# Basic and Advanced OOP Interview Questions in C#

## Basic OOP Interview Questions in C#

1. **What is Object-Oriented Programming (OOP)?**
   OOP is a programming paradigm based on the concept of objects. It helps organize code using four main principles:
   - Encapsulation
   - Inheritance
   - Polymorphism
   - Abstraction

2. **What is a class and an object in C#?**
   A class is a blueprint for creating objects.
   An object is an instance of a class.
   Example:

```
public class Car
{
    public string Color = "Red";
}
Car myCar = new Car();  // myCar is an object of class Car
```

3. **What is Encapsulation?**
   Encapsulation means hiding the internal state and requiring all interaction to be performed through methods.
   Example:

```
public class Student
{
    private int age;
    public void SetAge(int a) { age = a; }
    public int GetAge() { return age; }
}
```

4. **What is Inheritance?**
   Inheritance allows one class (child) to inherit fields and methods from another class (parent).
   Example:

```
public class Animal
{
    public void Eat() { Console.WriteLine("Eating"); }
}
public class Dog : Animal
{
    public void Bark() { Console.WriteLine("Barking"); }
}
```

5. **What is Polymorphism?**
   Polymorphism means "many forms"  the same method behaves differently.
   *Compile-time polymorphism* → method overloading
   *Run-time polymorphism* → method overriding
   Overloading:

```csharp
public class Math
{
    public int Add(int a, int b) => a + b;
    public double Add(double a, double b) => a + b;
}
```

Overriding:

```csharp
public class Animal
{
    public virtual void Speak() => Console.WriteLine("Animal sound");
}
public class Cat : Animal
{
    public override void Speak() => Console.WriteLine("Meow");
}
```

6. **What is Abstraction?**
   Abstraction means showing only essential details and hiding the internal logic.
   Example using abstract class:

```csharp
public abstract class Shape
{
    public abstract void Draw();
}
public class Circle : Shape
{
    public override void Draw() { Console.WriteLine("Drawing Circle"); }
}
```

7. **What is the difference between Abstract Class and Interface?**

| **Feature** | **Abstract Class** | **Interface** |
|---|---|---|
| Inheritance | Single | Multiple |
| Members | Can have fields and methods | Only methods/properties (no fields in C# $<$ 8) |
| Access Modifiers | Can use public/protected/private | All members are public by default |
| Use Case | When some base functionality needed | When no implementation is shared |

8. **Can a class implement multiple interfaces in C#?**
   Yes, C# supports multiple interface inheritance.
   Example:

```csharp
public interface IWalk { void Walk(); }
public interface IRun { void Run(); }
public class Person : IWalk, IRun
{
    public void Walk() { Console.WriteLine("Walking"); }
    public void Run() { Console.WriteLine("Running"); }
}
```

9. **What is the difference between Method Overloading and Overriding?**

| **Feature** | **Overloading** | **Overriding** |
|---|---|---|
| Time | Compile-time | Run-time |
| Class Type | Same class | Parent and child class |
| Signature | Different parameters | Same signature |
| Keyword | No keyword needed | Uses virtual and override |

10. **What is a Constructor in C#?**
    A constructor is a special method that runs automatically when an object is created.
    Example:

```csharp
public class Car
{
    public Car()
    {
        Console.WriteLine("Car object created");
    }
}
```

## Advanced OOP Interview Questions in C#

1. **What is the difference between an abstract class and an interface in C# (with C# 8.0+ features)?**

| **Feature** | **Abstract Class** | **Interface (C# 8.0+)** |
|---|---|---|
| Inheritance | Single | Multiple |
| Method Implementation | Can have both implemented and abstract methods | Can have default method implen |
| Constructor | Yes | No |
| Access Modifiers | Supports all (public, protected, etc.) | Members are public by default |

Example:

```csharp
interface IPrinter
{
    void Print();
    void Scan() => Console.WriteLine("Default Scan"); // C# 8.0+
}
```

2. **What is the use of the sealed keyword in C#?**
A *sealed* class cannot be inherited.
A *sealed* method cannot be overridden further.
Example:

```csharp
public sealed class FinalClass { }
public class Base
{
    public virtual void Show() { }
}
public class Derived : Base
{
    public sealed override void Show() { }
}
```

3. **What is the difference between virtual, override, and new keywords?**
*virtual*: Defines a method that can be overridden in a derived class.
*override*: Replaces a virtual method from the base class.
*new*: Hides a method from the base class (not polymorphism).
Example:

```csharp
public class Base
{
    public virtual void Show() => Console.WriteLine("Base");
}
public class Derived : Base
{
    public override void Show() => Console.WriteLine("Derived");
}
```

4. **What is object slicing? Does it happen in C#?**
Object slicing occurs when a derived class object is assigned to a base class by value, and derived members are "sliced" off.
C# prevents slicing because all class types are reference types  slicing is a C++ concept.

5. **Can you explain covariance and contravariance in C#?**
*Covariance*: You can assign a more derived type to a generic delegate or interface.
*Contravariance*: You can assign a less derived type.
Example:

```
IEnumerable<string> strList = new List<string>(); // Covariance
Action<object> objAction = (o) => Console.WriteLine(o);
Action<string> strAction = objAction;  // Contravariance
```

6. **What is the difference between shallow copy and deep copy?**
*Shallow Copy*: Copies reference to objects (same inner object).
*Deep Copy*: Copies the entire object and its nested objects.
Example (Shallow):

```
Person p1 = new Person();
Person p2 = p1;  // p1 and p2 refer to the same object
```

7. **What is a destructor? How does it work in C#?**
A destructor is called when an object is garbage collected.
Its declared using `C̃lassName()`.
Example:

```
class MyClass
{
    ~MyClass()
    {
        Console.WriteLine("Destructor called");
    }
}
```

**Note**: You cannot control when the destructor is called.

8. **What is the SOLID principle in OOP?**

| Principle | Description |
|---|---|
| S Single Responsibility | A class should have one reason to change |
| O Open/Closed | Open for extension, closed for modification |
| L Liskov Substitution | Subclasses should be substitutable for base classes |
| I Interface Segregation | Many small interfaces are better than one large one |
| D Dependency Inversion | Depend on abstractions, not concretions |

9. **What is composition vs inheritance? Which is better?**
*Inheritance*: "Is-a" relationship.
*Composition*: "Has-a" relationship (preferred for flexibility).
Example of Composition:

```
class Engine { }
class Car
{
    private Engine _engine = new Engine();  // Car *has-a* Engine
}
```

10. **What is late binding vs early binding?**
*Early Binding*: Method resolved at compile-time (e.g., normal method calls).
*Late Binding*: Method resolved at run-time using reflection or `dynamic`.
Example:

```
dynamic obj = "Hello";
Console.WriteLine(obj.Length);  // Late binding
```