Vector Class

Java provides a few classes that allow us to manage collections of objects. Today, we'll discuss the Vector class, which stores a group of objects.

One of the "limitations" of an array in C is that it can only store one type of object. In Java, a Vector can store different types of objects. (Actually so can an array…) Secondly, arrays in C are of a fixed size. (Actually you can also declare dynamically allocated arrays in C.)

The Vector class in Java essentially handles both of these issues very well. A Vector object can arbitrarily grow and shrink as needed without the user worrying about any of the details. Furthermore, a Vector can store different types of objects.

Well, this is a bit of a stretch actually. The manner in which Java gets away with having Vectors that can store different types of objects is through its "tree" of inheritance. All non-primitives are objects, and some objects inherit from others. For example, if I wrote a Car class, I could write a SportsCar class that inherited all the methods and variables of the Car class, but then maybe had some extra variables and methods that were specific to the SportsCar. Furthermore, then I could maybe have a Corvette class that inherited from SportsCar. It would retain all the attributes of the SportsCar class, but then have some of its extra attributes. In this example, we have the following:

SportsCar inherits from Car, so a SportsCar object IS-A Car object.

Corvette inherits from SportsCar, so a Corvette object IS-A SportsCar object AND a Car object.

Java also provides a very basic class called Object. Every non-primitive class automatically inherits from Object. Thus, in a way, all non-primitives are valid "Object" objects. Thus, if we create an array of Object or a Vector of Object, then that collection can store ANY sort of non-primitive, ranging from a Time object, to a String object, to a Corvette object.

In the old version of Java, the way to create an empty Vector was as follows:

Vector v = new Vector();

This looks like a regular default constructor.

In the new version of Java, generics are used. This essentially means that some classes can be based upon a "generic" class, where you fill in the class you desire. For a Vector object, you are allowed to declare a Vector of any type of class. Here is how we can declare a Vector of class Object:

Vector<Object> v = new Vector<Object>();

Thus, added to the name of the class is <T>, where T is replaced with the type of object you want the vector to store.

If you wanted to create a Vector of String objects, you'd do the following:

Vector<String> v = new Vector<String>();

(Some) Methods in the Vector Class

```
// Appends this vector with a new element with value v.
void addElement(T v)

// Inserts value v into this Vector such that v has index i.
// Any preexisting elements with indices i or greater are shifted
// backwards by one position.
void insertElementAt(T v, int i)

// If i is a valid index in this Vector, then the ith element is set
// to v. Otherwise, an ArrayIndexOutOfBoundsException
// is thrown.
void setElementAt(T v, int i)

// Returns the number of elements in this Vector.
int size();

// If i is a valid index in this Vector, it returns the ith element;
// otherwise an ArrayIndexOutOfBoundsException is thrown.
T elementAt(int i)

// Returns whether this Vector has an element with value v.
boolean contains(Object v)
```

In essence, a Vector is similar to an array, except that there is an extra level of data abstraction going on. Rather than directly accessing the array, the user indirectly interacts with the array through the public methods provided in the Vector class.