Software Engineering : Design Patterns 2CB105
Design Patterns for Games 2CB106

02b – Design Pattern Introduction

Dr Andy Guest

a.guest@yorksj.ac.uk

Room 104 44 Lord Mayor’s Walk

Introduction to Design Patterns

- Design patterns represent the best practices used by experienced object oriented software developers.
- Design patterns are solutions to general problems that software developers faced during software development
- These solutions were obtained by trial and error by numerous software developers over quite a substantial period of time.

The Gang of Four

- In 1994, four authors Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides published a book titled **Design Patterns - Elements of Reusable Object-Oriented Software** which initiated the concept of Design Pattern in Software development.
- These authors are collectively known as **Gang of Four (GOF)**. According to these authors design patterns are primarily based on the following principles of object orientated design.
  - Program to an interface not an implementation
  - Favour object composition over inheritance

Usage of Design Pattern

Design Patterns have two main usages in software development.

- Common platform for developers
  - Design patterns provide a standard terminology and are specific to particular scenario. For example, a singleton design pattern signifies use of single object so all developers familiar with single design pattern will make use of single object and they can tell each other that program is following a singleton pattern.
- Best Practices
  - Design patterns have been evolved over a long period of time and they provide best solutions to certain problems faced during software development. Learning these patterns helps unexperienced developers to learn software design in an easy and faster way.

Types of Design Patterns (From the GOF)

- **Creational Patterns**
  These design patterns provide a way to create objects while hiding the creation logic, rather than instantiating objects directly using new operator. This gives program more flexibility in deciding which objects need to be created for a given use case.
- **Structural Patterns**
  These design patterns concern class and object composition. Concept of inheritance is used to compose interfaces and define ways to compose objects to obtain new functionalities.
- **Behavioural Patterns**
  These design patterns are specifically concerned with communication between objects.

