Patrones de Diseño

Agenda

- Origen
- Definición
- Tipos de Patrones
- Patrones esenciales
- Conclusiones

Christopher Alexander Levy

- **Christopher Alexander** acuño el término *lenguaje de patrón*. Lo usó para referirse a los problemas normales del diseño arquitectónico y civil. Su uso iba desde la forma de estructurar una ciudad a como debían disponerse las ventanas en una habitación. La idea se popularizó gracias a su libro ***A Pattern Language*.**

- El libro de Alexander ***The Timeless Way of Building*** (**El modo intemporal de construir**) describe qué significa para él lenguaje de patrón y cómo se aplica al diseño de edificios y ciudades. Sin embargo, este sistema es aplicable a cualquier otro campo del diseño.

Patrón de Diseño

"Cada patrón describe un problema que ocurre una y otra vez en nuestro entorno, para describir después el núcleo de la solución a ese problema, de tal manera que esa solución pueda ser usada más de un millón de veces sin hacerlo ni siquiera dos veces de la misma forma".

Cristopher Alexander

Video: Patterns in Architecture

Historia

- En 1987, ***Ward Cunningham y Kent Beck*** trabajaron con Smaltalk y diseñaron interfaces de usuario. Decidieron, para ello, utilizar alguna de las ideas de Alexander para desarrollar un lenguaje pequeño de patrones para servir de guía a los programadores de **Smaltalk.** Así dieron lugar al libro "***Using Pattern Languajes for Object-Oriented Programs***".

Definición de Patrón de Diseño

"Solución a un problema en un determinado contexto"

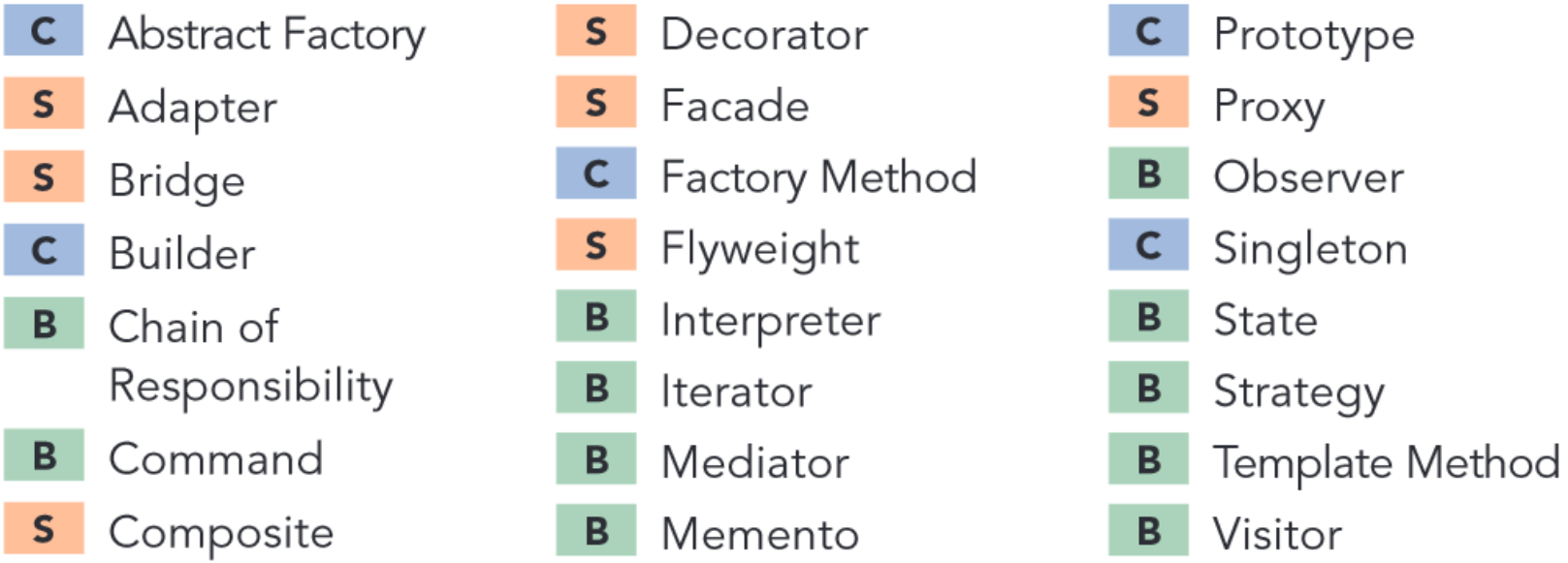
"Solución estándar para un problema común de programación"

- Un patrón es la abstracción de una forma concreta que puede repetirse en contextos específicos.
- Un patrón es una unidad de información nombrada, instructiva e intuitiva que captura la esencia de una familia exitosa de soluciones probadas a un problema recurrente dentro de un cierto contexto.

