Section 1 - Stack Data Structures

Introduction to Week Nine

We have used ***arrays*** outside of classes used to ***sort*** numbers. We even replaced numbers with a **Student** class and watched arrays ***sort*** those objects. Finally, we saw an ***array*** appear inside a class where it was used to store the frequency of a given letter as it occurred in a **String**.

An ***array*** is an example of something computer scientists call a ***data structure***, and ***sorting*** is an example of something those same practitioners call an ***algorithm***.

This lesson, we will introduce another common kind of ***data structure***, a ***stack***. Then, we will talk about a new ***algorithm*** called a ***linear search.*** In the second lesson of the week, we'll investigate how to make the search more efficient by introducing a refined algorithm, the ***binary search***.

Reading

As usual, first study this module completely. Then, if you want more, look up the relevant topics in your text.

9A.1.1 An Array-Based Stack

A *stack* satisfies these rules:

- It stores data for us; if we have a ***stack*** object, say **s**, then when we want **s** to store a ***String***, say "hotdog", we ***push*** it onto the stack with a call: **s.push("hotdog")**. "hotdog" is then placed into the ***stack*** **s** and then joins the other items that were on the ***stack***.
- Whenever we go to retrieve data (called ***popping***) from the stack, the item we get will always be the *most recent item* that we ***pushed***. So if we call **s.pop()** right after the above ***push***, the item returned by the **pop()** method would be "hotdog". This also has the effect of removing "hotdog" from the ***stack***, so it is no longer there.

Notice that in the last paragraph, we said "the item returned by **pop()**." That means **pop()** returns a value to the client - it *does not* make an independent decision to, say, print the ***popped*** item to the screen.

Note

pop() should not do any output - it should just return the popped value as a **functional return.**

An exception for our purposes is that **pop()** could output a console error message if the stack were empty. However, this is not ideal, and eventually we will use a technique called ***exceptions*** to do this. For now, we allow console error messages to be output from certain methods.

Let's say that we have defined a ***Stack*** class, and we have instantiated a ***Stack*** object, **s**. Then, using **s**, if we push three times:

```
s.push("four");
s.push("nine");
s.push("two");
```

Then the ***Stack*** **s** looks like this:

```
top ->
       "two"
       "nine"
       "four"
```

If you then **pop()** once, the "two" is returned from the ***stack*** and it will then contain:

```
top ->
       "nine"
       "four"
```

If you **push()** another item:

```
s.push("7.3");
```

the ***stack*** becomes:

```
top ->
       "7.3"
       "nine"
       "four"
```

If you **pop()** now, the "7.3" will be returned and if you **pop()** immediately after that, the "nine" will be returned, leaving only the "four" on the stack.

I think you get the idea.

Since **s** is an object of class **Stack**, we should be able to instantiate more than one **Stack** at a time, say **s1** and **s2**, and **push()** and **pop()** from them independently.

9A.1.2 A Stack Example

This example defines the **Stack** class using techniques we have already learned, and mixes up the ***pushing*** and ***popping*** of the two stacks. You'll have to compare the output to the source code carefully so you can associate the pops with the output that results.

```
// ------------- Class Foothill --------------
public class Foothill
{

// -------  main --------------
public static void main(String[] args)
{
   MyStack s1, s2;
   int k;

   // Initialize stacks  ------------
   s1 = new MyStack();
   s2 = new MyStack();

   // Test the Stack -----
   System.out.println(s1.pop());

   s1.push( "money" );
   s1.push( "in" );
   s1.push( "the" );
   s2.push( "bank" );
   s1.push( "a penny saved is" );
   s1.push( "123456789.123456789" );

   s2.push( "a penny earned" );
   s2.push( "2" );
   s1.push( "3" );
   s2.push( "4" );

   System.out.println("\n------------ First Stack ------------\n");
   for (k=0; k<8; k++)
      System.out.print(s1.pop() + " : ");

   System.out.println("\n------------ Second Stack ------------\n");
   for (k=0; k<8; k++)
      System.out.print(s2.pop() + " : ");
   }
} // end of class StackTest ---------------------
```

```
// ------------- Class MyStack --------------
class MyStack
{
   final private static int SIZE = 10;
   private String stck[];
   private int tos;

// ------------------------------------------------
public  MyStack()
{
   tos = 0;
   stck = new String[SIZE];
}

// ------------------------------------------------
public boolean push( String item )
{
   if (tos == SIZE)
      return false;
   stck[tos++] = new String(item);
   return true;
}

// ------------------------------------------------
public String pop()
{
   if (tos==0)
      return "Stack Empty";

   return stck[--tos];
}
} // end of class MyStack -------------------------
```

And the output:

```
Stack Empty

------------ First Stack ------------

3 : 123456789.123456789 : a penny saved is : the : in :
money : Stack Empty : Stack Empty :
------------ Second Stack ------------

4 : 2 : a penny earned : bank : Stack Empty : Stack Empty :
Stack Empty : Stack Empty :
```

You should easily be able to make the following improvements to this example:

- Allow the client to specify the size of the stack in an overloaded constructor.
- Use the return **boolean** of **push()** in the client to report a full stack.
- Add a method that could be called **init()**, whose job would be to reset the stack object, effectively removing all existing items on the stack, preparing it for a fresh reuse.

