Patrones de Diseño

M.C. Juan Carlos Olivares Rojas

Patrones de Diseño

- Son principios generales de soluciones que aplican ciertos estilos que ayudan a la creación de software.

- Es una descripción de un problema y la solución a la que le da el nombre y que se puede aplicar en nuevos contextos. Muchos patrones ayudan a asignar responsabilidades a los objetos.

Patrones de Diseño

- Un patrón es un par/problema solución con nombre que se puede aplicar en nuevos contextos, con consejos acerca de cómo aplicarlo en nuevas situaciones y discusiones sobre sus compromisos.

- “Un patrón de una persona es un bloque de construcción primitivo de otra”

Patrones repetitivos

- Los patrones sugieren algo repetitivo. No es su finalidad expresar nuevas ideas de diseño.

- Los patrones permiten codificar conocimientos, estilos y principios existentes y que se han probado que es valido.

Patrones

- Existen tanto a nivel arquitectura como de componentes.
- La Ingeniería del Software se encuentra en pañales en cuanto al uso de patrones.
- Se recomienda que tengan definidas las responsabilidades y colaboraciones, ejemplos de código.

Aplicaciones de patrones

- Generación de menús en cascada.
- Patrones para la generación de diálogos.
- Patrón de paginación.
- Patrón para la generación de aplicaciones portátiles

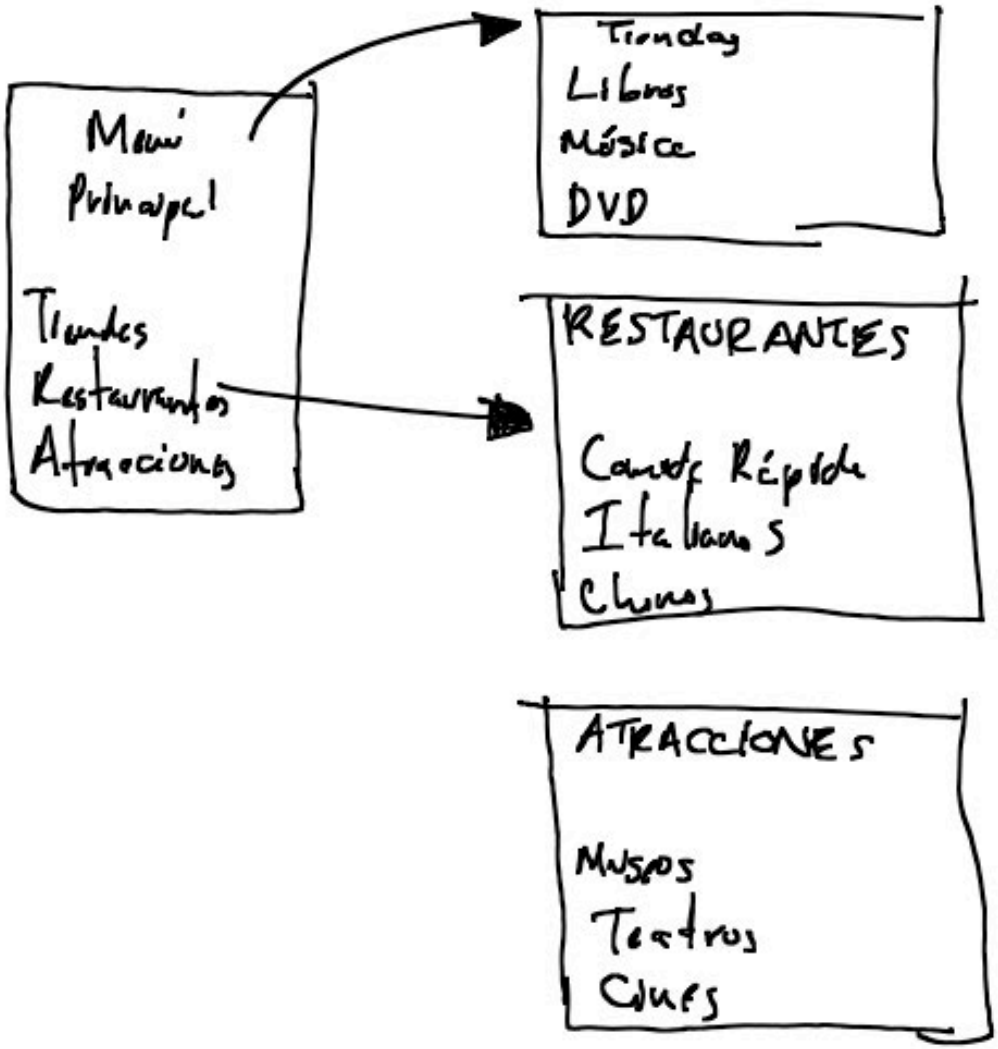
Patrones

- Son arquitecturas comprobadas para construir software orientado a objetos que sea flexible y se pueda mantener.
- Proveer la reutilización del diseño en sistemas posteriores.
- Ayudar a identificar los errores y obstáculos comunes que ocurren al crear sistemas.

