DESARROLLO DE SOFTWARE

BERTHIN TORRES

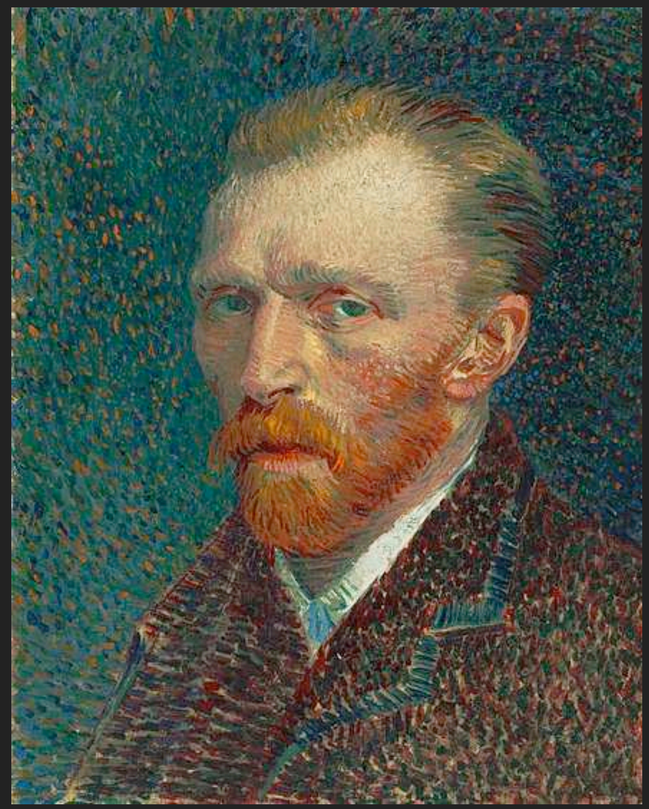
Diciembre 18, 2021

- Qué es desarrollo de software?
- Principios de Software (*software principles*)
- Patrones de diseño (*patterns design*)
- Revisión de código (*code review*)
- DEMO!
- Take aways
  - Soft skills
  - Entrevistas
  - Mastering the keyboard

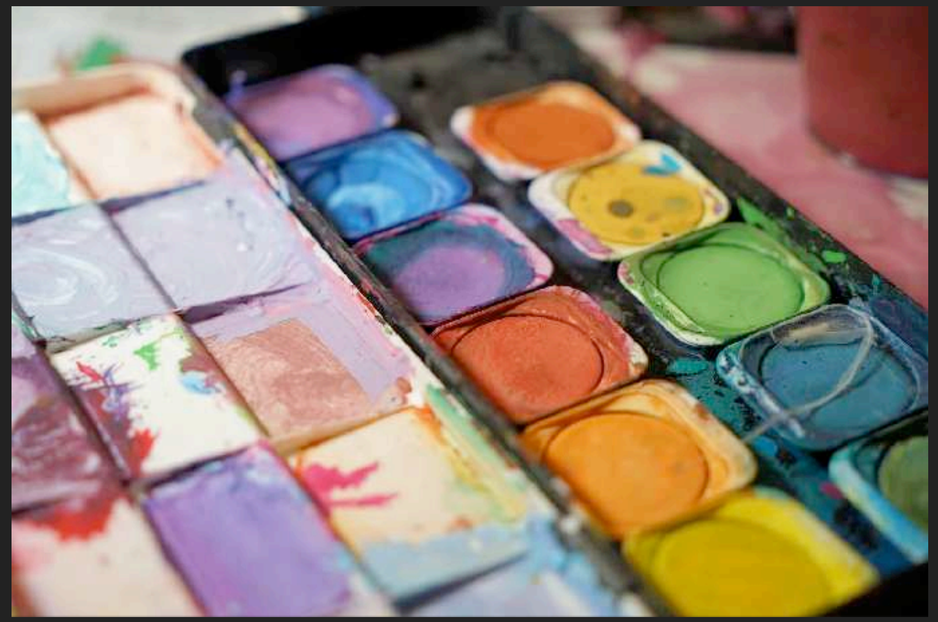
- Qué es desarrollo de software?
- Principios de Software (*software principles*)
- Patrones de diseño (*patterns design*)
- DEMO!
- Revisión de código (*code review*)
- Take aways
  - Soft skills
  - Entrevistas
  - Mastering the keyboard

Da vinci

Rembrandt



Van gogh



QUE ES? V1

- Conjunto de act
  al proceso de cr
  distribución y so
- Actores:
  -
  -
  - E

AGILE MANIFESTO

AGILE MANIFESTO

Estamos descubriendo formas mejores de desarrollar software tanto por nuestra propia experiencia como ayudando a terceros. A través de este trabajo hemos aprendido a valorar:

- Individuos e interacciones sobre procesos y herramientas
- Software funcionando sobre documentación extensiva
- Colaboración con el cliente sobre negociación contractual
- Respuesta ante el cambio sobre seguir un plan

Esto es, aunque valoramos los elementos de la derecha, valoramos más los de la izquierda.

QUE ES? V2

- V1
- Ser responsables para diseñar e implementar el producto. Velar para que el código fuente sea:
  - Entendible
  - Extensible
  - Reusable
  - Mantenible

ROBERT C. MARTIN (A.K.A. UNCLE BOB)

REGLAS PARA EL DESARROLLO DE SOFTWARE

- Seguir convenciones estándares
- Mantener el código simple y sencillo (KISS)
- Dejar el código mejor de lo que lo encontraste
- Buscar siempre la raíz del problema
- ...

General rules

Follow standard conventions

Keep it simple stupid. Simpler is always better Reduce complexity as much as possible

Boy scout rule. Leave the campground cleaner than you found it

Always find root cause Always look for the root cause of a problem

Design rules

Keep configurable data at high levels

Prefer polymorphism to if/else or switch/case

Separate multi-threading code

Prevent over-configurability

Use dependency injection

Follow Law of Demeter A class should know only its direct dependencies

Understandability tips

Be consistent If you do something a certain way, do all similar things in the same way

Use explanatory variables

Encapsulate boundary conditions. Boundary conditions are hard to keep track of Put the processing for them in one place

Prefer dedicated value objects to primitive type

Avoid logical dependency. Don't write methods which works correctly depending on something else in the same class

Avoid negative conditionals

Names rules

Choose descriptive and unambiguous names

Make meaningful distinction

Use pronounceable names

Use searchable names

Replace magic numbers with named constants

Avoid encodings. Don't append prefixes or type information

Functions rules

Small

Do one thing

Use descriptive names

Prefer fewer arguments

Have no side effects

Don't use flag arguments. Split method into several independent methods that can be called from the client without the flag

Comments rules

Always try to explain yourself in code

Don't be redundant

Don't add obvious noise

Don't use closing brace comments

Don't comment out code Just remove

Use as explanation of intent

Use as clarification of code

Use as warning of consequences

Source code structure

Separate concepts vertically

Related code should appear vertically dense

Declare variables close to their usage

Dependent functions should be close

Similar functions should be close

Place functions in the downward direction

Keep lines short

Don't use horizontal alignment

Use white space to associate related things and disassociate weakly related

Don't break indentation

Objects and data structures

Hide internal structure

Prefer data structures

Avoid hybrids structures (half object and half data)

Should be small

Do one thing

Small number of instance variables

Base class should know nothing about their derivatives

Better to have many functions than to pass some code into a function to select a behavior

Prefer non-static methods to static methods

Tests

One assert per test

Readable

Fast

Independent

Repeatable

Code smells

Rigidity. The software is difficult to change. A small change causes a cascade of subsequent changes

Fragility. The software breaks in many places due to a single change

Immobility. You cannot reuse parts of the code in other projects because of involved risks and high effort

Needless. Complexity

Needless. Repetition

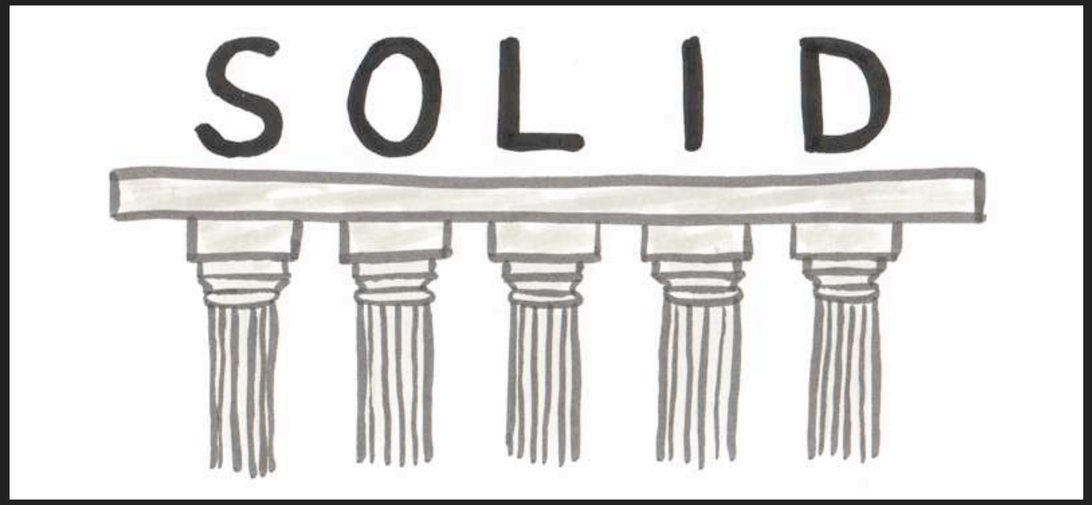
Opacity. The code is hard to understand

'**Clean code**' by Robert C. Martin summary
by Wojtek Lukaszuk

- Qué es desarrollo de software?
- Principios de Software (*software principles*)
- Patrones de diseño (*patterns design*)
- DEMO!
- Revisión de código (*code review*)
- Take aways
  - Soft skills
  - Entrevistas
  - Mastering the keyboard

PRINCIPIOS DE SOFTWARE

- **S**ingle responsibility principle
- **O**pen-close principle
- **L**iskov substitution principle
- **I**nterface segregation principle
- **D**ependency inversion principle

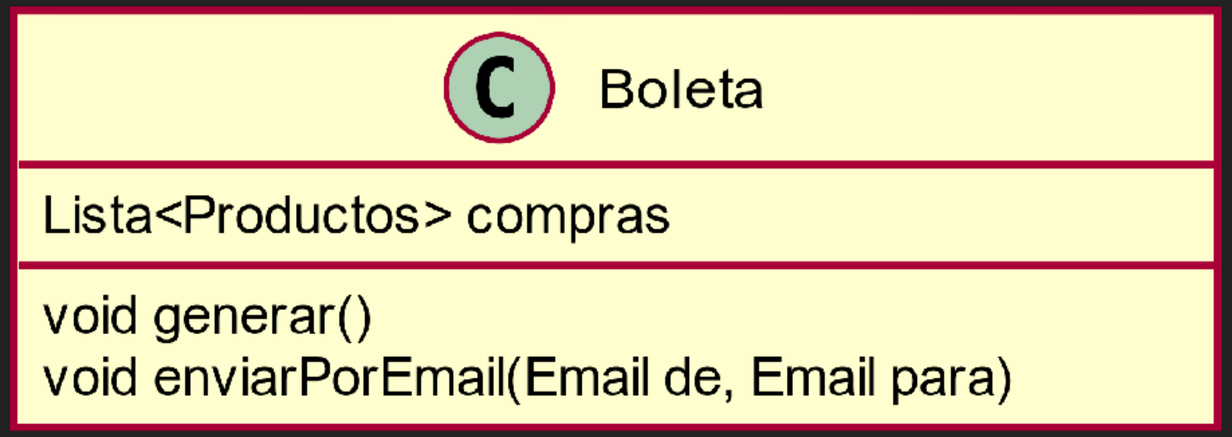


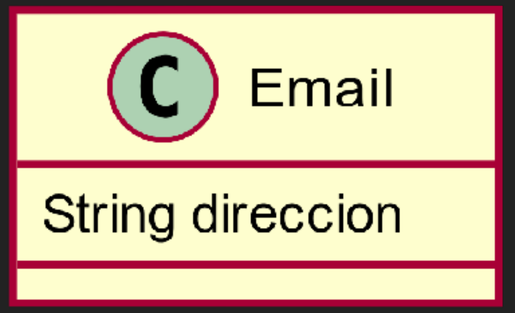
SOLID

Software Development is not a Jenga game

SINGLE RESPONSIBILITY PRINCIPLE

- Una clase/function no debe de tener más que una responsabilidad.

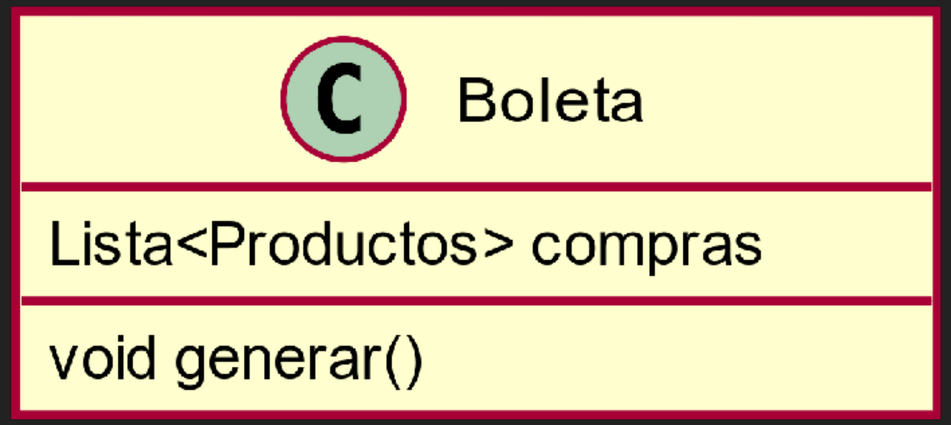


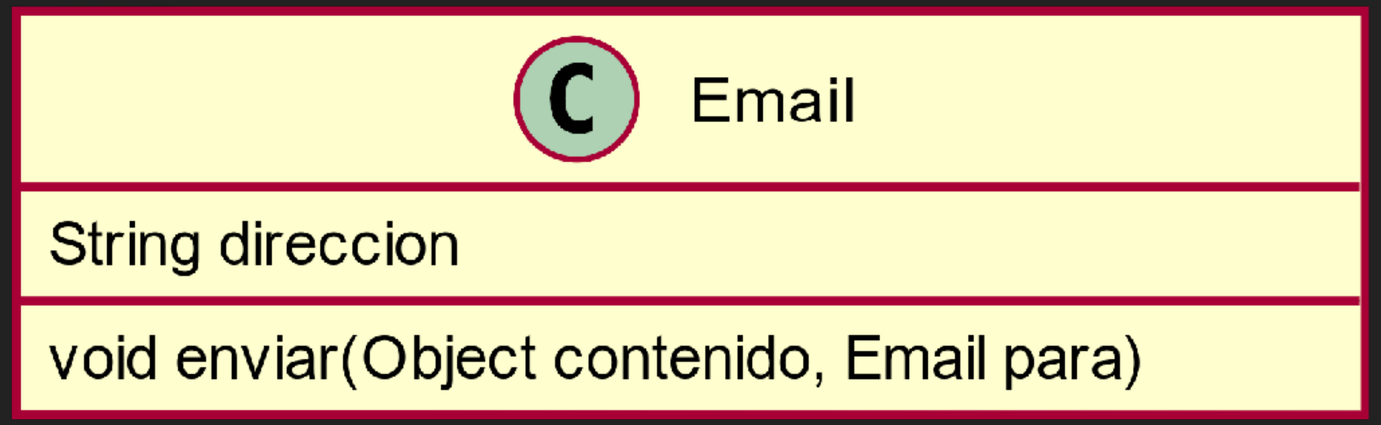



```
boleta.generar()
boleta.enviarPorEmail(de=empresa, para=cliente)
```

SINGLE RESPONSIBILITY PRINCIPLE

- Una clase/function no debe de tener más que una responsabilidad.

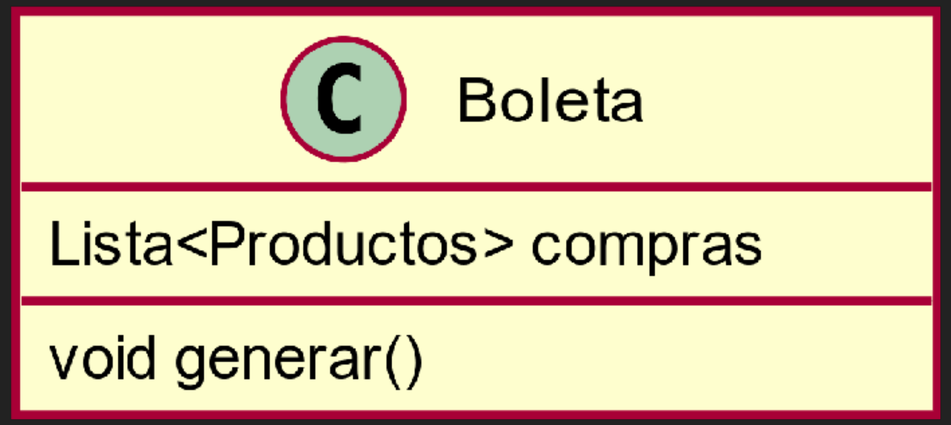


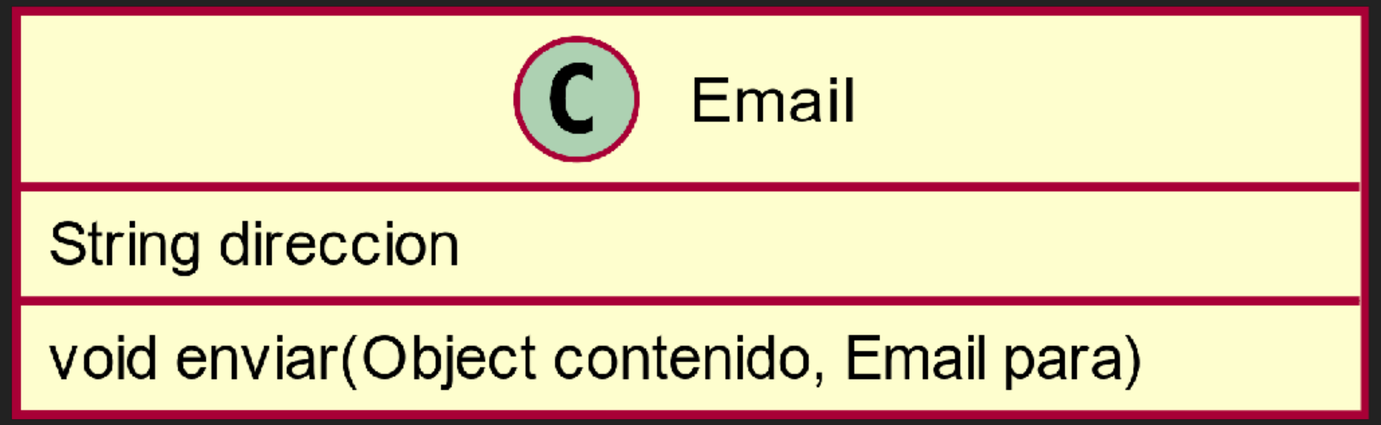



```
boleta.generar()
empresaEmail.enviar(contenido=boleta, para=clienteEmail)
```

SINGLE RESPONSIBILITY PRINCIPLE

- Una clase/function no debe de tener más que una responsabilidad.





