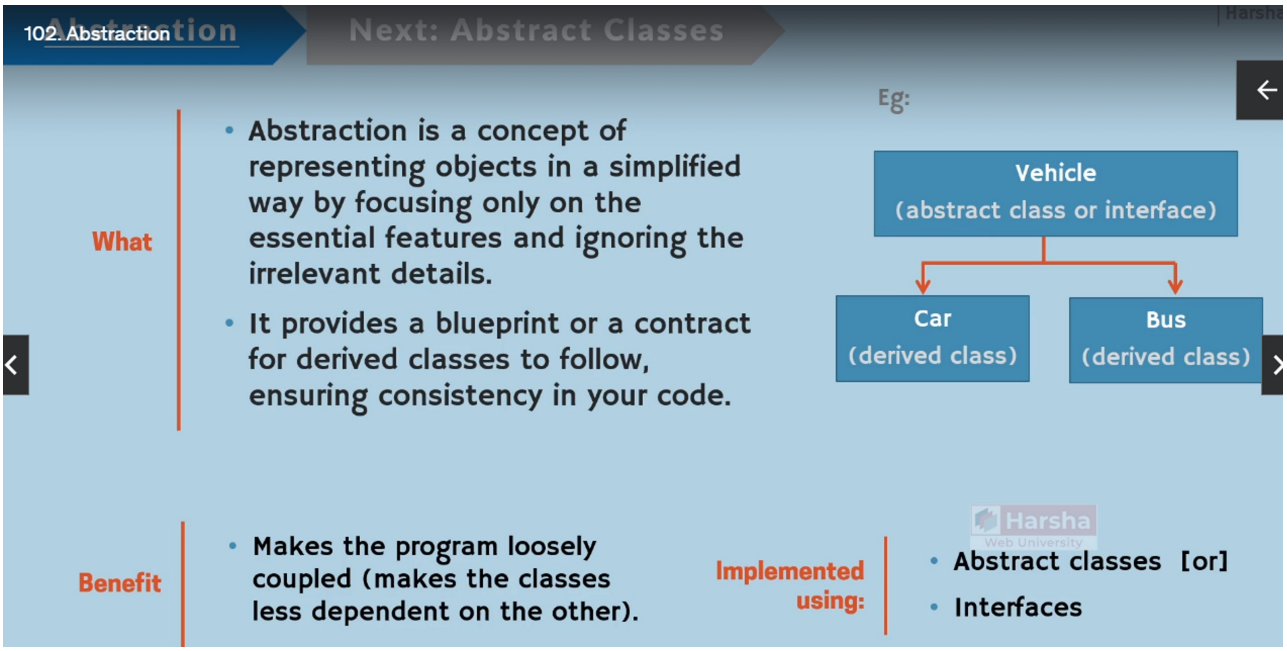
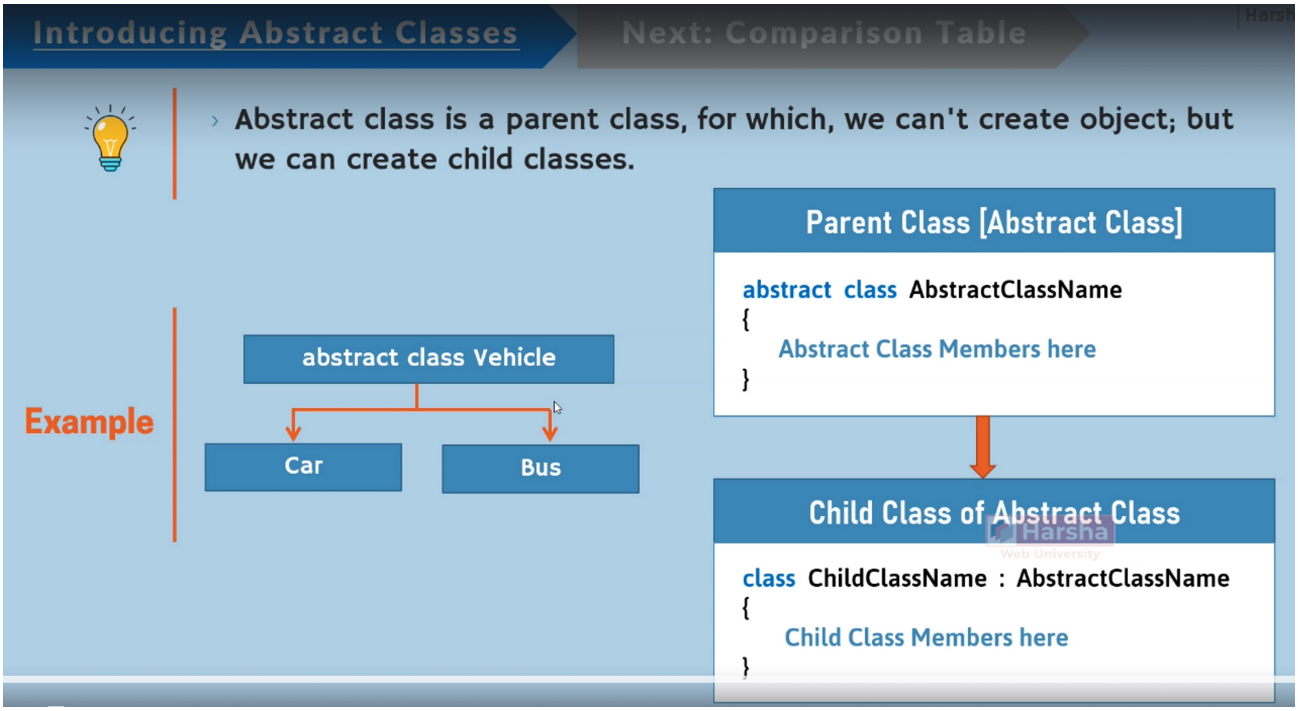
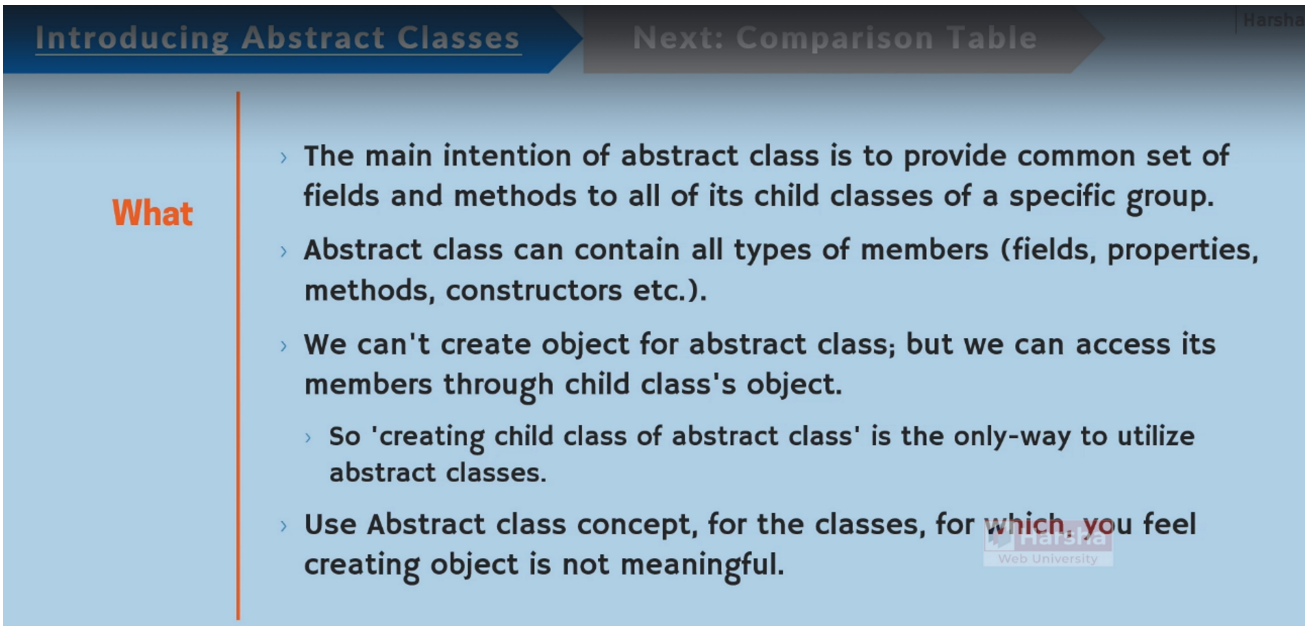
Section 11: Abstract Classes and Interfaces

