Heaven’s Light Is Our Guide

Rajshahi University of Engineering & Technology

Department of Computer Science & Engineering

LAB REPORT

Course Code: CSE 3206

Course Title: Software Engineering Sessional

Experiment Name: Observer pattern and state pattern.

Submitted By	**Submitted To**
Syed Abdullah Al Masum – 2003172 **Md F A Sadik – 2003173** **Nafis Ahmed - 2003174**	**Farzana Parvin** **Lecturer** **Department of CSE, RUET**

Introduction:

The Observer Design Pattern is a behavioral pattern where a publisher object notifies a list of observers whenever its state changes. This pattern establishes a one-to-many relationship, allowing multiple objects to be updated automatically when the publisher's state changes, without tight coupling between the two.

How to Implement the Observer Pattern

Publisher (Subject):

Maintains a list of observers and notifies them of state changes.

Observer Interface:

Defines an update() method that all observers must implement.

Concrete Observers:

Implement the Observer interface and handle updates from the publisher.

Client:

Registers observers with the publisher and triggers notifications.

Reasons to use:

Decoupling: The pattern reduces direct dependencies between the publisher and its observers.

Real-Time Notifications: Ideal for event-driven systems like stock updates, email subscriptions, or real-time messaging.

Efficiency: Observers are notified only when there’s a relevant change, minimizing unnecessary checks.

UML Diagram:

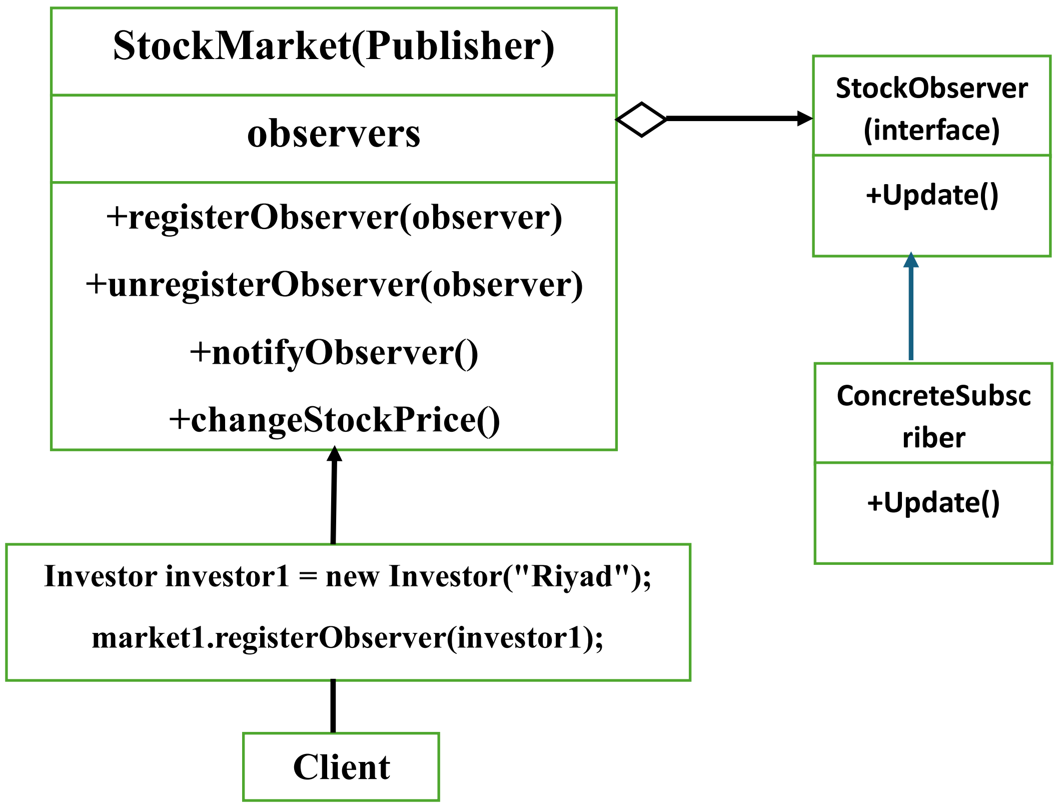
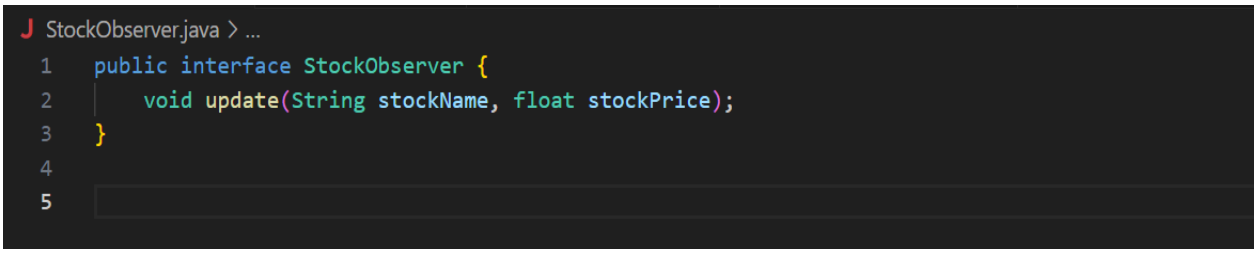

