
IS1S481 Coursework 1

Jake Real - 23056792

08/12/2023



Contents

Part A - Design Task	2
Part 1 User Login and Unique Pin	2
Part 2 - Employee Pay Calculator	5
Part B - Programming Task	14
Part 1 User Login and Unique Pin	14
Source Code	14
Program Unit Tests	17
Program Outputs	18
Part 2 - Employee Pay Calculator	18
Program Source Code	18
Program Unit Tests	30
Program Outputs	36
References	38

Part A - Design Task

Part 1 User Login and Unique Pin

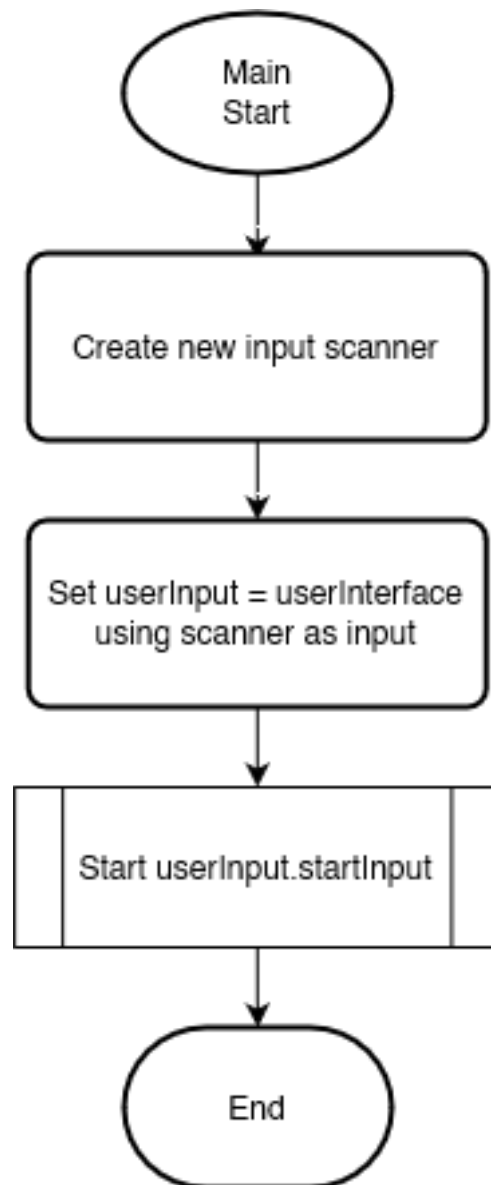
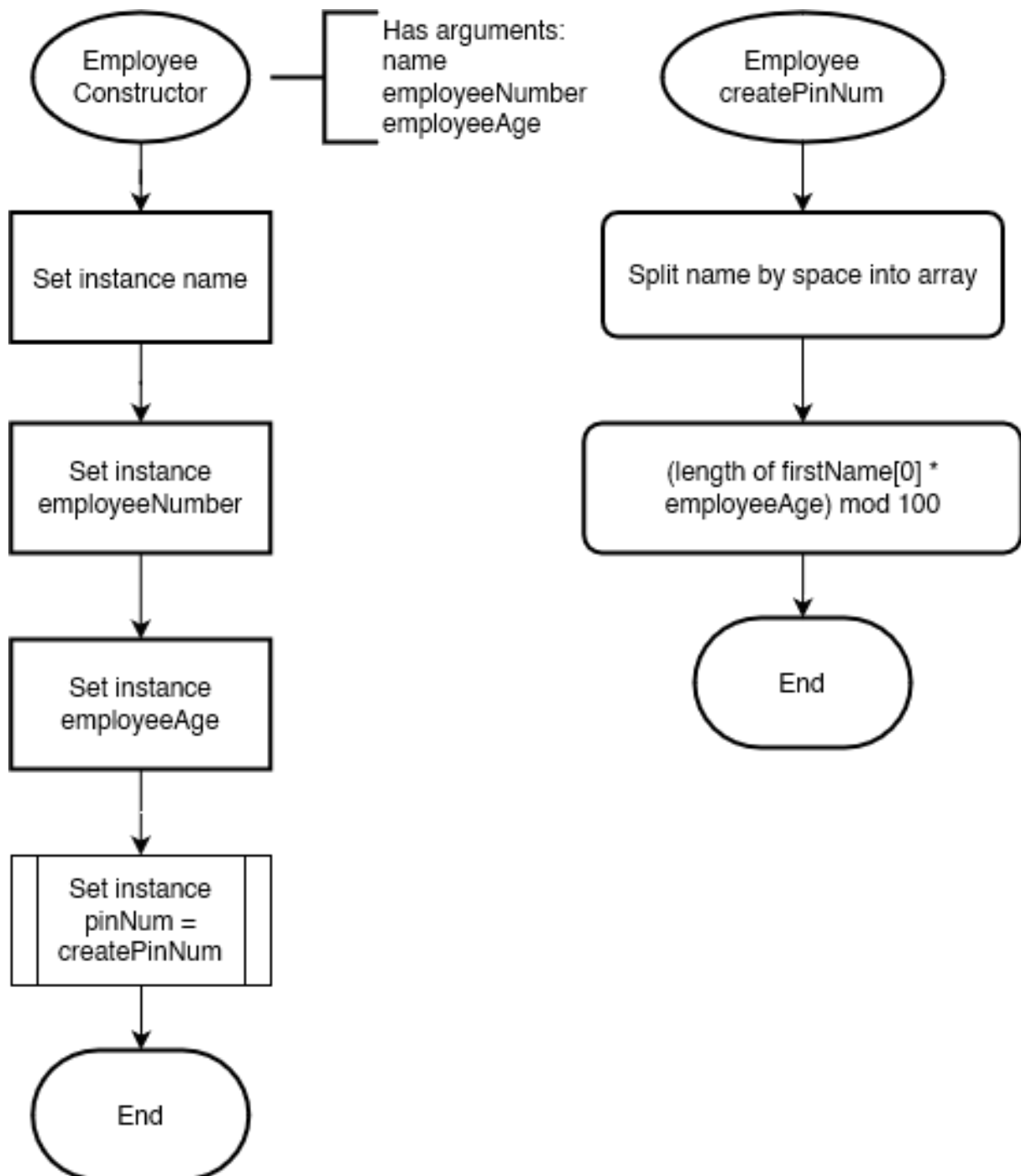
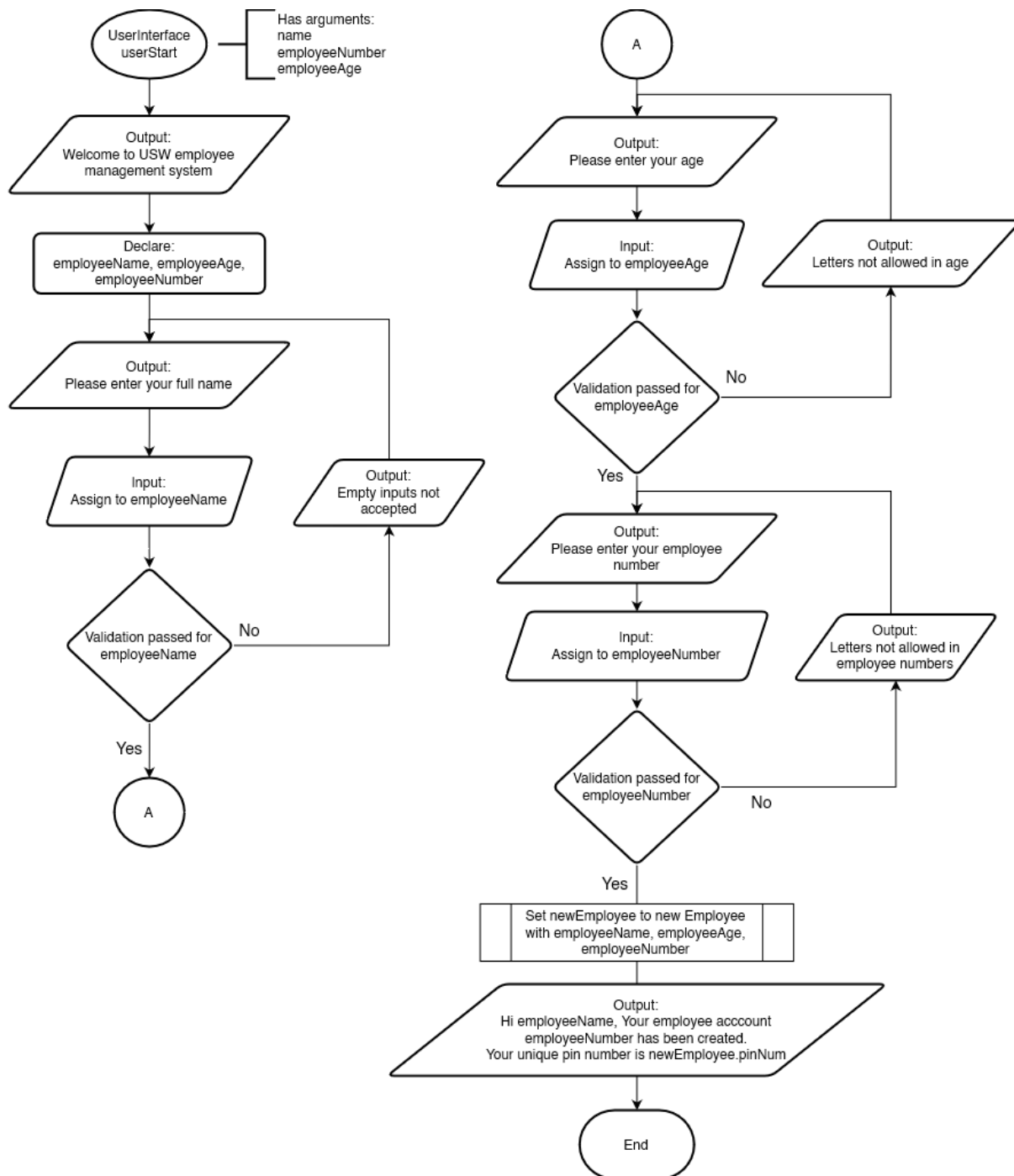


