CSE 11 **START EARLY!**
Spring 2015
Program Assignment #3 (100 points)

Due: 24 April 2015 at 1159pm (2359, Pacific Standard Time)

PROGRAM #3 : ArrayPlay

READ THE ENTIRE ASSIGNMENT BEFORE STARTING

In lecture, we described several common operations that can be performed on arrays. In this assignment you will implement a class, called `IntArray11`, that defines and implements these operations on integer arrays.. You will then create a program called `ArrayPlay`, that allows you interactively create an `IntArray11` instance, perform operations on that instance and print the results. The goals of this assignment are 1) to become comfortable with defining a class and using it, 2) utilize some methods defined in the java String class, 3) begin to understand how to better handle some user input errors and 4) become more comfortable with reading javadoc-created documentation and using it to guide implementation. This is a more complicated program that #1 or #2, so it will require you to start to develop some of your own debugging skills. It is roughly double the length of your previous assignment.

The `IntArray11` class

You are being provided a skeleton of the IntArray11 class that has been commented using Javadoc-style comments. Your first step should be to generate the documentation using the javadoc command-line tool, to see the details of each constructor and public method. See lecture slides for an example of how to run the javadoc command.

When you Implement IntArray11, **You may NOT**

1. Change the signature of any `public` method
2. Add any new `public` methods, you may (and probably will) add `private` methods
3. Add any `public` instance or class variables
4. Use any other pre-defined classes (e.g., Array, ArrayList, …) that provide similar functionality

In grading your assignment, we will write our test programs to utilize only your implemented `IntArray11` class to test its functionality. We will then test your `ArrayPlay` program against our implementation of `IntArray11` and finally we will test your two classes together.

As you look at the `IntArray11` definition, it should be clear that an instance of the class must internally store an array of `int`s. This instance variable makes up a key component of the state of an `IntArray11` object. The class must also define methods that manipulate an instance's internally-stored array. You should also note, that in the skeleton, `boolean` methods always return `false`. That is clearly incorrect in the full implementation. Those return values in the skeleton are present so that the initial code will compile. For clarity, we will list the constructors and methods defined in IntArray11 with a brief description

Constructors

Constructor and Description
`IntArray11``()` 0-argument constructor.
`IntArray11``(int size)` Store an array of size n.
`IntArray11``(int[] intArray)` Create an array of size n and store a copy of the contents of the input argument

Modifier and Type	Method and Description
`boolean`	**`delete`**`(int index)` Delete and element at index
`int`	**`getElement`**`(int index)` get the Element at index
`int`	**`getNelem`**`()` get the number of elements stored in the array
`boolean`	**`insert`**`(int index, int element)` Insert an element at index in the array
`void`	**`reverse`**`()` reverse the order of the elements in the array
`boolean`	**`reverse`**`(int start, int end)` reverse the order of the elements in the array from start to end index
`boolean`	**`setElement`**`(int index, int element)` set the value of an element in the stored array
`boolean`	**`swap`**`(int index1, int index2)` swap two elements in the array
`java.lang.String`	**`toString`**`()` Pretty Print -- Empty String "[]" else "[e1, e2, ..., en]"

For methods that return `boolean`, they should return `true` if method was successful, `false` if an error would occur. For example, if you attempted to swap two indices and either index is out of bounds, you should return `false`. **Your `intArray11` implementation should never generate a runtime error.** Nor should it print any error messages to the screen (though you will find such error messages useful during debugging)

Some Hints:

1. You may find an internal helper method that copies the contents of one array into another to be useful.
2. If the `insert` method is invoked, and your internal array is full, you should create a new internal array that is large enough to hold all the elements and the newly inserted element.

The ArrayPlay Program

This is a java program and you must therefore define a `main()` method. Its job is to take commands from the user and behave appropriately. The commands that ArrayPlay must understand are

Command	Arguments	Action
new		Create an IntArray11 instance with no elements. This becomes the current IntArray11 instance
new	size	Create an IntArray11 instance with array contents 1,2, …, size. This becomes the current IntArray11 instance
new	e0 e1 e2 … en	Create an IntArray11 instance with array contents e0,e1,e2,..,en. This becomes the current IntArray11 instance (n+1 elements)
print		Print the current IntArray11 using the toString() method
print	index	Print the element at index of current IntArray11 instance
delete	index	Delete the element at index of the current IntArray11 instance
insert	index element	Insert the element at index in the current IntArray11 instance
reverse		Reverse the order of elements in the current IntArray11 instance
reverse	start end	Reverse the order of elements in the current IntArray11 instance from the start to end index
set	index element	Store element at index in the current IntArray11 instance
size		Print the number elements currently stored in the current IntArray11 instance
swap	index1 index2	Swap the elements at index1 and index2 of the current IntArray11 instance
exit		Exit the program

Initialization

When your program begins, there is no current IntArray11 index, any operation, except new or exit should result in a BAD INPUT message to the screen.

