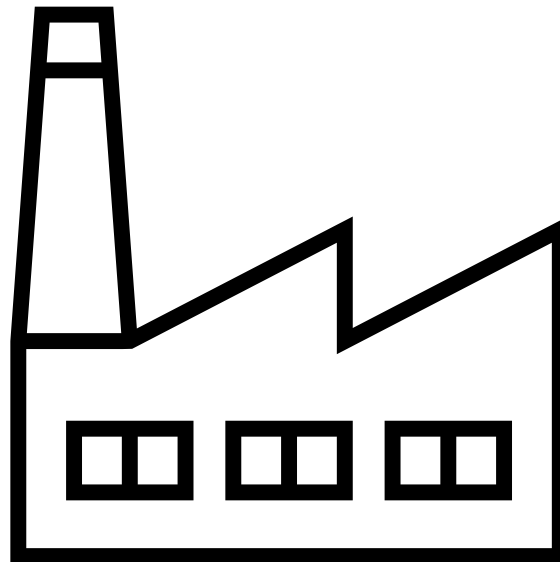


# Summative Assessment 4

18-23 July 2024

Jamie Myburgh



Designing  
Question 2

# Contents

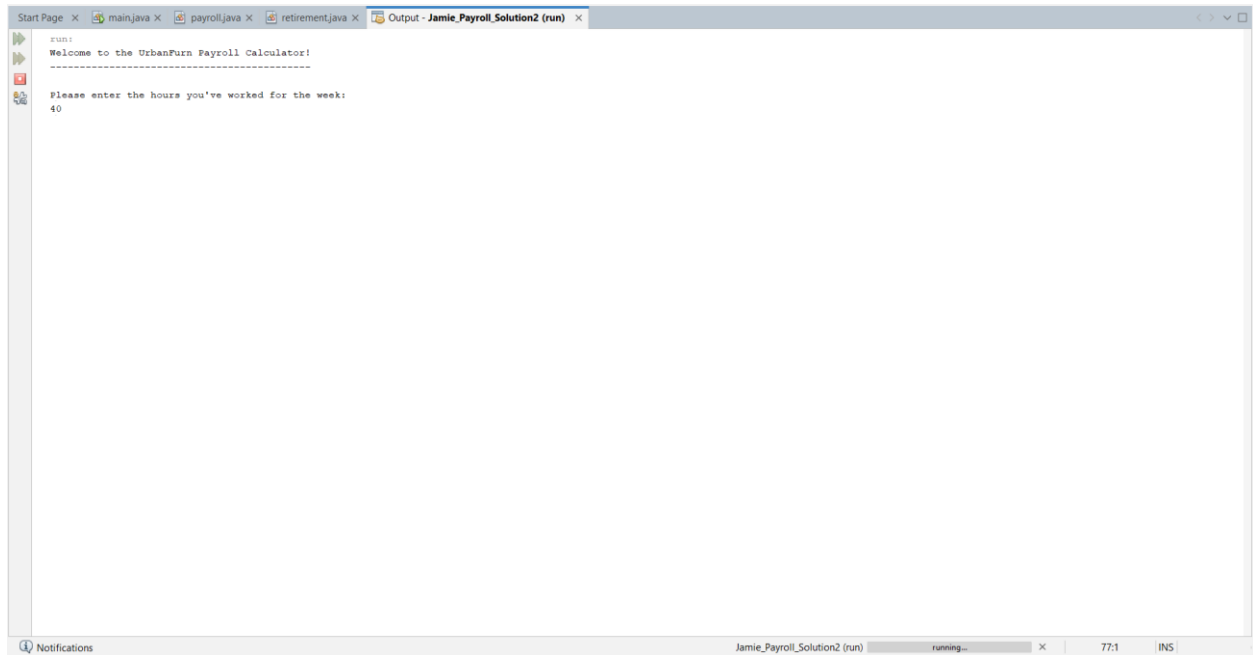
a) Desk Checking .....	3
i. User Hours Input .....	3
ii. User Shift Input.....	4
iii. Standard Wage.....	5
iv. Overtime .....	6
v. Retirement Plan.....	7
vi. Display Information .....	8
b) UML Diagram .....	9
ii. Worker and Hours .....	9
c) Flowchart and Pseudocode.....	10
Flowchart: .....	10
Pseudocode:.....	11
d) Data Structures .....	14
Date Structures.....	14
Classes.....	14
Objects .....	16
Random Access Manager.....	17

## a) Desk Checking

The following content displays a desk checking to ensure that the program executes its required tasks.

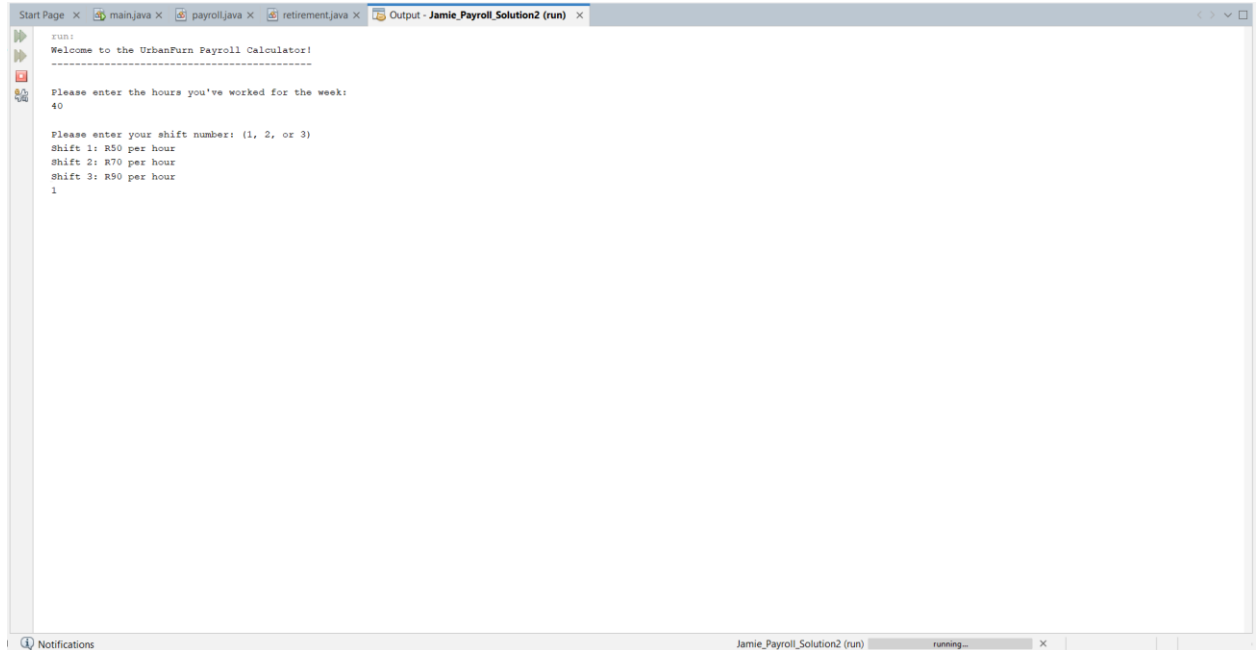
### i. User Hours Input

The program will prompt each worker to input their hours worked for the week.



## ii. User Shift Input

Workers will select their shift number and are paid one of three hourly rates depending on their shift. The first shift is R50 per hour, the second shift is R70 per hour, and the third shift is R90 per hour.



The screenshot shows a Java IDE with several tabs: 'Start Page', 'main.java', 'payroll.java', 'retirement.java', and 'Output - Jamie\_Payroll\_Solution2 (run)'. The 'Output' tab is active, displaying the following text:

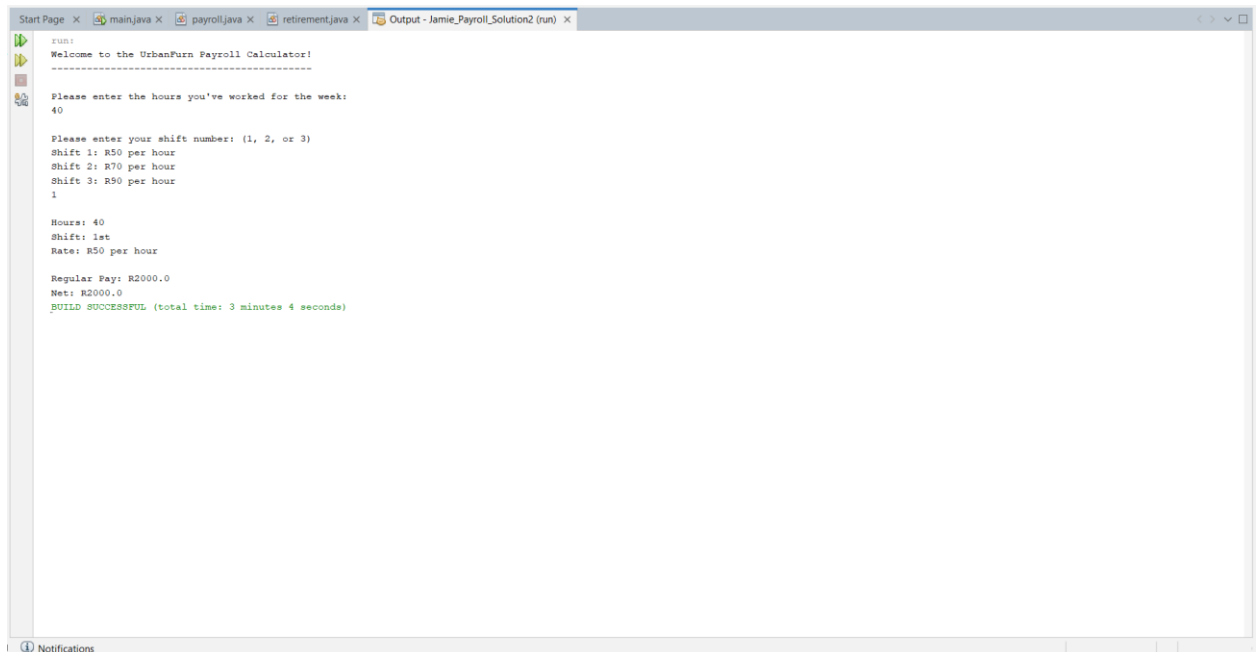
```
run:
Welcome to the UrbanFurn Payroll Calculator!
-----
Please enter the hours you've worked for the week:
40

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

At the bottom of the IDE, there is a 'Notifications' bar and a status bar showing 'Jamie\_Payroll\_Solution2 (run)' and 'running...'.

### iii. Standard Wage

Should a worker work the first shift for 40 hours, they will be paid the standard wage.



```
Start Page x main.java x payroll.java x retirement.java x Output - Jamie_Payroll_Solution2 (run) x
run:
Welcome to the UrbanFurn Payroll Calculator!
-----
Please enter the hours you've worked for the week:
40

