|  | **Carrera:** Técnico Universitario en Programación  **Materia:** Programación II  **Tema:** Resumen temas POO |
| --- | --- |

**RESUMEN DE POO**

**Un programa de POO** puede entenderse como la interacción de objetos de una o varias clases, dentro de un contexto específico definido (un sistema con límites que hace algo determinado).

Tiene **2 ámbitos principales de programación**: el **programa**, y la **clase**.

El programa es donde se produce la **interacción entre los objetos**. Puede estar contenido en uno o varias clases, o en funciones globales. Las características del programa dependen del lenguaje que se esté utilizando.

En las **clases** es donde se establece que tienen y que pueden hacer los objetos.

**Definiciones básicas**

**Objeto:** entidad del mundo real o virtual sobre el cual se desea registrar información, o que forma parte de la porción de la realidad que se quiere representar mediante un programa de computadora. En términos de programación son variables de una determinada clase. También se define a un objeto como una instancia de una clase.

**Clase:** plantilla o molde donde se definen las características (propiedades) y el comportamiento (métodos) de todos los individuos pertenecientes a esa clase. En términos de programación las propiedades son las variables de la clase, y los métodos las funciones de la clase; una clase es la definición de un tipo de dato.

**Programación de una clase**

**c**lass Nombre{

private:/// propiedades

public: ///métodos

};

**class:** palabra clave que indica que se está programando una clase. **Nombre** es el nombre que el programador decide darle a la clase. Está sujeto a las mismas restricciones que se aplican a los nombres de las variables.

**private y public** son **especificadores de acceso.** Indican la accesibilidad de los elementos contenidos en la clase. Lo que se encuentra en la parte privada (**private**) sólo es accesible dentro de la clase (generalmente se colocan las propiedades); lo que se encuentra en la parte pública (**public**) es accesible tanto dentro como fuera de la clase (por lo general los métodos). Dentro de la clase de manera directa; fuera de la clase mediante un objeto de esa clase. Por lo general se colocan en la parte pública los métodos.

La idea de colocar las propiedades en la parte privada es para garantizar que todos los accesos al contenido de un objeto se hagan de manera controlada, por medio de un método definido en la misma clase. Esto recibe el nombre de **encapsulamiento.**

**Diseño de una clase**

Cuando se programa una clase se piensa primero en lo que debe tener un objeto (sus propiedades), y luego en lo que tiene que hacer (sus métodos). Como se señaló más arriba la interacción entre objetos debe ser resuelta en el programa, por lo que la clase debe enfocarse en resolver los problemas de un solo objeto.

**Métodos:** los métodos son las funciones de la clase y determinan el comportamiento de los objetos. Los más usuales son

**Constructores:** tienen el mismo nombre de la clase. No devuelven nada, y pueden recibir parámetros. Son llamados de manera automática cuando se declara un objeto. Sirven para crear y/o asignar valores iniciales a las propiedades.

**Destructores:** tienen el mismo nombre de la clase antecedido por el símbolo ~. No devuelven nada, y no reciben parámetros. Son llamados de manera automática cuando se termina la vida de un objeto. Sirven para asociar código que se ejecuta cuando el objeto cae fuera de su ámbito (por ejemplo, para devolver memoria).

**Sets()**: permiten asignar valor a las propiedades. Puede dentro de ellos establecer criterios de validación para los datos.

**Gets():** permiten que el mundo exterior a la clase conozca el valor de las propiedades de los objetos.

Los métodos deben responder a la naturaleza de los objetos de la clase que se está programando. Por ejemplo, si una clase sirve para crear objetos gráficos, necesitará métodos para moverlo, presentarlo, ocultarlo, etc. Si el objeto es un registro que se graba en un archivo puede tener métodos para escribirlo en un archivo o leerlo de un archivo.

**Sobrecarga de operadores**

Otros métodos pueden desarrollarse en C++ por medio de la **sobrecarga de operadores**. Este mecanismo permite agregarles funcionalidad a los operadores del lenguaje, de manera de simplificar luego la operación entre los objetos. Por ejemplo, puede sobrecargarse el operador == para comparar de manera directa dos objetos de esa clase,

Si fuera útil, es posible tener más de una sobrecarga de un mismo operador, ya que el lenguaje acepta la **sobrecarga de funciones**, que nos permite tener varias funciones con el mismo nombre. La exigencia es que las funciones se diferencien entre ellas por los parámetros que reciben.

Otros elementos que pueden ser útiles en la programación de una clase son:

**Puntero this:** contiene la dirección del objeto que llama al método. Se lo suele definir como un puntero oculto, ya que no es imprescindible usarlo. En caso que sea de utilidad se lo puede utilizar. Para hacerlo hay que poner el nombre this dentro del método; según sea el caso, con el operador -> se puede acceder a las propiedades o los métodos. Sólo existe dentro de los métodos de una clase.

**Referencias**: una referencia es el alias de una variable. Es muy útil para pasar por referencia una variable como parámetro a una función, ya que evita hacer una copia local de la variable en la función. Una referencia se declara anteponiendo el & a un nombre.

Por ejemplo la función void f1(int &x), recibe una referencia a una variable de tipo entero. Al tener la referencia a la variable puede leer o escribir en la variable original.

**Funciones con parámetros con valores por defecto (o por omisión):** en C++ es posible asignar valores por defecto para los parámetros. De esta manera, si no se envían valores, se utilizan los definidos por defecto; si se envían, se usan esos valores. Los valores por defecto deben indicarse en el prototipo, o en la cabecera de la función. Por ejemplo:

void f2(int x=5, char letra=’o’);

Si la función f2() es llamada sin valores, se usan el 5 para x y la ‘o’ para letra; si se envían otros valores en la llamada se pisan los establecidos por defecto por los enviados.

**Mecanismos para la creación de clases**

**Herencia:** consiste en utilizar una clase, llamada **base**, para construir un conjunto de clases a las que se denomina **derivadas**. Las clases derivadas van a contener las propiedades y métodos definidas en la clase base, con excepción de los constructores.

Este mecanismo es útil cuando se dispone de un conjunto de clases que pueden ser agrupadas en una categoría de orden superior que las contenga (vaca, caballo, perro y gato **son** animales). Se define en la clase base lo que es común para todos, y luego se agrega lo particular en cada clase derivada.

La forma de asociar una clase con una clase base se realiza de la siguiente manera:

class Derivada: public Base{…};

de esta manera la clase Derivada contendrá todo lo contenido en la Base. La forma de heredar puede ser public, private, o protected. Este último es un especificador de acceso que se utiliza para que las clases derivadas puedan tener acceso directo a lo definido como protected en la clase base. Recordar que lo privado restringe el acceso a la misma clase.

**Composición:** consiste en tener un objeto de una clase como propiedad de otra clase. Por ejemplo, un objeto pedido contiene una fecha, por lo que en lugar de definir cada una de las propiedades de una fecha dentro de pedido, se declara un objeto fecha como propiedad de la clase pedido. El objeto agregado por composición mantiene el encapsulamiento, por lo que no podrá accederse a sus propiedades de manera directa si no por medio de los métodos de su clase.