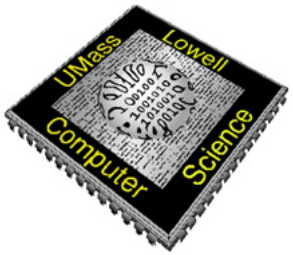


91.412: Software Engineering II

Design Patterns

Sarita Bassil, Ph.D.

Outline

1. What are design patterns?
2. Three categories of design patterns
3. “Observer” pattern
4. DoFactory Website
5. Get familiar with design patterns

Outline

1. **What are design patterns?**
2. Three categories of design patterns
3. “Observer” pattern
4. DoFactory Website
5. Get familiar with design patterns

1. What are design patterns?

- Pressman “quick” definition:
  - Design patterns are a **generic approach** for solving some **small problems** that can be **adapted** to a much wider variety of **specific problems**.
- GoF (The Gang of Four) - Gamma et al., 1995:
  - Suppose we have a number of OO systems, you will find **recurring** patterns of **classes and communicating objects**.
  - These patterns can solve specific design problems and make OO design more flexible, elegant, and **reusable**.
  - These patterns help designers reuse **successful designs** by basing new designs on prior experience.
  - As a designer, if you are **familiar** with such patterns, you can apply them immediately to design problems **without** having to **rediscover them**.

1. What are design patterns?

- Talking about the **assignment of responsibilities** when you move from the robustness diagram to the interaction diagram.
  - You should always look for every **opportunity** to reuse existing design patterns, when assigning responsibilities.

Outline

1. What are design patterns?
2. **Three categories of design patterns**
3. “Observer” pattern
4. DoFactory Website
5. Get familiar with design patterns

2. Three categories of design patterns

- The GoF patterns are generally considered the foundation for all other patterns.
- They are categorized in three groups:
  - Creational Patterns
    - Solutions for creating instances in a flexible way.
  - Structural Patterns
    - Solutions allowing to organize the structural layout of a set of classes and instances in a way to facilitate the maintenance.
  - Behavioral Patterns
    - Solutions allowing to organize the interactions between a number of instances in an efficient and maintainable way.

2. Three categories of design patterns

- Creational Patterns
  - Creational patterns are ones that create objects for you, rather than having you instantiate objects directly. This gives your program more flexibility in deciding which objects need to be created for a given case.
  - Disconnect the knowledge of an instance type (what) from the knowledge of the instance creation process (when and how).
    - **Abstract Factory**
    - **Builder**
    - **Factory Method**
    - **Prototype**
    - **Singleton**

2. Three categories of design patterns

- Structural Patterns
  - These concern class and object composition. They use inheritance to compose interfaces and define ways to compose objects to obtain new functionality.
    - **Adapter**
    - **Bridge**
    - **Composite**
    - **Decorator**
    - **Facade**
    - **Flyweight**
    - **Proxy**

2. Three categories of design patterns

- Behavioral Patterns
  - Most of these design patterns are specifically concerned with communication between objects.
    - **Chain of responsibility**
    - **Command**
    - **Iterator**
    - **Mediator**
    - **Memento**
    - **Observer (I will cover it in class)**
    - **State**
    - **Strategy**
    - **Template Method**
    - **Visitor**