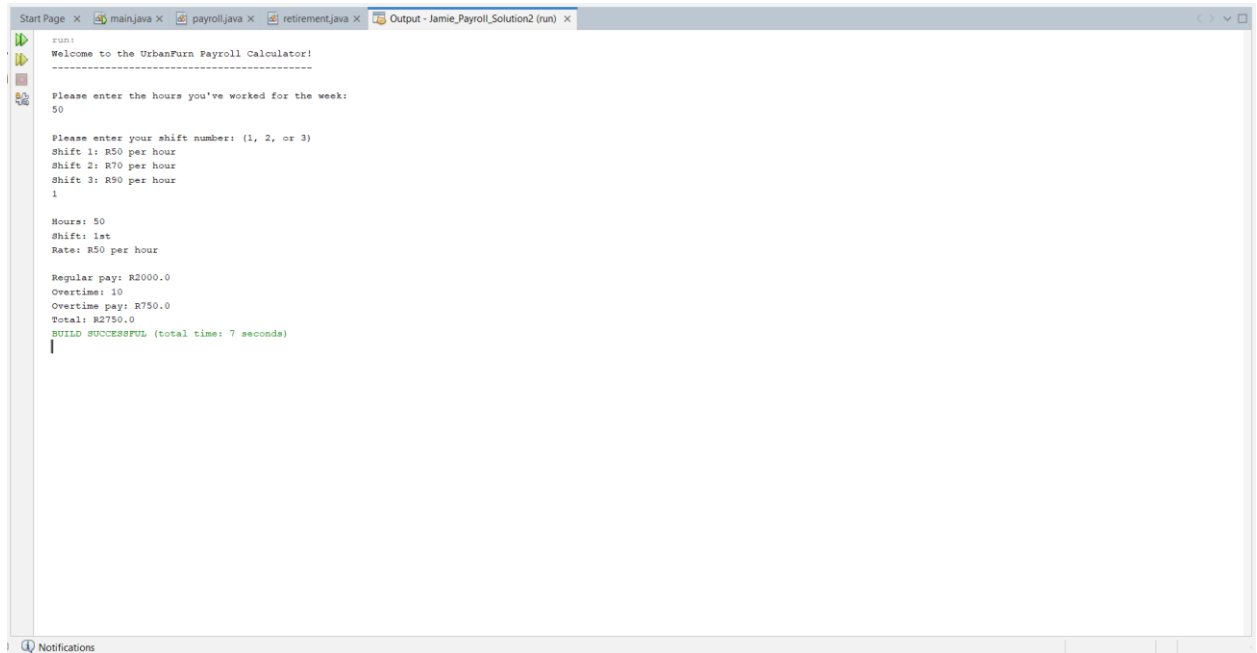
Please enter your shift number: (1, 2, or 3)
Shift 1: $50 per hour
Shift 2: $70 per hour
Shift 3: $90 per hour
1

Hours: 40
Shift: 1st
Rate: $50 per hour

Regular Pay: $2000.0
Net: $2000.0
BUILD SUCCESSFUL (total time: 3 minutes 4 seconds)
```

#### iv. Overtime

Any hours greater than 40 are paid at one and one-half of the usual rate.



The screenshot shows a Java IDE with several tabs: 'Start Page', 'main.java', 'payroll.java', 'retirement.java', and 'Output - Jamie\_Payroll\_Solution2 (run)'. The 'Output' tab is active, displaying the following text:

```
run:
Welcome to the UrbanFurn Payroll Calculator!
-----
Please enter the hours you've worked for the week:
50

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

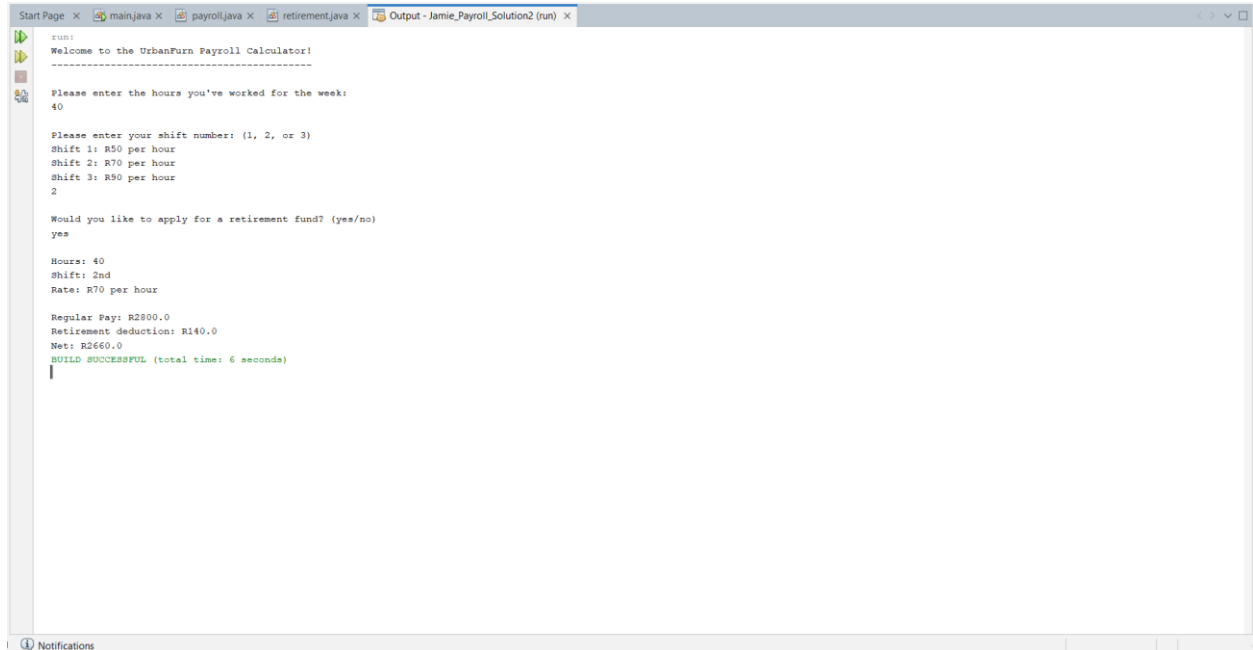
Hours: 50
Shift: 1st
Rate: R50 per hour

Regular pay: R2000.0
Overtime: 10
Overtime pay: R750.0
Total: R2750.0
BUILD SUCCESSFUL (total time: 7 seconds)
```

A notification bar at the bottom of the IDE shows a bell icon and the text 'Notifications'.

## v. Retirement Plan

Second and third-shift workers can elect to participate in the retirement plan, for which 5% of the worker's gross pay is deducted from their paycheck. A prompt will ask whether the user would like to participate in the retirement plan or not (yes/no).