13 July 2024 11:00





You can't create object of abstract class. You can create object of child class and thorough the object of child class, only you can access any member of the abstract class.



Comparison Table: Class (vs) Abstract Class

Next: Abstract 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

Comparison Table: Class (vs) Abstract Class

Next: Abstract Methods

Based on members:

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

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

Introducing Abstract Methods

What

- Abstract methods are declared in parent class, with "abstract" keyword; implemented in child classes, with "override" keyword.

When

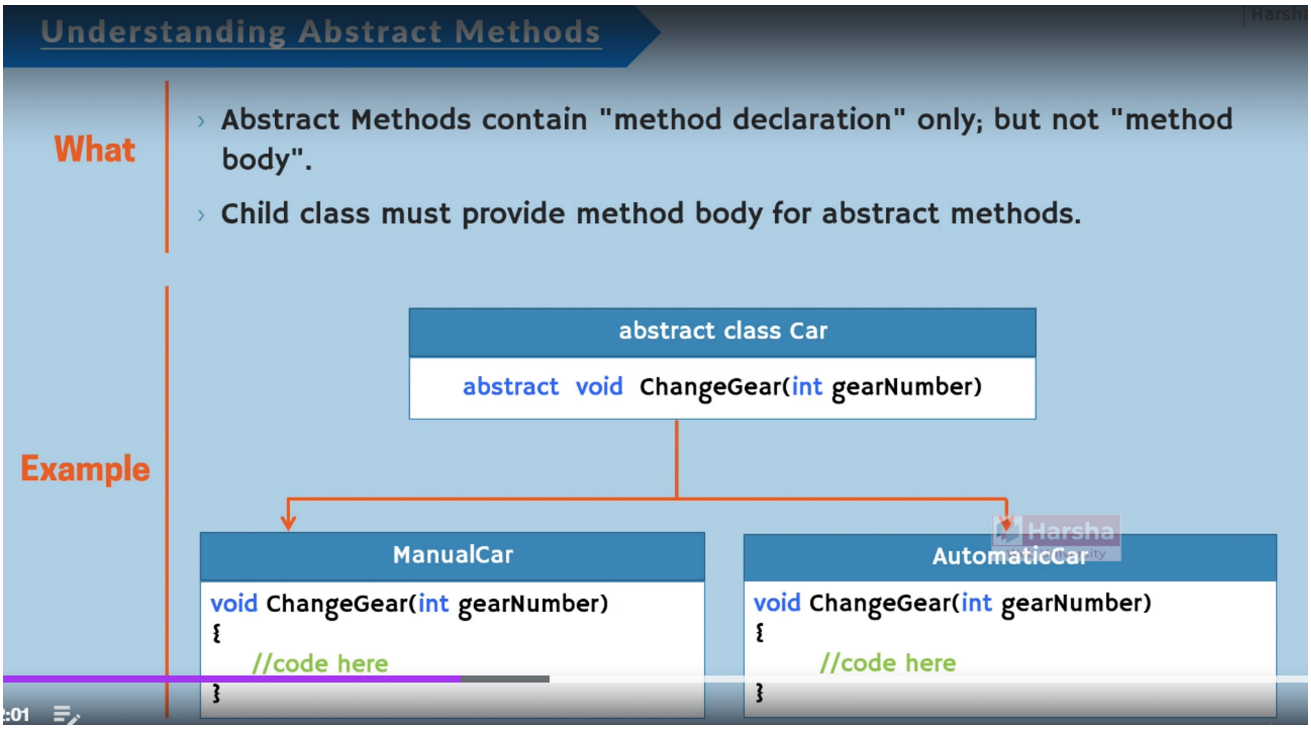
- When the parent class don't want to provide the definition of a method;
  - it wants to let child classes to implement the method.

