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. 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 late period. 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
- MoonTrip.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 ticket to a destination in space and then interpret and print the formatted ticket 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.

$$\frac{\sqrt{|23.7x^9| - x}}{(7.3x^2 + 5.2x + 3.1)}$$

Next, determine the number of 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 five 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 number	Program output
1 2 3 4 5	Enter a value for x: 0 Result: 0.0 # digits to left of decimal point: 1 # digits to right of decimal point: 1 Formatted Result: 0.0

Line number	Program output
1 2 3 4 5	Enter a value for x: 12.6 Result: 354.81017620992213 # digits to left of decimal point: 3 # digits to right of decimal point: 14 Formatted Result: 354.81018

Line number	Program output
1 2 3 4 5	Enter a value for x: -12.6 Result: 397.21157911023744 # digits to left of decimal point: 3 # digits to right of decimal point: 14 Formatted Result: 397.21158

Line number	Program output
1 2 3 4 5	Enter a value for x: 9876543210 Result: 6.464928605131014E24 # digits to left of decimal point: 1 # digits to right of decimal point: 18 Formatted Result: 6,464,928,605,131,014,000,000,000.0

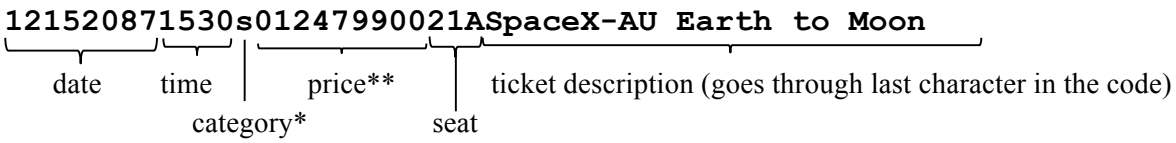
When the digits to the right of the decimal in the unformatted result end with E followed by one or more digits (e.g., E24 indicates an exponent of 26), the 'E' should be included in the count of the digits to the right of the decimal point.

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 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.

- **MoonTrip.java**

Requirements: The purpose of this program is to accept coded ticket information to the moon and back as input that includes the date, time, category, price, and seat, followed by the description of the travel. Note that the nine digits for price have an implied decimal point. The program should then print the ticket information including the actual cost, which is the price with discount applied as appropriate: 40% for a student ticket (`s`), 20% for employee ticket (`e`), and none for regular tickets (i.e., anything other than `s` or `e`). The last line of the ticket should contain a "prize number" between 1 and 9999 inclusive that should always be printed as 4 digits (e.g., 7 should be printed as 0007). The coded input is formatted as follows:



*category [`s`, `e`, any other character is a regular ticket with no discount; store this as type `char`]
**price [`012479900` *has an implied decimal point; the* `double` *value should be* `124799.00`]

Whitespace (e.g., spaces or tabs) before or after the coded information should be disregarded. (Hint: After the ticket code has been read in as a String, it should be trimmed.) Your program will need to print the date and time, seat, the itinerary (i.e., ticket description), the ticket price, the ticket category, the actual cost, and a random prize number in the range 1 to 9999. If the user enters a code that does not have at least 26 characters, then an error message should be printed and the program should end. [The 26th character of the code is part of the ticket description.]

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

Line #	Program output
1	`Enter ticket code: 123456789012345`
2	
3	`*** Invalid ticket code ***`
4	`Ticket code must have at least 26 characters.`

Note that the ticket code below results in the indicated output except for the prize number which is random. When more than one item is shown on the same line (e.g., date, time, and seat on line 4), there are three spaces between them (**do not use the tab escape sequence \t**).

Line #	Program output
1	`Enter ticket code: 121520871530s01247990021ASpaceX-AU Earth to Moon`
2	
3	`Date: 12/15/2087   Time: 15:30   Seat: 21A`
4	`Itinerary: SpaceX-AU Earth to Moon`
5	`Price: $124,799.00   Category: s   Cost: $74,879.40`
6	`Prize Number: 9230`

Line #	Program output
1	Enter ticket code: 121520871530e01247990021ASpaceX-AU Earth to Moon
2	
3	Date: 12/15/2087 Time: 15:30 Seat: 21A
4	Itinerary: SpaceX-AU Earth to Moon
5	Price: $124,799.00 Category: e Cost: $99,839.20
6	Prize Number: 0071

Note that the ticket code below has five leading spaces (be sure you are trimming the input code).

Line #	Program output
1	Enter ticket code: 121520871530z01247990021ASpaceX-AU Earth to Moon
2	
3	Date: 12/15/2087 Time: 15:30 Seat: 21A
4	Itinerary: SpaceX-AU Earth to Moon
5	Price: $124,799.00 Category: z Cost: $124,799.00
6	Prize Number: 1655

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 category character, 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 method 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 four 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 child discount constants. For example, the following statements could be placed above the main method.

```
static final double STUDENT_DISCOUNT = .40;
static final double EMPLOYEE_DISCOUNT = .20;
```

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.

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. Activity 1 (pages 5 and 6) describes how to create a jGRASP project containing both of your files.

Hints

MoonTrip class

1. The ticket 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. The String value for price should be converted to type double (using Double.parseDouble) so that it can be used to calculate cost. When printing the values for price and cost, they should be formatted properly by creating an appropriate DecimalFormat object and calling its format method.

   Since all items other than the price will not be used in arithmetic expressions, they can be left as String values (or char value in the case of category).

2. Since char values are primitive types, == and `!=` can be used to compare two values for equality and inequality respectively. Thus, if the category is extracted from the input using the String method charAt() which returns a char, then == and `!=` can be used to compare char values.

   Otherwise, if category is a String, you should not use == and `!=` to compare two String values. String values should be compared for equality using the String equals method which has a boolean return type. For example, if s1 and s2 are String objects, to check to see if their respective character strings are equal you should use

   ```
   s1.equals(s2)
   ```

   rather than

   ```
   s1 == s2
   ```

   which is only true if s1 and s2 are aliases.

The time and date should have leading zeros as appropriate. Therefore, these can be printed as String values by concatenating their components with `":"` and `"/"` as needed.