Object-Oriented Design

Lecturer: Raman Ramsin

Lecture 22

GoF Design Patterns – Behavioral
GoF Behavioral Patterns – Class

Class

- **Interpreter:** Given a language, define a representation for its grammar along with an interpreter that uses the representation to interpret sentences in the language.

- **Template Method:** Define the skeleton of an algorithm in an operation, deferring some steps to subclasses; lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure.
GoF Behavioral Patterns – Object

Object

- **Chain of Responsibility**: Avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request. Chain the receiving objects and pass the request along the chain until an object handles it.

- **Command**: Encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations.

- **Iterator**: Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation.

- **Mediator**: Define an object that encapsulates how a set of objects interact; promotes loose coupling by keeping objects from referring to each other explicitly.
GoF Behavioral Patterns – Object (Contd.)

Object (Contd.)

- **Memento**: Without violating encapsulation, capture and externalize an object's internal state so that the object can be restored to this state later.

- **Observer**: Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.

- **State**: Allow an object to alter its behavior when its internal state changes. The object will appear to change its class.

- **Strategy**: Define a family of algorithms, encapsulate each one, and make them interchangeable; lets the algorithm vary independently from clients that use it.

- **Visitor**: Represent an operation to be performed on the elements of an object structure; lets you define a new operation without changing the classes of the elements.
Chain of Responsibility

Intent:
- Avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request. Chain the receiving objects and pass the request along the chain until an object handles it.
Chain of Responsibility: Class Hierarchy

![Class Hierarchy Diagram]

- **Application**
- **Widget**
- **Dialog**
- **Button**

```
if can handle {
    ShowHelp()
} else {
    Handler::HandleHelp()
}
```
Chain of Responsibility: Structure

Diagram showing the structure of the Chain of Responsibility pattern.
Iterator

**Intent:**
- Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation.
Iterator: Structure
Mediator

- **Intent:**
  - Define an object that encapsulates how a set of objects interact: promotes loose coupling by keeping objects from referring to each other explicitly, and lets you vary their interaction independently.
Mediator: Typical Collaboration and Class Hierarchy
Mediator: Structure
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.
Observer: Structure
Observer: Collaboration

![Diagram showing the Observer pattern with state transitions and methods like Notify, Update, SetState, and GetState]
State

- **Intent:**
  - Allow an object to alter its behavior when its internal state changes. The object will appear to change its class.
State: Structure

Diagram:

- **Context**
  - Request()
  - state -> Handle()

- **State**
  - Handle()

- **ConcreteStateA**
  - Handle()

- **ConcreteStateB**
  - Handle()
Strategy

- **Intent:**
  - Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it.
Strategy: Structure
Visitor

**Intent:**

- Represent an operation to be performed on the elements of an object structure; lets you define a new operation without changing the classes of the elements on which it operates.
Visitor: Structure
Visitor: Collaborations
Reference