Figure 1: Flowchart of Main.java

**Figure 2:** Flowchart of Employee.java

**Figure 3:** Flowchart of UserInterface.java

Part 2 - Employee Pay Calculator

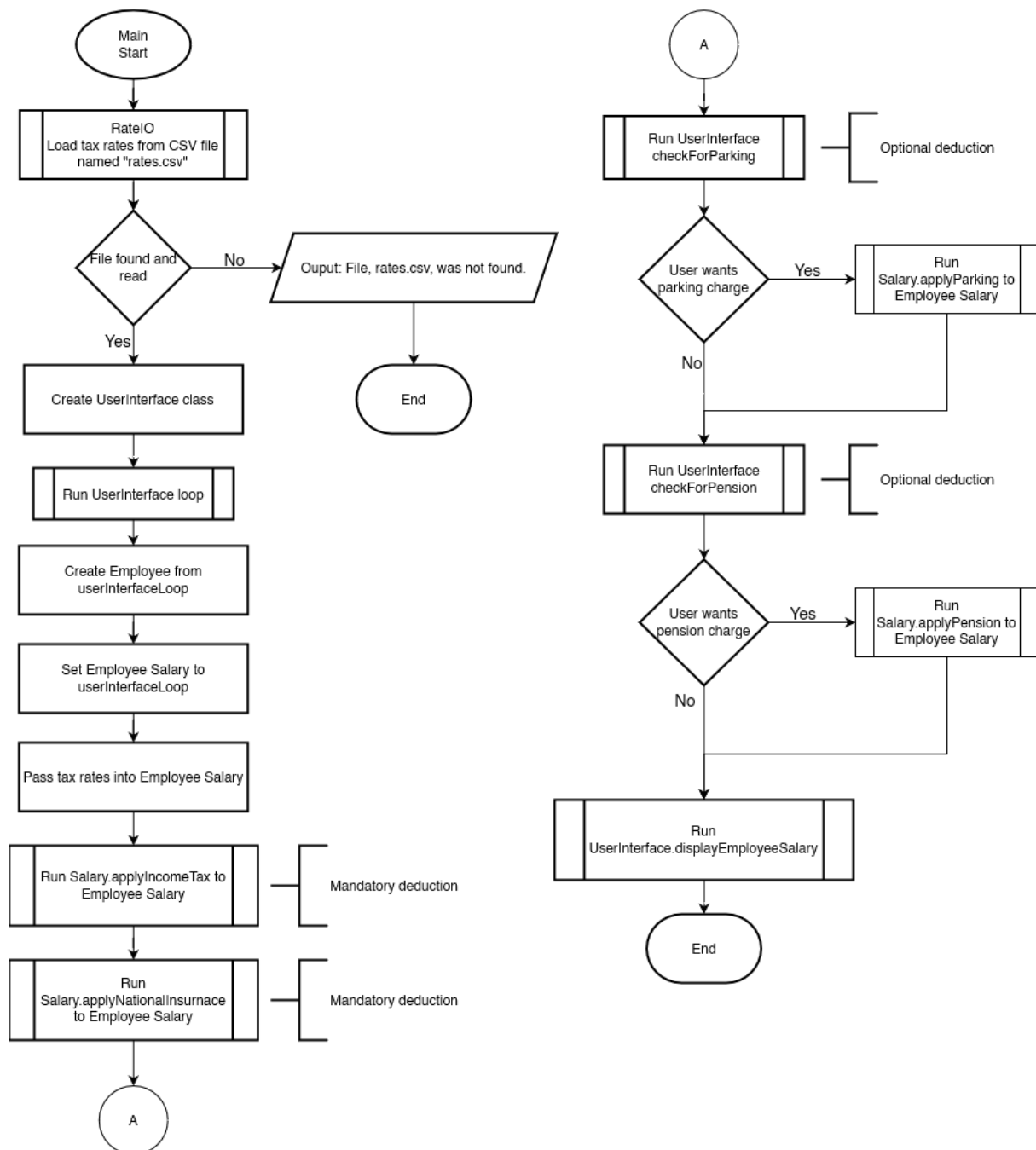
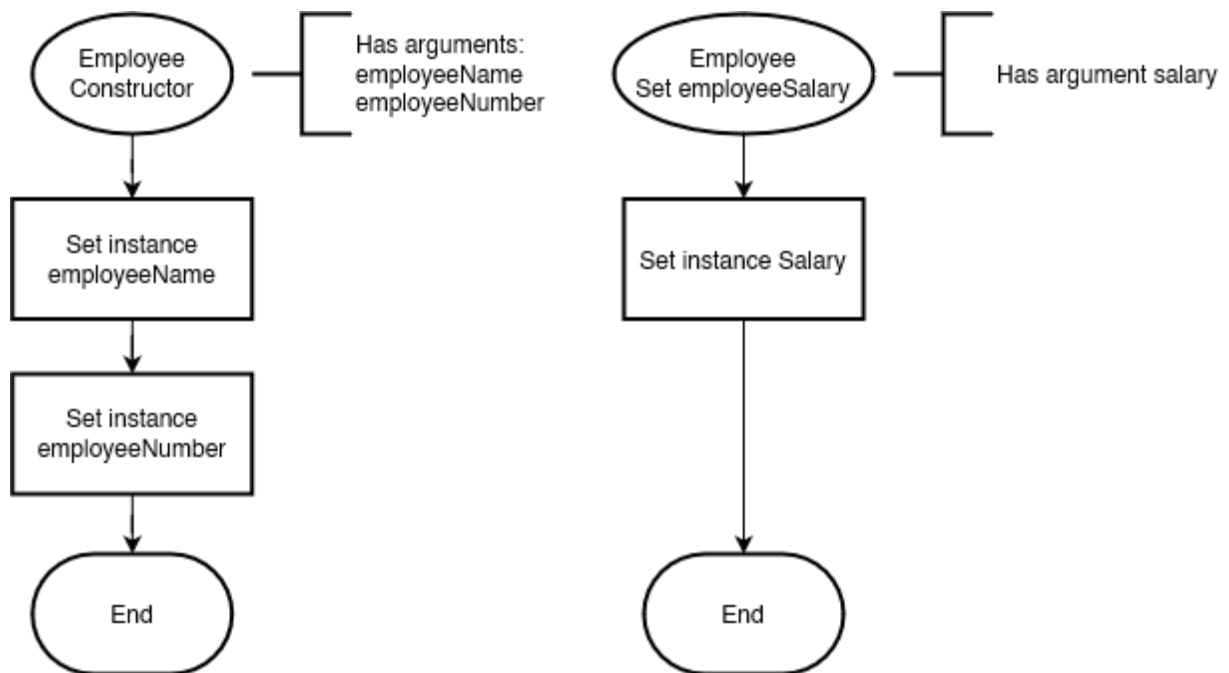


