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

Program Source Code

`Main.java`

```java
package usw.employeepay;

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

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


        } catch (IOException e) {
            System.out.println("File, rates.csv, was not found. Make
                sure rates.csv is run in same folder as the " +
                    "program");
            return;
```

```
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
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().
                        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                 }
```

```
                }
            }
        }
    }
```

Employee.java

```java
package usw.employeepay;

public class Employee {

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

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

    public String getName() {
        return name;
    }

    public int getEmployeeNum() {
        return employeeNum;
    }

    public Salary getSalary() {
        return employeeSalary;
    }

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

Salary.java

```java
package usw.employeepay;

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

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

    iRateIO rateIO;

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

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

    public static BigDecimal convertMonthly(BigDecimal amount) {
        return amount.divide(new BigDecimal("12"), 2, RoundingMode.
            HALF_UP);
    }

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

    public void applyIncomeTax() {
        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
97              /*
```

```
 98                * If the payment is in a band denoted with a negative
 99                * number then it is overflow, and applies
100                * that rate to rest of salary
101                */
102               if (entry.getKey().compareTo(BigDecimal.ZERO) < 0) {
103                   /* totalPayment = totalPayment +
104                    * (income - previousBand) * taxRate
105                    */
106                   totalPayment = totalPayment.add(
107                       income.subtract(previousBracket).multiply(entry.
                              getValue()).setScale(2,RoundingMode.HALF_UP)
108                   );
109               } else if (income.compareTo(entry.getKey()) > 0) {
110                   /* If the income is greater than the current
111                    * payment band
112                    */
113
114                   /* totalPayment = totalPayment +
115                    * (currentBracket - previousBand) * taxRate
116                    * It then rounds to 2 decimal places
117                    */
118                   totalPayment = totalPayment.add((
119                       entry.getKey().subtract(previousBracket)).multiply(
120                           entry.getValue()).setScale(2, RoundingMode.
                                  HALF_UP)
121                   );
122
123               } else if ((income.compareTo(previousBracket) > 0) && (
                      income.compareTo(entry.getKey()) < 0)) {
124                   /* If the income is smaller than the current payment
125                    * band
126                    */
127
128                   /* Get the leftover money in the band */
129                   BigDecimal bracketAmount = income.subtract(
                          previousBracket);
130                   /* apply tax to the leftover amount in the band
131                    * totalPayment = totalPayment +
132                    * (leftoverAmount * taxRate)
133                    */
134                   totalPayment = totalPayment.add(
135                       bracketAmount.multiply(entry.getValue()).setScale
                              (2, RoundingMode.HALF_UP)
136                   );
137                   /* Since income is smaller than current band, won't
138                    * make it to next band, break out of loop
139                    */
140                   break;
141               }
142               previousBracket = entry.getKey();
143           }
```

```
144         return totalPayment;
145     }
146 
147     // Setters
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     // Getters
161 
162     public BigDecimal getGrossSalary() {
163         return grossSalary;
164     }
165 
166     public BigDecimal getMonthlySalary() {
167         return convertMonthly(grossSalary);
168     }
169 
170     public BigDecimal getTaxableAmount() {
171         return grossSalary.subtract(new BigDecimal("12570"));
172     }
173 
174     public BigDecimal getIncomeTaxAmount() {
175         return totalIncomeTax;
176     }
177 
178     public BigDecimal getNIAmount() {
179         return totalNI;
180     }
181 
182     public BigDecimal getPensionAmount() {
183         return totalPension;
184     }
185 
186     public BigDecimal getTotalParking() {
187         return totalParking;
188     }
189 
190     public BigDecimal getTotalDeductions() {
191         return totalDeductions;
192     }
193 
194     public BigDecimal getNetSalary() {
```

```
195             return netSalary;
196         }
197
198         public BigDecimal getMonthlyNetSalary() {
199             return netSalary.divide(
200                 new BigDecimal("12"), 2, RoundingMode.HALF_UP
201             );
202         }
203     }
```

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
            LinkedHashMap<>();
13      private final LinkedHashMap<BigDecimal, BigDecimal> pensionBands =
            new LinkedHashMap<>();
```

```
14      private final LinkedHashMap<BigDecimal, BigDecimal>
            nationalInsurance = new LinkedHashMap<>();
15      private BigDecimal monthlyParking;
16
17      /**
18       * Reads a CSV for tax bands, national insurance, and
19       * @param filePath String of file path
20       * @throws IOException If file does not exist / is not found
21       */
22      public RateIO(String filePath) throws IOException {
23          List<String> lines = Files.readAllLines(Paths.get(filePath));
24          // Each line runs the parseLine function
25          lines.forEach(line ->
26              parseLine(Arrays.asList(line.split(",")))
27          );
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          return nationalInsurance;
61      }
62
```

```
63      public LinkedHashMap<BigDecimal, BigDecimal> getPensionBands() {
64          return pensionBands;
65      }
66
67      public BigDecimal getMonthlyParking() {
68          return monthlyParking;
69      }
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
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(
```

```
            testSalary.getTotalParking())
        );
        assertEquals(0, expectedNetSalary.compareTo(
            testSalary.getNetSalary())
        );
    }

    @Test
    @DisplayName("Total teachers pension")
    void getTotalTeachersPension() {
        BigDecimal expectedTeachersPension = new BigDecimal("3501.76");
        testSalary.applyPension();

        assertEquals(0, expectedTeachersPension.compareTo(
            testSalary.getPensionAmount())
        );
    }

    @Test
    @DisplayName("Total deductions")
    void getTotalDeductions() {
        BigDecimal expectedDeductions = new BigDecimal("10737.84");
        testSalary.applyMandatoryDeductions();

        assertEquals(0, expectedDeductions.compareTo(
            testSalary.getTotalDeductions())
        );
    }

    @Test
    @DisplayName("Net salary")
    void getNetSalary() {
        BigDecimal expectedNetSalary = new BigDecimal("34142.16");
        testSalary.applyMandatoryDeductions();
        testSalary.applyParkingCharge();

        assertEquals(0, expectedNetSalary.compareTo(
            testSalary.getNetSalary())
        );
    }
}
```

RateIOTest.java

```

package usw.employeepay;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.DisplayName;
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;
```

```java

class UserInterfaceTest {

    @Test
    @DisplayName("Valid input in name field")
    void nameValidInput() {

        String dataIn = "Jake Real\n4324324\n423432";
        ByteArrayInputStream in = new ByteArrayInputStream(
            dataIn.getBytes()
        );
        System.setIn(in);

        Scanner scanner = new Scanner(System.in);

        UserInterface userInput = new UserInterface(scanner);
        userInput.createEmployeeLoop();
    }
}
```

Program Outputs

Running Main.java:

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

Calculating yearly net pay...

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

Total deductions: £10737.84
Yearly net pay: £34262.16

Calculating monthly net pay...

Gross salary: £3750.00
Taxable amount: £2702.50
```

```
Tax paid: £540.50
National insurance paid: £354.32

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