner.nextInt(Scanner.java:2238)  
    at jamie_payroll_solution2.main.main(main.java:24)  
C:\Users\JAMIE.M1\AppData\Local\NetBeans\Cache\21\executor-snippets\run.xml:111: The following error occurred while executing this line:  
C:\Users\JAMIE.M1\AppData\Local\NetBeans\Cache\21\executor-snippets\run.xml:68: Java returned: 1  
BUILD FAILED (total time: 1 minute 29 seconds)
```


Shift

The shift variable was originally declared as an integer, and functions were given to the variable depending on the input. For the program to work effectively, the input should be either '1', '2', or '3'. To observe what the program would do if the variables were not these specific integers, the number '4' was inputted. This resulted in a build success. This should not occur.

```
Please enter your shift number: (1, 2, or 3)
Shift 1: R50 per hour
Shift 2: R70 per hour
Shift 3: R90 per hour
4
```

```
Please enter your shift number: (1, 2, or 3)
Shift 1: R50 per hour
Shift 2: R70 per hour
Shift 3: R90 per hour
4

BUILD SUCCESSFUL (total time: 1 minute 7 seconds)
||
```

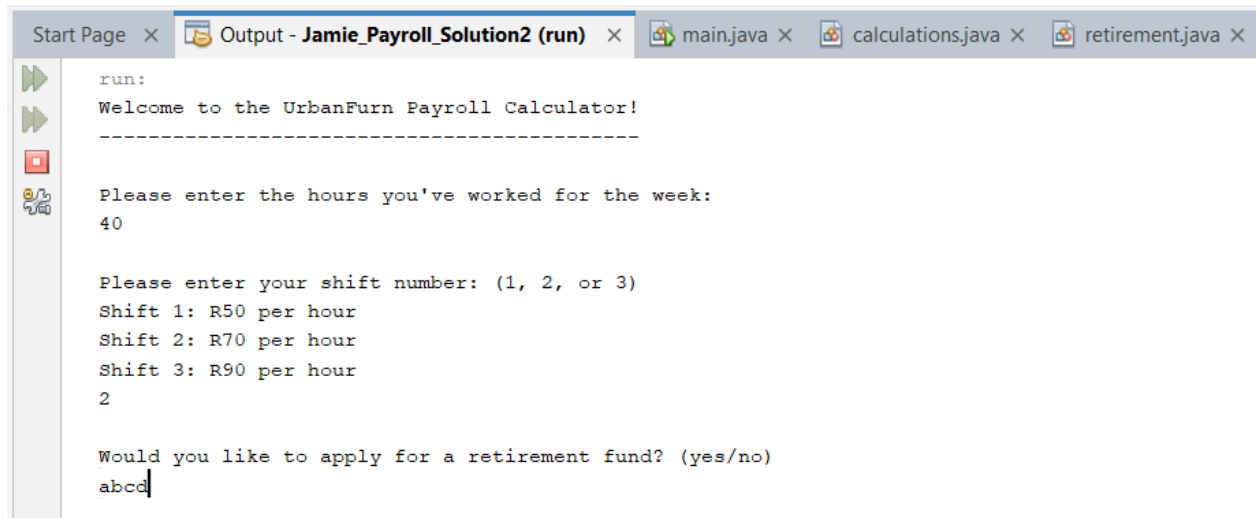
A String variable was then inputted to observe it's functionality. This resulted in a type-mismatch error.