Figure 4: Flowchart of Main.java

**Figure 5:** Flowchart of `Employee.java`

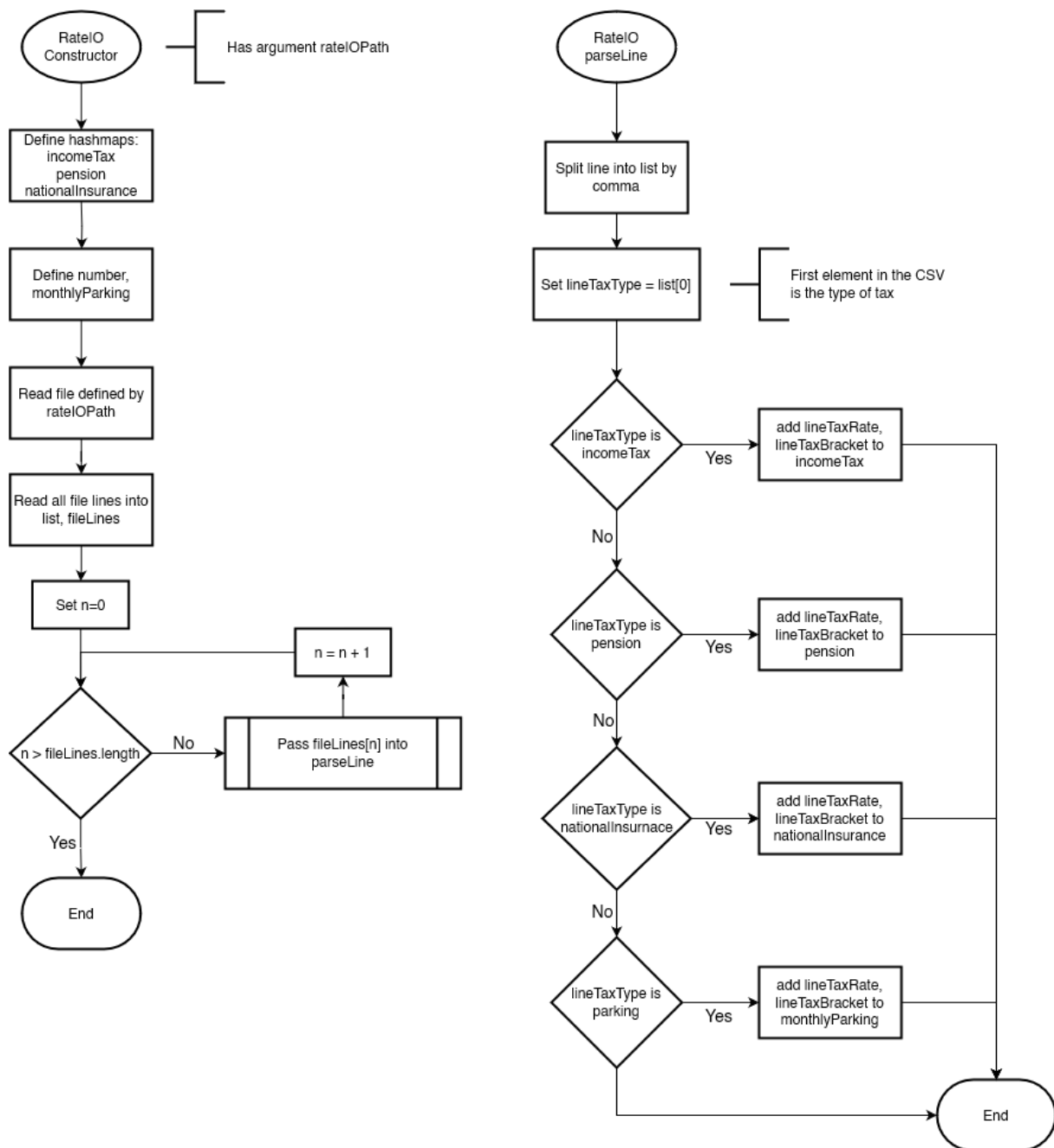
