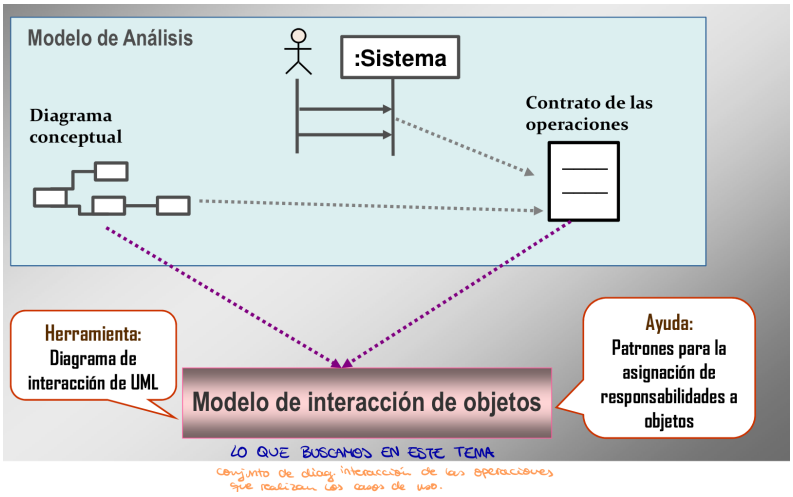
TEMA 3.3: Diseño de los casos de uso

---

1. INTRODUCCIÓN



2. PATRONES DE DISEÑO PARA ASIGNAR RESPONSABILIDADES

2.1. RESPONSABILIDAD

Una responsabilidad es un contrato u obligación que debe tener un objeto en su comportamiento. Se divide en dos categorías:

- **Hacer**:
  - Hacer algo en uno mismo
  - Iniciar una acción entre otros objetos
  - Controlar y coordinar actividades en otros objetos
- **Conocer**:
  - Estar enterado de los datos privados encapsulados
  - Estar enterado de la existencia de objetos conexos
  - Estar enterado de las cosas que se pueden derivar o calcular

No hay que confundir responsabilidad y método. Las responsabilidades se implementan usando métodos que operan solos o en colaboración con otros métodos. Los métodos cumplen las responsabilidades.

2.2. PATRÓN DE DISEÑO

Es una descripción de un problema con su solución en un determinado contexto. Son una forma de reutilizar el conocimiento y la experiencia de otros diseñadores.

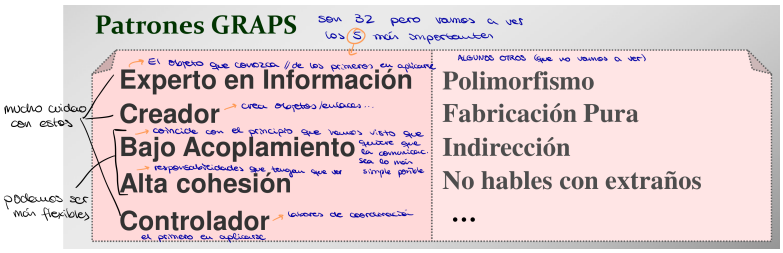
Las partes esenciales de un patrón son:

- **Nombre**: que sea una referencia significativa al patrón
- **Problema**: una descripción del problema que enuncie cuándo se puede aplicar el patrón
- **Solución**: una descripción de la solución de diseño, sus relaciones y responsabilidades. Es una plantilla para que una solución se instale en diferentes formas
- **Consecuencias**: los resultados (buenos y malos) y las negociaciones al aplicar el patrón. Ayudan a entender si es factible usar el patrón en una situación particular

3. PATRONES GRASP (General Responsability Assignment Software Patterns)

Son los más estándar. Describen los principios fundamentales del diseño de objetos y la asignación de responsabilidades, expresados como patrones. A un objeto se le puede aplicar más de un patrón.

Algunas características son:
- No expresan nuevos principios en la ingeniería del SW
- Codifican conocimiento, expresiones y principios ya existentes
- Son un ejemplo de fuerza de abstracción porque dan nombre a una idea compleja
- Apoyan la incorporación de conceptos a nuestro sistema cognitivo y a la memoria
- Facilitan la comunicación



3.1. EXPERTO EN INFORMACIÓN

Problema: ¿Cuál es el principio general para asignar responsabilidades a los objetos?
Solución: Asignar responsabilidad a la clase que contiene la información necesaria para llevarla a cabo
Consecuencias:
- MALAS: En ocasiones va en contra de los principios de acoplamiento o cohesión. Puede ocurrírsenos que un mismo objeto sepa de todo, por lo que tendría mucha interacción con otros, y habría alto acoplamiento (mucha comunicación) y baja cohesión (conoce cosas muy dispares)
- BUENAS: Mantiene el ocultamiento de información y distribuye el comportamiento

Ver ejemplo diapositiva 7.