Patrones de Diseño en Programación

Tipos de Patrones de Diseño

- Patrones de creación: se utilizan para construir objetos de manera que puedan desacoplarse de su sistema de implementación.
- Patrones estructurales: se utilizan para formar estructuras de objetos grandes entre muchos objetos dispares.
- Patrones de comportamiento: se utilizan para gestionar algoritmos, relaciones y responsabilidades entre objetos.

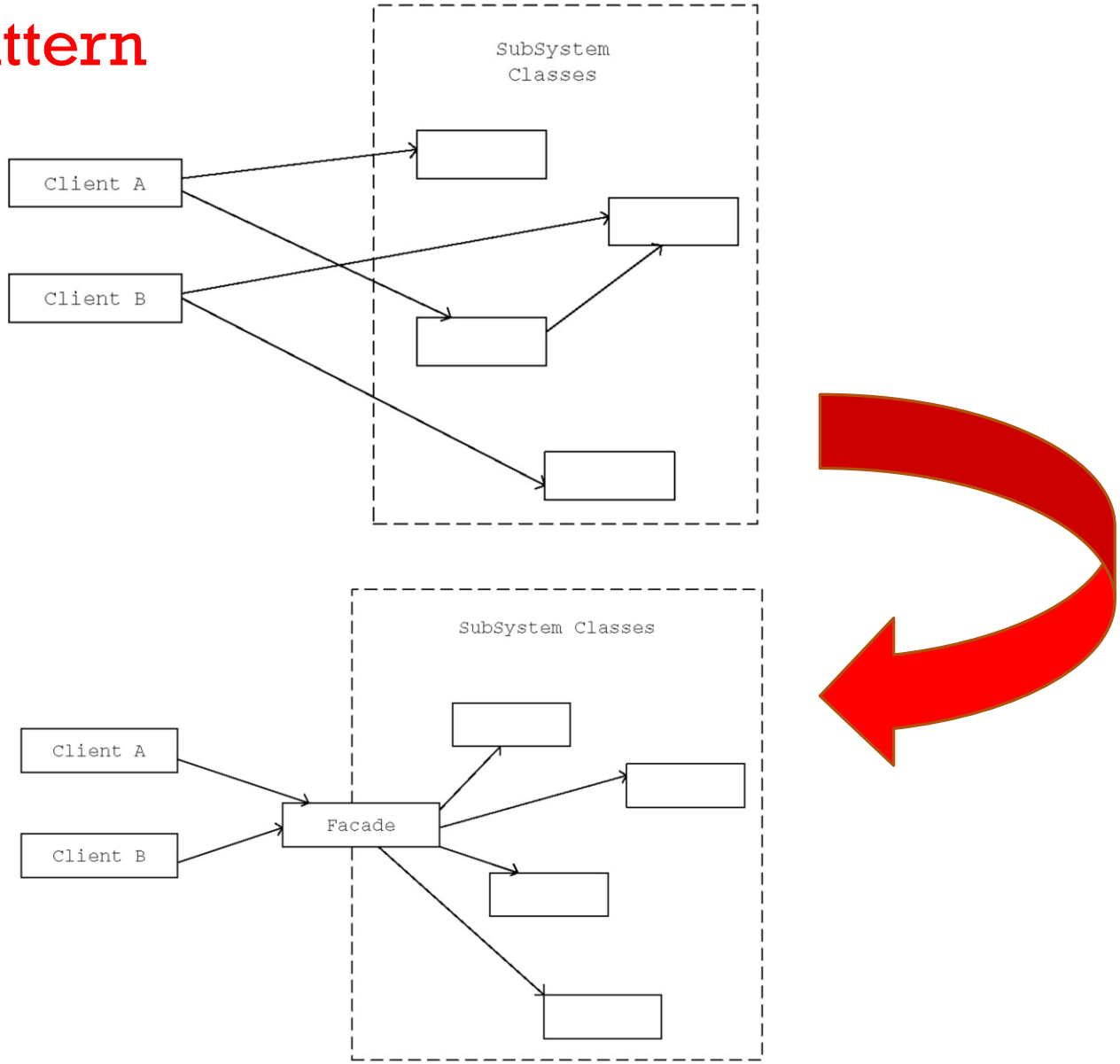


C=Creación **S=Estructural** **B=Comportamiento**

Características

- **Solucionar un problema**: los patrones capturan soluciones, no sólo principios o estrategias abstractas.
- **Ser un concepto probado**: los patrones capturan soluciones demostradas, no teorías o especulaciones.
- **La solución no es obvia**: muchas técnicas de solución de problemas tratan de hallar soluciones por medio de principios básicos. Los mejores patrones generan una solución a un problema de forma indirecta.