Parent Class [Abstract Class]

```
abstract class AbstractClassName
{
    AccessModifier abstract ReturnDataType MethodName(param1, ...);
}
```

Child Class of Abstract Class

```
class ChildClassName : AbstractClassName
{
    AccessModifier override ReturnDataType MethodName(param1, ...)
    {
    }
}
```



From Child Class point of view, it is optional to override 'virtual method'. but it is mandatory to override abstract method.

Agenda

1. **Understanding Interfaces**
   What is interface and its features.
2. **Dynamic Polymorphism**
   What is Polymorphism and different types of polymorphism.
   Achieving dynamic polymorphism using interfaces.
3. **Multiple Inheritance**
   Creating child classes with multiple parent interfaces.
4. **Interface Inheritance**
   Creating child interface with one or more parent interfaces.

Agenda

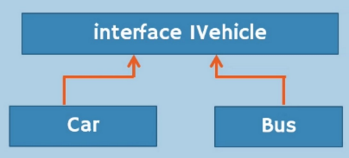
5. **Explicit Interface Implementation**
   Creating two methods with same signature in two different interfaces and implementing both interfaces in the same child class.

Introducing Interfaces

Next: Comparison Table

- Interface is a set of abstract methods, that must be implemented by the child classes.

Example



Interface

```
interface InterfaceName
{
    ReturnDataType MethodName(param1, ...);
}
```

Child Class of Interface

```
class ChildClassName : InterfaceName
{
    public ReturnDataType MethodName(param1, ...)
    {
    }
}
```

Introducing Interfaces

Next: Comparison Table

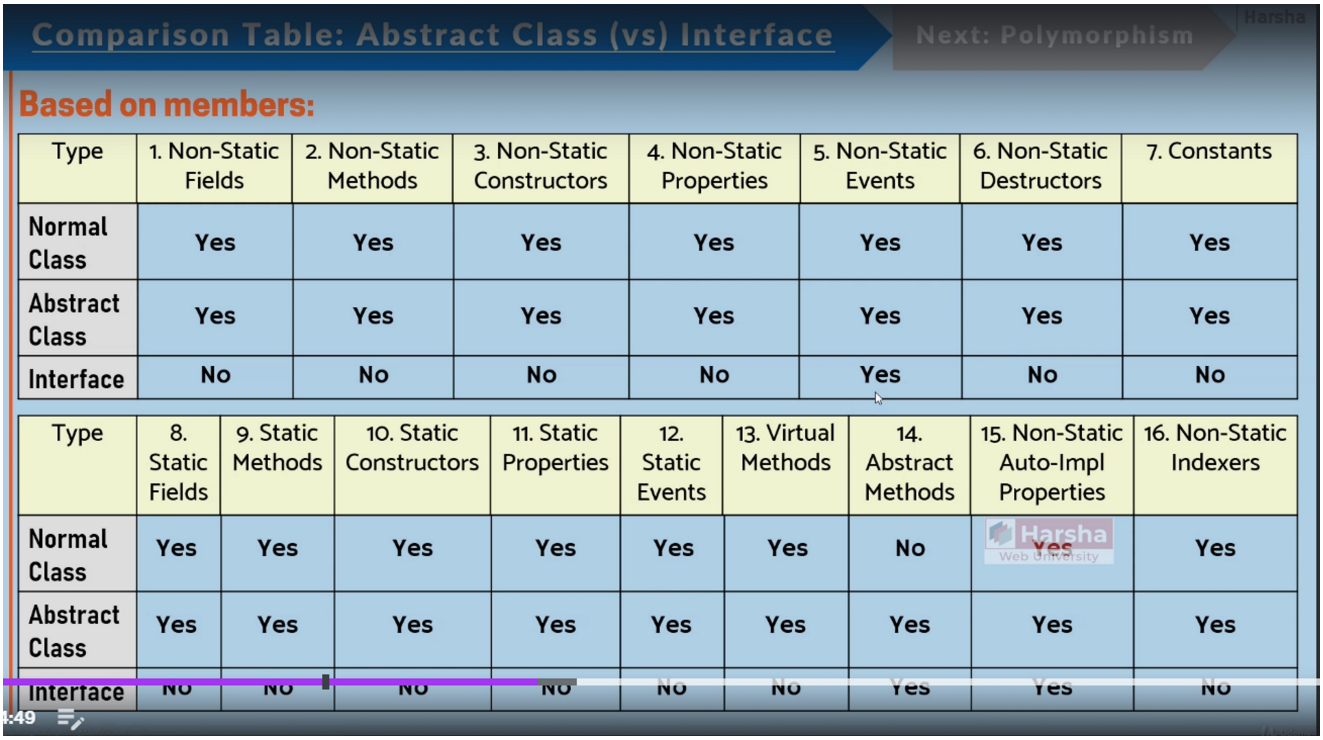
- The child class that implements the interface, MUST implement ALL METHODS of the interface.
- Interface methods are by default "public" and "abstract".
- The child class must implement all interface methods, with same signature.
- You can't create object for interface.
- You can create reference variable for the interface.
- The reference variable of interface type can only store the address of objects of any one of the corresponding child classes.
- You can implement multiple interfaces in the same child class [Multiple Inheritance].
- An interface can be child of another interface.

