
IS1S481 Coursework 1

Jake Real - 23056792

08/12/2023



Contents

Part A - Design Task	2
Task 1 User Login and Unique Pin	2
Task 2 Employee Pay Calculator	5
Part B - Programming Task	15
Task 1 User Login and Unique Pin	15
Source Code	15
Program Unit Tests	18
Program Outputs	18
Task 2 - Employee Pay Calculator	20
Program Source Code	20
Program Unit Tests	33
CSV File	38
Program Outputs	39
References	41

Part A - Design Task

Task 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.

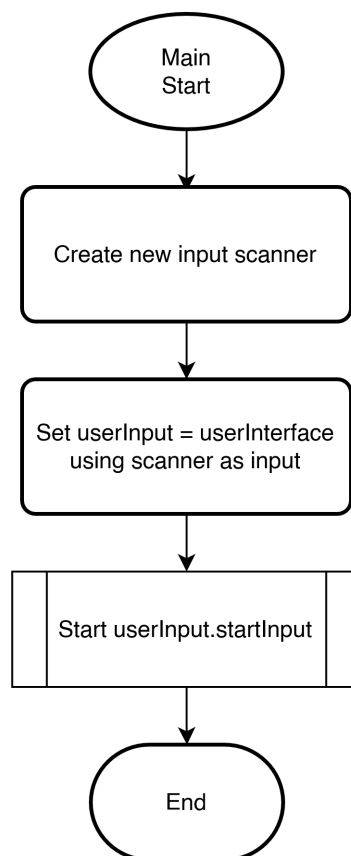
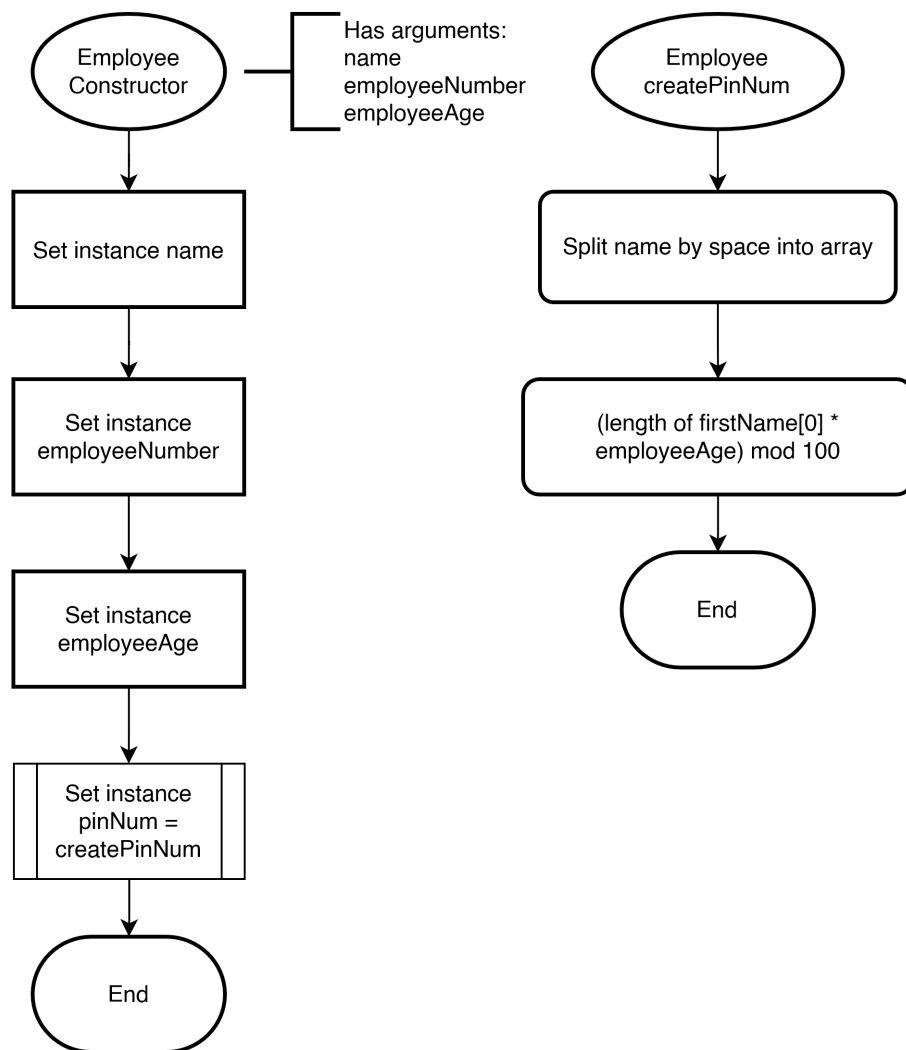
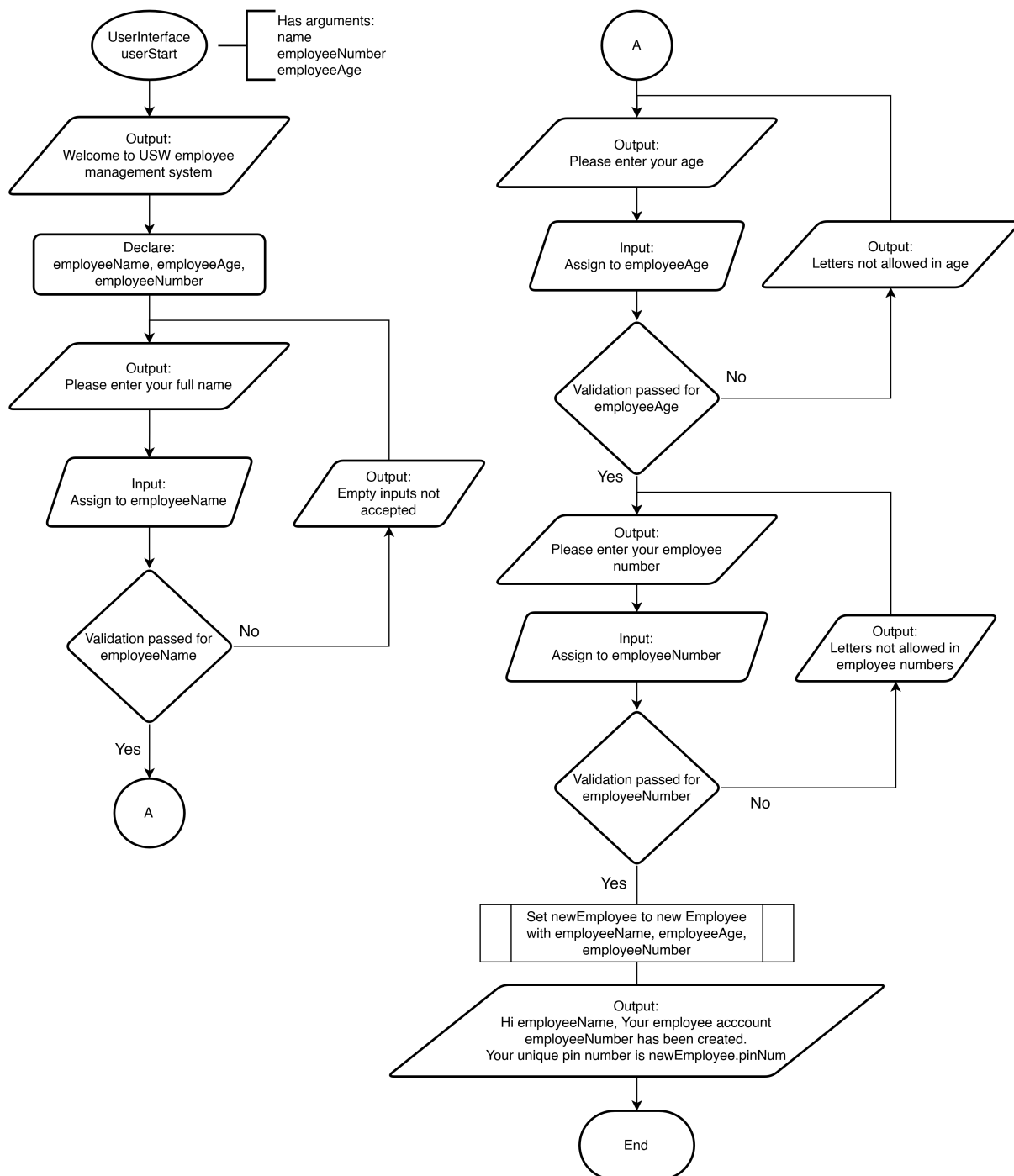


