Ingeniería de Software II

Patrones de Diseño

RESUMEN

Docente: Dr. Pedro E. Colla

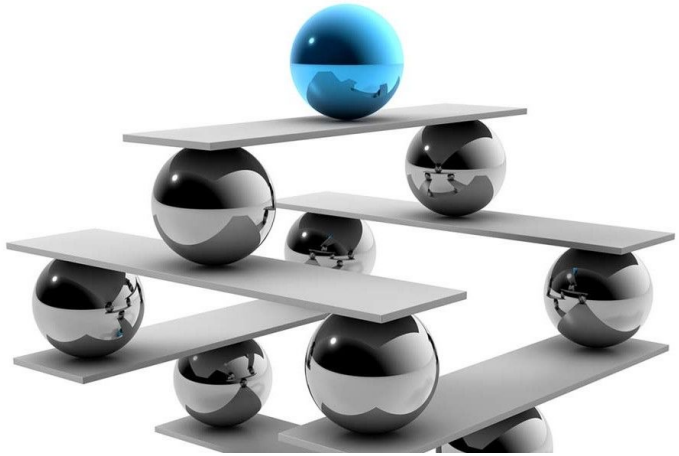
Patrones de Creación

	Singleton
¿Que hace?	Hace disponibles objetos de una determinada clase sin necesidad de generar uno diferente para cada demanda.
¿Por qué?	El arquitecto tiene que poder consolidar la creación de objetos similares.
¿Soluciona que?	(I) Consistencia. (II) Performance (creación, recursos).
¿Cómo se implementa?	Construcción de forma que el constructor esté dominado por el patrón y solo sea invocado una vez, siendo retornado en el resto de las invocaciones solo una copia (puntero).
¿Ejemplo?	Cualquier objeto que sea utilizado por toda la arquitectura y deba ser simultáneamente consistente para todas las instancias. Ej. AFIP.

	Prototype	Builder
¿Que hace?	Copia objetos existentes **sin** que el código para hacerlo dependa de las clases que se copian.	Construye un objeto paso por paso. Eso ocurre cuando para construir un objeto su constructor necesita construir múltiples objetos con dependencias complejas de una configuración
¿Por qué?	El arquitecto tiene que limitar la proliferación de código duplicado a nivel fuente.	Porque las dependencias complejas devienen en un constructor complejo y difícil de testear.
¿Soluciona que?	(I) Para copiar un objeto desde afuera no se ven los privados. (II) Si el copiador tiene que conocer los detalles de las clases habrá tantos copiadores como objetos.	Transforma un constructor complejo y que debe ser modificado permanentemente (por cada variación de la configuración) en uno simple y genérico que puede ser expandido "*por fuera*" mediante constructores también simples.
¿Cómo se implementa?	Que las clases implementen el método "*.clone()*" que copie al objeto "*desde adentro*". A quienes lo implementan se dice implementan el patrón **prototipo**.	Se separan los objetos "*internos*" en múltiples constructores que son invocados individualmente por un "*director*" (opcional) en función de las necesidades de configuración.
¿Ejemplo?	Cualquier objeto sobre el cual deban extraerse copias, por ejemplo un "*backup online*" para revertir cambios.	Construir cualquier objeto que tenga muchos comportamientos opcionales o una larga lista de parámetros de configuración en su constructor.