Comparison Table: Abstract Class (vs) Interface

Next: Polymorphism

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
Interface	No	Yes	Yes	No



Comparison Table: Abstract Class (vs) Interface

Next: Polymorphism

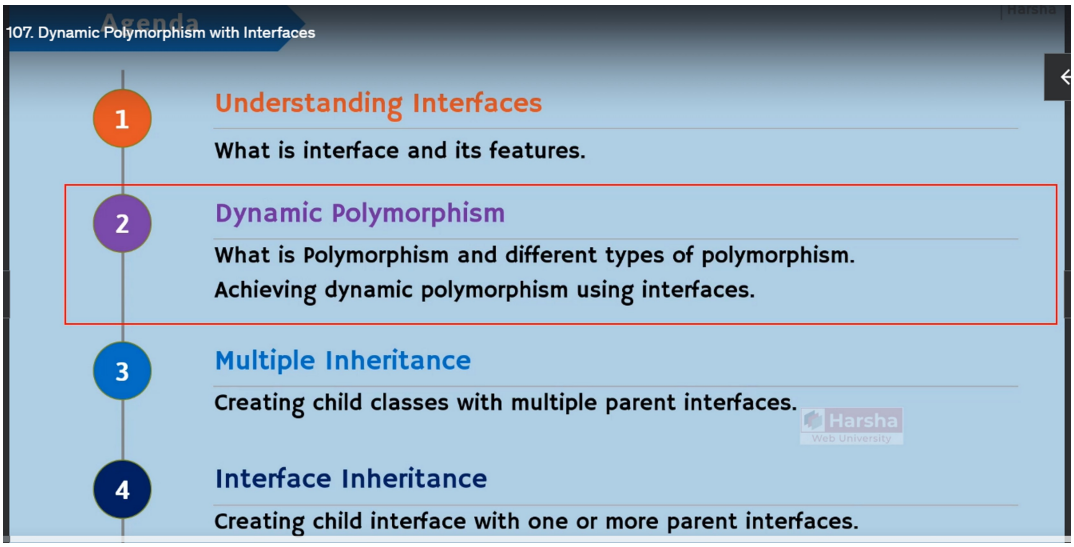
Based on members:

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
Interface	No	No	No	No	Yes	No	No

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
Interface	No	No	No	No	No	No	Yes	Yes	No

When exactly you will go for interface and when exactly you will go for abstract class?

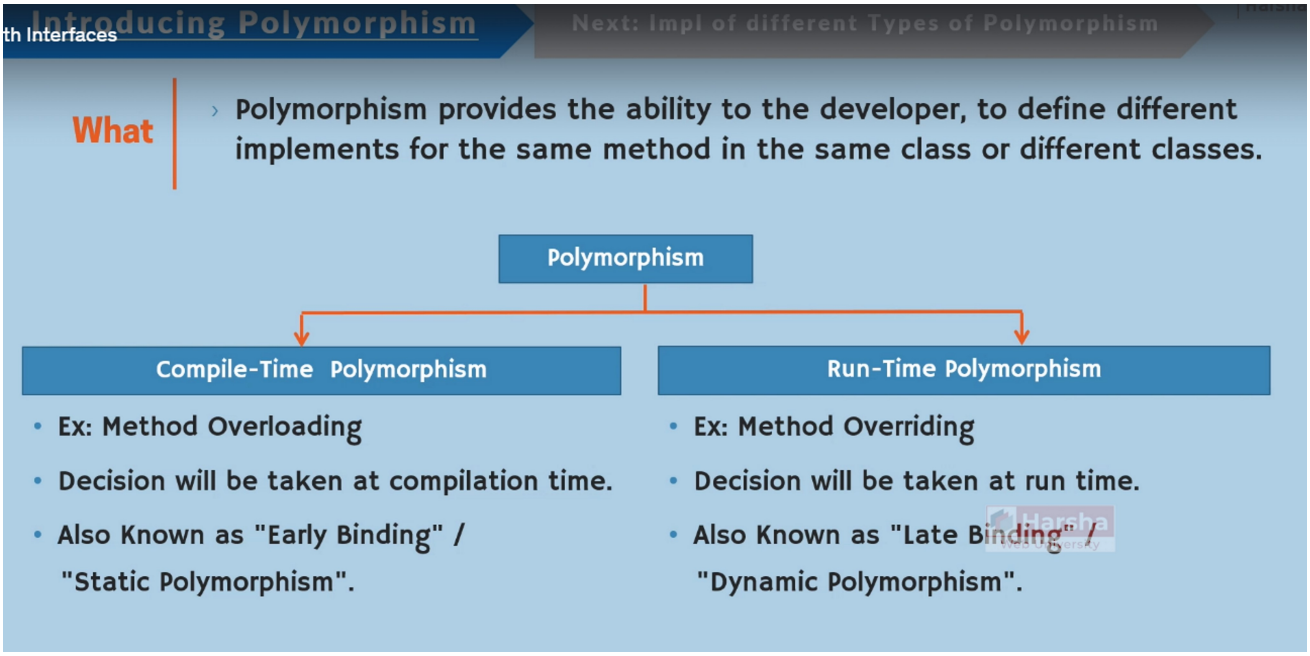
Answer: whenever you want to define some methods as abstract methods and remaining methods as non-abstract methods => go for abstract class.

