Inheritance

Introduction

Another fundamental concept in object-oriented programming is the concept of **inheritance**; it is considered the third pillar of OOP in addition to encapsulation and polymorphism. Inheritance is also known as **code reuse**. Code reuse means that when we build a class for a particular concept, we can reuse this class and inherit its properties and functionality to solve related problems without rewriting the same code again.

In the **C# language**, inheritance allows a class to inherit data members and methods from another class using the colon symbol :. This mechanism saves development time and improves program organization and maintainability.

Inheritance Advantages:

- Promotes code reusability.
- Enhances reliability since the base class code is already tested and debugged.
- Reduces development and maintenance costs.
- Minimizes code redundancy and supports extensibility.
- Facilitates the creation of reusable class libraries in C#.

This lecture presents the concept of inheritance in **C#**, explains its basic types, and demonstrates how to implement them using C# syntax. Before discussing the types of inheritance, we need to understand some essential definitions.

Inheritance in C# is an object-oriented programming (OOP) feature that allows one class to derive properties and behaviors from another class. It promotes code reusability, extensibility and establishes a natural hierarchical relationship between classes.

1. Base Class

A base class is the class whose data members and methods are inherited by another class. It is also called the **parent class** or **superclass**.

---

2. Derived Class

A derived class is the class that inherits data and/or functionality from a base class. It is also called the **child class** or **subclass**.

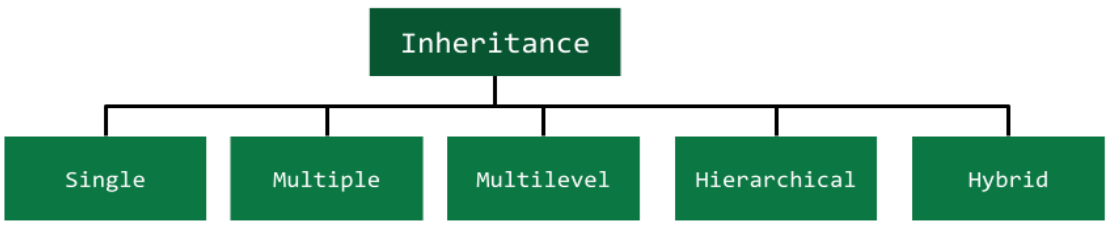
Example: Basic Inheritance

```
using System;

class Animal {
    public void Eat() {
        Console.WriteLine("This animal eats food.");
    }
}

class Dog : Animal {
    public void Bark() {
        Console.WriteLine("The dog barks.");
    }
}

class Program {
    static void Main() {
        Dog d = new Dog();
        d.Eat();   // Inherited method
        d.Bark();  // Derived class method
    }
}
```

Types of Inheritance

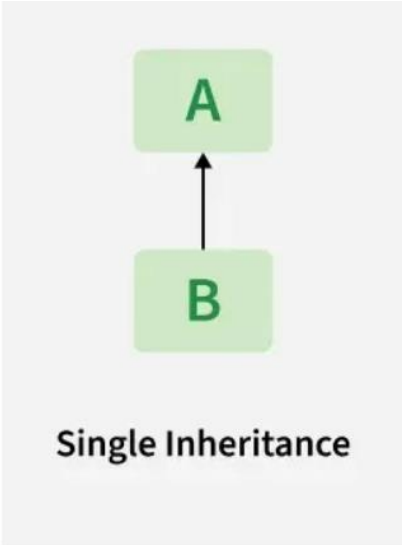
C# directly supports the following inheritance forms:

1. **Single Inheritance:** One class derives from one base class.
2. **Multilevel Inheritance:** A class derives from another derived class.
3. **Hierarchical Inheritance:** Multiple classes derive from a single base class.
4. **Multiple Inheritance (Through Interfaces):** A class can implement multiple interfaces, achieving multiple inheritance indirectly, since C# does not allow multiple base classes.



1. Single Inheritance

In single inheritance, subclasses inherit the features of one superclass.



In the above image, the class A serves as a base class for the derived class B.

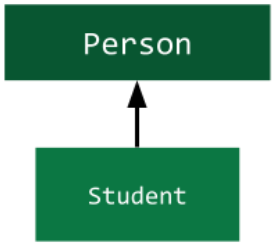
In object-oriented programming, classes represent real-world entities. In inheritance, the base class represents the more **general concept**, while the derived class represents the more **specific concept**.

