Dot Net Training

by Besant Technologies

OOPS Concepts

Procedural programming is about writing procedures or methods that perform operations on the data, while object-oriented programming is about creating objects that contain both data and methods.

- Polymorphism
- Inheritance
- Encapsulation
- Data Abstraction

Polymorphism

The word polymorphism is derived from two Greek words: poly and morphs. The word “Poly” means many and “morphs” means forms. Therefore, polymorphism means “many forms”

1. **Static Polymorphism / Compile-Time Polymorphism / Early Binding**
2. **Dynamic Polymorphism / Run-Time Polymorphism / Late Binding**

Static Polymorphism

- Method Overloading
- Operator Overloading
- Method Hiding

Dynamic Polymorphism

- Method Overriding

Method Overloading

Method Overloading means it is an approach to defining multiple methods under the class with a single name. So, we can define more than one method with the same name in a class. But the point that you need to remember the parameters of all those methods should be different (different in terms of number, type, and order of the parameters).

```
Public void function1()

{...}

Public void function1(int x)

{

}
```

Overriding

The process of re-implementing the superclass non-static, non-private, and non-sealed method in the subclass with the same signature is called Method Overriding in C#.

Syntax:

```
class Class1      {//Virtual Function (Overridable Method)
    public virtual void Show()
    {
    }
}
class Class2 : Class1
{//Overriding Method
    public override void Show()
    {
    }
}
```

Method Hiding

Method Hiding/Shadowing is also an approach of re-implementing the parent class methods under the child class exactly with the same signature (same name and same parameters).

Syntax:

```
class Class1      {//Virtual Function (Overridable Method)
    public void Show()
    {
    }
}
class Class2 : Class1
{//Overriding Method
    public new void Show()
    {
    }
}
```

Operator Overloading

To make operators work with user-defined data types like classes.

Syntax:

1. The return type is the return type of the function.
2. the operator is a keyword.
3. Op is the symbol of the operator that we want to overload. Like: +, <, -, ++, etc.
4. The type must be a class or struct. It can also have more parameters.
5. It should be a static function

operator(object) ;

Operator a;

Operators

+,-,!,++,--,/,*,= — 3+4a , 5+8i → 8+12i

Inheritance

it is possible to inherit fields and methods from one class to another. We group the "inheritance concept" into two categories:

- Derived Class (child) - the class that inherits from another class
- Base Class (parent) - the class being inherited from

In C#, there are 4 types of inheritance:

1. Single inheritance: A derived class that inherits from only one base class.
2. Multi-level inheritance: A derived class that inherits from a base class and the derived class itself becomes the base class for another derived class.
3. Hierarchical inheritance: A base class that serves as a parent class for two or more derived classes.
4. Multiple inheritance: A derived class that inherits from two or more base classes.
5. Hybrid:

Inheritance - Rules

Rule -1:

In C#, the parent classes constructor must be accessible to the child class, otherwise, the inheritance would not be possible because when we create the child class object first it goes and calls the parent class constructor so that the parent class variable will be initialized and we can consume them under the child class

Rule2:

In inheritance, the child class can access the parent class members but the parent classes can never access any members that are purely defined in the child class.

Rule 3:

In C# we don’t have support for multiple inheritances through classes, what we are provided is only Single Inheritance through classes. That means with classes, only one immediate parent class is allowed (i.e. Single, Multilevel and Hierarchical supported), and more than one immediate parent class is not allowed in C# with classes (i.e. Multiple and Hybrid are not supported).

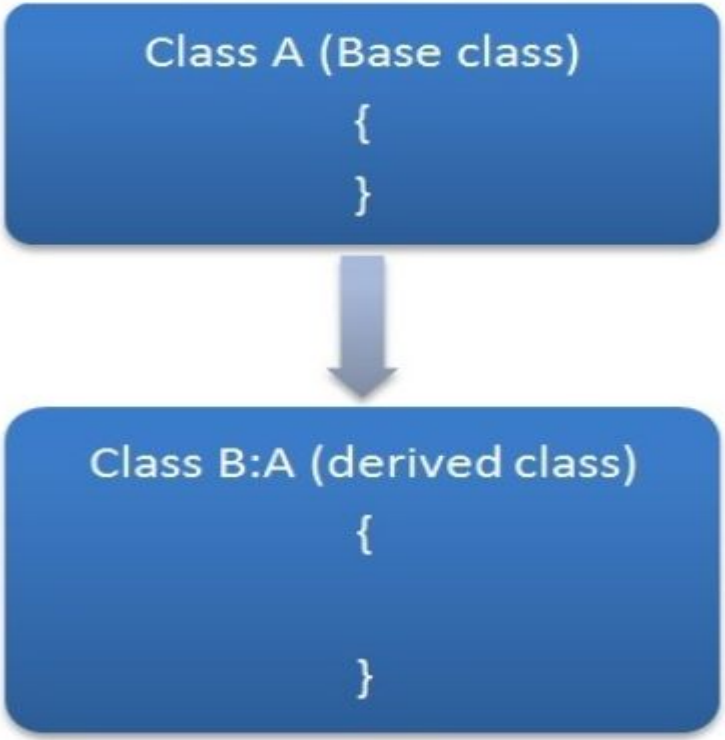
Advantages of Inheritance:

1. Code Reusability: Inheritance allows us to reuse existing code by inheriting properties and methods from an existing class.
2. Code Maintenance: Inheritance makes code maintenance easier by allowing us to modify the base class and have the changes automatically reflected in the derived classes.
3. Code Organization: Inheritance improves code organization by grouping related classes together in a hierarchical structure.

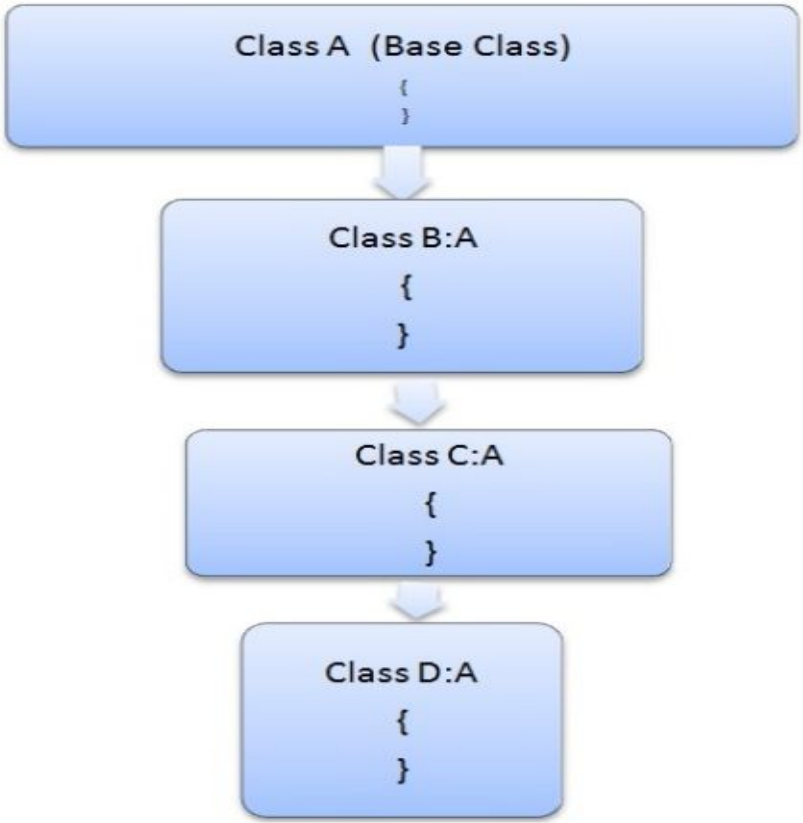
Disadvantages of Inheritance:

1. Tight Coupling: Inheritance creates a tight coupling between the base class and the derived class, which can make the code more difficult to maintain.
2. Complexity: Inheritance can increase the complexity of the code by introducing additional levels of abstraction.
3. Fragility: Inheritance can make the code more fragile by creating dependencies between the base class and the derived class.

