FUNDAMENTALS OF OBJECT ORIENTED PROGRAMMING

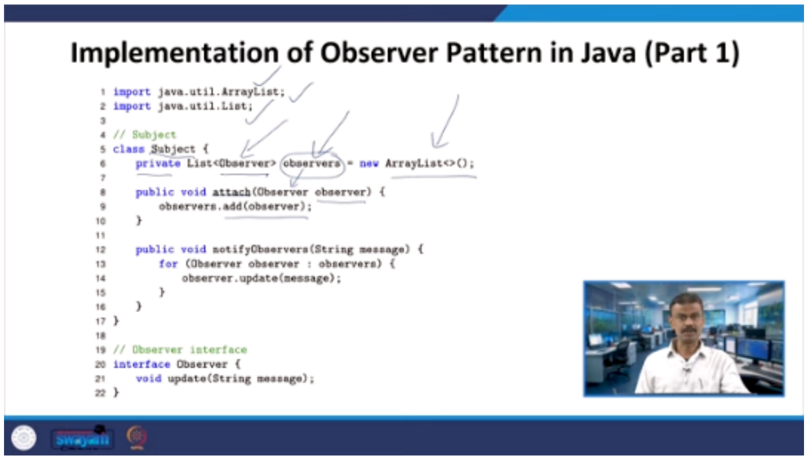
Lecture49

Lecture 49: Observer Pattern

Thank you. Welcome to lecture number 49, Design Patterns. So, in the last lecture, I talked about observer pattern in C++ and in today's lecture, I will start with the implementation of observer pattern in Java, alright. So, let us consider. So, we just import the utility array list and utility list, alright, utility call list.

Let us have the class subject, alright. So, let us have the class subject. and you have a private right private class called list right so which is a inbuilt so that contains the list of observer right and then you have a object under list is observers all right and dynamically allocate a memory with array list right array list constructor all right so array list constructor so which is available in array list right you have a method called attach So, the object is observer and the class is observer. So, this method where the observers, the object observers which is invoking the inbuilt at by pausing observer.

So, it is like what we had seen in C++. This will add whatever you are pausing. We are going to pause some message. So, that will be in the list. So, this example, exact example we had seen in C++.



Implementation of Observer Pattern in Java (Part 1)

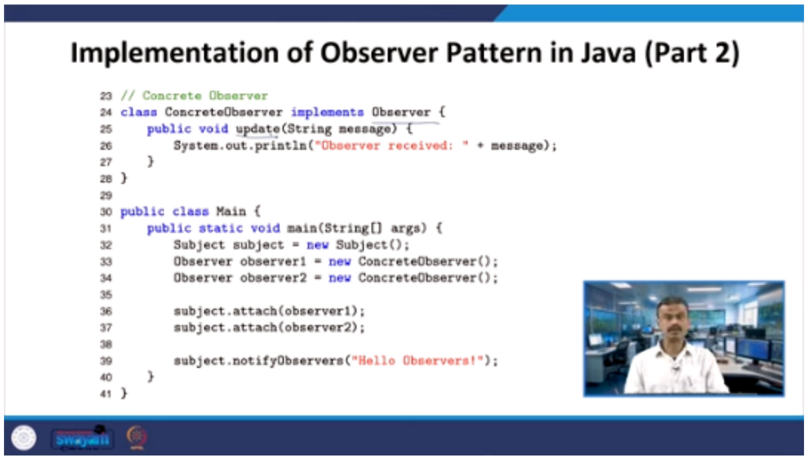
```java
import java.util.ArrayList;
import java.util.List;

// Subject
class Subject {
    private List<Observer> observers = new ArrayList<>();

    public void attach(Observer observer) {
        observers.add(observer);
    }

    public void notifyObservers(String message) {
        for (Observer observer : observers) {
            observer.update(message);
        }
    }
}

// Observer interface
interface Observer {
    void update(String message);
}
```

So, another method notify observers right notify observers where the argument is a message argument is a message and the data type is string ok data type is string return type is void in this method. So, now range for loop observers The identity ashmap of observers and observer both are same now. So, therefore, right. So, this will be copied into observer.

