a)

Purpose Description:

- The purpose of the program is to calculate overtime pay for the employees of the company UrbanFurn.

- The program will also include their regular pay, and retirement contributions only if the employee decides to participate in the retirement plan.

Scope and Functionality:

- **Input Handling:**
  - Accepts user input for hours worked and shift number.
  - Validates the input to ensure non-negative hours and valid shift numbers.
  - Asks for retirement plan participation for applicable shifts.

- **Payroll Calculation:**
  - Calculates regular pay based on shift-specific hourly rates.
  - Calculates overtime pay for hours worked beyond 40 hours at a rate of 1.5 times the regular hourly rate.
  - Calculates retirement deductions for eligible shifts and participants.
  - Computes the total pay and net pay after deductions.

- **File Handling:**
  - Stores hours worked data in a persistent text file.
  - Reads and writes data to the file for record-keeping and processing.

- **Output:**
  - Displays detailed payroll information to the user, including regular pay, overtime pay, total pay, retirement deductions, and net pay.
  - Ensures accurate reflection of user input in the stored file.

Program Architecture

- **Data Flow:**
  - **Input Stage:**
    - User inputs hours worked and shift number.
    - User decides on retirement plan participation for shifts 2 and 3.
  - **Processing Stage:**

        - Validate inputs.
        - Calculate payroll details based on input.
    - **Output Stage:**
        - Display payroll details.
        - Write the hours worked to the file.

Interfaces:

- **User Interface:**
    - Console-based input and output for interaction with the user.
    - Prompts for hours worked, shift number, and retirement plan participation.
- **File Interface:**
    - Interaction with hours_worked.txt for reading and writing hours worked.

b)

Code Layout

Main Class

```java
package andrew_payroll;

import java.util.*;
import java.io.*;

public class Andrew_Payroll {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        ArrayList<Employee> employees = new ArrayList<>();

        try {
            RandomAccessFile file = new RandomAccessFile("hours_worked.txt", "rw");

            System.out.println("Please enter how many hours you worked:");
            int hoursWorked = sc.nextInt();
            if (hoursWorked < 0) {
                System.out.println("Hours worked cannot be negative");
                return;
            }
```

```java
            System.out.println("Please enter your shift number:");
            int shiftNumber = sc.nextInt();
            if (shiftNumber < 1 || shiftNumber > 3) {
                System.out.println("Invalid shift number. Please enter 1, 2, or 3.");
                return;
            }

            boolean retirementPlan = false;

            if (shiftNumber == 2 || shiftNumber == 3) {
                System.out.println("Do you wish to participate in the retirement plan? (yes/no)");
                String choice = sc.next();
                if ("yes".equalsIgnoreCase(choice)) {
                    retirementPlan = true;
                }
            }
```

```java
            Employee employee = new Employee(hoursWorked, shiftNumber, retirementPlan);
            employees.add(employee);
            employee.calculatePay();

            displayEmployeeDetails(employee);

            file.seek(file.length());
            file.writeBytes(hoursWorked + "\n");

            file.close();
        } catch (IOException e) {
            System.out.println("An error occurred while handling the file.");
        }
    }
```

```java
private static void displayEmployeeDetails(Employee employee) {
    System.out.println("Employee Details:");
    System.out.println("Shift Number: " + employee.getShiftNumber());
    System.out.println("Hours Worked: " + employee.getHoursWorked());
    System.out.println("Regular Pay: " + "R" + employee.getRegularPay());
    System.out.println("Overtime Pay: " + "R" + employee.getOvertimePay());
    System.out.println("Total Pay: " + "R" + employee.getTotalPay());
    System.out.println("Retirement Deduction: " + "R" + employee.getRetirementDeduction());
    System.out.println("Net Pay: " + "R" + employee.getNetPay());
  }
}
```

- The main class uses consistent space, tabs and follows naming conventions for variables, functions and classes and is organised with clear separation.

