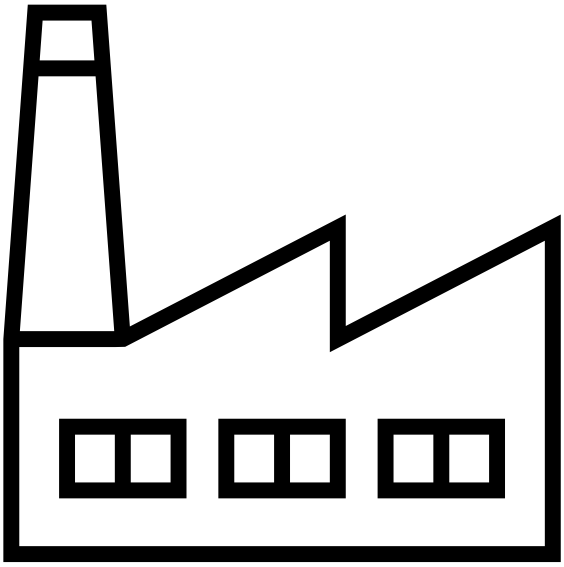
Summative Assessment 4

18-23 July 2024

Jamie Myburgh



Designing

Question 2

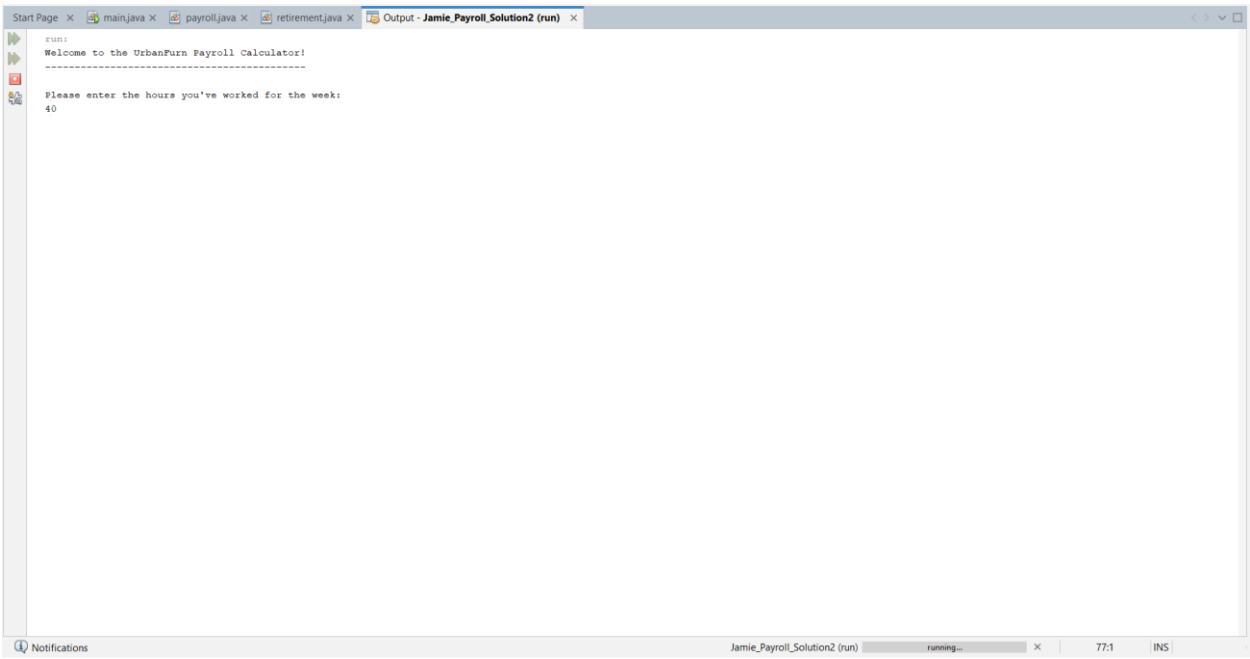
Contents

a) Desk Checking

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

