Lecture 4

Inheritance

- **Inheritance** enables new objects to inherit the properties of existing objects.

A class that is used as the basis for inheritance is called a superclass or base class. A class that inherits from a superclass is called a subclass or derived class

Encapsulation

- **Encapsulation** is a concept that binds together the data and methods that manipulate the data, and that keeps both safe from outside interference and misused.

Access Modifiers

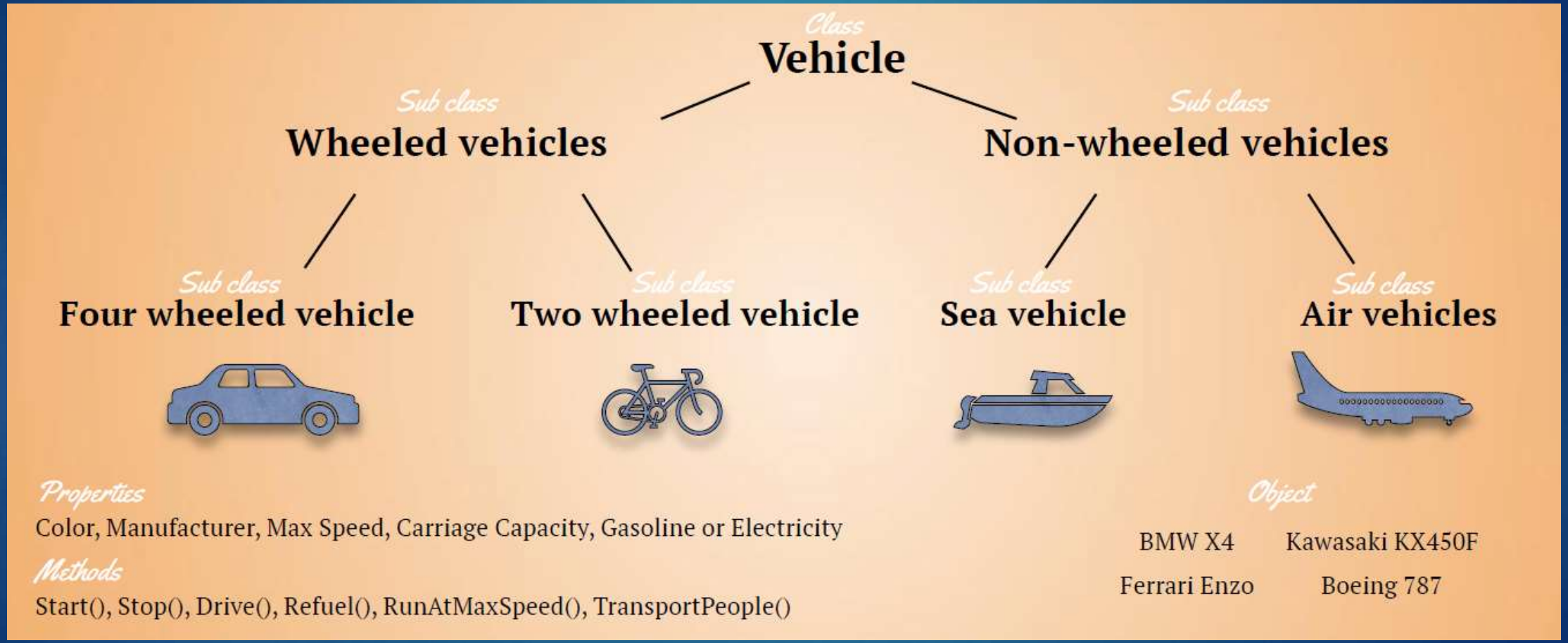
- ***Public***: Access is not restricted.
- ***Protected***: Access is limited to the containing class or types derived from the containing class.
- ***Private***: Access is limited to the containing type.
- ***Internal***: Access is limited to the current assembly.
- ***Protected internal***: Access is limited to the current assembly or types derived from the containing class.

Inheritance example

```
class Program
  {
     static void Main(string[] args)
     {
        Animal animal = new Animal();
        Dogs dog = new Dogs();
        Birds bird = new Birds();
        bird.FeedAnimal();

     }
  }

class Animal
  {
     public string animalName;
     public DateTime animalBirthDate;

     public void FeedAnimal()
     {

     }
  }
```

```
class Dogs : Animal
  {
     public string dogBreed;
     public string dogIntelligence;
     public bool isEasyToTrain;
  }

  class Birds : Animal
  {
     public string birdColor;
     public string birdCountry;
  }
```

Vehicle inheritance



Polymorphism

```
class Program
   {
      static void Main(string[] args)
      {
         Shapes[] shapes = new Shapes[4];
         shapes[0] = new Shapes();
         shapes[1] = new Circles();
         shapes[2] = new Lines();
         shapes[3] = new Triangle();

         foreach (var shape in shapes)
         {
            shape.Draw();
         }
      }
   }
```

```
class Shapes
   {
      public virtual void Draw()
      {
         Console.WriteLine("I am a simple shape");
      }
   }
```

```
class Circles : Shapes
   {
      public new void Draw()
      {
         Console.WriteLine("I am circle");
      }
   }

   class Lines : Shapes
   {
      public override void Draw()
      {
         Console.WriteLine("I am line");
      }
   }
```

```
class Triangle : Shapes
   {
      public override void Draw()
      {
         Console.WriteLine("I am triangle");
      }
   }
```

Polymorphism

- ***Polymorphism*** **means having many forms. usually expressed as 'one interface, multiple functions'.**
- ***Overriding*** **allows you to change the functionality of a method in a child class.**

Static type or compile time	***Dynamic type or runtime***
Overloading	**Overriding**

Abstract class

- ***Abstraction*** **is a concept or an idea not associated with any specific instance.**

```
class Program
  {
     static void Main(string[] args)
     {
        Lines line = new Lines();
        line.SayHi(); line.Draw();
     }
  }

abstract class Shapes
  {
     abstract public void Draw();
     public void SayHi()
     {
        Console.WriteLine("Hi from the
abstact class");
     }
  }
```

```
class Lines : Shapes
{
   public override void Draw()
   {
      Console.WriteLine("Hi I am a line");
   }
}
```