Outline

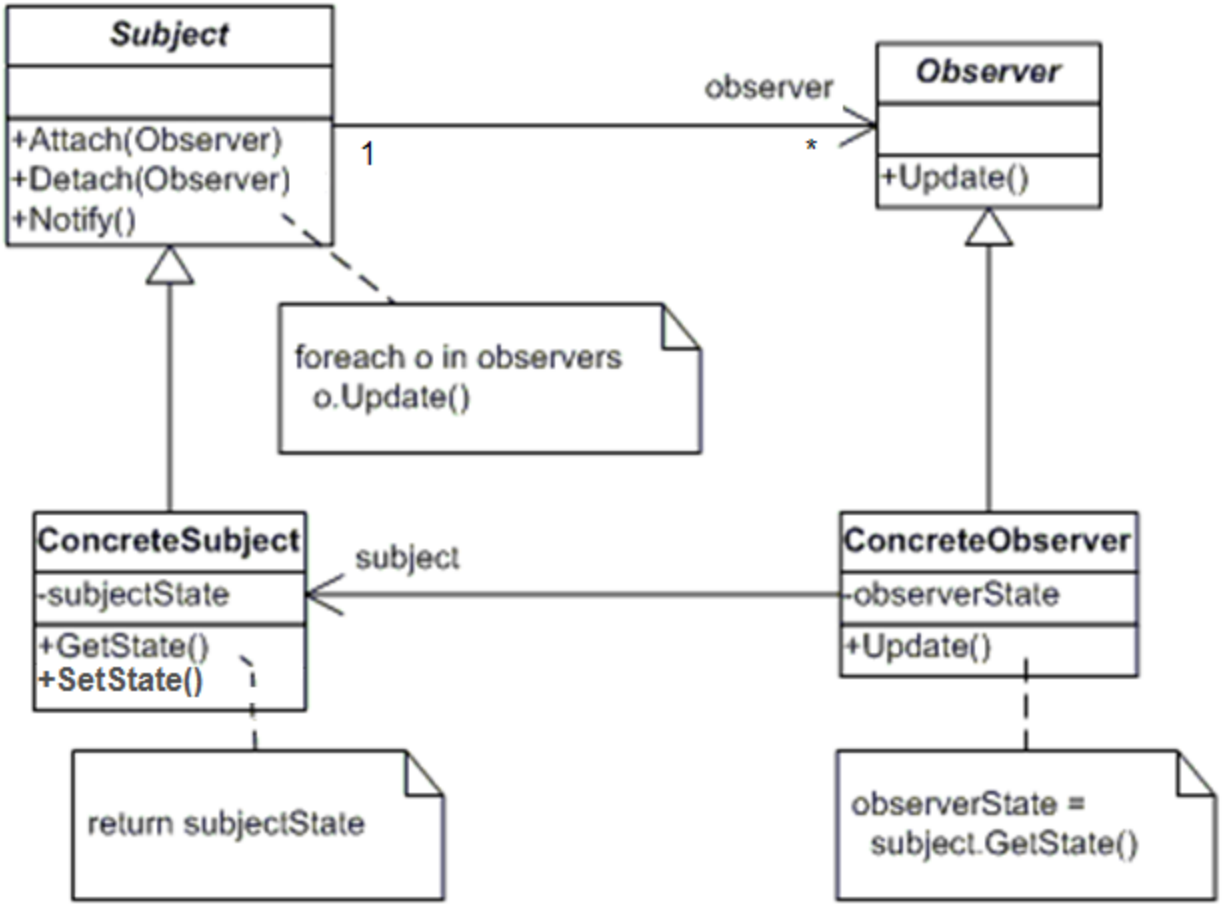
1. What are design patterns?
2. Three categories of design patterns
3. **“Observer” pattern**
4. DoFactory Website
5. Get familiar with design patterns

3. “Observer” pattern

- This is an example of a behavioral pattern.
- The “Observer” defines a one-to-many relationship, so that when one object (subject) changes state, the others (observers) are notified and updated automatically.
- Non-software example of the “Observer” pattern:
  - Some auctions demonstrate this pattern.
  - Each bidder posses a numbered paddle that is used to indicate a bid.
  - The auctioneer starts the bidding, and “observes” when a paddle is raised to accept the bid.
  - The acceptance of the bid changes the bid price, which is broadcast to all of the bidders on the form of a new bid.

