Lab 4: Dynamic Array

Due 11:59 pm Mon, Mar 3, 2008

1 Assignment

Implement the Array<Value> and Date2 classes specified below with all public methods **exactly** as printed here. You may add additional public methods if desired. You may add any private methods that you wish and may extend the documentation as needed to explain the behavior and assumption of the classes. You may use any code from lecture, previous lab solutions, or the textbook, but are responsible for ensuring that that code is correct, well-designed, and fully documented. Your Array class may only use Java arrays (e.g., Value[ ]) and integers (i.e., int) in the implementation. It may not invoke any methods on the java.util.Arrays class (if you don't know what that means, you're probably OK).

To submit your solution, use the command:

turnin –c 136 Array.java Date2.java

For this lab, your solution should be in a set of files named as above. You may submit additional classes that you used for testing and implementation (although do not submit TestArray.java if you did not modify it). Do not submit files ending in ".class" or "~".

You can download the outline code below for Array and Date2 from the course web page.

```
import structure5.Assert;

/** Dynamic array (similar to java.util.ArrayList) */
public class Array<Value> {
   /** Creates an empty array */
   public Array();

   /**  Resizes the array to contain N elements. Newly exposed elements
      are null. */
   public void setSize(int N);

   /** Number of elements stored in the Array. */
   public int size();

   /** Set element i to value v */
   public void set(int i, Value v);

   /** Invokes toString on every element and then returns the string
      "{ element1 , element2 , ... }"*/
   public String toString();

   /** Return the element at i */
   public Value get(int i);

   /** Adds v to the end of the array.*/
   public void add(Value v);

   /** Inserts the specified element at the specified position, moving all
      other elements over. */
   public void add(int i, Value v);

   /** Removes all elements */
   public void clear();

   /** Returns true iff there are no elements in the array. */
   public boolean isEmpty();

   /** Removes the element at position i and returns it. */
   public Value remove(int i);

   /** Returns a new Java array containing exactly the elements in this array. */
   public Value[] toArray();

   /** Makes the underlying storage of this Array take no more
      than size() elements */
   public void trimToSize() {

   /** Sorts the elements of this array from least to greatest
      according to the comparator.*/
   public void sort(Comparator<Value> comparator);
}
```

```
import structure5.Assert;
import java.util.Random;

/** Simple Date representation for lab 4. */
public class Date2 {

   /** Returns the number of days in month m of year y, where
      1 = January and 2000 = year 2000 */
   static public int numDays(int m, int y);

   /**
      Constructs a new Date.
      @pre 0 < m <= 12; y >= 1900; 0 < d <= num days in that month
    */
   public Date2(int d, int m, int y);

   /** Returns the day of the month, where 1 = the first day */
   public int getDay();

   /** Returns the month, where 1 = January */
   public int getMonth();

   /** Returns the year, where 2000 = the year 2000 */
   public int getYear();

   /**
    Returns the day of the week corresponding to this date,
    where 0 = Saturday
    */
   public int getDayOfWeek();

   /** Converts a day of the week, where 0 = Sat, to the equivalent string. */
   static public String dayOfWeekToString(int day);

   /** Produces a random date between 1900 and 2099 (no guarantees
    * about the distribution of dates, however). For a given run of the program,
    * the random number generator is always initialized with the same seed, so
    * generating a sequence of random dates always produces the
    * same sequence.
    */
   static public Date2 random();

   /** Generates a string representation with the month spelled out. */
   public String toString();
}
```

2 Evaluation

Correctness	**10**
Readability	**10**
Design	**10**
Efficiency	**10**
Total	**40**
If sort is not merge sort	+2 extra credit
If sort is expected O(n log n) time	+2 extra credit

3 Advice

I've provided TestArray.java, which contains some tests that you can use to aid in debugging. Your code may pass all of these tests and still be incorrect, however. I recommend extending my tests with your own, although this is not required. Use lots of assertions in your code to help detect problems. Comment out any tests that you don't want to run while debugging.

Implement the Array methods in the order that I specified them so that you can print the contents of your array when debugging.

Depending on how you choose to implement your sort, calling trimToSize from the beginning may make it easier to implement. Make sure that you import java.util.Comparator at the top of Array.java.

My complete solution for Array.java was about 230 lines of code, including assertions, comments, and whitespace. My Array.sort method was 50 lines (I chose to use a merge sort) and used two helper methods.