Observer Pattern:

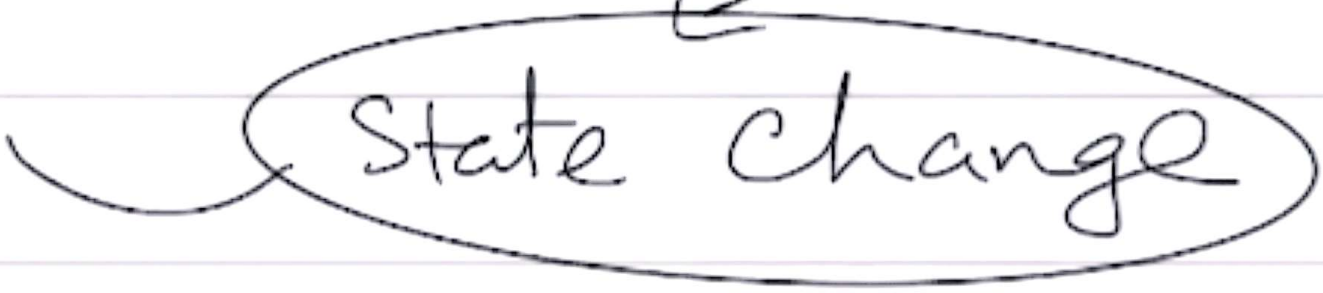
(one kind of behavioral pattern)

one-to-many dependency between

one-to-many dependency between objects

① YT channel (object)

Many users



structure

① Subject: Core component

list of observer (user)

subscribe or unsubscribe

Structure

(I) Subject: Core component

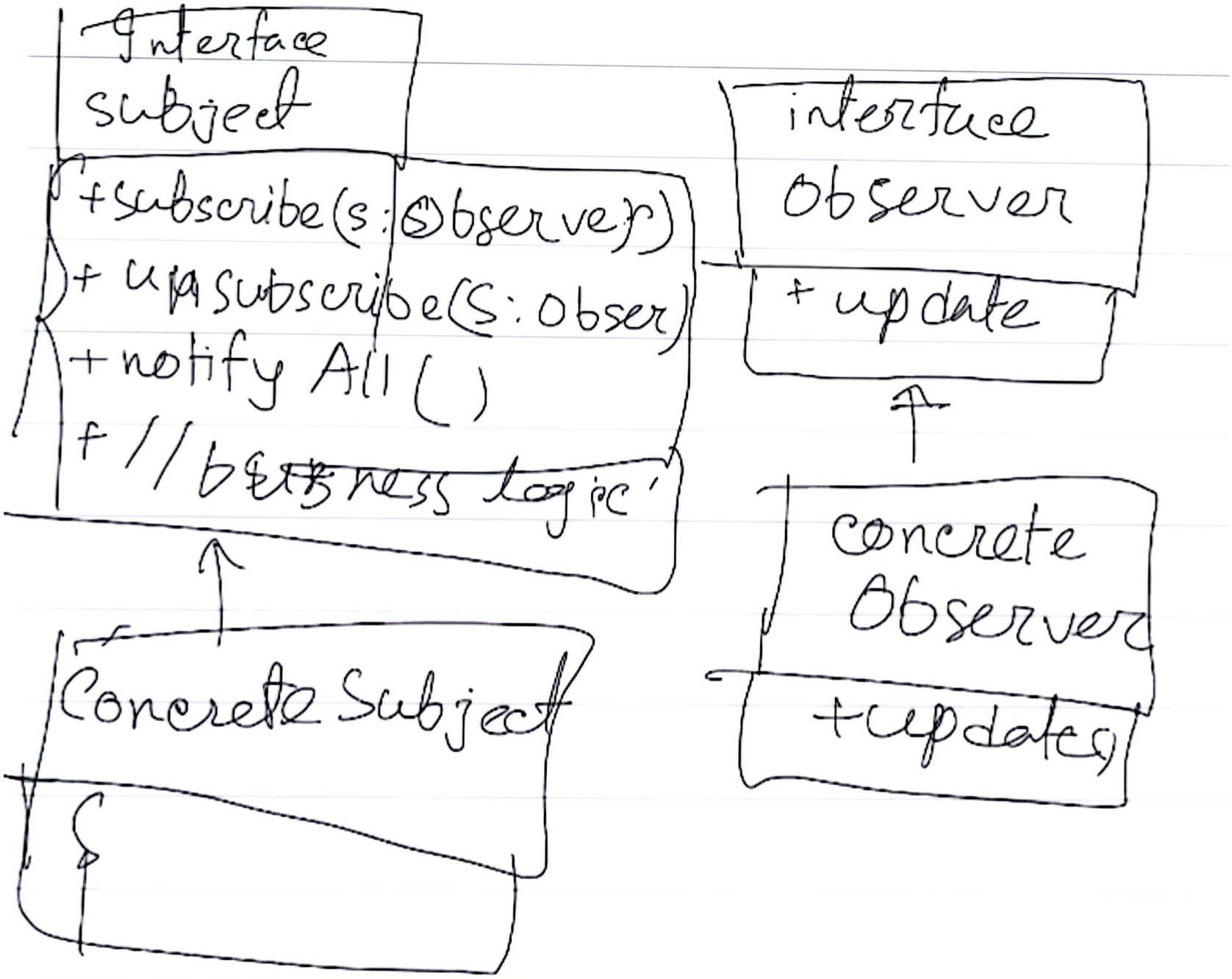
list of observer (user)