3. “Observer” pattern

- UML class diagram of the “Observer” pattern (static structure)



3. “Observer” pattern

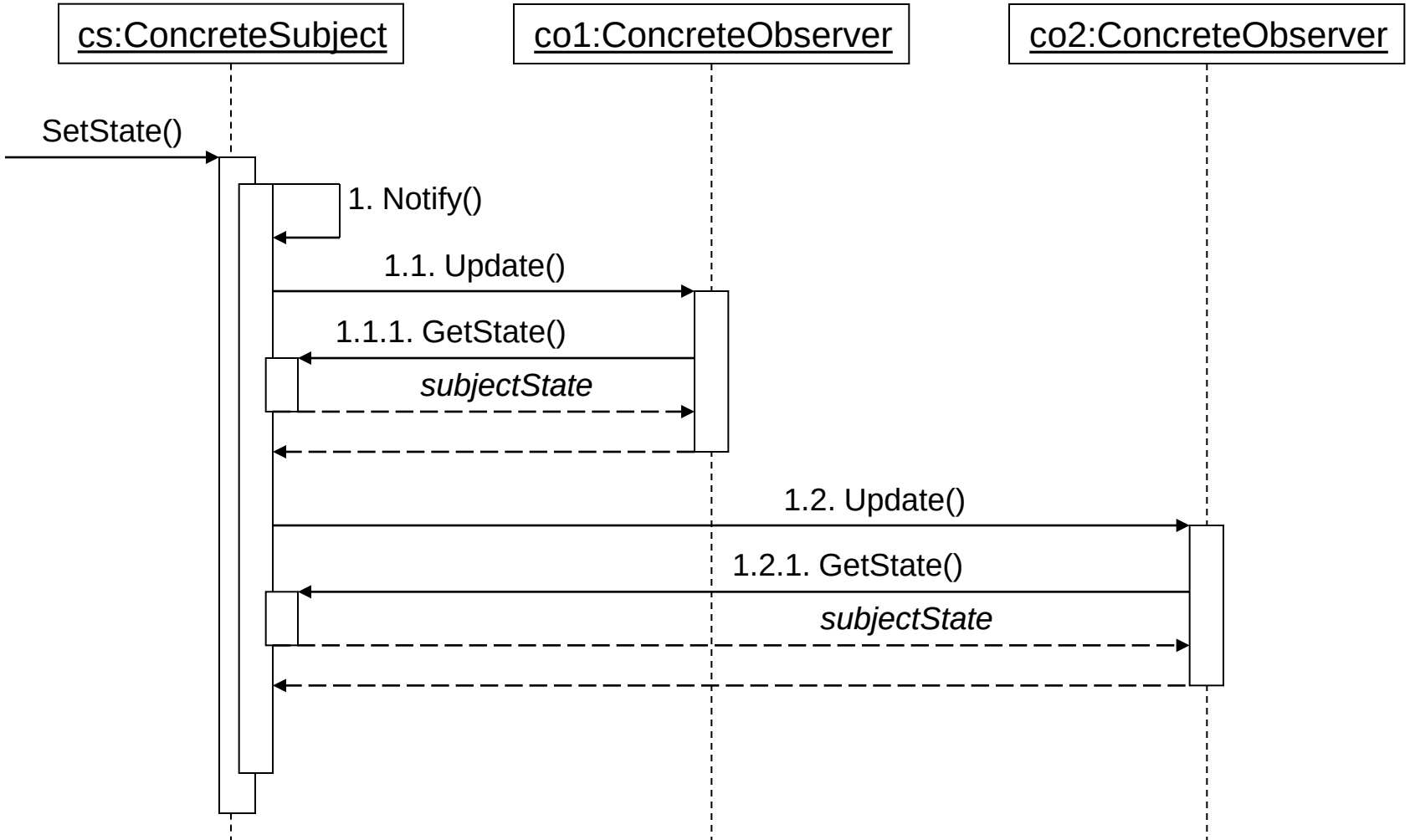
- Participants
  - The classes and/or objects participating in this pattern are:
    - **Subject (E.g., Stock)**
      - knows its observers. Any number of Observer objects may observe a subject.
      - provides an interface for attaching and detaching Observer objects.
    - **ConcreteSubject (E.g., IBM)**
      - stores state of interest to ConcreteObserver.
      - sends a notification to its observers when its state changes.

3. “Observer” pattern

- Participants
  - The classes and/or objects participating in this pattern are:
    - **Observer (E.g., IInvestor)**
      - defines an updating interface for objects that should be notified of changes in a subject.
    - **ConcreteObserver (E.g., Investor)**
      - maintains a reference to a ConcreteSubject object.
      - stores state that should stay consistent with the subject's.
      - implements the Observer updating interface to keep its state consistent with the subject's.

