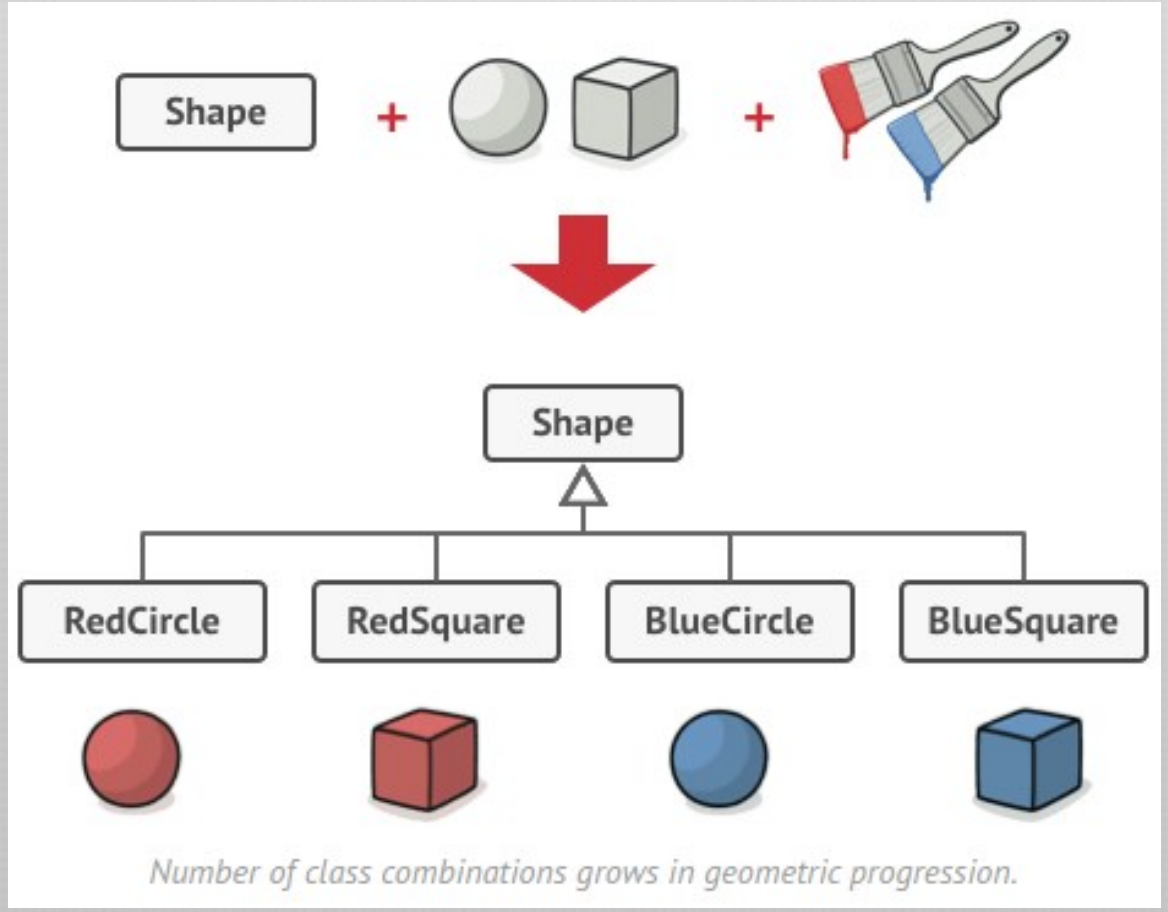
Tecnología de Programación

Martín L. Larrea

Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur

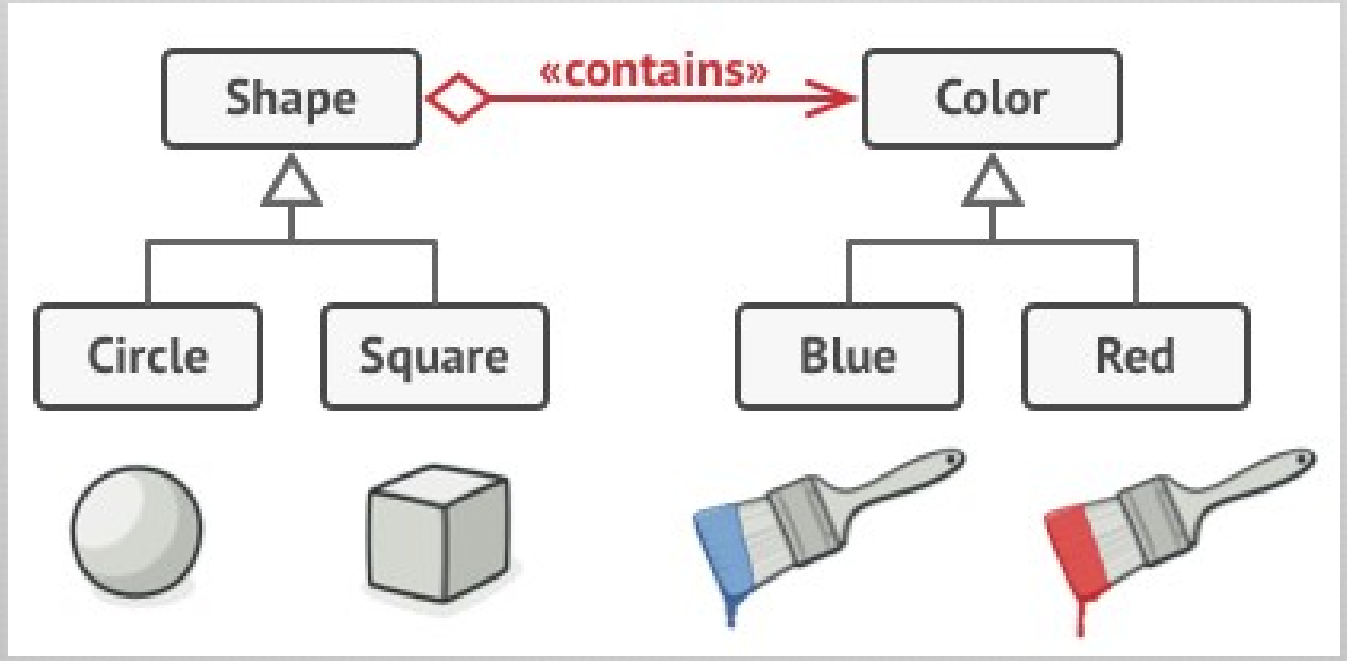
Patrón: Bridge

- Tipo: Estructural
- Intención: Separar una gran clase o conjunto de clases en dos jerarquías, **abstracción** e **implementación** las cuales pueden ser desarrolladas independientemente.



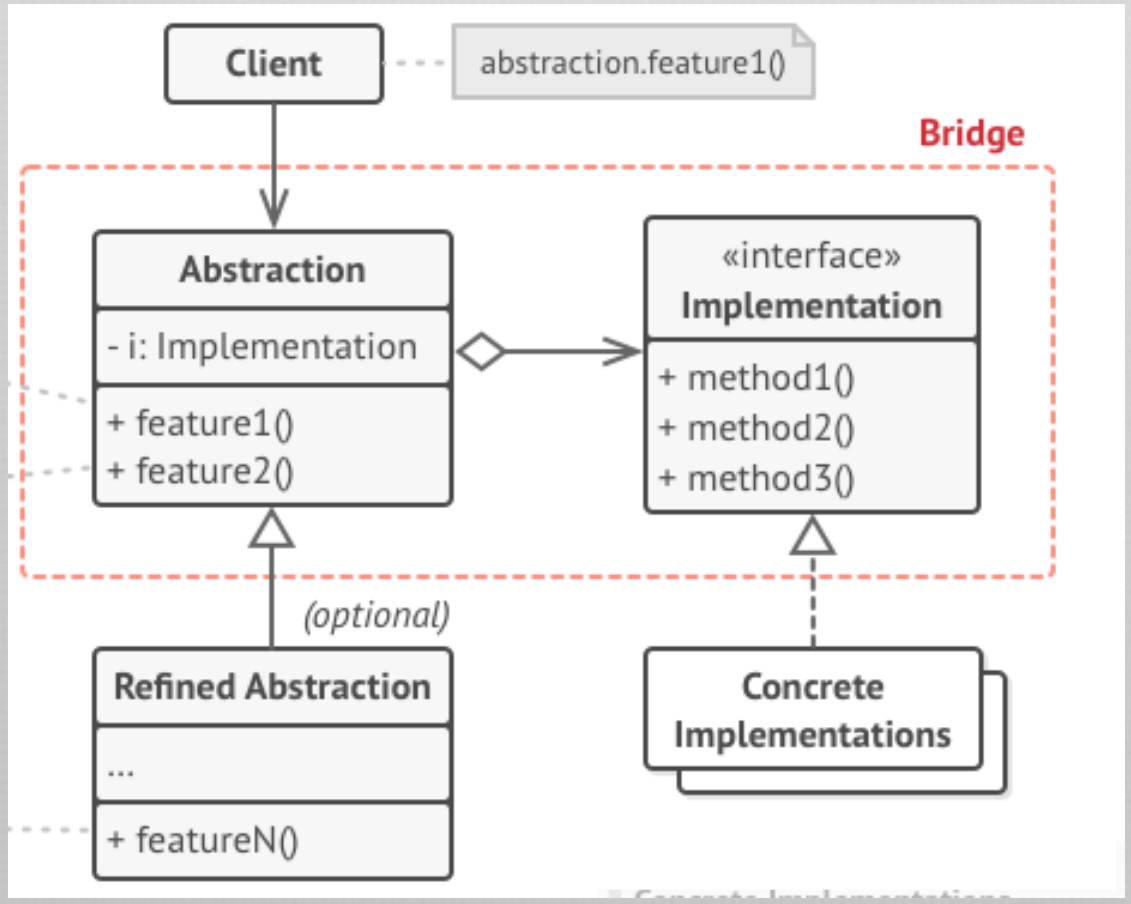
Patrón: Bridge

- Tipo: Estructural
- Intención: Separar una gran clase o conjunto de clases en dos jerarquías, **abstracción** e **implementación** las cuales pueden ser desarrolladas independientemente.



