Overriding & Overloading

Overriding

Method overriding, in object oriented programming, is a language feature that allows a subclass or child class to provide a specific implementation of a method that is already provided by one of its super-classes or parent classes. The implementation in the subclass overrides (replaces) the implementation in the superclass by providing a method that has same name, same parameters or signature, and same return type as the method in the parent class. The version of a method that is executed will be determined by the object that is used to invoke it. If an object of a parent class is used to invoke the method, then the version in the parent class will be executed, but if an object of the subclass is used to invoke the method, then the version in the child class will be executed.

A Few Rules

- An override method provides a new implementation of a member that is inherited from a base class.
- You cannot use the new, static, or virtual modifiers to modify an override method.
- An overriding property declaration must specify exactly the same access modifier, type, and name as the inherited property
- The base class method must be defined virtual.
- The base class method can be called from within the derived class using the base keyword.

By default, C# methods are not virtual. If a method is declared as virtual, any class inheriting the method can implement its own version. To make a method virtual, the virtual modifier is used in the method declaration of the base class. The derived class can then override the base virtual method by using the override keyword

Example:

```
**********************************************

public class Employee
  {
     public string name;
     // Basepay is defined as protected, so that it may be
     // accessed only by this class and derived classes.

     protected decimal basepay;

// Constructor to set the name and basepay values.
     public Employee(string name, decimal basepay)
     {
        this.name = name;
        this.basepay = basepay;

     // Declared virtual so it can be overridden.

     public virtual decimal CalculatePay()
     {
        return basepay;
     }
  }

**********************************************

public class SalesEmployee : Employee
  {
     // New field that will affect the base pay.
     private decimal salesbonus;

     // The constructor calls the base-class version, and
     // initializes the salesbonus field.
     public SalesEmployee(string name, decimal basepay,  decimal salesbonus) : base(name, basepay)
     {
        this.salesbonus = salesbonus;
     }

     // Override the CalculatePay method
     // to take bonus into account.

     public override decimal CalculatePay()
     {
        return basepay + salesbonus;
     }
  }


**********************************************
```

Overloading

Method overloading

Function overloading or method overloading is a feature found in various programming languages that allows creating several methods with the same name which differ from each other in the type of the input and the output of the function. It is simply defined as the ability of one function to perform different tasks.

For example, *doTask()* and *doTask(object O)* are overloaded methods. To call the latter, an object must be passed as a parameter, whereas the former does not require a parameter, and is called with an empty parameter field. A common error would be to assign a default value to the object in the second method, which would result in an *ambiguous call* error, as the compiler wouldn't know which of the two methods to use.

Example

```
public static int Compare(
    string strA,
    string strB
);

public static int Compare(
    string strA,
    string strB,
    bool ignoreCase
);
```

Constructor overloading

Constructors, used to create instances of an object, may also be overloaded in some object oriented programming languages. Because in many languages the constructor's name is predetermined by the name of the class, it would seem that there can be only one constructor.

```
class Program
{
    class C2
    {
          int A;
          int B;
          public C2(int a, int b)
          {
              A = a;
              B = b;
          }
    }

    static void Main()
    {
          C2 c = new C2(1, 2);
    }
}
```