División de Alta Tecnología

1

Sofware Design Patterns and Principles

Java Backend Developer I

Objetivos

- Conocer previamente los 3 paradigmas de la programación.
- Conocer los principios SOLID planteados por Robert C. Martin.
- Conocer los 6 patrones de diseño de software más importantes
- Explorar codigo fuente con los patrones GoF (laboratorios)

Agenda

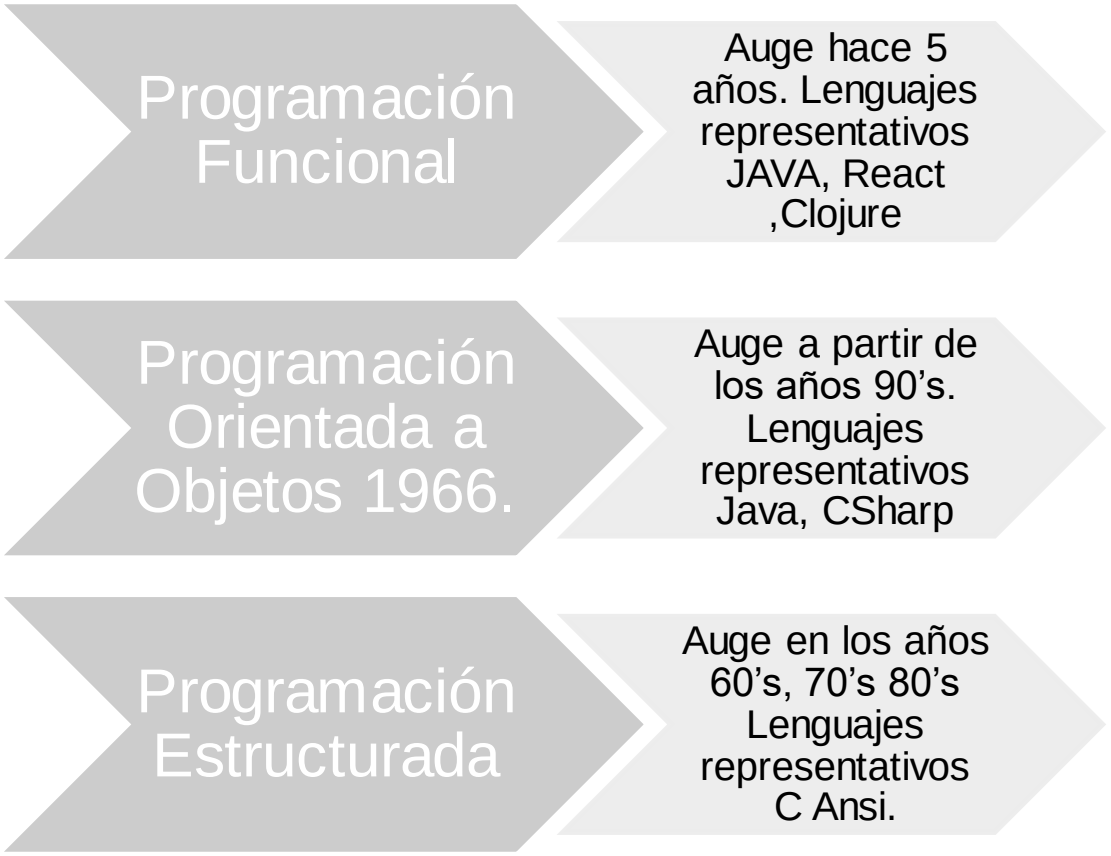
- Revisar los 3 paradigmas de la programación.
- Revisar los Principios de diseño SOLID
- Revisar los 6 patrones mas relevantes del GOF
  - Patrón Singleton
  - Patrón Factory Method
  - Patrón Iterator
  - Patrón Observer
  - Patrón Proxy
  - Patrón Bridge

Paradigmas de la Programación

- Un **paradigma** de **programación** es un estilo de desarrollo de programas.
- Representa un enfoque particular para construir software.
- Los lenguajes de **programación**, necesariamente, se encuadran en uno o varios **paradigmas** a la vez a partir del tipo de órdenes que permiten implementar, algo que tiene una relación directa con su sintaxis.
- A lo largo del tiempo se consideran tres paradigmas:
  - Programación estructurada
  - Programación orientada a objetos
  - Programación funcional.