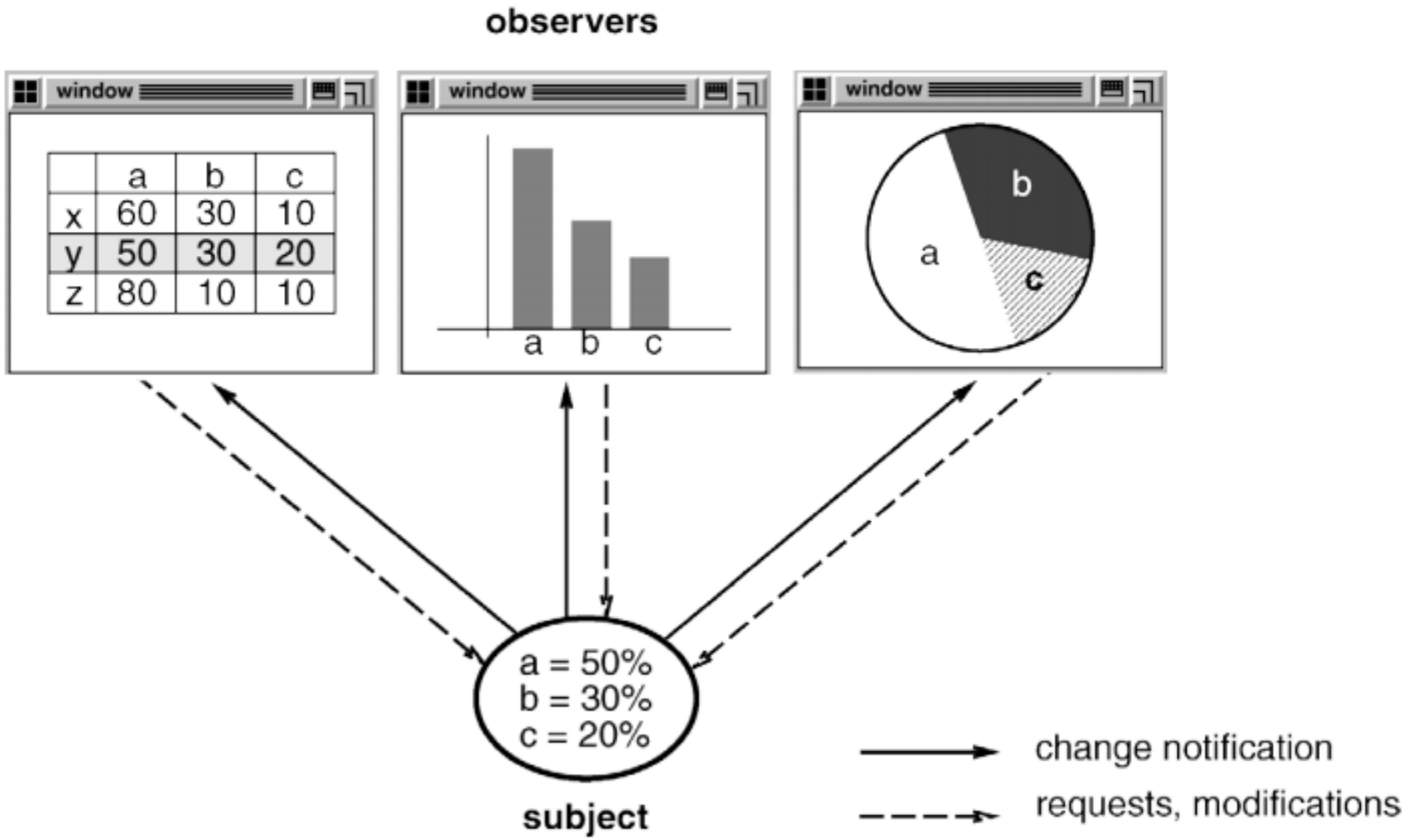
Design Pattern Benefits

- Design patterns help you to solve common design problems through a **proven approach**.
- Design patterns are **well documented** so that there is no ambiguity in the understanding.
- Design pattern may help you reduce the overall **development time** because rather than finding a solution you are applying a well known solution.
- Design patterns promote code reusability and loose coupling within the system. This helps you deal with **future extensions and modifications** with more ease than otherwise.
- Design patterns promote **clear communication** between technical team members due to their well documented nature. Once the team understands what a particular design pattern means, its meaning remains unambiguous to all the team members.
- Design patterns may **reduce errors** in the system since they are proven solutions to common problems.

Design Pattern Structure

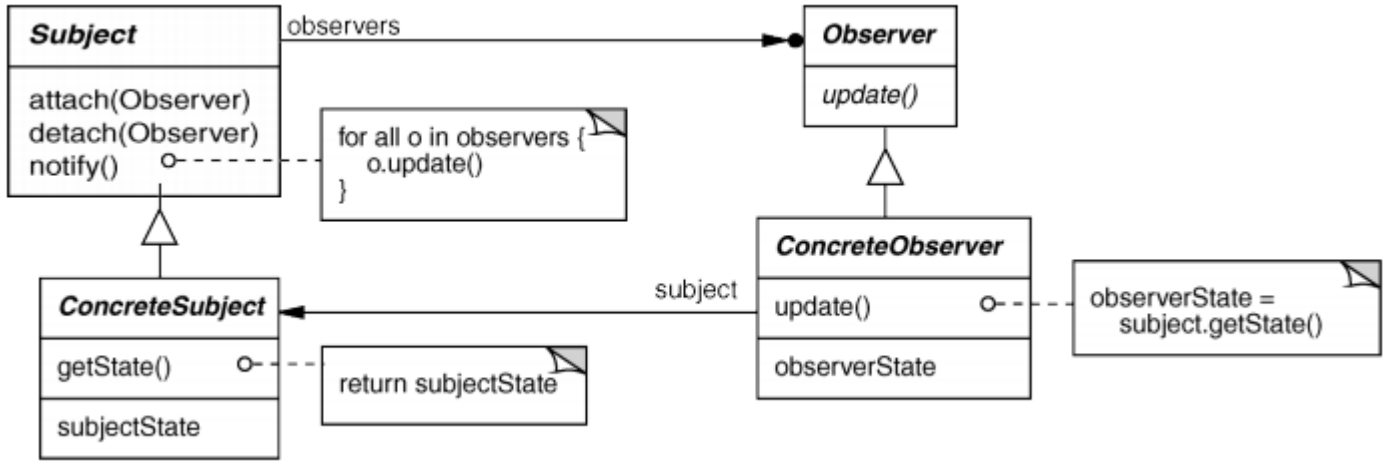
- **Name :** A design pattern usually has a name that expresses its purpose in nutshell. This name is used in the documentation or communication within the development team.
- **Intent :** Intent of a pattern states what that pattern does. Intent is usually a short statement(s) that captures the essence of the pattern being discussed.
- **Problem :** This refers to a software design problem under consideration. A design problem depicts what is to be addressed under a given system environment.
- **Solution :** A solution to the problem mentioned above. This includes the classes, interfaces, behaviours and their relationships involved while solving the problem.
- **Consequences :** Tradeoffs of using a design pattern. As mentioned earlier there can be more than one solution to a given problem. Knowing consequences of each will help you evaluate each solution and pick the right one based on your needs.

