IS1S481 Coursework 1

Jake Real - 23056792

08/12/2023

Contents

Part A - Design Task

Part 1 User Login and Unique Pin

Design Decisions

When creating the design for this program, the separation of user interface and data was important; therefore, `Employee` and `UserInterface` were placed into separate entities which are controlled by `Main`. `UserInterface` can only provide or use data that `Employee` allows it to, ensuring data that everything is encapsulated.

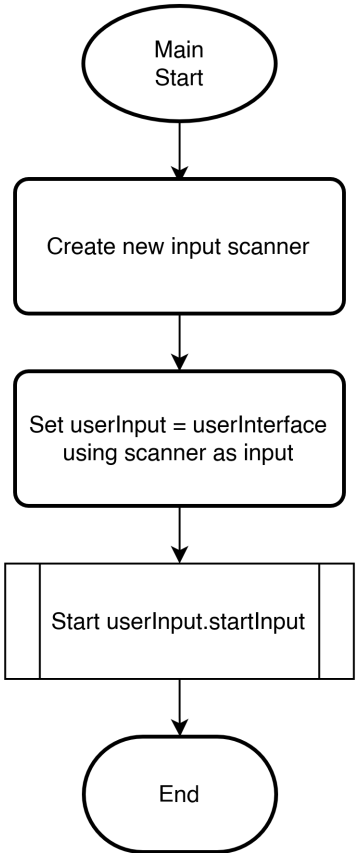

Figure 1: Flowchart of Main

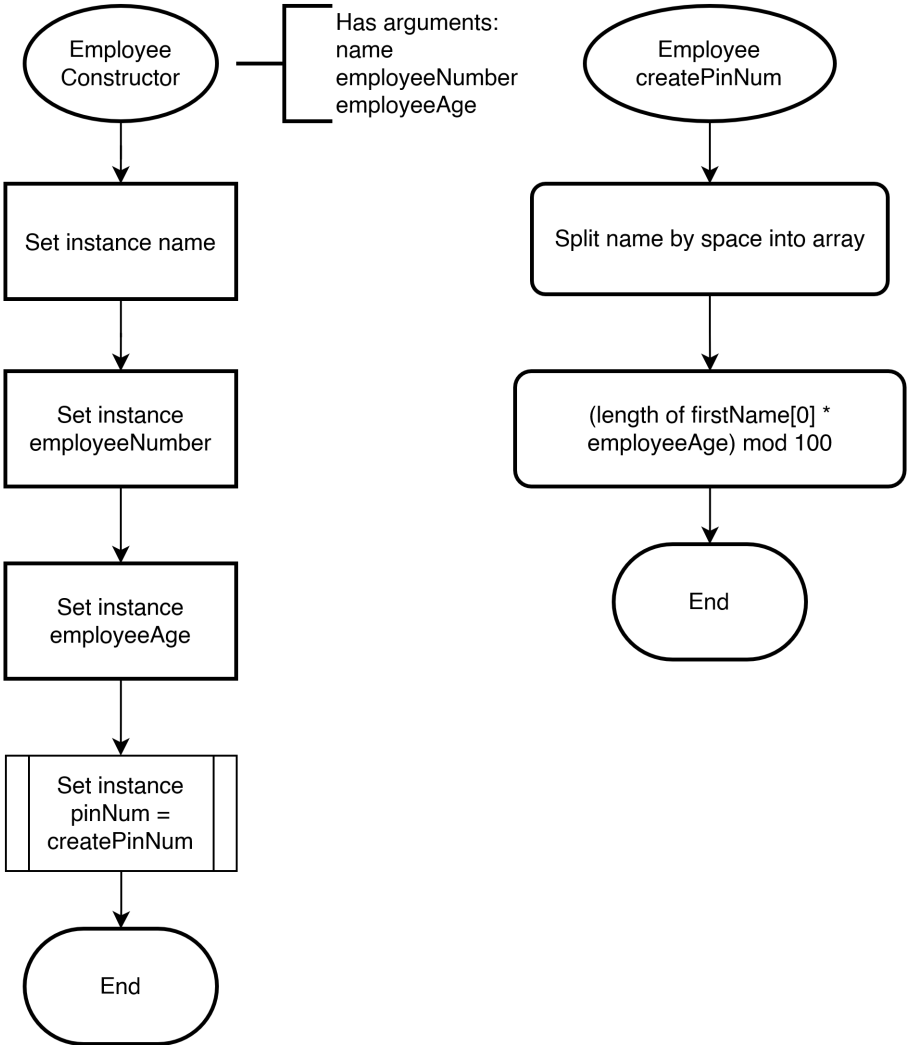

Figure 2: Flowchart of Employee

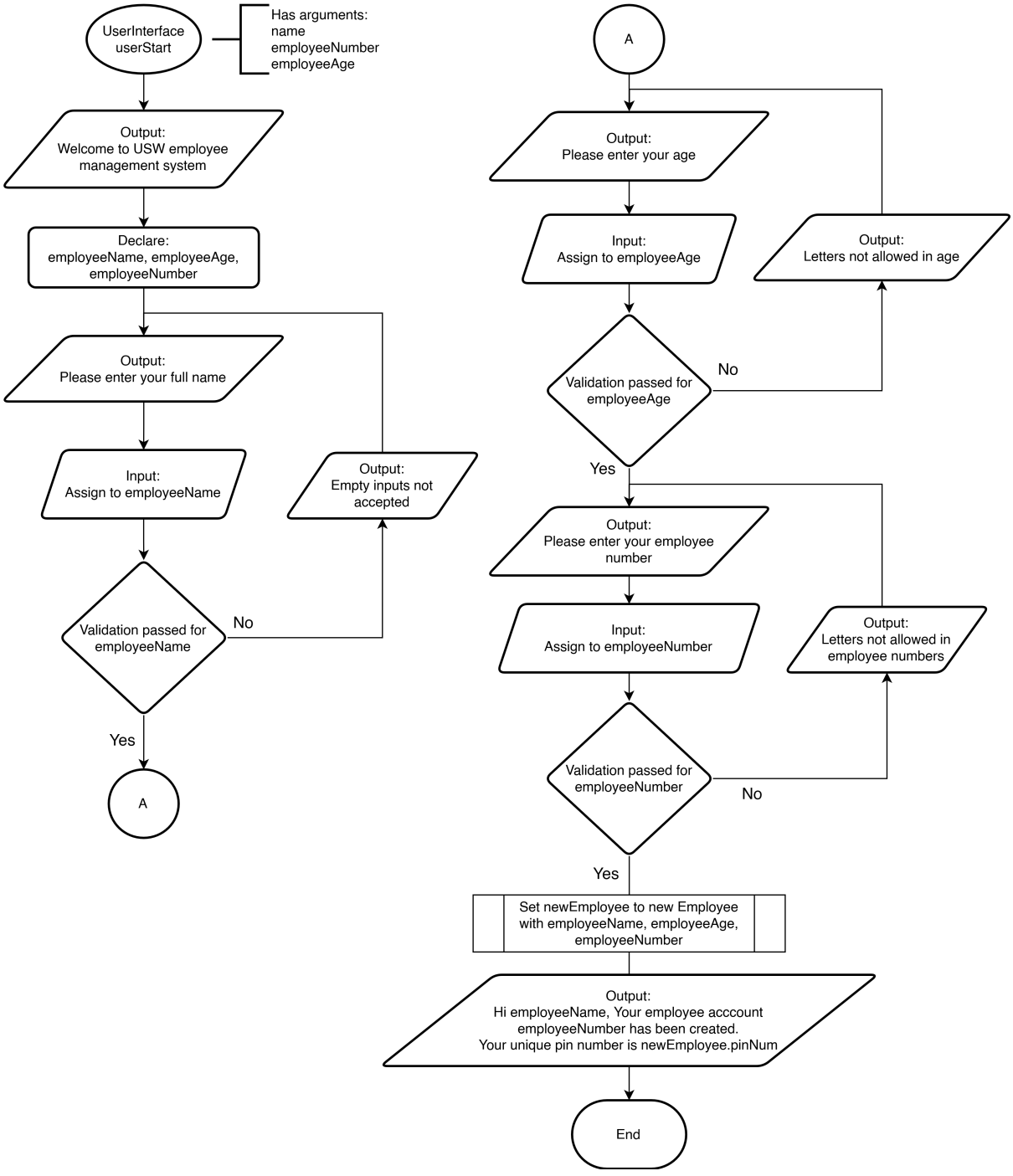

Figure 3: Flowchart of UserInterface

Part 2 - Employee Pay Calculator

Design Decisions:

Several important design choices were made prior to starting on the flowcharts and program. The following choices were made, salary calculations would be created through a process of test-driven development to ensure that they carried out the correct calculations. This necessitated the use of dependency injection in areas related to input and output as the tests had to be consistent and unaffected by changes to user input or files. Therefore, the flowcharts had to display dependency injection through providing object-oriented classes as arguments.

Moreover, separating control of the program was an important design goal. Classes associated with input and output should only serve as constructors to their calling class, such as `Main`. They should not perform any substantial data operations; said data operations should happen in their related classes, such as `Employee`, and `Salary`.