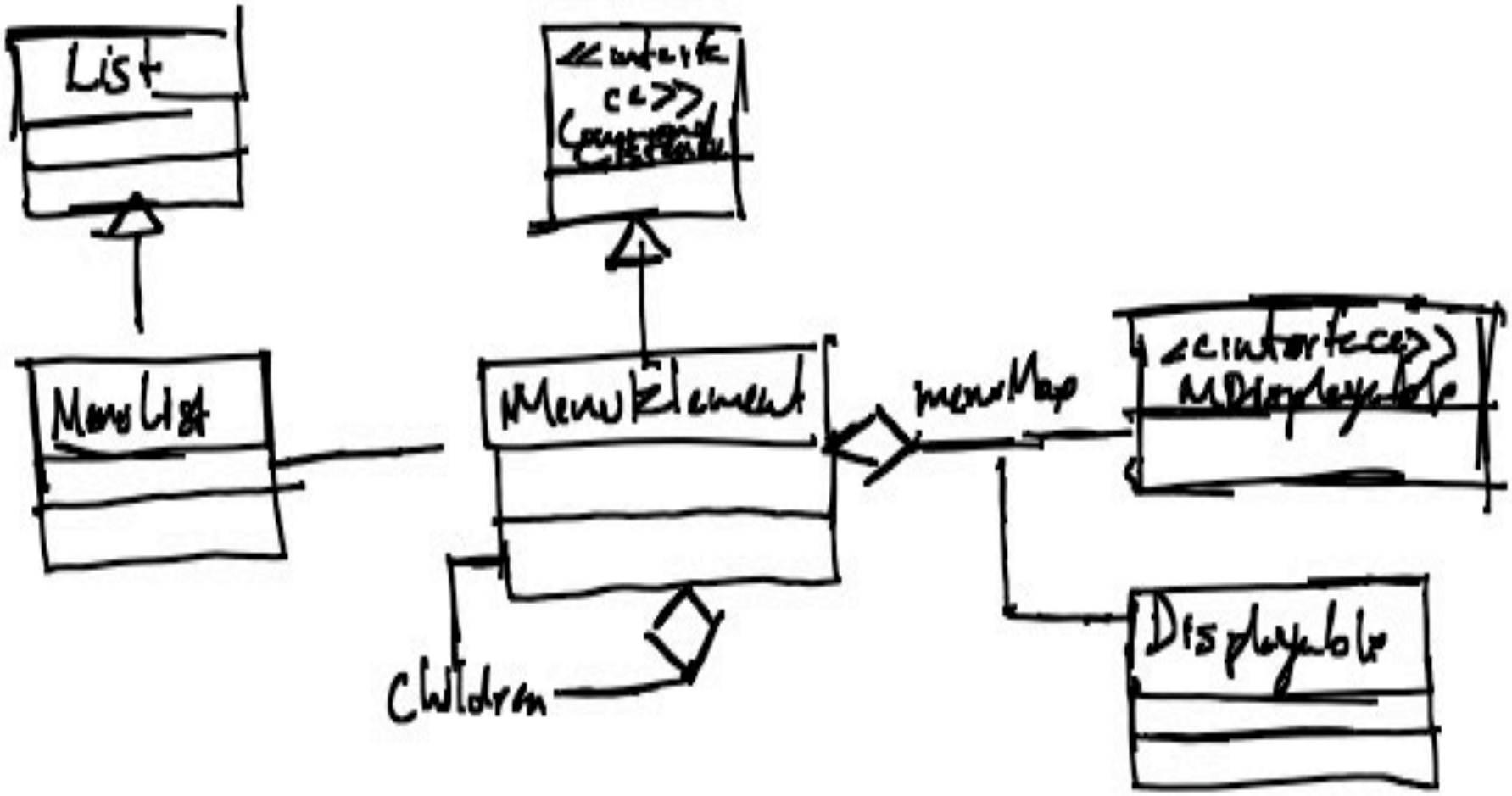
Patrones

- Ayudar a diseñar sistemas, independientes del lenguaje en el que se vayan a implementar.
- Establecer un vocabulario de diseño común entre los desarrolladores.
- Acortar la fase de diseño en un proceso de desarrollo de software.