	Factory	Abstract Factory
¿Que hace?	Permite crear objetos desde una “superclase” donde todas las subclases compartan la interfaz, pero permite alterar posteriormente el comportamiento de las sub-clases	Crea familias de objetos relacionados, que son parte de un objeto aglutinante superior pero que individualmente NO compartan el mismo comportamiento funcional (ni interfaz).
¿Por qué?	El arquitecto se concentra en el rol funcional del objeto y no en su implementación física	Debo poder aglutinar en tiempo real de acuerdo a las reglas de negocio que objetos selecciono.
¿Soluciona que?	(I) Evita código condicional. (II) Evita dejar librado al azar la interfaz (III) Aporta flexibilidad para modificar posteriormente clases similares a nivel conceptual (igual interfaz) pero diferente comportamiento.	El arquitecto tiene libertad en expandir no solo las familias (mediante un Factory) sino como distintas familias pueden ser invocadas, se pueden aplicar reglas de consistencia (por ejemplo determinadas reglas de formación de modelos).
¿Cómo se implementa?	En lugar de llamar al constructor específico de una clase se llama al Factory quien decide a que constructor específico llamar, el cual se implementa incrementalmente. Al objeto creado se lo llama “producto”.	Se hace una “Factory” por cada familia de objetos funcionalmente similar y se llama a los mismos mediante un Abstract Factory. El Abstract Factory es un Factory de Factory.
¿Ejemplo?	Cualquier objeto que funcionalmente haga lo mismo (= interfaz) pero que tenga comportamiento diferente (ej. Método de delivery)	Configuraciones complejas de productos, por ejemplo teléfono celular donde hay que coordinar que los distintos componentes son consistentes entre si (país, legislación local, etc.)

Patrones de Creación



Patrones de Estructura

	Adapter	Proxy
¿Que hace?	Permite conectar objetos con interfaces incompatibles o emular el comportamiento de un objeto tal como es visto por otro.	Substituto o representante para otro objeto al cual controla el acceso de forma que no puedan acceder directamente a el.
¿Por qué?	El arquitecto debe compatibilizar objetos de comportamiento diferente sin incrementar el acoplamiento.	El arquitecto debe desplegar políticas de restricción de acceso por integridad, seguridad u otras razones al objeto real.
¿Soluciona que?	(I) Objeto que tiene 2 interfaces para interactuar con objetos diferentes. Permite convertir datos, estrategia de procesamiento, reglas de negocio e incluso ajustar interfaces tecnológicas.	Usualmente problemas de implementación diferida, encolado y aplicación de políticas.
¿Cómo se implementa?	Al menos una interfaz para un objeto existente para que pueda llamar en forma segura al adaptador. Puede ser bi-direccional (requiere máquina de estados FSM). Herencia.	Par “servidor”/”listener” back to back.
¿Ejemplo?	RS-232 Network adapter	Colas, NAT, Store & forward. Funcionalmente: Tarjeta de crédito (proxy de cuenta).

	Bridge	Composite
¿Que hace?	Permite separar una clase grande o grupo de clases relacionadas en dos jerarquías. Abstracción e implementación.	Permite crear (componer) objetos en estructuras de árbol, respetar sus jerarquías sin generar un número significativo de clases diferentes.
¿Por qué?	El arquitecto puede gestionar la dimensionalidad de un problema sin incrementar el número de clases.	Al representar jerarquías de objetos similares con igual interfaz habilita la aplicación de algoritmos recursivos.
¿Soluciona que?	Gestión de clases con mas de una dimensión de atributos donde pueda ser necesario agregar variantes de atributos adicionales (genera explosión combinacional de clases).	Evitar la multiplicación de clases y código especializado según la ubicación en la jerarquía.
¿Cómo se implementa?	Separa algunos atributos a interfaces que se implementan por separado.	Con una interfaz común se apunta a un objeto genérico que establece las clases de gestión de árbol (.add/.remove/.get/.execute).
¿Ejemplo?	El mismo producto conceptual tiene implementaciones físicas diferentes (ej. Control remoto).	BOM. Algoritmos de implementación de estructuras de SWE (linked list, queue, hash,..), estructuras de travesía (Markov chains, Graphmaster).