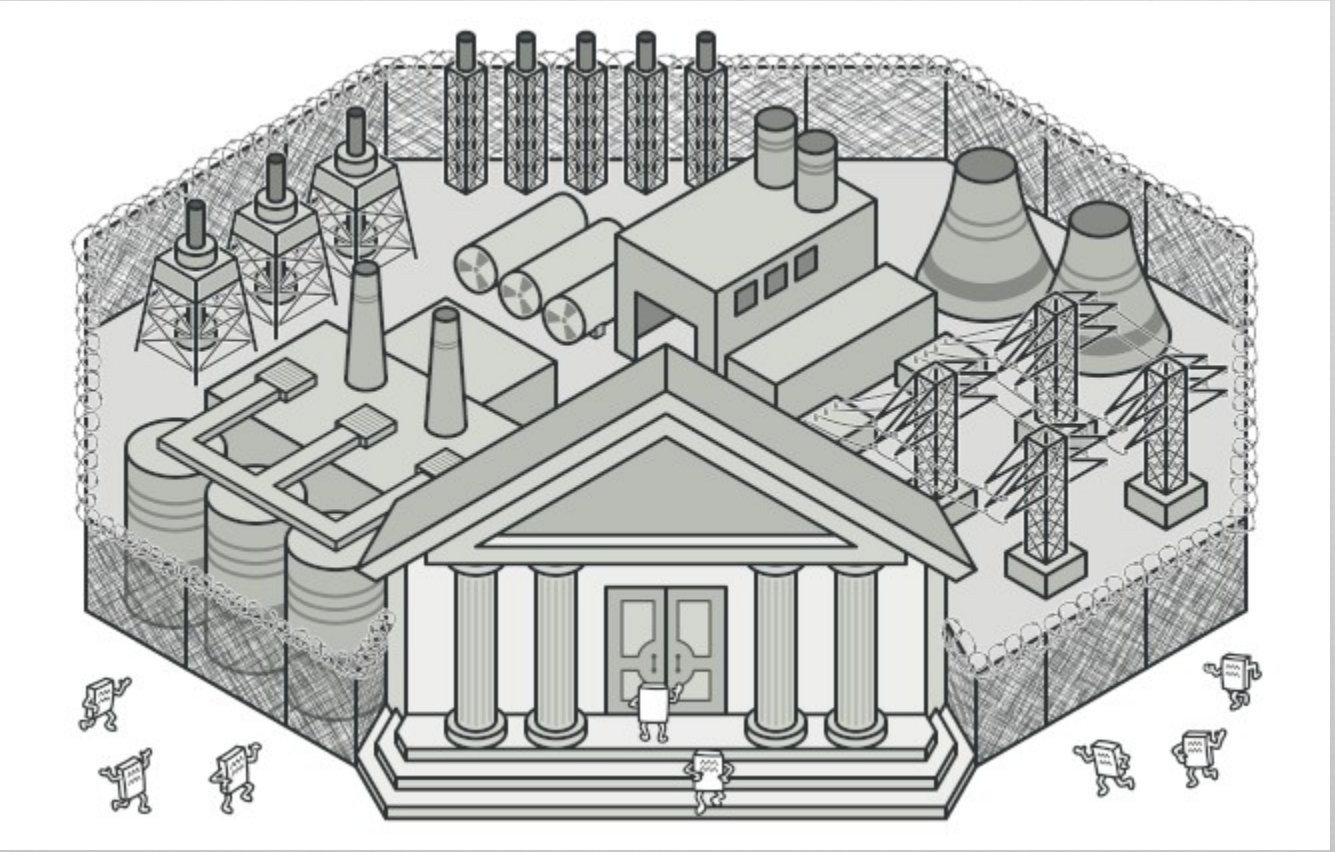
Patrón: Bridge

- Tipo: Estructural
- Intención: Separar una gran clase o conjunto de clases en dos jerarquías, **abstracción** e **implementación** las cuales pueden ser desarrolladas independientemente.