StockMarket(Publisher)
observers
+registerObserver(observer)
+unregisterObserver(observer)
+notifyObserver()
+changeStockPrice()
StockObserver (interface)
+Update()
ConcreteSubscriber
+Update()
Investor investor1 = new Investor("Riyad");
market1.registerObserver(investor1);
Client

Figure: UML diagram of Observer Design Pattern.

Code:

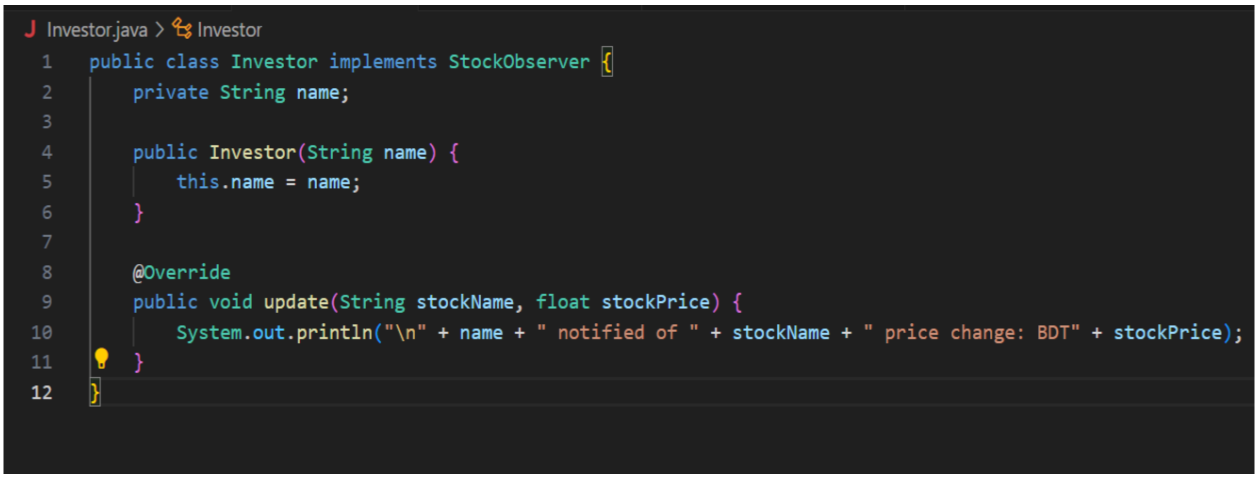
StockMarket Interface:



StockObserver.java > ...

```java
public interface StockObserver {
    void update(String stockName, float stockPrice);
}