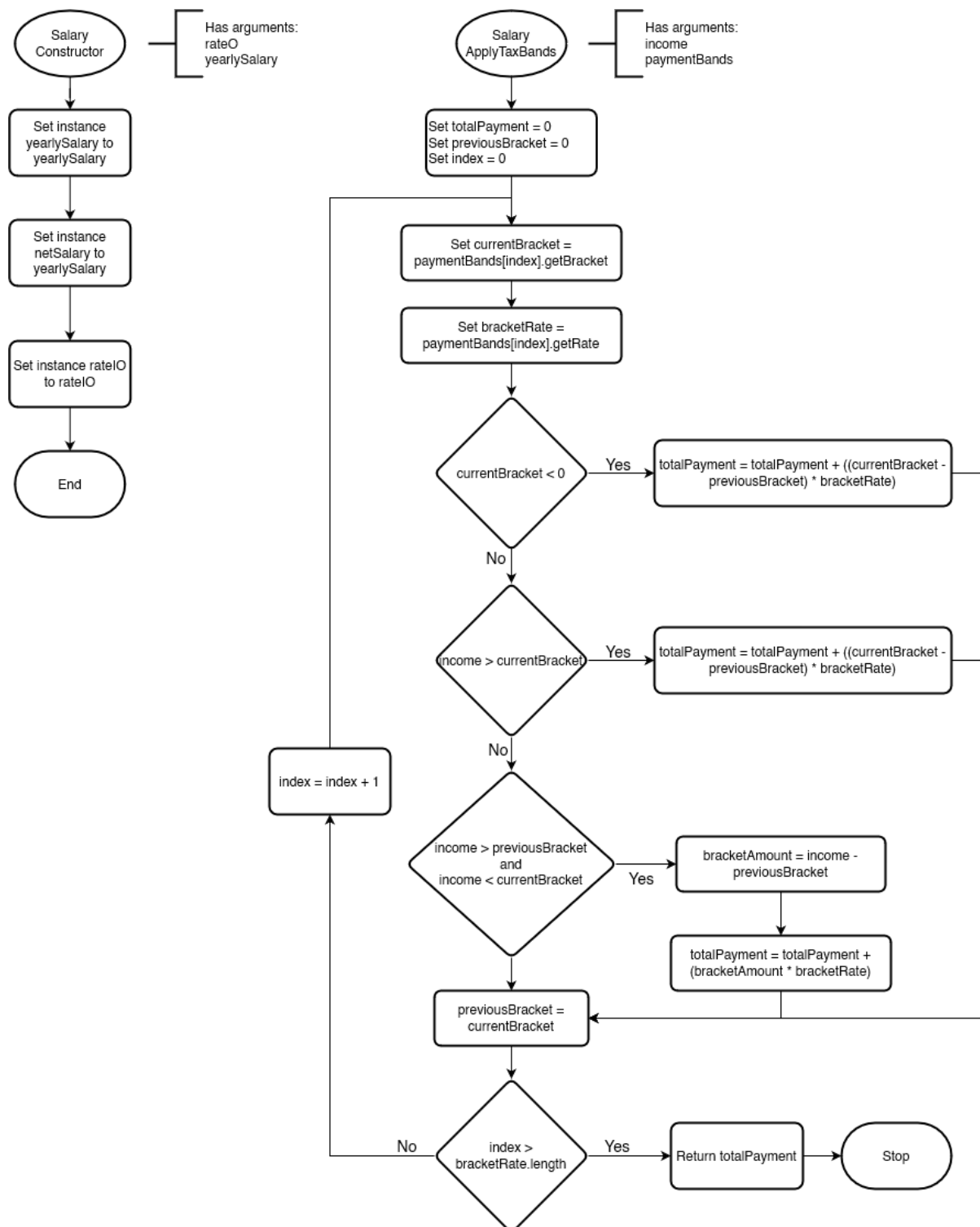
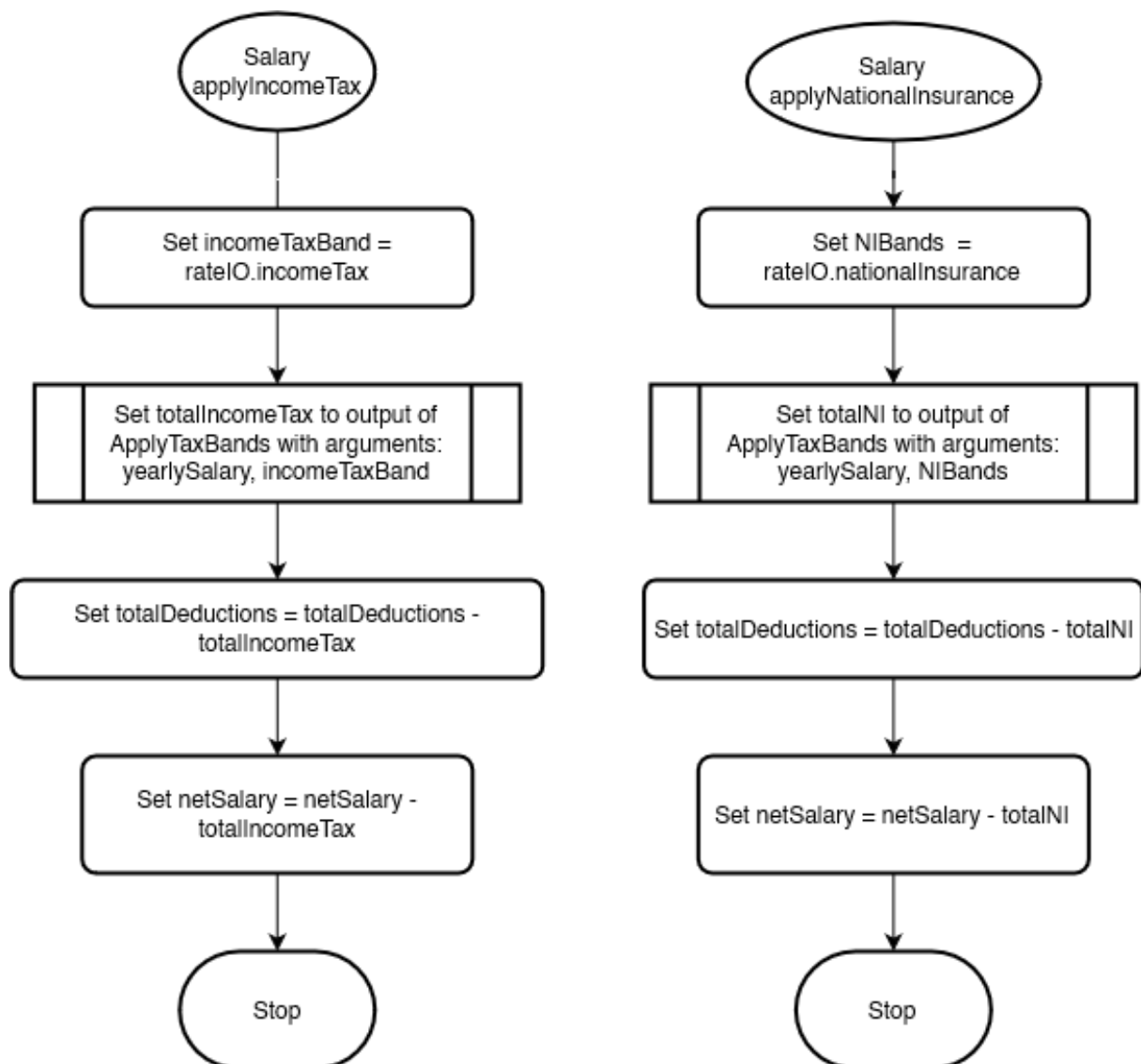
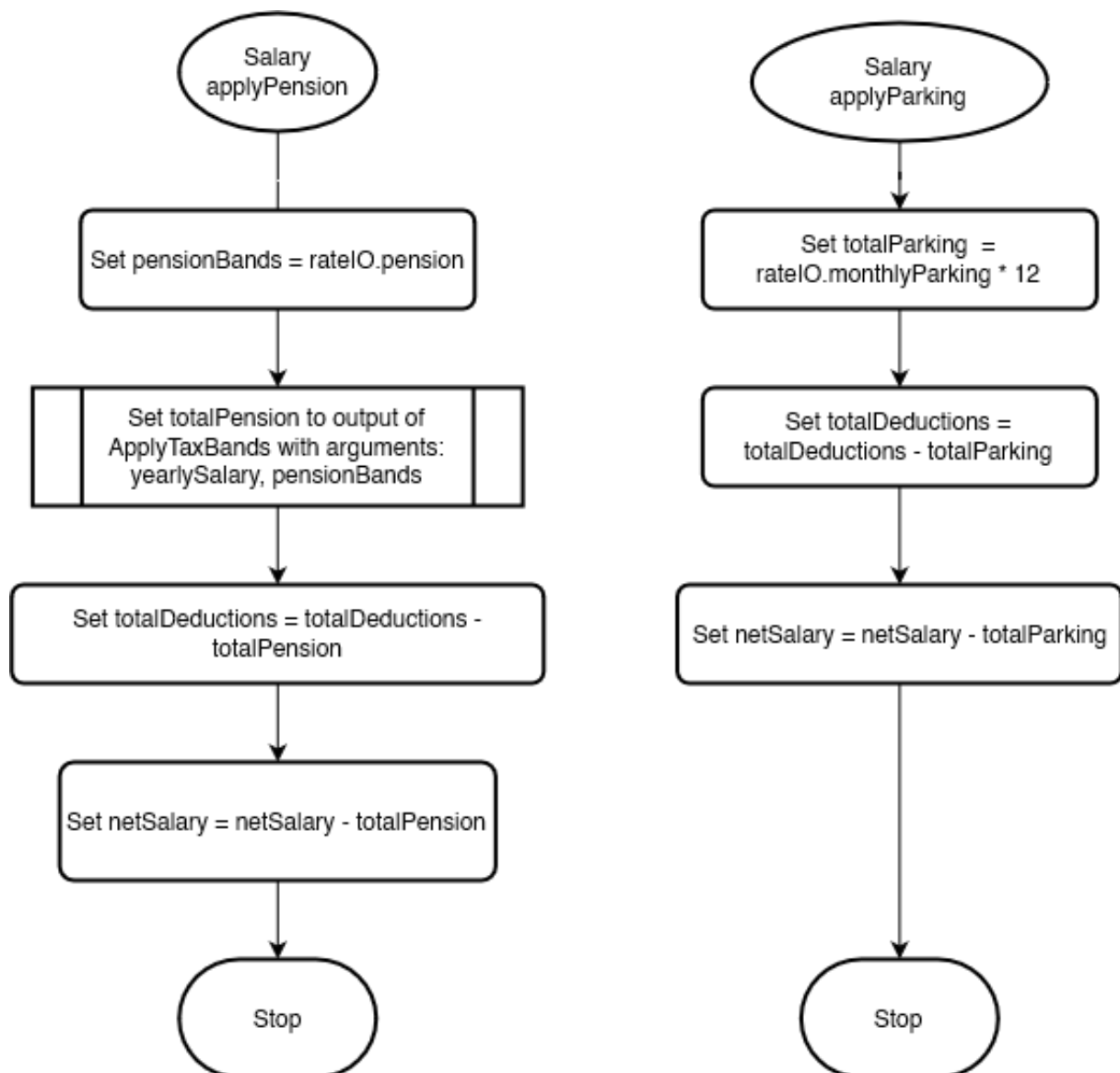