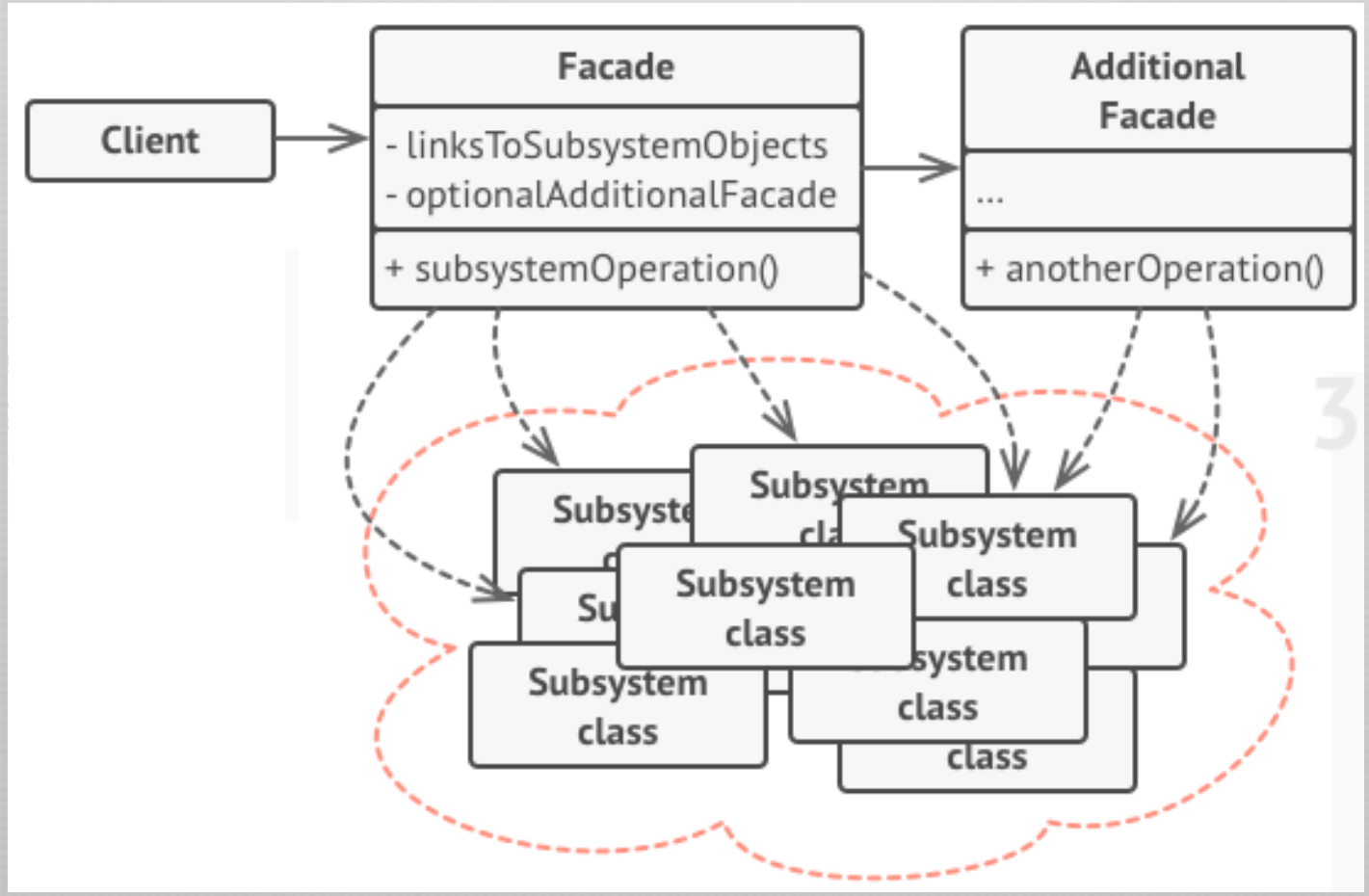
Patrón: Facade

- Tipo: Estructural
- Intención: Proveer una interfaz simplificada para acceder a una librería, framework o conjunto complejo de clases.



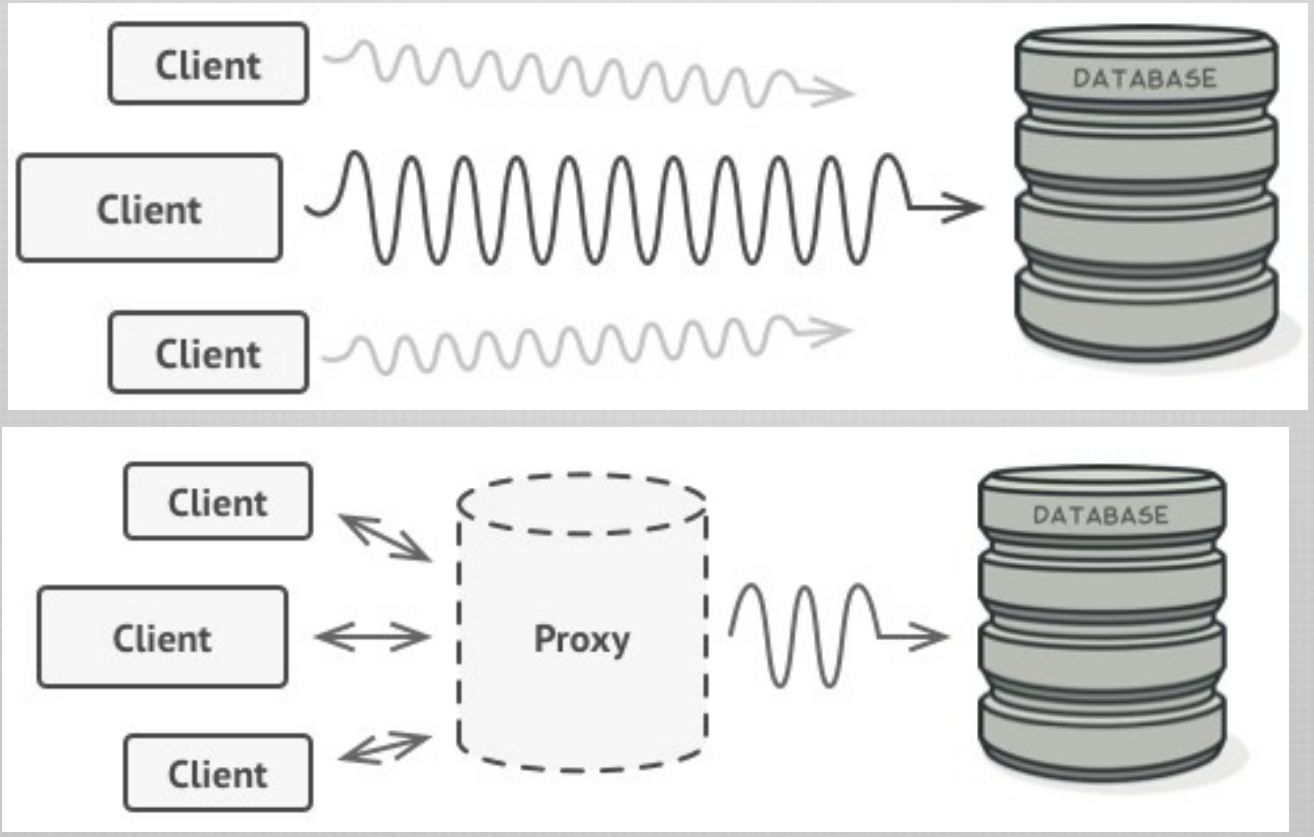
Patrón: Facade

- Tipo: Estructural
- Intención: Proveer una interfaz simplificada para acceder a una librería, framework o conjunto complejo de clases.



