Topics

1) How to best loop through some items?
2) How to best notify an object of a change?
3) How to best organize classes in an application?
4) How can design ideas be reused?
Iterator
Accessing Items in a Collection

**Java Iterator:**

```java
List<String> words = // <snip>

Iterator<String> iterator = words.iterator();
while (iterator.hasNext()) {
    String word = iterator.next();
    // <snip>
}
```

**Direct Link List Code**

```java
Node current = words.head();
while (current != null) {
    String word = current.getData();
    current = current.nextNode();
}
```

- What changes when switch to an ArrayList?
  - Using an iterator:...
  - Direct access:...

- What changes when switch to an binary tree?
  - Using an iterator:...
  - Direct access:...
Iterator Idea

- Iterator Idea:
  - An object which allows iteration over items.
  - If details are hidden..
  - Can have multiple iterators for a collection without them interfering.

```java
int count = 0;
Iterator<String> itr1 = cars.iterator();
while (itr1.hasNext()) {
    String car1 = itr1.next();
    Iterator<String> itr2 = cars.iterator();
    while (itr2.hasNext()) {
        String car2 = itr2.next();
        if (car1.equals(car2)) {
            count++;
        }
    }
}
```
Pattern

- **Software Design Pattern:**
  - Allows discussion, implementation, and reuse of proven software designs

- **Gang of Four**
  - A pioneering book on design patterns by 4 authors: Gamma, Helm, Johnson, Vlissides.
The Iterator Pattern

• Context
  – An aggregate object contains element objects
  – Clients need access to the element objects
  – The aggregate object should not expose its internal structure
  – Multiple clients may want independent access

• Solution
  – Iterator fetches one element at a time
  – Each iterator object..
  – Iterators use a common interface.
Iterator UML

- Client only depends on..
  - It gets a concrete iterator, but knows only its generic type.
- Mapping pattern to CarManager example:

<table>
<thead>
<tr>
<th>Design Pattern</th>
<th>CarManager Ex.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concrete Iterator</td>
<td>Anon. Iterator</td>
</tr>
<tr>
<td>Concrete Aggregate</td>
<td>CarManager</td>
</tr>
<tr>
<td>Aggregate &lt;&lt;I&gt;&gt;</td>
<td>nothing in this example.</td>
</tr>
</tbody>
</table>

isDone()... !hasNext()...
Observer pattern motivation

- Imagine you are writing an automatic day-planner:
  - It reads in the user's interests, plus information about the world, and suggest what they should do.

- Possible design idea:
  - You want to use different objects for cultural planning, sports planning, and sight-seeing.
  - Some objects bring in information about the world; your planning-objects use these info objects.

- Challenge:
  - All of these objects need to know the weather.
  - Your weather object gets updates now and then.
  - How do you tell..
Possible Idea

• Have the weather object call each info. object:

```java
class Weather

    void newDataUpdate() {
        String weatherData = ...;
        culturePlanner.update(weatherData);
        sportsPlanner.update(weatherData);
        sightseeingPlanner.update(weatherData);
        // Change here EVERY time you get a new planner.
    }
```

• Bad because:
  – Weather object is...
  – Every new planner you get, you'll have to change the weather object's code, recompile, and re-run.
The observer pattern

- Observer Pattern:

- Produces a one to many relationship:
  - one object observed (called the subject)
  - many objects observing (called the observers).

- Great because it loosely couples objects:
  - Object with something to report does not need a hard-coded list of who to tell; ...
Observer

- **Button Example**
  - Button knows of a click; TankUI *wants* to know.
  - TankUI creates anonymous ActionListener
    - TankUI registers it with button as a listener for..

- **Benefit:**..
Observer Pattern

• Context
  - An object, called the subject, is source of events
  - One or more observer objects want to be notified when such an event occurs.

• Solution
  - Define an observer interface type.
    All..
  - Subject maintains a collection of observers.
  - Subject supplies methods for attaching and detaching observers.
  - Whenever an event occurs, the subject..
Observer UML

- Subject object knows nothing about class observing it.

Design Pattern | TankUI Ex.
--- | ---
Subject | JButton
attach() | addActionListener()
Observer <<I>> | ActionListener<<I>>
notify() | actionPerformed()
Concrete Observer | Anon. ActionListener
Model View Controller Pattern
and
Facade Pattern
Terminology

• Model:
  - Not like a "model airplane": it's the brains of your system.

• View:
  - Numerous views (parts of UI) may register as observers to a model.
MVC

- Clean design
  Split business logic into..

- Model View Controller Pattern
  MVC splits off 3 things:
  - Hold data and logic
    - Ex: Histogram
  - Present information to user
    - Ex: HistogramIcon, UI components
  - Handles user interaction.
    - Ex: ActionListeners for buttons.
Facade Pattern

• Separate your model from your UI!
  – What if the model is complicated?
    UI gets to many classes in the model.

• Facade Pattern
  – Introduce a new class to the model to..
Facade Pattern Example: Music Player

Facade Pattern Example: Music Player

Facade Pattern Example: Music Player

Facade Pattern Example: Music Player
Recognizing Patterns
Applying Patterns

- Recognize a pattern by..
  - Iterator: cycle through a collection
  - Observer: register for events
  - Strategy: wrap part of an algorithm into a class

- Helps to remember examples
  - Pattern name a hint, but it's not always applicable.

- Ex: What strategy applies to..
  - Strategy?
  - Observer?
  - Iterator?
Summary

- Design patterns allow reuse of design ideas.
- Iterator: An object which abstracts iteration through items in a collection.
  - Decoupled: change collection without changing client code.
- Observer: Notify observing objects of a change without being coupled to those objects.
- MVC: Separate the model from the view.
  - Consider Facade Pattern to decouple UI from model complexity.
- Apply patterns based on patterns intention (not name or UML diagram).