Project: Using Java API Classes

Deliverables

Your project files should be submitted to Web-CAT by the due date and time specified. In order to avoid a late penalty for the project, you must submit your completed code files to Web-CAT by 11:59 p.m. on the due date. If you are unable to submit via Web-CAT, you should e-mail your project Java files in a zip file to your TA before the deadline.

Files to submit to Web-CAT:

- ExpressionEvaluator.java
- MealOrder.java

Specifications

Overview: You will write two programs this week. The first will compute the value generated by a specified expression and the second will read data for a meal order and then interpret and print the formatted meal order information.

- **ExpressionEvaluator.java**

  Requirements: Calculate the following expression for a value x of type double which is read in from the keyboard, and save the result of the expression in a variable of the type double. You must use the `sqrt()`, `abs()` and `pow()` methods of the Math class to perform the calculation. You may use a single assignment statement with a single expression, or you may break the expression into appropriate multiple assignment statements. The latter may easier to debug if you are not getting the correct result.

$$\frac{\sqrt{|1.2x^3 - 10x + 1| + (5x^4 - 7x^3)^2}}{(x^2 + 10)}$$

  Next, determine the number of characters (mostly digits) to the left and to the right of the decimal point in the unformatted result. [Hint: You should consider converting the type *double* result into a String using the static method `Double.toString(result)` and storing it into a String variable. Then, on this String variable use the `indexOf(".")` method from the String class to find the position of the period (i.e., decimal point) and the `length()` method to find the length of the String. Knowing the location of the decimal point and the length, you should be able to determine the number of digits on each side of the decimal point.]

  Finally, the result should be printed using the class java.text.DecimalFormat so that to the right of the decimal there are at most four digits and to the left of the decimal each group of three digits is separated by a comma in the traditional way. Also, there should also be at least one digit on each side of the decimal (e.g., 0 should be printed as 0.0). Hint: Use the pattern "#,##0.0###" in your DecimalFormat constructor. However, make sure you know what this pattern means and how to modify and use it in the future.

Design: Several examples of input/output for the ExpressionEvaluator program are shown below.

Line #	Program output
1	Enter a value for x: 0
2	Result: 0.1
3	# of characters to left of decimal point: 1
4	# of characters to right of decimal point: 1
5	Formatted Result: 0.1

Line #	Program output
1	Enter a value for x: 10.5
2	Result: 438.0203717219859
3	# of characters to left of decimal point: 3
4	# of characters to right of decimal point: 13
5	Formatted Result: 438.0204

Line #	Program output
1	Enter a value for x: -10.5
2	Result: 572.7958155057314
3	# of characters to left of decimal point: 3
4	# of characters to right of decimal point: 13
5	Formatted Result: 572.7958

Line #	Program output
1	Enter a value for x: 123456
2	Result: 7.620605543800056E10
3	# of characters to left of decimal point: 1
4	# of characters to right of decimal point: 18
5	Formatted Result: 76,206,055,438.0006

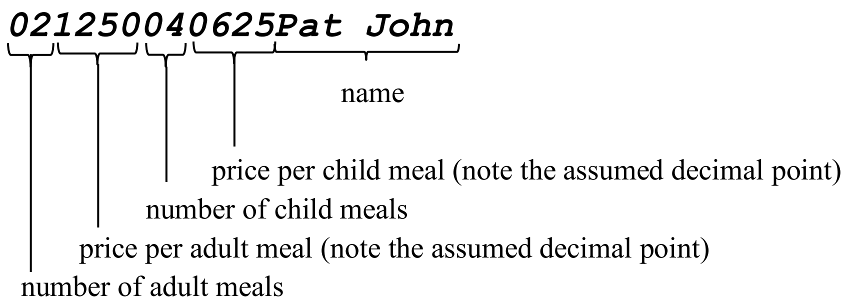
When the characters to the right of the decimal in the unformatted result end with E followed by one or more digits (e.g., E10 indicates an exponent of 10), the 'E' should be included in the count of the characters to the right of the decimal point.

Code: In order to receive full credit for this assignment, you must use the appropriate Java API classes and method to do the calculation and formatting. It is recommended as a practice that you do not modify the input value once it is stored.

Test: You will be responsible for testing your program, and it is important to not rely only on the examples above. Assume that the amount entered can be any positive or negative floating-point number.

