**Abstract Classes**

Looking back at our inheritance topic, we note that the Employee class is being used as the base class for Manager and Programmer. We can continue to extend the Employee class by creating as many sub classes as required for different employees in our application. However, when looking at our class hierarchy, does it make sense to be able to create an object of type Employee, directly?  Certainly the base class contains common properties but realistically we would prefer to only create objects of specific types of employees.

In order to enforce this behavior in our code, we should consider making the Employee class an abstract class.  Abstract classes are closely related to interfaces, which will be covered in the next topic. Abstract classes cannot be instantiated, which means we would not be able to create a new Employee object in code with this statement:

Employee newEmployee = new Employee();

 When you create an abstract class you may partially implement some of the behavior in the class, or not implement the behavior at all.  An abstract class requires the subclass to implement some, or all, of the functionality.  If we extend our previous example of the Employee and Manager classes, using abstract classes, we can demonstrate this concept better.  Note that the employee class now includes some methods to implement behaviors.

abstract class Employee  
    {  
        private string empNumber;  
        private string firstName;  
        private string lastName;  
        private string address;  
  
        .....  
  
        public virtual void Login()  
        {  
        }  
  
        public virtual void LogOff()  
        {  
        }  
  
        public abstract void EatLunch();  
  
    }

In this code sample, the ..... is used to denote that some code was snipped from this sample to shorten it for display.   Consider the properties still present in the code.

Also notice that we have now prepended the keyword abstract to our class: abstract class Employee.  Doing so converts our class to an abstract class and sets up some requirements.  Once you create an abstract class, you decide which methods "must" be implemented in the sub classes and which methods "can" be implemented, or overridden, in the sub class.  There is a clear difference.

Any method you declare in the abstract class that will contain some implementation in the abstract class, but can be overridden in the sub class, you decorate with the virtual keyword.  Note in the previous code sample,Login() and LogOff() are both decorated with the virtual keyword.   This means that you can write implementation code in the abstract class and sub classes are free to override the implementation , or accept the implementation that is inherited.

The EatLunch() method is decorated with the abstract keyword, like the class.  There are specific constraints around an abstract method:

* An abstract method cannot exist in non-abstract class
* An abstract method is not permitted to have any implementation, including curly braces
* An abstract method signature must end in a semi-colon
* An abstract method **MUST** be implemented in any sub class.  Failure to do so will generate a compiler warning in C#.