```
Start Page x main.java x payroll.java x retirement.java x Output - Jamie_Payroll_Solution2 (run) x
run:
Welcome to the UrbanPurn 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)
yes

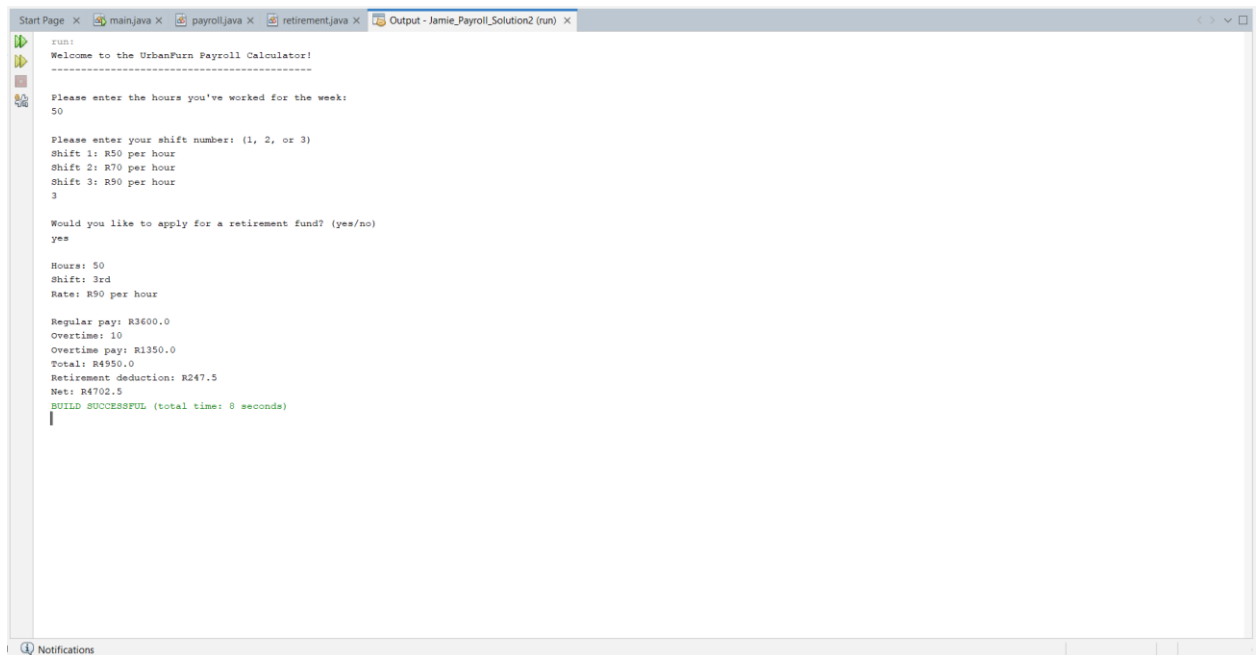
Hours: 40
Shift: 2nd
Rate: R70 per hour

Regular Pay: R2800.0
Retirement deduction: R140.0
Net: R2660.0
BUILD SUCCESSFUL (total time: 6 seconds)
```

## vi. Display Information

The output of the program will include a display providing a breakdown of the following information:

- The hours worked
- The shift number
- The hourly pay rate
- The standard pay
- The overtime pay, if there is any
- The total of the regular and overtime pay
- The retirement deduction, if applicable and chosen
- The net pay



The screenshot shows a Java IDE with several tabs: 'main.java', 'payroll.java', 'retirement.java', and 'Output - Jamie\_Payroll\_Solution2 (run)'. The 'Output' tab is active, displaying the following text:

```
run:
Welcome to the UrbanPurn Payroll Calculator!
-----
Please enter the hours you've worked for the week:
50

