“Plan Documentation”

Table of Contents

a) Problem Description.

For several months, Mr. Singh, the factory's payroll supervisor, has been struggling with discrepancies in overtime calculations. Despite the accurate record-keeping and the efforts of his dedicated team, errors in overtime pay have repeatedly surfaced triggering frustration and dissatisfaction among the factory workers. Employees from all shifts such as morning, afternoon, and night have reported a number of instances where their hard-earned overtime hours were not accurately reflected in their pay checks. These discrepancies have led to heated discussions during break times and after-shift meetings, making the growing tension among the workforces even worse.

b) Research Integration.

User Interface:

- The program will prompt each worker to input their hours worked for the week.
- Workers will select their shift number and are paid one of three hourly rates depending on their shift: first shift is R50 per hour, second shift is R70 per hour, and third shift is R90 per hour.
- Each factory worker might work any number of hours per week, any hours greater than 40 are paid at one and one-half times the usual rate.
- In addition, second and third-shift workers can elect to participate in the retirement plan, for which 5 percent of the worker's gross pay is deducted from the pay checks.
- A prompt will ask if the worker wishes to participate in the retirement plan (yes/no).

Calculation Logic:

- Regular pay will be calculated based on standard hourly rates.
- Overtime pay will be calculated for hours exceeding 40 per week, with rates adjusted per shift (1.5 times the usual rate).
- Retirement plan contributions (5%), if opted in, will be deducted from the gross pay.

Output and Transparency:

Write a program that prompts the user for hours worked and the shift number; if the shift number is 2 or 3, prompt the user to enter the worker's choice on whether they want to participate in the retirement plan. The display will provide a breakdown of the following:

- the hours worked
- the shift
- the hourly pay rate
- the regular pay
- overtime pay
- the total of regular and overtime pay
- the retirement deduction, if any, and

- the net pay

Storage:

- Saving the number of hours entered each time the program is run.

c) Viability Evaluation

UrbanFurn is experiencing discrepancies in overtime calculations, causing dissatisfaction among workers. The program needs to accurately calculate the regular and overtime pay for workers across different shifts and handle retirement plan contributions for second and third-shift workers.

Solution 1: Using a Console-based interface

This solution uses the Scanner class to create a console-based **interface** for the payroll program, making it simple and easy to understand.

Functions:

- `while-loops` to validate the employee's input and if invalid, allowing the employee to try again.
- `if` statements to allow only valid inputs and is also used to handle retirement plan selection of the program.
- `Random AccessFile` allowing the number of hours worked and the other additional payment breakdown to be stored in and read from a text file.

Costs:

- Development Time: Creating a console-based **interface** with the Scanner class requires less time and effort compared to simpler graphic-based approaches.
- Complexity: Integrating input validation directly within console-based interface can become less complex, leading to simple-to-maintain code.
- Cost: Approximately R15,000.00 with a time frame of 1-2 months due to the simplicity of the implementation.

Solution 2: Using Methods for Encapsulation within a Console-based interface

This solution uses methods to encapsulate different functionalities within the Console-based interface, enhancing code readability and maintainability. It involves setting up methods to handle various tasks such as running the calculation, validating the employee's input, and displaying the payment breakdowns.

Functions Used:

- `payrollCalculation` method to calculate the regular pay, overtime pay, total pay, retirement deduction (if any) and net pay.
- `validateInput` method to check if the employee's input is within the valid range.

- `displayPayment` method to show the whole breakdown of the payment to the employee.
- `Random AccessFile` allowing the number of hours worked and the other additional payment breakdown to be stored in and read from a text file.

Benefits:

- Modular Code: The code is easier to read and maintain, with each method encapsulating specific functionalities.
- Ease of Extension: Adding new features or modifying existing ones is straightforward as changes are confined to specific methods.
- Reusability: Methods like `displayPayment` can be reused, reducing redundancy in code.

Costs:

- Setup Time: Setting up methods and ensuring they integrate well with Console-based interface requires more time and effort.
- Cost: Approximately R20,000.00 with a time frame of 2-3 months due to the complexity of method-based implementation.

d) Evaluating the economic viability.

Solution 1: Console-based Interface

Cost:

- R15,000.00

Time Frame:

- 1-2 months

Benefits:

- This allows less development time and cost.
- This allows simple and straightforward implementation of the different functionalities.
- This allows the program to be easier to maintain in the short term.

Drawbacks:

- Less method used in the code, which could be harder to extend and maintain in the long term.
- The basic user experience with a console-based interface.

Solution 2: Methods for Encapsulation within a Console-based Interface

Cost:

- R20,000.00

Time Frame:

- 2-3 months

Benefits:

- This allows more structure (methods) and maintainable code.
- This allows the program to be easier to extend and modify in the future.
- This allows the program to better handling of complex functionality through the use of encapsulation.

Drawbacks:

- This leads to higher development cost and time.
- This type of program is more complex to implement.

e) Solution Selection.

*Chosen Solution: Solution 1 (*Console-based interface*) without the use of method*.

- Based on the evaluation, developing a desktop application using a high-level, class-based, object-oriented programming language like Java (specifically the Scanner class) is most suitable.

Program Features:

User Interface:

- Scanner with text fields, and prompts.

Functionality:

- Ask the employee to input their hours worked.
- Allow the employee to select shift number.
- Calculate the regular pay based on standard hourly rates of each of the shifts.
- Calculate the overtime pay for hours that exceed 40 (excluding 40).
- Deduct 5% for retirement plan contributions if opted in.
- Display the detailed payment breakdown:
    - Their numbers of hours worked
    - Their shift number
    - The hourly pay rate
    - Their regular pay
    - Their overtime pay
    - Their total pay
    - Their retirement deduction (if any)
    - Their net pay

Performance:

- The program handles the employee's input and provide the feedback based on the input.
- The program validates the employee's input to make sure only valid integers are allowed and accepted.
- The program stores the payment breakdown in and read from a text file.

Maintainability:

- The program contains clear and well-documented code.

- The program contains comments that explain the flow and functionality of the overall program.
- The program contains modular design to make future enhancements easer to implement.