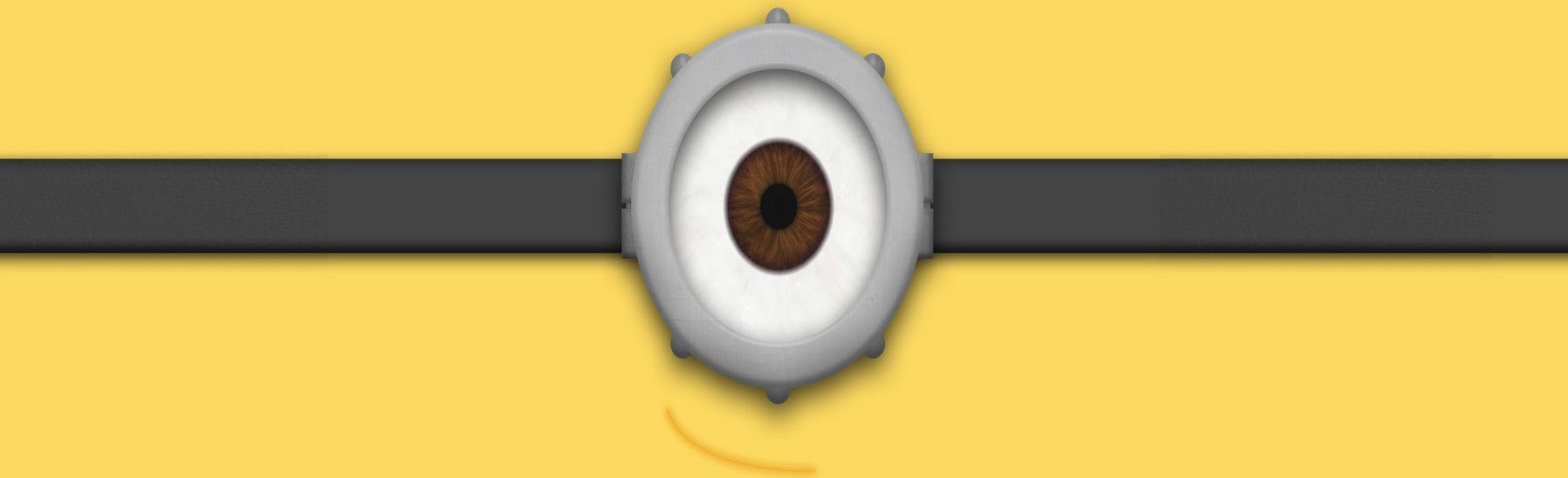


OO Design Patterns

SEASON 2

Xebia

Speaker

Abhishek Sharma

Consultant Java

Xebia

Agenda

SOLID Again !

"Walking on water and developing software from a specification are easy if both are frozen."

- Edward V. Berard

Xebia

Ewww... Bad code

When an application becomes hard to maintain. The signs which indicate a future code rot

Rigidity – small changes causes the entire system to rebuild.

Fragility – changes to one module causes other unrelated modules to misbehave. Imagine a car system in which changing the radio station affects windows.

Immobility – a module's internal components cannot be extracted and reused in new environments. For example, if an application's login module cannot be used in entirely different system then this module is immobile, caused by couplings and dependencies between different modules. The strategy is to decouple central abstractions from low-level details, like a particular database schema or UI implementation (web, desktop) or specific frameworks.

Viscosity – when building and testing are difficult to perform and take a long time to execute. When even a simple change is costly to make, and requires you to make changes in multiple places/levels.

Xebia

Good code :)

Code makes an application work for users. Getting tests passing or removing duplication is useful, but the code needs to be shipped so that users can use it.

What kind of code is good?

Code **needs to work**. Assuming that as a prerequisite…

When code is easy to understand, we can change or extend it faster and are less likely to make a mistake in the process. **Readability** counts.

When code is **easy to change**, we can iterate on user feedback faster.

When code is fun to work with, we’re happier. Building software is our full time job. If we go home every day feeling wrung dry, something is wrong with our process.

Xebia

SOLID software !!

