# *“Design Documentation”*

# Table of Contents

## *a) Desk-checking.*

### *Review the Design Documents:*

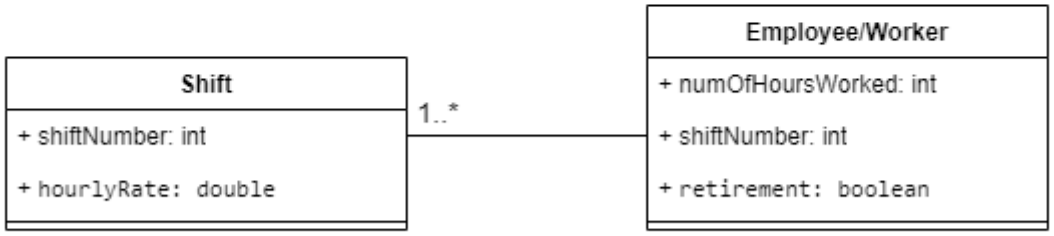
The following points have been checked and tested.

- Making sure that all business requirements are translated and detailed correctly into the design.
- Checking if all the inputs (e.g., hours worked, shift number, retirement plan participation) are validated and handled well.
- Confirming that all the calculations (e.g., regular pay, overtime pay, retirement deductions) are accurate.
- Making sure that the program can write to and read from a file properly.
- Confirming that the design sticks to the coding standards and best practices.

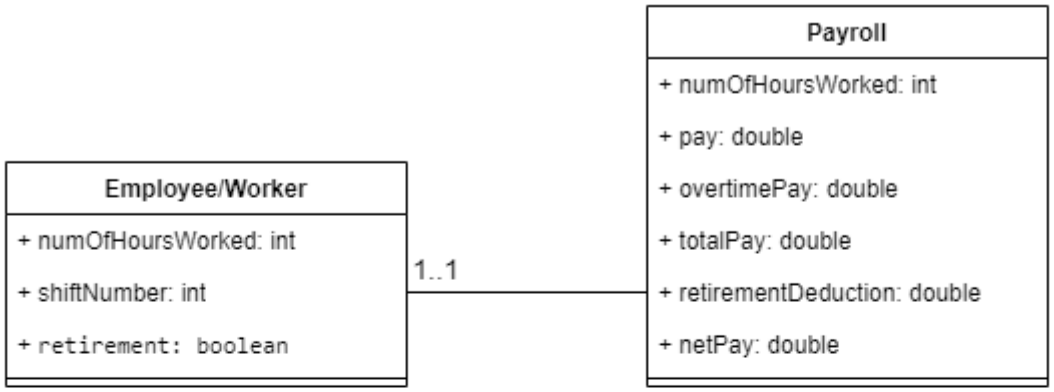
## *b) Use UML diagrams to document the overall system architecture, including class diagrams for object-oriented design.*

*UML Diagram*

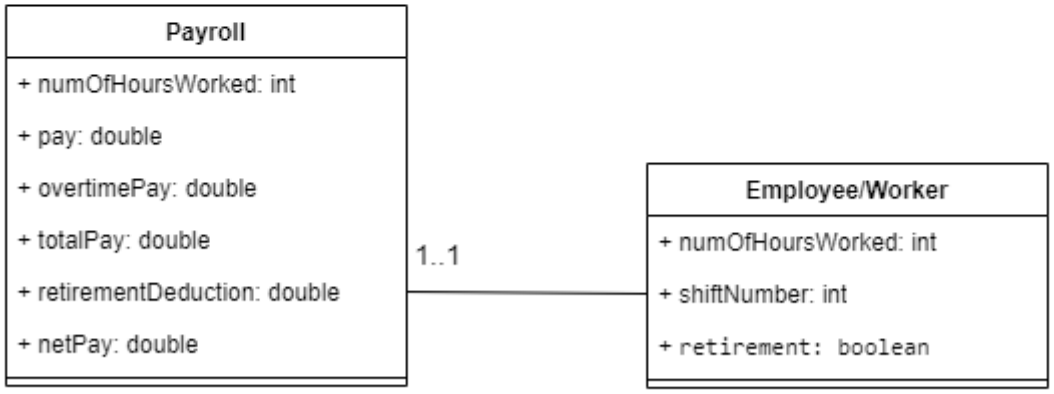

Each employee is associated with a shift.




A shift can be associated with multiple employees.




Each employee has a payroll.




Each payroll can be associated with an employee.

