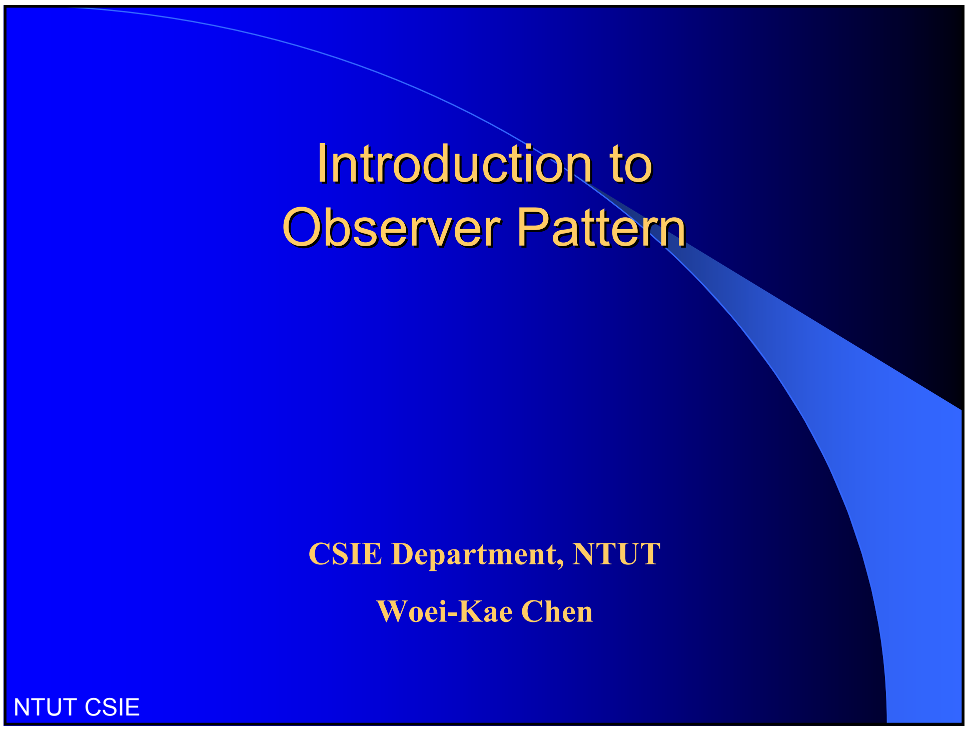
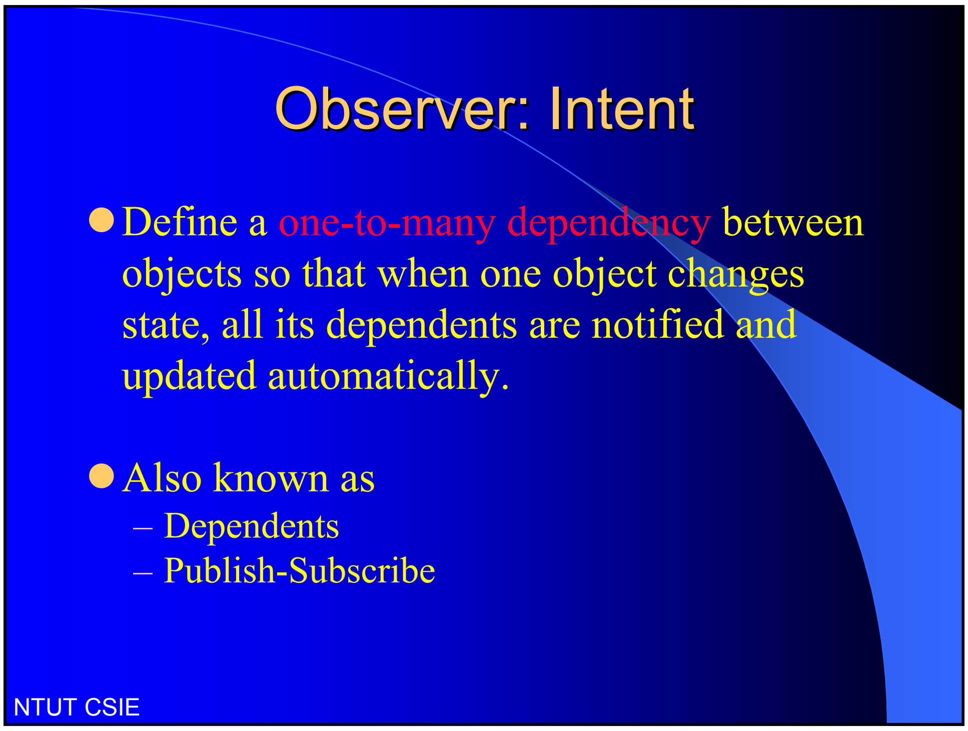


