Inheritance

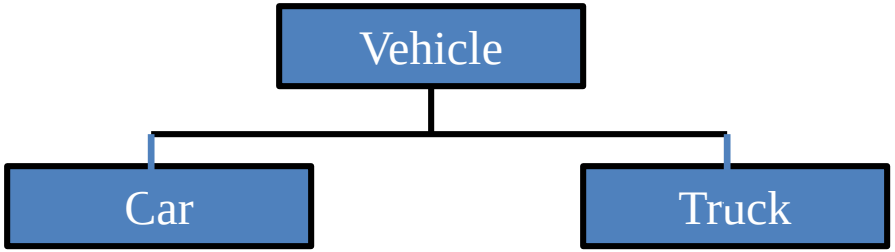
Objectives

- Define inheritance
- Why Inheritance
- Types of Inheritance
- base keyword
- new modifier
- Construction and Inheritance
- Sealed modifier
- Abstract Class and Interface

What is Inheritance

Inheritance is a process which enables you to create new types ,based on an already existing types:

- For example, Car and Truck classes might inherit from Vehicle class
- Fields and methods in the Vehicle class are inherited by Car and Truck
- Car and Truck can define their own fields and behavior



Relationships of Inheritance

AlmaMate

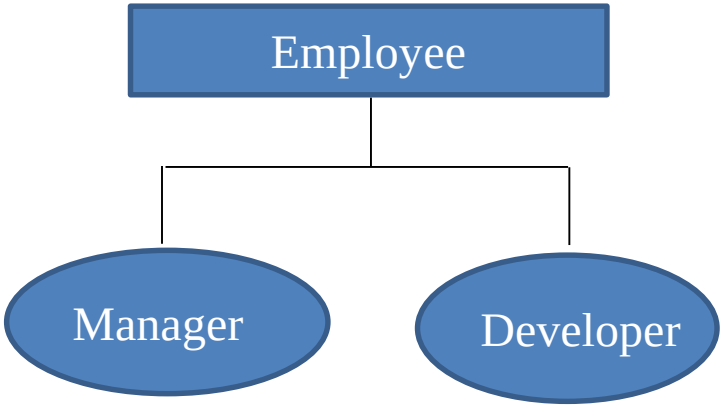
"is-a" relationship

- a subclass can be used wherever a base class can be used
- Is implemented in C# by extending a class

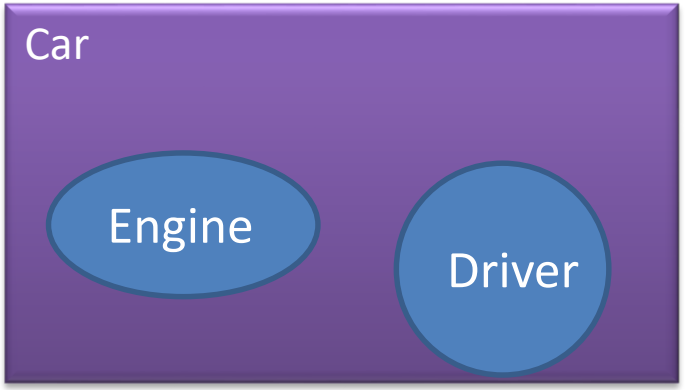
"has-a" relationship

- a whole-class relationship between a class and its parts
- also known as composition
- implemented in C# by instantiating an object inside a class

