IS1S481 Coursework 1

Jake Real

17/11/2023

Contents

Part A - Design Task

Part 1 User Login and Unique Pin

Part 2 - Employee Pay Calculator

Part B - Programming Task

Part 1 User Login and Unique Pin

Program Source Code

`Main.java`

Program Unit Tests

Program Outputs

Part 2 - Employee Pay Calculator

Design Process

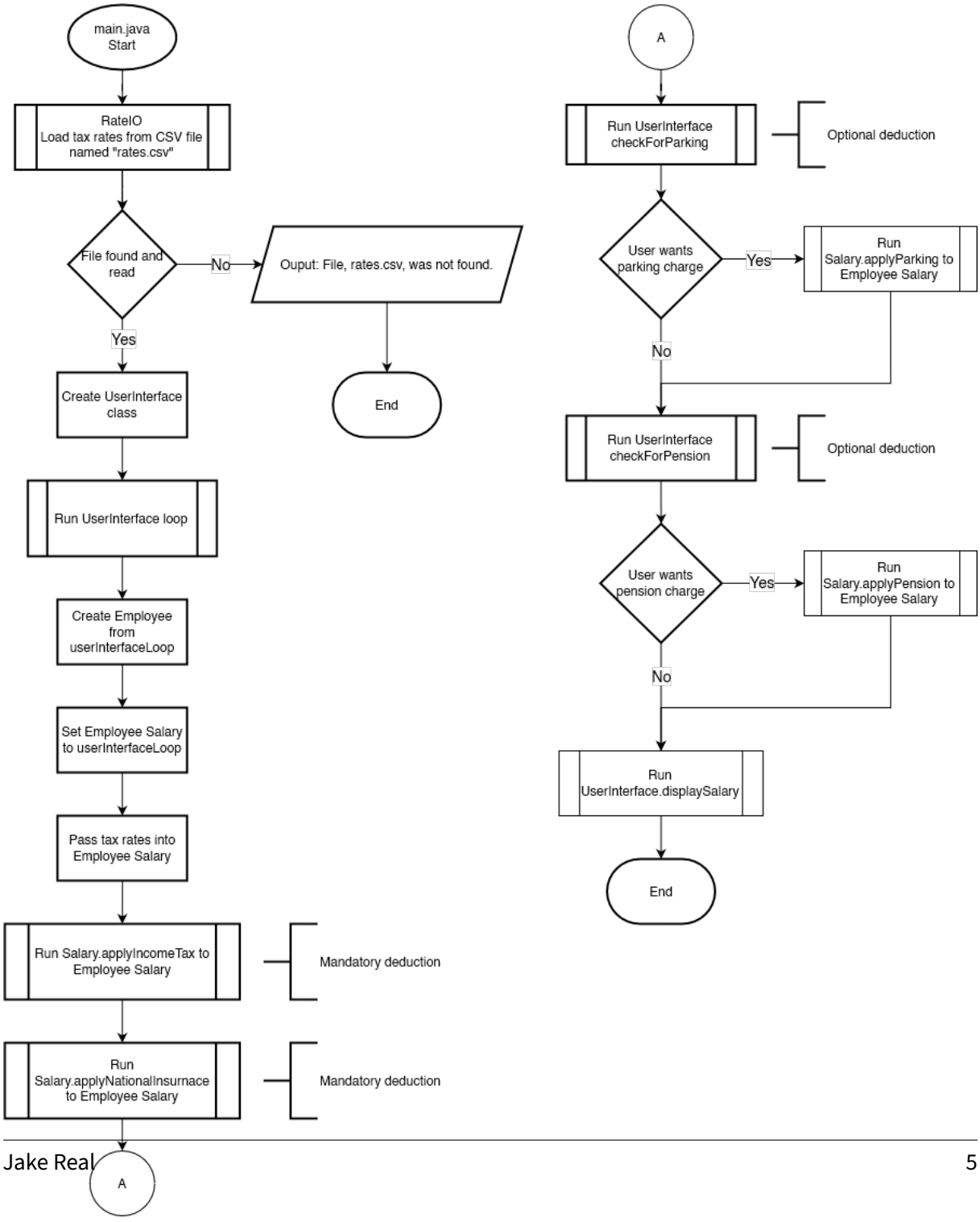

Figure 1: Flowchart of Main.java

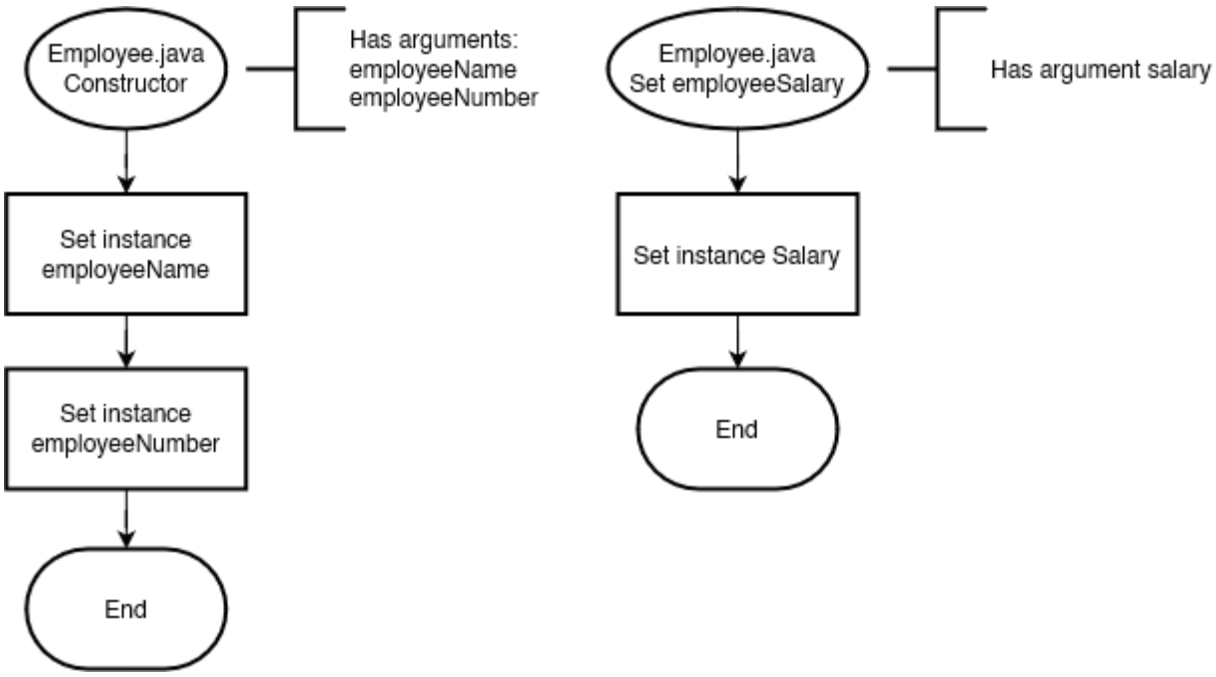

Figure 2: Flowchart of Employee.java

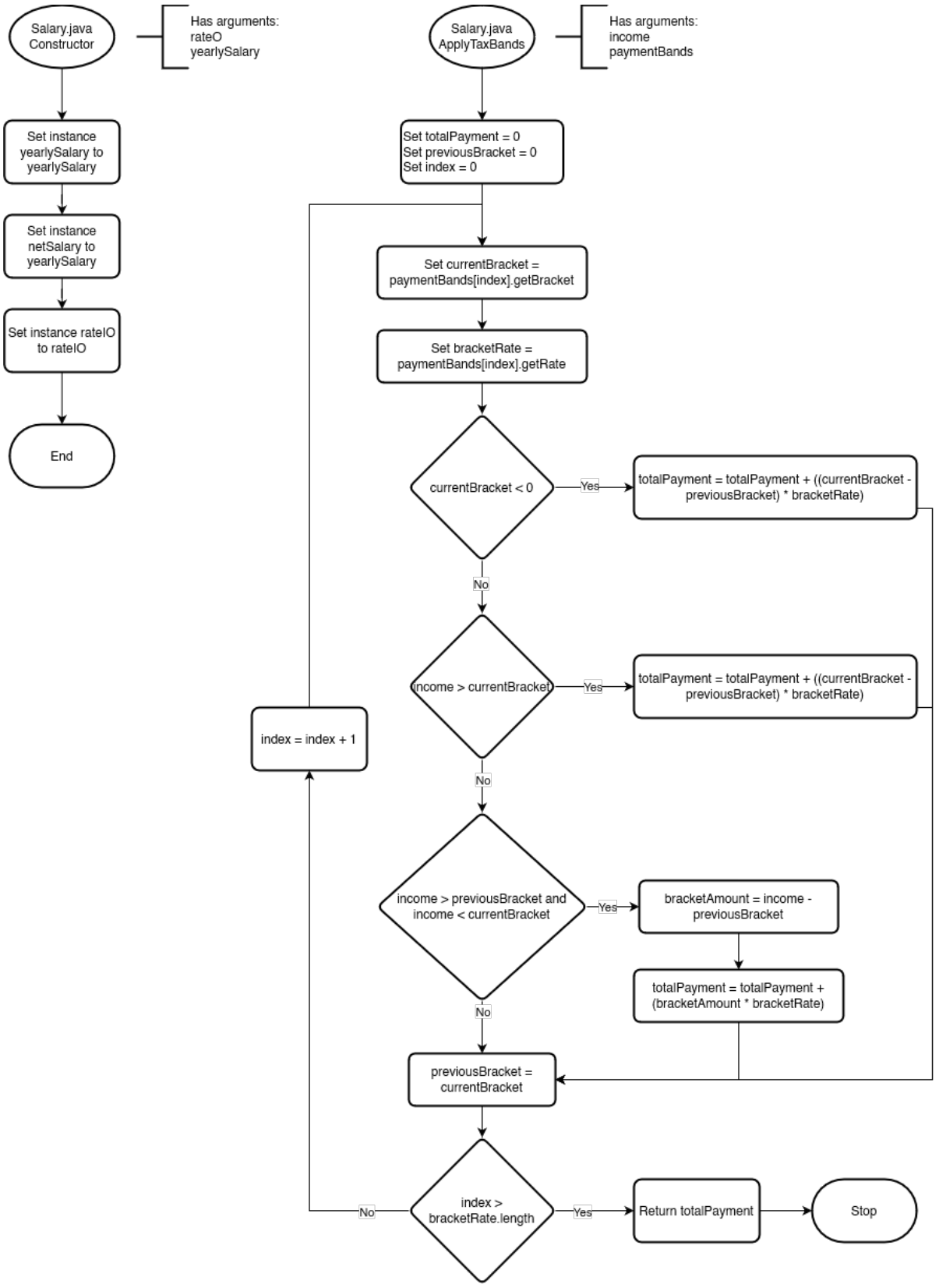

Figure 3: Flowchart of Salary.java

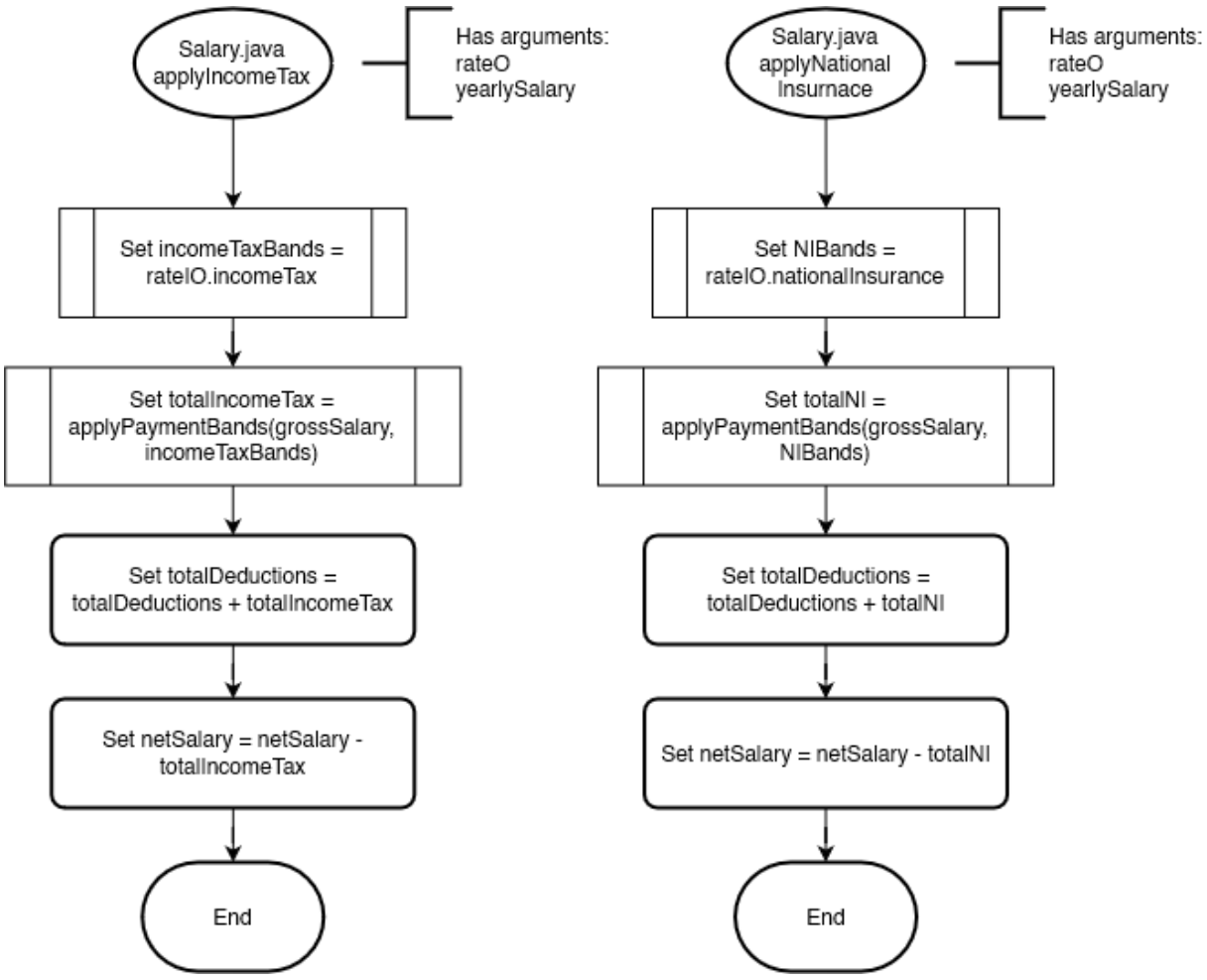

Figure 4: 2nd Flowchart of Salary.java

