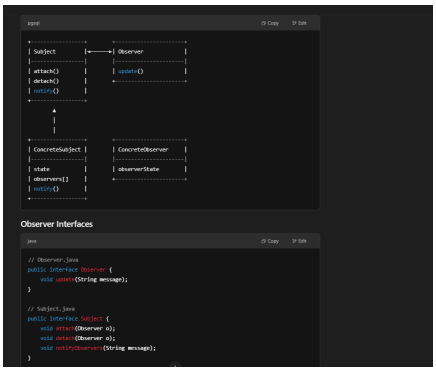
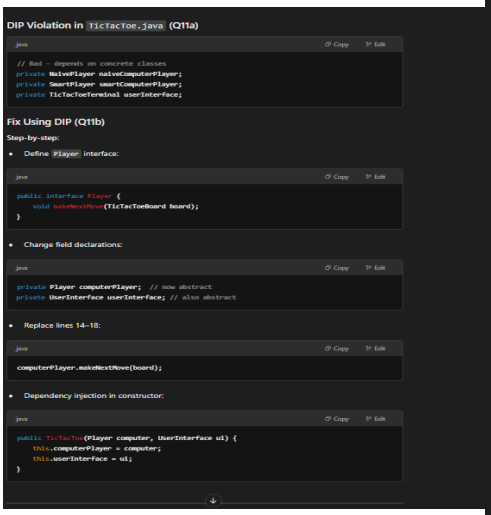
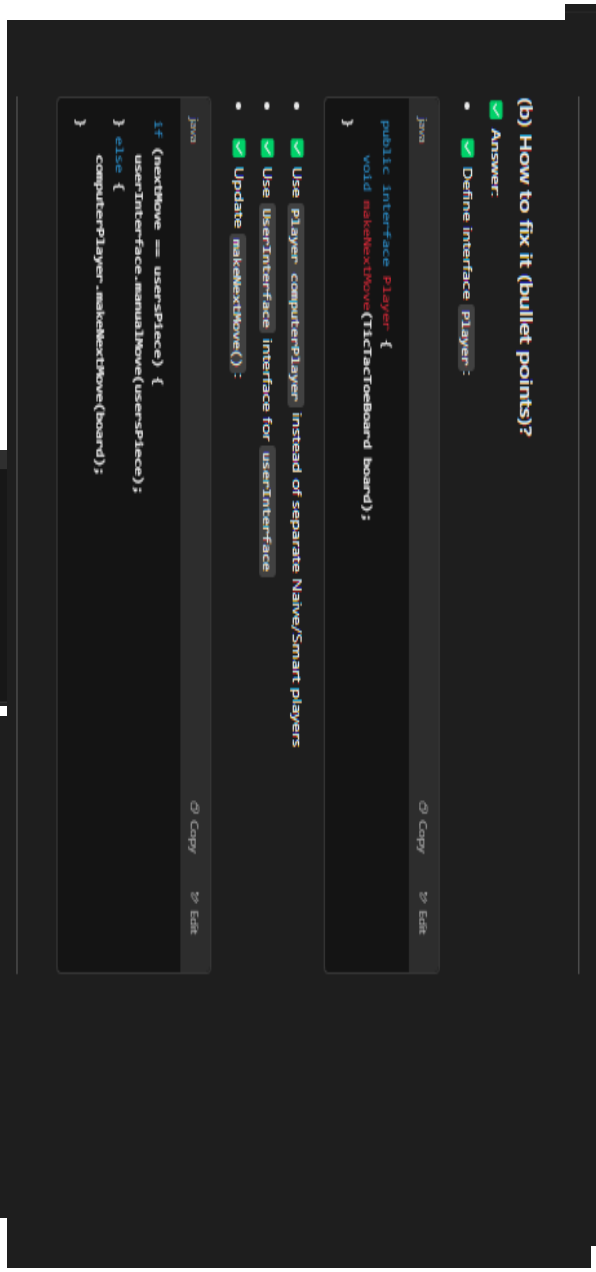
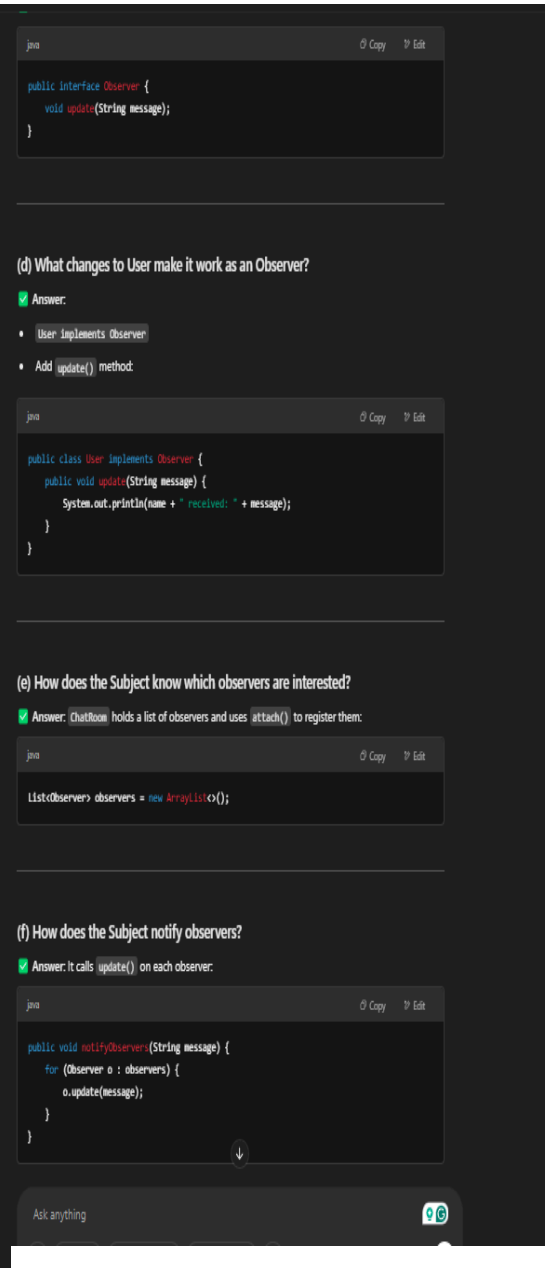
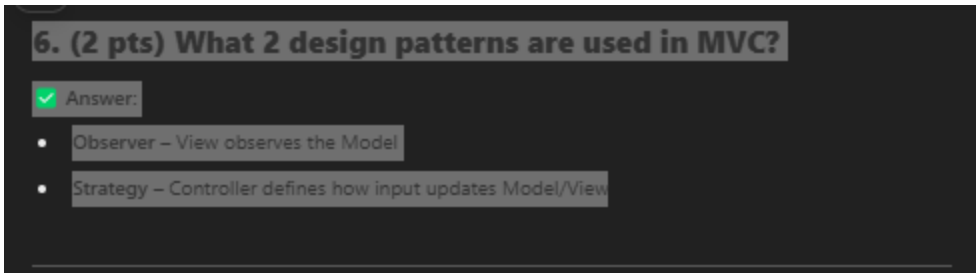