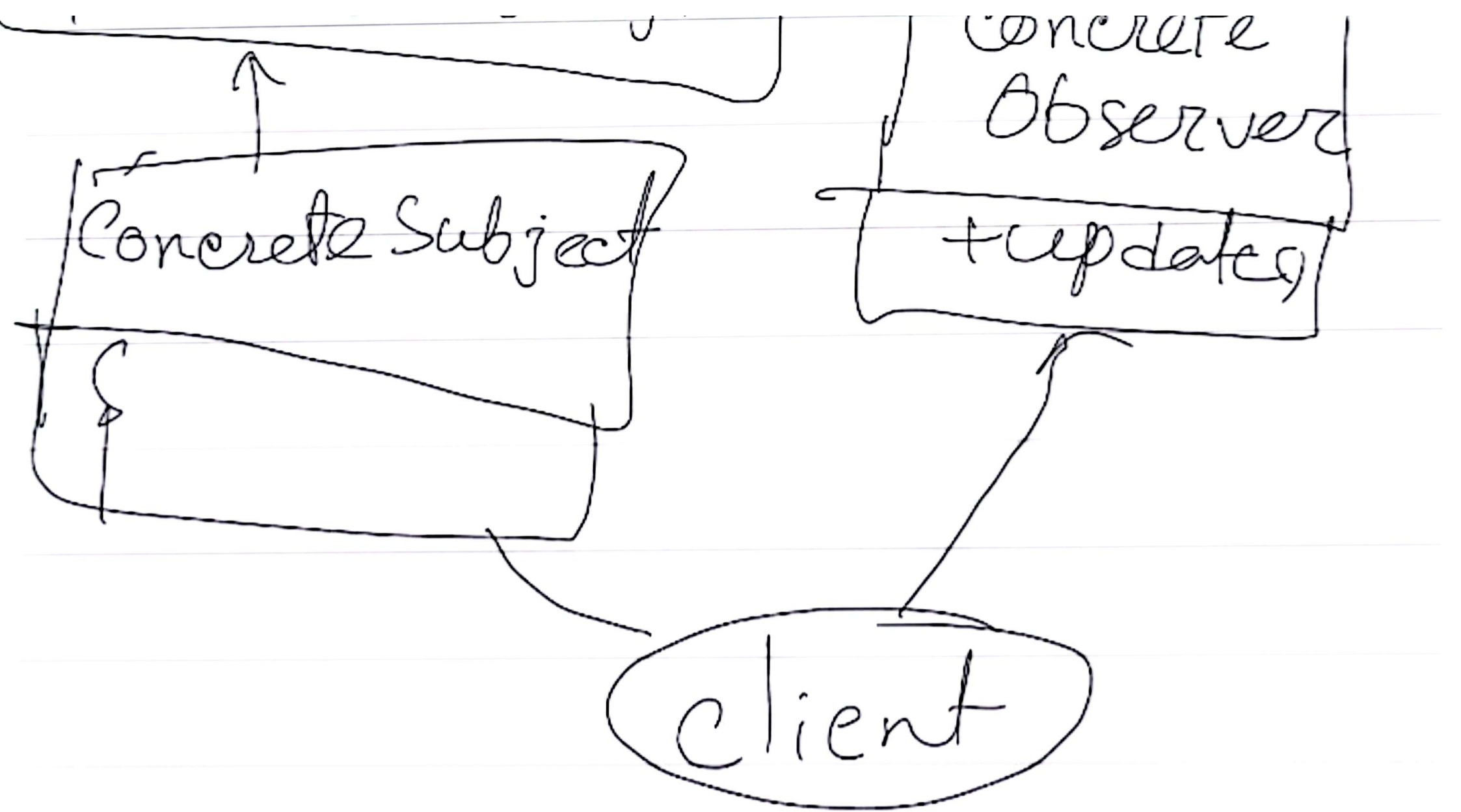
subscribe or unsubscribe

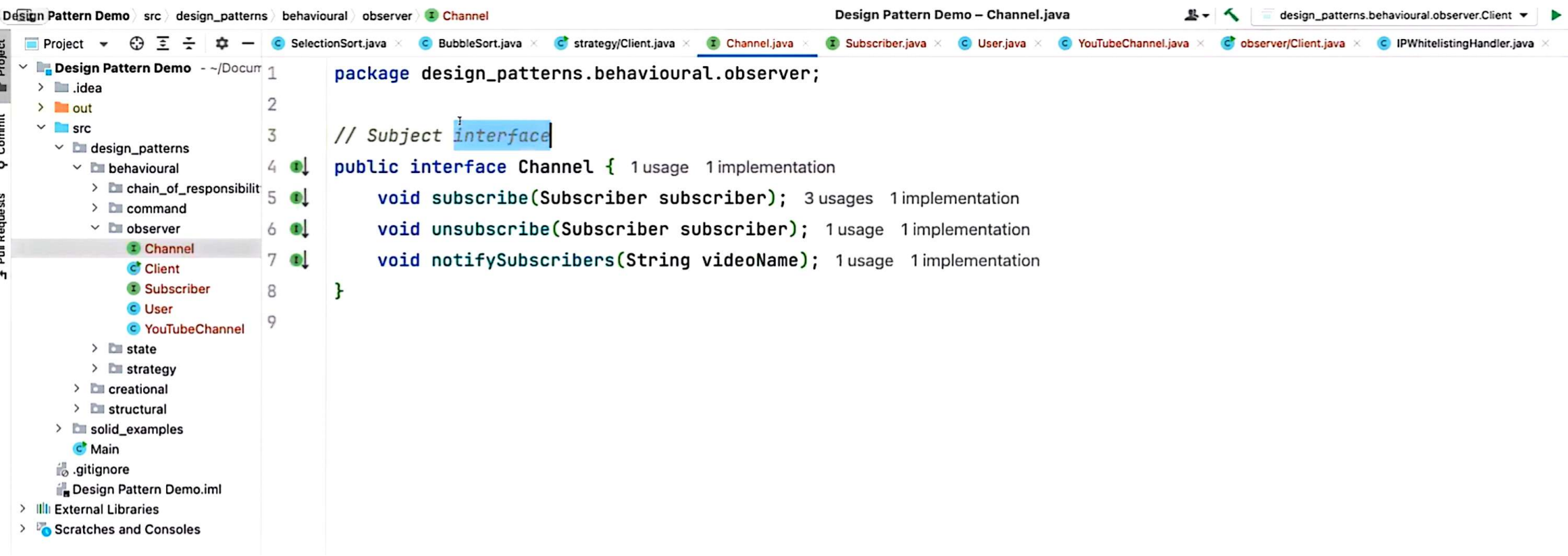
(II) Observer

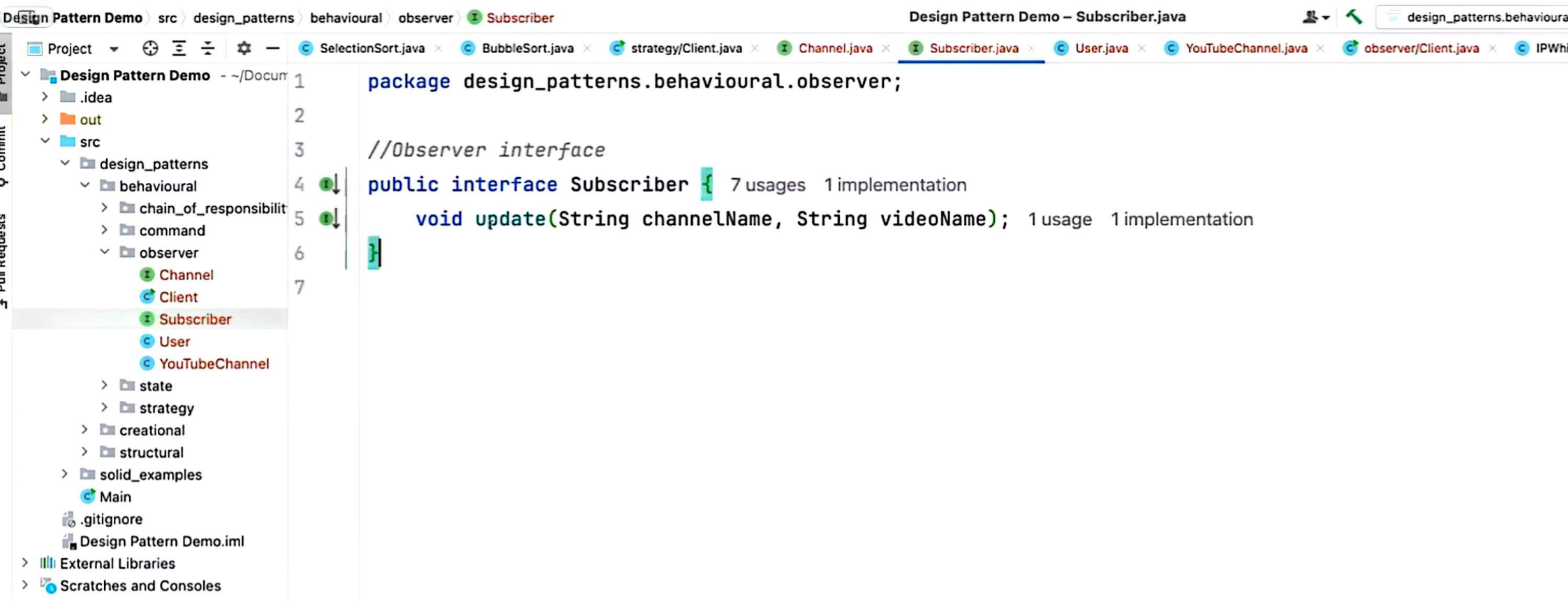
(III) Concrete Subject

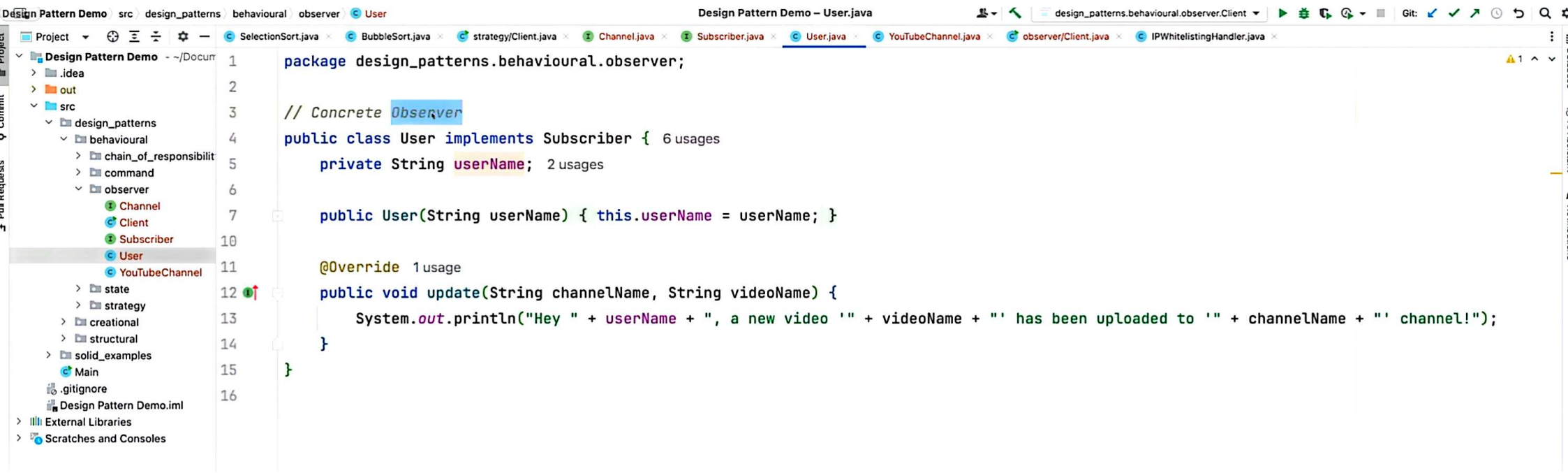
(IV) Concrete Observer

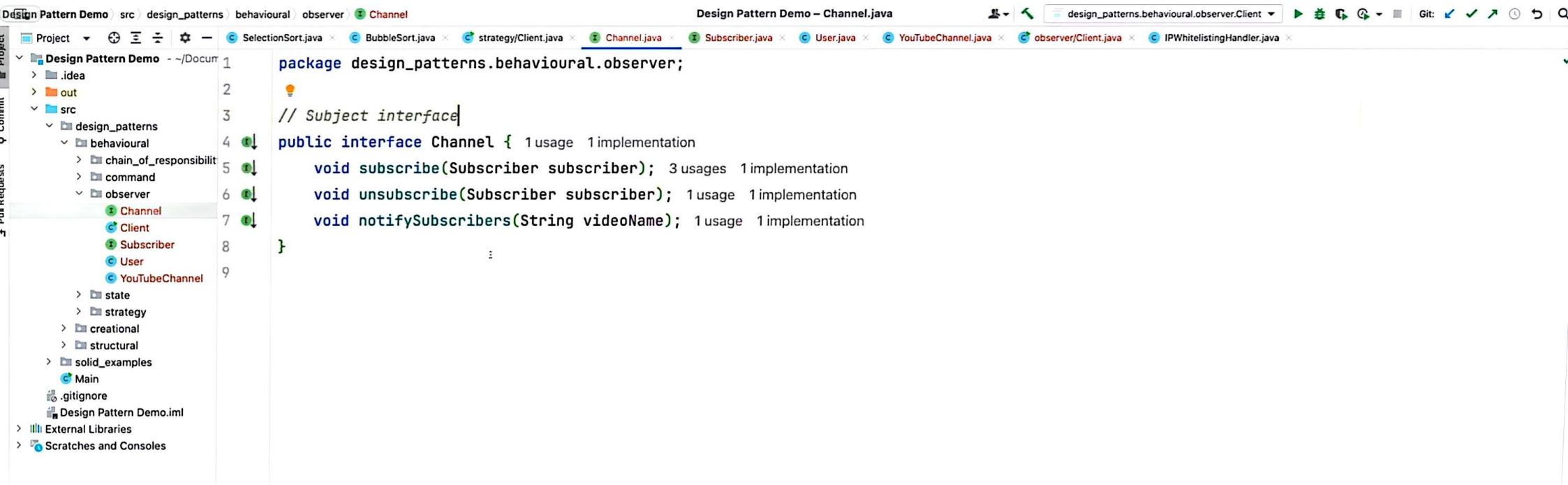

Interface
subject
+subscribe(s:Observer)
+unsubscribe(s:obser)
+notifyAll()
+//business logic'
interface
Observer
+update
concrete
Observer
+update()
Concrete Subject
{


Concrete
Observer
+update()
ConcreteSubject
client


Design Pattern Demo
src
design_patterns
behavioural
observer
Channel
Design Pattern Demo – Channel.java
design_patterns.behavioural.observer.Client
Project
SelectionSort.java
BubbleSort.java
strategy/Client.java
Channel.java
Subscriber.java
User.java
YouTubeChannel.java
observer/Client.java
IPWhitelistingHandler.java
Design Pattern Demo - ~/Docum
.idea
out
src
design_patterns
behavioural
chain_of_responsibilit
command
observer
Channel
Client
Subscriber
User
YouTubeChannel
state
strategy
creational
structural
solid_examples
Main
.gitignore
Design Pattern Demo.iml
External Libraries
Scratches and Consoles
1 package design_patterns.behavioural.observer;
2
3 // Subject interface
4 public interface Channel { 1 usage 1 implementation
5 void subscribe(Subscriber subscriber); 3 usages 1 implementation
6 void unsubscribe(Subscriber subscriber); 1 usage 1 implementation
7 void notifySubscribers(String videoName); 1 usage 1 implementation
8 }
9


Design Pattern Demo › src › design_patterns › behavioural › observer › Subscriber
Design Pattern Demo – Subscriber.java
Project
SelectionSort.java
BubbleSort.java
strategy/Client.java
Channel.java
Subscriber.java
User.java
YouTubeChannel.java
observer/Client.java
Design Pattern Demo - ~/Docum
.idea
out
src
design_patterns
behavioural
chain_of_responsibilit
command
observer
Channel
Client
Subscriber
User
YouTubeChannel
state
strategy
creational
structural
solid_examples
Main
.gitignore
Design Pattern Demo.iml
External Libraries
Scratches and Consoles
1 package design_patterns.behavioural.observer;
2
3 //Observer interface
4 public interface Subscriber { 7 usages 1 implementation
5 void update(String channelName, String videoName); 1 usage 1 implementation
6 }
7


Design Pattern Demo
src
design_patterns
behavioural
observer
User
Design Pattern Demo – User.java
design_patterns.behavioural.observer.Client
Git:
Project
SelectionSort.java
BubbleSort.java
strategy/Client.java
Channel.java
Subscriber.java
User.java
YouTubeChannel.java
observer/Client.java
IPWhitelistingHandler.java
Design Pattern Demo - ~/Docum
.idea
out
src
design_patterns
behavioural
chain_of_responsibilit
command
observer
Channel
Client
Subscriber
User
YouTubeChannel
state
strategy
creational
structural
solid_examples
Main
.gitignore
Design Pattern Demo.iml
External Libraries
Scratches and Consoles
1 package design_patterns.behavioural.observer;
2
3 // Concrete Observer
4 public class User implements Subscriber { 6 usages
5 private String userName; 2 usages
6
7 public User(String userName) { this.userName = userName; }
10
11 @Override 1 usage
12 public void update(String channelName, String videoName) {
13 System.out.println("Hey " + userName + ", a new video '" + videoName + "' has been uploaded to '" + channelName + "' channel!");
14 }
15 }
16


Design Pattern Demo › src › design_patterns › behavioural › observer › Channel
Design Pattern Demo – Channel.java
design_patterns.behavioural.observer.Client
Git:
Project
SelectionSort.java
BubbleSort.java
strategy/Client.java
Channel.java
Subscriber.java
User.java
YouTubeChannel.java
observer/Client.java
IPWhitelistingHandler.java
Design Pattern Demo - ~/Docum
.idea
out
src
design_patterns
behavioural
chain_of_responsibilit
command
observer
Channel
Client
Subscriber
User
YouTubeChannel
state
strategy
creational
structural
solid_examples
Main
.gitignore
Design Pattern Demo.iml
External Libraries
Scratches and Consoles
1 package design_patterns.behavioural.observer;
2
3 // Subject interface
4 public interface Channel { 1 usage 1 implementation
5 void subscribe(Subscriber subscriber); 3 usages 1 implementation
6 void unsubscribe(Subscriber subscriber); 1 usage 1 implementation
7 void notifySubscribers(String videoName); 1 usage 1 implementation
8 }
9

Design Pattern Demo - ~/Docum
.idea
out
src
design_patterns
behavioural
chain_of_responsibilit
command
observer
Channel
Client
Subscriber
User
YouTubeChannel
state
strategy
creational
structural
solid_examples
Main
.gitignore
Design Pattern Demo.iml
External Libraries
Scratches and Consoles

```java
package design_patterns.behavioural.observer;

import java.util.ArrayList;
import java.util.List;

// Concrete Subject
public class YouTubeChannel implements Channel {  no usages
    private List<Subscriber> subscribers = new ArrayList<>();  3 usages
    private String channelName;  3 usages

    public YouTubeChannel(String channelName) {  no usages
        this.channelName = channelName;
    }

    @Override  no usages
    public void subscribe(Subscriber subscriber) {
        subscribers.add(subscriber);
    }

    @Override  no usages
    public void unsubscribe(Subscriber subscriber) {
        subscribers.remove(subscriber);
    }

    @Override  1 usage
    public void notifySubscribers(String videoName) {
        for (Subscriber subscriber : subscribers) {
            subscriber.update(channelName, videoName);
        }
    }
```