	Decorator	Facade
¿Que hace?	Permite agregar funcionalidad a objetos agregando la misma mediante el encapsulado en contenedores especiales con la misma interfaz.	Proporciona una interfaz simplificada a una biblioteca, un marco, una herramienta o cualquier otro grupo complejo de clases
¿Por qué?	El arquitecto debe proveer elementos para extender la funcionalidad sin modificar las clases usuarias.	Permite diferir las decisiones de implementación asegurando a las clases usuarias la interfaz al servicio que permita su desarrollo.
¿Soluciona que?	Casos donde una misma acción conceptual a nivel funcional puede requerir implementaciones diferentes.	Necesidad de “stubs” funcionales. Casos donde se requiere flexibilidad de implementación o implementación diferida de la funcionalidad subyacente.
¿Cómo se implementa?	Se identifican las primitivas de operación funcional y se define una interfaz que capture las mismas. Luego el encapsulado usa las primitivas para la implementación especial.	Se establece una interfaz común y una implementación básica que luego puede ser incrementada por herencia.
¿Ejemplo?	Notificaciones sobre distintas plataformas.	Sistemas de order entry, gestión de catálogos y menú de servicios.

	Flyweight
¿Que hace?	Permite mantener más objetos dentro de la cantidad de memoria compartiendo las partes comunes del estado entre varios objetos en lugar de mantener toda la información en cada objeto
¿Por qué?	El arquitecto tiene que poder equilibrar requerimientos físicos mediante la reducción controlada de figuras de calidad de acoplamiento.
¿Soluciona que?	Proveer una interfaz consolidada para el acceso a partes comunes de manera de gestionar el acoplamiento cuando consideraciones de performance lo hacen necesario.
¿Cómo se implementa?	Se identifica la información que se desea consolidar y se destina una interfaz a accederla, en caso de lectura/grabación se protegen las operaciones con semáforos/critical code segment.
¿Ejemplo?	Cálculos basados en modelos de elementos finítos. Trayectorias astronómicas. Problema de los n-cuerpor. Navier-Stokes.

Patrones de Estructura

Patrones de Comportamiento

	Chain of command	Iterator
¿Que hace?	Permite pasar solicitudes a lo largo de una cadena de manejadores. Al recibir una solicitud, cada manejador decide si la procesa o si la pasa al siguiente manejador de la cadena.	Recorre elementos de un conjunto de datos sin exponer la implementación subyacente de los mismos.
¿Por qué?	El arquitecto debe proveer mecanismos para agregar comportamientos sin modificar la implementación de los existentes.	Distintas implementaciones aportan ventajas y desventajas que pueden variar con el tiempo y hacer conveniente su cambio sin impactar el resto.
¿Soluciona que?	Un mismo mensaje o solicitud que deba ser procesado potencial o realmente por múltiples módulos.	Acceso a estructuras abstractas de datos.
¿Cómo se implementa?	Cada comportamiento particular se implementa por un segmento de código u objeto llamado “manejador” (*handler*), se establece un criterio se pasaje de mensajes.	Servicios para crear, borrar, listar, agregar, remover y contar elementos en una colección de datos.
¿Ejemplo?	Configuración de productos. Bill of material (BOM).	Array, Linked List (simple/doble), hash, árbol, cola, statck, etc.

	Observer	Memento
¿Que hace?	Permite definir un mecanismo de suscripción para notificar a varios objetos sobre cualquier evento que le suceda al objeto que están observando.	Permite guardar y restaurar el estado previo de un objeto sin revelar los detalles de su implementación.
¿Por qué?	El arquitecto puede estructurar una topología de sub-sistemas que actúen sobre los mismos eventos sin interactuar entre ellos.	El arquitecto puede establecer puntos de control o sincronismo (checkpoint) de manera que el objeto disparador no conozca los detalles de los datos que deben ser preservados.
¿Soluciona que?	Gestión de mensajes únicos que provocan comportamientos diferentes según sea quien los procese.	Establecimiento de backups, resguardos y puntos de control.
¿Cómo se implementa?	Una clase despachadora que alimenta a una cola de mensajes desde donde todos interactúan.	Con una interfaz común se apunta a un objeto genérico que establece las clases de gestión punto de control las que al ser invocadas preservan la estructura privada de datos.
¿Ejemplo?	Distribución multimedia de contenidos.	Soporte de mecanismos “*undo*” o historia de estados previos.