Figure 1: Flowchart of Main

**Figure 2:** Flowchart of Employee

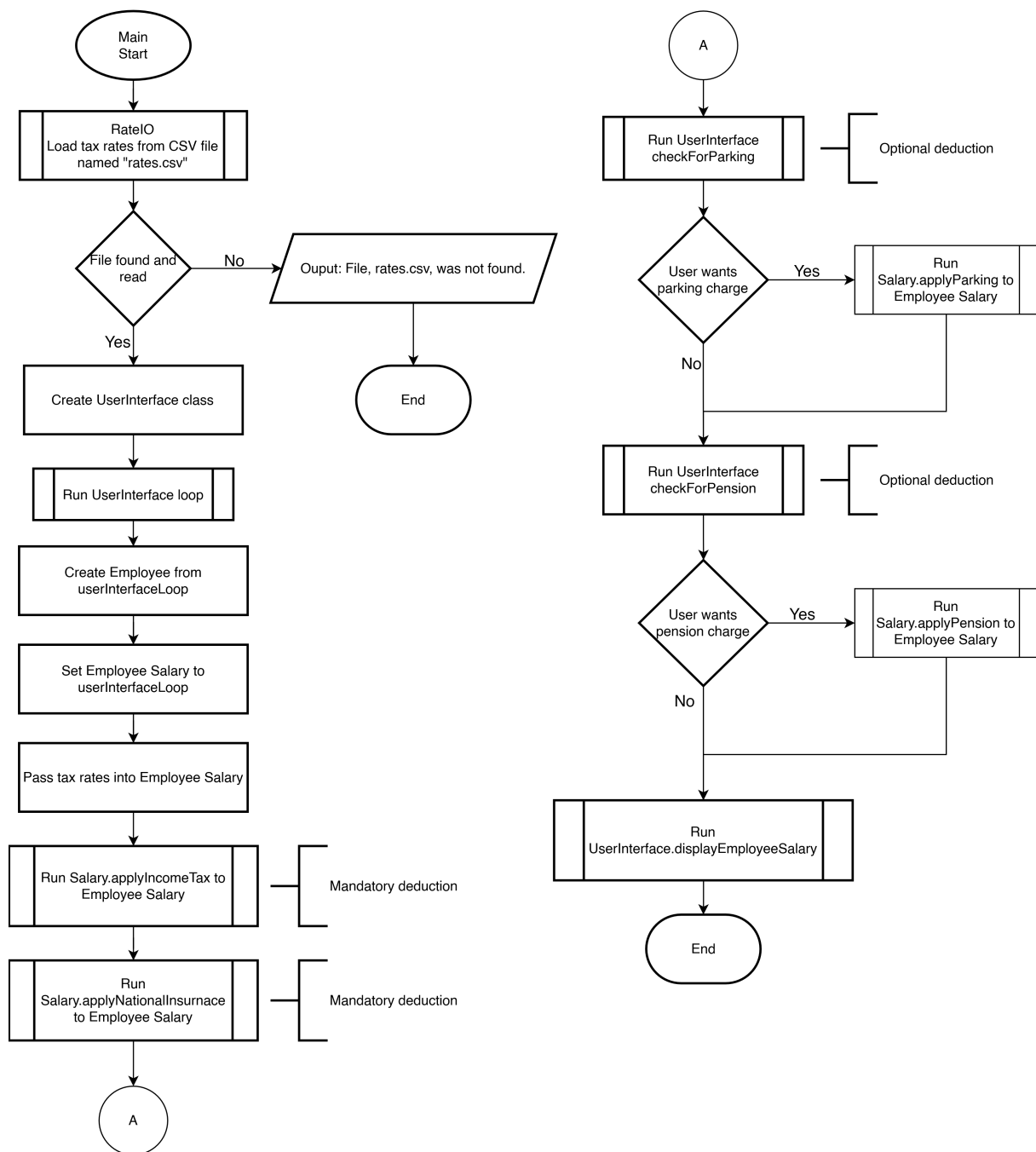
**Figure 3:** Flowchart of UserInterface

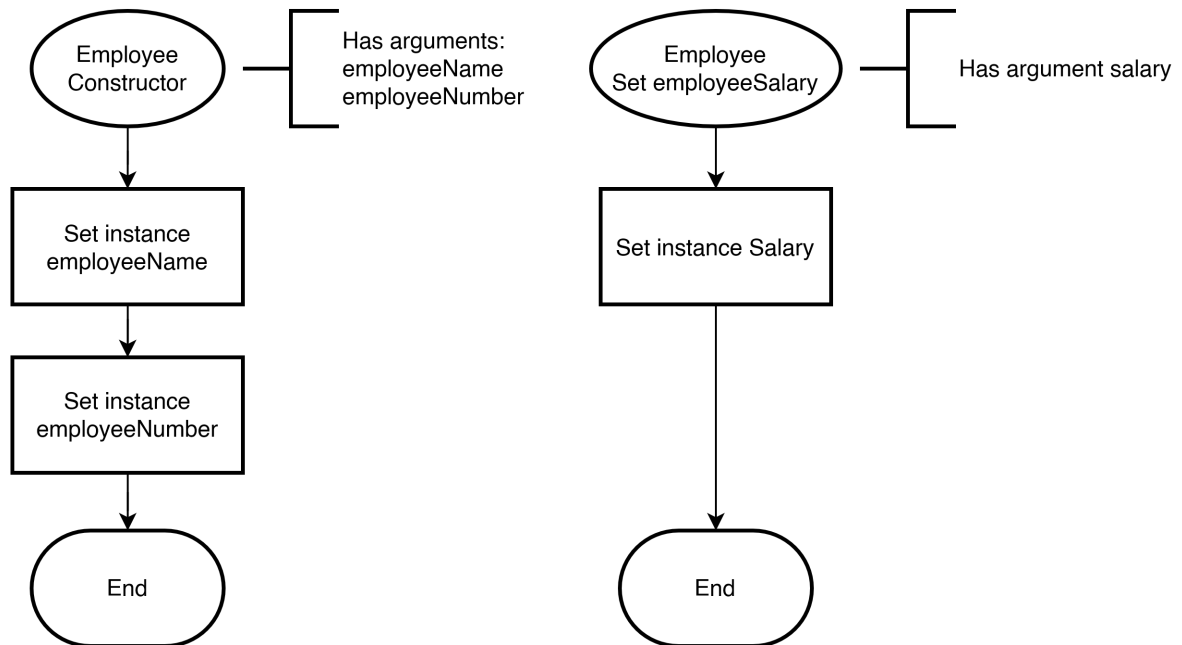
Task 2 Employee Pay Calculator

Design Decisions:

Several important design choices were made prior to starting on the flowcharts and program. The following choices were made. The initial designs would include the additional features to reduce refactoring and rewriting of code. This increased complexity as the tax calculation implementation had to be dynamic, adapting to different numbers and sizes of brackets. To adjust for this, test-driven development would be used when programming areas related to salary and file IO, allowing for quick and easy testing of the mathematical components to correct manually calculated test values. This increased my development speed as these tests could easily be quickly run with little user input. However, implementing effective unit tests requires dependency injection with mocked fake classes in methods related to file reading. The mock classes and unit tests have not been planned through flowcharts as they only serve ease development. Therefore, an effort was made to show dependency injection in the flow charts' argument comments.

