Design Patterns

Reusable solution to a commonly occurring problem

Slides are based on scripts from Prof. Grüneis

Introduction

- **Design Patterns** are **reusable solutions to a commonly occuring problem**
- Is like a **blueprint** to be used and customized to fit your particular use case
- Are formalized **best practices**
- Are **independent from programming language**
- Are usually described using **UML diagrams**
- **"Gang Of Four (GoF)"**: Eric **Gamma**, Richard **Helm**, Ralph **Johnson**, John **Vlissides**
  - Book: **"Design Patterns - Elements of Reusable Object-Oriented Software"** (1995)
  - Introduced **23 Design Patterns**

Goals

- **Common language** when talking about software development concepts
- Provides **solutions** and **documentation** to solve various typical use cases
- Basic idea when implementing components:
  - **loose couppling**
  - **DRY** principle
  - **reusability**
  - **extendibility**

Groups

- **3 main groups** of Design Patterns:

- **Creational Design Patterns**
- **Structural Design Patterns**
- **Behavioral Design Patterns**

Creational Design Patterns

- Creation of complex objects
- Hide the creation process
- Abstract Factory
- Builder
- Factory Method
- Prototype
- Singleton

Structural Design Patterns

- Composition of classes and objects
- Creation of bigger structures
- Adapter
- Bridge
- Composite
- Decorator
- Facade
- Flyweight
- Proxy

Behavioral Design Patterns

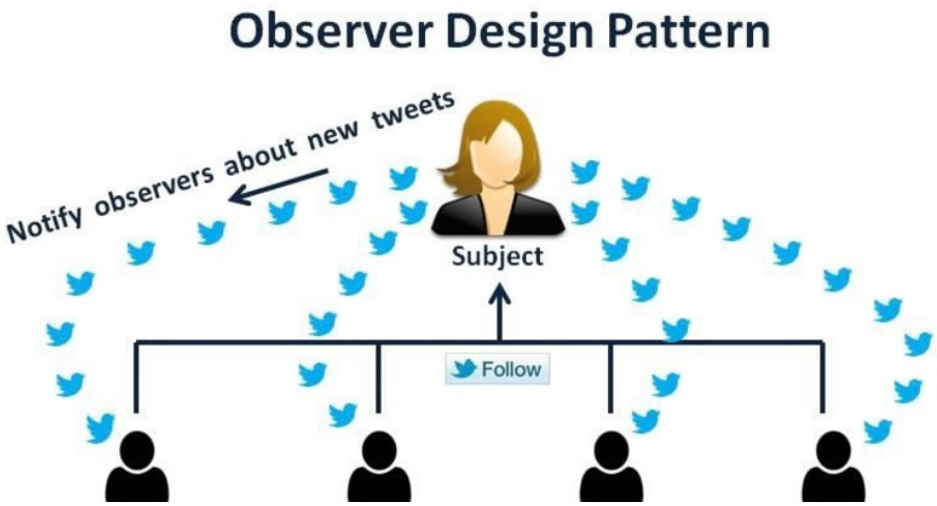
- Definition of responsibility of objects
- Describe interaction between those objects
- Chain of Responsibility
- Command
- Interpreter
- Iterator
- Mediator
- Memento
- Observer
- State
- Strategy
- Template Method
- Visitor

Observer

- One of the **most used** behavioral design patterns
- Basic idea is to enable **communication between objects** using a very **loose coupling**
- Objects **subscribe** (and **unsubscribe**) to **subject** to be **notified** when something happens
- Subject **notifies** all subscribed objects when something happens

---

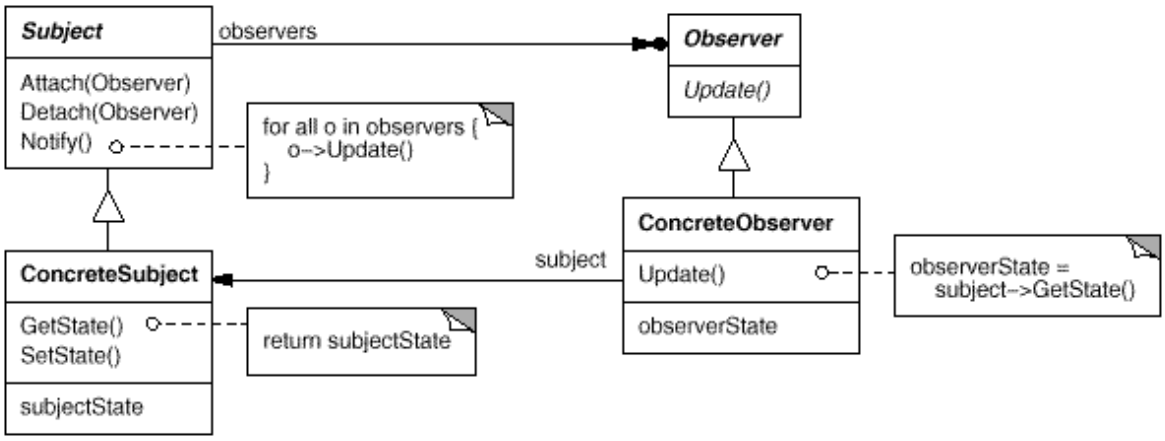
Observer (cont.)



Source: https://dev.to/danlee0528/design-pattern-the-observer-pattern-3oha

---

Observer (UML)



Source: https://www.cs.mcgill.ca/~hv/classes/CS400/01.hchen/doc/observer/observer.html

- `Observer` is usially an **interface** (`IObserver`)
- `ConcreteObserver` registers to the `ConcreteSubject` in its **constructor**
- Usually, there ist **exactly one `ConcreteSubject`**
- There can be **as many `Observer`-Objects** as you wish

- `GetState() / SetState()` often implemented in C# as **properties**
- The `Subject` does not know how many `Observers` exist (even if any exist)

When to use the Observer Design Pattern?

- If you want / have to react to a state change of an object without knowing when that happens
- Changes of states have to be dealt with immediatelly
- Many different objects need to have the same information about an object's state
- An object has to call a method of another object without having an explicit reference to this object

---

Use Cases for the Observer Design Pattern

- Chatting apps of any kind
- Events in C# (object of type `event` is the `Subject` in the UML diagram)
- MVC / MVVM
- Elements in Angular HTML-Templates (are observers for the variables in the TypeScript class)
- RxJs
- SignalR

Pros / Cons of Observer Design Pattern

- **Pros:**
  - Loose coupling between objects; if a concrete observer changes, there are no implications on the subject
  - Subject does not know the observers -> high rate of reusability
  - The basis structure is independent from the concrete problem
- **Cons:**
  - You have to remember to **unsubscribe** from the subject
  - Possible **recursive invocation** of notifications