Lesson 25 - Inheritance & Polymorphism I

Inheritance

Inheritance allows us to define a class based on another class. This makes creating and maintaining an application easy. The class whose properties are inherited by another class is called the **Base** class. The class which inherits the properties is called the **Derived** class. For example, base class **Animal** can be used to derive **Cat** and **Dog** classes. The derived class inherits all the features from the base class, and can have its own additional features.

> Inheritance allows us to define a class based on another class.

Animal base class:

```
class Animal
{
        public int Legs { get; set; }
        public int Age { get; set; }
}
```

Now we can derive the Dog class:

```
class Dog : Animal
{
        public Dog()
                => Legs = 4;

        public void Bark()
                => Console.WriteLine("Woof");
}
```

Note the syntax for a derived class. A **colon** and the name of the **base** class follow the name of the derived class. All public members of **Animal** become public members of **Dog**. That is why we can access the **Legs** member in the **Dog** constructor. Now we can instantiate an object of type **Dog** and access the inherited members as well as call its own **Bark** method.

```
Dog d = new();
Console.WriteLine(d.Legs);
d.Bark();
```

A base class can have multiple derived classes. For example, a **Cat** class can inherit from **Animal**.

> Inheritance allows the derived class to reuse the code in the base class without having to rewrite it. And the derived class can be customized by adding more members. So, the derived class extends the functionality of the base class.

A derived class inherits all the members of the base class, including its methods.

```
class Person
{
        public void Speak()
                => Console.WriteLine("Hi there");
}

class Student : Person
{
        int number;
}
```

```
Student s = new();
s.Speak();
```

We created a **Student** object and called the **Speak** method, which was declared in the base class **Person**.

> C# does not support multiple inheritance, so you cannot inherit from multiple classes. However, you can use interfaces to implement multiple inheritance.

Protected Members

Up to this point, we have worked exclusively with `public` and `private` access modifiers. Public members may be accessed from anywhere outside of the class, while access to private members is limited to their class. The `protected` access modifier is very similar to `private` with one difference; it can be accessed in the derived classes. So, a `protected` member is accessible only from derived classes.

```
class Person
{
        protected int Age { get; set; }
        protected string Name { get; set; }
}

class Student : Person
{
        public Student(string nm)
                => Name = nm;
        public void Speak()
                => Console.WriteLine("Name: " + Name);
}
```

```
Student s = new("David");
s.Speak();
```

As you can see, we can access and modify the `Name` property of the base class from the derived class. But, if we try to access it from outside code, we will get an error:

```
Student s = new("David");
s.Name = "Bob"; //Error
```

Sealed Classes

A class can prevent other classes from inheriting it, or any of its members, by using the `sealed` modifier.

```
sealed class Animal
{
        // class members
}

class Dog : Animal { } //Error
```

In this case, we cannot derive the Dog class from the Animal class because Animal is `sealed`.

> The sealed keyword provides a level of protection to your class so that other classes cannot inherit from it.