For example, suppose we want to represent a **Person** and a **Student** in C#. We could create two separate classes, each containing its own data members and methods. However, many attributes are common between them. Every person has a name, age, height, and weight. Likewise, every student also has these same attributes.

Instead of rewriting these common members in both classes, we define them once in the base class (**Person**) and allow the derived class (**Student**) to inherit them. This reduces code duplication and improves reusability.

In this example:

- Person is the **base class** because it represents the more general concept.
- Student is the **derived class** because it represents a more specific concept.



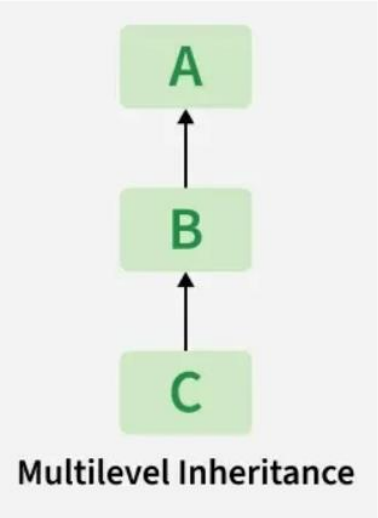
We can say:
Every student is a person, but not every person is a student.

```
using System;

class Person
{
   protected string name;
   protected float age, length, weight;

   public Person()
   {
      Console.Write("Enter Name: ");
      name = Console.ReadLine();

      Console.Write("Enter Age: ");
      age = float.Parse(Console.ReadLine());

      Console.Write("Enter Length: ");
      length = float.Parse(Console.ReadLine());

      Console.Write("Enter Weight: ");
      weight = float.Parse(Console.ReadLine());
   }

   public virtual void Print()
   {
      Console.WriteLine("Name: " + name);
      Console.WriteLine("Age: " + age);
      Console.WriteLine("Weight: " + weight);
   }
```

```
}

class Student : Person
{
   private int level;
   private float avg;

   public Student() : base()
   {
      Console.Write("Enter Average: ");
      avg = float.Parse(Console.ReadLine());

      Console.Write("Enter Level: ");
      level = int.Parse(Console.ReadLine());
   }

   public override void Print()
   {
      base.Print();   // يستدعي دالة الأب
      Console.WriteLine("Average: " + avg);
      Console.WriteLine("Level: " + level);
   }
}

class Program
{
   static void Main()
   {
      Student s = new Student();
      s.Print();
   }
}
```

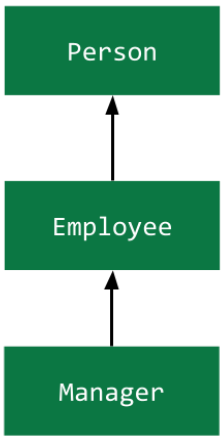
2. Multilevel Inheritance

In Multilevel Inheritance, a derived class will inherit a base class and as well as the derived class also act as the base class for another class.



In the above image, class A serves as a base class for the derived class B,
which serves as a base class for the derived class

The most general concept is considered a first-level base class; the next specific class is considered the base class for the more specific class; and so on. This process can be extended to any number of levels. To illustrate the idea, let's take the following example: Since every manager is an employee and every employee is a person, we can organize the three classes in multilevel inheritance, as Figure bellow explains, and is implemented in Program.