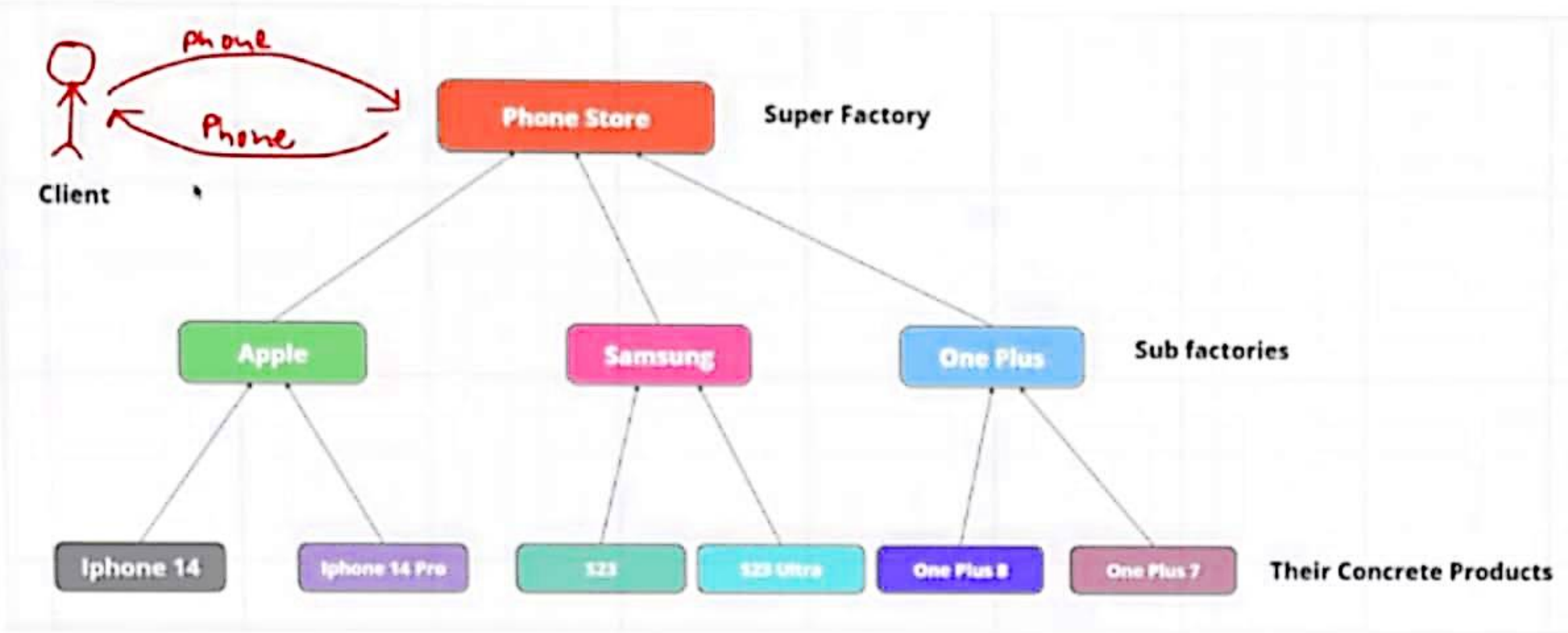
Intent

intention or purpose

Provide an interface for creating families of related or dependent objects without specifying their concrete classes.

- provide a simple creational interface for a complex family of classes
  - Client does not have to know any of those details.
- avoid naming concrete classes
  - Clients use abstract creational interfaces and abstract product interfaces. Concrete classes can be changed without affecting clients.

Example Scenario



- Images are taken from "medium"

Example Scenario

	Chair	Sofa	Coffee Table
Art Deco			
Victorian			
Modern			

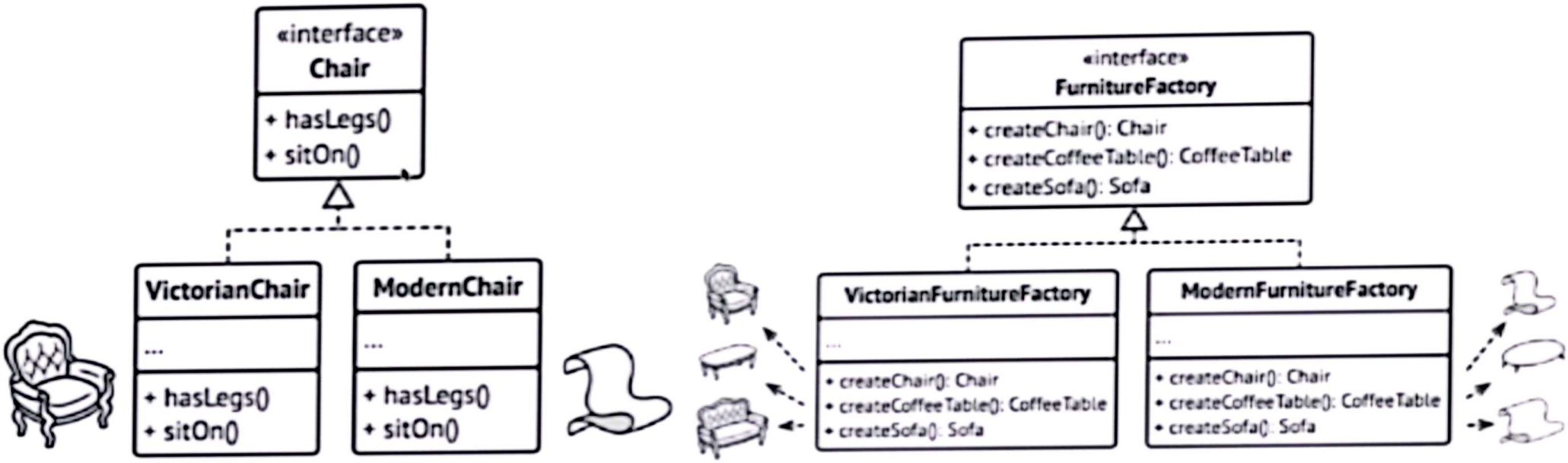
Sowmik Roy has left the meeting

Applicability

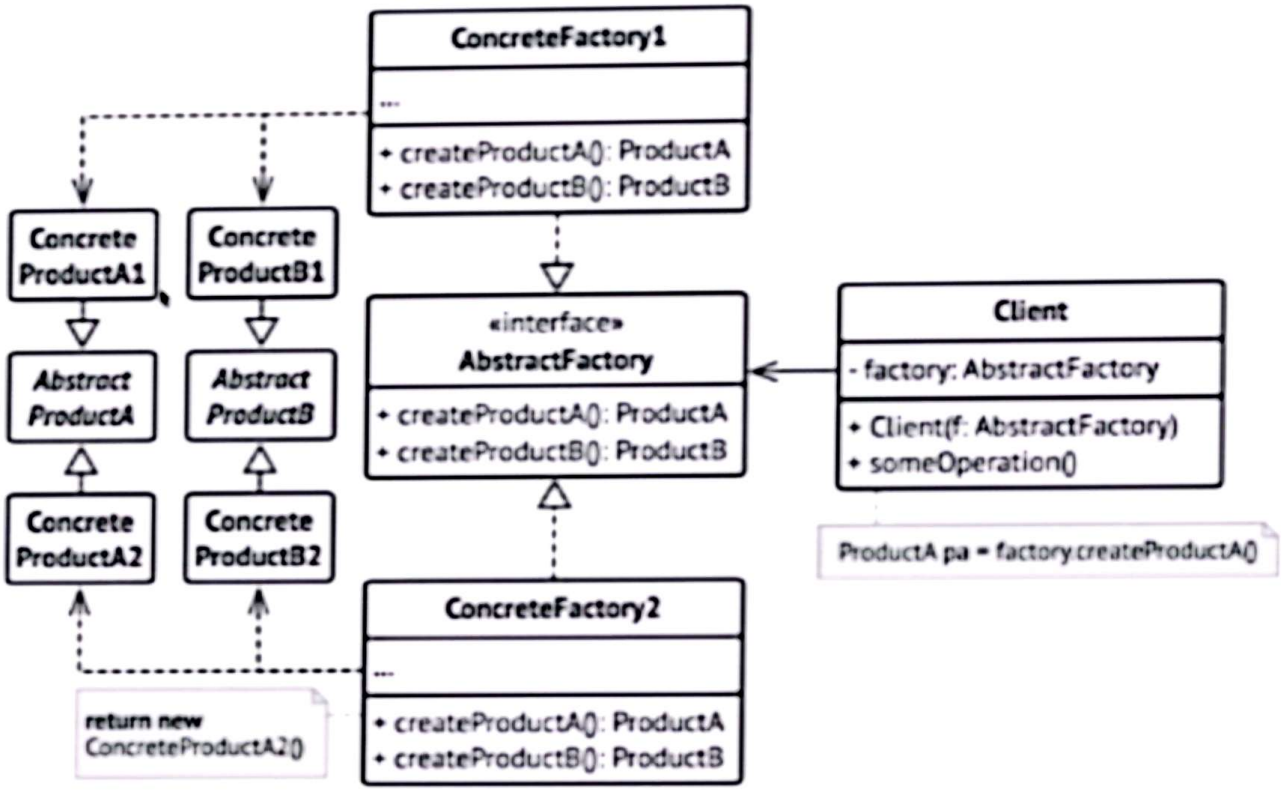
Use the Abstract Factory Pattern if

- clients need to be ignorant of how servers are created, composed, and represented.
- clients need to operate with one of several families of products
- a family of products must be used together, not mixed with products of other families
- provide a library and want to show just the interface, not implementation of the library components.