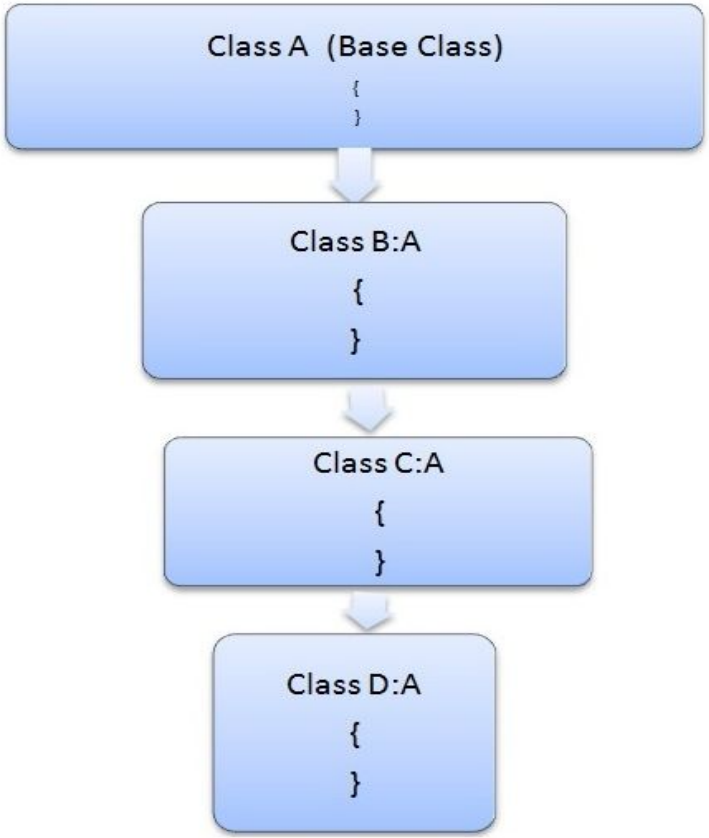
Single Inheritance



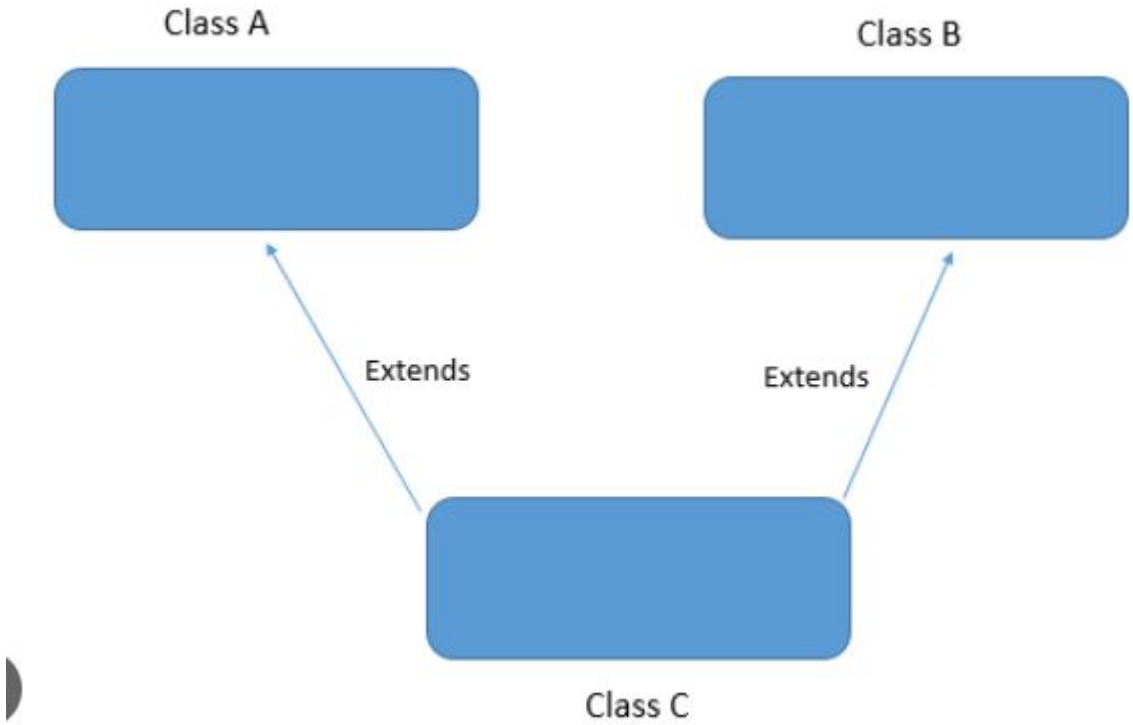
Multi Level



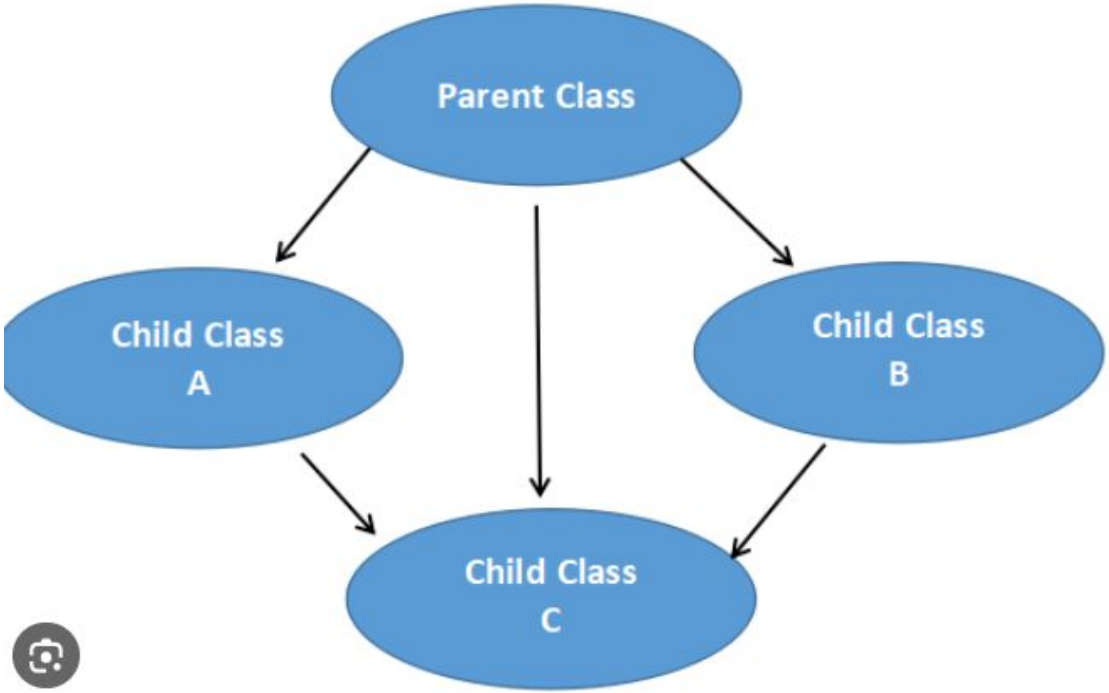
Hierarchical inheritance



Multiple inheritances



Hybrid inheritances



Encapsulation

Encapsulation Hides the internal state and functionality of an object and only allows access through a public set of functions

Encapsulation Hides the internal state and functionality of an object and only allows access through a public set of functions

Encapsulation

1. **public**: The public members can be accessed by any other code in the same assembly or another assembly that references it.
2. **private**: The private members can be accessed only by code in the same class.
3. **protected**: The protected Members in C# are available within the same class as well as to the classes that are derived from that class.
4. **internal**: The internal members can be accessed by any code in the same assembly, but not from another assembly.
5. **protected internal**: The protected internal members can be accessed by any code in the assembly in which it's declared, or from within a derived class in another assembly.
6. **private protected:** The private protected members can be accessed by types derived from the class that is declared within its containing assembly.

Getter and Setter:

Encapsulation

Advantages of Encapsulation in C#:

1. **Protection of data:** You can validate the data before storing it in the variable.
2. **Achieving Data Hiding:** The user will have no idea about the inner implementation of the class.
3. **Security:** The encapsulation Principle helps to secure our code since it ensures that other units(classes, interfaces, etc) can not access the data directly.
4. **Flexibility:** The encapsulation Principle in C# makes our code more flexible, which in turn allows the programmer to change or update the code easily.
5. **Control:** The encapsulation Principle gives more control over the data stored in the variables. For example, we can control the data by validating whether the data is good enough to store in the variable.

Abstraction

The process of representing the essential features without including the background details is called Abstraction.

In C#, we can implement the abstraction OOPs principle in two ways. They are as follows:

1. **Using Interface**
2. **Using Abstract Classes and Abstract Methods**

Abstraction

Advantages of Abstraction Principle in C#