Paradigmas de la Programación Cuando Surgieron

Paradigmas de la Programación Auge y Lenguajes Representativos



Programación Funcional

- Fue el primero en inventarse. Anterior a la programación de computadoras en sí.
- La programación funcional es el resultado directo del trabajo de Alonzo Church, quien en 1936 inventó el cálculo l. Su cálculo l es la base del lenguaje LISP, inventado en 1958 por John McCarthy.
- No tiene una declaración de asignación. La mayoría de los lenguajes funcionales, de hecho, tienen algunos medios para alterar el valor de una variable, pero solo bajo una disciplina muy estricta.
- La programación funcional impone disciplina a la asignación.
- Nota: Permite la inmutabilidad de las variables, con ello nos proporciona tener código seguro, no existe riesgo que las variables se modifiquen de manera arbitraria.

Programación Orientada a Objetos

- La programación orientada a objetos (POO) viene a innovar la forma de obtener resultados. Los objetos manipulan los datos de entrada para la obtención de datos de salida específicos.
- Esta basado en: herencia, cohesión, abstracción, polimorfismo, acoplamiento y encapsulamiento.
- Su uso se popularizó a principios de la década de 1990. En la actualidad, existe una gran variedad de lenguajes de programación que soportan la orientación a objetos.
- La programación orientada a objetos imponen disciplina a la transferencia indirecta de control.

Programación Estructurada

- Descubierta por Dijkstra en 1968
- Identifico que el uso de saltos (goto) es perjudicial para la estructura del programa.
- Reemplazo los saltos por sentencias (if, else, do while, until).
- La programación estructurada impone disciplina en la transferencia directa de control.

Paradigmas de la Programación
Conclusiones:

- Cada uno de los tres paradigmas de desarrollo más utilizados hoy en día impone una serie de restricciones a los desarrolladores:
  - La programación estructurada nos libra del uso directo de los punteros, de las instrucciones “goto”.
  - La programación orientada a objetos permite la inversión de dependencia gracias al polimorfismo.
  - La programación funcional nos brinda la no asignación de las variables.

Principios de diseño SOLID

- Robert C. Martin definió 5 principios de diseño de aplicaciones de software los cuales los basa en la programación orientada a objetos.
  - SRP: Principio de responsabilidad única
  - OCP: Principio abierto-cerrado
  - LSP: Principio de sustitución de Liskov
  - ISP: Principio de segregación de interfaz
  - DIP: Principio de inversión de dependencia

Principios de diseño SOLID

- Estos principios permiten:
  - Crear un software eficaz, que cumpla con su objetivo y que sea robusto y estable.
  - Escribir un código limpio y flexible ante los cambios: que se pueda modificar fácilmente según necesidad, que sea reutilizable y mantenible.
  - Permitir escalabilidad: que acepte ser ampliado con nuevas funcionalidades de manera ágil.
- Desarrollar software de calidad.
- Todo arquitecto y desarrollador de software debe conocerlo.

Principios de diseño SOLID
SRP: Principio de responsabilidad única

- Robert C. Martin nos señala que un módulo debe ser responsable ante un solo actor.
- Una clase debería tener una, y solo una, razón para cambiar. La razón para cambiar es la R de responsabilidad.
- Actualmente en la programación se puede señalar que este principio establece que una clase, componente o microservicio debe ser responsable de una sola cosa (es decir “decoupled” en inglés).
- Si por el contrario, una clase tiene varias responsabilidades, esto implica que el cambio en una responsabilidad provocará la modificación en otra responsabilidad.