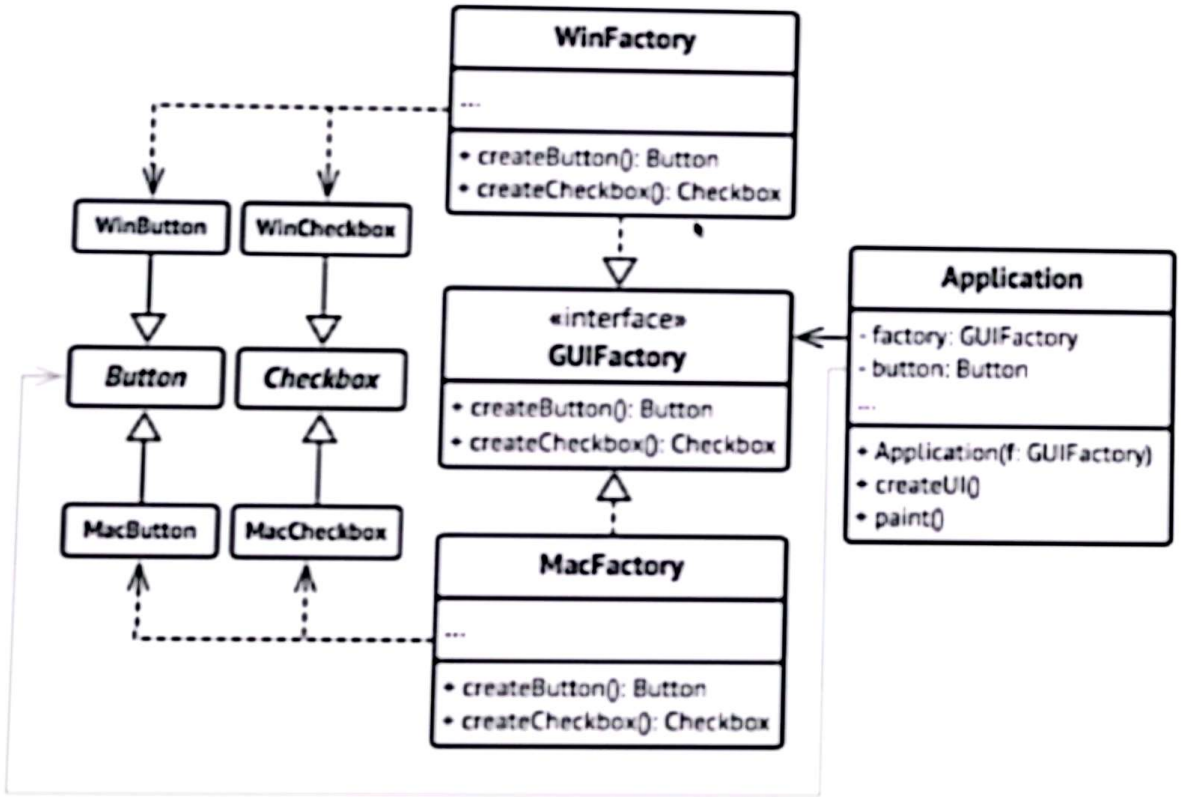
Solution to our problem



Class Diagram



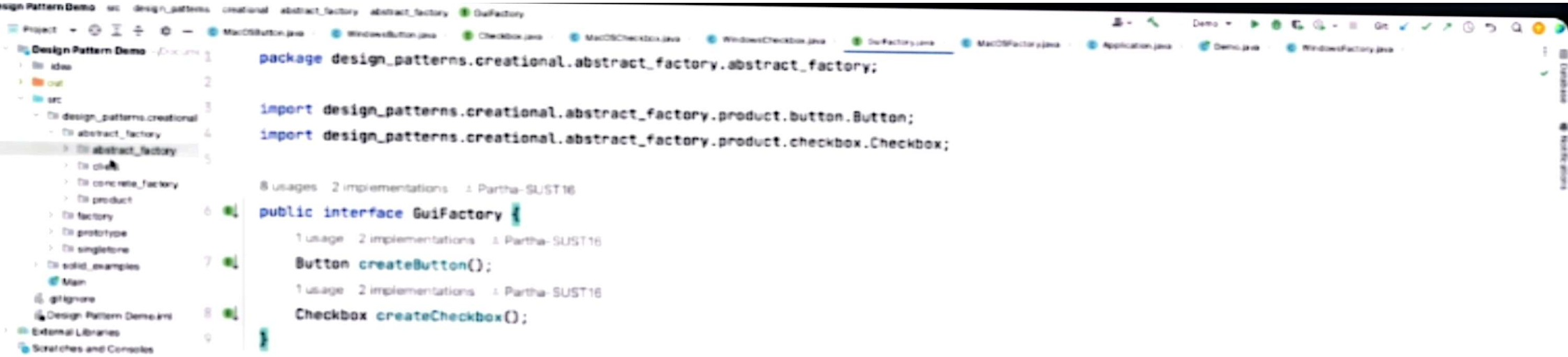
Example UI Class Diagram

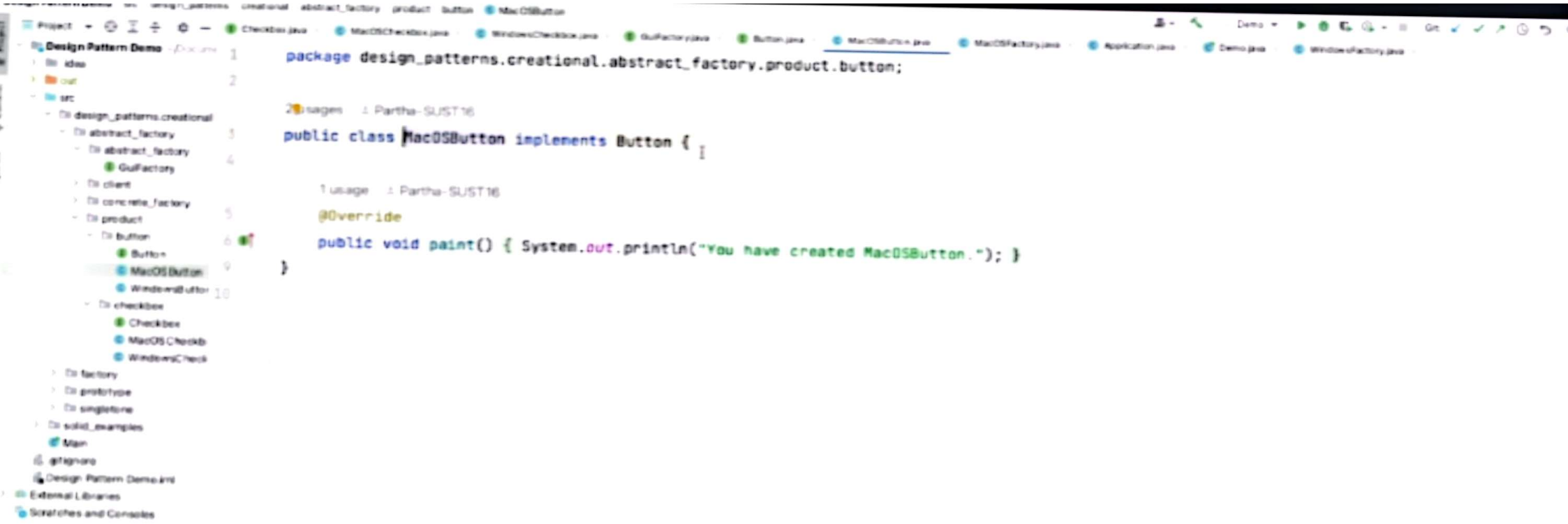


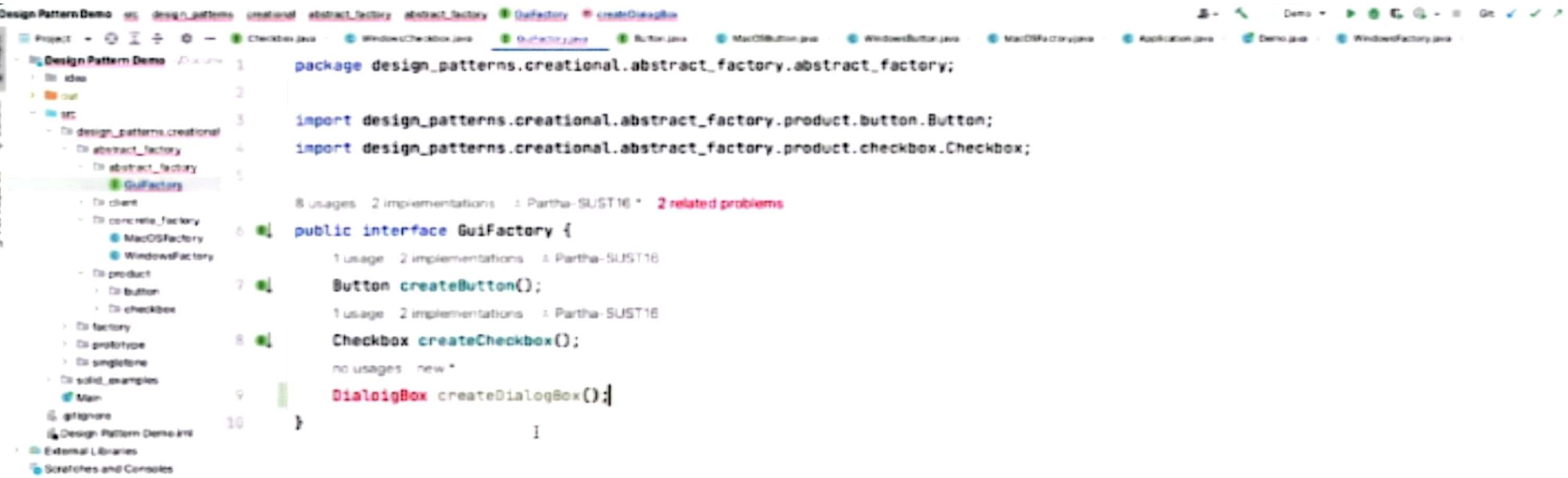
Consequences

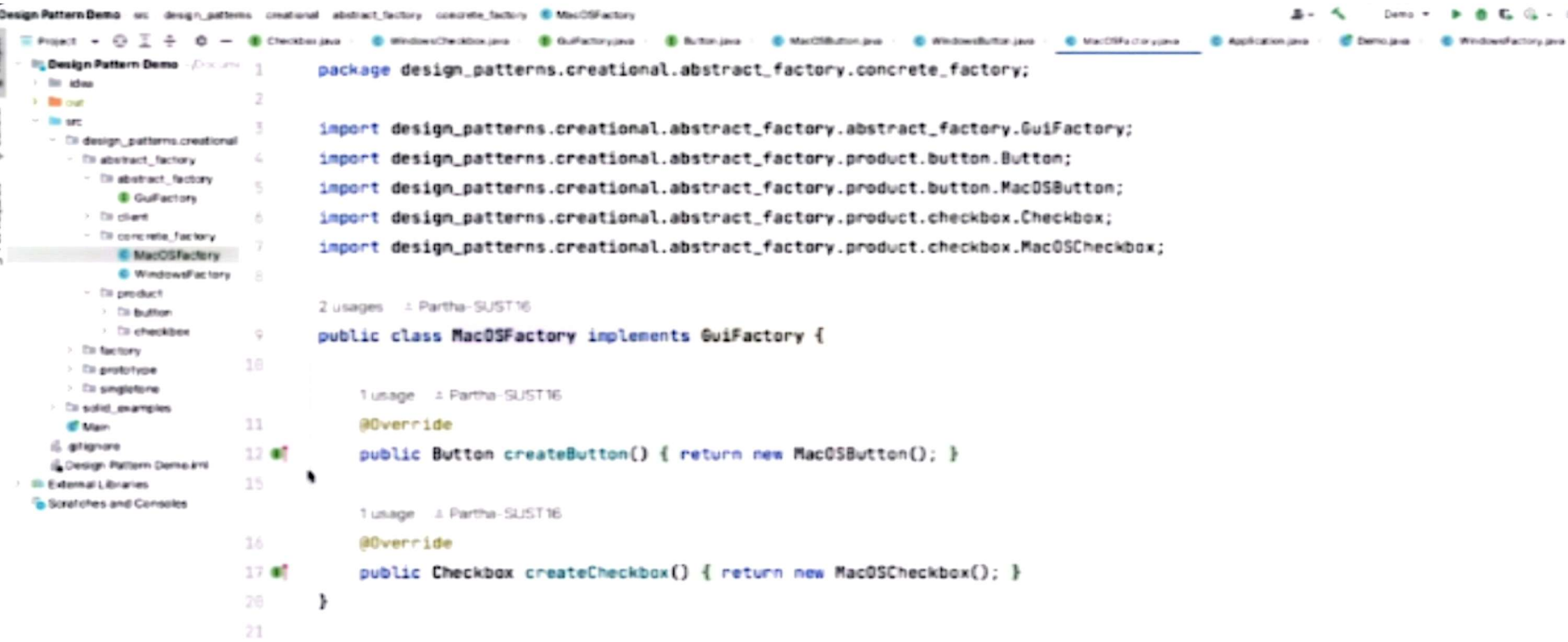
The Abstract Factory Pattern has the following benefits:

- It isolates concrete classes from the client.
  - You use the Abstract Factory to control the classes of objects the client creates.
  - Product names are isolated in the implementation of the Concrete Factory, clients use the instances through their abstract interfaces
- Exchanging defined product families is easy.
  - None of the client code breaks because the abstract interfaces don't change.
  - Because the abstract factory creates a complete family of products, the whole product family changes when the concrete factory is changed.
- It promotes consistency among products.
  - It is the concrete factory's job to make sure that the right products are used together.


Design Pattern Demo
Project
Design Pattern Demo
idea
out
src
design_patterns.creational
abstract_factory
abstract_factory
client
concrete_factory
product
factory
prototype
singletone
solid_examples
Main
.gitignore
Design Pattern Demo.iml
External Libraries
Scratches and Consoles
package design_patterns.creational.abstract_factory.abstract_factory;
import design_patterns.creational.abstract_factory.product.button.Button;
import design_patterns.creational.abstract_factory.product.checkbox.Checkbox;
8 usages 2 implementations Partha- SUST16
public interface GuiFactory {
1 usage 2 implementations Partha- SUST16
Button createButton();
1 usage 2 implementations Partha- SUST16
Checkbox createCheckbox();
}


Design Pattern Demo
idea
out
src
design_patterns.creational
abstract_factory
abstract_factory
GuiFactory
client
concrete_factory
product
button
Button
MacOSButton
WindowsButton
checkbox
Checkbox
MacOSCheckb
WindowsCheck
factory
prototype
singletone
solid_examples
Main
gitignore
Design Pattern Demo.iml
External Libraries
Scratches and Consoles
package design_patterns.creational.abstract_factory.product.button;
2 usages Partha-SUST'16
public class MacOSButton implements Button {
1 usage Partha-SUST'16
@Override
public void paint() { System.out.println("You have created MacOSButton."); }
}

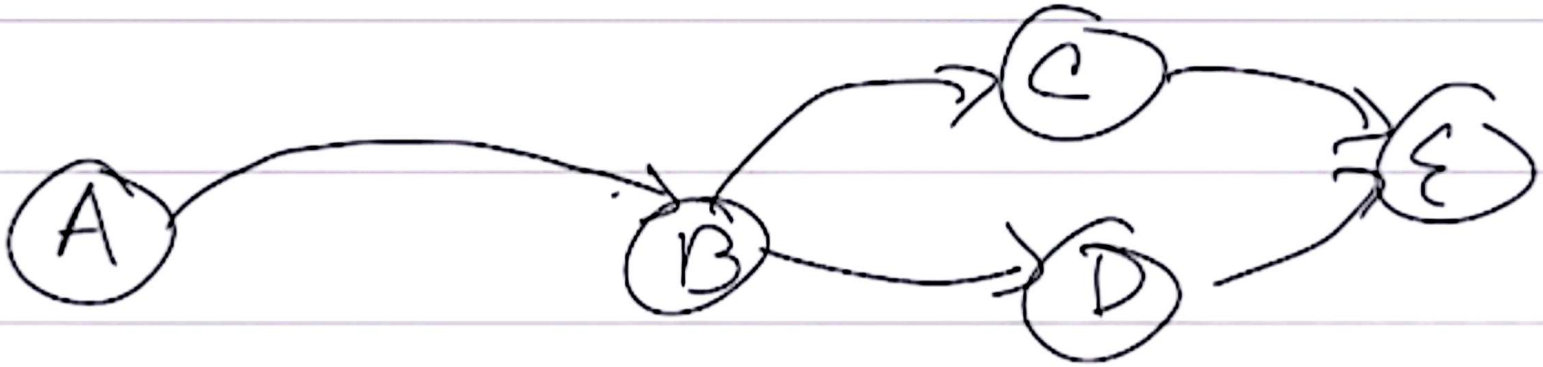

Design Pattern Demo
design_patterns.creational.abstract_factory.abstract_factory;
import design_patterns.creational.abstract_factory.product.button.Button;
import design_patterns.creational.abstract_factory.product.checkbox.Checkbox;
8 usages 2 implementations Partha-SUST16 * 2 related problems
public interface GuiFactory {
1 usage 2 implementations Partha-SUST16
Button createButton();
1 usage 2 implementations Partha-SUST16
Checkbox createCheckbox();
no usages new *
DialogBox createDialogBox();
}