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

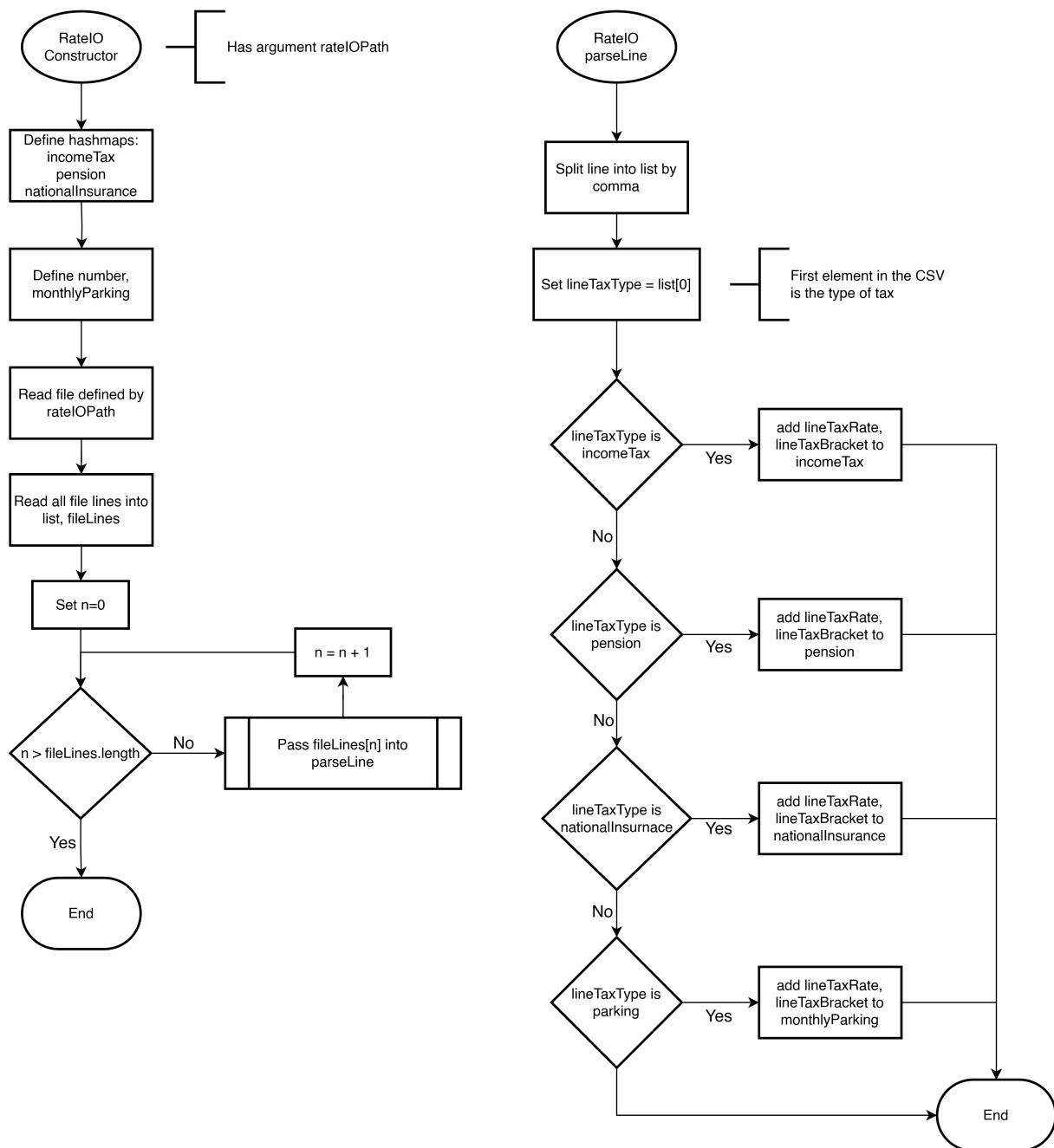
