Unit Seven Lecture One:

Topic 1: Parts of a class

Up to this point in our study of Java, I have only provided you with enough information about classes to get routine tasks accomplished. It is time to learn about classes in more depth and begin our investigation of Object Oriented Programming (OOP).

To illustrate the various parts of a class, I'm going to design a WestComputerStudent class.

```
public class WestComputerStudent
{
```

Every class (except perhaps a driver class) will have private data. These are called instance variables (fields). These variables will hold different values for each object of type WestComputerStudent. Class constants (final) are also placed in this part of the class. If these constants are to be applied to all objects of type WestComputerStudent they should be declared as static.

```
private int progAccum;
private int testAccum;
private int progCount;
private int testCount;
private String grade;
private int computerNum;
private static final double progRate = .2;
private static final double testRate = .8;
```

Some classes will have static variables. A static variable, like a static final, holds a value that is the same for the entire class. In other words, it is global to all objects declared to be of type WestComputerStudent, within a project.

```
private static int lastAssignedNum = 0;
```

In some classes you will see all the private data at the end of the class, this is an acceptable option.

Placing the access specifier 'private' in front of all the instance variables means they can be accessed only by methods of the same class. Thus, the instance variables of an object are effectively hidden from clients of the class. Typically, instance variables are of concern only to the creator of the class. The process of hiding the data and providing methods for data access is called ***encapsulation***.

All classes have at least one constructor, called the default constructor. It assigns initial values to all of the private data. A constructor must have the same name as the class itself.

```
public WestComputerStudent()
{
      progAccum = 0;
      testAccum = 0;
      progCount = 0;
      testCount = 0;

      grade = "Incomplete";

      lastAssignedNum++;
      computerNum = lastAssignedNum;
}
```

Many classes have additional constructors called parametric constructors. In a parametric constructor, values for some or all of the private data are passed to the class from the client program. When creating parametric constructors the creator of the class is making major decisions on how the client is going to interface with the class.

```
public WestComputerStudent(int pSum, int pCount, int tSum, int tCount)
{
      progAccum = pSum;
      progCount = pCount;
      testAccum = tSum;
      testCount = tCount;

      grade = setGrade();

      lastAssignedNum++;
      computerNum = lastAssignedNum;
}
```

Other parametric constructors are possible. Can you think of some examples? Using more than one constructor is called ***method overloading***.

Notice in the parametric constructor that a method called setGrade() is called to fill the instance variable grade. Any method, besides a constructor, that is used to initialize the value of private data is called a mutator method. The name of a mutator method usually starts with the word 'set' and mutator methods are usually private because there is no need for a client to have access to them. Notice the student's grade remains incomplete until at least one program score and one test score have been inputted.

```
private String setGrade()
{
      double ave1, ave2, ans;
      String gr = "Incomplete";

      if (progAccum == 0 || testAccum == 0)
            return "Incomplete";

      ave1 = (double)progAccum/progCount;
      ave2 = (double)testAccum/testCount;

      ans = ave1 * progRate + ave2 * testRate;

      if (ans >= 89.5)  gr = "A";
      if ((ans >= 79.5) && (ans < 89.5))  gr = "B";
      if ((ans >= 69.5) && (ans < 79.5))  gr = "C";
      if ((ans >= 59.5) && (ans < 69.5))  gr = "D";
      if (ans < 59.5)  gr = "F";

      return gr;
}
```

At this point in our class we have, one way or another, initialized all of our private data. We must now try to anticipate how a client of the class might want to manipulate the private data. Methods used to do this are called public member (instance) methods.

```
public void addProg(int score)
{
      progAccum += score;
      progCount++;

      grade = setGrade();
}
```

This method allows the client to add a program score and update the student's grade. A similar method should exist to add a test score.