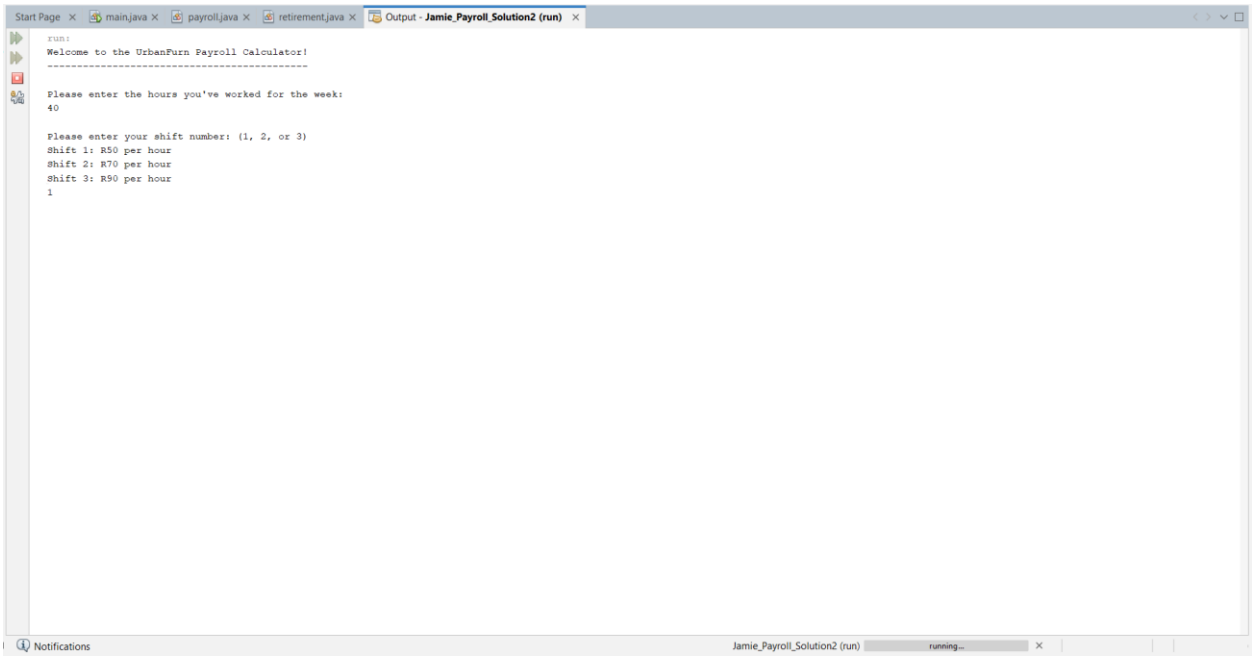
i. User Hours Input

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



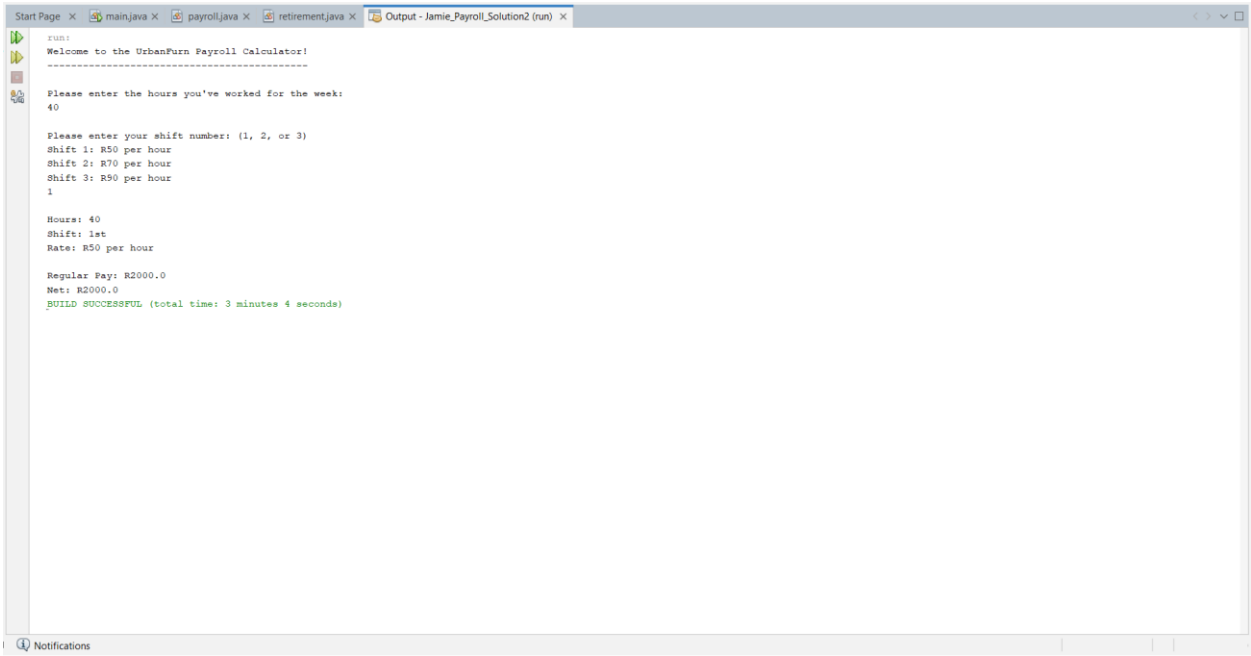
ii. User Shift Input

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.