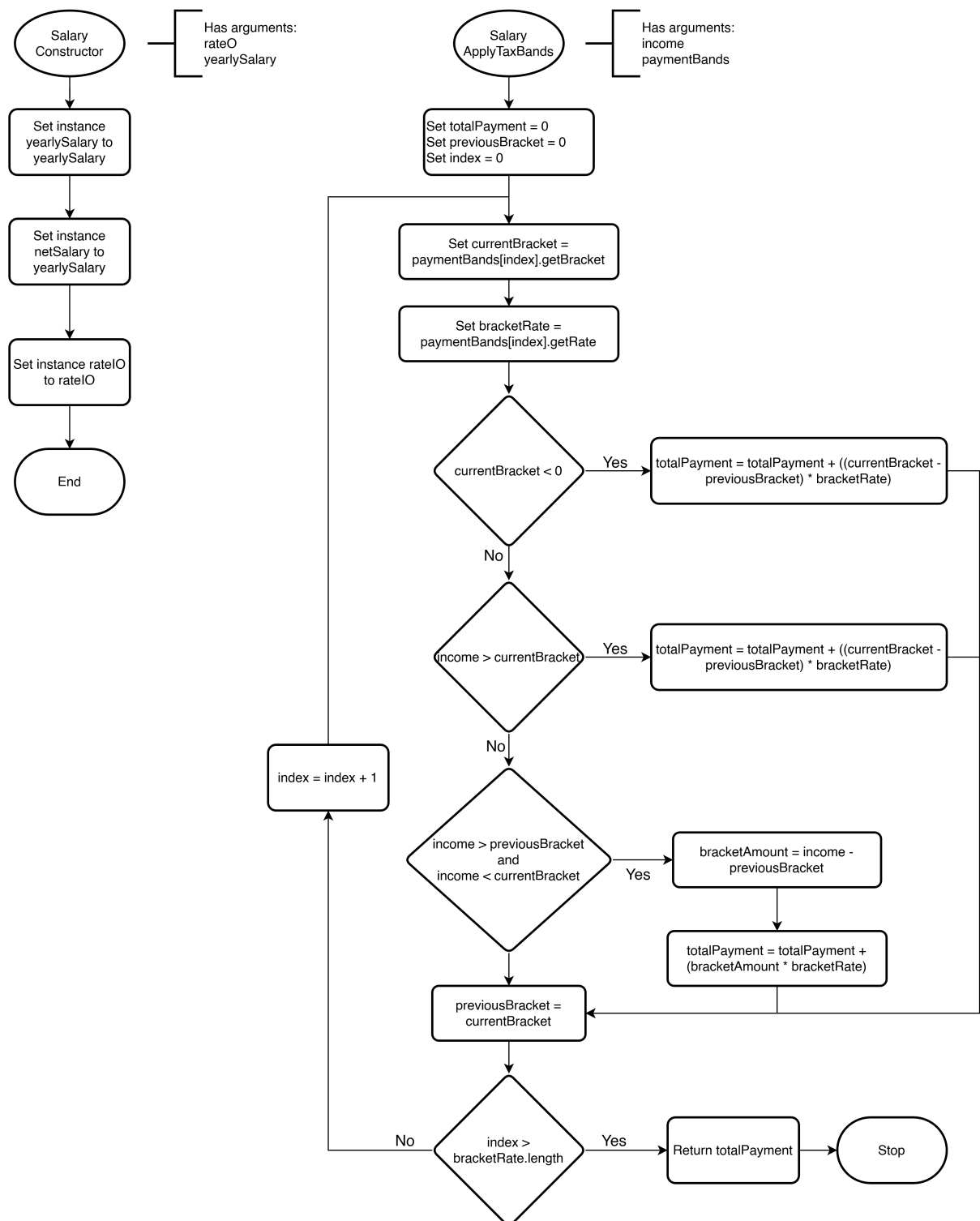
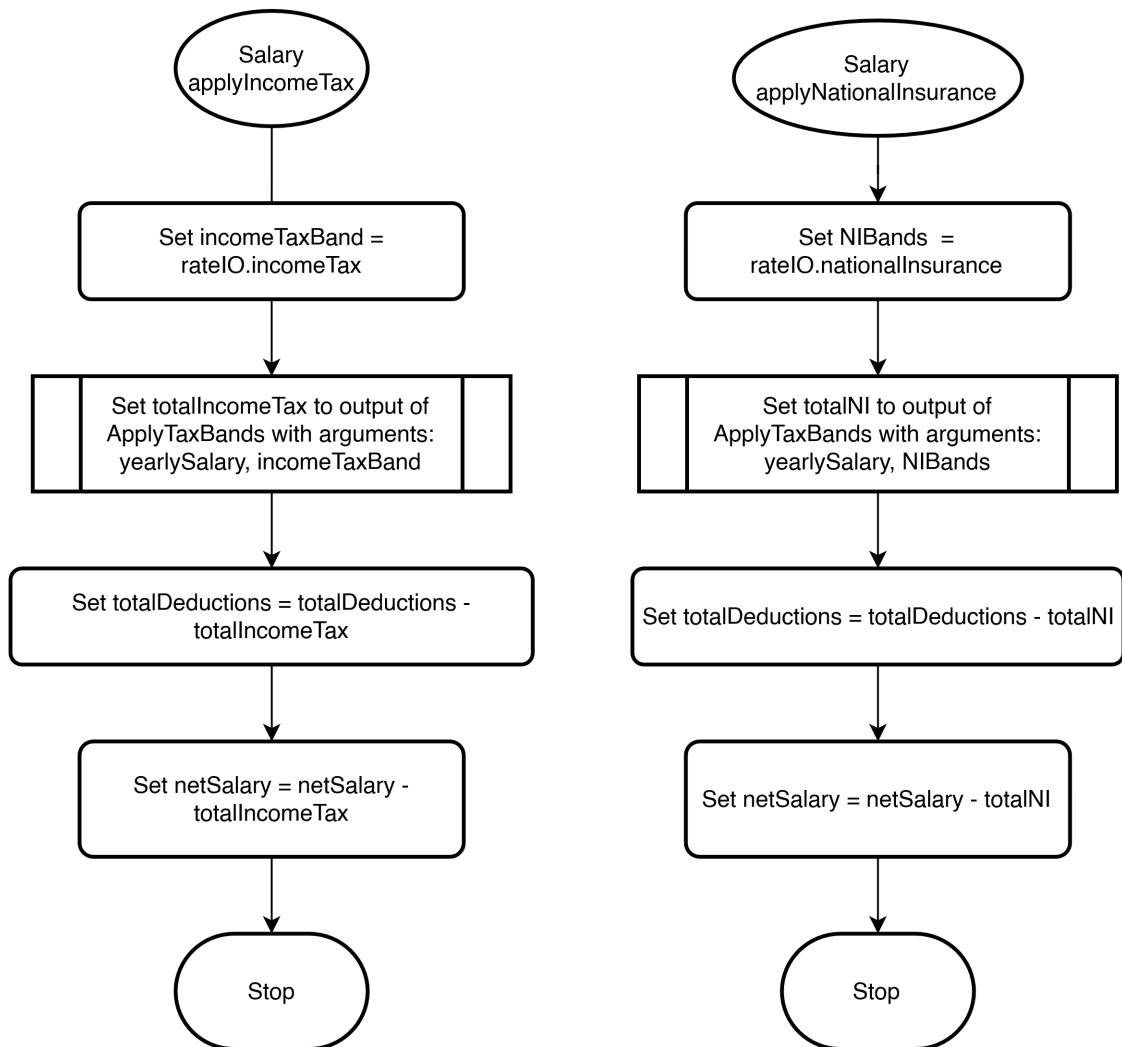
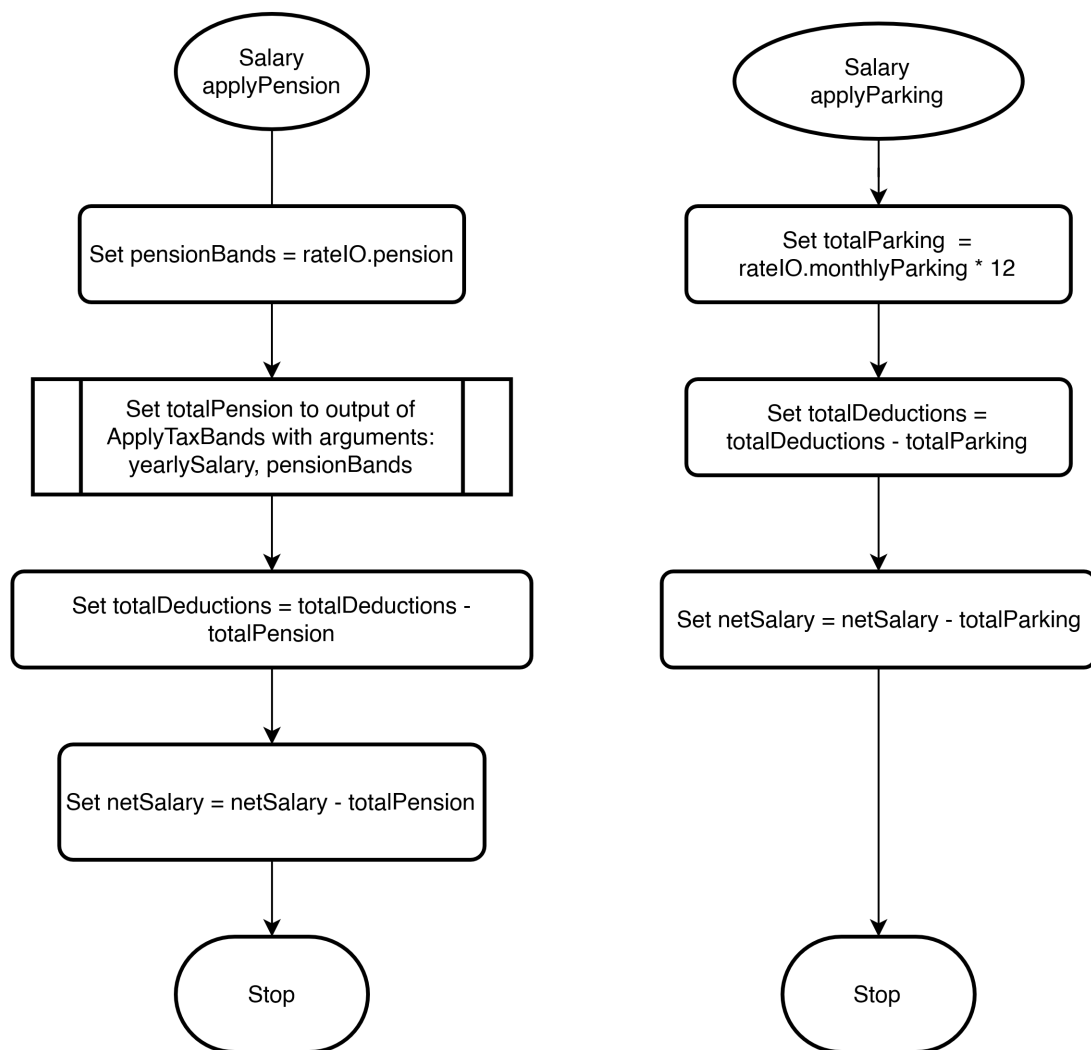


