LAB # 9

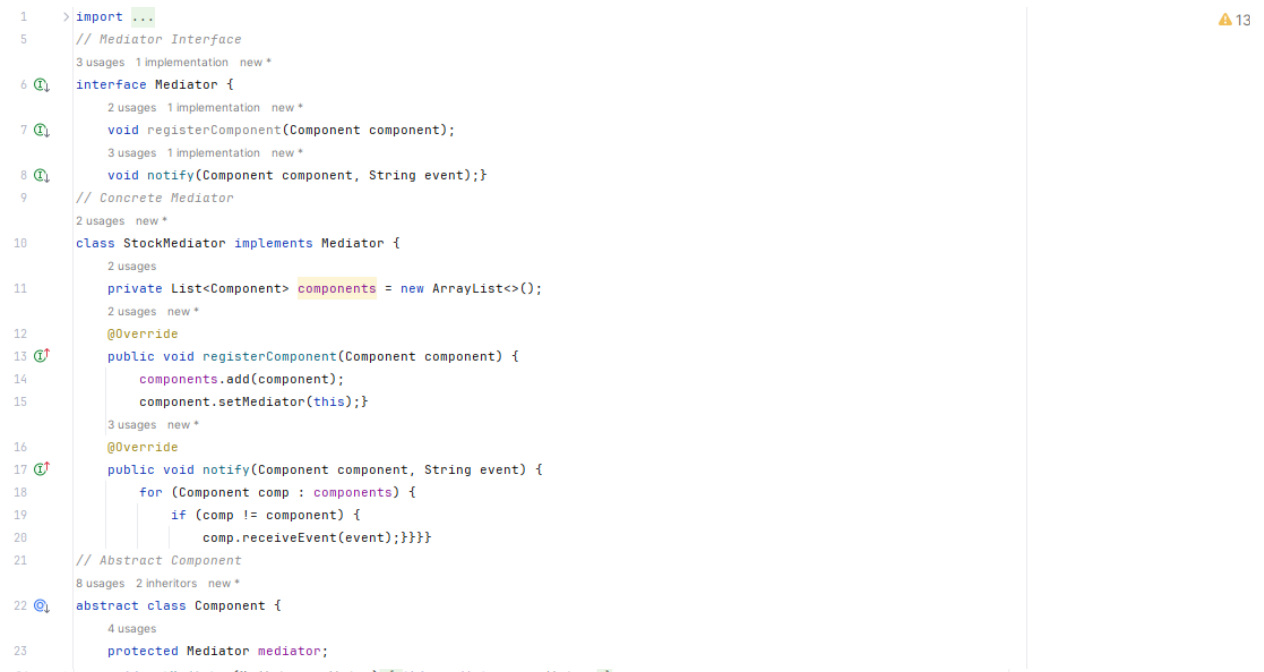
Behavioral Patterns: Mediator Pattern and Observer Pattern