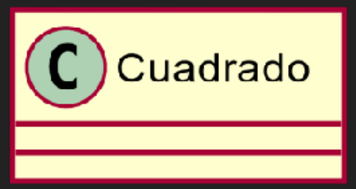

```
boleta.generar()
empresaEmail.enviar(contenido=boleta, para=clienteEmail)
```

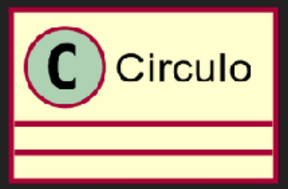
SINGLE RESPONSIBILITY PRINCIPLE

Just Because You Can, Doesn't Mean You Should

OPEN CLOSE PRINCIPLE

- Módulos deben de estar abiertos para extenderse pero cerrados para modificarse

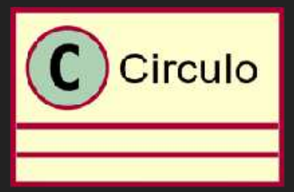





```
void Pantalla::dibujar(Objeto objeto) {
    if (objeto.instanceOf(Circulo)) {
        // dibujar circulo
    } else if (objeto.instanceOf(Cuadrado)) {
      // dibujar cuadrado
    }
}
```

OPEN CLOSE PRINCIPLE

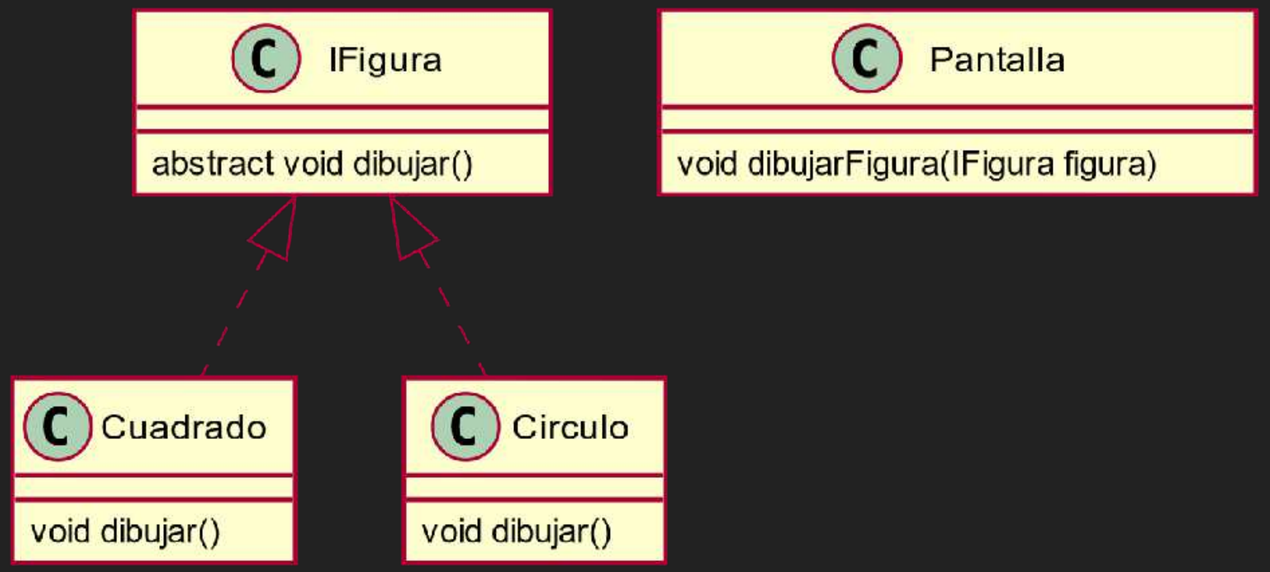
- Módulos deben de estar abiertos para extenderse pero cerrados para modificarse




```
void Pantalla::dibujar(Objeto objeto) {
    if (objeto.instanceOf(Circulo)) {
        // dibujar circulo
    } else if (objeto.instanceOf(Cuadrado)) {
      // dibujar cuadrado
    } else if (objeto.instanceOf(Triangulo)) {
      // dibujar triangulo
    }
}
```

OPEN CLOSE PRINCIPLE

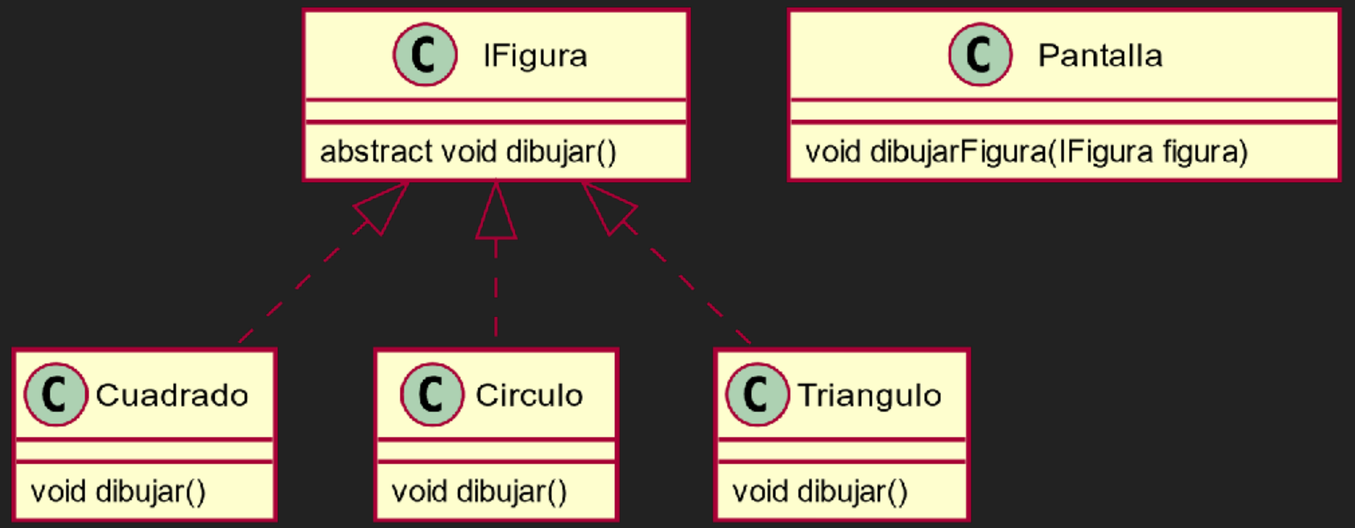
- Módulos deben de estar abiertos para extenderse pero cerrados para modificarse




```
void Pantalla::dibujarFigura(IFigura figura) {
    figura.dibujar()
}
```

OPEN CLOSE PRINCIPLE

- Módulos deben de estar abiertos para extenderse pero cerrados para modificarse



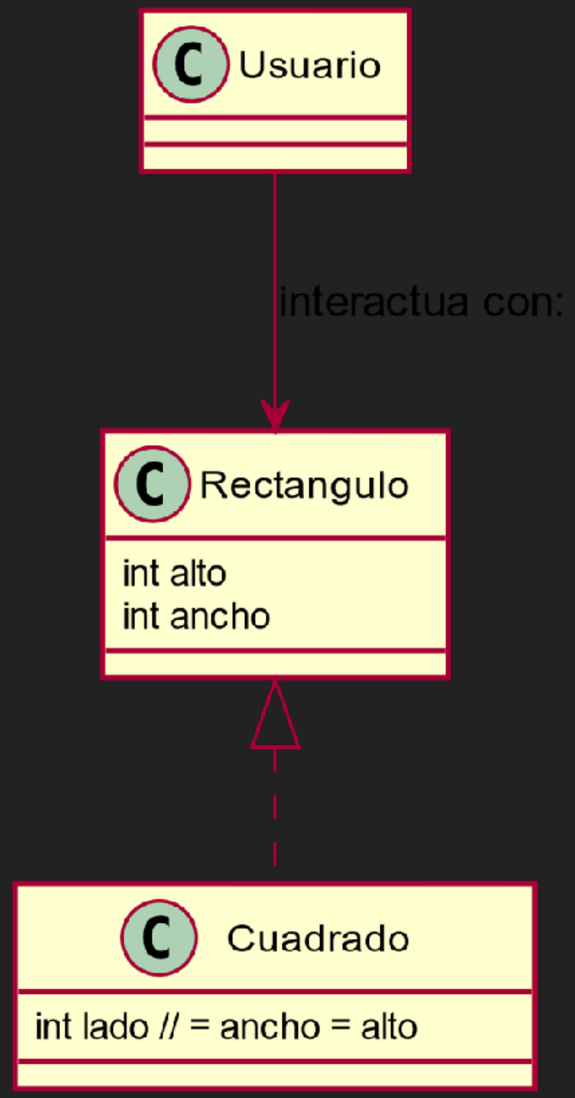

```
void Pantalla::dibujarFigura(IFigura figura) {
    figura.dibujar()
}
```

OPEN CLOSED PRINCIPLE

Open Chest Surgery Is Not Needed When Putting On A Coat

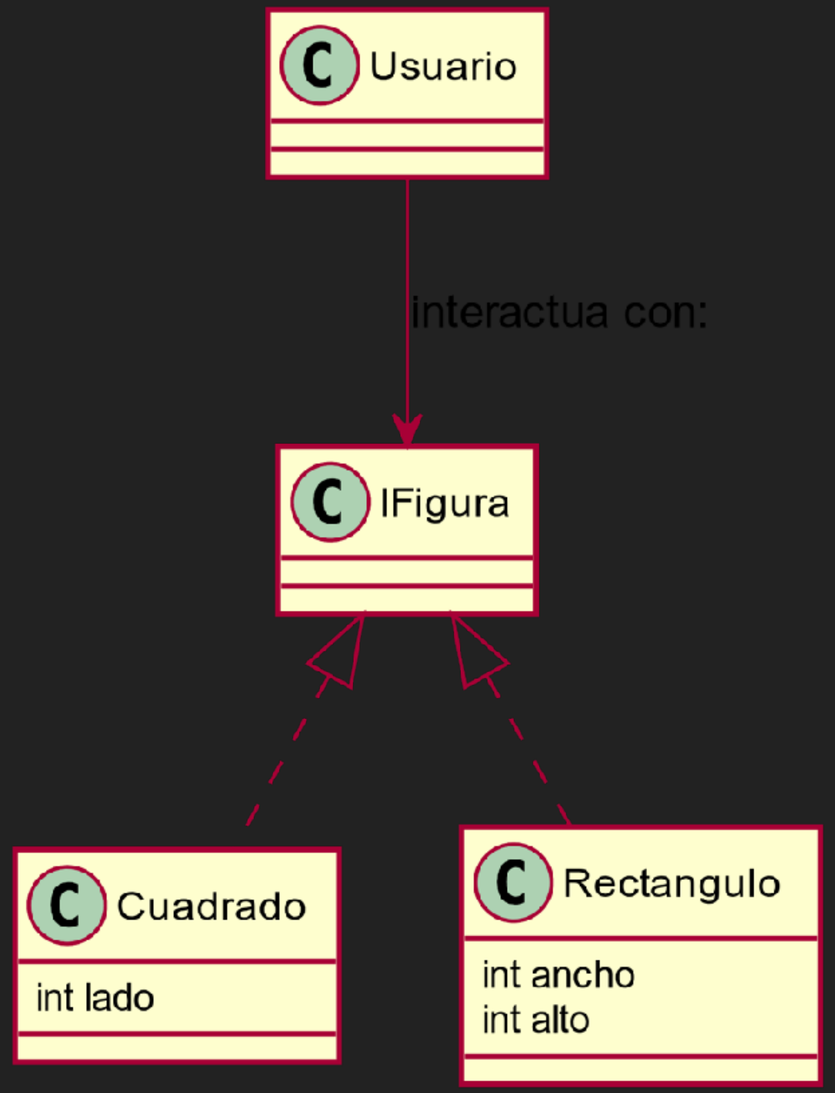
LISKOV SUBSTITUTION PRINCIPLE

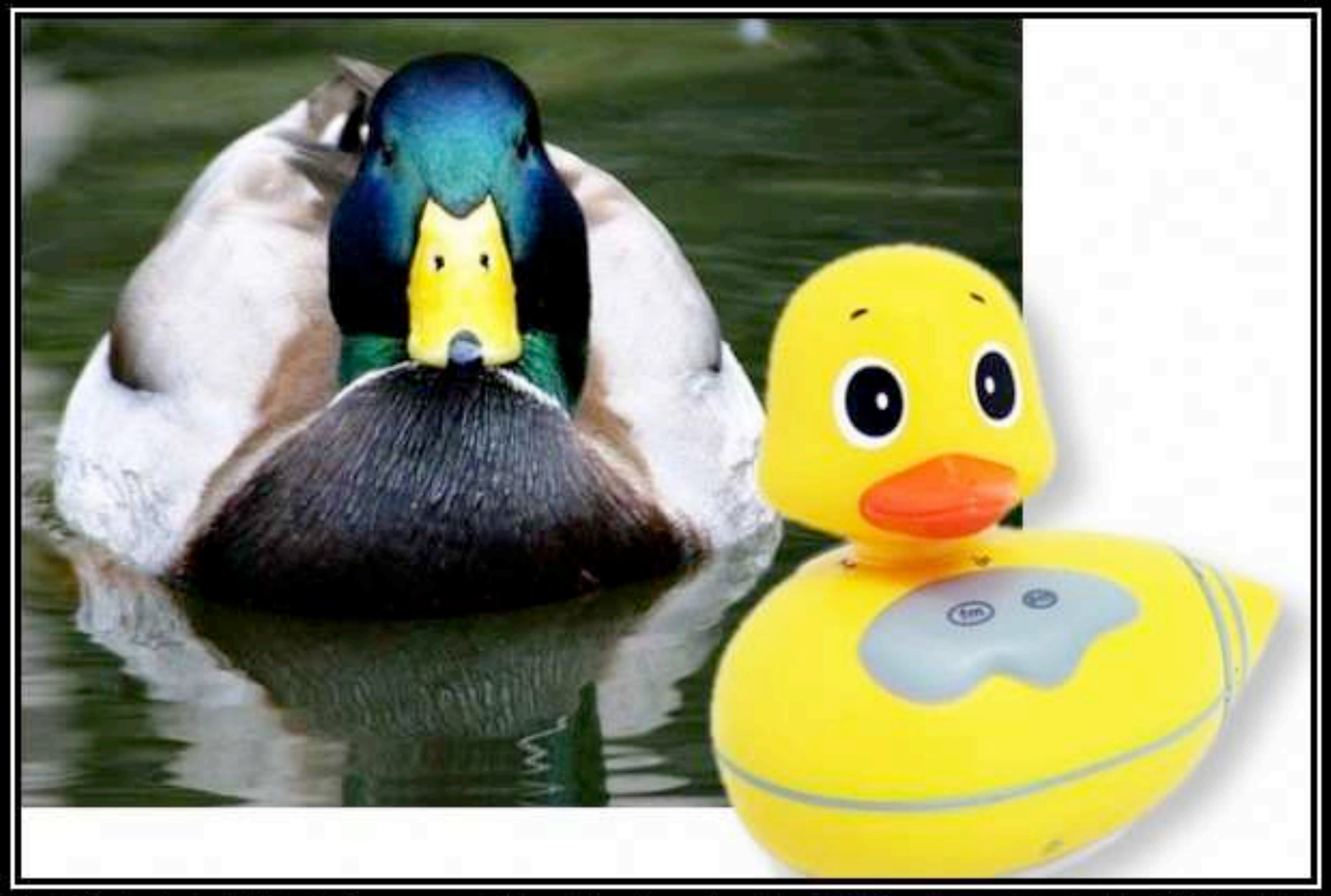
- Una función que utilice referencias a una clase base debe de poder usar objetos de las clases derivadas sin tener conocimiento de ellas



LISKOV SUBSTITUTION PRINCIPLE

- Una función que utilice referencias a una clase base debe de poder usar objetos de las clases derivadas sin tener conocimiento de ellas



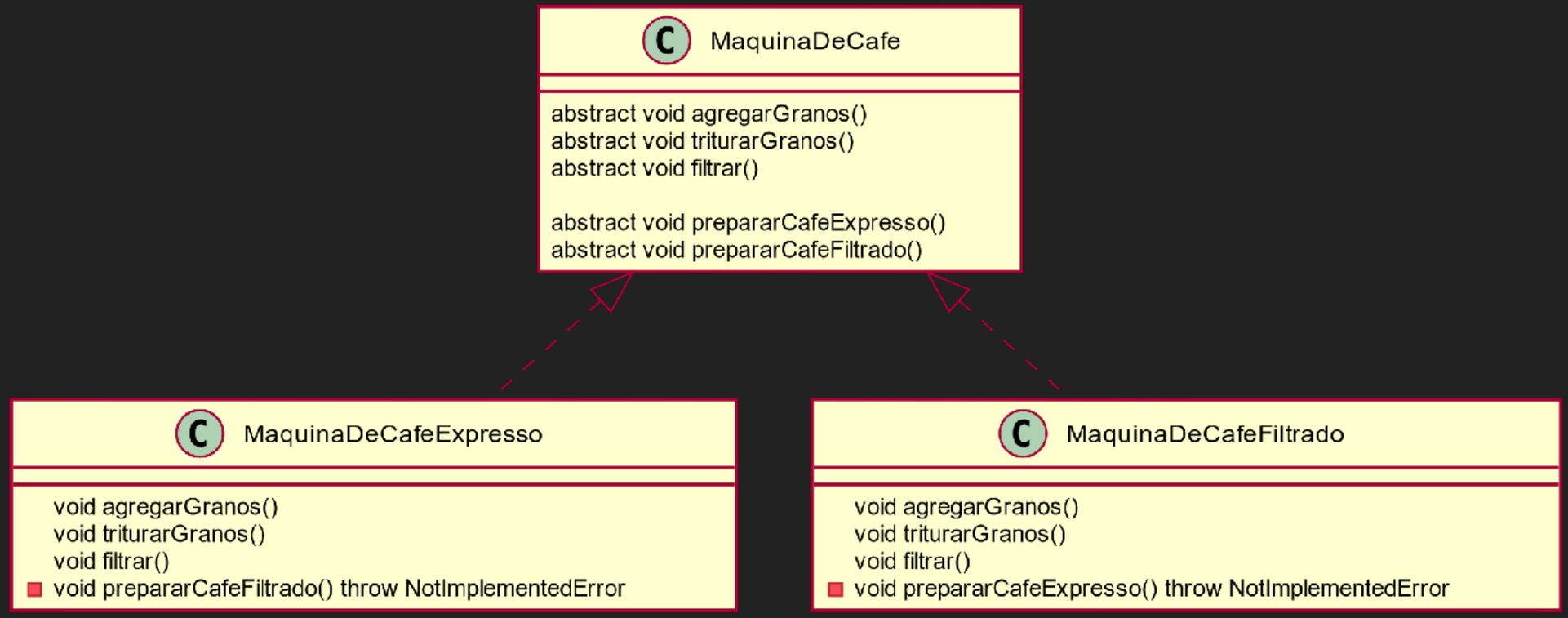


LISKOV SUBSTITUTION PRINCIPLE

If It Looks Like A Duck, Quacks Like A Duck, But Needs Batteries - You Probably Have The Wrong Abstraction

