**Directions**: Launch BlueJ and load the project Z:\csa\csa\_exercises. *Create the project if it does not already exist.* We will create several classes for various exercises inside this project.

Do not forget to test!!!

***Java Directions****: If you are doing this as a Java exercise, you will need to create a class file for each class and test it in a main method. Sample Java main method follows:*

*Public static void main(String[] args) {*

*}*

**MyMath Class**

**Note**: All of the methods for the MyMath class must be static which will allow you to execute the methods without creating an instance of MyMath.

**DO NOT** create an instance of MyMath during your testing!!!

1. **evenInt()** **method**
   1. Receives an integer as input and returns true if the number is even, otherwise it returns false.
   2. Make the evenInt() method static so you do not need to create an instance of MyMath before calling evenInt()
   3. Call evenInt() from within a main() method.
   4. Print the value returned by the evenInt() method.
   5. Create at least four tests (two even and two odd integers).
2. **maxInt()** **method**
   1. Receives two or three integers as input and returns the highest of those values.
   2. It must be **overloaded** to accept two or three integers. This means creating two maxInt() methods. One accepting two integers and one accepting three.
   3. Make both versions of the maxInt() method static so you do not need to create an instance of MyMath before calling maxInt()
   4. Call maxInt() from within a main() method.
   5. Print the value returned by the method.
   6. Test this method the completely *(all possible combinations)*
      1. How many tests must be created to adequately test this method?

**Paycheck Class**

1. Define a class called Paycheck that calculates the employee’s total pay and prints details about taxes to withdraw and the amount to print on the paycheck.
   1. Attributes:
      1. name (String)
      2. hoursWorked (double)
      3. hourlyRate (double)
   2. Methods:
      1. printCheck()
      2. Tax Calculations - Employees pay:
         1. No taxes if they earned less than $7 per hour and worked less than 40 hours that week.
         2. 3% tax if they earned less than $7 per hour and worked at least 40 hours that week.
         3. 5% tax if they earn between $7 and $10 per hour and they work at least 30 hours in the week **or** worked 40 or more hours in the given week and earn $10 or less per hour.
         4. 7% if they make more than $10 per hour.
      3. After performing the above calculations, the printCheck() method should print something like the sample output seen below:

Employee: Pete Newbee Hours Worked: 33 Hourly Rate: $10 Total taxes: $16.5 Total pay: $313.5

* 1. Create a constructor that accepts name, hoursWorked, and hourlyRate as parameters and sets the appropriate instance variables.
  2. Create getters and setters for each of the instance variables.
  3. **Test this with as many possible combinations as it takes to test each condition!!!**
     1. Create one object instance per test.
     2. Make sure you use getters and setters during testing.
     3. *Remember* – Copy/Paste is your friend when creating this many tests.

**MonthlyStatement Class**

1. Define a class called MonthlyStatement that calculates, and prints, monthly statements sent out to our customers. *This is a common part of Accounts Receivable in today’s accounting systems.*
   1. Attributes:
      1. customerName (String)
      2. invoiceDueDays (int)
      3. currentBalance (double)
   2. Methods:
      1. statementDetails() The details of the statement. Formatted output:

Customer: Peter Newbee Current Balance: $600.9 Age of debt: 20 days

* + 1. calcStatement() **Example message to print:**
       1. - *pay\_message*
  1. Create a constructor that accepts customerName, invoiceDueDays, and currentBalance as parameters and sets the appropriate instance variables.
  2. Create getters and setters for each of the instance variables.
  3. **You must use nested if statements in this method.**
  4. **Test this with as many possible combinations as it takes to test each condition!**
     1. Create one object instance.
     2. After the initial test, use the same object instance by modifying the data values using setters.
     3. Print the next statement using the statementDetails() and calcStatement() methods.
     4. Repeat until you have tested all the conditions.

**Complete Output**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Customer: Pete Newbee Current Balance: 300.0 Age of Debt: 20

First Notice

Please pay us by the first of next month.

**Note:** The first two lines above (including the asterisks) are printed in the statementDetails() method. Each of the last two lines are printed in *separate parts* of the calcStatement() method.

**Conditions**

|  |  |
| --- | --- |
| **if** (Condition) | **pay\_message** |
| invoiceDueDays <= 30 and amountDue <= $1000 | First notice:  Please pay us by the first next month. |
| invoiceDueDays <= 30 amountDue > $1000 | First Notice:  Warning - Bill over $1000. Please pay by the first of next month. |
| invoiceDueDays > 30 and <= 60 and amountDue <= $1000 | Second notice: Past Due bill  Please pay as soon as possible. |
| invoiceDueDays > 30 and <= 60 and amountDue > $1000 | Second notice: Past Due bill  Over $1000 – Please pay as soon as possible. |
| invoiceDueDays > 60 and <= 90 and amountDue <= $1000 | Final notice: Past Due bill  Please pay as soon as possible. |
| invoiceDueDays > 60 and <= 90 and amountDue > $1000 | Final notice: Past Due bill  over $1000 – Please pay as soon as possible. |
| If invoiceDueDays > 90 | You have two weeks to pay or we hand this account over to a collection agency. |