Figure 6: Flowchart of RatelO.java

**Figure 7:** Flowchart of Salary.java

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

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

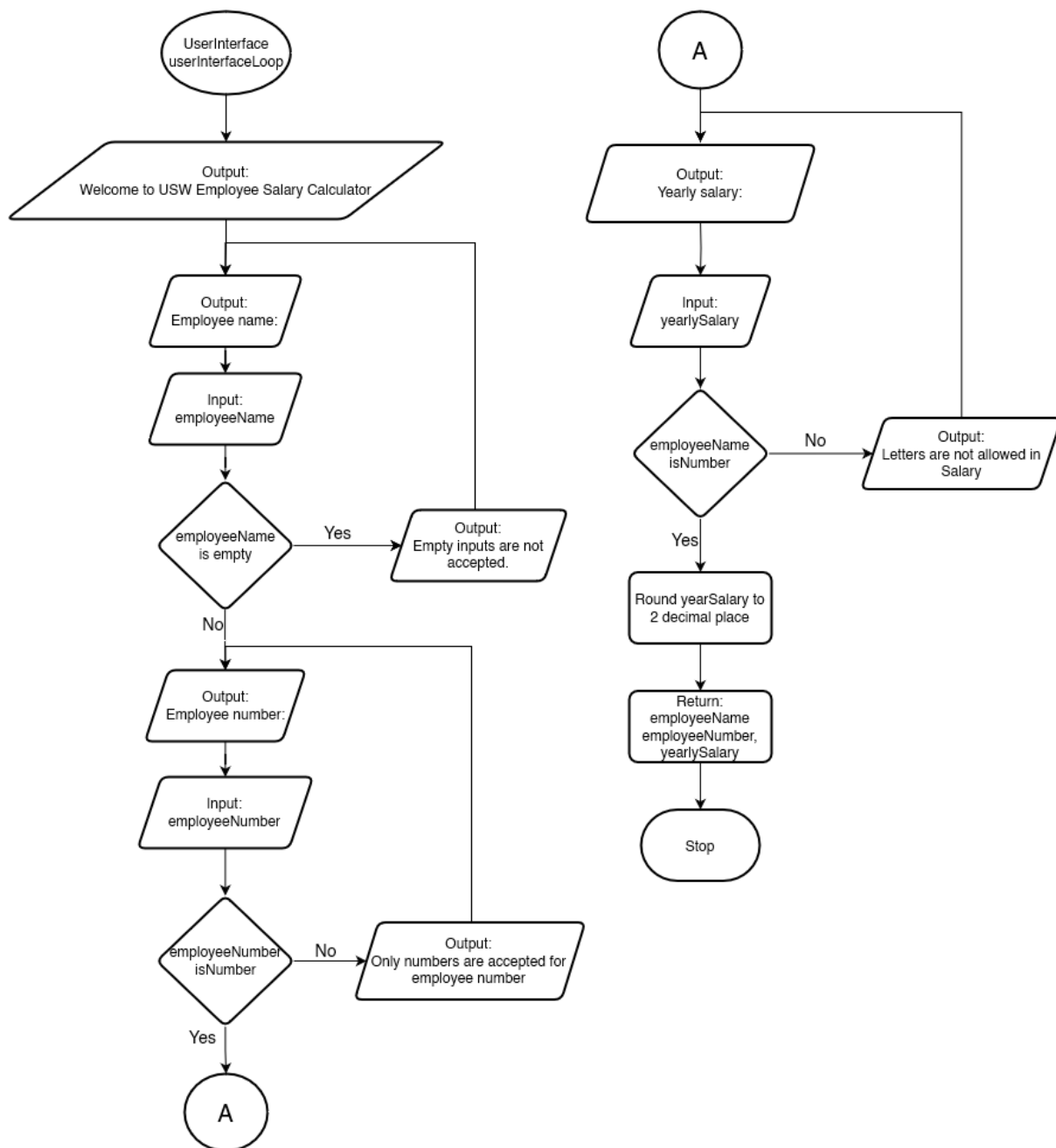
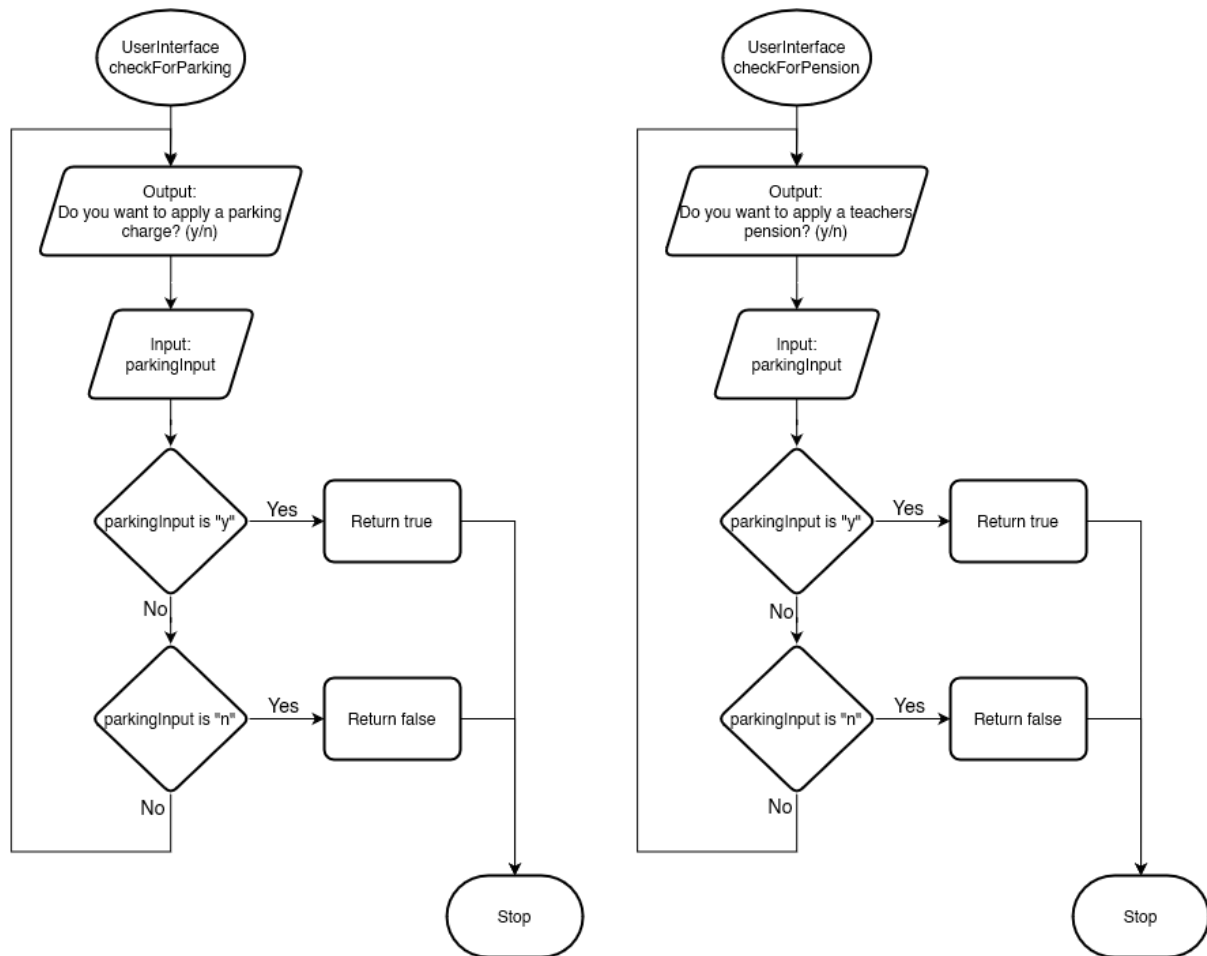
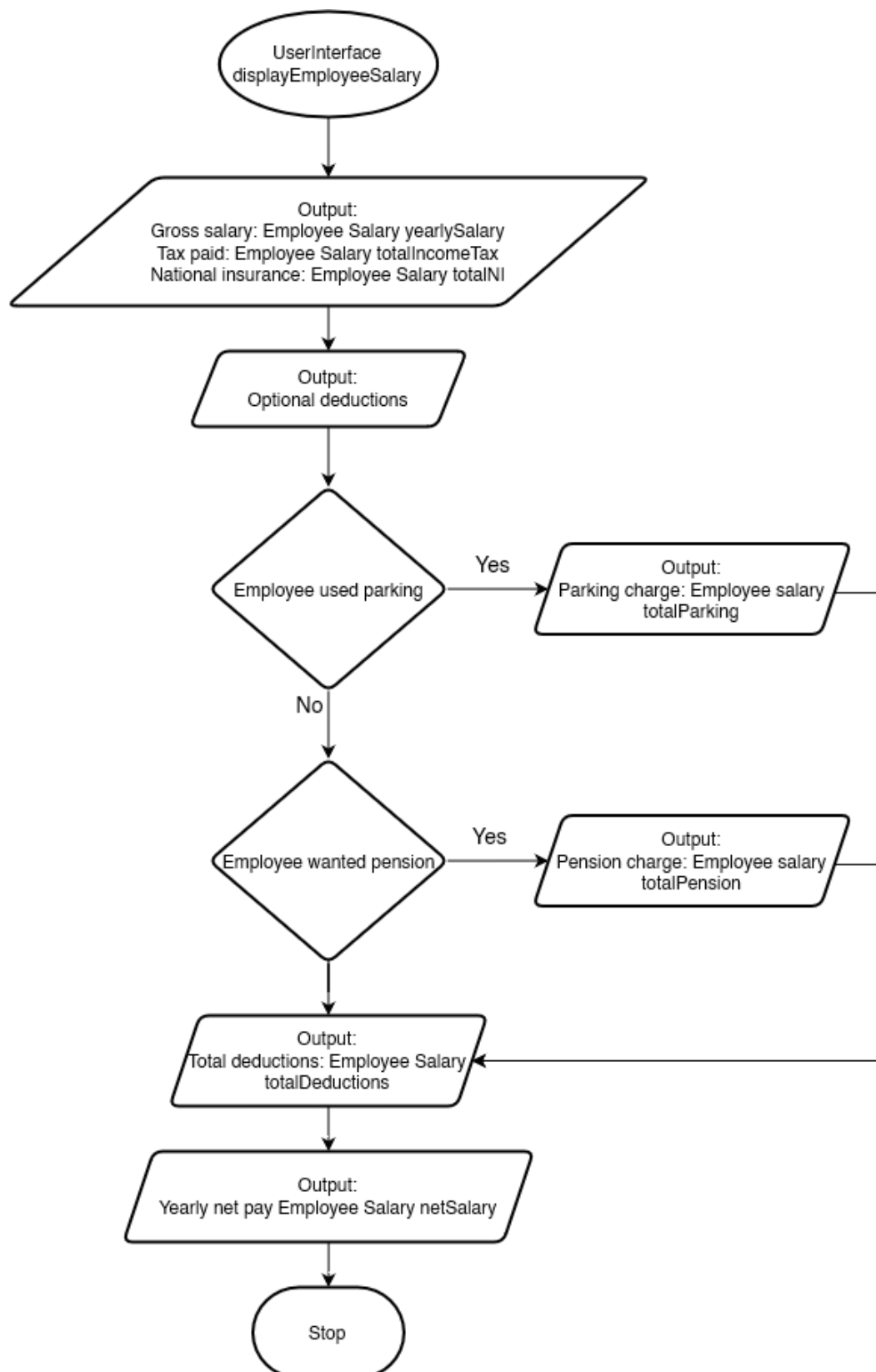


Figure 10: Flowchart of UserInterface.java

**Figure 11:** 2nd Flowchart of `UserInterface.java`

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

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, unaffected by changes to user input or files.(Products, 2021)

Part B - Programming Task**Part 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      * @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