Design Pattern Demo
Project
Design Pattern Demo
idea
out
src
design_patterns.creational
abstract_factory
abstract_factory
GuiFactory
client
concrete_factory
MacOSFactory
WindowsFactory
product
button
checkbox
factory
prototype
singletone
solid_examples
Main
.gitignore
Design Pattern Demo.iml
External Libraries
Scratches and Consoles
1 package design_patterns.creational.abstract_factory.concrete_factory;
2
3 import design_patterns.creational.abstract_factory.abstract_factory.GuiFactory;
4 import design_patterns.creational.abstract_factory.product.button.Button;
5 import design_patterns.creational.abstract_factory.product.button.MacOSButton;
6 import design_patterns.creational.abstract_factory.product.checkbox.Checkbox;
7 import design_patterns.creational.abstract_factory.product.checkbox.MacOSCheckbox;
8
2 usages Partha-SUST16
9 public class MacOSFactory implements GuiFactory {
10
1 usage Partha-SUST16
11 @Override
12 public Button createButton() { return new MacOSButton(); }
15
1 usage Partha-SUST16
16 @Override
17 public Checkbox createCheckbox() { return new MacOSCheckbox(); }
20 }
21

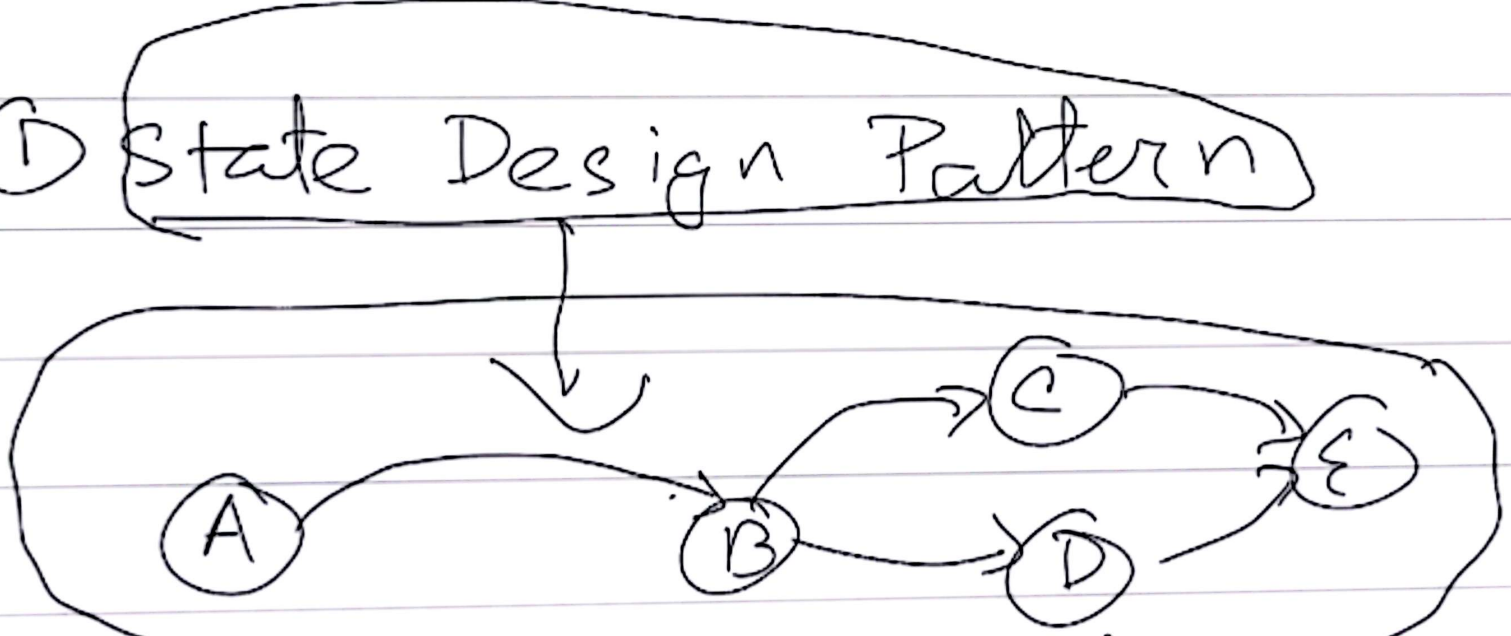

Design Pattern Demo
Project
Design Pattern Demo
idea
out
src
design_patterns.creational
abstract_factory
abstract_factory
GuiFactory
client
Application
Demo
concrete_factory
product
button
checkbox
factory
prototype
singleton
solid_examples
Main
.gitignore
Design Pattern Demo.iml
External Libraries
Scratches and Consoles
Demo
configureApplication
Demo
Checkbox.java
WindowsCheckbox.java
GuiFactory.java
Button.java
MacOSButton.java
WindowsButton.java
MacOSFactory.java
Application.java
Demo.java
WindowsFactory.java
1 package design_patterns.creational.abstract_factory.client;
2
3 import ...
6
Partha SUST16
7 public class Demo {
1 usage Partha SUST16
8 private static Application configureApplication() {
9 Application app;
10 GuiFactory factory;
11 String osName = System.getProperty("os.name").toLowerCase();
12 if (osName.contains("mac")) {
13 factory = new MacOSFactory();
14 } else {
15 factory = new WindowsFactory();
16 }
17 app = new Application(factory);
18 return app;
19 }
20
Run
Demo
/Users/pp_paul/Library/Java/JavaVirtualMachines/openjdk-19.0.1/Contents/Home/bin/java -javaagent:/Users/pp_paul/Applications/IntelliJ IDEA Ultimate.
You have created MacOSButton.
You have created MacOSCheckbox.
Process finished with exit code 0

Behavioral Design Pattern:

1. State Design Pattern



① State Design Pattern



Finite-state machine

at any given time

prog

at every state prog. behaviour is different

at every state prog. behaviour is different

* switch to another state

* Google doc / publish

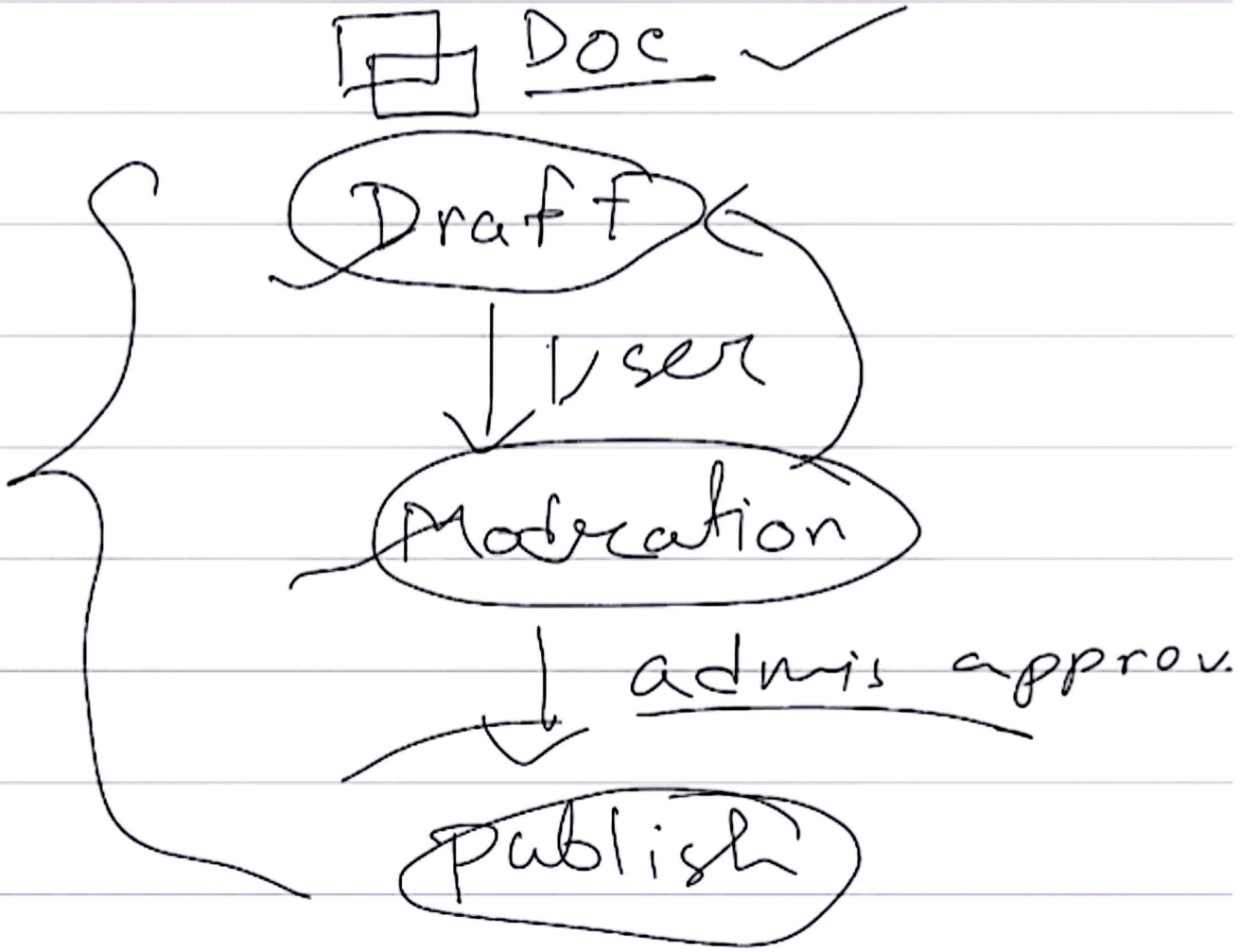
Document

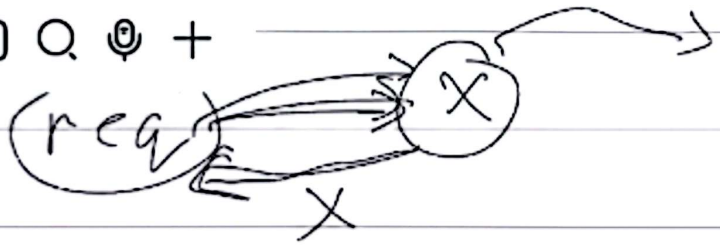
(I) Draft (II) Moderation

(III) Publish

(I) Draft (II) Moderation

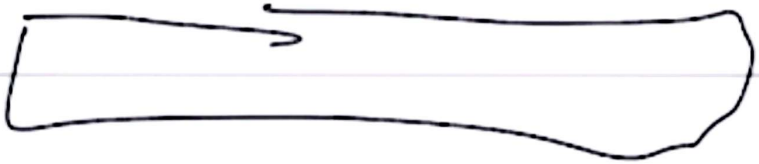
(III) Publish → public user is admin.



