Resumen Teoría POO

Tema 1 - Introducción a la POO

Podemos diferencias dos tipos de programación según el nivel de abstracción:

- **Programación con estructura de datos (ED):** Bajo nivel de abstracción.
- **Programación tipos abstractos de datos (TAD):** Alto nivel de abstracción.

ED: es un método de almacenamiento de datos en la memoria del ordenador. Es una forma de estructurar los datos para su posterior acceso y manipulación.

TAD: mejora la programación, crea programas más fáciles y menos complejos.

1.1 - Encapsulamiento

Un TAD dispone de dos vistas:

- Una **vista exterior,** en la que se ocultan detalles, estructuras de datos y los secretos de su comportamiento interno.
- Una **vista interior,** en la que se puede ver cómo está hecho por dentro y como se comporta internamente cada operación, las estructuras de datos utilizadas, el código...

Esta separación es muy importante, ya que en el diseño de un TAD no debe intervenir la vista interna. Igual ocurre con el cliente, que no debe conocer el interior (la vista interna) de un TAD para poder usarlo. Es mejor que el cliente se limite a usar **la interfaz del TAD**, es decir, las operaciones del TAD.

Esto tiene como **utilidad** facilitar el diseño y hacerlo de más calidad, además de hacer que la labor del cliente sea más sencilla y potente.

También evita que nadie haga modificaciones de forma inadecuada, protege de operaciones no permitidas sobre los objetos, simplifica la comprensión del código (al no ser necesario conocer sus datos internos), y en el caso futuro en el que se haga una modificación interna a los datos de un objeto, no afectará a ningún programa que utilice dicho objeto, ya que nunca se accede a los datos internos.

1.2 - Software de calidad

Podemos clasificar las cualidades del software en dos tipos:

- **La calidad funcional:** tiene que ver con el grado de acercamiento del funcionamiento del software a lo que se determinó en sus especificaciones y requisitos funcionales.
- **La calidad estructural:** cumple las características internas (no funcionales) que dan soporte a otras características internas (factor de calidad interno) del software.

Basándonos en el modelo de McCall, podemos distinguir diferentes **factores de calidad**, lo cuales se basan en tres ejes:

1 - Operación del producto

Características funcionales y operacionales (pueden ser factores externos o internos):

- **Corrección** (externo): capacidad del producto para realizar de forma adecuada aquello para lo que fue creado.
- **Robustez** (externo): capacidad del producto de realizar aquello por lo que se creó de forma satisfactoria, exacta y precisa.
- **Eficiencia** (externo e interno): capacidad del producto de realizar aquello por lo que se creó de la mejor forma posible.
- **Integridad** (externo): capacidad del producto de no corromperse, comprometerse o ser vulnerable en ninguna de sus partes.
- **Facilidad de uso** (externo): capacidad del producto de ser fácil de utilizar al introducir e interpretar datos, comprender errores, interpretar resultados... Sea cual sea el usuario final.

2 - Transición del producto

Capacidad de adaptación a nuevos entornos:

- **Portabilidad** (externo e interno): capacidad del producto de ejecutarse en otro hardware o sistema operativo diferente.
- **Reutilidad/ Reusabilidad** (interno): capacidad del producto de ser reutilizado en su totalidad o en parte por otros productos.
- **Compatibilidad** (externa): es la capacidad que tienen los programas de combinarse entre sí. La clave es la estandarización.

3 - Revisión del producto

Aptitudes ante la modificación y el mantenimiento del producto (factores de calidad internos):

- **Mantenibilidad** (interno): capacidad de encontrar y corregir un defecto en el software.
- **Extensibilidad** (interno): facilidad de adaptar el producto a cambios en la especificación de requisitos.
- **Testable** (interno): habilidad de poder validarse el software.

Otros factores importantes

Otros factores también importantes son los siguientes:

- **Seguridad** (externo): es la capacidad del producto de proteger sus componentes de usos no autorizados y de situaciones excepcionales de pérdida de información.
- **Accesibilidad** (externo): acceso a la información sin limitación alguna por razones de deficiencia, incapacidad o minusvalía.

