# Resumen POO

## Tema 1:

### Abstracción

Consiste es facilitar la compresión de un sistema complejo añadiendo una capa de abstracción sobre cada módulo software.

Pasos:

1. Analizar un problema.
2. Identificar cada componente a nivel abstracto y sus funciones.
3. Diseñar una solución con dichos componentes.
4. Implementación.

### Diseño

Proceso: Elaborar un plan de lo que se quiere hacer.

Resultado: Software de calidad.

### TAD (Tipo Abstracto de Dato)

Es un modelo matemático compuesto por una serie de operaciones definidas sobre un conjunto de datos. Durante la implementación se definirá cómo se realizan esas operaciones y dará lugar a las estructuras de datos.

### EncapsulamientoImpide acceso directo al estado interno de un objeto:

1. Impidiendo operaciones no permitidas.
2. Simplificando su comprensión.
3. Sin afectar a ningún programa que use dicho objeto ya que nunca se accede a ellos.
4. Solo podrá interactuarse con el objeto mediante su interfaz pública.

## Tema 2:

### Software de Calidad

Tipos:

* La calidad funcional (externa): funcionamiento del software con sus especificaciones y requisitos.
* La calidad estructural (interna): características internas no-funcionales del software que dan soporte características internas del software como la mantenibilidad, la facilidad de extensión, la reutilización, etc.

Factores de calidad: Internos y externos.

### Revisión del Producto

Mantenibilidad, Flexibilidad y Testabilidad.

### Adaptación a otros entornos e interrelación con otros softwares

Portabilidad, Reutilidad y Compatibilidad.

### Funcionalidad

Corrección, Robustez, Eficiencia, Integridad y Facilidad de Uso.

## Tema 3:

### Descomposición Modular

Los objetos y las clases son independientes entre sí, y la unidad primaria de descomposición modular.

### Criterios

Descomposición, Composición, Comprensibilidad, Continuidad y Protección.

### Reglas

* Correspondencia directa.
* Pocas, Pequeñas y Explícitas interfaces.
* Ocultación de la información.

### Principios

* Unidades modulares lingüísticas.
* Auto documentación.
* Acceso uniforme.
* Puerto abierto-cerrado.
* Elección única.

### Base de la Programación Orientada a Objetos (POO)

* Representa un objeto del mundo real, con atributos, comportamiento y entidad.
* Tiene sentido, se entiende, y se comprende fácilmente.
* Se puede reutilizar.
* Se puede establecer relaciones de distintos tipos.

## Tema 4:

### TDD vs OOD

* Descomposición TDD (Top-Down Desing): Descomposición de arriba a abajo (descomposición funcional).
* Descomposición OOD (Object Oriented Desing): Descomposición orientada a objetos.

### Especificación e Implementación de TAD

* TAD: es una tabla que gestiona un conjunto de enteros y sus estadísticas. Esta realiza una serie de operaciones como: añade, total , infonum e infofrec.
* La especificación informa de manera precisa los detalles relevantes del TAD mediante una serie de operaciones como: concat, deleteDuplicate, binarySearch y factorial.

Requisitos de la implementación: Tiene que ser pequeña, cerrada o abierta, y tiene que seguir una seria de criterios, reglas y principios de descomposición modular.

### Reutilización del Software:

La reutilización tiene 2 aspectos:

* Producción (creación, autoría): desarrollar componentes consumiendo y aprendiendo.
* Consumo: estudia, aprende e imita: El estilo, el diseño, las propiedades de un componente…

¿Qué reutilizar?

* Metodologías de Programación: POO: objetos/clases, herencia, polimorfismo, templates, etc.
* Diseños: los patrones de diseños.

¿Cómo reutilizar?

* Realizando mejoras constantes en estructuras de reutilización: lenguajes de programación, repositorios, comunidades de desarrolladores, etc.

La repetición: detectarla, entenderla y describirla bien para desarrollarla como módulo.

Módulo ideal para la reutilización, es aquel que:

1. Es clara, concreta y bien descrita.
2. Tiene facilidad de uso, con el menor tamaño posible y la mayor calidad.
3. Cumple los criterios, reglas y principios de la descomposición modular.

Obstáculos: falta de formación, fallos en el diseño y gestión, y desconfianza de reutilización fuera de tu propio equipo de desarrollo.

Estructuras de reutilización: rutinas, bibliotecas, librerías, etc.

