Declaring an Object with Parent Class or Interface Type

In C#, you can create an object using the **type of the parent class or interface**, even if the object is created from a **child class**.

```
ParentClass obj = new ChildClass();
```

```
IMyInterface obj = new MyClass();
```

This is not just allowed—it's **powerful**. It allows us to write flexible, reusable, and loosely coupled code.

🔷 Why Do We Use Parent Class or Interface Type?

1. Polymorphism

You can treat different child classes as the same type, which simplifies your code and allows for dynamic behavior.

2. Abstraction

You can hide complex implementation details and only expose necessary functionality.

3. Flexibility & Extensibility

It allows you to change the concrete implementation without changing the calling code.

🧠 Real-Life Analogy

Think of IVehicle as an interface:

```
interface IVehicle
{
   void Start();
}
```

Now, many classes implement IVehicle:

```
class Car : IVehicle
{
   public void Start() => Console.WriteLine("Car is starting...");
}

class Motorcycle : IVehicle
{
   public void Start() => Console.WriteLine("Motorcycle is starting...");
}
```

Now instead of:

```
Car myCar = new Car();
Motorcycle myBike = new Motorcycle();
```

You can do:

```
IVehicle vehicle1 = new Car();
IVehicle vehicle2 = new Motorcycle();
```

Now you can store both in a list and treat them the same:

```
List<IVehicle> vehicles = new List<IVehicle>
{
   new Car(),
   new Motorcycle()
};
foreach (var v in vehicles)
{
   v.Start(); // Calls the correct version automatically
}
```

🔶 **Another Example with Inheritance**

```
class Animal
{
   public virtual void Speak()
   {
      Console.WriteLine("Animal makes a sound");
   }
}
class Dog : Animal
{
   public override void Speak()
   {
      Console.WriteLine("Dog barks");
   }
}
```

```
class Cat : Animal
{
    public override void Speak()
    {
        Console.WriteLine("Cat meows");
    }
}
```

Use parent class to refer to children:

```
Animal a1 = new Dog();
Animal a2 = new Cat();
a1.Speak(); // Dog barks
a2.Speak(); // Cat meows
```

This allows:

```
List<Animal> animals = new List<Animal>
{
    new Dog(),
    new Cat()
};
foreach (var a in animals)
{
    a.Speak(); // Correct version called
}
```

🔶 Interface Example in a System

Imagine you're building a **payment system**:

```
interface IPaymentMethod
{
    void Pay(decimal amount);
}
```

You have several payment methods:

```
class CreditCard : IPaymentMethod
{  public void Pay(decimal amount)
   {
      Console.WriteLine($"Paid {amount} with Credit Card");
   }
}
class PayPal : IPaymentMethod
{
   public void Pay(decimal amount)
   {
      Console.WriteLine($"Paid {amount} with PayPal");
   }
}
```

Use interface to treat all payment methods the same:

```
void ProcessPayment(IPaymentMethod method, decimal amount)
{
   method.Pay(amount); // Interface hides the implementation
}
```

Usage:

```
ProcessPayment(new CreditCard(), 100);
ProcessPayment(new PayPal(), 50);
```

🔷 **Summary**

Concept	**Benefit**
Declaring object as Parent Class / Interface	Enables **polymorphism**
Allows working with **collections of mixed types**	Simplifies code
Can **swap implementations** without rewriting code	Adds flexibility
Encourages **coding to an interface**, not an implementation	Best practice