Lesson 15…..Classes and Objects

A **class** is like a cookie cutter and the "cookies" it produces are the **objects**:
One cookie cutter……..many possible cookies.
One **class**……...…many possible **objects**.

Building a *Circle* class:

Let's build a class and begin to understand its parts. Our class will be called *Circle*. When we create one of our *Circle* objects (just like creating a cookie), we will want to specify the radius of each circle. We will want to have the ability to interrogate the various *Circle* objects we might have created and ask for the area, circumference, or diameter.

```
public class Circle
{
        //This part is called the constructor and lets us specify the radius of a
        //particular circle.
        public Circle(double r)
        {
                radius = r;
        }

        //This is a method. It performs some action (in this case it calculates the
        //area of the circle and returns it.
        public double area( )  //area method
        {
                double a = Math.PI * radius * radius;
                return a;
        }

        public double circumference( )  //circumference method
        {
                double c = 2 * Math.PI * radius;
                return c;
        }

        public double radius;  //This is a State Variable…also called Instance
                               //Field and Data Member. It is available to code
                               // in ALL the methods in this class.
}
```

Instantiating an object:

Now, let's use our cookie cutter (the *Circle* class) to create two cookies (*Circle* objects). Place the following code in the *main* method of a different class (*Tester*).

```
Circle cir1 = new Circle(5.1);
Circle cir2 = new Circle(20.6);
```

With a cookie-cutter we say we **create** a cookie. With a class we **instantiate** an object. So, we just instantiated an object called *cir1* having a radius of 5.1 and another object

called *cir2* having a radius of 20.6…. From this point on we don't refer to *Circle*. Instead we refer to *cir1* and *cir2*.

Let's suppose we wish to store the radius of *cir1* in a variable called *xx*.
Here's the code to do this:

```
double xx = cir1.radius;
```

Now let's ask for and printout the area of *cir2*:

```
System.out.println ( cir2.area( ) );
```

A closer look at methods:

We will now look at the **signature** (also called a **method declaration**) of this *area* method and then examine each part.

```
public double area( )   //this is the signature
```

Access control (*public*, *private*, etc.):

The word ***public*** gives us access from outside the *Circle* class. Notice above that we used *cir2.area( )* and this code was in some other class…so "public" lets us have access to the *area( )* method from the outside world. It is also possible to use the word ***private*** here. (more on this later)… Strictly speaking, *public* and *private* are not officially part of the signature; however, since they generally always preface the actual signature, we will consider them part of the signature for the remainder of this book.

Returned data type (*double*, *int*, *String*, etc):

The word ***double*** above tells us what type variable is returned. When we issue the statement *System.out.println(cir2.area( ));*, what do we expect to be "returned" from the call to the *area* method? The answer is that we expect a double precision number since the area calculation may very well yield a decimal fraction result.

Method name:

The word ***area*** as part of the signature above is the name of the method and could be any name you like…even your dog's name. However, it is wise not to use cute names. Rather, use names that are suggestive of the action this method performs.

Naming convention:

Notice all our methods begin with a small letter. This is not a hard-and-fast rule; however, it is conventional for variables and objects to begin with lower case letters.

Parameters:

The parenthesis that follows the name of the method normally will contain parameters. So far, in our circle class none of the methods have parameters so the parenthesis are all empty; however, the parenthesis must still be there.

Let's create a new method in which the parenthesis is **not** empty. Our new method will be called *setRadius*. The purpose of this is so that after the object has been created (at which time a radius is initially set), we can change our mind and establish a **new** radius for this particular circle. The new signature (and code) will be as follows:

```
public void setRadius(double nr)
{
        radius = nr;  //set the state variable radius to the new radius
}                     //value, nr
```

We see two new things here:

a. ***void*** means we are **not** returning a value from this method. Notice there is no *return* in the code as with the other methods.

b. *double nr* means the method expects us to send it a *double* and that it will be called *nr* within the code of this method. *nr* is called a **parameter**.

   Here is how we would call this method from within some other class:

   ```
   cir2.setRadius(40.1);  //set the radius of cir2 to 40.1
   ```

   40.1 is called an **argument**. The terms arguments and parameters are often carelessly interchanged; however, the correct usage of both has been presented here.

   Notice that there is no equal sign in the above call to *setRadius*. This is because it's void (returns nothing)… therefore, we need not assign it to anything.

Have you noticed another way we could change the radius?

```
cir2.radius = 40.1;  //We store directly into the public instance field.
```

Understanding *main*:

At this point we are capable of understanding three things that have remained mysterious up to now. Consider the line of code that's a part of all our programs:

```
public static void main(String args[ ])
```

1. ***main*** is the name of this special **method**

2. ***public*** gives us access to this method from outside its class

3. ***void*** indicates that this method doesn't return anything

The other parts will have to remain a mystery for now.

The constructor:

Next, we will look at the constructor for the *Circle* class.

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

The entire purpose of the constructor is to set values for some of the state variables of an object at the time of its creation (construction). In our *Circle* class we set the value of the state variable, *radius*, according to a double precision number that is passed to the constructor as a parameter. The parameter is called *r* within the constructor method; however, it could be given any legal variable name.

The constructor is itself a method; albeit a very special one with slightly different rules from ordinary methods.

1. **public** is always specified.