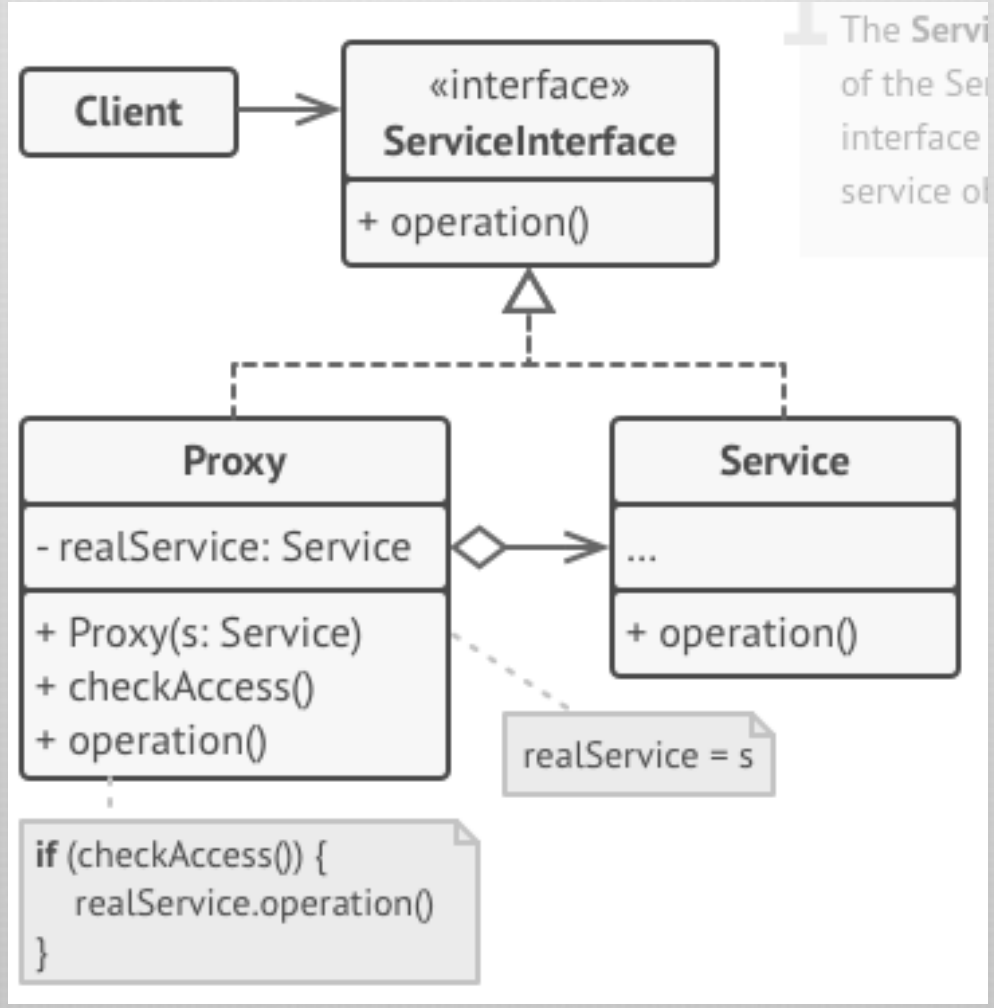
Patrón: Proxy

- Tipo: Estructural
- Intención: Proveer un control al acceso a una clase.



Patrón: Proxy

- Tipo: Estructural
- Intención: Proveer un control al acceso a una clase.



Patrones

The Catalog of Design Patterns

https://refactoring.guru/design-patterns/catalog

Apps | Buenos dias | UNS-DCIC | En revision | Para revisar | Libros & Articulos | Recursos Electrónic | Biblioteca Electrónic | Proyectos | Otros | Other bookmarks

Hooray! After 3 years of work, I've finally released the ebook on design patterns! Check it out »

REFACTORING GURU

- Premium Content
- Refactoring
- Design Patterns
  - What is a Pattern
  - Catalog
    - Creational Patterns
    - Structural Patterns
    - Behavioral Patterns
  - Code Examples

Log in | Contact us

The Catalog of Design Patterns

Creational patterns

These patterns provide various object creation mechanisms, which increase flexibility and reuse of existing code.

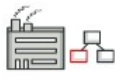
Structural patterns

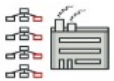
These patterns explain how to assemble objects and classes into larger structures while keeping these structures flexible and efficient.

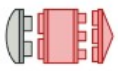
Behavioral patterns

These patterns are concerned with algorithms and the assignment of responsibilities between objects.

feedback


Factory Method


Abstract Factory

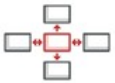

Adapter


Bridge


Chain of Responsibility


Command


Iterator


Mediator

https://refactoring.guru/design-patterns/chain-of-responsibility

2019-2do-tdp-Clase-11-Patr... | The Catalog of Design Patte... | Skype | 08:46

Patterns - Antipatterns

Los patrones de diseño proveen una forma efectiva de guía en el desarrollo de software. Formaliza la forma de pensar correcta en esta tarea.

Hay que conocerlos para poder usarlos.

