Introduction to Comp. Sci., Homework 7

Due at 12pm on Monday, April 8

Readings from Liang

Read chapter 11, "Inheritance and Polymorphism."

Optional readings and exercises from HFJ

Read chapter 8 of Head First Java, "Serious Polymorphism." Do the exercises, and check your answers against those provided at the ends of the chapter.

To be turned in: Stack

For this part, you'll build a `Stack` class (in the `hw7` package) that represents a last-in-first-out collection of objects. You may not use any class in `java.util`.

- It should implement the `testing.predefined.StackInterface` interface, which contains the `push`, `pop`, and `empty` methods, described in `testing/predefined/StackInterface.java`.
- It should have a constructor that takes a single `int` argument specifying the maximum capacity of the stack.
- It should override the `toString` method of `java.lang.Object`. This method should return a string of the form `(VAL1,VAL2,VAL3,...,VALN)` where `VAL1`, `VAL2`, etc. are the elements on the stack. If the first element of a stack is the `String "foo"` and the second (top) one is the `Integer 7`, `toString` would return `(foo,7)`.
- It should override the `equals` method of `java.lang.Object`. Two `Stack` objects are equal if and only if the elements on them are the same. Two `Stack` objects with the same elements but different capacities are equal.

For example:

```
Stack stack = new Stack(10);        // create empty stack
System.out.println(stack.empty());  // should print "true"
stack.push(Integer.valueOf(5));     // stack now contains (5).
stack.push(Character.valueOf('x')); // stack now contains (5,x).
System.out.println(stack);          // should print (5,x)
System.out.println(stack.pop());    // should print "x"; stack contains (5)
System.out.println(stack.pop());    // should print "5"; stack now empty
```

We'll talk more about this in class on April 1. I have provided a simple test for the `Stack` class called `Hw7StackTest`, which you can run in the usual way.

To be turned in: Matcher

Use your `Stack` class to match balanced parentheses and brackets. Create a class, `hw7.Matcher`, with a method:

```
public static boolean matchParentheses(String s);
```

It takes a `String` containing `(`, `)`, `[`, and `]` and returns `true` if the parentheses are balanced. For example:

```
Matcher.matchParentheses("()")       // returns true
Matcher.matchParentheses("([])")     // returns true
Matcher.matchParentheses("([])[()]") // returns true
Matcher.matchParentheses("([)]")     // returns false
Matcher.matchParentheses("(")        // returns false
Matcher.matchParentheses("[]]")      // returns false
```

I suggest implementing this by creating a `Stack`, pushing a `(` or `[` onto the stack whenever one of those characters is seen, and popping it off when the corresponding closing character is seen. If the wrong character is at the top of the stack, the parentheses do not match, and the method should return `false`. Note that your `Stack` class can hold `Character` objects, but not `char` values, since `char` is a primitive type.

I have provided a simple test for the Matcher class called `Hw7MatcherTest`, which you can run in the usual way.