Example: Observer



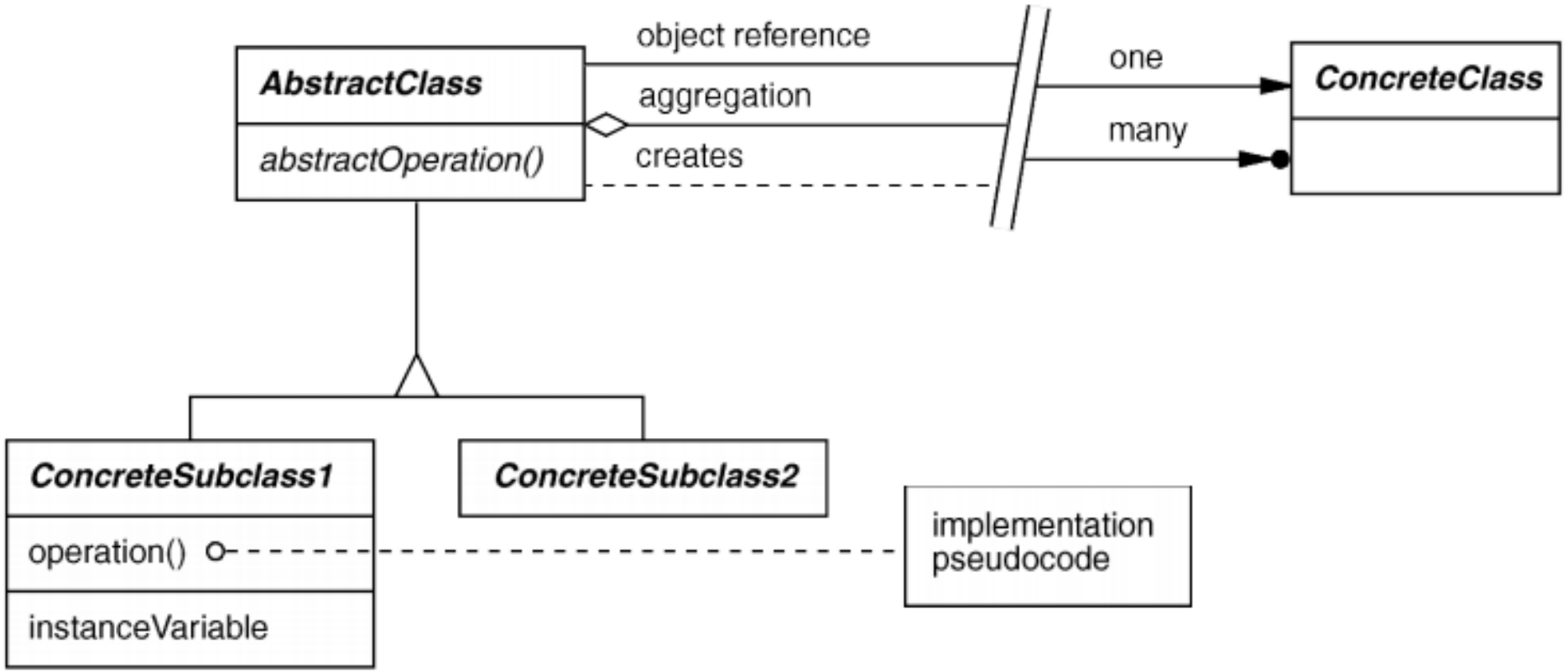
Observer Design Pattern

- **Intent**
  - define a one-to-many dependency between objects so that when one object changes state, all dependents are notified & updated
- **Applicability**
  - an abstraction has two aspects, one dependent on the other
  - a change to one object requires changing untold others
  - an object should notify unknown other objects
- **Structure**



