**Reverse Polish (HP) Style Calculator - Part 2**

The purpose of this assignment is to incorporate an abstract class and inheritance and add it to the interface of the business calculator created in Topic 4.

* Implement a pure abstract stack class named AbstractStack that has no implementation. It will ***not*** have a private array of double, because that is an implementation detail of ArrayStack that would not be found in other implementations such as LinkedStack. Nor will it have the constructor **public** AbstractClass(int size), or methods **public** double peek(int n) or **public** int count(), since these methods are convenience features that are easily implemented for ArrayStack, but not other implementations of stack.
* Include the method **public** double peek(), and add the method public void clear(), resulting in the following abstract methods.
  + **public** **abstract** void push(double item) – Standard stack action; Throws an exception if the stack is full.
  + **public** **abstract** double pop() – Standard stack action; Throws an exception if the stack is empty.
  + **public abstract** boolean isEmpty() – Returns true if the stack is empty and false otherwise.
  + **public** **abstract** double peek() – 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.
  + **public** **abstract** void clear() – empties the stack if it is not already empty.
* Do not specify a constructor. Depend on the default constructor.
* Modify the existing ArrayStack.java file to include the phrase "extends AbstractStack."
* Add an implementation for the methods void clear() and double peek(), which overloads the existing double peek(int n). Provide a default constructor that creates an array that will hold three elements.
  + **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.
* Modify the existing TestArrayStack.java file to include tests for void clear() and double peek(). Note that it is easiest to do "incremental testing" in which you code a little then test a little.
* Create an interface with additional methods.
* Create a new file called Forth.java. Make this a **public** **interface** file. Add the following methods to the interface.
  + **public** add() – pops two values from the stack, adds them together, and pushes the result back onto the stack. Throws an exception if there are not at least 2 items on the stack.
  + **public** sub() – pops two values from the stack, subtracts the second number popped from the first number popped, and pushes the result back onto the stack. Throws an exception if there are not at least 2 items on the stack.
  + **public** mult() – pops two values from the stack, multiplies them together, and pushes the result back onto the stack. Throws an exception if there are not at least 2 items on the stack.
  + **public** div() – pops two values from the stack, divides the second number popped by the first number popped, and pushes the result back onto the stack. Throws an exception if there are not at least 2 items on the stack.
  + **public** dup() – peeks at the top value on the stack and pushes a copy of that value onto the stack. Throws an exception if the stack is empty or full.
  + **public** twoDup() – peeks at the top two values on the stack and pushes a copy of both values onto the stack (in the same order). Throws an exception if the stack does not have at least 2 items or room for 2 additional items.
* Create another file called ForthStack.java. Make this file **extend** ArrayStack and **implement** Forth. Code concrete implementations of each of the methods in the interface.
* Test by making a copy of the file TestArrayStack.java and name it TestForth. Change the name of the class inside the file. Add tests for add, sub, mult, div, dup, and twoDup.
* After thoroughly testing the program, submit the AbstractStack.java, ArrayStack.java, Forth.java, ForthStack.java, and TestForthStack.java files to the instructor.