"is-a" relationship



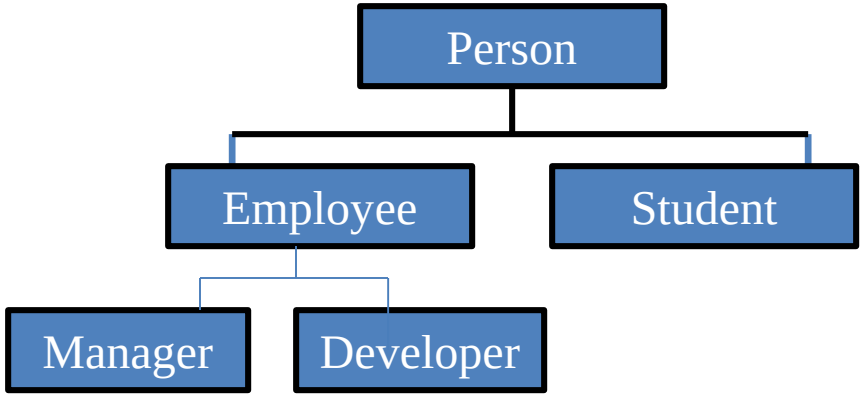
"has-a" relationship

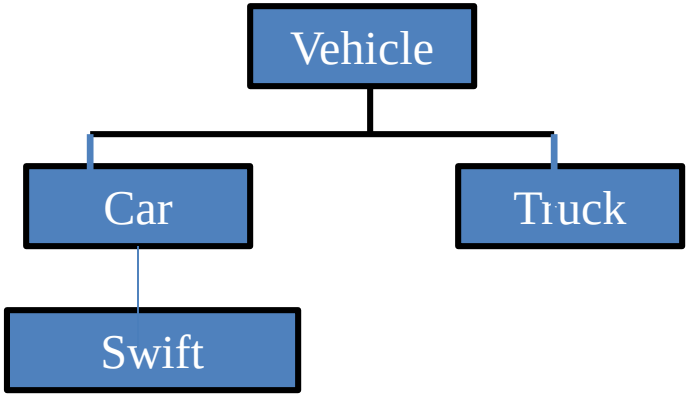


Why Inheritance

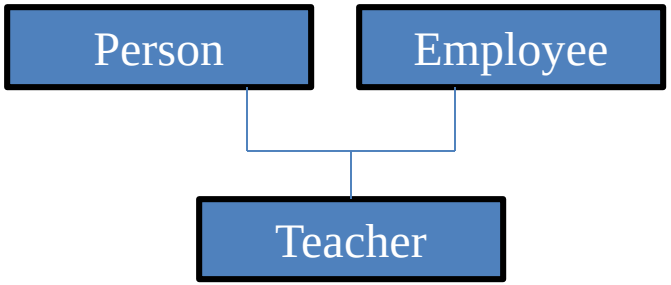
- It provides a way of organizing classes
- To simplify the functionality in a complex system
- To define common properties of classes only once.
- It provides delegation of some functions to different classes, to easily manage the class functionalities.

Types of Inheritance





Single Inheritance



Multiple Inheritance

C# don't support Multiple Inheritance

Rules of Inheritance

- You can inherit a class only from one class (single inheritance)
- A subclass inherits all the properties of base class.
- Sub class can modify or hide behavior and properties of base class
- Inheritance syntax is **:** followed by base class name eg: class child **:** base
- private members are not inheritable
- Object class is the base class of all types

Inheritance Example: 1

```
class Product
    {
        protected string product_ID;
        protected  int price;
}
 public class Book : Product
   {
        string author;
        string title;

}
```

Inheritance operator

Inheritance Example: 2

```
class Product
  {
    string product_ID;
    int price;
    public string Product_ID
    {
      get { return product_ID; }
      set { product_ID = value; }
    }
    public int Price
    {
      get { return price; }
      set { price = value; }
    }
  }
```

```
public class Book : Product
 {
    string author;
    string title;
    static void Main(string[] args)
    {
      Book b1 = new Book();

      b1.Product_ID = "p001";
      b1.Price = 350;

      b1.author = "Tom";
      b1.title = "C# Development";
    }
 }
```

Accessing base class(Product) members

Accessing child class(Book) members

base keyword

```
public class Product
   {
      protected string product_ID;
       protected int price;
   }
    public class Book : Product
  {
       string author;        string title;
      public Book(string product_ID, int
price,  string author, string title)
      {
          base.product_ID =product_ID  ;
          base.price = price;
          this.author = author;
          this.title = title;
      }
```

```
 public void PrintInfo()
       {

System.Console.WriteLine(product_ID);
           System.Console.WriteLine(title);
       }
      static void Main(string[] args)
      {
          Book b1 = new
Book("p001",200,"ABC","GOD");
          b1.printInfo();
                }
}
```

Used to access members of base class and not available in static context

Using new keyword as modifier

In C#, apart from as an operator ,new keyword can be used as a modifier too.

