Pair Assignment: Making an Interpreter

Java's use of Reflection means that we can access classes and methods on the fly, and we can create new objects and invoke methods from String input. To illustrate these abilities and to cement your understanding of interpreters, you will create a very small scale "interpreter" for Java. This program will work to load and execute methods from user input. However, this program will only work on a small subset of Java. We will only deal with literals that are ints or Strings, and we will only read two types of Java statements: object creation and method calls. To refer to a standard Java class,we will use the fully specified class name. Furthermore, we will not really do syntax-checking. If a line has the word "new" in it, we will assume it's an object creation line. We won't deal with compound method calls, either. Be sure to catch each kind of exception that can occur and give a detailed error message rather than crashing.

Here is a sample interaction with the Interpreter:

```
This is a simple interpreter. I'm not a good compiler, so be careful and follow my special
rules:
Each class name should be fully qualified,
I only create objects and call methods.
I only use literals of integers and Strings.
Enter 'Q' to quit.
java.lang.String word = new java.lang.String("Interpreter");
ok. I have a new java.lang.String called word
java.lang.String nextWord = word.substring(0,4);
I made a new object. Result was Inte
word = word.concat(nextWord);
I changed the value of word to my result, InterpreterInte
java.lang.Integer eIndex = word.indexOf("e");
I made a new object. Result was 3
```

When you finish reading this write-up, start by checking out the code written for you.

Some Definitions

Before starting, make sure you understand the difference between *formal* and *actual* parameters (See http://en.wikipedia.org/wiki/Parameter_%28computer_programming%29). Formal parameters are the holding place for a variable given in the method header. Actual parameters (also called "arguments") are the values those methods take on once the method is called. For example, I could have a method header:

public int add(int x, int y)

And call the method like this:

int sum = add(first,second);

The variables x and y are formal parameters and first and second are actual parameters.

Why is this important? Because in this assignment you will be dealing with list of Objects, and those Objects can be literally anything in all of Java. You have to keep up with what the Object is supposed to represent - especially between formal and actual parameters.

Using Non-standard Classes

Of course, since Reflection is dynamic, you don't have to be restricted to using standard java objects. There is an object in the project called MyClass. Here is a sample interaction using that non-standard class:

```
MyClass obj = new MyClass(3, "Hey!");
ok. I have a new MyClass called obj
String output = obj.toString();
I made a new object. Result was 3 : Hey!
Integer num = obj.getNumber();
I made a new object. Result was 3
output = obj.getMessage();
I changed the value of output to my result, Hey!
obj.print();
3 : Hey!
I performed the operation. My answer is: null
```

Go Code!

The two objects that you need to fill in are ReflectionUtilities and Interpreter. Look for the "TODO" marking. Make sure you understand the rest of the code, though, so that you can use it appropriately in your code. You might want to tackle ReflectionUtilities first, since you will call those methods in Interpreter. There is a JUnit File called Tester.java, and it will test your ReflectionUtilities code for you somewhat. Please feel free to add test cases to it.

Turn it in

Please turn in one zip file per team containing all of your code in d2l.