Employee Class

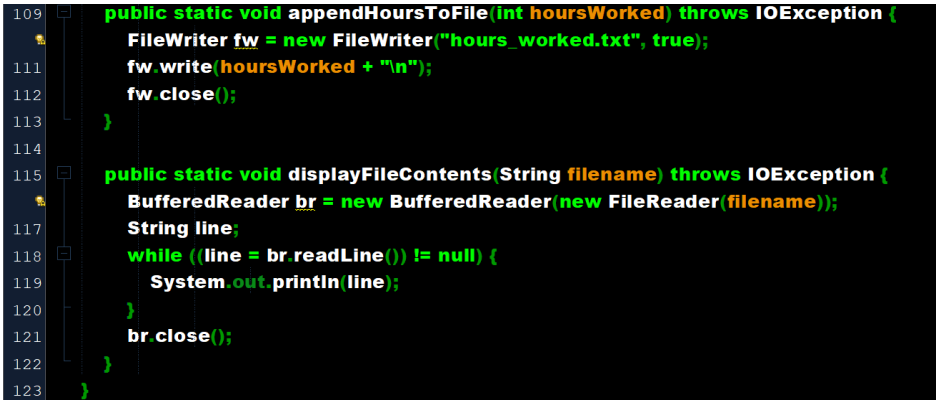
```java
package andrew_payroll;

import java.io.*;

public class Employee {

    private int hoursWorked;
    private int shiftNumber;
    private boolean hasRetirementPlan;
    private double regularPay;
    private double overtimePay;
    private double totalPay;
    private double retirementDeduction;
    private double netPay;
    private boolean validShift;

    public Employee(int hoursWorked, int shiftNumber, boolean hasRetirementPlan) {
        this.hoursWorked = hoursWorked;
        this.shiftNumber = shiftNumber;
        this.hasRetirementPlan = hasRetirementPlan;
        this.validShift = shiftNumber >= 1 && shiftNumber <= 3;
    }

```

```
24    public void calculatePay() {
25        if (hoursWorked < 0) {
26            throw new IllegalArgumentException("Hours worked cannot be negative.");
27        }
28
29        if (!validShift) {
30            regularPay = 0;
31            overtimePay = 0;
32            totalPay = 0;
33            retirementDeduction = 0;
34            netPay = 0;
35            return;
36        }
37
38        double hourlyPayRate;
          switch (shiftNumber) {
40            case 1:
41                hourlyPayRate = 50;
42                break;
43            case 2:
44                hourlyPayRate = 70;
45                break;
```

```
46            case 3:
47                hourlyPayRate = 90;
48                break;
49            default:
50                hourlyPayRate = 0;
51                break;
52        }
53
54        if (hoursWorked > 40) {
55            regularPay = 40 * hourlyPayRate;
56            overtimePay = (hoursWorked - 40) * hourlyPayRate * 1.5;
57        } else {
58            regularPay = hoursWorked * hourlyPayRate;
59            overtimePay = 0;
60        }
61
62        totalPay = regularPay + overtimePay;
63
```

```
64      if (shiftNumber != 1 && hasRetirementPlan) {
65          retirementDeduction = 0.05 * totalPay;
66      } else {
67          retirementDeduction = 0;
68      }
69
70      netPay = totalPay - retirementDeduction;
71  }
72
73  public int getHoursWorked() {
74      return hoursWorked;
75  }
76
77  public int getShiftNumber() {
78      return shiftNumber;
79  }
80
81  public boolean hasRetirementPlan() {
82      return hasRetirementPlan;
83  }
84
```

```
85  public double getRegularPay() {
86      return regularPay;
87  }
88
89  public double getOvertimePay() {
90      return overtimePay;
91  }
92
93  public double getTotalPay() {
94      return totalPay;
95  }
96
97  public double getRetirementDeduction() {
98      return retirementDeduction;
99  }
100
101 public double getNetPay() {
102     return netPay;
103 }
104
105 public boolean isValidShift() {
106     return validShift;
107 }
```



