Homework 1 (due by 4:00 pm on Oct 31)

Objectives:

- Review basic concepts and limitations of Arrays
- Reinforce the concept of object-oriented programming

In this homework, you are to implement a Java class

public class MyArray

You are not allowed to use Java Collections framework. Instead, you must use String[] to store and manipulate words in your class. Similar to Java's ArrayList class, 0 can be passed as initial capacity.

Step-by-step guide:

- Think carefully to find out what methods you need to implement.
  - You are more than welcome to use private helper methods. In fact, I encourage you to do so.
- Write your code for the whole MyArray class on paper.
  - Remember you need to submit this paper in class.
  - Give enough space per line as you write.
  - *Try to finish this within 15 minutes. We will not deduct any points because of mistakes made at this point*.
- Implement the class on Eclipse or any other IDEs you prefer.
  - You also need to submit the source file on Blackboard.
- Test your code extensively!
  - Use the provided text file to test your program. When you run the program with the given childrensbible.txt file, you should see the output similar to the following.
  - A word is a sequence of letters [a..zA..Z] that does not include digits [0..9] and the underscore character.

```
god is found
Number of words in the file is : 9398
Capacity of words array is: 16384
the children s bible selections from the old and new
testaments translated and …..
Number of words w/o duplicates is : 1425
the children s bible selections from old and new
testaments translated arranged …….
```

- Update your code on paper with comments
    - Mostly, focus on the segments that are not correct. Use a pen of a different color.

All of the necessary methods that you are required to implement and additional information can be found from the following MainDriver class.

```
import java.util.*;
import java.io.*;

public class MainDriver {
     public static void main(String[] args) {

          // creates MyArray object with initial capacity
          MyArray words = new MyArray(0);
          Scanner scanner = null;

          try {
               scanner = new Scanner(new
File("childrensbible.txt"));
               while(scanner.hasNextLine()) {
                    String line = scanner.nextLine();
                    String[] wordsFromText = line.split("\\W");

                   /*
                    * when inserting, duplicates are allowed
                    * add method takes care of validating words
                    * array doubles its size when necessary
                    */
                    for(String word:wordsFromText)
                         words.add(word.toLowerCase());
               }
          } catch(FileNotFoundException e) {
               System.err.println("Cannot find the file");
          } finally {
               if(scanner!= null) scanner.close();
          }

          // find a word in the words array
          if(words.search("god"))
               System.out.println("god is found");
```

```
           else
                System.out.println("not found");

           // print current number of words
           System.out.println("Number of words in the file is :
"+words.size());

        // print capacity, or the current length, of the array
        // capacity will be increased as necessary based on
doubling-up resizing policy
        System.out.println("Capacity of words array is:
"+words.getCapacity());

        // print words in the words array
       words.display();

        /*
         *  remove all of the duplicates in the words array
         *  Think carefully about how you would perform!
         *  1. You are not allowed to use Java Collections
Framework
         *  2. You are not allowed to use any other data
structures to save your memory
         *  3. You are not allowed to use any sorting algorithms
         */
         words.removeDups();

         System.out.println("Number of words w/o duplicates is :
"+words.size());
         words.display();
     }
}
```

Deliverables:

- A few sheets of paper that have your initial code as well as your comments (Submit this in class that is on the due date.)
    - **In your comments, clearly write worst-case time complexity using Big-O notation for each of the public methods.**
- Your source code file. (Submit this on Blackboard by 4 pm on the due date.)

Grading:

Your homework will be graded first by compiling and testing it. After that, we will read your code to determine appropriate methods/classes are used. In addition, we will judge the overall style and modularity of your code.

Points will be deducted for poor design decisions, uncommented and unreadable code. Your comments on your paper should be able to demonstrate your proper understanding of the code.

- Working code: 60 points
- Coding style (refer to the guideline): 20 points
- Your paper code's comments: 20 points

Late submission will NOT be allowed and, if you have multiple versions of your code file, make sure you do **submit the correct version. Only the version that is submitted before the due will be graded.**

Coding style guideline:

Coding style is very important and you would get deduction based on your coding style.

Here is a quick style guide for you all to reference. This is not a comprehensive list but just a starting point.