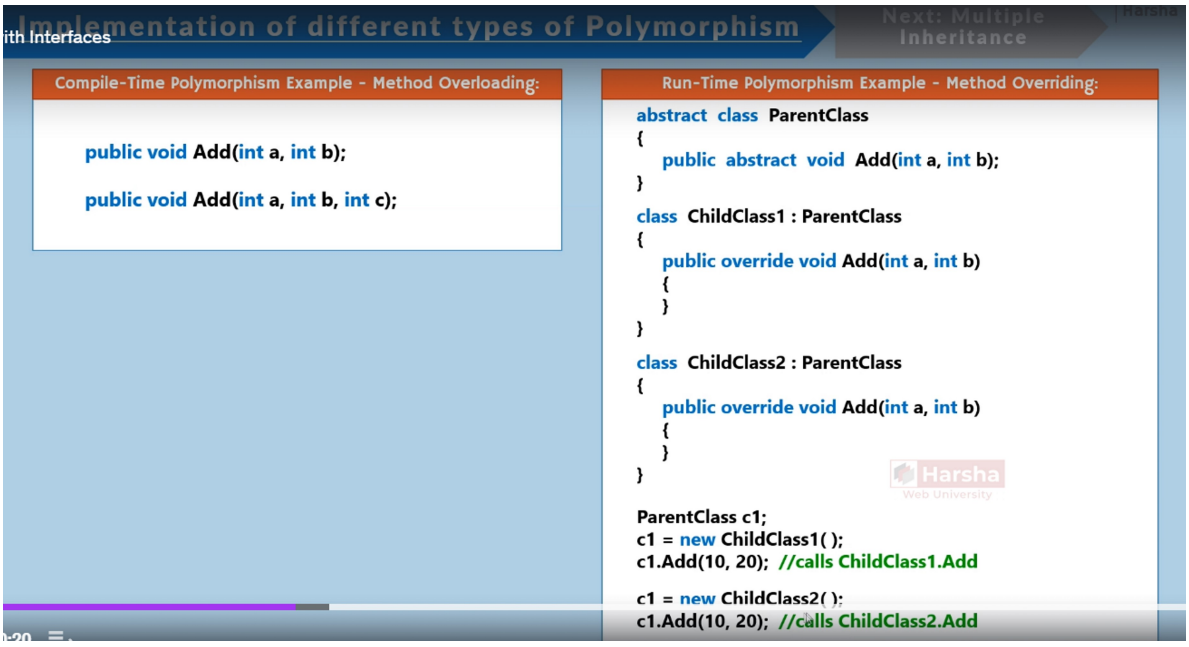


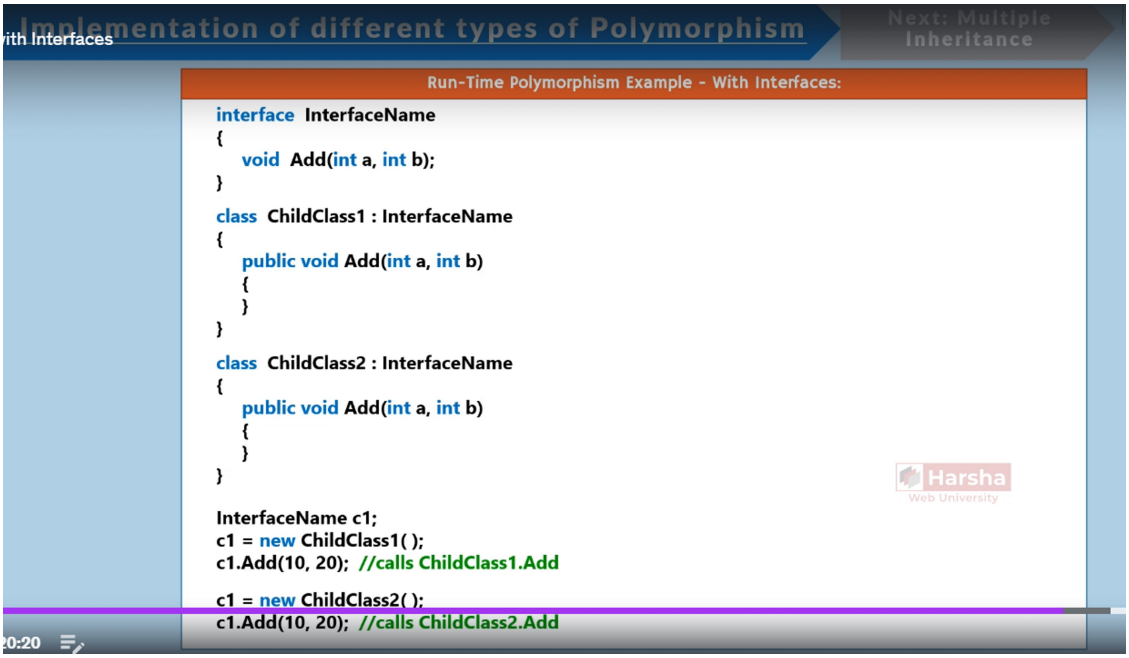
Agenda

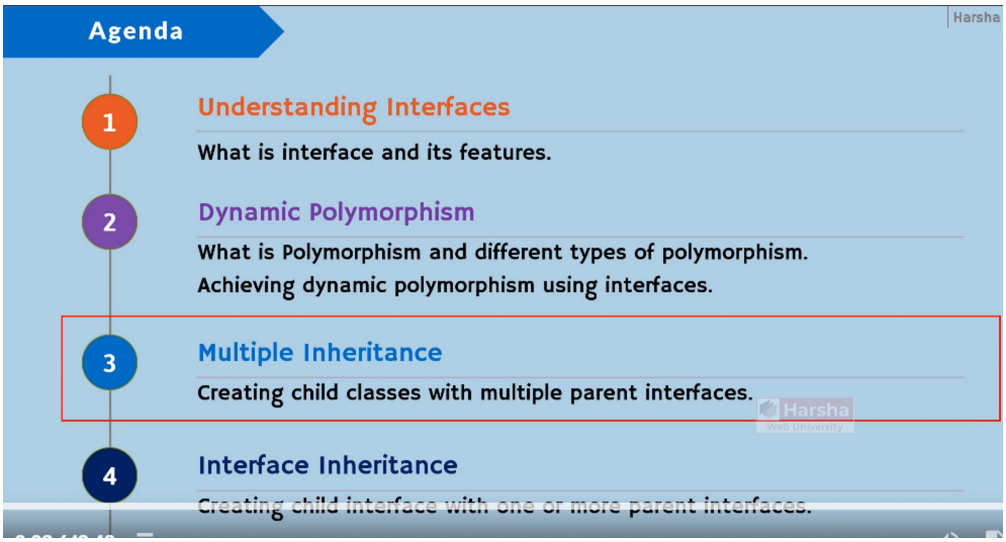
107. Dynamic Polymorphism with Interfaces

