OOPs in C#

Explore the fundamentals of object-oriented programming (OOP) in C#

What is Object-Oriented Programming (OOP)?

Core Concept

OOP is a programming paradigm based on the concept of "objects" that contain data and methods, representing real-world entities.

Emphasis

OOP focuses on data abstraction, encapsulation, inheritance, and polymorphism to create modular, reusable, and maintainable code.

Key Principles of OOP

Encapsulation

Hides data and methods within objects, allowing controlled access.

Inheritance

Creates new classes based on existing ones, promoting code reuse.

Polymorphism

Allows objects of different classes to respond to the same method call in different ways.

Abstraction

Simplifies complex systems by focusing on essential features while hiding unnecessary details.

```
// Class
public class Animal
{
    public string Name;
    public string Sound;

    public void Speak()
    {
        Console.WriteLine($"{Name} makes a {Sound} sound.");
    }
}

// Object
Animal dog = new Animal { Name = "Dog", Sound = "Bark" };
dog.Speak(); // Output: Dog makes a Bark sound.
```

Classes and Objects in C#

Classes

Blueprints for creating objects, defining data (fields) and behavior (methods).

Objects

Instances of classes, containing specific data and methods that operate on that data.

```
public class Animal
{
    private string _name;

    public string Name
    {
        get { return _name; }
        set { _name = value; }
    }

    public void DisplayName()
    {
        Console.WriteLine($"Animal Name: {Name}");
    }
}

var cat = new Animal();
cat.Name = "Cat"; // Setting name using property
cat.DisplayName(); // Output: Animal Name: Cat
```

Encapsulation in C#

Data Hiding

Private fields protect data from direct modification, ensuring integrity.

Controlled Access

Public methods provide a controlled interface for interacting with the encapsulated data.

Inheritance in C#

1

Base Class

Defines common attributes and behaviors shared by derived classes.

2

Derived Class

Inherits properties and methods from the base class, adding or overriding specific features.

```
public class Animal
{
    public void Eat() => Console.WriteLine("This animal eats food.");
}

public class Dog : Animal
{
    public void Bark() => Console.WriteLine("Dog barks.");
}

Dog dog = new Dog();
dog.Eat(); // Output: This animal eats food.
dog.Bark(); // Output: Dog barks.
```

Polymorphism in C#

1

Overriding

Derived classes redefine methods inherited from the base class to provide specific functionality.

2

Overloading

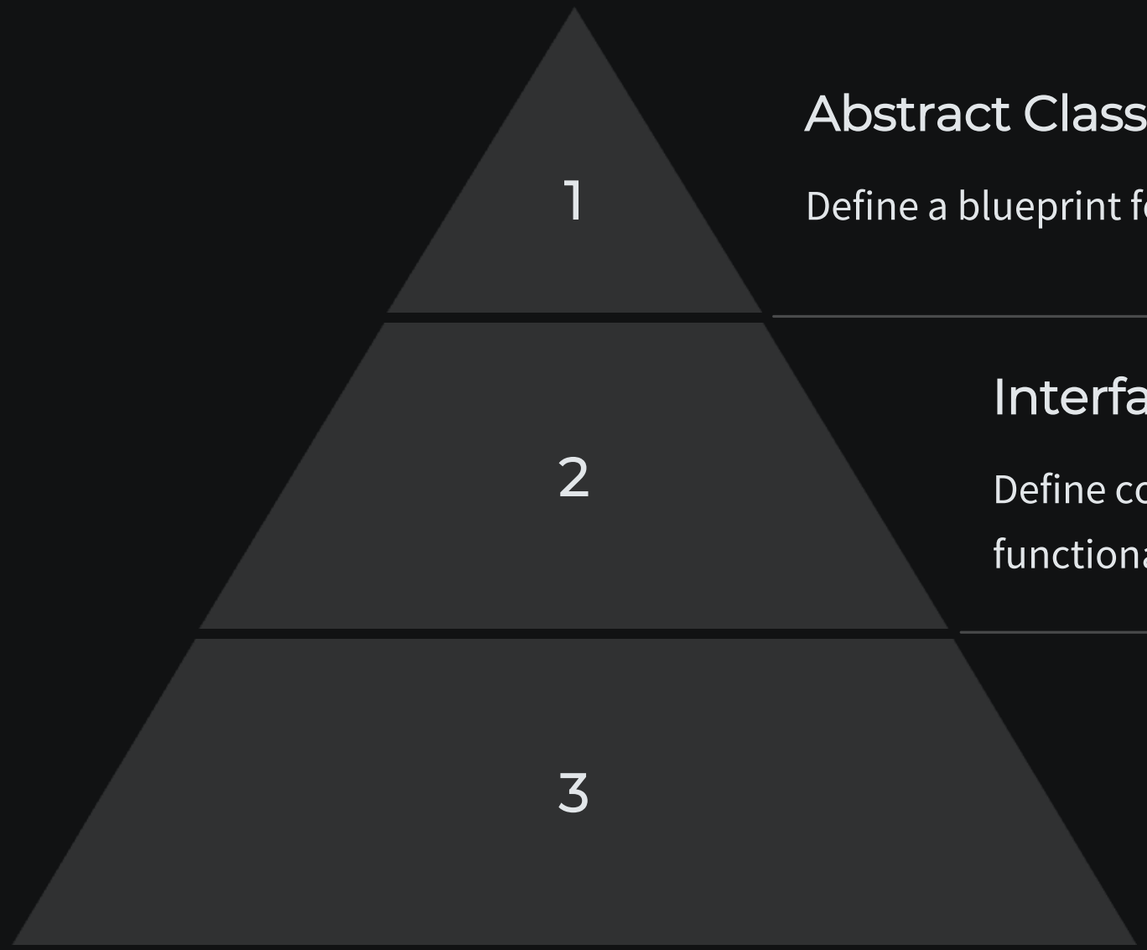
Multiple methods with the same name but different parameters, allowing different behaviors based on input.

```
public class Animal
{
    public virtual void Speak() => Console.WriteLine("Animal makes a sound.");
}

public class Dog : Animal
{
    public override void Speak() => Console.WriteLine("Dog barks.");
}

Animal animal = new Dog();
animal.Speak(); // Output: Dog barks.
```

Abstraction in C#



Abstract Classes

Define a blueprint for common behaviors, but can't be instantiated.

Interfaces

Define contracts that classes must implement, ensuring specific functionalities.

Benefits

Simplifies code by focusing on essential features and hiding implementation details.

Example :

```
public abstract class Animal
{
    public abstract void MakeSound();
}

public class Cow : Animal
{
    public override void MakeSound() => Console.WriteLine("Cow says Moo.");
}

Animal cow = new Cow();
cow.MakeSound(); // Output: Cow says Moo.
```

Benefits of using OOP in C#

1

Reusability

Code can be reused across different projects, reducing development time.

2

Maintainability

Code becomes easier to understand and modify, making it simpler to fix bugs and add new features.

3

Flexibility

OOP allows for flexible and extensible code, adapting to changing requirements.

Practical Examples of OOP in C# Applications

1

GUI Design

Objects represent buttons, labels, and other UI elements, enabling modular design and easy modifications.

2

Database Interaction

Objects model database tables and records, facilitating data access and manipulation.