```

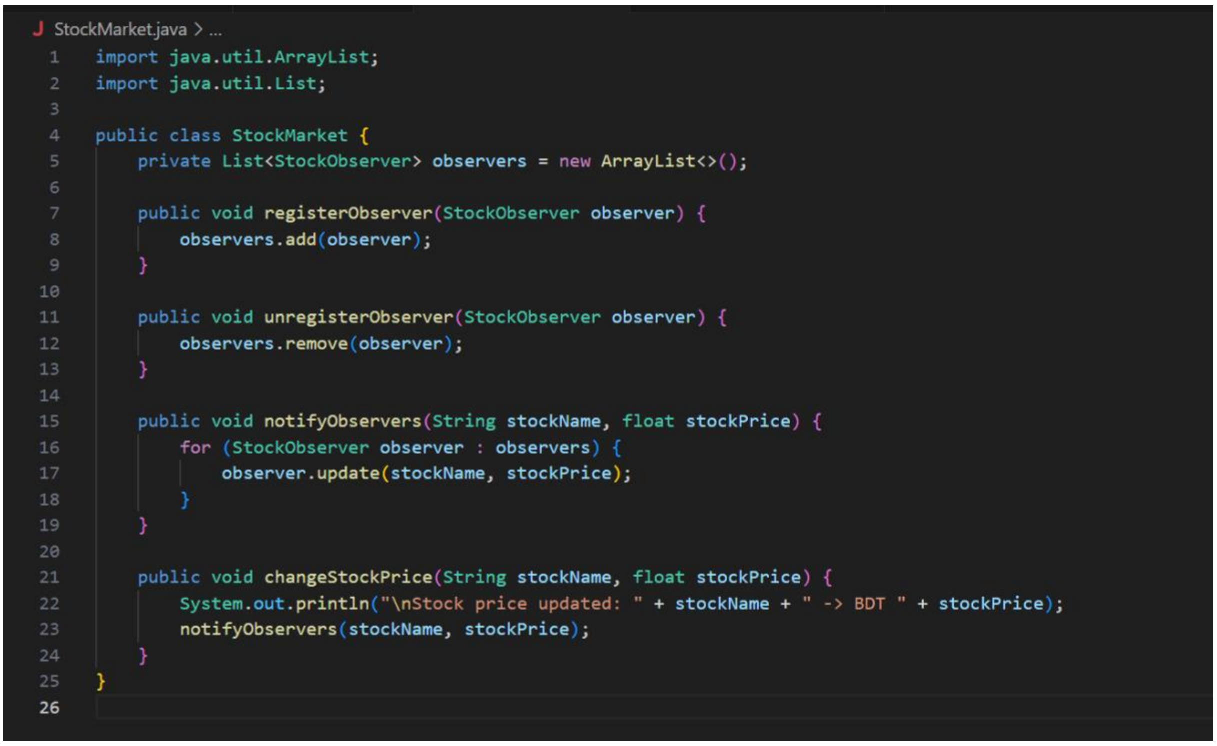
Investor(Subscriber):



Investor.java > Investor

```java
public class Investor implements StockObserver {
    private String name;

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

    @Override
    public void update(String stockName, float stockPrice) {
        System.out.println("\n" + name + " notified of " + stockName + " price change: BDT" + stockPrice);
    }
}
```

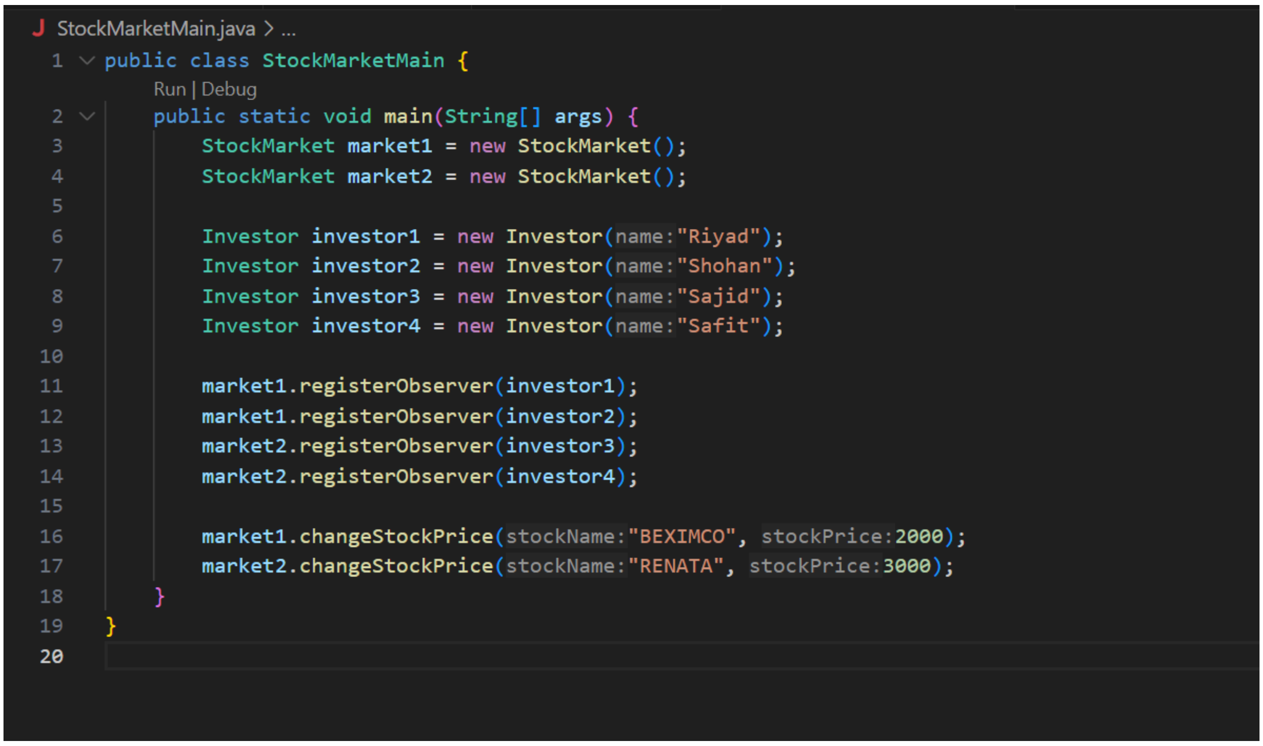
StockMarket(Publisher):