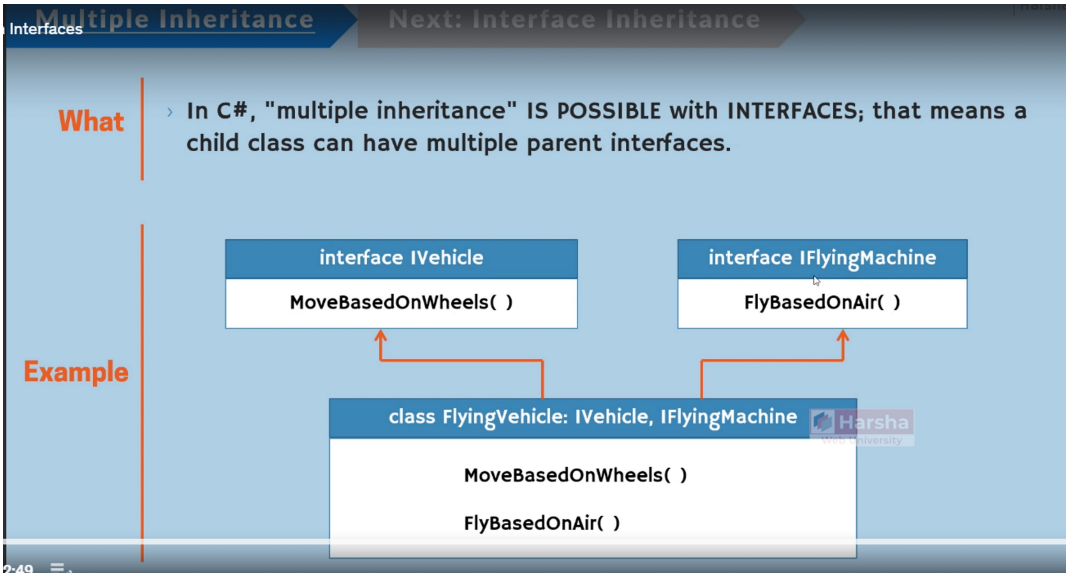
1. **Understanding Interfaces**
   What is interface and its features.
2. **Dynamic Polymorphism**
   What is Polymorphism and different types of polymorphism.
   Achieving dynamic polymorphism using interfaces.
3. **Multiple Inheritance**
   Creating child classes with multiple parent interfaces.
4. **Interface Inheritance**
   Creating child interface with one or more parent interfaces.


th Interfaces
Introducing Polymorphism
Next: Impl of different Types of Polymorphism
What
Polymorphism provides the ability to the developer, to define different implements for the same method in the same class or different classes.
Polymorphism
Compile-Time Polymorphism
Ex: Method Overloading
Decision will be taken at compilation time.
Also Known as "Early Binding" / "Static Polymorphism".
Run-Time Polymorphism
Ex: Method Overriding
Decision will be taken at run time.
Also Known as "Late Binding" / "Dynamic Polymorphism".

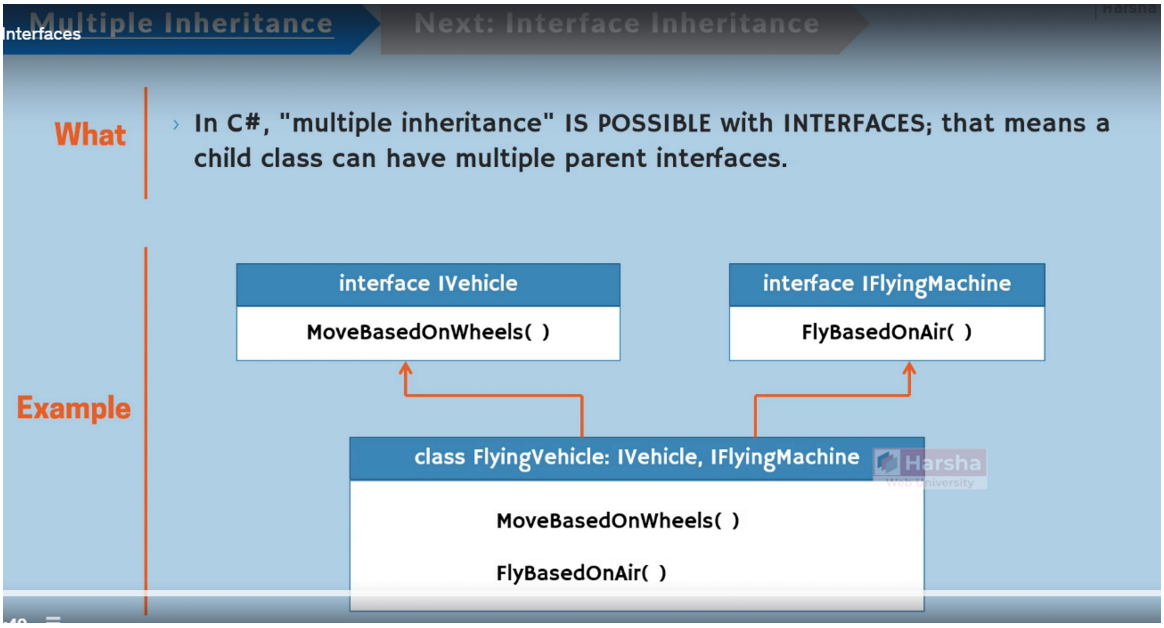

ith Interfaces
Implementation of different types of Polymorphism
Next: Multiple Inheritance
Compile-Time Polymorphism Example - Method Overloading:
public void Add(int a, int b);
public void Add(int a, int b, int c);
Run-Time Polymorphism Example - Method Overriding:
abstract class ParentClass
{
public abstract void Add(int a, int b);
}
class ChildClass1 : ParentClass
{
public override void Add(int a, int b)
{
}
}
class ChildClass2 : ParentClass
{
public override void Add(int a, int b)
{
}
}
ParentClass c1;
c1 = new ChildClass1( );
c1.Add(10, 20); //calls ChildClass1.Add
c1 = new ChildClass2( );
c1.Add(10, 20); //calls ChildClass2.Add