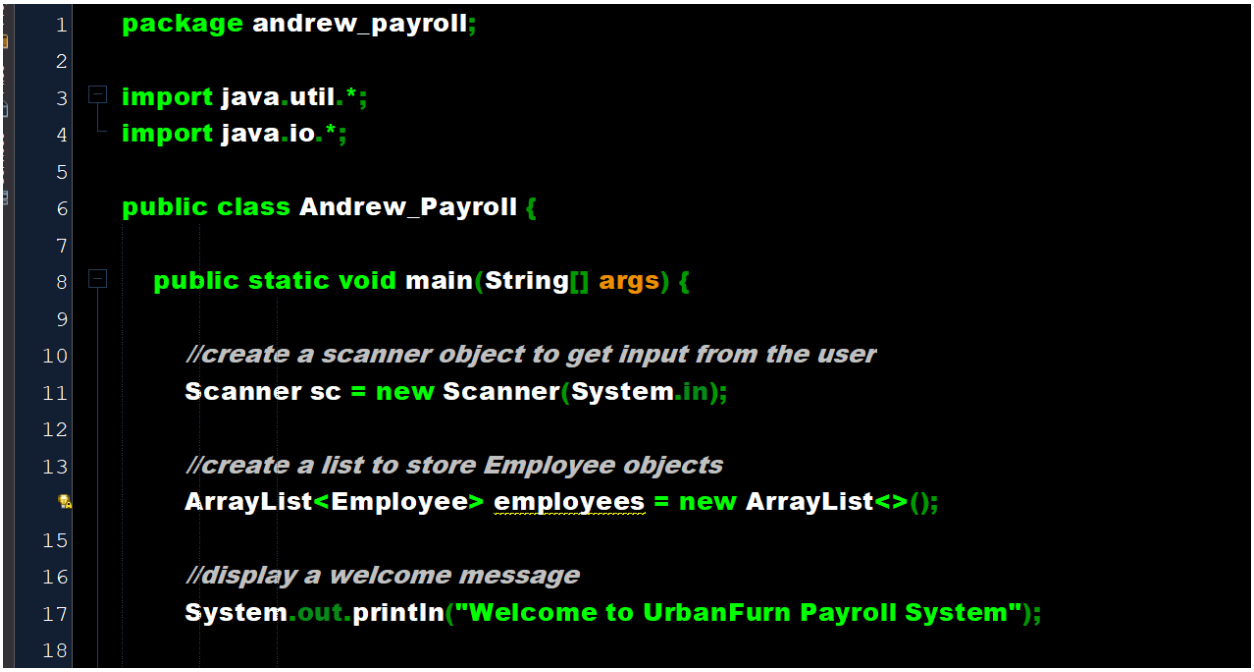
```
109     public static void appendHoursToFile(int hoursWorked) throws IOException {
            FileWriter fw = new FileWriter("hours_worked.txt", true);
111         fw.write(hoursWorked + "\n");
112         fw.close();
113     }
114
115     public static void displayFileContents(String filename) throws IOException {
            BufferedReader br = new BufferedReader(new FileReader(filename));
117         String line;
118         while ((line = br.readLine()) != null) {
119             System.out.println(line);
120         }
121         br.close();
122     }
123 }
```

- The employee class uses consistent space, tabs and follows naming conventions for variables, functions and classes and is organised with clear separation.

c)