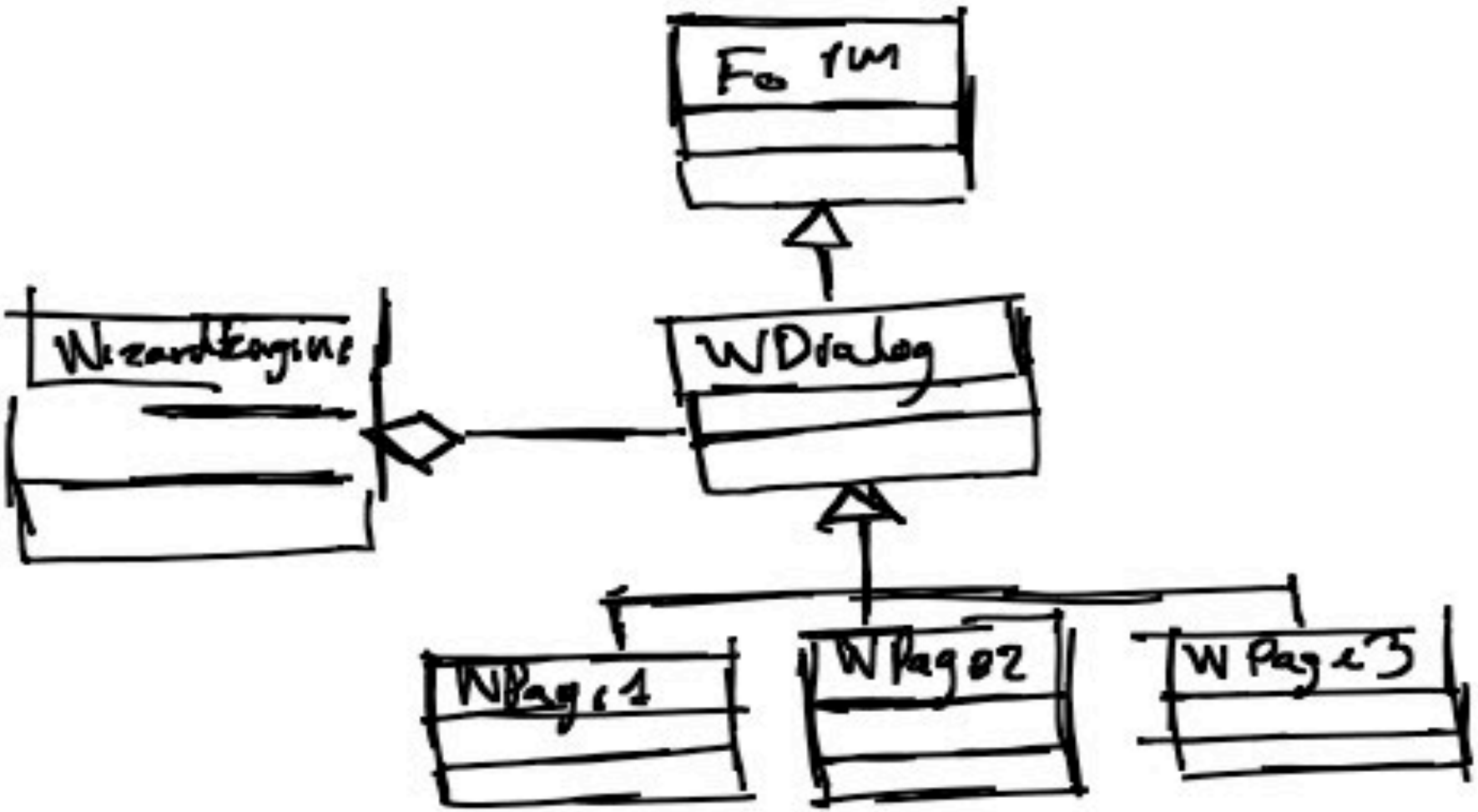
Ejemplo


Menú
Principal
Tiendas
Restaurantes
Atracciones
Tiendas
Libros
Música
DVD
RESTAURANTES
Comida Rápida
Italianos
Chinos
ATRACCIONES
Museos
Teatros
Clubs