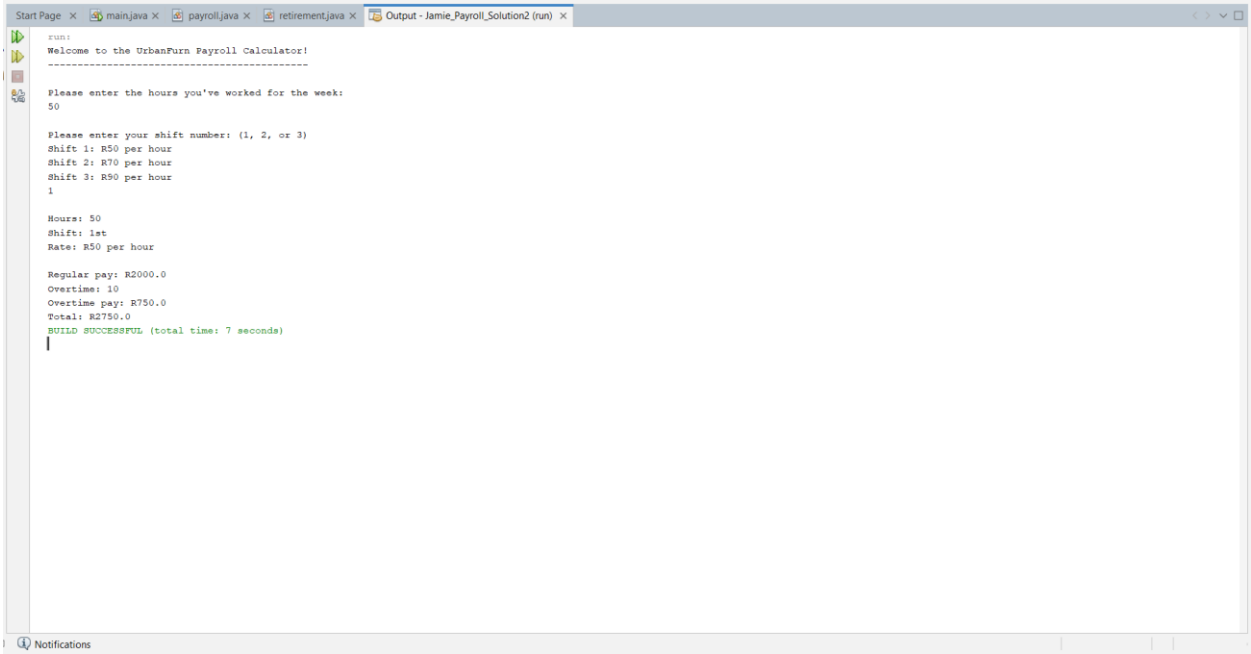
iii. Standard Wage

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



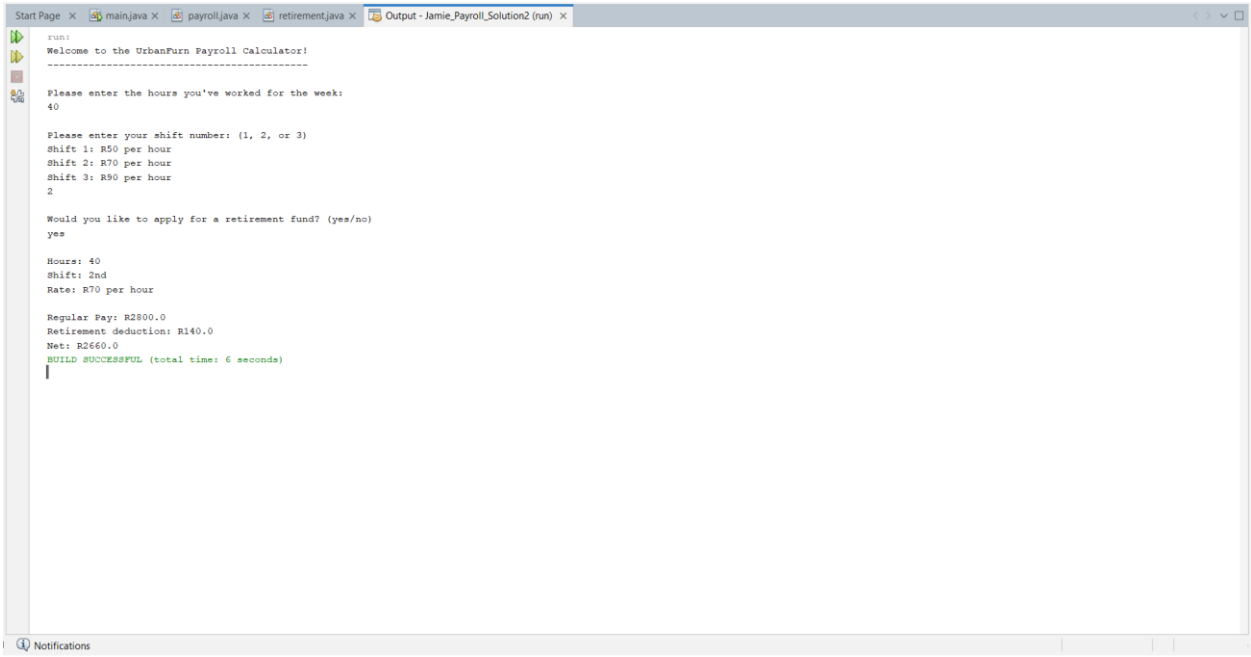
iv. Overtime

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



v. Retirement Plan

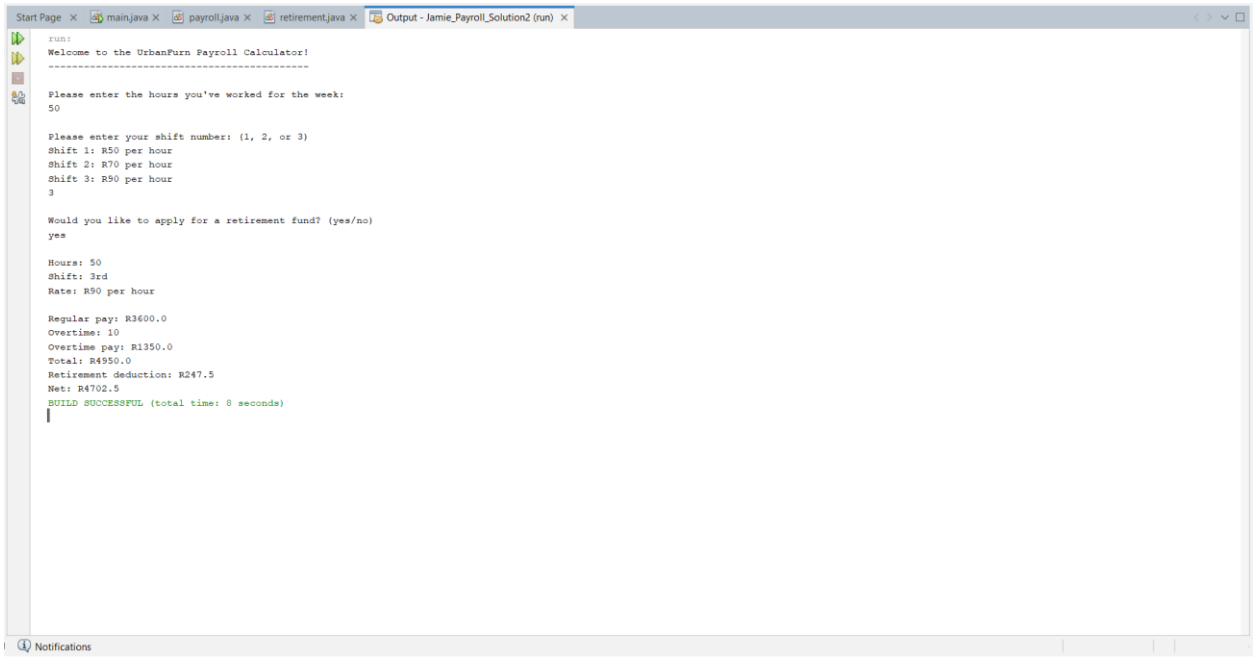