Facade Pattern



Facade (P. Estructural)

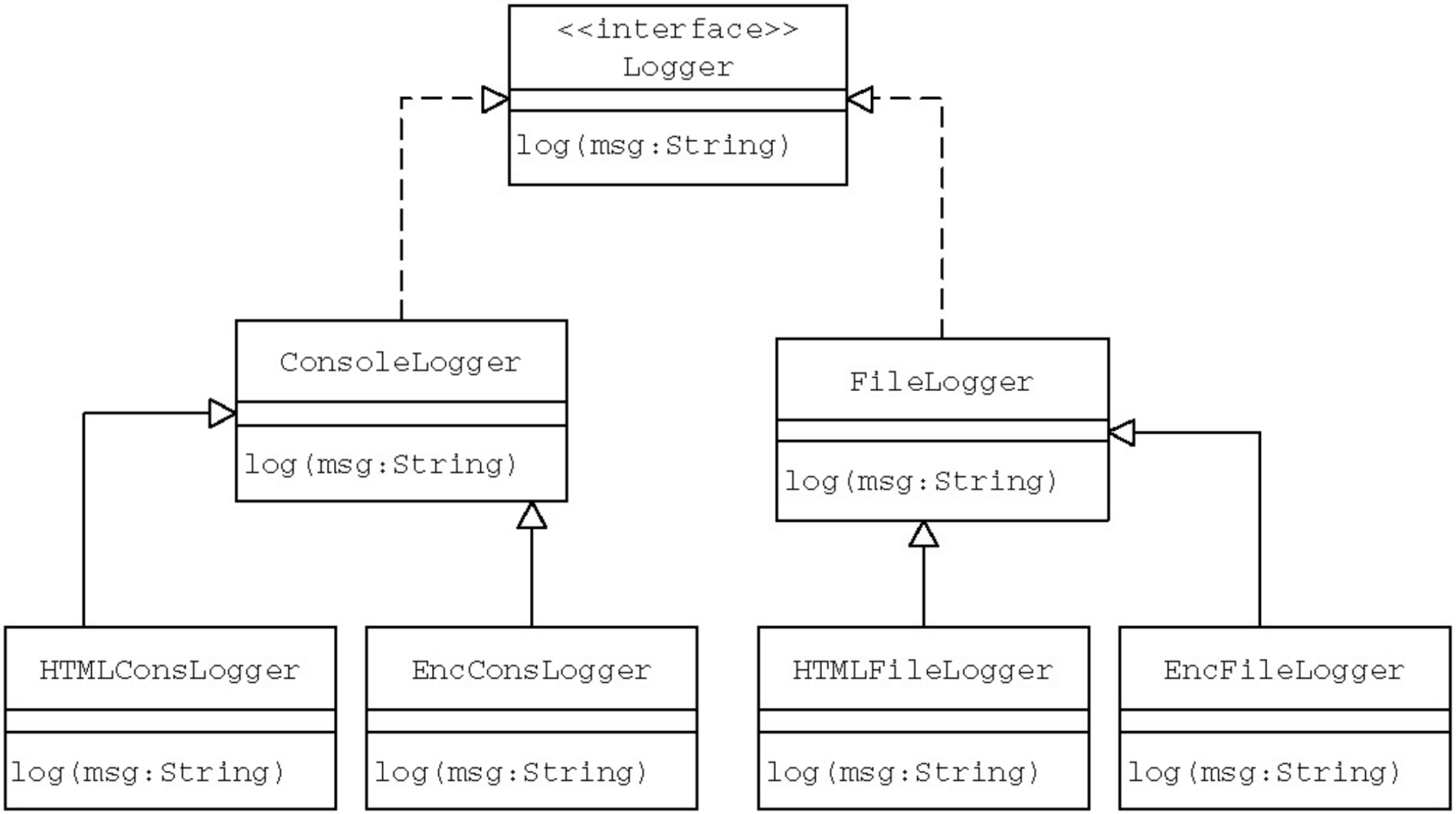
- El patrón de Fachada se ocupa de un subsistema de clases. Un subsistema es un conjunto de clases que trabajan en conjunto entre sí con el propósito de proporcionar un conjunto de funcionalidades (métodos) relacionados a un negocio.
- Por ejemplo, una clase de cuenta, una clase de dirección y una clase de tarjeta de crédito que trabajan juntas, como parte de un subsistema, proporcionan funcionalidades a un cliente en línea.
- Este tipo de interacción directa de los clientes con las clases del subsistema conduce a un alto grado de acoplamiento. Siempre que una clase de subsistema sufre un cambio, como un cambio en su interfaz, todas sus clases de cliente dependientes pueden verse afectadas.

