

UT03.-DISEÑO ORIENTADO A OBJETOS. DIAGRAMAS ESTRUCTURALES.

Índice

1.- Introducción a la orientación a objetos

2.- Conceptos de orientación a objetos

2.1.- Ventajas de la orientación

a objetos. 2.2.- Clases, atributos
y métodos.

2.3.- Visibilidad

2.4.- Objetos. Instanciación.

3.- UML

3.1.- Elementos de los diagramas UML.

3.2.- Tipos de diagramas UML.

3.3.- Herramientas para la elaboración de diagramas UML.

3.4.- Diagramas de clases.

3.5.- Relaciones entre clases.

3.6.- Tipos de relaciones entre clases.

3.7.- Paso de los requisitos de un sistema al diagrama de
clases.

3.8.- Generación de código a partir del diagrama de clases.

3.9.- Generación de la documentación.

4.- Ingeniería inversa.

1. Introducción a la orientación a objetos.

Introducción.

- **En el análisis y diseño de una aplicación con un enfoque estructurado:**
 - ▣ Proceso centrado en los **procedimientos**.
 - ▣ Se codifican mediante funciones que actúan sobre **estructuras de datos** → **Programación estructurada**.
 - ▣ **Qué** hay que hacer → funcionalidad.
- **En el análisis y diseño de una aplicación con un enfoque orientado a objetos:**
 - ▣ Un sistema se entiende como un **conjunto de objetos** con propiedades y un comportamiento.
 - ▣ **Objeto:** consta de una estructura de datos (**propiedades**) y un conjunto de operaciones (**comportamiento**).
 - ▣ Los datos definidos dentro del objeto son los atributos y las operaciones definen el comportamiento y permiten cambiar el valor de los atributos.
 - ▣ Los objetos se comunican mediante el paso de **mensajes**.
 - ▣ **Clase: Abstracción** de un conjunto de objetos → plantilla para crear objetos.
 - ▣ Cuando se crea un objeto se ha de especificar de qué clase es para que el compilador sepa cuáles son sus características.
 - ▣ Para el análisis y diseño orientado a objetos utilizamos el **lenguaje UML** (Unified Modeling Language – Lenguaje de Modelado Unificado)

Introducción

- **UML** es un lenguaje de **modelo basado en diagramas** que sirve para expresar modelos (representaciones de la realidad donde se ignoran detalles de menor importancia).
- UML se ha convertido en un estándar de facto de la mayor parte de metodologías de desarrollo orientado a objetos.
- Ejecución de una aplicación OO:
 - ▣ Creación de objetos a medida que se necesitan
 - ▣ Los mensajes se mueven de un objeto a otro (o del usuario a un objeto).
 - ▣ Borrado de objetos cuando ya no se necesitan → liberación de la memoria → proceso automático en Java.

2. Conceptos de orientación a objetos

Conceptos de orientación a objetos

□ Objeto

- ▣ Unidad dentro de un programa de computadora que consta de un **estado** (**propiedades** = datos almacenados con un determinado valor) y de un **comportamiento** (**métodos** = tareas realizables durante el tiempo de ejecución).
- ▣ Un objeto puede ser creado:
 - Instanciando una clase (POO)
 - Mediante escritura directa de código y replicación (Programación basada en prototipos)
- ▣ Es un ente dinámico → existen en tiempo de ejecución y ocupan memoria.

□ Clase

- ▣ Define el tipo de objeto, cómo funciona un determinado tipo de objeto.
- ▣ Concepto estático → abstracción de un conjunto de objetos.

□ Método

- ▣ Operación de un determinado objeto.
- ▣ Mensaje \cong llamada a una operación de un objeto.

Conceptos de orientación a objetos

□ **Abstracción.**

- Captura características y comportamientos similares de un conjunto de objetos → conjunto de clases.

□ **Encapsulación.**

- Significa agrupar todos los elementos que pueden considerarse pertenecientes a una misma entidad, al mismo nivel de abstracción → cohesión de los componentes del sistema.
- Ocultar ciertos detalles de los objetos → separar el aspecto interno del externo de un objeto → ocultar los atributos y métodos de los objetos.

