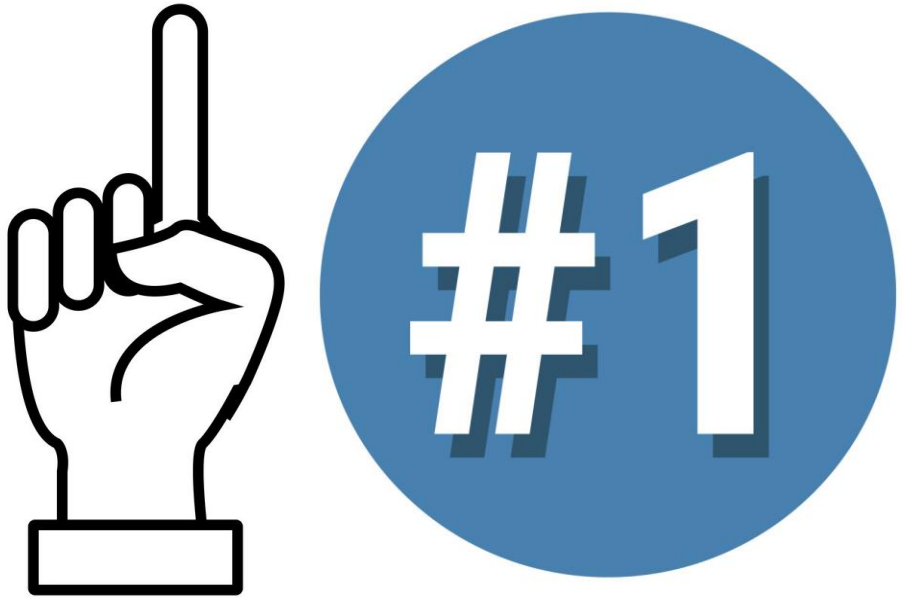
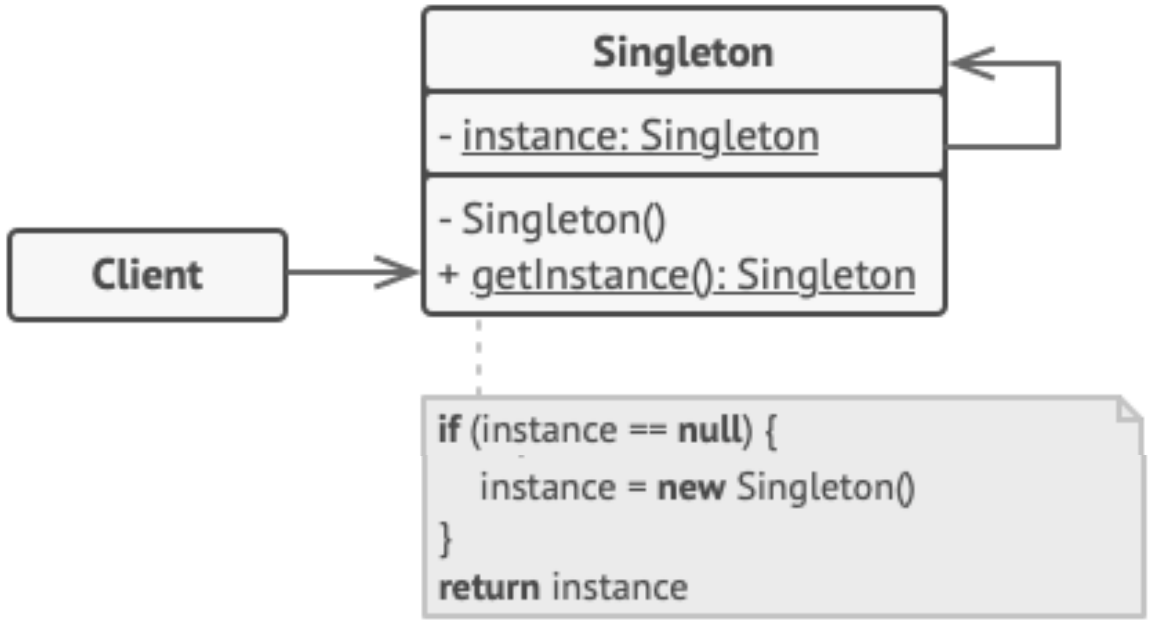
Singleton

Singleton is a creational design pattern that lets you ensure that a class has only one instance, while providing a global access point to this instance.



