



**Inheritance**



Harsha Vardhan

- > UI Expert
- > .NET Expert
- > Module Lead
- > Corporate Instructor

 Harsha  
Web University

### Agenda

- 1 Understanding Inheritance  
What is Inheritance and why to use it.
- 2 Types of inheritance  
Single inheritance, Multiple inheritance, Multi-level inheritance, Hierarchical inheritance and Hybrid inheritance
- 3 'base' keyword  
Accessing parent class's members in child class.
- 4 Parent class's constructor  
Calling constructor of parent class, from child class.

### Agenda

- 5 Method Hiding  
Hiding parent class's method using 'new' keyword.
- 6 Method Overriding  
Using 'virtual' keyword at parent class's method; 'override' keyword in child class's method.

Introducing Inheritance → Next: Syntax of Inheritance → Harsha

**Goal**

- > This allows classes to be arranged in a hierarchy that represents "is-a-type-of" relationships.
- > The "parent class" acts as a "base type" of "one or more child classes".
- > Child classes are derived from parent class.

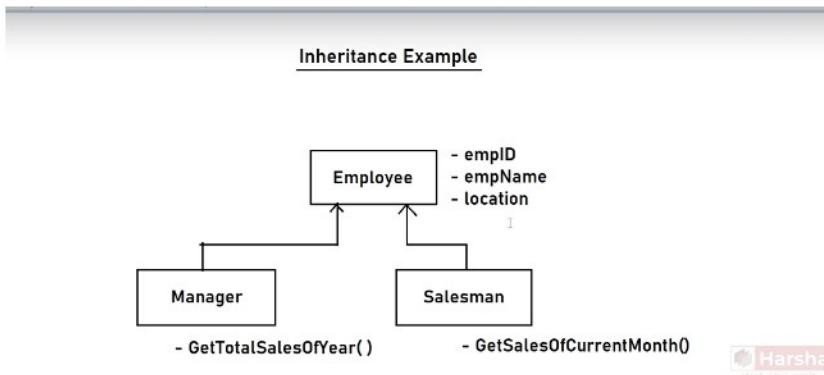
|  |  |
|--|--|
|  | <p><b>Parent Class</b></p> <ul style="list-style-type: none"> <li>- Parent class's fields</li> <li>- Parent class's methods</li> </ul> <p style="text-align: center;">↓</p> <p><b>Child Class</b></p> <ul style="list-style-type: none"> <li>- Child class's fields</li> <li>- Child class's methods</li> </ul> <p style="border: 1px solid black; padding: 2px; margin-top: 5px;">[also can access parent class's fields and methods]</p> |
|--|--|

**Introducing Inheritance** → **Next: Types of Inheritance**

|  |   |
|--|---|
| <b>What</b> <ul style="list-style-type: none"> <li>Concept of extending the parent class, by creating child class.</li> <li>"Child class" extends "parent class".</li> </ul>  <ul style="list-style-type: none"> <li>The child class's object stores members of both child class and parent class.</li> </ul> | <b>Creating Parent Class</b> <pre>class ParentClassName {     Parent Class Members here }</pre><br><b>Creating Child Class</b> <pre>class ChildClassName : ParentClassName {     Child Class Members here }</pre> |
|--|---|

**Object of Child Class**

|                       |                      |
|-----------------------|----------------------|
| Parent Class's Fields | Child Class's Fields |
|-----------------------|----------------------|



**Agenda**

- 1 Understanding Inheritance**  
What is Inheritance and why to use it.
- 2 Types of inheritance**  
Single inheritance, Multiple inheritance, Multi-level inheritance, Hierarchical inheritance and Hybrid inheritance
- 3 'base' keyword**  
Accessing parent class's members in child class.
- 4 Parent class's constructor**  
Calling constructor of parent class, from child class.

**Types of Inheritance** → **Next: 'base' keyword**

|  |   |
|--|---|
| <b>I. Single Inheritance</b> <pre>class ParentClassName { } class ChildClassName : ParentClassName { }</pre> <p>One Parent Class, One Child Class.</p> | <b>2. Multi-Level Inheritance</b> <pre>class ParentClassName { } class ChildClass1 : ParentClassName { } class ChildClass2 : ChildClass1 { }</pre> <p>One Parent Class, One Child Class; and the Child class has another Child class.</p> |
|--|---|

**Types of Inheritance** → **Next: 'base' keyword**

|  |  |
|--|--|
| <b>3. Hierarchical Inheritance</b> <pre>class ParentClassName { } class ChildClass1 : ParentClassName { } class ChildClass2 : ParentClassName { }</pre> <p>One Parent Class, Multiple Child Classes.</p> | <b>4. Multiple Inheritance</b> <pre>class ParentClass1 { } class ParentClass2 { } class ChildClass : ParentClass1, ParentClass2 { }</pre> <p>Multiple Parent Classes, One Child class.</p> |
|--|--|

C# does support Multiple Inheritance directly, we can achieve multiple inheritance by using interfaces.

