Unit Nine Lecture One:

Topic 1: Inheritance

Inheritance is a method for extending existing classes by adding methods and fields.

Topic 2: Superclass

A general class from which other classes will inherit methods and fields.

Topic 3: Subclass

A specialized class that inherits methods and fields from the superclass. It also adds instance variables and methods or overrides methods of the superclass.

Topic 4: Overridden Method

If both the superclass and the subclass have a method with the same name and same parameter types, the method of the subclass takes precedence. In other words, the subclass method overrides the superclass method.

Topic 5: How is an interface different from inheritance?

An interface is not a class. It does not have instance fields and it does not have coded methods. It merely tells you which methods you should implement. A superclass has both instance fields and coded methods, and subclasses inherit them.

Topic 6: Why is inheritance important?

"One important reason for inheritance is code reuse. By inheriting from an existing class, you do not have to replicate the effort that went into designing and perfecting that class."

Topic 7: Implementing Inheritance

Step 1: Create a superclass.

- Create instance fields that will be common to all the classes inheriting from the superclass.

```
public class Circle
{
    private double radius;
    private static final double pi = 3.14;
```

- Create a parametric constructor that will initialize the instance fields.

```
    public Circle(double x)
    {
          radius = x;
    }
```

- Create any methods that would be appropriate for all the classes inheriting from the superclass.

```
    public double areaCircle()
    {
          double ans;

          ans = pi * radius * radius;

          return ans;
    }

    public double circumference()
    {
          double ans;

          ans = 2 * pi * radius;

          return ans;
    }

    public double getRadius()
    {
          return radius;
    }

    public static double getPi()
    {
          return pi;
    }
}
```

Step 2: Create a subclass.

- Create instance fields that will be specific to the subclass.

```
public class Cylinder extends Circle
{
   private double height;
```

- Create a parametric constructor that will send the appropriate variables to the super class and initialize the other instance fields. The call to super must be the first line of the constructor!

```
   public Cylinder(double r, double h)
   {
         super(r);
         height = h;
   }
```

- Create any methods that would be specific to your subclass.

```
   public double volume()
   {
         double ans;

         ans = areaCircle() * height;

         return ans;
   }

   public double surfaceArea()
   {
         double ans;

         ans = 2 * areaCircle() +  circumference() * height;

         return ans;
   }
}
```

Notice that methods of the superclass are called as if they were methods of the subclass!

Step 3: Create all other subclasses that will inherit from the superclass.

Step 4: Create the driver class that will test the
        superclass and its subclasses.

```
public class driver extends JFrame
{
    .
    .
    .
    public void TestCylinder()
    {
          double volume;
          double surfaceArea;

          Cylinder solid1 = new Cylinder(2.5, 5.1);

          volume = solid1.volume();
          surfaceArea = solid1.surfaceArea();
          .
          .
          .
```

*** Special Situation ***

What if both the superclass and the subclass had the same method name, but they served different purposes?

```
public double myMethod(int x)
{
      .
      .
      .
}
```

In the subclass, you would call the method in the superclass like this...

```
int y = 5;
double ans = super.myMethod(y);
```

Without the reserved word 'super', the local method would override the method of the superclass.

In the subclass, you would call the method of that subclass like this...

```
int y = 8;
double ans = this.myMethod(y);
            or
double ans = myMethod(y);
```

The reserved word 'this' is optional. It explicitly informs the compiler that you want to use the local method.