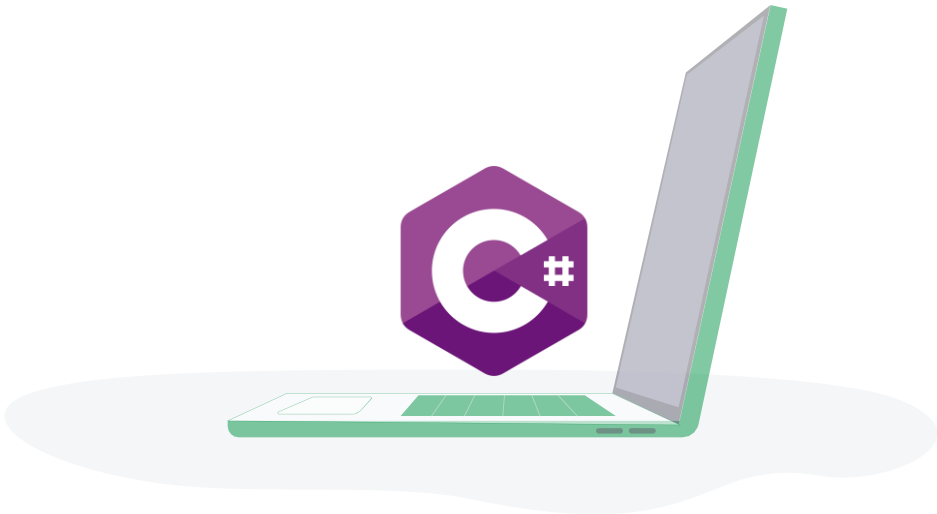
The Learning Hub ©

1. Inheritance Overview
2. Single Inheritance
3. Multilevel Inheritance
4. Hierarchical Inheritance
5. Sealed Class
6. Access Modifiers

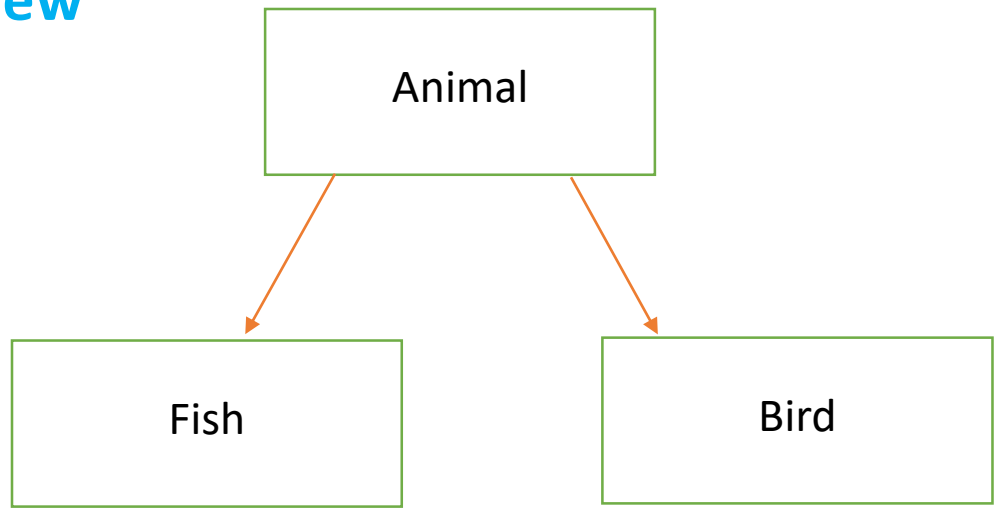



Inheritance Overview

## Inheritance Overview

One of the most primary characteristics in OOP is Inheritance which enables programmers not to repeat themselves while coding. As a result, using inheritance will reduce code redundancy which will reflect on whole program performance.

# Inheritance Overview



## Inheritance Benefits

- Code reusability.
- Extend and modify the defined behavior.
- Create more specialized behavior.

Note that C# and .Net support single inheritance only and we can declare multiple inheritance by using interfaces as will be covered in the next chapter.

## Points to Distinguish

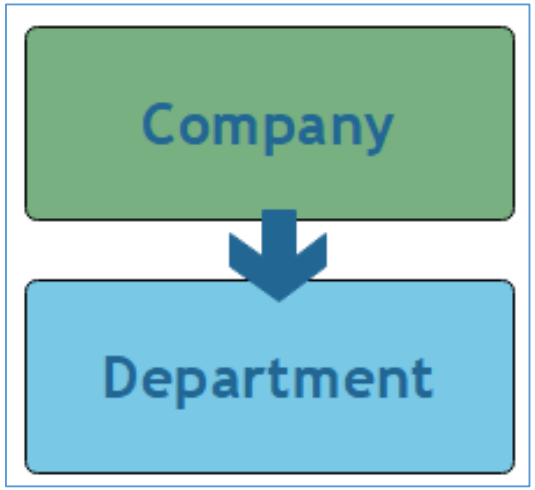
- The more generalized class which its members are inherited based on access modifier is called base class or parent.
- The more specialized class which inherits these members is called derived class or child.