23 Patrones GoF
33 Patrones de Buschmann
72 Patrones de análisis
38 Patrones de Diseño CORBA
...

Los anti-patrones (antipatterns) son un concepto relativamente nuevo en la ingeniería de software y una extensión natural a los patrones de diseño.

Pretenden ayudar en el proceso de desarrollo,
de forma similar a como lo hacen los patrones de diseño.

Los anti-patrones son soluciones negativas que presentan más problemas que los que soluciona.

Hay que conocerlos para poder evitarlos.

Antipatterns

Un anti-patrón es una forma literaria que describe una solución común y recurrente a un problema, que genera decididamente consecuencias negativas. (*Brown*).

Es una ilustración de la aplicación de lo que parece ser la solución correcta en el momento incorrecto.

Es una muestra de ciertas construcciones dañinas de software, lo cual es la antítesis de la idea propuesta por Gof.

Es mucho más que la consecuencia de malos hábitos.

Denotan usualmente problemas de desarrollo. El propósito de catalogar los antipatrones es que puedan ser reconocidos y remediados.

Pueden ser tan complejos como un diseño incorrecto de clases y objetos y tan simples como código mal estructurado.

El término antipatterns se origina con el trabajo de Mark Akroyd en 1996 titulado "AntiPatterns: Vaccinations Against Object Misuse"

Antipatterns

¿Por qué estudiar los antipatrones?

La idea no es tanto decir
“no hagas esto”
sino
“tal vez no sepas que haces esto... pero no funciona”

Permite reconocer problemas comunes y sus causas subyacentes.

Proveen un plan para revertir estas causas e implementar soluciones productivas.

Proveen un vocabulario común para la identificación de problemas y sus soluciones.

Antipatterns

Los patrones son abstracciones de experiencia previa.
Los patrones determinan un proceso de escritura deductivo:
identificación del problema,
identificación unívoca de la solución de acuerdo al contexto del problema.

Los antipatrones determinan un proceso de escritura iterativo:

identificar el antipatrón,
definir la solución,
definir síntomas y consecuencias,
revisar y elaborar.

La intención es advertir y tomar conciencia de situaciones y soluciones alternativas.

La idea de anti-patrón se extiende a varios niveles,
desde el código individual de una clase o módulo
hasta la organización humana que produce software...

Antipatterns - descripción

NO existe un estándar para la descripción de antipatrones, pero los siguientes son algunos items recurrentes:

AntiPattern Name – nombre descriptivo del antipatrón.

Also Known As – otros nombres por los cuales es conocido.

Most Frequent Scale –area en la cual el antipatrón es aplicable. *(micro-architecture, framework, application, system, enterprise, global/industry)*

Refactored Solution Name – nombre significativo describiendo la solución al antipatrón.

Refactored Solution Type – la categoría en la cual la solución refactorizada cae *(software, technology, process, or role).*

Root Causes – raíz (causa) del problema.

Unbalanced Forces – fuerzas (aspectos) desbalanceados en la causa del antipatrón. (*functionality, management of performance, resources, complexity, change, IT resources, or technology transfer).*

Anecdotal Evidence – evidencia anecdótica de casos y consecuencias del antipatrón en acción.

Antipatterns - descripción

Background – ejemplos del antipatrón.

General Form – diagrama o detalles adicionales acerca del antipatrón.

Symptoms and Consequences – sintomas y consecuencias generales.

Typical Causes – causas específicas que conjuntamente con la raíz del problema llevan a la creación del antipatrón.

Known Exceptions – instancias en las cuales el antipatrón es aceptable.

Refactored Solutions – explica cómo una solución refactorizada resuelve el desbalance de las fuerzas antes mencionadas.

Variations – variaciones comunes del antipatrón

Example – ejemplo descriptivo de cómo la solución refactorizada puede aplicarse.

Related Solutions – referencias cruzadas a otros antipatrones

Applicability to Other Viewpoints and Scales – explica cómo el antipatron corresponde con otros puntos de vista y/o escalas.

Antipatterns

Existe una gran cantidad de antipatterns en la literatura y en la web.

No existe un trabajo paradigmático como el de GoF al que podamos seguir para estudiar algún catálogo.

Es imposible que veamos todos en este curso, pero es importante revisar muchos de ellos y conocer las propuestas.
Entran en la misma órbita que los patrones, al enseñar por medio del ejemplo.

Si exploramos todos los antipatrones formalizados, muchos nos resultarán muy triviales y algunos, lamentablemente, muy familiares.

Mencionaremos algunos antipatrones populares.
Muchos más pueden ser encontrados en

http://c2.com/cgi/wiki?AntiPatternsCatalog
http://sourcemaking.com/antipatterns

y en otros libros (como *Bitter Java*).

Tipos comunes de antipatrones

Los antipatrones pueden agruparse en

Antipatrones del desarrollo de software
Describen situaciones usualmente candidatas a la refactorización.

Antipatrones de la arquitectura del software
Se enfocan en la estructura general del sistema y sus componentes.

Antipatrones de Administración del Proyecto
Se enfocan en aspectos relacionados con la administración del proyecto, de las personas involucradas y de la comunicación humana.

Existen otras clasificaciones. Por ejemplo, según Wikipedia.