1. The Abstraction Principle reduces the complexity of viewing things. It only provides the method signature by hiding how the method is actually implemented.
2. The Abstraction Principle helps to increase the security of an application or program as we are only providing the necessary details to call the method by hiding how the methods are actually implemented.
3. With the Abstraction Principle, the enhancement will become very easy because without affecting end-users we can able to perform any type of changes in our internal system.
4. Without the Abstraction principle, maintaining application code is very complex. Abstraction gives one structure to program code.

Encapsulation vs Abstraction

1. The Encapsulation Principle is all about data hiding (or information hiding). On the other hand, the Abstraction Principle is all about detailed hiding (implementation hiding).
2. Using the Encapsulation principle, we can protect our data i.e. from outside the class, nobody can access the data directly. We are exposing the data through publicly exposed methods and properties. The advantage is that we can validate the data before storing and returning it. On the other hand, using the Abstraction Principle, we are exposing only the services so that the user can consume the services but how the services/methods are implemented is hidden from the user. The user will never know, how the method is implemented.
3. With the Encapsulation Principle, we group data members and member functions into a single unit called class, interface, enum, etc. On the other hand, with the Abstraction Principle, we are exposing the interface or abstract class to the user and hiding implementation details i.e. hiding the child class information.
4. We can implement Encapsulation by declaring the data members as private and exposing the data members only through public exposed methods and properties with proper validation. On the other hand, we can implement abstraction through abstract classes and interfaces.
5. In abstraction, only the abstract view is presented to the user while complex and detailed data is hidden from the user. On the other hand, in encapsulation, data member and member functions are bundled as a single unit and can be protected or made accessible using access modifiers and getter and setter methods.
6. Abstraction in C# is used to hide unwanted data and shows only the required properties and methods to the user. Encapsulation in C# is used to bind data members and member functions into a single unit to prevent outsiders from accessing it directly.