Software Architecture & Design SEC3071

Lecture No. 40

Muhammad Shahid
Department of Computer Science
National Textile University

shahid.abdullah@hotmail.com

Last Lecture Review

- Behavioral Design Patterns
- Design Principle
  - Encapsulate What Varies
  - Program to an Interface not to an Implementation
  - Favor Composition Over Inheritance
- Strategy Design Pattern
  - Applicability
  - Implementation

Agenda – What will you Learn Today?

Observer Design Pattern

Observer Design Pattern

Observer Pattern – Motivation

- When we partition a system into a collection of cooperation classes, it is desired that consistent state between participating objects is to be maintained
- This should be not achieved via tight coupling as against our basis design principle because for obvious reason this will reduce reusability

Observer Pattern – Observations

- There exists a consistent communication model between a set of dependent objects and an object that they are dependent on
- This allows the dependent objects to have their state synchronized with the object that they are dependent on

Observers and Subjects

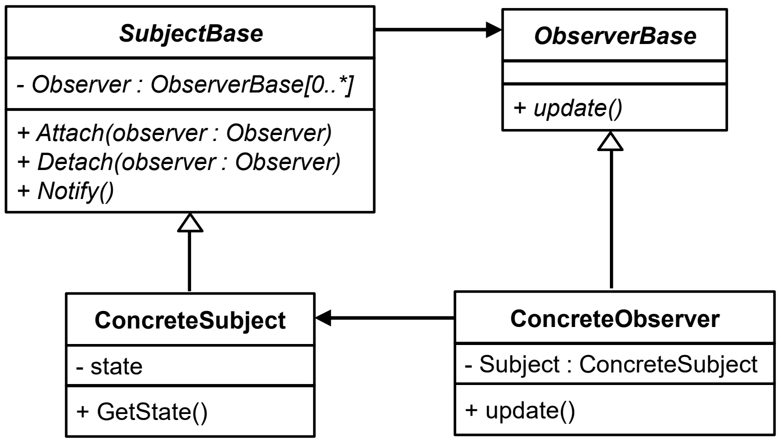
- The set of dependent objects are referred to as Observers

- The object on which Observer dependent is referred to as the Subject

Observer Pattern Defined

- "This pattern defines a one-to-many relationship between objects so that when there is change in the state of the one object it should be notified and automatically updated to all of it’s dependent."

Observer Pattern – Class Diagram


SubjectBase
- *Observer : ObserverBase[0..*]*
+ *Attach(observer : Observer)*
+ *Detach(observer : Observer)*
+ *Notify()*
ObserverBase
+ *update()*
ConcreteSubject
- state
+ GetState()
ConcreteObserver
- Subject : ConcreteSubject
+ update()

Observer – Real Life Example

Observer – Real Life Example

```
public abstract class Stock
{
      private string symbol;
      private double price;

      private List<IInvestor> investors = new List<IInvestor>();

      public Stock(string symbol, double price)
      {
            this.symbol = symbol;
            this.price = price;
      }

      public void Attach(IInvestor investor)
      {
            investors.Add(investor);
      }
```

Observer – Real Life Example

```
      public void Detach(IInvestor investor)
      {
            investors.Remove(investor);
      }

      public void Notify()
      {
            foreach (IInvestor investor in investors)
            {
                investor.Update(this);
            }
            Console.WriteLine("");
      }
```

Observer – Real Life Example

```
        // Gets or sets the price
        public double Price
        {
              get { return price; }
              set
              {
                  price = value;
                  Notify();
              }
        }

        // Gets the symbol
        public string Symbol
        {
              get { return symbol; }
        }
} // End f Stock class
```

Observer – Real Life Example

```
public class Gold : Stock
{
    // Constructor
    public Gold(string symbol, double price):base(symbol, price)
    { }
}

public interface IInvestor
{
    void Update(Stock stock);
}
```

Observer – Real Life Example

```
public class Investor : IInvestor
{
    private string name;
    private Stock stock;

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

    public void Update(Stock stock)
    {
            Console.WriteLine("Dear {0}, {1}'s prices change to",
                                    name, stock.Symbol, stock.Price);
    }

} // End of Investor class
```