Section 2 - The Linear Search Algorithm

9A.2.1 Searching for a Student

We reprise the example of the **StudentArrayUtilities** class which processes an ***array of Student objects***. You can refer to the original example by returning to the lesson on arrays. We will introduce a new class method, **arraySearch()** that takes two parameters:

1. ***A Student array*** to search.
2. A ***first*** and ***last name*** (two **Strings**) for which to search (together, called the ***search key***).

If **arraySearch()** finds a student in the array that matches the search key, it returns the index of that **Student**, i.e., the array index where the **Student** was found. If not, it returns a -1. Here is how **arraySearch()** can be used:

```
String first, last;
int found;

first = "pamela"; last = "jacobs";
found = StudentArrayUtilities.arraySearch(myClass, first, last );
if ( found >= 0 )
   System.out.println(
         first + " " + last + " IS in list at position " + found);
else
   System.out.println( first + " " + last + " is NOT in list.");
```

This static class method is very easy to write. We just plow through the array, comparing the first and last names until we find a match. Here it is:

```
public static int arraySearch(Student[] array,
   String keyFirst, String keyLast)
{
   for (int k = 0; k < array.length; k++)
      if ( array[k].getLastName().equals(keyLast)
         && array[k].getFirstName().equals(keyFirst) )
         return k;  // found match, return index

   return -1;   // fell through - no match
}
```

There isn't much fancy going on here. This is called a ***linear search*** because we are going through the array in a straight line from bottom to top, until we find a match.

Finally, here is the whole program with an output, demonstrating the linear search.

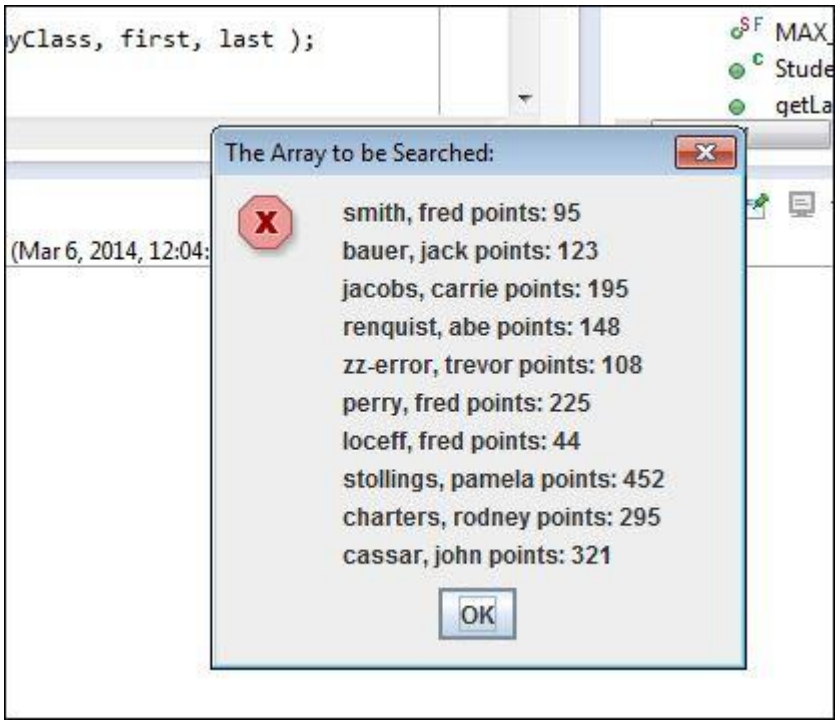
```
import javax.swing.*;

public class Foothill
{
   public static void main (String[] args)
   {
      Student[] myClass = { new Student("smith","fred", 95),
         new Student("bauer","jack",123),
         new Student("jacobs","carrie", 195),
         new Student("renquist","abe",148),
         new Student("3ackson","trevor", 108),
         new Student("perry","fred",225),
         new Student("loceff","fred", 44),
         new Student("stollings","pamela",452),
         new Student("charters","rodney", 295),
         new Student("cassar","john",321),
         };

      StudentArrayUtilities.printArray("The Array to be Searched:", myClass);

      String first, last;
      int found;

      first = "pamela"; last = "stollings";
      found = StudentArrayUtilities.arraySearch(myClass, first, last );
      if ( found >= 0 )
         System.out.println( first + " " + last
               + " IS in list at position " + found);
      else
         System.out.println( first + " " + last + " is NOT in list.");

      first = "pamela"; last = "jacobs";
      found = StudentArrayUtilities.arraySearch(myClass, first, last );
      if ( found >= 0 )
         System.out.println( first + " " + last
               + " IS in list at position " + found);
      else
         System.out.println( first + " " + last + " is NOT in list.");

      first = "carrie"; last = "jacobs";
      found = StudentArrayUtilities.arraySearch(myClass, first, last );
      if ( found >= 0 )
         System.out.println( first + " " + last
               + " IS in list at position " + found);
      else
         System.out.println( first + " " + last + " is NOT in list.");
   }
}
```

```
class Student
{
   private String lastName;
   private String firstName;
   private int totalPoints;

   public static final String DEFAULT_NAME = "zz-error";
   public static final int DEFAULT_POINTS = 0;
   public static final int MAX_POINTS = 1000;

   // constructor requires parameters - no default supplied
   public Student( String last, String first, int points)
   {
      if ( !setLastName(last) )
         lastName = DEFAULT_NAME;
      if ( !setFirstName(first) )
         firstName = DEFAULT_NAME;
      if ( !setPoints(points) )
         totalPoints = DEFAULT_POINTS;
   }

   public String getLastName() { return lastName; }
   public String getFirstName() { return firstName; }
   public int getTotalPoints() { return totalPoints; }

   public boolean setLastName(String last)
   {
      if ( !validString(last) )
         return false;
      lastName = last;
      return true;
   }

   public boolean setFirstName(String first)
   {
      if ( !validString(first) )
         return false;
      firstName = first;
      return true;
   }

   public boolean setPoints(int pts)
   {
      if ( !validPoints(pts) )
         return false;
      totalPoints = pts;
      return true;
   }
```