There are some situations when the creator of a class feels a client of that class may need access to an instance variable. This can be done through an accessor method. The name of an accessor method usually starts with the word 'get' and is public so the client can use it.

```
public String getGrade()
{
      return grade;
}

public int getComputerNum()
{
      return computerNum;
}
```

Lastly, methods exist which don't operate on a particular instance of an object. These methods are called static methods. For example, Math.sqrt(x); can be executed without first declaring an object of type Math.

```
public static String seeRates()
{
      String ans;
      double percent1, percent2;

      percent1 = 100 * progRate;
      percent2 = 100 * testRate;

      ans = "Programs count " + percent1 + " % and ";
      ans += "Tests count " + percent2 + " %\n";

      return ans;
}
```

As you can see in this example, the static method seeRates() is completely independent of any of the objects declared to be of type WestComputerStudent. Therefore, it would be called from the client program like this...

```
String output = WestComputerStudent.seeRates();
```

Below you will find a driver class for the WestComputerStudent class. I tried to document it so you would understand when each member of the WestComputerStudent class was called.

```
import javax.swing.JApplet;
import javax.swing.JTextArea;
import java.awt.Container;

public class CompStuTest extends JApplet
{
      public void init()
      {
            int num1, num2;
            String gr1, gr2, rates;

            // calling default constructor
            WestComputerStudent x = new WestComputerStudent();

            // calling parametric constructor
            WestComputerStudent y = new WestComputerStudent(198, 2, 76, 1);

            // calling the accessor method getComputerNum
            num1 = x.getComputerNum();
            num2 = y.getComputerNum();

            // calling public member (instance) methods addProg & addTest
            x.addProg(100);
            x.addProg(90);
            x.addTest(57);
            y.addProg(95);
            y.addTest(70);
```

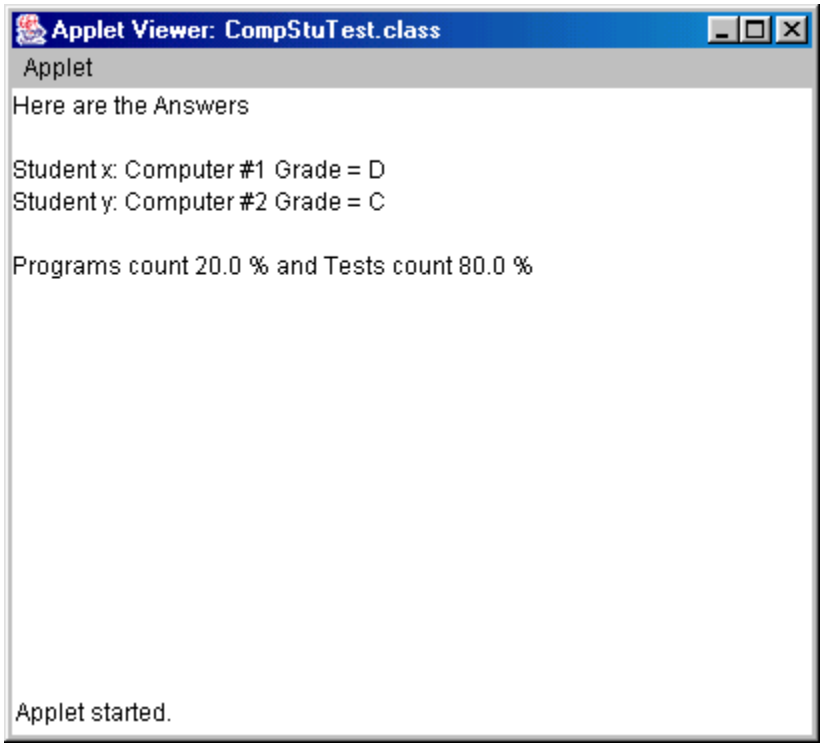
```
            // calling the accessor method getGrade
            gr1 = x.getGrade();
            gr2 = y.getGrade();


            // calling static method seeRates
            rates = WestComputerStudent.seeRates();

            // outputting the data
            JTextArea output = new JTextArea();
            output.setText("Here are the Answers\n\n");
            output.append("Student x: Computer #" + num1);
            output.append(" Grade = " + gr1 + "\n");
            output.append("Student y: Computer #" + num2);
            output.append(" Grade = " + gr2 + "\n\n");
            output.append(rates);

            Container container = getContentPane();
            container.add(output);
      }
}
```

The applet would look like this...



Topic 2: Scope

The scope of a variable is the block of code in which you can refer to the variable by its name. In Java, you cannot have two local variables with overlapping scope.

It is possible in Java to have a local variable use the same name as an instance variable. If this occurs, the local variable will have precedence inside the block of code in which it was declared. Inside this same block of code, it is still possible to access the instance variable but it must be preceded by the word 'this'. The word 'this' automatically refers to the present object.

Unit Seven Assignment One: The BankAccount Class