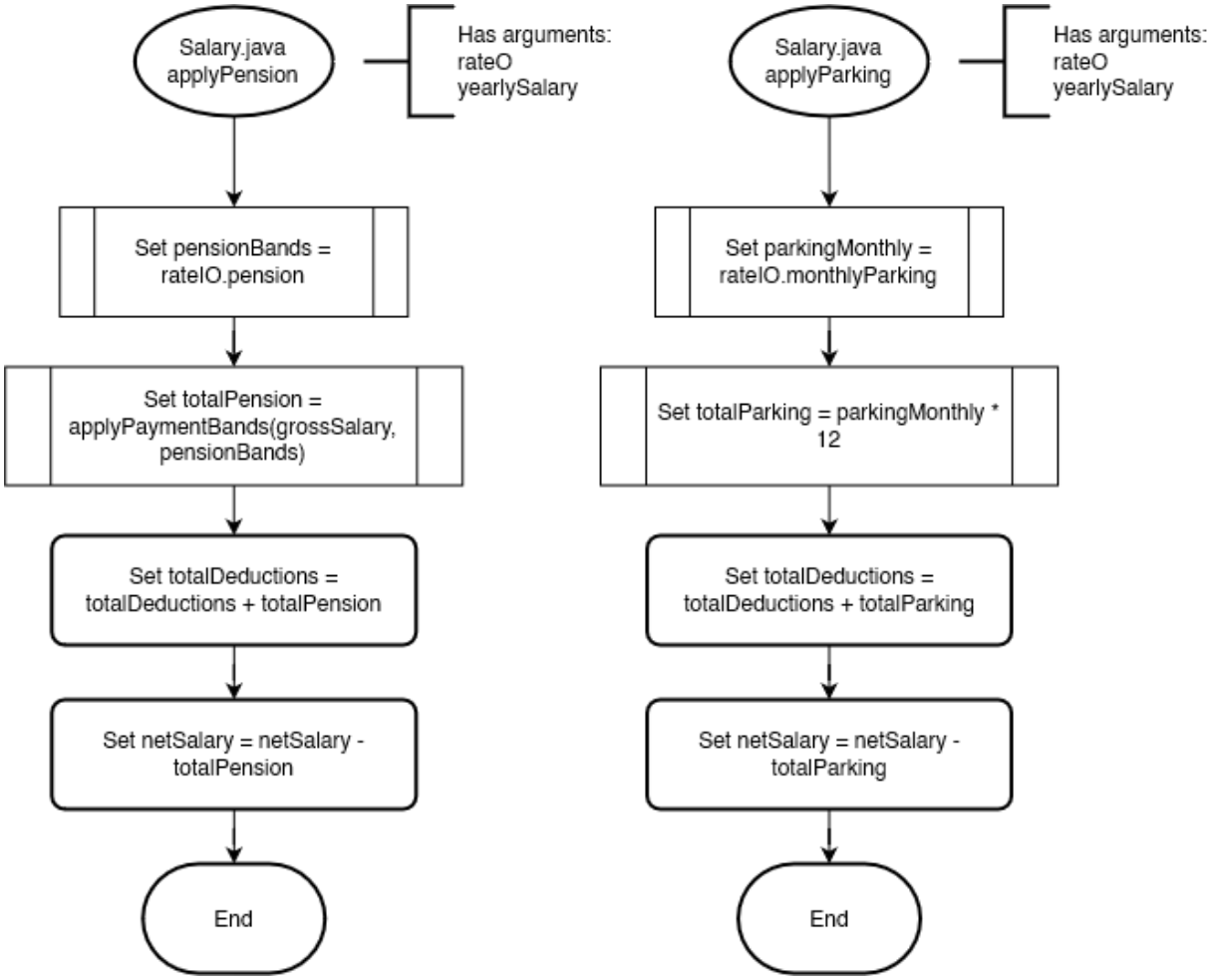

Figure 5: 3rd Flowchart of Salary.java

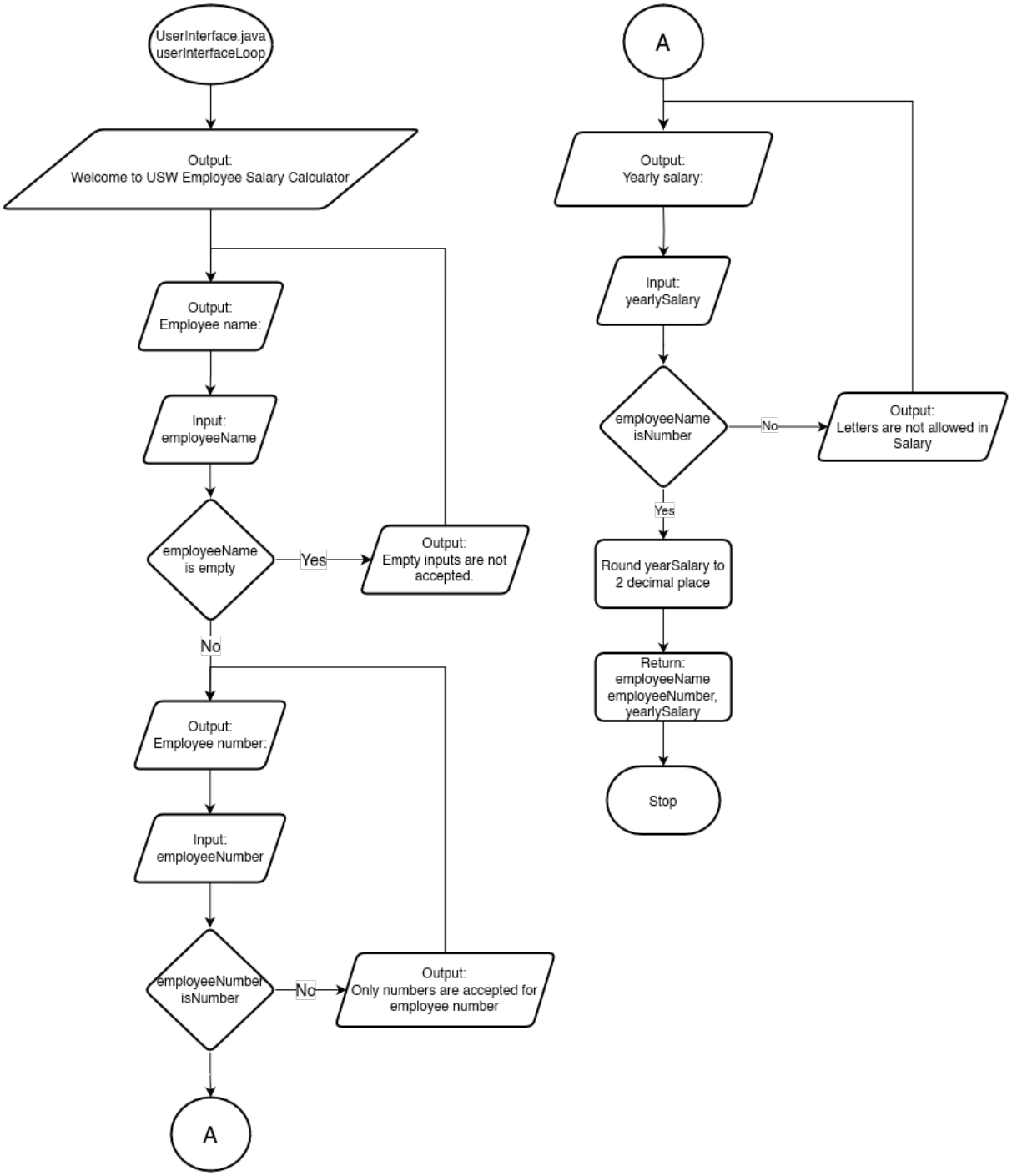

Figure 6: Flowchart of UserInterface.java

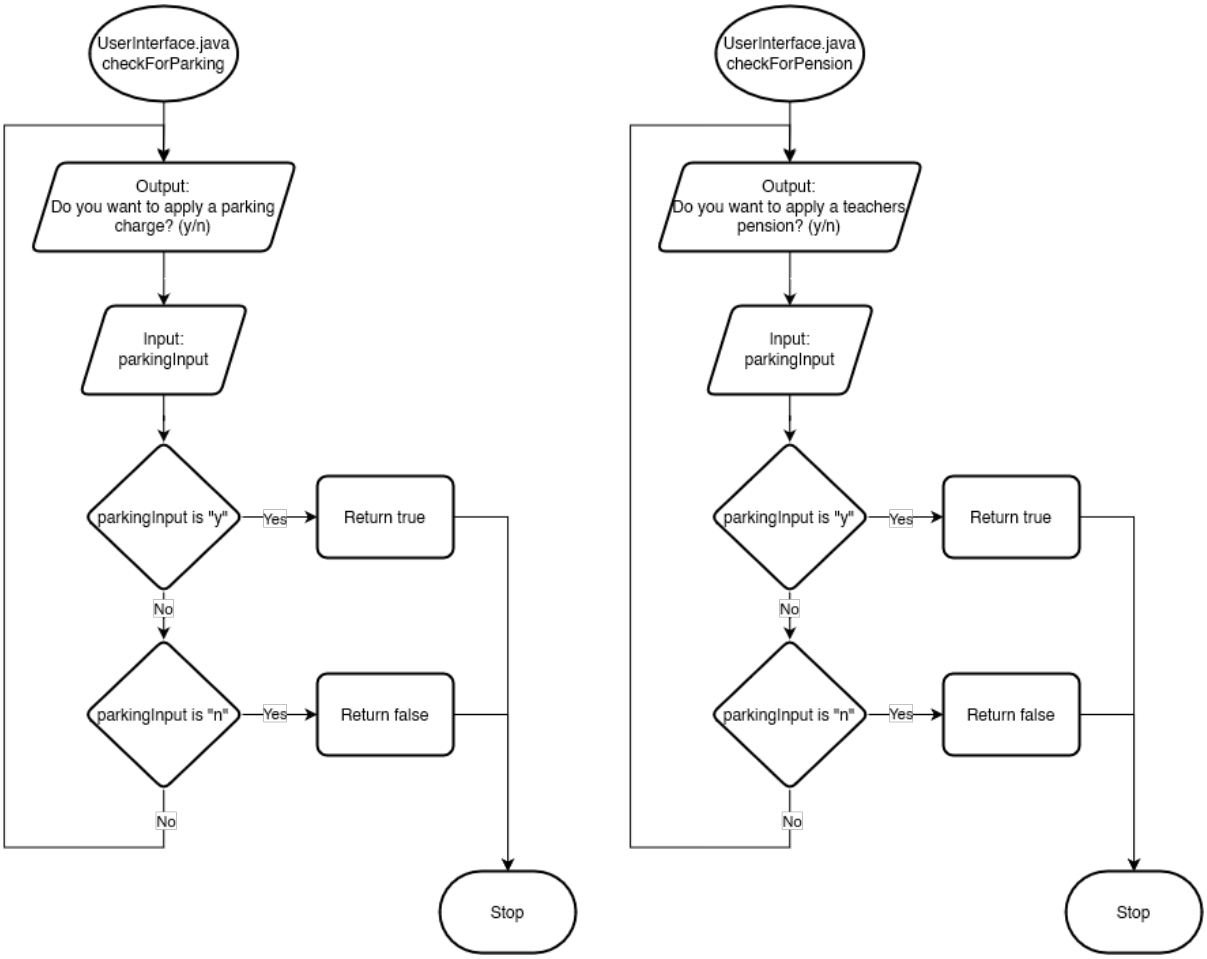

Figure 7: 2nd Flowchart of UserInterface.java

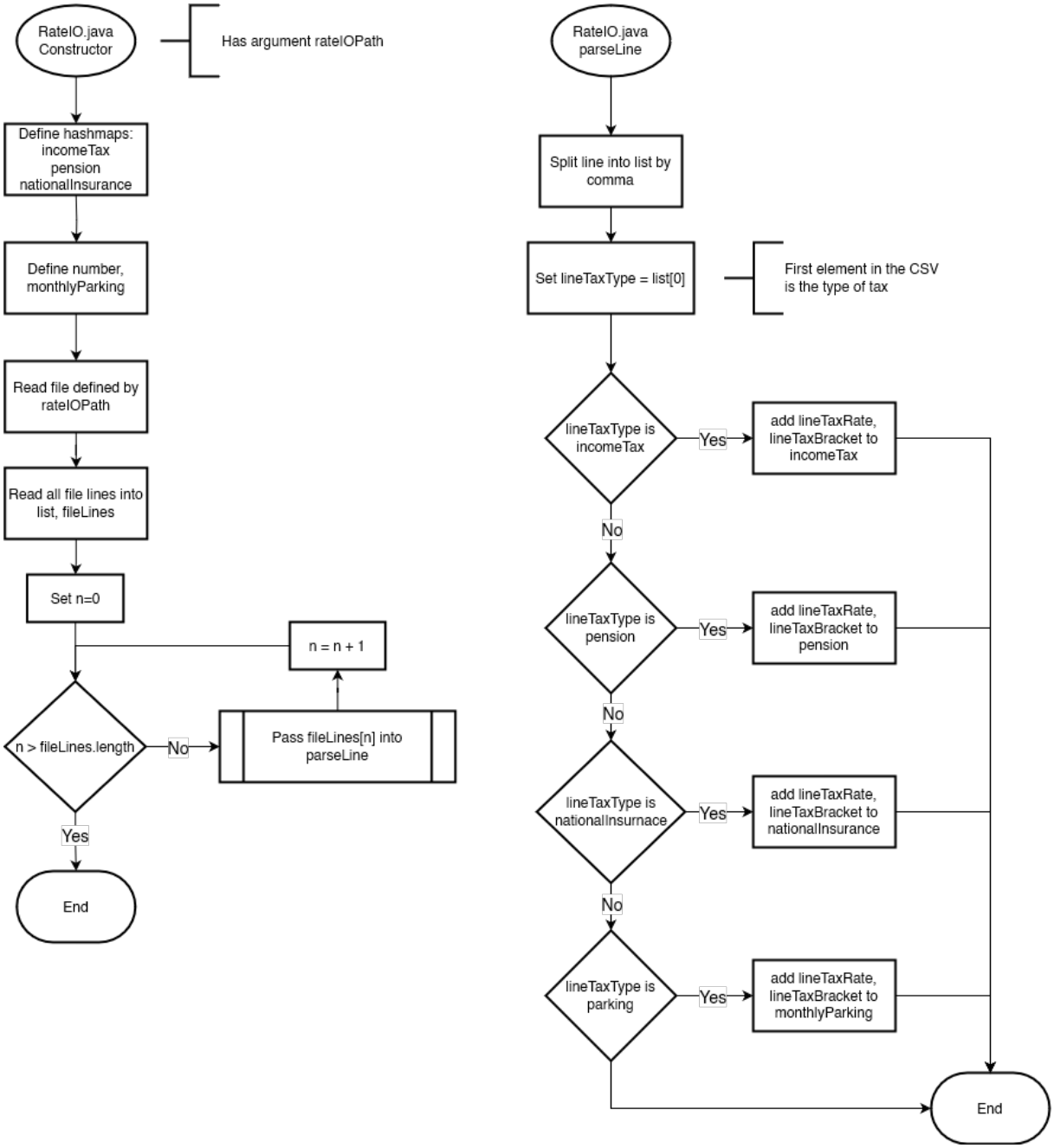

Figure 8: Flowchart of RateIO.java

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
                   sure rates.csv is run in same folder as the " +
15                     "program");
16             return;
17         }
18         Scanner scanner = new Scanner(System.in);
19         UserInterface userInput = new UserInterface(scanner);
20         Employee employee = userInput.createEmployeeLoop();
21         employee.setEmployeeSalary(userInput.getSalaryLoop(rateIO));
22
23         /* Apply income tax and national insurance */
24         employee.getSalary().applyMandatoryDeductions();
25
26         /* Check if user wants to apply optional deductions */
27         if (userInput.userApplyParking()) {
28             employee.getSalary().applyParkingCharge();
29         }
30         if (userInput.userApplyPension()) {
31             employee.getSalary().applyPension();
32         }
33         UserInterface.displayEmployeeSalary(employee);
34     }
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
```