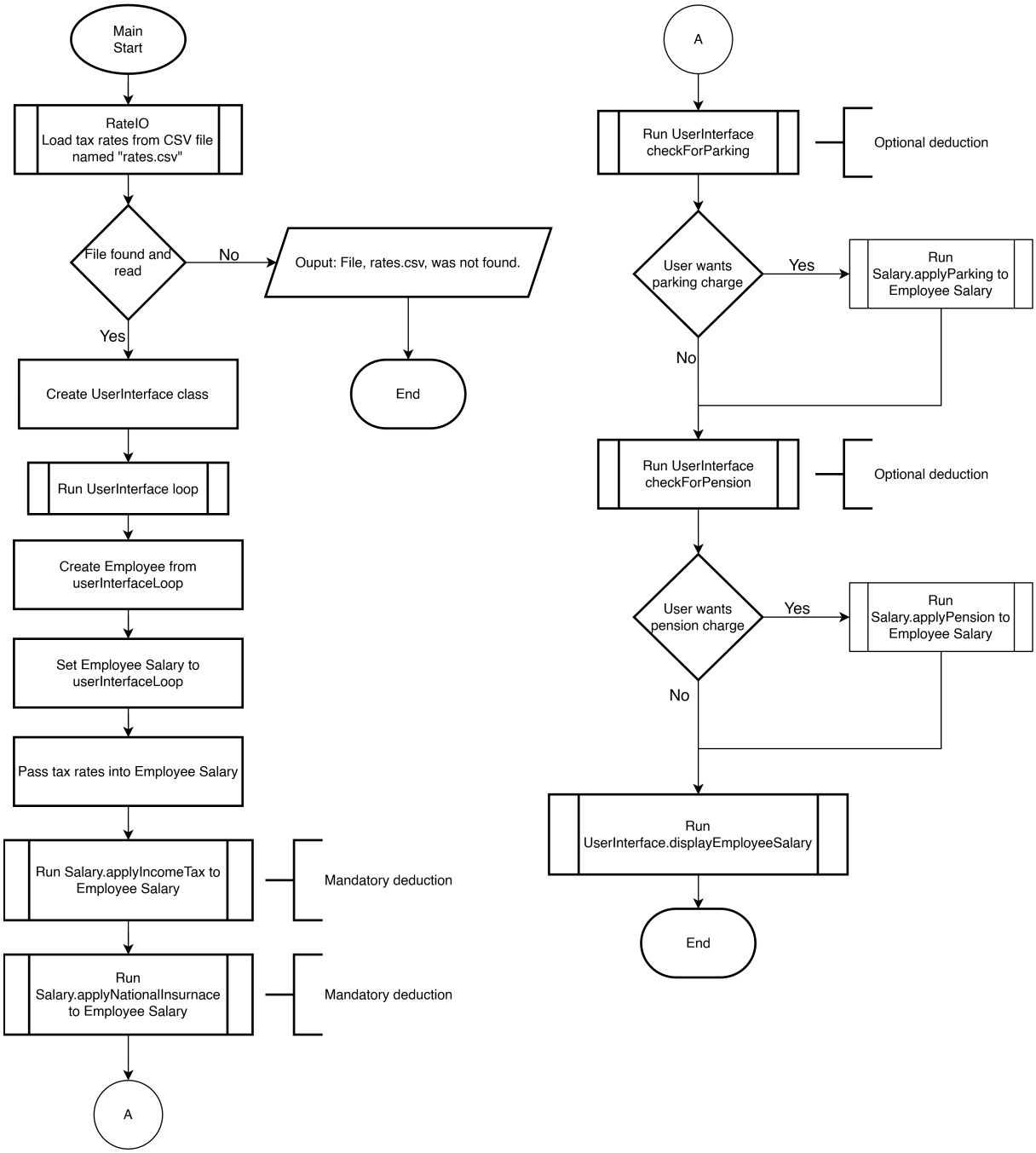

Figure 4: Flowchart of Main

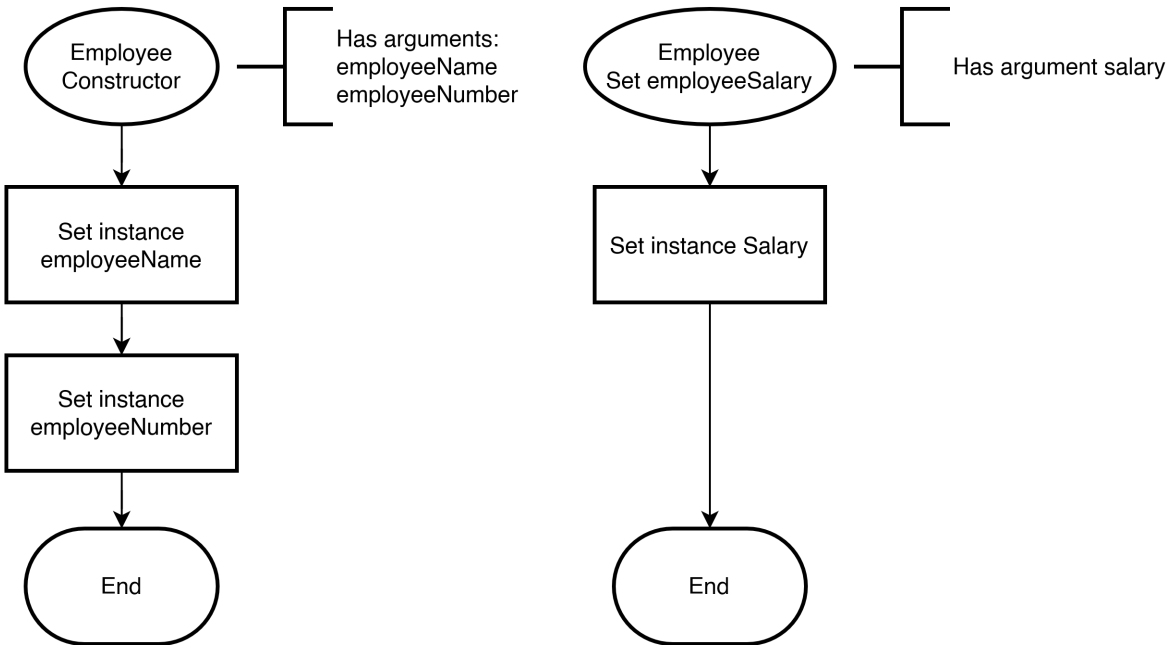

Figure 5: Flowchart of Employee

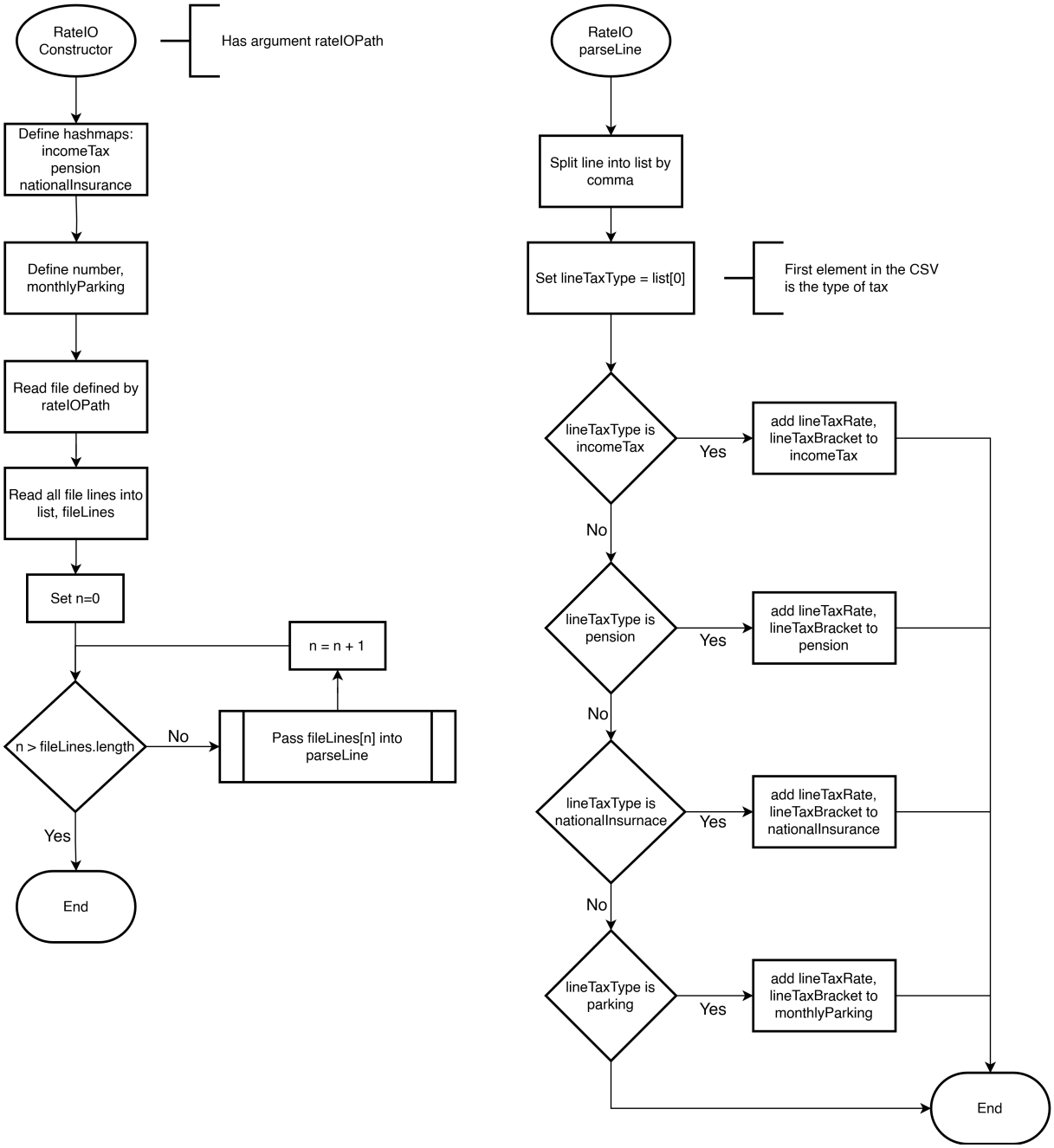

Figure 6: Flowchart of RateIO

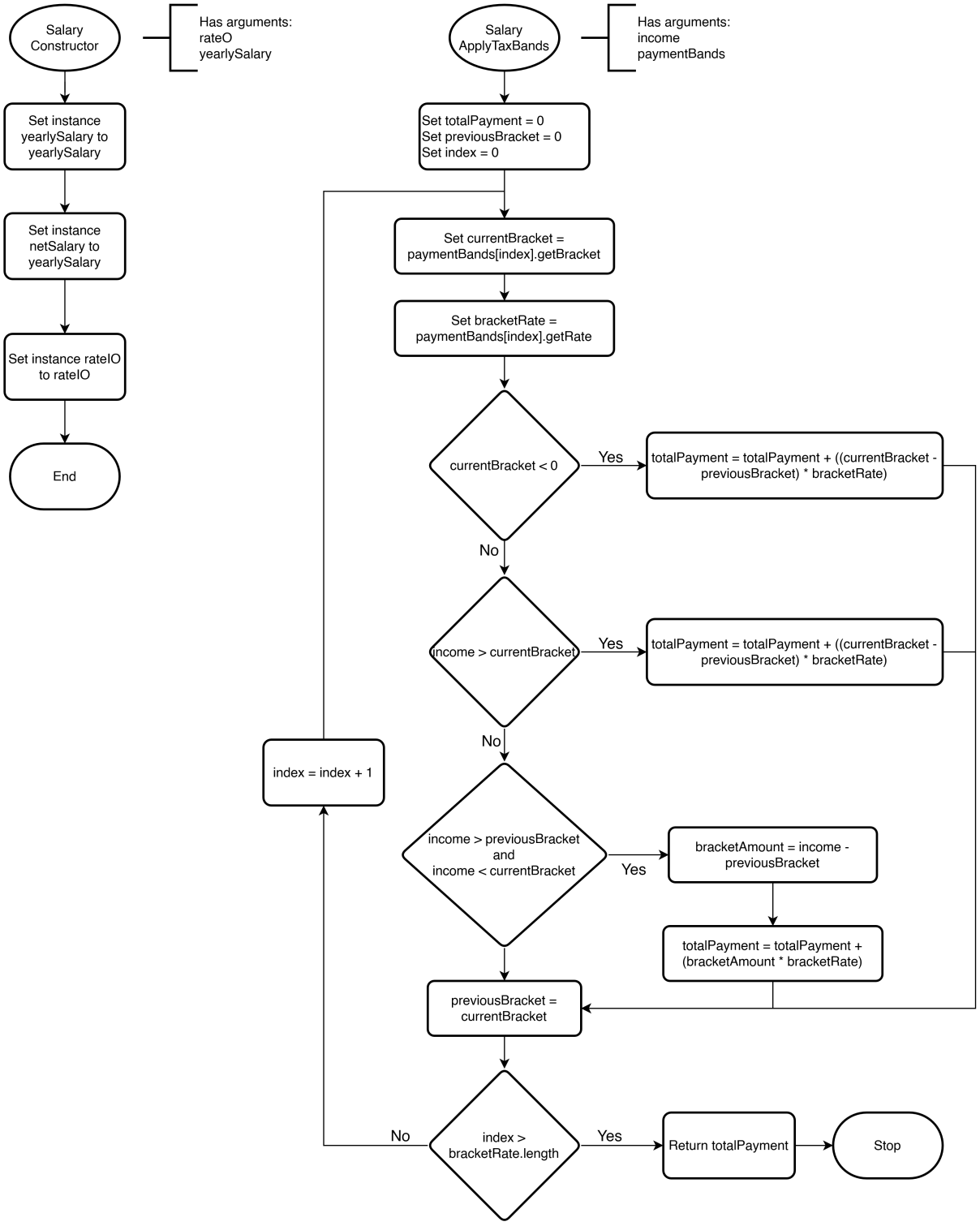

Figure 7: Flowchart of Salary

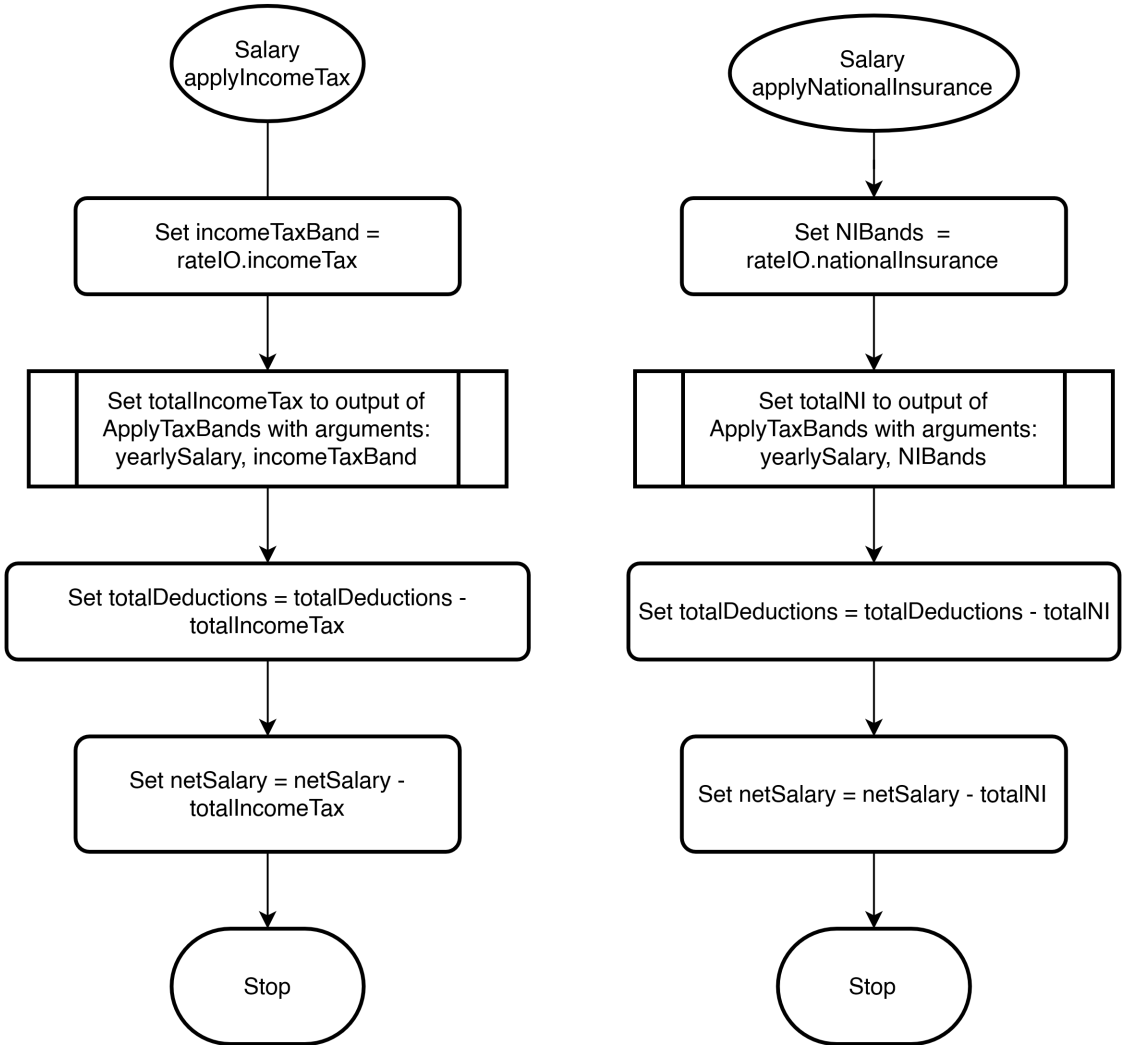

Figure 8: 2nd Flowchart of Salary

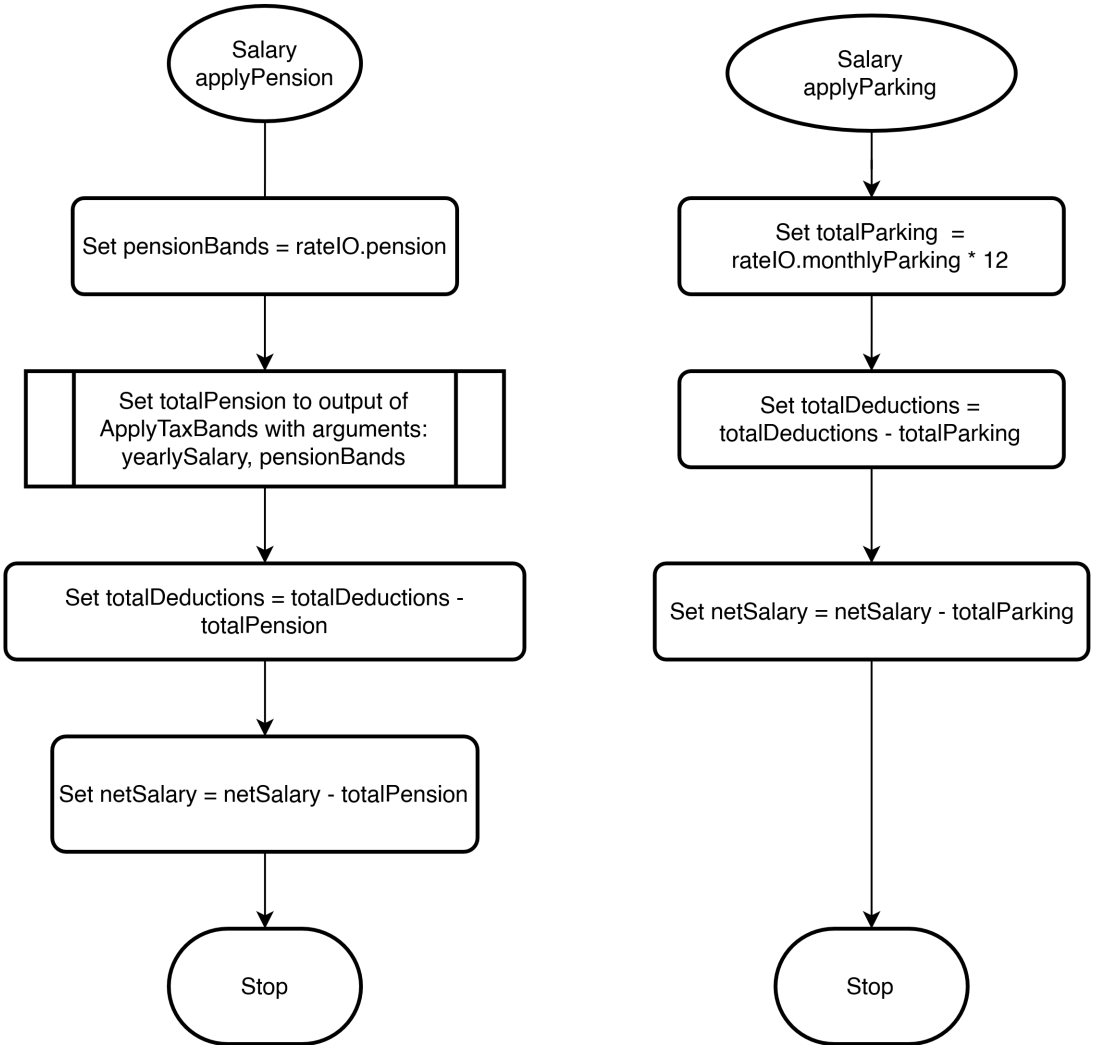

Figure 9: 3rd Flowchart of Salary

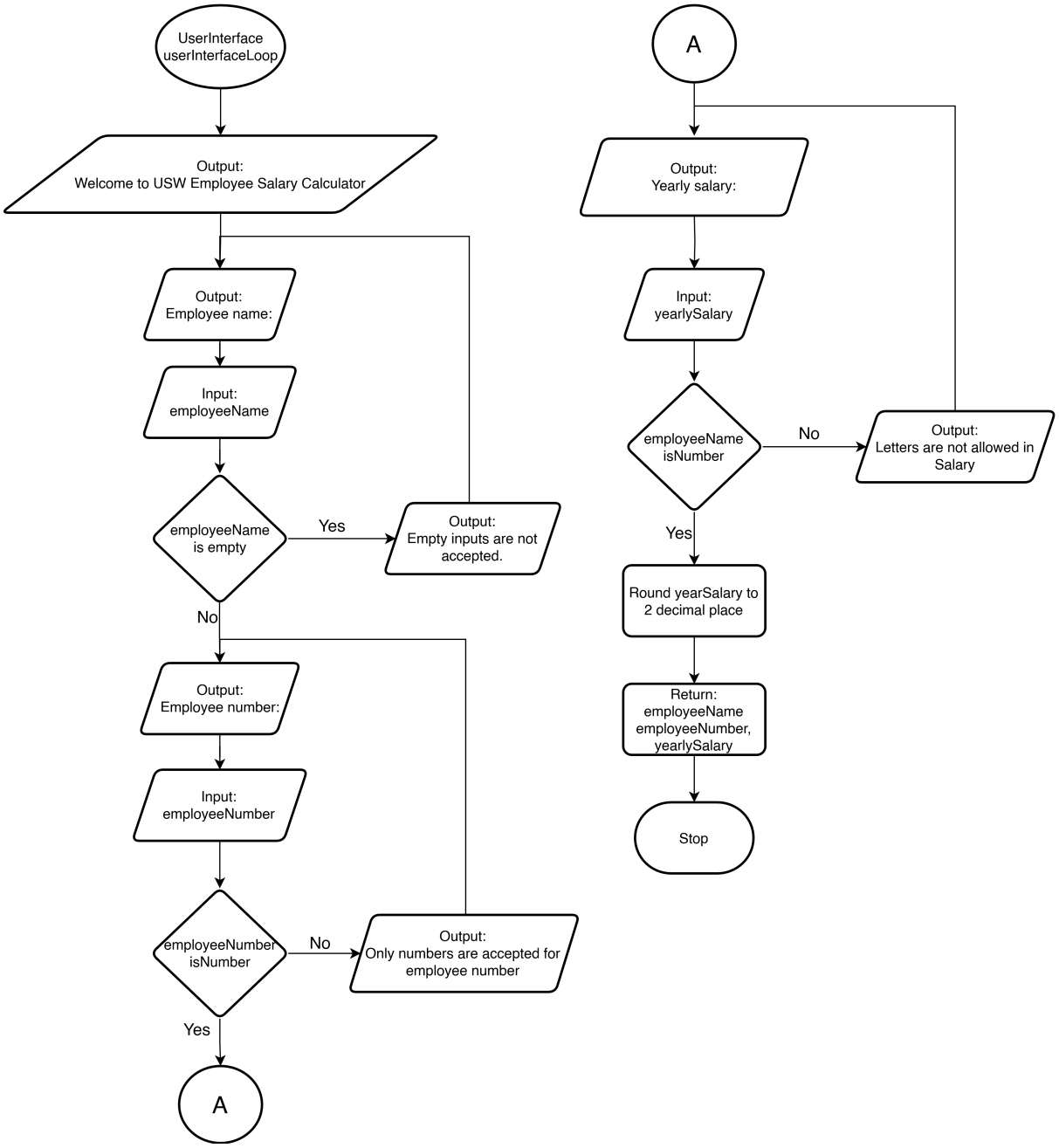

Figure 10: Flowchart of UserInterface

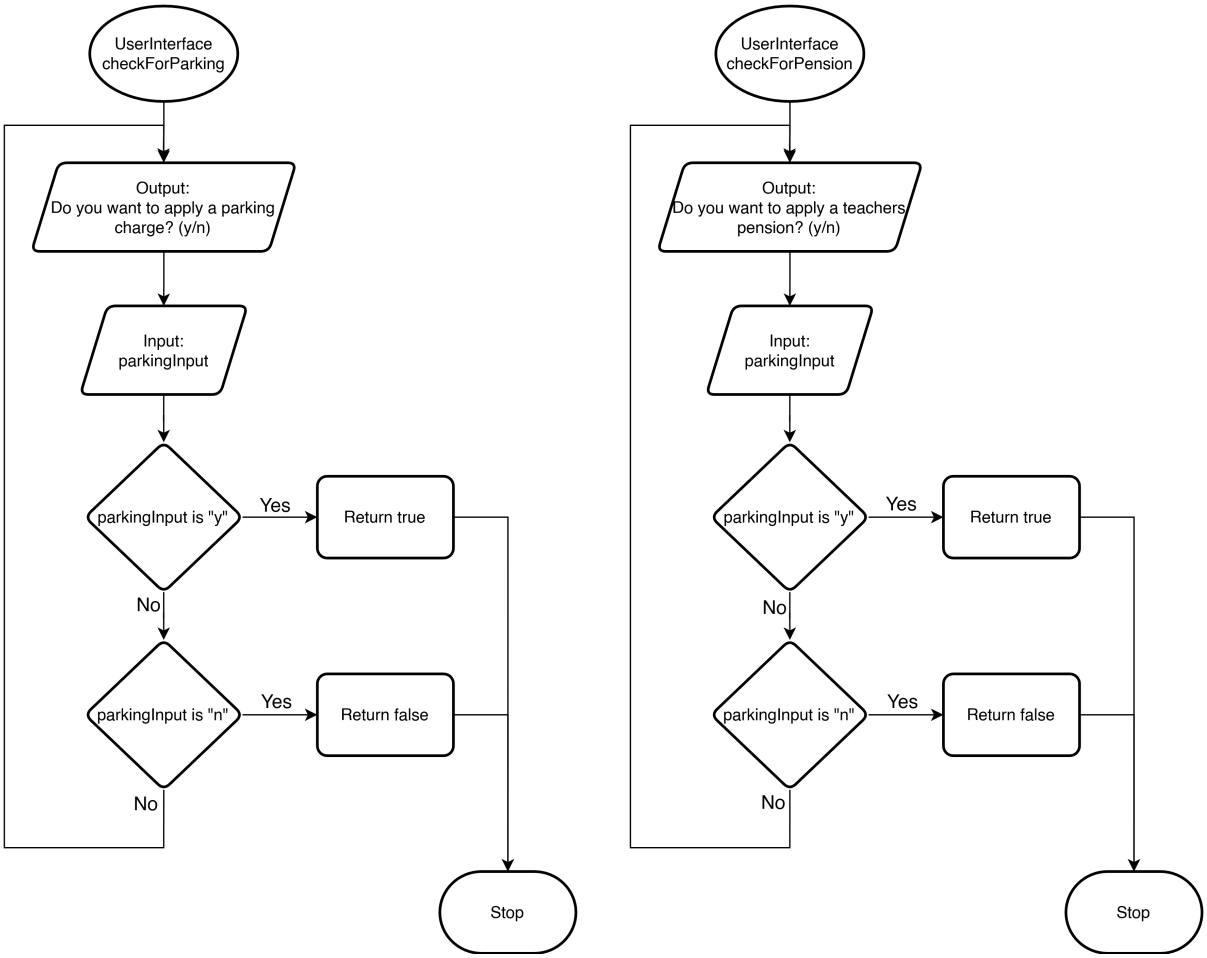

Figure 11: 2nd Flowchart of UserInterface

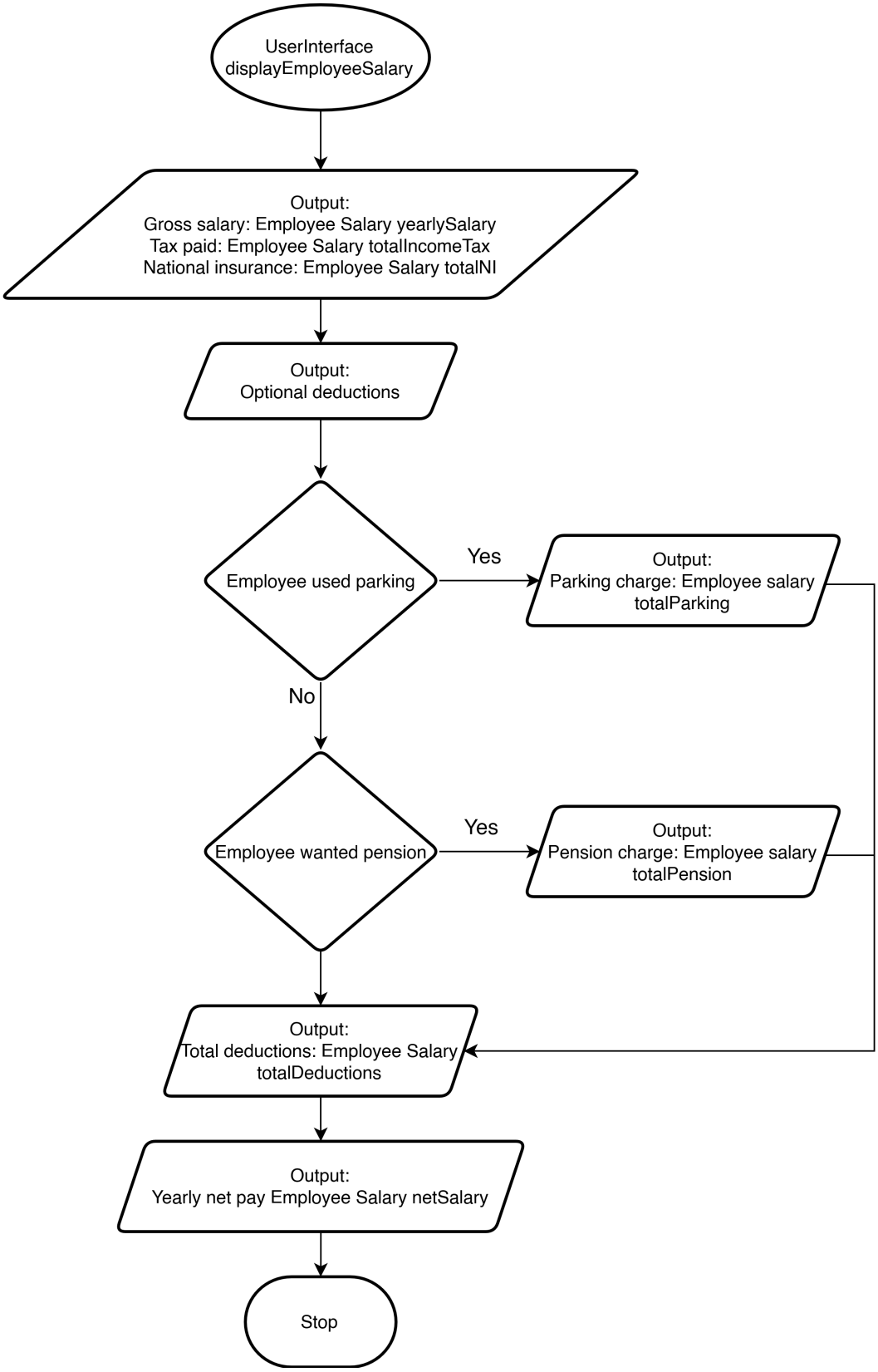

Figure 12: 3rd Flowchart of UserInterface

Part B - Programming Task

Part 1 User Login and Unique Pin

Source Code

`Main.java`

```java
package usw.employeelogin;

import java.util.Scanner;

public class Main {
    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);
        UserInterface userInput = new UserInterface(scanner);
        userInput.userStart();
    }
}
```

`Employee.java`

```java
package usw.employeelogin;

public class Employee {
    private final String name;
    private final int employeeNum;
    private final int employeeAge;
    private final int pinNum;

    /**
     * Creates an employee
     *
     * @param name        Name of employee
     * @param employeeNum Number employee
     * @param employeeAge Age of employee
     */
    public Employee(String name, int employeeNum, int employeeAge) {
        this.name = name;
        this.employeeNum = employeeNum;
        this.employeeAge = employeeAge;
        this.pinNum = createPinNum();
    }

    /**
     * Internal class that returns code. Used in construction of class
     *
```

```
26       * @return Returns the person's PIN
27       */
28      private int createPinNum() {
29          // Gets the person's PIN
