Class Methods

There are two modifiers for Class Methods in Apex: Public or Protected. Return type is mandatory for method and if method is not returning anything then you must mention void as return type. Body is required for method.

Syntax:

```
[public | private | protected | global]
[override]
[static]
return_data_type method_name (input parameters)
{
// Method body goes here
}
```

Explanation of Syntax:

Those parameters mentioned in the square brackets are optional. However, below components are required:

- return_data_type
- method_name

Access Modifiers for Class Methods:

Using access modifiers, you could specify access level for class methods. For Example, Public method will be accessible from anywhere in the class and outside of the Class. Private method will be accessible only within class. Global will be accessible by all the Apex classes and can be exposed as web service method accessible by other apex classes.

Example:

```
//Method definition and body
public static Integer getCalculatedValue () {
    //do some calculation
    myValue = myValue+10;
    return myValue;
}
```

This method has return type as Integer and takes no parameter.

A Method can have parameters as shown in the below example:

```
//Method definition and body, this method takes parameter price which will then be used
in method.
public static Integer getCalculatedValueViaPrice (Decimal price) {
 //do some calculation
 myValue = myValue+price;
 return myValue;
}
```

Class Constructors

A constructor is a code that is invoked when an object is created from the class blueprint. It has the same name as class name.

We don't need to define the constructor for every class, as by default a no-argument constructor gets called. Constructors are useful when we would like to have some initialization of variables or process to be done at the time of class initialization. For example: You would like to assign values to certain Integer variables as 0 when the class gets called.

Example:

```
//Class definition and body
public class MySampleApexClass2 {
public static Double myValue;   //Class Member variable
public static String myString;  //Class Member variable

public MySampleApexClass2 () {
    myValue = 100;  //initialized variable when class is called

}

public static Double getCalculatedValue () {    //Method definition and body
    //do some calculation
    myValue = myValue+10;
    return myValue;
}

public static Double getCalculatedValueViaPrice (Decimal price) {   //Method definition
and body
    //do some calculation
    myValue = myValue+price;//Final Price would be 100+100=200.00
    return myValue;
}
}
```

You could call the method of class via constructor as well. This may be useful when programming Apex for visual force controller. When class object is created, then constructor is called as shown below:

```
//Class and constructor has been instantiated
MySampleApexClass2 objClass = new MySampleApexClass2();
Double FinalPrice = MySampleApexClass2.getCalculatedValueViaPrice(100);
System.debug('FinalPrice: '+FinalPrice);
```

Overloading Constructors

Constructors can be overloaded, i.e. a class can have more than one constructors defined with different parameters.

Example:

```
public class MySampleApexClass3 {   //Class definition and body
public static Double myValue;       //Class Member variable
public static String myString;      //Class Member variable

public MySampleApexClass3 () {
    myValue = 100;  //initialized variable when class is called
    System.debug('myValue  variable with no Overaloading'+myValue);
}

public MySampleApexClass3 (Integer newPrice) {  //Overloaded constructor
    myValue = newPrice; //initialized variable when class is called
    System.debug('myValue  variable with Overaloading'+myValue);
}

public static Double getCalculatedValue () {    //Method definition and body
    //do some calculation
    myValue = myValue+10;
    return myValue;
}


public static Double getCalculatedValueViaPrice (Decimal price) {   //Method definition
and body
    //do some calculation
    myValue = myValue+price;
    return myValue;
}
}
```

You could execute this class as we have executed it in previous example.

```
//Developer Console Code
MySampleApexClass3 objClass = new MySampleApexClass3();
Double FinalPrice = MySampleApexClass3.getCalculatedValueViaPrice(100);
System.debug('FinalPrice: '+FinalPrice);
```