EASTERN INTERNATIONAL UNIVERSITY

SCHOOL OF COMPUTING

AND INFORMATION TECHNOLOGY

Lab 8

Practice Assignment – Quarter 2, 2023-2024

Course Name: OOP

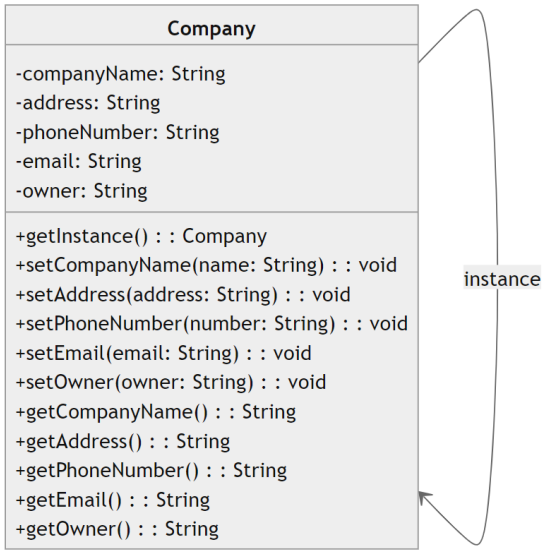
Course Code: CSE 203

Student's Full Name:

Student ID:

1. Singleton Pattern

Hong Ky Manufacturing Company deploys ERP software to manage all business activities. Company information includes (companyName, address, phoneNumber, email, owner). For convenience in calling company information displayed on company reports. The software development team has implemented these objects according to the Singleton Pattern.



Class CompanyInfo

```java
public class CompanyInfo {
    private static CompanyInfo instance;

    private String companyName;
    private String address;
    private String phoneNumber;
    private String email;
    private String owner;

    // Private constructor to prevent instantiation from outside
    private CompanyInfo() {
        // Initialize company information
        this.companyName = "MyCompany";
        this.address = "123 Main Street, City, Country";
        this.phoneNumber = "123-456-7890";
        this.email = "info@eiu.edu.com";
        this.owner = "";
    }
```

```java
// Public static method to get the single instance of the class
public static synchronized CompanyInfo getInstance() {
    if (instance == null) {
        instance = new CompanyInfo();
    }
    return instance;
}

// Getters and setters for company information
public String getCompanyName() {
    return companyName;
}

public void setCompanyName(String companyName) {
    this.companyName = companyName;
}

public String getAddress() {
    return address;
}

public void setAddress(String address) {
    this.address = address;
}

public String getPhoneNumber() {
    return phoneNumber;
}

public void setPhoneNumber(String phoneNumber) {
    this.phoneNumber = phoneNumber;
}

public String getEmail() {
    return email;
}

public void setEmail(String email) {
    this.email = email;
}
```

```
    public void setOwner(String owner) {
        this.owner = owner;
    }

    // Method to display company information
    public void displayInfo() {
        System.out.println("Company Name: " + companyName);
        System.out.println("Address: " + address);
        System.out.println("Phone Number: " + phoneNumber);
        System.out.println("Email: " + email);
        System.out.println("Owner: " + owner);
    }
```

Class SingletonPattern

```
public class SingletonPattern {
    public static void main(String[] args) {
        // Get the instance of CompanyInfo
        CompanyInfo companyInfo = CompanyInfo.getInstance();

        // Display the company information
        companyInfo.displayInfo();

        // Update the company information
        companyInfo.setCompanyName("EIU-Đại học Quốc tế Miền Đông");
        companyInfo.setAddress("3M38+6F3, Nam Kỳ Khởi Nghĩa, Định Hoà, Thủ Dầu Một, Bình Dương");
        companyInfo.setPhoneNumber("(+84) 0274 222 0372");

        // Display the updated company information
        companyInfo.displayInfo();
    }

}
```

2. Observer Pattern

Project Overview:

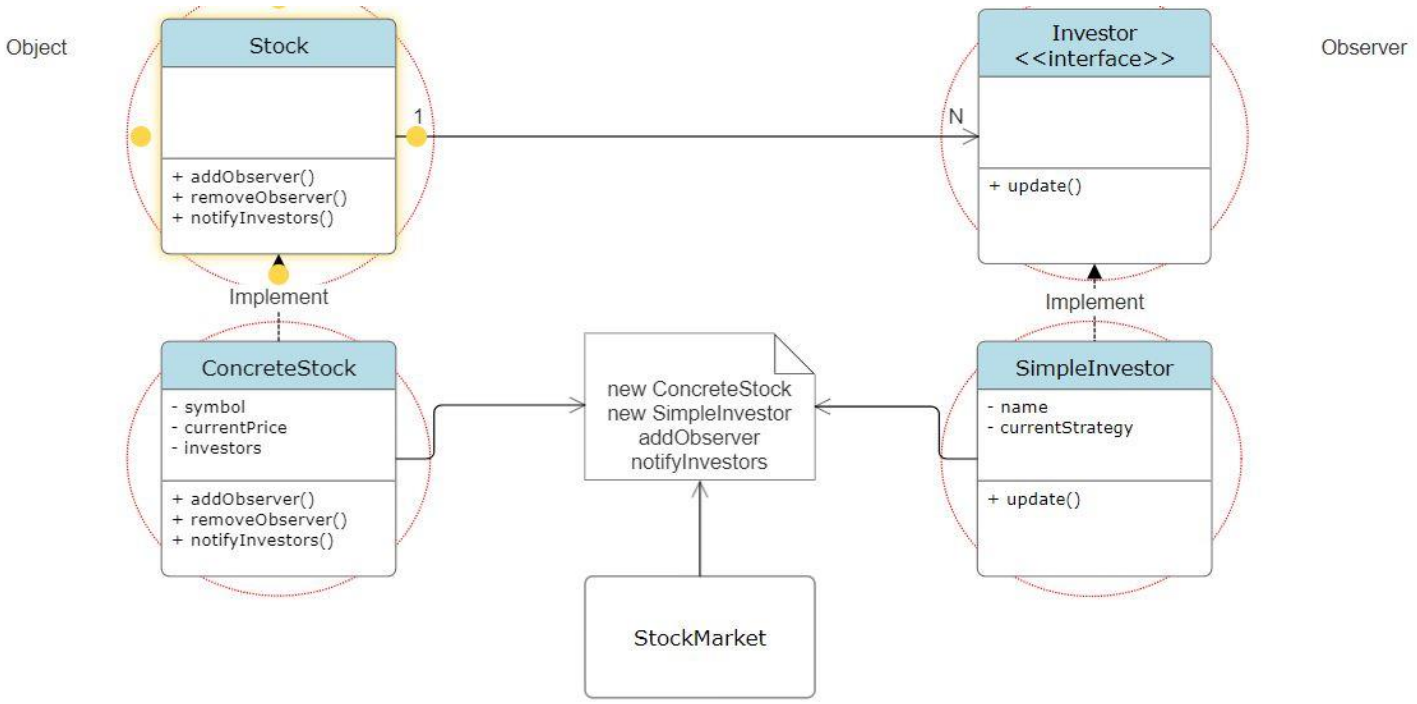
The Stock Market Monitoring System is designed to provide real-time updates to investors about significant changes in stock prices. Investors subscribe to receive real-time updates about specific stocks they are interested in, and when there is a significant change in the stock prices they will be notified so that they can change their investment Strategy at once. This is one of the practical scenarios for applying the Observer pattern. The system utilizes the Observer pattern to establish a one-to-many relationship between stocks (**subjects**) and investors (**observers**).

Project Components:

1. Stock (**Subject**):
    - Interface outlines the operations a Stock operation including:
    - **Add** and **remove** investors from list.

- **Notifies** investors when there is a significant change in the stock price: For the sake of simplicity, notify method provide a new stock price (100, 120…) instead of the difference in value (+10, +20, -30...).