OBJECTIVE

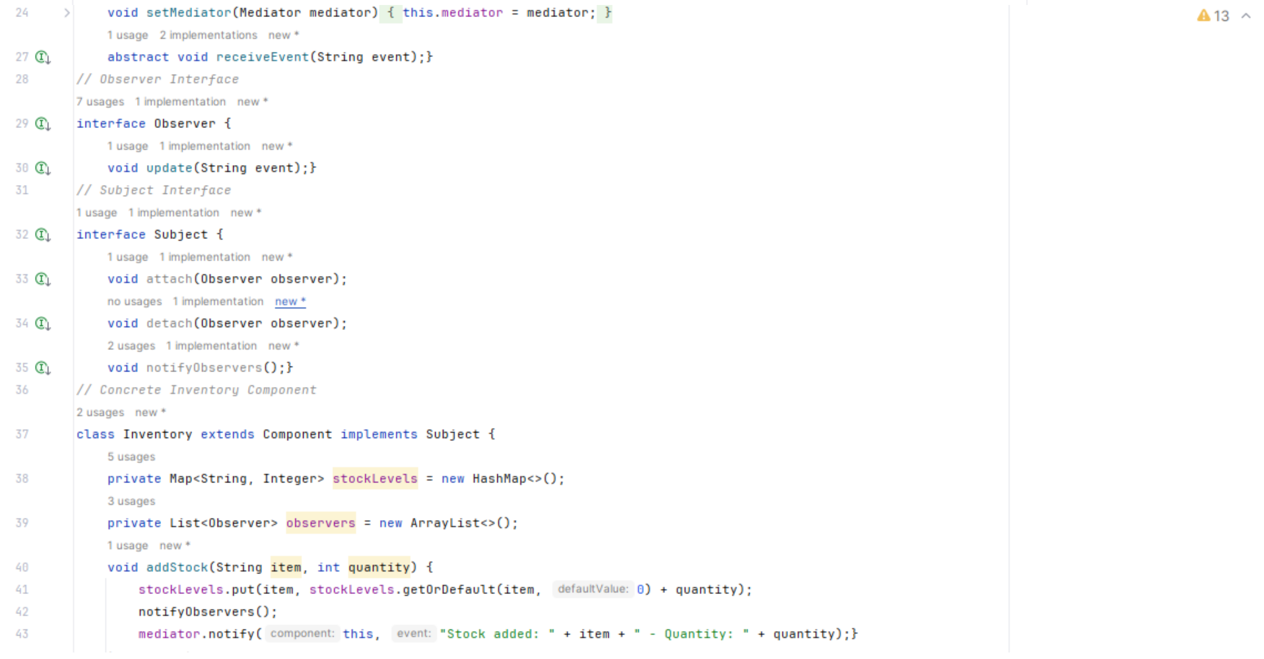
- Understand behavioral design patterns: Mediator Pattern and Observer Pattern

Exercise: Implement a Mediator Pattern and Observer Pattern on your selected scenario.

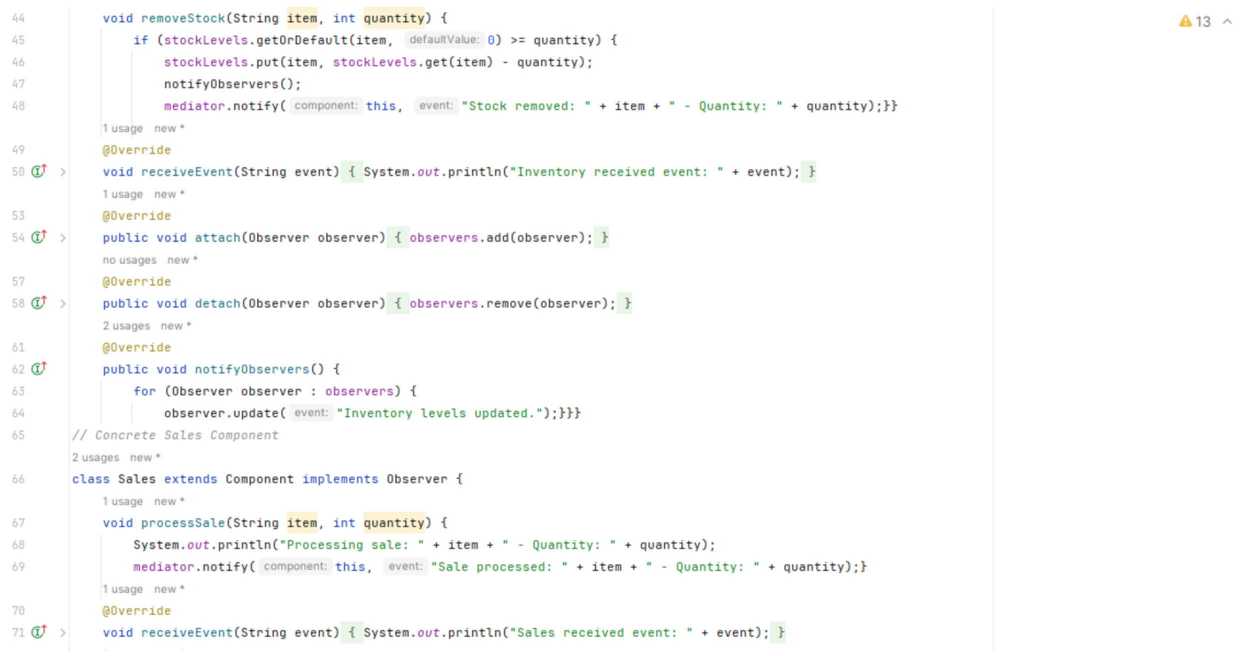
SOURCE CODE :