Single Inheritance

## Single and Simple Inheritance

When there are a base class and one and only one derived class, this is what is meant by single inheritance.

Note that all members of base class will be inherited to derived class based on their access modifiers.



## Let’s Take an Example 

# Company class as base class

```
class Company
    {
        public int Id { get; set; }
        public string Name { get ; set; }
        public string Location { get; set; }
        public string Description { get; set; }
        public void CompanyInfo() =>
Console.WriteLine($"{Name} Company with {Id} id " +
                  $"which is located at{Location}
provide {Description}");
    }
```

# Department class as derived class

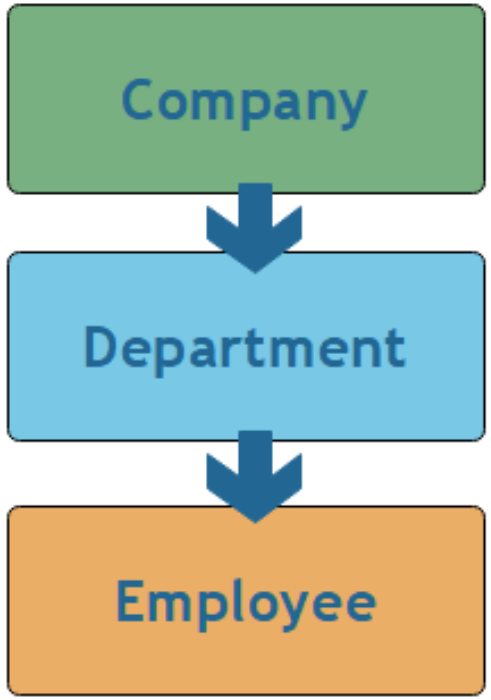
```
class Department:Company
    {
        public int DepartmentId { get; set; }
        public string DepartmentName { get; set; }
        public int NumberOfEmployees { get; set; }
        public string ManagerName { get; set; }

public void DepartmentInfo()=>
            Console.WriteLine($"{DepartmentName}
with {DepartmentId} id" +
                $" under {ManagerName} management
has {NumberOfEmployees} employees");
    }
```

# Multilevel Inheritance

## How can multilevel inheritance be achieved?

When a derived class inherits its members to another class, this is what is calling by multilevel inheritance.

- Here, the members of base class and the first derived class are inherited to the second derived class.



## Let’s Take an Example 

# Employee class inherits Department

```
class Employee:Department
    {
        public int EmployeeId { get; set; }
        public string EmployeeName { get; set; }
        public double Salary { get; set; }
        public double Age { get; set; }
        public void EmployeeInfo()
        {
Console.WriteLine($"{EmployeeName} with {EmployeeId}
id" +
                $" is {Age} years old works at
{DepartmentName} in {Name} company" +
                $" and receives {Salary} JOD ");
        }
    }
```

## Call base class constructor

Programmers can call base class constructor using base keyword, which sometimes will be optional if the base class has overloading constructors, and sometimes will be mandatory when it has one constructor with parameters.

Syntax public Derived(): base()

# Call base class constructor

```
class Company
    {
        public Company(int id)
        {
            Id= id;
        }
    }
class Department: Company
    {
        public Department(int companyId,int depId) :
base(companyId)
        {
            DepartmentId = depId;
        }
    }
```