Second and third-shift workers can elect to participate in the retirement plan, for which 5% of the worker's gross pay is deducted from their paycheck. A prompt will ask whether the user would like to participate in the retirement plan or not (yes/no).



vi. Display Information

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

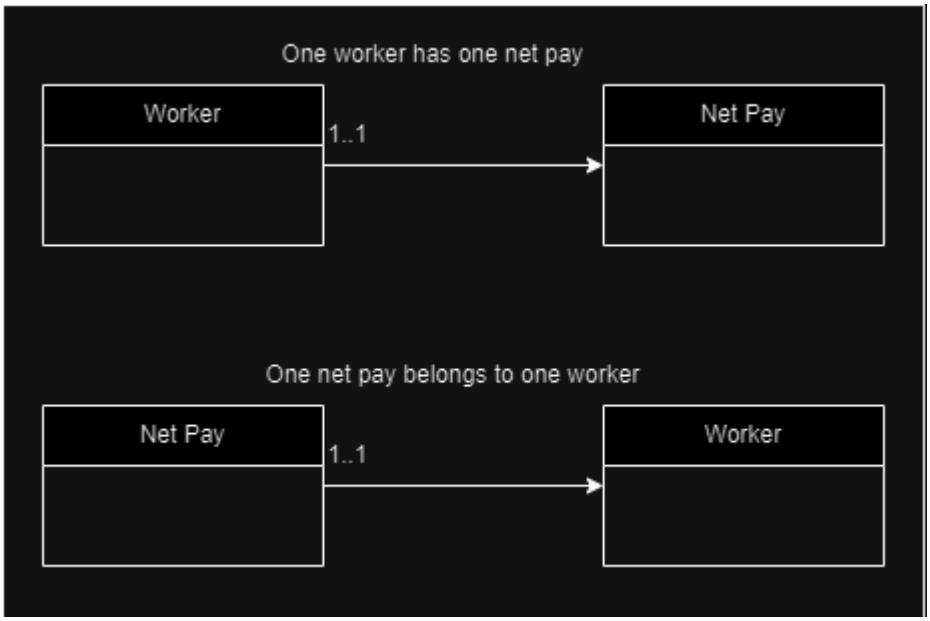
- The hours worked
- The shift number
- The hourly pay rate
- The standard pay
- The overtime pay, if there is any
- The total of the regular and overtime pay
- The retirement deduction, if applicable and chosen
- The net pay