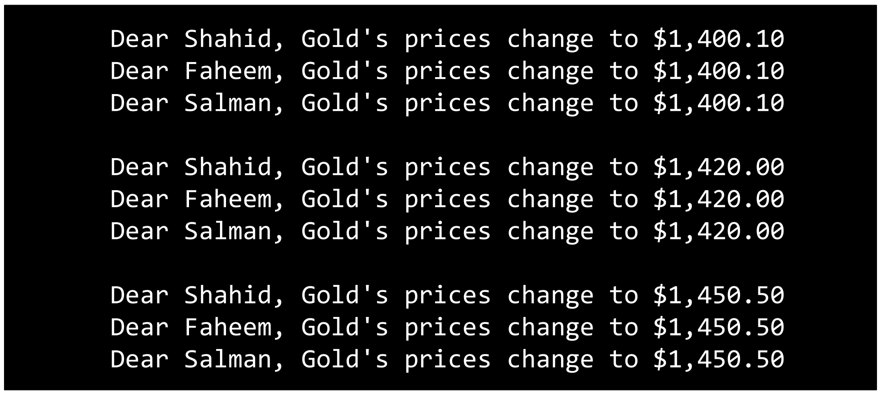
Observer – Real Life Example

```
static void Main(string[] args)
{
        // Create and attach investors
        Gold gold = new Gold("Gold", 1400.00);
        Investor shahid = new Investor("Shahid");
        Investor faheem = new Investor("Faheem");
        Investor salman = new Investor("Salman");

        gold.Attach(shahid);
        gold.Attach(faheem);
        gold.Attach(salman);

        // Fluctuating prices will notify investors
        gold.Price = 1400.10;
        gold.Price = 1420.00;
        gold.Price = 1450.50;
} // End of Main()
```

Observer Pattern – Implementation



```
Dear Shahid, Gold's prices change to $1,400.10
Dear Faheem, Gold's prices change to $1,400.10
Dear Salman, Gold's prices change to $1,400.10

Dear Shahid, Gold's prices change to $1,420.00
Dear Faheem, Gold's prices change to $1,420.00
Dear Salman, Gold's prices change to $1,420.00

Dear Shahid, Gold's prices change to $1,450.50
Dear Faheem, Gold's prices change to $1,450.50
Dear Salman, Gold's prices change to $1,450.50
```

Observer – Real Life Example

```
static void Main(string[] args)
{
        ...
        ...
        // Detching investor
        gold.Detach(shahid);

        // Fluctuating prices will notify investors
        gold.Price = 1400.10;
        gold.Price = 1420.00;
        gold.Price = 1450.50;
} // End of Main()
```

Observer Pattern – Implementation

```
Dear Faheem, Gold's prices change to $1,400.10
Dear Salman, Gold's prices change to $1,400.10

Dear Faheem, Gold's prices change to $1,420.00
Dear Salman, Gold's prices change to $1,420.00

Dear Faheem, Gold's prices change to $1,450.50
Dear Salman, Gold's prices change to $1,450.50
```

Exercise

- Design and implement Animal Information System (AIS). The system should be able to handle a variety of different animal e.g. cat, lion, horse etc. The animals can eat, run and have other behaviors. Animals are of different types. Each animal has different behavior of eating and running. The system should be able to add more animals and behavior in future requirements.

Recap

- Behavioral Design Patterns
- Observer Pattern
  - Motivation
  - Observations
  - Observers and Subjects
  - Definition
  - Communication Mechanism
  - Class Diagram
  - Code Implementation
- Observer Pattern – Real Life Example

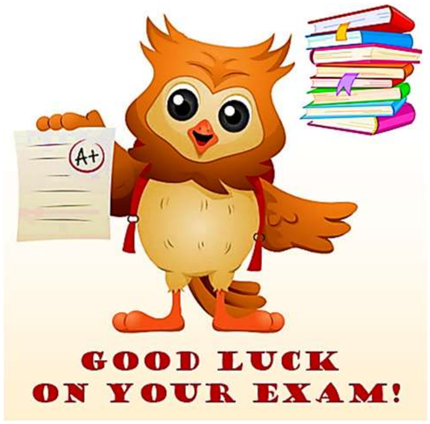
Thanks to Allah Almighty

Thank You All

Done

Your Comments are Welcomed

Good Luck for Exams



Questions

27 Software Architecture & Design – SEC3071