30          // PINs are user's name length multiplied by their age
31          // Modulo prevents pins above 9999
32          String[] firstName = this.name.split(" ");
33          return (firstName[0].length() * employeeAge) % 1000;
34      }
35
36      public String getName() {
37          return name;
38      }
39
40      public int getPinNum() {
41          return pinNum;
42      }
43
44      public int getEmployeeNum() {
45          return employeeNum;
46      }
47  }
```

UserInterface.java

```
1   package usw.employeelogin;
2
3   import java.util.InputMismatchException;
4   import java.util.Scanner;
5
6   public class UserInterface {
7
8       private final Scanner scanner;
9
10      public UserInterface(Scanner scanner) {
11          this.scanner = scanner;
12      }
13
14      public void userStart() {
15          System.out.println(
16              "Welcome to USW employee management system
17          ");
18
19          String employeeName;
20          int employeeAge;
21          int employeeNumber;
22
23
24          while (true) {
25              System.out.print("Please enter your full name: ");
```

```
            employeeName = scanner.nextLine();
            if (!employeeName.isEmpty()) {
                break;
            }
            System.out.println("Empty inputs are not accepted");
        }

        while (true) {
            System.out.print("What's your age: ");
            try {
                employeeAge = scanner.nextInt();
                if (employeeAge < 0) {
                    System.out.println(
                        "Negative ages not allowed"
                    );
                    scanner.nextLine();
                    continue;
                }
                break;
            } catch (InputMismatchException e) {
                System.out.println(
                    "Letters not allowed in age"