Principios de diseño SOLID
OCP: Principio abierto-cerrado

- Robert C. Martin nos señala que un artefacto de software debe estar abierto para extensión pero cerrado para modificación.
- Actualmente en la programación se puede señalar que una clase debería ser capaz de extender el comportamiento de una clase, sin modificarla". En otras palabras: las clases que usas deberían estar abiertas para poder extenderse y cerradas para modificarse.
- También es importante tener en cuenta el Open/Closed Principle (OCP) a la hora de desarrollar clases, librerías o frameworks, para llevar un orden de clases extendidas.

Principios de diseño SOLID
LSP Principio de sustitución de Liskov

- La L deriva de Barbara Liskov quien planteo que: Las clases derivadas deben poder ser sustituidas por sus clases base.
- Significa que los objetos deben poder ser reemplazados por instancias de sus subtipos sin alterar el correcto funcionamiento del sistema o lo que es lo mismo: si en un programa utilizamos cierta clase, deberíamos poder usar cualquiera de sus subclases sin interferir en la funcionalidad del programa.
- Según Robert C. Martin incumplir el Liskov Substitution Principle (LSP) implica violar también el principio de Abierto/Cerrado.

Principios de diseño SOLID
ISP: Principio de segregación de interfaz

- En el cuarto principio de SOLID, Robert C. Martin sugiere: “Haz interfaces que sean específicas para un tipo de cliente”, es decir, para una finalidad concreta.
- En este sentido, según este principio, es preferible contar con muchas interfaces que definan pocos métodos que tener una interface forzada a implementar muchos métodos a los que no dará uso.
- Así, cada clase implementa las interfaces de la que realmente necesita implementar sus métodos. A la hora de añadir nuevas funcionalidades, esto nos ahorrará bastante tiempo, y además, cumplimos con el primer principio (Responsabilidad Única).

Principios de diseño SOLID
DIP: Principio de inversión de dependencia

- Último principio, el cual se puede enfocar en la siguiente frase: “Depende de abstracciones, no de clases concretas”.
- Según Robert C. Martin los módulos de alto nivel no deberían depender de módulos de bajo nivel. Ambos deberían depender de abstracciones.
- Las abstracciones no deberían depender de los detalles. Los detalles deberían depender de las abstracciones.
- El objetivo del principio consiste en reducir las dependencias entre los módulos del código, es decir, alcanzar un bajo acoplamiento de las clases.

Principios de diseño SOLID
Conclusiones

- Aplicar estos cinco principios puede parecer algo complejo y a la larga costoso, pero mediante la práctica se volverán parte de nuestra forma correcta de programar.
- Nos permite finalmente que nuestros programas sean más sencillo de mantener, pero no solo para nosotros, si no más aún para los desarrolladores que vengan después, ya que verán un programa con una estructura bien definida y clara.

Patrones de Diseño GOF
Concepto de Patrón

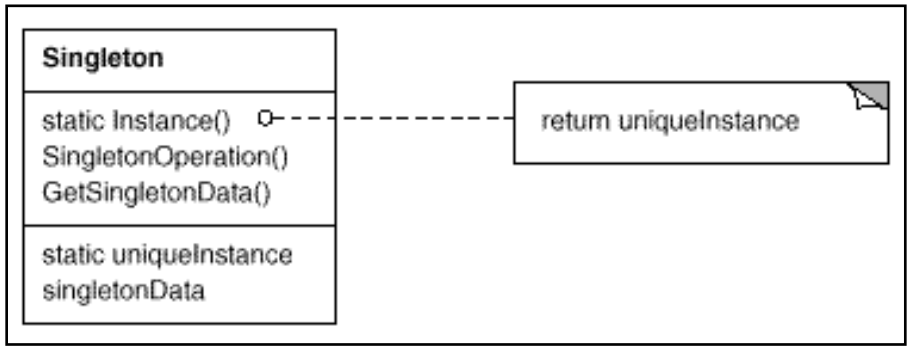
- Los patrones de diseño son unas técnicas para resolver problemas comunes en el desarrollo de software.
- Para que una solución sea considerada un patrón debe poseer ciertas características:
  - Debe haber comprobado su efectividad resolviendo problemas similares en ocasiones anteriores.
  - Debe ser reutilizable, lo que significa que es aplicable a diferentes problemas de diseño en distintas circunstancias.
- GOF los clasificó en: creacionales, estructurales y de comportamiento.

