CS1110 Lab 05. Class Vector Fall 2010

Name ____________________ Netid______________________

Class `java.util.Vector` provides the ability to maintain a growable/shrinkable list of objects, which can be of great utility in cases where you don't know ahead of time how many objects will be in the final list. In this lab, you will gain some experience with class `Vector` and learn just how useful it can be. This material is covered in Sec. 5.3 (pp. 184-188) of the text. After the lab, study that section. Also, the API is your friend. Use it! Clicking this link will open the API spec of class Vector in a new window.

A History Lesson

The developers of Java knew early on that they wanted some kind of growable list, so they created class `Vector` and shipped it out with Java v1.0. Later, however, they wanted to generalize the idea of a list. So they created new classes that provide a more general implementation than `Vector`. Rather than get rid of `Vector` —for "backward compatability" reasons, you can't simply throw out old stuff— for Java v1.2 the developers added new methods to class `Vector` so that it would be consistent with the other, newer, classes. Many of these new methods do the same thing as the old ones. Some of the old ones are "deprecated" (to deprecate means to disapprove of, but often only mildly so). But they will NOT go away, and you can use them. There is also a new class `ArrayList`, which can be used in place of `Vector`.

Caution: Another important history lesson

Several years ago, Java switched from version 1.4 of the language to version 1.5. Java 1.5 and 1.6 make it easier to work with `Vector`s of a particular class of elements, like a `Vector` of elements of class `Character`, which we will be working with here. If your computer has Java 1.4, you have two options:

1. Deinstall the Java 1.4 compiler and install Java 1.5 or 1.6.
2. For this lab, follow the instructions in the comment that appears at the top of file Lab05.java, which you will obtain later from the course web page.

What a Vector contains:

A `Vector v` contains a list of elements, numbered 0, 1, 2, ... Function `v.size()` tells how many elements are in the list. We use the following **non-Java** notation to refer to parts of the list. The notation helps us write things more clearly and succinctly. We refer to the elements in the list as `v[0]`, `v[1]`, ..., `v[v.size()—1]`. We refer to part of the list, say elements `v[h]`, `v[h+1]`, ..., `v[k]`, using `v[h..k]`. We also write `v[h..]` to mean `v[h],v[h+1],...,v[v.size()-1]`.

In Java 1.5 or later, use this assignment statement to create a `Vector` that can contain elements only of class `C` and store its name in `v`:

```
Vector <C> v= new Vector <C>();
```

The appearance of <C> says that the `Vector` may contain only elements of class `C`. In this lab, we will be working with `Vector<Character>`, meaning a `Vector` whose elements are of class `Character`.

`Vector v` has a *capacity*, which is the number of elements for which space in the computer memory has been allocated. This is different from its size, which is the number of elements in it. When an element is to be added to `v` but the size is already equal to the capacity, Java allocates space for more elements —for efficiency reasons, the capacity is usually doubled. The capacity can also be controlled by the programmer.

Here is a list of the old methods, the corresponding new ones, and what they do:

Old method	*New method*	*Purpose*
v.addElement(Object ob)	v.add(Object ob)	append ob to v's list. If v's type is class Vector<C>, ob should be of class C or a subtype of C.
v.insertElementAt(int k, Object ob)	v.add(int k, Object ob)	change v's list to v[0..k-1], ob, v[k..]. If v's type is class Vector<C>, ob should be of class C or a subtype of C.
v.elementAt(int k)	v.get(int k)	= v[k]
v.removeElement(Object ob)	v.remove(Object ob)	remove ob from the list in v (if it is there)
v.removeElementAt(int k)	v.remove(int k)	remove v[k] from v's list, changing it to v[0..k-1], v[k+1..]
v.removeAllElements()	v.clear()	remove all elements from v
v.setElementAt(Object ob, int k)	v.set(int k, Object ob)	replace v[k] by ob

Other useful methods in class `Vector` are:

v.size()	= the number of elements in v's list
v.capacity()	= the number of elements that are currently allocated for v's list —this can be different from the number of elements that are actually IN v's list!
v.indexOf(Object ob)	= i, where v[i] is the first occurrence of ob in the list, or -1 if ob is not in v. (Occurrence is checked with method **`equals`**, not with ==.)
v.lastIndexOf(Object ob)	= i, where v[i] is the last occurrence of ob in the list , or -1 if ob is not in v.
v.toString()	= a comma-separated list of the elements in v, enclosed in brackets

Task 1. Experimenting with Vector

Download file Lab05.java from the course website or from here. This program will help you understand exactly what is happening when you call various methods of a `Vector`.

Look over the code that we have provided. We have defined a `Vector<Character> v`, which you will use throughout this lab. You can access it from the Interactions pane of DrJava. We have also defined two constructors, which will illustrate different qualities of `Vector`. Read the specifications so you understand what each one does (according to the Vector API, the capacity increment is the amount by which the capacity of the vector is automatically incremented when its size becomes greater than its capacity). Don't worry about the stubbed-in methods yet —the ones you have to write; you'll get to them later.