So, when observer is invoking update right by passing message this function will be executed right. So, whatever number of observers correct. So, if there are number of observers in the list. So, that means, if there are two who are there. So, two times this for loop will be executed if it is n times.

So, n times this will be executed. So let us consider the interface observer right. So let us consider the interface observer. So under this interface you have so this is abstract method update. So that means this has to be inherited and you have the message the argument correct or parameter whose data type is string.

So, this has to be inherited this interface has to be inherited further because it is 100 percent abstraction. So, now let us consider class concrete observer right the concrete observers implements observer right. So, that means it is inheriting observer and here you have the update method correct. So, in the update method you have the message Whose data type is string.



Implementation of Observer Pattern in Java (Part 2)

```java
// Concrete Observer
class ConcreteObserver implements Observer {
    public void update(String message) {
        System.out.println("Observer received: " + message);
    }
}

public class Main {
    public static void main(String[] args) {
        Subject subject = new Subject();
        Observer observer1 = new ConcreteObserver();
        Observer observer2 = new ConcreteObserver();

        subject.attach(observer1);
        subject.attach(observer2);

        subject.notifyObservers("Hello Observers!");
    }
}
```

So, here you have one print statement. So, that print statement observer received. And in fact, whatever the message that we are going to pass. Alright. So, based on this, which constructor it is going to be called.

Alright. So, we are going to have the upcasting, etc. Let us see what exactly it will be printed. Alright. So, let us go to the main program.

So, you have the main class, which is a driver class. And you have a public static void main method. I am creating an object, subject, right under the class subject, right? Dynamically allocating memory with a constructor, subject, right? So, here I have one observer 1, which is an object under observer, and it will be upcasting with concrete observer, right?

So, concrete observer. So, concrete observer, you are having the method over here. which is implementing observer, so that means, assume that whenever you are calling this, right? Whenever you are calling this particular, let us say, observer 1, correct? This object, when you are passing it, so this particular constructor will be called. That means whatever the update method, whenever it is invoking, so this will be printed, right? So, right now, You have instantiated two objects. Observer 1 and observer 2.

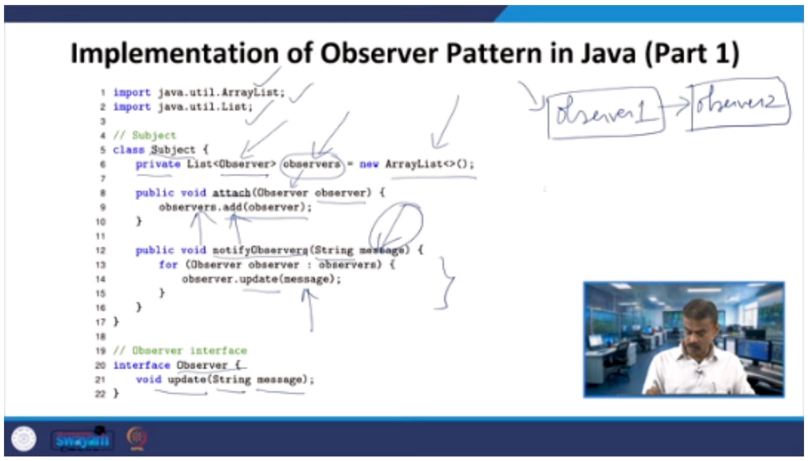
Under observer. So now the subject. This object. Subject is invoking. Attach by passing.

observer 1 all right it is invoking attach by pausing observer 1 so what is happening when it is invoking so observers object will invoking add right so whatever you have passed so what you have passed observer 1 so now observer and observer one both the identity ash map is same so the first observer one right so that will be in the list observer one right so that will be in the list and then subject is invoking attach you are pausing observer two correct so in that case the next one is observer two without loss of generality i take a linked list so it is attaching at the last right so now subject This object is invoking notify observer. Right. Notify observers.