3. “Observer” pattern

- Sequence diagram of the “Observer” pattern (dynamic structure)



3. “Observer” pattern

- Here is the code in C# that demonstrates the “Observer” pattern in which registered investors are notified every time a stock changes value.

  [*Taken from: http://www.dofactory.com/*]

```
namespace DoFactory.GangOfFour.Observer.RealWorld
{
  /// <summary>
  /// MainApp startup class for Real-World
  /// Observer Design Pattern.
  /// </summary>
  class MainApp
  {
    /// <summary>
    /// Entry point into console application.
    /// </summary>
    static void Main()
    {
      // Create IBM stock and attach investors
      IBM ibm = new IBM("IBM", 120.00);
      ibm.Attach(new Investor("Sorros"));
      ibm.Attach(new Investor("Berkshire"));

      // Fluctuating prices will notify investors
      ibm.Price = 120.10;
      ibm.Price = 121.00;
      ibm.Price = 120.50;
      ibm.Price = 120.75;

      // Wait for user
      Console.ReadKey();
    }
  }
```

1/5

```
/// The 'Subject' abstract class
abstract class Stock
{
  private string _symbol;
  private double _price;
  private List<IInvestor> _investors = new List<IInvestor>();

  // Constructor
  public Stock(string symbol, double price)
  {
    this._symbol = symbol;
    this._price = price;
  }

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

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

2/5

```
  public void Notify()
  {
    foreach (IInvestor investor in _investors)
    {
      investor.Update(this);
    }

    Console.WriteLine("");
  }

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

  // Gets the symbol
  public string Symbol
  {
    get { return _symbol; }
  }
}
```

3/5

```
/// <summary>
/// The 'ConcreteSubject' class
/// </summary>
class IBM : Stock
{
  // Constructor
  public IBM(string symbol, double price)
    : base(symbol, price)
  {
  }
}

/// <summary>
/// The 'Observer' interface
/// </summary>
interface IInvestor
{
  void Update(Stock stock);
}
```

4/5

```
  /// <summary>
  /// The 'ConcreteObserver' class
  /// </summary>
  class Investor : IInvestor
  {
    private string _name;
    private Stock _stock;

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

    public void Update(Stock stock)
    {
      Console.WriteLine("Notified {0} of {1}'s " +
        "change to {2:C}", _name, stock.Symbol, stock.Price);
    }