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

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

Hours: 50
Shift: 3rd
Rate: R90 per hour

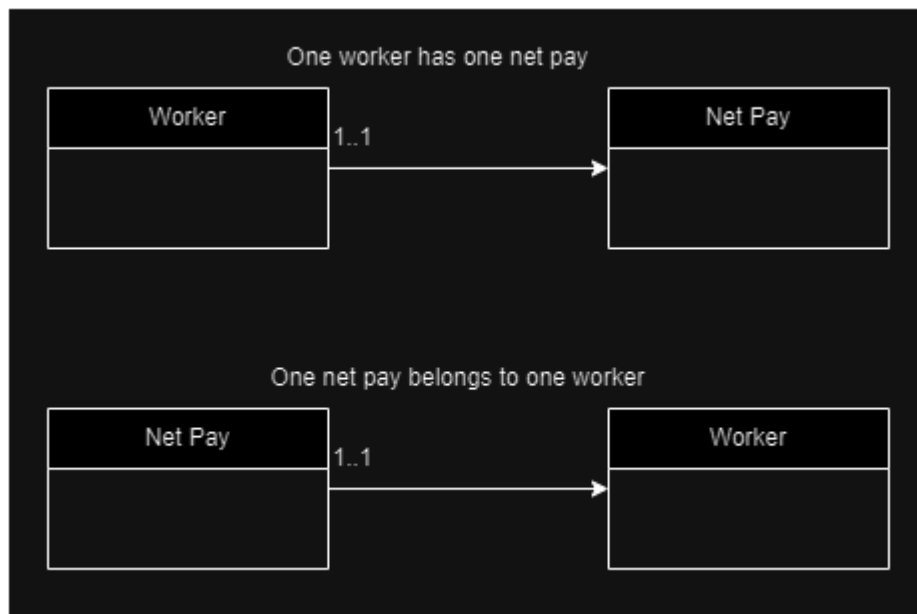
Regular pay: R3600.0
Overtime: 10
Overtime pay: R1350.0
Total: R4950.0
Retirement Deduction: R247.5
Net: R4702.5
BUILD SUCCESSFUL (total time: 8 seconds)
```



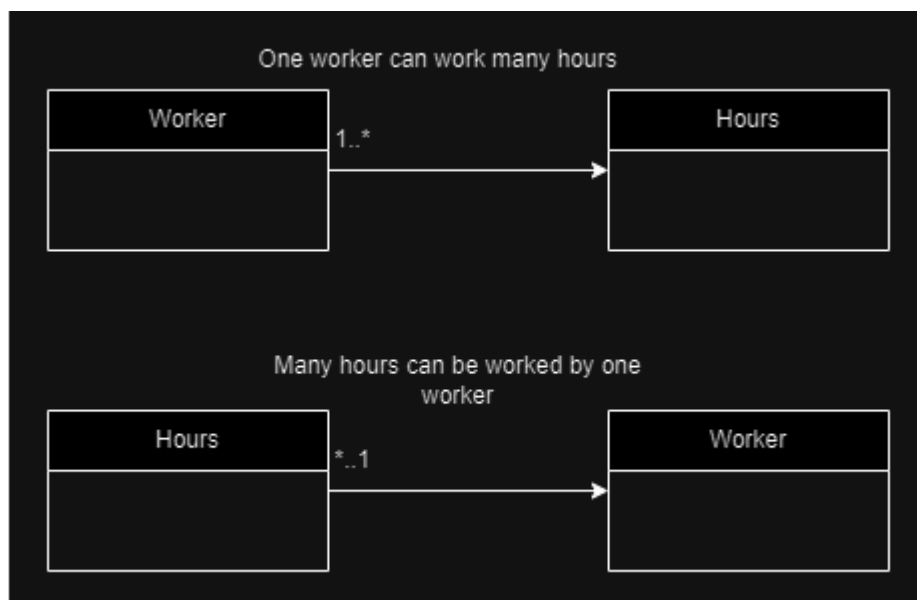
## b) UML Diagram

A unified modelling language diagram has been created to represent the relationships between the entities within the program.

### i. Worker and Net Pay



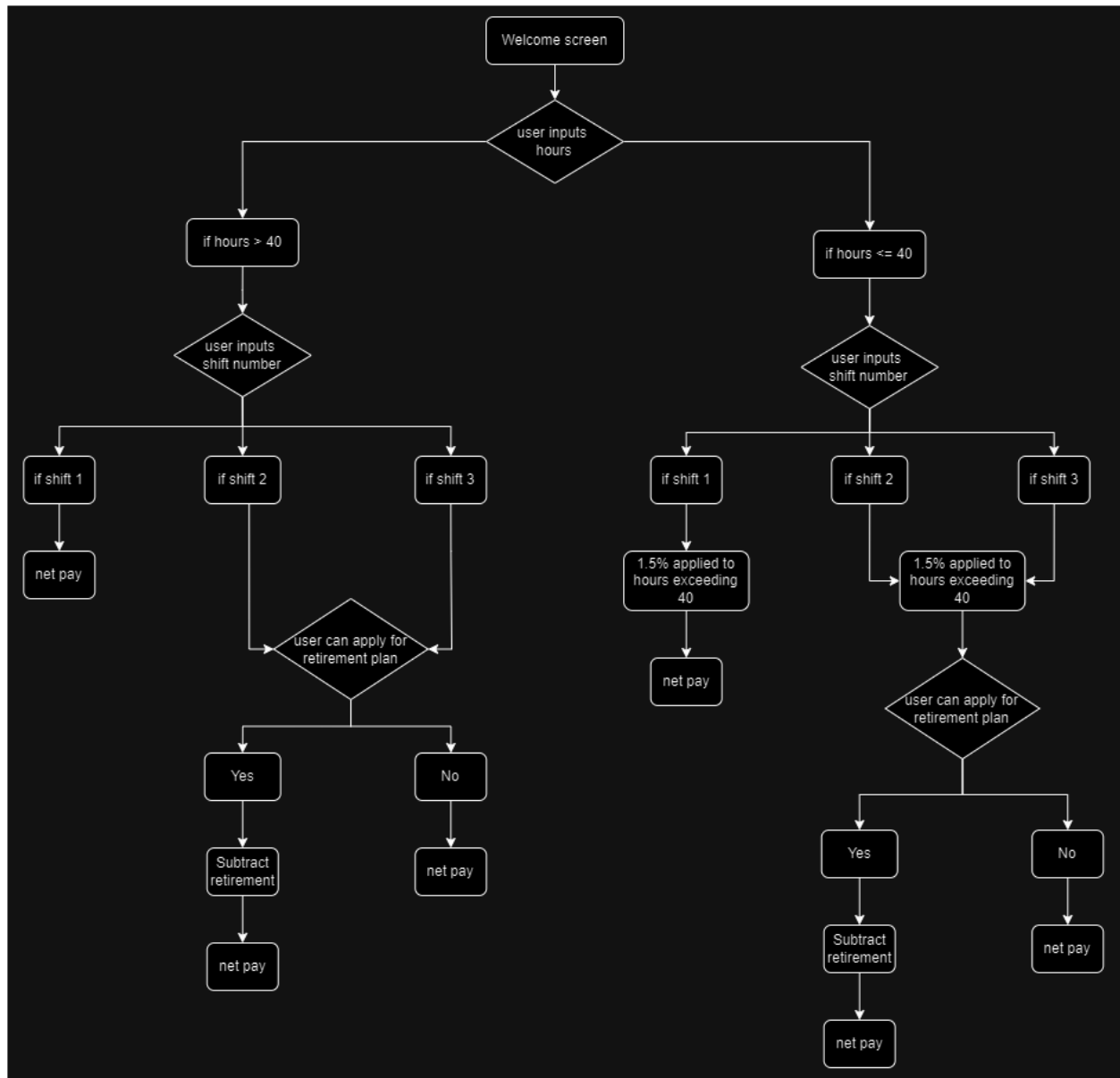
### ii. Worker and Hours



## c) Flowchart and Pseudocode

### Flowchart:

A flow chart has been created to visualize the program's operational flow as well as to document the overall system's architecture.



## Pseudocode:

### Start

(Main class)

Initialize integer variables "hours" and "shift" and a shift variable "input"

Initialize a string variable "input"

Instantiate a new Scanner instance called "sc"

Instantiate instances of the other classes

Print the welcome message into the console

Ask the user to enter the hours they've worked for this week

Collect input with Scanner and store input in "hours" variable

Ask user to enter their shift number (1, 2, or 3)

Collect input with Scanner and store input in the "shift integer" integer-variable

If "shift integer" variable is equal to 1:

Set "shift" to 50

Print the hours, shift, and rate into the console

Call method to display calculations from the calculations class

Call method to display the regular net pay from the retirement class

Else if "shift integer" is equal to 2:

Set "shift" to 70

Ask the user if they'd like to apply for a retirement fund

Store the user's input in the "input" variable

Print the hours, shift, and rate into the console

Call method to display calculations from the calculations class

Call method to display the regular net pay from the retirement class

Else if "shift integer" is equal to 3:

Set "shift" to 90

Ask the user if they'd like to apply for a retirement fund

Store the user's input in the "input" variable

Print the hours, shift, and rate into the console

Call method to display calculations from the calculations class

Call method to display the regular net pay from the retirement class