Patrón para un menú desplegable

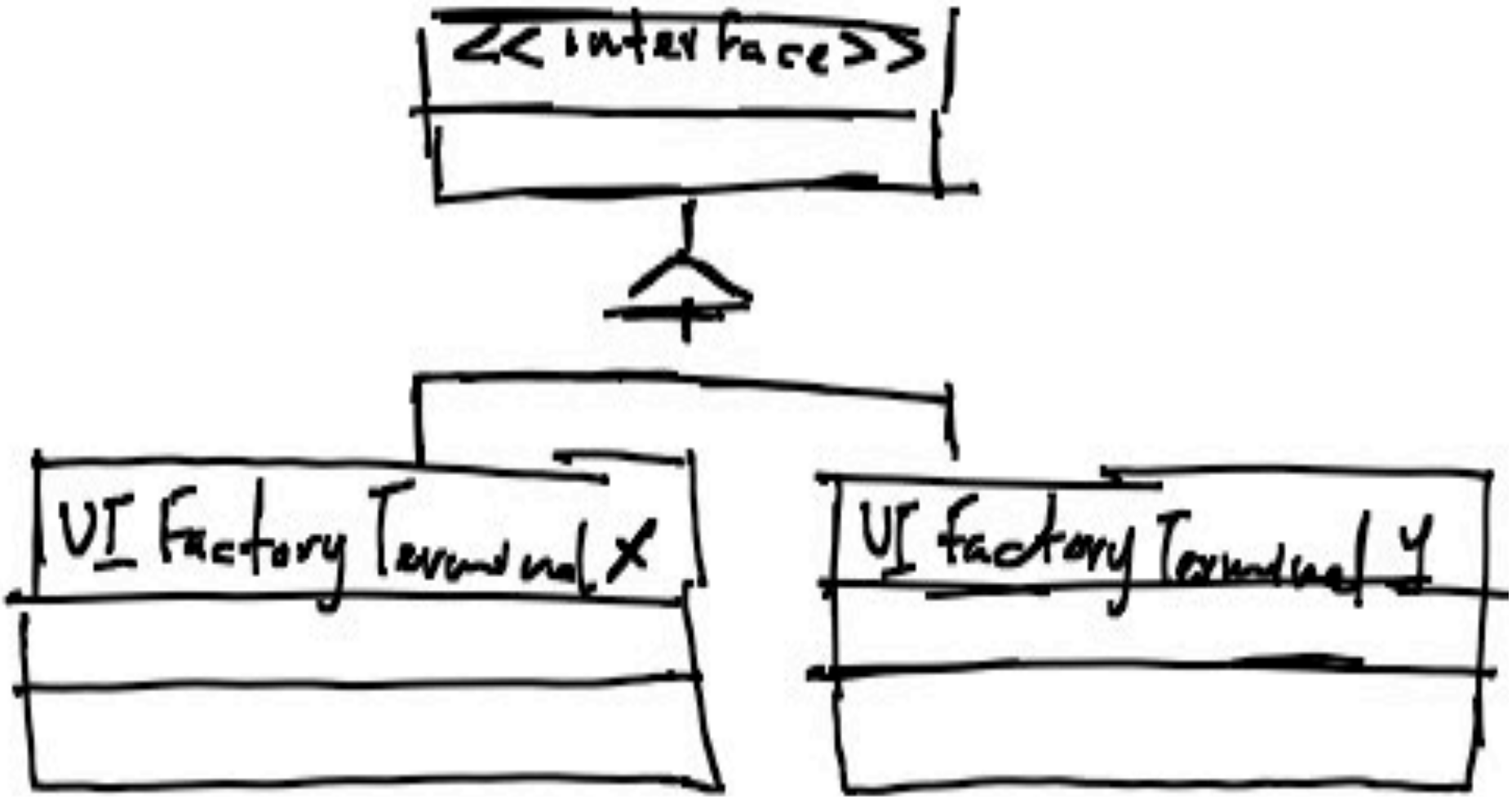


Patrón para páginas de interfaces



Herencia Simple

Interfaz genérica



Historia de los patrones

- Surgió con los patrones arquitectónicos (construcción) de Cristofer Alenxander (1977). Los patrones de software se originaron en los 80s con Kent Beck.

- Existen patrones como GRASP

Historia de los patrones

- De 1991 a 1994 Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides (banda de los cuatro).

- “La actividad más importante en la escritura de patrones es la reflexión”

Principio abierto-cerrado

- PAC descrito por Bertrand Meyer (1988) indica que los módlos deberían ser tanto abiertos (para extenderse: adaptable) como cerrados (ocultación de la información).

Arquitectura del software

- Conjunto de decisiones significativas sobre la organización del sistema de software, la selección d elos elementos estructurales y sus interfaces, junto con su comportamiento y colaboraciones, interfaces y su composición.

Tipos de patrones

- Patrones de arquitecturas: relacionados con el diseño a gran escala y de grano grueso; y que se aplican típicamente en las primeras iteraciones (fase de elaboración).

- Estilo: soluciones de diseño de bajo nivel orientadas a la implementación o al lenguaje.

Tipos de patrones

- Patrones de diseño: relacionados con el diseño de objetos y frameworks de pequeña y mediana escala. También se les llama patrones de micro-arquitectura

- Otros: patrones del proceso de desarrollo de software y organizacionales, patrones de interfaz de usuario, patrones de pruebas.

Patrones de arquitectura

- Capas (layers). Modelo de n niveles. Ejemplo: arquitectura lógica de un sistema de información. Capas: Presentación (UI, Vista, Interfaz), Aplicación (Flujo de trabajo, proceso, controlador), Dominio (servicios del negocio, modelo), Infraestructura del negocio (servicios de negocio de bajo nivel), Servicios técnicos (infraestructura técnica) y Base (servicios del núcleo, servicios básicos)

Patrones arquitectónicos

- Arquitectura de tres niveles:
- 1. Interfaz: ventanas, informes, etcétera
- 2. Lógica de la aplicación: tareas y reglas que dirigen el proceso.
- 3. Almacenamiento: mecanismos de almacenamiento persistente.

Patrones arquitectónicos

- La arquitectura de tres niveles se puede simplificar con un diseño en dos niveles.

- Si la interfaz y lógica de aplicación la lógica de la aplicación están en el cliente, dichos clientes se llaman gruesos.

- Si la interfaz únicamente se encuentra en el cliente, se les denomina clientes ligeros.

Patrones arquitectónicos

- Patrón MVC (Modelo-Vista-Controlador) se encarga de separar interfaces, en general el Modelo de la vista.

- Fue creado con Smalltalk-80.

- El modelo es la capa del dominio, la vista es la capa de presentación y el controlador son los objetos de flujo de trabajo.

Patrones GoF

- Gang-of-Four (“pandilla de los cuatro”) descritos en el libro Design Patterns (Gama 1995) definieron un ctaálogo con 23 patrones básicos.

Adaptador

- Problema: Resolver interfaces incompatibles, o proporcionar una interfaz estable para componentes parecidos con diferentes interfaces

- Solución: convertir la interfaz original de un componente en otra interfaz, mediante un objeto adaptador intermedio.

Factoría

- Principio: diseñe para mantener una separación de interfaces (separation of concerns)

- Los objetos factoría separan las responsabilidad de la creación compleja de objetos de apoyo, ocultan la lógica de creación de la parte compleja.

Factoría

- Problema: ¿Quién debe ser el responsable de la creación de los objetos cuando existen consideracioes especiales como una lógica de creación compleja, el deseo de separar las responsabilidades de la creación para mejorar la cohesión, etc.

- Solución: crear un objeto fabricación pura denominado factoría que maneje la creación.

Fábrica

- Su propósito es crear objetos, permitiendo al sistema identificar que clase se debe instanciar en tiempo de ejecución.

- Adaptador: proporciona un objeto con una nueva interfaz que se adapta a la interfaz de otro objeto, permitiendo que ambos objetos colaboren entre sí. Similar a los enchufes eléctricos.

Fábrica abstracta

- Permite a un sistema determinar la subclase a partir de la cual se va a instanciar un objeto en tiempo de ejecución. A menudo, esta subclase es desconocida drante el desarrollo.

- Decorador: permite a un objeto obtener una funcionalidad adicional en forma dinámica.

