**The Scenario**:

Imagine you have a text editor, and you want to perform actions like inserting and deleting text. You also want to keep track of these actions so that you can undo them if needed.

**Command Design Pattern Explained**:

* **Encapsulating Actions**: The Command design pattern is all about encapsulating actions or operations as objects. Instead of directly calling methods on objects, you create objects that represent actions and can be executed.
* **Command Interface**: You define a common ICommand interface. This interface includes methods like Execute() and Undo(). These methods represent the actions you want to perform and the ability to undo them.
* **Concrete Command Classes**: You create concrete command classes that implement the ICommand interface. These classes define specific actions. In this code, you have InsertTextCommand and DeleteTextCommand classes that represent inserting and deleting text.
* **Receiver Object**: The receiver is the object that performs the actual action. In this case, the TextEditor class holds the text and methods to manipulate it.
* **Command History**: The CommandHistory class maintains a record of executed commands. It also provides a way to execute and undo commands.
* **Using the Command Pattern**: In the Main method, you create instances of the TextEditor and CommandHistory classes. You also create instances of concrete command classes and associate them with the receiver (the text editor). You then execute these commands using the CommandHistory, which manages their execution and undoing.

**In Simple Words**: The Command design pattern is like ordering food at a restaurant. Instead of directly cooking the food, you give orders to the waiter (command objects). The waiter takes your order, brings it to the kitchen (receiver), and the chef (receiver) cooks the food according to the order (command). The waiter also keeps a record of the orders you've placed, so if you change your mind, they can undo or redo the orders.

**Step 1: Define Interfaces and Commands**

In this step, you define the ICommand interface, which includes methods for executing and undoing commands. You also create concrete command classes (InsertTextCommand and DeleteTextCommand) that implement this interface.

internal class Program

{

**// ICommand interface with Execute and Undo methods**

public interface ICommand

{

void Execute();

void Undo();

}

**// Concrete InsertTextCommand class implementing ICommand**

public class InsertTextCommand : ICommand

{

**// ...**

}

**// Concrete DeleteTextCommand class implementing ICommand**

public class DeleteTextCommand : ICommand

{

**// ...**

}

**// ...**

}

**Step 2: Implement TextEditor and CommandHistory**

In this step, you implement the TextEditor class, which has methods for inserting and removing text. You also create a CommandHistory class that maintains a stack of executed commands.

public class TextEditor

{

**// ...**

}

public class CommandHistory

{

**// ...**

}

**Step 3: Using the Command Pattern**

In the Main method, you create instances of the TextEditor and CommandHistory classes. You also create instances of concrete command classes (InsertTextCommand and DeleteTextCommand) and execute them using the CommandHistory.

static void Main(string[] args)

{

TextEditor editor = new TextEditor();

CommandHistory history = new CommandHistory();

ICommand insertCommand = new InsertTextCommand(editor, "Hello, ");

ICommand deleteCommand = new DeleteTextCommand(editor, "Hello, ");

history.ExecuteCommand(insertCommand);

**// ...**

history.ExecuteCommand(deleteCommand);

**// ...**

history.Undo();

**// ...**

}

**Explanation of Command Pattern**:

The Command design pattern is used to encapsulate a request as an object, allowing for parameterization of clients with different requests, queuing of requests, and logging of the requests. It separates the sender (client) of a request from the receiver (the object that performs the action).

In this example,

* ICommand interface defines the basic operations of executing and undoing a command.
* The concrete command classes (InsertTextCommand and DeleteTextCommand) encapsulate specific actions to be performed on the TextEditor. They implement the ICommand interface and hold references to the TextEditor instance and the necessary data for the command.
* The TextEditor class contains methods to modify the text content.
* The CommandHistory class maintains a stack of executed commands and provides methods to execute and undo commands.
* In the Main method, you create instances of the TextEditor and CommandHistory classes. You create concrete command instances and execute them using the command history. This allows you to insert and delete text while keeping track of executed commands and enabling undo functionality.

In this code example, the Command pattern provides a way to encapsulate and manage operations as objects, enabling greater flexibility and control over the execution of actions on the TextEditor.