                );
                scanner.nextLine();
            }
        }

        while (true) {
            System.out.print("Please enter your employee number: ");
            try {
                employeeNumber = scanner.nextInt();
                if (employeeNumber < 0) {
                    System.out.println(
                        "Negative employee numbers not allowed"
                    );
                    scanner.nextLine();
                    continue;
                }
                break;
            } catch (InputMismatchException e) {
                System.out.println(
                    "Letters not allowed in employee number"
                );
                scanner.nextLine();
            }
        }

        Employee newEmployee = new Employee(employeeName,
            employeeNumber, employeeAge
        );
        System.out.printf(
```

```
77              "Hi %s. Your employee account %d has been created.
78              Your unique pin number is %04d.",
79              newEmployee.getName(),
80              newEmployee.getEmployeeNum(),
81              newEmployee.getPinNum()
82          );
83      }
84  }
```

Program Unit Tests

`EmployeeTest.java`

```java
package usw.employeelogin;

import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertEquals;

class EmployeeTest {
    Employee testEmployee = new Employee("jim", 330, 20);
    Employee testEmployeeLong = new Employee("12345", 203, 4321);

    @Test
    @DisplayName("Pin number generated correctly")
    public void getPinNum() {
        assertEquals(60, testEmployee.getPinNum());
        assertEquals(605, testEmployeeLong.getPinNum());
    }
}
```

Program Outputs

Running `Main.java` with typical inputs,

```
Welcome to USW employee management system
Please enter your full name: Jake
What's your age: 19