Singleton

- Problema: se admite exactamente una instancia de una clase. Los objetos necesitan un único punto de acceso global.

- Solución: Defina un método estático de la clase que devuelva el singleton

Estrategia

- Problema: ¿Cómo diseñar diversos algoritmos o políticas que están relacionadas?¿Cómo diseñar que estos algoritmos o políticas puedan cambiar?

- Solución: defina cada algoritmo/estrategia en una clase independiente, con una interfaz común.

Estrategia

- Es parecido al patrón Estado, solo que el patrón Estrategia encapsula un algoritmo y no la información de estado.

- Plantilla: también trabaja con algoritmos, pero aquí todos los objetos comparten un solo algoritmo definido por una superclase.

Composite

- Problema: ¿Cómo tratar un grupo o una estructura compuesta del mismo modo (polimórficamente) que un objeto no compuesto (atómico)?

- Solución: defina las clases para los objetos compuestos y atómicos de manera que implementen el mismo interfaz.

Compuesto

- Cada componente en una estructura jerárquica implementa la misma interfaz o extiende de una superclase común.

- Con este mecanimo al recorrer todos los elementos (nodos u hojas) no se tendrá que determinar de que clase son los elementos.

Fachada

- Problema: se requiere de una interfaz común, unificada para un conjunto de implementaciones o interfaces dispares --como en un subsistema--. Podría no ser conveniente acoplarla con muchas cosas del subsistema, o la implementación del subsistema podría cambiar.

Fachada

- Solución: defina un único punto de conexión con el subsistema --un objeto fachada que envuelva el subsistema--. Este objeto fachada presenta una única interfaz unificada y es responsable de colaborar con los componentes del subsistema.

Fachada

- Permite a un objeto (conocido como fachada) proporcionar una interfaz simple para los comportamientos de un subsistema.

- Niveles o capas: el modelo de tres capas corresponde al nivel de información, nivel medio y al nivel cliente.

Observador

- Problema: diferentes tipos de objetos subscriptores están interesados en el cambio de estado o eventos de un objeto emisor, y quieren reaccionar cada uno a su manera cuando el emisor genere un evento. Además, el emisor quiere mantener bajo acoplamiento con los subscriptores

Observador

- Solución:defina una interfaz “subscriptor” u “oyente” (listener). Los subscriptores implementan esta interfaz. El emisor dinámicamente puede registrar subscriptores que están interesados en un evento, y notificarles cuando ocurre un evento.

Observador

- Promueve el poco acoplamiento entre un objeto sujeto y objetos observadores; un sujeto notifica a los observadores cuado éste cambia de estado. Al ser notificados, los observadores cambian en respuesta al cambio del sujeto.

Estado

- Problema: el comportamiento de un objeto depende de su estado, y sus métodos contienen la lógica de casos que reflejan las acciones condicionales dependiendo del estado.

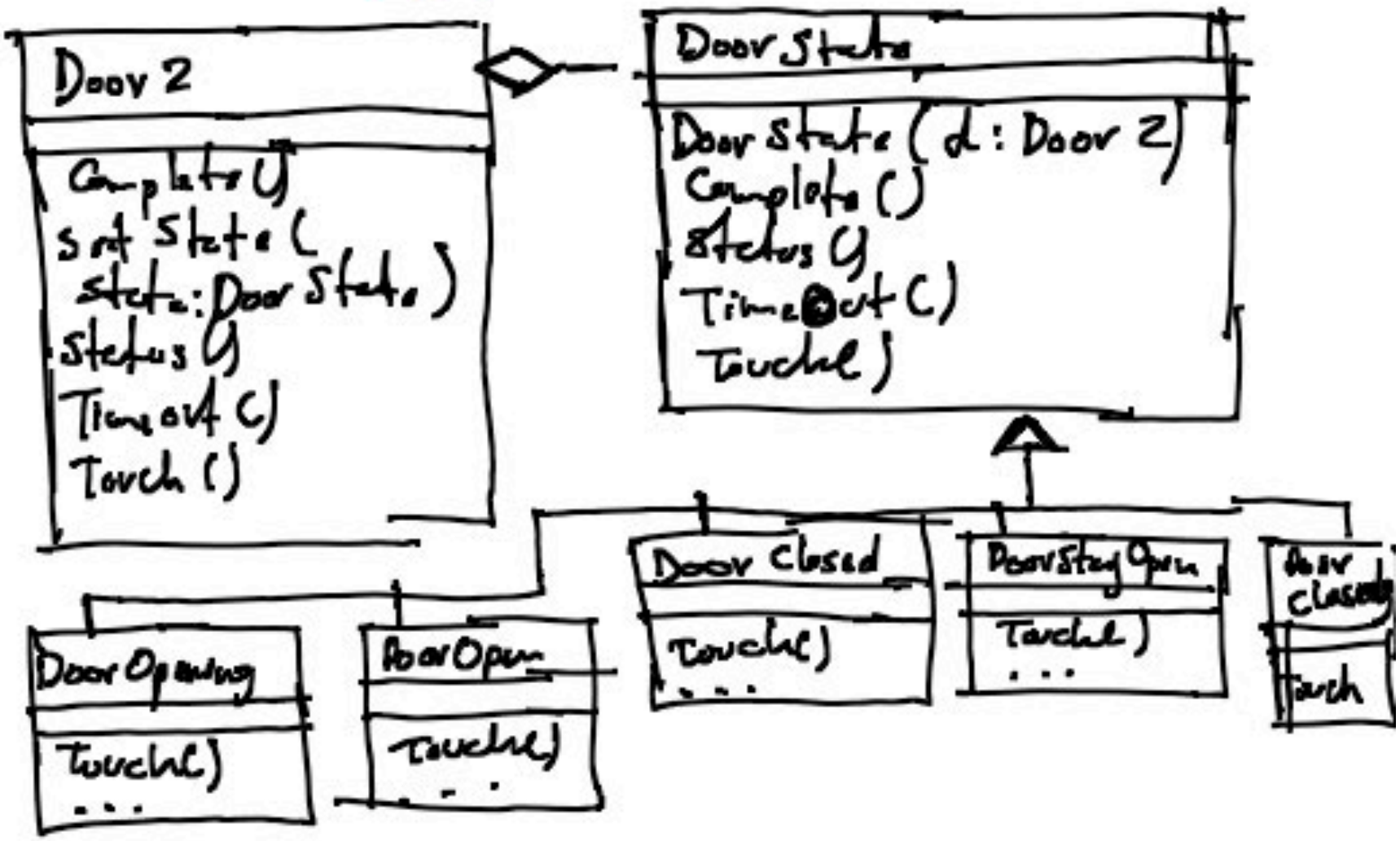
- Solución: cree clases de estado para cada estado que implementan una interfaz común.