Observer Design Pattern

- **Modified UML/OMT Notation**

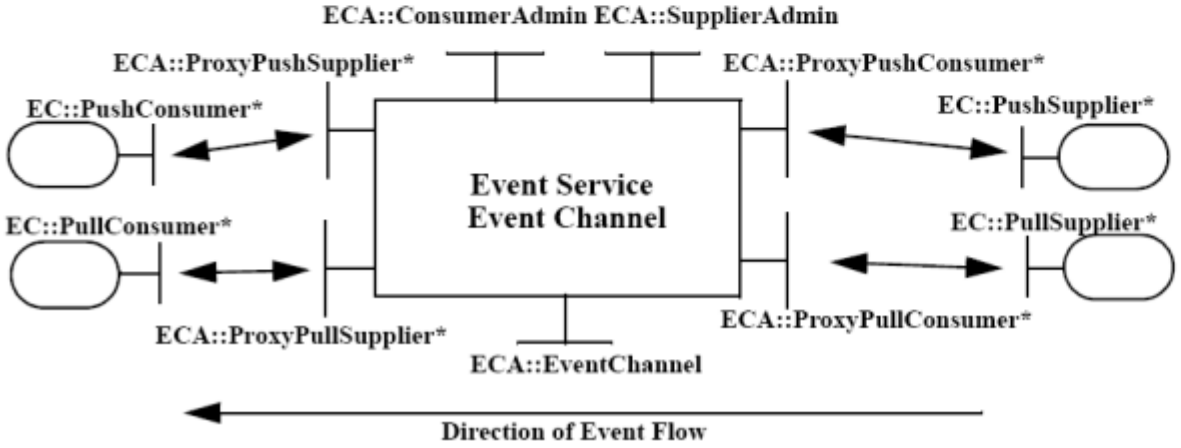


Observer Design Pattern

- **Implementation**

```
class ProxyPushConsumer : public // …
  virtual void push (const CORBA::Any &event) {
    for (std::vector<PushConsumer>::iterator i
          (consumers.begin ()); i != consumers.end (); i++)
      (*i).push (event);
  }

class MyPushConsumer : public // ….
  virtual void push
   (const CORBA::Any &event) { /* consume the event. */ }
```



CORBA Notification Service example using C++ Standard Template Library (STL) iterators (which is an example of the Iterator pattern from GoF)

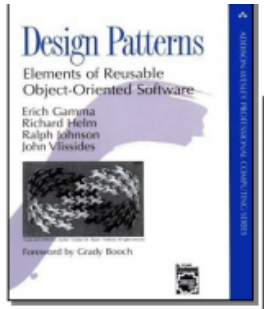
Observer Design Pattern

- Consequences
  - + modularity: subject & observers may vary independently
  - + extensibility: can define & add any number of observers
  - + customizability: different observers offer different views of subject
  - - unexpected updates: observers don’t know about each other
  - - update overhead: might need hints or filtering
- Implementation
  - subject-observer mapping
  - dangling references
  - update protocols: the push & pull models
  - registering modifications of interest explicitly

Observer Design Pattern

- Known Uses
  - Smalltalk Model-ViewController (MVC)
  - InterViews (Subjects & Views, Observer/Observable)
  - Andrew (Data Objects & Views)
  - Symbian event framework
  - Pub/sub middleware (e.g., CORBA Notification Service, Java Message Service)
  - Mailing lists

Design Space for GOF Patterns



		Purpose		
		Creational	**Structural**	**Behavioral**
Scope	**Class**	Factory Method √	Adapter (class) √	Interpreter √ Template Method √
	Object	Abstract Factory √ Builder √ Prototype √ Singleton √	Adapter (object) Bridge √ Composite √ Decorator Flyweight Facade Proxy √	Chain of Responsibility Command √ Iterator √ Mediator Memento Observer √ State √ Strategy √ Visitor √

Scope: domain over which a pattern applies

Purpose: reflects what a pattern does

UML?