Unit Five Lecture One:

Topic 1: One Dimensional Arrays

An array is a fixed-length sequence of values of the same type, usually of a primitive data type.

In most cases you will want to store a reference to the array in a variable so that you can access it later.

For example: `int[] data = new int[5];`

In this example, data is a reference to an array of integers. The call new int[5] creates the actual array of 5 integers. The reserved word new is mandatory. When an array of primitive data types is first created, all values are initialized with 0. The positions of arrays are numbered starting at 0.

data[0]	data[1]	data[2]	data[3]	data[4]
0	0	0	0	0

To change a value in the data array, you need to specify which position in the array you want to access.

```
data[3] = 7;
```

Now the position with index (or subscript) 3 is filled with the value 7.

data[0]	data[1]	data[2]	data[3]	data[4]
0	0	0	7	0

In Java, it is an error to try to access nonexistent positions.

```
int x = data[5];
```

There is no data[5]. Remember, the array has length 5, so index values range from 0 to 4. The compiler does not catch this error. However, when an invalid index is detected during the execution of a program, the program will terminate.

In Java, an array has an instance variable *length*, which you can access to find out the size of the array.

data.length == 5, the value given to it when the new command was executed.

```
For Example:   for (int j=0; j<data.length; j++)
               {
                    data[j] = 2*j;
               }
```

data[0]	data[1]	data[2]	data[3]	data[4]
0	2	4	6	8

Note that there are no parentheses following *length*. In Java, *length* is an instance variable not a method. It is highly recommended that you use the *length* variable rather than some constant value.

Reminder: When working with a String object, *length* is a method and would require parentheses.

In Java, you are allowed to list all elements that you want to include in the array, enclosed in braces and separated by commas.

```
For Example:   double[] myData = {1.6, 2.3, 4.5, 7.1};
```

The Java compiler counts how many elements you want to place in the array, allocates an array of the correct size, and fills it with the elements that you specify. The new command is not used in this situation and myData.length == 4.

myData[0]	myData[1]	myData[2]	myData[3]
1.6	2.3	4.5	7.1

Topic 2: The Enhanced for loop

Java 1.5 introduces a very convenient shortcut for a common loop type. Often, you need to iterate through a sequence of elements - such as the elements of an array. The enhanced for loop (sometimes called the "for each" loop) makes this process particularly easy to program.

Example:

```
int[] data = {7, 4, 5, 8, 9, 2};
int sum = 0;

for (int x : data)
{
      sum += x;
}
```

Notice, the enhanced for loop is used to visit all elements of an array. The first time through the loop, x takes on the 7, the second time through the loop, x takes on the 4, so on and so forth.

The enhanced for loop can't be used to initially fill an array.

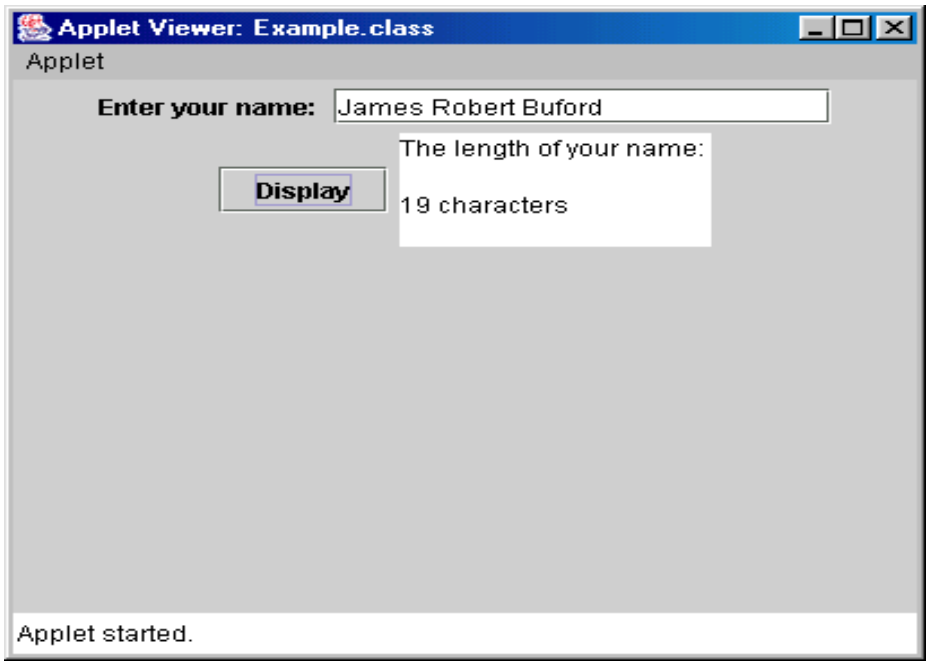
Assignment U5A1: The Correlation Problem