Types of Inheritance      Next: 'base' keyword

5. Hybrid Inheritance

```
class ParentClassName
{
}

class ChildClass1 : ParentClassName
{
}

class ChildClass2 : ChildClass1
{
}

class ChildClass3 : ChildClass1
```

Hierarchical Inheritance + Multi Level Inheritance

Harsha Web University

Agenda

- 1 Understanding Inheritance  
What is Inheritance and why to use it.
- 2 Types of inheritance  
Single inheritance, Multiple inheritance, Multi-level inheritance, Hierarchical inheritance and Hybrid inheritance
- 3 'base' keyword  
Accessing parent class's members in child class.
- 4 Parent class's constructor

'base' keyword      Next: Parent Class's Constructor

What

- > The "base" keyword represents parent class's members in the child class.
- > It is optional to use, by default.
- > It is must to use, when there is "name ambiguity" between parent class's member and child class's member.

base

Object of Child Class

Parent Class's Members    Child Class's Members

Harsha Web University

Agenda

- 1 Understanding Inheritance  
What is Inheritance and why to use it.
- 2 Types of inheritance  
Single inheritance, Multiple inheritance, Multi-level inheritance, Hierarchical inheritance and Hybrid inheritance
- 3 'base' keyword  
Accessing parent class's members in child class.
- 4 Parent class's constructor  
Calling constructor of parent class, from child class.

Harsha Web University

By Default, when you write the statement 'new childClassName()' to create object of the child class, by default it will directly call the child class's constructor only.

Parent Class's Constructor      Next: Method Hiding

Creating Parent Class

```
class ParentClassName
{
    public ParentClassName( param1, ... )
    {
    }
}
```

Creating Child Class

```
class ChildClassName : ParentClassName
{
```

Harsha Web University

### Creating Child Class

```
class ChildClassName : ParentClassName
{
    public ChildClassName( ... ): ParentClassName(arg1, arg2, ...)
    {
    }
}
```

Harsha  
Web University

It's optional to call 'parent Class' parameter-less constructor from 'Child Class'. Because the parameterless constructor of the parent class will be called automatically while creating object of the child class.

### Parent Class's Constructor



### Next: Method Hiding

- It is OPTIONAL to call "Parent Class's Parameter-less Constructor" from "Child Class".
- It is MUST to call "Parent Class's Parameterized Constructor" from "Child Class" and pass necessary arguments.

### Agenda

- 5** Method Hiding  
Hiding parent class's method using 'new' keyword.
- 6** Method Overriding  
Using 'virtual' keyword at parent class's method; 'override' keyword in child class's method.

Harsha  
Web University

### Method Hiding

### What

- It is a concept, which is used to hide (overwrite) the parent class's method, by creating another method in the child class with same name and same parameters.

### Creating Parent Class

```
class ParentClassName
{
    public void MethodName( param1, ... )
    {
    }
}
```

### Creating Child Class

```
class ChildClassName : ParentClassName
{
    public new void MethodName( param1, ... )
    {
    }
}
```

Harsha  
Web University

## Method Hiding

## Next: Method Overriding

### What

- It is a concept, which is used to hide (overwrite) the parent class's method, by creating another method in the child class with same name and same parameters.

**Creating Parent Class**

```
class ParentClassName
{
    public void MethodName( param1, ... )
    {
    }
}
```

**Creating Child Class**

```
class ChildClassName : ParentClassName
{
    public new void MethodName( param1, ... )
    {
    }
}
```

## Method Hiding

## Next: Method Overriding



- When method hiding is done, if the method is called using child class's object; the child class's method only executes; parent class's method will not be executed.
- Method hiding is done automatically, but is recommended to use "new" keyword (but not must).

## Agenda

5

### Method Hiding

Hiding parent class's method using 'new' keyword.

6

### Method Overriding

Using 'virtual' keyword at parent class's method; 'override' keyword in child class's method.

## Method Overriding

### What

- It is a concept, which is used to extend the parent class's method, by creating another method in the child class with same name and same parameters.