Patrones de Diseño GOF

- Patrones a revisar:
- Patrón Singleton (Creacional)
- Patrón Factory Method (Creacional)
- Patrón Iterator (Comportamiento)
- Patrón Observer (Comportamiento)
- Patrón Proxy (Estructural)
- Patrón Bridge.(Estructural)

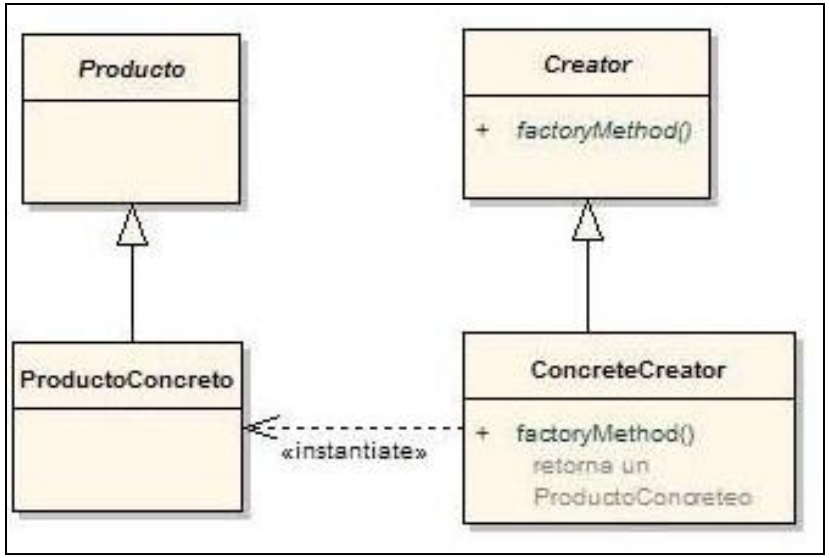
Patrones de Diseño GOF
Creacional: Patrón Singleton

- Patrón Singleton:
- Permite asegurar que una clase solo tenga una instancia y proporcione un punto de acceso global.
- Garantiza la existencia de una única instancia para una clase y la creación de un mecanismo de acceso global a dicha instancia. Restringe la instanciación de una clase o valor de un tipo a un solo objeto.



Patrones de Diseño GOF
Creacional: Patrón Factory Method

- Patrón Factory Method:
- Define una interfaz para crear un objeto, pero permita que las subclases decidan qué clase instanciar. El método Factory permite que una clase difiera la creación de instancias en subclases.

