Communication Between Objects in Command Pattern

In the Command Design Pattern, communication between objects is central to how the pattern operates. Since it's a behavioral pattern, it focuses on how objects interact and how responsibilities are distributed among them. Here's how the communication happens in the text editor example we discussed earlier.

1. Command Interface (`Command`):

- Methods: `execute()` and `undo()`

- This interface defines the contract for executing and undoing commands.

2. Concrete Command Classes:

- `TypeTextCommand`: Represents the action of typing text.

- `DeleteTextCommand`: Represents the action of deleting the last character of text.

3. Receiver (`StringBuilder document`):

- The `StringBuilder` object represents the document being edited. It knows how to append text, delete characters, and more.

4. Invoker (`TextEditor`):

- The `TextEditor` class is responsible for storing and executing commands. It triggers the command's `execute()` method and also keeps track of the commands for undoing actions.

5. Client (`CommandPatternDemo`):

- The client sets up the commands, assigns them to the invoker, and initiates actions.

Communication Flow in the Text Document Example:

1. Client Setup:

- The client (`CommandPatternDemo`) creates a `StringBuilder` instance to represent the document.

- It then creates instances of `TypeTextCommand` and `DeleteTextCommand`, passing the `StringBuilder` as the

receiver to each command.

- The client also creates an instance of `TextEditor` (the invoker) and assigns the command objects to it.

2. Invoker Executes a Command:

- The client sets the `typeHello` command in the invoker and calls `executeCommand()` on the invoker.

- The `TextEditor` (invoker) calls `execute()` on the `typeHello` command.

- The `TypeTextCommand`'s `execute()` method is invoked.

3. Command Executes Action on Receiver:

- Inside the `TypeTextCommand`'s `execute()` method, the command appends "Hello" to the document by calling `document.append(text)`.

4. Result:

- The document now contains the text "Hello".

- The invoker doesn't need to know how the text was added; it simply knows that the command was executed.

5. Undoing the Action:

- To undo the typing action, the client calls `undoCommand()` on the invoker.

- The `TextEditor` (invoker) calls the `undo()` method on the `typeHello` command.

- The `TypeTextCommand`'s `undo()` method deletes the last 5 characters from the document by calling `document.deleteCharAt()`.

6. Deleting the Last Character:

- The client can now set the `deleteLastChar` command and execute it to delete the last character from the document.

- The `DeleteTextCommand`'s `execute()` method is called, which deletes the last character by calling `document.deleteCharAt()` on the receiver (`StringBuilder`).

Summary of Communication in the Command Pattern:

- Client sets up and configures the command objects with their respective receivers (in this case, the document).

- Invoker (TextEditor) triggers the execution of commands. It doesn't need to know the details of the command's actions; it just calls `execute()` or `undo()` on the command.

- Command Objects (TypeTextCommand, DeleteTextCommand) encapsulate all the details needed to perform an action on the receiver. They communicate with the receiver by invoking its methods to execute or undo actions.

- Receiver (StringBuilder document) performs the actual work (e.g., modifying the document) based on the commands it receives.