*Relationships*

**Employee** and **Shift**:

- Each employee is associated with a shift.
- A shift can be associated with multiple employees.
- Relationship: One-to-Many (One Shift can have many Employees)

**Employee** and **Payroll**:

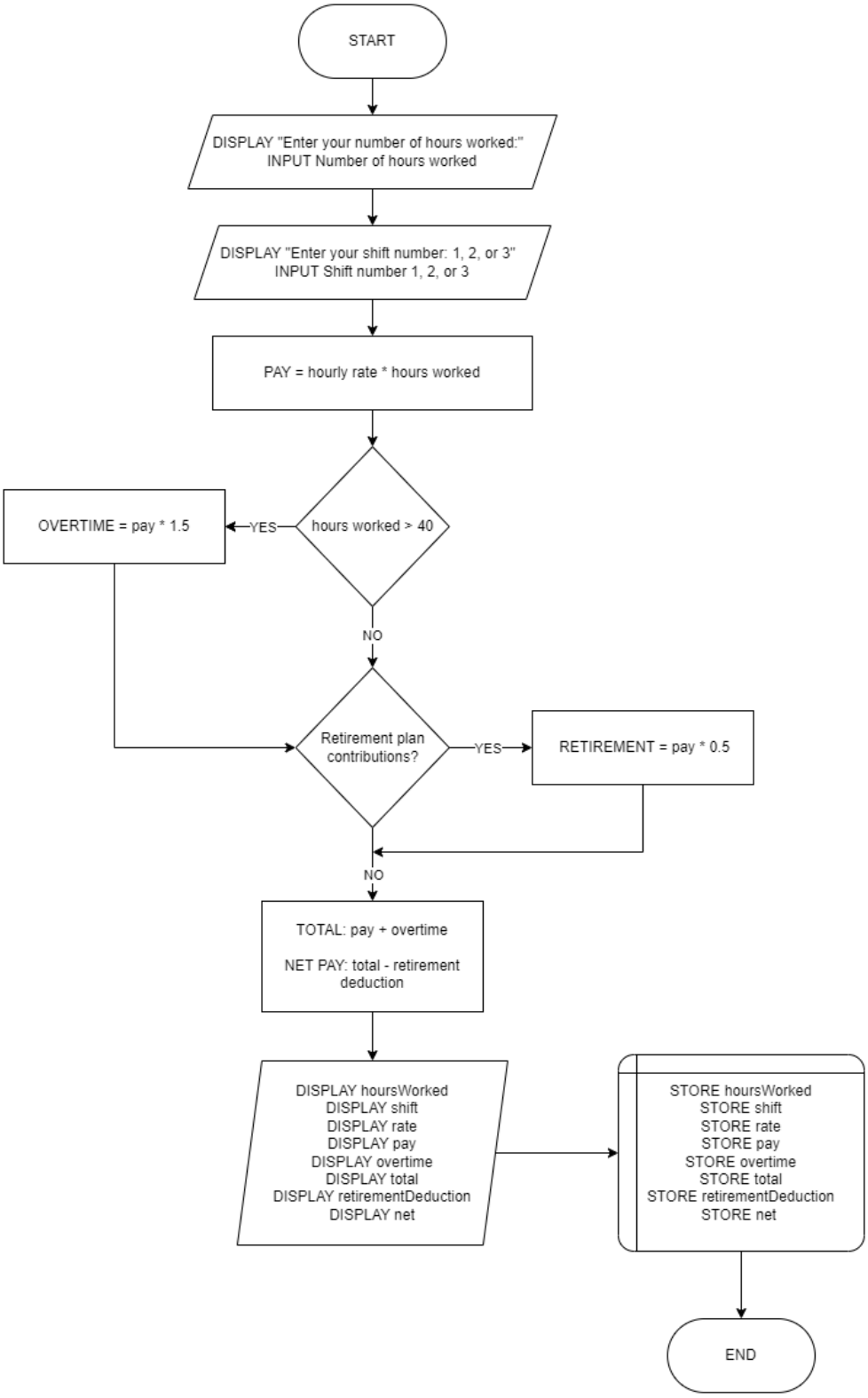
- Each employee has a payroll.
- Relationship: One-to-One (Each Employee has one Payroll record at a time)

*Description*

- **Employee** class includes details about the employee's hours worked, shift number, and participation in the retirement plan.
- **Shift** class includes details about different shifts and their corresponding hourly rates.
- **Payroll** class h6andles payroll calculations such as regular pay, overtime pay, total pay, retirement deductions, and net pay.