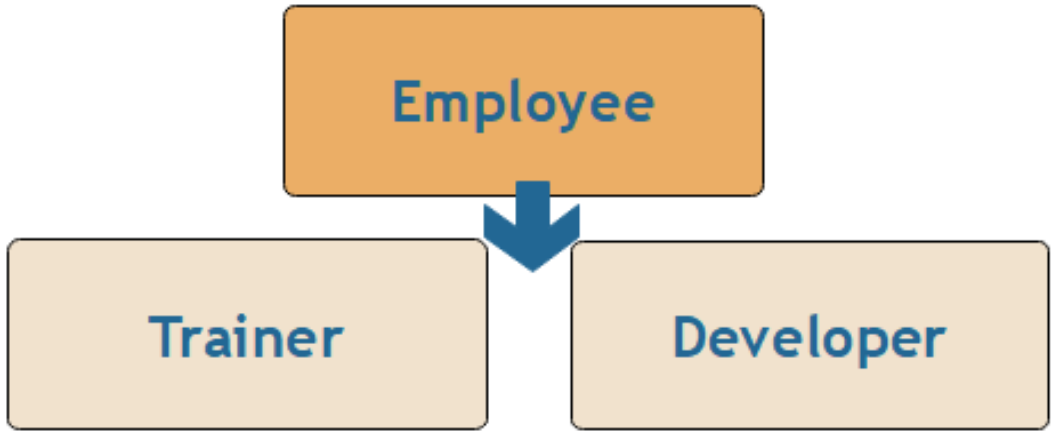
# Continue..

```
class Employee:Department
    {
        public Employee(int companyId,int depId,int
empId) : base(companyId, depId)
        {
            EmployeeId = empId;
        }
    }
```


Hierarchal Inheritance

## Hierarchical Inheritance

When some features or members are needed to be inherited to more than one derived class, programmers use hierarchical inheritance.



## Let’s Take an Example 

# Employee class as base class

```
class Employee
    {
        public int EmployeeId { get; set; }
        public string EmployeeName { get; set; }
        public double Salary { get; set; }
        public double Age { get; set; }
        public void EmployeeInfo()
        {
Console.WriteLine($"{EmployeeName} with {EmployeeId}
id" +
                $" is {Age} years old recieves
{Salary} JOD");
        }
    }
```

# Trainer class inherits Employee class

```
class Trainer : Employee
   {
        public string CourseName { get; set; }
        public int NumberOfStudents { get; set; }
        public DateTime StartDate { get; set; }
        public DateTime EndDate { get; set; }
        public void TrainerInfo(){
        Console.WriteLine($"{EmployeeName} presents
{CourseName}\n " +
                  $" to {NumberOfStudents} students
which starts at\n" +
                  $" {StartDate} and completes at
{EndDate}");
        }
   }
```

# Developer class inherits Employee class

```
class Developer : Employee
    {
        public string Specialty { get; set; }
        public string Position { get; set; }
        public void DeveloperInfo()
        {
            Console.WriteLine($"{EmployeeName} is
working in {Specialty} " +
                $" as {position} ");
        }
    }
```

# Main

```
Trainer trainer = new Trainer();
trainer.CourseName = "OOP";
trainer.NumberOfStudents = 33;
DateTime date = new DateTime(2022, 10, 10);
TimeSpan duration = new TimeSpan(8,0,0,0);
trainer.StartDate = date;
trainer.EndDate = date.Add(duration);
trainer.TrainerInfo();
```


Sealed Class

## What if programmers want to prevent inheritance?

In some cases, programmers need to make some classes non-heritable, they can accomplish this by declaring classes as sealed ones.

Note that programmers still can declare an object from sealed class.

## Let’s Take an Example 

# Sponsors class

```
sealed class Sponsors
    {
        public string Id { get; set; }
        public string Name { get; set; }
        public void SponsorInfo()
        {
            Console.WriteLine($"{Name} with id {Id});
        }
    }
//in main
Sponsors sponsor = new Sponsors();
sponsor.Id = 1;
sponsor.Name = "Sir";
sponsor.SponsorInfo();
```


Access Modifier

## Overview of Access Modifier

By declaring access modifier, we determine the visibility and accessibility level of class and its members from outside the class in the same assembly or from another one references it, note that all types and members have an accessibility level in program.

## Access modifiers

Public members can be accessed from whole program in the same assembly or another assembly that references it.

Private members can be accessed just by the owner class or struct itself.

Internal members can be accessed by the owner class surely and within current assembly in derived classes or as instances from.

## Access modifiers

Protected members can be accessed by the owner class and from within any derived classes in the same assembly or in another assembly that references it.

Private Protected members can be accessed by the owner class and from within derived classes in the same assembly.

## Access modifiers

Protected Internal members combine the accessibility level from both internal and protected access modifiers. Which mean they can be accessed by the owner class and within the same assembly in derived classes or as instances from. Additionally, they can be accessed from within derived classes in another assembly.

## Inherited Members

What about access modifiers of members in base class? As it mentioned in the previous slide, derived class inherits the base class members based on these access modifiers.

- Now, let’s add some members in our base class with all explained access modifiers and see the differences.

