Polymorphism in C# is a core principle of object-oriented programming (OOP) that allows methods in derived classes to have different behaviors while sharing the same name as those in a base class. It provides flexibility and allows code to be more reusable and maintainable.

There are two main types of polymorphism in C#: **Compile-time (or static)** and **Run-time (or dynamic)**.

**1. Compile-time Polymorphism (Method Overloading)**

This type of polymorphism is achieved by method overloading or operator overloading. It is resolved at compile time.

**Method Overloading**

In method overloading, multiple methods in the same class share the same name but differ in the number or type of parameters.

using System;

class CompileTimePolymorphism

{

public void Display()

{

Console.WriteLine("Display method with no parameters.");

}

public void Display(string message)

{

Console.WriteLine("Display method with a message: " + message);

}

public void Display(string message, int number)

{

Console.WriteLine($"Display method with a message: {message} and a number: {number}");

}

}

class Program

{

static void Main()

{

CompileTimePolymorphism obj = new CompileTimePolymorphism();

obj.Display();

obj.Display("Hello, World!");

obj.Display("Hello", 42);

}

}

**2. Run-time Polymorphism (Method Overriding)**

This type of polymorphism is achieved using method overriding and works through inheritance and interfaces. It is resolved at run time.

**Method Overriding**

In method overriding, a derived class provides a specific implementation of a method that is already defined in its base class. To enable overriding, the virtual keyword is used in the base class, and the override keyword is used in the derived class.

using System;

class BaseClass

{

public virtual void Display()

{

Console.WriteLine("Display method in BaseClass.");

}

}

class DerivedClass : BaseClass

{

public override void Display()

{

Console.WriteLine("Display method in DerivedClass.");

}

}

class Program

{

static void Main()

{

BaseClass obj = new DerivedClass(); // Polymorphism in action

obj.Display(); // Calls the Display method in DerivedClass

}

}

**Key Features of Run-time Polymorphism:**

1. **Inheritance**: The derived class must inherit from the base class.
2. **Virtual Methods**: The base class method must be declared with the virtual keyword.
3. **Override Keyword**: The derived class method must use the override keyword.
4. **Dynamic Dispatch**: The method called is determined at runtime.

**Other Forms of Polymorphism**

* **Interfaces**: Interfaces allow different classes to implement the same methods, enabling polymorphism through a common contract.

interface IShape

{

void Draw();

}

class Circle : IShape

{

public void Draw()

{

Console.WriteLine("Drawing a Circle.");

}

}

class Rectangle : IShape

{

public void Draw()

{

Console.WriteLine("Drawing a Rectangle.");

}

}

class Program

{

static void Main()

{

IShape shape;

shape = new Circle();

shape.Draw();

shape = new Rectangle();

shape.Draw();

}

}

**Abstract Classes**: Abstract methods in abstract classes also enable polymorphism by requiring derived classes to implement them.