Patrón Estado

- Utiliza una superclase abstracta (clase de estado), la cual contiene métodos que describen los comportamientos para los estados de un objeto.

- Patrones de diseño de creación: proporcionan formas de instanciar objetos en un sistema.

Patrón Estado



Command

- Problema: ¿Cómo gestionar las solicitudes o tareas que necesitan funciones como ordenar (estableciendo prioridades), poner en cola, retrasar, anotar en registro o deshacer?

- Solución: definir una clase por cada tarea que implementan una interfaz común.

Comando

- Permite a los programadores encapsular la funcionalidad deseada una vez en un objeto reutilizable.

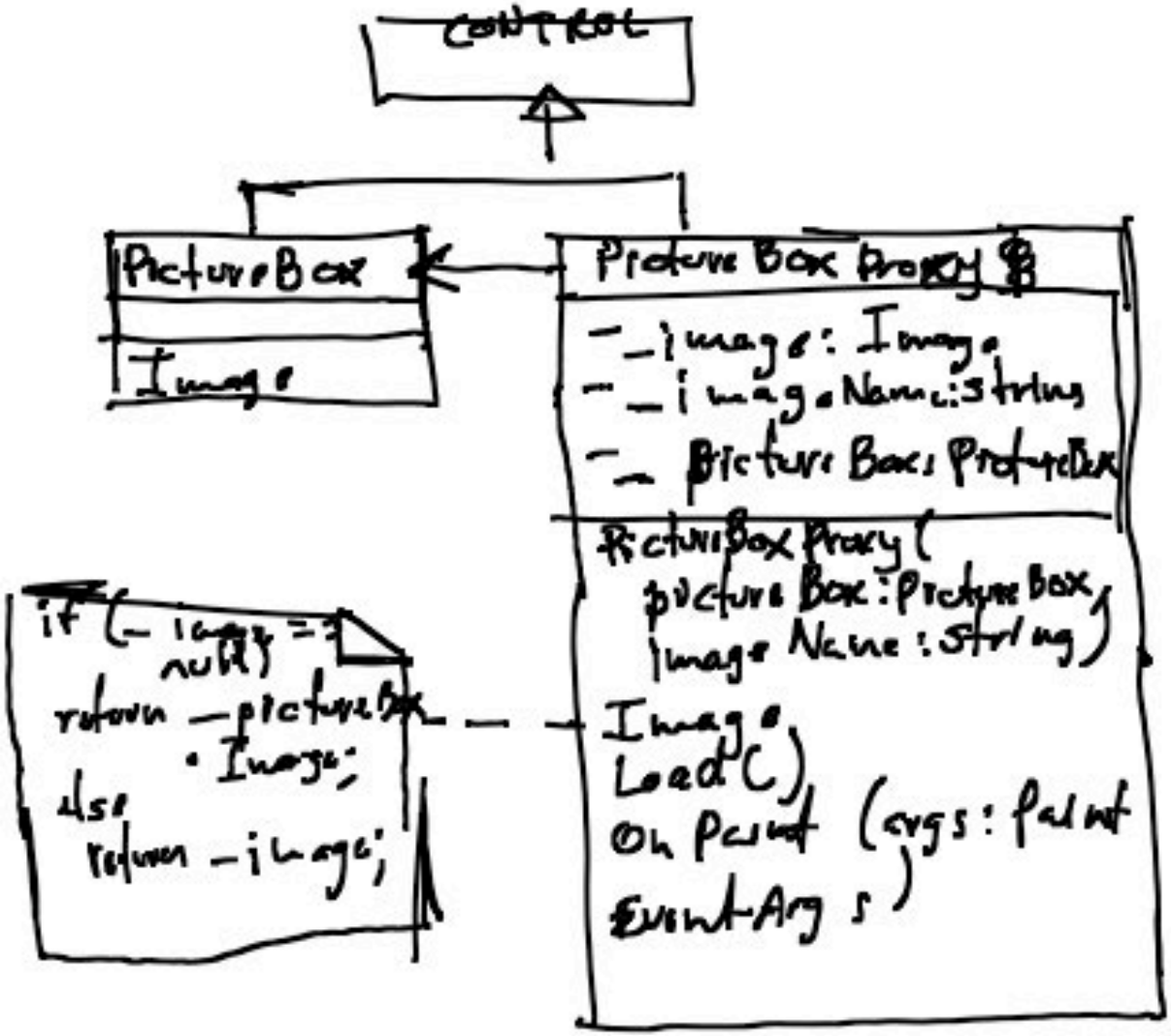
- Dicha funcionalidad puede agregarse a un menú, barra de herramientas, menú contextual o cualquier otro mecanismo.