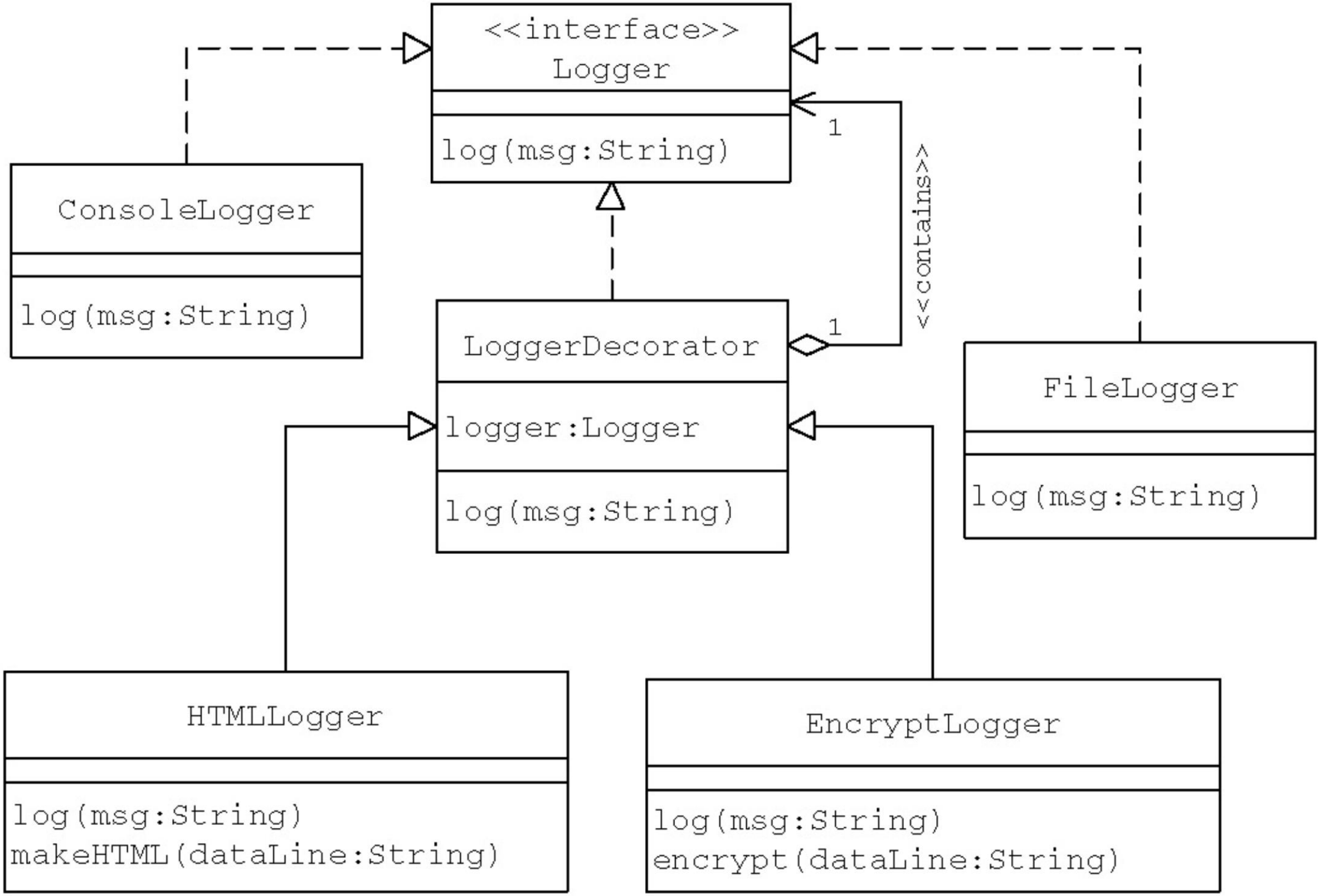
Facade (P. Estructural)

- Un objetivo de diseño común es minimizar la comunicación y las dependencias entre subsistemas.
- Que proporcione una interfaz única y simplificada a las instalaciones más generales de un subsistema.

Decorator (P.Estructural)

- El patrón de decorador se utiliza para ampliar la funcionalidad de un objeto de forma dinámica sin tener que cambiar el origen de la clase original o utilizar la herencia. Esto se logra creando una envoltura de objetos denominada Decorador alrededor del objeto real.


