**Reverse Polish (HP) Style Calculator - Part 1**

The purpose of this assignment is to use array as stack to begin constructing a business calculator.

Throughout the remainder of the course, you will complete a series of programming functions that will allow you to develop an emulator for a Hewlett-Packard business calculator that uses the Reverse Polish Notation (RPN) discussed in the Topic Materials.

Reverse Polish Notation is also the basis for a computer programming language for embedded applications called Forth, which you read about in the Topic Materials. As part of this project, you will develop a simplified Forth interpreter.

The heart of a RPN calculator is a data structure called a stack. Review the Topic Materials video tutorials and textbook sections 14.3 and 14.5 and study the examples related to stack prior to completing the programming steps below.

* Implement a stack class named ArrayStack that "has-a" **private** array of double.
* Implement the methods:
  + **public** ArrayClass(int size) - constructor that creates an ArrayClass instance containing an array of the specified size. Remember that in Java, arrays are indexed from 0!
  + **public** push(double item) – standard stack action; throws an exception if the array bounds are exceeded.
  + **public** double pop() – standard stack action; throws an exception if the array bounds are exceeded.
  + **public** boolean isEmpty() – returns true if the stack is empty, and false otherwise.
  + **public** double peek(int n) – returns the value of the item located at the specified position on the stack; throws an exception if the array bounds are exceeded or if a nonexistent element is requested; peek(0) will return the top element of the stack.
  + **public** int count() - returns the number of items currently pushed onto the stack.
* Implement a console application class named TestArrayStack containing a main method that tests each of the above methods of the ArrayStack by creating an instance of ArrayStack and calling each of the methods to test that each functions normally as described and fails with exceptions as described.
* When a failure occurs, print a descriptive error message to the console and stop.
* Keep adding tests and correcting bugs until all tests pass and you have "covered" all aspects of the specification. When this happens, print out "SUCCESS" and stop.

**Note:** The details of the methods specified above may be somewhat different from the discussion in the text and the videos. You will have to think about adapting the ideas in these sources, as is usual in programming. Available examples are only "good approximations" of what you need to do. You need to add the creative energy to make the adaptations.

* Use the debugging features of Eclipse to step though the program to find and correct bugs.
* After thoroughly testing the program, submit the ArrayStack.java and TestArrayStack.java files to the instructor.