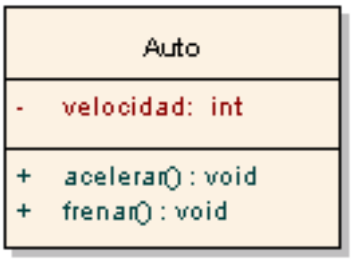
La Visibilidad

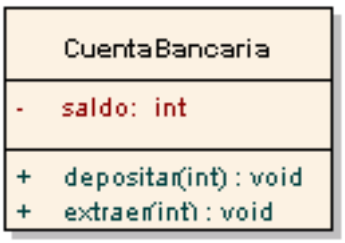
La visibilidad o accesibilidad determina el alcance que tiene un atributo u operación. El alcance (scope) de un miembro representa qué partes pueden acceder a dicho miembro. El alcance está determinado por los modificadores de visibilidad, presentados como "público" (su simbología es el signo +) y como "privado" (su simbología es el signo -)

Que un atributo u operación sea privado significa que aquel atributo u operación solo puede ser accedido desde dentro del objeto que lo contiene. En cambio si este atributo u operación fuera público podría ser accedido por cualquier objeto.

En el siguiente ejemplo podemos ver los miembros públicos y privados de la clase "Auto" diferenciados por los signos "-" y "+":



Esto mismo se puede aplicar con la clase "CuentaBancaria":



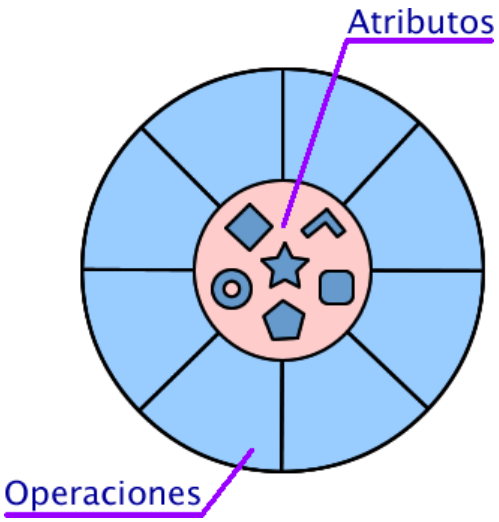
Una de las dudas que surgen a la hora de entender los modificadores de acceso es "¿por qué debería de tener un miembro privado?". Hacer que un atributo sea privado se utiliza para tratar con datos o valores los cuales no queremos que sean accedidos más que por el objeto contenedor. Las operaciones privadas cobran sentido como procesamientos internos del objeto que no queremos sean accedidas por otros objetos (tanto porque no tiene sentido que la accedan otros objetos como porque podría resultar peligroso para la integridad del objeto contenedor).

El porqué de utilizar miembros privados se aclarará a continuación junto con el tema "Encapsulamiento".
En resumen, la visibilidad determinan el alcance (scope) del atributo u operación en cuestión.

El Encapsulamiento

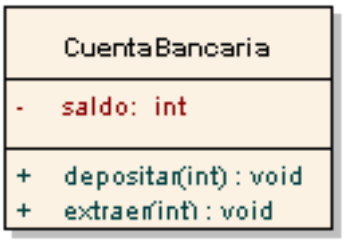
El encapsulamiento es una de las características representativas de la POO. Este concepto tiene que ver con la forma correcta de pensar en clases y objetos. Significa aislar o independizar las clases de nuestro programa de tal manera que su funcionamiento sea independiente del de las otras clases.

La clase "Automóvil" posee un atributo privado que representa la velocidad y posee una operación pública que le permite interactuar con otros objetos. Esta operación recibe como parámetro un valor que representa la velocidad que se le quiere asignar a dicho automovil. Si esta operación recibiera un valor negativo, podría pasarlo a un número positivo, mantener la velocidad anterior o hacer cualquier otra cosa evitando que aparezcan errores en el objeto por un dato inválido (velocidad negativa). Si en vez de utilizar un atributo privado y una operación pública, utilizáramos directamente un atributo público, este tipo de casos no podrían ser manejados y se atentaría contra la integridad del objeto.
La ocultación puede visualizarse mejor en el siguiente gráfico:



El gráfico representa que los atributos están encerrados por operaciones, es decir que los atributos son privados y las operaciones son públicas. Esto significa que para poder acceder a un atributo es necesario utilizar una operación, es decir que no hay posible acceso al atributo sino no se utiliza una operación.

A continuación se presenta el encapsulamiento con la clase "CuentaBancaria":



Para poder modificar el atributo saldo, es necesario hacerlo a través de las operaciones depositar o extraer.

En resumen, el encapsulamiento es la capacidad que tienen las clases de permitir que ciertos miembros sean accesibles y ciertos miembros no lo sean, y así lograr ocultar el estado interno de sus instancias.

Las principales ventajas que ello aporta son:

- Se facilita a los programadores que vayan a usar el tipo de dato el aprendizaje de cómo trabajar con él, pues se le pueden ocultar todos los detalles relativos a su implementación interna y sólo dejarle visibles

aquellos que puedan usar con seguridad. Además, así se les evita que cometan errores por manipular inadecuadamente miembros que no deberían tocar.

- Se facilita al creador del tipo la posterior modificación del mismo, pues si los programadores no pueden acceder a los miembros no visibles, sus aplicaciones no se verán afectadas si éstos cambian o se eliminan. Gracias a esto es posible crear inicialmente tipos de datos con un diseño sencillo aunque poco eficiente, y si posteriormente es necesario modificarlos para aumentar su eficiencia, ello puede hacerse sin afectar al código escrito.

La encapsulación se consigue añadiendo modificadores de acceso en las definiciones de miembros y tipos de datos.