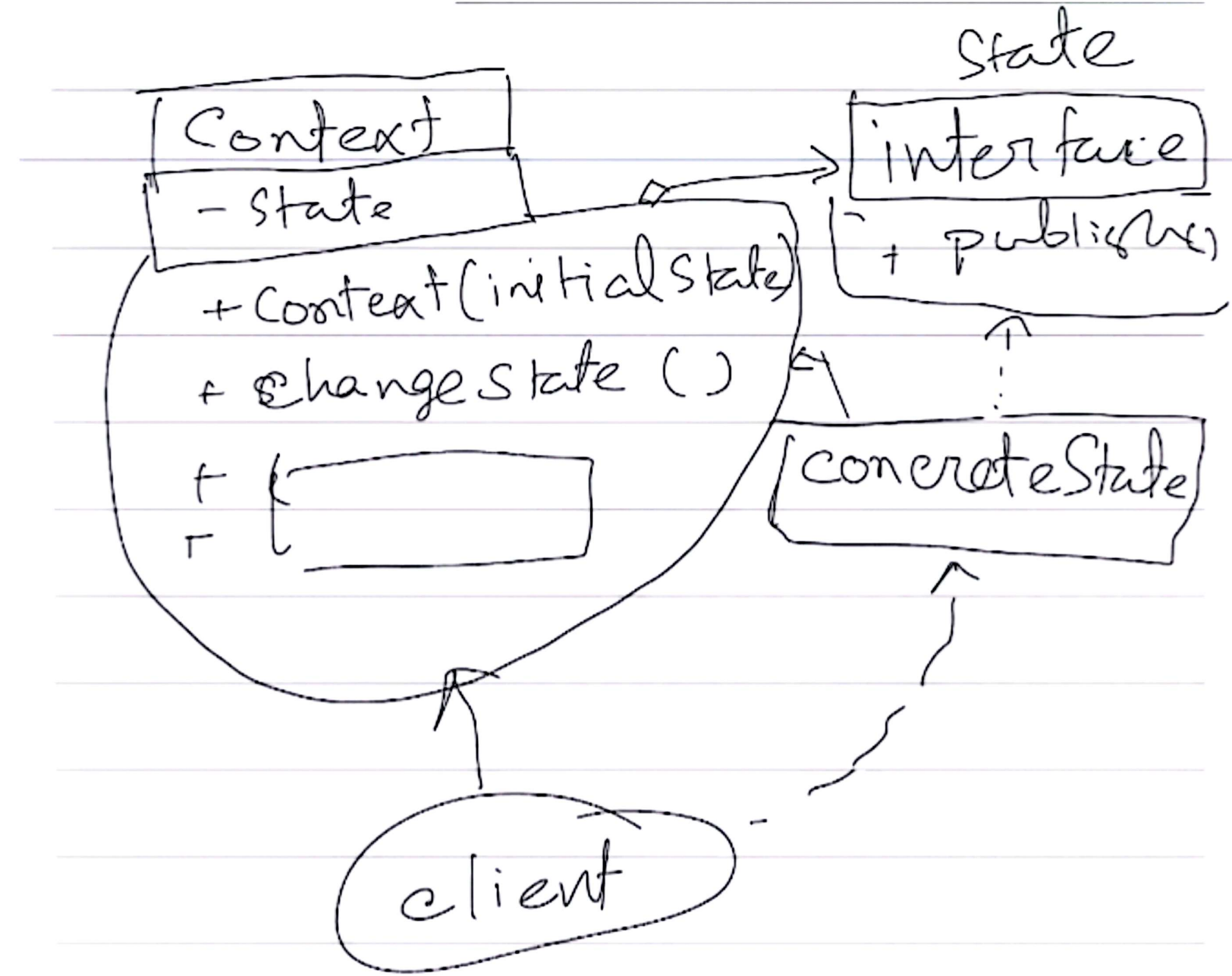


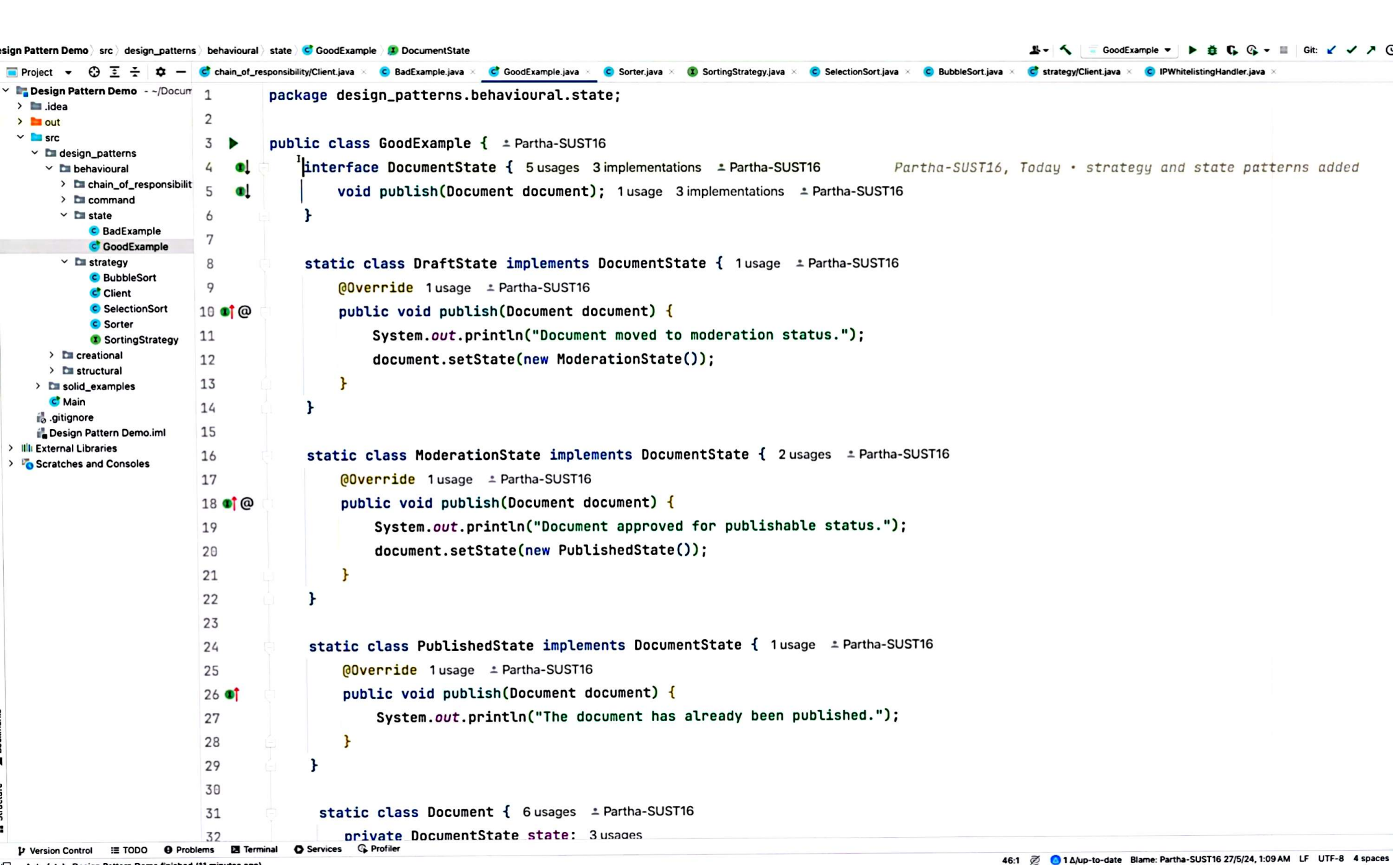

```
class Doc
    state : string
    publish ( )

    if (state == "draft")
        state = "moder"
    else if (state == "mod"
        if (currentuser ==
            "admin")
            state = publish
    else if (state = "pub")
```

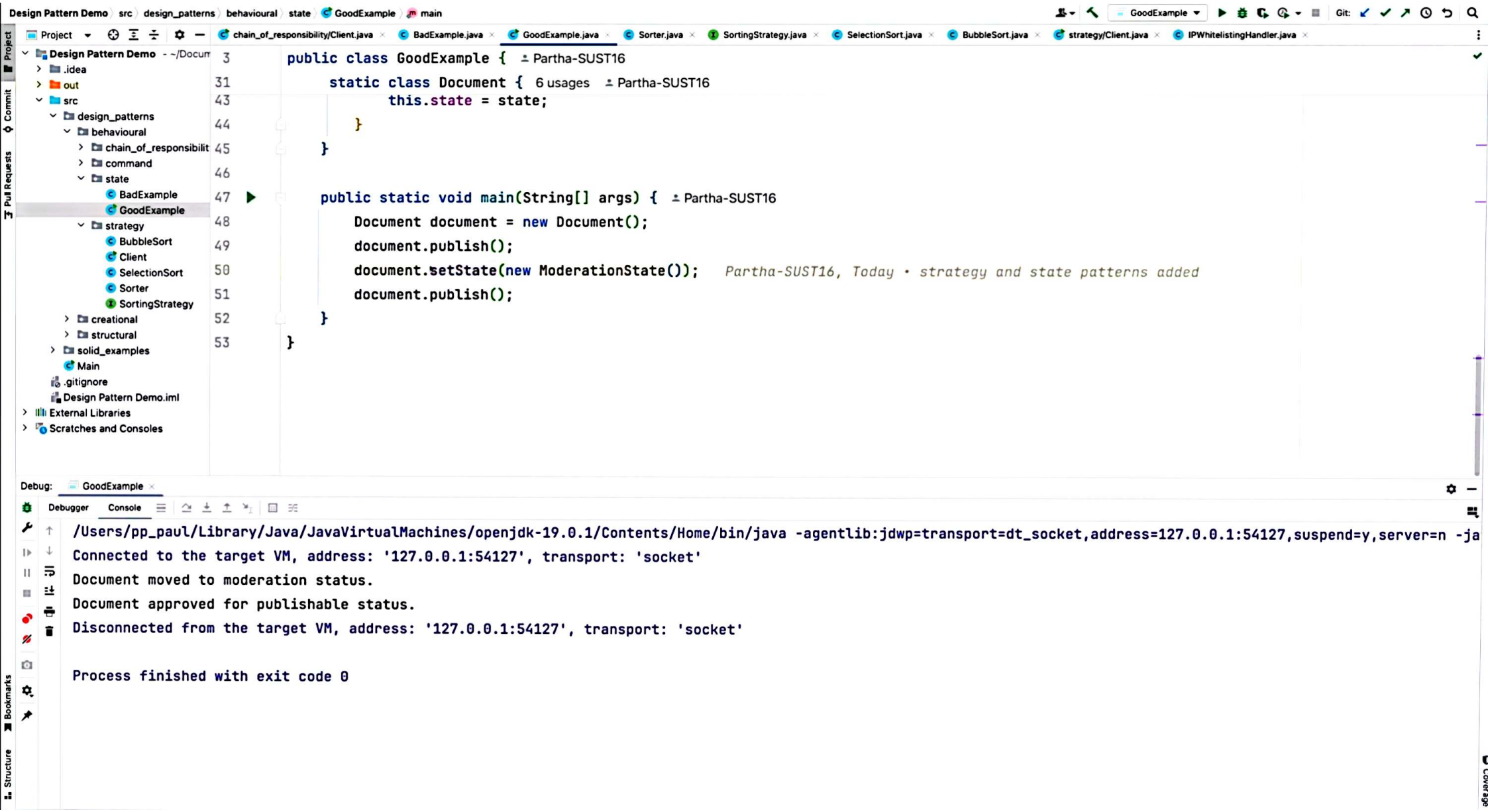
```
        state = "moder"
    } else if (state == "mod
        if (currentuser ==
            "admin")
                state = publish
    else if (state = "pub")
    }
```

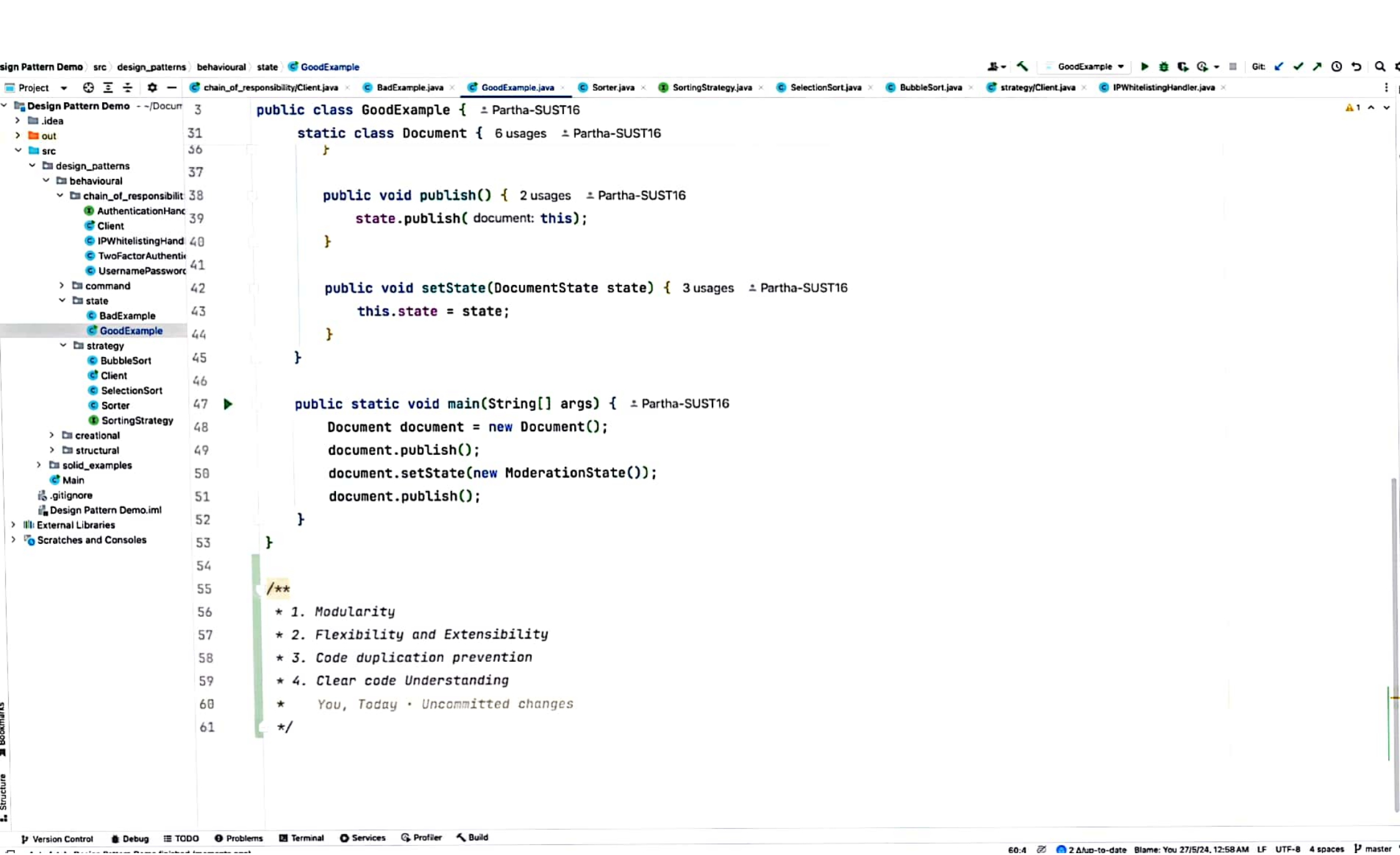


"code Smell"

