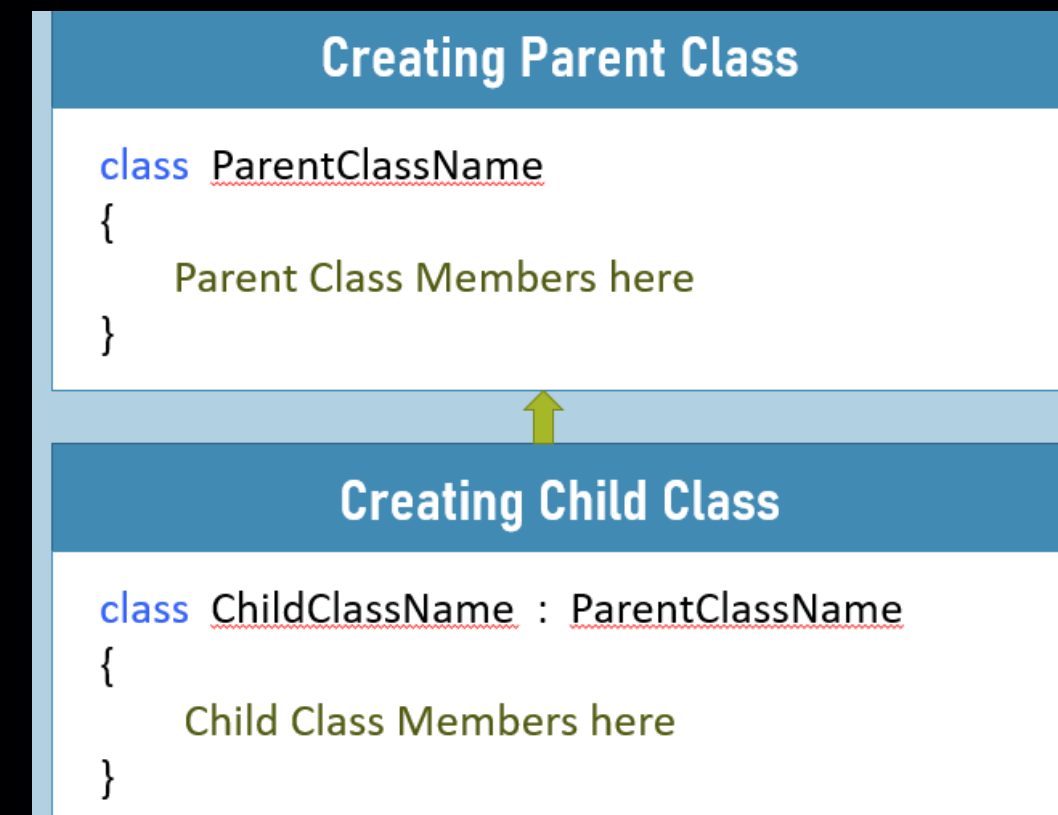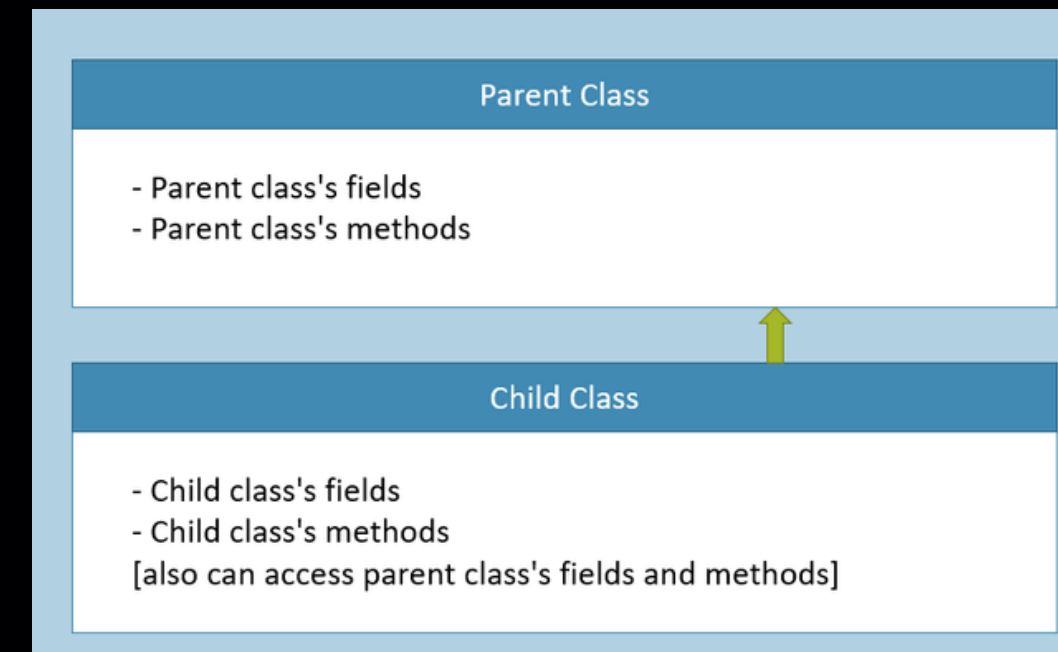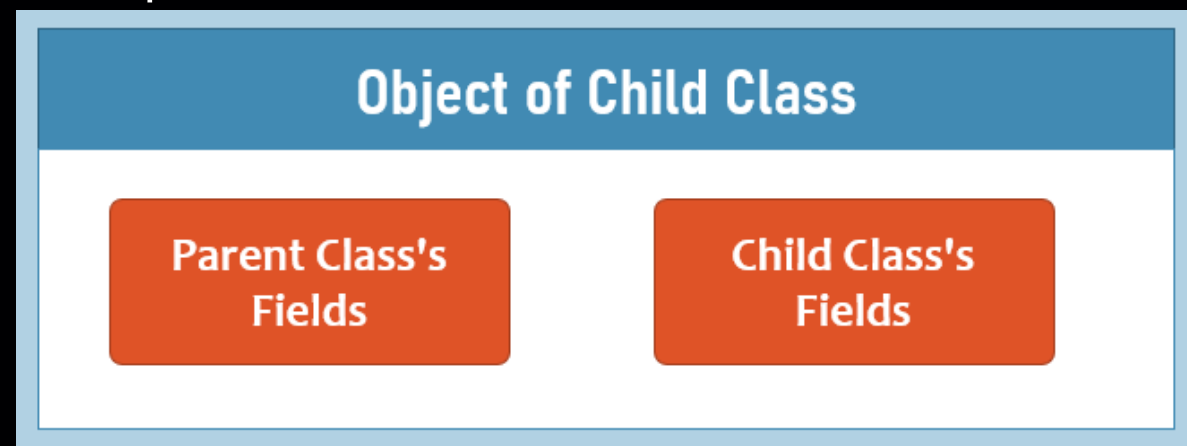# INHERITANCE

