DYNAMIC ARRAYS

In each of the above examples, an arbitrary limit was set on the number of items -- 100 ints, 10 *Players*, 100 *Shapes*. Since the size of an array is fixed, a given array can only hold a certain maximum number of items. In many cases, such an arbitrary limit is undesirable. Why should a program work for 100 data values, but not for 101? The obvious alternative of making an array that's so big that it will work in any practical case is not usually a good solution to the problem. It means that in most cases, a lot of computer memory will be wasted on unused space in the array. That memory might be better used for something else. And what if someone is using a computer that could handle as many data values as the user actually wants to process, but doesn't have enough memory to accommodate all the extra space that you've allocated for your huge array?

Clearly, it would be nice if we could increase the size of an array at will. This is not possible, but what **is** possible is almost as good. Remember that an array variable does not actually hold an array. It just holds a reference to an array object. We can't make the array bigger, but we can make a new, bigger array object and change the value of the array variable so that it refers to the bigger array. Of course, we also have to copy the contents of the old array into the new array. The array variable then refers to an array object that contains all the data of the old array, with room for additional data. The old array will be garbage collected, since it is no longer in use.

Let's look back at the game example, in which `playerList` is an array of type `Player[]` and `playerCt` is the number of spaces that have been used in the array. Suppose that we don't want to put a pre-set limit on the number of players. If a new player joins the game and the current array is full, we just make a new, bigger one. The same variable, `playerList`, will refer to the new array. Note that after this is done,`playerList[0]` will refer to a different memory location, but the value stored in `playerList[0]` will still be the same as it was before. Here is some code that will do this:

```
// Add a new player, even if the current array is full.

if (playerCt == playerList.length) {
        // Array is full.  Make a new, bigger array,
        // copy the contents of the old array into it,
        // and set playerList to refer to the new array.
   int newSize = 2 * playerList.length;  // Size of new
array.
   Player[] temp = new Player[newSize];  // The new array.
   System.arraycopy(playerList, 0, temp, 0,
playerList.length);
   playerList = temp;  // Set playerList to refer to new
array.
}

// At this point, we KNOW there is room in the array.

playerList[playerCt] = newPlayer; // Add the new player...
playerCt++;                       //    ...and count it.
```

If we are going to be doing things like this regularly, it would be nice to define a reusable class to handle the details. An array-like object that changes size to accommodate the amount of data that it actually contains is called a dynamic array. A dynamic array supports the same operations as an array: putting a value at a given position and getting the value that is stored at a given position. But there is no upper limit on the positions that can be used (except those imposed by the size of the computer's memory). In a dynamic array class, the `put` and `get` operations must be implemented as instance methods. Here, for example, is a class that implements a dynamic array of ints:

```
/**
 *  An object of type DynamicArrayOfInt acts like an array
of int
 *  of unlimited size.  The notation A.get(i) must be used
instead
```

```
 *  of A[i], and A.set(i,v) must be used instead of A[i] =
v.
 */
public class DynamicArrayOfInt {

   private int[] data;  // An array to hold the data.

   /**
    * Constructor creates an array with an initial size of
1,
    * but the array size will be increased whenever a
reference
    * is made to an array position that does not yet
exist.
    */
   public DynamicArrayOfInt() {
      data = new int[1];
   }

   /**
    *  Get the value from the specified position in the
array.
    *  Since all array elements are initialized to zero,
when the
    *  specified position lies outside the actual physical
size
    *  of the data array, a value of 0 is returned.  Note
that
    *  a negative value of position will still produce an
    *  ArrayIndexOutOfBoundsException.
    */
   public int get(int position) {
      if (position >= data.length)
         return 0;
      else
         return data[position];
   }

   /**
    *  Store the value in the specified position in the
array.
    *  The data array will increase in size to include
this
    *  position, if necessary.
    */
   public void put(int position, int value) {
      if (position >= data.length) {
             // The specified position is outside the
actual size of
             // the data array.  Double the size, or if
that still does
```

```
                // not include the specified position, set
the new size
                // to 2*position.
            int newSize = 2 * data.length;
            if (position >= newSize)
               newSize = 2 * position;
            int[] newData = new int[newSize];
            System.arraycopy(data, 0, newData, 0,
data.length);
            data = newData;
                // The following line is for demonstration
purposes only !!
            System.out.println("Size of dynamic array
increased to " + newSize);
         }
         data[position] = value;
      }

} // end class DynamicArrayOfInt
```

The data in a *DynamicArrayOfInt* object is actually stored in a regular array, but that array is discarded and replaced by a bigger array whenever necessary. If `numbers` is a variable of type*DynamicArrayOfInt*, then the command `numbers.put(pos,val)` stores the value `val` at position number `pos` in the dynamic array. The function `numbers.get(pos)` returns the value stored at position number `pos`.

The first example in this section used an array to store positive integers input by the user. We can rewrite that example to use a *DynamicArrayOfInt*. A reference to `numbers[i]` is replaced by`numbers.get(i)`. The statement "`numbers[numCount] = num`;" is replaced by "`numbers.put(numCount,num);`". Here's the program:

```
public class ReverseWithDynamicArray {

   public static void main(String[] args) {

      DynamicArrayOfInt numbers;  // To hold the input
numbers.
      int numCount;  // The number of numbers stored in
the array.
```

```
      int num;    // One of the numbers input by the user.

      numbers = new DynamicArrayOfInt();
      numCount = 0;

      TextIO.putln("Enter some positive integers; Enter 0 
to end");
      while (true) {  // Get numbers and put them in the 
dynamic array.
         TextIO.put("? ");
         num = TextIO.getlnInt();
         if (num <= 0)
            break;
         numbers.put(numCount, num);  // Store num in the 
dynamic array.
         numCount++;
      }

      TextIO.putln("\nYour numbers in reverse order 
are:\n");

      for (int i = numCount - 1; i >= 0; i--) {
          TextIO.putln( numbers.get(i) );  // Print the i-
th number.
      }

   } // end main();

}  // end class ReverseWithDynamicArray
```

Source : http://math.hws.edu/javanotes/c7/s3.html