PLANNING DOCUMENTATION

Martin Rugani [Learnership - Curro Private College Cape Town - All Schools]

Contents

A) Problem Description

UrbanFurn faces discrepancies when calculating their employees' overtime calculations. Errors with employee overtime pay are constantly surfacing which has led to frustration among the workers of the UrbanFurn factory. There have been numerous complaints from employees about overtime not accurately reflected in their paychecks which has increased tension in the workplace. A meeting with management and the representatives of employees was held where they decided to contract developers to create a program that will:

- Be implemented in 4 days
- Accurately Calculate employee overtime and deductions
- Display the user's total income
- How much overtime they've earned
- Calculate total with overtime
- Calculate retirement Deductions
- Record the data into a text file
- Read the data from the text file

B) Research integration

The program will calculate a user's overtime and deductions, the program will also allow the user to register for a retirement plan that will be deducted from their total monthly income to do this the program will collect the following:

Input

1. Hours worked
2. Shift type (First, Second, Third)
3. Whether the user wants to opt into a retirement plan if they work the second or third shift

Processing

The program will use the user input to calculate the following:

1. Determine the user's hourly rates by checking the shift type
2. Calculate the total regular pay for the user
3. Determine if the user qualifies for overtime by checking if they've worked over 40 Hours
4. Calculate how many extra hours they've worked
5. Multiply their hourly rate by 1.5 to determine their overtime rate
6. Multiply their extra hours by their overtime rate to get their overtime pay
7. Add their overtime pay to their regular pay
8. If the user qualifies and opts into a retirement plan deduct 5% from their total
9. Calculate the user's net pay by deducting their retirement fund from their total

Output

After processing the program will display:

1. The user's hours worked
2. The user's shift type
3. The user's pay rate
4. Their Regular pay
5. Their overtime pay
6. The total of their regular and total pay added together
7. Retirement deduction if any
8. Their Net pay

C) Viability Evaluation

The viability of this project is measured by time to develop and the complexity of the code, the program has been specified to calculate different units of data which is related to employee income and deductions, most of these calculations are an automated process requiring at most 5 lines of code for each function to determine a returnable variable, this reduces the complexity of the code substantially making it fully feasible to complete development and testing within a day and at most 2 days, the learning curve and the onboarding process is expected to go smoothly due to the lack of user input required for calculations and outputs to be processed. The program can be made with current technology within it's constraints the program can calculate transactions and store hours for multiple users and can be run multiple times to record different instances of user data and transactions

This program will help with the payroll issues that UrbanFurn faces by automating their overtime calculation process, the program aims to produce accurate data quickly and efficiently to reduce discrepancies and frustration within the factory.

D) Economic Viability

Given the history of UrbanFurn's precise record-keeping in other aspects of the organization, I believe this program will benefit the company. The program will help Mr Singh accurately record data by automating the calculation process which reduces the time it will take to process payments for employees. The program is also aimed at improving UrbanFurn's technical and automated processes I believe improving these aspects of the company will help them integrate more employees into the workforce while reducing the load on the accounting team. Automating transactions will allow more employees to be added and accounted for with little to no discrepancies between them. This will also improve the reputation of management among employees which in turn will also improve Urbanfurn's reputation as a workplace within Cape town due to word of mouth and positive feedback from employees. The estimated cost of this program is R30 000, I believe the organization will profit from investing this program by ensuring they accurately process transactions and reduce complaints and backlash from employees which will improve productivity and lessen costs

In summary, I believe this program will further automate the transaction process which will allow for quicker and more payments to be handled, accurate transaction amounts will also ensure that the company doesn't lose more money than required and improve the reputation of the company as a whole.

E) Solution Selection

Solution 1

This solution aims to calculate the Overtime pay and deductions and write data to the file using randomAccessFile this method allows me to insert data anywhere in a textfile or database it also stores data as their type according to the program which can help prevent rounding errors and inaccurate translations,

This solution helps with record–keeping when data needs to be stored in a specific position in a file however this method may perform slower because it writes data types as they are in an unbuffered format.