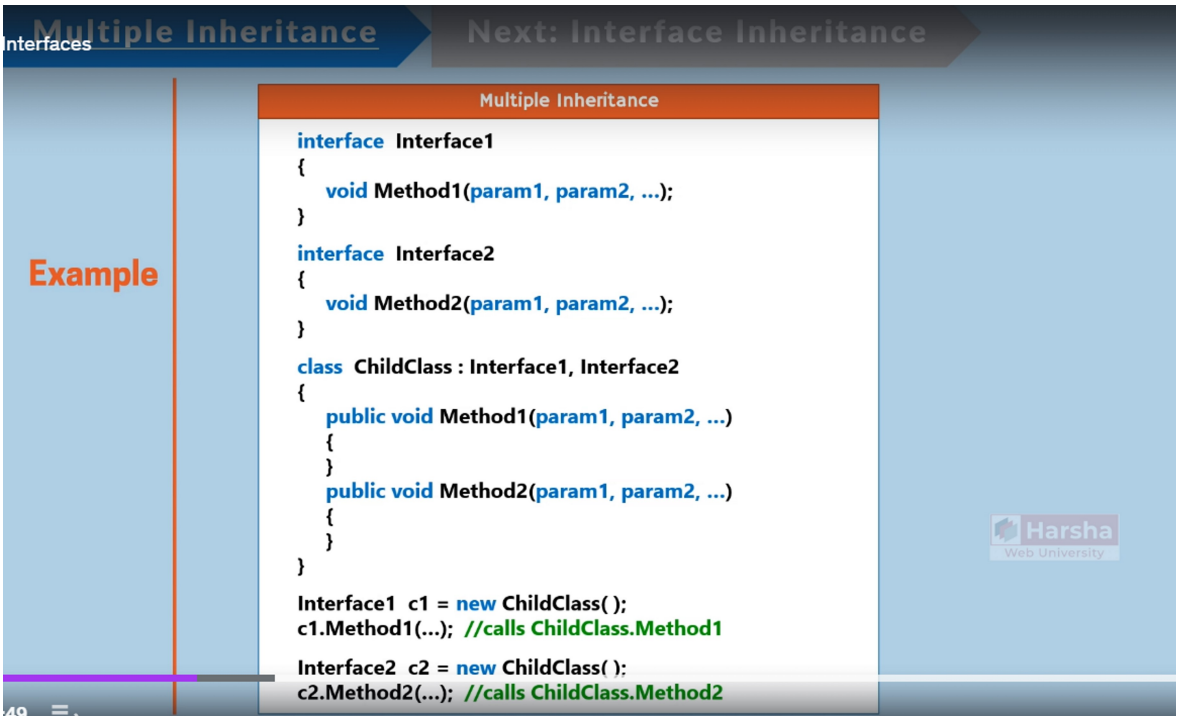

ith Interfaces
Implementation of different types of Polymorphism
Next: Multiple Inheritance
Run-Time Polymorphism Example - With Interfaces:
interface InterfaceName
{
void Add(int a, int b);
}
class ChildClass1 : InterfaceName
{
public void Add(int a, int b)
{
}
}
class ChildClass2 : InterfaceName
{
public void Add(int a, int b)
{
}
}
InterfaceName c1;
c1 = new ChildClass1( );
c1.Add(10, 20); //calls ChildClass1.Add
c1 = new ChildClass2( );
c1.Add(10, 20); //calls ChildClass2.Add
Harsha
Web University
20:20


Agenda
Harsha
1
Understanding Interfaces
What is interface and its features.
2
Dynamic Polymorphism
What is Polymorphism and different types of polymorphism.
Achieving dynamic polymorphism using interfaces.
3
Multiple Inheritance
Creating child classes with multiple parent interfaces.
Harsha
Web University
4
Interface Inheritance
Creating child interface with one or more parent interfaces.



You can say that through a reference variable of interface, you can access only the methods of the same interface.


Interfaces
Multiple Inheritance
Next: Interface Inheritance
What
In C#, "multiple inheritance" IS POSSIBLE with INTERFACES; that means a child class can have multiple parent interfaces.
Example
interface IVehicle
MoveBasedOnWheels( )
interface IFlyingMachine
FlyBasedOnAir( )
class FlyingVehicle: IVehicle, IFlyingMachine
MoveBasedOnWheels( )
FlyBasedOnAir( )
Harsha
Web University


Interfaces
Multiple Inheritance
Next: Interface Inheritance
Example
Multiple Inheritance
interface Interface1
{
void Method1(param1, param2, ...);
}
interface Interface2
{
void Method2(param1, param2, ...);
}
class ChildClass : Interface1, Interface2
{
public void Method1(param1, param2, ...)
{
}
public void Method2(param1, param2, ...)
{
}
}
Interface1 c1 = new ChildClass( );
c1.Method1(...); //calls ChildClass.Method1
Interface2 c2 = new ChildClass( );
c2.Method2(...); //calls ChildClass.Method2
Harsha
Web University

Multiple Inheritance

Next: Interface Inheritance

- "One Child - Multiple Parent Classes / Parent Interfaces" is called as "Multiple Inheritance".
- In C#.NET, "multiple inheritance" IS NOT POSSIBLE with CLASSES; that means you can't specify multiple parent classes.
- The child class MUST IMPLEMENT all methods of all the interfaces, that are inherited from.

Agenda

1. **Understanding Interfaces**
   What is interface and its features.
2. **Dynamic Polymorphism**
   What is Polymorphism and different types of polymorphism.
   Achieving dynamic polymorphism using interfaces.
3. **Multiple Inheritance**
   Creating child classes with multiple parent interfaces.
4. **Interface Inheritance**
   Creating child interface with one or more parent interfaces.

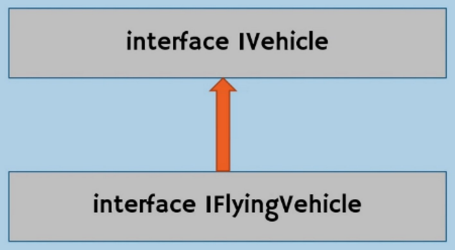
Interface Inheritance

Next: Explicit Interface Implementation

What

- If an interface inherits from another interface, we call it as "Interface Inheritance".
- The child class that implements the child interface must implement all the members of both parent interface and child interface too.

