Command Pattern

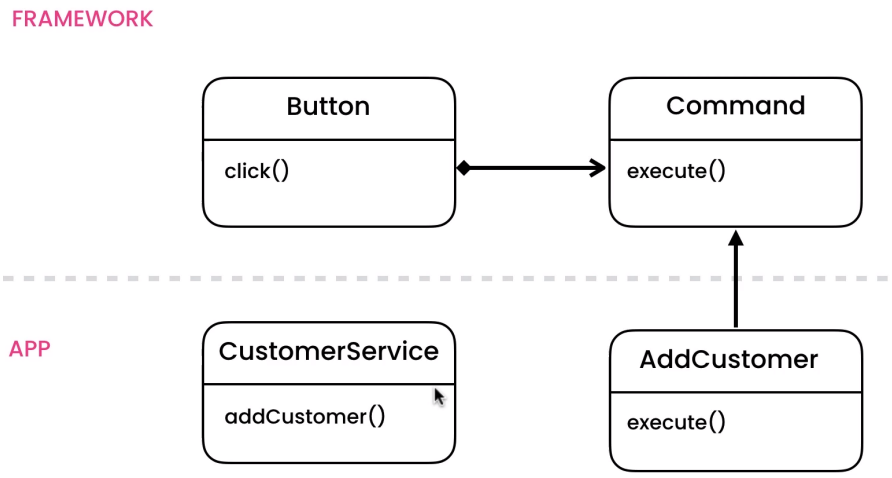
Useful for executing and unexecuting tasks - sets aets an interface which enables action then in the concrete implemenatation the conmmnds are completed or undone e.g. save a document, resize a document, change the color of an image etc

Example Problem - framework for GUI is being developed - framework specifies a Button, Checkbox and and TextBox class allowing other developers to implement them.

A button has a String label variable with getters and setters and a click method for actions. these actions will change dependent on the context which is what the command pattern aims to solve

Solution -

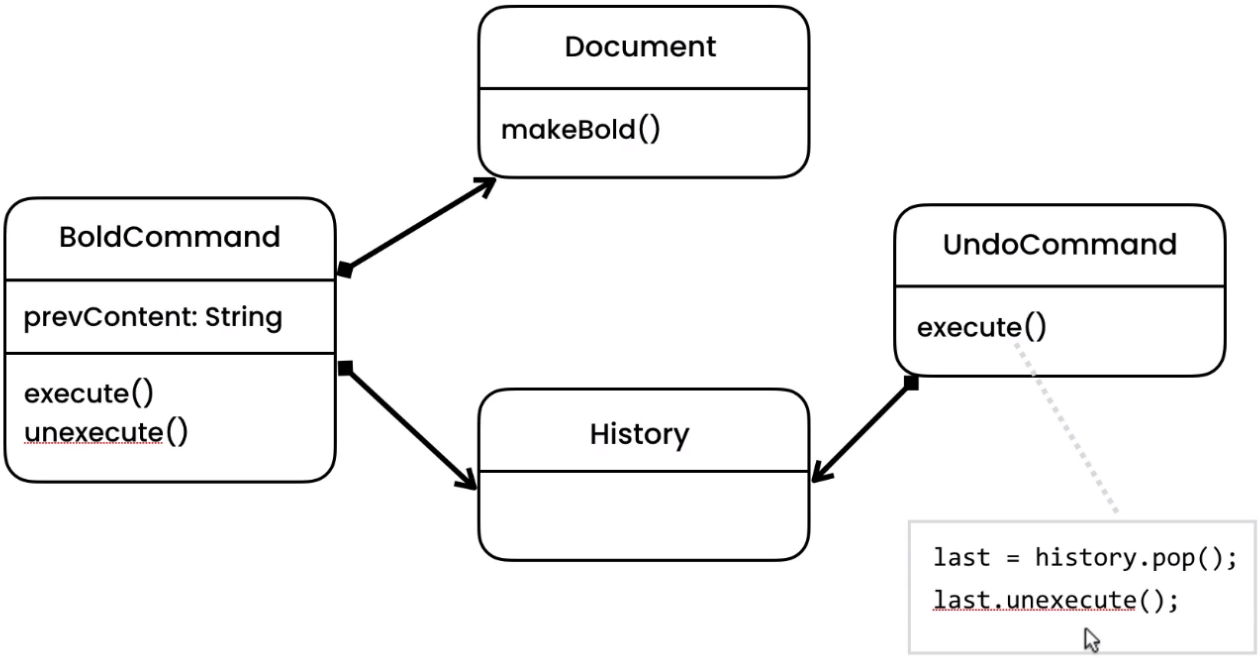
Implement a Command interface in the Button class which delegates that actions when the botton is clicked to the exectute method which is then implemented in the App itself. Allowing The Button (invokker) to talk with the CustomerServioce class (receiver) without being aware of it via the Command iterface which has a concrete implementation in the App - AddCustomer



