18
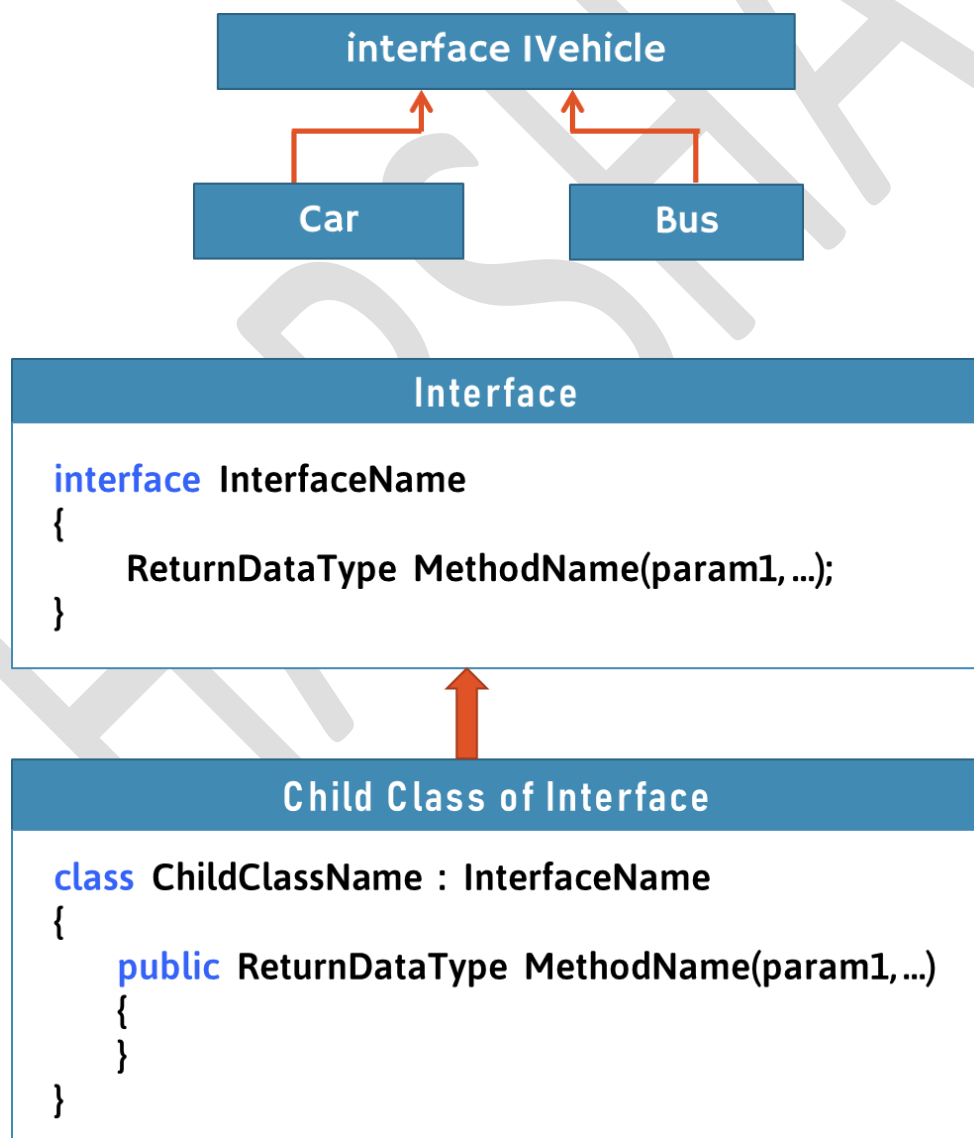
Interfaces

Introducing Interfaces

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

Interface

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

Child Class of Interface

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

› 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.

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

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

Polymorphism

› Polymorphism provides the ability to the developer, to define different implements for the same method in the same class or different classes.

› Two types of Polymorphism

› Compile-Time Polymorphism

› Run-Time 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".

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:

```csharp
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
```