## *c) Create flowcharts to visualize the program's operational flow. Write pseudo code to outline the algorithm.*

*Flowchart*



*Pseudocode*

```
START
    // this is creating an object of the Scanner class named scanner
    CREATE scanner object

    INITIALIZE x as false
    INITIALIZE numOfHours as 0

    // this is a while loop that is used to validate the input of the employee's number of hours worked
    WHILE x is false
      PRINT "Enter your number of hours worked: "
      numOfHours = SCANNER.nextInt()

      IF numOfHours > 0
        SET x as true
      ELSE
        PRINT "Invalid option. Number of hours has to be > 1."

    INITIALIZE y as false
    INITIALIZE shiftNumber as 0

    // this is a while loop that is used to validate the input of the employee's shift number
    WHILE y is false
      PRINT "Enter your shift number (1, 2, or 3):"
      PRINT "1 - First shift"
      PRINT "2 - Second shift"
      PRINT "3 - Third shift"
      shiftNumber = SCANNER.nextInt()

      IF shiftNumber > 0 AND shiftNumber < 4
        SET y as true
      ELSE
        PRINT "Invalid option. Please choose 1, 2, or 3."

    // this is a if statement that asks the employee about participating in the retirement plan
    IF shiftNumber is 2 OR 3
      PRINT "Would you like to participate in the retirement plan? (yes/no): "
```

```
    choice = SCANNER.next()
    y = choice.equalsIgnoreCase("yes")
// this is a switch case to initalaise the hourlyRate depaending on the shift number of the employees
SWITCH shiftNumber
    CASE 1
        hourlyRate = 50
    CASE 2
        hourlyRate = 70
    CASE 3
        hourlyRate = 90
    DEFAULT
        hourlyRate = 0

// this is used to calculate the overtime rate of the employee
overtimeRate = hourlyRate * 1.5

// this is a if statement used to calculate the regular pay and the overtime pay of the employee (if any)
IF numOfHours > 40
    regularPay = hourlyRate * 40
    overtimePay = (numOfHours - 40) * overtimeRate
ELSE
    regularPay = hourlyRate * numOfHours
    overtimePay = 0

// this is used to calculate the total pay of the employee
totalPay = regularPay + overtimePay

// this is a if statement used to calculate the retirement deduction of the employee (if any)
IF y
    retirementDeduction = totalPay * 0.05
ELSE
    retirementDeduction = 0

// this is used to calculate the net pay of the employee
netPay = totalPay - retirementDeduction

// this is the payment breakdown
PRINT payment breakdown
```

```
        CALL writeFile(numOfHours, shiftNumber, hourlyRate, regularPay, overtimePay,
        totalPay, retirementDeduction, netPay)
        CALL readFile()
STOP
        // this is a methoed that is used to write to the payroll file
        FUNCTION writeFile(numOfHours, shiftNumber, hourlyRate, regularPay,
        overtimePay, totalPay, retirementDeduction, netPay)

          TRY
            // this is use to create a object of the RandomAccessFile class called
            payrollFile
            CREATE RandomAccessFile object "payrollFile" with "rw" mode

            // this is use to create a object of the RandomAccessFile class called
            payrollFile
            payrollFile.seek(payrollFile.length())

            // this is the payment breakdown that will saved into the payroll file
            WRITE payment breakdown to payrollFile

            // this is used to close the payroll file
            payrollFile.close()
          CATCH Exception e
            PRINT e

        // this is a methoed that is used to read from the payroll file
        FUNCTION readFile()
          TRY
            // this is use to create a object of the RandomAccessFile class called
            payrollFile
            CREATE RandomAccessFile object "payrollFile" with "r" mode

            // this is used to start reading from the beginning of the file
            payrollFile.seek(0)

            // this is used to read and display each line from the payroll file
            WHILE (line = payrollFile.readLine()) != null
              PRINT line

            // this is used to close the payroll file
```

```
    payrollFile.close()
CATCH Exception e
    PRINT e
```

## *d) Data Structures.*

*Loops:*

- `while-loops` to ask the employees to enter their number of hours worked and if invalid, allowing the employee to try again.
- `while-loops` to ask the employees to enter their shift number and if invalid, allowing the employee to try again.

*If Statements:*

- `if Statement` to allow only valid the input of the number of hours they have worked.
- `if Statement` to allow only valid the input of the shift number they have worked.
- `if Statement` to ask the employees that worked for shift 2 and 3 if they would you like to participate in the retirement plan.
- `If Statement` used to calculate the regular pay and the overtime pay of the employee (if any)

*Data Storage:*

- Use `RandomAccessFile` for direct file access, allowing reading from and writing to any location in the file.
- Ensure data persistence by saving the number of hours entered each time the program runs.

*Switch Case*

- Switch case to initialize the hourlyRate depending on the shift number of the employees.