Encapsulamiento

En general, pilar de 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**.

En programación orientada a objetos, generalmente (no siempre) es recomendable que los atributos tengan un nivel encapsulación cerrado, es decir, los atributos deben de ser privados y manipularlos mediante sus métodos Get y Set (Java u otros) o Propiedades (.Net). Para mayor seguridad, podemos agregar validaciones en los métodos Get y Set.

El fin de tener atributos o métodos privados del objeto, es evitar que el usuario de la clase pueda cambiar su estado de manera imprevista e incontrolada.

Puedes usar el nivel de encapsulación protegido, cuando una clase tendrá clases que deriven de ella (Herencia), y es necesario que las clases hijas (clases derivadas) requieran acceso a los atributos de la clase padre (súper clase), entonces los atributos o métodos serán protected.

Puedes usar el nivel de encapsulación abierto (atributos o métodos públicos), si quieres obviar la implementación de los métodos y propiedades, para así concentrarse solamente en cómo usarlos, de esa manera el desarrollo de software será más ágil y rápido.

Sin embargo, elegir el nivel de encapsulación no es porque uno quiere así por así, debemos de tener criterio (abstracción) para tener atributos o métodos, privados, públicos o protegidos.

Para entender mejor realicemos algunos ejemplos.

Ejemplo 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.

Clase Persona

```
public class Persona
{

     //Atributos
     public string nombre;
     public DateTime fechaNacimiento;
     private int edad;

     //Métodos
     public void registrar()
     {
          calcularEdad(fechaNacimiento);
          Console.WriteLine(nombre+ " con "+edad+ " años de edad, ha sido registrado correctamente");
     }

     private void calcularEdad(DateTime fechaNacimientoPersona)
     {
          DateTime fechaActual = DateTime.Now;
          edad= fechaActual.Year - fechaNacimientoPersona.Year;
     }
}
```

Clase Principal

```
static void Main(string[] args)
{
     Persona alumno = new Persona();
     alumno.nombre = "Roger";
     alumno.fechaNacimiento =Convert.ToDateTime("10/10/1996");
     alumno.registrar();
     Console.ReadKey();
}
```

Ejemplo 2 C# – Propiedades

También comenté, 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.

Clase Persona

```
public class Persona
{

      //Atributos
      private string _nombre;
      private DateTime _fechaNacimiento;
      private int _edad;

      //Propiedades
      public int Edad
      {
           get //Encapsulación nivel abierto, por defecto es publico porque la propiedad es publico.
           {
                return _edad;
           }

           private set //Encapsulación nivel cerrado- Privado
           {
                _edad = value;
           }
      }

      public string Nombre
      {
           get
           {
                return _nombre;
           }

           set
           {
                _nombre = value;
           }
     }

     public DateTime FechaNacimiento
     {
           get
           {
                return _fechaNacimiento;
           }

           set
           {
                _fechaNacimiento = value;
           }
}

      //Métodos
      public void registrar()
      {
           calcularEdad(FechaNacimiento);
           Console.WriteLine(Nombre+ " ha sido registrado correctamente");
      }
```

```
      private void calcularEdad(DateTime fechaNacimientoPersona)
      {
           DateTime fechaActual = DateTime.Now;
           Edad= fechaActual.Year - fechaNacimientoPersona.Year;
      }
}
```

Clase Principal

```
static void Main(string[] args)
{
     Persona alumno = new Persona()
     alumno.Nombre = "Roger";
     alumno.FechaNacimiento =Convert.ToDateTime("10/10/1996");
     alumno.registrar();
     Console.WriteLine("la edad del alumno "+alumno.Nombre+" es "+alumno.Edad);
     //el valor de la edad se puede obtener por que la propiedad GET es Publico.
     Console.ReadKey();
}
```

Sin embargo, no se puede asignar el valor de edad.

```
alumno.Edad = 18;
//Marcará error por el nivel de protección
```

Bibliografía:

- Extraído de <<Programación Orientada a Objetos - Encapsulamiento(POO - Parte3)>>. 2022. Consultado el 28 de noviembre del 2022