All right. So here you go. Notify observers. You are passing what? Allow observer message.

Right. You are passing allow observers message. Right. So notify observer. So message will be nothing but this one will be nothing but allow observers.

Right. So that you are passing. And now, for the observer, I already said. So here you have the range for loop. So the range for loop contains these two.



Implementation of Observer Pattern in Java (Part 1)

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

// Subject
class Subject {
    private List<Observer> observers = new ArrayList<>();

    public void attach(Observer observer) {
        observers.add(observer);
    }

    public void notifyObservers(String message) {
        for (Observer observer : observers) {
            observer.update(message);
        }
    }
}

// Observer interface
interface Observer {
    void update(String message);
}
```

Right now. So suppose you have, let us say, so many attached. Let us say observer 3, 4, 5, 6, etc. So n you are attaching. So the list will be more.

Right. So it will be like observer 1, 2, 3, etc. Up to n. So right now, you have two attached methods you are calling. So these two will be there. So the for loop will go like this.

So observer dot update. So it is invoking update. So update it will be overridden in the interface. So first message you are passing observer 1. First one will be observer 1.

Anyway you are passing allow observers. Right. So allow observers you are passing. So update. So here you go update.

So the string will be right. The message that you have passed. The message is nothing but allow observers. So system.out.println you have observer received. followed by allow observers.

So, this is for observer 1 right. So, then in the list you have observer 2 is arranged for loop. So, observer 2 this will again happen. So, that means same observer received and allow observers will be printed. So, that means when I run the code I have to get 2 times the output.

So, this is what exactly we are seeing over here. So, this is how observer pattern will work in Java. So, we had seen observer pattern in C++ and here you go observer pattern in Java. And similarly, we will see one more program. This is with the help of C++ right.

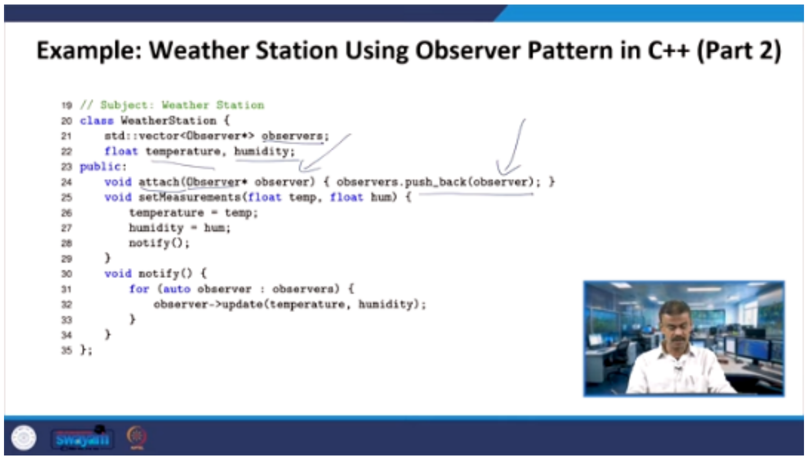
So, here weather station using observer pattern in C++ right. So, let us have class observer. So, which is having a virtual function right update pure virtual function. So, that is equal to 0. So that means it is abstract class observer.

So then, this means the update function has to be defined in the subclass. So, the subclass is PhoneDisplay, right? So, which is publicly inheriting Observer. So here, you have the update function. We will pass temperature and humidity, right?

So, this will be overridden. So, PhoneDisplay temperature—whatever the temperature is—will be displayed. And degree Celsius humidity—the humidity will be displayed, right? So, this is your PhoneDisplay function. And here, another class called WeatherStation, right?

You have another class called WeatherStation. So, here you have the object observers under vector. So, the vector as Observer—the pointer object Observer. So, here you have two private member data: temperature and humidity, and in the public, you have a member function called attach. So, attach you have Observer object.

So, it is a pointer object under observer all right. So, now this function call the pushback in fact we had seen the similar kind of problem right. So, pushback observer right. So, pushback by passing observer. So, the observer is invoking this



Example: Weather Station Using Observer Pattern in C++ (Part 2)