<<interface>>
Logger
log(msg:String)
ConsoleLogger
log(msg:String)
FileLogger
log(msg:String)
HTMLConsLogger
log(msg:String)
EncConsLogger
log(msg:String)
HTMLFileLogger
log(msg:String)
EncFileLogger
log(msg:String)


<<interface>>
Logger
log(msg:String)
ConsoleLogger
log(msg:String)
LoggerDecorator
logger:Logger
log(msg:String)
FileLogger
log(msg:String)
HTMLLogger
log(msg:String)
makeHTML(dataLine:String)
EncryptLogger
log(msg:String)
encrypt(dataLine:String)
1
1
<<contains>>

Singleton (P. Creación)

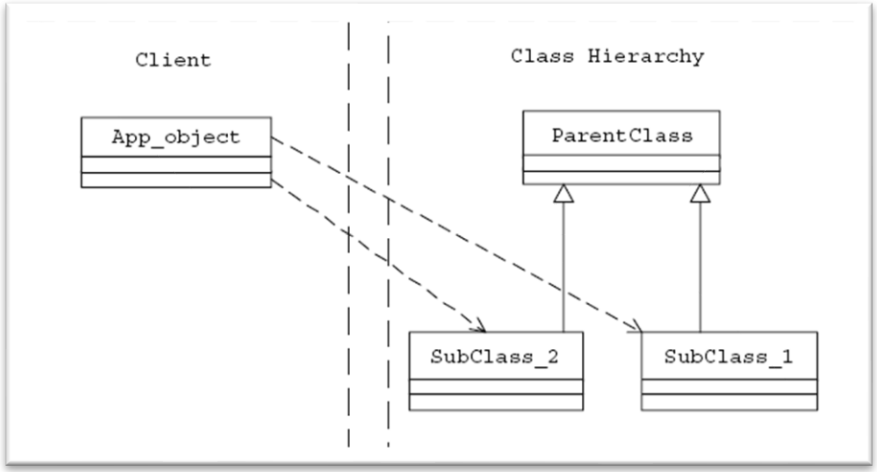
- El objetivo es tener una sola instancia de una clase en todo el sistema de tal manera que todas las clases puedan acceder a la misma instancia.
- 1 Sola instancia para la VM
- No se podrá instanciar directamente.

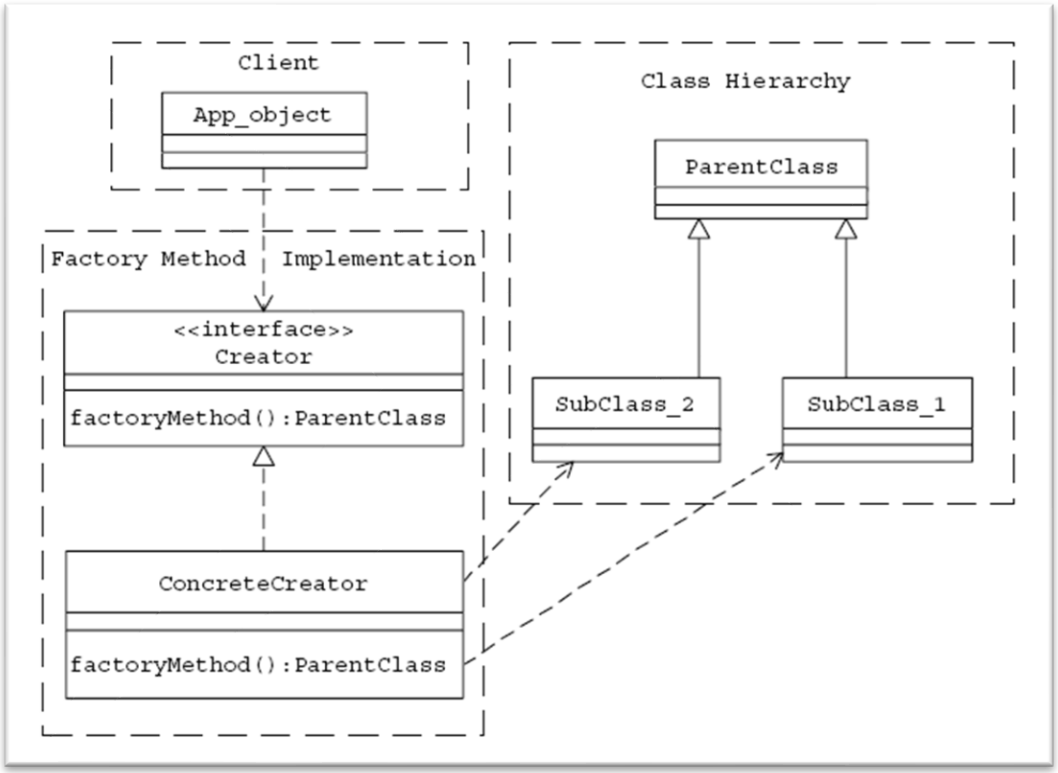
Recomendaciones:

- Tener cuidado con la concurrencia de usuarios que acceden a su métodos o propiedades.

