CS 1130, LAB 3: VECTOR EXERCISES

Name: _____ **Net-ID**: _____

There is an online version of these instructions at

http://www.cs.cornell.edu/courses/cs1130

You may wish to use that version of the instructions.

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 do not know ahead of time how many objects will be in the final list. The purpose of this lab is to gain some experience with class `Vector` and learn just how useful it can be.

Requirements For This Lab. The very first thing that you should do in this lab is to download the file `VisualVector.java` from the course web page:

http://www.cs.cornell.edu/courses/cs1130/2012fa/labs/lab3/VisualVector.java

You will note that this is a subclass of `JFrame`. That is because, instead of a JUnit test, we are going to use a GUI program to help you "test" the lab. The GUI will give you a visual representation of the changes that you make to the class.

Within `VisualVector.java` you will notice several function (and procedure) stubs. They stand out because they have the comments

```
//-------- IMPLEMENT ME! --------//
```

in their method bodies. As with the last lab, you will be filling in code here.

For this lab you will show your instructor the contents of `VisualVector.java` and what you have written on this sheet; there is no JUnit test this time. As always, you should try to finish the lab during your section. However, if you do not finish during section, you have **until the beginning of lab next week to finish it**. You should always do your best to finish during lab hours; remember that labs are graded on effort, not correctness.

---

1. Understanding the Vector Class

The material in this lab is covered in Section 5.3 (pp. 184-188) of the text. After the lab, you should study that section.

Basic Terminology. A `Vector v` contains a list of elements, numbered 0, 1, 2, ... The 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, such as elements `v[h]`, `v[h+1]`, ..., `v[k]` as `v[h..k]`. We also write `v[h..]` to mean elements `v[h]`, `v[h+1]`, ..., `v[v[v.size()-1]`.

As Vector is a generic class, we use the following assignment statement to create a `Vector` that can contain only elements of the class `C`, and store the name of this `Vector` 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`.

Finally, `Vector` whose name is in `v` has a ¡i¿capacity¡/i¿, 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, and it os sometimes possible to save memory or make a program a little faster in this way (though you should never worry about this).

The Vector API. You can find the API for `Vector` following the Java API link from the website for CS 1110. Remember that `Vector` is in the package `java.util` and not `java.lang`. Alternatiely, you can simply copy and paste this URL into your browser:

http://docs.oracle.com/javase/6/docs/api/java/util/Vector.html

We highly recommend that you do not Google for the `Vector` API. For historical reasons (see below), the top Google link is to the Java 1.4.2 version of `Vector`. There have been a lot of changes to this class over the years and it is important that you only look at the API for the current (for this course, Java 1.6) version of `Vector`.

Remember from class that `Vector` is a generic class. That means that its type is specified by what it contains, which is put inside "angled brackets" (e.g. <, >). On the API page for `Vector`, you will notice that it puts the letter E in the angled brackets and then uses E as a type through the web page. The way to read this page is to place the letter E with whatever you have put in your angled brackets (which for this lab is the class Character).

In this class, we will use a small subset of the API. For your convenience, here are the important methods:

Method	Description
`v.add(E ob)`	Append `ob` to the list `v`. If the type of `v` is `Vector<E>`, then `ob` should be of class `E` or a subtype of `E`.
`v.add(int k, E ob)`	Change the list `v` to `v[0..k-1]`, `ob`, `v[k..]`. If the type of `v` is `Vector<E>`, then `ob` should be of class `E` or a subtype of `E`.
`v.get(int k)`	Yields: `v[k]`
`v.remove(E ob)`	Remove ob from the list in v (if it is there)
`v.remove(int k)`	Remove `v[k]` from the list `v`, changing it to `v[0..k-1]`, `v[k+1..]`
`v.clear()`	Remove all elements from `v`
`set(int k, E ob)`	Replace `v[k]` by `ob`
`v.size()`	Yields: the number of elements in the list `v`
`v.capacity()`	Yields: the number of elements that are currently allocated for the list `v`. This can be different from the number of elements that are actually **in** the list `v`.
`v.indexOf(E ob)`	Yields: i, where `v[i]` is the first occurrence of `ob` in the list; -1 if not in the list.
`v.lastIndexOf(E ob)`	Yields: i, where `v[i]` is the last occurrence of `ob` in the list; -1 if not in the list.
`v.toString()`	Yields: a comma-separated list of the elements in `v`, enclosed in brackets

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 cannot simply throw out old stuff), the developers of Java v1.2 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. This is why you will see some methods in the documentation for the Vector class that seem redundant – they are.