### Stop

Start

Calculations class extends main class

Initialize the double variables "total", "payroll", and "overtime"

Create method to perform calculations

If inputted "hours" is greater than 40:

Calculate "total" as 40 times the "shift"

Calculate overtime as 1.5 times the shift rate for hours exceeding 40

Calculate "payroll" as the sum of "total" and "overtime"

Else:

Calculate payroll as product of "hours" and "shift"

Create method to display calculations

In inputted "hours" is greater than 40:

Call method to perform calculations

Print the regular pay with the "total" value

Print the overtime pay with the "overtime" value

Print the total pay with the "payroll" value

Else:

Call method to perform calculations

Print the regular pay with the "payroll" value

Stop

Start

Retirement class extends calculations class

Initialize double variables for "retirement" and "net"

Create method to perform retirement calculations

Store user's input in "input"

If user's "input" is equal to "yes"

Call the method to perform the calculations of the calculations

class

Calculate "retirement" as "payroll" times 0.05

Calculate "net" as "payroll" minus "retirement"

Print the net payment with the "net" value

Else:

Call the method to perform the calculations of the calculations

class

Set "net" as "payroll" value

Print the net payment with the "net" value

Create method to perform the regular net payment calculation

If inputted "hours" <= 40:

Set "net" as "payroll" value

Print the net payment with the "net" value

Stop

## d) Data Structures

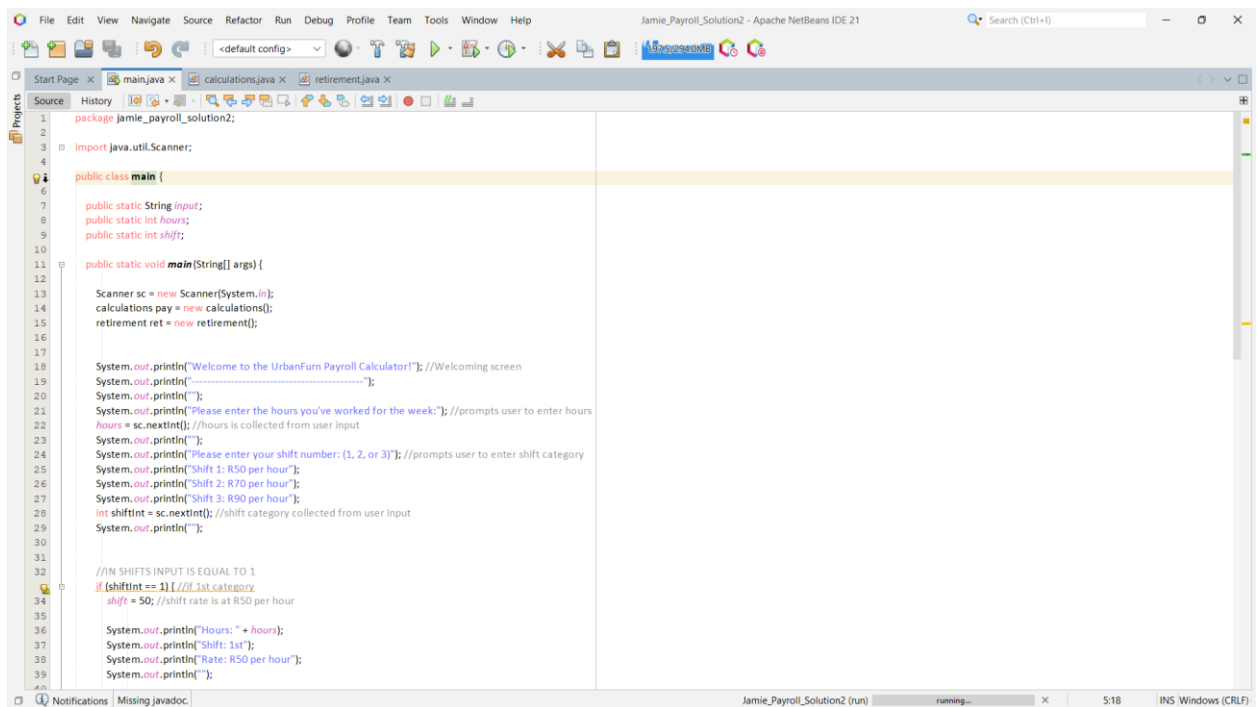
### Date Structures

As seen within the pseudocode, the program makes use of multi-level inheritance. Inheritance is when an object or class is based on another object or class, using the same implementation (inheriting from a class) or specifying implementation to maintain the same behaviour (realizing an interface, inheriting behaviours).

Multi-level inheritance occurs when derived classes inherit properties and methods from other derived classes and not one common base class. This occurs when the class to performs the retirement calculations inherits from the class that performs the payroll calculations.

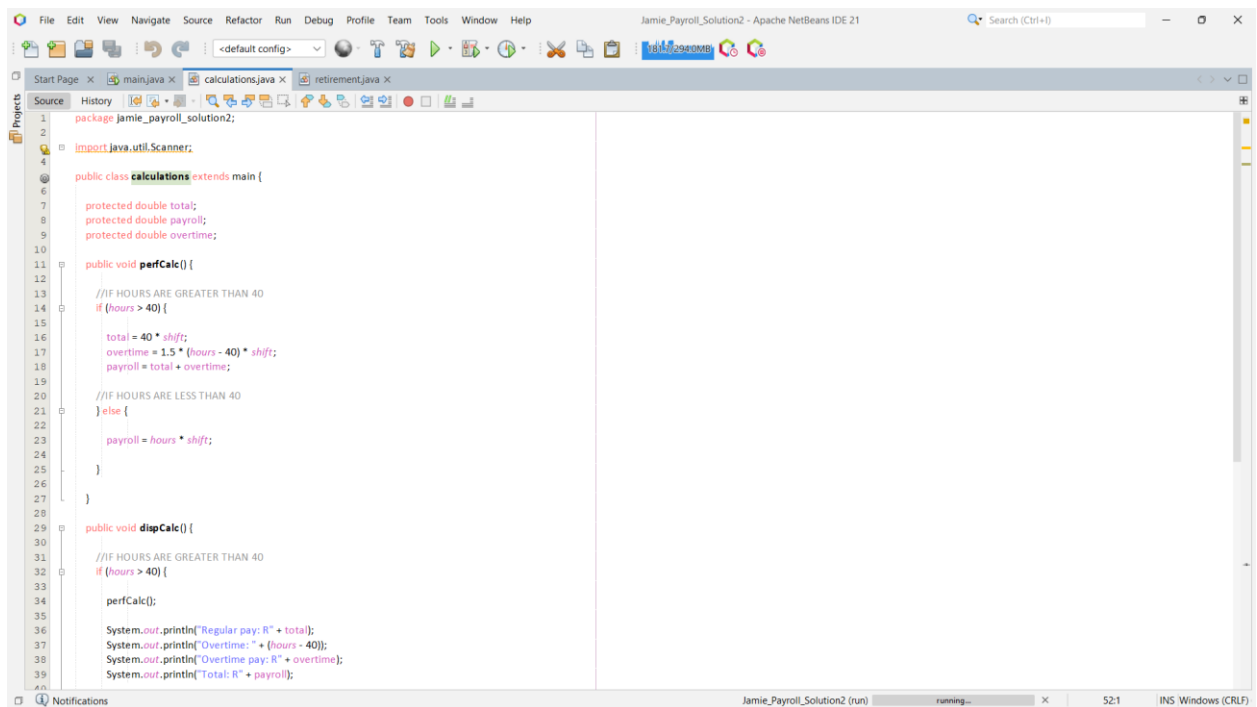
### Classes

The main class:



```
1 package jamie_payroll_solution2;
2
3 import java.util.Scanner;
4
5 public class main {
6
7     public static String input;
8     public static int hours;
9     public static int shift;
10
11     public static void main(String[] args) {
12
13         Scanner sc = new Scanner(System.in);
14         calculations pay = new calculations();
15         retirement ret = new retirement();
16
17         System.out.println("Welcome to the UrbanFurn Payroll Calculator!"); //Welcoming screen
18         System.out.println("-----");
19         System.out.println("");
20         System.out.println("Please enter the hours you've worked for the week:"); //prompts user to enter hours
21         hours = sc.nextInt(); //hours is collected from user input
22         System.out.println("");
23         System.out.println("Please enter your shift number: (1, 2, or 3)"); //prompts user to enter shift category
24         System.out.println("Shift 1: R50 per hour");
25         System.out.println("Shift 2: R70 per hour");
26         System.out.println("Shift 3: R90 per hour");
27         int shiftint = sc.nextInt(); //shift category collected from user input
28         System.out.println("");
29
30         //IN SHIFTS INPUT IS EQUAL TO 1
31         if (shiftint == 1) { //if 1st category
32             shift = 50; //shift rate is at R50 per hour
33
34             System.out.println("Hours: " + hours);
35             System.out.println("Shift: 1st");
36             System.out.println("Rate: R50 per hour");
37             System.out.println("");
38         }
39     }
40 }
```

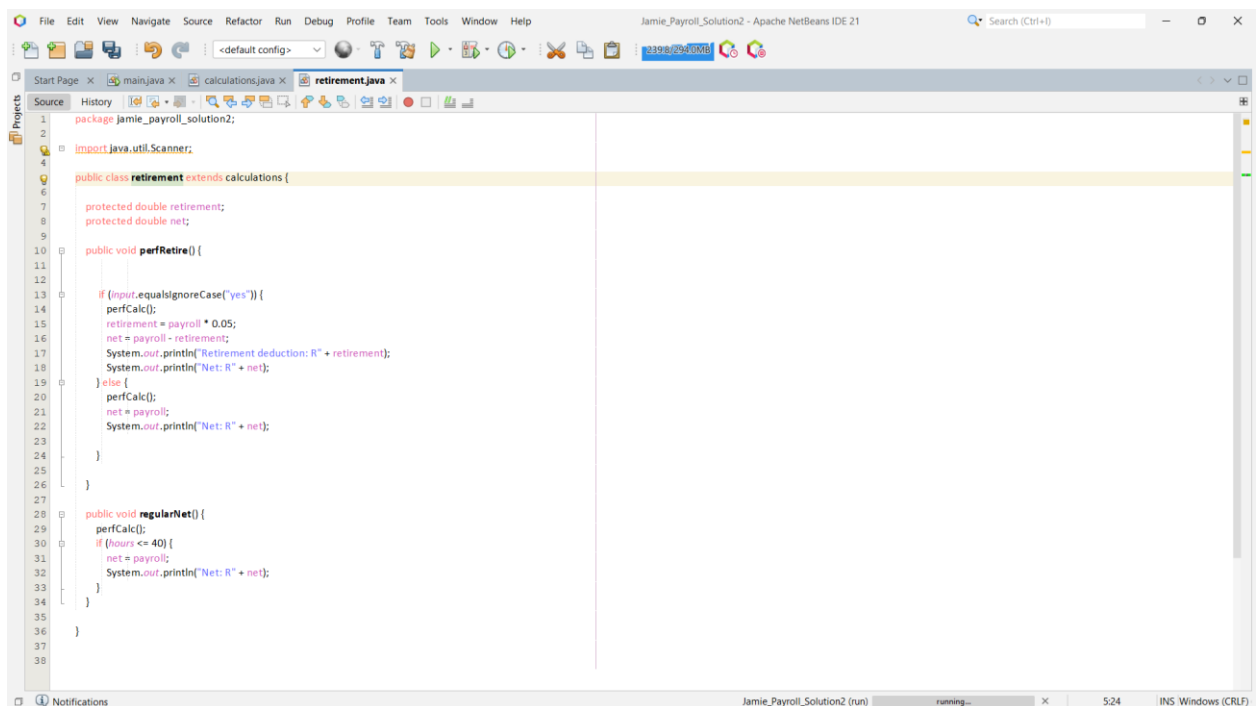
## The payroll calculations class:



The screenshot shows the Apache NetBeans IDE with the file 'calculations.java' open. The code defines a package 'jamie\_payroll\_solution2', imports 'java.util.Scanner', and defines a public class 'calculations' that extends 'main'. The class contains three methods: 'perfCalc()', 'dispCalc()', and 'regularNet()'. 'perfCalc()' calculates total, overtime, and payroll based on hours. 'dispCalc()' displays the results. 'regularNet()' calculates the regular net pay.