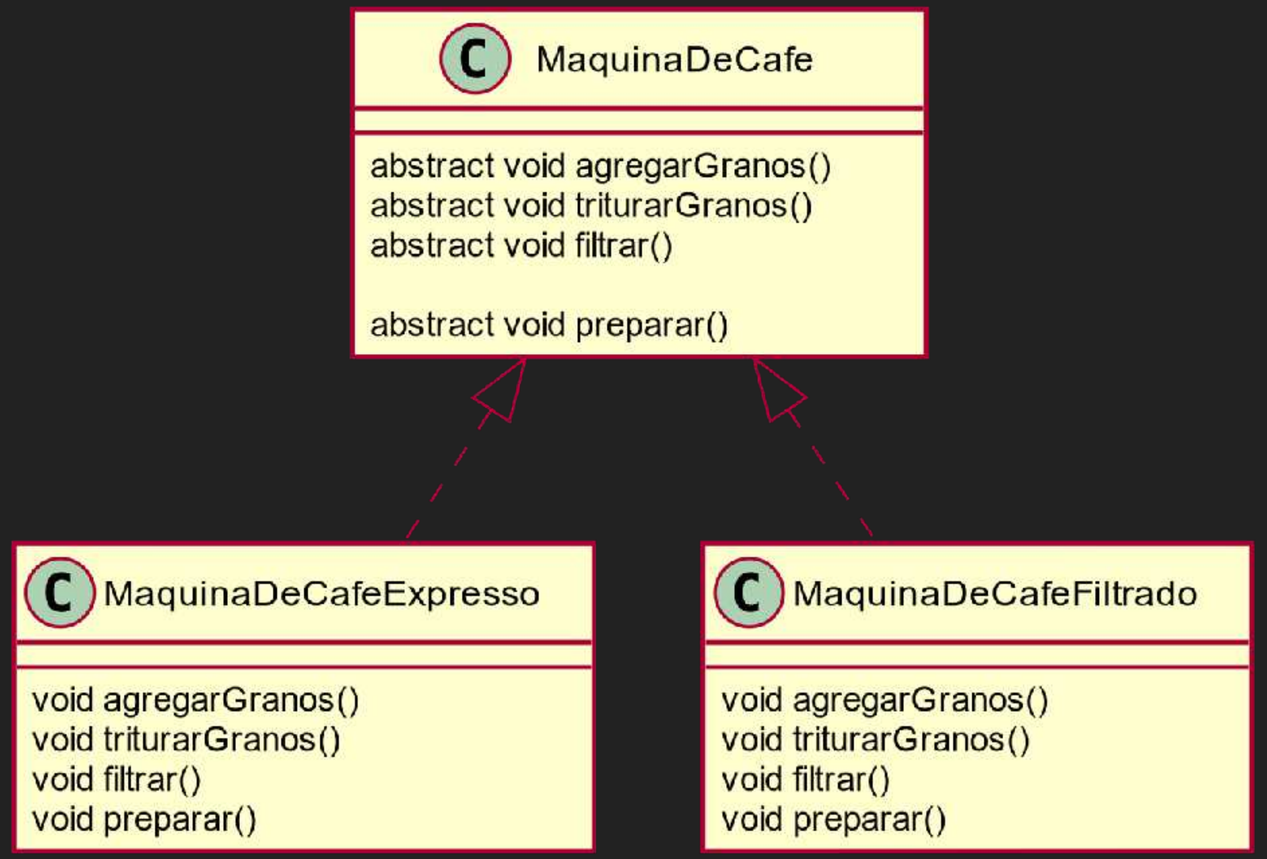
INTERFACE SEGREGATION PRINCIPLE

- Clientes no deben ser forzados a depender de una interfaz que no utilizan



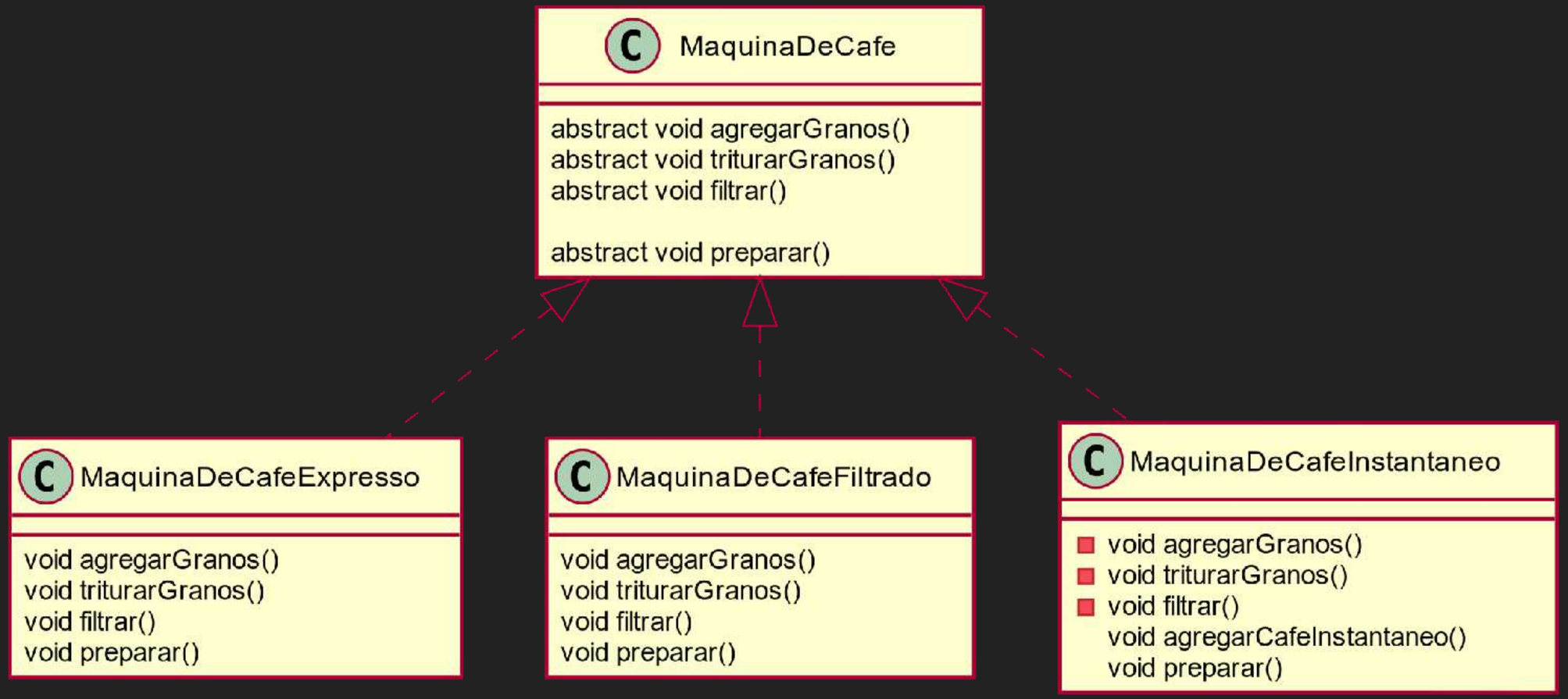
INTERFACE SEGREGATION PRINCIPLE

- Clientes no deben ser forzados a depender de una interfaz que no utilizan



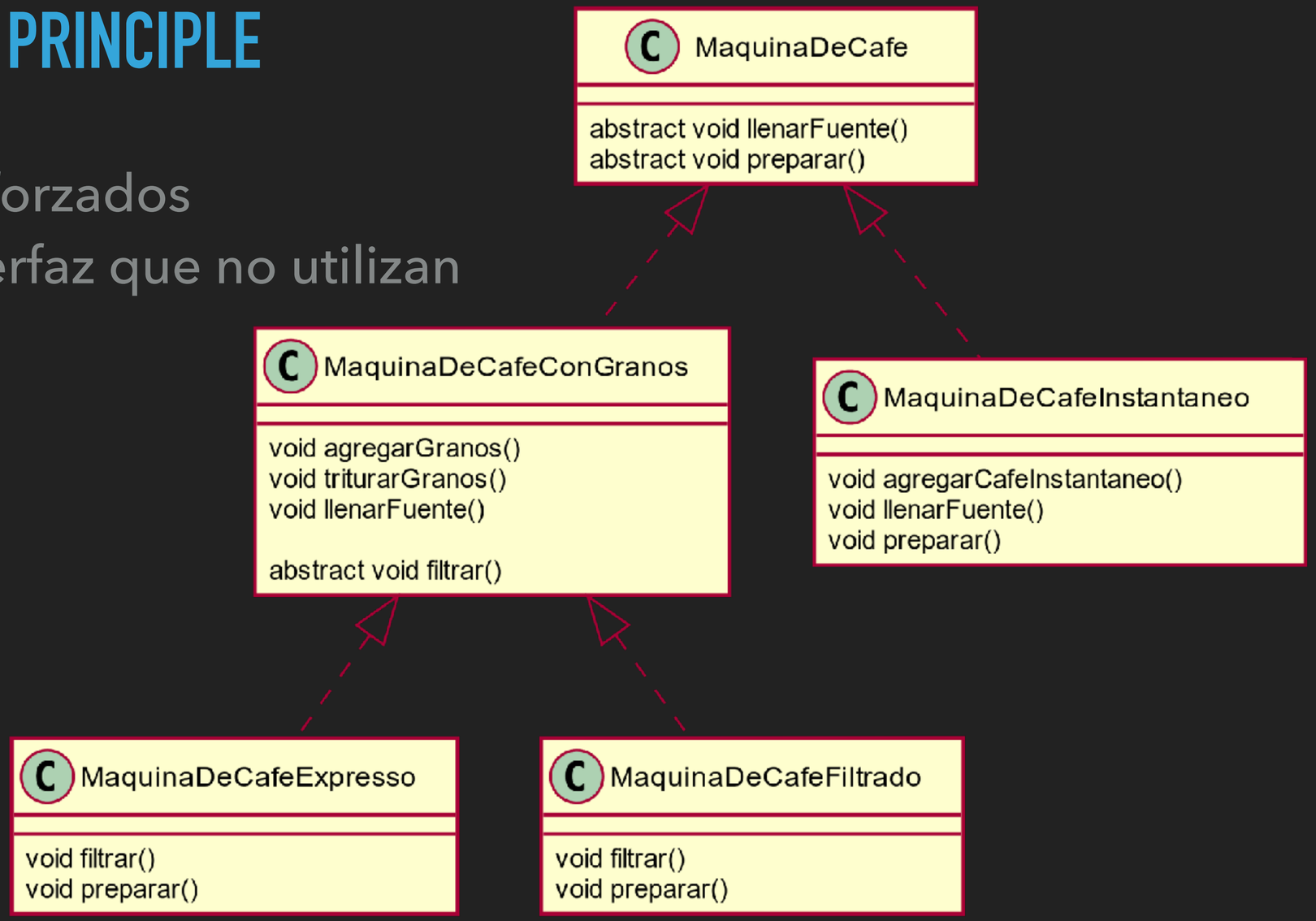
INTERFACE SEGREGATION PRINCIPLE

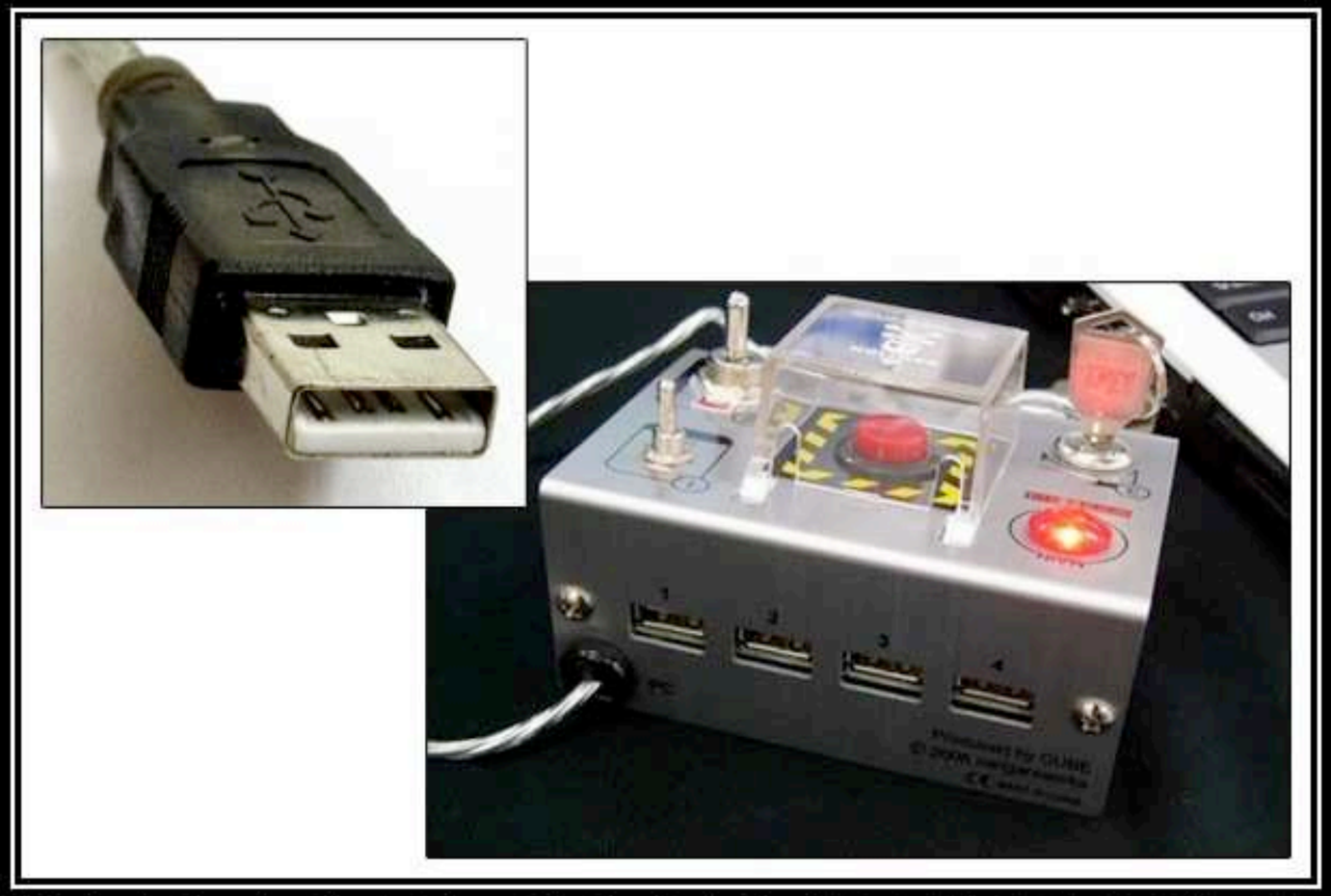
- Clientes no deben ser forzados a depender de una interfaz que no utilizan



INTERFACE SEGREGATION PRINCIPLE

- Clientes no deben ser forzados
  a depender de una interfaz que no utilizan



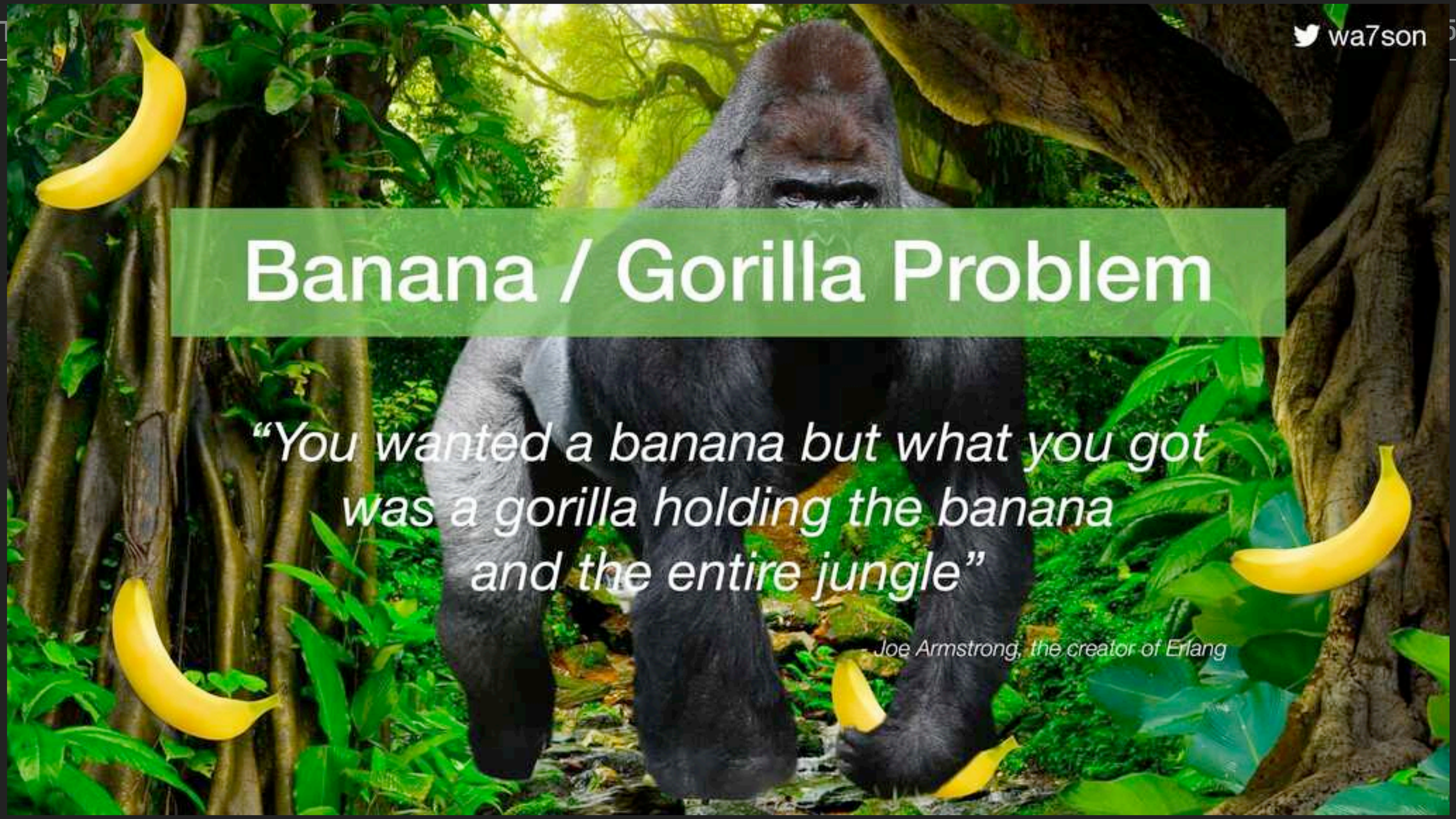


INTERFACE SEGREGATION PRINCIPLE

You Want Me To Plug This In, Where?