Figure 6: Flowchart of RateIO

**Figure 7:** Flowchart of Salary

**Figure 8:** 2nd Flowchart of Salary

**Figure 9:** 3rd Flowchart of Salary

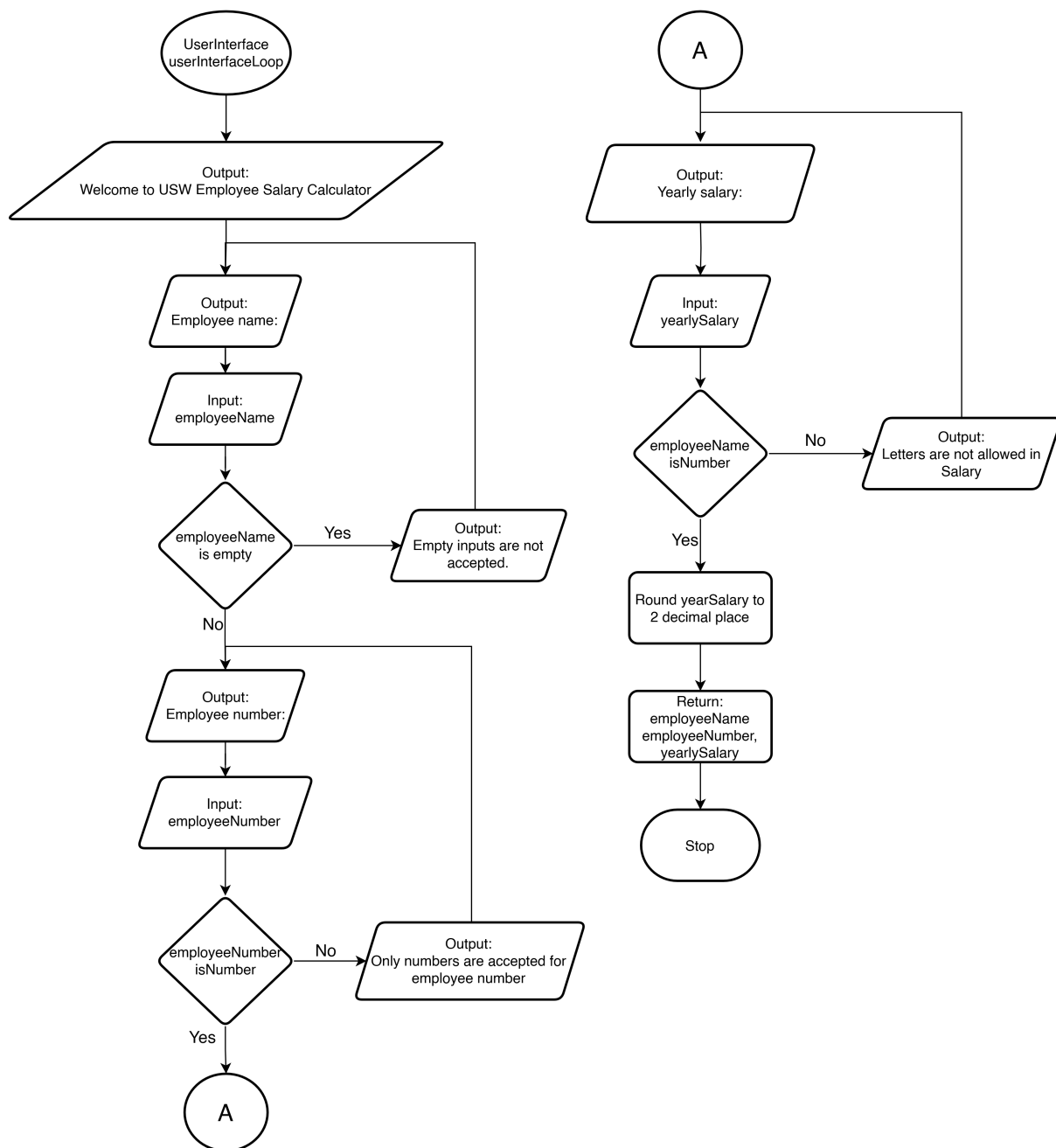
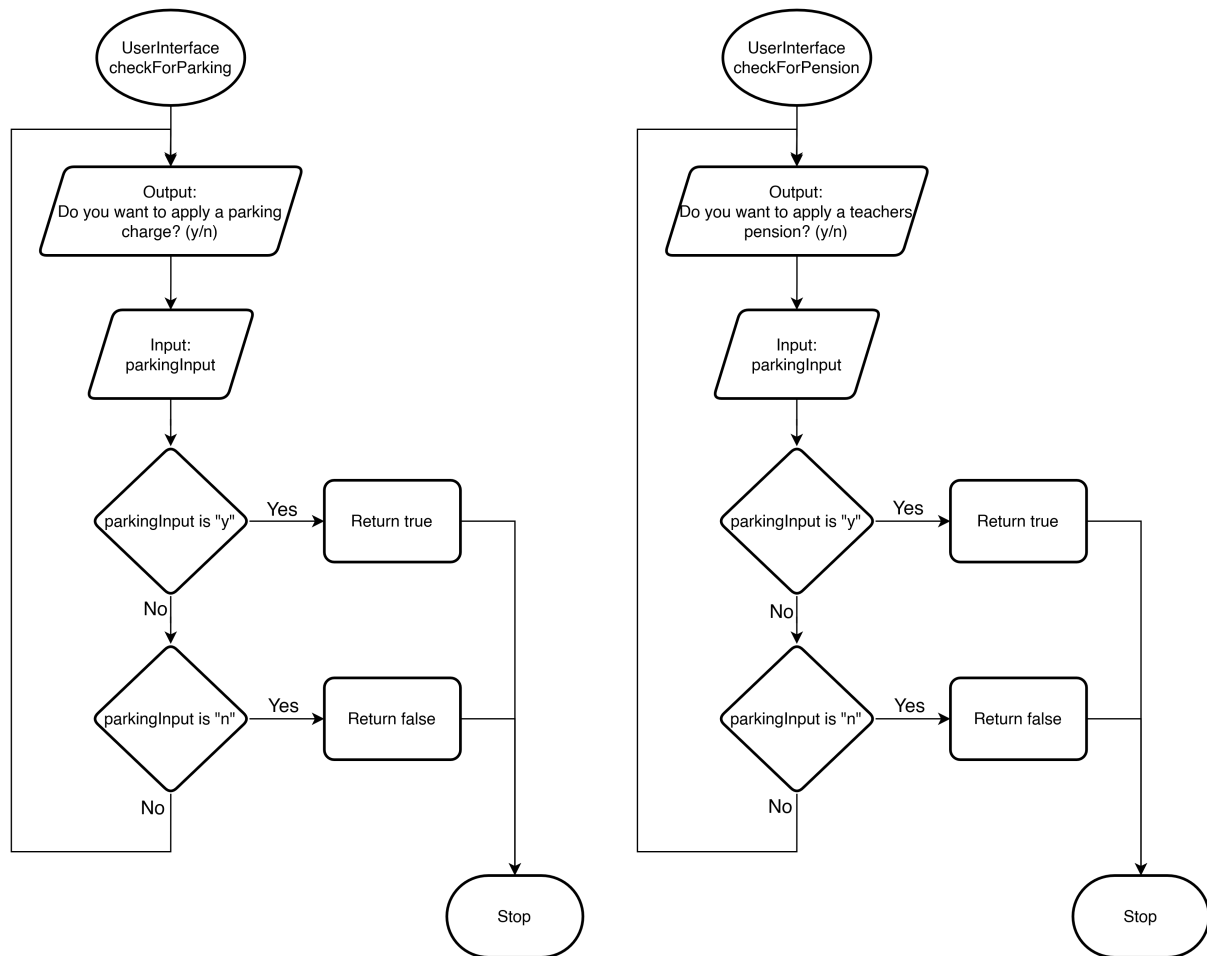
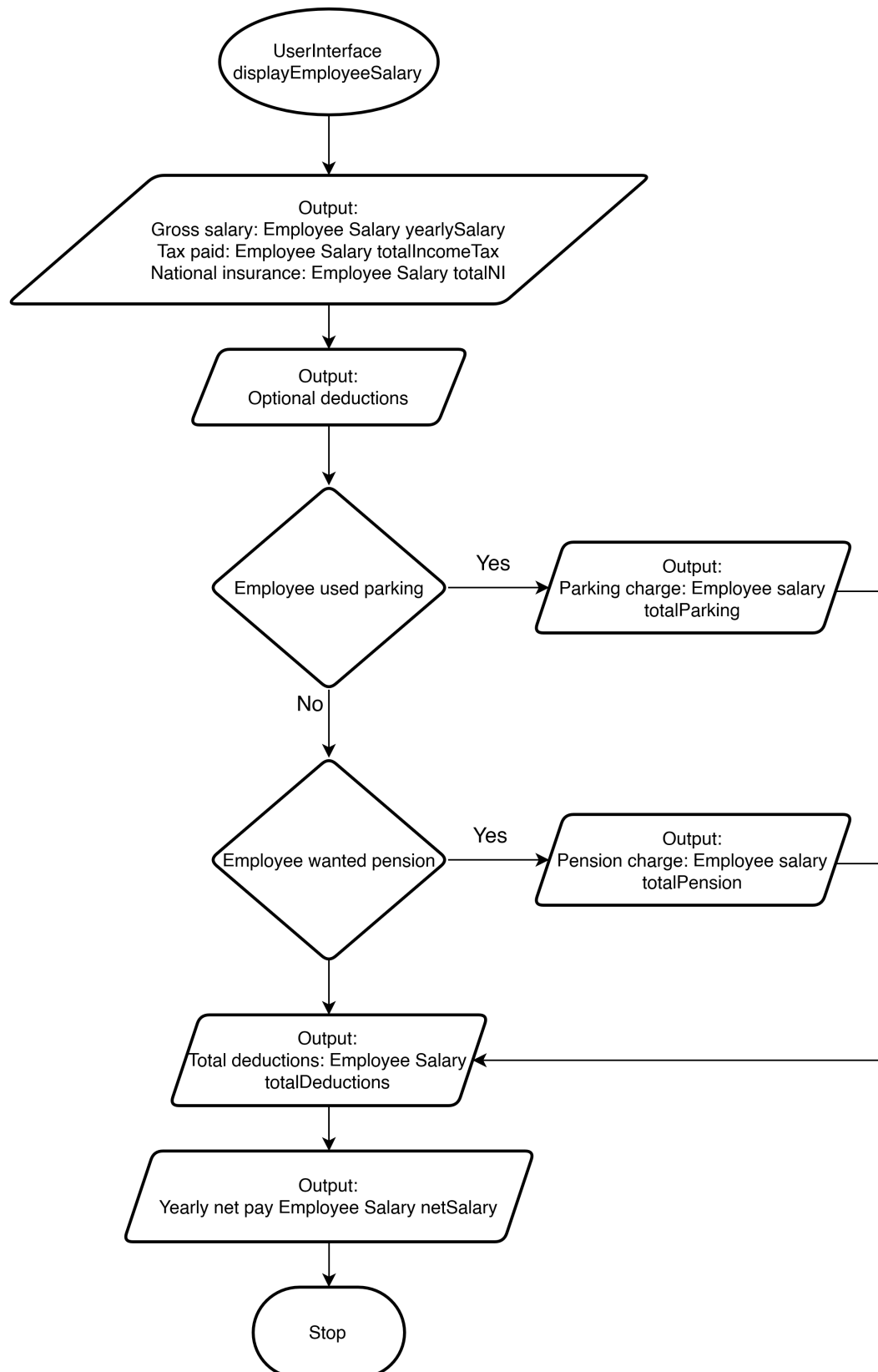


