

CMSC 132:

OBJECT-ORIENTED PROGRAMMING II



Iterator, Marker, Observer Patterns

Department of Computer Science
University of Maryland, College Park

Design Patterns

- Descriptions of **reusable** solutions to common software design problems (e.g, Iterator pattern)
- Captures the experience of experts
- Goals
 - Solve common programming challenges
 - Improve reliability of solution
 - Aid rapid software development
 - Useful for real-world applications
- Design patterns are like recipes – generic solutions to expected situations
- Design patterns are language independent
- Recognizing when and where to use design patterns requires familiarity & experience
- Design pattern libraries serve as a glossary of idioms for understanding common, but complex solutions
- Design patterns are used throughout the Java Class Libraries

Iterator Pattern

- **Definition**
 - Move through collection of objects without knowing its internal representation
- **Where to use & benefits**
 - Use a standard interface to represent data objects
 - Uses standard iterator built in each standard collection, like List, Set, or Map
 - Need to distinguish variations in the traversal of an aggregate
- **Example**
 - Iterator for collection
 - Original
 - Examine elements of collection directly
 - Using pattern
 - Collection provides Iterator class for examining elements in collection

Iterator Example

```
public interface Iterator<V> {  
    bool hasNext();  
    V next();  
    void remove();  
}
```

```
Iterator<V> it = myCollection.iterator();
```

```
while ( it.hasNext() ) {  
    V x = it.next(); // finds all objects  
    ...             // in collection  
}
```

Marker Interface Pattern

- **Definition**
 - Label semantic attributes of a class
- **Where to use & benefits**
 - Need to indicate attribute(s) of a class
 - Allows identification of attributes of objects without assuming they are instances of any particular class
- **Example**
 - Classes with desirable property GoodProperty
 - *Original*
 - Store flag for GoodProperty in each class
 - *Using pattern*
 - Label class using GoodProperty interface
- **Examples from Java**
 - Cloneable
 - Serializable

Marker Interface Example

```
public interface SafePet { }    // no methods
```

```
class Dog implements SafePet { ... }  
class Piranha { ... }
```

```
Dog dog = new Dog();  
Piranha piranha = new Piranha();
```

```
if (dog instanceof SafePet) ...    // True  
if (piranha instanceof SafePet) ... // False
```

Observer Pattern

- **Definition**

- Updates all dependents of object automatically once object changes state

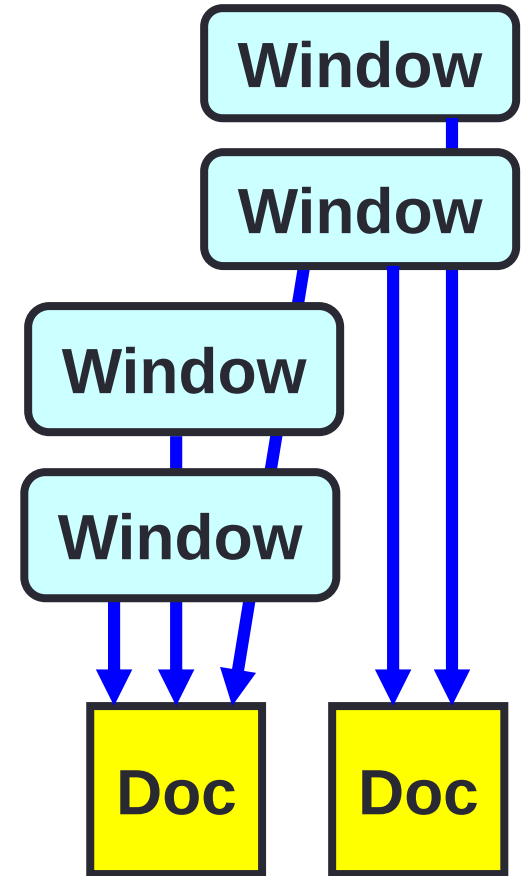
- **Where to use & benefits**

- One change affects one or many objects
- Many object behavior depends on one object state
- Need broadcast communication
- Maintain consistency between objects
- Observers do not need to constantly check for changes

Observer Pattern

- **Example**

- Multiple windows (views) for single document
- Original
 - Each window checks document
 - Window updates image if document changes
 - Think of window as asking “Are we there yet?”
- Using pattern
 - Each window registers as observer for document
 - Document notifies all of its observers when it changes



Observer Example

```
public interface Observer {  
    // called when observed object o changes  
    public void update(Observable o, Object a)  
}
```

```
public class Observable {  
    protected void setChanged()  
    protected void clearChanged()  
    boolean hasChanged()  
  
    void addObserver(Observer o)  
    void notifyObservers()  
    void notifyObservers(Object a)  
}
```

```
public class MyWindow implements Observer {  
    public openDoc(Observable doc) {  
        doc.addObserver(this); // add window to list  
    }  
    public void update(Observable doc, Object arg) {  
        redraw(doc); // display updated document  
    }  
}
```

```
public class MyDoc extends Observable {  
    public void edit() {  
        ... // edit document  
        setChanged(); // mark change  
        notifyObservers(arg); // invokes update()  
    }  
}
```