```
Please enter your shift number: (1, 2, or 3)
Shift 1: R50 per hour
Shift 2: R70 per hour
Shift 3: R90 per hour
abcd
```

```
Exception in thread "main" java.util.InputMismatchException
    at java.base/java.util.Scanner.throwFor(Scanner.java:964)
    at java.base/java.util.Scanner.next(Scanner.java:1619)
    at java.base/java.util.Scanner.nextInt(Scanner.java:2284)
    at java.base/java.util.Scanner.nextInt(Scanner.java:2238)
    at jamie_payroll_solution2.main.main(main.java:24)
C:\Users\JAMIE.M1\AppData\Local\NetBeans\Cache\21\executor-snippets\run.xml:111: The following error occurred while executing this line:
C:\Users\JAMIE.M1\AppData\Local\NetBeans\Cache\21\executor-snippets\run.xml:68: Java returned: 1
BUILD FAILED (total time: 1 minute 29 seconds)
```

Retirement

Should a user select the 2nd or 3rd shift, they will be given the option to apply for a retirement fund. Because this is a 'yes or no', they should only be able to enter one of these two values. The program should not accept any other values. To test this, 'abcd' was inputted. The result was that it automatically assumed that the user did not apply for the retirement fund, and therefore the program continued as normal. This should not occur.

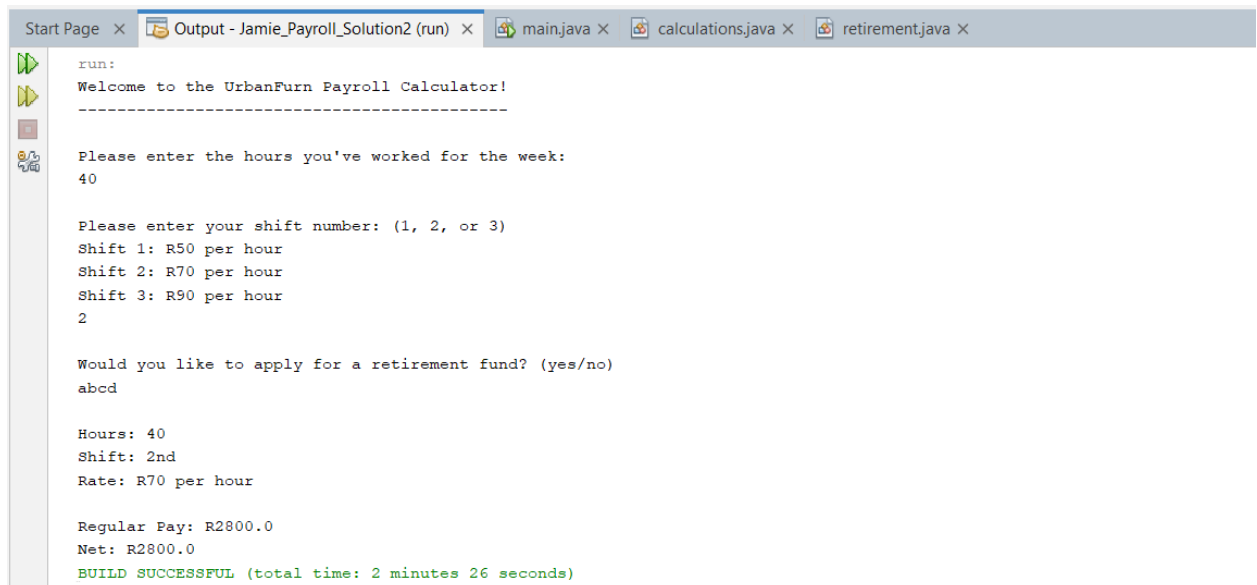


```
run:
Welcome to the UrbanFurn Payroll Calculator!
-----

Please enter the hours you've worked for the week:
40

Please enter your shift number: (1, 2, or 3)
Shift 1: R50 per hour
Shift 2: R70 per hour
Shift 3: R90 per hour
2

Would you like to apply for a retirement fund? (yes/no)
abcd
```



```
run:
Welcome to the UrbanFurn Payroll Calculator!
-----

Please enter the hours you've worked for the week:
40

Please enter your shift number: (1, 2, or 3)
Shift 1: R50 per hour
Shift 2: R70 per hour
Shift 3: R90 per hour
2