```cpp
19 // Subject: Weather Station
20 class WeatherStation {
21     std::vector<Observer*> observers;
22     float temperature, humidity;
23 public:
24     void attach(Observer* observer) { observers.push_back(observer); }
25     void setMeasurements(float temp, float hum) {
26         temperature = temp;
27         humidity = hum;
28         notify();
29     }
30     void notify() {
31         for (auto observer : observers) {
32             observer->update(temperature, humidity);
33         }
34     }
35 };
```

function assume that you are passing some information i mean it will be the list right so it will be pushed back into the list so now line number 25 you have set measurements the member function called set measurements you are passing two arguments temp and humidity right temperature and humidity so and then you are copying temp into temperature into humidity and it is also calling the notify function right. So, the notify function is defined as a usual function you have a range for loop right observers will be copied into observer and then observer is invoking update by passing temperature and humidity right. So, update here we have in the subclass the subclass is the phone display whereas the abstract class is observer. All right.

So this will be called. So this we will see when we call this in the main program. So let us go to the main program. Main program, you have an object station. Correct.

And the class is a weather station. Right. The class is a weather station. You have an object phone, and the class is phone display. OK.

So now, the object station. Right. Which is invoking attach. And you are passing the reference of phone. Right.

You are passing the reference of phone. So, attach. So, let us say what will happen, right? So, the attach, in fact, you have line number 24. So, in line number 24, you are passing the reference of phone, right?

Observer star, right? So, observer star equals observer, which is equal to the address of phone. So, that means this observer is pointing to the reference of phone now, right? And the observers, right, which is a private object, In fact, the pointers of observer in vector, right.

So, push back observer means you are passing the phone. So, it will attach—I mean, the push back. So, it will put the phone in the list, right. It will put the phone in the list, right. So, it will put the phone in the list.

So, now the station object, which is invoking set measurements. Right, which is invoking set measurements by passing 25.0 and 60.0, all right. So, here you go, all right. So, here you have—you are passing 25.0 and 60.0, all right. So, now temperature will be 25.0, and the humidity will be 60.0, all right.

So, it is invoking notify, line number 28. All right, this particular class is invoking. In fact, this particular function is invoking notify. So, notify here observer and observers—observers, right?—which will be copied into observer. So, you have one phone data, right, in the list, and here, update correctly. So, here you update temperature and humidity, right.

So, in the list, the phone list. So, let us say 25.0 and 60.0, all right. So, update. So, let us see what update is doing, right? Because observer is invoking update, correct. So, observer is invoking update.

So, when it is invoking update, right. So, here you go—it is the virtual function. So, therefore, it was inherited here and will be overridden. So, here you go—you are passing this temperature and humidity, 25.0 and 60.0, right. So, this will be displayed—see, out phone display temperature, right, 25.0.

And degree Celsius humidity 60.0 right. So these two will be printed right. And now station is invoking set measurements by passing 30.0 and 55.0 right. You are passing this. So that means this will be updated.

Rest are all same. This will be updated with what is the value 30.0 and 55.0 all right. So, maybe here itself I can put I do not need to put one more. So, here right.

