Program #3

Due: **11/7/18 at 10:00 PM**
Late Submissions: **11/8/18 at 10:00 PM**
11/9/18 at 10:00 PM
Files (10): **AdditionExpression.java, DivisionExpression.java, Expression.java, InvalidQuestionException.java, MultiplicationExpression.java, NumberConvertor.java, NumberWord.java, QuestionTemplate.java, SubtractionExpression.java, WordMath.java**

For this assignment, you will write a program that can solve simple math problems in a conversational style. You will demonstrate your understanding of abstract classes and interfaces, while also reinforcing your ability to work with strings and exceptions. The details of each class are given in UML form below. In terms of total lines of code, this will be one of the longest programs you write this semester. Therefore, I encourage you to start early. Also, the program will be more manageable if you develop the classes in the order described and test them as you go. Your program should robustly handle any input the user gives. Note, it is essential that the names and types of all data fields and the names, return types and parameter lists of all methods appear exactly as described. Otherwise, you may lose points on the assignment.

NumberWord	*implements the Comparable<> interface*
-value:int	The value of the number as an integer
-inWords:String	The word(s) used to express the number
+NumberWord(value:int, inWords:String)	Initialize the fields.
+getValue():int	Get method for **value**
+getInWords():String	Get method for **inWords.**
+compareTo(obj:NumberWord):int	Compares the current number to the **obj**, using **inWords.** That is, numbers are ordered alphabetically by their spellings.

NumberConvertor	A utility for converting numbers to words and vice versa.
-FIRST_NUMBERS:String[]	A constant containing the word forms for the numbers 0 through 19, inclusive.
-TENS_NUMBERS:String[]	A constant containing the word forms for all of the multiples of 10 between 20 and 90 inclusive.
-numbersList:ArrayList<NumberWord>	An ArrayList containing NumberWords for 0-19 and the multiples of 10 from 20-90.
+NumberConvertor()	Initializes **numbersList** by creating the appropriate NumberWords objects and then sorts the list by the spellings of the words.
-lookupValueOfWord(numberWord:String):int	Given a single number word, uses binary search (see Sect. 7.10.2, pp. 268-271) to look it up and return the associated int value. If not found, returns -1.

+toNumber(numberWords:String):int	Using the **numbersList**, converts the words for any number between 0 and 99 inclusive into its integer equivalent. Returns -1 for any number that it does not recognize.
+toWords(value:int):String	Converts a numeric value between 0 to 99 into the equivalent word(s). If the number is outside the range, returns "not a number I can put into words (*value*)"

InvalidQuestionException	*A subclass of Exception.* Should be thrown whenever the program is unable to interpret the user's question (or a part of it).
	no fields
+InvalidQuestionException()	Set the exception message to "Question not understood"

Expression	Represents a simple arithmetic problem.
#leftOpInt:int	The value of the left operand
#rightOpInt:int	The value of the right operand
+numConvertor:NumberConvertor	A single **NumberConvertor** that can be used by any object that descends fromExpression.
+Expression(leftOp:String, rightOp:String)	Initialize the **leftOpInt** and the **rightOpInt** using strings that express their values in words. If either operand cannot be interpreted as a number, throws an InvalidQuestionException.
+*evaluate()*:int	Perform the arithmetic operation using the left and right operands and return the result
+evaluateInWords():String	Perform the arithmetic operation and return the numeric result expressed as words.

There should be four concrete subclasses of **Expression: AdditionExpression, SubtractionExpression, MultiplicationExpression** and **DivisionExpression.** Each should only have a constructor and an implementation of the **evaluate()** method appropriate to the operation. Note, **DivisionExpression** should perform integer division (i.e., drop fractions).

QuestionTemplate	A pattern for one type of arithmetic question.
-preText:String	Text that must occur at the beginning of the question.
-middleText:String	Text that must occur somewhere in the question.
-endText:String	Text that must occur at the end of the question.
-operation:int	An integer that identifies the operation specified by the question. Uses the constants defined in the WordMath class.
+QuestionTemplate(preText:String, middleText:String, endText: String, operation:int	Initialize the fields.

+isMatch(question:String):boolean	Returns true if **question** matches the template. To match, it must start with **preText**, end with **endText**, and contain **middleText.**
+parseQuestion(question:String):Expression	Given a **question** for which **isMatch()** is known to be true, returns the equivalent subtype of **Expression** for the question. The left operand for the expression is assumed to be the string between the **pretext** and the **middleText.** The right operand is the string between **middleText** and **endText.** If either operand is invalid, then throw an InvalidQuestionException.

