VT Academy
Thinking Ahead, Beyond Boundaries

# Object Oriented Programming

# Inheritance and Polymorphism

## Session 5

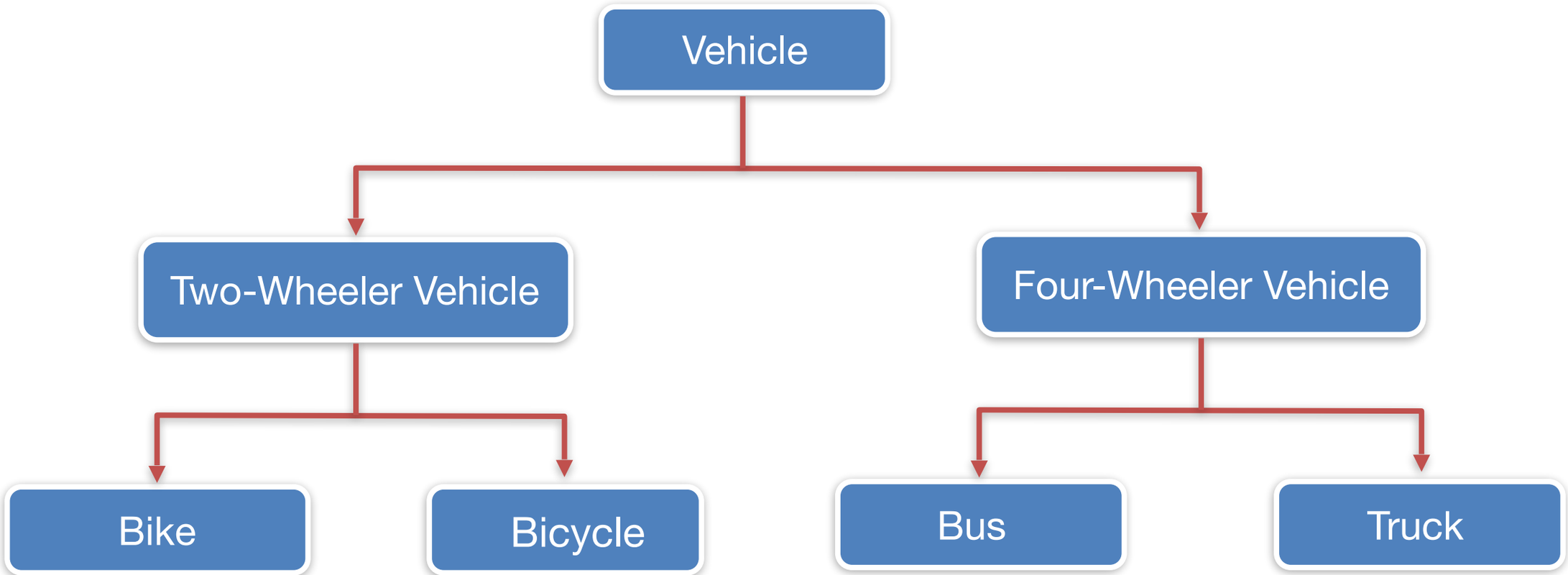
# Objectives

VTC Academy
Thinking Ahead, Beyond Boundaries

- Inheritance
- Polymorphism

# Inheritance

- Inheritance allows you to create a class by deriving the common attributes and methods of an existing class



# Purpose of Inheritance

- Inheritance allows us to define a class in terms of another class, which makes it easier to create and maintain an application
- Inheritance also provides an opportunity to reuse the code functionality and speeds up implementation time
- Apart from reusability, inheritance is widely used for:
  - Generalization
  - Specialization
  - Extension

# Base and Derived Classes

- A class can be derived from only one class or more interfaces, which means that it can inherit data and functions from multiple base classes or interfaces
- Syntax used in C# for creating derived classes

```
<acess-specifier> class <base_class>
{
   ...
}
class <derived_class> : <base_class>
{
   ...
}
```

# Base and Derived Classes

- Example:

```
// Base class
class Shape
{
   public void setWidth(int w)
   {
      width = w;
   }
   public void setHeight(int h)
   {
      height = h;
   }
   protected int width;
   protected int height;
}

// Derived class
class Rectangle: Shape
{
   public int getArea()
   {
      return (width * height);
   }
}
```