Figure 10: Flowchart of UserInterface

**Figure 11:** 2nd Flowchart of UserInterface

**Figure 12:** 3rd Flowchart of UserInterface

Part B - Programming Task

Task 1 User Login and Unique Pin

Source Code

Main.java

```
1 package usw.employeeLogin;
2
3 import java.util.Scanner;
4
5 public class Main {
6     public static void main(String[] args) {
7
8         Scanner scanner = new Scanner(System.in);
9         UserInterface userInput = new UserInterface(scanner);
10        userInput.userStart();
11    }
12 }
```

Employee.java

```
1 package usw.employeeLogin;
2
3 public class Employee {
4     private final String name;
5     private final int employeeNum;
6     private final int employeeAge;
7     private final int pinNum;
8
9     /**
10      * Creates an employee
11      *
12      * @param name      Name of employee
13      * @param employeeNum Number employee
14      * @param employeeAge Age of employee
15      */
16     public Employee(String name, int employeeNum, int employeeAge) {
17         this.name = name;
18         this.employeeNum = employeeNum;
19         this.employeeAge = employeeAge;
20         this.pinNum = createPinNum();
21     }
22
23     /**
24      * Internal class that returns code. Used in construction of class
25      */
26 }
```



```
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: ");
```

```
26         employeeName = scanner.nextLine();
27         if (!employeeName.isEmpty()) {
28             break;
29         }
30         System.out.println("Empty inputs are not accepted");
31     }
32
33     while (true) {
34         System.out.print("What's your age: ");
35         try {
36             employeeAge = scanner.nextInt();
37             if (employeeAge < 0) {
38                 System.out.println(
39                     "Negative ages not allowed"
40                 );
41                 scanner.nextLine();
42                 continue;
43             }
44             break;
45         } catch (InputMismatchException e) {
46             System.out.println(
47                 "Letters not allowed in age"
48             );
49             scanner.nextLine();
50         }
51     }
52
53     while (true) {
54         System.out.print("Please enter your employee number: ");
55         try {
56             employeeNumber = scanner.nextInt();
57             if (employeeNumber < 0) {
58                 System.out.println(
59                     "Negative employee numbers not allowed"
60                 );
61                 scanner.nextLine();
62                 continue;
63             }
64             break;
65         } catch (InputMismatchException e) {
66             System.out.println(
67                 "Letters not allowed in employee number"
68             );
69             scanner.nextLine();
70         }
71     }
72
73     Employee newEmployee = new Employee(employeeName,
74         employeeNumber, employeeAge
75     );
76     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

```
1  package usw.employeeLogin;  
2  
3  import org.junit.jupiter.api.DisplayName;  
4  import org.junit.jupiter.api.Test;  
5  
6  import static org.junit.jupiter.api.Assertions.assertEquals;  
7  
8  class EmployeeTest {  
9      Employee testEmployee = new Employee("jim", 330, 20);  
10     Employee testEmployeeLong = new Employee("12345", 203, 4321);  
11  
12     @Test  
13     @DisplayName("Pin number generated correctly")  
14     public void getPinNum() {  
15         assertEquals(60, testEmployee.getPinNum());  
16         assertEquals(605, testEmployeeLong.getPinNum());  
17     }  
18 }
```

Program Outputs

Running `Main.java` with typical inputs,

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

Using longer details,

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

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

Testing input validation:

Full name,

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

Age,

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

Employee number,

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

Unit Test Outputs,

```
1 [INFO] -----
2 [INFO]  T E S T S
3 [INFO] -----
4 [INFO] Running usw.employeelogin.EmployeeTest
5 [INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed:
    0.063 s - in usw.employeelogin.EmployeeTest
6 [INFO]
7 [INFO] Results:
8 [INFO]
9 [INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
10 [INFO]
11 [INFO]
12 [INFO] --- jar:3.3.0:jar (default-jar) @ pop-coursework ---
13 [INFO] Building jar: /home/jake/Code/usw/pop-coursework-1/target/pop-
    coursework-1.0-SNAPSHOT.jar
14 [INFO] -----
15 [INFO] BUILD SUCCESS
16 [INFO] -----
```

```
17 [INFO] Total time: 3.098 s
18 [INFO] Finished at: 2023-12-08T17:37:55Z
19 [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.