    // Gets or sets the stock
    public Stock Stock
    {
      get { return _stock; }
      set { _stock = value; }
    }
  }
}
```

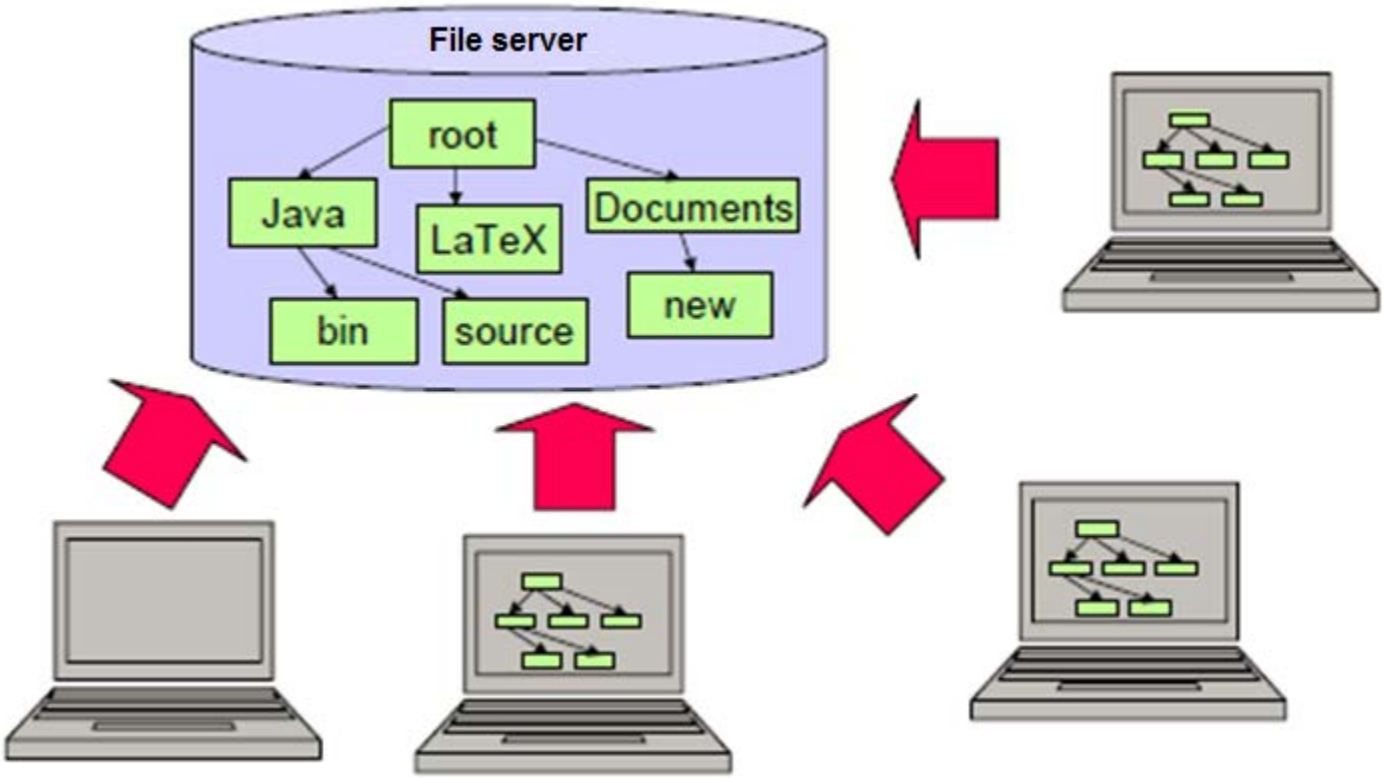
5/5

Output

```
Notified Sorros of IBM's change to $120.10
Notified Berkshire of IBM's change to $120.10

Notified Sorros of IBM's change to $121.00
Notified Berkshire of IBM's change to $121.00

Notified Sorros of IBM's change to $120.50
Notified Berkshire of IBM's change to $120.50