Part 2 - Employee Pay Calculator

Program Source Code

Main.java

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

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     /**
14      * Class that handles outputting and accepting user input
15      *
16      * @param scanner Input handling
17      */
18     public UserInterface(Scanner scanner) {
19         this.scanner = scanner;
20     }
21
22
23     /**
24      * Outputs the information concerning an employee's salary
25      *
26      * @param employee Employee to display salary of
27      */
28     public static void displayEmployeeSalary(Employee employee) {
29
30         System.out.println("\nCalculating yearly net pay...\n");
31         System.out.printf("
32             Gross salary: £%s
33             Taxable amount: £%s
34             Tax paid: £%s
35             National insurance paid: £%s
36             ",
37             employee.getSalary().getGrossSalary(),
38             employee.getSalary().getTaxableAmount(),
39             employee.getSalary().getIncomeTaxAmount(),
40             employee.getSalary().getNIAmount()
41         );
42
43         /* Non-required deductions */
44         if (!(employee.getSalary().getTotalParking() == null)) {
45             System.out.printf("Parking charge: £%s\n",
46                 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().
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())
98         );
99     }
100
101     System.out.printf("\nMonthly total deductions: £%s\n",
102         Salary.convertMonthly(employee.getSalary().
103             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
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
168     BigDecimal yearSalary;
169
170     while (true) {
171         System.out.print("Yearly salary: ");
172         try {
173             String inputSalary = scanner.next();
174             yearSalary = new BigDecimal(inputSalary);
175             yearSalary = yearSalary.setScale(2, RoundingMode.
176                 HALF_UP);
177             /* Clear the newline character from scanner buffer
178              * Otherwise next question would appear twice, as the
179              * scanner would pick up the leftover newline
180              */
181             scanner.nextLine();
182
183             // Check if the number is negative
184             if (yearSalary.compareTo(BigDecimal.ZERO) < 0) {
185                 System.out.println(
186                     "Negative salaries are not accepted"
187                 );
188                 continue;
189             }
190             break;
191         } catch (NumberFormatException e) {
192             System.out.println(
193                 "Letter are not allowed in the employee number"
194             );
195         }
196     }
197     return new Salary(yearSalary, rateIO);
198 }
199
200 /**
201  * Asks user if they want to apply a parking charge
202  */
```

```
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.employee;
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      public static BigDecimal convertMonthly(BigDecimal amount) {
36          return amount.divide(new BigDecimal("12"), 2, RoundingMode.
37              HALF_UP);
38      }
39      /**
40       * Applies required deductions: income tax, national insurance
41       */
42      public void applyMandatoryDeductions() {
43          applyIncomeTax();
44          applyNationalInsurance();
45      }
46
47      public void applyIncomeTax() {
48          totalIncomeTax = applyPaymentBands(grossSalary,
49              rateIO.getTaxBands()
50          );
51          totalDeductions = totalDeductions.add(totalIncomeTax);
52          netSalary = netSalary.subtract(totalIncomeTax);
53      }
54
55      public void applyNationalInsurance() {
56          totalNI = applyPaymentBands(grossSalary,
57              rateIO.getNationalInsurance()
58          );
59          totalDeductions = totalDeductions.add(totalNI);
60          netSalary = netSalary.subtract(totalNI);
```

```
61     }
62
63     public void applyPension() {
64         totalPension = applyPaymentBands(grossSalary,
65             rateIO.getPensionBands()
66         );
67         totalDeductions = totalDeductions.add(totalPension);
68         netSalary = netSalary.subtract(totalPension);
69     }
70
71     public void applyParkingCharge() {
72         // Monthly parking * 12
73         totalParking = rateIO.getMonthlyParking().multiply(
74             new BigDecimal("12")
75         );
76         totalDeductions = totalDeductions.add(totalParking);
77         netSalary = netSalary.subtract(totalParking);
78     }
79
80     /**
81      * Applies payment bands to income dynamically
82      *
83      * @param income      Accepts BigDecimals, no negatives
84      * @param paymentBands LinkedHashMap containing, the taxBand first,
85      *                      then the taxRate, overflow tax rates
86      *                      should be denoted with a negative
87      *                      on the band
88      * @return Total payment on income after paymentBands applied
89      */
90     private BigDecimal applyPaymentBands(BigDecimal income,
91         LinkedHashMap<BigDecimal, BigDecimal> paymentBands) {
92
93         BigDecimal totalPayment = new BigDecimal("0");
94         BigDecimal previousBracket = new BigDecimal("0");
95
96         for (Map.Entry<BigDecimal, BigDecimal> entry : paymentBands.
97             entrySet()) {
98             BigDecimal currentBracket = entry.getKey();
99             BigDecimal bracketRate = entry.getValue();
100
101             /*
102              * If the payment is in a band denoted with a negative
103              * number then it is overflow, and applies
104              * that rate to rest of salary
105              */
106             if (currentBracket.compareTo(BigDecimal.ZERO) < 0) {
107                 /* totalPayment = totalPayment +
108                  * (income - previousBand) * taxRate
109                  */
110                 totalPayment = totalPayment.add(
111                     income.subtract(previousBracket).multiply(
```

```

        bracketRate).setScale(2,RoundingMode.HALF_UP)
110    );
111    } else if (income.compareTo(entry.getKey()) > 0) {
112        /* If the income is greater than the current
113        * payment band
114        */
115
116        /* totalPayment = totalPayment +
117        * (currentBracket - previousBand) * taxRate
118        * It then rounds to 2 decimal places
119        */
120        totalPayment = totalPayment.add((
121            entry.getKey().subtract(previousBracket)).multiply(
122            entry.getValue()).setScale(2, RoundingMode.
                HALF_UP)
123    );
124
125    } else if ((income.compareTo(previousBracket) > 0) && (
        income.compareTo(entry.getKey()) < 0)) {
126        /* If the income is smaller than the current payment
127        * band
128        */
129
130        /* Get the leftover money in the band */
131        BigDecimal bracketAmount = income.subtract(
            previousBracket);
132        /* apply tax to the leftover amount in the band
133        * totalPayment = totalPayment +
134        * (leftoverAmount * taxRate)
135        */
136        totalPayment = totalPayment.add(
137            bracketAmount.multiply(entry.getValue()).setScale
                (2, RoundingMode.HALF_UP)
138    );
139        /* Since income is smaller than current band, won't
140        * make it to next band, break out of loop
141        */
142        break;
143    }
144    previousBracket = entry.getKey();
145    }
146    return totalPayment;
147    }
148
149    public void setSalary(BigDecimal grossSalary) {
150        this.grossSalary = grossSalary;
151        netSalary = grossSalary;
152        applyMandatoryDeductions();
153    }
154
155    public void setRateIO(iRateIO rateIO) {
```

```
156         this.rateIO = rateIO;
157         applyMandatoryDeductions();
158     }
159
160     public BigDecimal getGrossSalary() {
161         return grossSalary;
162     }
163
164     public BigDecimal getMonthlySalary() {
165         return convertMonthly(grossSalary);
166     }
167
168     public BigDecimal getTaxableAmount() {
169         return grossSalary.subtract(new BigDecimal("12570"));
170     }
171
172     public BigDecimal getIncomeTaxAmount() {
173         return totalIncomeTax;
174     }
175
176     public BigDecimal getNIAmount() {
177         return totalNI;
178     }
179
180     public BigDecimal getPensionAmount() {
181         return totalPension;
182     }
183
184     public BigDecimal getTotalParking() {
185         return totalParking;
186     }
187
188     public BigDecimal getTotalDeductions() {
189         return totalDeductions;
190     }
191
192     public BigDecimal getNetSalary() {
193         return netSalary;
194     }
195
196     public BigDecimal getMonthlyNetSalary() {
197         return netSalary.divide(
198             new BigDecimal("12"), 2, RoundingMode.HALF_UP
199         );
200     }
201 }
```

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 = new
13         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     }
```

```
28     }
29
30     /**
31     * Handle the separated line and add it to a tax band
32     * @param line Line to parse
33     */
34     private void parseLine(List<String> line) {
35         /* Each type of deduction possible in CSV */
36         switch (line.get(0)) {
37             case "tax" -> taxBands.put(
38                 new BigDecimal(line.get(1)),
39                 new BigDecimal(line.get(2))
40             );
41             case "pension" -> pensionBands.put(
42                 new BigDecimal(line.get(1)),
43                 new BigDecimal(line.get(2))
44             );
45             case "nationalInsurance" -> nationalInsurance.put(
46                 new BigDecimal(line.get(1)),
47                 new BigDecimal(line.get(2))
48             );
49             case "parking" -> monthlyParking = (
50                 new BigDecimal(line.get(1))
51             );
52         }
53     }
54
55     public LinkedHashMap<BigDecimal, BigDecimal> getTaxBands() {
56         return taxBands;
57     }
58
59     public LinkedHashMap<BigDecimal, BigDecimal> getNationalInsurance()
60     {
61         return nationalInsurance;
62     }
63
64     public LinkedHashMap<BigDecimal, BigDecimal> getPensionBands() {
65         return pensionBands;
66     }
67
68     public BigDecimal getMonthlyParking() {
69         return monthlyParking;
70     }
```