DEPENDENCY INVERSION PRINCIPLE

- Módulos de alto nivel no deben de depender en módulos de bajo nivel, cada cual tiene que usar sus abstracciones.
- Abstracciones no dependen de detalles de implementación. Detalles no deben de depender de abstracciones.


wa7son
Banana / Gorilla Problem
“You wanted a banana but what you got
was a gorilla holding the banana
and the entire jungle”
Joe Armstrong, the creator of Erlang

DEPENDENCY INVERSION PRINCIPLE

- Módulos de alto nivel no deben de depender en módulos de bajo nivel, cada cual tiene que usar sus abstracciones.
- Abstracciones no dependen de detalles de implementación. Detalles no deben de depender de abstracciones.

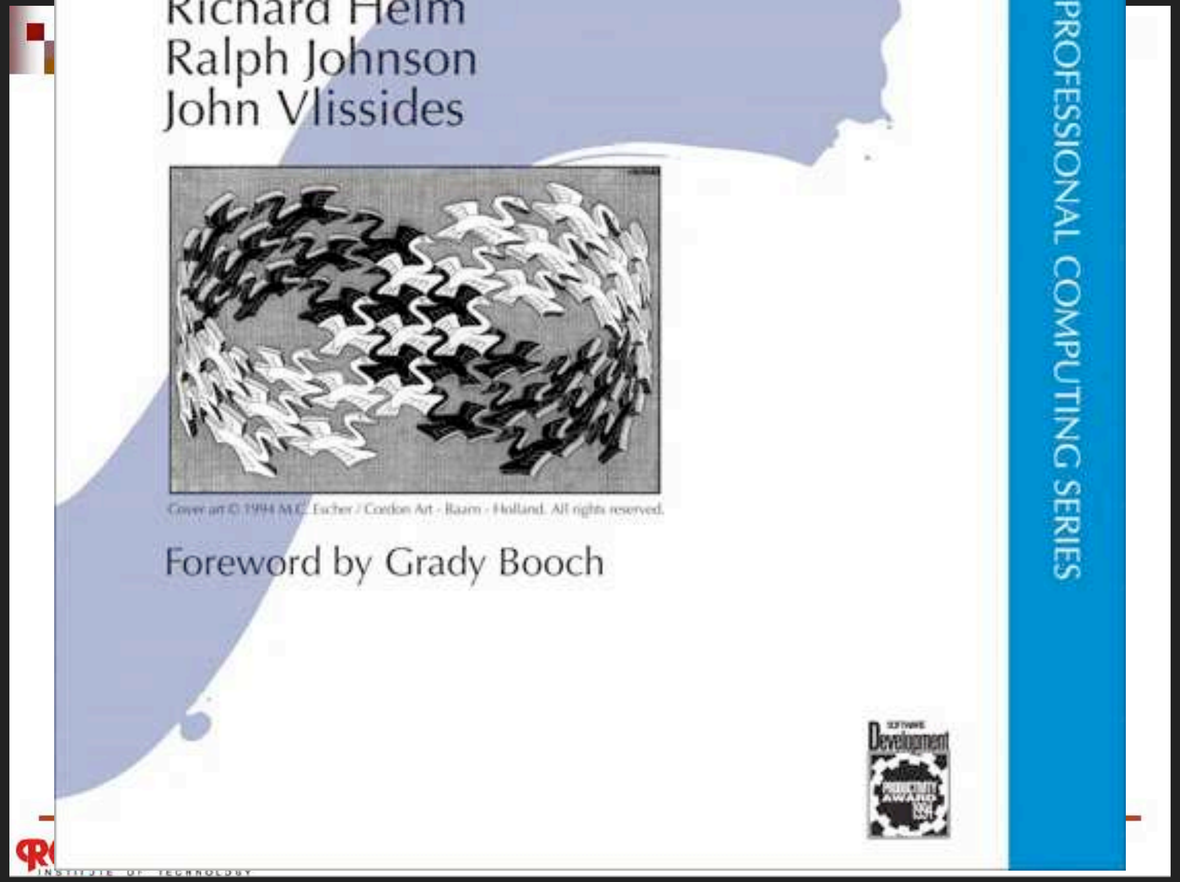
DEPENDENCY INVERSION PRINCIPLE

Would You Solder A Lamp Directly To The Electrical Wiring In A Wall?

- Qué es desarrollo de software?
- Principios de Software (*software principles*)
- Patrones de diseño (*patterns design*)
- DEMO!
- Revisión de código (*code review*)
- Take aways
  - Soft skills
  - Entrevistas
  - Mastering the keyboard

THE GANG OF 4 (GOF)

- Motivación: solucion rentes projectos



ELEMENTS OF

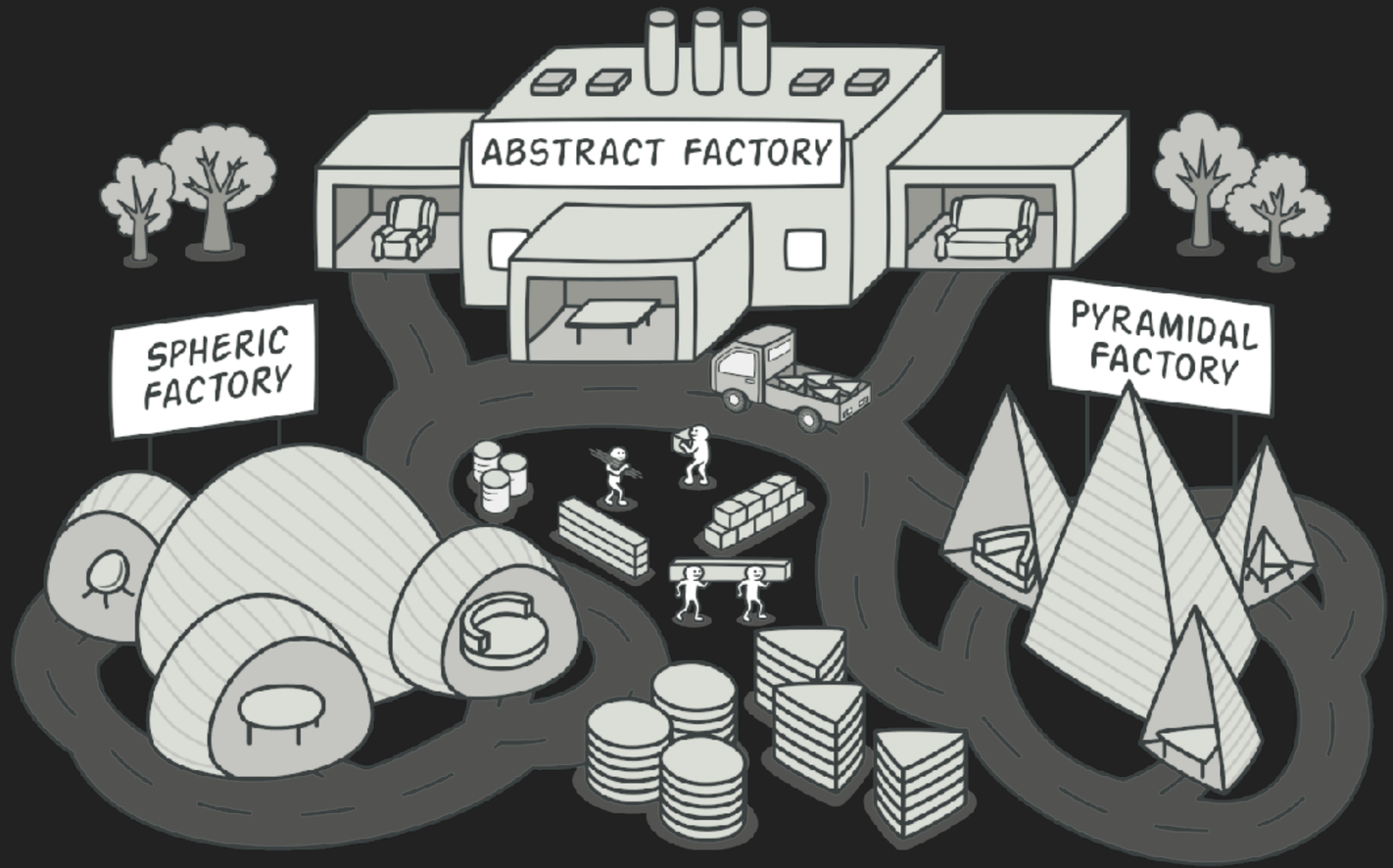
- Colección
en 3 categ
  - Patrones
  - Patrones
  - Patrones

THE 23 GANG OF FOUR DESIGN PATTERNS

C	Abstract Factory	S	Facade	S	Proxy
S	Adapter	C	Factory Method	B	Observer
S	Bridge	S	Flyweight	C	Singleton
C	Builder	B	Interpreter	B	State
B	Chain of Responsibility	B	Iterator	B	Strategy
B	Command	B	Mediator	B	Template Method
S	Composite	B	Memento	B	Visitor
S	Decorator	C	Prototype		

