Polymorphism & Virtual Methods

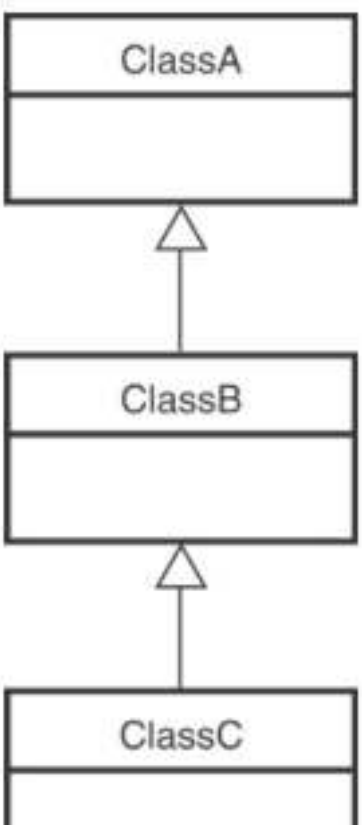
The lesson discusses the concepts of polymorphism including subtyping and virtual methods in detail using examples

WE'LL COVER THE FOLLOWING

- Polymorphism Definition
- Subtyping
  - Example
- Virtual Methods
  - Example

Polymorphism Definition

The term **Polymorphism** means the ability to take many forms. It occurs if there is a hierarchy of *classes* which are all related to each other by *inheritance*.



Subtyping

Subtyping is a form of *polymorphism* in which another *class inherits* from a **base** *class* to *generalize* a similar behavior.

You basically have a *method inherited* from **base** *class* but it is implemented differently in the **child** *classes*.

Example

Let's take a look at an example of *polymorphism* below.

```
using System;

class Car {
  //refuel method defined in base class
  void refuel(){Console.WriteLine("Refueling a car");}
}

class NormalCar: Car { //child class inheriting from base class Car
  public void refuel() { //changing definition of refuel() in child class using subtyping pol
    Console.WriteLine("Refueling with petrol");
  }
}

class ElectricCar: Car { //child class inheriting from base class Car
  public void refuel() { //changing definition of refuel() in child class using subtyping pol
    Console.WriteLine("Charging battery");
  }
}

class PolymorphismExample {
  static void Main() {
    NormalCar car1 = new NormalCar(); //making an abject of child class NormalCar
    car1.refuel(); //calling the refuel() function of the NormalCar class object

    ElectricCar car2 = new ElectricCar(); //making an abject of child class ElectricCar
    car2.refuel(); //calling the refuel() function of the ElectricCar class object
  }
}
```

As you can see from the example above both classes `NormalCar` and `ElectricCar` now have a *method* to `refuel`, but their own *implementations*.

Virtual Methods

A **virtual** *method* is a *member* function which is declared in the *base* class using the *keyword* `virtual` and is re-defined (*Overriden*) by the *derived* class.

> **Note:** To override a **virtual** *method*, use the *keyword* `override`

Virtual functions can be used while implementing *Polymorphism*.

Example

Now let's look at an example which exhibits **Polymorphism** and makes use of **virtual functions** to do so.

```
using System;

class Vehicle {
  protected int NumberOfWheels {get;set;} = 0;
  public Vehicle() {}
  public virtual void Display() { //We declare Display method virtual so that the child class
    Console.WriteLine("The number of wheels of Vehicle are: {0}" , NumberOfWheels);
  }
}

class Ducati: Vehicle {
  public Ducati() {
    NumberOfWheels = 2;
  }
  public override void Display() { // The override keyword indicates we want new logic behind
    Console.WriteLine("The number of wheels of Ducati are: {0}", NumberOfWheels); //new defir
  }
}

class Lamborghini: Vehicle {
    public Lamborghini() {
      NumberOfWheels = 4;
    }
    public override void Display() { // The override keyword indicates we want new logic behi
      Console.WriteLine("The number of wheels of Lamborghini are: {0}", NumberOfWheels); //ne
      }
}

class VirtualExample {
      static void Main() {
        Vehicle car1 = new Vehicle();
        car1.Display();
        car1 = new Ducati();
        car1.Display();
        car1 = new Lamborghini();
        car1.Display();
      }
}
```

In the above code:

- The class `Vehicle` takes *multiple* forms as a **base** *class*.
- The **derived** classes `Ducati` and `Lamborghini` *inherit* from `Vehicle` and `override` the *base class's* `Display()` *method*, to display its own `NumberOfWheels`.
- In **line 31** the *object* is created for the *base* type `Vehicle` using a *variable* `car1`.
    - In **line 32** it calls the **base** class *method* `Display()`.
- In **line 33**, the `car1` *object* is pointed to the **derived** *class* `Ducati` and calls its `Display()` *method* in **line 34**.
    - Here comes the **polymorphic** behavior, even though the *object* `car1` is of type `Vehicle`, it calls the **derived** class *method* `Display()` as the type `Ducati` *overrides* the **base** class `Display()` *method*, since the `car1` *object* is pointed towards `Ducati`.

The same explanation is applicable when it invokes the `Lamborghini` type's `Display()` *method*.

In the next lesson we will be discussing the concept of *interface* in C#.