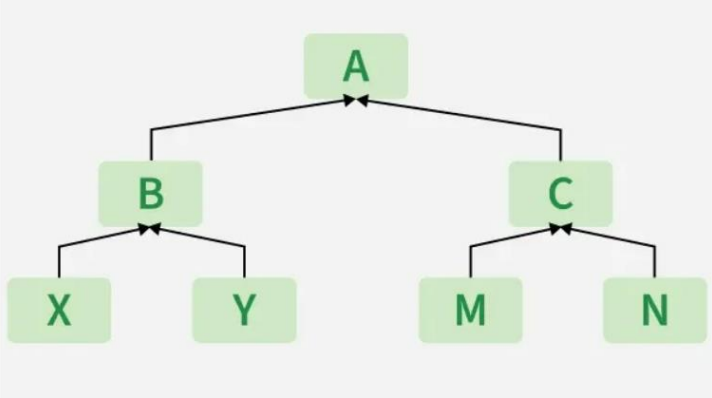


```
using System;

class Person
{
    private string name;
    private int age;

    public Person(string n, int a)
    {
        name = n;
        age = a;
    }

    public virtual void Print()
    {
        Console.WriteLine("Name: " + name);
        Console.WriteLine("Age: " + age);
    }
}

class Employee : Person
{
    private int salary;

    public Employee(string n, int a, int s) : base(n, a)
    {
        salary = s;
    }

    public override void Print()
    {
        base.Print();
        Console.WriteLine("Salary: " + salary);
    }
}

class Manager : Employee
{
    private string title;

    public Manager(string n, int a, int s, string t) : base(n, a, s)
    {
        title = t;
    }

    public override void Print()
    {
        base.Print();
        Console.WriteLine("Title: " + title);
```

```
    }
}

class Program
{
    static void Main()
    {
        Manager M = new Manager("Ali", 54, 7000, "HR Manager");
        M.Print();

        Console.WriteLine();

        Employee A = new Employee("Ahmed", 43, 5000);
        A.Print();

        Console.WriteLine();

        Person S = new Person("Saif", 34);
        S.Print();
    }
}
```

3. Hierarchical Inheritance

In Hierarchical Inheritance, one class serves as a superclass (base class) for more than one subclass.



In the above image, X and Y are sub-class (child class) that inherits property from class B and M and N are sub-class (child class) that inherits property from class C. B is the parent class of X and Y and C is the parent class of M and N.

In hierarchical inheritance, multiple derived classes inherit from a single base class.

For example, suppose we have an **Employee** class that contains common data members such as:

- `name`
- `age`
- `salary`

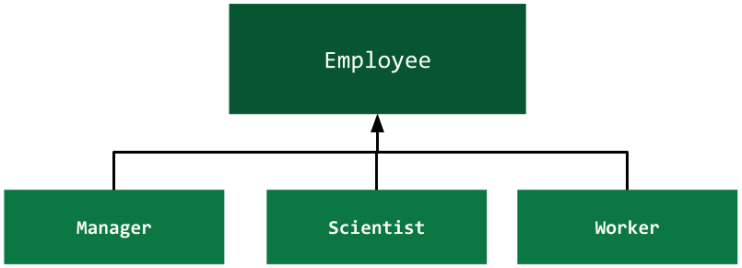
We can derive several classes from the **Employee** class, where each derived class may add its own specific data members.

- The **Manager** class inherits all data members from the `Employee` class and adds an additional data member:
  - `title` (string)
- The **Scientist** class also inherits from the `Employee` class and adds:
  - `publication` (string)
- The **Worker** class inherits from the `Employee` class but does not add any new data members of its own.

In this case, all derived classes share the common properties defined in the base class (`Employee`), while each derived class may extend the functionality by adding specialized attributes.

This type of inheritance is called **Hierarchical Inheritance**, because multiple classes are derived from a single base class.

The relationship among these classes is illustrated in Figure bellow:



```
using System;

class Emp
{
   private string name;
   private int age;
   private int salary;

   public Emp(string n, int a, int s)
   {
      name = n;
      age = a;
      salary = s;
   }

   public virtual void Print()
   {
      Console.WriteLine("Name: " + name);
      Console.WriteLine("Age: " + age);
      Console.WriteLine("Salary: " + salary);
   }
}

class Manager : Emp
{
   private string title;

   public Manager(string n, int a, int s, string t) : base(n, a, s)
   {
      title = t;
   }

   public override void Print()
   {
      base.Print();
      Console.WriteLine("Title: " + title);
   }
}

class Scientist : Emp
{
   private string publication;

   public Scientist(string n, int a, int s, string p) : base(n, a, s)
   {
      publication = p;
   }

   public override void Print()
   {
      base.Print();
      Console.WriteLine("Publication: " + publication);
   }
}

class Worker : Emp
```

```
{
   public Worker(string n, int a, int s) : base(n, a, s)
   {
   }

   public override void Print()
   {
      base.Print();
   }
}

class Program
{
   static void Main()
   {
      Manager M = new Manager("Ali", 54, 7000, "HR Manager");
      M.Print();

      Console.WriteLine();

      Scientist S = new Scientist("Ahmed", 43, 5000, "AI Tech.");
      S.Print();

      Console.WriteLine();

      Worker W = new Worker("Hasan", 22, 2000);
      W.Print();
   }
}
```