Proxy

- Los patrones de diseño de estructura describen manera comunes de organizar las clases y objetos en un sistema.

- Proxy es un patrón de diseño de estructura que permite ejecutar un objeto mientras se ejecuta otro. Carga de varias imágenes cuando se da una animación.

Patrón Proxy



Patrones de diseño de comportamiento

- Proporcionan estrategias comprobadas para modelar la manera en que los objetos colaboran entre sí en un sistema.

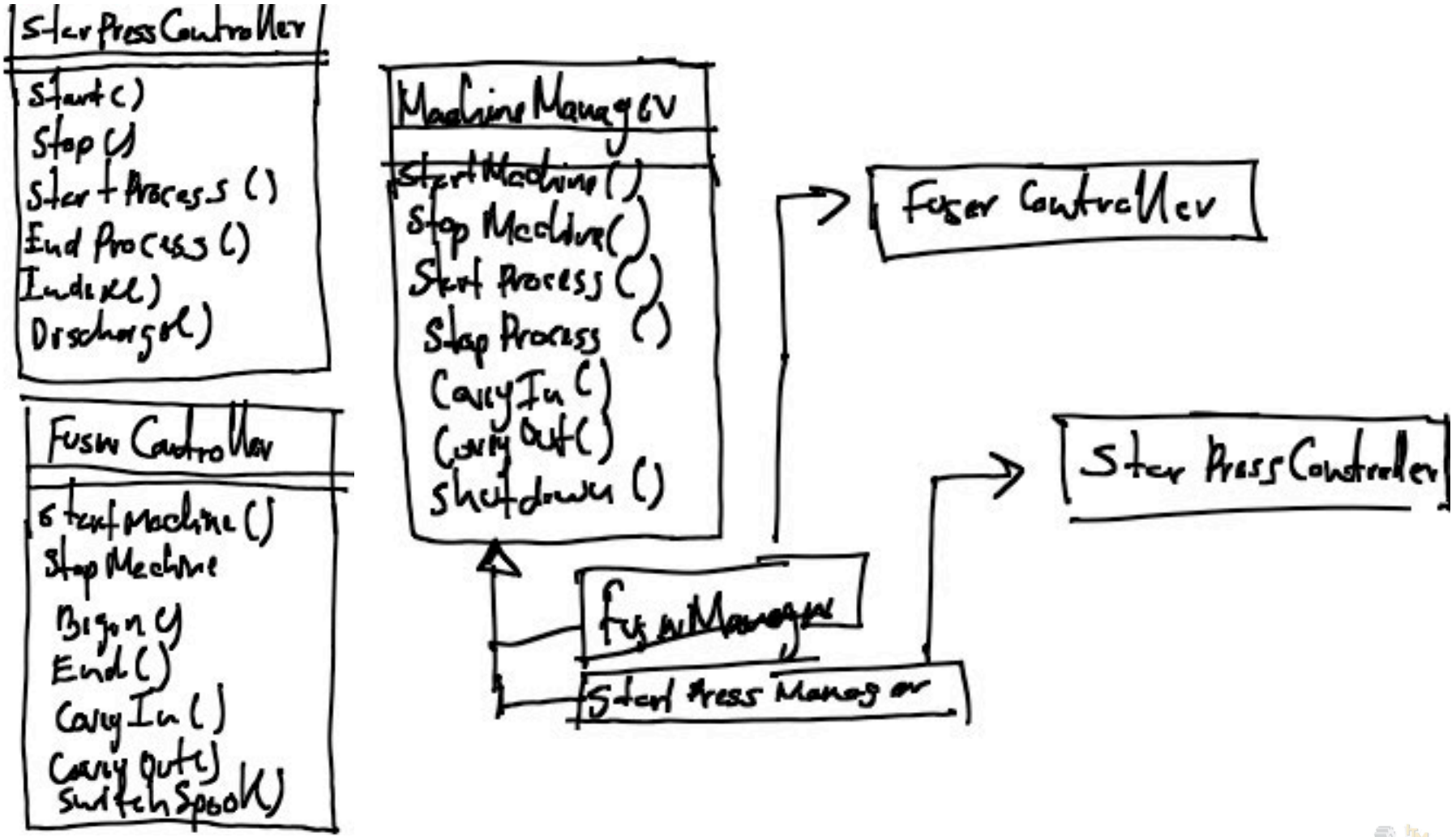
- Memento: implementa funciones de “deshacer”. Este patrón permite a un objeto guardar su estado, de tal forma de que si se requiere, se pueda restaurar fácilmente.

Puente

- Ayuda a mejorar la independencia de la plataforma.

- Divide un problema en una abstracción y sus implementaciones en jerarquías de clases separadas.

