**LOS PILARES DE LA POO**

**1)** Recordemos que tres son las principales características de un lenguaje orientado a objetos, es decir, se considera que un lenguaje está totalmente orientado a objetos si es capaz de proveer estas tres características:

* Encapsulación
* Herencia
* Polimorfismo

**2)**  La **ENCAPSULACIÓN** es la cualidad de unificar los datos y la forma de manipularlos, de esta forma podemos ocultar el funcionamiento de una clase y exponer solo los datos que manipula (mediante propiedades), así como proveer de medios para poder manipular dichos datos (mediante métodos).

**3)** La **HERENCIA** es la cualidad de poder crear nuevas clases (o tipos) basadas en otras clases, de forma que la nueva clase obtenga todas las características de la clase que ha heredado, tanto los datos que contiene como la forma de manipularlos, pudiendo añadir nuevas características e incluso cambiar el comportamiento de algunas de las incluidas en la clase base, (siempre que así se haya previsto). Mediante la herencia podemos crear de forma fácil una jerarquía de clases que comparten un mismo comportamiento básico pero que cada nueva generación puede tener (y de hecho tiene) un nuevo comportamiento.

**4)** El **POLIMORFISMO** es la cualidad de implementar de forma particular algunas de las características que tienen las clases, de forma que cuando necesitemos usarlas no nos preocupe la implementación interna que cada una tenga, lo que realmente nos interesa o nos debe importar es que podemos usar esas características e incluso podamos acceder a ellas de forma anónima... o casi.

### 5) LAS CLASES Y ESTRUCTURAS

En los lenguajes orientados a objetos, existe el concepto clase. Cuando hablamos de clases, también podemos extenderlo a estructuras.

Una clase es una pieza de código en la que podemos definir una serie de datos y al mismo tiempo unos métodos (funciones o procedimientos) que nos permitirán acceder a esos datos.

Cuando definimos una clase, lo que estamos haciendo es definir una plantilla, a partir de la cual podemos crear objetos en la memoria. Por tanto, la clase es el molde con el cual podemos crear nuevos objetos. Para poder crear algo "tangible" a partir de una clase, tenemos que crear en la memoria un nuevo objeto del tipo de la clase, en estos casos lo que decimos es que instanciamos un nuevo objeto de la clase.

**6)** La diferencia principal entre una clase y una estructura es la forma en que se crean los objetos que representan a esas "ideas". Los objetos creados a partir de las clases son objetos por referencia, es decir, si declaramos una variable para manipular ese objeto, lo que tendremos será una referencia (o puntero) a una dirección de memoria en la que realmente está el objeto.

**7)** Las interfaces representan un contrato, de forma que cualquier clase que la implemente debe utilizar los miembros de la interfaz usando la misma forma en que ésta la ha descrito: mismo número de argumentos, mismo tipo de datos devuelto, etc.

**8)** Los constructores son procedimientos especiales (funciones que no devuelven un valor) en los que podemos escribir toda la lógica que debe usarse para la correcta creación del objeto. Por ejemplo, podemos inicializar las variables usadas, podemos asignarle algunos valores predeterminados, etc.

### LOS MIEMBROS DE LAS CLASES: CAMPOS, PROPIEDADES Y MÉTODOS

### 9) Los campos no son otra cosa que variables declaradas en la clase, habitualmente declaradas de forma privada.

**10 Las propiedades** son funciones especiales que nos permiten acceder a esos datos, aunque para ser más precisos, las propiedades realmente representan a los datos que una clase contiene, al menos de forma pública.

**11) Los métodos** nos permitirán realizar acciones sobre los datos, por ejemplo devolver un rango de valores o simplemente una representación amigable de la información contenida.

### EL ÁMBITO DE LOS MIEMBROS DE LAS CLASES

1. **Private .** Para uso privado. Cuando declaramos un miembro como privado sólo lo podremos acceder desde la propia clase. Este es el más restrictivo y el que se recomienda para los campos y las funciones de uso interno.
2. **Protected .** Uso protegido. Los elementos declarados como protegidos sólo estarán accesibles, además de en la propia clase, por cualquier clase derivada.
3. **Friend.** Para uso dentro de la propia aplicación. Cuando declaramos un miembro con este modificador, solo podremos acceder a él desde la propia clase o desde cualquier código que se encuentre en el mismo ensamblado (proyecto).
4. **Protected Friend.** Una mezcla de Protected y Friend, es decir solo accesible desde las clases derivadas o desde el mismo proyecto.
5. **Public.** Este modificador de ámbito nos permite exponer públicamente cualquier miembro de la clase, de forma que no haya restricciones para acceder a él.