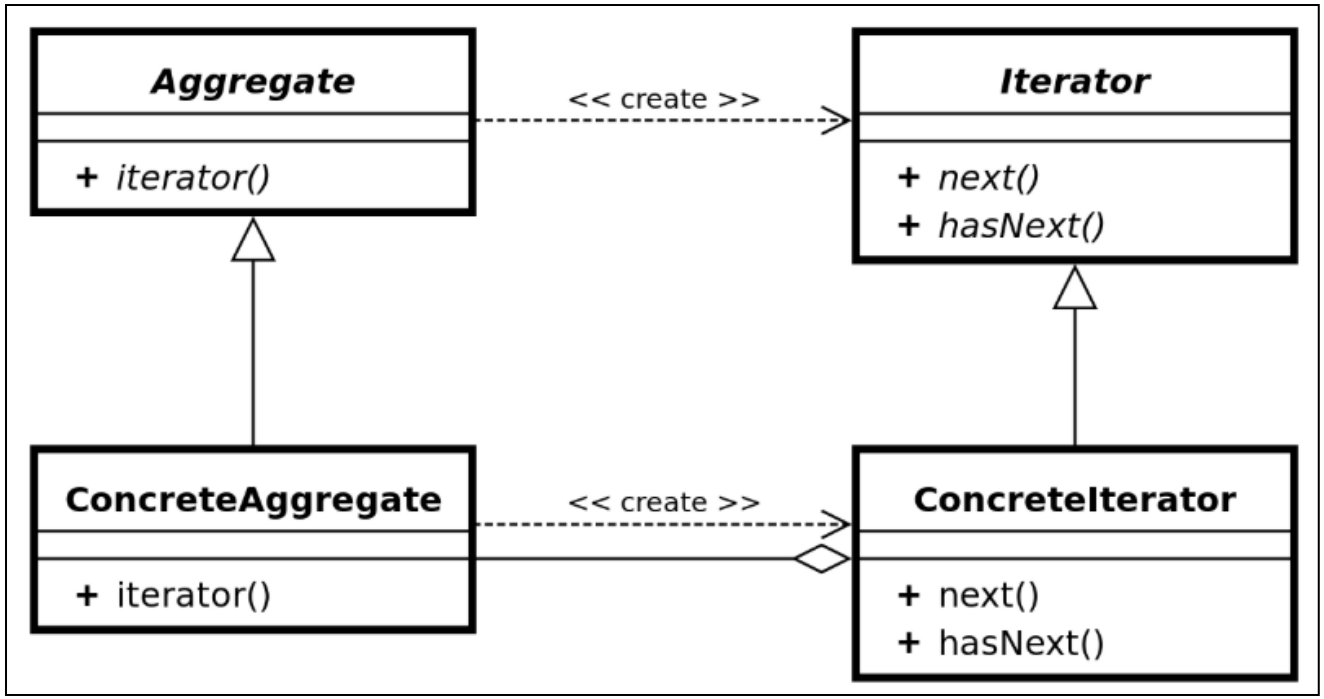


Patrones de Diseño GOF
Comportamiento: Patrón Patron Iterator

- Patrón Iterator:
- Nos permite recorrer los elementos de un conjunto sin importar cómo se representen internamente.
- En Java este concepto es familiar, ya que su librería estándar trae implementado este patrón. Todos los elementos de tipo lista poseén un método que devuelve un iterator, con el cual podemos trabajar cómodamente sobre colecciones.

Patrones de Diseño GOF
Comportamiento: Patrón Iterator

- Patrón Iterator:

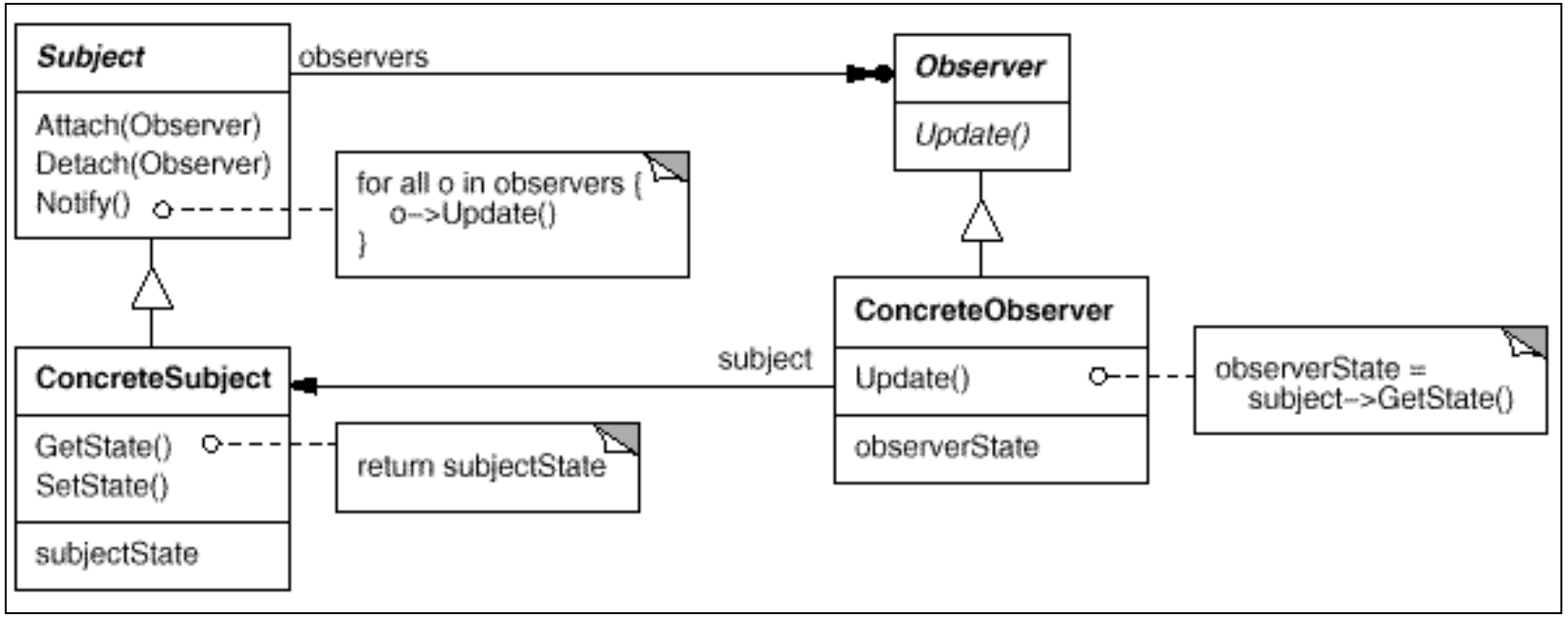


Patrones de Diseño GOF
Comportamiento: Patrón Observer

- Patrón Observer:
- Nos permite implementar una estrategia que reaccione a los cambios de estado en el objeto observado.
- Se identifica los siguientes actores en la implementación del patrón:
- Objetos observables: Deberán implementar un mecanismo para poder añadir y eliminar objetos observadores.
- Objetos observadores: Objetos que implementen la interfaz IObserver donde vendrá definido, al menos, un método que defina la reacción correspondiente al cambio de estado.

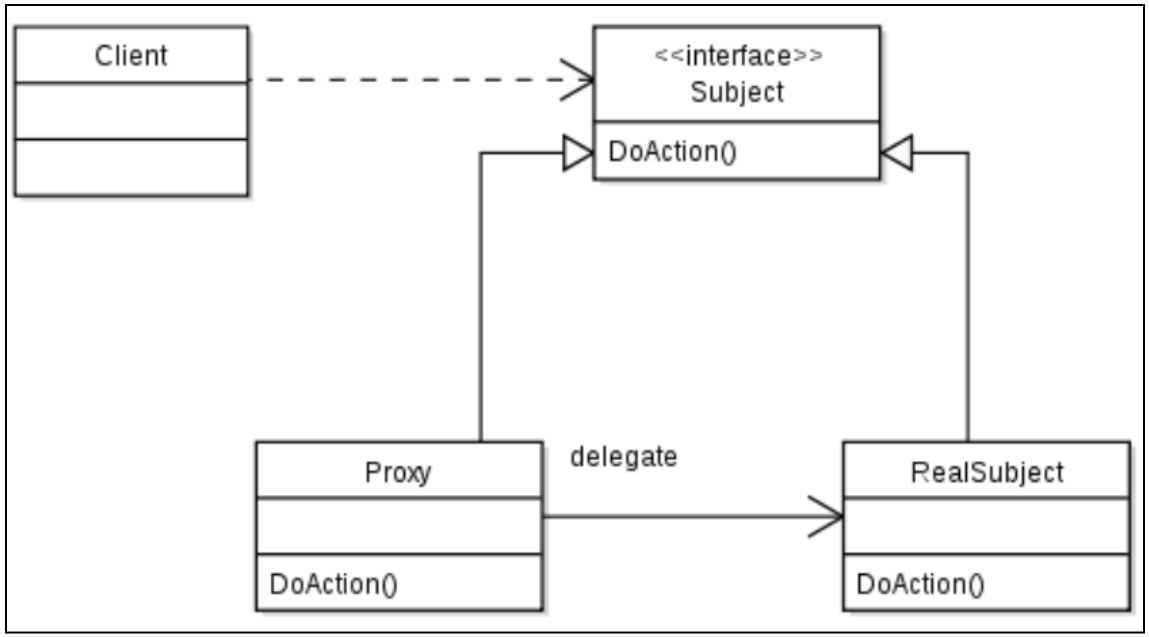
Patrones de Diseño GOF
Comportamiento: Patrón Observer