Let’s make some changes 

# Add these properties in Company class

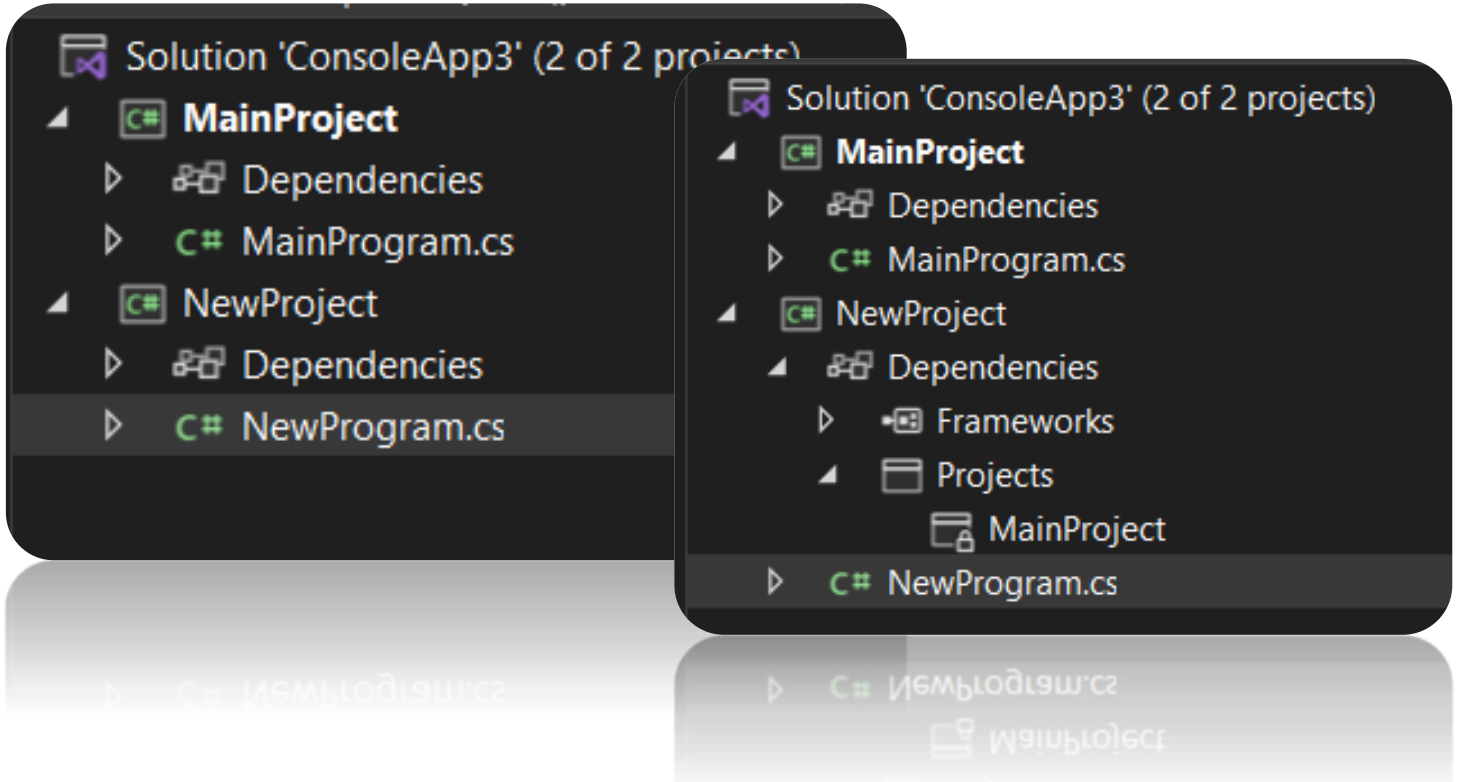
```
class Company
    {    ……….……
         private double SerialNumber { get; set; }
         internal int NumberOfEmployees{ get; set; }
         protected string Owner { get; set; }
         protected internal string Sponsor{ get; set;}
         private protected double Cost { get; set; }
         ……….……
    }
```

## Add New Assembly

After creating a new assembly, make a reference between them and add derived class from Company class and see the inherited members based on access modifiers in these two situations:

- In derived class.
- When an instance is created from company class.

# Our Two Projects



# Add these properties in Company class

```
public class Company1:Company
    {
        public Company1()
        {
            Id = 222;
            Name = "Harmony IT Solutions";
            Owner = "Sir";
            Sponsor = "Sir";
        }
```

# Continue..

```
public void PrintInheritedMembers()
        {
            Console.WriteLine($"{base.Id}-
{base.Name}: the owner is {base.Owner} " +
                $"and it is sponsored by
{base.Sponsor}");
        }
    }
```