1. Storing user input

```
public static void main(String[] args) {

    //Declaring variables that hold user input
    int Shift = 0;
    int Hoursworked = 0;
    //Declaring variables used to process transactions and cakculations
    double Rate;
    double Regularpay;
    double Overtimepay = 0;
    double Overtimepayrate;
    double total = 0;
    double retirementdeduction = 0;
    double netpay = 0;
    String Retirement = "";

    //Creating an object of scanner to collect user input
    Scanner scanner = new Scanner(System.in);

    //I ask the user if their using stored data or new data
    System.out.println("Which data are you using? \n 1)New Data \n 2)Stored Data");
```

2. Collecting hours worked

```
//I ask the user if their using stored data or new data
System.out.println("Which data are you using? \n 1)New Data \n 2)Stored Data");
int choice = Integer.parseInt(scanner.nextLine());

//if the user is entering new data the program will ask the user to enter their
//hours worked

if(choice == 1){

//I use a try statement to ensure that the user enters hours as a number
try {
    //Prompting the user to enter their hours worked
    System.out.println("Please enter you hours worked this week");
    Hoursworked = Integer.parseInt(scanner.nextLine());

} catch (NumberFormatException e) {
    System.out.println("Invalid! Please enter a numeric value");
    //The program will reprompt the user to enter their hours if it's invalid
    Hoursworked = Integer.parseInt(scanner.nextLine());
}
```

3. Reading the hours worked from the text file using RandomAccessFile

```
//if the user selects stored data the program will read the hours stored in the textfile
}else if(choice ==2){
  try{
      //I create a new instance of random access file in read mode "r"

      RandomAccessFile file = new RandomAccessFile("Hours.txt", "r");

      //I declare a variable that reads lines from the text file
      String line;

      //The program will read each line in the file as long as they aren't empty
      while((line = file.readLine()) != null){
          //I extract the line from the textfile until the semicolon at the end
          //of each line to seperate them and store each of them in an array
          String[]Hour =line.split(";");
          //Then i assign the read hour to the hours worked variable while trimming
          //any white space
          Hoursworked = Integer.parseInt(Hour[0].trim());

  }
```

4. Validating users shift type

```
try {

    System.out.println("Plese select your shift number (Enter the corresponding number) \n 1) First s
    Shift = Integer.parseInt(scanner.nextLine());

} catch (NumberFormatException e) {
    System.out.println("Invalid! Please enter the number corresponding to your shift");
    Shift = Integer.parseInt(scanner.nextLine());
}

//I create a boolean that checks if the user selects a valid shift
boolean validshift = true;

//if the user enters a number greater than 3 and less than 1 then it'll be assigned
//as invalid
if (Shift > 3) {
    validshift = false;
} else if (Shift < 1) {
    validshift = false;
```

```
//The program will reprompt the user to input their shift number again if it
//isn't valid
while (!validshift) {
    System.out.println("Invalid Shift selected, please enter the correct corresponding number");
    Shift = Integer.parseInt(scanner.nextLine());
}

//Setting hourly rates according to the employee shift
if (Shift == 1) {
    Rate = 50;

} else if (Shift == 2) {
    Rate = 70;
} else {
    Rate = 90;
}

//Asking the second and third shift workers if they want to opt in for a retirement plan
if (Shift > 1) {
    System.out.println("Would you like to opt in for a retirem
    Retirement = scanner.next();
```

5. Asking the user if they want to opt into a retirement plan and displaying input data

```
//Asking the second and third shift workers if they want to opt in for a retirement plan
if (Shift > 1) {
    System.out.println("Would you like to opt in for a retirement plan? (Yes/no");
    Retirement = scanner.next();
}

//Displaying the users pay rate, hours worked and shift
System.out.println("Hours Worked: " + Hoursworked + " Hours");
System.out.println("Shift: " + Shift);
System.out.println("Hourly pay Rate: R" + Rate);
```

6. Calculating the users' regular pay

```
System.out.println("Hourly pay Rate: R" + Rate);

//Calculating employees regular pay
Regularpay = Rate * Hoursworked;

//Setting the total variable and regular pay
total = Regularpay;
```

