CS 158 Lab 3 January 21, 2010
Implementing a Stack as an array

1. Create a new project called Stacks. Add an interface called **Stack**. In it, define methods **push**, **pop**, **peek**, and **isEmpty** as we did in class yesterday, where the type of object that will be on the stack is **Object**, and **push** returns nothing.

2. Create a new class called **ArrayStack** that implements the **Stack** interface. Use a protected array inside that class. The array must grow as needed so that the **push** operation will always work (assuming that there is enough memory available in the computer). Use an initial size of 1000. You should use a protected variable that points to the top of the stack, and in fact, is the subscript of the array that contains the top element. Your constructor should probably initialize it to -1. Make sure that every element that changes the stack also updates that variable. Test and implement each of the required methods. For this lab only, you can assume that the stack will be used properly. That is, no one will ever try to peek or pop an empty stack. So you don't have to test for those situations, and if someone actually does it, the program would generate a subscript out of bounds or null pointer exception (but that's not your problem). But you DO need to handle the situation where the stack grows to be bigger than 1000 (or whatever its current size is). In that case, create a new array with an additional 1000 elements and copy over all of the existing elements. Be sure this is tested thoroughly.

3. Create a static method in your main class called **bracesMatch** that takes a String argument and returns true if the input string contains matching and non-overlapping braces: ( ) [ ] { }. For example, "`[N(ov)(emb)er]1st`" returns true, and "`{Oc[t}ober2]3rd`" returns false. To do this, use one of your stack objects. Go through the input string one character at a time. Push opening characters onto the stack. When a closing character is found, pop the stack to see if the braces match. You can ignore other characters. Since character primitives are not Objects, you will have to "wrap" characters in an object. You can use the Character class.

   ```
   Character ch = new Character('x');
   ```

   places the character 'x' into the object ***ch***, and the Character class provides a method called ***charValue*** that returns the character in the object.