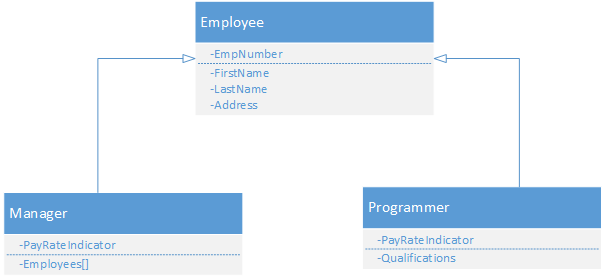
**Introducing Inheritance**  
  
Inheritance is yet another pillar in the world of object-oriented programming. You can use inheritance as an aspect of code reuse by defining different classes that will contain common features and have a relationship to one another.  An example could be employees as a general classification and which could contain managers, non-management workers, and any other employee classification.

Consider creating an application to simulate an office workspace that includes all the employees.  Then consider the common features that all employee classifications have followed by a list of attributes that are different for each employee type.  For example, they all might have an employee number, first and last names, addresses, etc., but managers have different responsibilities than other employee classifications.

Inheritance allows you to create a base class containing the core, shared attributes, and then each different class of employee would inherit these attributes whole extending them for their own special needs.  The class that inherits from the base class is referred to as the derived class but also commonly referred to as a subclass.  When using the term subclass, some also refer to the base class as a super class.  In programming languages such as Objective-C, this is reinforced by the use of statements such like this example where the keyword super is used to initialize a nib file in a super class;

self = [super initWithNibName:nibNameOrNil bundle:nibBundleOrNil];

Consider the following simplified UML Class diagram as an example.



**Applying Inheritance**

The C# programming language does not support multiple inheritance directly.  Multiple inheritance is a concept whereby multiple base classes can be inherited by a single subclass.  In C#, a derived class can only have one base class.

To inherit from base class in C#, you append your derived class name with a colon and the name of the base class.  The following example demonstrates the Manager class inheriting the Employee base class from the previous topic's UML diagram.

class Manager : Employee  
{  
    private char payRateIndicator;  
    private Employee[] emps;  
}

This simple class definition in C# lists the keyword class followed by the class name Manager, a colon and then the name of the base class Employee.  Looking at this snippet we can't tell what the Manager class has inherited from Employee so we would need to look at that class as well to understand all the properties available for us.  The Employee class is shown here:

class Employee  
{  
    private string empNumber;  
    private string firstName;  
    private string lastName;  
    private string address;  
  
    public string EmpNumber  
    {  
        get  
        {  
            return empNumber;  
        }  
  
        set  
        {  
            empNumber = value;  
        }  
    }  
  
    public string FirstName  
    {  
        get  
        {  
            return firstName;  
        }  
  
        set  
        {  
            firstName = value;  
        }  
    }  
  
    public string LastName  
    {  
        get  
        {  
            return lastName;  
        }  
  
        set  
        {  
            lastName = value;  
        }  
    }  
  
    public string Address  
    {  
        get  
        {  
            return address;  
        }  
  
        set  
        {  
            address = value;  
        }  
    }  
}

When working in Visual Studio, the Intellisense feature will provide you with a visual representation of the inherited members.  As an example, if we were to instantiate an object of type Manager in our code and then use the dot notation to bring up the list of properties for the Manager class, we would also see the properties from the base class Employee in that list as well.  This is shown in the following image:

