# Summary what we learn from class today with your own words:

Today we learned 5 patterns from 23 patterns: Singleton, builder, observer, proxy, and factory. Those 23 patterns can be sorted according to 3 types: Creational Pattern, Structural Pattern and Behavioral Pattern.

Creational Pattern hides the creation logic, so we do not new an object directly. Like Singleton, it uses static property, private constructor which means we cannot new an object, and a static method to create an object; builder call static method to return an object of inner class then set property and return the class object. I think factory also belongs to Creational Pattern because If we need a Samsung we do not new a Samsung object directly, instead we new a factory object and call factory’s method to create a Samsung object.

Structural Pattern composite class another class and delegate it. Proxy belongs to structural pattern.

Behavioral Pattern describe the communication among classes or objects. Observer Pattern belongs to

Behavioral Pattern.

1. Singleton Pattern:

Singletion Pattern has only one object, this class create object by itself. Its constructor is private so we can not new an object but we can call it’s static method to create an instance. This class has only one this instance. Usually like hardware printer should use Singleton Pattern.

In version 1, there is eager loading problem. In Sequelize, eager loading means join table but here eager loading means the opposite of lazy loading. Eager loading in Singleton means the instance is created too early.

Version 2 solved the eager loading problem by using lazy loading. Create the instance only the static method being called. But there is still a problem that is Thread-Not-Safe. When I first heard about Thread Safe and not safe was when I was learning HashTable and HashMap. So here Thread-Not-Safe means if multiply threads call the method to create the instance together, multiple objects might be created while Singleton Pattern is only one object in the class.

Version 3 solved the Thread-Not-Safe problem by using ‘synchronized’ key word to lock it which means only one thread can access in one time. New problem comes that it is not efficient because only one thread can access this method in one time.

Version 4 solved the efficiency problem by double check which allow multiple threads access the method in one time, if instance not null, just return the instance; Only if the instance is null, lock here to restrict only one thread can access to create an instance. Still, there is a problem that is instruction reorder. ‘instruction reorder’ is a word relate to the machine language. For here, if it is a single thread, there would be no problem. However, multiple threads make the order not what we expected. Thus, we need to add a volatile key word.

1. Builder Pattern:

When there are too many property in a class and we need many constructors to initial the property or if we want to do constructor: Demo(string firstName, int id){…} and constructor: Demo(string LastName, int id){…}, there would be an error I believe because the rules of the overload. In this situation we can use Builder Pattern to solve this problem: create inner class with set method and return this inner class to support chaining and another method to return outer class’s object. In main we can call outer class’s method to return an inner class object and chaining set properties we need, finally call the method which can return the outer class object. Compare to the default pattern, Builder Pattern solves the too many constructors overload problem and the overload error problem which I just mentioned. However, the disadvantage is obviously that is too much code.

1. Proxy Pattern

Sometimes we have problems to visit some objects or we want to add interface to other system, we need Proxy. Proxy Pattern can add some control or data management like filter. In code example, we use proxy class to do the same thing that the original class can do. Like the method ‘printFullName’ in original class is to print the name. In Proxy class the method ‘printFullName’ with same name in original class can do the same thing but add a data count feature. We can add multiple proxies to the same original class. However, if we can visit the object directly, Proxy Pattern seems complicated because it needs more steps and consume more time. This Proxy Pattern is static proxy because if we create an object from Proxy class, this object can only visit the only one original class object which we passed into the constructor. Dynamic proxy can visit multiply objects.

1. Factory Pattern

We have Samsung class and iphone class. We do not create those class directly. Instead, we just need to tell the Factory that we need a Samsung, the Factory will create a Samsung for us. Thus, we do not have to know about how Samsung created. Meanwhile, if we want to know, it’s also hard to read. (Mock and Stub searched from Google below)

1. Observer Pattern

When there is one to many relationship among objects, we can use Observer Pattern. When the Subject do some change, it will notify the observers. Observer Pattern allows subject to send data to observers by an efficient way. We can add or remove without let the subject know. However, we cannot control the order of observer receiving the notification.

# google all the concept you don't understand:

The single-responsibility principle (SRP) is a computer-programming principle that states that every class in a computer program should have responsibility over a single part of that program's functionality, which it should encapsulate. All of that module, class or function's services should be narrowly aligned with that responsibility.

A Logger object is used to log messages for a specific system or application component. Loggers are normally named, using a hierarchical dot-separated namespace. Logger names can be arbitrary strings, but they should normally be based on the package name or class name of the logged component, such as java.net or javax.swing. In addition it is possible to create "anonymous" Loggers that are not stored in the Logger namespace.

A cache's primary purpose is to increase data retrieval performance by reducing the need to access the underlying slower storage layer. Trading off capacity for speed, a cache typically stores a subset of data transiently, in contrast to databases whose data is usually complete and durable.

A Java virtual machine (JVM) is a virtual machine that enables a computer to run Java programs as well as programs written in other languages that are also compiled to Java bytecode. The JVM is detailed by a specification that formally describes what is required in a JVM implementation. Having a specification ensures interoperability of Java programs across different implementations so that program authors using the Java Development Kit (JDK) need not worry about idiosyncrasies of the underlying hardware platform.

Stub: a dummy piece of code that lets the test run, but you don't care what happens to it.

Mock: a dummy piece of code, that you VERIFY is called correctly as part of the test.

A stub is application facing, and a mock is test facing. It's important to know and distinguish the two since many frameworks for mocking use these terms for different kinds of objects. The biggest distinction is that a stub you've already written with predetermined behavior.