- Patrón Observer:



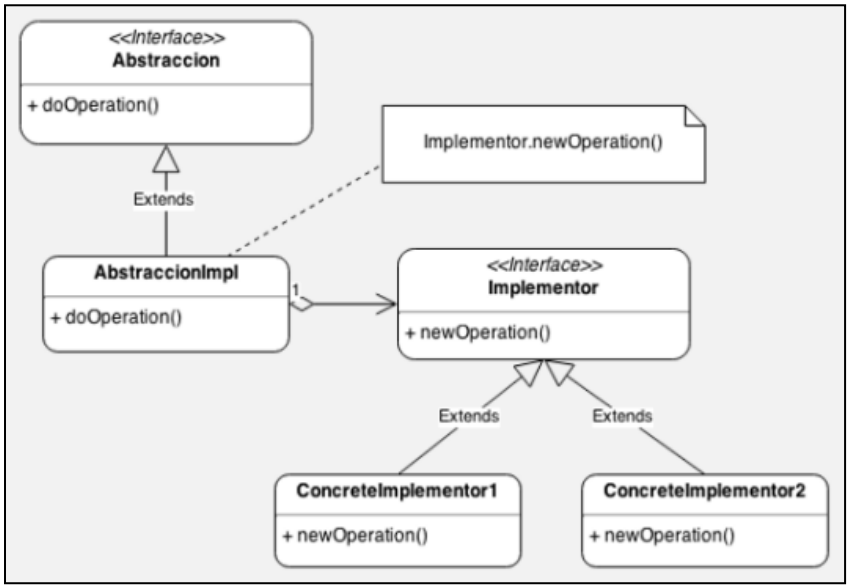
Patrones de Diseño GOF
Estructural: Patrón Proxy

- Patrón Proxy:
- Nos permite controlar el acceso a un objeto.
- Se debería proporcionar funcionalidad adicional al acceder a un objeto.



Patrones de Diseño GOF
Estructural: Patrón Bridge

- Patrón Bridge:
- Usada en programación para desacoplar una abstracción de su implementación, de manera que ambas puedan ser modificadas independientemente sin necesidad de alterar por ello la otra.



Lecturas adicionales

Para obtener información adicional, puede consultar los siguientes libros:

- Clean Architecture: A Craftsman's Guide to Software Structure and Design, First Edition
- Java Design Patterns: A Hands-On Experience with Real-World Examples

Resumen

En este capítulo, usted aprendió:

- Los 3 paradigmas de la programación.
- Los Principios de diseño SOLID
- Los 6 patrones mas relevantes del GOF
  - Patrón Singleton
  - Patrón Factory Method
  - Patrón Iterator
  - Patrón Observer
  - Patrón Proxy
  - Patrón Bridge

Tareas 1.1: Investigue los conceptos patrones de diseño de programación

Investigar y presentar 4 patrones adicionales del GOF

- Presentar un archivo Word de no más de 3 páginas explicando los patrones presentado en su tarea

Tareas 1.2: Desarrolle e implemente Patrones

Desarrollar 4 ejemplos donde utilice patrones para solucionar los problemas.

- Plantee 4 problemas a resolver.
- Presentar el código en JAVA que permita verificar el uso de los 4 patrones investigados en la tarea 1.1