**ENCAPSULAMIENTO**

La encapsulación se refiere al ocultamiento de los datos miembros de un objeto, es decir, encapsular los atributos y métodos del objeto, de manera que sólo se pueda cambiar mediante las operaciones definidas para ese objeto.

Entonces la encapsulación es un mecanismo de protección o aislamiento de atributos y métodos, es decir, el aislamiento protege a los datos asociados de un objeto contra su modificación por quien no tenga derecho a acceder a ellos, eliminando efectos secundarios e interacciones en cuanto al ocultamiento de los datos miembros de un objeto.

En otros términos, es la capacidad de visibilidad de atributos y métodos de un objeto, esta visibilidad va de acuerdo al nivel de encapsulamiento, tenemos tres niveles principales:

**Niveles de encapsulamiento**

* **Nivel cerrado:** los atributos y métodos del objeto sólo es accesible desde la misma clase.
* **Nivel protegido:** los atributos y métodos del objeto sólo es accesible desde la clase y las clases que heredan
* **Nivel abierto:** los atributos y métodos del objeto puede ser accedido desde cualquier clase.

Estos niveles se manejan mediante los modificadores de acceso: privado (privated), protegido (protected) y público (public).

**Modificadores de acceso**

El modificador de acceso **privated**, corresponde al nivel cerrado de acceso.

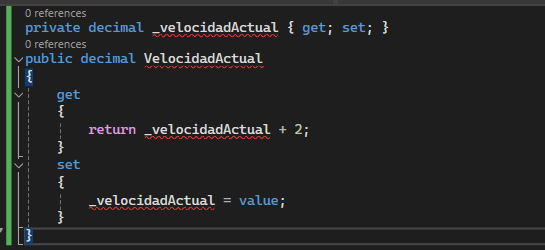
El modificador de acceso **protected** corresponde al nivel protegido.

El modificador de acceso **public** corresponde al nivel abierto.

El encapsulamiento en la programación orientada a objetos es cuando limitamos el acceso o damos un acceso restringido de una propiedad a los elementos que necesita un miembro y no a ninguno más.

El elemento más común de encapsulamiento son las clases, donde encapsulamos y englobamos tanto métodos como propiedades.

Otro ejemplo muy común de encapsulamiento son los getters y setters de las propiedades dentro de una clase. Por defecto nos dan el valor “normal” pero podemos modificarlos para que cambie.



En el ejemplo que acabamos de ver tenemos dos propiedades, ambas hacen referencia a la velocidad actual, pero hay ligeras diferencias.

Una es privada, por lo que desde fuera de la clase no podemos acceder a su valor.

* **Public** acceso total
* **Private**, acceso únicamente desde la clase o struct que los contiene
* **Internal**, acceso desde el mismo Proyecto
* **Protected** desde la misma clase o desde las clases que heredan
* **Protected** **internal**, combina protected e internal y nos permite acceder desde el mismo Proyecto o de clases que la derivan.
* **Private** **protected**, que permite acceder desde la clase actual o de la que derivan de ella.

**El ejercicio 1.1 y 1.2 son lo mismo**

**Ejemplo 1.1 C#**

Crearemos una clase Persona, como atributos: nombre, fecha de nacimiento y edad, como métodos: registrar persona y calcular edad.

Bien, como dije en la teoría, la encapsulación es un mecanismo de protección o aislamiento de atributos y métodos de un objeto contra modificaciones imprevistas o incontroladas. Para ellos debemos tener criterio para aplicar el nivel de encapsulamiento de los atributos o métodos del objeto/clase.

Por lo tanto, teniendo criterio y aplicando conocimientos de abstracción, el atributo nombre, fecha de nacimiento, y el método registrar persona, deben de tener la posibilidad de ser accedida desde fuera de la clase, así poder asignar valores e invocar el método del objeto. Entonces estos atributos y el método deben de ser públicos.

Por otro lado, el atributo edad y método calcular edad, solo deben ser accedidas por la misma clase; así la propia clase asignaría el valor de la edad de la persona, como también usar el método para calcular la edad a partir de la fecha de nacimiento de la persona y la fecha actual (edad exacta asignado por propia clase, además recuerden que la edad no siempre será el mismo, cambia al pasar los años). En cambio, si la edad tendría el modificador de acceso en público, este atributo puede ser accedido desde cualquier parte, y así tener la posibilidad asignar la edad con datos erróneos.

el atributo edad es privado, por lo tanto, no es posible asignar un valor y obtener el valor desde fuera de la clase.