□ **Modularidad**

- Subdividir una aplicación en partes más pequeñas (módulos), cada una de las cuales debe ser tan independiente como sea posible de la aplicación en sí y de las restantes partes.
- En POO, clase \cong módulo más básico del sistema

□ **Cohesión**

- En la POO las clases tendrán alta cohesión cuando se refieran a una única entidad. Podemos garantizar una fuerte cohesión disminuyendo al mínimo las responsabilidades de una clase: si una clase tiene muchas responsabilidades probablemente haya que dividirla en dos o más.
- A mayor cohesión, mejor: el módulo en cuestión será más sencillo de diseñar, programar, probar y mantener.

□ **Principio de ocultación**

- Aísla las propiedades de un objeto contra su modificación por quien no tenga derecho a acceder a ellas. Reduce la propagación de efectos colaterales cuando se producen cambios.

Conceptos de OO. II

□ **Polimorfismo.**

- ▣ Consiste en reunir bajo el mismo nombre comportamientos diferentes.
- ▣ La selección del comportamiento dependerá del objeto que lo ejecute

□ **Herencia.**

- ▣ Relación que se establece entre objetos en los que unos utilizan las propiedades y comportamientos de otros formando una jerarquía. Los objetos heredan las propiedades y el comportamiento de todas las clases a las que pertenecen.

□ **Recolección de basura**

- ▣ Destrucción automática de los objetos ➔ desvinculación de la memoria asociada.

Ventajas OO

- ❑ Desarrollo de software en
 - ▣ menos tiempo
 - ▣ con menos coste
 - ▣ mayor calidad gracias a la reutilización → código reusable en otras Aplicaciones (clases).
- ❑ Aumento de la calidad de los sistemas, haciéndolos más extensibles → facilidad para aumentar o modificar la funcionalidad de la aplicación.
- ❑ Facilidad de modificación y mantenimiento por la modularidad y encapsulación
- ❑ Adaptación al entorno y el cambio con aplicaciones escalables → propiedad para ampliar un sistema sin rehacer su diseño y sin disminuir su rendimiento

Clases, atributos y métodos.

- Los objetos de un sistema se abstraen en **clases** formada por un conjunto de procedimientos y datos.
- Propósito de la clase: definir **abstracciones** y favorecer la **modularidad**
- Miembros:
 - ▣ **Nombre.**
 - ▣ **Atributos:** conjunto de características asociadas a una clase. Definen el **estado** del objeto. Se definen por su nombre y su tipo, que puede ser simple o compuesto como otra clase.
 - ▣ **Protocolo:** Operaciones (métodos, mensajes) que manipulan el estado.
 - Un **método** es el procedimiento o función que se invoca para actuar sobre un objeto.
 - Un **mensaje** es el resultado de cierta acción efectuada por un objeto. El conjunto de mensajes a los cuales puede responder un objeto se le conoce como *protocolo del objeto*.

Visibilidad.

- **Principio de ocultación** → aísla el **estado** de manera que sólo se pueda cambiar mediante las operaciones definidas dentro de una clase → protege los datos de modificaciones por alguien que no tenga derecho a acceder a ellos → las clases se dividan en dos partes:
 - **Interfaz**: visión externa de una clase.
 - **Implementación**: representación de la abstracción, y mecanismos que conducen al comportamiento deseado.
- **Niveles de ocultación** → **visibilidad** → define el tipo de acceso que se permite a atributos o métodos.
 - **Público**: Se pueden acceder desde cualquier clase y cualquier parte del programa.
 - **Privado**: Sólo se pueden acceder desde operaciones de la clase.
 - **Protegido**: Sólo se pueden acceder desde operaciones de la clase o de clases derivadas en cualquier nivel.

Visibilidad. II

- Norma general
 - ▣ Estado (atributos) → privado
 - ▣ Operaciones del comportamiento → públicas
 - ▣ Operaciones auxiliares para definir el comportamiento → privadas/protegidas

Objetos. Instanciación

- Clase → abstracción de un conjunto de objetos.
- Creación de objeto de clase → **instancia de clase**.
- Un objeto se define por:
 - ▣ **Su estado**: definido por el conjunto de valores de atributos.
 - ▣ **Su comportamiento**: definido por los métodos públicos de su clase.
 - ▣ **Su tiempo de vida**: intervalo de tiempo a lo largo del programa en el que el objeto existe, desde su creación (**instanciación**) hasta la destrucción del objeto.
- **Clase abstracta**: no puede ser instanciada.
 - ▣ Uso: definir métodos genéricos para sus clases derivadas.