Unidad Nº1: Introducción a la POO

*Preguntas orientadoras*

1. Se denominan **paradigmas de programación** a las formas de clasificar los [lenguajes de programación](https://es.wikipedia.org/wiki/Lenguajes_de_programaci%C3%B3n) en función de sus características. Según los conceptos en que se basa un lenguaje de programación, se tienen distintas maneras y estilos para plantear la solución.

El paradigma de programación que se utilizó hasta ahora es la **Programación procedural**, también llamada **Programación Imperativa (C).** La idea básica es definir los algoritmos o procedimientos más eficaces para tratar los datos del problema.

Las características son:

* Código y datos separados, sin ninguna conexión formal.

* Programa: es un conjunto de llamadas a funciones (secuencia de instrucciones que describen la solución).

1. Un **TAD**es un **encapsulamiento** que contiene la definición de un nuevo tipo de datos y todas las operaciones que se pueden realizar con él (matriz, string, complejo). **TERMINAR**
2. La **Programación Orientada a Objetos (POO)** es un paradigma de programación, es decir, un modelo o un estilo de programación que nos da unas guías sobre cómo trabajar con él. Se base en el concepto de clases y objetos.

**Las ventajas de la POO son:**

* Reutilización de código:

- Ahorra tiempo en el desarrollo de programas.  
- Empleo de software que ya ha sido probado.

* Fácil mantenimiento y depuración de los programas.
* Extensibilidad: posibilidad de ampliar la funcionalidad de la aplicación de manera sencilla.
* Modularidad y encapsulamiento: el sistema se descompone en objetos con  
  responsabilidades claramente especificadas.

**Las características de la POO son:**

* [**Abstracción**](https://es.wikipedia.org/wiki/Abstracci%C3%B3n_(programaci%C3%B3n_orientada_a_objetos))**:** Permite seleccionar las características relevantes dentro de un conjunto e identificar comportamientos comunes para definir nuevos tipos de entidades.
* [**Encapsulamiento**](https://es.wikipedia.org/wiki/Encapsulamiento_(programaci%C3%B3n_orientada_a_objetos))**:** Cada objeto está aislado del exterior. Cada tipo de objeto expone una interfaz a otros objetos donde se presenta el prototipo de las funciones que puede realizar, el  
  usuario no necesita conocer la implementación. Facilita el mantenimiento y depuración de los programas. Ante una variación en el código de la clase los usuarios no necesitarán cambiar sus líneas de código si no cambia su interfaz.
* [**Polimorfismo**](https://es.wikipedia.org/wiki/Polimorfismo_(programaci%C3%B3n_orientada_a_objetos))**:** Permite utilizar el mismo nombre para funciones de comportamientos diferentes (asociadas a objetos distintos), por ejemplo una función *suma* () que calcule la suma de matrices o de números complejos. Cuál de los métodos será invocado se resuelve en *tiempo de ejecución* cuando un objeto particular lo necesite. Esta característica se denomina asignación tardía o dinámica. Algunos lenguajes proporcionan también medios más estáticos (*tiempo* de compilación): plantillas (templates) y sobrecarga de operadores.
* [**Herencia**](https://es.wikipedia.org/wiki/Herencia_(programaci%C3%B3n_orientada_a_objetos))**:** Permite definir nuevas clases (subclase o derivada) en base a otras ya existentes (base o superclase). Una subclase hereda todos los métodos y atributos de su superclase, además de poder definir miembros adicionales (ya sean datos o funciones). Las subclases también pueden redefinir (override) métodos definidos por su superclase: responden al **mismo mensaje** que su superclase con su propio método (polimorfismo).
* [**Modularidad**](https://es.wikipedia.org/wiki/Modularidad_(programaci%C3%B3n_orientada_a_objetos)) **:** Permite 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. Estos módulos se pueden compilar por separado, pero tienen conexiones con otros módulos.
* **Cohesión y acoplamiento:** el principal objetivo al plantear el diseño de una aplicación es minimizar el costo de mantenimiento, por los cambios que surgen en el sistema. Un diseño efectivo minimiza la probabilidad de que se propaguen los cambios. Se debe **reducir** el **acoplamiento** entre componentes y **aumentar** la **cohesión** interna de cada uno.
* **Acoplamiento:** Grado de interdependencia que hay entre los distintos módulos de un programa. Dos elementos están acoplados (no deseable) en la medida en el que los  
  cambios en uno tienden a necesitar cambios en el otro. Un acoplamiento simple se presenta cuando un componente accede directamente a un dato que pertenece a otro.
* **Cohesión:** Medida de fuerza o relación funcional existente entre las sentencias o grupos de sentencias de un mismo módulo. Un módulo cohesionado (deseable) ejecutará una única tarea sencilla interactuando muy poco o nada con el resto de módulos del programa. Un elemento puede tener poca cohesión tanto por ser muy grande o muy pequeño.
* **Clase (class):** Es un tipo de dato definido por el usuario (TAD). Se puede considerar como  
  una generalización de las *estructuras* de C.
* **Objetos, instances (instancias) o data objects:** Son las variables concretas que se crean de una determinada clase, la cual describe su estructura (información y comportamiento), mientras que el estado de la instancia se define por las operaciones ejecutadas. Un objeto tiene su propio conjunto de datos o variables (atributos) y funciones (métodos) miembro. Para proteger a las variables de modificaciones no deseadas se recurre al concepto de *encapsulamiento, ocultamiento* o *abstracción de datos*.

**Nivel de acceso a los miembros de una clase:**

* **Public:** Se pueden acceder libremente desde fuera de la clase. Los métodos y atributos públicos forman la **interfaz** del objeto, a través de la cuál se comunica con otros objetos.
* **Private:** Sólo pueden ser accedidos por las funciones y operadores de la misma clase.
* **Protected:** Son privados para las funciones externas pero públicos para las clases derivadas.

1. Los métodos de una clase se definen anteponiendo a su nombre el nombre de la clase y *el operador de resolución de visibilidad*

(**::** = *scope resolution operator*), de la forma:

*tipo\_dato\_retorno* <nombre\_clase>**::**<nombre\_metodo> (parametros) { .......; }

Mediante esta notación se puede distinguir entre funciones que tengan el mismo nombre pero distintas visibilidades y permite también acceder a funciones desde puntos del programa en que éstas no son visibles.

1. **UML** son las siglas de **U**nified **M**odeling **L**anguage (Lenguaje Unificado de Modelado).No es un lenguaje de programación propiamente dicho, sino una serie de **normas** y **estándares gráficos** respecto a cómo se deben representar los esquemas y diagramas relativos al software.

**Tipos de Diagramas UML:**

* **De clases:** Para UML una clase es una entidad. Puede ser un diagrama o representación de conceptos que intervienen en un problema, o un diagrama de clases software (class). El sentido de se lo da la persona que lo construye.
* **De estados:** Se usan para representar objetos de software y el intercambio de mensajes entre ellos, representando la aparición de nuevos objetos de izquierda a derecha. Se puede utilizar también para indicar cómo evolucionan los métodos en el tiempo.
* **De colaboración:** Representan objetos o clases y la forma en que se transmiten mensajes y colaboran entre ellos para cumplir un objetivo.
* **De casos de uso:** Representan a los actores y casos de uso (procesos principales) que intervienen en un desarrollo de software.
* **Otros diagramas:** De actividad, de paquetes, de arquitectura de software, etc.