For the first or two of assignments, I will be very generous with style points, since you all are still catching up.

One major thing I want to emphasize is that there are many aspects of style where multiple options are all considered "good" style. I don't differentiate between these as long as you are consistent. You won't lose points if you don't use the style that I prefer, but you will lose points if you mix and match.

Curly braces. Ending curly braces, }, should always be on a new line that has nothing else on it except for whitespace (with the exception of "cuddled elses", which I'll talk about in a second). Opening curly braces, {, can be on the same line as its corresponding statement or on a new line -- either way is fine with me, as long as you are consistent. It's also acceptable to omit (or not omit) curly braces for one line statements -- again, either way is fine with me as long as you are consistent. For example, either of the following bits of code is OK:

```
for(int i=0; i<n; i++) count++;
for(int i=0; i<n; i++){
   count++;
}
for(int i=0; i<n; i++)
{
   count++;
}
```

However, the following bit of code is _not_ OK:

```
for(int i=0; i<n; i++){ count++; }
```

If-else statements. If-else statements have a few quirky bits of style. For example, there is one exception to the "} on a new line" rule: cuddled elses. This style is OK (in fact, I think it can make certain bits of code look much nicer); here is an example:

```
if(x == 0) {
   doSomething();
   andSomethingElse();
} else if(x == 1) {
   doSomethingDifferent();
   doAnotherThing();
} else {
   doSomeDefaultThing();
}
```

Another note about if-else statements -- don't have unnecessarily bloated if-else statements. For example, the following is poor style:

```
boolean withinRange1(int x) {
   if(x < MAX_VALUE && x > MIN_VALUE) return true;
   else return false;
}
```

Instead, you should write:

```
boolean withinRange2(int x) {
   return x < MAX_VALUE && x > MIN_VALUE;
}
```

Keep in mind that you can also negate boolean statements with the ! operator; if the return statements in withinRange1 were switched, for withinRange2 you could write return !(x < MAX_VALUE && x > MIN_VALUE).

Indentation. People have very passionate opinions about tabs vs spaces. I don't care, as long as you're consistent. Do not mix tabs and spaces; if you are using spaces, make sure you use a consistent amount of spaces per tab length (I prefer 4, but won't take off points if you use another size). You should also indent appropriately, like inside loops, methods, etc...

For example, this is _not_ OK (with or without curly braces):

```
for(int i=0; i<n; i++)
{
count++;
}
```

This is the correct way (again, with or without curly braces):

```
for(int i=0; i<n; i++)
{
   count++;
}
```

Note: Some IDEs, like Eclipse, have a nifty auto-indent feature. If I recall correctly, Eclipse's is ctrl+i when your code is selected.

Variable and method names. Java, by convention, uses "camel caps" for its variable and method names -- like withinRange instead of within_range (the latter naming convention may be familiar to some of you from other languages, like C). Additionally, make sure you use descriptive -- but not overly-lengthy -- variable names. For example, "count" is a good name for an int used as a counter; "num" is not.

Line length. Do not have lines longer than 80 characters. I'm not exceptionally strict about this (eg, if you have one 83-character line, I won't take off points), but it's very easy for me to see where this happens.

Method length and code reuse. I don't have a hard-and-fast rule for method length. If it seems exceptionally long (say, longer than 50 lines), then consider how to rewrite it and break it up into helper methods. Similarly, if you have identical (or almost identical) bits of code used in more than one place -- like a binary search, or inserting into some data structure -- put it in a helper method.

Commenting. Commenting is messy business sometimes; generally speaking, you should comment any code that is non-obvious. You don't need to say, for example:

```
// sums the numbers from 1 to 10
for(int i=1; i<=10; i++) sum += i;
```

Any more complicated or sophisticated code, or larger pieces of code that do some bigger overall procedure (e.g. binary search), should be preceded with a comment. You should also include a comment at the beginning of each method saying what it does, although for small, obvious methods (like the withinRange method above). Commenting is highly recommended as a good style.

I hope this is enough to get you all started! This is just a guideline.

There are many more nuances to style, but these are the major things I look for. Again, if you have questions, you can come see me in office hours.