Please enter your employee number: 234212
Hi Jake. Your employee account 234212 has been created. Your unique pin
    number is 0076.
```

Using longer details,

```
Welcome to USW employee management system
Please enter your full name: Jefferson Jame
```

```
What's your age: 34
Please enter your employee number: 43244
Hi Jefferson Jame. Your employee account 43244 has been created. Your
    unique pin number is 0306.
```

Testing input validation:

Full name,

```
Welcome to USW employee management system
Please enter your full name:
Empty inputs are not accepted
Please enter your full name:
```

Age,

```
Welcome to USW employee management system
Please enter your full name: Jake
What's your age: af
Letters not allowed in age
What's your age:
```

Employee number,

```
Welcome to USW employee management system
Please enter your full name: Jake
What's your age: 43
Please enter your employee number: a
Letters not allowed in employee number
Please enter your employee number:
```

Unit Test Outputs,

```
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
[INFO] Running usw.employeelogin.EmployeeTest
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed:
    0.063 s - in usw.employeelogin.EmployeeTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO]
[INFO] --- jar:3.3.0:jar (default-jar) @ pop-coursework ---
[INFO] Building jar: /home/jake/Code/usw/pop-coursework-1/target/pop-
    coursework-1.0-SNAPSHOT.jar
[INFO] -------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] -------------------------------------------------------
```

```
[INFO] Total time:  3.098 s
[INFO] Finished at: 2023-12-08T17:37:55Z
[INFO] ------------------------------------------------------------
```

This test determines if the employee's pin is calculated correctly. Two sets of test data are used; data that results in short pin, and data that generates a longer pin.