Introduction to Observer Pattern

CSIE Department, NTUT

Woei-Kae Chen

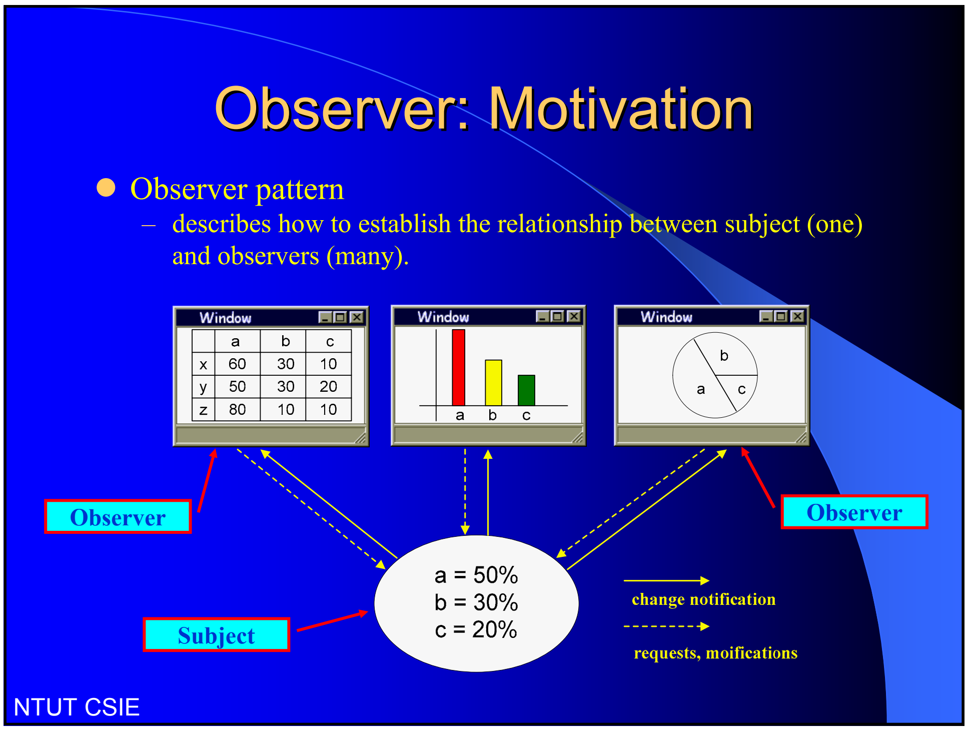
NTUT CSIE



Observer: Intent

- Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.
- Also known as
  - Dependents
  - Publish-Subscribe

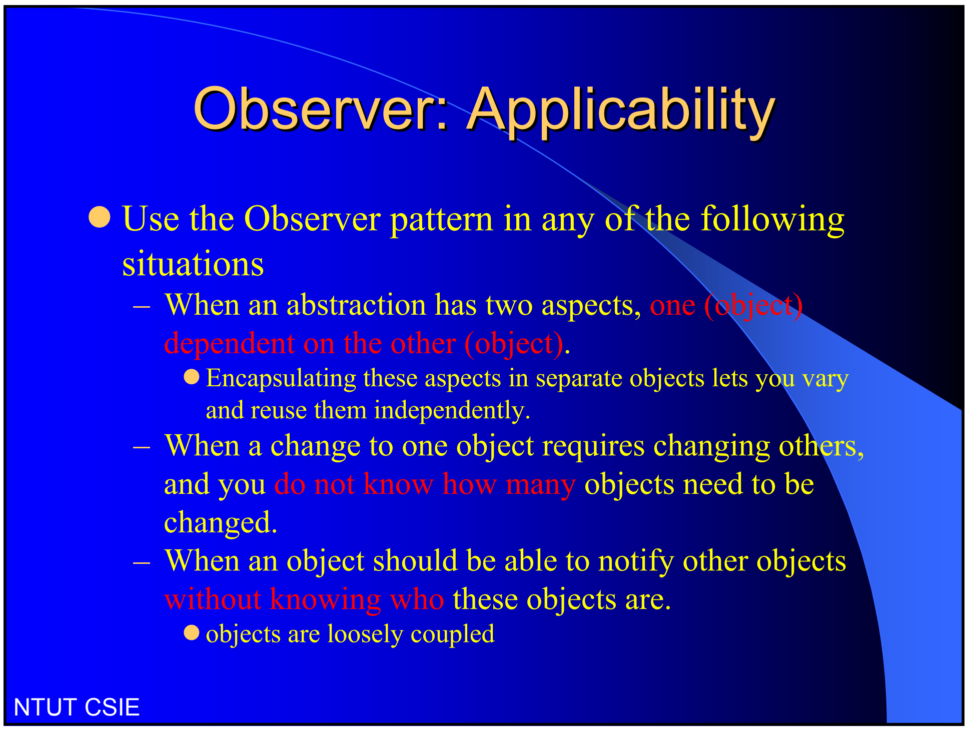
NTUT CSIE



Observer: Motivation

- Observer pattern
  - describes how to establish the relationship between subject (one) and observers (many).