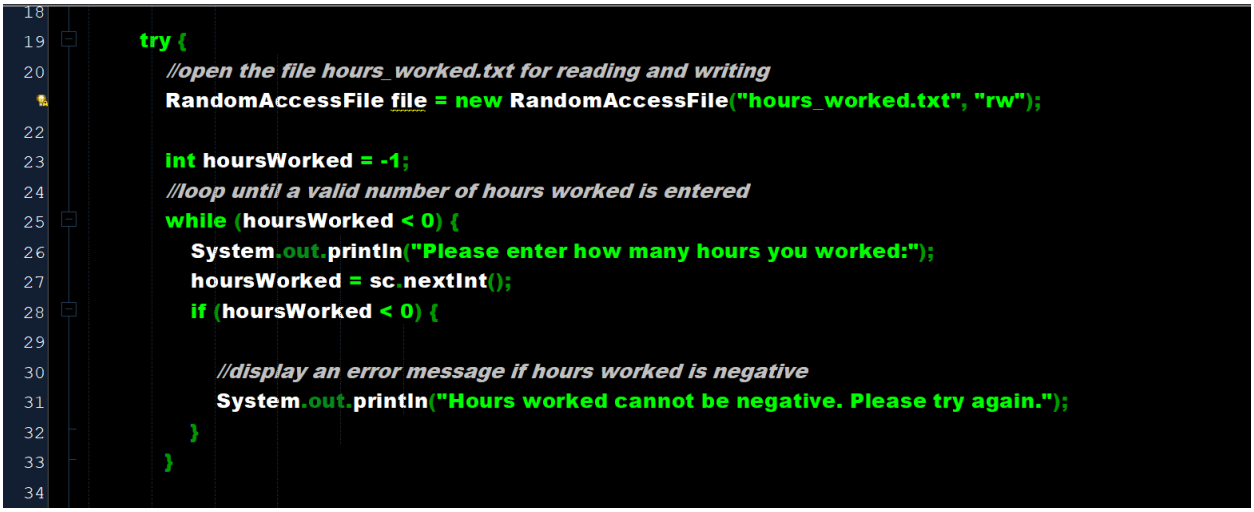
Internal Documentation:

Main Class



```
1  package andrew_payroll;
2
3  import java.util.*;
4  import java.io.*;
5
6  public class Andrew_Payroll {
7
8      public static void main(String[] args) {
9
10         //create a scanner object to get input from the user
11         Scanner sc = new Scanner(System.in);
12
13         //create a list to store Employee objects
14         ArrayList<Employee> employees = new ArrayList<>();
15
16         //display a welcome message
17         System.out.println("Welcome to UrbanFurn Payroll System");
18
```

- Line 11 creates a new scanner object which allows for input from the user.
- Line 14 creates a list in order to store employee objects.
- Line 17 displays a welcome message that welcomes the user to the program.



```
18
19      try {
20          //open the file hours_worked.txt for reading and writing
            RandomAccessFile file = new RandomAccessFile("hours_worked.txt", "rw");
22
23          int hoursWorked = -1;
24          //loop until a valid number of hours worked is entered
25          while (hoursWorked < 0) {
26              System.out.println("Please enter how many hours you worked:");
27              hoursWorked = sc.nextInt();
28              if (hoursWorked < 0) {
29
30                  //display an error message if hours worked is negative
31                  System.out.println("Hours worked cannot be negative. Please try again.");
32              }
33          }
34
```

- Line 21 opens the text document hours_worked.txt for reading and writing.
- Line 24 loops the user until they enter a valid hours worked value if they enter an incorrect one.
- Line 31 displays an error message if the hours worked entered is negative

```
35          int shiftNumber = 0;
36
37          //loop until a valid shift number is entered
38          while (shiftNumber < 1 || shiftNumber > 3) {
39             System.out.println("Please enter your shift number:");
40             shiftNumber = sc.nextInt();
41             if (shiftNumber < 1 || shiftNumber > 3) {
42
43                //display error message if shift number is not valid
44                System.out.println("Invalid shift number. Please enter 1, 2, or 3.");
45             }
46          }
47
48          boolean retirementPlan = false;
49
```

- Line 37 uses a while loop that keeps on asking the user to enter a valid shift number should they enter the incorrect value.