StockMarket.java > ...

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

public class StockMarket {
    private List<StockObserver> observers = new ArrayList<>();

    public void registerObserver(StockObserver observer) {
        observers.add(observer);
    }

    public void unregisterObserver(StockObserver observer) {
        observers.remove(observer);
    }

    public void notifyObservers(String stockName, float stockPrice) {
        for (StockObserver observer : observers) {
            observer.update(stockName, stockPrice);
        }
    }

    public void changeStockPrice(String stockName, float stockPrice) {
        System.out.println("\nStock price updated: " + stockName + " -> BDT " + stockPrice);
        notifyObservers(stockName, stockPrice);
    }
}

```

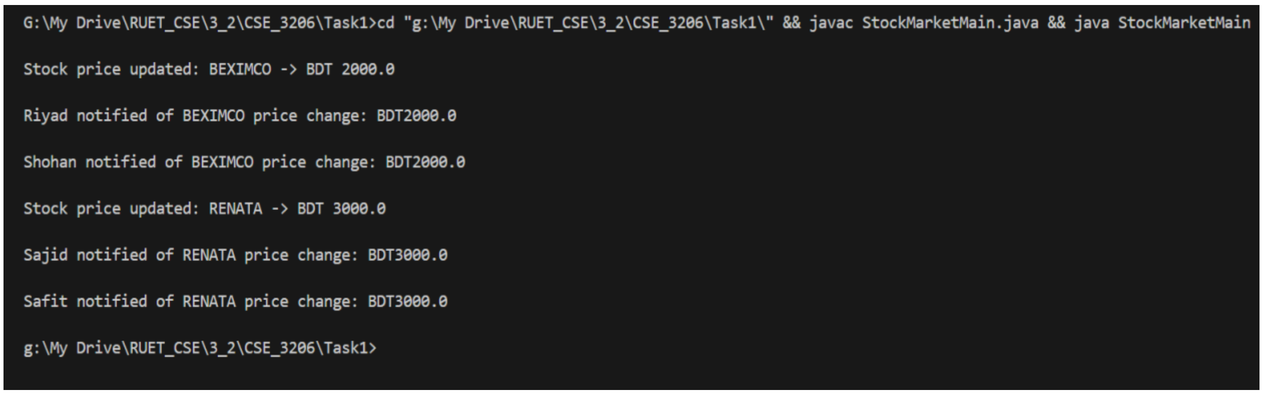
StockMarketMain:



StockMarketMain.java > ...

```java
public class StockMarketMain {
    Run | Debug
    public static void main(String[] args) {
        StockMarket market1 = new StockMarket();
        StockMarket market2 = new StockMarket();

        Investor investor1 = new Investor(name:"Riyad");
        Investor investor2 = new Investor(name:"Shohan");
        Investor investor3 = new Investor(name:"Sajid");
        Investor investor4 = new Investor(name:"Safit");

        market1.registerObserver(investor1);
        market1.registerObserver(investor2);
        market2.registerObserver(investor3);
        market2.registerObserver(investor4);

        market1.changeStockPrice(stockName:"BEXIMCO", stockPrice:2000);
        market2.changeStockPrice(stockName:"RENATA", stockPrice:3000);
    }
}
```

Output:



```
G:\My Drive\RUET_CSE\3_2\CSE_3206\Task1>cd "g:\My Drive\RUET_CSE\3_2\CSE_3206\Task1\" && javac StockMarketMain.java && java StockMarketMain

Stock price updated: BEXIMCO -> BDT 2000.0

Riyad notified of BEXIMCO price change: BDT2000.0

Shohan notified of BEXIMCO price change: BDT2000.0

Stock price updated: RENATA -> BDT 3000.0

Sajid notified of RENATA price change: BDT3000.0

Safit notified of RENATA price change: BDT3000.0

g:\My Drive\RUET_CSE\3_2\CSE_3206\Task1>
```

State behavioral design pattern

Objective: The State Design Pattern is a behavioral design pattern in software engineering that allows an object to change its behavior dynamically based on its internal state. By encapsulating state-specific behaviors into separate classes, this pattern promotes open-closed principle, making it easier to extend and maintain the code. It is particularly useful in scenarios where an object's behavior is highly dependent on its current state, such as finite state machines, UI components, or workflows.

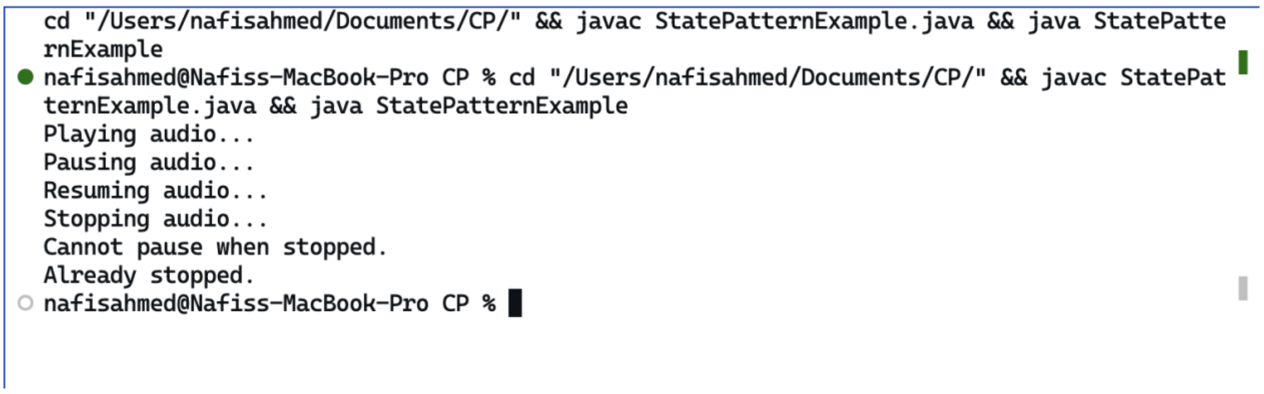
Example: Audio Player State Control

Code:

```
// State Interface
interface PlayerState {
    void play();
    void pause();
    void stop();
}

// Concrete States
class StoppedState implements PlayerState {
    private AudioPlayer player;

    public StoppedState(AudioPlayer player) {
        this.player = player;
    }

    @Override
    public void play() {
        System.out.println("Playing audio...");
        player.setState(new PlayingState(player)); // Change state
    }

    @Override
    public void pause() {
        System.out.println("Cannot pause when stopped.");
    }