As a modifier it is used to explicitly hide an inherited member from a base class.

```
public class BaseClass
 {
public int j;
  public void Start()
     {   ..........
     }
 }
```

```
public class DerivedClass : BaseClass
{
new public void Start()
{  ............
}
}
```

Construction and Inheritance

- It is must to initialize the entire object(base class part and derived class part) when creating derived class object.
- Base class part should initialize first.
- To invoke base class constructor use :base(…).
- Base class constructor without argument invoked automatically.
- If base class don’t have default constructor then explicitly call available constructor.

Example

```
class Person
 {
     private string name;
      private  string address;
public Person( string name, string
     address)
     {
       this.name = name;
       this.address= address;
      }
 }
```

Overloaded base class constructor (non-default)

```
class Customer : Person
{
     string cust_ID;

      //sub class constructor
     public Customer (string name, string
     address, string cust_ID) : base (name,
     address)
      {
          this.cust_ID = cust_ID;
      }
     public static void Main(string[] args)
     {
      Customer  cust = new
     Customer("Manish", "XYZ","C001");
     } }
```

Constructor invocation: Base to Derived

Calling base constructor

Sealed Modifier

- Used to restrict the inheritance
  - class is defined as **sealed class,** cannot be inherited.

```
sealed class ExampleSealedClass
{
.......
}
```

Abstract Class and method

Provides common behavior across a set of subclasses

- Abstract class can't be instantiated.
- To create use abstract modifier followed by class name.
- May contains abstract and non-abstract methods and properties .
- Abstract method , a method without implementation.
- Non-abstract method, a method with implementation ,can be overridden.
- Use the abstract modifier in a method or property declaration to indicate that the method or property does not contain implementation
- An abstract method is implicitly a virtual method
- The implementation is provided by an overriding method, which is generally a member of a non-abstract class.
- Abstract method declaration don't use static or virtual modifiers
- Class containing abstract method required to be abstract
- Class derived from an abstract class must include actual implementations of all inherited abstract methods and properties.

Example

```
public abstract class MyAbstractClass
  {
     public abstract void Display();
     public void Start()
  {
     System.Console.WriteLine("Start invoked");
  }
 public virtual void Stop()
  {
     System.Console.WriteLine("Stop invoked");
  }
 }
```

Abstract class declaration

Abstract method declaration

Non-Abstract method

Virtual method ,can be overridden in derived class

Example Contd….

Example: Contd….

```
public class DerivedAbstract : MyAbstractClass
  {
    public override void display()
    {
        System.Console.WriteLine("display method implementation ");
    }
    public override void stop()
    {
        System.Console.WriteLine("Stop method is overridden");
    }
     static void Main(string[] args)
     {
          ……————
     }
  }
```

Abstract method implementation

Overriding virtual method

Interface

A contract that specifies what methods must be exposed by an implementing class.

- Defines what a class can do but not how the class will do it.
- Contains abstract members which are independent of the implementation.
- Can inherit from multiple base interfaces.
- Can be implemented by any class.
- Cannot implement any data type.
- Cannot be instantiated.
- Can contain methods, properties, events.
- A class or struct can implement several interfaces, thus enabling multiple inheritance.

Creating and Implementing an Interface

```
interface ICalculator
{
    double Add();
    double Subtract();
    double Multiply();
    double Divide();
}
```

```
class Calculator : ICalculator
{
    public double Add(){}
    public double Subtract(){}
    public double Multiply(){}
    public double Divide(){}
}
```

Single class can implement more than one Interface

AlmaMate

Thank You