Notified Sorros of IBM's change to $120.75
Notified Berkshire of IBM's change to $120.75
```

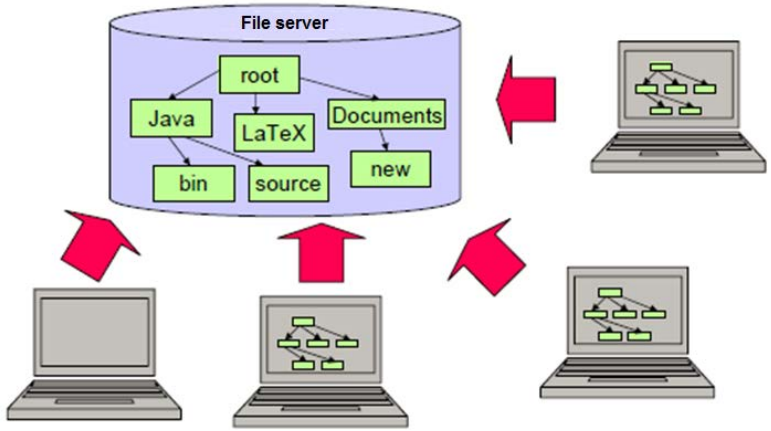
3. “Observer” pattern

- Example of a “remote file server”



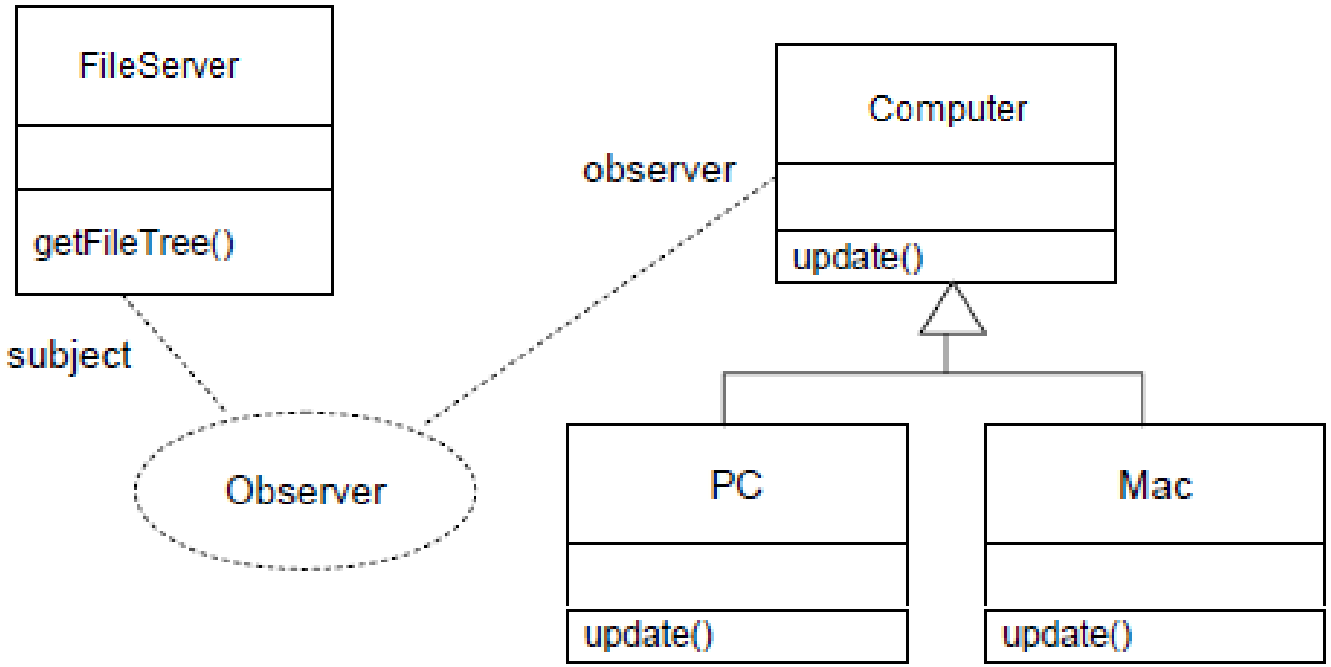
3. “Observer” pattern

- Example of a “remote file server”
  - Constraints:
    - The identity and the number of machines connecting to the server are unknown in advance.
    - New machines can be added to the system.
    - Polling is inappropriate, i.e., impossible or expensive solution especially with a big number of machines.
  - Solution:
    - “Observer” pattern.



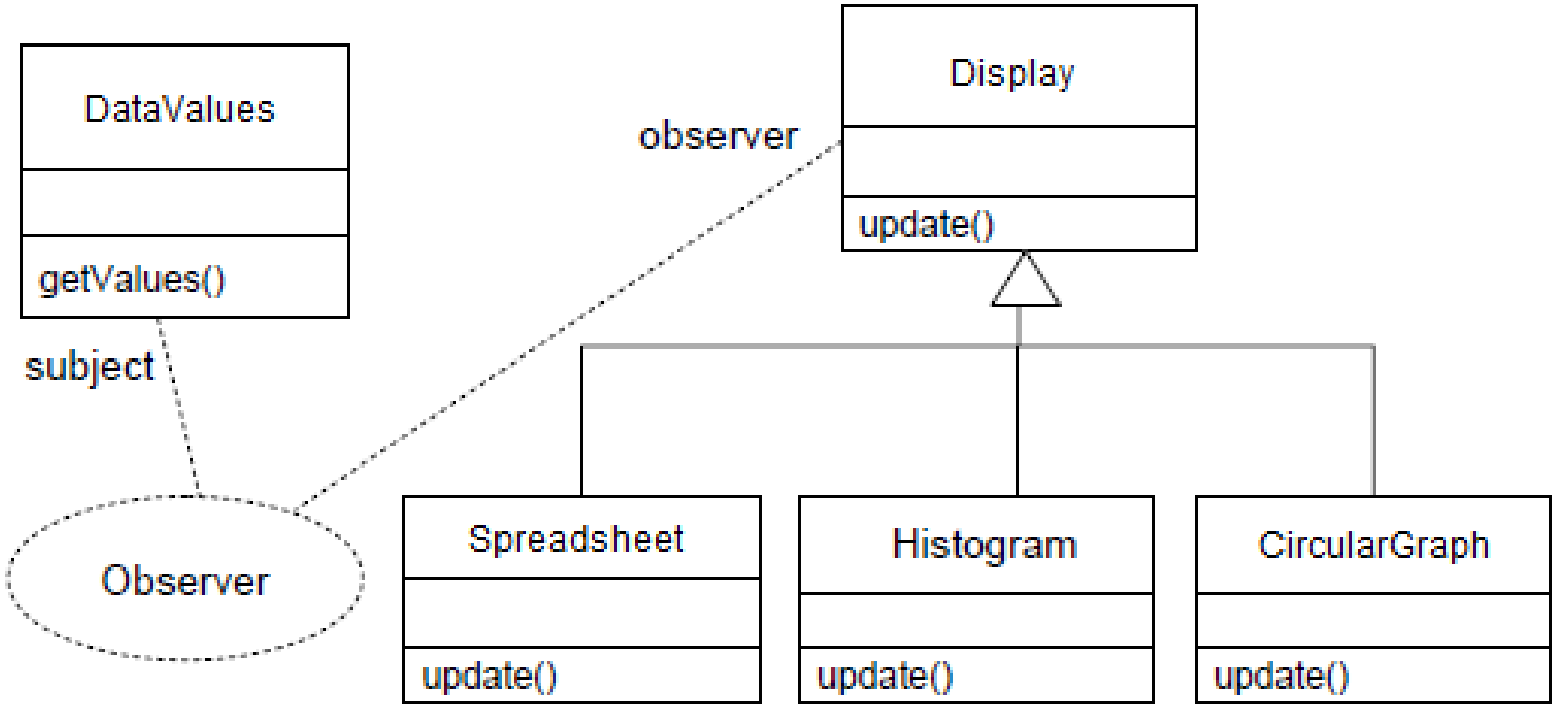
3. “Observer” pattern

- Example of a “remote file server”



3. “Observer” pattern

- Example: Visualizing data using three different forms



Outline

1. What are design patterns?
2. Three categories of design patterns
3. “Observer” pattern
4. **DoFactory Website**
5. Get familiar with design patterns

4. DoFactory Website

- http://www.dofactory.com
  - Go to: Tutorials → design patterns

  (http://www.dofactory.com/Patterns/Patterns.aspx)
    - Gives a catalog of the GoF patterns.
    - For every single pattern:
      - Definition
      - Frequency of use in industry (out of 5)
      - UML class diagram
      - List of the participants (classes participating to this pattern)
      - Real-world example
      - Sample Code in C#
        - » A **structural code** demonstrating the pattern.
        - » A **real-world code** demonstrating the pattern.
        - » A **.NET optimized code** (need to buy "Design Pattern Framework")

Outline

1. What are design patterns?
2. Three categories of design patterns
3. “Observer” pattern
4. DoFactory Website
5. **Get familiar with design patterns**

5. Get familiar with design patterns!

- I would like you to cover at least the following patterns:
  - **Factory method**
  - **Facade**
  - **Iterator**
  - **Composite**
  - **Singleton**
  - **Proxy**
  - **Command**

5. Get familiar with design patterns!

- Every two students are responsible of one pattern to be picked from the list on the previous slide.
- Access the description of the pattern on dofactory.com, generate a presentation of 10-15 minutes (slides are required) to be exposed and explained in class.
  - You will learn about each others patterns ☺
- Presentation will be done on:
  - **Friday 10/5/2012** and **Wednesday 10/10/2012**
- This will count as part of the midterm exam (50%).