Compile class `Lab05` and type this into the Interactions pane.

```
Lab05 lab= new Lab05();
```

A window should appear at the top of your screen containing a drawing of numbered boxes. This drawing represents `Vector<Character>` object `v` in class `Lab05`. Note that there are 10 empty boxes, numbered 0-9. The numbers are called *indices* or *indexes*. You use them to refer to the objects in the boxes.

Method **`add`** has a parameter of class `Object`. However, since `Vector v` was created using `new Vector<Character>()`, only objects of class `Character` can be added to `v`. If `v` had been created using `new Vector<Object>()`, you would have been able to add ANY Java object to `v`. But you cannot add primitive-type values such as **`int`** or **`char`** values. This is one situation where wrapper classes are useful!

To avoid the problem of drawing arbitrary objects in the numbered boxes, only instances of class `Character` are drawn. Any other object will be drawn as a red question mark. (What other objects could there be? We'll see at least one example in this lab.)

Resize your DrJava window so that it doesn't block the drawing. Now try the following in the Interactions pane:

lab.v.add(new Character('A'));

A `Character` object that wraps `'A'` has been added to `Vector v`, and you can see it in box 0.

Note: If you are using Java 1.5 or later, you can write simply "`lab.v.add('A')`"; `'A'` will automatically be wrapped in an object of class `Character` for you.

Now type:

lab.v.remove(new Character('A'));

(Do not use "`lab.v.remove('A')`" regardless of your version of Java, because the 'A' is, probably unexpectedly, interpreted as an int, and probably not the int you want!) And it's gone from `v`. Note that you passed in two different objects to methods `add` and `remove`. `Vector` uses method `equals` of each element `v[i]` of `Vector v`, and for elements of class `Character`, `v[i].equals(ob)` yields **`true`** if the character in `v[i]` is the same as the character in `ob`.

Type the following command to put some more objects in `Vector v`:

lab.initializeV();

Look over the drawing. Now enter the commands on the left in the table below into the Interactions pane. On the right, write down what the command returned (if anything) and what happened to the Vector drawing. If you don't understand WHY certain commands do certain things, ask!

Tip 1: Use the up arrow key to get your previous command instead of repeatedly typing in "new Character...".
Tip 2: Make sure you're watching the Vector drawing when you hit Enter to execute your commands in the Interactions pane! It will be much easier to see what happened.
Tip 3: Make sure you leave off the semicolon when you make a function call; otherwise, you will not see not what the function returned.
Tip 4: If you mis-type something, you can start over with the call "lab.initializeV();" .

lab.v.add(new Character('B'));	
lab.v.remove(new Character('3'));	
lab.v.remove(new Character('7'));	
lab.v.indexOf(new Character('1'))	
lab.v.indexOf(new Character('2'))	
lab.v.get(5)	
lab.v.get(12)	
lab.v.indexOf(lab.v.get(2))	What is this call doing?
lab.v.indexOf(lab.v.get(8))	Why doesn't this return 8?
lab.v.firstElement()	
lab.v.set(1, new Character('O'));	
lab.v.capacity()	
lab.v.size()	
lab.v.toString()	
lab.v.trimToSize();	
lab.v.setSize(12);	What is in the cells that have red question marks? Is the new capacity also 12?

Task 2: Writing methods to manipulate a Vector

We have written four method stubs for you to implement; implement them. For the last, you do not need a loop or recursion.

As usual, we suggest you write and test the methods one at a time, thoroughly testing one before moving on to the next. We created the window that visualizes `Vector v` precisely to give you an easy way to test. For example, after writing the swap method, try `lab.swap(0,1)` and see what happens in the window.

public void swap(int first, int second)	Swap the objects at v[first] and v[second]
public boolean moreThanOne(Object obj)	= "there is more than one occurrence of obj in v" *Hint:* Does looking at the first and last occurrences of obj in v help? *Hint:* Did you test the case where there are zero occurrences of obj? (Technical note: here, "occurrence" is in the sense of method `equals`, not `==`. That is, if we executed `lab.v.add(0, new Character 'A')` and `lab.v.add(1, new Character 'A')`, then `lab.moreThanOne(new Character 'A')` should return **true**: `v.get(0).equals(v.get(1))` is **true** even though `lab.v.get(0) == lab.v.get(1)` is **false**.)
public boolean hasExtraSpace()	= "there is space allotted to v that is not being used"
public String toString()	= a string that contains the characters of v, in the order in which they occur in v. *Note*: Do NOT use loops or recursion in writing toString. *Hint*: Use the existing toString of class Vector, but the resultant String has square brackets and commas in it, which you then need to remove using methods of class String. Remove the brackets first. Because of some syntax issues involving "regular expressions" (don't worry if you don't know what those are), don't use function replaceAll() on Strings containing brackets.

Show your work to your TA or a consultant.