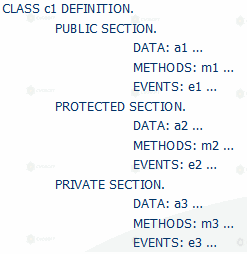
1. Las Propiedades de la Programación Orientada a Objetos

Existen tres propiedades fundamentales que presentan los objetos:

* **Encapsulación**
* **Herencia**
* **Polimorfismo**

1. **Encapsulación**

También conocido como **“ocultamiento”**, se refiere a que todos los objetos restringen la visibilidad de sus recursos (atributos y métodos) al resto de los usuarios. Las tres áreas de visibilidad (pública, privada y protegida) son la base de la encapsulación. Cada objeto posee una interfase que determina la manera de interactuar con él. La implementación (su interior) es encapsulada, lo que quiere decir que desde fuera el objeto es invisible, simplemente se usa.



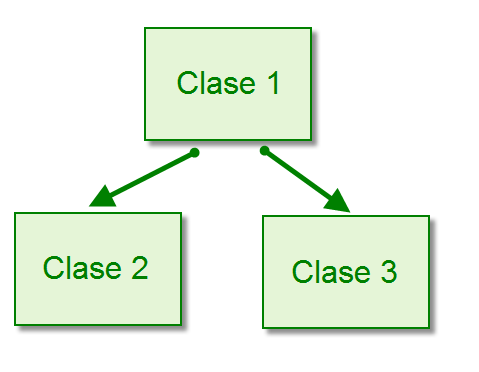
Cuando se define una clase hay que tener mucho cuidado en el diseño de los componentes públicos, intentando declarar tan pocos como sea posible. Los componentes públicos de las clases globales no pueden ser cambiados una vez que se ha liberado la clase. Además de definir la visibilidad de un atributo, se puede proteger también de los cambios usando la adición **READ-­ONLY.**

1. Herencia

Es la capacidad que tiene una clase de heredar las propiedades y métodos de otra clase. La herencia permite crear una nueva clase a partir de una existente, heredando la nueva clase sus propiedades. Esto se realiza añadiendo la adición **INHERITING FROM** a la sentencia de definición de la clase:

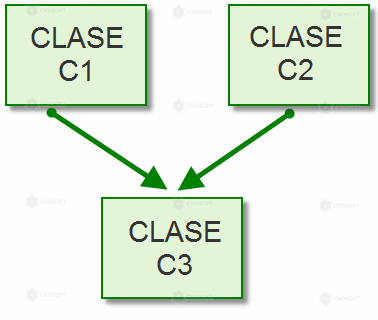
CLASS <SUBCLASS> DEFINITION INHERITING FROM <SUPERCLASS>. La clase original se conoce como la superclase de la nueva clase que será la subclase. Solo los componentes públicos y protegidos son visibles a la subclase. Aunque los componentes privados de la superclase existen en la subclase, no son visible.

Denominaremos Especialización a la relación donde una clase, la subclase hereda las principales características de la superclase. La Generalización es lo inverso. Ocurre si la superclase no tiene una sección privada la subclase es una réplica exacta de la superclase. Podemos añadir nuevos componentes en la subclase, convirtiendo así a la subclase en una versión especializada de la superclase. Se obtiene un nuevo nivel de especialización si la subclase es a su vez una superclase de otras subclases.



**TIPOS DE HERENCIA:**

* **Simple:** una clase puede tener más de una subclase de las cuales es superclase, pero sólo puede tener una superclase de la cual es subclase

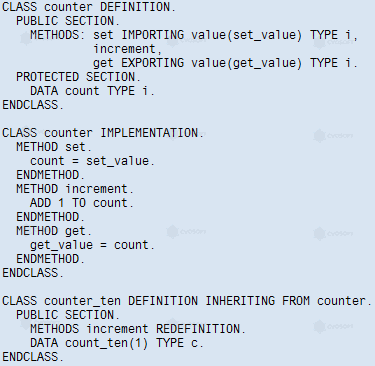


* **Múltiple:** una clase hereda de varias superclases. Donde se forma una estructura de árbol en la cual el grado de especialización aumenta con cada nivel jerárquico que se añada.

El nodo raíz de todos los árboles de herencia en los objetos ABAP es la clase predefinida vacía OBJECT.

Las adiciones ABSTRACT y FINAL en las sentencias Method y Class (en método y clases) permiten definir métodos o clases abstractas y finales. Un método abstracto se define en una clase abstracta y no puede ser implementado en esa clase sino en una subclase de esta. Las clases abstractas no pueden ser instanciadas. Un método final no puede ser redefinido en una subclase. Las clases finales no pueden tener subclase, estas son las que finalizan el árbol de herencia.

ABAP solo soporta herencia simple.

1. Polimorfismo

Significa que métodos que se llaman exactamente igual pueden comportarse de manera distinta en clases diferentes. Conocido como REDEFINICIÓN, consiste en conseguir que un objeto de una clase se comporte como un objeto de cualquiera de sus subclases, dependiendo de la forma de llamar a los métodos de dicha clase o subclases.

Se utiliza la adición REDEFINITION en la sentencia METHODS para redefinir un método público o protegido dependiente de instancia en una subclase y hacer que realice una función más especializada. Ejemplo:

La implementación de la redefinición en la subclase ‘oculta’ la implementación original en la superclase. Esto se aplica particularmente a la referencia a sí mismo **me-­>**.

Ejemplo de un método M1 de una superclase: CALL METHOD [ME->]M2

Dentro de un método redefinido se puede usar la referencia **SUPER­->** para acceder al método ‘oculto’. Esto permite usar la funcionalidad existente en el método de la superclase sin tener que codificarla de nuevo en la subclase.

Ejemplo de polimorfismo:

Tenemos una clase y una subclase y ambas tienen definido un mismo método, pero cada una de ellas presenta una declaración particular del método, adaptado a sus propias necesidades. Por ejemplo supongamos que tenemos la clase VEHÍCULO y las subclases CAMIÓN, BARCO y AVIÓN. En la clase VEHÍCULO podemos tener definido el método CARGAR\_NAFTA y luego en cada subclase tendremos una implementación particular del método, ya que no es lo mismo cargar nafta a un camión, barco y avión.