    @Override
```

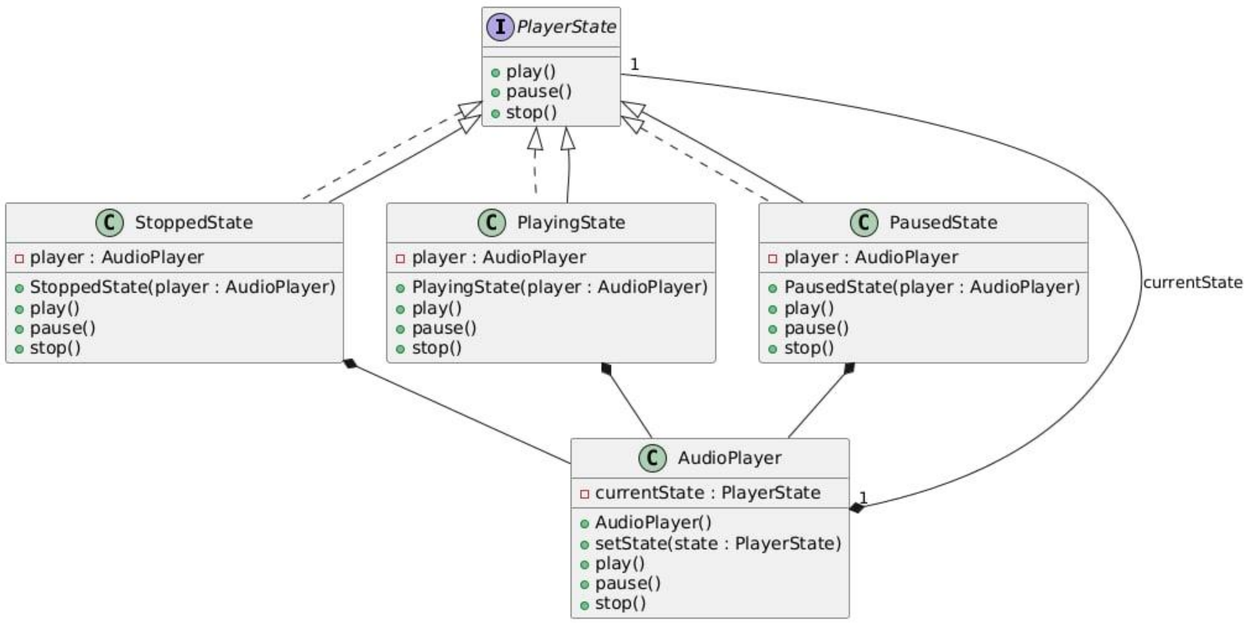
```java
    public void stop() {
        System.out.println("Already stopped.");
    }
}

class PlayingState implements PlayerState {
    private AudioPlayer player;

    public PlayingState(AudioPlayer player) {
        this.player = player;
    }

    @Override
    public void play() {
        System.out.println("Already playing.");
    }

    @Override
    public void pause() {
        System.out.println("Pausing audio...");
        player.setState(new PausedState(player)); // Change state
    }

    @Override
    public void stop() {
        System.out.println("Stopping audio...");
        player.setState(new StoppedState(player)); // Change state
    }
}

class PausedState implements PlayerState {
    private AudioPlayer player;

    public PausedState(AudioPlayer player) {
        this.player = player;
    }

    @Override
    public void play() {
        System.out.println("Resuming audio...");
        player.setState(new PlayingState(player)); // Change state
    }

    @Override
    public void pause() {
        System.out.println("Already paused.");
    }
    @Override
    public void stop() {
        System.out.println("Stopping audio...");
        player.setState(new StoppedState(player)); // Change state
    }
}

// Context
class AudioPlayer {
    private PlayerState currentState;

    public AudioPlayer() {
        currentState = new StoppedState(this); // Initial state
    }

    public void setState(PlayerState state) {
        this.currentState = state;
    }

    public void play() {
        currentState.play();
    }

    public void pause() {
        currentState.pause();
    }

    public void stop() {
        currentState.stop();
    }
}

// Client
public class StatePatternExample {
    public static void main(String[] args) {
        AudioPlayer player = new AudioPlayer();

        player.play(); // Playing audio...
        player.pause(); // Pausing audio...
        player.play(); // Resuming audio...
        player.stop(); // Stopping audio...
        player.pause(); // Cannot pause when stopped.
        player.stop();// Already stopped.
    }
}
```

Output:



```
  cd "/Users/nafisahmed/Documents/CP/" && javac StatePatternExample.java && java StatePatte
  rnExample
● nafisahmed@Nafiss-MacBook-Pro CP % cd "/Users/nafisahmed/Documents/CP/" && javac StatePat
  ternExample.java && java StatePatternExample
  Playing audio...
  Pausing audio...
  Resuming audio...
  Stopping audio...
  Cannot pause when stopped.
  Already stopped.
○ nafisahmed@Nafiss-MacBook-Pro CP %
```

UML Diagram:


PlayerState
+ play()
+ pause()
+ stop()
1
StoppedState
- player : AudioPlayer
+ StoppedState(player : AudioPlayer)
+ play()
+ pause()
+ stop()
PlayingState
- player : AudioPlayer
+ PlayingState(player : AudioPlayer)
+ play()
+ pause()
+ stop()
PausedState
- player : AudioPlayer
+ PausedState(player : AudioPlayer)
+ play()
+ pause()
+ stop()
currentState
AudioPlayer
- currentState : PlayerState
+ AudioPlayer()
+ setState(state : PlayerState)
+ play()
+ pause()
+ stop()
1