* Agrupaciones de unidades: Se adaptan mediante argumentos y resuelven problemas concretos y complejos de reducido tamaño.
* Clases: Son piezas modulares reutilizables.
* Paquete: Es similar a una rutina, pero más avanzada.
* Framework/entorno de desarrollo: Facilitan el desarrollo de cualquier aplicación, pueden incorporar elementos estructurales de diseño de aplicaciones y especializarse en plataformas.

## Tema 6:

### Pilares de la Tecnología Orientada a Objetos:

Triángulo básico de la computación, las tres fuerzas de la computación:

Objeto o dato

Acción o funciones

Procesador

Hay 2 enfoques:

1. Enfoque tradicional: Se basa en las funciones y toma decisiones de diseño prematuras y de poca calidad. Le da mucha importancia a la interfaz y es poco adecuado para sistemas de medio/gran tamaño.
2. Enfoque Orientado a Objetos: Se basa en los objetos que se manipulan, presenta un diseño pensando en la reutilización, la extensibilidad, el mantenimiento, y es mejor para sistemas grandes.

Abstracción

Encapsulamiento

Herencia

Polimorfismo

### Abstracción

Clases y objetos.

* Acceso a una clase: se puede acceder mediante el uso de métodos, la interfaz, las funciones (proporcionan + eficiencia) y los atributos (ligados a la representación interna y su acceso dificulta cambios).
* Para el cliente, los atributos y las funciones deben diferenciarse lo mínimo posible.

### Encapsulamiento

Oculta la implementación y la visión interna con un acceso mediante determinados métodos.

Impide el acceso directo al estado interno de un objeto, y hace que solo pueda interactuarse con el objeto mediante su interfaz pública.

### Herencia

Forma una estructura de reutilización jerárquica específica de la POO, facilita la reutilización y la extensibilidad, es útil en la descomposición modular, y forma una clase base y una derivada.

### Polimorfismo

Existe cuando se usa la misma interfaz para objetos de distinto tipo.

Hay varios tipos:

* Estático: Es el + simple y define una función/operador diferente para cada tipo.
* Estático paramétrico: La interfaz es independiente de la instancia y usa plantillas de clase y de función.
* Dinámico: Permite construir código genérico independiente del tipo y usa punteros a la clase base y funciones virtuales.

Sistema Software Orientado a Objetos: Es un método simple que ofrece servicios descritos por el sistema. Se considera sistema cerrado si contiene todas las clases que necesita la clase raíz.

Gestión de memoria: Hay varias formas de gestionar la memoria: memoria estática, en pila, dinámica, desconexión de memoria, aparición de objetos inalcanzables

## Tema 7:

### Patrones de Diseño

Definición: Es una solución para la resolución de problemas de diseño de software.

Contenidos: Está compuesto por objetos, clases y las relaciones entre estos y cada uno está especializado en resolver un problema de diseño concreto en un determinado contexto.

Utilidad: Ayuda a la resolución de los problemas de diseño para que nuestros diseños serán: más sencillos, menos costosos y de mayor calidad.

### Tipos

* Patrones de Comportamiento (BSC): objetos, interacciones, algoritmos, etc.
* Patrones Estructurales (S): componen estructuras de clases entre varios objetos distintos.
* Patrones de Creación (.): crean instancias de objetos.

### Elementos esenciales de un Patrón de Diseño (PD)

* Tipo: BSC
* Estructura: Conjunto de clases/objetos que cooperan.
* Función: Identificar cuando y bajo que circunstancias hay aplicar el patrón.

### Template Method

* “Separa las cosas que cambian de aquellas que permanecen igual.”
* Tipo: S.
* Función: Describe un comportamiento genérico para concretarlo en las clases derivadas mediante el uso de placeholders Ventajas: modificabilidad.

### Parameterized Types o Generics Design Pattern:

* Tipo: S.
* Función: Define un nuevo tipo sin tener que especificar los tipos de todos sus componentes.
* Ventajas: Genericidad.

### Iterator

* Proporcionan una manera de acceder a los elementos de un objeto secuencial. Se puede aplicar en listas, vectores, etc.
* Ventajas: facilidad de uso, favorece la abstracción, independencia de la representación interna y acceso uniforme.

### Observer

* Estructura: un objeto(sujeto) y varios suscriptores que dependen de unos datos (el sujeto). Este patrón de suscripción permite la actualización continua de los datos.
* Esta estructura de tipo B se define por el par: sujeto-observador.
* El sujeto contiene todos los datos que se publican y registra a los observadores (suscriptores) en su interfaz. Puede haber varios observadores del mismo sujeto.
* El sujeto y el observador son independientes. Las vistas son un componente del MVC.