- **Oportunidad** (externo): capacidad de un sistema de ser lanzado cuando el usuario o el mercado lo necesita o antes.
- **Economía** (externo): los costes del producto.

Tema 2 - Descomposición. Abstracción y Especificación

2.1 - Criterios modulares

Un método de descomposición modular para ser bueno debe satisfacer los siguientes 5 requisitos fundamentales:

- **1 - Descomposición modular:** debe permitir el problema de software en un pequeño número de subproblemas menos complejos y suficientemente independientes para permitir que el trabajo futuro pueda proseguir por separado en cada uno de ellos.
- **2 - Composición modular:** favorece la producción de elementos de software que se puedan combinar libremente unos con otros para producir nuevos sistemas.
- **3 - Comprensibilidad modular:** capacidad de producir software en el cual un lector pueda entender cada módulo sin tener que conocer a los otros (o muy poco de ellos).
- **4 - Continuidad modular:** un pequeño cambio en la especificación o en los requisitos provoca solo cambios en un único módulo (o en un número reducido de ellos).
- **5 - Protección modular:** si hubiera un error solo afectaría a dicho módulo o en el peor de los casos a los módulos vecinos.

2.2 - Reglas de descomposición modular

Estas 5 reglas deben seguirse para asegurarnos cumplir con los requisitos anteriores:

- **1 - Correspondencia directa:** la estructura modular obtenida debe ser compatible con la estructura del dominio del problema.
- **2 - Pocas interfaces**: un módulo debe comunicarse con el menor número de posibles módulos.
- **3 - Pequeñas interfaces:** los módulos deben intercambiar la menor información posible.
- **4 - Interfaces explícitas:** las interfaces deben ser obvias a partir de su simple lectura.
- **5 - Ocultación de la información:** se seleccionan un subconjunto de propiedades del módulo como información oficial sobre el módulo para ponerla a disposición de los autores de módulos clientes.

2.3 - Principios modulares

A partir de las reglas y criterios anteriores, pueden derivar algunos principios para seguir con una descomposición modular de calidad:

- **1 - Unidades modulares lingüísticas:** los módulos deben corresponderse con las unidades sintácticas del lenguaje de programación que se utilice.
- **2 - Auto-documentación:** el diseñador debe lograr que toda la información relativa al módulo forme parte de él.
- **3 - Acceso uniforme:** todos los servicios ofrecidos por un módulo deben estar disponibles a través de una notación uniforme sin importar su implementación.
- **4 - Principio abierto-cerrado:** los módulos deben ser a la vez abiertos (para su posterior modificación, ampliación...) y cerrados (cuando está disponible para ser usado e integrado en el sistema).
- **5 - Elección única:** solo un módulo accede a una estructura de datos y elige sus opciones.

Tema 3 - Patrones de diseño

El patrón de diseño es un concepto utilizado en la etapa de diseño de software. Es una solución reutilizable a un problema de diseño de software: soluciones de diseño elegantes y de calidad a problemas complejos. Son elementos en el diseño de software que se repiten una y otra vez.

Está compuesto por objetos, clases y las relaciones entre ellos. Cada patrón está especializado en resolver un problema de diseño concreto en un determinado contexto.

Las ventajas que implican el uso de patrones de diseño son: diseños futuros más sencillos, menos costosos y de mayor calidad.

