Diagramas de clases y patrones de diseño

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(Información del tutor**

**Kevin Zeledón, 8487-6944,** [**magneticok@gmail.com**](mailto:magneticok@gmail.com)**)**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Etapa de análisis (Entender que es lo que quiero/quieren)

Etapa de modelaje/Diseño (Proponer una serie de soluciones para el problema)

Etapa de implementación (Programar el código)

# UML (Unified modeling language)

Se utiliza durante todo el ciclo de vida de desarrollo de software (SDLS). Pero se utiliza principalmente en la etapa de modelaje

Es una notación formal que utiliza la gente que desarrolla software.

## ¿para qué modelar software?

* Comunicar

Brinda la estructura y el comportamiento deseado del sistema que se va a construir

* Visualizar y controlar

Manejar la arquitectura del sistema en desarrollo (darle seguimiento, y manejar los cambios)

* Entender

Cuando se modela se entiende tanto el problema a resolver como la manera que se está implementando (Entender problema y solución(sistema))

* Gestionar Riegos

Identificar, analizar y prevenir (evitar) o aceptar riesgos (con la idea de mitigar) o delegar (asignar dicho riego a un tercero) el riesgo

No modelar es un riego que se podría correr para cosas pequeñas

Un modelo permite crear una abstracción, es decir representar algo que existe en la vida real. También puede ser considerado como una simplificación.

## Diagramas del UML

### Diagrama de clases (Vista estática)

Estático: porque no se puede ver el programa en tiempo de ejecución. Permite ver la solución al problema.

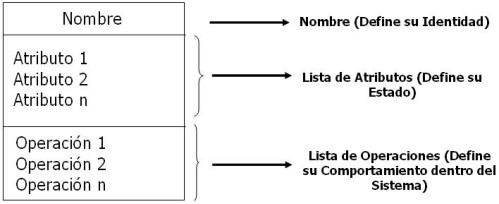
Incluye todas las estructuras de datos y todas las operaciones para solucionar el problema.

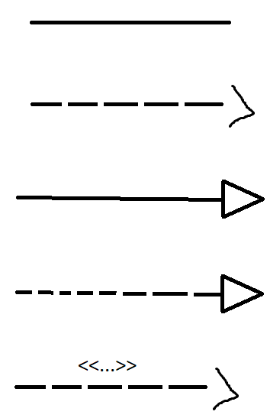
Incluye todas las clases que requiere la implementación de la solución.

Tiene clases con nombre, métodos y atributos, etc…

(modelo dominio: entender el problema, es más general, solo incluyen objetos que forman parte del problema)

Representación de una clase:



En un diagrama de clases, existen 5 tipos de relaciones:

Asociación (relación simple con un verbo)

Dependencia

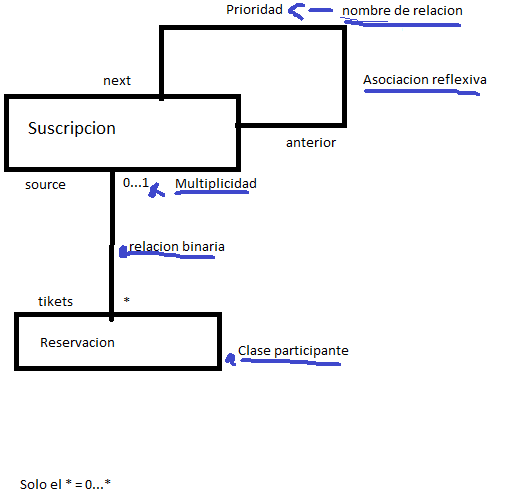
Generalización (de quien heredo, de hija a padre)

Realización

(es como dependencia, pero indirecta (algo como sobreescritura)

Uso (explica uso)

# Tipos de relaciones de asociación



La multiplicidad indica la cantidad posible de cada objeto.

