**Imagine a Real-World Scenario:**

Think of a shopping scenario where you want to calculate the total price of items in a shopping cart. However, different items might have different types of discounts applied to them. Some items might have no discount, some might have a percentage-based discount, and some might have a fixed amount discount. You want to be able to calculate the total price based on these different discount strategies.

**Strategy Interface (IDiscountStrategy):**

* This is like a contract that defines how different discount strategies should work.
* It has a method ApplyDiscount that calculates the discounted price based on the strategy.

**Concrete Strategy Classes (NoDiscountStrategy, PercentageDiscountStrategy, FixedAmountDiscountStrategy):**

* These are like the actual implementations of the different discount strategies.
* They implement the IDiscountStrategy interface and provide specific ways of applying discounts.

**Context Class (ShoppingCart):**

* This is like the shopping cart where you want to calculate the total price.
* It has a reference to the currently selected discount strategy.
* It uses the selected strategy to calculate the total price based on the items in the cart.

**Client Code (Main Method):**

* This is where you set up everything.
* You create instances of different discount strategies.
* You create instances of the shopping cart, each with a different discount strategy.
* You calculate and print the total prices for different carts.

**In Simple English:**

The Strategy pattern is like having different ways to calculate discounts while shopping. Imagine shopping for items with various types of discounts: no discount, a percentage-based discount, or a fixed amount discount. The Strategy pattern allows you to define different strategies for applying discounts and then use these strategies interchangeably. Just like you can choose different ways to apply discounts while shopping, in your code, you can choose different discount strategies to calculate the total price of items in a shopping cart. Each strategy knows how to apply its own type of discount. This pattern helps keep different discount calculations separate and flexible, without having complex conditional logic for each type of discount.

**Step 1: Define the Strategy Interface (IDiscountStrategy)**

public interface IDiscountStrategy

{

double ApplyDiscount(double originalPrice);

}

The IDiscountStrategy interface defines a method ApplyDiscount that different discount strategies should implement.

**Step 2: Implement Concrete Strategies (NoDiscountStrategy, PercentageDiscountStrategy, FixedAmountDiscountStrategy)**

public class NoDiscountStrategy : IDiscountStrategy

{

public double ApplyDiscount(double originalPrice)

{

return originalPrice;

}

}

public class PercentageDiscountStrategy : IDiscountStrategy

{

**// Implement methods for percentage-based discount**

}

public class FixedAmountDiscountStrategy : IDiscountStrategy

{

**// Implement methods for fixed amount discount**

}

The NoDiscountStrategy, PercentageDiscountStrategy, and FixedAmountDiscountStrategy classes implement the IDiscountStrategy interface.

Each class provides a different implementation of the ApplyDiscount method, representing various discount strategies.

**Step 3: Implement the Context (ShoppingCart)**

public class ShoppingCart

{

private IDiscountStrategy \_discountStrategy;

public ShoppingCart(IDiscountStrategy discountStrategy)

{

\_discountStrategy = discountStrategy;

}

public double CalculateTotalPrice(double originalPrice)

{

return \_discountStrategy.ApplyDiscount(originalPrice);

}

}

The ShoppingCart class represents the context that uses different discount strategies.

It has a field \_discountStrategy that holds a reference to the currently selected discount strategy.

The CalculateTotalPrice method takes the original price and applies the discount strategy to calculate the final price.

**Step 4: Client Code (Main Method)**

static void Main(string[] args)

{

**// Create instances of different discount strategies**

IDiscountStrategy noDiscount = new NoDiscountStrategy();

IDiscountStrategy tenPercentDiscount = new PercentageDiscountStrategy(10);

IDiscountStrategy twentyDollarsDiscount = new FixedAmountDiscountStrategy(20);

**// Create instances of ShoppingCart with different discount strategies**

ShoppingCart cart1 = new ShoppingCart(noDiscount);

ShoppingCart cart2 = new ShoppingCart(tenPercentDiscount);

ShoppingCart cart3 = new ShoppingCart(twentyDollarsDiscount);

**// Calculate and print total prices for different carts**

}

In the Main method, you create instances of different discount strategies and instances of ShoppingCart using these strategies.

**Explanation:**

**Strategy Pattern**: The Strategy pattern allows you to define a family of interchangeable algorithms, encapsulate each algorithm, and make them interchangeable. This way, you can switch the algorithm being used without altering the client code.

**Step 1 and 2:** The IDiscountStrategy interface and the NoDiscountStrategy, PercentageDiscountStrategy, and FixedAmountDiscountStrategy classes represent different strategies for applying discounts.

**Step 3**: The ShoppingCart class acts as the context that uses different discount strategies. Depending on the strategy provided during instantiation, it calculates the total price using the corresponding discount strategy.

**Step 4**: In the Main method, you create instances of different discount strategies and associate them with different ShoppingCart instances. When you calculate the total prices for these carts, the behavior varies based on the selected discount strategy.

The output of the program demonstrates the flexibility of the Strategy pattern by applying different discount strategies to calculate the total prices for different shopping carts. This shows how you can encapsulate different algorithms (discount strategies) and make them interchangeable, which is the essence of the Strategy pattern.