# “base” Keyword

- **base** keyword allows you to access the variables and methods of the base class from the derived class.
- When you inherit a class, the methods and variables defined in the base class can be re-declared in the derived class.
- Cannot use the base keyword for invoking the static methods of the base class.
- Syntax demonstrates the use of base keyword:

# “new” Keyword

- **new** keyword used to instantiate a class by creating its object
- This instantiate finally invokes the constructor of the class
- As a modifier, the “new” keyword is used to hide the methods or variables of the base class that are inherited in the derived class
- Allows you to redefine the inherited methods or variables in the derived class
- Syntax demonstrates the use of the new modifier:

# Contructor Inheritance

- In C#, you cannot inherit constructors like you inherit methods.
- You can invoke the base class constructor by either instantiating the derived class or the bas class
- The instance of the derived class will always first invoke the constructor of the base class followed by the constructor of the derived class
- You can explicitly invoke the derived class constructor by using “base” keyword in the derived class constructor declaration

# Contructor Inheritance

Invoked first

```
Base Class : Animal {
        Animal () { }
}
Derived Class : Cat {
        Cat() { }

        public void Create() {
                Cat objCat = new Cat();
        }
}
```

Invoked second

# Method Overloading

- Method overriding is a feature that allows the derived class to override or redefine the methods of the base class
- Overriding a method in the derived class can change the body of the method that was declared in the base class

# “virtual” and “overide” Keywords

- You can override a base class method in the derived class using appropriate C# keywords such as “virtual” and “override”
- Override a particular method of the base class in the derived class, you need to declare the method in the base class using virtual keyword
- A method declared using the virtual keyword is referred to as a virtual method
- In the derived class, you need to declare the inherited virtual method using the override keyword
- The overrides keyword overrides the base class method in the derived class

# Sealed Classes

- Sealed class is a class that prevents inheritance. If a class tries to derive a sealed class, the C# compiler generates an error.
- You can declare a sealed class by preceding the class keyword with the sealed keyword
- The sealed keyword prevents a class from being inherited by any other class
- The sealed class cannot be a base class as it cannot be inherited by any other class
- Syntax:

# Purpose of Sealed Classes

- Ensure Security
- Protect from Copyright Problems
- Prevent Inheritance

## Guidelines

- If overriding the methods of a class might result in unexpected functioning of the class
- When you want to prevent any third party from modifying your class

# Polymorphism

- Polymorphism is derived from two Greek words, namely poly and Morphos
- Poly means many and Morphos means forms
- Polymorphism means existing in multiple forms
- Polymorphism is the ability of an entity to behave differently in different situations
- Polymorphism allows methods to function differently based on the parameters and their data types

# Polymorphism Implementation

- You can implement polymorphism in C# though method overloading and method overriding
- You can create multiple methods with the same name in a class or in different classes having different method body or different signatures
- Methods having the same name but different signatures in a class are referred to as overloaded methods

# Compile-time vs Runtime Polymorphism

VTC Academy
Thinking Ahead, Beyond Boundaries

| Compile-time Polymorphism | Runtime Polymorphism |
| --- | --- |
| In Compile time Polymorphism, call is resolved by the **compiler**. | In Run time Polymorphism, call is **not** resolved by the compiler. |
| It is also known as **Static binding, Early binding** and **overloading** as well. | It is also known as **Dynamic binding, Late binding** and **overriding** as well. |
| **Overloading** is compile time polymorphism where more than one methods share the same name with different parameters or signature and different return type. | **Overriding** is run time polymorphism having same method with same parameters or signature, but associated in a class & its subclass. |
| It is achieved by **function** overloading and **operator** overloading. | It is achieved by **virtual functions** and **pointers**. |
| It provides **fast execution** because known early at compile time. | It provides **slow execution** as compare to early binding because it is known at runtime. |
| Compile time polymorphism is **less flexible** as all things execute at compile time. | Run time polymorphism is **more flexible** as all things execute at run time. |

# Summary

- Inheritance allows us to define a class in terms of another class, which makes it easier to create and maintain an application
- In C#, you cannot inherit constructors like you inherit methods.
- Sealed class is a class that prevents inheritance. If a class tries to derive a sealed class, the C# compiler generates an error.
- You can override a base class method in the derived class using appropriate C# keywords such as “virtual” and “override”
- Polymorphism is the ability of an entity to behave differently in different situations
- You can implement polymorphism in C# though method overloading and method overriding