Organizational anti-patterns
Project management anti-patterns
Analysis anti-patterns
Software design anti-patterns
Object-oriented design anti-patterns
Programming anti-patterns
Methodological anti-patterns
Configuration management anti-patterns

Antipatrones de desarrollo de software

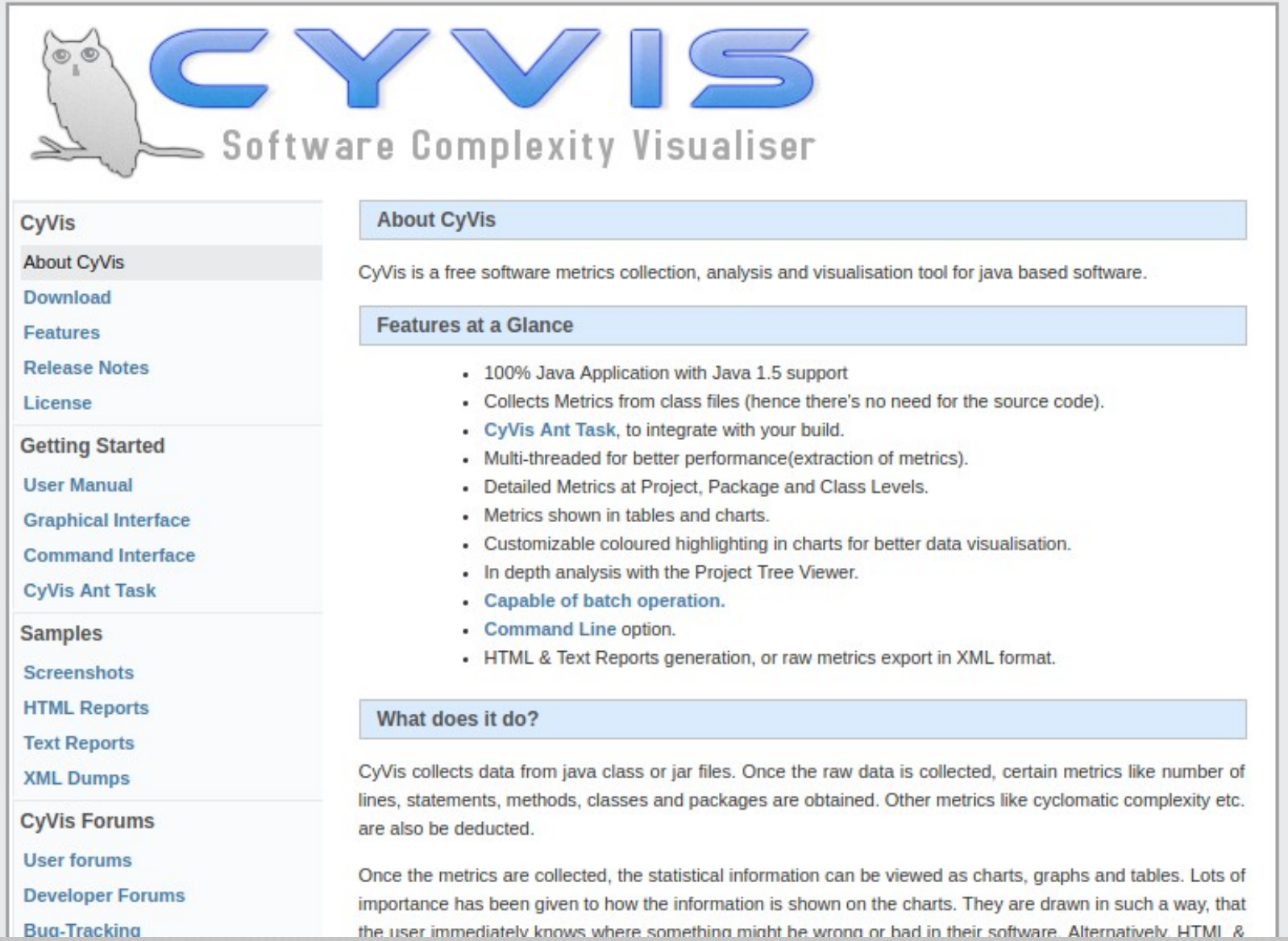
Spaghetti Code

Scale: *Application*
Descripción: código en el cual el flujo de control es muy variado y complicado (*spaghetti*). En OO, los objetos son básicamente funciones con poca interacción entre ellos.

Síntomas:
Un cuerpo de código realizando más de una función.
Es más fácil reescribir todo que modificar algo.
Falta documentación

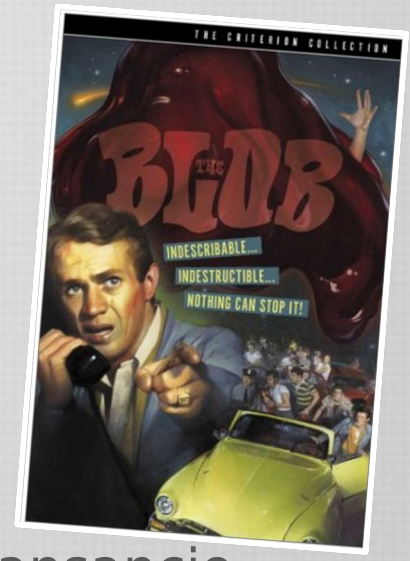
Causa: pereza, ignorancia, presión de tiempo de desarrollo.
Refactored Solution: código limpio, factorizado, funciones de alto nivel, jerarquía de objetos.