2. ConcreteStock (**Concrete Subject**):
   - Represents a stock with a unique symbol: e.g., AAPL for Apple
   - Maintains the current stock price: Market value.
   - ConcreteStock is the concrete subject implementing the “Subject" interface.
   - It maintains a list of observers (“Investor") and provides methods to manage this list
3. Investor (**Observer**):
   - An interface or abstract class representing the observer: Contains a method (e.g., **update**) that is called by the stock when there is a significant price change.
4. Strategy (enum):
   - A simple enumeration presents the common Strategy that investor may adapt: MonitorOnly, QuickTransactions, HoldPositions, HedgingStrategies.
5. SimpleInvestor (**Concrete Observers**):
   - A SimpleInvestor class that implements the observer interface.
   - SimpleInvestor has a name and keeps their current Strategy.
   - SimpleInvestor default behavior is “MonitorOnly”.
   - SimpleInvestor prefers adapting “HedgingStrategies” when stock price is lower than 50 but they are likely to adopt “QuickTransactions” when the price go higher than 200. (implement this behavior by simply **printing out** name, stock symbol, current price and current Strategy).
6. StockMarket (Client Code):
   - Write client code to demonstrate the usage of the Observer pattern.
   - Create instances of stocks, investors, and the stock market.
   - Register investors for specific stocks.
   - Simulate changes in stock prices and observe how investors are notified.



Implement stock market monitoring system using Observer pattern

Interface Stock

```
package Observer;

import java.util.ArrayList;
import java.util.List;
public interface Stock {
    public void addObserver(Investor investor);
    public void removeObserver(Investor investor) ;
    public void notifyInvestors(double priceChange);

}
```

Class ConcreteStock

```java
package Observer;

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

public class ConcreteStock implements Stock{
        private String symbol;
    private double currentPrice;
    private List<Investor> investors = new ArrayList<>();
    public ConcreteStock(String sysbol, double currentPrice)
    {
        this.symbol = sysbol;
        this.currentPrice = currentPrice;
    }
    public void addObserver(Investor investor) {
        this.investors.add(investor);
    }

    public void removeObserver(Investor investor) {
        this.investors.remove(investor);
    }

    public void notifyInvestors(double priceChange) {
        currentPrice = priceChange;
        for (Investor investor : investors) {
            investor.update(symbol, priceChange);
        }
    }
}
```

Class Investor

```java
package Observer;

interface Investor {
        String getName();
        Strategy addaptStrategy();
    void update(String stockSymbol, double priceChange);
}
```

Class SimpleInvestor

```
package Observer;


public class SimpleInvestor implements Investor {
        private String name;
        private Strategy currentStrategy;
        public SimpleInvestor(String name)
        {
                this.name = name;
                this.currentStrategy = Strategy.MonitorOnly;
        }
        @Override
        public String getName() {
                return name;
        }
        @Override
        public Strategy addaptStrategy() {
                return currentStrategy;
        }
        @Override
        public void update(String stockSymbol, double priceChange) {
                System.out.print(getName() +" was notified by the price change in " + stockSymbol + ": $" + priceChange)
                if(priceChange <50)
                {
                        System.out.println(" and change their strategy to "+Strategy.HedgingStrategies);
                }
                else {
                        if(priceChange >200)
                        {
                                System.out.println(" and change their strategy to "+Strategy.QuickTransactions);
                        }
                        else {
                                System.out.println (" and decide to do nothing");
                        }
                }
        }

}
```

Enum Strategy

```
package Observer;

public enum Strategy {
        MonitorOnly,
        QuickTransactions,
        HoldPositions,
        HedgingStrategies
}
```

Class StockMarket

```java
package Observer;

public class StockMarket {
    public static void main(String[] args) {
        Stock appleStock = new ConcreteStock("APP",100);
        Stock dellStock = new ConcreteStock("DEll",100);
        // Register basic investor
        Investor tomInvestor = new SimpleInvestor("Tom");
        Investor bobInvestor = new SimpleInvestor("Bob");
        appleStock.addObserver(tomInvestor);
        appleStock.addObserver(bobInvestor);
        dellStock.addObserver(tomInvestor);


        // Simulate a significant price change
        appleStock.notifyInvestors(100);
        appleStock.notifyInvestors(40.0);
        appleStock.notifyInvestors(10.0);
        appleStock.notifyInvestors(5.0);
        dellStock.notifyInvestors(300.0);
        dellStock.notifyInvestors(200.0);
    }
}
```

Submitted: github link (public) and all source code (*.rar file) submitted to moodle