La prioridad es un verbo simple que describe la acción que se realiza

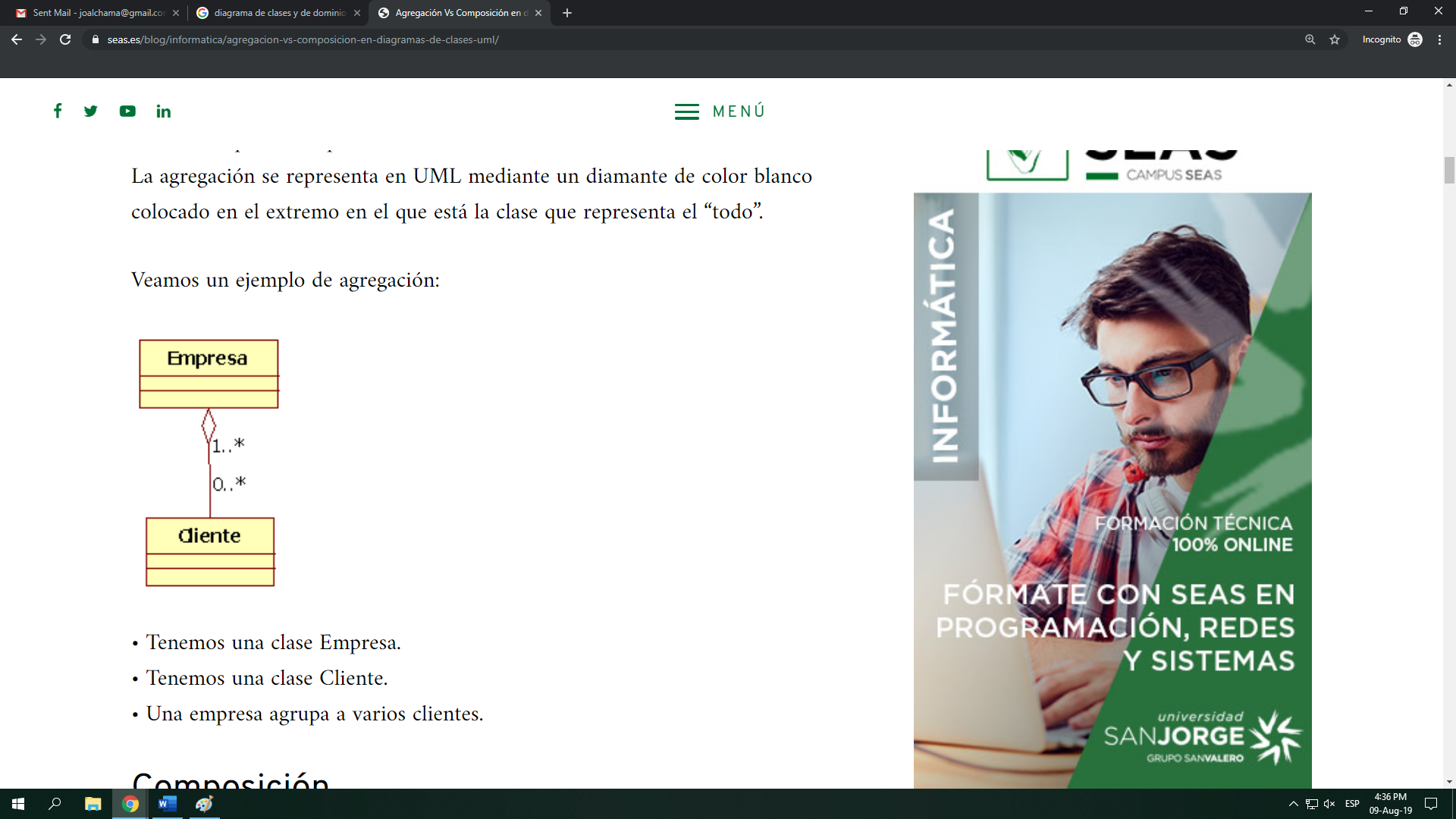
(ver imagen)