Topic 2: Intermediate GUI's

The next step in your understanding of Java GUI's, is to get away from JOptionPanes and start putting buttons right on your applet. This may sound easy, but taking this step also requires an understanding of event-handling methods.

Let's take a look at a simple applet, that will allow the user to enter a name into a JTextField, hit a JButton, and the number of characters in the name will appear in a JTextArea.

The applet should look like the one below:



After you type in the name, you must hit "Enter". This will invoke the ActionListener for the JTextField which places the name into memory.

After you hit "Enter", then you may click on the Display button. This will invoke the ActionListener for the JButton which counts the characters in the name and prints the result to the JTextArea.

Now let's take a look at the code that produced this Applet.

```
import javax.swing.JApplet;
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.JTextArea;
import javax.swing.JButton;
import java.awt.Container;
import java.awt.FlowLayout;
import java.awt.event.*;

public class Example extends JApplet implements ActionListener
{
      private JTextField field1;
      private JTextArea output;
      private JButton display;
      private String name;      // must be an instance field even though
                                // it appears in only one method
      public void init()
      {
            JLabel label1;

            Container container = getContentPane();
            container.setLayout(new FlowLayout());

            label1 = new JLabel("Enter your name: ");
            container.add(label1);

            field1 = new JTextField(20);
            field1.addActionListener(this);
            container.add(field1);

            display = new JButton("Display");
            display.addActionListener(this);
            container.add(display);

            output = new JTextArea();
            container.add(output);
            output.setText("The length of your name: \n\n");
      }

      public void actionPerformed(ActionEvent event)
      {
            int len;

            if (event.getSource() == field1)
                  name = event.getActionCommand();

            if (event.getSource() == display)
            {
                  len = name.length();
                  output.append(len + " characters\n");
            }
      }
}
```

Most of the import statements are pretty straight forward. But, let's look at **import java.awt.FlowLayout;** . This class sets up the most basic layout manager. GUI components are placed on a container from left to right in the order in which they are added to the container. When the edge of the container is reached, components are continued on the next line. Let's also look at **import java.awt.event.*;** . This package contains many data types that enable a program to process a user's interactions with a program's GUI. In this program we use the ActionListener and ActionEvent data types from this package.

In the line **public class Example extends JApplet implements ActionListener** , extends means that Example is inheriting from the JApplet class and implements means that it is using an interface called ActionListener. Both inheritance and interfaces will be dealt with later in the course.

The next line that is new to you is **container.setLayout(new FlowLayout());** . This simply tells us that the container will use a FlowLayout, as described above.

After this line, notice that a label, textfield, button, and textarea are added to the containter. Also notice, that an ActionListener was added to both the textfield and the button. The line **field1.addActionListener(this);** is used to tell field1 that the applet(this) can listen for action events related to the textfield. When this line is added to the code, there must also be an actionPerformed method added, that defines what happens when the user uses the textfield.

In the actionPerformed method, the line **if (event.getSource() == field1)** is telling the compiler that if an action is performed on the textfield (a name is typed and Enter is hit) execute the following line. The line **name = event.getActionCommand();** is telling the compiler to take the String entered into the textfield and store it in the String variable called name.

Assignment U5A1: The Dealing Cards Problem