**Abstraction in C#: Real-Life Example**

**Definition**:  
Abstraction in C# is the process of hiding implementation details and showing only the essential features of an object. It allows you to focus on "what" an object does rather than "how" it does it.

**Real-Life Example**: **ATM (Automated Teller Machine)**

When you interact with an ATM, you perform operations like withdrawing money, checking balance, or depositing money. You don't need to know how the ATM communicates with the bank's server or processes the transaction. This is abstraction — focusing on the functionality without worrying about the internal implementation.

// Abstract class defining the contract

public abstract class ATM

{

public abstract void WithdrawMoney(decimal amount);

public abstract void CheckBalance();

public abstract void DepositMoney(decimal amount);

}

// Concrete implementation of ATM for a specific bank

public class BankATM : ATM

{

private decimal balance;

public BankATM(decimal initialBalance)

{

balance = initialBalance;

}

public override void WithdrawMoney(decimal amount)

{

if (amount > 0 && amount <= balance)

{

balance -= amount;

Console.WriteLine($"You have withdrawn {amount:C}. Remaining balance: {balance:C}");

}

else

{

Console.WriteLine("Insufficient balance or invalid amount.");

}

}

public override void CheckBalance()

{

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

}

public override void DepositMoney(decimal amount)

{

if (amount > 0)

{

balance += amount;

Console.WriteLine($"You have deposited {amount:C}. New balance: {balance:C}");

}

else

{

Console.WriteLine("Invalid deposit amount.");

}

}

}

class Program

{

static void Main(string[] args)

{

// Creating an ATM instance with an initial balance

ATM myATM = new BankATM(500.00m);

// Performing operations

myATM.CheckBalance();

myATM.WithdrawMoney(100.00m);

myATM.DepositMoney(200.00m);

myATM.CheckBalance();

}

}

**Explanation**

1. **Abstract Class (ATM)**:
   * Defines the essential operations (withdraw, check balance, deposit) without revealing the internal implementation.
   * Acts as a blueprint for specific implementations.
2. **Concrete Class (BankATM)**:
   * Implements the abstract methods and defines the specific behavior.
   * Hides internal logic like balance manipulation.
3. **Main Program**:
   * Interacts with the ATM using high-level operations without worrying about the internal details.