# Composición y agregación

Tienen un contenedor y los contenidos

Agregación: relación débil. Si el contenedor deja de existir los contenidos pueden seguir existiendo

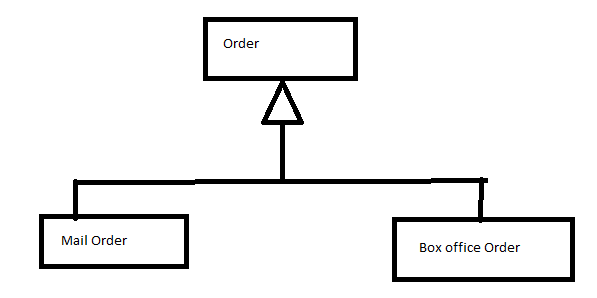
Composición: relación fuerte. Si el contenedor deja de existir los contenidos NO pueden seguir existiendo



Los clientes siguen existiendo si la empresa quiebra

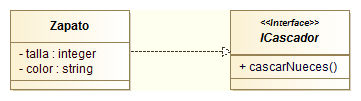


# Relaciones de generalización



Es una relación de herencia

# Realización



El tipo bloque de selección depende de lo que voy a pedir. (ejemplo de radio botón o check box)



# Forward engineering (normal) y reverse engineering

\*Programas para crear diagramas de clase\*

Microsoft Visio

Lucid Chart

IBM rational software architect

Diseño orientado a objetos

# ¿Qué es un buen diseño?

Que sea:

Eficiente: Que lo haga de la mejor manera (el contexto determina esta “mejor” forma)

Eficaz: Que cumpla con lo que se necesita

Siga buenas prácticas: para dar una sensación de correctitud en el trabajo

Pero algunos aspectos van a ser subjetivos

El diseño orientado a objetos comienza después el análisis, antes de iniciar el proyecto. Una vez que se tiene luz verde inicia el proceso (cuando hay plata).

# Modelado Conceptual

Es la fase de preguntas al cliente, todo el análisis inicial.

Comprender a lo que nos vamos a enfrentar desde una perspectiva general.

Un diagrama a un alto nivel (garabatos).

Un bosquejo conceptual del funcionamiento del ambiente en que se va a implementar el sistema

# Fase de Análisis

Una investigación a profundidad acerca del problema.

Aquí se pueden usar User Stories (“Scroom”)

Nace de los modelos agiles de desarrollo, basando se en la idea de “el cambio es lo único seguro”. Simplifican la documentación para dedicar tiempo a prototipos entregables.

Permite recibir constantemente retroalimentación del cliente

* Quien tiene la necesidad “yo como…, quiero…, para…” (quien, qué y para que)
* Analizar el impacto de cada cosa que necesita el cliente, e inclusive poder sugerir cambios, esto por tener un conocimiento más profundo de lo que se necesita.

Prototipo: Versión muy básica de la solución. Mostrar al cliente como se vería el producto que se quiere entregar.

# Fase de diseño

Implementar la información recolectada en las fases anteriores

El output final es el diagrama de clases, que sin embargo no va a estar listo, pues lo más probable es tener que agregar más clases o bien considerar clases de más

Se hace en tres etapas:

* Identificar clases:

Elaborar historias de usuario a partir de sustantivos, es decir conseguir las clases.

Cada clase debería tener una y solo una responsabilidad

* Identificar asociaciones:  
  Identificar como las diferentes clases interactúan entre ellas.

Si una clase no tiene relación con nadie, no debería existir.

Este proceso podría llevar a agregar o quitar clases

* Identificar atributos y métodos:

Marcar los adjetivos de la especificación (atributos)

Marcar los verbos de la especificación (métodos)

Patrones de diseño

Patrón: Regularidad discernible en la realidad

Al crear programas siempre se busca ajustar diseños o soluciones anteriormente creados para el problema de que se presenta.

(POO) Patrón: Solución general reutilizable a un problema común (“La forma de hacer la pizza”). Es además una buena práctica, es decir está probada.