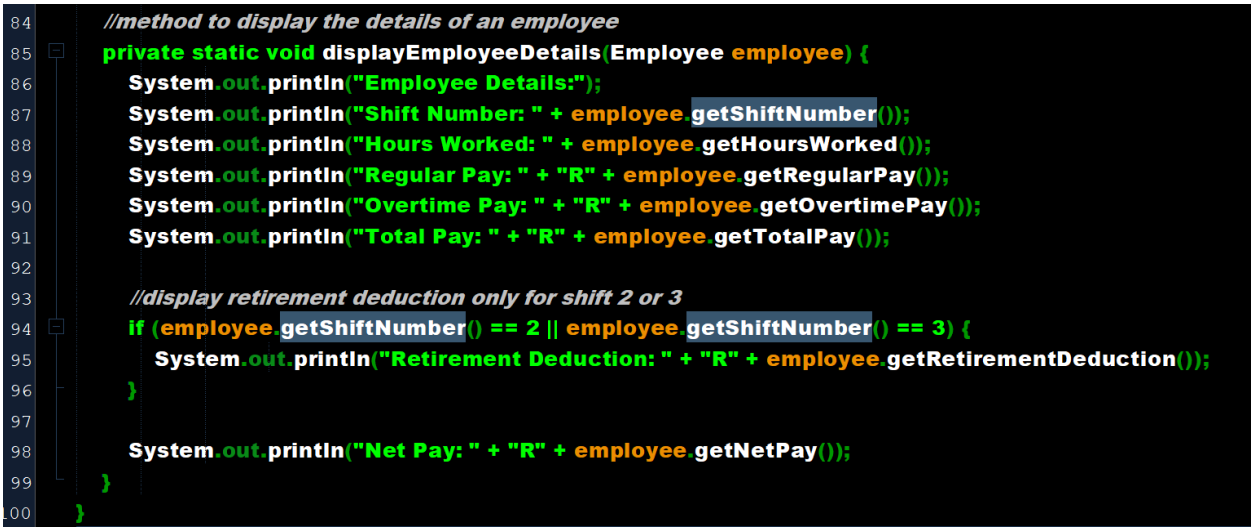
- Line 43 displays an error message if the shift number is not valid.

```
50          //ask the user if they want to participate in the retirement plan for shift 2 or 3
51          if (shiftNumber == 2 || shiftNumber == 3) {
52              System.out.println("Do you wish to participate in the retirement plan? (yes/no)");
53              String choice = sc.next();
54              if ("yes".equalsIgnoreCase(choice)) {
55                  //set retirement plan to true if user answers yes
56                  retirementPlan = true;
57              }
58          }
59
60          //create a new Employee object with the input values
61          Employee employee = new Employee(hoursWorked, shiftNumber, retirementPlan);
62
63          //add the employee to the list of employees
64          employees.add(employee);
65
66          //calculate the pay for the employee
67          employee.calculatePay();
68
```

- Lines 51 – 54 ask the user if they want to participate in the retirement plan shift 2 or 3.
- Line 56 sets the retirement plan to true if the user answers yes
- Line 61 creates a new employee object with the input values
- Line 61 adds the employee to the list of employees
- Line 67 calculates the pay for the employee.

```
69            //display the details of the employee
70            displayEmployeeDetails(employee);
71
72            //move the file pointer to the end of the file and write the hours worked to the file
73            file.seek(file.length());
74            file.writeBytes(hoursWorked + "\n");
75
76            //close the file
77            file.close();
78        } catch (IOException e) {
79            //display error message if an exception occurs while handling the file
80            System.out.println("An error occurred while handling the file.");
81        }
82    }
83
```