Another major change happened in Java 1.5. This is the first version of Java to support generic classes. Before then, Vector to only hold objects whose class is `Object`. Since `Object` is the "superest" class of them all, this was not a problem. However, it made programming with `Vector` much, much harder for beginners. That is why it is important that you do not use any API for `Vector` that is older than Java 1.5.

---

2. Experimenting with Vector

Once you have downloaded `VisualVector.java`, you should first play with it a bit. We have already provided you with quite a bit of code there, and this code is intended to make the lab a bit more interesting. In particular, this is some of the first GUI code that we have seen in this course.

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).

Do not worry about the stubbed-in methods yet (the ones you have to write); you will get to them later.

Working with `VisualVector`. Compile the class `VisualVector` and type this into the Interactions pane:

```
VisualVector lab = new VisualVector();
```

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 `VisualVector`. Note that there are 10 empty boxes, numbered 0-9. The numbers are called the *indices* or *indexes*. You use them to refer to the objects in the boxes.

Resize your DrJava window so it does not block the representation. In the Interactions pane, type:

```
> 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.

Now type the following:

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

It is now gone from `v`. Note that you passed in two different objects (every time you use `new` it makes a new object) to the methods `add` and `remove`. `Vector` uses the method `equals` of each element `v[i]` of `Vector v` to find a match; 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`.

Note: As we said in class, the latest version of Java supports "boxing" which allows us to write

```
> lab.v.add('A');
```

to do the same thing as the `add` statement above. However, you should avoid doing this, because it is dangerous. In particular,

```
> lab.v.remove('A');
```

is very bad as Java will actually try to cast 'A' to an int (meaning an index to remove from) before it tries to box it in a `Character` object.

`VisualVector` **Exercises.** Type the following command to put some more objects in `Vector v`:

```
> lab.initializeV();
```

Look over the Vector representation in the JFrame. Note that an object can appear many times in the same list ('3' and '2' both appear twice). In the interactions pane, try the commands on the left in the table below. On the right, write down what the command returned (if anything) and what happened to the `Vector` representation. If you do not 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 ae watching the `Vector` representation 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

Command	**Result/Explanation**
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('B'))	
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 does this not return 8?

Command	Result/Explanation
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?

---

3. Writing Methods to Manipulate a Vector

We have written four method stubs for you to implement; implement them. As usual, we suggest you write and test the methods one at a time, thoroughly testing one before moving on to the next. However, you do not need to make a `VisualVectorTester` JUnit class this time. You can simply use the visual representation in the JFrame to test your code in an ad-hoc fashion. For example, after writing the swap method, try `lab.swap(0,1)` and see what happens in the JFrame window.

Method	Description (JavaDoc Specification)
`swap(int first, int second)`	Swap the objects at `v[first]` and `v[second]`
`moreThanOne(Character obj)`	Yields: "there is more than one occurrence of `obj` in `v`" **Hint:** Does looking at the first and last occurrences of `obj` in **v** help?
`hasExtraSpace()`	Yields: "there is space allotted to **v** that is not being used"
`toString()`	Yields: a string that has contains the characters of v, in the order in which they occur in v. Precondition: v may not contain commas or spaces (see the hint below).

Important: **Do not use loops or recursion** (if you know what they are) in this lab. Everything can be done using the methods of either `Vector` or `String`. Here is a hint for the last method: Use the existing toString of class Vector, but remove the square brackets, commas, and spaces using methods of class `String`. Furthermore, remove the brackets first; because of some syntax issues involving "regular expressions", you should not use the function `replaceAll()` on Strings containing brackets. Furthermore, this is the reason for the precondition on `toString()`.

When you are done, show your work to your TA or a consultant.