Abstraction in C#

Abstraction in C# is one of the four fundamental principles of Object-Oriented Programming (OOP), along with **Encapsulation**, **Inheritance**, and **Polymorphism**. Abstraction is the process of hiding the complex implementation details of a system and exposing only the necessary and relevant parts to the user. This allows developers to work with simplified, abstract representations of objects rather than the full complexity behind them.

In C#, abstraction is mainly implemented using **abstract classes** and **interfaces**.

What is Abstraction?

Definition: Abstraction allows you to define the essential properties and behaviors of an object without revealing the implementation details. This ensures that the internal workings of objects are hidden, and only relevant information is exposed.

For example:

- **Real-world analogy:** When you drive a car, you don't need to understand how the engine works. You only interact with the steering wheel, pedals, and gear shift, while the complex internal details of the car's engine are abstracted away.

Abstract Classes in C#

An **abstract class** is a class that cannot be instantiated directly. It can contain both abstract methods (methods without implementation) and non-abstract methods (methods with implementation). Any class inheriting from an abstract class must implement the abstract methods.

Key Points of Abstract Classes:

- **Cannot be instantiated** directly.
- Can contain both **abstract** and **non-abstract methods**.
- Abstract methods **must** be implemented by derived (child) classes.
- Used to represent base concepts, where common functionality is shared, but each derived class must implement specific behaviors.

Syntax for Abstract Class:

```
abstract class Animal

{       // Abstract method (no implementation)

        public abstract void MakeSound();



        // Non-abstract method (has implementation)

        public void Sleep()

        {

        Console.WriteLine("Sleeping...");

        }

}
```

In this example, the `Animal` class has an abstract method `MakeSound()` that must be implemented by any class that inherits from `Animal`, but it also has a regular method `Sleep()` that can be used directly.

Example of Abstract Class:

```
abstract class Animal {

// Abstract method (must be implemented in derived classes)

public abstract void MakeSound();

 // Regular method public void Sleep()

{ Console.WriteLine("Sleeping..."); }

}
```

```
class Dog : Animal
{// Implementation of the abstract method
        public override void MakeSound()
        {       Console.WriteLine("Bark!");       }
}
class Cat : Animal
{       // Implementation of the abstract method
        public override void MakeSound()
        {       Console.WriteLine("Meow!");
        }
}
```

```
class Program
{         static void Main(string[] args)
          {
          Animal myDog = new Dog();
          myDog.MakeSound();  // Output: Bark!
          myDog.Sleep();      // Output: Sleeping...
          Animal myCat = new Cat();
          myCat.MakeSound();  // Output: Meow!
          myCat.Sleep();      // Output: Sleeping...
          }
}
```

Explanation:

- The `Animal` class is an abstract class that defines an abstract method `MakeSound()`.
- The `Dog` and `Cat` classes inherit from `Animal` and must provide their own implementation of `MakeSound()`.
- The `Sleep()` method is shared by both `Dog` and `Cat` without modification.

Abstract Methods

An **abstract method** is a method that has no body, and it must be implemented in any non-abstract class that derives from the abstract class.

Syntax for Abstract Method:

```
public abstract void MethodName();
```

Abstract methods can only be declared in an abstract class.

These methods provide a contract that derived classes must follow.

Example: Using Abstract Methods

Here’s a more practical example using a banking system:

```
using System;

abstract class Account

{

        public abstract void Withdraw(decimal amount);  // Abstract method

        public void DisplayBalance(decimal balance)

        {

        Console.WriteLine($"The current balance is: {balance:C}");

        }

}
```

```
class SavingsAccount : Account

{               private decimal balance = 1000;

                // Implementing the abstract method

                public override void Withdraw(decimal amount){

                if (amount > balance)

                {

                Console.WriteLine("Insufficient funds.");

                }               else

                {

                balance -= amount;

                Console.WriteLine($"Withdrawal of {amount:C} was successful.");

                DisplayBalance(balance);

                }               }

}
```

```
class Program
{
    static void Main(string[] args)
    {
    Account myAccount = new SavingsAccount();
    myAccount.Withdraw(150);  // Successful withdrawal
    myAccount.Withdraw(1000); // Insufficient funds
    }
}
```

Explanation:

- The `Account` class defines an abstract method `Withdraw()` that each account type must implement.
- The `SavingsAccount` class implements the `Withdraw()` method, providing the specific behavior for a savings account.
- The `DisplayBalance()` method in the abstract class is used to show the account balance, which is shared by all account types.