Factory (P. Creación)

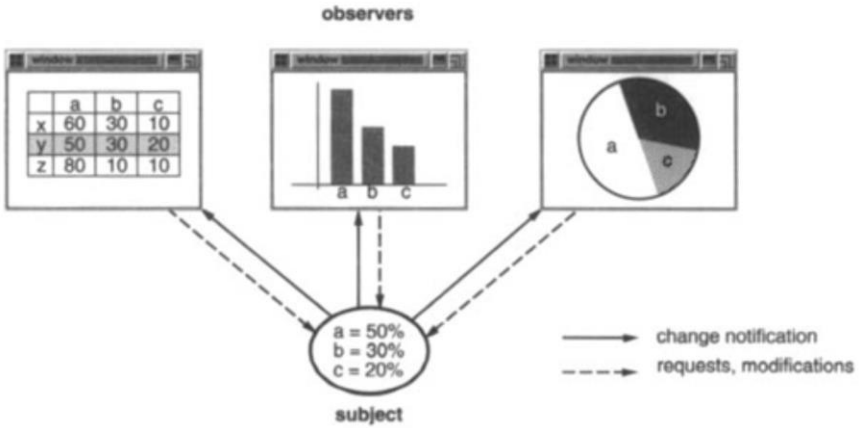
- Se usa bastante debido a su ahorro de recursos y utilidad en aplicaciones.
- Su objetivo es devolver una instancia de múltiples tipos de objetos, normalmente estos provienen de una misma clase padre, mientras que se diferencian entre ellos por algún aspecto de comportamiento.
- La idea es no estar usando new directamente a cada momento.
- Como se usa interfaces/herencia, esto reduce el acoplamiento.


Client
Class Hierarchy
App_object
ParentClass
SubClass_2
SubClass_1


Client
App_object
Class Hierarchy
ParentClass
Factory Method
Implementation
<<interface>>
Creator
factoryMethod():ParentClass
SubClass_2
SubClass_1
ConcreteCreator
factoryMethod():ParentClass

Observer (P. Comportamiento)

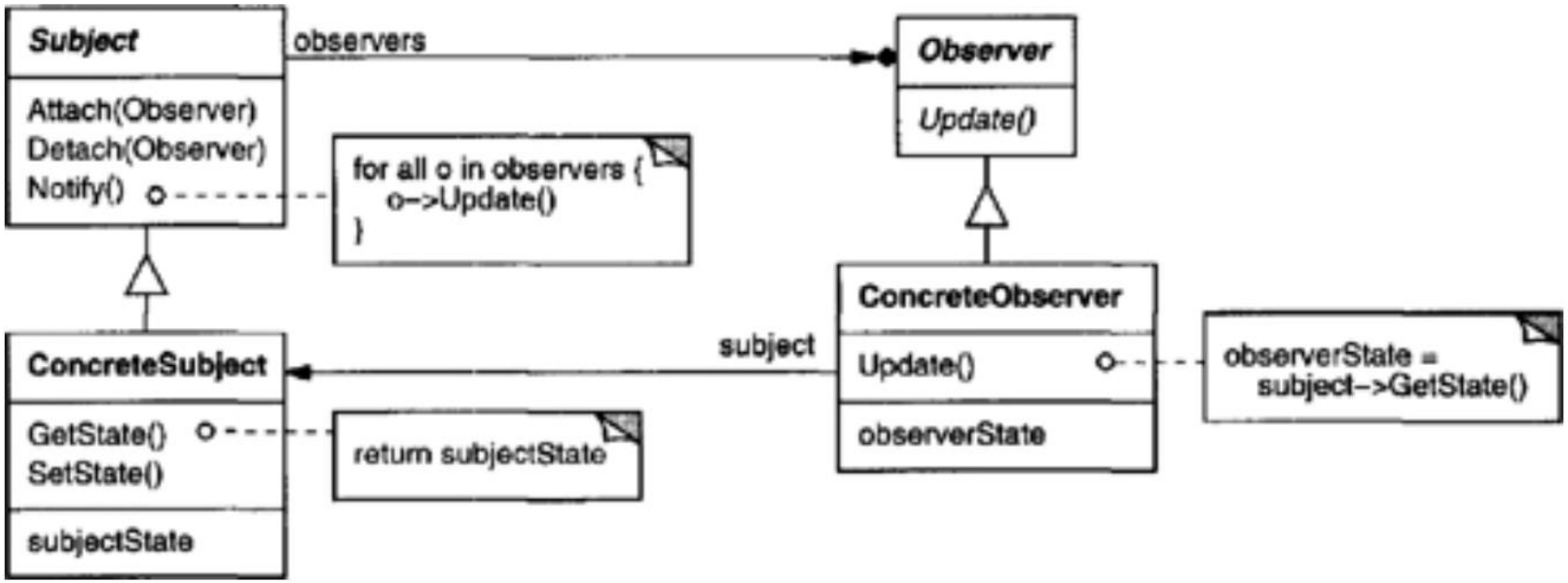
- **Intención**: Defina una dependencia de uno a muchos entre objetos para que cuando un objeto cambie de estado, todos sus dependientes sean notificados y actualizados automáticamente.