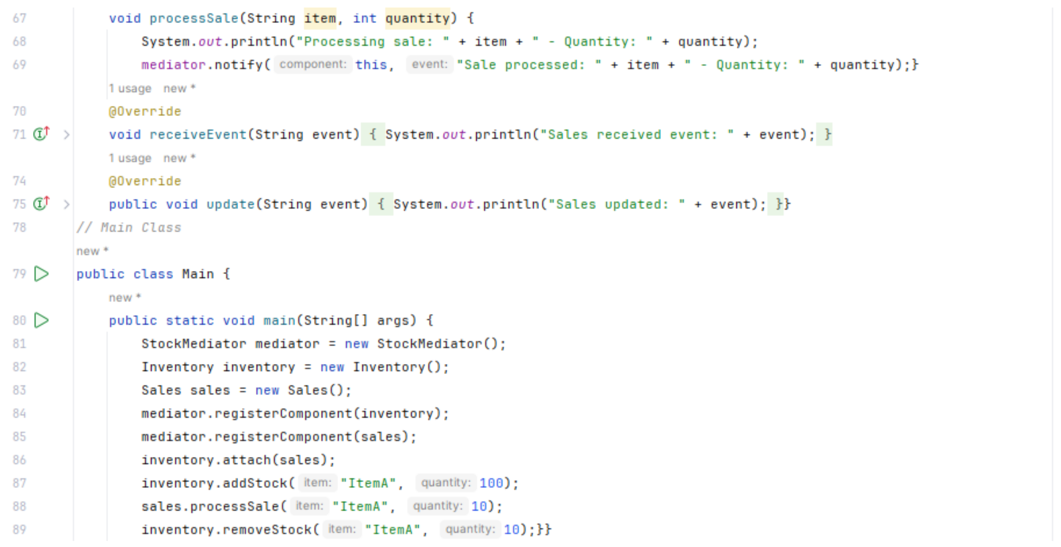
```java
import ...
// Mediator Interface
interface Mediator {
    void registerComponent(Component component);
    void notify(Component component, String event);}
// Concrete Mediator
class StockMediator implements Mediator {
    private List<Component> components = new ArrayList<>();
    @Override
    public void registerComponent(Component component) {
        components.add(component);
        component.setMediator(this);}
    @Override
    public void notify(Component component, String event) {
        for (Component comp : components) {
            if (comp != component) {
                comp.receiveEvent(event);}}}}
// Abstract Component
abstract class Component {
    protected Mediator mediator;
```



```java
    void setMediator(Mediator mediator) { this.mediator = mediator; }
    abstract void receiveEvent(String event);}
// Observer Interface
interface Observer {
    void update(String event);}
// Subject Interface
interface Subject {
    void attach(Observer observer);
    void detach(Observer observer);
    void notifyObservers();}
// Concrete Inventory Component
class Inventory extends Component implements Subject {
    private Map<String, Integer> stockLevels = new HashMap<>();
    private List<Observer> observers = new ArrayList<>();
    void addStock(String item, int quantity) {
        stockLevels.put(item, stockLevels.getOrDefault(item, 0) + quantity);
        notifyObservers();
        mediator.notify(this, "Stock added: " + item + " - Quantity: " + quantity);}
```