La utilidad de los patrones reside en su capacidad de ser ajustado al contexto.

Primer catálogo (1994) (Patrones de diseño):

* Erich Gamma
* Richard Helm
* Ralph Johnson
* John Vissides

# Categorías

Alcance: Hacia que está orientado el patrón (Hacia clases o hacia objetos).

Clases 🡪 se encargan del proceso con el que se crean objetos

Objetos🡪Modifican a instancias de clases (Es decir se enfocan en el tiempo de ejecución)

Propósito: ¿Qué es lo que intenta hacer el patrón?

* Creaciones: Específicos para crear objetos
* Estructurales: Composición de clases y objetos (ponerlos a trabajar juntos) (para conseguir un resultado)
* Comportamiento: Comunicación entre objetos (para conseguir un resultado)

## Creacionales

Se encargan de abstraer el proceso de instanciación. Que el proceso mediante el cual se crean los objetos sea completamente independiente al resto de procesos que ejecuta el sistema.

* Encapsular el conocimiento acerca de cómo el sistema usa las clases concretas (no abstractas)
* Esconder los detalles de cómo se crean esas clases

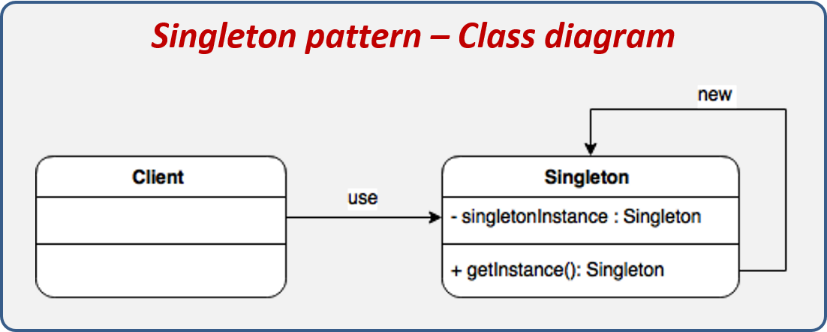
### Singleton

Que exista una única instancia de una clase. Con un método constructor que se pregunta si la instancia ya existe, si no la crea, si si entonces devuelve esa clase.

Colaboradores: una clase y el cliente

Cuando aplica: una única instancia, un único punto de acceso

Diagrama de clase:



### Abstract Factory

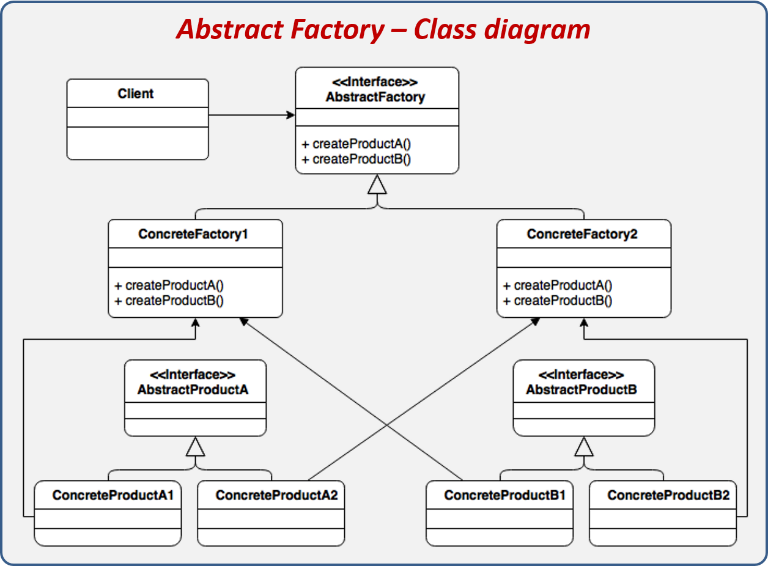
Proveer una interfaz para crear familias de objetos (ejemplo de los carros y ejemplo de enemigos) dependientes o relacionados mediante la especificación de clases concretas. Con el fin de simplificar el proceso de creación. “Líneas de ensamblaje”.