Command Loop

The program should print out a command prompt of `"> ",` (please take note of the space that follows the `'>'`) and then wait for user input. If a command is successful, it should print **OK**, otherwise is should print **BAD INPUT**. When testing your program, we will always enter a command followed by zero or more integers. You do not, for example, have to properly handle a bad input string like "swap eleven 5" (the first argument to the command is "eleven" and is not an integer). Inside of your command loop, do NOT create a new scanner instance. That is, create a single scanner instance and then your command loop can use the hasNext() method to determine if input is still available. For example, a code snippet for your loop could be

```
Scanner scnr = new Scanner(System.in);
String prompt = "> ";
System.out.print(prompt)
while (scnr.hasNext())
{
      // … your command loop code is here
}
```

Case insensitivity

Command are insensitive to case. That is "new", "NEW", "New", "neW" are all valid forms of the new command. Be smart and use a String built in method to make this easy.

Whitespace insensitivity

Commands and their arguments can be separated by more than one space. For example, "reverse 8 11" and "reverse 8 11" are both valid input strings.

(Hint: see Hints below for how to split a string with arbitrary amounts of white space)

Definition of "BAD INPUT"

1. A command is given the wrong number of arguments. For example "swap 10", "swap", and "swap 10 11 12" are all bad input strings, because swap expects exactly two arguments.
2. If the form of the command is ok, then BAD INPUT should be displayed if the `IntArray11` method that was invoked returned an indication of error (See the Javadoc-created documentation)
3. exit followed by any other input is NOT bad input. If exit is the command, the program should exit.

Hints:

1. You will likely find it easier to read a complete line of input and then extract the command and any arguments. The `nextLine()` method in the `Scanner` class is likely useful to you. In other words, if you know you have some input, then valid input must be of the form: "<command> [argument1] [argument2] …". If you read an entire line of input, the first word in that input is the command. Based on the actual command, your program will need to process the rest of the command line.
2. We haven't covered regular expressions (and may not this quarter), but to split a String (e.g. input) into an array of Strings separated by arbitrary amounts of white space, you can do the following in your code
   - `String [] args = input.split("\\s+");`
3. You may find it convenient to define two helper functions, one that returns the command typed in by user, the other that returns an `int` array of the arguments.
4. There are several ways to code the main command loop, but `if … else if .. else if … else`, might be very convenient for you. You can also use the `switch` statement
5. Define helper functions when you need them. Think about small things that need to be performed by some/all of the commands. For example, all commands need to read the input line. Some (but not all) commands need to process arguments.
6. Read the Javadoc for valid input parameter ranges. For example, if the current IntArray instance has N elements (valid indices are 0, 1, … (N-1)), a valid insert index is 0,1,…,N. Inserting at index K means that all the elements in locations 0,..,(K-1) are unchanged. In

particular, for an IntArray instance with N elements, an insert at location N, appends to the array. You may have to copy the original array (the internal state of your IntArray instance!) to a newer array with more capacity to make this work.

Example Sessions of Running ArrayPlay

I. new is never typed in by the user

```
[cse11@centos32bit PR3]$ java ArrayPlay
> swap 10 11
BAD INPUT
> print
BAD INPUT
> delete 0
BAD INPUT
> exit
OK
[cse11@centos32bit PR3]$
```

II. An array of size 10 is created, printed, reversed, and then printed again.

```
[cse11@centos32bit PR3]$ java ArrayPlay
> new 10
OK
> print
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
OK
> reverse
OK
> print
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
OK
> set 5 25
OK
> print
[10, 9, 8, 7, 6, 25, 4, 3, 2, 1]
> exit
OK
[cse11@centos32bit PR3]$
```