Observer

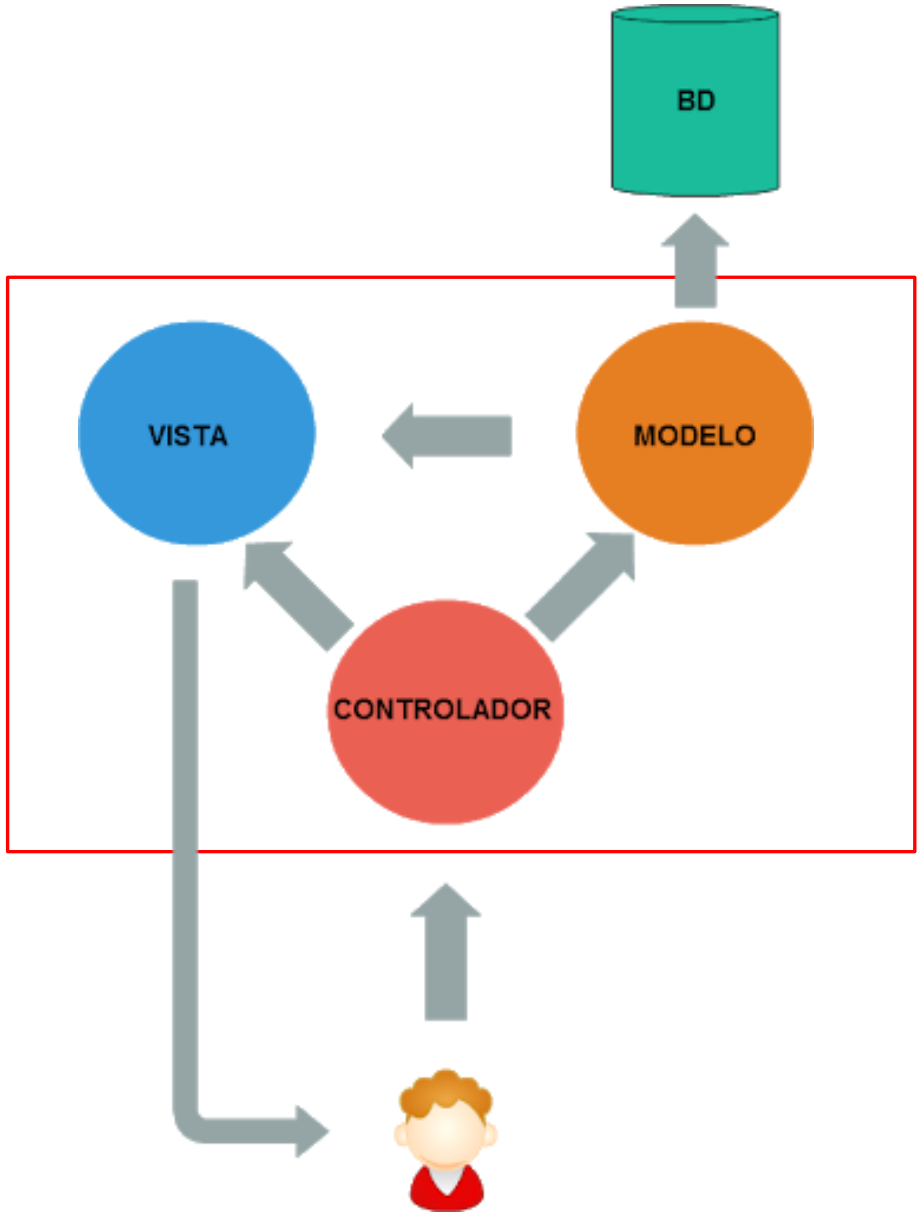
(P. Comportamiento)

- El patrón Observer es útil para diseñar un modelo de comunicación coherente entre un conjunto de objetos dependientes y un objeto del que dependen.
- Esto permite que los objetos dependientes tengan su estado sincronizado con el objeto del que dependen.
- El conjunto de objetos dependientes se denomina observadores y el objeto del que dependen se denomina sujeto.
- Para lograr esto, el patrón Observer sugiere un modelo de editor-suscriptor que conduce a un límite claro entre el conjunto de objetos. Observer y el objeto Subject.


