# Project 3: Using Java API Classes

**Due**: Monday, February 8, 2016 by 11:59 PM

## Deliverables

Your project files should be submitted to Web-CAT by the due date and time specified above (see the Lab Guidelines for information on submitting project files). In order to avoid a late penalty for the project, you must submit your completed code files to Web-CAT no later than 11:59 PM on the due date for the completed code. You may submit your project up to 24 hours after the due date, but there is a late penalty of 15 points. No projects will be accepted after the one day grace period. If you are unable to submit via Web-CAT, you should e-mail your project Java files in a zip file to your lab instructor before the deadline.

Files to submit to Web-CAT:

- RadicalFormula.java
- SuperBowlBoxLunch.java

## Specifications

**Overview:** You will write two programs this week. The first will compute the value generated by a specified formula and the second will calculate the total for a food order from a set menu.

- **RadicalFormula.java**

  **Requirements**: Calculate the following expression for any value of the variable x, and save the result 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 somewhat large expression, or you may break the formula 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 digits to the left and to the right of the decimal point in the result. [Hint: You should consider converting the result of type *double* 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 and the *length( )* method to find the length. 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).

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

| Line number | 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` |

| Line number | 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` |

| Line number | 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` |

| Line number | 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` |

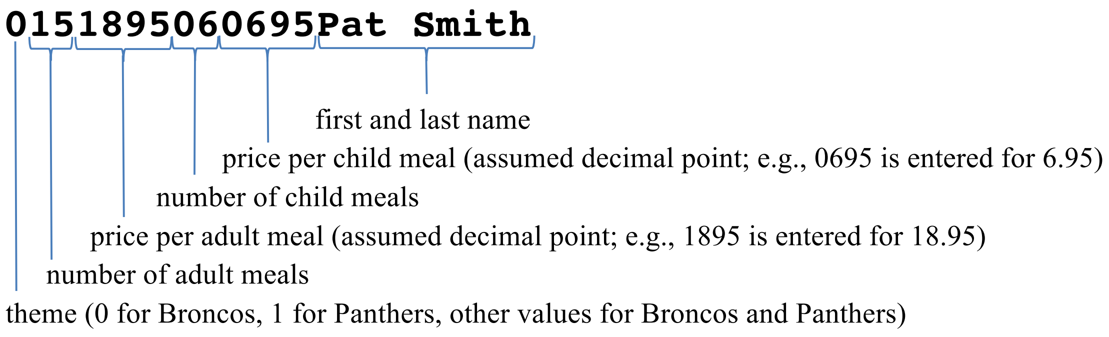
Note that in the example above the digits to the right of the decimal in the unformatted result are followed by E25 (indicating an exponent of 25). For this program, the "E25" should be included in the count of the digits to the right of the decimal point.

**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 input values once they are 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.

- **SuperBowlBoxLunch.java**

**Requirements**: The purpose of this program is to accept coded information for a Super Bowl box lunch order as input that includes the Super Bowl theme, the number of adult meals, price of adult meal, number of child meals, price of child meal, and name of the customer. The program should then print the user's order information including the total, as well as a "lucky number" between 1 and 9999 inclusive. The coded information is formatted as follows:


0151895060695Pat Smith
first and last name
price per child meal (assumed decimal point; e.g., 0695 is entered for 6.95)
number of child meals
price per adult meal (assumed decimal point; e.g., 1895 is entered for 18.95)
number of adult meals
theme (0 for Broncos, 1 for Panthers, other values for Broncos and Panthers)

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, total, the theme and a random "lucky" number in the range 1 to 9999. If the user enters a code that does not have at least 13 characters, then an error message should be printed. For this assignment, the first 13 characters, if present, will need to be digits; otherwise a runtime exception will occur. The 14th character if present is part of the name.

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

| Line number | Program output |
|---|---|
| 1 | `Enter order code: 123456789` |
| 2 (blank line) | |
| 3 | `*** Invalid Order Code ***` |
| 4 | `Order code must have at least 13 characters.` |

Note that the order code entered below has five leading spaces.

| Line number | Program output |
|---|---|
| 1 | `Enter order code: 0151895060695Pat Smith` |
| 2 | |
| 3 | `Name: Pat Smith` |
| 4 | `Adult meals: 15 at $18.95` |
| 5 | `Child meals: 6 at $6.95` |
| 6 | `Total: $325.95` |
| 7 | `Theme: Broncos` |
| 8 | `Lucky Number: 3808` |

| Line number | Program output |
|---|---|
| 1 | `Enter order code: 1091895020695Sam Jones` |
| 2 | |
| 3 | `Name: Sam Jones` |
| 4 | `Adult meals: 9 at $18.95` |
| 5 | `Child meals: 2 at $6.95` |
| 6 | `Total: $184.45` |
| 7 | `Theme: Panthers` |
| 8 | `Lucky Number: 2632` |

| Line number | Program output |
|---|---|
| 1 | `Enter order code: 7431550290425Becky's Big Party` |
| 2 | |
| 3 | `Name: Becky's Big Party` |
| 4 | `Adult meals: 43 at $15.50` |
| 5 | `Child meals: 29 at $4.25` |
| 6 | `Total: $789.75` |
| 7 | `Theme: Broncos and Panthers` |
| 8 | `Lucky Number: 0914` |

**Code**: In order to receive full credit for this assignment, you must use the appropriate Java API classes and methods to do the extraction of the substrings, calculation, and formatting. These include the String methods trim 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 lucky number should be formatted so that four digits are displayed including leading zeros if needed. It is recommended as a practice that you do 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 ticket info data each time you run your program. You should use the jGRASP debugger and viewer canvas to view the entered code. "Step" in the debugger until the code is read in, then open a viewer on code variable and set "Viewer" to "Presentation Char Array" to see the index of each character in the code.

## 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 have passed Checkstyle. **If you do not submit both programs at the same time, the submission will receive zero points for correctness because Web-CAT will attempt to compile both programs with the JUnit test cases but will fail**. Activity 1 describes how to create a jGRASP project containing both of your files.