Xebia

SOLID – by the power of Gray Skull

SOLID

Software Development is not a Jenga game

Xebia

SOLID for a better Software

SOLID are five basic principles which help to create good software architecture. SOLID is an acronym where:-

S stands for SRP (Single responsibility principle)

O stands for OCP (Open closed principle)

L stands for LSP (Liskov substitution principle)

I stands for ISP (Interface segregation principle)

D stands for DIP (Dependency inversion principle)

Single responsibility principle

Xebia

Single responsibility principle

The Single Responsibility Principle (SRP) states that there should never be more than one reason for a class to change. This means that every class, or similar structure, in your code should have only one job to do.

Everything in the class should be related to that single purpose, i.e. be cohesive. It does not mean that your classes should only contain one method or property.

Xebia

Cohesion and Coupling

Coupling or dependency is the degree to which each program module relies on each one of the other modules.

Cohesion refers to the degree to which the elements of a module belong together. Thus, it is a measure of how strongly related each piece of functionality expressed by the source code of a software module is.

A system with a higher degree of coupling and lower cohesion implies increased risk.

Xebia

Single responsibility principle - exercise

The **Email** example

```
interface Email{
    void send(MailSender sender, String content);
}
```

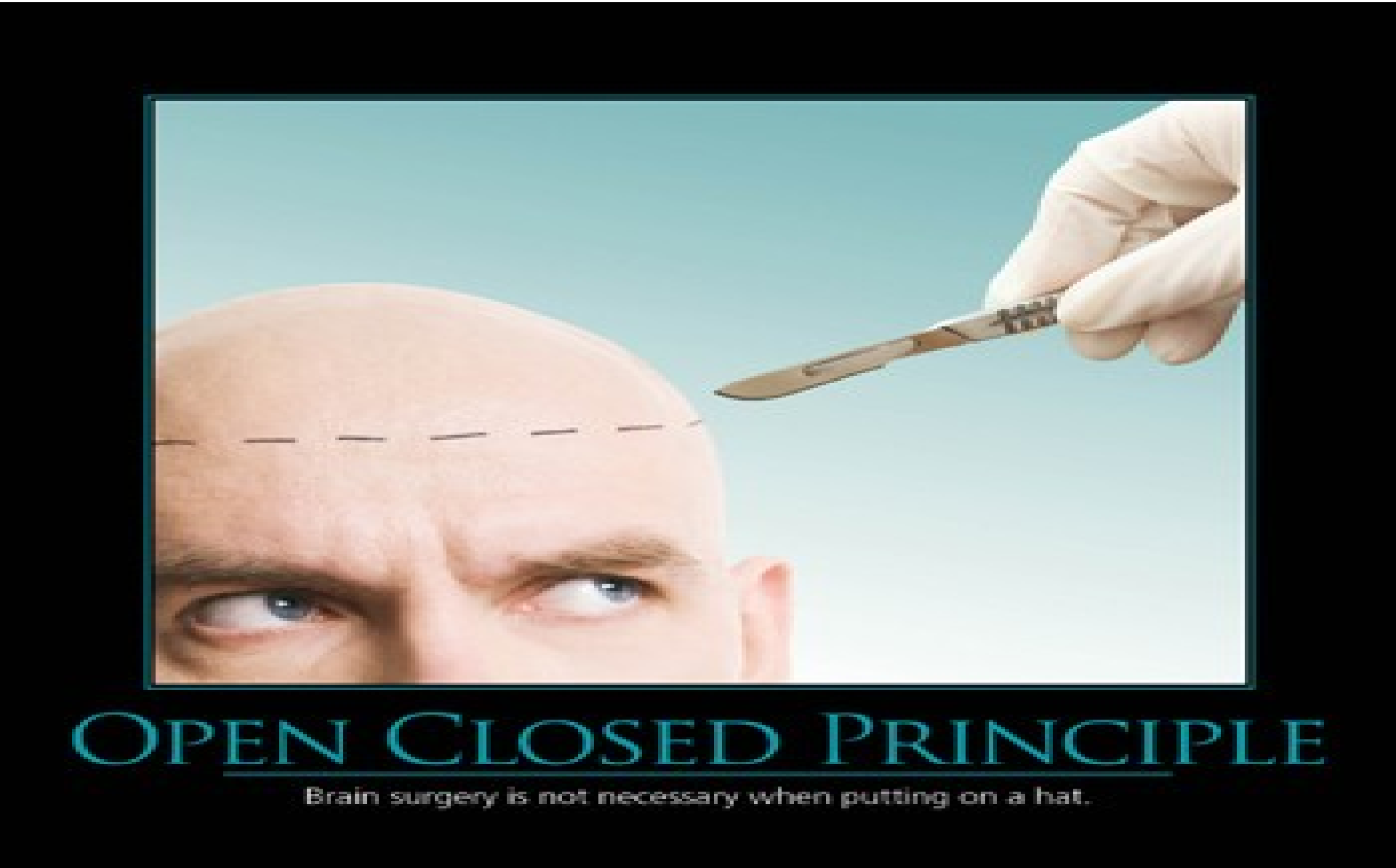
Add these functionality:

- –save as draft in database,
- –print to the nearest printer,
- –display on browser/screen,
- –send HTML,
- –Send XML data,
- –Send attachments

“A violation”

Xebia

Open closed principle



Xebia

Open closed principle

Software entities (classes, modules, functions, etc.) should be

open for extension,

but

closed for modification.

Xebia

Liskov substitution principle

If it looks like an apple but does not taste like one

You have probably the wrong abstraction!

Xebia

Liskov substitution principle - excercise

Design a OO class hierarchy for the Flying Birds

Sample Impl classes:

Parrot

Sparrow

Kingfisher

Ostrich

Xebia

Liskov substitution principle

The Liskov substitution principle is the L in SOLID principle and it states that subtypes must be substitutable for their base types. An LSP can be achieved by :

- Child classes must not remove the behavior of base classes
- Child class must not violate base class invariants

Xebia

Interface segregation principle

Interface Segregation Principle

If IRequireFood, I want to Eat(Food food) not,

LightCandelabra() or LayoutCutlery(CutleryLayout preferredLayout)

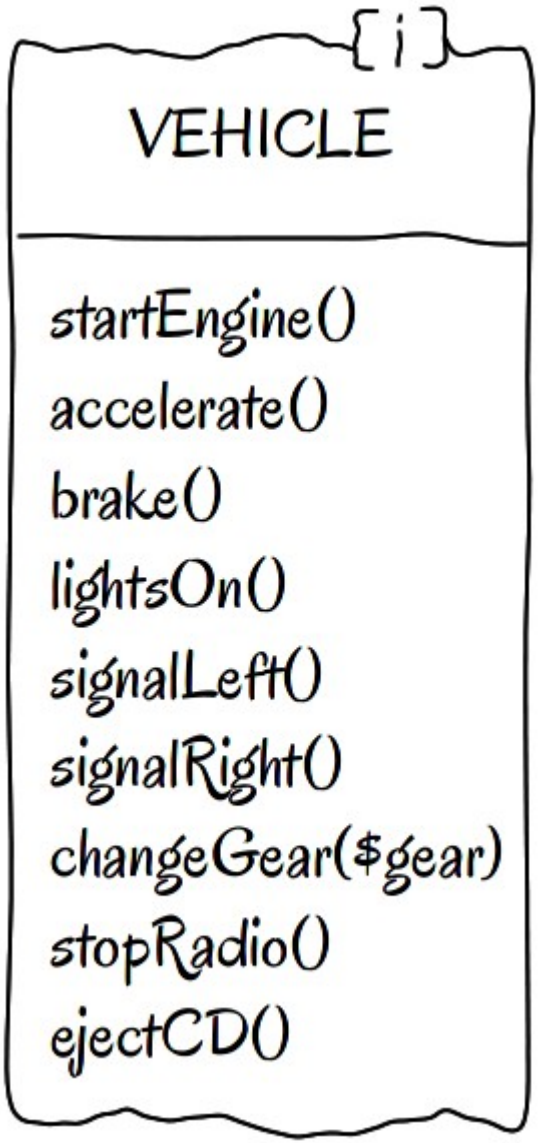
Xebia

Interface segregation principle

The interface-segregation principle (ISP) states that no client should be forced to depend on methods it does not use.