NTUT CSIE

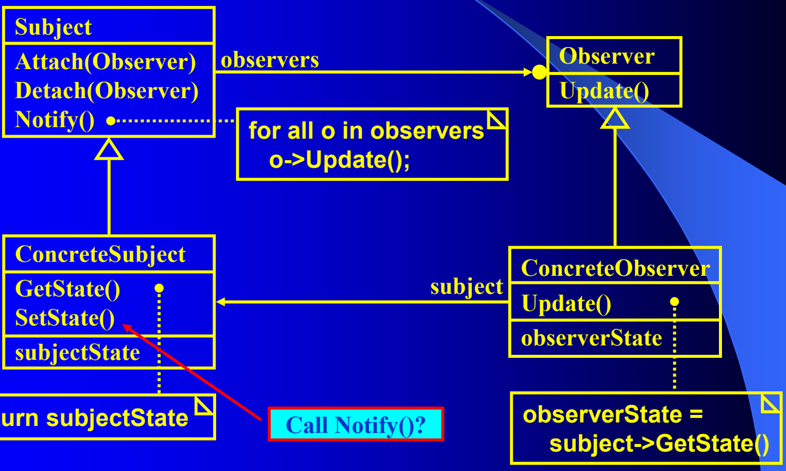


Observer: Applicability

- Use the Observer pattern in any of the following situations
  - When an abstraction has two aspects, one (object) dependent on the other (object).
    - Encapsulating these aspects in separate objects lets you vary and reuse them independently.
  - When a change to one object requires changing others, and you do not know how many objects need to be changed.
  - When an object should be able to notify other objects without knowing who these objects are.
    - objects are loosely coupled