7. Calculating Overtime

```
//Setting the users Overtime if they work over 40 hours a week
if (Hoursworked > 40) {

    //If the employee exceeds 40 hours their regular pay will be calcul
    //using 40 hours
    Regularpay = Rate * 40;

    //Then I determine how many extra hours the employee worked for
    int overhours = Hoursworked - 40;

    //Determining the new pay rate for overtime workers
    Overtimepayrate = Rate * 1.5;

    //Calculating overtime pay by implementing the new rate in the forr
    //and extra hours
    Overtimepay = overhours * Overtimepayrate;

    //Calculating the total gross pay of employees by overwritting the
    //variable
    total = Regularpay + Overtimepay;
```

8. Calculating deductions and the user's total netpay

```
//Checking if the user opted in for a retirement plan
if (Retirement.equalsIgnoreCase("yes")) {

    //if the user is an overtime worker the retirement will be deducted from the total
    //of their regular pay and overtime
    retirementdeduction = total * 0.05;

    //The retirement deduction will be removed from the total of their regular and
    //overtime pay
    netpay = total - retirementdeduction;

    //Displaying the retirement deduction
    System.out.println("Retirement deduction: R" + retirementdeduction);
}

System.out.println("Netpay R" + netpay);
```

9. Storing the user's hours worked into a text file using random access file

```
//Writing the data to a text file
try {
    //I create a new instance of random access file called file in read-write mode
    //which means this method will read data then write it
    //"rw" means readwrite
    RandomAccessFile file = new RandomAccessFile("Hours.txt", "rw");

    //This moves the cursor in the text file to the end of the file
    //meaning we add new data after each other
    file.seek(file.length());

    //Using write bytes to add "hours worked" as single bytes in the text file
    //the label and data are both their own byte hence I use bytes insted of byte
    file.writeBytes(Hoursworked + ";\n");

    file.close();

    System.out.println("Your hours this week have been recorded");
} catch (IOException e) {
    System.out.println("Error");
}
```

Solution 2

This solution will calculate the overtime pay and deductions for employees just like the previous solution, but this program differentiates itself by storing the data in an external file this process is done using filewriter and bufferedwritter these methods write data directly to a textfile to assist with record however this solution can only write data as strings or characters so it converts data in the program which could cause rounding errors or data loss.

This Method generally performs faster since it writes data as strings, but it is best used when the user wants to enter streams of characters like names or positions or when the user wants to overwrite data.

1. Storing user input

```
public static void main(String[] args) {

    //Declaring variables that hold user input
    int Shift = 0;
    int Hoursworked = 0;
    //Declaring variables used to process transactions and cakculations
    double Rate;
    double Regularpay;
    double Overtimepay = 0;
    double Overtimepayrate;
    double total = 0;
    double retirementdeduction = 0;
    double netpay = 0;
    String Retirement = "";

    //Creating an object of scanner to collect user input
    Scanner scanner = new Scanner(System.in);

    //I ask the user if their using stored data or new data
    System.out.println("Which data are you using? \n 1)New Data \n 2)Stored Data");
```

2. Collecting hours worked

```
//I ask the user if their using stored data or new data
System.out.println("Which data are you using? \n 1)New Data \n 2)Stored Data");
int choice = Integer.parseInt(scanner.nextLine());

//if the user is entering new data the program will ask the user to enter their
//hours worked

if(choice == 1){

//I use a try statement to ensure that the user enters hours as a number
try {
    //Prompting the user to enter their hours worked
    System.out.println("Please enter you hours worked this week");
    Hoursworked = Integer.parseInt(scanner.nextLine());

} catch (NumberFormatException e) {
    System.out.println("Invalid! Please enter a numeric value");
    //The program will reprompt the user to enter their hours if it's invalid
    Hoursworked = Integer.parseInt(scanner.nextLine());
}
```

