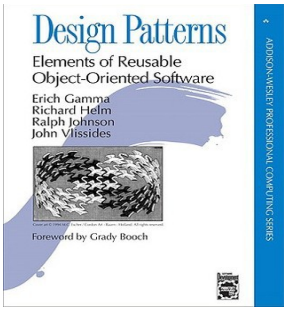
## Software Design Principles and Design Patterns

Object-oriented design patterns describe relationships and interactions between classes or objects, without reference to the applications or objects that are involved.
Design patterns are general description of a strategy to solve a problem that can be used in many different situations. Think of them as the "best practices guide" to a programmer indicating well-known strategies to solve a common problem.

The classic book that introduced Design Patterns
*Design Patterns: Elements of Reusable Object-Oriented Software*
by Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides with a foreword by Grady Booch.
Addison Wesley Publishers 1994
ISBN 0-201-63361-2
The authors are often referred to as the Gang of Four (GoF)




In much the same way that a novice learns a craft from a master craftsman(woman), the design patterns are intended to be words of wisdom from master programmers to other programmers outlining the best strategies used to deal with situations that arise frequently in software programming. While these *patterns* do not tell you anything about how to write code for a specific application or in a specific situation, they outline general strategies to follow in situations that often arise. Think of them as “distilled wisdom” after many years of programming and solving problems.

But remember that these words of wisdom are constrained by the available tools, such as programming languages and programming paradigms that currently exist. And these patterns apply only to Object Oriented programming paradigms, not to functional programming, for example.

### Examples of Design Patterns

In software engineering,  a Design Pattern is a general reusable solution to a commonly occurring problem. Design patterns encourage design reuse. Design patterns allow you to reuse clean, time-tested designs.

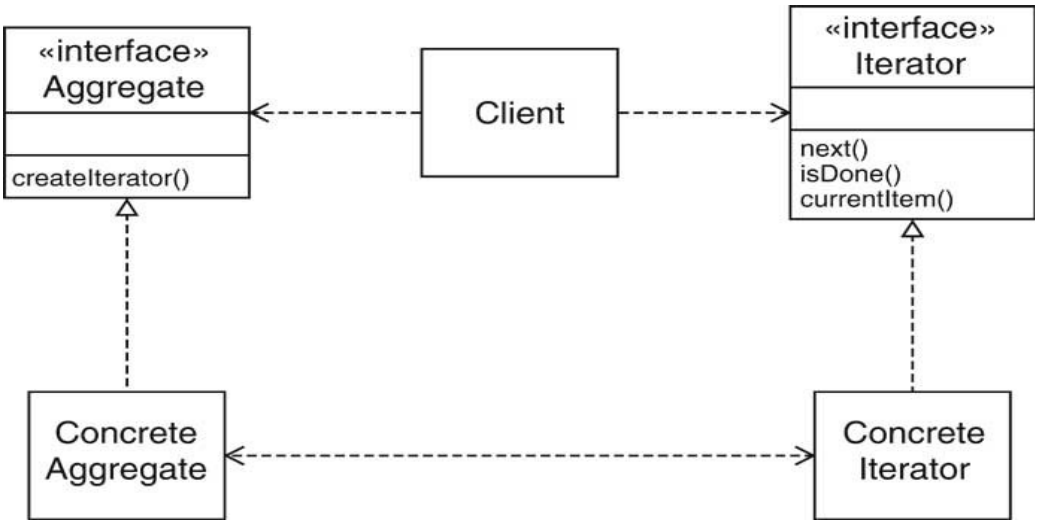
# The Iterator Pattern

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

- Define an iterator that fetches one element at a time
- Each iterator object keeps track of the position of the next element
- If there are several aggregate/iterator variations, it is best if the aggregate and iterator classes realize common interface types.




- Names in pattern are *examples*
- Names differ in each occurrence of pattern

| **Name in Design Pattern** | **Actual Name (linked lists)** |
|---|---|
| `Aggregate` | `List` |
| `ConcreteAggregate` | `LinkedList` |
| `Iterator` | `ListIterator` |
| `ConcreteIterator` | anonymous class implementing `ListIterator` |
| `createIterator()` | `listIterator()` |
| `next()` | `next()` |

| | |
|---|---|
| `isDone()` | opposite of `hasNext()` |
| `currentItem()` | return value of `hasNext()` |

# Observer Pattern

The Observer Pattern is a Software Design Pattern in which an object, called the subject, maintains a list of its dependents, called observers, and notifies them automatically of any state changes, usually by calling one of their methods. It is mainly used to implement distributed event handling systems.

## Context

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

## Solution

1. Define an observer interface type. All concrete observers implement it.
2. The subject maintains a collection of observers.
3. The subject supplies methods for attaching and detaching observers.
4. Whenever an event occurs, the subject notifies all observers.

| **Name in Design Pattern** | **Actual Name (Swing buttons)** |
|---|---|
| `Subject` | `JButton` |
| `Observer` | `ActionListener` |
| `ConcreteObserver` | the class that implements the `ActionListener` interface type |
| `attach()` | `addActionListener()` |
| `notify()` | `actionPerformed()` |