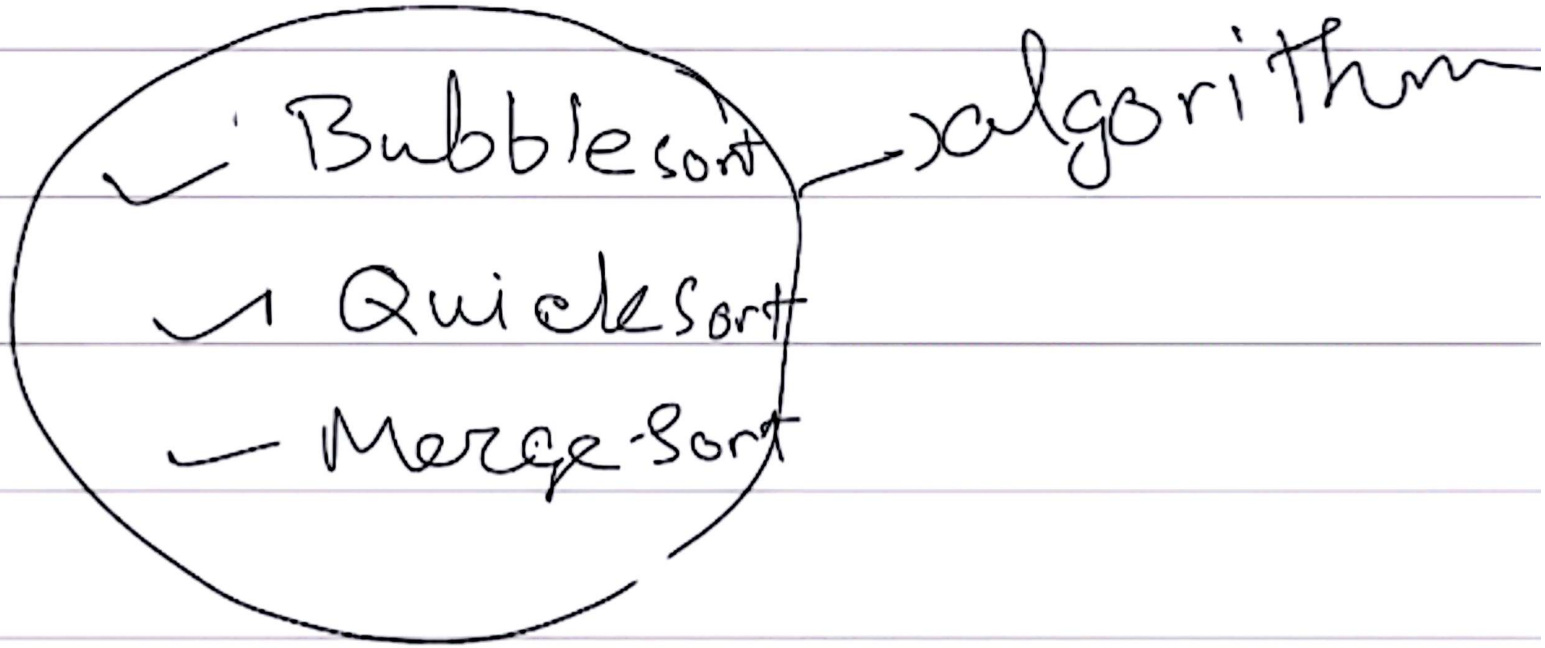

State
Context
- State
+ Context(initialState)
+ changeState()
interface
+ publicFn()
concreteState
client


esign Pattern Demo 〉 src 〉 design_patterns 〉 behavioural 〉 state 〉 GoodExample 〉 DocumentState
GoodExample
Git:
Project
chain_of_responsibility/Client.java
BadExample.java
GoodExample.java
Sorter.java
SortingStrategy.java
SelectionSort.java
BubbleSort.java
strategy/Client.java
IPWhitelistingHandler.java
Design Pattern Demo - ~/Docum
.idea
out
src
design_patterns
behavioural
chain_of_responsibilit
command
state
BadExample
GoodExample
strategy
BubbleSort
Client
SelectionSort
Sorter
SortingStrategy
creational
structural
solid_examples
Main
.gitignore
Design Pattern Demo.iml
External Libraries
Scratches and Consoles
1 package design_patterns.behavioural.state;
2
3 public class GoodExample { Partha-SUST16
4 interface DocumentState { 5 usages 3 implementations Partha-SUST16 Partha-SUST16, Today • strategy and state patterns added
5 void publish(Document document); 1 usage 3 implementations Partha-SUST16
6 }
7
8 static class DraftState implements DocumentState { 1 usage Partha-SUST16
9 @Override 1 usage Partha-SUST16
10 @ public void publish(Document document) {
11 System.out.println("Document moved to moderation status.");
12 document.setState(new ModerationState());
13 }
14 }
15
16 static class ModerationState implements DocumentState { 2 usages Partha-SUST16
17 @Override 1 usage Partha-SUST16
18 @ public void publish(Document document) {
19 System.out.println("Document approved for publishable status.");
20 document.setState(new PublishedState());
21 }
22 }
23
24 static class PublishedState implements DocumentState { 1 usage Partha-SUST16
25 @Override 1 usage Partha-SUST16
26 public void publish(Document document) {
27 System.out.println("The document has already been published.");
28 }
29 }
30
31 static class Document { 6 usages Partha-SUST16
32 private DocumentState state; 3 usages
Version Control
TODO
Problems
Terminal
Services
Profiler
46:1
1 Δ/up-to-date
Blame: Partha-SUST16 27/5/24, 1:09 AM
LF
UTF-8
4 spaces


Design Pattern Demo › src › design_patterns › behavioural › state › GoodExample › main
GoodExample
Git:
chain_of_responsibility/Client.java
BadExample.java
GoodExample.java
Sorter.java
SortingStrategy.java
SelectionSort.java
BubbleSort.java
strategy/Client.java
IPWhitelistingHandler.java
Project
Design Pattern Demo
.idea
out
src
design_patterns
behavioural
chain_of_responsibilit
command
state
BadExample
GoodExample
strategy
BubbleSort
Client
SelectionSort
Sorter
SortingStrategy
creational
structural
solid_examples
Main
.gitignore
Design Pattern Demo.iml
External Libraries
Scratches and Consoles
3 public class GoodExample { Partha-SUST16
31 static class Document { 6 usages Partha-SUST16
43 this.state = state;
44 }
45 }
46
47 public static void main(String[] args) { Partha-SUST16
48 Document document = new Document();
49 document.publish();
50 document.setState(new ModerationState()); Partha-SUST16, Today • strategy and state patterns added
51 document.publish();
52 }
53 }
Debug:
GoodExample
Debugger
Console
/Users/pp_paul/Library/Java/JavaVirtualMachines/openjdk-19.0.1/Contents/Home/bin/java -agentlib:jdwp=transport=dt_socket,address=127.0.0.1:54127,suspend=y,server=n -ja
Connected to the target VM, address: '127.0.0.1:54127', transport: 'socket'
Document moved to moderation status.
Document approved for publishable status.
Disconnected from the target VM, address: '127.0.0.1:54127', transport: 'socket'
Process finished with exit code 0

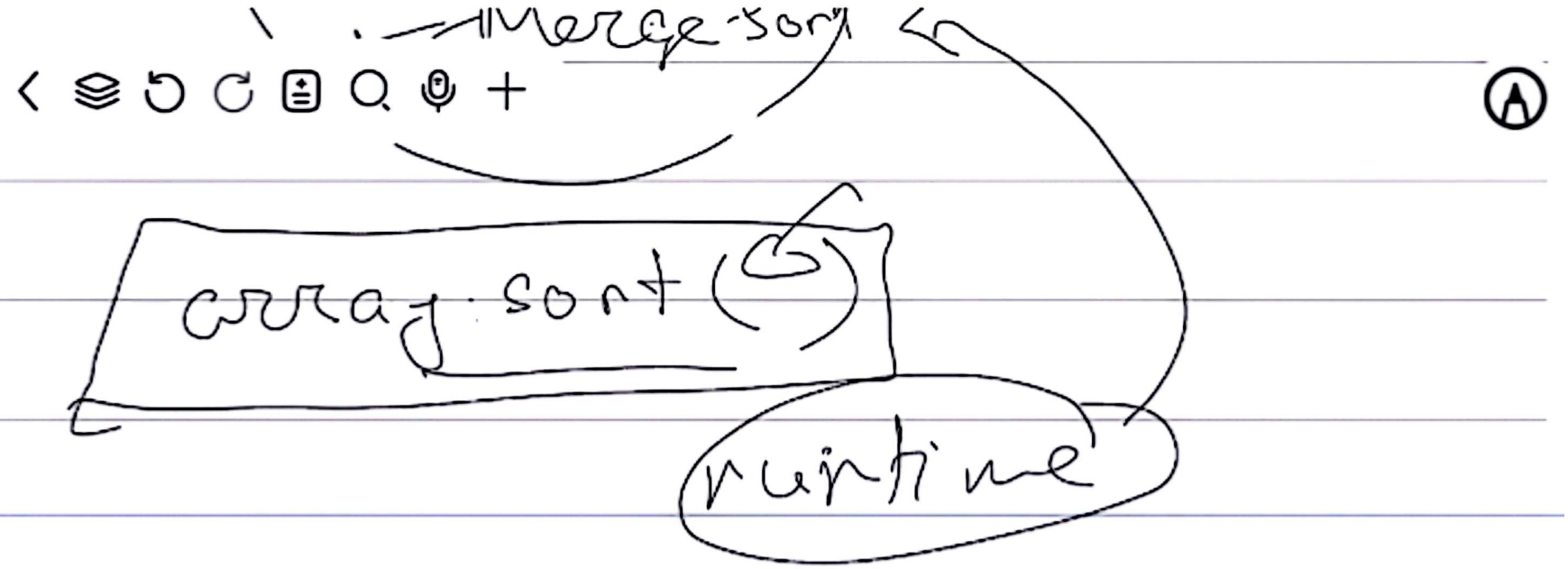

sign Pattern Demo src design_patterns behavioural state GoodExample
Project
chain_of_responsibility/Client.java
BadExample.java
GoodExample.java
Sorter.java
SortingStrategy.java
SelectionSort.java
BubbleSort.java
strategy/Client.java
IPWhitelistingHandler.java
Design Pattern Demo
.idea
out
src
design_patterns
behavioural
chain_of_responsibilit
AuthenticationHanc
Client
IPWhitelistingHand
TwoFactorAuthentic
UsernamePasswor
command
state
BadExample
GoodExample
strategy
BubbleSort
Client
SelectionSort
Sorter
SortingStrategy
creational
structural
solid_examples
Main
.gitignore
Design Pattern Demo.iml
External Libraries
Scratches and Consoles
3 public class GoodExample { Partha-SUST16
31 static class Document { 6 usages Partha-SUST16
36 }
37
38 public void publish() { 2 usages Partha-SUST16
39 state.publish(document: this);
40 }
41
42 public void setState(DocumentState state) { 3 usages Partha-SUST16
43 this.state = state;
44 }
45 }
46
47 public static void main(String[] args) { Partha-SUST16
48 Document document = new Document();
49 document.publish();
50 document.setState(new ModerationState());
51 document.publish();
52 }
53 }
54
55 /**
56 * 1. Modularity
57 * 2. Flexibility and Extensibility
58 * 3. Code duplication prevention
59 * 4. Clear code Understanding
60 * You, Today · Uncommitted changes
61 */
Version Control
Debug
TODO
Problems
Terminal
Services
Profiler
Build
60:4
Blame: You 27/5/24, 12:58 AM
LF
UTF-8
4 spaces
master