3. Validating users shift type

```
try {

    System.out.println("Plese select your shift number (Enter the corresponding number) \n 1) First s
    Shift = Integer.parseInt(scanner.nextLine());

} catch (NumberFormatException e) {
    System.out.println("Invalid! Please enter the number corresponding to your shift");
    Shift = Integer.parseInt(scanner.nextLine());
}

//I create a boolean that checks if the user selects a valid shift
boolean validshift = true;

//if the user enters a number greater than 3 and less than 1 then it'll be assigned
//as invalid
if (Shift > 3) {
    validshift = false;
} else if (Shift < 1) {
    validshift = false;
```

```
//The program will reprompt the user to input their shift number again if it
//isn't valid
while (!validshift) {
    System.out.println("Invalid Shift selected, please enter the correct corresponding number");
    Shift = Integer.parseInt(scanner.nextLine());
}

//Setting hourly rates according to the employee shift
if (Shift == 1) {
    Rate = 50;

} else if (Shift == 2) {
    Rate = 70;
} else {
    Rate = 90;
}

//Asking the second and third shift workers if they want to opt in for a retirement plan
if (Shift > 1) {
    System.out.println("Would you like to opt in for a retirem
    Retirement = scanner.next();
```

4. Asking the user if they want to opt into a retirement plan and displaying input data

```
//Asking the second and third shift workers if they want to opt in for a retirement plan
if (Shift > 1) {
    System.out.println("Would you like to opt in for a retirement plan? (Yes/no");
    Retirement = scanner.next();
}

//Displaying the users pay rate, hours worked and shift
System.out.println("Hours Worked: " + Hoursworked + " Hours");
System.out.println("Shift: " + Shift);
System.out.println("Hourly pay Rate: R" + Rate);
```

5. Calculating the users' regular pay

```
System.out.println("Hourly pay Rate: R" + Rate);

//Calculating employees regular pay
Regularpay = Rate * Hoursworked;

//Setting the total variable and regular pay
total = Regularpay;
```

6. Calculating Overtime

```
//Setting the users Overtime if they work over 40 hours a week
if (Hoursworked > 40) {

    //If the employee exceeds 40 hours their regular pay will be calcul
    //using 40 hours
    Regularpay = Rate * 40;

    //Then I determine how many extra hours the employee worked for
    int overhours = Hoursworked - 40;

    //Determining the new pay rate for overtime workers
    Overtimepayrate = Rate * 1.5;

    //Calculating overtime pay by implementing the new rate in the form
    //and extra hours
    Overtimepay = overhours * Overtimepayrate;

    //Calculating the total gross pay of employees by overwritting the
    //variable
    total = Regularpay + Overtimepay;
```

7. Calculating deductions and the user's total Net pay

```
//Checking if the user opted in for a retirement plan
if (Retirement.equalsIgnoreCase("yes")) {

    //if the user is an overtime worker the retirement will be deducted from the total
    //of their regular pay and overtime
    retirementdeduction = total * 0.05;

    //The retirement deduction will be removed from the total of their regular and
    //overtime pay
    netpay = total - retirementdeduction;

    //Displaying the retirement deduction
    System.out.println("Retirement deduction: R" + retirementdeduction);
}

System.out.println("Netpay R" + netpay);
```

8. Storing the user's hours worked into a text file using File Writer

```
        System.out.println("Netpay R" + netpay);

    try{
        FileWriter filewriter = new FileWriter("Output.txt",true);
        try(BufferedWriter writer = new BufferedWriter(filewriter)){
            writer.write("Hours Worked: " + Hoursworked);

        }
    }catch(IOException e){
        System.out.println("Error");
    }
```

Chosen Solution

In terms of complexity, the methods used in both solutions aren't complex to implement but I chose to implement solution 1, the RandomAccessfile is a better all-rounder for adding data to a file than file writer, Random-access allows the user to enter any kind of data to a file which helps with this scenario in case the company wants to record names and transaction details.

This means that the program is scalable and can be expanded to add new data to a file when necessary, which reduces the complexity and cost of expanding the program and guarantees its longevity.

RandomAccessFile also allows data to be retrieved from any part of a file which helps when additional data that has been stored is needed for additional transactions.

The program will approximately take a week to create and implement within the system, a team of 2 developers will be assigned to develop and test the program among each other considering their salary of 12 000 a week each that brings development costs to 24 000 along with integration and device costs the total cost of this solution will be R30 000, However considering that this program automates accounting by calculating and recording data which severely lowers the workload on the accounting team which enables less costs to be assigned to them for manual bookkeeping and also allows them to focus on the funds elsewhere within the organization ensuring a return on investment