Part 2 - Employee Pay Calculator

Program Source Code

`Main.java`

```java
package usw.employeepay;

import java.io.IOException;
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        RateIO rateIO;
        try {
            rateIO = new RateIO("rates.csv");


        } catch (IOException e) {
            System.out.println("File, rates.csv, was not found. Make
                sure rates.csv is run in same folder as the " +
                    "program");
            return;
        }
        Scanner scanner = new Scanner(System.in);
        UserInterface userInput = new UserInterface(scanner);
        Employee employee = userInput.createEmployeeLoop();
        employee.setEmployeeSalary(userInput.getSalaryLoop(rateIO));

        /* Apply income tax and national insurance */
        employee.getSalary().applyMandatoryDeductions();

        /* Check if user wants to apply optional deductions */
        if (userInput.userApplyParking()) {
            employee.getSalary().applyParkingCharge();
        }
        if (userInput.userApplyPension()) {
            employee.getSalary().applyPension();
        }
        UserInterface.displayEmployeeSalary(employee);
    }
```

```
35 }
```

UserInterface.java

```java
package usw.employeepay;

import java.math.BigDecimal;
import java.math.RoundingMode;
import java.util.InputMismatchException;
import java.util.Scanner;

public class UserInterface {

    private final Scanner scanner;


    /**
     * Class that handles outputting and accepting user input
     *
     * @param scanner Input handling
     */
    public UserInterface(Scanner scanner) {
        this.scanner = scanner;
    }


    /**
     * Outputs the information concerning an employee's salary
     *
     * @param employee Employee to display salary of
     */
    public static void displayEmployeeSalary(Employee employee) {

        System.out.println("\nCalculating yearly net pay...\n");
        System.out.printf("""
            Gross salary: £%s
            Taxable amount: £%s
            Tax paid: £%s
            National insurance paid: £%s
            """,
            employee.getSalary().getGrossSalary(),
            employee.getSalary().getTaxableAmount(),
            employee.getSalary().getIncomeTaxAmount(),
            employee.getSalary().getNIAmount()
        );

        /* Non-required deductions */
        if (!(employee.getSalary().getTotalParking() == null)) {
            System.out.printf("Parking charge: £%s\n",
                employee.getSalary().getTotalParking()
```

```
47              );
48          }
49
50          if (!(employee.getSalary().getPensionAmount() == null)) {
51              System.out.printf("Pension charge: £%s\n",
52                  employee.getSalary().getPensionAmount()
53              );
54          }
55
56          System.out.printf("\nTotal deductions: £%s\n",
57              employee.getSalary().getTotalDeductions()
58          );
59          System.out.printf("Yearly net pay: £%s\n",
60              employee.getSalary().getNetSalary()
61          );
62
63
64          System.out.println("\nCalculating monthly net pay...\n");
65          System.out.printf("""
66              Gross salary: £%s
67              Taxable amount: £%s
68              Tax paid: £%s
69              National insurance paid: £%s
70              """,
71              Salary.convertMonthly(
72                  employee.getSalary().getGrossSalary()
73              ),
74              Salary.convertMonthly(
75                  employee.getSalary().getTaxableAmount()
76              ),
77              Salary.convertMonthly(
78                  employee.getSalary().getIncomeTaxAmount()
79              ),
80              Salary.convertMonthly(
81                  employee.getSalary().getNIAmount()
82              )
83          );
84
85
86          /* Non-required deductions */
87          if (!(employee.getSalary().getTotalParking() == null)) {
88              System.out.printf("Parking charge: £%s\n",
89                  Salary.convertMonthly(employee.getSalary().
                        getTotalParking())
90              );
91          }
92
93          if (!(employee.getSalary().getPensionAmount() == null)) {
94              System.out.printf("Pension charge: £%s\n",
95                  Salary.convertMonthly(employee.getSalary().
                        getPensionAmount())
```

```
96              );
97          }
98
99          System.out.printf("\nMonthly total deductions: £%s\n",
100             Salary.convertMonthly(employee.getSalary().
                    getTotalDeductions())
101         );
102
103         System.out.printf("Monthly net pay: £%s\n",
104             employee.getSalary().getMonthlyNetSalary()
105         );
106     }
107
108     /**
109      * UI loop constructs an Employee class and returns it
110      * Uses validation
111      *
112      * @return Constructed Employee object
113      */
114     public Employee createEmployeeLoop() {
115
116         String employeeName;
117         int employeeNumber;
118
119         System.out.println(
120             "Welcome to USW Employee Salary Calculator"
121         );
122         System.out.println(
123             "----------------------------------------"
124         );
125
126         while (true) {
127             System.out.print("Employee Name: ");
128             employeeName = scanner.nextLine();
129             if (!employeeName.isEmpty()) {
130                 break;
131             }
132             System.out.println("Empty inputs are not accepted.");
133         }
134
135         while (true) {
136             System.out.print("Employee number: ");
137             try {
138                 employeeNumber = scanner.nextInt();
139                 if (employeeNumber < 0) {
140                     System.out.println(
141                         "Negative numbers not accepted"
142                     );
143                     continue;
144                 }
145                 break;
```

```
146             } catch (InputMismatchException e) {
147                 System.out.println(
148                     "Letter are not allowed employee number"
149                 );
150                 /* nextLine clears the newline from nextInt() avoiding
151                 duplicates of above message */
152                 scanner.nextLine();
153             }
154         }
155         return new Employee(employeeName, employeeNumber);
156     }
157 
158     /**
159      * UI loop that constructs Salary that is filled with tax
            information
160      *
161      * @param rateIO The tax bands to use in initial instantiation of
            taxes, pension, etc.
162      * @return Constructed Salary object
163      */
164     public Salary getSalaryLoop(RateIO rateIO) {
165 
166         BigDecimal yearSalary;
167 
168         while (true) {
169             System.out.print("Yearly salary: ");
170             try {
171                 String inputSalary = scanner.next();
172                 yearSalary = new BigDecimal(inputSalary);
173                 yearSalary = yearSalary.setScale(2, RoundingMode.
                        HALF_UP);
174                 /* Clear the newline character from scanner buffer
175                  * Otherwise next question would appear twice, as the
176                  * scanner would pick up the leftover newline
177                  */
178                 scanner.nextLine();
179 
180                 // Check if the number is negative
181                 if (yearSalary.compareTo(BigDecimal.ZERO) < 0) {
182                     System.out.println(
183                         "Negative salaries are not accepted"
184                     );
185                     continue;
186                 }
187                 break;
188             } catch (NumberFormatException e) {
189                 System.out.println(
190                     "Letter are not allowed in the employee number"
191                 );
192             }
193         }
```

```
194             return new Salary(yearSalary, rateIO);
195         }
196 
197         /**
198          * Asks user if they want to apply a parking charge
199          *
200          * @return To apply parking charge or not
201          */
202         public boolean userApplyParking() {
203 
204             while (true) {
205                 System.out.println(
206                     "Do you want to apply a parking charge? (y/n)"
207                 );
208                 // Normalise characters to lowercase
209                 String parkingInput = scanner.nextLine().toLowerCase();
210                 switch (parkingInput) {
211                     case "y": {
212                         return true;
213                     }
214                     case "n": {
215                         return false;
216                     }
217                 }
218             }
219         }
220 
221         /**
222          * Asks the user if they want to apply a teacher's pension
223          *
224          * @return bool indicating to apply pension or not
225          */
226         public boolean userApplyPension() {
227             while (true) {
228                 System.out.println(
229                     "Do you want to apply a teachers pension? (y/n)"
230                 );
231                 // Normalise characters to lowercase
232                 String parkingInput = scanner.nextLine().toLowerCase();
233                 switch (parkingInput) {
234                     case "y": {
235                         return true;
236                     }
237                     case "n": {
238                         return false;
239                     }
240                 }
241             }
242         }
243     }
```

Employee.java

```java
package usw.employeepay;

public class Employee {

    private final int employeeNum;
    private final String name;
    private Salary employeeSalary;

    /**
     * Creates employee
     *
     * @param name        Employee name
     * @param employeeNum Employee number
     */
    public Employee(String name, int employeeNum) {
        this.name = name;
        this.employeeNum = employeeNum;
    }

    public String getName() {
        return name;
    }

    public int getEmployeeNum() {
        return employeeNum;
    }

    public Salary getSalary() {
        return employeeSalary;
    }

    /**
     * Adds Salary to Employee
     *
     * @param employeeSalary Salary object
     */
    public void setEmployeeSalary(Salary employeeSalary) {
        this.employeeSalary = employeeSalary;
    }
}
```

