Introduction to Java
Notes for Packet 4: Classes and Objects

Classes. A class is generally used to serve as a blueprint to create objects. It is a way to define a set of data and define how to interact with that data.

Here is an example of two classes. The Runner class creates and uses two objects of the Box class.

```
public class Box{
    private int side;

    public Box( int len ) {
        side = len;
    }

    public int getVolume() {
        int vol = side * side * side;
        return vol;
    }

    public void change( int x ) {
        side = x;
    }
}
```

```
public class Runner {
    public static void main(String[] args) {
        Box bob = new Box( 3 );
        int v = bob.getVolume();
        System.out.println( v + " cu.in." );

        Box ann = new Box( 4 );
        v = ann.getVolume();
        System.out.println( v + " cu.in."  );

        ann.change( 2 );
        v = ann.getVolume();
        System.out.println( v + " cu.in."  );
    }
}
```

A class can be divided into three sections:

Instance variables **This is the data in each object. Declared at the top (in this course). Every constructor/method can use them**

Constructors **Used to create objects and initialize the instance variables. Constructors must have the same name as their class.**

Methods. A method is a block of code in a class that has:

1) a visibility modifier **public (means other classes can use it)**

2) a return type **Describes the type of data a method returns.**

3) a name **Generally same rules as naming a variable.**

4) a parameter list. **list of inputs to a method.**

A method may or may not refer to the class's instance variables.

The variables in a class can be grouped into three categories.

```
public class Box{
    private int side;

    public Box( int len ) {
        side = len;
    }

    public int getVolume() {
        int vol = side * side * side;
        return vol;
    }

    public void change( int x ) {
        side = x;
    }
}
```

1) **instance variables (e.g. side)**

2) **parameters (e.g. len, x) are declared in the first line of a method or constructor. They are initialized when the method or constructor is called.**

3) **local variables (e.g. vol) are declared within a method or constructor. All of the variables we have used previous to this unit have been local variables.**

Methods. Every method must have a return type but not every method returns something.

Example 1 has a return type of int and returns an int. Example 2 has a return type of **void** which means the method does not return anything.

Example Method 1.

```
public int doubleIt( int num ) {
        return 2 * num;
}
```

Example Method 2.

```
public void circleArea( double radius ) {
        if ( radius <= 0 ) {
                System.out.println( "Must be positive.");
                return;
        }

        double area = radius * radius * 3.14;
        System.out.println( "The area is " + area );
}
```

Example Method 3.

```
public double findMin( double a, double b ) {
        if ( a < b )
                return a;
        else
                return b;
}
```

IMPORTANT. A return statement causes a method to stop and return control to the calling method.

The process of creating an object is called _____**instantiation**_____. In general, you must first ___**instantiate**___ an object before you can call any of its methods. When calling a constructor, use the keyword ___**new**___

Let's go back to the example that we started with.

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

        Box bob = new Box( 3 );
        int v = bob.getVolume();
        System.out.println( v + " cu.in." );

        Box ann = new Box( 4 );
        v = ann.getVolume();
        System.out.println( v + " cu.in."  );

        ann.change( 2 );
        v = ann.getVolume();
        System.out.println( v + " cu.in."  );
    }
}
```

Arguments and Parameters. An argument is the name of the variable or value being passed to a corresponding parameter in a method. If you change the value of a parameter, **the value of the argument does not change.**

```
public class Rock {
        private int weight;

        public Rock( int w ) {
                weight = w;
        }

        public void methodX( int a ){
                a = a + 5;
                System.out.println( a );
        }
}
```

```
public class Runner {
        public static void main(String[] args) {
                Rock r = new Rock( 100 );
                int n = 22;
                r.methodX( n );
                System.out.println( n );
                r.methodX( 13 );
        }
}
```

Name the parameters in the Rock class. _____**w, a**_____

Name the arguments in the Runner class. _____**100, n, 13**_____

Another Data Type. So far we have been using two types of primitive variables: ints and doubles. Another data type is the boolean data type. Variables of type boolean have a value of ___**true**___ or ___**false**___. Wherever you can use a boolean expression, you may also use a boolean variable.

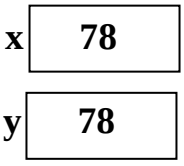
```
public class Runner {
    public static void main(String[] args) {
        Calculator c = new Calculator();
        int x = c.squareIt( 7 );
        System.out.println( x );

        c.set( true );
        x = c.squareIt( 5 );
        System.out.println( x );

        c.set( false );
        x = c.squareIt( 9 );
        System.out.println( x );
    }
}
```

```
public class Calculator {
    private boolean on;

    public Calculator() {
        on = false;
    }

    public int squareIt( int a ) {
        if ( on )
            return a*a;
        else
            return 0;
    }

    public void set( boolean b ) {
        on = b;
    }
}
```

The Difference between Primitive Data Types and Object Data Types.

First, remember the definition of a variable. A variable is a ___**named storage location in memory**___. So, an obvious question is: what is stored in a particular variable? For primitive data types (e.g. ___**double**___ and ___**int**___) the answer is easy. The variable stores the data.

```
int x = 78;
int y = x;
```



For object variables, the answer is more complicated. An object variable contains a ___**reference**___ to an object - not the object itself.

```
Point p1 = new Point( 3, 4 );
Point p2;
p2 = p1;
```