- 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.

- Concept of extending the parent class, by creating child class.

- "Child class" extends "parent class".

- The child class's object stores members of both child class and parent class.

Parent Class

- Parent class's fields
- Parent class's methods

Child Class

- Child class's fields
- Child class's methods
[also can access parent class's fields and methods]

Creating Parent Class

```
class ParentClassName
{
      Parent Class Members here
}
```

Creating Child Class

```
class ChildClassName : ParentClassName
{
      Child Class Members here
}
```

Object of Child Class

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

INHERITANCE

Types

1. Single Inheritance

One Parent Class, One Child Class.

2. Multi-Level Inheritance

One Parent Class, One Child Class; and the Child class has another Child class.

3. Hierarchical Inheritance

One Parent Class, Multiple Child Classes.

4. Multiple Inheritance

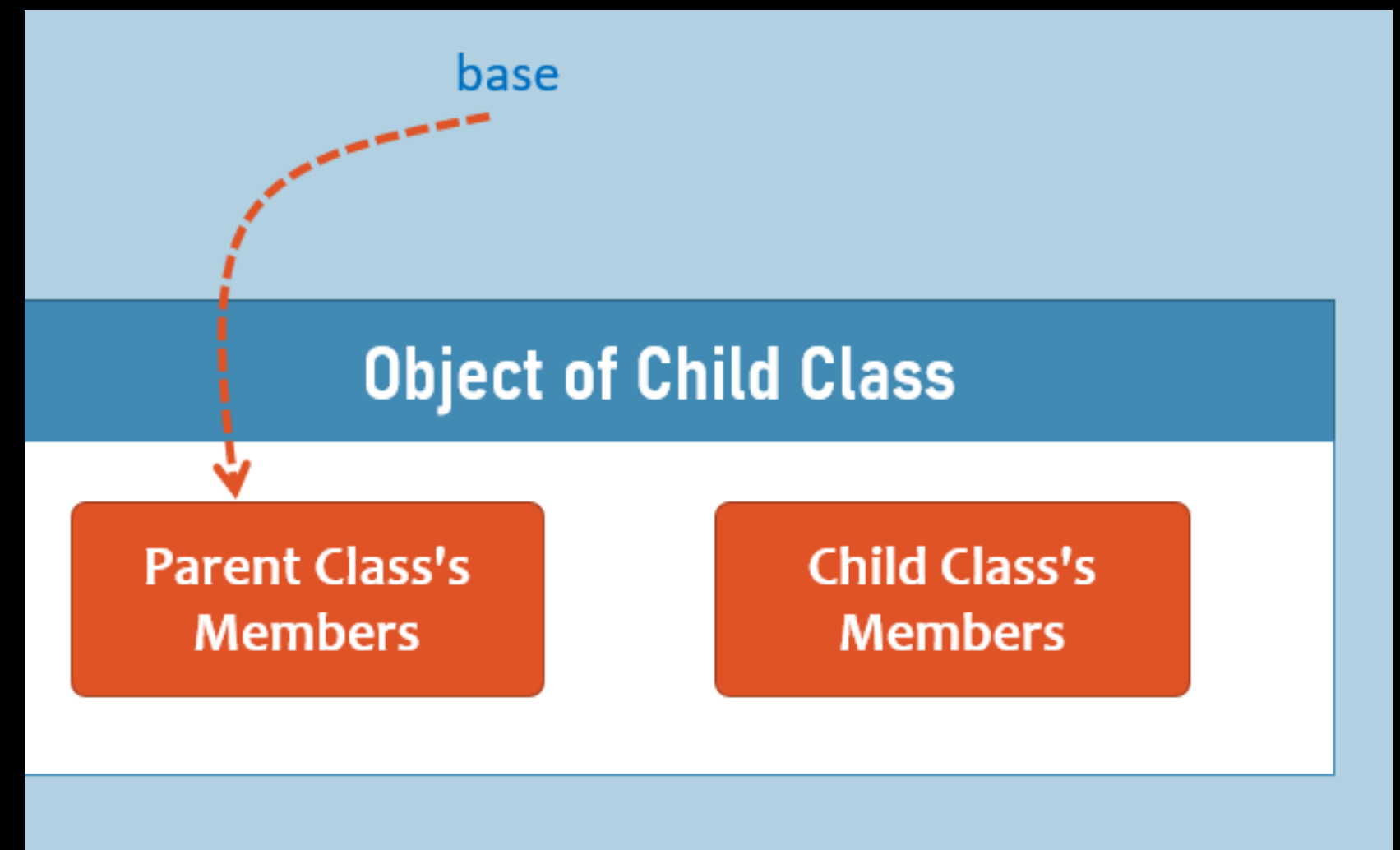Multiple Parent Classes, One Child class.