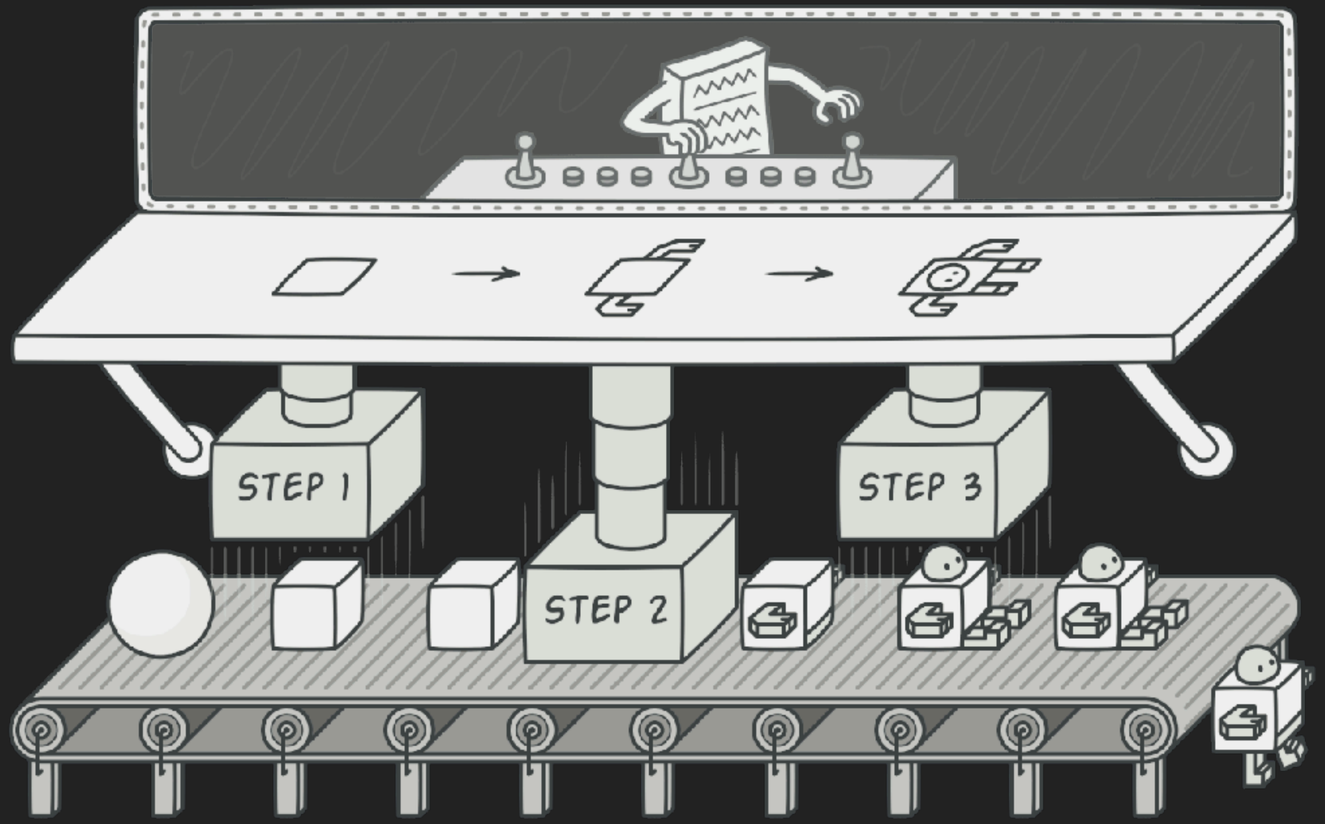
PATRONES DE CREACIÓN

- Abstract factory
- Builder
- Factory method
- Prototype
- Singleton



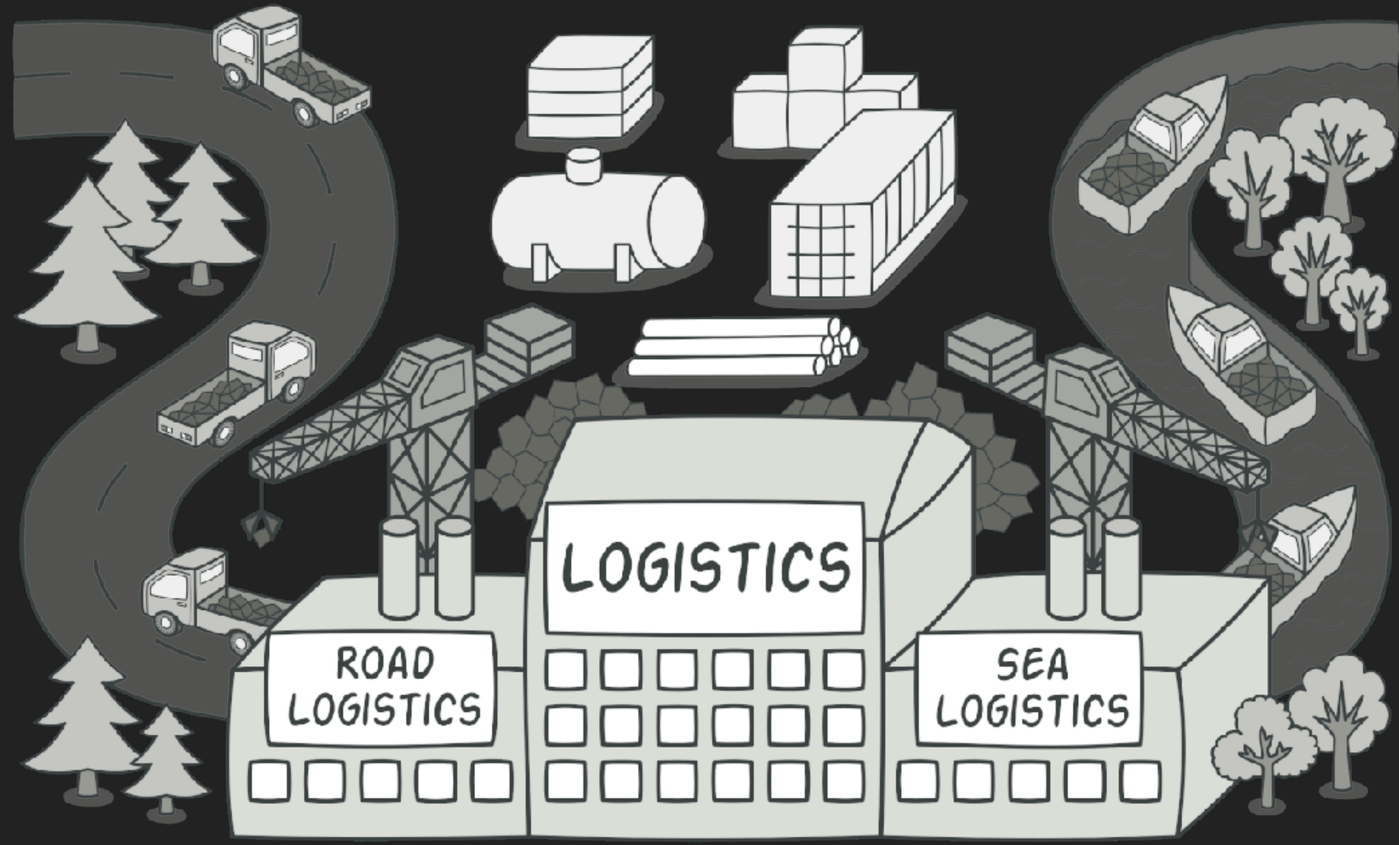
PATRONES DE CREACIÓN

- Abstract factory
- Builder
- Factory method
- Prototype
- Singleton



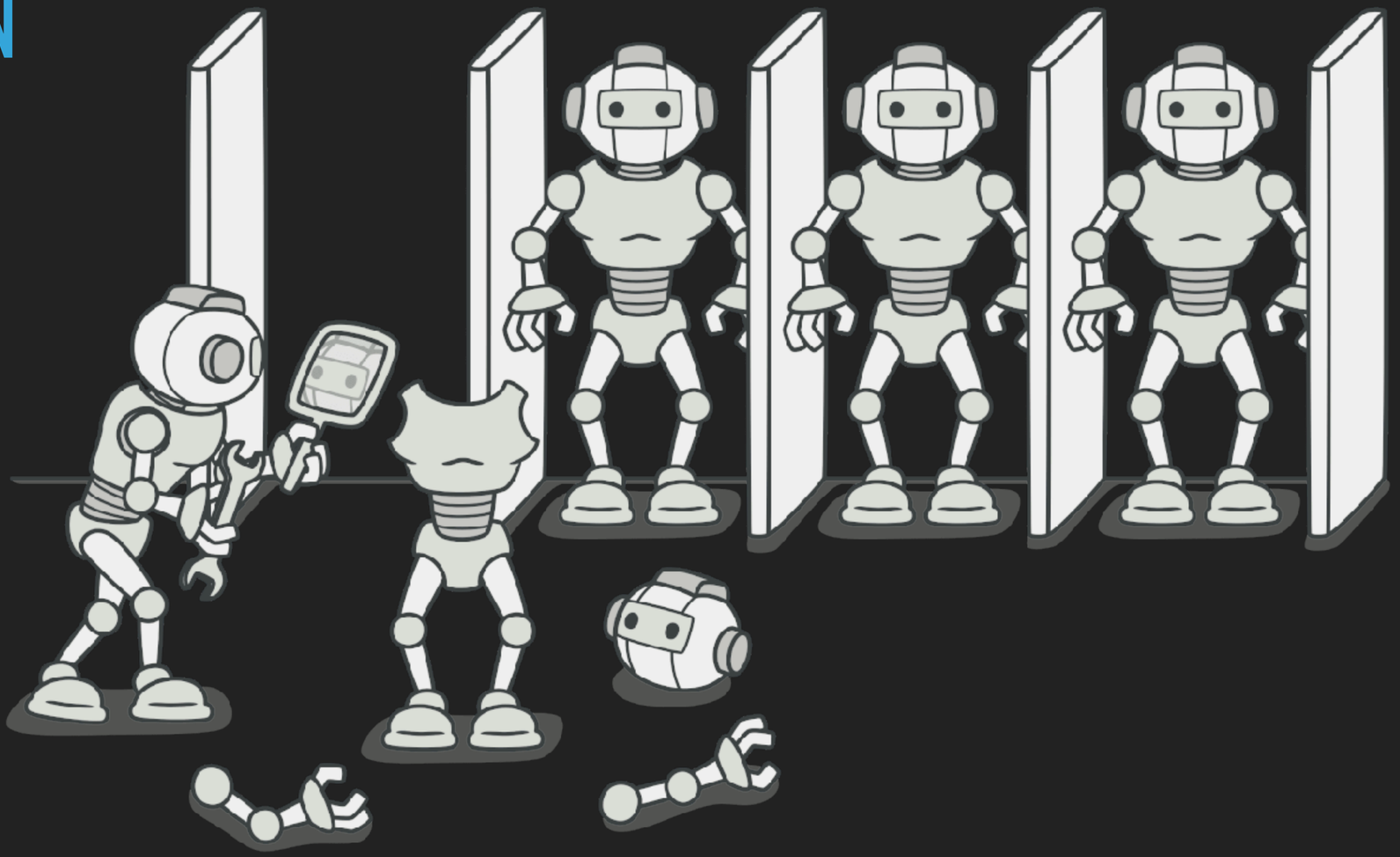
PATRONES DE CREACIÓN

- Abstract factory
- Builder
- Factory method
- Prototype
- Singleton



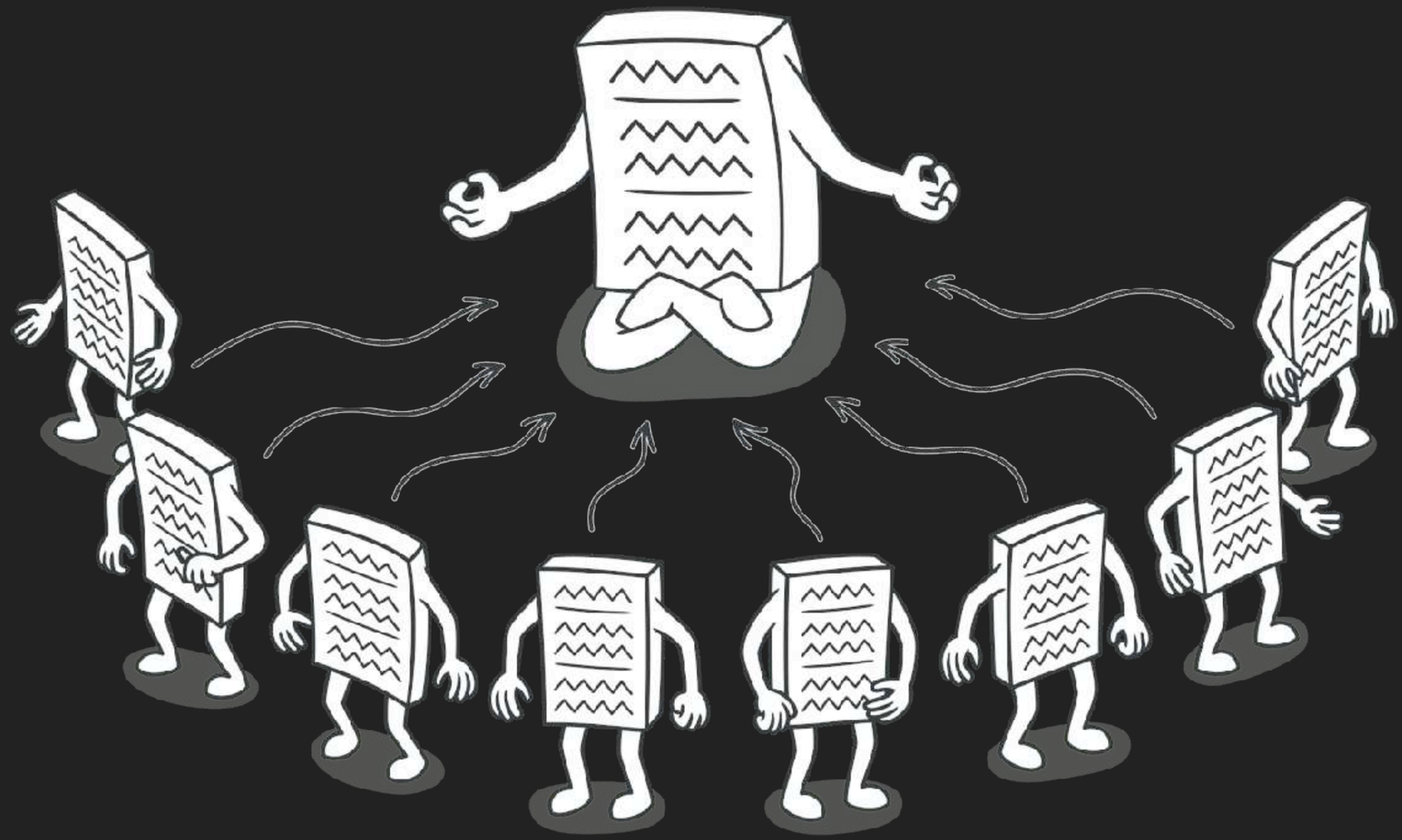
PATRONES DE CREACIÓN

- Abstract factory
- Builder
- Factory method
- Prototype
- Singleton



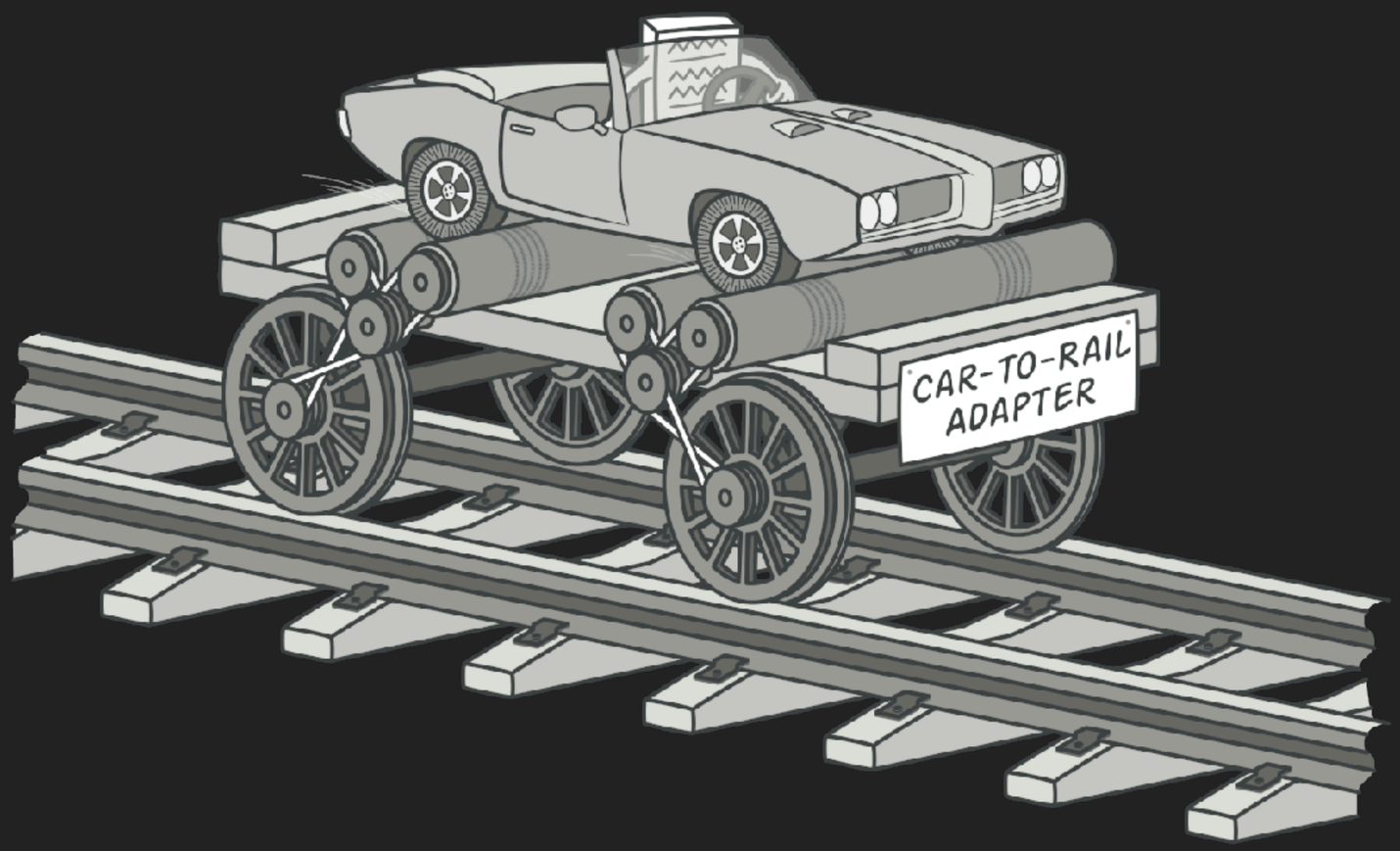
PATRONES DE CREACIÓN

- Abstract factory
- Builder
- Factory method
- Prototype
- Singleton



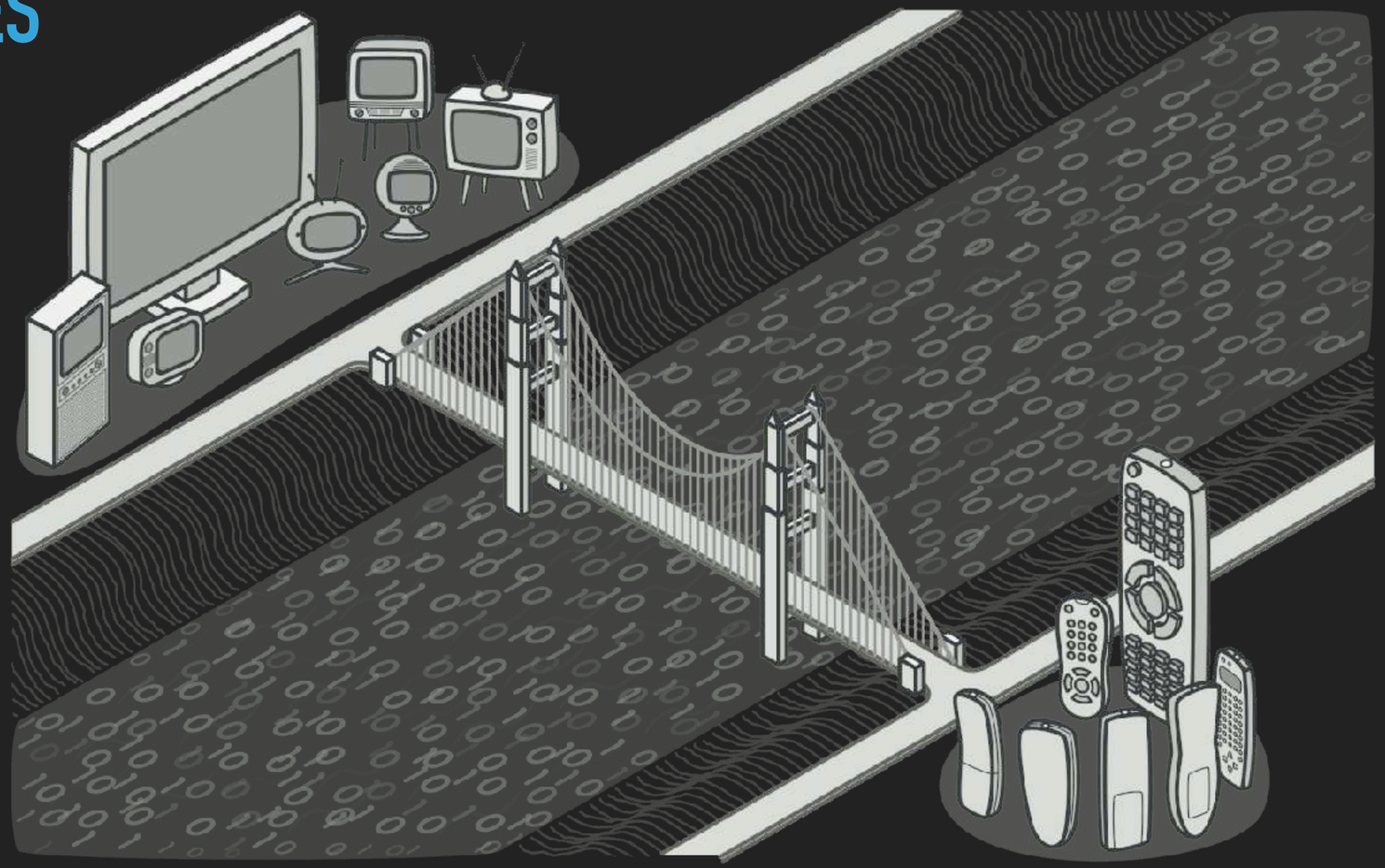
PATRONES ESTRUCTURALES

- Adapter
- Bridge
- Composite
- Decorator
- Facade
- Flyweight
- Proxy



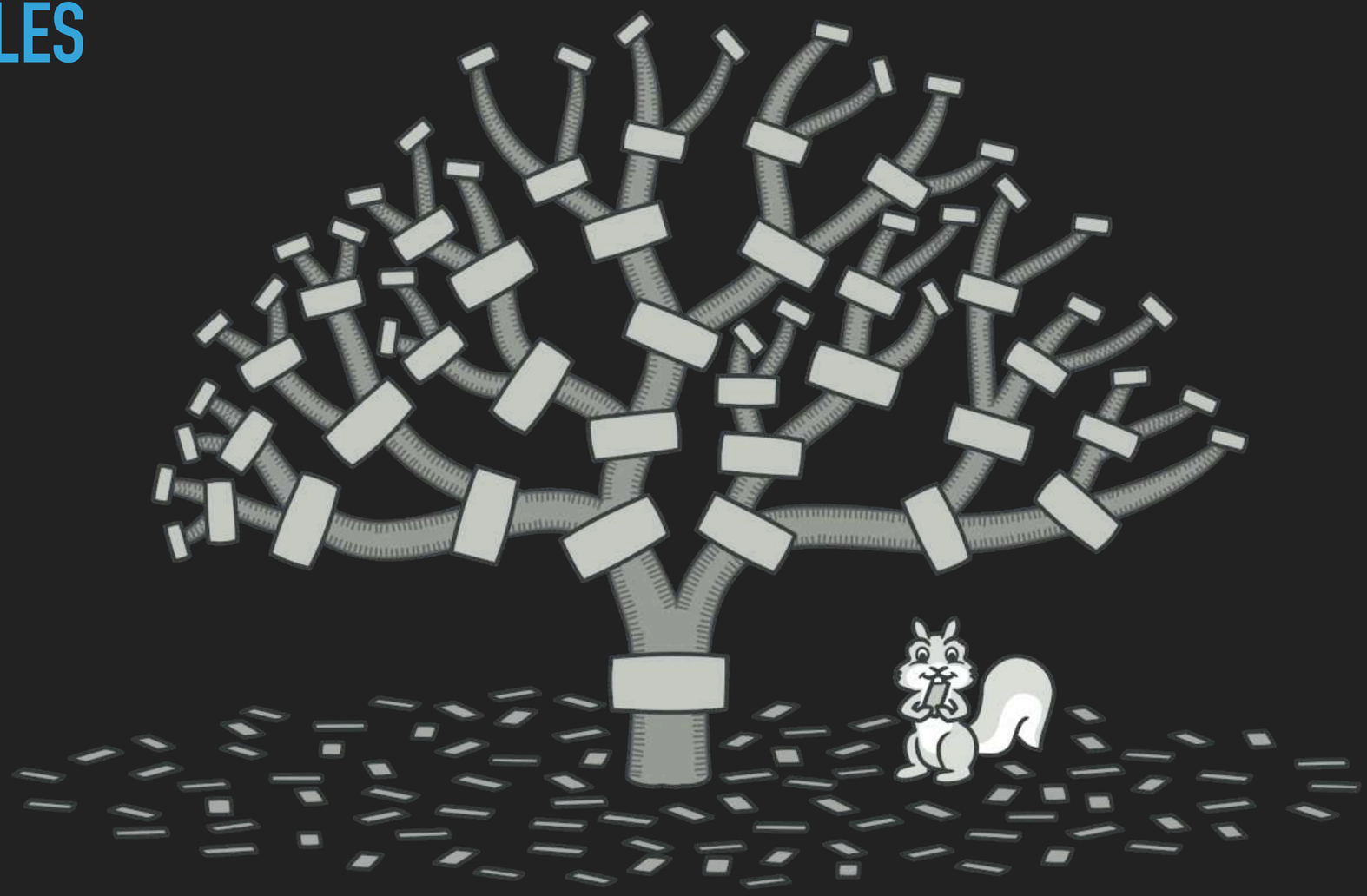
PATRONES ESTRUCTURALES

- Adapter
- Bridge
- Composite
- Decorator
- Facade
- Flyweight
- Proxy



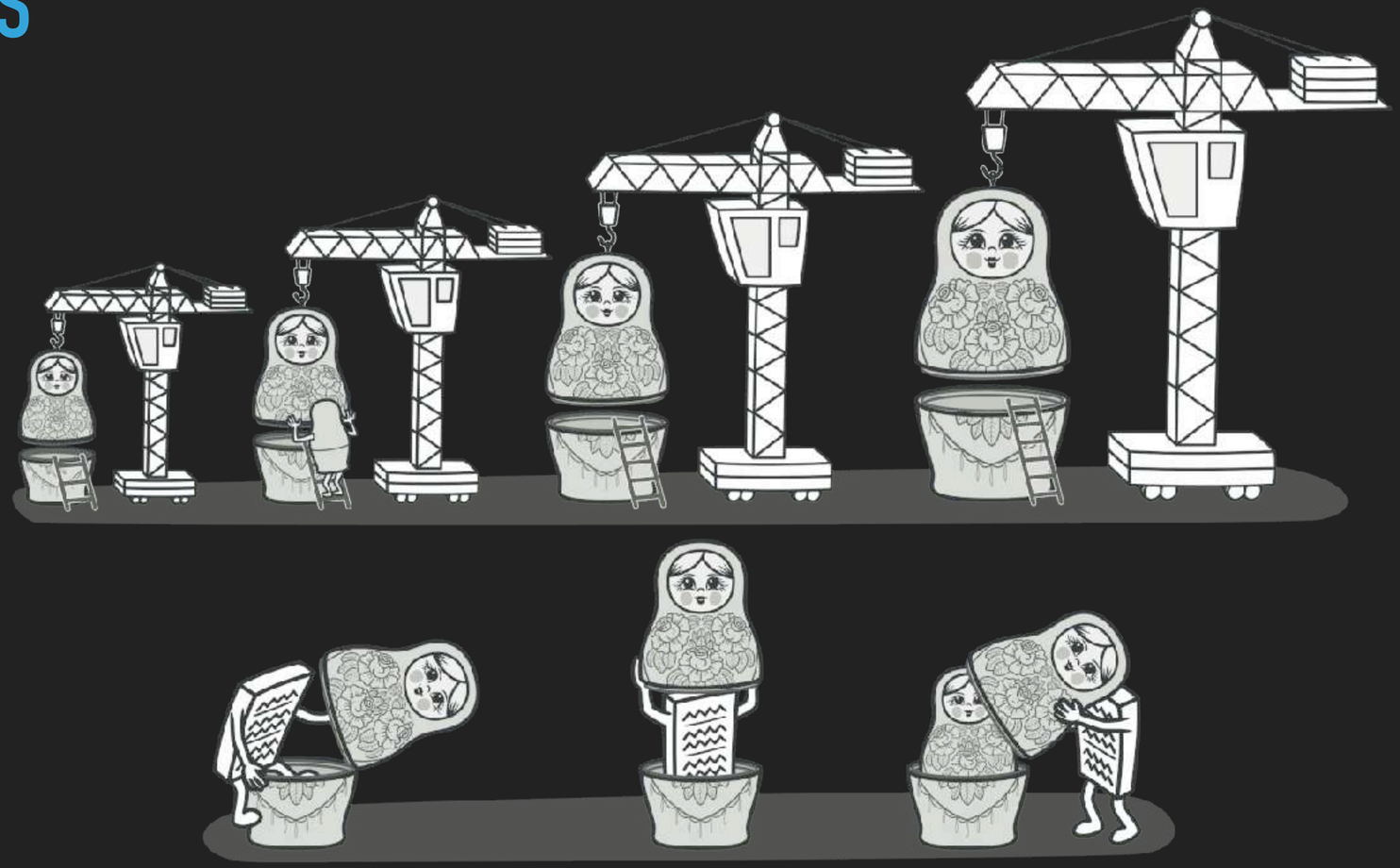
PATRONES ESTRUCTURALES

- Adapter
- Bridge
- Composite
- Decorator
- Facade
- Flyweight
- Proxy



PATRONES ESTRUCTURALES

- Adapter
- Bridge
- Composite
- Decorator
- Facade
- Flyweight
- Proxy

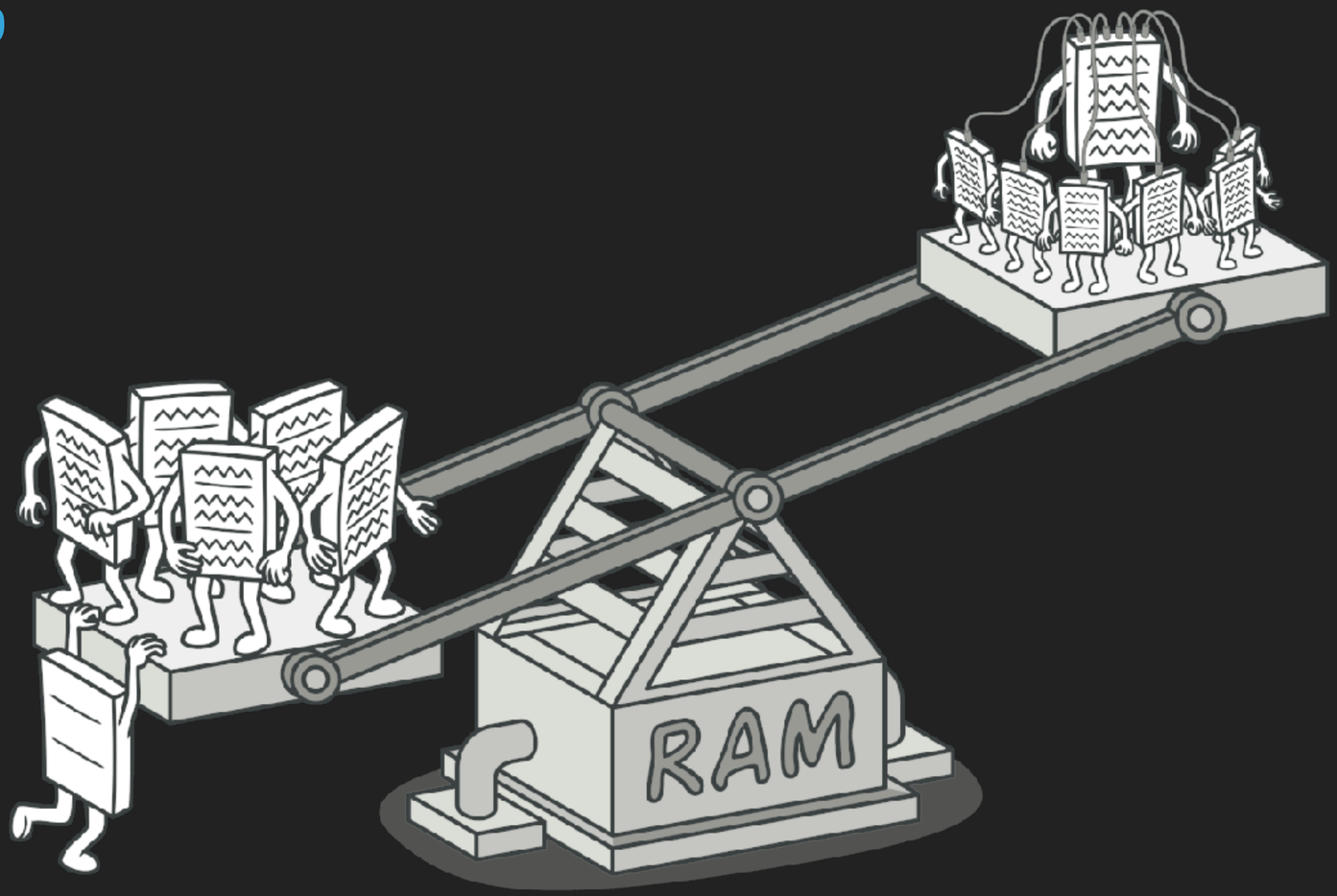


PATRONES ESTRUCTURALES

- Adapter
- Bridge
- Composite
- Decorator
- Facade
- Flyweight
- Proxy

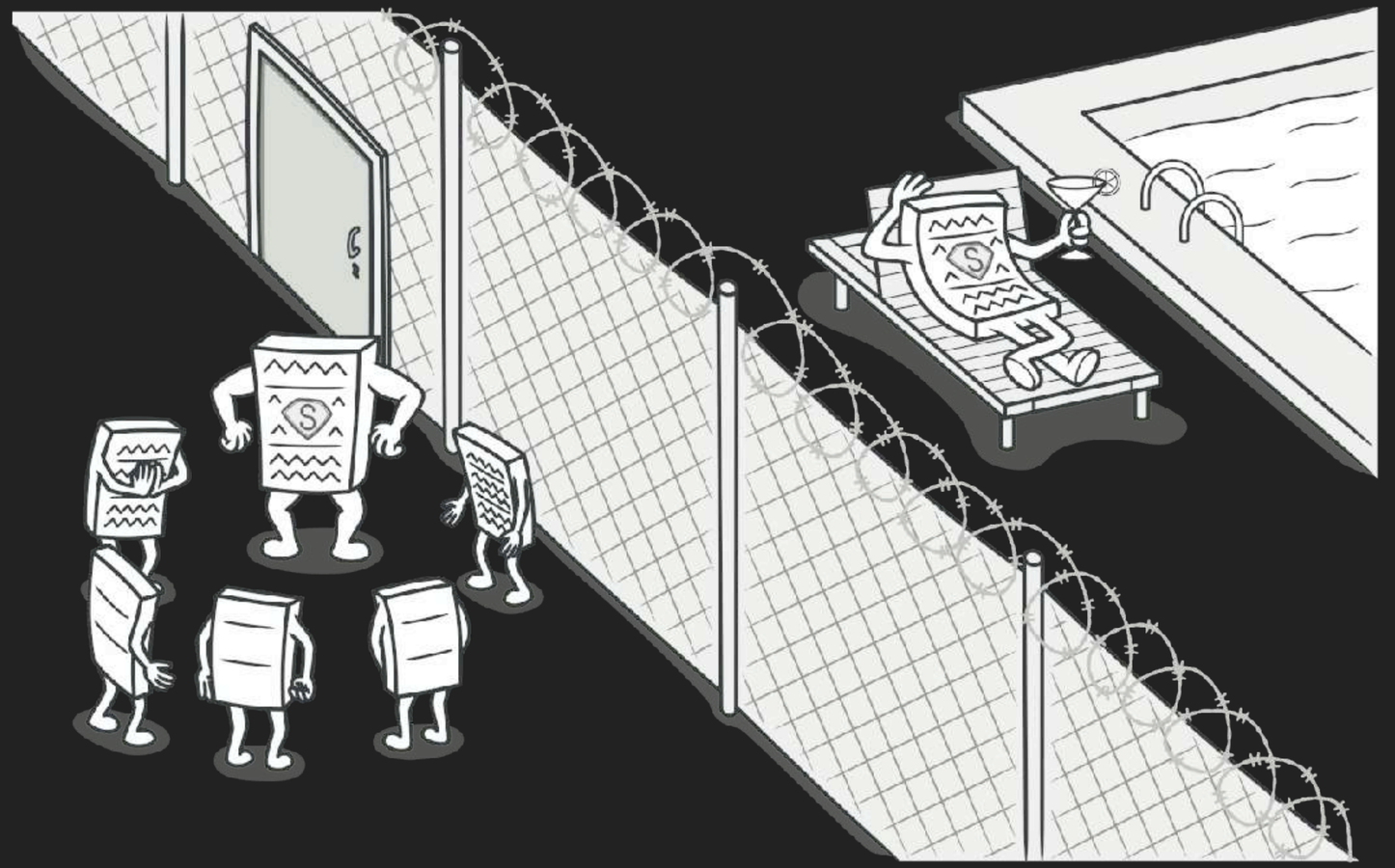
PATRONES ESTRUCTURALES

- Adapter
- Bridge
- Composite
- Decorator
- Facade
- Flyweight
- Proxy



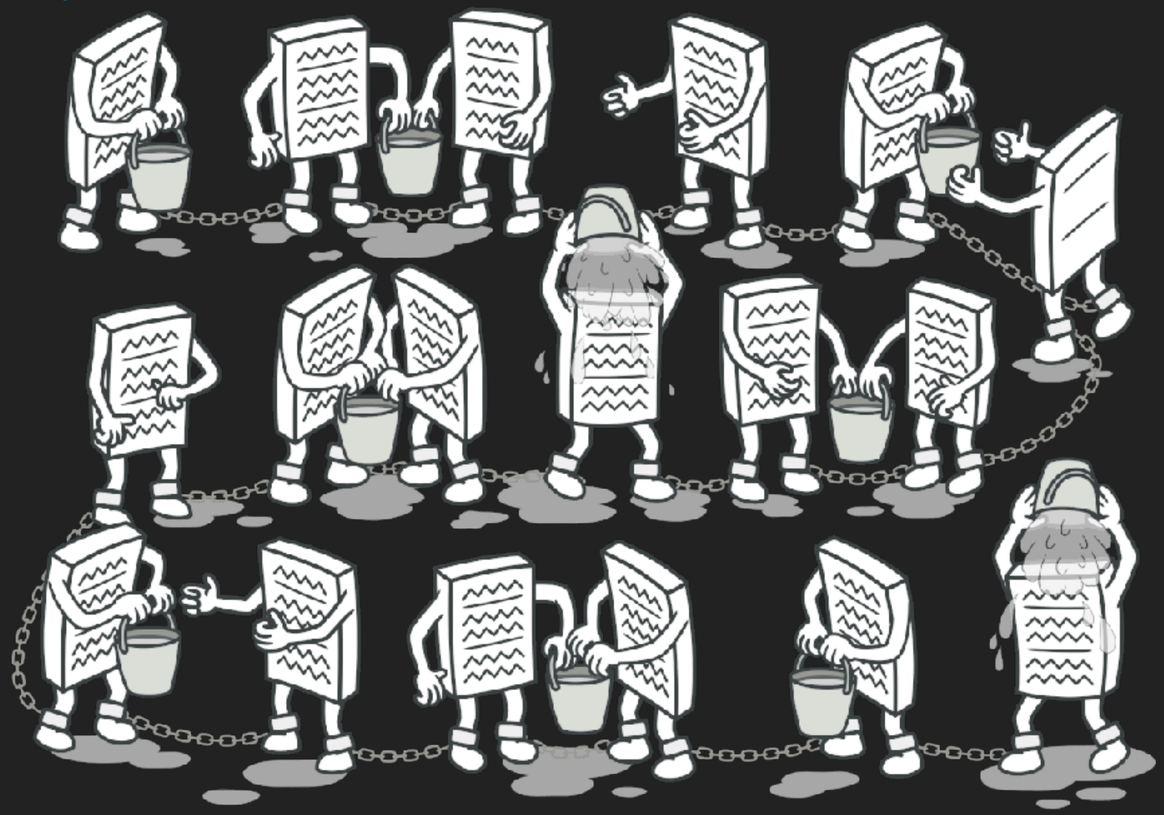
PATRONES DE COMPORTAMIENTO

- Adapter
- Bridge
- Composite
- Decorator
- Facade
- Flyweight
- Proxy



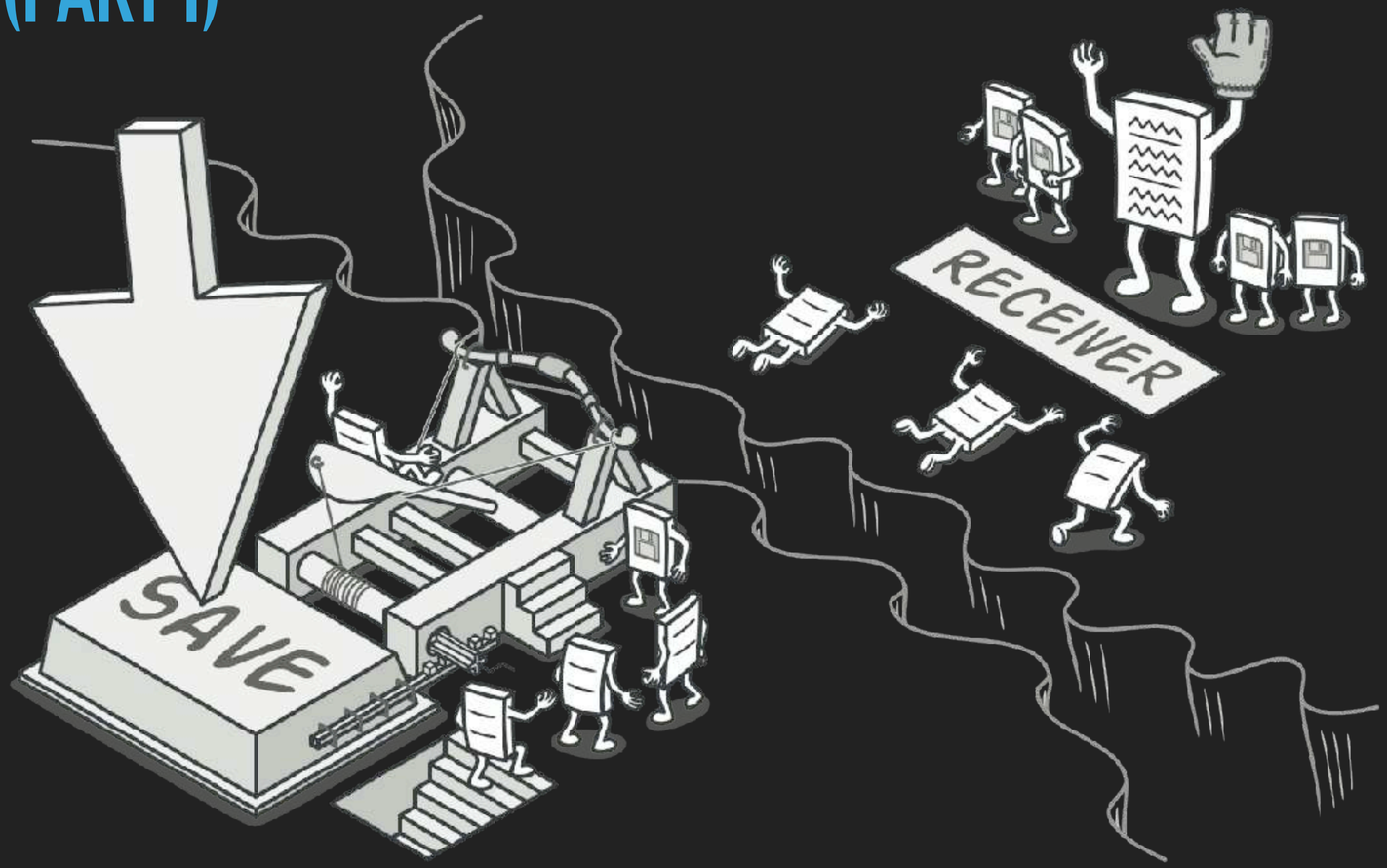
PATRONES ESTRUCTURALES (PART I)