```java
    /**
     * 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()
            );
        }

        if (!(employee.getSalary().getPensionAmount() == null)) {
            System.out.printf("Pension charge: £%s\n",
                employee.getSalary().getPensionAmount()
            );
        }

        System.out.printf("\nTotal deductions: £%s\n",
            employee.getSalary().getTotalDeductions()
        );
        System.out.printf("Yearly net pay: £%s\n",
            employee.getSalary().getNetSalary()
        );


```

```
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                    break;
140                } catch (InputMismatchException e) {
141                    System.out.println(
142                        "Letter are not allowed employee number"
143                    );
144                    /* nextLine clears the newline from nextInt() avoiding
145                    duplicates of above message */
146                    scanner.nextLine();
147                }
148            }
149            return new Employee(employeeName, employeeNumber);
150        }
151
152        /**
153         * UI loop that constructs Salary that is filled with tax
                information
154         *
155         * @param rateIO The tax bands to use in initial instantiation of
                taxes, pension, etc.
156         * @return Constructed Salary object
157         */
158        public Salary getSalaryLoop(RateIO rateIO) {
159
160            BigDecimal yearSalary;
```

```
161
162         while (true) {
163             System.out.print("Yearly salary: ");
164             try {
165                 String inputSalary = scanner.next();
166                 yearSalary = new BigDecimal(inputSalary);
167                 yearSalary = yearSalary.setScale(2, RoundingMode.
                        HALF_UP);
168                 /* Clear the newline character from scanner buffer
169                  * Otherwise next question would appear twice, as the
170                  * scanner would pick up the leftover newline
171                  */
172                 scanner.nextLine();
173                 System.out.println(yearSalary);
174                 break;
175             } catch (NumberFormatException e) {
176                 System.out.println(
177                     "Letter are not allowed in the employee number"
178                 );
179             }
180         }
181         return new Salary(yearSalary, rateIO);
182     }
183
184     /**
185      * Asks user if they want to apply a parking charge
186      *
187      * @return To apply parking charge or not
188      */
189     public boolean userApplyParking() {
190
191         while (true) {
192             System.out.println(
193                 "Do you want to apply a parking charge? (y/n)"
194             );
195             // Normalise characters to lowercase
196             String parkingInput = scanner.nextLine().toLowerCase();
197             switch (parkingInput) {
198                 case "y": {
199                     return true;
200                 }
201                 case "n": {
202                     return false;
203                 }
204             }
205         }
206     }
207
208     /**
209      * Asks the user if they want to apply a teacher's pension
210      *
```