Examples: The complexity of buying a TV

The Flying Bird example revisited



Xebia

Dependency inversion principle

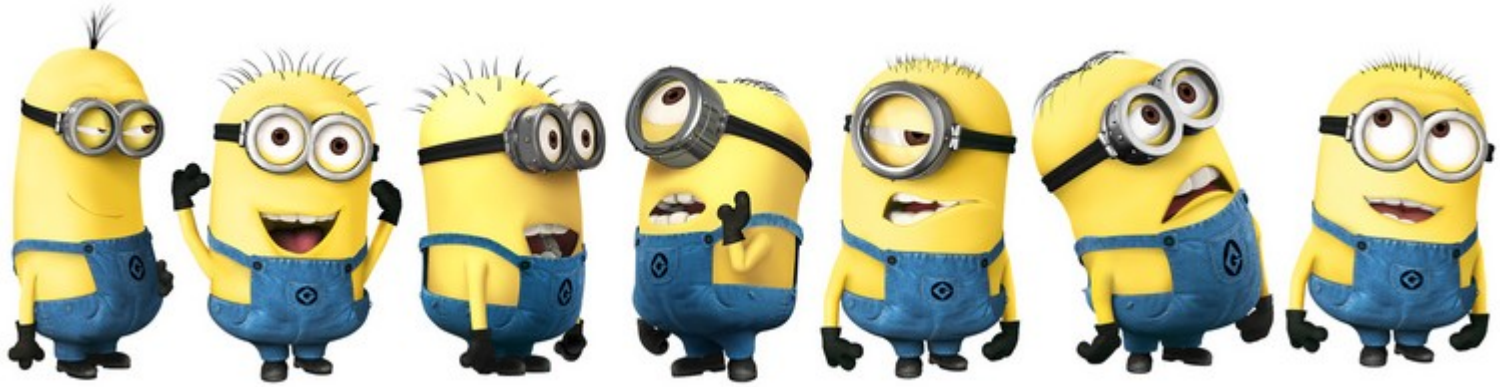
Resolving **dependencies**

and

Hollywood principle

Xebia

Some Design Patterns



Xebia

Command

The command pattern is a behavioral design pattern in which an object is used to represent and encapsulate all the information needed to call a method at a later time. This information includes the method name, the object that owns the method and values for the method parameters.

It allows us to achieve complete decoupling between the sender and the receiver. (A sender is an object that invokes an operation, and a receiver is an object that receives the request to execute a certain operation. With decoupling, the sender has no knowledge of the Receiver 's interface.)

Xebia

Command

The Command pattern has the following advantages:

1. A command decouples the object that invokes the operation from the one that knows how to perform it.
2. A command is a First-class object. It can be manipulated and extended like any other object.
3. Commands can be assembled into a composite command.
4. It's easy to add new commands, because you don't have to change the existing classes.

Xebia

Decorator

Extending an object's functionality can be done statically (at compile time) by using inheritance however it might be necessary to extend an object's functionality dynamically (at runtime) as an object is used.

Decoration is more convenient for adding functionalities to objects instead of entire classes at runtime. With decoration it is also possible to remove the added functionalities dynamically. Decoration adds functionality to objects at runtime which would make debugging system functionality harder.

Xebia

Strategy

Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it.

Xebia

Adapter

In the Adapter Design Pattern, a class converts the interface of one class to be what another class expects.

The adapter does this by taking an instance of the class to be converted (the adaptee) and uses the methods the adaptee has available to create the methods which

The basic premise of the adapter is that you either can not or do not want to change the adaptee. This might be because you purchased the adaptee, and do not have the source code.

Xebia

Proxy

In proxy pattern, a class represents functionality of another class. This type of design pattern comes under structural pattern.