Patrón Puente



Cadena de responsabilidad

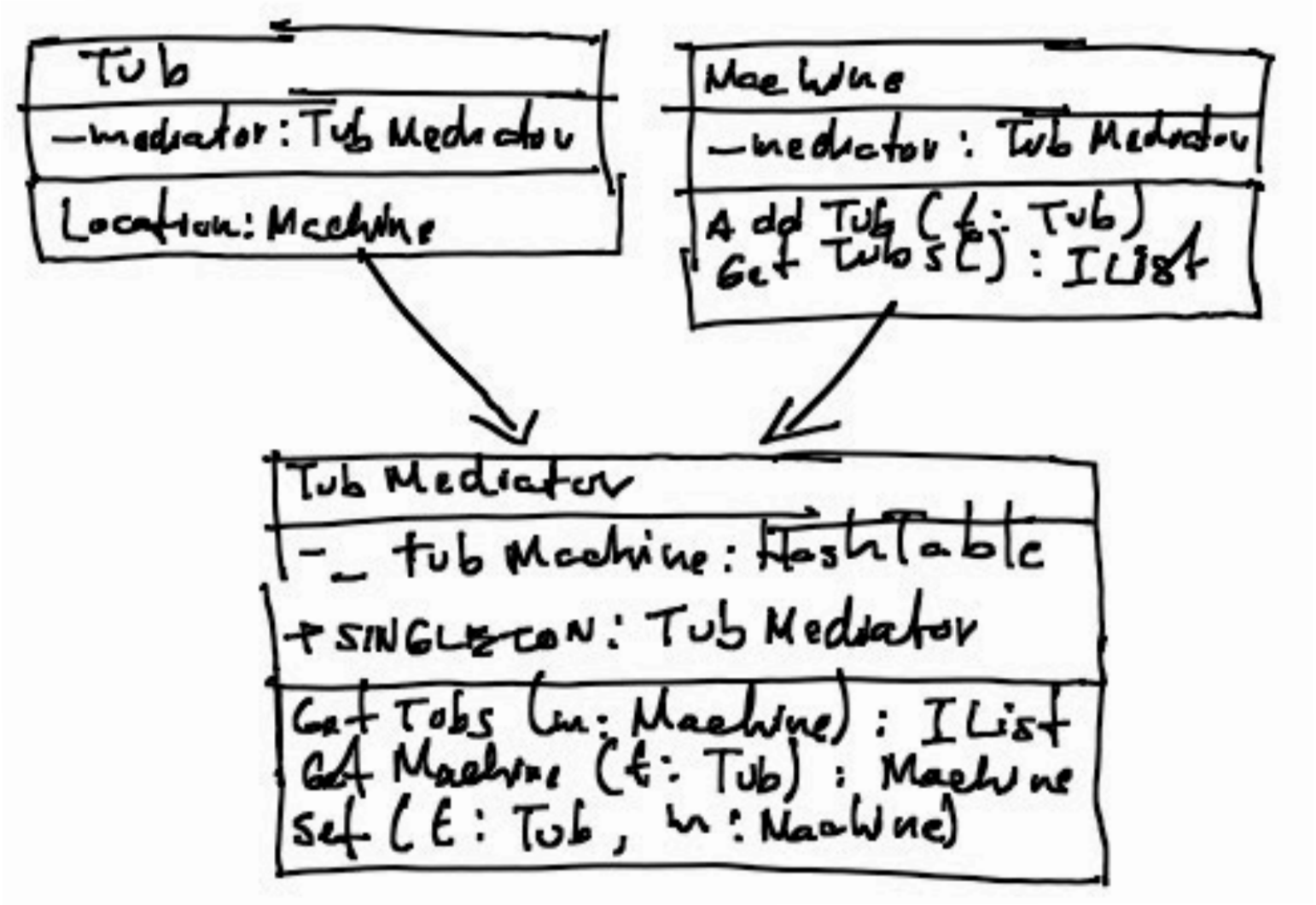
- Permite aun sistema determinar, en tiempo de ejecución, el objeto que se encargará de un mensaje.

- Permite a un objeto enviar un mensaje a varios objetos en una cadena de objetos. Cada objeto de la cadena puede manejar el mensaje o pasarlo al siguiente objeto de la cadena.

Patrones de diseño de concurrencia

- Ejecución de un solo subproceso (Synchronized).
- Suspensión precavida.
- Frustación
- Bloqueo de lectura/escritura
- Término de dos fases

Patrón Mediador



Referencias

- A. Froufe, P. Cárdenas, “J2ME Java “ Micro Edition”, Alfaomega Ra-Ma, México, 2004, ISBN: 970-15-1022-4.

- M. Prieto, “Desarrollo de juegos con J2ME”, Alfaomega Ra-Ma, México, 2005, ISBN: 970-15-1093-3.

Referencias

- C. Larman, “UML y Patrones”, Segunda edición, Pearson Prentice Hall, España, 2003, ISBN: 84-205-3438-2, pp. 624.

- P. Kimmel, “Manual de UML”, McGraw-Hill, México, 2007, ISBN: 970-10-5899-2.

- R. Pressman, “Ingeniería del Software”, McGraw-Hill, España, 2002, ISBN: 84-481-3214-9

Referencias

- H. Deitel, et al., “Como programar en Java”, Quinta edición, Pearson Prentice Hall, México, 2004, ISBN: 970-26-0518-0, pp. 1268.

- S. Metsker, “Design Patterns in C#”, Addison Wesley, Estados Unidos, 2004, ISBN: 0-321-12697-1.

¿Preguntas?