```
211         * @return bool indicating to apply pension or not
212         */
213        public boolean userApplyPension() {
214            while (true) {
215                System.out.println(
216                    "Do you want to apply a teachers pension? (y/n)"
217                );
218                // Normalise characters to lowercase
219                String parkingInput = scanner.nextLine().toLowerCase();
220                switch (parkingInput) {
221                    case "y": {
222                        return true;
223                    }
224                    case "n": {
225                        return false;
226                    }
227                }
228            }
229        }
230    }
```

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      public static BigDecimal convertMonthly(BigDecimal amount) {
36          return amount.divide(new BigDecimal("12"), 2, RoundingMode.
                HALF_UP);
37      }
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,
            LinkedHashMap<BigDecimal, BigDecimal> paymentBands) {
91
92          BigDecimal totalPayment = new BigDecimal("0");
93          BigDecimal previousBracket = new BigDecimal("0");
94
95          for (Map.Entry<BigDecimal, BigDecimal> entry : paymentBands.
                entrySet()) {
96              BigDecimal currentBracket = entry.getKey();
97              BigDecimal bracketRate = entry.getValue;
98
99
100             /*
101              * If the payment is in a band denoted with a negative
102              * number then it is overflow, and applies
103              * that rate to rest of salary
104              */
105             if (currentBracket.compareTo(BigDecimal.ZERO) < 0) {
106                 /* totalPayment = totalPayment +
107                  * (income - previousBand) * taxRate
108                  */
109                 totalPayment = totalPayment.add(
110                     income.subtract(previousBracket).multiply(
                            bracketRate).setScale(2,RoundingMode.HALF_UP)
111                 );
112             } else if (income.compareTo(entry.getKey()) > 0) {
113                 /* If the income is greater than the current
114                  * payment band
115                  */
116
117                 /* totalPayment = totalPayment +
118                  * (currentBracket - previousBand) * taxRate
119                  * It then rounds to 2 decimal places
120                  */
121                 totalPayment = totalPayment.add((
122                     entry.getKey().subtract(previousBracket)).multiply(
123                         entry.getValue()).setScale(2, RoundingMode.
                                HALF_UP)
124                 );
125
126             } else if ((income.compareTo(previousBracket) > 0) && (
                    income.compareTo(entry.getKey()) < 0)) {
127                 /* If the income is smaller than the current payment
128                  * band
129                  */
130
```

```
131                 /* Get the leftover money in the band */
132                 BigDecimal bracketAmount = income.subtract(
                        previousBracket);
133                 /* apply tax to the leftover amount in the band
134                  * totalPayment = totalPayment +
135                  * (leftoverAmount * taxRate)
136                  */
137                 totalPayment = totalPayment.add(
138                     bracketAmount.multiply(entry.getValue()).setScale
                            (2, RoundingMode.HALF_UP)
139                 );
140                 /* Since income is smaller than current band, won't
141                  * make it to next band, break out of loop
142                  */
143                 break;
144             }
145             previousBracket = entry.getKey();
146         }
147         return totalPayment;
148     }
149 
150     // Setters
151 
152     public void setSalary(BigDecimal grossSalary) {
153         this.grossSalary = grossSalary;
154         netSalary = grossSalary;
155         applyMandatoryDeductions();
156     }
157 
158     public void setRateIO(iRateIO rateIO) {
159         this.rateIO = rateIO;
160         applyMandatoryDeductions();
161     }
162 
163     // Getters
164 
165     public BigDecimal getGrossSalary() {
166         return grossSalary;
167     }
168 
169     public BigDecimal getMonthlySalary() {
170         return convertMonthly(grossSalary);
171     }
172 
173     public BigDecimal getTaxableAmount() {
174         return grossSalary.subtract(new BigDecimal("12570"));
175     }
176 
177     public BigDecimal getIncomeTaxAmount() {
178         return totalIncomeTax;
179     }
```

```
180
181     public BigDecimal getNIAmount() {
182         return totalNI;
183     }
184
185     public BigDecimal getPensionAmount() {
186         return totalPension;
187     }
188
189     public BigDecimal getTotalParking() {
190         return totalParking;
191     }
192
193     public BigDecimal getTotalDeductions() {
194         return totalDeductions;
195     }
196
197     public BigDecimal getNetSalary() {
198         return netSalary;
199     }
200
201     public BigDecimal getMonthlyNetSalary() {
202         return netSalary.divide(
203             new BigDecimal("12"), 2, RoundingMode.HALF_UP
204         );
205     }
206 }
```

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

```java
package usw.employeepay;

import java.io.IOException;
import java.math.BigDecimal;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.Arrays;
import java.util.LinkedHashMap;
import java.util.List;

public class RateIO implements iRateIO {
    private final LinkedHashMap<BigDecimal, BigDecimal> taxBands = new LinkedHashMap<>();
    private final LinkedHashMap<BigDecimal, BigDecimal> pensionBands = new LinkedHashMap<>();
    private final LinkedHashMap<BigDecimal, BigDecimal> nationalInsurance = new LinkedHashMap<>();
    private BigDecimal monthlyParking;

    /**
     * Reads a CSV for tax bands, national insurance, and
     * @param filePath String of file path
     * @throws IOException If file does not exist / is not found
     */
    public RateIO(String filePath) throws IOException {
        List<String> lines = Files.readAllLines(Paths.get(filePath));
        // Each line runs the parseLine function
        lines.forEach(line ->
            parseLine(Arrays.asList(line.split(",")))
        );
    }

    /**
     * Handle the separated line and add it to a tax band
     * @param line Line to parse
     */
    private void parseLine(List<String> line) {
        /* Each type of deduction possible in CSV */
        switch (line.get(0)) {
            case "tax" -> taxBands.put(
                new BigDecimal(line.get(1)),
                new BigDecimal(line.get(2))
            );
            case "pension" -> pensionBands.put(
                new BigDecimal(line.get(1)),
                new BigDecimal(line.get(2))
            );
            case "nationalInsurance" -> nationalInsurance.put(
                new BigDecimal(line.get(1)),
```

```
                    new BigDecimal(line.get(2))
                );
                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 {

    TestingFakeRateIO testingRateIO = new TestingFakeRateIO();
    Salary testSalary = new Salary(
        new BigDecimal("45000"), testingRateIO
    );
    Salary testSalaryDecimal = new Salary(
        new BigDecimal("50000"), testingRateIO
    );
    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
40      @Test
41      @DisplayName("Calculate taxable amount")
42      public void getTaxableAmount() {
43          BigDecimal expectedTaxableAmount = new BigDecimal("32430.00");
44
45          assertEquals(0, expectedTaxableAmount.compareTo(
46              testSalary.getTaxableAmount())
47          );
48      }
49
50      @Test
51      @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.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          LinkedHashMap<BigDecimal, BigDecimal> expectedTaxBands = new
                LinkedHashMap<>();
31          expectedTaxBands.put(
32              new BigDecimal("12570"), new BigDecimal("0.00")
33          );
34          expectedTaxBands.put(
35              new BigDecimal("50270"), new BigDecimal("0.20")
36          );
37          expectedTaxBands.put(
38              new BigDecimal("125140"), new BigDecimal("0.40")
39          );
```

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

```
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
```