Button Class	Command Interface	Add Customer	Customer Service

`public class Button {`
`  private String label;`
`  private Command command;`
 	
`  public Button(Command command) {`
`  this.command = command;`
`  }`
 	
`  public void click() {`
`  command.execute();`
`  }`
 	
`  public String getLabel() {`
`  return label;`
`  }`
 	
`  public void setLabel(String label) {`
`  this.label = label;`
`  }`
`}`
	`public interface Command {`
	`  void execute();`
	`}`
		`public class AddCustomerCommand implements Command {`
		`  private CustomerService service;`
		
		`  public AddCustomerCommand(CustomerService service) {`
		`  this.service = service;`
		`  }`
		
		`  @Override`
		`  public void execute() {`
		`  service.addCustomer();`
		`  }`
		`}`
			`public class CustomerService {`
			`  public void addCustomer() {`
			`  System.out.println("Add customer");`
			`  }`
			`}`

Undo operation with the command pattern - below is an implementation of a html text editor which crrently can make text bod and then undo that command




```
prevContent = doc.getContent();
document.makeBold();
history.push(this);
```

```
doc.setContent(prevContent)
```

```
public class BoldCommand implements UndoableCommand {
private String prevContent;
private HtmlDocument document;
private History history;

public BoldCommand(HtmlDocument document, History history) {
this.document = document;
this.history = history;
}

@Override
public void unexecute() {
document.setContent(prevContent);
}

@Override
public void execute() {
prevContent = document.getContent();
document.makeBold();
history.push(this);
}
}
```

```
public class History {
  private Deque<UndoableCommand> commands = new ArrayDeque<>();

 public void push(UndoableCommand command) {
    commands.add(command);
 }

  public UndoableCommand pop() {
    return commands.pop();
 }

  public int size() {
    return commands.size();
 }
}
```

```
public class UndoCommand implements Command {
  private History history;

 public UndoCommand(History history) {
    this.history = history;
 }

  @Override
  public void execute() {
    if (history.size() > 0)

      history.pop().unexecute();
 }
}
```

```
public class HtmlDocument {
  private String content;

 public void makeBold() {
    content = "<b>" + content + "</b>";
 }

  public String getContent() {
    return content;
 }

  public void setContent(String content) {
    this.content = content;
 }

}
```


```
public
interface Comman
d {
  void execute();

}
```

```
public
interface Undoab
leCommand extend
s Command {
  void unexecute
();
}
```

```
public class HistoryStorage {
  public void save(History history) {
    try {
      var fileStream = new FileOutputStream
("history.txt");
 var objectStream = new ObjectOutputStream
(fileStream);
 objectStream.writeObject(history);
 objectStream.close();
 } catch (IOException e) {
      e.printStackTrace();
 }
  }

  public History restore() {
    try {
      var fileStream = new FileInputStream(
"history.txt");
 var objectStream = new ObjectInputStream
(fileStream);
 var history = (History) objectStream.
readObject();
 return history;
 } catch (IOException |
ClassNotFoundException e) {
      e.printStackTrace();
 return null;
 }
  }
}
```

```
public class Main {
  public static void main (String[] args) {

   var history = new History();

var document = new HtmlDocument();

document.setContent("Hello World");



var boldCommand = new BoldCoommand(document,
history);

boldCommand.execute();

//outputs <b>Hello World</b>

System.out.println(document.getContent)



var undoCommand = new UndoCommad(history);

undoCommand.execute();

//outputs Hello World

System.out.println(document.getContent)

};
}
```