

# Computer Program Solutions

## QUESTION 1

Ethan Van Rensburg – System Development  
Learnership Candidate

*18 – 23 June 2024*

## Contents

a) Problem Description:.....	1
b) Research Integration: .....	2
1. Data Needed by the Payroll System: .....	2
2. Important Functions for the Payroll Procedure: .....	2
c) Viability Evaluation: .....	3
d) Evaluating Economics: .....	4
e) Solution Selection .....	5
<b>Solution 1: Multilevel Inheritance and Objects</b> .....	5
<b>Solution 2: Single Main Class Approach</b> .....	7
<b>Factors:</b> .....	10
<i>Development cost:</i> .....	10
<i>Time and Resources:</i> .....	10
<i>Return on Investment (ROI):</i> .....	11
<i>Budget constraints:</i> .....	11
<b>Final Solution:</b> .....	11

## a) Problem Description:

The program was requested for end user satisfaction and delivery, with regards to its financial domain. Before the program was coded up, UrbanFurn faced confusion towards their payroll management and delivering accurate results to their workforce due to technical issues regarding the recording of work hours. This program now aims to eliminate this issue by asking for the worker's shift, then asking for their number of hours, this will then be taken and, based on your shift and pay, will produce all the required details and results.

The first shift will take in the number of hours, and depending on if they are normal or overtime, will produce the needed results, such as regular pay, overtime, total of regular and overtime and the net pay.

The second shift will take in the number of hours, and depending on if they are normal or overtime, will produce suitable results like shift one. Shift two will also prompt the end user for a retirement plan, if the user says yes, they will have a retirement plan deduction and their net pay will be affected. If they say no, they will receive no deductions and their net pay will not be affected.

The third shift follows the same format and execution of the second shift, which works with normal hours, overtime and a potential retirement plan.

## b) Research Integration:

### 1. Data Needed by the Payroll System:

**Working Shifts:** The system has to keep track of every employee working shifts, including the starting and ending schedules. This data ensures the accurate recording of work shifts.

**Hourly Pay Rate:** It is critical to monitor the hourly pay rate of every employee. It makes it possible to accurately calculate their salary based on the number of hours they put in.

### 2. Important Functions for the Payroll Procedure:

**Inputs:** Create a function that measures the total hours worked compared to shift input in order to calculate shifts and hours. For payroll calculations to be accurate, many factors are required.

**How to Determine Payroll:** Create a function that uses the number of hours worked and the hourly pay rate to determine the weekly salary. This ensures that employees are paid accurately and correctly.

**Outputs:** After the inputs and calculations are completed, the results must be displayed via print methods and code to show the suitable pay for the workers.

**Recording and Compliance:** Include recording features that process the information given by end users.

### c) Viability Evaluation:

The request was defined as a necessary action, but the team needed to consider the value and impact of this project. The current issue that was faced was merely inaccurate calculations, which the team were able to solve and evaluate into fruition. From the display example in the research, it was proven that the capabilities for this solution are not out of reach and can be achieved.

Based on the current tools at hand, these being the Apache NetBeans IDE, version 22, and the Java Development Kit updated to version 22, the problem can be coded and solved within a java-based development environment.

## d) Evaluating Economics:

First, development costs include the initial costs associated with designing, coding and assessing the payroll system. These costs include developer salaries, software licenses, and any third-party tools or libraries used. In addition, the ongoing costs of system maintenance, updates, bug fixes and compliance adjustments add to the overall cost.

### **Costs:**

The majority of costs will initially be within regards to the programming team and development process. The costs of production will be estimated with programmer salaries within creating, testing and documenting the program. Approximately, the costs will range to about **R308** per hour for coding, which is **R2464** a day on coding and testing, then **R267** per hour for technical writing and professional documentation, which is **R6408** within the last remaining 3 days of the schedule.

Therefore, the final costs will range around **R8000 to R10000**, approximately, if not more based on operational tasks and timing.

On the other hand, the benefits of implementing a payroll program are significant:

1. **Efficiency:** The automated system streamlines payroll, reduces manual work and minimizes errors. Such efficiency means saving time for HR staff.
2. **Accuracy:** Automatic calculations greatly reduce the risk of human error, ensuring accurate pay for workers.
3. **Compliance:** The system manages calculations and potential deductions based on the shifts.
4. **Employee satisfaction:** Accurate pay improves employee productivity.
5. **Long-term savings:** Over time, the program's efficiency and accuracy help save costs.