Task 2 - Employee Pay Calculator

Program Source Code

Main.java

```
1 package usw.employee;
2
3 import java.io.IOException;
4 import java.util.Scanner;
5
6 public class Main {
7     public static void main(String[] args) {
8         RateIO rateIO;
9         try {
10             rateIO = new RateIO("rates.csv");
11
12
13         } catch (IOException e) {
14             System.out.println("File, rates.csv, was not found. Make
15                 sure rates.csv is run in same folder as the " +
16                 "program");
17             return;
18         }
19         Scanner scanner = new Scanner(System.in);
20         UserInterface userInput = new UserInterface(scanner);
21         Employee employee = userInput.createEmployeeLoop();
22         employee.setEmployeeSalary(userInput.getSalaryLoop(rateIO));
23
24         /* Apply income tax and national insurance */
25         employee.getSalary().applyMandatoryDeductions();
26
27         /* Check if user wants to apply optional deductions */
28         if (userInput.userApplyParking()) {
29             employee.getSalary().applyParkingCharge();
30         }
31         if (userInput.userApplyPension()) {
32             employee.getSalary().applyPension();
33         }
34         UserInterface.displayEmployeeSalary(employee);
35     }
36 }
```

```
35 }
```

UserInterface.java

```
1 package usw.employeepay;
2
3 import java.math.BigDecimal;
4 import java.math.RoundingMode;
5 import java.util.InputMismatchException;
6 import java.util.Scanner;
7
8 public class UserInterface {
9
10     private final Scanner scanner;
11
12     /**
13      * Class that handles outputting and accepting user input
14      *
15      * @param scanner Input handling
16      */
17     public UserInterface(Scanner scanner) {
18         this.scanner = scanner;
19     }
20
21
22     /**
23      * Outputs the information concerning an employee's salary
24      *
25      * @param employee Employee to display salary of
26      */
27     public static void displayEmployeeSalary(Employee employee) {
28
29         System.out.println("\nCalculating yearly net pay...\n");
30         System.out.printf("Gross salary: £%s\n",
31             employee.getSalary().getGrossSalary(),
32             employee.getSalary().getTaxableAmount(),
33             employee.getSalary().getIncomeTaxAmount(),
34             employee.getSalary().getNIAmount());
35
36         /* Non-required deductions */
37         if (!(employee.getSalary().getTotalParking() == null)) {
38             System.out.printf("Parking charge: £%s\n",
39                 employee.getSalary().getTotalParking());
40         }
41     }
42
43
44
45
46 }
```

```
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().
90                 getTotalParking())
91         );
92     }
93
94     if (!(employee.getSalary().getPensionAmount() == null)) {
95         System.out.printf("Pension charge: £%s\n",
96             Salary.convertMonthly(employee.getSalary().
97                 getPensionAmount())
```

```
96         );
97     }
98
99     System.out.printf("\nMonthly total deductions: £%s\n",
100         Salary.convertMonthly(employee.getSalary().
101             getTotalDeductions())
102     );
103
104     System.out.printf("Monthly net pay: £%s\n",
105         employee.getSalary().getMonthlyNetSalary()
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
160  * information
161  *
162  * @param rateIO The tax bands to use in initial instantiation of
163  * taxes, pension, etc.
164  * @return Constructed Salary object
165  */
166 public Salary getSalaryLoop(RateIO rateIO) {
167     BigDecimal yearSalary;
168
169     while (true) {
170         System.out.print("Yearly salary: ");
171         try {
172             String inputSalary = scanner.next();
173             yearSalary = new BigDecimal(inputSalary);
174             yearSalary = yearSalary.setScale(2, RoundingMode.
175                 HALF_UP);
176             /* Clear the newline character from scanner buffer
177             * Otherwise next question would appear twice, as the
178             * scanner would pick up the leftover newline
179             */
180             scanner.nextLine();
181
182             // Check if the number is negative
183             if (yearSalary.compareTo(BigDecimal.ZERO) < 0) {
184                 System.out.println(
185                     "Negative salaries are not accepted"
186                 );
187                 continue;
188             }
189             break;
190         } catch (NumberFormatException e) {
191             System.out.println(
192                 "Letter are not allowed in the employee number"
193             );
194         }
195     }
196 }
```

```
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

```
1 package usw.employeepay;
2
3 public class Employee {
4
5     private final int employeeNum;
6     private final String name;
7     private Salary employeeSalary;
8
9     /**
10      * Creates employee
11      *
12      * @param name Employee name
13      * @param employeeNum Employee number
14      */
15     public Employee(String name, int employeeNum) {
16         this.name = name;
17         this.employeeNum = employeeNum;
18     }
19
20     public String getName() {
21         return name;
22     }
23
24     public int getEmployeeNum() {
25         return employeeNum;
26     }
27
28     public Salary getSalary() {
29         return employeeSalary;
30     }
31
32     /**
33      * Adds Salary to Employee
34      *
35      * @param employeeSalary Salary object
36      */
37     public void setEmployeeSalary(Salary employeeSalary) {
38         this.employeeSalary = employeeSalary;
39     }
40 }
```

Salary.java

```
1 package usw.employeepay;
2
3 import java.math.BigDecimal;
4 import java.math.RoundingMode;
5 import java.util.LinkedHashMap;
```

```
6 import java.util.Map;
7
8 /**
9  * Class that contains information and methods related to Salary.
10 * Includes: income tax, national insurance, pensions, and
11 * parking charges
12 */
13 public class Salary {
14
15     iRateIO rateIO;
16
17     /*
18     * BigDecimal used as we are working with money
19     * Avoids errors concerning floating-point representation
20     */
21     private BigDecimal grossSalary;
22     private BigDecimal netSalary;
23     private BigDecimal totalDeductions = new BigDecimal("0");
24     private BigDecimal totalIncomeTax;
25     private BigDecimal totalNI;
26     private BigDecimal totalPension;
27     private BigDecimal totalParking;
28
29     public Salary(BigDecimal grossSalary, iRateIO rateIO) {
30         this.grossSalary = grossSalary;
31         netSalary = grossSalary;
32         this.rateIO = rateIO;
33     }
34
35     /**
36     * Converts a yearly BigDecimal amount into its monthly amount and
37     * rounds to 2 D.P
38     * @param amount Value to divide by 12
39     * @return BigDecimal containing the value divided by 12 and
40     * rounded
41     */
42     public static BigDecimal convertMonthly(BigDecimal amount) {
43         // Divide and round to 2 d.p with standard maths rounding
44         return amount.divide(new BigDecimal("12"), 2, RoundingMode.
45             HALF_UP);
46     }
47
48     /**
49     * Applies required deductions: income tax, national insurance
50     */
51     public void applyMandatoryDeductions() {
52         applyIncomeTax();
53         applyNationalInsurance();
54     }
55
56     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,
99         LinkedHashMap<BigDecimal, BigDecimal> paymentBands) {
100
101         BigDecimal totalPayment = new BigDecimal("0");
102         BigDecimal previousBracket = new BigDecimal("0");
103
104         for (Map.Entry<BigDecimal, BigDecimal> entry :
105             paymentBands.entrySet()) {
106             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 = (
```