<div style="background:#E8632B; color:white; text-align:center">Run-Time Polymorphism Example - With Interfaces:</div>

```csharp
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
```
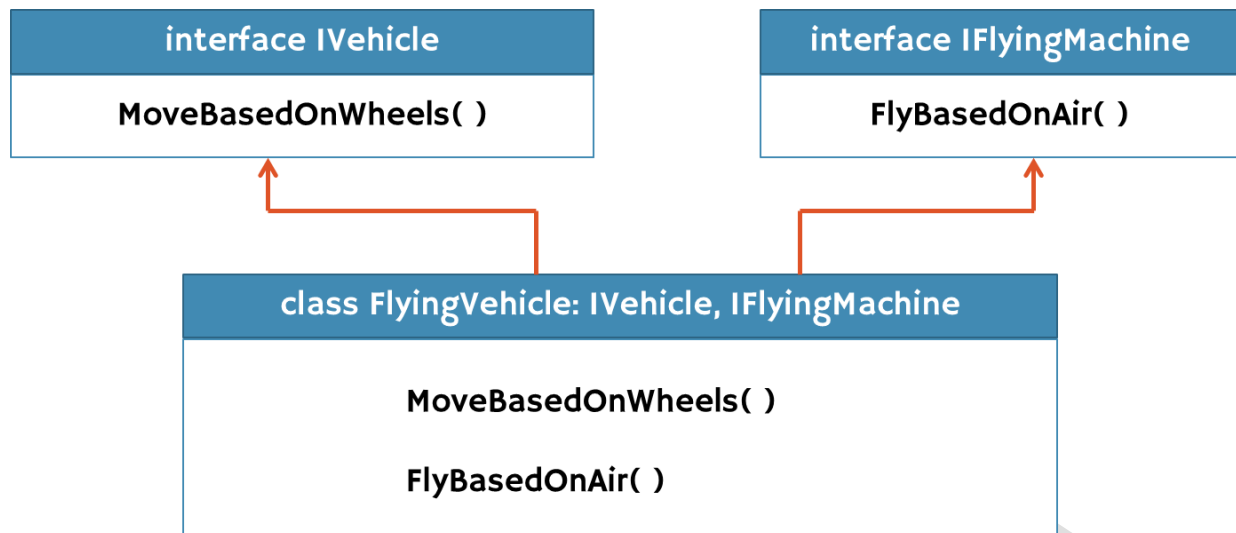
Multiple Inheritance

› In C#, "multiple inheritance" IS POSSIBLE with INTERFACES; that means a child class can have multiple parent interfaces.

interface IVehicle	interface IFlyingMachine
MoveBasedOnWheels( )	FlyBasedOnAir( )

class FlyingVehicle: IVehicle, IFlyingMachine

MoveBasedOnWheels( )

FlyBasedOnAir( )

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
```
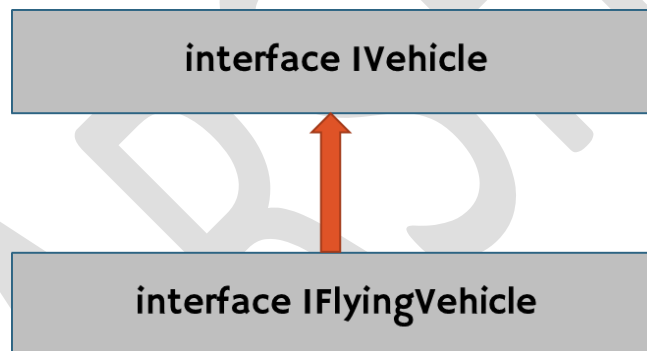
› "One Child - Multiple Parent Classes / Parent Interfaces" is called as "Multiple Inheritance".

› 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.

Interface Inheritance

› 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.

```
interface IVehicle
```

```
interface IFlyingVehicle
```

Interface Inheritance

```csharp
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
```
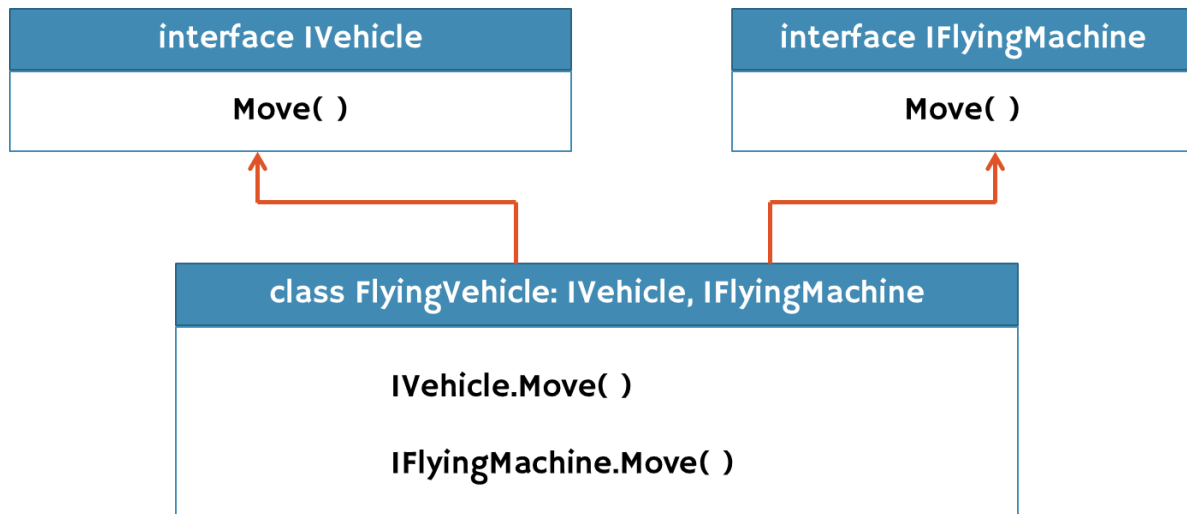
Explicit Interface Implementation

› "Explicit Interface Implementation" is used to implement an interface method privately; that means the interface method becomes as "private member" to the child class.

interface IVehicle	interface IFlyingMachine
Move( )	Move( )

class FlyingVehicle: IVehicle, IFlyingMachine

IVehicle.Move( )

IFlyingMachine.Move( )

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
```

› 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.