## e) Solution Selection

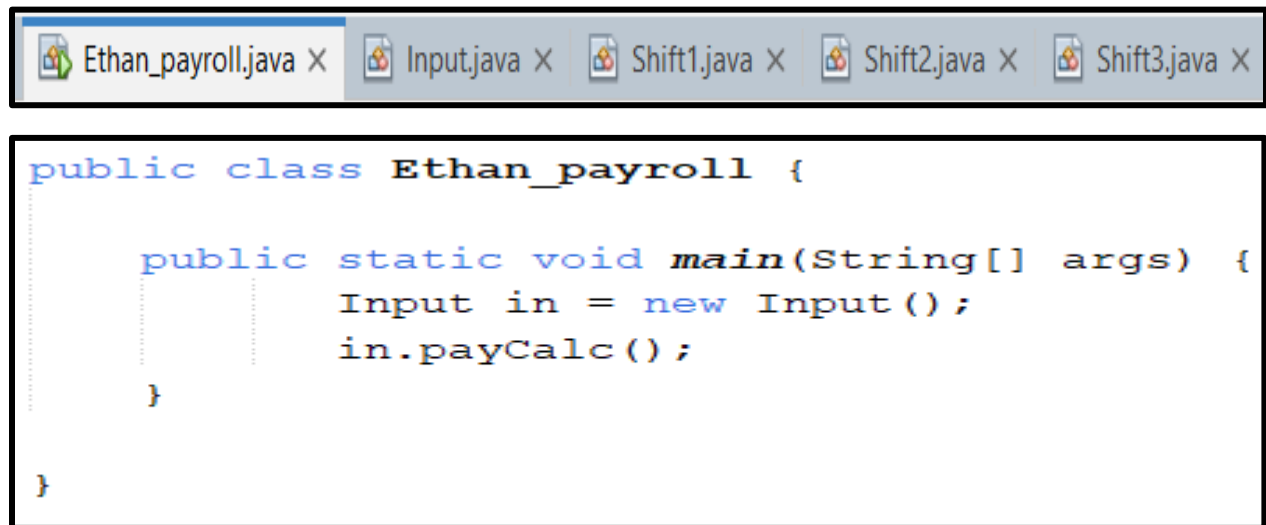
### Solution 1: Multilevel Inheritance and Objects

#### Implementation Details:

1. The main class calls the input class and its functions to manage prompts and results.
2. The input class uses different shift classes (day shift, night shift, morning shift). These categories contain specific functions related to each type of exchange.
3. Inheritance ensures efficient sharing of methods between exchange classes, promoting code reusability.

#### Advantages:

- **Code efficiency:** By inheriting methods from a common base class, redundancy is avoided, and maintainability is improved.
- **Structured Design:** A hierarchy of classes ensures a clear organization of functions.



```
public class Ethan_payroll {  
  
    public static void main(String[] args) {  
        Input in = new Input();  
        in.payCalc();  
    }  
  
}
```

```

public class Input {

    public void payCalc() {
        Scanner scan = new Scanner(System.in);
        Shift1 one = new Shift1();
        Shift2 two = new Shift2();
        Shift3 three = new Shift3();
        System.out.println("""
            Please select your assigned shift:
            1 - Morning Shift
            2 - Day Shift
            3 - Night Shift""");
        int shifChoice = scan.nextInt();
        switch (shifChoice) {
            case 1 ->
                one.payCalc();
            case 2 ->
                two.payCalc();
            case 3 ->
                three.payCalc();
            default ->
                System.out.println("Invalid input");
        }
    }
}

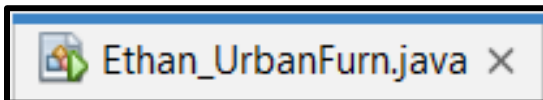
```



## Solution 2: Single Main Class Approach

### Implementation Details:

1. The main class performs the task directly using the methods of the Input and shift classes (similar to solution 1).
2. This approach results in a more compact codebase, where all related methods are stacked into the main class.
3. Added benefits: The implemented design and code brings in expansion. For example, read and write functions can be added to save data (such as saving employee hours to a text file).



```
public class Ethan_UrbanFurn {  
  
    public static void main(String[] args) {  
        Scanner scan = new Scanner(System.in);  
        Ethan_UrbanFurn plan = new Ethan_UrbanFurn();  
  
        int hours;  
        double hourly1 = 50;  
        double hourly2 = 70;  
        double hourly3 = 90;  
  
        System.out.println("""  
                                Please select your assigned shift:  
                                1 - Morning Shift  
                                2 - Day Shift  
                                3 - Night Shift""");  
        int shifChoice = Integer.parseInt(scan.nextLine());  
    }  
}
```

```

switch (shifChoice) {

    case 1 -> {
        System.out.println("How many hours do you work in a week?");
        hours = scan.nextInt();
        try {
            String content = ("Hours Worked: " + hours + "\n");
            File fil = new File("C:\\Users\\ETHAN.V\\Documents\\NetBeansProjects\\Ethan_UrbanFurn\\Hours.txt");
            FileWriter fw = new FileWriter((fil.getAbsolutePath()), true);
            BufferedWriter bw = new BufferedWriter(fw);
            bw.write(content);
            bw.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
        if (hours <= 40 && hours !=0) {
            int weekPay = hours * 50;
            System.out.println();
            System.out.println("Payroll:");
            System.out.println("-----");
            System.out.println("Hours Worked: " + hours);
            System.out.println("Shift: Morning");
            System.out.println("Hourly Pay Rate: R" + hourly1);
            System.out.println("Regular Pay: R" + weekPay);
            System.out.println("Overtime Pay: R0.00");
            System.out.println("Total of Regular and Overtime: R" + weekPay);
            System.out.println("Retirment Deduction: R0.00");
            System.out.println("Net Pay: R" + weekPay);
        } else if (hours > 40 && hours !=0) {
            int overTime = hours - 40;
            int calcOver = overTime * (int) (50 * 1.5);
            float weekOver = calcOver + (40 * 50);
            System.out.println();
            System.out.println("Payroll:");
            System.out.println("-----");
            System.out.println("Hours Worked: " + hours);
            System.out.println("Shift: Morning");
            System.out.println("Hourly Pay Rate: R" + hourly1);
            System.out.println("Regular Pay: R" + (40 * 50));
            System.out.println("Overtime Pay: R" + calcOver);
            System.out.println("Total of Regular and Overtime: R" + weekOver);
            System.out.println("Retirment Deduction: R0.00");
            System.out.println("Net Pay: R" + weekOver);
        } else {
            System.out.println("Invalid input");
        }
    }
}