- Line 70 displays the details of the employee.
- Line 73 -74 moves the file pointer to the end and write the hours worked to the file.
- Line 77 closes the file.
- Line 80 displays an error message if an exception happens while handling the file.



```
84      //method to display the details of an employee
85      private static void displayEmployeeDetails(Employee employee) {
86          System.out.println("Employee Details:");
87          System.out.println("Shift Number: " + employee.getShiftNumber());
88          System.out.println("Hours Worked: " + employee.getHoursWorked());
89          System.out.println("Regular Pay: " + "R" + employee.getRegularPay());
90          System.out.println("Overtime Pay: " + "R" + employee.getOvertimePay());
91          System.out.println("Total Pay: " + "R" + employee.getTotalPay());
92
93          //display retirement deduction only for shift 2 or 3
94          if (employee.getShiftNumber() == 2 || employee.getShiftNumber() == 3) {
95              System.out.println("Retirement Deduction: " + "R" + employee.getRetirementDeduction());
96          }
97
98          System.out.println("Net Pay: " + "R" + employee.getNetPay());
99      }
100 }
```

- Line 85 creates a method to display the details of an employee.
- Line 93 displays retirement deduction for only shift 2 and shift 3.

Employee Class

```
1   package andrew_payroll;
2
3   import java.io.*;
4
5   public class Employee {
6
7       //attributes to store employee details
8       private int hoursWorked;
9       private int shiftNumber;
10      private boolean hasRetirementPlan;
11      private double regularPay;
12      private double overtimePay;
13      private double totalPay;
14      private double retirementDeduction;
15      private double netPay;
16      private boolean validShift;
17
```

- Line 3 imports the java.io library
- Line 8 – 16 are attributes that are declared to store employee details.

```
18      //constructor to initialize employee attributes
19      public Employee(int hoursWorked, int shiftNumber, boolean hasRetirementPlan) {
20          this.hoursWorked = hoursWorked;
21          this.shiftNumber = shiftNumber;
22          this.hasRetirementPlan = hasRetirementPlan;
23          //check if shift number is valid
24          this.validShift = shiftNumber >= 1 && shiftNumber <= 3;
25      }
26
27      //method to calculate the pay based on hours worked and shift number
28      public void calculatePay() {
29          //check for negative hours worked
30          if (hoursWorked < 0) {
31              throw new IllegalArgumentException("Hours worked cannot be negative.");
32          }
33
```

- Line 19 – 24 is a constructor that initializes employee attributes and checks whether shift number is valid or not.
- Line 28 creates a method to calculate the pay based on hours worked and shift number.
- Line 30 checks whether hours worked are negative

```
34          //if the shift number is invalid, set all pay attributes to zero
35          if (!validShift) {
36              regularPay = 0;
37              overtimePay = 0;
38              totalPay = 0;
39              retirementDeduction = 0;
40              netPay = 0;
41              return;
42          }
43
```