	Mediator	State
¿Que hace?	Permite reducir las dependencias caóticas entre objetos. El patrón restringe las comunicaciones directas entre los objetos, forzándolos a colaborar únicamente a través de un objeto mediador.	Permite a un objeto alterar su comportamiento cuando su estado interno cambia. Parece como si el objeto cambiara su clase.
¿Por qué?	El arquitecto debe proveer elementos para reducir el acoplamiento en una configuración de interacciones n-arias entre objetos.	El arquitecto debe poder implementar autómatas de estados finitos sin que le produzca una explosión de clases para representar estados.
¿Soluciona que?	Problemas de cohesión y acoplamiento cuando existen múltiples interacciones complejas entre objetos (problema de (n)(n-1)/2 relaciones).	Implementación de procesos de negocios que requieren su gestión mediante un autómata de estados finitos.
¿Cómo se implementa?	Objeto *dispatcher*.	Los métodos operan con un esquema polimórfico en base al estado de la clase (la que puede ser privada).
¿Ejemplo?	Cursado de mensajes de control peer-to-peer entre objetos disimiles.	Reproductor multimedia (los comandos generan acciones diferentes según el estado de reproducción).

	Command	Strategy
¿Que hace?	Permite generar una solicitud a ser satisfecha por otra clase de manera que tenga toda la información necesaria para su procesamiento independientemente de los requerimientos de otras clases y aún así operar con una interfaz común.	Permite definir una familia de algoritmos, colocar cada uno de ellos en una clase separada y hacer sus objetos intercambiables.
¿Por qué?	El arquitecto debe gestionar colas de procesamiento genéricas de formato común pero donde los mensajes pueden tener tratamientos muy distintos.	Permite que algoritmos diferentes en su implementación puedan ser manipulados mediante una interfaz común la que es resuelta en tiempo de ejecución.
¿Soluciona que?	Requerimientos de procesamiento en un ambiente distribuido.	Necesidad de evitar acoplamiento entre clases conceptualmente similares pero con implementaciones muy diferentes.
¿Cómo se implementa?	Un objeto mediador gestiona la entrada y salida de una cola de mensaje, cada clase que ingresa o extrae un mensaje puede “consumirlo” o “compartirlo” con otras.	Interfaz común que implementa diferencialmente en forma privada.
¿Ejemplo?	Colas en ambiente distribuidos. Colas de mensajes en GUI.	Implementación de estructuras de datos abstractas.

	Visitor	Template
¿Que hace?	Permite separar algoritmos de los objetos sobre los que operan.	Define el esqueleto de un algoritmo en la superclase pero permite que las subclases sobrescriban pasos del algoritmo sin cambiar su estructura (su interfaz permanece igual).
¿Por qué?	El arquitecto debe poder separar implementaciones de los objetos que las implementan.	Permite separar la implementación del algoritmo de su utilización en la arquitectura y desacoplar quien usa al servicio de quien lo provee.
¿Soluciona que?	Reducir acoplamiento en caso de funcionalidad adicional en una clase o agregado de clases con algoritmos nuevos.	Atiempamiento de construcción, necesidad de definición detallada en forma no homogénea dentro de la arquitecturra
¿Cómo se implementa?	El nuevo comportamiento se despliega en una clase separada llamada visitante, en lugar de intentar integrarlo dentro de clases existentes. El objeto que originalmente tenía que realizar el comportamiento se pasa ahora a uno de los métodos del visitante como argumento	Se divide un algoritmo en una serie de pasos, se convierten estos pasos en métodos y colocan una serie de llamadas a esos métodos dentro de un único método *plantilla*. Los pasos pueden ser abstractos, o contar con una implementación por defecto. Para utilizar el algoritmo, el cliente debe aportar su propia subclase, implementar todos los pasos abstractos y sobrescribir algunos de los opcionales si es necesario.
¿Ejemplo?	Una misma acción que difiere según el contexto u objetivo de ejecución (por ej. Orientación geográfica, de industria, de producto, etc.)	Permitir un producto base con múltiples variaciones definidas en tiempo de ejecución.

Patrones de Comportamiento