Colaboradores: un abstract factory, con dos concrete factories

Aplica: Cuando un sistema necesita independencia de como x objetos son creados

Configurar el sistema con múltiples familias de productos.

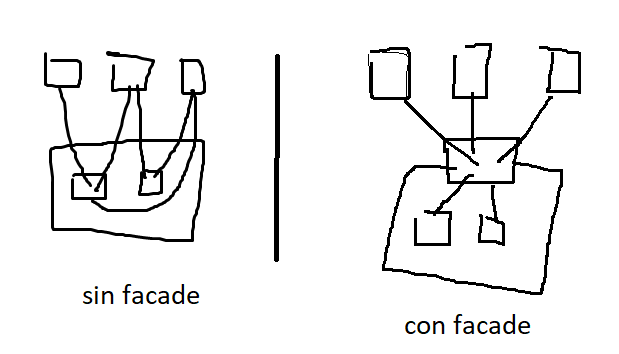
Diagrama de clase:



## Estructurales

### Facade (Fachada)

Establecer una interfaz (de alto nivel) unificada, para un conjunto de (sub)interfaces en un sistema complejo. En vez de interactuar con mucho, se interactúa solo con uno. Con el fin de simplificar la interacción con un sistema complejo. Los clientes no deberían tener acceso a los subsistemas



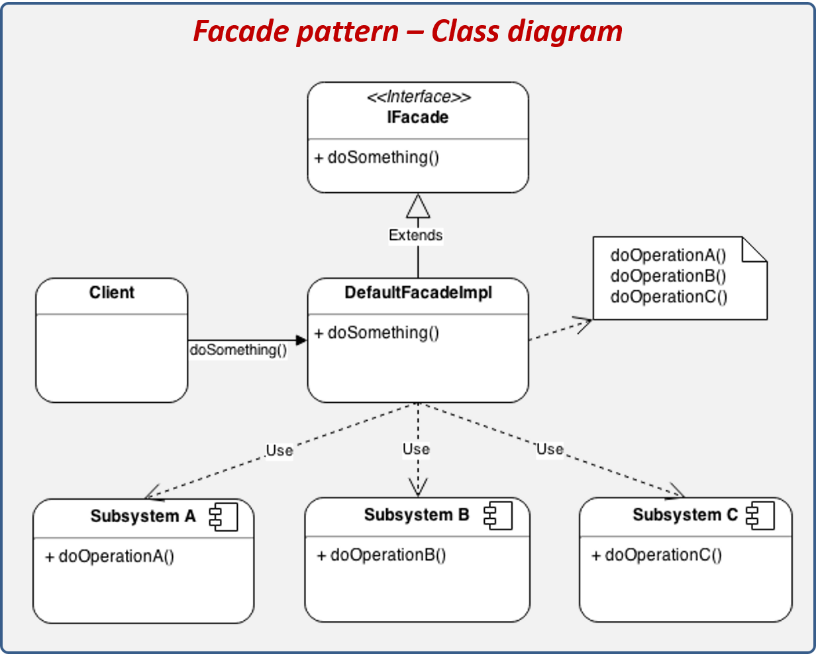
Cuando aplica: Cuando quiero darle una interfaz simple a un sistema complejo, que sea suficiente para el cliente. Cuando hay muchas dependencias. Un único punto de entrada al sistema.

El facade desacopla a las clases del sistema.