Observer Interfaces
// Observer.java
public interface Observer {
void update(String message);
}
// Subject.java
public interface Subject {
void attach(Observer o);
void detach(Observer o);
void notifyObservers(String message);
}


// User.java (ConcreteObserver)
public class User implements Observer {
private String name;
public User(String name) {
this.name = name;
}
public void update(String message) {
System.out.println(name + " received: " + message);
}
}


DIP Violation in TicTacToe.java (Q11a)
// Bad - depends on concrete classes
private NaivePlayer naiveComputerPlayer;
private SmartPlayer smartComputerPlayer;
private TicTacToeTerminal userInterface;
Fix Using DIP (Q11b)
Step-by-step:
Define Player interface:
public interface Player {
void makeNextMove(TicTacToeBoard board);
}
Change field declarations:
private Player computerPlayer; // now abstract
private UserInterface userInterface; // also abstract
Replace lines 14–18:
computerPlayer.makeNextMove(board);
Dependency injection in constructor:
public TicTacToe(Player computer, UserInterface ui) {
this.computerPlayer = computer;
this.userInterface = ui;
}


(b) How to fix it (bullet points)?
Answer:
Define interface Player:
public interface Player {
void makeNextMove(TicTacToeBoard board);
}
Use Player computerPlayer instead of separate Naive/Smart players
Use UserInterface interface for userInterface
Update makeNextMove():
if (nextMove == usersPiece) {
userInterface.manualMove(usersPiece);
} else {
computerPlayer.makeNextMove(board);
}


public interface Observer {
void update(String message);
}
(d) What changes to User make it work as an Observer?
Answer:
User implements Observer
Add update() method:
public class User implements Observer {
public void update(String message) {
System.out.println(name + " received: " + message);
}
}
(e) How does the Subject know which observers are interested?
Answer: ChatRoom holds a list of observers and uses attach() to register them:
List<Observer> observers = new ArrayList<>();
(f) How does the Subject notify observers?
Answer: It calls update() on each observer:
public void notifyObservers(String message) {
for (Observer o : observers) {
o.update(message);
}
}
Ask anything


6. (2 pts) What 2 design patterns are used in MVC?
Answer:
Observer – View observes the Model
Strategy – Controller defines how input updates Model/View