Leer ;)
“How To Write Unmaintainable Code”


CYVIS
Software Complexity Visualiser
CyVis
About CyVis
Download
Features
Release Notes
License
Getting Started
User Manual
Graphical Interface
Command Interface
CyVis Ant Task
Samples
Screenshots
HTML Reports
Text Reports
XML Dumps
CyVis Forums
User forums
Developer Forums
Bug-Tracking
About CyVis
CyVis is a free software metrics collection, analysis and visualisation tool for java based software.
Features at a Glance
100% Java Application with Java 1.5 support
Collects Metrics from class files (hence there's no need for the source code).
CyVis Ant Task, to integrate with your build.
Multi-threaded for better performance(extraction of metrics).
Detailed Metrics at Project, Package and Class Levels.
Metrics shown in tables and charts.
Customizable coloured highlighting in charts for better data visualisation.
In depth analysis with the Project Tree Viewer.
Capable of batch operation.
Command Line option.
HTML & Text Reports generation, or raw metrics export in XML format.
What does it do?
CyVis collects data from java class or jar files. Once the raw data is collected, certain metrics like number of lines, statements, methods, classes and packages are obtained. Other metrics like cyclomatic complexity etc. are also be deducted.
Once the metrics are collected, the statistical information can be viewed as charts, graphs and tables. Lots of importance has been given to how the information is shown on the charts. They are drawn in such a way, that the user immediately knows where something might be wrong or bad in their software. Alternatively, HTML &

Antipatrones de desarrollo de software

The Blob (o *God Class*)



Scale: *Application*
Síntomas:

Una clase *grande*, muchos métodos no relacionados
Muchos métodos sin argumentos.
"Esta clase es el corazón de nuestro sistema"

Causa: falta de experiencia en el diseño, vagancia, cansancio.
Consecuencias: pérdida de las ventajas de la OO. Difícil reuso o testeo.

Refactored Solution: Separar en pequeñas clases, evitar asociaciones transitivas, minimizar la complejidad del Blob quitando responsabilidades y repartiéndolas en otras clases en forma sensata.

Ejemplos: *Library_main_control*,
clase *Frame1* en algunos proyectos.

Antipatrones de desarrollo de software

Lava Flow

Existe código muerto u información de diseño olvidada que queda inalterada mientras el proyecto evoluciona.

Scale: *Application*
Síntomas:

- *Código que nadie entiende.*
- *Código "abandonado", comentado.*
- *Diseño no documentado.*
- *Código que no puede ser testeado.*

Causa: fracaso al aplicar un sistema de control de revisión. Programadores no experimentados tienden a conservar inútilmente código viejo.

Consecuencias: baja perfomance en el sistema. Dificultad para extenderlo o depurarlo.

Refactored Solution: Aplicar técnicas de revisión, eliminar código muerto, refactorizar.

Antipatrones de desarrollo de software

Lava Flow

```
// This class was written by someone earlier (Alex?) to manage the indexing
// or something (maybe).  It's probably important.  Don't delete.  I don't
// think it's used anywhere - at least not in the new MacroINdexer module which
// may actually replace whatever this was used for…
class IndexFrame extends Frame
{
   // IndexFrame constructor
   //=============================================================================
   public IndexFrame(String index_parameter_1)
   {
   // Note: need to add additional stuff here…
   super (str);
   }
   //-----------------------------------------------------------------------
…
```

Antipatrones de desarrollo de software

Golden Hammer

Una tecnología particular o producto es usada por todos pues el equipo de desarrollo obtuvo beneficios de esta forma.

Scale: *Application*
Alias*: Old Yeller, Head-in-the sand*
Síntomas:

- *Las soluciones encontradas son inferiores en calidad a otras existentes.*
- *La arquitectura del sistema se describe mejor en función de un producto particular.*
- *Nuevos desarrollos dependen fuertemente de la tecnología usada.*
- *Productos existentes dictan la estructura del diseño y del sistema*

Causa: excesiva dependencia y costumbrismo.
Refactored Solution: expandir el conocimiento de los desarrolladores constantemente (entrenamiento, libros, etc)

Antipatrones de la arquitectura del software

Reinvent de wheel

Un proyecto apenas reutiliza soluciones previas, aún cuando existan sistemas anteriores con funcionalidad solapada.

“Hecho 100% a medida“

Escala: Sistema

Alias: Greenfield Systems

Síntomas:

- *Repetición de funcionalidad de otro software comercial.*
- *Arquitecturas inmaduras e inestables.*
- *Arquitecturas cerrados, útiles acotadamente, sin previsión de reusabilidad.*
- *Inadecuado soporte para la administración de cambios e interoperabilidad.*

Causa: poca comunicación entre proyectos. Ausencia de procesos que incluyan conocimiento del dominio.

Antipatrones de la administración del proyecto

Analysis Paralysis

Ocurre cuando se busca la perfección y la completitud en la etapa de análisis, apenas avanzando en el diseño.

- *Frecuentes reinicios del proyecto.*
- *Aspectos de diseño e implementación se adelantan a la etapa de analisis.*
- *La etapa de análisis ya no tiene interacción con el usuario/cliente.*
- *El costo del análisis excede los tiempos estimados.*

Death by Planning

Ocurre cuando el proyecto se demora por excesiva dedicación a la preparación de planes para el desarrollo del proyecto.

- *Mucho tiempo de planificación.*
- *Demoras continuas en avanzar sobre la próxima etapa.*

Antipatrones de la administración del proyecto

Smoke and mirrors

Ocurre cuando el equipo se compromete a cumplir expectativas del cliente que escapan las capacidades tecnológicas de la organización.

El principal afectado es el cliente, que no obtiene lo esperado.

Es importante la correcta administración de las expectativas del cliente.