Salary.java

```java
package usw.employeepay;

import java.math.BigDecimal;
import java.math.RoundingMode;
import java.util.LinkedHashMap;
```

```java
import java.util.Map;

/**
 * Class that contains information and methods related to Salary.
 * Includes: income tax, national insurance, pensions, and
 * parking charges
 */
public class Salary {

    iRateIO rateIO;

    /*
     * BigDecimal used as we are working with money
     * Avoids errors concerning floating-point representation
     */
    private BigDecimal grossSalary;
    private BigDecimal netSalary;
    private BigDecimal totalDeductions = new BigDecimal("0");
    private BigDecimal totalIncomeTax;
    private BigDecimal totalNI;
    private BigDecimal totalPension;
    private BigDecimal totalParking;

    public Salary(BigDecimal grossSalary, iRateIO rateIO) {
        this.grossSalary = grossSalary;
        netSalary = grossSalary;
        this.rateIO = rateIO;
    }

    /**
     * Converts a yearly BigDecimal amount into its monthly amount and
     * rounds to 2 D.P
     * @param amount Value to divide by 12
     * @return BigDecimal containing the value divided by 12 and
     * rounded
     */
    public static BigDecimal convertMonthly(BigDecimal amount) {
        // Divide and round to 2 d.p with standard maths rounding
        return amount.divide(new BigDecimal("12"), 2, RoundingMode.
            HALF_UP);
    }

    /**
     * Applies required deductions: income tax, national insurance
     */
    public void applyMandatoryDeductions() {
        applyIncomeTax();
        applyNationalInsurance();
    }

    public void applyIncomeTax() {
```

```
56          totalIncomeTax = applyPaymentBands(grossSalary,
57              rateIO.getTaxBands()
58          );
59          totalDeductions = totalDeductions.add(totalIncomeTax);
60          netSalary = netSalary.subtract(totalIncomeTax);
61      }
62
63      public void applyNationalInsurance() {
64          totalNI = applyPaymentBands(grossSalary,
65              rateIO.getNationalInsurance()
66          );
67          totalDeductions = totalDeductions.add(totalNI);
68          netSalary = netSalary.subtract(totalNI);
69      }
70
71      public void applyPension() {
72          totalPension = applyPaymentBands(grossSalary,
73              rateIO.getPensionBands()
74          );
75          totalDeductions = totalDeductions.add(totalPension);
76          netSalary = netSalary.subtract(totalPension);
77      }
78
79      public void applyParkingCharge() {
80          // Monthly parking * 12
81          totalParking = rateIO.getMonthlyParking().multiply(
82              new BigDecimal("12")
83          );
84          totalDeductions = totalDeductions.add(totalParking);
85          netSalary = netSalary.subtract(totalParking);
86      }
87
88      /**
89       * Applies payment bands to income dynamically
90       *
91       * @param income       Accepts BigDecimals, no negatives
92       * @param paymentBands LinkedHashMap containing, the taxBand first,
93       *                     then the taxRate, overflow tax rates
94       *                     should be denoted with a negative
95       *                     on the band
96       * @return Total payment on income after paymentBands applied
97       */
98      private BigDecimal applyPaymentBands(BigDecimal income,
            LinkedHashMap<BigDecimal, BigDecimal> paymentBands) {
99
100         BigDecimal totalPayment = new BigDecimal("0");
101         BigDecimal previousBracket = new BigDecimal("0");
102
103         for (Map.Entry<BigDecimal, BigDecimal> entry :
104             paymentBands.entrySet()) {
105             BigDecimal currentBracket = entry.getKey();
```

```
106             BigDecimal bracketRate = entry.getValue;
107
108             /*
109              * If the payment is in a band denoted with a negative
110              * number then it is overflow, and applies
111              * that rate to rest of salary
112              */
113             if (currentBracket.compareTo(BigDecimal.ZERO) < 0) {
114                 /* totalPayment = totalPayment +
115                  * (income - previousBand) * taxRate
116                  */
117                 totalPayment = totalPayment.add(
118                     income.subtract(
119                         previousBracket
120                     ).multiply(
121                         bracketRate
122                     ).setScale(
123                         2, RoundingMode.HALF_UP
124                     )
125                 );
126
127             } else if (income.compareTo(entry.getKey()) > 0) {
128                 /* If the income is greater than the current
129                  * payment band
130                  */
131
132                 /* totalPayment = totalPayment +
133                  * (currentBracket - previousBand) * taxRate
134                  * It then rounds to 2 decimal places
135                  */
136                 totalPayment = totalPayment.add(
137                     (entry.getKey().subtract(
138                         previousBracket
139                     )).multiply(
140                         entry.getValue()
141                     ).setScale(
142                         2, RoundingMode.HALF_UP
143                     )
144                 );
145
146             } else if ((income.compareTo(previousBracket) > 0) &&
147                        (income.compareTo(entry.getKey()) < 0))
148             {
149                 /* If the income is smaller than the current payment
150                  * band
151                  */
152
153                 /* Get the leftover money in the band */
154                 BigDecimal bracketAmount = income.subtract(
155                     previousBracket
156                 );
```

```
157                 /* apply tax to the leftover amount in the band
158                  * totalPayment = totalPayment +
159                  * (leftoverAmount * taxRate)
160                  */
161                 totalPayment = totalPayment.add(
162                     bracketAmount.multiply(
163                         entry.getValue()
164                     ).setScale(
165                         2, RoundingMode.HALF_UP
166                     )
167                 );
168                 /* Since income is smaller than current band, won't
169                  * make it to next band, break out of loop
170                  */
171                 break;
172             }
173             previousBracket = entry.getKey();
174         }
175         return totalPayment;
176     }
177 
178     public void setSalary(BigDecimal grossSalary) {
179         this.grossSalary = grossSalary;
180         netSalary = grossSalary;
181         applyMandatoryDeductions();
182     }
183 
184     public void setRateIO(iRateIO rateIO) {
185         this.rateIO = rateIO;
186         applyMandatoryDeductions();
187     }
188 
189     public BigDecimal getGrossSalary() {
190         return grossSalary;
191     }
192 
193     public BigDecimal getMonthlySalary() {
194         return convertMonthly(grossSalary);
195     }
196 
197     public BigDecimal getTaxableAmount() {
198         return grossSalary.subtract(new BigDecimal("12570"));
199     }
200 
201     public BigDecimal getIncomeTaxAmount() {
202         return totalIncomeTax;
203     }
204 
205     public BigDecimal getNIAmount() {
206         return totalNI;
207     }
```

```
208
209     public BigDecimal getPensionAmount() {
210         return totalPension;
211     }
212
213     public BigDecimal getTotalParking() {
214         return totalParking;
215     }
216
217     public BigDecimal getTotalDeductions() {
218         return totalDeductions;
219     }
220
221     public BigDecimal getNetSalary() {
222         return netSalary;
223     }
224
225     public BigDecimal getMonthlyNetSalary() {
226         return netSalary.divide(
227             new BigDecimal("12"), 2, RoundingMode.HALF_UP
228         );
229     }
230 }
```

iRateIO.java

```
1  package usw.employeepay;
2
3  import java.math.BigDecimal;
4  import java.util.LinkedHashMap;
5
6  /**
7   * Interface for RateIO. Multiple implementations that use file
8   * reading, and mocked set values for testing purposes
9   */
10 public interface iRateIO {
11     LinkedHashMap<BigDecimal, BigDecimal> getTaxBands();
12
13     LinkedHashMap<BigDecimal, BigDecimal> getNationalInsurance();
14
15     LinkedHashMap<BigDecimal, BigDecimal> getPensionBands();
16
17     BigDecimal getMonthlyParking();
18 }
```

RateIO.java

```
1  package usw.employeepay;
```

```
2
3  import java.io.IOException;
4  import java.math.BigDecimal;
5  import java.nio.file.Files;
6  import java.nio.file.Paths;
7  import java.util.Arrays;
8  import java.util.LinkedHashMap;
9  import java.util.List;
10
11 public class RateIO implements iRateIO {
12     private final LinkedHashMap<BigDecimal, BigDecimal> taxBands =
13         new LinkedHashMap<>();
14     private final LinkedHashMap<BigDecimal, BigDecimal> pensionBands =
15         new LinkedHashMap<>();
16     private final LinkedHashMap<BigDecimal, BigDecimal>
17         nationalInsurance = new LinkedHashMap<>();
18     private BigDecimal monthlyParking;
19
20     /**
21      * Reads a CSV for tax bands, national insurance, and
22      * @param filePath String of file path
23      * @throws IOException If file does not exist / is not found
24      */
25     public RateIO(String filePath) throws IOException {
26         List<String> lines = Files.readAllLines(Paths.get(filePath));
27         // Each line runs the parseLine function
28         lines.forEach(line ->
29             parseLine(Arrays.asList(line.split(",")))
30         );
31     }
32
33     /**
34      * Handle the separated line and add it to a tax band
35      * @param line Line to parse
36      */
37     private void parseLine(List<String> line) {
38         /* Each type of deduction possible in CSV */
39         switch (line.get(0)) {
40             case "tax" -> taxBands.put(
41                 new BigDecimal(line.get(1)),
42                 new BigDecimal(line.get(2))
43             );
44             case "pension" -> pensionBands.put(
45                 new BigDecimal(line.get(1)),
46                 new BigDecimal(line.get(2))
47             );
48             case "nationalInsurance" -> nationalInsurance.put(
49                 new BigDecimal(line.get(1)),
50                 new BigDecimal(line.get(2))
51             );
52             case "parking" -> monthlyParking = (
```

```
                    new BigDecimal(line.get(1))
                );
            }
        }

        public LinkedHashMap<BigDecimal, BigDecimal> getTaxBands() {
            return taxBands;
        }

        public LinkedHashMap<BigDecimal, BigDecimal> getNationalInsurance()
        {
            return nationalInsurance;
        }

        public LinkedHashMap<BigDecimal, BigDecimal> getPensionBands() {
            return pensionBands;
        }

        public BigDecimal getMonthlyParking() {
            return monthlyParking;
        }
    }
```