3.2. CREADOR

Problema: ¿Quién debería ser el responsable de la creación de una instancia de alguna clase
Solución: Asignar a la clase B la responsabilidad de crear instancias de A cuando:
- B agrega objetos de A
- B contiene objetos de A
- B registra objetos de A

- B utiliza objetos de A
- B tiene los datos de inicialización de A

Consecuencias:
- MALAS: No es conveniente su uso cuando se construye a partir de instancias existentes
- BUENAS: Produce bajo acoplamiento

Ver ejemplo diapositiva 9.

3.3. BAJO ACOPLAMIENTO

Problema: ¿Cómo soportar bajas dependencias, bajo impacto del cambio e incremento de la reutilización?
Solución: Asignar una responsabilidad de manera que el acoplamiento permanezca bajo (ppio de independencia)
Consecuencias:
- MALAS: Llevado a extremo puede ocasionar diseños pobres; en un conjunto de
- clases debe haber un nivel de acoplamiento moderado y adecuado
- BUENAS: No afectan los cambios en otros componentes, son fáciles de entender de manera aislada, es conveniente para reutilizar

Cuánto más independiente es un módulo, más reutilizable es.

Ver ejemplo diapositiva 11.

Pregunta examen: responder adecuadamente qué clase conoce al objeto y quién es el encargado de crear/destruir al objeto.

Formas comunes de acoplamiento (de más débil a más fuerte):

- El **Tipo X** tiene un atributo que referencia a una instancia de **Tipo Y**, o al propio **Tipo Y**
- Un objeto de **Tipo X** invoca los servicios de un objeto de **Tipo Y**
- El **Tipo X** tiene un método que referencia a una instancia de **Tipo Y**, o al propio **Tipo Y**, de algún modo
- El **Tipo X** es una subclase, directa o indirecta, del **Tipo Y**
- El **Tipo Y** es una interfaz y el **Tipo X** implementa esa interfaz

3.4. ALTA COHESIÓN

Problema: ¿Cómo mantener la complejidad manejable?
Solución: Asignar una responsabilidad de manera que la cohesión permanezca alta. No es lo mismo un módulo con muchas acciones que muchos módulos con pocas acciones
Consecuencias:
- MALAS: Ninguna, renunciar a la alta cohesión sólo cuando esté muy justificado. Tener objetos que encapsulen muchas funciones siempre es malo, mejor separar
- BUENAS: Claridad y facilidad de entendimiento del diseño, simplificación del mantenimiento y de las mejoras, a menudo soporta bajo acoplamiento, incremento de la reutilización

Ver ejemplo diapositiva 14.

Grados de cohesión funcional:

- Muy baja cohesión
  Una única clase es responsable de muchas cosas en áreas funcionales diferentes
- Baja cohesión
  Una única clase tiene la responsabilidad de una tarea compleja en un área funcional
- Alta cohesión
  Una única clase tiene una responsabilidad moderada en un área funcional y colabora con otras clases para llevar a cabo las tareas
- Moderada cohesión
  Una única clase tiene responsabilidades ligeras y únicas en unas pocas áreas diferentes que están lógicamente relacionadas con el concepto de la clase, pero no entre ellas

3.5. CONTROLADOR

Problema: ¿Quién debe ser 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 represente

- El sistema global, dispositivo o subsistema (controlador de fachada)
- El escenario de caso de uso en el que tiene lugar el evento del sistema (controlador de caso de uso)

Consecuencias:

- MALAS: Controladores saturados (van más lentos)
- BUENAS: Se asegura que la lógica de la aplicación no se maneje en la interfaz del sistema, hay un aumento de la reutilización y bajo nivel de acoplamiento, hay posibilidad de razonar sobre el estado de los casos de uso

La única responsabilidad del controlador es captar los eventos (mensajes) de fuera y descomponerlos para distribuirlos. No tiene por qué representar a clases del modelo conceptual. Además, puede haber más de un controlador cuando el sistema es muy grande, para no saturar a uno único.