Subject
observers
Attach(Observer)
Detach(Observer)
Notify()
for all o in observers {
o->Update()
}
Observer
Update()
ConcreteSubject
GetState()
SetState()
subjectState
return subjectState
subject
ConcreteObserver
Update()
observerState
observerState =
subject->GetState()

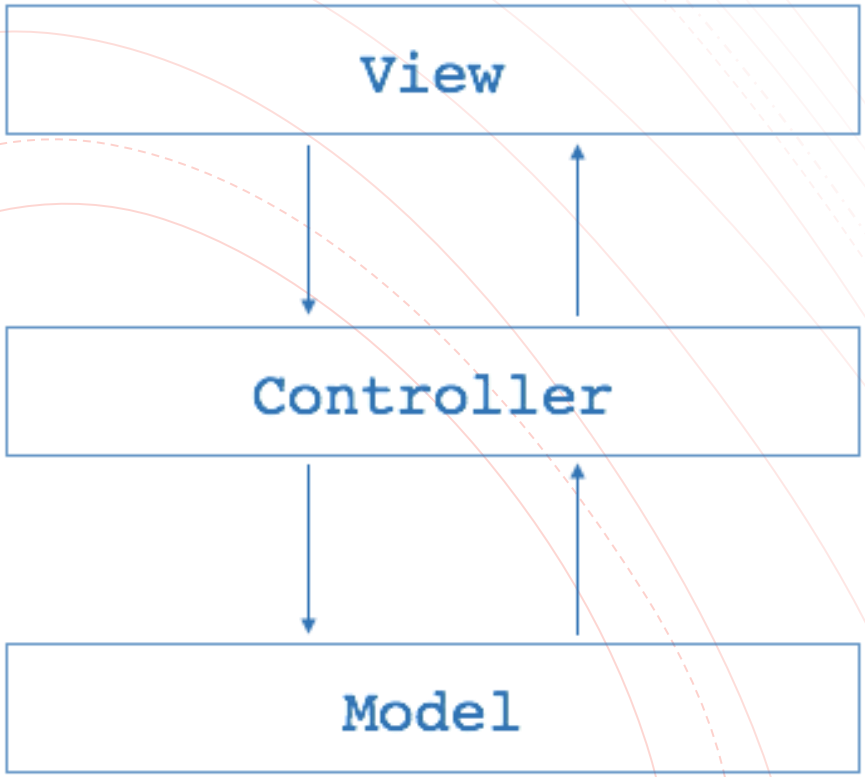

Model View Controller
(MVC)

MVC



MVC

- **Modelo**: Los objetos de modelo encapsulan los datos específicos de una aplicación y definen la lógica y el cálculo que manipulan y procesan esos datos. El modelo puede tener relaciones a uno y a muchos con otros objetos de modelo.
- **Vista**: Un objeto de vista sabe cómo dibujarse a sí mismo y puede responder a las acciones del usuario. Un propósito principal de los objetos de visualización es mostrar datos de los objetos del modelo de la aplicación y permitir la edición de esos datos, generalmente se desacoplan de los objetos del modelo.
- **Controlador**: Un objeto controlador actúa como intermediario entre uno o más de los objetos de vista de una aplicación y uno o más de sus objetos modelo. Los objetos controladores son, por tanto, un conducto a través del cual los objetos de vista aprenden sobre los cambios en los objetos del modelo y viceversa. Los objetos del controlador también pueden realizar tareas de configuración y coordinación para una aplicación y administrar los ciclos de vida de otros objetos.


View
Controller
Model

MVC

- Muchos objetos de estas aplicaciones tienden a ser más reutilizables y sus interfaces tienden a estar mejor definidas. Las aplicaciones que tienen un diseño MVC también son más fácilmente extensibles que otras aplicaciones. Además, muchas tecnologías y arquitecturas se basan en MVC y requieren que sus objetos personalizados desempeñen uno de los roles de MVC.

Conclusiones

- Proveen las "mejores prácticas" y "lecciones aprendidas" que deben ser suficientemente y formalmente documentadas, compiladas, escrutadas y ampliamente difundidas como patrones.
- Una vez que una solución ha sido expresada en forma de patrón, está podría ser aplicada y reaplicada a otros contextos, y facilitar ampliamente la reutilización.
- Los patrones exponen conocimiento sobre la construcción de software que ha sido fruto de muchos expertos durante muchos años.
- Todo diseñador de software debería ser capaz de utilizar patrones correctamente cuando construye sistemas software.
- Hace más sencillo el desarrollo y control de las aplicaciones.