So, it is understood that. So, this will be updated with 30.0 and 55.0 all right. So, this is a change. So, you are passing this that means observer is invoking the update right the both are same the location is same I will just cross down right. So, temperature humidity you are passing 30.0 and 55.0 and that will be displayed because it is invoking update and this will be displayed.

So, first time it will be displaying 25.0 and 60.0 and second time it will be displaying 30.0 and 55.0. So, when I run the code I will get the output this. So, that is what we are expecting 25 and 60, 30 and 55 all right. So, here you are attaching the reference is phone all right. So, that is what this is the one all right.

So, right now you call this is the observer. So, observers correct. So, that you are copying into observer. So, that means I have already attached the phone. So, the phone

reference the address of phone is nothing but the observer right. So, that is what the meaning observer star observer equal to address of phone. So, that means observer and reference of phone is same right and then you are having a range for loop in fact the same location right. So, which is nothing but address of phone and when you are passing into the function the attach function. So, this is nothing but observer

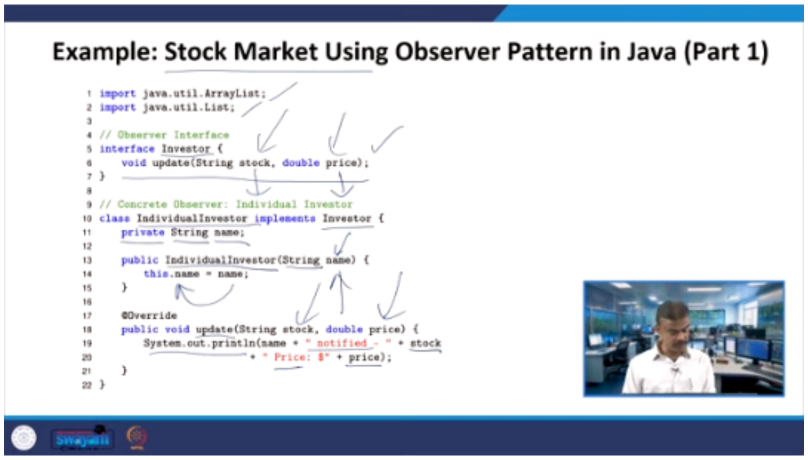
So, both are pointing to the same location address of phone and observer both are pointing to the same location that is the meaning. So, they are all coming under the class observer which is capital O right. So, this is how observer pattern is working and then we have seen the output as well. So, let us consider one more example called stock market right. So, this is using observer pattern in java.

So, let us have the utility array list and list like exactly what we had seen the previous examples. Here we have the interface call investor all right. So, that means the function that you are writing or a method that you are writing is a abstract method update you have two arguments stock and price right. So, whose data types are strings and double respectively now you have the subclass. right individual investor all right so which is implements the investor under this you have a private member data called name whose data type is string and then here you have a constructor so one argument constructor name whose data type is string so now name you are copying into this dot name

So, name you are copying into this dot name. So, now assume that since I have defined update. So, I have declared kind of declaration. It is a inside abstract method. Since you have abstract method update in investor interface, I have to define in the subclass.

So, here we have update. So, this will be overridden. You have two arguments stock and price whose data types are string and double respectively. So, now here you have one print statement system dot out dot printl and name notified and stock is price right. So, name plus that means the name will be printed all right.

So, the name will be printed anyway this dot name equal to name that name will be printed notified will be there and stock all right so that will be printed whatever you are passing that stock will be printed all right and then you have a price price will be printed stock price right so all these will be printed name so whatever be the name name will be printed and notified is nothing but the statement that you are writing under the code that will be printed as it is Whatever value of the stock in string will be printed, price in dollar will be printed and the price will be printed, ok. So, here this class is over, right. So, which is a concrete observer.



Example: Stock Market Using Observer Pattern in Java (Part 1)

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

// Observer Interface
interface Investor {
    void update(String stock, double price);
}

// Concrete Observer: Individual Investor
class IndividualInvestor implements Investor {
    private String name;

    public IndividualInvestor(String name) {
        this.name = name;
    }

    @Override
    public void update(String stock, double price) {
        System.out.println(name + " notified - " + stock
                          + " Price: $" + price);
    }
}
```