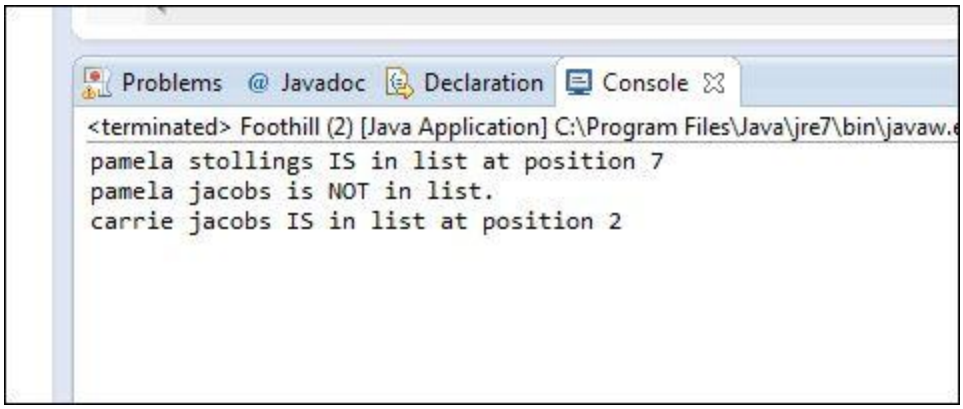
```
   // could be an instance method and, if so, would take one parameter
   public static int compareTwoStudents( Student firstStud, Student secondStud )
   {
      int result;

      // this particular version based on last name only (case insensitive)
      result = firstStud.lastName.compareToIgnoreCase(secondStud.lastName);

      return result;
   }

   public String toString()
   {
      String resultString;

      resultString = " "+ lastName
         + ", " + firstName
         + " points: " + totalPoints
         + "\n";
      return resultString;
   }

   private static boolean validString( String testStr )
   {
      if (testStr != null && Character.isLetter(testStr.charAt(0)))
         return true;
      return false;
   }

   private static boolean validPoints( int testPoints )
   {
      if (testPoints >= 0 && testPoints <= MAX_POINTS)
         return true;
      return false;
   }
}

class StudentArrayUtilities
{
   // print the array with string as a title for the message box
   // this is somewhat controversial - we may or may not want an I/O
   // methods in this class.  we'll accept it today
   public static void printArray(String title, Student[] data)
   {
      String output = "";

      // build the output string from the individual Students:
      for (int k = 0; k < data.length; k++)
         output += " "+ data[k].toString();

      // now put it in a JOptionPane
      JOptionPane.showMessageDialog( null, output, title,
         JOptionPane.OK_OPTION);
   }
```

```
   // returns true if a modification was made to the array
   private static boolean floatLargestToTop(Student[] data, int top)
   {
      boolean changed = false;
      Student temp;

      // compare with client call to see where the loop stops
      for (int k = 0; k < top; k++)
         if ( Student.compareTwoStudents(data[k], data[k+1]) > 0 )
         {
            temp = data[k];
            data[k] = data[k+1];
            data[k+1] = temp;
            changed = true;
         }
      return changed;
   }

   // public callable arraySort() - assumes Student class has a compareTo()
   public static void arraySort(Student[] array)
   {
      for (int k = 0; k < array.length; k++)
         // compare with method def to see where inner loop stops
         if ( !floatLargestToTop(array, array.length-1-k) )
            return;
   }

   public static int arraySearch(Student[] array,
      String keyFirst, String keyLast)
   {
      for (int k = 0; k < array.length; k++)
         if ( array[k].getLastName().equals(keyLast)
            && array[k].getFirstName().equals(keyFirst) )
            return k;  // found match, return index

      return -1;   // fell through - no match
   }
}
```

The output:



... followed by ...



As you can see, this is a mixture of fake-GUI (JOptionPane) and console. We wouldn't really do this normally, but it doesn't hurt to see the combination.