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     TestingFakeRateIO testingRateIO = new TestingFakeRateIO();
13     Salary testSalary = new Salary(
14         new BigDecimal("45000"), testingRateIO
15     );
16     Salary testSalaryDecimal = new Salary(
17         new BigDecimal("50000"), testingRateIO
18     );
19     Salary testSalaryLarge = new Salary(
20         new BigDecimal("140000"), testingRateIO
21     );
22
23     @Test
24     @DisplayName("Calculate monthly salary")
25     public void monthlySalaryCalculations() {
26         BigDecimal expectedMonthlySalary2 = new BigDecimal("3750");
27
28         assertEquals(0, expectedMonthlySalary2.compareTo(
29             testSalary.getMonthlySalary()
30         ));
31
32         BigDecimal expectedMonthlySalary1 = new BigDecimal("4166.67");
33
34         assertEquals(0, expectedMonthlySalary1.compareTo(
35             testSalaryDecimal.getMonthlySalary()
36         ));
37     }
38
39     @Test
40     @DisplayName("Calculate taxable amount")
41     public void getTaxableAmount() {
42         BigDecimal expectedTaxableAmount = new BigDecimal("32430.00");
43
44         assertEquals(0, expectedTaxableAmount.compareTo(
45             testSalaryLarge.getTaxableAmount()
46         ));
47     }
48
49     @Test
50     @DisplayName("Calculate income tax")
```



```
52     public void calculateIncomeTax() {
53         BigDecimal expectedTax = new BigDecimal("6486");
54         testSalary.applyIncomeTax();
55
56         assertEquals(0, expectedTax.compareTo(
57             testSalary.getIncomeTaxAmount())
58         );
59
60         BigDecimal expectedTaxLarge = new BigDecimal("44175");
61         testSalaryLarge.applyIncomeTax();
62
63         assertEquals(0, expectedTaxLarge.compareTo(
64             testSalaryLarge.getIncomeTaxAmount())
65         );
66     }
67
68     @Test
69     @DisplayName("Calculate national insurance")
70     void calculateNationalInsurance() {
71         BigDecimal expectedNI = new BigDecimal("4251.84");
72         testSalary.applyNationalInsurance();
73
74         assertEquals(0, expectedNI.compareTo(
75             testSalary.getNIAmount())
76         );
77     }
78
79     @Test
80     @DisplayName("Parking charge applies")
81     void useParkingCharge() {
82         BigDecimal expectedNetSalary = new BigDecimal("34142.16");
83         BigDecimal monthlyParking = new BigDecimal("120.00");
84         testSalary.applyMandatoryDeductions();
85         testSalary.applyParkingCharge();
86
87         assertEquals(0, monthlyParking.compareTo(
88             testSalary.getTotalParking())
89         );
90         assertEquals(0, expectedNetSalary.compareTo(
91             testSalary.getNetSalary())
92         );
93     }
94
95     @Test
96     @DisplayName("Total teachers pension")
97     void getTotalTeachersPension() {
98         BigDecimal expectedTeachersPension = new BigDecimal("3501.76");
99         testSalary.applyPension();
100
101         assertEquals(0, expectedTeachersPension.compareTo(
102             testSalary.getPensionAmount())
```

```
103     );
104 }
105
106 @Test
107 @DisplayName("Total deductions")
108 void getTotalDeductions() {
109     BigDecimal expectedDeductions = new BigDecimal("10737.84");
110     testSalary.applyMandatoryDeductions();
111
112     assertEquals(0, expectedDeductions.compareTo(
113         testSalary.getTotalDeductions())
114     );
115 }
116
117 @Test
118 @DisplayName("Net salary")
119 void getNetSalary() {
120     BigDecimal expectedNetSalary = new BigDecimal("34142.16");
121     testSalary.applyMandatoryDeductions();
122     testSalary.applyParkingCharge();
123
124     assertEquals(0, expectedNetSalary.compareTo(
125         testSalary.getNetSalary())
126     );
127 }
128 }
```

RateIOTest.java

```
1
2 package usw.employeeepay;
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     LinkedHashMap<BigDecimal, BigDecimal> expectedTaxBands = new
31         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     expectedTaxBands.put(
42         new BigDecimal("-1"), new BigDecimal("0.45")
43     );
44     assertEquals(expectedTaxBands, rateIO.getTaxBands());
45 }
46
47 @Test
48 @DisplayName("NI tax bands")
49 void getNationalInsurance() {
50     LinkedHashMap<BigDecimal, BigDecimal> expectedNationalInsurance
51         = new LinkedHashMap<>();
52     expectedNationalInsurance.put(
53         new BigDecimal("9568"), new BigDecimal("0.00")
54     );
55     expectedNationalInsurance.put(
56         new BigDecimal("-1"), new BigDecimal("0.12")
57     );
58     assertEquals(expectedNationalInsurance, rateIO.
59         getNationalInsurance());
60 }
61
62 @Test
63 @DisplayName("Pension tax bands")
64 void getPensionBands() {
65     LinkedHashMap<BigDecimal, BigDecimal> expectedPensionBands =
66         new LinkedHashMap<>();
67
68     expectedPensionBands.put(
69         new BigDecimal("32135.99"), new BigDecimal("0.074")
70     );
71     expectedPensionBands.put(
72         new BigDecimal("43259.99"), new BigDecimal("0.086")
73     );
74 }
```

```
69         );
70         expectedPensionBands.put(
71             new BigDecimal("51292.99"), new BigDecimal("0.096")
72         );
73         expectedPensionBands.put(
74             new BigDecimal("67980.99"), new BigDecimal("0.102")
75         );
76         expectedPensionBands.put(
77             new BigDecimal("92597.99"), new BigDecimal("0.113")
78         );
79         expectedPensionBands.put(
80             new BigDecimal("-1"), new BigDecimal("0.117")
81         );
82         assertEquals(expectedPensionBands, rateIO.getPensionBands());
83     }
84
85     @Test
86     @DisplayName("CSV parking fee")
87     void getMonthlyParking() {
88         BigDecimal expectedMonthlyParking = new BigDecimal("10.00");
89         assertEquals(0, expectedMonthlyParking.compareTo(
90             rateIO.getMonthlyParking()
91         ));
92     }
93 }
```

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`:

```
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

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

Products, M.I. (2021) *MAX6675 cold-junction-compensated k-thermocouple-to-digital converter (0°C to +1024°C)*. Available at: <https://datasheets.maximintegrated.com/en/ds/MAX6675.pdf>.

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