(acoplamiento: dependencia entre módulos (debería ser bajo) (entre módulos))

(cohesión: todas las partes están altamente relacionado (debería ser alta) (entre clases))

Diagrama de clase:



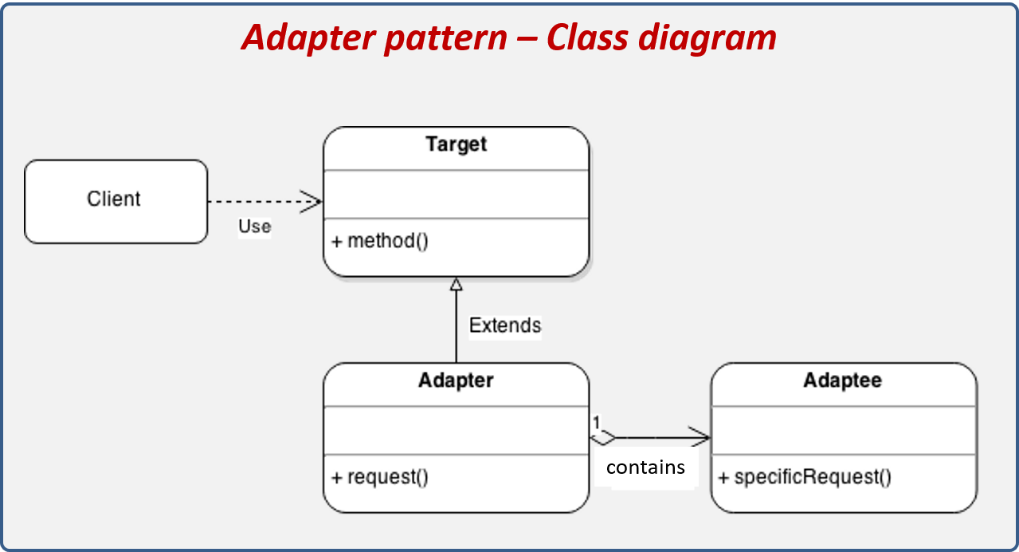
### Adapter

Convertir la interfaz de una clase a una interfaz distinta para que puedan relacionarse objetos que de otra forma no podrían comunicarse.

Colaboradores: Un cliente adaptado, un adapter y un cliente receptor

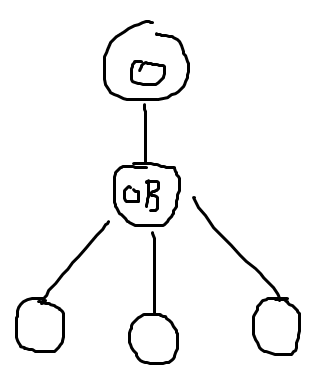
Aplica cuando: Tengo una clase que quiero que se comunique con otra con la que no se puede comunicar

Diagrama de clase:



## Comportamiento

### Observer

Establecer una relación de 1 a muchos. De tal forma que cuando algo pasa con el objeto observado le notifica al observer quien, notifica a todos los observadores. Los observadores toman una decisión según la infamación dada.

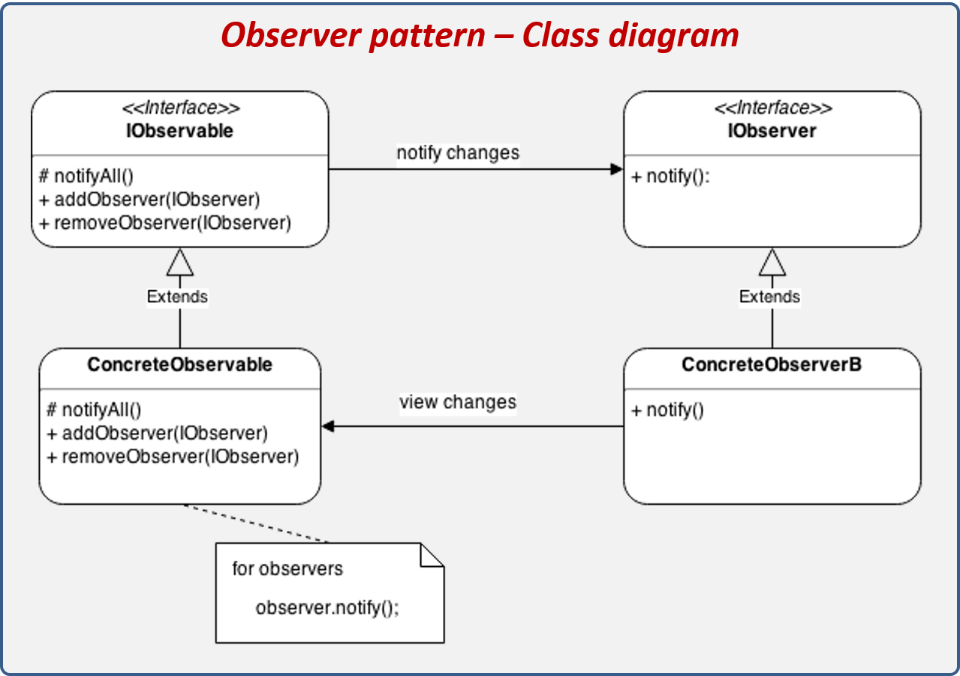
Colaboradores: Un objeto, un observer y observadores.