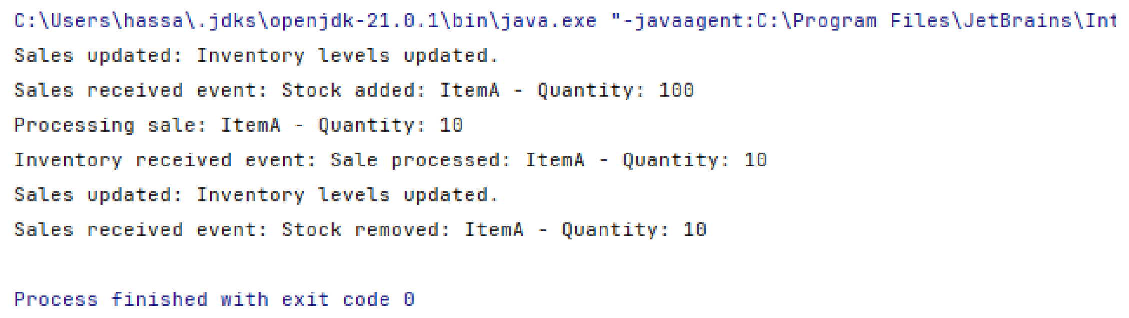


```
    void removeStock(String item, int quantity) {
        if (stockLevels.getOrDefault(item, defaultValue: 0) >= quantity) {
            stockLevels.put(item, stockLevels.get(item) - quantity);
            notifyObservers();
            mediator.notify( component: this,  event: "Stock removed: " + item + " - Quantity: " + quantity);}}
    1 usage  new *
    @Override
    void receiveEvent(String event) { System.out.println("Inventory received event: " + event); }
    1 usage  new *
    @Override
    public void attach(Observer observer) { observers.add(observer); }
    no usages  new *
    @Override
    public void detach(Observer observer) { observers.remove(observer); }
    2 usages  new *
    @Override
    public void notifyObservers() {
        for (Observer observer : observers) {
            observer.update( event: "Inventory levels updated.");}}}
// Concrete Sales Component
2 usages  new *
class Sales extends Component implements Observer {
    1 usage  new *
    void processSale(String item, int quantity) {
        System.out.println("Processing sale: " + item + " - Quantity: " + quantity);
        mediator.notify( component: this,  event: "Sale processed: " + item + " - Quantity: " + quantity);}
    1 usage  new *
    @Override
    void receiveEvent(String event) { System.out.println("Sales received event: " + event); }
```



```
    void processSale(String item, int quantity) {
        System.out.println("Processing sale: " + item + " - Quantity: " + quantity);
        mediator.notify( component: this,  event: "Sale processed: " + item + " - Quantity: " + quantity);}
    1 usage  new *
    @Override
    void receiveEvent(String event) { System.out.println("Sales received event: " + event); }
    1 usage  new *
    @Override
    public void update(String event) { System.out.println("Sales updated: " + event); }}
// Main Class
new *
public class Main {
    new *
    public static void main(String[] args) {
        StockMediator mediator = new StockMediator();
        Inventory inventory = new Inventory();
        Sales sales = new Sales();
        mediator.registerComponent(inventory);
        mediator.registerComponent(sales);
        inventory.attach(sales);
        inventory.addStock( item: "ItemA",  quantity: 100);
        sales.processSale( item: "ItemA",  quantity: 10);
        inventory.removeStock( item: "ItemA",  quantity: 10);}}
```

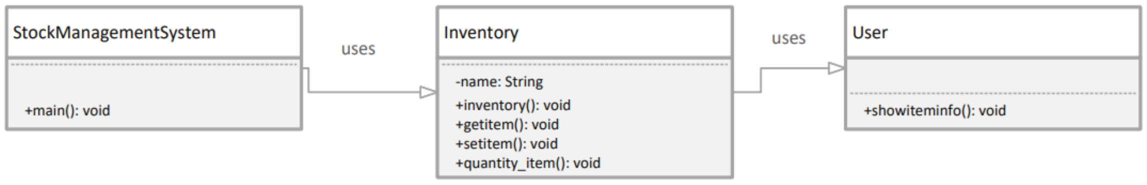
OUTPUT:



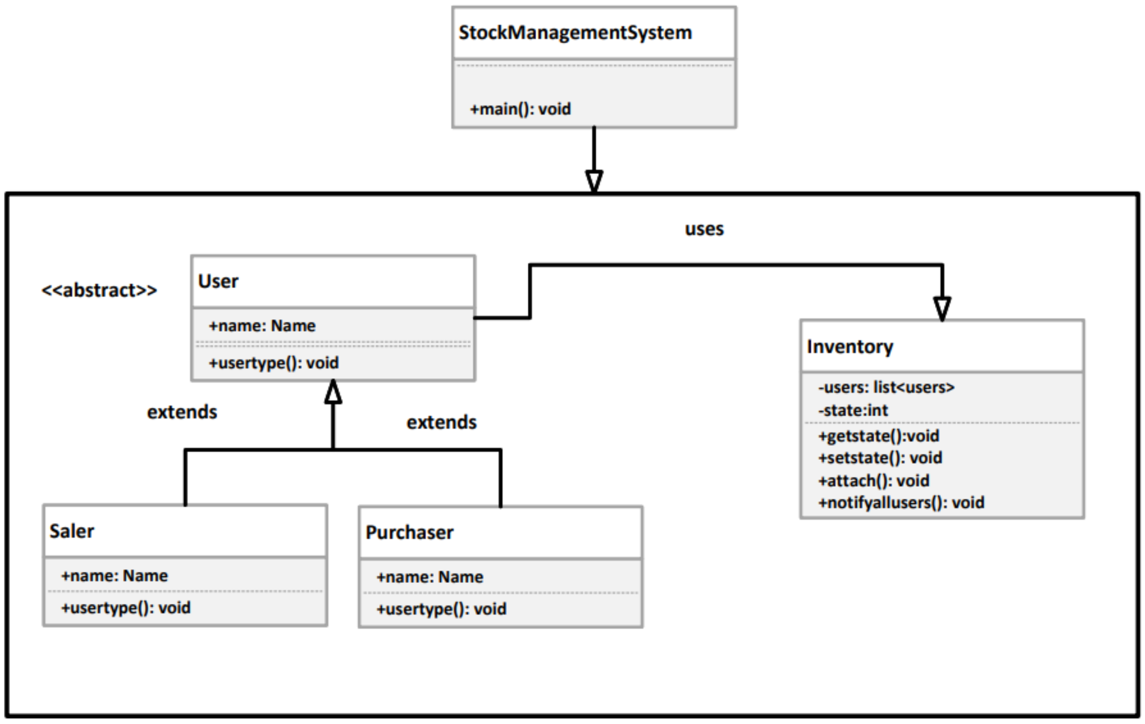
```
C:\Users\hassa\.jdks\openjdk-21.0.1\bin\java.exe "-javaagent:C:\Program Files\JetBrains\Int
Sales updated: Inventory levels updated.
Sales received event: Stock added: ItemA - Quantity: 100
Processing sale: ItemA - Quantity: 10
Inventory received event: Sale processed: ItemA - Quantity: 10
Sales updated: Inventory levels updated.
Sales received event: Stock removed: ItemA - Quantity: 10

Process finished with exit code 0
```

DIAGRAM :

MEDIATOR PATTERN



OBSERVER PATTERN


StockManagementSystem
+main(): void
uses
<<abstract>>
User
+name: Name
+usertype(): void
extends
extends
Inventory
-users: list<users>
-state:int
+getstate():void
+setstate(): void
+attach(): void
+notifyallusers(): void
Saler
+name: Name
+usertype(): void
Purchaser
+name: Name
+usertype(): void