In proxy pattern, we create object having original object to interface its functionality to outer world.

Xebia

Template

If we take a look at the dictionary definition of a template we can see that a template is a preset format, used as a starting point for a particular application so that the format does not have to be recreated each time it is used.

On the same idea is the template method is based. A template method defines an algorithm in a base class using abstract operations that subclasses override to provide concrete behavior.

Xebia

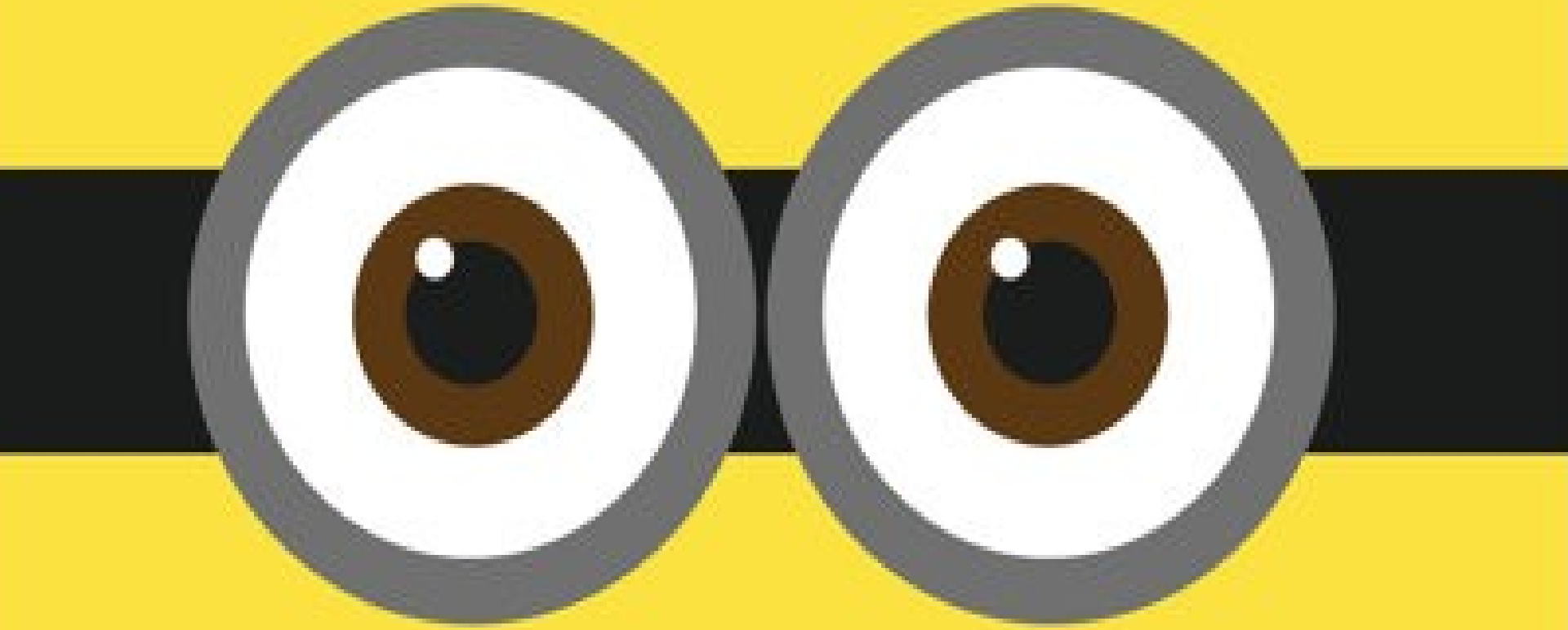
Chain of Responsibility

The Chain of Responsibility design pattern allows an object to send a command without knowing what object will receive and handle it. The request is sent from one object to another making them parts of a chain and each object in this chain can handle the ommand, pass it on or do both.

As the name suggests, the chain of responsibility pattern creates a chain of receiver objects for a request. This pattern decouples sender and receiver of a request based on type of request.

Xebia

Case Studies



THANK YOU

so very much!