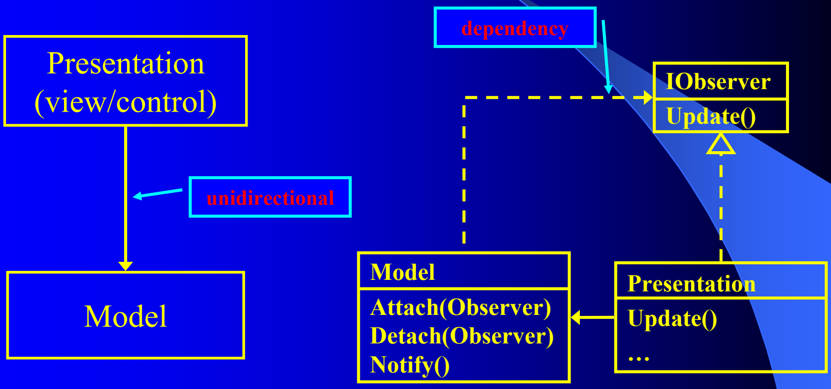
NTUT CSIE

Observer: Structure

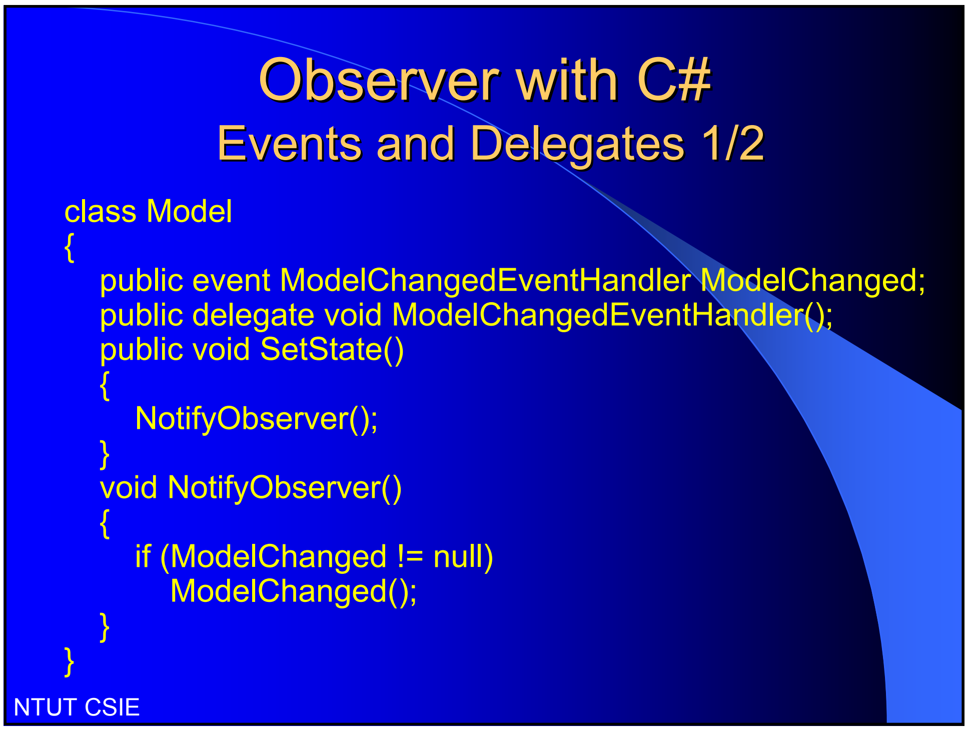


NTUT CSIE

Observer and MVC



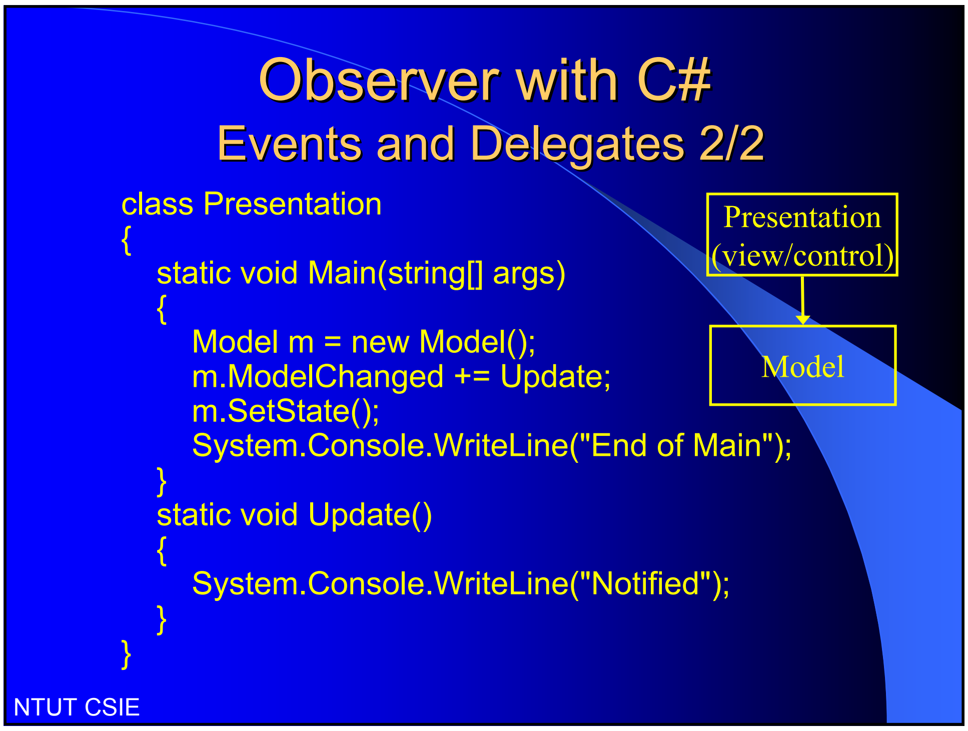
NTUT CSIE



Observer with C#
Events and Delegates 1/2

```
class Model
{
    public event ModelChangedEventHandler ModelChanged;
    public delegate void ModelChangedEventHandler();
    public void SetState()
    {
        NotifyObserver();
    }
    void NotifyObserver()
    {
        if (ModelChanged != null)
            ModelChanged();
    }
}
```