**Ejemplo 1.2 C# – Propiedades**

También en programación orientada a objetos, es recomendable que los atributos siempre sean privados y manipularlos por métodos Get y Set, o propiedades, además es buena práctica y ventajoso. En el ejemplo anterior, el atributo edad es privado, por lo tanto, no es posible asignar un valor y obtener el valor desde fuera de la clase. Pero hay la necesidad de poder obtener la edad desde fuera de la clase (desde la instancia- objeto). Se podría colocar la edad como público, pero no tendría sentido, debido a los inconvenientes mencionados en el ejemplo anterior.

Bueno, es aquí donde los descriptores de acceso (métodos Get, Set o propiedades) entran en juego. Es posible que valor del atributo edad solo pueda ser definida por la misma clase (privado), pero el valor del atributo edad se pueda obtener desde fuera de la clase (publico).

Para ello podemos de indicar que el método Get (Obtener valor) sea público, y el método Set (Asignar valor) sea privado.

**Ejercicio2:**

Crear una clase Libro que contenga los siguientes atributos:

– ISBN

– Titulo

– Autor

– Número de páginas

Crear sus respectivos métodos get y set correspondientes para cada atributo.

Crear el método toString() para mostrar la información relativa al libro con el

siguiente formato:

«El libro con ISBN creado por el autor tiene páginas»

En el fichero main, crear 2 objetos Libro (los valores que se quieran) y mostrarlos

por pantalla.

Por último, indicar cuál de los 2 tiene más páginas.

**Ejercicio3:**

Realizar una clase llamada **Raices**, donde representaremos los valores de una ecuación de 2º grado. Tendremos los 3 coeficientes como atributos, llamémosles a, b y c.

Hay que insertar estos 3 valores para construir el objeto.

Las operaciones que se podrán hacer son las siguientes:

**obtenerRaices**(): imprime las 2 posibles soluciones

**obtenerRaiz**(): imprime única raíz, que será cuando solo tenga una solución posible.

**getDiscriminante**(): devuelve el valor del discriminante (double), el discriminante

tiene la siguiente formula, (b ^ 2)-4\*a\*c

**tieneRaices**(): devuelve un booleano indicando si tiene dos soluciones, para que esto ocurra, el discriminante debe ser mayor o igual que 0.

**tieneRaiz**(): devuelve un booleano indicando si tiene una única solución, para que esto ocurra, el discriminante debe ser igual que 0.

**calcular**(): mostrara por consola las posibles soluciones que tiene nuestra ecuación, en caso

de no existir solución, mostrarlo también.

**Formula ecuación 2º grado: (-b±√((b ^ 2) - (4 \* a \* c)))/ (2 \* a)**

Solo varia el signo delante de -b

**Ejercicio4:**

**Desarrollo de un Sistema de Gestión de Bibliotecas con Diferentes Modificadores de Acceso**

Se te pide desarrollar un sistema de gestión de bibliotecas utilizando C# y el principio de encapsulamiento en programación orientada a objetos, empleando diferentes modificadores de acceso (private, protected, internal, y public). El sistema debe permitir la administración de libros, usuarios y préstamos de libros.

Requisitos:

1. **Clase Libro**:
   * Propiedades con diferentes modificadores de acceso: ID (private), Título (protected), Autor (internal), Género (protected internal), Disponible (public).
   * Métodos públicos para acceder y modificar estas propiedades de manera controlada.
   * Método para mostrar información del libro.
2. **Clase Usuario**:
   * Propiedades con diferentes modificadores de acceso: ID (private), Nombre (protected), Apellido (internal), LibrosPrestados (protected internal).
   * Métodos públicos para acceder y modificar estas propiedades de manera controlada.
   * Método para mostrar información del usuario.
3. **Clase Prestamo**:
   * Propiedades con diferentes modificadores de acceso: Usuario (internal), Libro (private), FechaPrestamo (protected internal), FechaDevolucion (protected).
   * Métodos públicos para crear y gestionar un préstamo.
   * Método para mostrar información del préstamo.
4. **Clase Biblioteca**:
   * Propiedades con diferentes modificadores de acceso: ListaLibros (private), ListaUsuarios (protected), ListaPrestamos (internal).
   * Métodos públicos para agregar libros, usuarios y registrar préstamos.
   * Método para devolver un libro y actualizar la disponibilidad.
   * Métodos para buscar libros y usuarios.

### Resumen de Uso en el Ejercicio:

* **private:** Protege los datos internos para que solo sean accesibles dentro de la misma clase.
* **protected:** Permite que las clases derivadas accedan a los datos, facilitando la extensión de las clases.
* **internal:** Facilita la colaboración dentro del mismo ensamblado, manteniendo los datos inaccesibles desde fuera.
* **protected internal:** Combina la visibilidad dentro del ensamblado y en las clases derivadas, proporcionando una flexibilidad adicional.
* **public:** Hace que los datos sean accesibles desde cualquier parte del código, útil para miembros que necesitan ser ampliamente accesibles.