```
1 package jamie_payroll_solution2;
2
3 import java.util.Scanner;
4
5 public class calculations extends main {
6
7     protected double total;
8     protected double payroll;
9     protected double overtime;
10
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     public void dispCalc() {
28
29         //IF HOURS ARE GREATER THAN 40
30         if (hours > 40) {
31
32             perfCalc();
33
34             System.out.println("Regular pay: R" + total);
35             System.out.println("Overtime: " + (hours - 40));
36             System.out.println("Overtime pay: R" + overtime);
37             System.out.println("Total: R" + payroll);
38         }
39     }
40
41     public void regularNet() {
42
43         perfCalc();
44         if (hours <= 40) {
45             net = payroll;
46             System.out.println("Net: R" + net);
47         }
48     }
49 }
```

## The retirement calculations class:



The screenshot shows the Apache NetBeans IDE with the file 'retirement.java' open. The code defines a package 'jamie\_payroll\_solution2', imports 'java.util.Scanner', and defines a public class 'retirement' that extends 'calculations'. The class contains two methods: 'perfRetire()' and 'regularNet()'. 'perfRetire()' calculates retirement deduction and net pay. 'regularNet()' calculates the regular net pay.

```
1 package jamie_payroll_solution2;
2
3 import java.util.Scanner;
4
5 public class retirement extends calculations {
6
7     protected double retirement;
8     protected double net;
9
10     public void perfRetire() {
11
12         if (input.equalsIgnoreCase("yes")) {
13             perfCalc();
14             retirement = payroll * 0.05;
15             net = payroll - retirement;
16             System.out.println("Retirement deduction: R" + retirement);
17             System.out.println("Net: R" + net);
18         } else {
19             perfCalc();
20             net = payroll;
21             System.out.println("Net: R" + net);
22         }
23     }
24
25     public void regularNet() {
26
27         perfCalc();
28         if (hours <= 40) {
29             net = payroll;
30             System.out.println("Net: R" + net);
31         }
32     }
33
34 }
35
36
37
38 }
```

## Objects

Two objects were created in the main class. These were instances of the payroll and retirement calculations class, that were instantiated to call upon their methods within the main class. A representation of this is found below:

```
Scanner sc = new Scanner(System.in);
calculations pay = new calculations();
retirement ret = new retirement();
```

The objects were called within the if-statements

```
32 //IN SHIFTS INPUT IS EQUAL TO 1
33 if (shiftInt == 1) { //if 1st category
34     shift = 50; //shift rate is at R50 per hour
35
36     System.out.println("Hours: " + hours);
37     System.out.println("Shift: 1st");
38     System.out.println("Rate: R50 per hour");
39     System.out.println("");
40
41     pay.dispCalc(); //performs calculation
42     ret.regularNet();
43 }
```

```
44 //IF SHIFTS INPUT IS EQUAL TO 2
45 } else if (shiftInt == 2) {
46     shift = 70;
47
48     System.out.println("Would you like to apply for a retirement fund? (yes/no)");
49     input = sc.next();
50
51     System.out.println("");
52     System.out.println("Hours: " + hours);
53     System.out.println("Shift: 2nd");
54     System.out.println("Rate: R70 per hour");
55     System.out.println("");
56
57     pay.dispCalc();
58     ret.perfRetire();
59 }
```

```
60 //IF SHIFTS INPUT IS EQUAL TO 3
61 } else if (shiftInt == 3) {
62     shift = 90;
63
64     System.out.println("Would you like to apply for a retirement fund? (yes/no)");
65     input = sc.next();
66
67     System.out.println("");
68     System.out.println("Hours: " + hours);
69     System.out.println("Shift: 3rd");
70     System.out.println("Rate: R90 per hour");
71     System.out.println("");
72
73     pay.dispCalc();
74     ret.perfRetire();
75 }
```



## Random Access Manager

RandomAccessManager had been implemented in the code to store the number of hours entered each time the program is run. This feature allows one to read and write to any location in the file.

```
79  try {  
80      RandomAccessFile file = new RandomAccessFile("hours.txt", "rw");  
81      file.seek(file.length());  
82      file.writeBytes("Hours: " + String.valueOf(hours) + "\n");  
83      file.writeBytes("Shift: R" + String.valueOf(shift) + " per hour" + "\n");  
84      file.writeBytes("");  
85      file.close();  
86  } catch (IOException exception) {  
87      exception.printStackTrace();  
88  }
```