NTUT CSIE

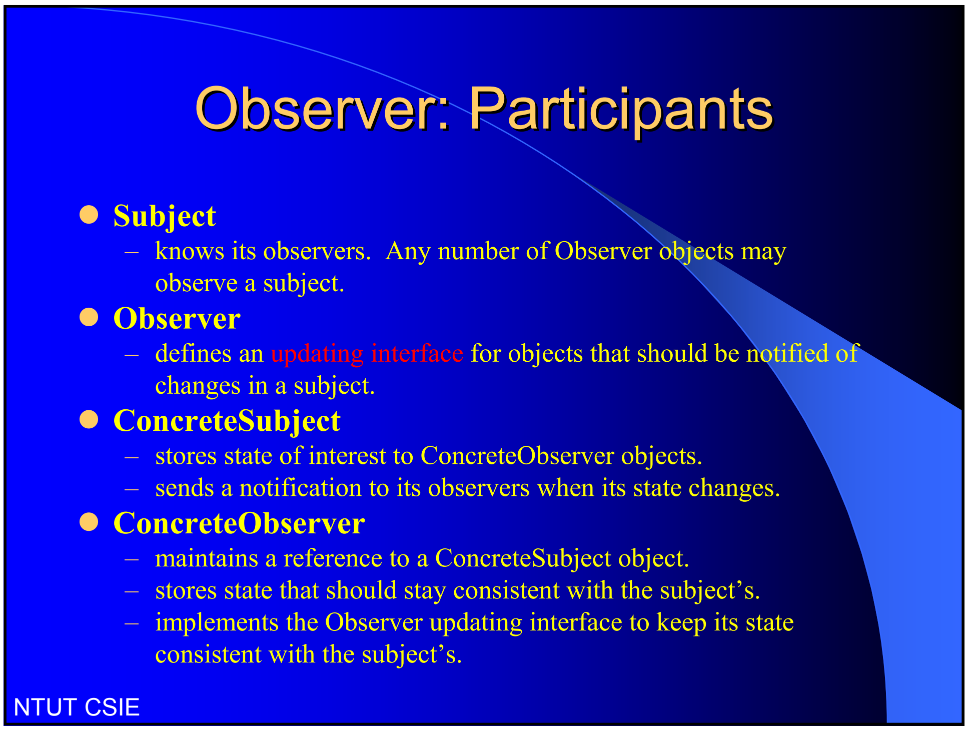


Observer with C#
Events and Delegates 2/2

```
class Presentation
{
    static void Main(string[] args)
    {
        Model m = new Model();
        m.ModelChanged += Update;
        m.SetState();
        System.Console.WriteLine("End of Main");
    }
    static void Update()
    {
        System.Console.WriteLine("Notified");
    }
}
```

$$\begin{tikzcd}\boxed{\text{Presentation (view/control)}} \arrow[d] \\ \boxed{\text{Model}}\end{tikzcd}$$

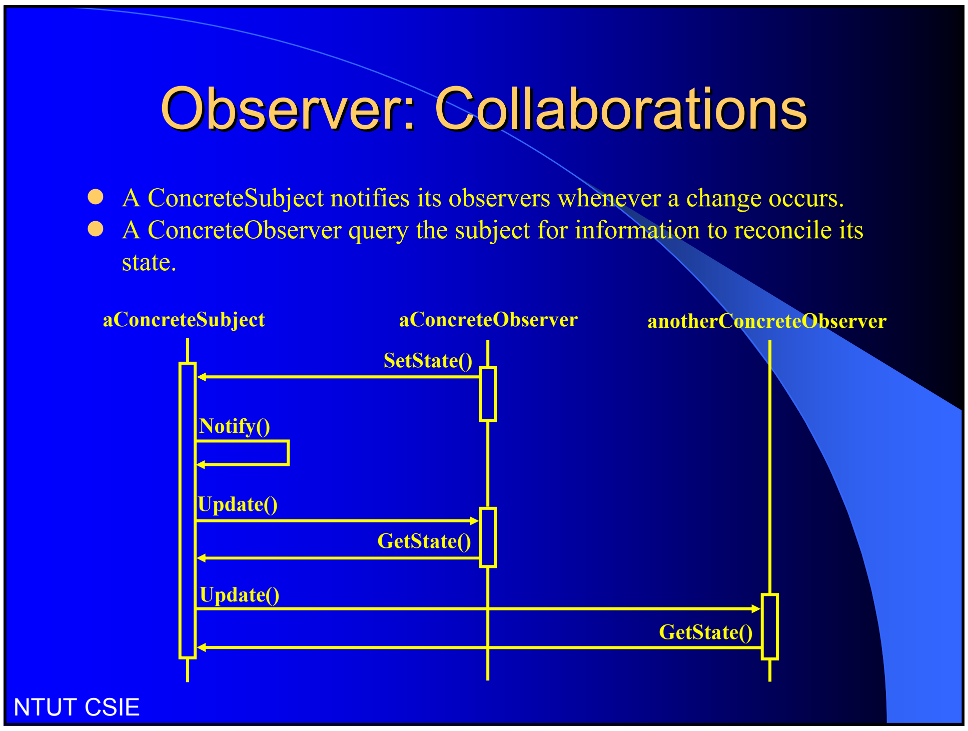
NTUT CSIE



Observer: Participants

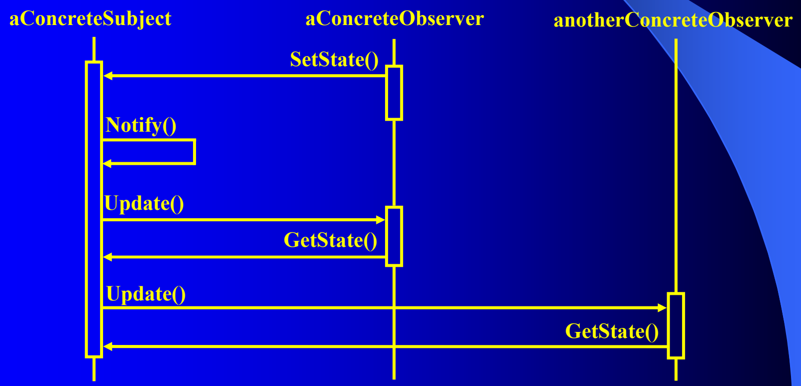
- **Subject**
  - knows its observers. Any number of Observer objects may observe a subject.
- **Observer**
  - defines an updating interface for objects that should be notified of changes in a subject.
- **ConcreteSubject**
  - stores state of interest to ConcreteObserver objects.
  - sends a notification to its observers when its state changes.
- **ConcreteObserver**
  - 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.