WordMath	
+ADDITION:int +SUBTRACTION:int +MULTIPLICATION:int +DIVISION:int	Constants representing the possible arithmetic operations. You can set them to any integer value, as long as each has a unique value and you always use the constants instead of the values. Typically, programmers start at 0 and count up.
+parseQuestion(question:String, templates:ArrayList):Expression	Attempts to match **question** to one of the entries in **templates.** For the first match found, parses the question using the template and returns the resulting **Expression.** If no matching template is found, throws an InvalidQuestionException.
+main(args:String[]):void	The main method. Creates an ArrayList containing **QuestionTemplate** objects for each of the question types described below. Asks the user to enter a question and then attempts to parse the question. If the question can be parsed, evaluates the expression and returns the answer in word form. If the question cannot be parsed, prints an error message. Note, a single run should never give the user an opportunity to reenter the question or enter a new question.

Numbers as Words

Every number can be written as an English word or word-phrase, such as “four,” “twelve,” “seventeen”, “thirty”, and “fifty-seven.” Note that for numbers over twenty, a hyphen is used between the number indicating the tens place (e.g., “fifty”), and the number indicating the ones place (e.g., “seven”). Of course, if the ones place is 0, then we only use the tens place word (i.e., “thirty” not “thirty-zero”).

Question Templates
This program is designed to answer arithmetic questions expressed in English. It is very difficult for programs to understand arbitrary English sentences, but if we control the kinds of sentences that we expect, then we can actually do quite well. Our program should be able to handle questions of the forms below:

What is *number* plus *number*?
Tell me the sum of *number* and *number*?
What do I get if I add *number* and *number* together?
What is *number* minus *number*?
Number less *number* is what?
What is *number* times *number*?
What do I get when I multiply *number* and *number*?
What is *number* divided by *number*?

Note, in all of the above examples, *number* stands for any number word (or words) for numbers between 0 and 99 inclusive.

The templates can be represented using the **QuestionTemplate** class. For example, the object for the "What is *number* plus *number*?" template would have the following field values:

preText: "What is"
middleText: "plus"
endText: "?"
operation: WordMath.ADDITION

Note, in the case of the "*Number* less *number* is what?" template , the **preText** is simply the empty string "".

Hints:

- Recall that constants are **static** and **final.**
- You can use the static method **sort** in the java.util.Collections class to sort an ArrayList of **Comparable**<> objects based on the compareTo() method that has been defined for them (e.g., an ArrayList of **NumberWord**).
- Given that there is a standard way to construct the word forms for numbers between 20 and 99, you should avoid explicitly enumerating the word forms for each possible value. Instead, make use of the **numbersList** field and the **lookupValueOfWord()** method as defined in **NumberConvertor** when implementing the **toNumber()** method.
- Use the FIRST_MUMBERS and TENS_NUMBERS constants to efficiently implement the **toWords()** method in NumberConvertor.
- Don't forget that you can initialize a static object reference variable at the same time that you declare it. Do not initialize static variables in constructors.
- There is a version of **indexOf** in the String class that takes a String as its parameter.
- When trying to match strings, remember that even a single extra whitespace on the end of one can result in it not being equivalent to the other. To remove all whitespace from the ends, use the **trim()** method.
- For one type of phrase, the number may start the sentence, and thus begin with a capital letter. Be sure you can handle such number words.

Sample output:

Below are the results of some example runs.

```
> java WordMath
Enter a question:
What is thirty-three plus eleven?
The answer is forty-four

> java WordMath
Enter a question:
Seventy-two less twenty-four is what?
The answer is forty-eight

> java WordMath
Enter a question:
What do I get when I multiply three and eleven?
The answer is thirty-three

> java WordMath
Enter a question:
What is the product of eight and nine?
I'm sorry, I don't understand the question. Please rerun and
rephrase your question.

> java WordMath
Enter a question:
What is twenty-two times six?
The answer is not a number I can put into words (132)
>
```

Comments

At a minimum provide Javadoc comments that summarize each class and explain what each method does. Remember, Javadoc comments start with the characters "/**" and end with "*/". Include additional comments for lines that are especially complicated. At the top of the WordMath.java file include a comment with the following form:

```
/*
CSE 17
Your name
Your user id
Tutor: name (e-mail)   [only include if a tutor aided you]
Program #3      DEADLINE: November 7, 2018
Program Description: Conversational Arithmetic
*/
```

Submission

Make sure that you have named your classes and files as specified, and that your main method is in the **WordMath** class. Once the program compiles, runs, and has been tested to your satisfaction, upload all of the **.java** files to Course Site. To do so, click on the name of the assignment in the Course Site page, and then press the "Add submission" button at the bottom of the next page. Drag and drop each file into the area under "File submissions". If necessary, you can update your submission at any time before the deadline passes.