Program Unit Tests

SalaryTest.java

```
package usw.employeepay;

import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;

import java.math.BigDecimal;

import static org.junit.jupiter.api.Assertions.assertEquals;

class SalaryTest {

    /* Use the fake rateIO, changing CSV won't mess up unit tests */
    TestingFakeRateIO testingRateIO = new TestingFakeRateIO();

    /* Various salaries to test different tax bands */
    Salary testSalary = new Salary(
        new BigDecimal("45000"), testingRateIO
    );
    Salary testSalaryDecimal = new Salary(
        new BigDecimal("50000"), testingRateIO
    );
    Salary testSalaryLarge = new Salary(
```

```
23          new BigDecimal("140000"), testingRateIO
24      );
25
26      @Test
27      @DisplayName("Calculate monthly salary")
28      public void monthlySalaryCalculations() {
29          BigDecimal expectedMonthlySalary2 = new BigDecimal("3750");
30
31          assertEquals(0, expectedMonthlySalary2.compareTo(
32              testSalary.getMonthlySalary()
33          ));
34
35          BigDecimal expectedMonthlySalary1 = new BigDecimal("4166.67");
36
37          assertEquals(0, expectedMonthlySalary1.compareTo(
38              testSalaryDecimal.getMonthlySalary()
39          ));
40
41      }
42
43      @Test
44      @DisplayName("Calculate taxable amount")
45      public void getTaxableAmount() {
46          BigDecimal expectedTaxableAmount = new BigDecimal("32430.00");
47
48          assertEquals(0, expectedTaxableAmount.compareTo(
49              testSalary.getTaxableAmount()
50          ));
51      }
52
53      @Test
54      @DisplayName("Calculate and apply income tax")
55      public void calculateIncomeTax() {
56          BigDecimal expectedTax = new BigDecimal("6486");
57          testSalary.applyIncomeTax();
58
59          assertEquals(0, expectedTax.compareTo(
60              testSalary.getIncomeTaxAmount()
61          ));
62
63          BigDecimal expectedTaxLarge = new BigDecimal("44175");
64          testSalaryLarge.applyIncomeTax();
65
66          assertEquals(0, expectedTaxLarge.compareTo(
67              testSalaryLarge.getIncomeTaxAmount()
68          ));
69      }
70
71      @Test
72      @DisplayName("Calculate and apply national insurance")
73      void calculateNationalInsurance() {
```

```
74          BigDecimal expectedNI = new BigDecimal("4251.84");
75          testSalary.applyNationalInsurance();
76
77          assertEquals(0, expectedNI.compareTo(
78              testSalary.getNIAmount()
79          ));
80      }
81
82      @Test
83      @DisplayName("Calculate and apply parking charge")
84      void useParkingCharge() {
85          BigDecimal expectedNetSalary = new BigDecimal("34142.16");
86          BigDecimal monthlyParking = new BigDecimal("120.00");
87          testSalary.applyMandatoryDeductions();
88          testSalary.applyParkingCharge();
89
90          assertEquals(0, monthlyParking.compareTo(
91              testSalary.getTotalParking()
92          ));
93          assertEquals(0, expectedNetSalary.compareTo(
94              testSalary.getNetSalary()
95          ));
96      }
97
98      @Test
99      @DisplayName("Calculate and apply teachers pension")
100     void getTotalTeachersPension() {
101         BigDecimal expectedTeachersPension = new BigDecimal("3501.76");
102         testSalary.applyPension();
103
104         assertEquals(0, expectedTeachersPension.compareTo(
105             testSalary.getPensionAmount()
106         ));
107     }
108
109     @Test
110     @DisplayName("Calculate total deductions")
111     void getTotalDeductions() {
112         BigDecimal expectedDeductions = new BigDecimal("10737.84");
113         testSalary.applyMandatoryDeductions();
114
115         assertEquals(0, expectedDeductions.compareTo(
116             testSalary.getTotalDeductions()
117         ));
118     }
119
120     @Test
121     @DisplayName("Calculate net salary")
122     void getNetSalary() {
123         BigDecimal expectedNetSalary = new BigDecimal("34142.16");
124         testSalary.applyMandatoryDeductions();
```

```
125             testSalary.applyParkingCharge();
126
127             assertEquals(0, expectedNetSalary.compareTo(
128                 testSalary.getNetSalary()
129             ));
130         }
131     }
```

RateIOTest.java

```
1
2   package usw.employeepay;
3
4   import org.junit.jupiter.api.BeforeEach;
5   import org.junit.jupiter.api.DisplayName;
6   import org.junit.jupiter.api.Test;
7
8   import java.io.IOException;
9   import java.math.BigDecimal;
10  import java.util.LinkedHashMap;
11
12  import static org.junit.jupiter.api.Assertions.assertEquals;
13
14  class RateIOTest {
15      private RateIO rateIO;
16
17      @BeforeEach
18      void setUp() {
19          try {
20              rateIO = new RateIO("rates.csv");
21
22          } catch (IOException e) {
23              System.out.println(e);
24          }
25      }
26
27      @Test
28      @DisplayName("CSV tax bands")
29      void getTaxBands() {
30          /* Create test LinkedHashMap to check against read file */
31          LinkedHashMap<BigDecimal, BigDecimal> expectedTaxBands =
32              new LinkedHashMap<>();
33          expectedTaxBands.put(
34              new BigDecimal("12570"), new BigDecimal("0.00")
35          );
36          expectedTaxBands.put(
37              new BigDecimal("50270"), new BigDecimal("0.20")
38          );
39          expectedTaxBands.put(
40              new BigDecimal("125140"), new BigDecimal("0.40")
```

```
41          );
42          expectedTaxBands.put(
43              new BigDecimal("-1"), new BigDecimal("0.45")
44          );
45          assertEquals(expectedTaxBands, rateIO.getTaxBands());
46      }
47
48      @Test
49      @DisplayName("CSV NI bands")
50      void getNationalInsurance() {
51          /* Create test LinkedHashMap to check against read file */
52          LinkedHashMap<BigDecimal, BigDecimal> expectedNationalInsurance
                =
53              new LinkedHashMap<>();
54          expectedNationalInsurance.put(
55              new BigDecimal("9568"), new BigDecimal("0.00")
56          );
57          expectedNationalInsurance.put(
58              new BigDecimal("-1"), new BigDecimal("0.12")
59          );
60          assertEquals(expectedNationalInsurance, rateIO.
                getNationalInsurance());
61      }
62
63      @Test
64      @DisplayName("CSV pension bands")
65      void getPensionBands() {
66          /* Create test LinkedHashMap to check against read file */
67          LinkedHashMap<BigDecimal, BigDecimal> expectedPensionBands =
68              new LinkedHashMap<>();
69
70          expectedPensionBands.put(
71              new BigDecimal("32135.99"), new BigDecimal("0.074")
72          );
73          expectedPensionBands.put(
74              new BigDecimal("43259.99"), new BigDecimal("0.086")
75          );
76          expectedPensionBands.put(
77              new BigDecimal("51292.99"), new BigDecimal("0.096")
78          );
79          expectedPensionBands.put(
80              new BigDecimal("67980.99"), new BigDecimal("0.102")
81          );
82          expectedPensionBands.put(
83              new BigDecimal("92597.99"), new BigDecimal("0.113")
84          );
85          expectedPensionBands.put(
86              new BigDecimal("-1"), new BigDecimal("0.117")
87          );
88          assertEquals(expectedPensionBands, rateIO.getPensionBands());
89      }
```

```
90
91      @Test
92      @DisplayName("CSV parking fee")
93      void getMonthlyParking() {
94          /* Test BigDecimal to compare to read file */
95          BigDecimal expectedMonthlyParking = new BigDecimal("10.00");
96          /* Check if they are equal */
97          assertEquals(0, expectedMonthlyParking.compareTo(
98              rateIO.getMonthlyParking())
99          );
100     }
101 }
```

`UserInterfaceTest.java`

```
1  package usw.employeepay;
2
3  import org.junit.jupiter.api.DisplayName;
4  import org.junit.jupiter.api.Test;
5
6  import java.io.ByteArrayInputStream;
7  import java.util.Scanner;
8
9  class UserInterfaceTest {
10
11     @Test
12     @DisplayName("Valid input in name field")
13     void nameValidInput() {
14
15         String dataIn = "Jake Real\n4324324\n423432";
16         ByteArrayInputStream in = new ByteArrayInputStream(
17             dataIn.getBytes()
18         );
19         System.setIn(in);
20
21         Scanner scanner = new Scanner(System.in);
22
23         UserInterface userInput = new UserInterface(scanner);
24         userInput.createEmployeeLoop();
25     }
26 }
```

Program Outputs

Running, `Main.java` with typical inputs,

```
1  Welcome to USW Employee Salary Calculator
2  ----------------------------------------
```

```
Employee Name: jake
Employee number: 43232
Yearly salary: 45000
45000.00
Do you want to apply a parking charge? (y/n)
n
Do you want to apply a teachers pension? (y/n)
n

Calculating yearly net pay...

Gross salary: £45000.00
Taxable amount: £32430.00
Tax paid: £6486.00
National insurance paid: £4251.84

Total deductions: £10737.84
Yearly net pay: £34262.16

Calculating monthly net pay...

Gross salary: £3750.00
Taxable amount: £2702.50
Tax paid: £540.50
National insurance paid: £354.32

Monthly total deductions: £894.82
Monthly net pay: £2855.18
```

Testing input validation:

Employee number,

```
Welcome to USW Employee Salary Calculator
----------------------------------------
Employee Name: Jake
Employee number: -3242
Negative numbers not accepted
Employee number:
```

Salary,

```
Welcome to USW Employee Salary Calculator
----------------------------------------
Employee Name: Jake
Employee number: 5000
Yearly salary: -2
Negative salaries are not accepted
Yearly salary:
```

Unit Test Outputs,

```
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
[INFO] Running usw.employeepay.RateIOTest
[INFO] Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed:
    0.105 s -- in usw.employeepay.RateIOTest
[INFO] Running usw.employeepay.UserInterfaceTest
Welcome to USW Employee Salary Calculator
---------------------------------------
Employee Name: Employee number: [INFO] Tests run: 1, Failures: 0,
    Errors: 0, Skipped: 0, Time elapsed: 0.032 s -- in usw.employeepay.
    UserInterfaceTest
[INFO] Running usw.employeepay.SalaryTest
[INFO] Tests run: 8, Failures: 0, Errors: 0, Skipped: 0, Time elapsed:
    0.034 s -- in usw.employeepay.SalaryTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 13, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] -------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] -------------------------------------------------------
[INFO] Total time:  1.859 s
[INFO] Finished at: 2023-12-01T11:10:30Z
[INFO] -------------------------------------------------------
```

These tests include:

- RateIO
  - CSV tax bands
  - CSV NI bands
  - CSV pension bands
  - CSV parking fee
- Salary
  - Calculate monthly salary
  - Calculate and apply parking charge
  - Calculate taxable amount
  - Calculate total deductions
  - Calculate and apply national insurance
  - Calculate net salary
  - Calculate and apply income tax
  - Calculate and apply teachers pension

- UserInterface
  - Valid input in name field

All tests used the specification examples as test values.

Salary tests use a mock implementation of the interface `iRateIO` based on the coursework specification to avoid failing tests due to a change in the `RateIO` CSV file.

References

Baeldung (2023a) *A guide to LinkedHashMap in java*. Available at: https://www.cdc.gov/foodsafety/food-poisoning.html.

Baeldung (2023b) *Java interfaces*.

Oracle ([no date]) *Files (java platform SE 8)*. Available at: https://www.cdc.gov/foodsafety/food-poisoning.html.

The Apache Software Foundation (2023a) *Maven in 5 minutes*.

The Apache Software Foundation (2023b) *Maven surefire plugin - usage*.