NTUT CSIE

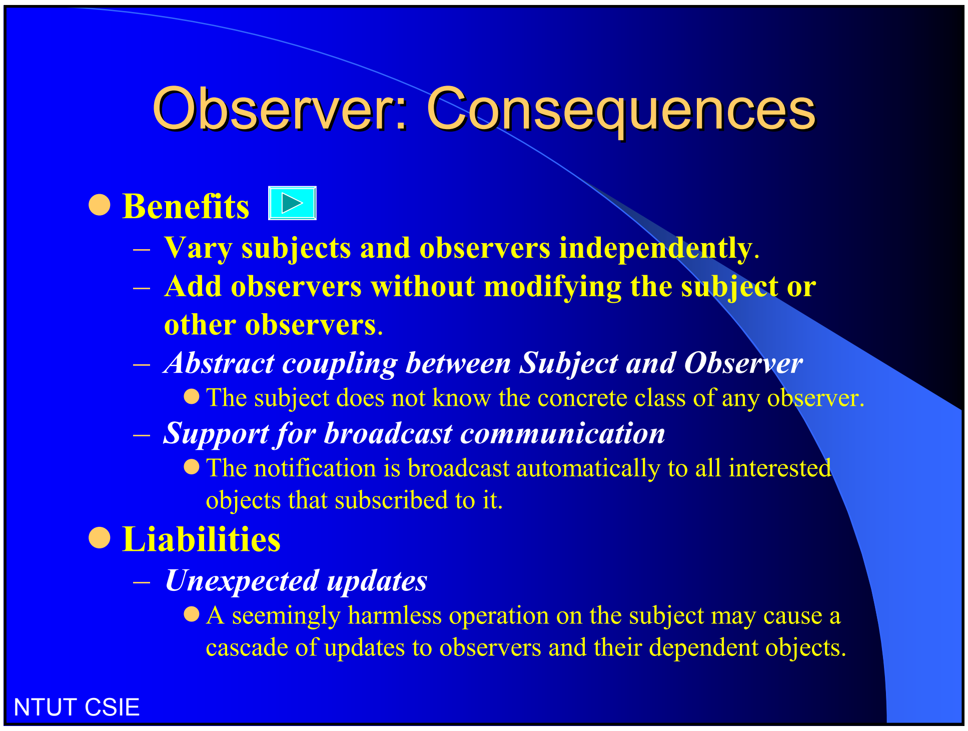


Observer: Collaborations

- A ConcreteSubject notifies its observers whenever a change occurs.
- A ConcreteObserver query the subject for information to reconcile its state.



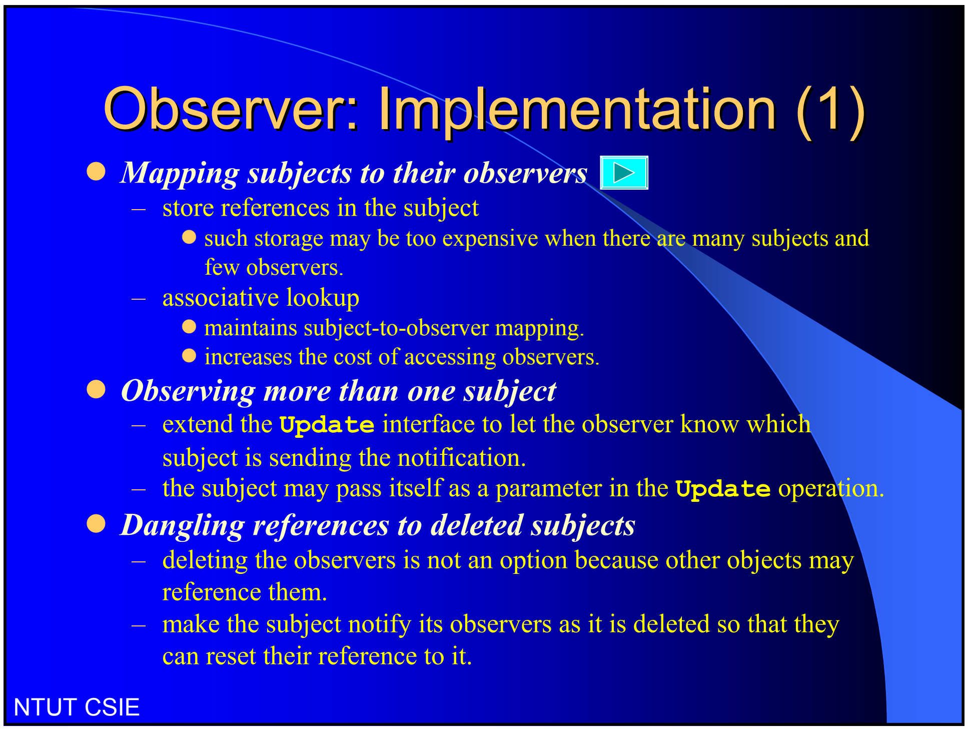
NTUT CSIE



Observer: Consequences

- **Benefits** ▷
  - **Vary subjects and observers independently.**
  - **Add observers without modifying the subject or other observers.**
  - ***Abstract coupling between Subject and Observer***
    - The subject does not know the concrete class of any observer.
  - ***Support for broadcast communication***
    - The notification is broadcast automatically to all interested objects that subscribed to it.
- **Liabilities**
  - ***Unexpected updates***
    - A seemingly harmless operation on the subject may cause a cascade of updates to observers and their dependent objects.