5. Hybrid Inheritance

Hierarchical Inheritance + Multi Level Inheritance

INHERITANCE

'base' keyword

- 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.
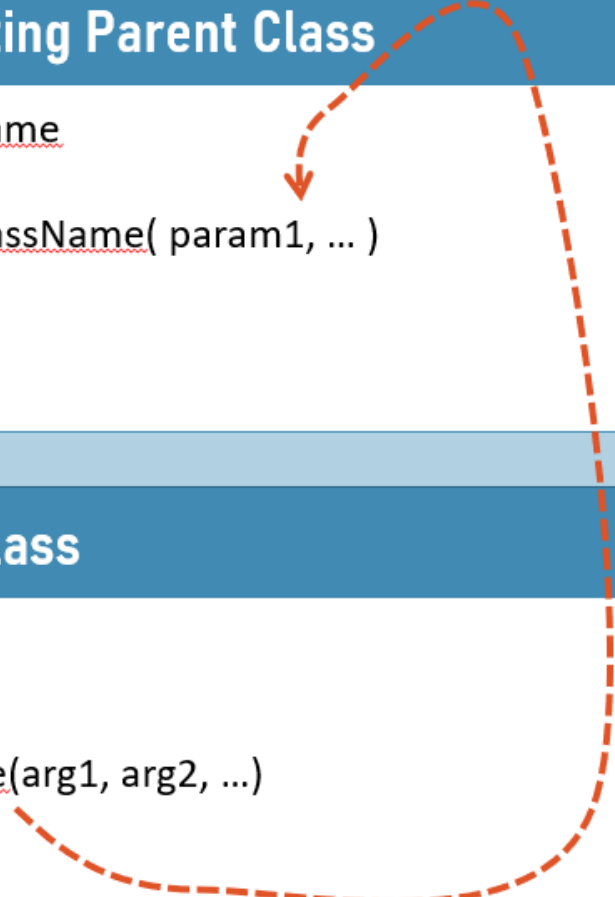
INHERITANCE

Parent Class's Constructor

- 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.

Creating Parent Class

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

Creating Child Class

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

INHERITANCE

Method Hiding

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.

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).

Creating Parent Class

```
class ParentClassName
{
    public void M    od     e( param1, ... )
    {
    }
}
```

Creating Child Class

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

INHERITANCE

Method Hiding

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.

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).

Creating Parent Class

```
class ParentClassName
{
    public void M      odN    e( param1, … )
    {
    }
}
```

Creating Child Class

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

81

INHERITANCE

Method Overriding

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.

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'.

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
    }
}
```

INHERITANCE

Key Points to Remember

- The "parent class" acts as a "base type" of "one or more child classes".

- The 'base' keyword inside the child class, can access the members of parent class.

- The child class's object stores parent class's fields also, automatically; however they are accessible in child class if they are not 'private'.

- Method hiding is a concept of 'overwriting' the parent class's method, by creating another method in child class, with same signature.

- Method overriding is a concept of 'extending' the parent class's method, by creating another method in child class, with same signature.

- C# doesn't support multiple inheritance (with multiple parent classes). That means, a child class can have ONLY ONE parent class; however, a child class can have MULTIPLE parent interfaces; so in this way, C# supports multiple inheritance.

SEALED

Sealed Classes

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.

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

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
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
Sealed Class	Yes	Yes	Yes	Yes	Yes	No	No	Yes	Yes

84

SEALED

Sealed Methods

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

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

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