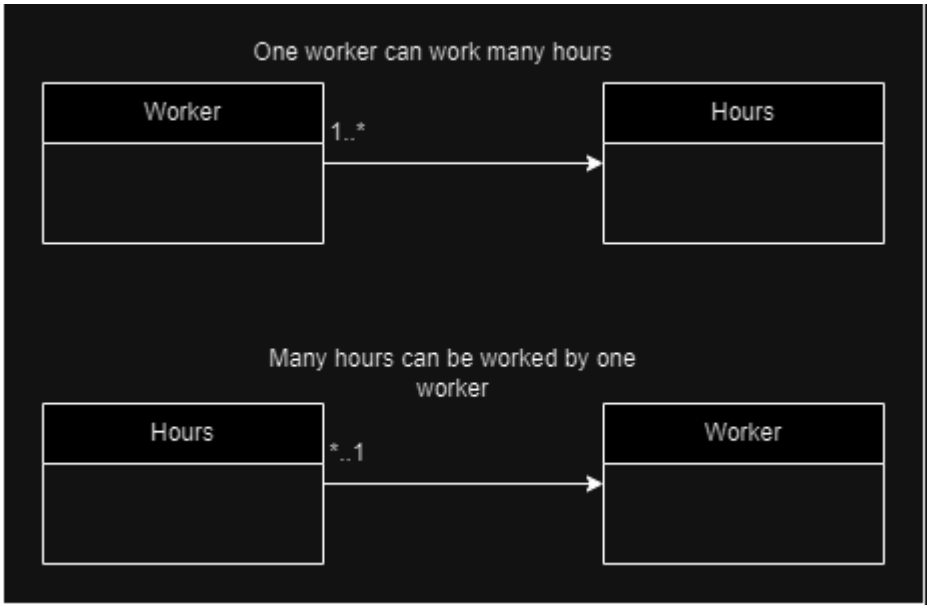
b) UML Diagram

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

i. Worker and Net Pay



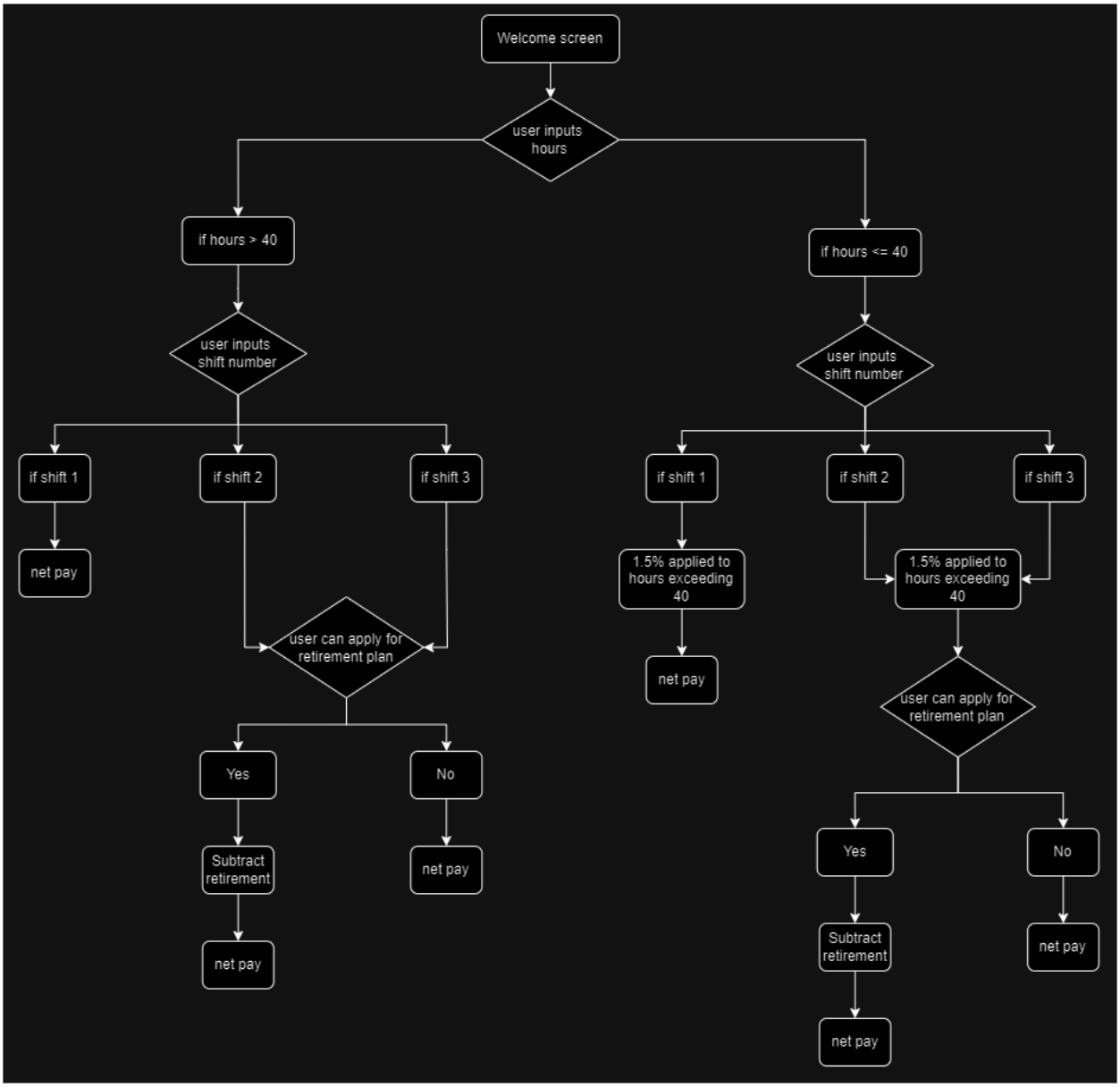
ii. Worker and Hours



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

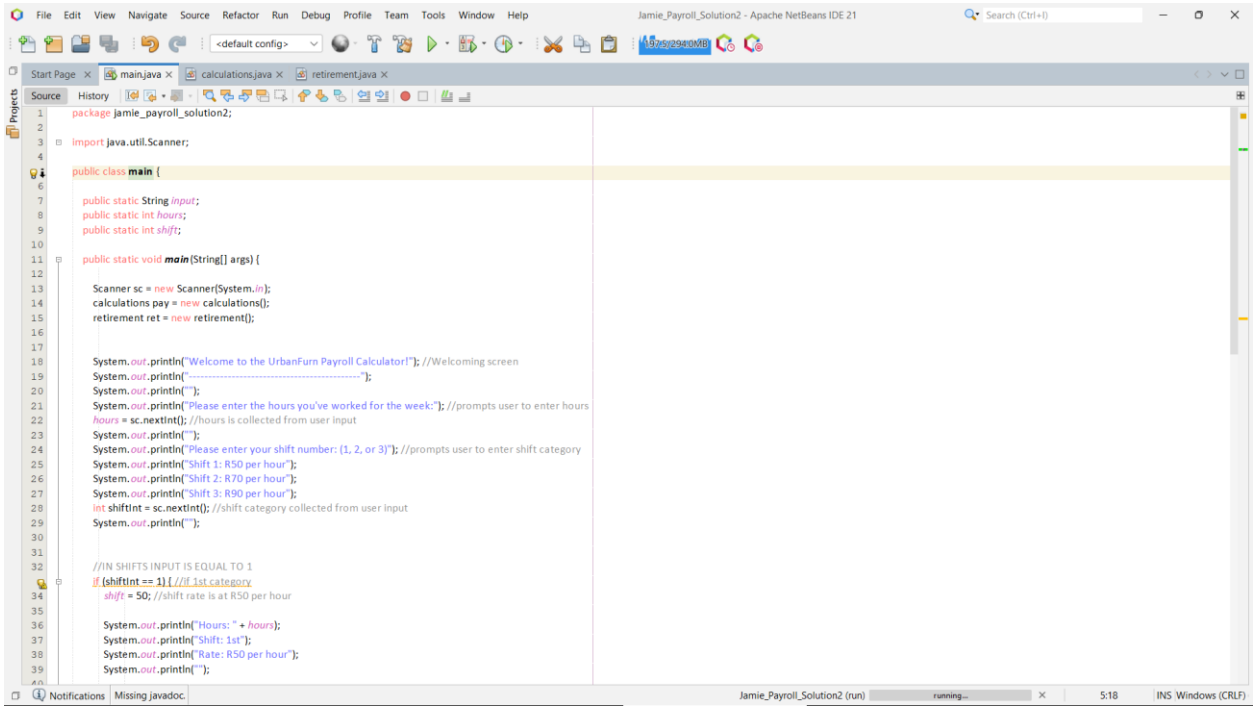
Date Structures

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:



The payroll calculations class:

```java
package jamie_payroll_solution2;

import java.util.Scanner;

public class calculations extends main {

  protected double total;
  protected double payroll;
  protected double overtime;

  public void perfCalc() {

    //IF HOURS ARE GREATER THAN 40
    if (hours > 40) {

      total = 40 * shift;
      overtime = 1.5 * (hours - 40) * shift;
      payroll = total + overtime;

    //IF HOURS ARE LESS THAN 40
    } else {

      payroll = hours * shift;

    }

  }

  public void dispCalc() {

    //IF HOURS ARE GREATER THAN 40
    if (hours > 40) {

      perfCalc();

      System.out.println("Regular pay: R" + total);
      System.out.println("Overtime: " + (hours - 40));
      System.out.println("Overtime pay: R" + overtime);
      System.out.println("Total: R" + payroll);