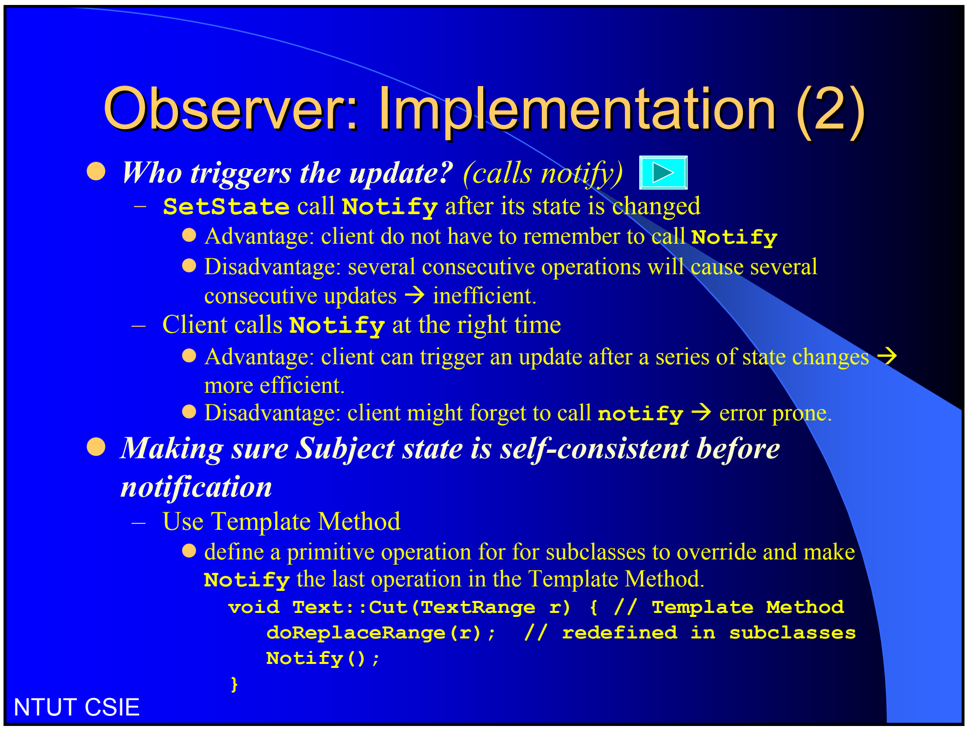
NTUT CSIE



Observer: Implementation (1)

- ***Mapping subjects to their observers*** ▷
  - store references in the subject
    - such storage may be too expensive when there are many subjects and few observers.
  - associative lookup
    - maintains subject-to-observer mapping.
    - increases the cost of accessing observers.
- ***Observing more than one subject***
  - extend the **`Update`** interface to let the observer know which subject is sending the notification.
  - the subject may pass itself as a parameter in the **`Update`** operation.
- ***Dangling references to deleted subjects***
  - deleting the observers is not an option because other objects may reference them.
  - make the subject notify its observers as it is deleted so that they can reset their reference to it.