3.1 - Patrones de diseño

Algunos patrones de diseño son:

Nombre	Tipo	Descripción	Aplicaciones	Consecuencias
Template Method	Estructural	Un método describe un comportamiento que se concreta en clases derivadas	Un mismo proceso se concreta de forma distinta en distintas clases	Modificalidad
Composite	Estructural	Jerarquía de objetos con comportamiento parecido y con la misma interfaz	Simplificar interfaces	Simplificación, interfaz, sencilla y acceso uniforme
Parameterized types	Estructural	Define un nuevo tipo sin especificar los tipos de sus componentes	Genericidad	Genericidad
Builder	Creacional	Crear gran variedad de objetos a través de objeto puente	Crear objetos complejos, constructor con muchos	Facilita creación de objetos complejos

			parámetros, muchos atributos, alta complejidad en su configuración	
Factory	Creacional	Crea instancias de clase que pertenecen a la misma familia	Crear objetos de una misma familia por una sola llamada	Sencillez de crear objetos y sencillez ampliar código
Singleton	Creacional	Crear elementos que sean accesibles desde cualquier punto del sistema	Configurar una app compleja	Una instancia y simplifica datos globales
Iterator	Comportamiento	Dar acceso a elementos de un objeto secuencial	Listas, vectores, contendores...	Sencillez, acceso uniforme e independencia interna
Observer	Comportamiento	Esquema de clases cooperan entre sí (sujeto con datos y observadores se nutren de datos)	Muchas clases cooperantes	Elementos independientes, reutilizables por separado y añadir nuevos observadores
Strategy	Comportamiento	Crea algoritmo intercambiable	Apps con distintos comportamientos en el futuro o estructura de datos compleja en futuro	Mejor eficiencia y rendimiento, facilita ampliación y permite estrategia

3.2 - MVC: Modelo-Vista-Controlador

MVC es una triada de patrones. Sus componentes son:

- Modelo: objeto.
- Vista: su representación.
- Controlador: modo en el que reacciona el usuario.

Usa las propiedades de varios patrones de diseño que cooperan: observer, composite, strategy...

Se emplea siempre en el desarrollo de aplicaciones web. Está presente en casi todos los frameworks de desarrollo modernos. Se puede aplicar a cualquier aplicación. Provoca una simplificación del desarrollo, separa desarrollos y proporciona varias presentaciones para un mismo modelo.

Tema 4 - Reutilización

La reutilización beneficia a desarrolladores, consumidores y a la calidad del producto final. Algunos de sus beneficios son:

- Rapidez de desarrollo.
- Fiabilidad.
- Eficiencia.
- Menor mantenimiento.
- Mayor consistencia en nuestros desarrollos.
- Mejora costes y es una inversión rentable.

- Se espera la mejoría de todos los factores de calidad.

La comunidad de desarrolladores nos proporciona las siguientes ventajas:

- Simplificación del desarrollo.
- Detección/Corrección de errores y mantenimiento.
- Portabilidad a otras plataformas.
- Mejora de la funcionalidad.
- Mejora de la interfaz.
- Mayor difusión.
- Aprendizaje, etc.

Tema 5 - POO

5.1 - Pilares de la POO

Los 4 pilares de la POO son:

- **Herencia**: estructura jerárquica de la POO que facilita la reutilización y extensibilidad del código mediante clases. Existen dos tipos:
  - **Clase diferida**: clase a la cuál le faltan métodos por definir y funciona como una interfaz descendiente.
  - **Clase efectiva**: no es diferida.
- **Abstracción**: proceso mental de aislar un elemento de su contexto o del resto de elementos que lo acompañan.
- **Polimorfismo**: ocurre cuando se dispone de la misma interfaz sobre entidades (objetos) de distinto tipo. Existen tres tipos:
  - **Estático**: sobrecarga de funciones y operaciones (se define una para cada tipo).
  - **Dinámico**: se vincula en tiempo de ejecución el código que es independiente del tipi.
  - **Paramétrico**: se vincula en tiempo de ejecución el código que es independiente del tipo.
- **Encapsulamiento**: ocultar la visión interna del tipo/ clase/ objeto a modo de "cápsula" a la que solo se accede con los métodos determinados.

5.2 - STL

Standard Template Library de C++. Es una librería en la que existen una serie de clases y plantillas definidas que se usan fácilmente en los programas. A muchos de estos tipos se les da el nombre de "contenedores", porque de forma genérica pueden "contener" internamente cualquier tipo base.

Están los contenedores clásicos como vector y list. Y otros contenedores como los asociativos: multiset, set, map y multimap.