- Chain of Responsibility
- Command
- Iterator
- Mediator
- Memento
- Observer



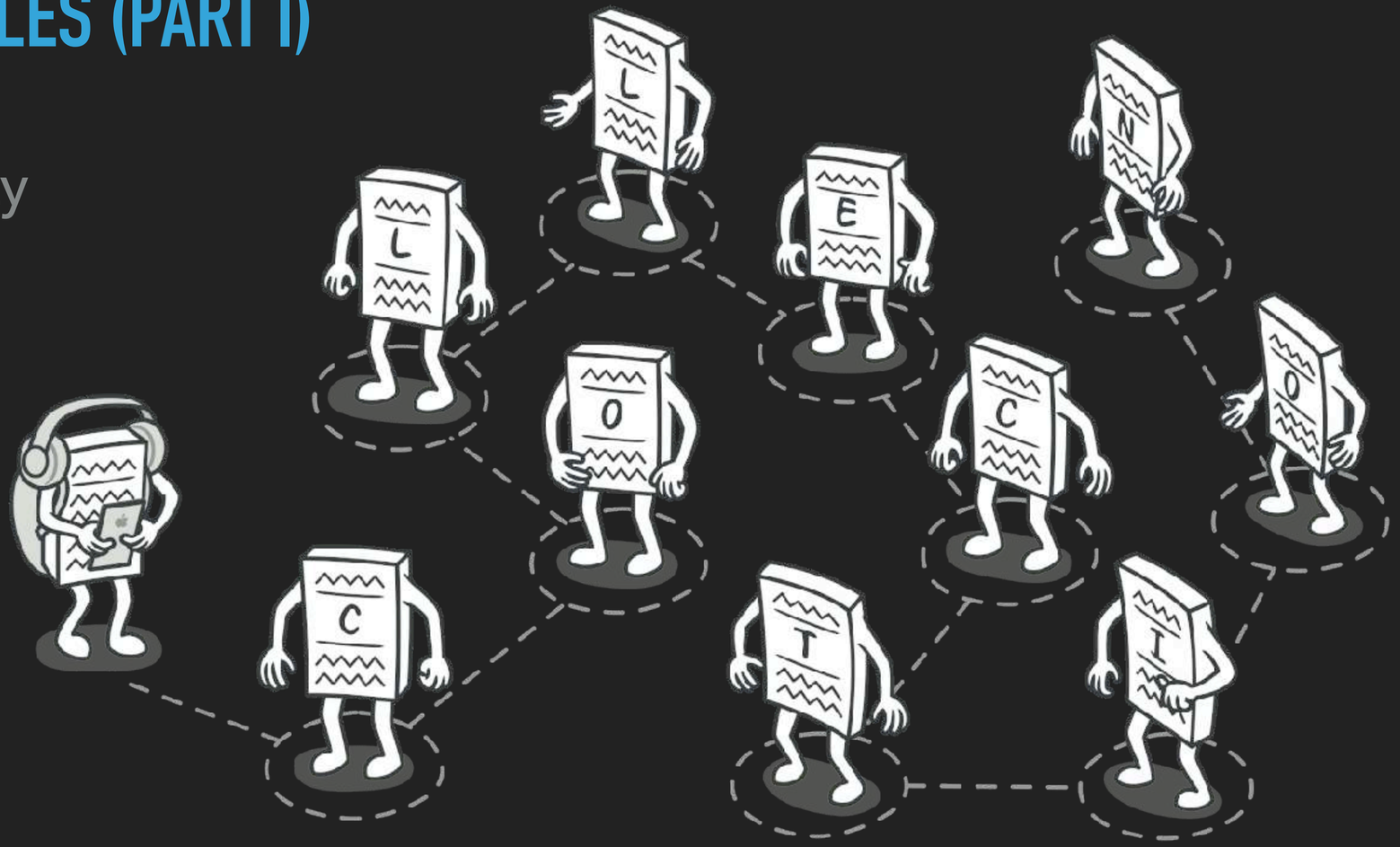
PATRONES ESTRUCTURALES (PART I)

- Chain of Responsibility
- Command
- Iterator
- Mediator
- Memento
- Observer



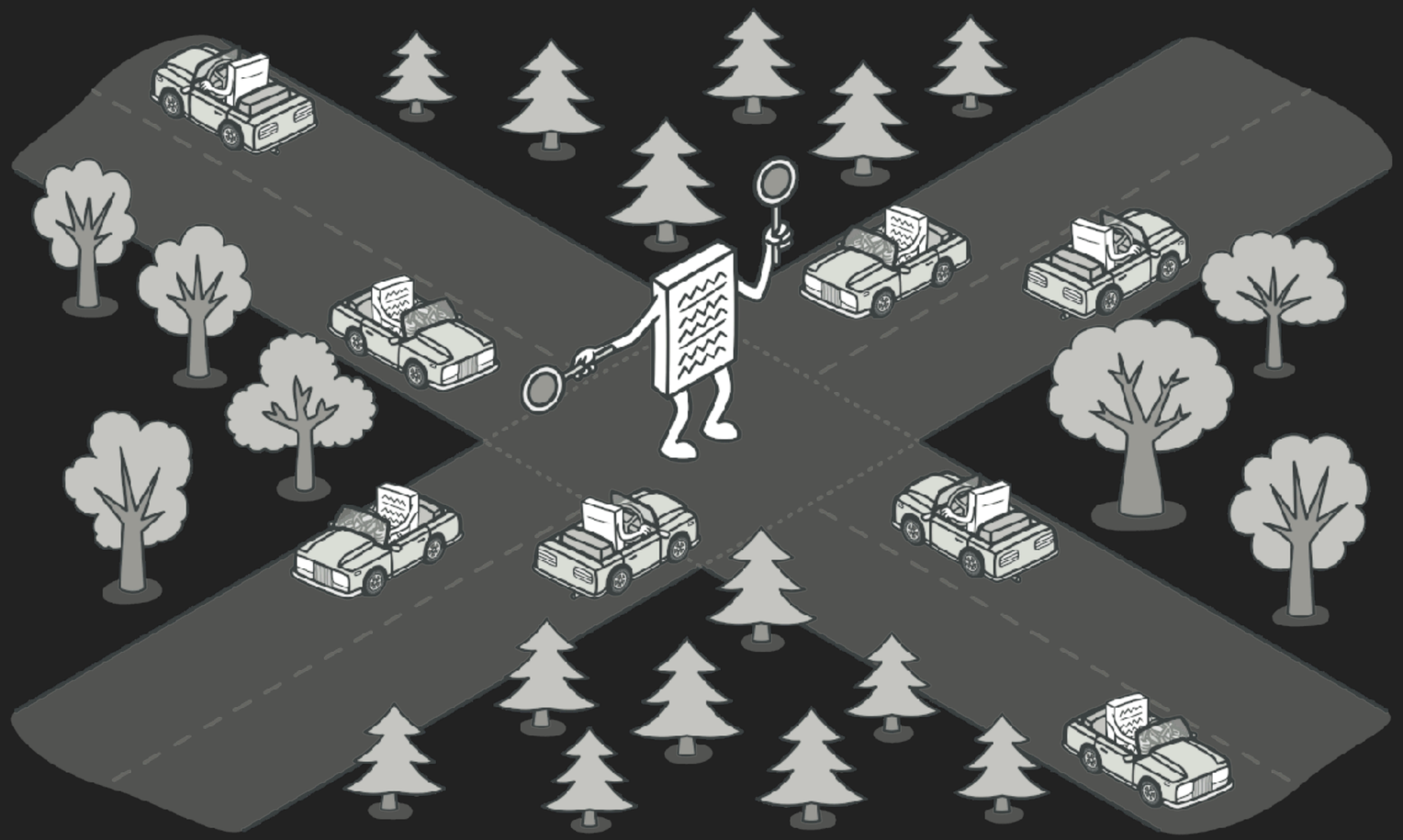
PATRONES ESTRUCTURALES (PART I)

- Chain of Responsibility
- Command
- Iterator
- Mediator
- Memento
- Observer



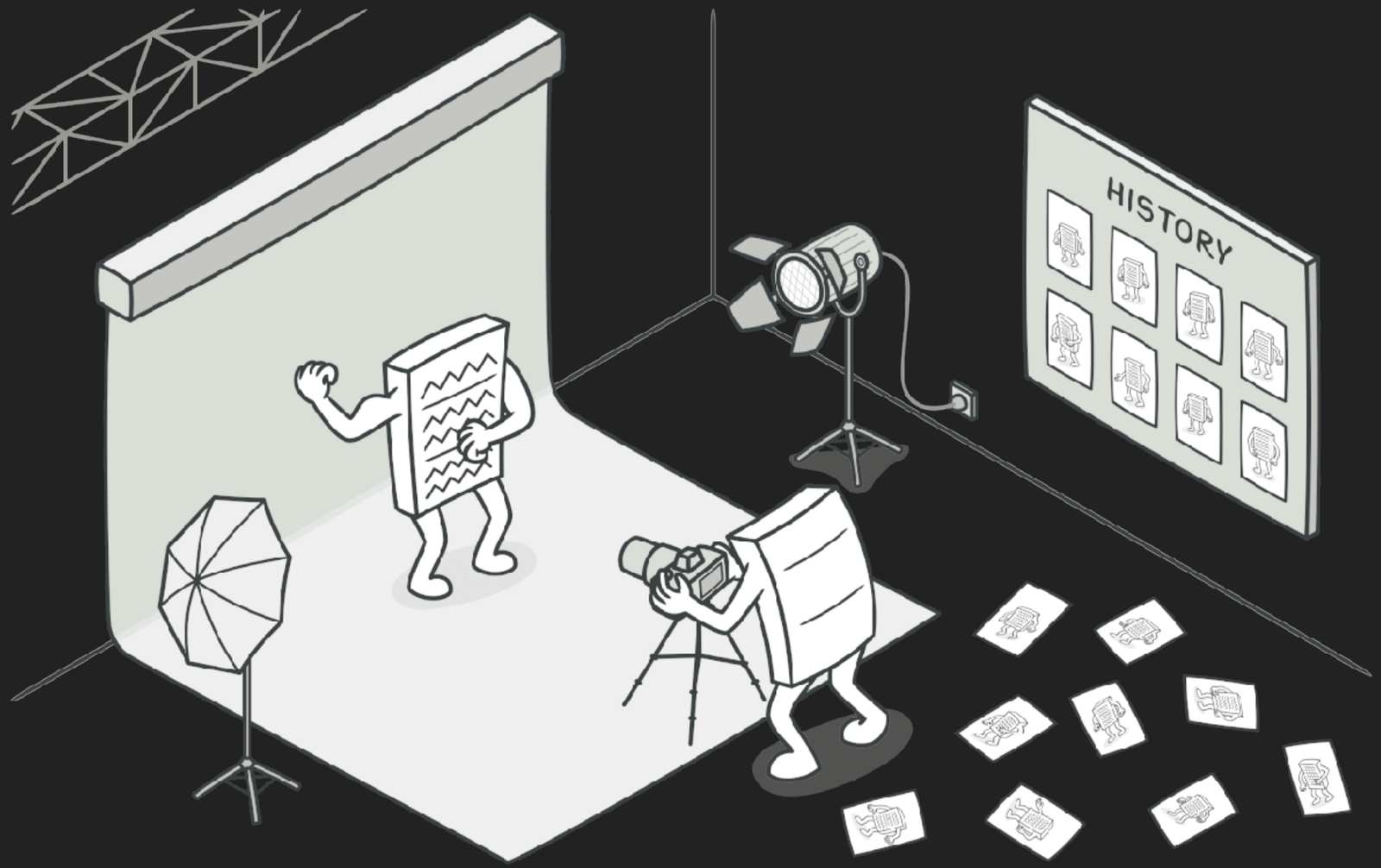
PATRONES ESTRUCTURALES (PART I)

- Chain of Responsibility
- Command
- Iterator
- Mediator
- Memento
- Observer



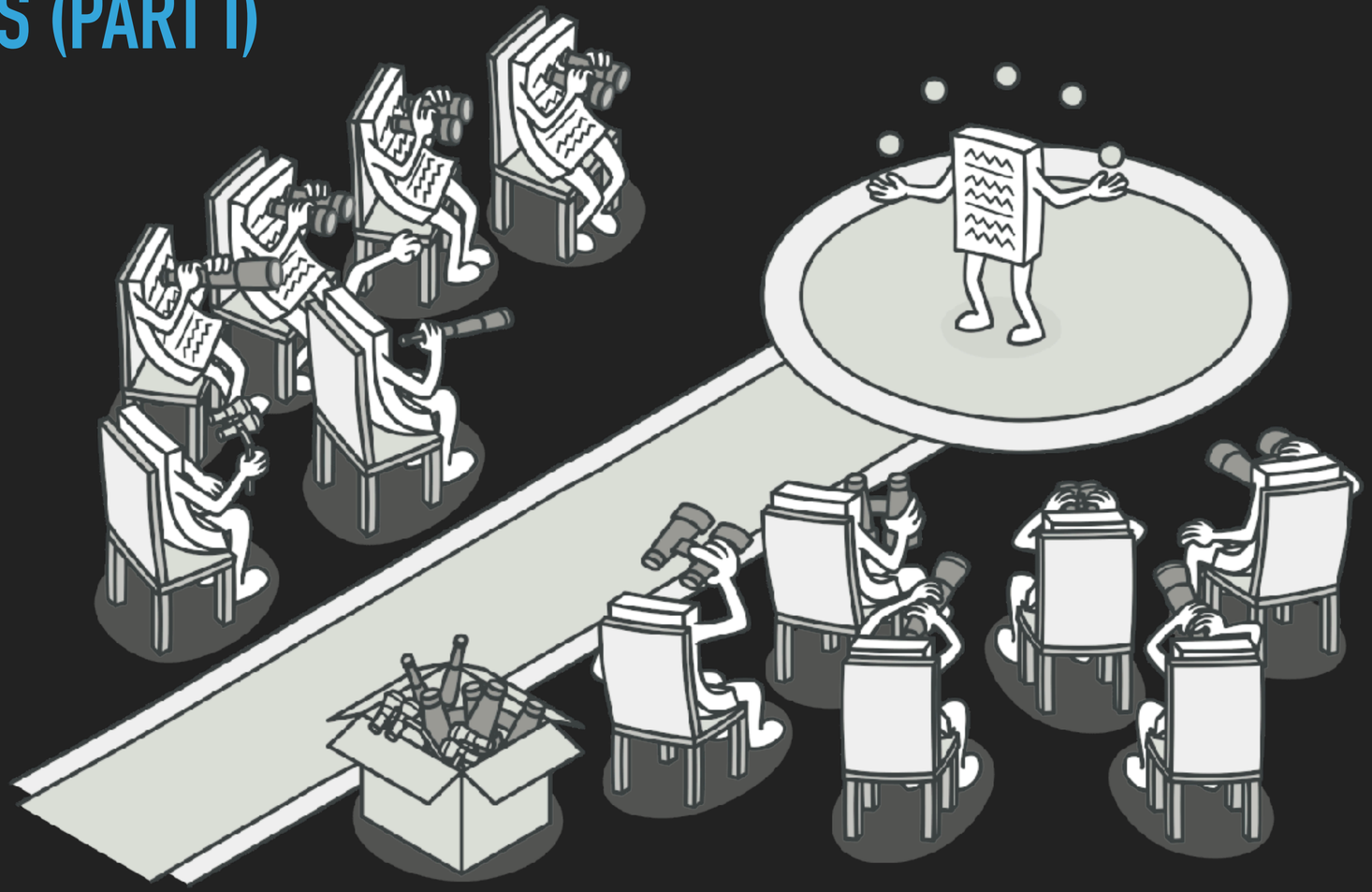
PATRONES ESTRUCTURALES (PART I)

- Chain of Responsibility
- Command
- Iterator
- Mediator
- Memento
- Observer



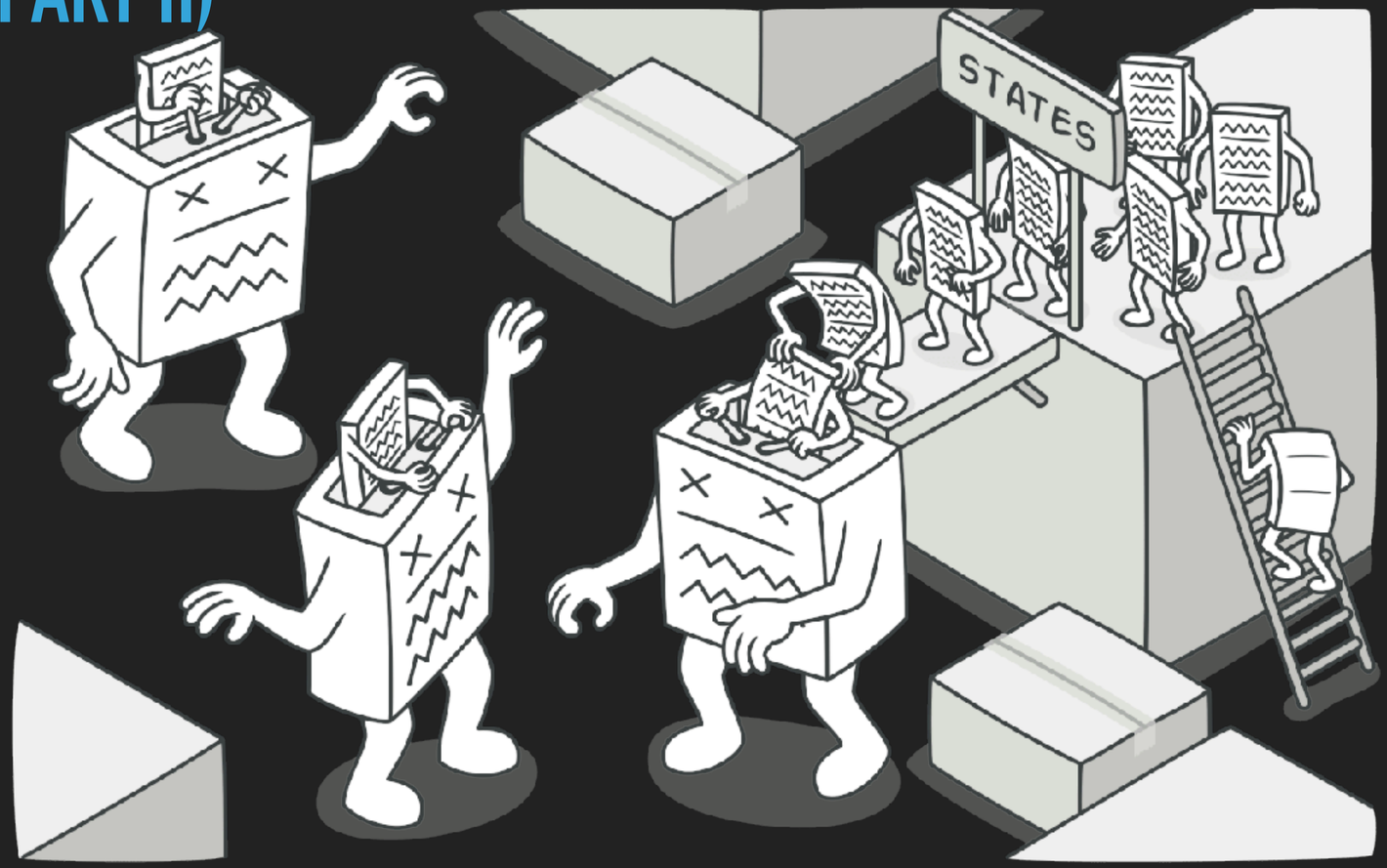
PATRONES ESTRUCTURALES (PART I)