NTUT CSIE

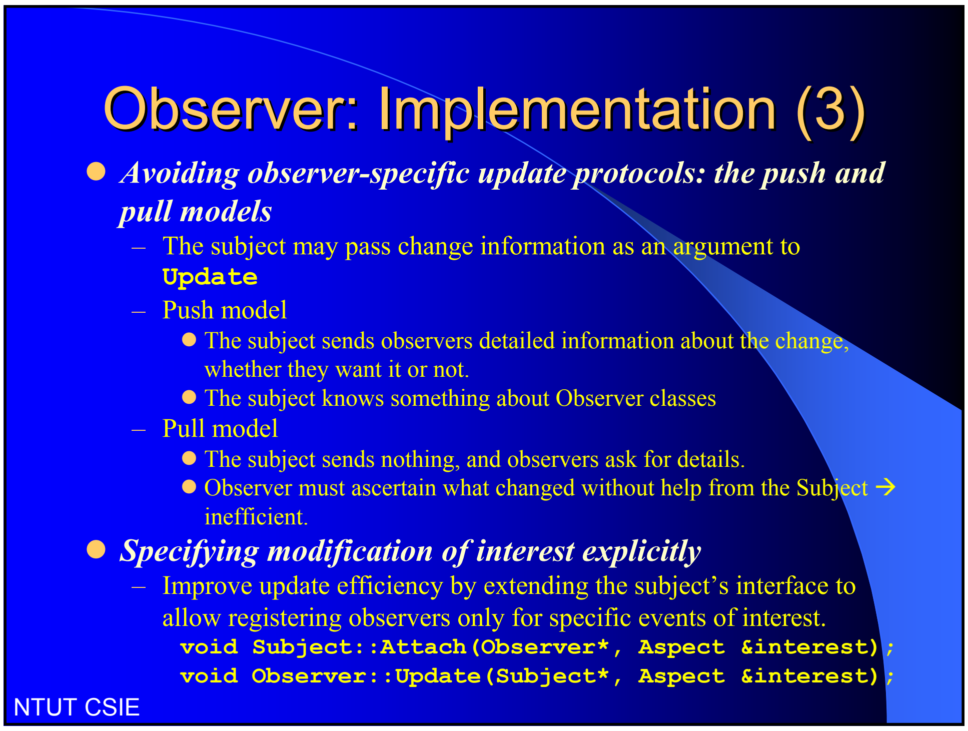


Observer: Implementation (2)

- ***Who triggers the update?*** *(calls notify)*
  - `SetState` call `Notify` after its state is changed
    - Advantage: client do not have to remember to call `Notify`
    - Disadvantage: several consecutive operations will cause several consecutive updates → inefficient.
  - Client calls `Notify` at the right time
    - Advantage: client can trigger an update after a series of state changes → more efficient.
    - Disadvantage: client might forget to call `notify` → error prone.
- ***Making sure Subject state is self-consistent before notification***
  - Use Template Method
    - define a primitive operation for for subclasses to override and make `Notify` the last operation in the Template Method.

```
void Text::Cut(TextRange r) { // Template Method
    doReplaceRange(r);   // redefined in subclasses
    Notify();
}
```

NTUT CSIE

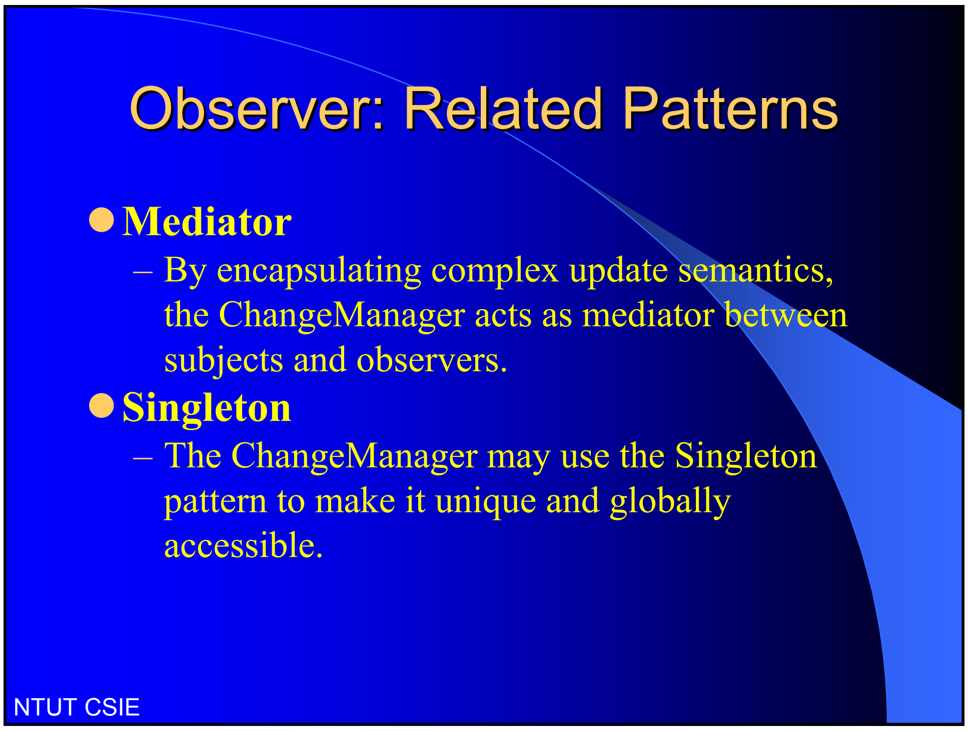


Observer: Implementation (3)

- ***Avoiding observer-specific update protocols: the push and pull models***
  - The subject may pass change information as an argument to `Update`
  - Push model
    - The subject sends observers detailed information about the change, whether they want it or not.
    - The subject knows something about Observer classes
  - Pull model
    - The subject sends nothing, and observers ask for details.
    - Observer must ascertain what changed without help from the Subject → inefficient.
- ***Specifying modification of interest explicitly***
  - Improve update efficiency by extending the subject's interface to allow registering observers only for specific events of interest.

```
void Subject::Attach(Observer*, Aspect &interest);
void Observer::Update(Subject*, Aspect &interest);
```

NTUT CSIE



Observer: Related Patterns

- **Mediator**
  - By encapsulating complex update semantics, the ChangeManager acts as mediator between subjects and observers.
- **Singleton**
  - The ChangeManager may use the Singleton pattern to make it unique and globally accessible.

NTUT CSIE