**5)**

**Alta Cohesión**

Problema: ¿Cómo mantener la complejidad dentro de límites manejables?

Solución: Asignar una responsabilidad de manera que los métodos de las clases sean coherentes y completos, logrando mayor cohesión entre las clases.

Como ejemplo, proponemos el siguiente caso:



Podemos observar que las responsabilidades asignadas a la clase ExpertoValidarAcceso son sólo las necesarias para lograr el objetivo del caso de uso, que sería verificar si la persona que está tratando de acceder a determinado lugar del establecimiento, se encuentra con acceso al mismo. Sin las responsabilidades asignadas, no sería posible la finalización exitosa del caso de uso, y son las responsabilidades propias del caso de uso.

Por ejemplo, se le podría haber asignado a este experto la responsabilidad de gestionar los permisos, pero esta responsabilidad debe ser asignada a otro experto como sería por ejemplo el ExpertoArmarPermisos, quien conoce los mecanismos necesarios para llevar a cabo dicha tarea manteniendo alta la cohesión.

**Bajo Acoplamiento**

Problema: ¿Cómo soportar bajas dependencias, bajo impacto frente al cambio e incrementar la reutilización?

Solución: Asignar una responsabilidad de manera que el acoplamiento permanezca bajo.

Un buen ejemplo de bajo acoplamiento sería:



El que hacemos al relacionar el ExpertoValidarAcceso hacia la clase IndirecciónPersistencia. Para llevar a cabo la realización del C.U. validar acceso, sólo acoplamos el experto del caso de uso con la clase IndirecciónPersistencia. De ésta manera nos permite tener un bajo impacto frente al cambio, ya que ante cualquier modificación que sufra la case IndireccionPersistencia, sólo debamos modificar el experto del caso de uso, sin que esto genere cambios en las otras clases que necesita el caso de uso para cumplir su objetivo.

**Experto**

Problema: ¿Cuál es el principio general para asignar responsabilidades a los objetos?

Solución: Asignar una responsabilidad al experto en información: la clase que contiene la información necesaria para llevar a cabo la responsabilidad.



En el ejemplo seleccionado, la clase Zona tiene la responsabilidad de conocer las puertas que pese, es el experto en información ya que tiene los comportamientos necesarios para llevar a cabo dicha responsabilidad.

**Creador**

Problema: ¿Quién debería ser el responsable de la creación de una nueva instancia de alguna clase?

Solución: Asignar a la clase B la responsabilidad de crear una instancia de A si se cumple uno o más de los siguientes casos:

* B agrega objetos de A
* B contiene objetos de A
* B utiliza instancias de objetos de A
* B utiliza más estrechamente objetos de A
* B tiene los datos de inicialización que se pasarán al objeto de A cuando sea creado (por tanto, B es un Experto con respecto a la creación de A)

B es un creador de A

Por ejemplo:



Podríamos preguntarnos cuál es la clase que tiene la responsabilidad que crear las distintas instancias de la clase Semana: sería la misma clase Calendario, ya que es el experto de información, tiene todos los datos de inicialización necesarios para la creación de la semana.

**Indirección**

Problema: ¿Dónde asignar una responsabilidad, para evitar el acoplamiento directo entre dos (o más) cosas? ¿Cómo desacoplar los objetos de manera que se soporte el bajo acoplamiento y el potencial para reutilizar permanezca más alto?

Solución: Asigne la responsabilidad a un objeto intermedio que medie entre otros componentes o servicios de manera que no se acoplen directamente.

Utilizamos la clase indirecciónPersistencia como una indirección, que nos permite implementar bien el modelado en capas y abstraernos de la capa de persistencia. Solo definimos métodos genéricos para la búsqueda y el guardado de los distintos objetos de forma permanente. Esto permite la independencia ante cualquier tipo de base de datos.



**Fabricación Pura**

Problema: ¿Qué objetos deberían tener la responsabilidad cuando no quiere violar los objetivos de Alta Cohesión y Bajo Acoplamiento, u otros, pero las soluciones que ofrece el Experto (por ejemplo) no son adecuadas?

Solución: Asigne un conjunto de responsabilidades altamente cohesivo a una clase artificial o de conveniencia que no representa un concepto del dominio del problema: algo inventado para soportar alta cohesión, bajo acoplamiento y reutilización.

Una clase creada a partir de este patrón ha sido la IndirecciónPersistencia.

Podríamos preguntarnos quién es el responsable de hacer persistente el objeto Acceso por ejemplo: en principio podríamos asignarle esa responsabilidad al experto del caso de uso (ExpertoValidarAcceso), porque es quien conoce todos los datos para llevar a cabo dicha tarea, pero si así lo hiciésemos, quedaría fuertemente acoplado con clases que nos permitieran guardar objeto creado de manera persistente en una base de datos. Si se llegase cambiar el motor de base de datos por decisión de la empresa, o cambiara el tipo de almacenamiento (a XML por ejemplo), se haría necesario cambiar los métodos del experto, para que pueda llevar a cabo esta tarea. También bajaría la cohesión, ya que ahora debería conocer sobre materialización y desmaterialización de objetos.

Por eso la solución a nuestro problema es crear la clase, IndirecciónPersistencia que no es una clase de nuestro negocio, sino que la creamos para quitarle esta responsabilidad a los expertos, dejándolos que solo conozcan de reglas de negocio.

**Controlador**

Problema: ¿Quién debe ser el responsable de gestionar un evento de entrada al sistema?

Solución: Asignar la responsabilidad de recibir o manejar un mensaje de evento del sistema a una clase que representa una de las siguientes opciones:

* Representa el sistema global, dispositivo o subsistema (controlador de fachada)
* Representa un escenario de caso de uso en el que tiene lugar el evento del sistema (controlador de sesión o de caso de uso)
* Utilice la misma clase controlador para todos los eventos del sistema en el mismo escenario de caso de uso.
* Informalmente, una sesión es una instancia de una conversación con un actor. Las sesiones pueden tener cualquier duración, pero se organizan a menudo en función de los casos de uso (sesiones de caso de uso)

Creamos la clase ControladorValidarAcceso, que recibe los eventos a través de la interfaz, y encomienda la responsabilidad de llevar a cabo la lógica de negocio que corresponda (validar permiso en este caso), al experto del caso de uso. Esto permite desacoplar la interfaz del experto.