First one is interface. So, now you have a class called stock exchange, right. Another class called stock exchange. Now, you have the object investors under list. of investor all right list of investor and in the right hand side you have dynamically allocating a memory with the constructor array list all right and you have a stock private member data under string right that means whose data type is string and price whose data type is double okay so these three are private and you have a public method called attach investor is an object and the class is the investor

So, this method is invoking that means under investors this object which is invoking add by pausing investor. So, add it is a inbuilt function. So, which is available in the utility. So, that means whatever you are pausing. So, that will be added into the list that is the meaning.

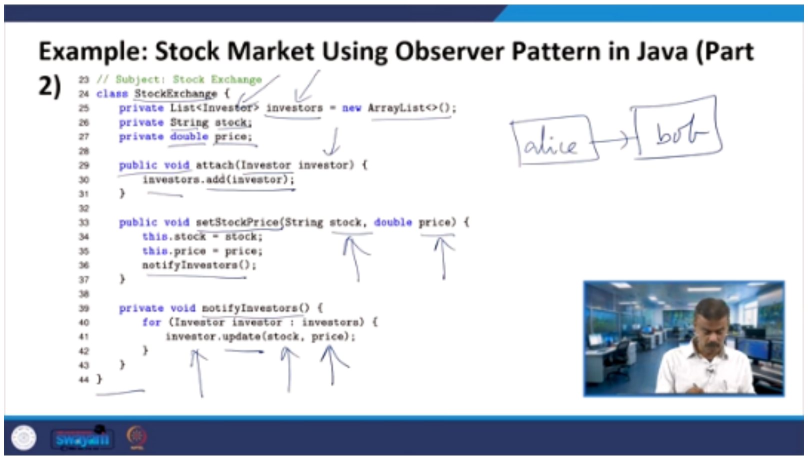
You have one more member function. In fact, it is a set stock price which is a method by pausing stock and price. right. So, stock will be copied into this dot stock and price will be copied into this dot price and this is calling notify investors and notify investors are here inside the class, all right. It is the range for loop, correct.

So, whatever you are adding, let us say in the list you have n, n number, all right. So, n number of investors. So, these investors, all right. So, will be called every time 1, 2, 3, 4, 5, etc. up to n times.

So, let us say you have two So, 2 times this for loop will be called and the investor object which is invoking update, right? You are pausing stock and price, right? So, this is how the stock exchange class up to here is working. So, now let us go to the main class, right?

So, here you have the main class, right? So, here you have the main method. I have an object exchange, right, under the class stock exchange, dynamically allocating a memory with a constructor stock exchange, right. So, now, in line number 48, I have an object allies, right, I have an object allies under investor and we are upcasting with individual investor, that means, this constructor will be called by pausing allies, right. right so the constructor individual investor is over here the constructor so this dot name is equal to allies now so for the object allies this dot name is allies and similarly the object bob this dot name is bob the same right the similar all right so line number 51 the object exchange under stock exchange with invoking attach right so you are attaching by pausing allies

So let us see. Attach. What attach is doing? So here you go. The method attach.



Example: Stock Market Using Observer Pattern in Java (Part 2)

```java
// Subject: Stock Exchange
class StockExchange {
    private List<Investor> investors = new ArrayList<>();
    private String stock;
    private double price;

    public void attach(Investor investor) {
        investors.add(investor);
    }

    public void setStockPrice(String stock, double price) {
        this.stock = stock;
        this.price = price;
        notifyInvestors();
    }

    private void notifyInvestors() {
        for (Investor investor : investors) {
            investor.update(stock, price);
        }
    }
}
```

All right. So you are attaching the object allies. So investor is nothing but allies now. All right. So this is adding investors.

The object investors is adding investor. Right. What is the investor? Allies. So that means your first object in the list is allies.

Right. The first object in the list is allies. And then you are attaching Bob. right so the second object in the list is bob okay so this is what exactly happening over here so now exchange the object exchange is invoking set stock price by pausing a a p l and 150.0 right a a p l apple and 150.0 right so apple stock is copied into this dot stock Price is copied into this dot price and notify investors will be invoked.

Right. So, notify investors will be invoked. So, notify investors. You have the for loop. So, the for loop.

So, we know. All right. So, here what we have done, you have. Passed Apple and 150.0, right. So, here the investor for, the for is nothing but the allies and Bob, right.

So, investor dot update, you are passing stock and price, Apple and 150.0, correct. So, the first object is allies. It is invoking update, right. So, when it is invoking update, it is over here. So, what is the name?

