6. Object Oriented Concepts and Techniques

Lesson 1:
Basics of Object Orientation – Part 2

6.3. Methods

- Methods are group of related statements in a class of objects that act on themselves and on other classes and objects.
- Methods are used to accomplish specific tasks. Objects communicate with each other using methods.
- Instance methods
  - Apply to an instance object of the class.
  - If the method makes a change to an individual object, it must be an instance method.
- Class methods
  - Apply to the class itself.

6.3. Methods
6.3.1. Defining methods

```
<modifier> <return type> <method name> (<parameter list>)
{
    <statements>
}
```

- **<modifier>** is optional
- **<return type>** can be any primitive type or a class Name or void (meaning that there is no return statement)
- Usually the **<modifier>** for methods is **public** and for attributes it is **private**

6.3. Methods
6.3.2. Signature of a Method

- The **<return type>**, **<method name>** and the **<parameter list>** defines the Signature of the method

- It is possible to define two ore more methods with the same name within the same class with different signatures. This is known as Method Overloading.
  - For example,

```
public void CreatePoint()
{
}
public void CreatePoint(int x , int y)
{
}
```

6.3. Methods
6.3.3. Accessing methods

- An instance method can be accessed as:

  <object name>.<method name>(<value list>);

  – For example,

  person1.SetName("Tom");

- An class method can be accessed as:

  <class name>.<method name>(<value list>);

  – For example,

  Integer.parseInt("25");

6.3. Methods
6.3.4. Recursion

- A method might call another method from within its body. In fact, it might even call itself. A method calling itself is known as recursion.

- Though this might seem a little odd, recursion is a very common and versatile concept.

- Here example of method that implements the well known mathematical function “factorial”.

```
public int factorial(int x)
{
        if(x==0)
        {
                return 1;
        }
        return x*factorial(x);
```

6.4. Parameter Passing

- All primitive data types are passed by value. Any modifications done within the method does not affect the original variable.
- Object types (instances of classes) and Arrays are passed by reference, Any modifications done within the method affects the original variable.
- If you require to modify the original variables of primitive data types, and this needs to be done by passing them to a method, we must declare the primitive data type variables as instance variables in a class and pass an object of that class to the method.
- Alternatively we can use **wrapper classes**. Wrapper classes are special classes that represent primitive data types and hence can be passed reference to methods.
  - For example, the wrapper class **Integer** represents primitive data type **int**.

6.5. Constructor Methods

- These methods are used to initialize objects
- They have the same name as the class and have no return type
- These methods are called automatically when the **new** operator is used to allocate memory for an object.
- A class can have multiple Constructors (Overloaded Constructors). They have either different number of arguments or different types of arguments (that is, they have different signatures)

6.5. Constructor Methods

6.5.1. Example for Constructors

```
public class TwoDimentionalPoint
{
        private float x, y;
        public TwoDimentionalPoint ()
        {
                x = 0;
                y = 0;
        }
        public TwoDimentionalPoint (float a,float b)
        {
                x=a;
                y=b;
        }
}
```

- **Note, the first constructor has no parameters passed to it. Also, note that the constructor has been overloaded.**