⑥ Strategy design Pattern



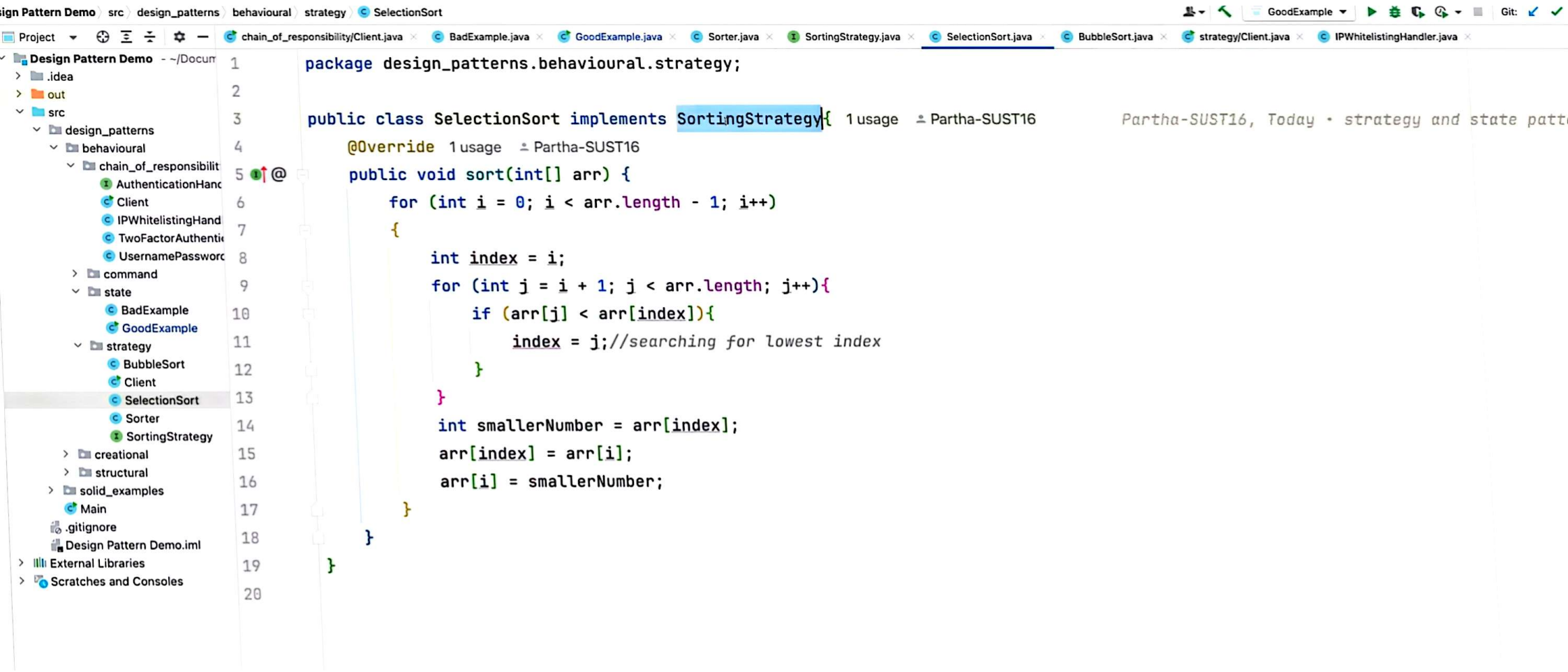
array.sort( )

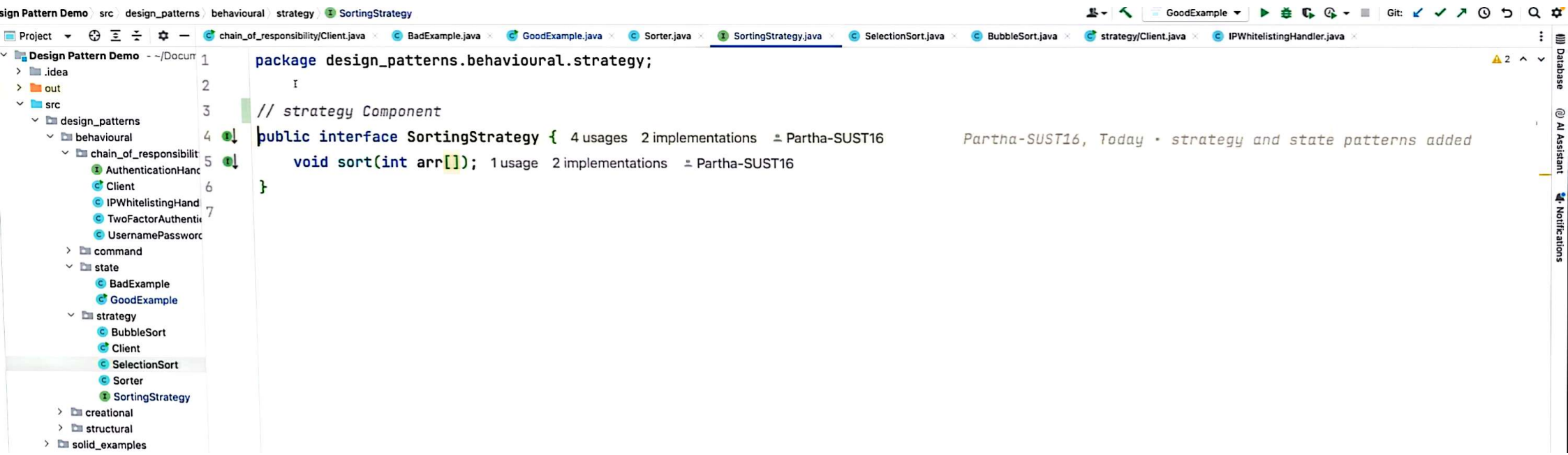
Untitled Notebook 1



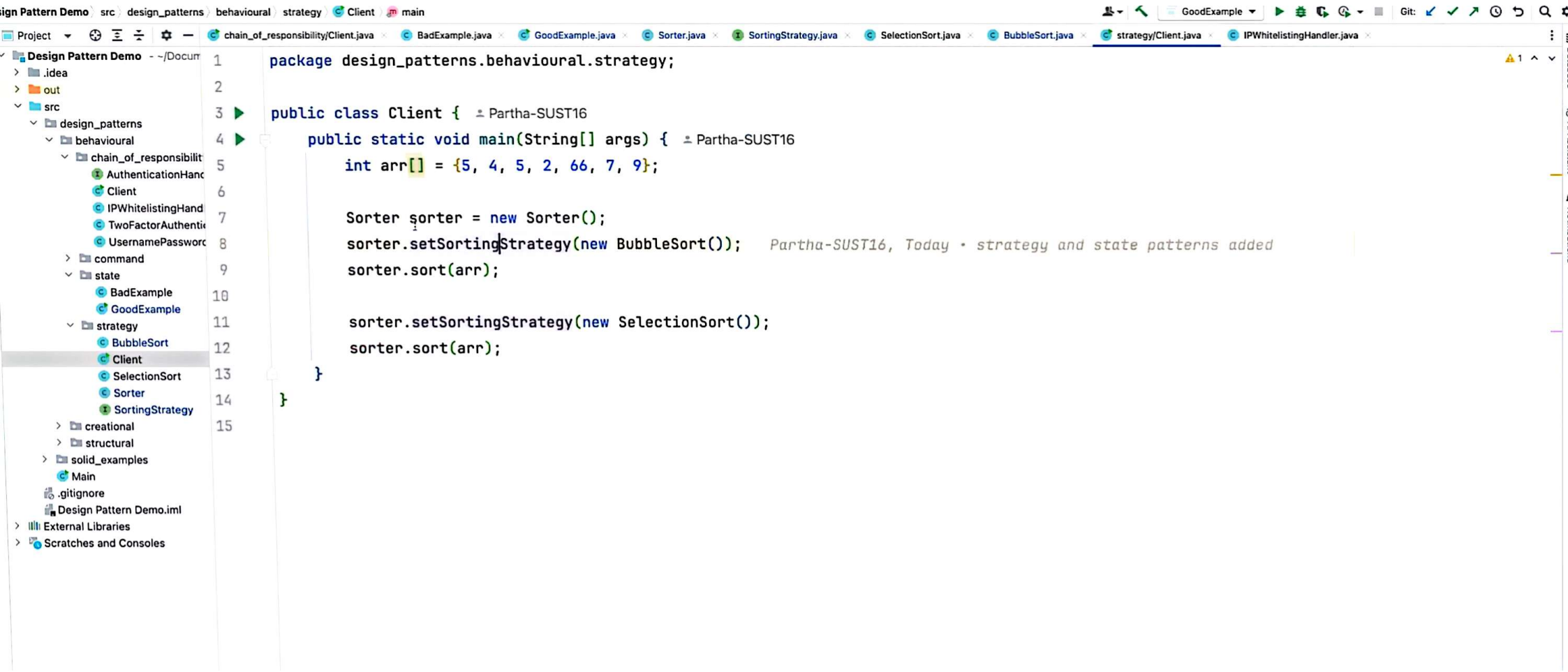
razorpay payment

card


ign Pattern Demo src design_patterns behavioural strategy SelectionSort
GoodExample
Git:
Project
chain_of_responsibility/Client.java
BadExample.java
GoodExample.java
Sorter.java
SortingStrategy.java
SelectionSort.java
BubbleSort.java
strategy/Client.java
IPWhitelistingHandler.java
Design Pattern Demo - ~/Docum
.idea
out
src
design_patterns
behavioural
chain_of_responsibilit
AuthenticationHanc
Client
IPWhitelistingHand
TwoFactorAuthentic
UsernamePasswor
command
state
BadExample
GoodExample
strategy
BubbleSort
Client
SelectionSort
Sorter
SortingStrategy
creational
structural
solid_examples
Main
.gitignore
Design Pattern Demo.iml
External Libraries
Scratches and Consoles
1 package design_patterns.behavioural.strategy;
2
3 public class SelectionSort implements SortingStrategy{ 1 usage Partha-SUST16 Partha-SUST16, Today • strategy and state patt
4 @Override 1 usage Partha-SUST16
5 public void sort(int[] arr) {
6 for (int i = 0; i < arr.length - 1; i++)
7 {
8 int index = i;
9 for (int j = i + 1; j < arr.length; j++){
10 if (arr[j] < arr[index]){
11 index = j;//searching for lowest index
12 }
13 }
14 int smallerNumber = arr[index];
15 arr[index] = arr[i];
16 arr[i] = smallerNumber;
17 }
18 }
19 }
20


sign Pattern Demo src design_patterns behavioural strategy SortingStrategy
GoodExample
Git:
Project
chain_of_responsibility/Client.java
BadExample.java
GoodExample.java
Sorter.java
SortingStrategy.java
SelectionSort.java
BubbleSort.java
strategy/Client.java
IPWhitelistingHandler.java
Design Pattern Demo - ~/Docum
.idea
out
src
design_patterns
behavioural
chain_of_responsibilit
AuthenticationHanc
Client
IPWhitelistingHand
TwoFactorAuthentic
UsernamePasswor
command
state
BadExample
GoodExample
strategy
BubbleSort
Client
SelectionSort
Sorter
SortingStrategy
creational
structural
solid_examples
package design_patterns.behavioural.strategy;
// strategy Component
public interface SortingStrategy { 4 usages 2 implementations Partha-SUST16
Partha-SUST16, Today • strategy and state patterns added
void sort(int arr[]); 1 usage 2 implementations Partha-SUST16
}
Database
AI Assistant
Notifications


sign Pattern Demo src design_patterns behavioural strategy BubbleSort
GoodExample
Git:
Project
chain_of_responsibility/Client.java
BadExample.java
GoodExample.java
Sorter.java
SortingStrategy.java
SelectionSort.java
BubbleSort.java
strategy/Client.java
IPWhitelistingHandler.java
Design Pattern Demo - ~/Docum
.idea
out
src
design_patterns
behavioural
chain_of_responsibilit
AuthenticationHanc
Client
IPWhitelistingHand
TwoFactorAuthentic
UsernamePasswor
command
state
BadExample
GoodExample
strategy
BubbleSort
Client
SelectionSort
Sorter
SortingStrategy
creational
structural
solid_examples
Main
.gitignore
Design Pattern Demo.iml
External Libraries
Scratches and Consoles
1 package design_patterns.behavioural.strategy;
2
3 // Concrete Strategy
4 public class BubbleSort implements SortingStrategy{ 1 usage Partha-SUST16 Partha-SUST16, Today • strategy and state patterns added
5 @Override 1 usage Partha-SUST16
6 public void sort(int[] arr) {
7 int size = arr.length;
8
9 // loop to access each array element
10 for (int i = 0; i < size - 1; i++) {
11 for (int j = 0; j < size - i - 1; j++)
12 if (arr[j] > arr[j + 1]) {
13 int temp = arr[j];
14 arr[j] = arr[j + 1];
15 arr[j + 1] = temp;
16 }
17 }
18 }
19 }
20


ign Pattern Demo src design_patterns behavioural strategy Client main
GoodExample
Git:
Project
chain_of_responsibility/Client.java
BadExample.java
GoodExample.java
Sorter.java
SortingStrategy.java
SelectionSort.java
BubbleSort.java
strategy/Client.java
IPWhitelistingHandler.java
Design Pattern Demo - ~/Docum
.idea
out
src
design_patterns
behavioural
chain_of_responsibilit
AuthenticationHanc
Client
IPWhitelistingHand
TwoFactorAuthentic
UsernamePasswor
command
state
BadExample
GoodExample
strategy
BubbleSort
Client
SelectionSort
Sorter
SortingStrategy
creational
structural
solid_examples
Main
.gitignore
Design Pattern Demo.iml
External Libraries
Scratches and Consoles
1 package design_patterns.behavioural.strategy;
2
3 public class Client { Partha-SUST16
4 public static void main(String[] args) { Partha-SUST16
5 int arr[] = {5, 4, 5, 2, 66, 7, 9};
6
7 Sorter sorter = new Sorter();
8 sorter.setSortingStrategy(new BubbleSort()); Partha-SUST16, Today • strategy and state patterns added
9 sorter.sort(arr);
10
11 sorter.setSortingStrategy(new SelectionSort());
12 sorter.sort(arr);
13 }
14 }
15
1