Allies notified, right. Allies notified. What is stock? Apple. And what is the price?

Price in dollar is 150.0. So the first one it will be printing. The name allies. Right. The name is allies.

Notified will be printed. Dash. And stock will be Apple. Price in dollar will be printed. And that is 150.0.

So that is the first output we can expect. Right. Right. That is the first output we can expect. And then.

Right. So it is moving. Right. So further, when it moves. So, because of the range for loop.

Right. So now, when it is moving to Bob, right? So again, the investor is invoking update by passing the same stock and price. Right? By passing, I mean the stock and price have not changed because your cursor is here, and it is executing. Your cursor is here. So, Apple and 150.0, all right. So, what will happen? Bob is notified: Apple and the price is 150.0. Correct.

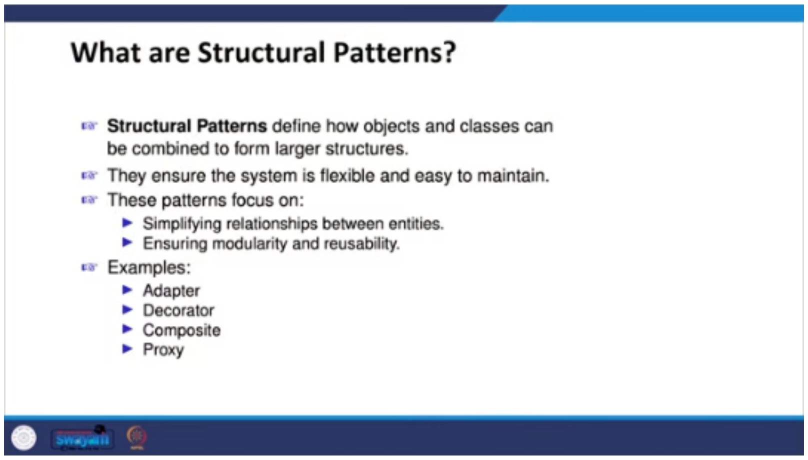
So, now again right so now exchange set stock price i mean google right and 2800 you are passing all right so now again your stock will be this and this stock will be google and this dot price is 2800 and now again notify investors so notify investors so right now right it will point right the investor you have the investors you have nothing but it is pointing here correct so again allies notified that stock is google and the price is 2800 again it will be printing correct so then for bob right so if you look at the output so you will be getting the output like this allies notified apple price is this bob notified apple price is this allies notified google price is this and bob notified google prices right. So, it is very clear, right.

So, how it is when the for loop is moving, right. The range for loop is moving, right. So, this is how the observer pattern is working, right. We have seen in C++ and we had also seen in Java. So, now if we compare observer pattern versus publish subscribe pattern, right.

So, the observers are directly tied to the subject in the case of observer pattern. And notification are sent to all registered observers, right? So that is what the example that we had seen both in C++ and Java. Whereas the publish-subscribe pattern, so it decouples publishers and subscribers via a message broker, right? And subscribers receive only the message they subscribe to, right?

So these are all the differences between the observer pattern and the publish-subscribe pattern. And what are all the benefits and drawbacks of the observer pattern? So, the benefits you have are: it decouples subject and observers, simplifies dynamic updates to multiple objects, and provides a flexible and extensible design for reactive systems. The main drawbacks are: the observer pattern can lead to performance overhead with many observers, and complexity increases with nested observer relationships, right? So, the observer pattern also has a risk of memory leaks.

If observers are not properly detached, right? So, these are all the drawbacks. So, in the next lecture, what we can do is we will talk about structural patterns, right? The structural patterns we have are adapter, decorator, composite, and proxy. So, this we will see in the next class.



What are Structural Patterns?

- **Structural Patterns** define how objects and classes can be combined to form larger structures.
- They ensure the system is flexible and easy to maintain.
- These patterns focus on:
  - Simplifying relationships between entities.
  - Ensuring modularity and reusability.
- Examples:
  - Adapter
  - Decorator
  - Composite
  - Proxy

Thank you very much.