```
53         new BigDecimal(line.get(1))
54     );
55     }
56 }
57
58 public LinkedHashMap<BigDecimal, BigDecimal> getTaxBands() {
59     return taxBands;
60 }
61
62 public LinkedHashMap<BigDecimal, BigDecimal> getNationalInsurance()
63 {
64     return nationalInsurance;
65 }
66
67 public LinkedHashMap<BigDecimal, BigDecimal> getPensionBands() {
68     return pensionBands;
69 }
70
71 public BigDecimal getMonthlyParking() {
72     return monthlyParking;
73 }
74 }
```

Program Unit Tests

SalaryTest.java

```
1 package usw.employeepay;
2
3 import org.junit.jupiter.api.DisplayName;
4 import org.junit.jupiter.api.Test;
5
6 import java.math.BigDecimal;
7
8 import static org.junit.jupiter.api.Assertions.assertEquals;
9
10 class SalaryTest {
11
12     /* Use the fake rateIO, changing CSV won't mess up unit tests */
13     TestingFakeRateIO testingRateIO = new TestingFakeRateIO();
14
15     /* Various salaries to test different tax bands */
16     Salary testSalary = new Salary(
17         new BigDecimal("45000"), testingRateIO
18     );
19     Salary testSalaryDecimal = new Salary(
20         new BigDecimal("50000"), testingRateIO
21     );
22     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         } catch (IOException e) {
22             System.out.println(e);
23         }
24     }
25
26     @Test
27     @DisplayName("CSV tax bands")
28     void getTaxBands() {
29         /* Create test LinkedHashMap to check against read file */
30         LinkedHashMap<BigDecimal, BigDecimal> expectedTaxBands =
31             new LinkedHashMap<>();
32         expectedTaxBands.put(
33             new BigDecimal("12570"), new BigDecimal("0.00")
34         );
35         expectedTaxBands.put(
36             new BigDecimal("50270"), new BigDecimal("0.20")
37         );
38         expectedTaxBands.put(
39             new BigDecimal("125140"), new BigDecimal("0.40")
40         );
41     }
42 }
```

```
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             =
54             new LinkedHashMap<>();
55         expectedNationalInsurance.put(
56             new BigDecimal("9568"), new BigDecimal("0.00")
57         );
58         expectedNationalInsurance.put(
59             new BigDecimal("-1"), new BigDecimal("0.12")
60         );
61         assertEquals(expectedNationalInsurance, rateIO.
62             getNationalInsurance());
63     }
64
65     @Test
66     @DisplayName("CSV pension bands")
67     void getPensionBands() {
68         /* Create test LinkedHashMap to check against read file */
69         LinkedHashMap<BigDecimal, BigDecimal> expectedPensionBands =
70             new LinkedHashMap<>();
71         expectedPensionBands.put(
72             new BigDecimal("32135.99"), new BigDecimal("0.074")
73         );
74         expectedPensionBands.put(
75             new BigDecimal("43259.99"), new BigDecimal("0.086")
76         );
77         expectedPensionBands.put(
78             new BigDecimal("51292.99"), new BigDecimal("0.096")
79         );
80         expectedPensionBands.put(
81             new BigDecimal("67980.99"), new BigDecimal("0.102")
82         );
83         expectedPensionBands.put(
84             new BigDecimal("92597.99"), new BigDecimal("0.113")
85         );
86         expectedPensionBands.put(
87             new BigDecimal("-1"), new BigDecimal("0.117")
88         );
89         assertEquals(expectedPensionBands, rateIO.getPensionBands());
90     }
```

```
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 }
```

CSV File

The CSV file used the following format scheme for band based deductions,

deduction type, bracket upper bound, deduction percentage

-1 in the bracket upper bound denotes an overflow bracket. This means that the bracket continues indefinitely.

The parking charge, which can not be applied through band calculations, uses the below format,

deduction type, monthly amount

```
1 tax,12570,0.00
2 tax,50270,0.20
3 tax,125140,0.40
4 tax,-1,0.45
5 pension,32135.99,0.074
6 pension,43259.99,0.086
7 pension,51292.99,0.096
8 pension,67980.99,0.102
9 pension,92597.99,0.113
10 pension,-1,0.117
11 nationalInsurance,9568,0.00
12 nationalInsurance,-1,0.12
13 parking,10
```

Program Outputs

Running, `Main.java` with typical inputs,

```
1 Welcome to USW Employee Salary Calculator
2 -----
3 Employee Name: jake
4 Employee number: 43232
5 Yearly salary: 45000
6 45000.00
7 Do you want to apply a parking charge? (y/n)
8 n
9 Do you want to apply a teachers pension? (y/n)
10 n
11
12 Calculating yearly net pay...
13
14 Gross salary: £45000.00
15 Taxable amount: £32430.00
16 Tax paid: £6486.00
17 National insurance paid: £4251.84
18
19 Total deductions: £10737.84
20 Yearly net pay: £34262.16
```



```
21
22 Calculating monthly net pay...
23
24 Gross salary: £3750.00
25 Taxable amount: £2702.50
26 Tax paid: £540.50
27 National insurance paid: £354.32
28
29 Monthly total deductions: £894.82
30 Monthly net pay: £2855.18
```

Testing input validation:

Employee number,

```
1 Welcome to USW Employee Salary Calculator
2 -----
3 Employee Name: Jake
4 Employee number: -3242
5 Negative numbers not accepted
6 Employee number:
```

Salary,

```
1 Welcome to USW Employee Salary Calculator
2 -----
3 Employee Name: Jake
4 Employee number: 5000
5 Yearly salary: -2
6 Negative salaries are not accepted
7 Yearly salary:
```

Unit Test Outputs,

```
1 [INFO] -----
2 [INFO]  T E S T S
3 [INFO] -----
4 [INFO] Running usw.employeepay.RateIOTest
5 [INFO] Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed:
    0.105 s -- in usw.employeepay.RateIOTest
6 [INFO] Running usw.employeepay.UserInterfaceTest
7 Welcome to USW Employee Salary Calculator
8 -----
9 Employee Name: Employee number: [INFO] Tests run: 1, Failures: 0,
    Errors: 0, Skipped: 0, Time elapsed: 0.032 s -- in usw.employeepay.
    UserInterfaceTest
10 [INFO] Running usw.employeepay.SalaryTest
11 [INFO] Tests run: 8, Failures: 0, Errors: 0, Skipped: 0, Time elapsed:
    0.034 s -- in usw.employeepay.SalaryTest
12 [INFO]
13 [INFO] Results:
```

```
14 [INFO]
15 [INFO] Tests run: 13, Failures: 0, Errors: 0, Skipped: 0
16 [INFO]
17 [INFO] -----
18 [INFO] BUILD SUCCESS
19 [INFO] -----
20 [INFO] Total time: 1.859 s
21 [INFO] Finished at: 2023-12-01T11:10:30Z
22 [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://docs.oracle.com/javase/8/docs/api/java/nio/file/Files.html>.

Baeldung (2023b) *Java interfaces*. Available at: <https://www.baeldung.com/java-interfaces>.

Oracle ([no date]) *Files (java platform SE 8)*. Available at: <https://docs.oracle.com/javase/8/docs/api/java/nio/file/Files.html>.

The Apache Software Foundation (2023a) *Maven in 5 minutes*. Available at: <https://maven.apache.org/guides/getting-started/maven-in-five-minutes.html>.

The Apache Software Foundation (2023b) *Maven surefire plugin - usage*. Available at: <https://maven.apache.org/surefire/maven-surefire-plugin/usage.html>.