**Creating Parent Class**

```
class ParentClassName
{
    public virtual void MethodName( param1, ... )
    {
    }
    1
}
```

**Creating Child Class**

```
class ChildClassName : ParentClassName
{
    public override void MethodName( param1, ... )
    {
        base.MethodName();
    }
    2
}
```

In case of **Method hiding**, you will override or shadow the parent class's method. That means in that case, the parent class's method will not execute.

But in case of **Method overriding**, both parent class's method and child class's method also will execute.

The 'virtual' keyword in the parent class's method says that, this parent method can be overridden in the child classes if required.

The 'override' keyword says that, this child class's method is overriding the parent class's method. That means, extending the

The 'virtual' keyword in the parent class's method says that, this parent method can be overridden in the child classes if required. 

The 'override' keyword says that, this child class's method is overriding the parent class's method. That means, extending the parent class's method.

## Method Overriding



- When method overriding is done, if the method is called using child class's object; the parent class's method first and child's method executed next.
- Method Overriding is done with "virtual" keyword at parent class; and "override" keyword at child class's method.
- The parent class's method invoked using "base" keyword.
- Without 'virtual' keyword are parent class's method; the child class's method can't be 'override'.



Harsha Vardhan

- UI Expert
- .NET Expert
- Module Lead
- Corporate Instructor

## Sealed Classes & Sealed Methods



### Sealed Classes

Next: Comparison Table

**What**

- Sealed class is a class, which is instantiable; but not inheritable.
- Use sealed class, whenever you don't want to let other developers to create child classes for the specific class.

Sealed Class

```
sealed class Class1
{
}

class Class2 : Class1 //not possible
```

### Comparison Table: Class (vs) Sealed Class

Next: Sealed Methods

**Based on inheritance and object**

| Class Type     | Can Inherit from Other Classes | Can Inherit from Other Interfaces | Can be Inherited | Can be Instantiated |
|----------------|--------------------------------|-----------------------------------|------------------|---------------------|
| Normal Class   | Yes                            | Yes                               | Yes              | Yes                 |
| Abstract Class | Yes                            | Yes                               | Yes              | No                  |
| Sealed Class   | Yes                            | Yes                               | No               | Yes                 |



## Comparison Table: Class (vs) Sealed Class

Next: Sealed Methods

| Type           | 1. Non-Static Fields | 2. Non-Static Methods | 3. Non-Static Constructors | 4. Non-Static Properties | 5. Non-Static Events | 6. Non-Static Destructors | 7. Constants |
|----------------|----------------------|-----------------------|----------------------------|--------------------------|----------------------|---------------------------|--------------|
| Normal Class   | Yes                  | Yes                   | Yes                        | Yes                      | Yes                  | Yes                       | Yes          |
| Abstract Class | Yes                  | Yes                   | Yes                        | Yes                      | Yes                  | Yes                       | Yes          |
| Sealed Class   | Yes                  | Yes                   | Yes                        | Yes                      | Yes                  | Yes                       | Yes          |

| Type           | 8. Static Fields | 9. Static Methods | 10. Static Constructors | 11. Static Properties | 12. Static Events | 13. Virtual Methods | 14. Abstract Methods | 15. Non-Static Auto-Impl Properties | 16. Non-Static Indexers |
|----------------|------------------|-------------------|-------------------------|-----------------------|-------------------|---------------------|----------------------|-------------------------------------|-------------------------|
| Normal Class   | Yes              | Yes               | Yes                     | Yes                   | Yes               | Yes                 | No                   | Yes                                 | Yes                     |
| Abstract Class | Yes              | Yes               | Yes                     | Yes                   | Yes               | Yes                 | Yes                  | Yes                                 | Yes                     |
| Sealed Class   | Yes              | Yes               | Yes                     | Yes                   | Yes               | No                  | No                   | Yes                                 | Yes                     |

## Sealed Methods

### What

- Sealed Methods must be "override methods", which can't be overridden in the corresponding child classes.

### Purpose

- Use sealed methods to prevent overriding that particular methods in the corresponding child classes.

Harsha

## Sealed Methods

### Creating Parent Class

```
class ParentClassName
{
    public virtual void MethodName( param1, ... )
    {
    }
}
```

1

### Creating Child Class

```
class ChildClass1 : ParentClassName
{
    public sealed override void MethodName()
    {
    }
}
```

2

### Creating Child Class 2

```
class ChildClass2 : ChildClass1
{
    public override void MethodName() //Doesn't compile!
    {
    }
}
```

3

Harsha  
Web University

You can not apply the sealed method for normal methods.

Sealed Method is the opposite of the Virtual Method. this method can't be overriden in the corresponding child classes.