### Composite

* Características: similitud entre objetos de distintos niveles en el sistema y una jerarquía de tareas.
* Función: construir cosas a partir de otras más pequeñas pero similares. En la POO consiste en coger objetos pequeños para construir otros + grandes y complejos, se agrupan componentes para construir super componentes.
* En este patrón de tipo S, se tratan de la misma manera a los objetos individuales y a los compuestos.
* Ventajas: simplificación, interfaz más sencilla, y el acceso uniforme.

### Strategy

* Es un patrón de tipo B que permite que se intercambie un algoritmo por otro dependiendo del problema en cuestión. Se usa cuando se prevé que algo tendrá distintos comportamientos en el futuro o cuando hay estructuras de datos complejas que podrían implementarse de distintas formas en el futuro.
* Ventajas: posibilidad de mejorar la eficiencia y facilita las ampliaciones.

### Strategy vs Template Method

* Strategy: Permite que un objeto cambie su comportamiento en tiempo de ejecución, intercambiando el algoritmo.
* Template: Permite que una función se desarrolle de formas distintas, mediante el uso de placeholders.

### MVC

* Es un grupo de tres elementos vinculados entre sí, de tipo S.
* Está presente en la mayoría de los frameworks de desarrollo modernos y se puede aplicar en cualquier aplicación.
* Ventajas: simplificación del desarrollo, flexibilidad, desacoplación de funciones, y reparto de roles (frontend, backend).

### Builder

* Es un patrón de tipo C, que facilita la creación y configuración de objetos complejos. Se usa cuando el proceso de construcción y configuración de un objeto es complejo
* Ventaja: Facilita la creación de objetos complejos.

### Factory

* Es un patrón de tipo C, que facilita la creación de instancias de diferentes clases que pertenecen a la misma familia. En este existen 2 implementaciones:

1. Mediante el “Factory Method”, que recibe una descripción del objeto deseado y devuelve un puntero de la clase base que apunta a ese objeto.
2. Mediante la “Abstract Factory”, que define la interfaz de cada subfactory.

* Ventajas: Facilita la manipulación de objetos pertenecientes a una misma familia y la ampliación de clases de la misma familia.

### Software Libre (SL)

* Un SL es aquel que puede ser copiado/modificado por cualquier persona a nivel de código fuente.
* El creador del SL, Richard Stallman tambien fue el creador de GNU, le siguió Linus Torvalds, creador de Linux.

Las 5 libertades del SL: Usar, copiar, estudiar, modificar y redistribuir

### Open Source vs. Software Libre:

* Open Source es un método de desarrollo que sacrifica la propiedad intelectual a cambio de obtener cooperación.
* Software Libre es un movimiento social.

Un proyecto de software libre se coordina a través de internet, foros, etc. El Software Libre se financia a través de contribuciones altruistas, y empresas que quieren desarrollar el sector y que quieren captar clientes. Hay que saber que éste no es lo mismo que Software Gratis, se puede vender y puede atraer inversiones.

Hay 2 modos de negocio:

* Servicios: Configuración, Mantenimiento, Programación, etc.
* Productos: El portfolio, empresas ofertando SL, etc.

### Licencias

Propiedad intelectual: al crear una obra, ésta ya está protegida y su uso requiere autorización.

Licencias del SL:

* GNU General Public License (GNU GLP): Garantiza la libertad de ejecutar, estudiar, modificar, mejorar un programa para cualquier propósito, de redistribuir copias y de volver a publicarlo.
* BSD: Permite que trabajos derivados sean redistribuidos como software propietario.
* GPL: Posee un mecanismo copyleft y Open Software Definition.
* Creative Commons: Se tiene libertad para adaptar, transformar, compartir, copiar y redistribuir el material en cualquier medio o formato.

Mientras que se sigan los siguientes términos la licenciante no puede revocar esas libertades:

1. Atribución: se debe dar crédito de manera adecuada e indicar si se han realizado cambios.
2. NoComercial: no se puede hacer uso del material con propósitos comerciales.
3. CompartirIgual: si remezcla, transforma o crea a partir del material, debe distribuir su contribución bajo la misma licencia del original.

Ejemplos de Software Libre: Apache, Chrome, Firefox, Android, Python, Java, compiladores como son gcc y g++, etc.

Distribuciones:

* GNU/Linux: Programas de utilidad + Kernel.
* Otras distribuciones: Debian, Ubuntu, etc.