Example



Interface Inheritance

Next: Explicit Interface Implementation

Example

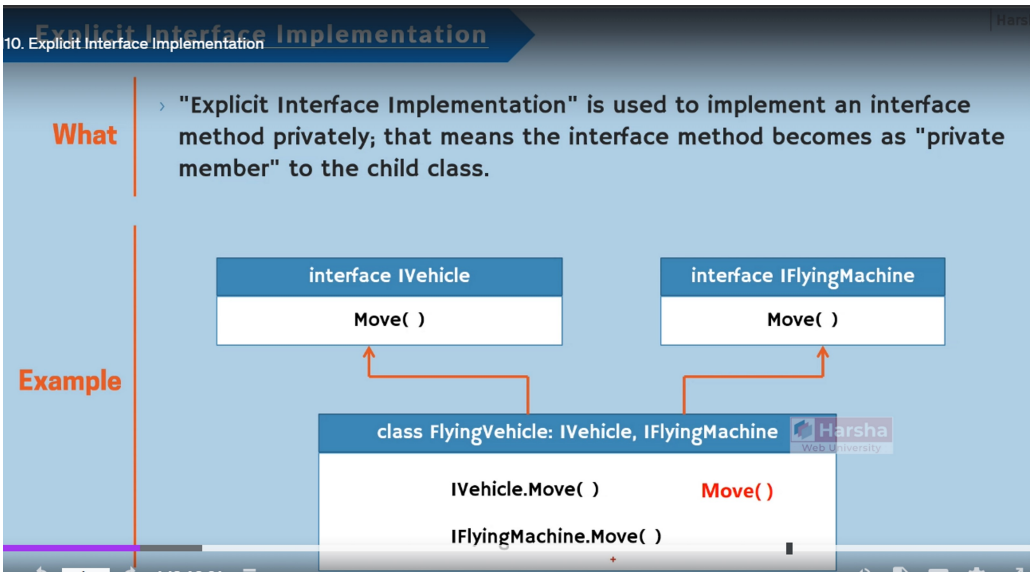
Interface Inheritance

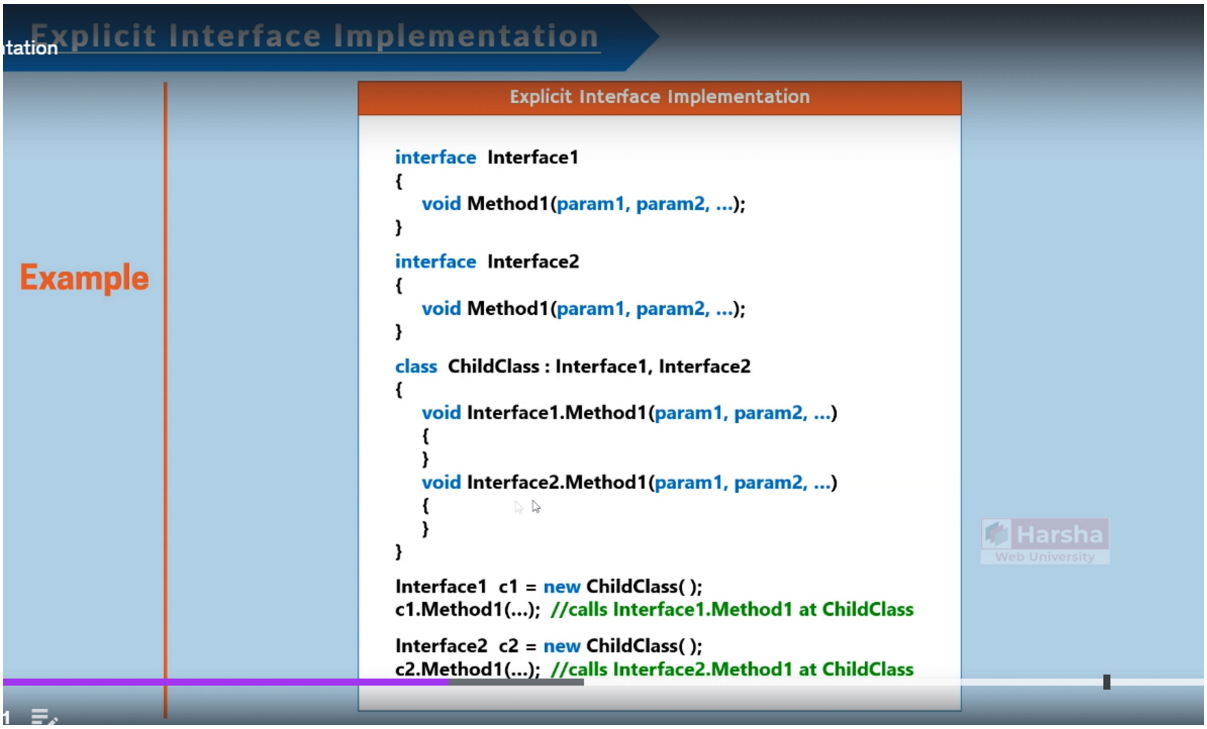
```
interface  Interface1
{
    public void Method1(param1, param2, ...);
}

interface  Interface2 : Interface1
{
    public void Method2(param1, param2, ...);
}

class  ChildClass : Interface2
{
    public void Method1(param1, param2, ...)
    {
    }
    public void Method2(param1, param2, ...)
    {
    }
}

Interface1  c1 = new ChildClass( );
c1.Method1(...);  //calls ChildClass.Method1

Interface2  c2 = new ChildClass( );
c2.Method1(...);  //calls ChildClass.Method1
c2.Method2(...);  //calls ChildClass.Method2
```


5

Explicit Interface Implementation

Creating two methods with same signature in two different interfaces and implementing both interfaces in the same child class.


10. Explicit Interface Implementation
Explicit Interface Implementation
What
"Explicit Interface Implementation" is used to implement an interface method privately; that means the interface method becomes as "private member" to the child class.
Example
interface IVehicle
Move( )
interface IFlyingMachine
Move( )
class FlyingVehicle: IVehicle, IFlyingMachine
IVehicle.Move( )
Move( )
IFlyingMachine.Move( )


Explicit Interface Implementation
Example
Explicit Interface Implementation
interface Interface1
{
void Method1(param1, param2, ...);
}
interface Interface2
{
void Method1(param1, param2, ...);
}
class ChildClass : Interface1, Interface2
{
void Interface1.Method1(param1, param2, ...)
{
}
void Interface2.Method1(param1, param2, ...)
{
}
}
Interface1 c1 = new ChildClass( );
c1.Method1(...); //calls Interface1.Method1 at ChildClass
Interface2 c2 = new ChildClass( );
c2.Method1(...); //calls Interface2.Method1 at ChildClass

Explicit Interface Implementation

- If a child class inherits from two or more interfaces, and there is a duplicate method (having same name and parameters) among those interfaces; then use "Explicit Interface Implementation", to provide different implementations for different interface methods respectively.
- You can use "Explicit Interface Implementation" to create private implementation of interface method; so that you can create abstraction for those methods.