Would you like to apply for a retirement fund? (yes/no)
abcd

Hours: 40
Shift: 2nd
Rate: R70 per hour

Regular Pay: R2800.0
Net: R2800.0
BUILD SUCCESSFUL (total time: 2 minutes 26 seconds)
```

Cycle 2: Existing Data

Regular Salary

```
20      @Test
21      public void testPayrollRegular() {
22          calculations calc = new calculations();
23          main.hours = 40;
24          main.shift = 70;
25          calc.perfCalc();
26          assertEquals(2800, calc.payroll, 0.01);
27      }
```

This test method sets the hours variable to 40, and the shift's pay rate to 70. What should occur according to the method, is that the system picks up that the inputted hours are not above 40. Therefore, it multiplies the value by 70 to acquire the payroll. The product of 40 and 70 is 2800.

Overtime Salary

```
11      @Test
12      public void testTotal() {
13          calculations calc = new calculations();
14          main.hours = 50;
15          main.shift = 70;
16          calc.perfCalc();
17          assertEquals(2800, calc.total, 0.01);
18      }
```

This test method sets the hours variable to 50, and the shift's pay rate to 70. What should occur according to the method, is that the system picks up that the inputted hours are above 40. Regardless, it multiplies only 40 by 70 to acquire the regular payroll. The product of 40 and 70 is 2800.

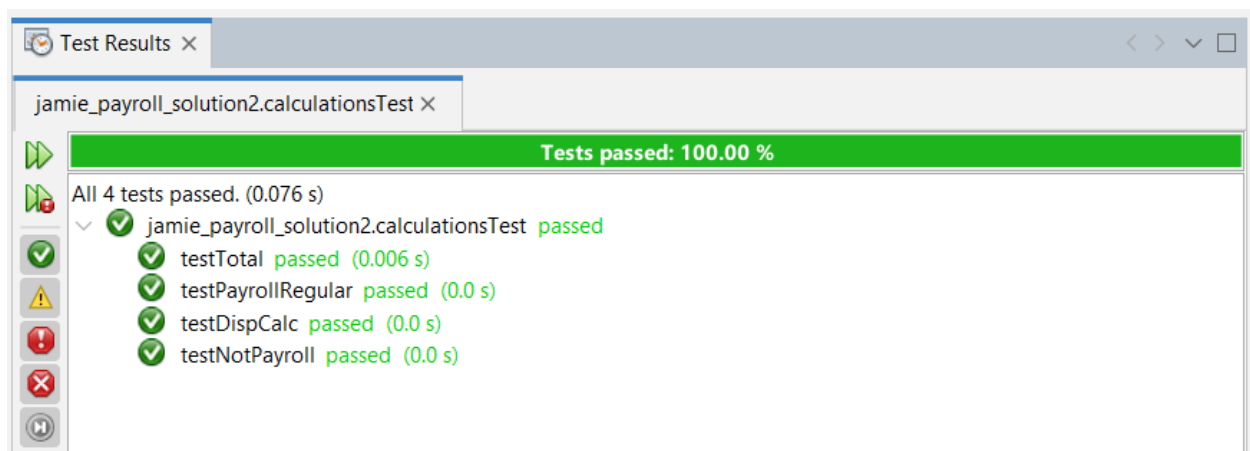
Total Payroll

This method sets the hours variable to 50, and the shift's pay rate to 70. What should occur, according to the method, is that the system picks up that the inputted hours are above 40. Therefore, the extra ten hours are overtime. The rate for the extra ten hours is multiplied by 70, produced by one and one-half. 40 multiplied by 70 is 2800, and ten multiplied by 70 produced by one and one-half, is 1050. The total should be 3850.

Not-Equals

```
37  @Test
38  public void testNotPayroll() {
39      calculations calc = new calculations();
40      main.hours = 50;
41      main.shift = 70;
42      calc.perfCalc();
43      assertEquals(3500, calc.total, 0.01);
44  }
```

This method tests that an answer is not true. Although the system recognizes that 50 has been inputted for the hours, and the shift rate is at 70. The method should not multiply 50 by 70 (which is 3500) because overtime is not calculated for the entire shift duration but only the hours exceeding 40.



All four tests passed.

Retirement with Overtime