III. A specific array input, the element at index 5 is removed, the first and last elements are swapped, and then the array is printed.

```
[cse11@centos32bit PR3]$ java ArrayPlay
> new 34 10 14 8 11 17 93 45
OK
> print
[34, 10, 14, 8, 11, 17, 93, 45]
OK
> delete 5
OK
> print
```

```
[34, 10, 14, 8, 11, 93, 45]
OK
> size
7
OK
> swap 0 6
OK
> print
[45, 10, 14, 8, 11, 93, 34]
OK
> exit
OK
[cse11@centos32bit PR3]$
```

IV. Various kinds of bad input. Improper arguments, bad indices, etc.

```
[cse11@centos32bit PR3]$ java ArrayPlay
> new 15
OK
> delete 16
BAD INPUT
> delete -5
BAD INPUT
> swap 10
BAD INPUT
> reverse 7 18
BAD INPUT
> exit
OK
[cse11@centos32bit PR3]$
```

V. Creating several new arrays

```
[cse11@centos32bit PR3]$ java ArrayPlay
> new 10
OK
> print
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
OK
> new 3
OK
> print
[1, 2, 3]
OK
> new
OK
> print
[]
OK
> new 17 29 53
OK
```

```
> print
[17, 29, 53]
OK
> exit
OK
[cse11@centos32bit PR3]$
```

Make Copies of your Program Files as you go along, If you make a big mistake you can go back to the previously working code

Turning in your Program

YOU MUST BE ON THE LAB MACHINES FOR THIS TO WORK. PLEASE VERIFY WELL BEFORE THE DEADLINE THAT YOU CAN TURNIN FILES

You will be using the "bundlePR3" program that will turn in the file

`ArrayPlay.java IntArray11.java`

No other files will be turned in and and it **must be named exactly as above.** BundlePR3 uses the department's standard turnin program underneath.

To turn-in your program, you must be in the directory that has your source code and then you execute the following

```
$ /home/linux/ieng6/cs11w/public/bin/bundlePR3
```

The output of the turnin should be similar to what you saw in your first programming assignment.

You can turn in your program multiple times. The turnin program will ask you if you want to overwrite a previously-turned in project. **ONLY THE LAST TURNIN IS USED!**

Don't forget to turn in your best version of the assignment.

Frequently asked questions

Can I add extra output? No. We attempt to autograde your programs to the extent possible. Having extra output can cause you to lose points.

What if my programs don't compile? Can I get partial credit? No. The bundle program will not allow you to turn in a program that does not compile.

Does ArrayPlay have to use my IntArray11 class. Yes! That is a key part of the assignment.

If the user generates too many arguments for a command is that "BAD INPUT"?. Yes. (See note on the exit command, above)

If the user generates too few arguments for a command is that "BAD INPUT"?. Yes.

Do I have to check for all kinds of crazy inputs? Check for what we've asked for and have your program respond properly. We will test with reasonable inputs in all other cases. In particular, we will test with commands that don't exist, but we won't ever give non-integer arguments to your commands. We will eventually cover exceptions in this course, to get better error handling. That means that if a command is typed in, it will only be of the form "command int0 int1 int2 …" Arguments will never be non-integers.

I don't understand what a deep copy is, can you explain? A deep copy of an instance (the source) creates a new instance of the appropriate class (the destination) it then copies all of the internal state of the source instance to the destination instance. For array objects, you copy each of the elements in the source array to the destination array.

I know how to use the ArrayList class, can I use it in this project? No. I want you to use arrays. Some of what we are doing is re-implementing some of the features of ArrayList.

I know about other classes that easily provide similar functionality, can I use them? No. one of the purposes of this assignment is write and debug some common array methods.

Can my IntArray11 or ArrayPlay class extend an existing class (using the extends keyword)? No.

Can you give is more sample inputs and outputs?. Possibly

I don't understand how to use toString() in ArrayPlay, can you explain? Every Object in java has a `toString()` method defined. When the compiler requires a String version of the object, it will invoke that instance's `toString()` method. If you have a valid instance of your `IntArray11` class (call it `myArray`), then `System.out.println(myArray)` will invoke the `toString()` method on your `myArray` instance and print the result.

Will you grade program style? Yes. In particular, indentation should be proper, variable names should be sensible. We will also look for code clarity, too. Overly long or complex codes are frowned upon. For example, the professor's implementation of ArrayPlay.java without comments (but with whitespace lines for readability) is 102 lines. The amount of code he added (including any private methods) to IntArray11.java was 93 (including comments). Your code may be longer or shorter in both cases. If you find yourself writing many 100s of lines of code, you need to ask for advice.

START EARLY! ASK QUESTIONS!