# Software Design Pattern

By Chris Zou

Software Design Pattern is a general reusable solution to a commonly occurring problem.

## Four of Gang

Four people published Design Patterns - Elements of Reusable Object-Oriented Software, which initiated the concept of Design Pattern in Software Development.

## Three Categories

Based on GOF's book, there are 23 design patterns which fall in three kinds:

- **Creation Patterns**
- **Structure Patterns**
- **Behaviour Patterns**

### 1. Creation (deal with creation of objects)

a. Factory

- i. ***Use For***: **creating objects without exposing the instantiation logic**
- ii. ***Use Case***: **creating objects that require a lot of setup/configuration, or to have flexibility in creating many different kinds of similar classes.**
- iii. **e.g. : creating a new user.**

b. Singleton

- i. ***Use For***: **only one instance of a class, only one global point of access to this instance.**
- ii. ***Use Case***: **Only one instance of the class should ever exist in the system.**
- iii. **e.g.: software may have SecurityManager class and there is only one instance of this class, accessed by SecurityManager.getInstance(). To check a permission using the SecurityManager, correct call is: SecurityManager.getInstance().checkPermission(Permission.READ)**

c. Templates

- i. ***Use For***: **An object is used as the basiss for making manu copies; the copies can then be customized as necessary.**
- ii. ***Use When***: **similar objects will appear often in the system, or to avoid multiple creation routines for the same/similar objects.**
- iii. **E.g. The company file tax information with the government every month. Define a template with some fields with in (SIN, name, address).**

d. Object Pool

- i. ***Use For***: **A set of already-initialized objects are kept on-hand, ready to use (instead of allocating and destroying the objects on demand).**
- ii. ***Use When***: **allocation and release of resource is expensive, or done very frequently.**

iii. **E.g: A pool of workers to process requests for student transcripts.**

## 2. Structure (used to design relationship between entities of the system)

### a. Get-and-Set

i. we always use get and set methods (not to expose the variables of a class to the outside **)**

### b. Adapter

i. Adapter converts the interface of one object into another to make classes compatible.

*ii.* *Use Case:* you have two objects or modules which are fundamentally capable of interacting, but have incompatible interfaces.

*iii.* *E.g. :* The Linux software known as WINE. It allows (some) Windows programs to work on a Linux system. A call to a Windows system function (such as “draw window”) is handled by wine AND it converts that request to the equivalent Linux form and send the converted request to the Linux system.

### c. Proxy

i. A proxy is an intermediate object that mediates access to the “real” object. Instead of accessing the target directly, other classes access the proxy only. The proxy can do one or more things, such as check security or log some data.

*ii.* *Use Case:* when access to a target, object should be controlled or monitored.

iii. *E.g.:* you want to log database queries.

## 3. Behaviour(model common communications between different objects in the system)

### a. Iterator

i. An Iterator is used to access all of the elements of a collection (list, queue, stack) without knowing the internal structure is. (next() method in Java)

*ii.* *Use Case:* examining the elements of a collection, if the collection might be of more than one type.

### b. Memento

i. Capture a copy of the internal state and provide a means for restoring the object to that state.

ii. *Use Case:* when it should be possible to save and restore.

*iii.* E.g.: Editing a document. The user can make changes, but if they click on cancel then the object is reverted to the previous state.

### c. Visitor

i. A way of separating an algorithm from an object. It is a way to have a new operation on ode that you cannot modify. (since the class with the algorithm typically “visits” the elements of the structure, it is called a visitor.

ii. *Use Case:* defining a new operation without changing the elements on which it operates.

iii. *E*.g.: You have some third party code which you want to analyse. You can write a visitor that goes to each element and print it to the screen.

d. **Listener**

  i. A listener is a class or method that executes in reponse to a change in the system. It is called a listener because it “listens” for changes in the target object, and when it “hears” such a change, it takes action.
  ii. Listener needs to be registered: tell the system that your listener exists and what it is interested in listening for .
  iii. *Use Case:* when an object needs to be notified of a change in another object.
  iv. E.g.: the UI of programs often has a listener that listens for changes to the data being displayed. When the data is changed, the listener is executed, and updates the screen so the change is visible to the user.

For more detailed knowledge about Design Pattern, refer to TutorialsPoint:
http://www.tutorialspoint.com/design_pattern/design_pattern_overview.htm