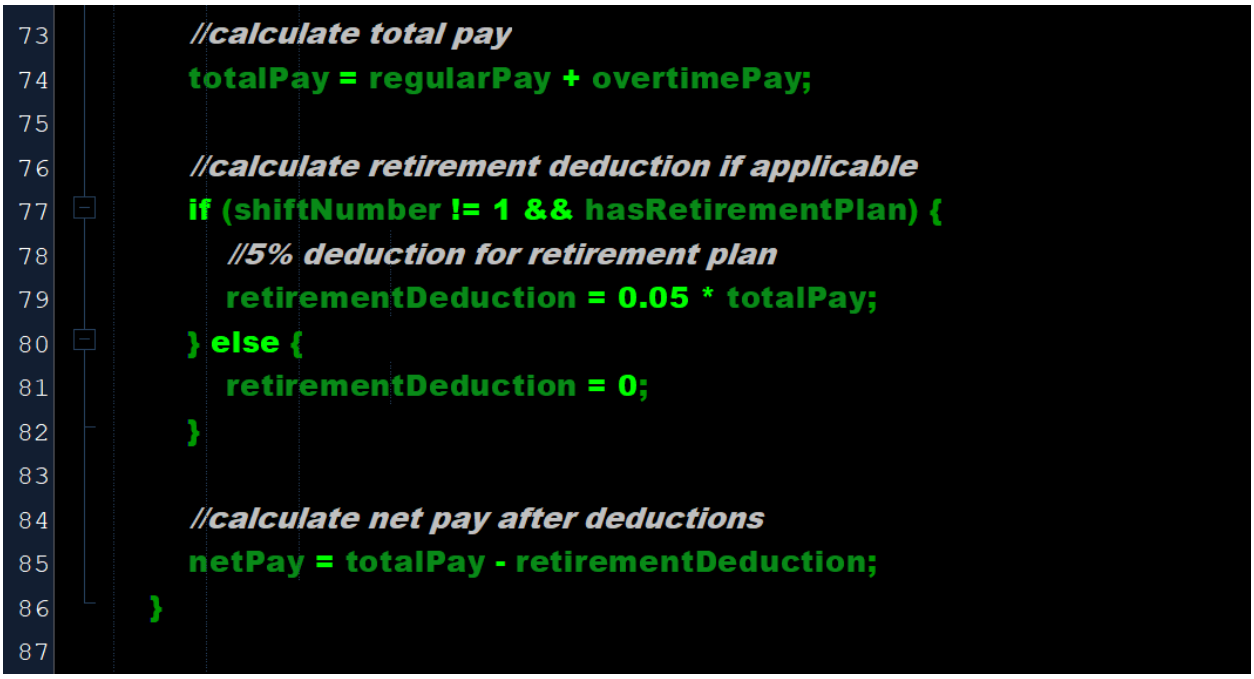
- Line 35 – 41 checks if the shift number is invaliff then all the attributes are set to 0.

```
44          //determine hourly pay rate based on the shift number
45          double hourlyPayRate;
            switch (shiftNumber) {
47              case 1:
48                  hourlyPayRate = 50;
49                  break;
50              case 2:
51                  hourlyPayRate = 70;
52                  break;
53              case 3:
54                  hourlyPayRate = 90;
55                  break;
56              default:
57                  hourlyPayRate = 0;
58                  break;
59          }
60
```

- Line 45 to 58 determines the hourly rate that is based on the employee's shift number.

```
61          //calculate regular and overtime pay
62          if (hoursWorked > 40) {
63              //regular pay for 40 hours
64              regularPay = 40 * hourlyPayRate;
65              //overtime pay for hours over 40
66              overtimePay = (hoursWorked - 40) * hourlyPayRate * 1.5;
67          } else {
68              //regular pay for hours worked
69              regularPay = hoursWorked * hourlyPayRate;
70              overtimePay = 0;
71          }
72
73          //calculate total pay
74          totalPay = regularPay + overtimePay;
75
```

- Line 62 calculates regular and overtime pay based on the employee's worked hours.
- Line 64 shows regular pay for 40 hours.
- Line 65 shows overtime pay for hours over 40.
- Line 69 to 70 is the regular for hours worked that week.