- Chain of Responsibility
- Command
- Iterator
- Mediator
- Memento
- Observer



PATRONES ESTRUCTURALES (PART II)

- State
- Strategy
- Template method
- Visitor

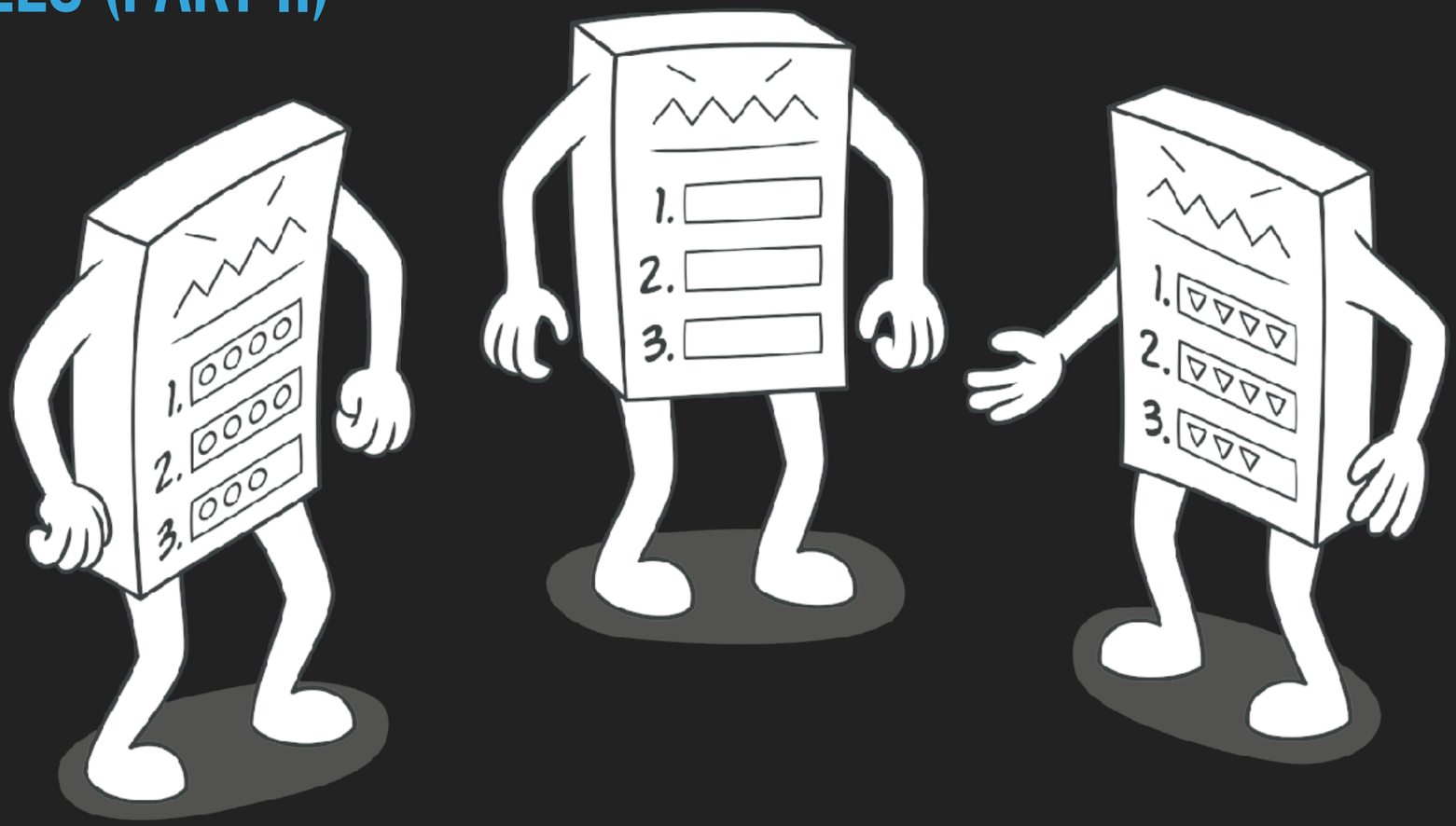


PATRONES ESTRUCTURALES (PART II)

- State
- Strategy
- Template method
- Visitor

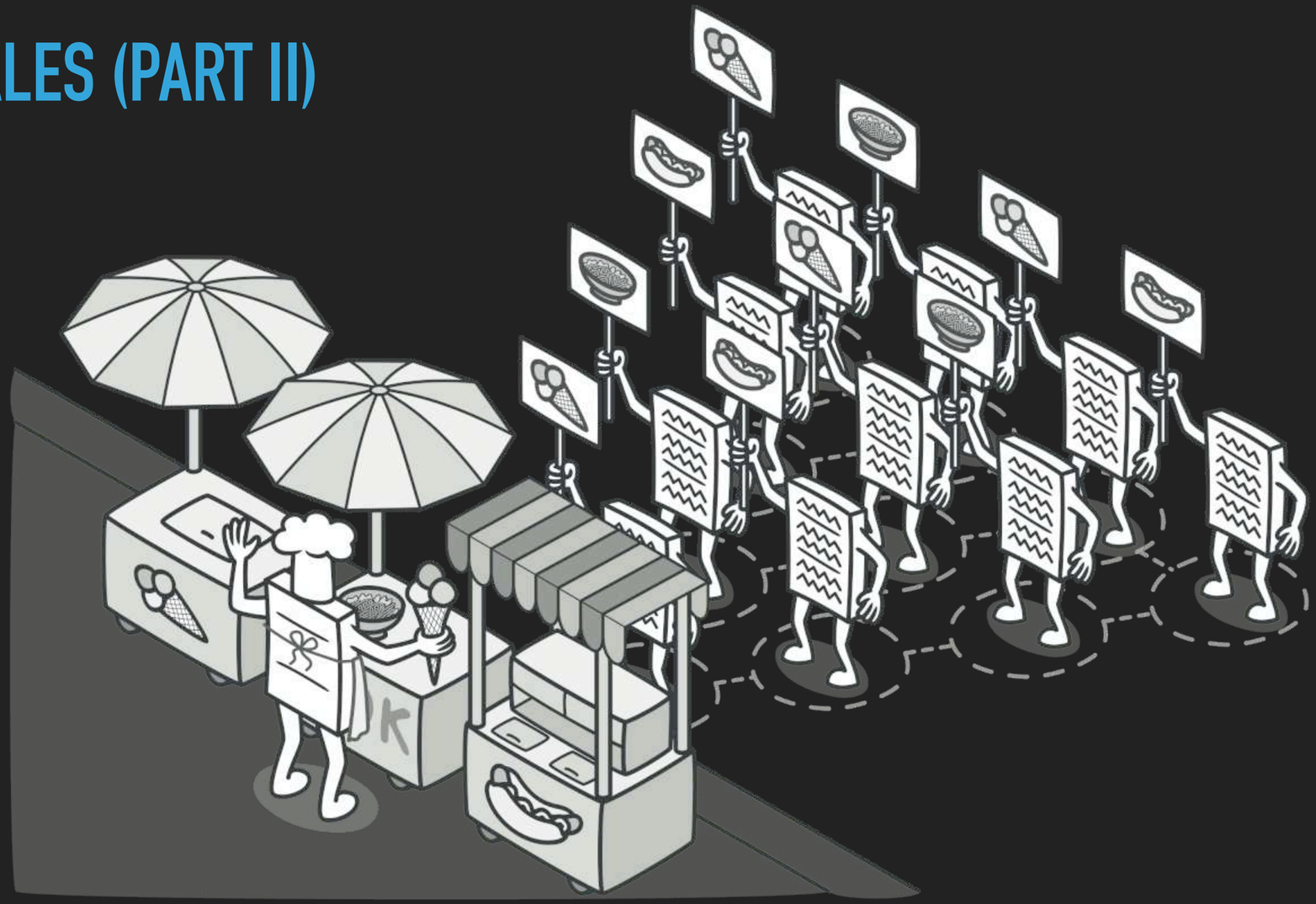
PATRONES ESTRUCTURALES (PART II)

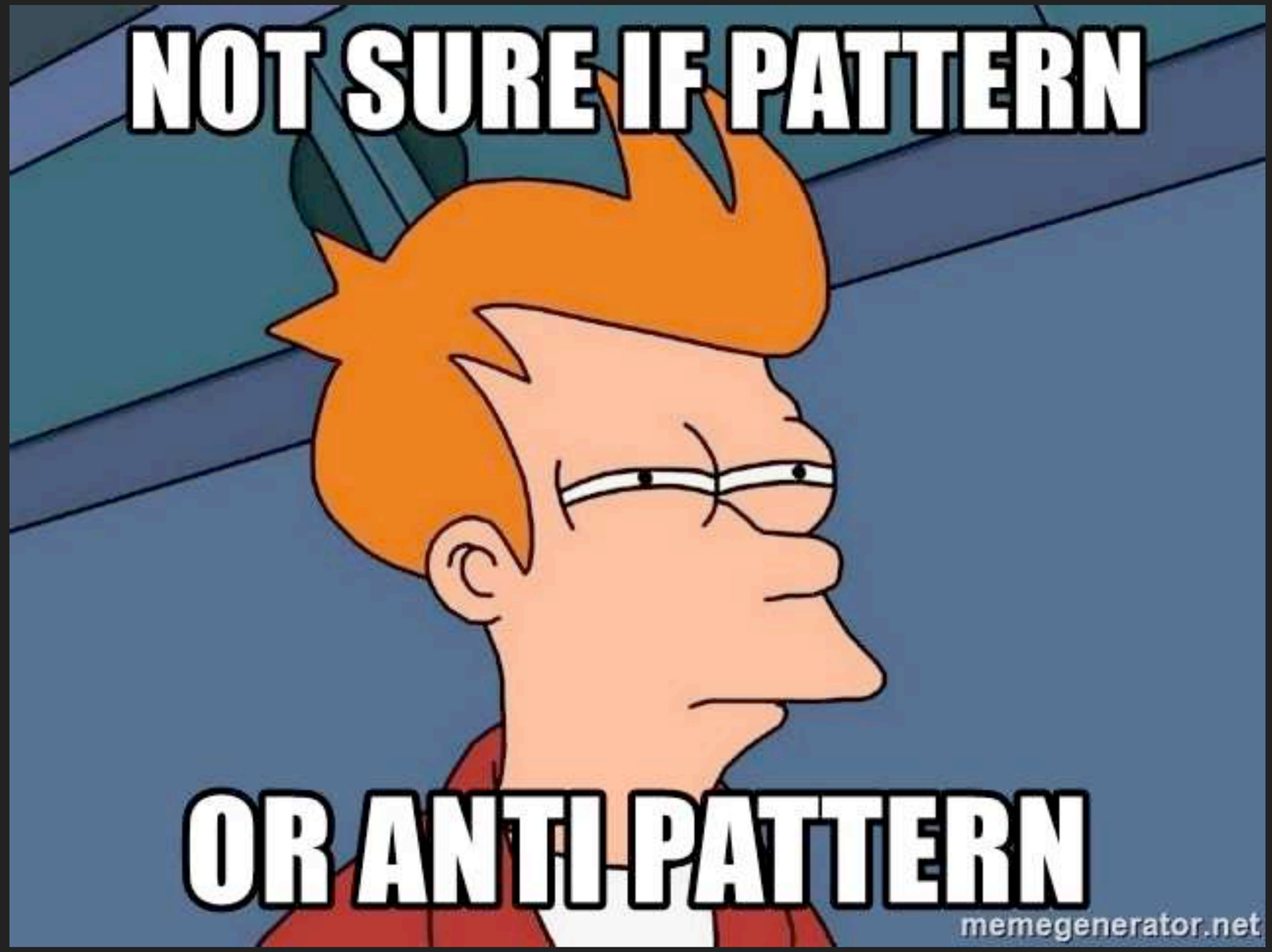
- State
- Strategy
- Template method
- Visitor



PATRONES ESTRUCTURALES (PART II)

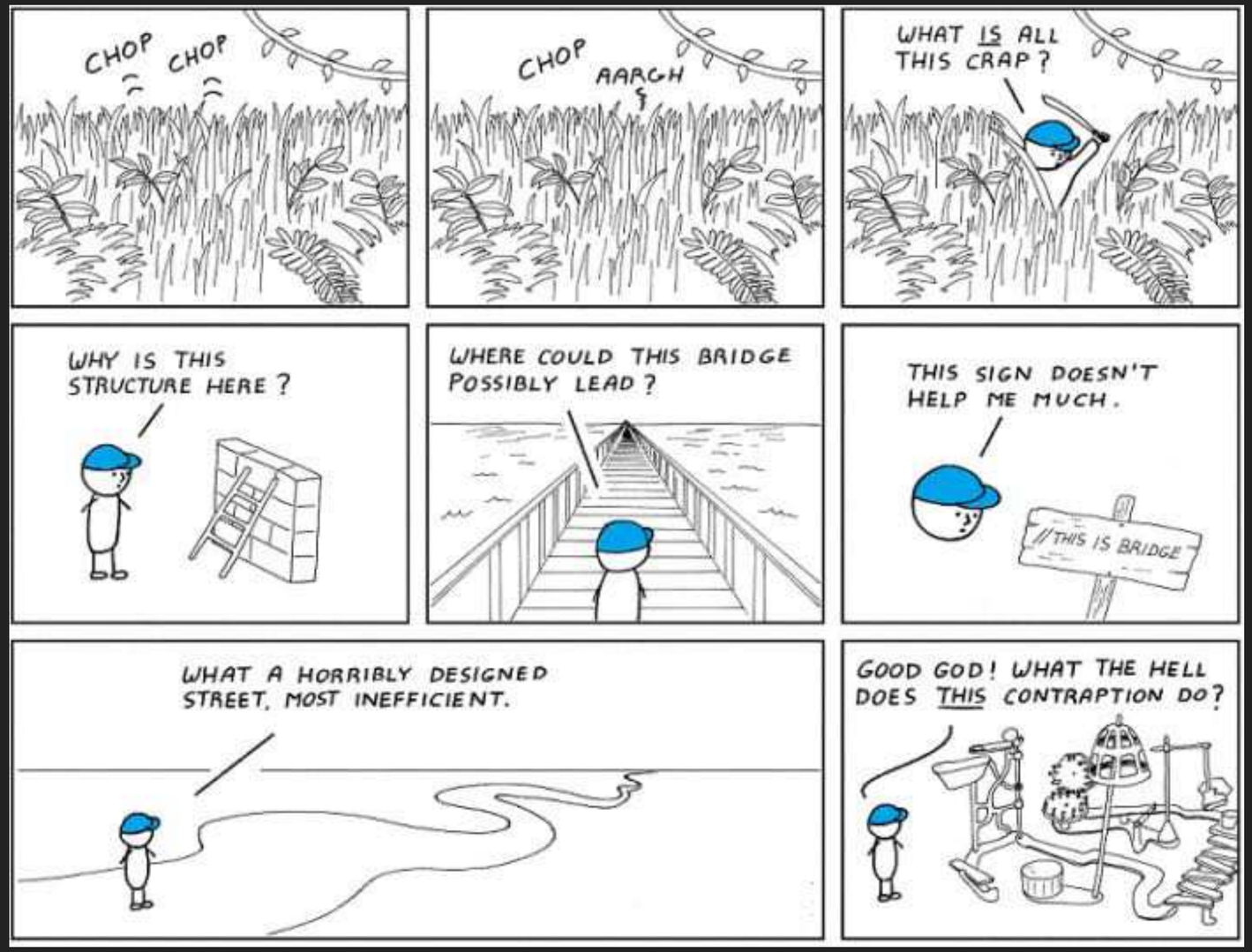
- State
- Strategy
- Template method
- Visitor




NOT SURE IF PATTERN
OR ANTI-PATTERN
memegenerator.net

ANTI PATTERNS

- God object
- Circular dependency
- Yoyo problem
- Magic numbers
- Dead code
- Spaghetti code
- CtrlC+CtrlV programming
- ...



- Qué es desarrollo de software?
- Principios de Software (*software principles*)
- Patrones de diseño (*patterns design*)
- DEMO!
- Revisión de código (*code review*)
- Take aways
  - Soft skills
  - Entrevistas
  - Mastering the keyboard

PIEDRA, PAPEL, Y TIJERAS

- Dos modos de juego:
  - Solo: humano vs maquina. El juego recibe el objeto seleccionado por el jugador (humano) y lo compara contra un objeto que la maquina selecciona para determinar al vencedor. La maquina puede tener dos niveles de dificultad, modo fácil (siempre saca piedra), y modo normal (elige aleatoriamente algún objeto)
  - Multijugador: humano vs humano. El juego recibe el objeto seleccionado por dos jugadores y determina al vencedor.
- Los datos de entrada serán los objetos seleccionados (piedra, papel, o tijera) por el(los) jugador(es).
- Asumir que los datos siempre serán correctos. No hay necesidad de validaciones.

PIEDRA, PAPEL, Y TIJERAS

Ejemplo:

```
$ ./piedra_papel_tijera --modo=multijugador
Jugador1: piedra
Jugador2: papel
Vencedor: Jugador1

$ ./piedra_papel_tijera --modo=solo --dificultad=facil
Jugador1: papel
Computador: piedra
Vencedor: Jugador1
```

- Qué es desarrollo de software?
- Principios de Software (*software principles*)
- Patrones de diseño (*patterns design*)
- DEMO!
- Revisión de código (*code review*)
- Take aways
  - Soft skills
  - Entrevistas
  - Mastering the keyboard


LETS DO A QUICK CODE REVIEW
This little maneuver is gonna cost us 51 years
imgflip.com

- Qué es desarrollo de software?
- Principios de Software (*software principles*)
- Patrones de diseño (*patterns design*)
- DEMO!
- Revisión de código (*code review*)
- Take aways
  - Soft skills
  - Entrevistas
  - Mastering the keyboard

CREDITOS

- Clean code:
  - https://gist.github.com/wojteklu/73c6914cc446146b8b533c0988cf8d29
- Design patterns:
  - https://refactoring.guru/