```

```

case 2 -> {
    System.out.println("How many hours do you work in a week?");
    hours = scan.nextInt();
    try {
        String content = ("Hours Worked: " + hours + "\n");
        File fil = new File("C:\\Users\\ETHAN.V\\Documents\\NetBeansProjects\\Ethan_UrbanFurn\\Hours.txt");
        FileWriter fw = new FileWriter((fil.getAbsolutePath()), true);
        BufferedWriter bw = new BufferedWriter(fw);
        bw.write(content);
        bw.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
    if (hours <= 40 && hours !=0) {
        int weekPay = hours * 70;
        plan.retire2(weekPay, hours, hourly2);
    } else if (hours > 40 && hours !=0) {
        int overTime = hours - 40;
        double calcOver = overTime * (70 * 1.5);
        double weekOver = calcOver + (40 * 70);
        plan.retireOver2(weekOver, hours, calcOver, hourly2);
    } else {
        System.out.println("Invalid input");
    }
}
}

```

```

case 3 -> {
    System.out.println("How many hours do you work in a week?");
    hours = scan.nextInt();
    try {
        String content = ("Hours Worked: " + hours + "\n");
        File fil = new File("C:\\Users\\ETHAN.V\\Documents\\NetBeansProjects\\Ethan_UrbanFurn\\Hours.txt");
        FileWriter fw = new FileWriter((fil.getAbsolutePath()), true);
        BufferedWriter bw = new BufferedWriter(fw);
        bw.write(content);
        bw.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
    if (hours <= 40 && hours !=0) {
        int weekPay = hours * 90;
        plan.retire3(weekPay, hours, hourly3);
    } else if (hours > 40 && hours !=0) {
        int overTime = hours - 40;
        double calcOver = overTime * (90 * 1.5);
        double weekOver = calcOver + (40 * 90);
        plan.retireOver3(weekOver, hours, calcOver, hourly3);
    } else {
        System.out.println("Invalid input");
    }
}
default ->
    System.out.println("Invalid input");
}
}

```

## Factors:

### *Development cost:*

#### **Solution 1:**

1. **Developer Salaries:** Reasonable, as it requires designing and implementing several classes. The costs will be calculated across the 4 days of the allocated salaries of the development team. [\(Click\)](#)
2. **Software Licenses:** Minimal as it is based mostly on standard Java libraries.
3. **Third Party Services:** Not important, assuming no external services are used.

#### **Solution 2:**

1. **Developer Salaries:** Similar to solution 1 as it also requires coding and testing. [\(Click\)](#)
2. **Software Licenses:** Minimal, Same as Solution 1.
3. **Third Party Services:** Same as Solution 1.

### *Time and Resources:*

#### **Solution 1:**

1. **Development Time:** Longer due to multi-class design and testing.
2. **Project Management:** Requires coordination of applications between classes.
3. **Testing:** More work needs to be done to ensure that class-to-class communication works properly.
4. **Documentation:** Additional class hierarchy documentation.

#### **Solution 2:**

1. **Development Time:** Shorter because it focuses on one core class.
2. **Project Management:** Simpler, fewer components.
3. **Testing:** Less complex due to a compact code base.
4. **Documentation:** Still necessary, but less extensive.

### *Return on Investment (ROI):*

#### **Solution 1:**

1. **Efficiency Advantage:** Similar to Solution 2 because both automate payroll.
2. **Error Reduction:** Comparable as both aim for accuracy.
3. **Employee Satisfaction:** Indirectly affects ROI, consistently across both solutions.

#### **Solution 2:**

1. Offers the same benefits as solution 1.

### *Budget constraints:*

#### **Solution 1:**

1. **Initial Budget:** Estimated R10,000+, potentially higher due to additional development time. ([Click](#))
2. **Long-term Maintenance:** Similar to solution 2.

#### **Solution 2:**

1. **Initial Budget:** Estimated R10,000+, potentially lower due to shorter development time. ([Click](#))
2. **Long-term Maintenance:** Comparable to solution 1.

### **Final Solution:**

Considering the factors, both solutions have similar costs and benefits. However, **Solution 2** stands out for its compact design, easy expansion and read/write functionality. Therefore, UrbanFurn should continue with solution 2 in its payroll software.