Ver ejemplo diapositiva 17.

4. ELABORACIÓN DEL MODELO DE INTERACCIÓN DE OBJETOS

4.1. DIRECTRICES GENERALES

- Las bases principales para obtener los diagramas de interacción son los contratos y el modelo conceptual
- El modelo conceptual sirve de guía para saber qué objetos pueden interaccionar en una operación
- Todo lo especificado en el contrato, especialmente las poscondiciones, excepciones y salidas, tienen que satisfacerse en el correspondiente diagrama de comunicación
- Para la elaboración de cada diagrama de comunicación se aplican los patrones de diseño

4.2. PASOS A SEGUIR

A. Elaborar los diagramas de interacción

Para cada operación especificada en los diagramas de secuencia

1. Tener presente el diagrama de conceptos y el contrato de la operación
2. Representar las relaciones del controlador con los objetos que intervienen en la interacción
3. Asignar responsabilidades a objetos
4. Establecer tipos de enlaces entre objetos

B. Inicialización del sistema

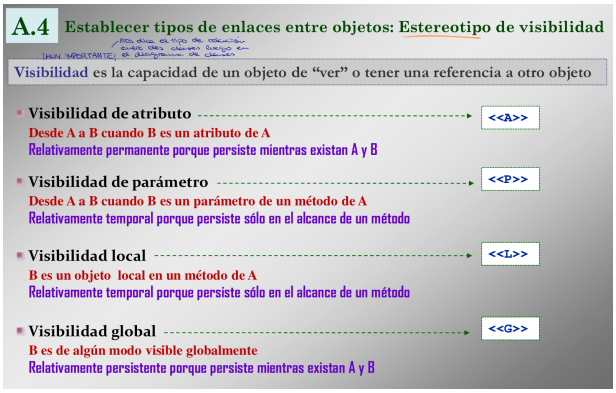
C. Establecer relaciones entre el modelo y la Interfaz de Usuario

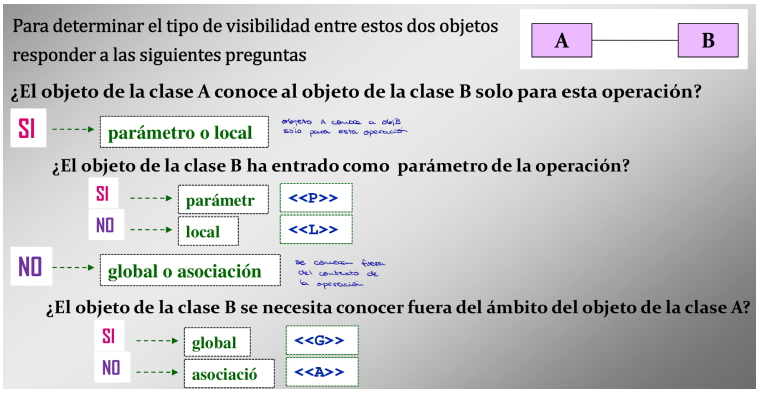
A modo de ampliación de ese esquema de pasos:

A2) ¿Qué objetos necesita conocer directamente el controlador?

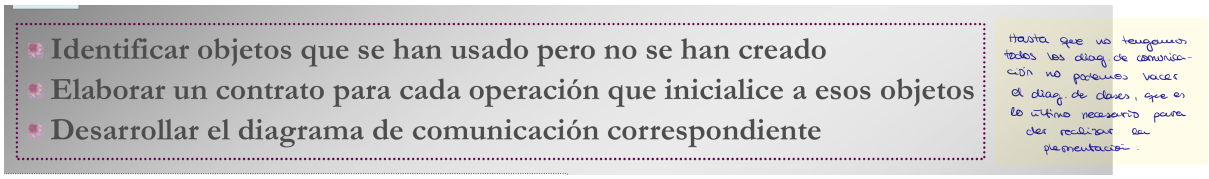
A3) Según el nivel en el que se encuentre la elaboración del diagrama, para cada objeto se debe formular la siguiente pregunta:
- De todo lo que se dice en el contrato, ¿de qué es responsable?
- La respuesta es aplicar los patrones de diseño, fundamentalmente el experto en información y el creador

A4)





B)



C)