2. The name of the constructor method is always the **same** as the name of the class.

3. This is actually a void method (since it doesn't return anything); however, the *void* specifier is omitted.

4. The required parenthesis may or may not have parameters. Our example above does. Following is another example of a *Circle* constructor with **no** parameters. A constructor with no parameters is called the **default constructor**.

```
public Circle( )
{
        radius =100;
}
```

   What this constructor does is to just blindly set the radii to 100 of all *Circle* objects that it creates.

Project… What's That Diameter?

Create a new method for the *Circle* class called *diameter*. Add this method to the *Circle* class described on page 15-1. It should return a *double* that is the diameter of the circle. No parameters are passed to this method.

In a *Tester* class, test the performance of your new *diameter* method as follows:
(Your project should have two classes, *Tester* and *Circle.*)

```
public class Tester
{
        public static void main( String args[] )
        {
                Circle cir1 = new Circle(35.5);
                System.out.println( cir1.diameter( ) );  // should give 71.0 as the answer.

        }
}
```

Exercise on Lesson 15

1. double length = 44.0;
   int width =13;
   Rectangle myRect = new Rectangle(length, width);
   a. Identify the class
   b. Identify the object
   c. What type of parameter(s) are passed to the constructor?

2. Write out the signature for the constructor of the *Rectangle* class from #1 above.

3. Suppose a constructor for the *Lunch* class is as follows:

```
public Lunch(boolean diet, int cal)
{
        diet_yes_no = diet;
        calories = cal;
}
```

   Write appropriate code that will create a *Lunch* object called *yummy5*. Tell the constructor that, yes, you are on a diet, and the number of calories should be 900.

4. BankAccount account39 = new BankAccount(500.43);
   a. Identify the class
   b. Identify the object
   c. What type of parameter(s) are passed to the constructor?

5. A class is like a _____. An object is like a _____.
   Fill in the blanks above using the word "cookie" and "cookie cutter".

6. What's wrong (if anything) with the following constructor for the *School* class?

```
public void school(int d, String m)
{  … some code …  }
```

7. Which of the following is a correct association?
   a. One class, many objects
   b. One object, many classes

8. Which must exist first?
   a. The class
   b. The object

9. Is the following legal? If not, why?

```
//Constructor
public House(int j, boolean k)
{  …some code…  }
```

```
//This code is in main of  Tester class
int p = 3, q = 9;
House myHouse = new House(p, q);
```

10.

```
//Constructor
public Band(int numMembers, int numInstruments, String director, double amount)
{  …code…}
```

```
Band ourBnd = new Band(mem, instrmnts, "Mr. Perkins", budget);
```

What should be the data types of:

a. mem
b. instrmnts
c. budget

****

```
public class BibleStory
{
    public int var1;
    public double var2;
    public String sss;

    public void Samson(double zorro)  {   …some code…}
    public String getDelilah( )  {   …some code…}
    public BibleStory(String x, int y, double z)  {   …some code…  }
}
```

11. From the *BibleStory* class above, write the signature of the constructor.

12. From the *BibleStory* class above, what is/are the instance field(s).

13. From the *BibleStory* class above, write the signature(s) of the all the method(s).

14. Write code that instantiates an object called *philistine* from the *BibleStory* class. Pass the following parameters to the constructor:
    The integer should be 19, the *String* "Ralph", and the *double* 24.18.

15. Assume an object called *gravy* has been created from the *BibleStory* class. Write code that will set the state variable *var2* to 106.9 for the *gravy* object.

16. Write code that will print the value of the *BibleStory* data member, *sss*. Assume you have already created an object called *bart*.

17. Again, assume we have an object called *bart* instantiated from the *BibleStory* class. What should you fill in for **<#1>** below in order that *sss* be stored in the variable *jj*?
    <#1> jj = bart.sss;

18. Create a class called *Trail*. It should have instance fields *x* and *y* that are integers. Instance field *s* should be a *String*. The constructor should receive a *String* which is used to initialize *s*. The constructor should automatically set *x* and *y* both equal to 10. There should be a method called *met* that returns a *String* that is the hex equivalent of *x* y*. This method receives no parameters.

19. Consider a method whose signature is: *public double peachyDandy(int z)*

    Write code that would call this method (assume we have an object name *zippo*). Also assume that this code will be placed in the *main* method of a *Tester* class and that the *peachyDandy* method is in some other class.

20. Refer to the information in 19 above. What's wrong with trying to call this method in the following fashion? *double hamburger = zippo.peachyDandy(127.31);*

Project … Overdrawn at the Bank

Create a class called *BankAccount*. It should have the following properties:

1. Two state variables:
   double balance… This is how much money is currently in the account.
   String name…The name of the person owning the account.

2. Constructor should accept two parameters.
   a. One should be a *double* variable that is used to initialize the state variable, *balance*.

   b. The other should be a *String* that is used to initialize the state variable, *name*.

3. Two methods:
   a. *deposit*…returns nothing…accepts a *double* that is the amount of money being deposited. It is added to the *balance* to produce a new balance.

   b. *withdraw*…returns nothing…accepts a *double* that is the amount of money being taken out of the account. It is subtracted from the *balance* to produce a new *balance*.

Create a *Tester* class that has a *main( )* method. In that method you should input from the keyboard the amount (1000) of money initially to be put into the account (via the constructor) along with the name of the person to whom the account belongs.

1. Use these two pieces of data to create a new *BankAccount* object called *myAccount*.

2. Call the *deposit* method to deposit $505.22.

3. Print the *balance* state variable.

4. Call the *withdraw* method to withdraw $100.

5. Print the remaining *balance* in this form:

```
The Sally Jones account balance is, $1405.22
```