Project: Using Java API Classes

Deliverables

Your project files should be submitted to Web-CAT by the due date and time specified. 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:

- RadicalFormula.java
- BoxLunch.java

Specifications

Overview: You will write two programs this week. The first will solve for the result of a specified expression using methods from the Math class, and the second will read data for a coded box lunch order and then interpret and print the formatted order information.

- **RadicalFormula.java**

  Requirements: Solve for the result of the expression in the formula below for a value x of type double which is read in from the keyboard, and save the result in a variable of the type double. You must use the `pow`, `sqrt`, and `abs` 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.

$$\sqrt{(3x^8 - 2x^3)^2 + |3x^5 - 2x^3|}$$

  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 invoke 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 three 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##"` when you create your DecimalFormat object. 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 RadicalFormula program are shown below.

Example #1

Line #	Program output
1	`Enter a value for x: 1`
2	`Result: 1.4142135623730951`
3	`# digits to left of decimal point: 1`
4	`# digits to right of decimal point: 16`
5	`Formatted Result: 1.414`
6	

Example #2

Line #	Program output
1	`Enter a value for x: -2.5`
2	`Result: 4608.915111500769`
3	`# digits to left of decimal point: 4`
4	`# digits to right of decimal point: 12`
5	`Formatted Result: 4,608.915`
6	

Example #3

Line #	Program output
1	`Enter a value for x: 5.1`
2	`Result: 1372773.0387854169`
3	`# digits to left of decimal point: 7`
4	`# digits to right of decimal point: 10`
5	`Formatted Result: 1,372,773.039`
6	

Example #4

Line #	Program output
1	`Enter a value for x: 1234.567`
2	`Result: 1.618969128395518E25`
3	`# digits to left of decimal point: 1`
4	`# digits to right of decimal point: 18`
5	`Formatted Result: 16,189,691,283,955,180,000,000,000.0`
6	

When the characters to the right of the decimal in the unformatted result end with E followed by one or more digits (e.g., E25 indicates an exponent of 25), the ‘E’ should be included in the count of the characters to the right of the decimal point.

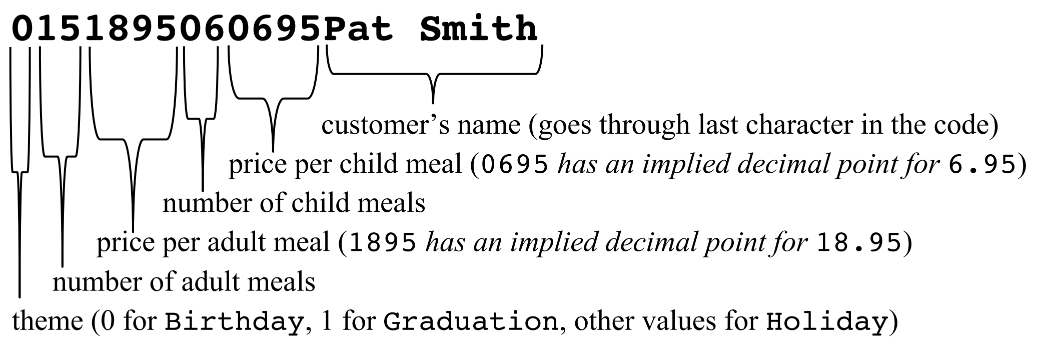
Code: To receive full credit for this assignment, you must use the appropriate Java API classes and method to do the calculation and formatting. It is recommended as a practice that you do not modify the input value once it is stored.

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.

- **BoxLunch.java**

Requirements: The purpose of this program is to accept coded box lunch order as input that includes the theme, the number of adult meals, price of adult meal, number of child meals, price of child meal, and name of the customer. Note that the four digits for the two price fields have an implied decimal point. The program should then print the user's order information including the total, as well as a "lucky number" between 1 and 9999 inclusive that should always be printed as four digits (e.g., 1 should be printed as 0001).

The coded input is formatted as follows:



0151895060695Pat Smith
customer's name (goes through last character in the code)
price per child meal (0695 *has an implied decimal point for* 6.95)
number of child meals
price per adult meal (1895 *has an implied decimal point for* 18.95)
number of adult meals
theme (0 for Birthday, 1 for Graduation, other values for Holiday)

Whitespace before or after the coded information should be disregarded (e.g., if the user enters spaces or tabs before or after the coded information, these 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, total, the theme, and a random "lucky" number in the range 1 to 9999 as shown in the examples below. If the user enters a code that does not have at least 13 characters, then an error message should be printed. [The 14th character if present is part of the customer's name.]

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

The box lunch order code below is invalid since it does not include the required minimum of 13 characters.

Example #1

Line #	Program output
1	`Enter order code: 123456789`
2	
3	`*** Invalid Order Code ***`
4	`Order code must have at least 13 characters.`
5	

Note that the order codes entered in the examples below result in the indicated output except for the prize number which is random.

Example #2

```
Enter order code: 0151895060695Pat Smith

Name: Pat Smith
Adult meals: 15 at $18.95
Child meals: 6 at $6.95
Total: $325.95
Theme: Birthday
Lucky Number: 1008

```

Example #3

```
Enter order code: 1091895020695Sam Jones

Name: Sam Jones
Adult meals: 9 at $18.95
Child meals: 2 at $6.95
Total: $184.45
Theme: Graduation
Lucky Number: 3737

```

Example #4 The order code below has five leading spaces (<u>be sure to trim the input code</u>).

```
Enter order code:      7431550290425Becky's Big Party

Name: Becky's Big Party
Adult meals: 43 at $15.50
Child meals: 29 at $4.25
Total: $789.75
Theme: Holiday
Lucky Number: 9645

```

Code: To receive full credit for this assignment, you must use the appropriate Java API classes and methods to trim the input string, to extract the substrings, conversion of substrings of digits to numeric values as appropriate, and formatting. These include the String methods trim, and substring, as well as wrapper class methods such Double.parseDouble and Integer.parseInt which can be used to convert a String of digits into a numeric value for price and number of meals. 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 seven 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.

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 order code each time you run your program.

Hints:

1. The order code should be read in all at once using the Scanner's nextLine method and stored in a variable of type String. Then the individual values should be extracted using the substring method. The String value for the meal prices should be converted to type double (using Double.parseDouble) and the String value for number of meals should be converted to type int (using Integer.parseInt) so that these can be used to Total cost. When printing the values for meal prices, total, and prize number, they should be formatted properly by creating an appropriate DecimalFormat object (see patterns below) and calling its format method.

   To format meal prices and total, use the pattern `"$#,##0.00"` when you create your DecimalFormat object. The number of meals does not require formatting.

   For prize number, use the pattern `"0000"` when you create your DecimalFormat object.

2. Since all items in the order code other than the prices and number of meals will not be used in arithmetic expressions, they can be left as type String.

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.