Singleton


Singleton
- instance: Singleton
- Singleton()
+ getInstance(): Singleton
Client
if (instance == null) {
instance = new Singleton()
}
return instance

Singleton. Eager

```
public class EagerInitializedSingleton {

    private static final EagerInitializedSingleton instance = new EagerInitializedSingleton();

    private EagerInitializedSingleton(){}

    public static EagerInitializedSingleton getInstance() {
        return instance;
    }
}
```

Singleton. Eager

```
public enum EnumSingleton {

    INSTANCE;

    public static void doSomething() {
        System.out.println("I am singleton");
    }
}
```

"Effective Java" J.Bloch

Singleton. Lazy

```
public class Singleton {

    private static Singleton instance;
    private String data;

    private Singleton(String data) {
        this.data = data;
    }

    public static Singleton getInstance(String data) {
        if (instance == null) {
            instance = new Singleton(data);
        }
        return instance;
    }

}
```

Not Thread save!

Singleton. Lazy

```
public class Singleton {

    private static Singleton instance;
    private String data;

    private Singleton(String data) {
        this.data = data;
    }

    public static Singleton getInstance(String data) {


        synchronized (Singleton.class) {

            if (instance == null) {
                instance = new Singleton(data);
            }
        }

        return instance;
    }

}
```

Thread save, but not optimized!

Singleton. Lazy

```
public class Singleton {

    private static volatile Singleton instance;
    private String data;

    private Singleton(String data) {
        this.data = data;
    }

    public static Singleton getInstance(String data) {

        if (instance == null) {
            synchronized (Singleton.class) {

                if (instance == null) {
                    instance = new Singleton(data);
                }
            }
        }
        return instance;
    }

}
```

Why Singleton is BAD?

- Static methods tightly couple our code.
- Makes code less flexible
- Violates SRP (be a singleton, perform business logic)
- Makes testing harder
- Can't track state changes

Spring and Singleton

A bean in the Spring framework is an object created, managed, and destroyed in the Spring IoC Container.

In effect, an object can define its dependencies without creating them and delegate that work to the IoC Container.

Spring and Singleton

- Spring Singleton is single instance per IoC container.
- Spring framework injects a bean in all classes that use it, but retains the flexibility to replace or extend it.
- Framework control the bean's lifecycle.
- Spring beans are easy to mock and the framework can inject them into test classes.

Spring Beans are Eager. Why?

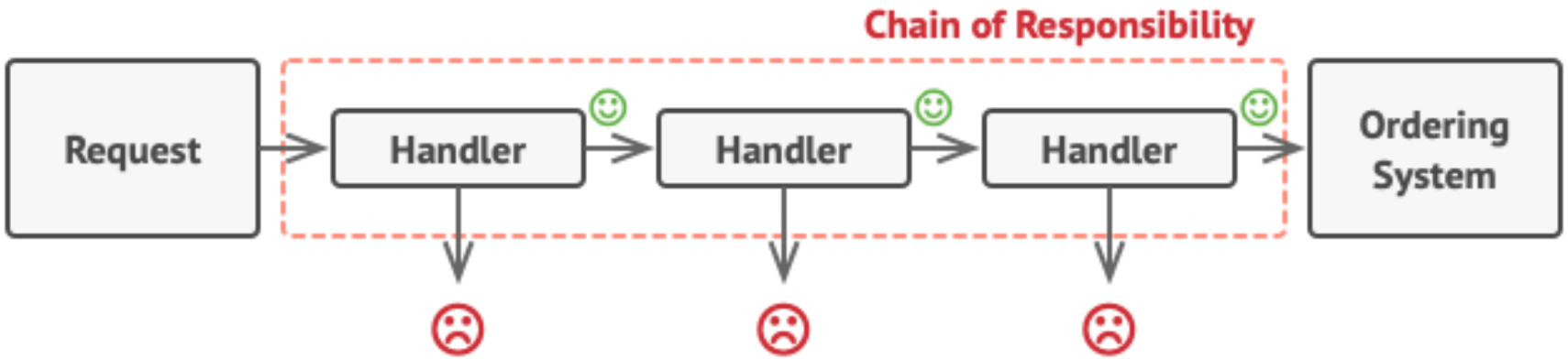
- Bootstrap time vs Runtime
- FailFast

Chain of Responsibility

Chain of Responsibility is a behavioral design pattern that lets you pass requests along a chain of handlers.

Upon receiving a request, each handler decides either to process the request or to pass it to the next handler in the chain.

Chain of Responsibility



Chain of Responsibility