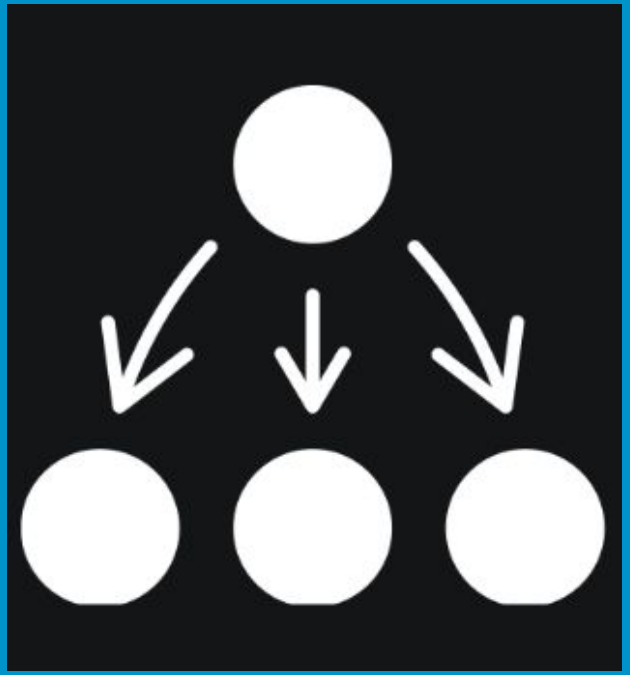
Inheritance



OOPs A PIE

Stands for:

A - abstraction

P - polymorphism

I - inheritance

E - encapsulation

Inheritance allows us to define a class based on another class.

What’s the point?

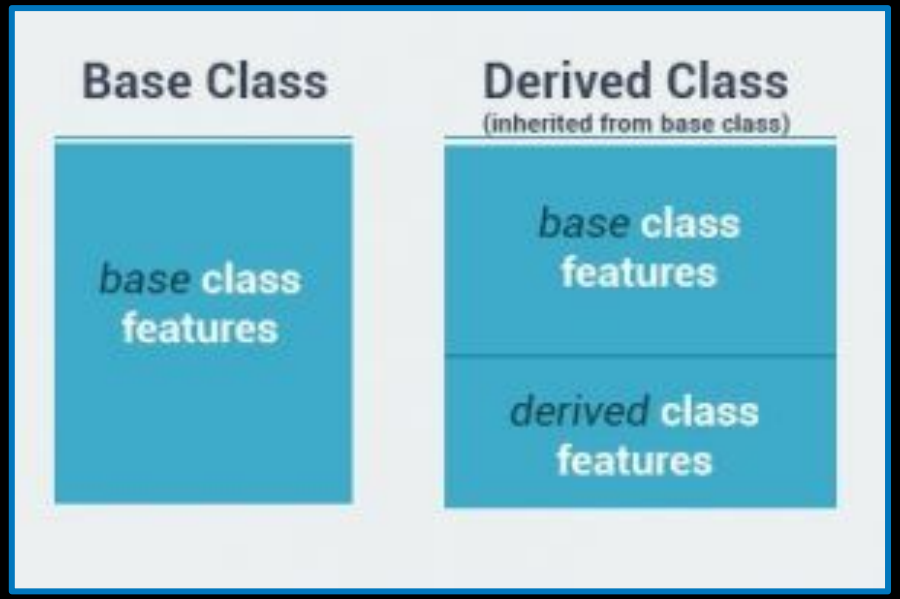
- **Inheritance** enables you to create new classes that reuse, extend, and modify the behavior that is defined in other classes.
- This makes creating and maintaining an application easy.

Definitions

Base Class - The class whose members are inherited from

Derived Class - and the class that inherits those members

Inheritance



DERIVED CLASS

(inherited from base class)

BASE CLASS

Base Class Features

Base Class Features

Derived Class Features

DRY - Do Not Repeat Yourself

Inheritance allows us to keep our code DRY.

DRY

- Inheritance allows the derived class to reuse the code in the base class without having to rewrite it.
- And the derived class can be customized by adding more members. In this manner, **the derived class extends the functionality of the base class.**

Inheritance

The derived class can thereby reuse the code in the base class without having to re-implement it.

Inheritance

When you define a class to derive from another class, **the derived class implicitly gains all the members of the base class, except for its constructors , finalizers, private members, and static members.**

- Except for Constructors: Constructors are special methods that are called to create a new instance of a class. Constructors are not inherited in C# because they are specific to the class they are defined in. Each class must define its own constructors. However, the derived class can call a constructor of the base class using the `base` keyword.
- Except for Finalizers: Finalizers (or destructors) are also special methods that are called when an instance of a class is about to be destroyed. Like constructors, they are specific to the class they are defined in and are not inherited.
- Except for Private Members: Private members of a class are accessible only within the same class and are not accessible to its derived classes. This is by design to encapsulate and hide specific details within the class itself.
- Except for Static Members: Static members belong to the class itself rather than to any specific instance of the class. While they are not inherited, they are still accessible from derived classes through the base class name. However, they do not behave as inherited members because they are not tied to an instance of the deriving class.

Can’t inherit non-instance members

Moreover, the **derived class cannot inherit the constructor of the base class** because constructors are not instance members of a class.

Inheritance

- The class whose members are inherited from is called the base class (PARENT) and the class that inherits those members is called the derived class (CHILD).

Parent → Base class

Child → Derived class

How To:

- **This is done by using a colon : after the derived class and then typing the name of the base class.**

Derived : Base

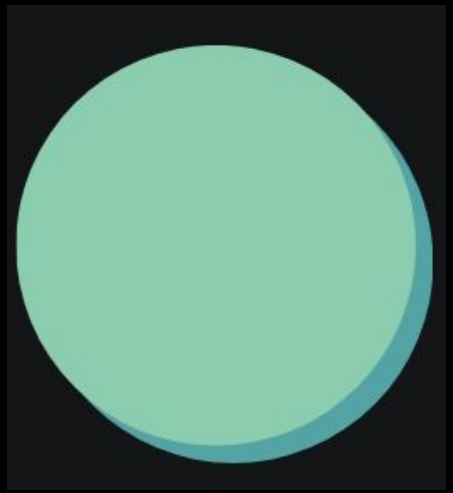
How To:

- **This is done by using a colon : after the derived class and then typing the name of the base class.**

Child : Parent

Checking Account : Bank Account

Car : Vehicle



Circle : Shape

Dog : Animal

Apple : Fruit

Note:

C# does not support multiple inheritance, so you cannot inherit from multiple classes.

Takeaway

- **Inheritance** allows you to extend the behavior of classes