Aplica cuando: hay un objeto que depende de otros objetos

Permite una comunicación masiva. Siempre y cuando el observado sea quien notifique.

El observado no sabe cuando está siendo observado.

Diagrama de clase:



# Presentaciones

Sigleton: Una clase de una sola instancia. Patrón Creacional

Abstract Factory: Simplificar al cliente. Una clase que se Toda una familia de clases, que se encarga de crear objetos similares

Facade: Se encarga de simplificar la interacción con el cliente, reducir la comunicación entre el cliente y los subsistemas. Ej (la solicitud de beca) Una interfaz que simplifica la interacción entre múltiples sistemas. “Evitar la burocracia en un sistema”

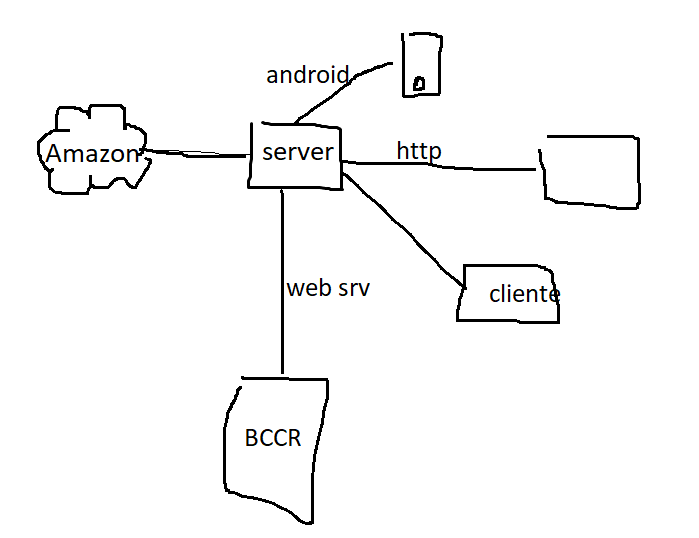
Observer: Crea objetos de tipo observador y un sujeto que notifica si uno de los objetos/clases (crea dependencia entre objetos) cambian. El sujeto notifica que uno de los observados cambios.

Adapter: Adatar clases incompatibles para que funcionen correctamente. Es un intermediario entre dos clases/objetos. Adaptado -> Adaptador

Patrones de Arquitectura

Arquitectura: las tecnologías que se van a emplear para darle solución a un problema.

Por ejemplo, WEB, un server que va a comunicar con varios clientes. Y usamos amazon cloud.

Esta a un nivel mucho más alto que el diseño (y lo patrones de diseño). Pues se genera primero.

# MVC

**Tres capas:**

Modelo, Vista, Controlador

Modelo: Base de datos, información almacenada. Que la información permanezca

Vista: Con lo que interactúa el usuario, GUI, gestiona los comandos del usuario

Controller: Recibe de la vista instrucciones, y toda la lógica necesaria, y da instrucciones al modelo.