- **MealOrder.java**

  Requirements: The purpose of this program is to accept coded information as input that includes the number of adult meals, price of adult meal, number of child meals, price of child meal, and name of the customer. Orders of $100 or more receive a 15% discount (see constants on page 5). The program should then print the user's order information including the total, as well as a "lucky number" between 1 and 99999 inclusive. If the order is less than $100, total is printed; otherwise subtotal, discount, and total are printed. The coded information is formatted as follows:


021250040625Pat John
name
price per child meal (note the assumed decimal point)
number of child meals
price per adult meal (note the assumed decimal point)
number of adult meals

  Whitespace before or after the coded information should be disregarded (e.g., if the user enters spaces or tabs before the coded information then the spaces should be disregarded). Your program will need to print the customer's name, number of adult meals, price per adult meal, number of child meals, price per child meal. If order is less than $100, total is printed; otherwise, subtotal, discount, and total are printed. Finally, a random "lucky" number in the range 1 to 99999 is printed. If the user enters a code that does not have at least 12 characters, then an error message should be printed. [For this assignment, the first 12 characters, if present, will need to be digits; otherwise a runtime exception will occur.]

  Design: Several examples of input/output for the program are shown below.

Line #	Program output
1	`Enter your order code: 123456789`
2 (blank)	
3	`Invalid Order Code.`
4	`Order code must have at least 12 characters.`

  Note that the code below has five leading spaces.

Line #	Program output
1	`Enter your order code:      021250040625Pat John`
2 (blank)	
3	`Name: Pat John`
4	`Adult meals: 2 at $12.50`
5	`Child meals: 4 at $6.25`
6	`Total: $50.00`
7	`Lucky Number: 68408`

Line #	Program output
1	`Enter your order code: 241495000000Sam Smith`
2 (blank)	
3	`Name: Sam Smith`
4	`Adult meals: 24 at $14.95`
5	`Child meals: 0 at $0.00`
6	`Subtotal: $358.80`
7	`15% Discount: -$53.82`
8	`Total: $304.98`
9	`Lucky Number: 35344`

Line #	Program output
1	`Enter your order code: 021500120500Penny's Party`
2 (blank)	
3	`Name: Penny's Party`
4	`Adult meals: 2 at $15.00`
5	`Child meals: 12 at $5.00`
6	`Total: $90.00`
7	`Lucky Number: 28930`

Line #	Program output
1	`Enter your order code: 202500501000Jackie's Big Party`
2 (blank)	
3	`Name: Jackie's Big Party`
4	`Adult meals: 20 at $25.00`
5	`Child meals: 50 at $10.00`
6	`Subtotal: $1,000.00`
7	`15% Discount: -$150.00`
	`Total: $850.00`
	`Lucky Number: 02304`

Code: In order to receive full credit for this assignment, you must use the appropriate Java API classes and methods to trim the input string, to do the extraction of the substrings, conversion of substrings of digits to numeric values as appropriate, and formatting. These include the String methods trim, charAt, and substring, as well as wrapper class methods such as Integer.parseInt and Double.parseDouble which can be used to convert a String of digits into a numeric value. The dollar amounts should be formatted so that both small and large amounts are displayed properly, and the prize number should be formatted so that five digits are displayed including leading zeroes, if needed, as shown in the examples above. It is recommended as a practice that you not modify input values once they are stored. While not a requirement, you should consider making the student discount and faculty/staff discount constants. For example, the following statements could be placed above the main method.

```
static final double DISCOUNT = 0.15;
static final double DISCOUNT_THRESHOLD = 100.0;
```

Test: You are responsible for testing your program, and it is important to not rely only on the examples above. Remember, when entering standard input in the Run I/O window, you can use the up-arrow on the keyboard to get the previous values you have entered. This will avoid having to retype the meal order data each time you run your program.

Grading

Web-CAT Submission: You must submit both "completed" programs to Web-CAT at the same time. Prior to submitting, be sure that your programs are working correctly and that they have passed Checkstyle. **If you do not submit both programs at once, Web-CAT will not be able to compile and run its test files with your programs which means the submission will receive zero points for correctness**. I recommend that you create a jGRASP project and add the two files. Then you will be able to submit the project to Web-CAT from jGRASP. Activity 1 (pages 5 and 6) describes how to create a jGRASP project containing both of your files.

Hints

MealOrder.java

1. The meal order code should be read in all at once and stored in a variable of type String, after which the individual values should be extracted using the substring method.
2. The String value for the number of meals should be converted to type int (using Integer.parseInt) so that it can be used in calculations.
3. The String value for price values should be converted to type double (using Double.parseDouble) so that it can be used in calculations.
4. Use an if-else statement to print total versus subtotal, discount, total. That is, convert the following pseudo code into Java:

```
if total is less than DISCOUNT_THRESHOLD
   print total
else
   print subtotal, discount, and total.
```

5. An appropriate DecimalFormat object should be created using the pattern `"$#,##0.00"` so that its format method can be called when printing the values for meal prices, subtotal, discount, and total for proper formatting.