```
11  @Test
12  public void testOvertime() {
13      calculations calc = new calculations();
14      retirement ret = new retirement();
15
16      main.input = "yes";
17      main.shift = 70;
18      main.hours = 50;
19
20      calc.perfCalc();
21      ret.perfRetire();
22
23      assertEquals(3850, calc.payroll, 0.01);
24      assertEquals(192.5, ret.retirement, 0.01);
25
26  }
```

This method tests the integrity and validity of a retirement fund if the user's input is to confirm the option. It first calculates the payroll, including the overtime pay, and then calculates the retirement by setting it to 5% of the payroll.

Retirement without Overtime

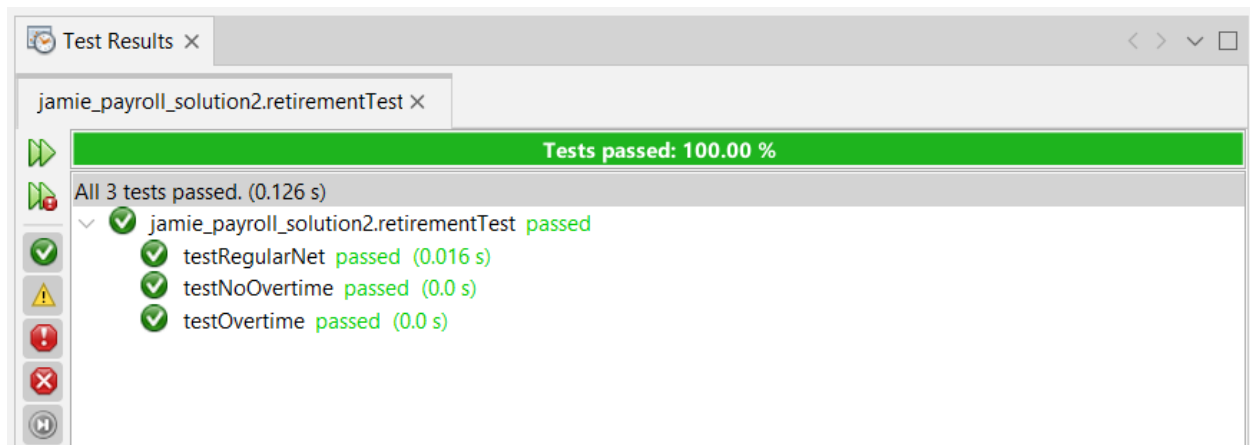
```
30  @Test
31  public void testNoOvertime() {
32
33      calculations calc = new calculations();
34      retirement ret = new retirement();
35
36      main.input = "yes";
37      main.shift = 70;
38      main.hours = 40;
39
40      calc.perfCalc();
41      ret.perfRetire();
42
43      assertEquals(2800, calc.payroll, 0.01);
44      assertEquals(140, ret.retirement, 0.01);
45
46  }
```

This method tests the integrity and validity of a retirement fund if the user's input is to confirm the option. It first calculates the payroll, excluding the overtime pay, and then calculates the retirement by setting it to 5% of the payroll.

Net Income for Basic Salary

```
48  @Test
49  public void testRegularNet() {
50      calculations calc = new calculations();
51      retirement ret = new retirement();
52
53      main.input = "yes";
54      main.shift = 50;
55      main.hours = 40;
56
57      calc.perfCalc();
58      ret.regularNet();
59
60      assertEquals(ret.net, calc.payroll, 0.01);
61  }
```

This method tests the integrity and validity of a net income of a basic salary. It first calculates the payroll, by multiplying the shift pay-rate by the hours worked, and then sets the payroll equal to the net.



All three tests passed.