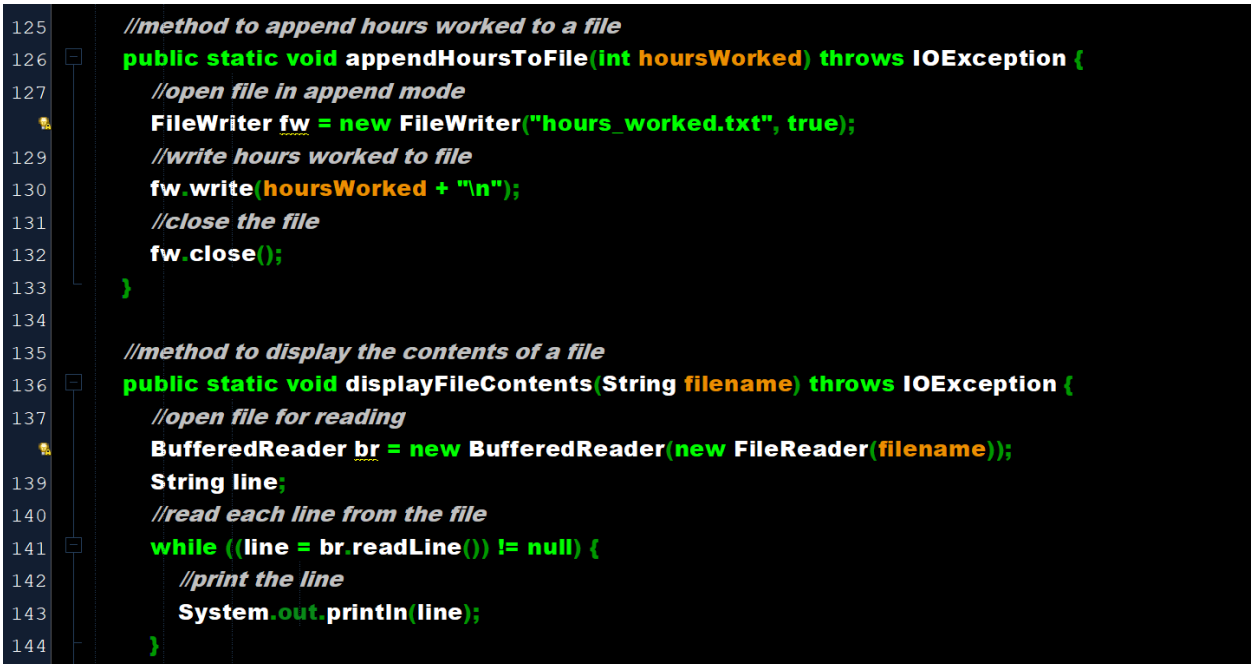
```
73          //calculate total pay
74          totalPay = regularPay + overtimePay;
75
76          //calculate retirement deduction if applicable
77          if (shiftNumber != 1 && hasRetirementPlan) {
78              //5% deduction for retirement plan
79              retirementDeduction = 0.05 * totalPay;
80          } else {
81              retirementDeduction = 0;
82          }
83
84          //calculate net pay after deductions
85          netPay = totalPay - retirementDeduction;
86      }
87
```

- Line 74 calculates the total pay.
- Line 77 calculates the retirement deduction if the user chooses to opt in.
- Line 79 shows that 5% is deduction from net pay.
- Line 85 calculates the net pay after all deductions.

```
88      //getter methods to access employee details
89      public int getHoursWorked() {
90          return hoursWorked;
91      }
92
93      public int getShiftNumber() {
94          return shiftNumber;
95      }
96
97      public boolean hasRetirementPlan() {
98          return hasRetirementPlan;
99      }
100
101     public double getRegularPay() {
102         return regularPay;
103     }
104
105     public double getOvertimePay() {
106         return overtimePay;
107     }
108
```

```
109     public double getTotalPay() {
110         return totalPay;
111     }
112
113     public double getRetirementDeduction() {
114         return retirementDeduction;
115     }
116
117     public double getNetPay() {
118         return netPay;
119     }
120
121     public boolean isValidShift() {
122         return validShift;
123     }
124
```

- Line 89 to 122 creates getter methods to be able to access employee details.



```
125     //method to append hours worked to a file
126     public static void appendHoursToFile(int hoursWorked) throws IOException {
127         //open file in append mode
            FileWriter fw = new FileWriter("hours_worked.txt", true);
129         //write hours worked to file
130         fw.write(hoursWorked + "\n");
131         //close the file
132         fw.close();
133     }
134
135     //method to display the contents of a file
136     public static void displayFileContents(String filename) throws IOException {
137         //open file for reading
            BufferedReader br = new BufferedReader(new FileReader(filename));
139         String line;
140         //read each line from the file
141         while ((line = br.readLine()) != null) {
142             //print the line
143             System.out.println(line);
144         }
```

- Line 126 creates a method that appends hours worked to a file.
- Line 126 opens the file in append mode.
- Line 129 writes the hours worked to a file.
- Line 135 creates a method to display the contents of the file.
- Line 138 opens the file for reading.
- Line 141 reads each line from the file.
- Line 143 prints the line.