```

The retirement calculations class:

```java
package jamie_payroll_solution2;

import java.util.Scanner;

public class retirement extends calculations {

  protected double retirement;
  protected double net;

  public void perfRetire() {


    if (input.equalsIgnoreCase("yes")) {
      perfCalc();
      retirement = payroll * 0.05;
      net = payroll - retirement;
      System.out.println("Retirement deduction: R" + retirement);
      System.out.println("Net: R" + net);
    } else {
      perfCalc();
      net = payroll;
      System.out.println("Net: R" + net);

    }

  }

  public void regularNet() {
    perfCalc();
    if (hours <= 40) {
      net = payroll;
      System.out.println("Net: R" + net);
    }
  }

}
```

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

```
//IN SHIFTS INPUT IS EQUAL TO 1
if (shiftInt == 1) { //if 1st category
    shift = 50; //shift rate is at R50 per hour

    System.out.println("Hours: " + hours);
    System.out.println("Shift: 1st");
    System.out.println("Rate: R50 per hour");
    System.out.println("");

    pay.dispCalc(); //performs calculation
    ret.regularNet();

//IF SHIFTS INPUT IS EQUAL TO 2
} else if (shiftInt == 2) {
    shift = 70;

    System.out.println("Would you like to apply for a retirement fund? (yes/no)");
    input = sc.next();

    System.out.println("");
    System.out.println("Hours: " + hours);
    System.out.println("Shift: 2nd");
    System.out.println("Rate: R70 per hour");
    System.out.println("");

    pay.dispCalc();
    ret.perfRetire();

//IF SHIFTS INPUT IS EQUAL TO 3
} else if (shiftInt == 3) {
    shift = 90;

    System.out.println("Would you like to apply for a retirement fund? (yes/no)");
    input = sc.next();

    System.out.println("");
    System.out.println("Hours: " + hours);
    System.out.println("Shift: 3rd");
    System.out.println("Rate: R90 per hour");
    System.out.println("");

    pay.dispCalc();
    ret.perfRetire();
}
```

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.

```
try {
    RandomAccessFile file = new RandomAccessFile("hours.txt", "rw");
    file.seek(file.length());
    file.writeBytes("Hours: " + String.valueOf(hours) + "\n");
    file.writeBytes("Shift: R" + String.valueOf(shift) + " per hour" + "\n");
    file.writeBytes("");
    file.close();
} catch (IOException exception) {
    exception.printStackTrace();
}
```