Apuntes Clase 2

Programación Orientada a Objetos (P.O.O)

Definición

- Es una manera de construir software basada en un nuevo paradigma.
- Propone resolver problemas de la realidad a través de identificar objetos y relaciones de colaboración entre ellos.
- El **objeto** y **mensaje** son sus elementos fundamentales.
- Se basa en 4 pilares -> Abstracción, Encapsulamiento, Herencia, Polimorfismo

Abstracción

Básicamente la abstracción es analizar un objeto de la realidad, ya sea físico o no físico y tener la capacidad de pasarlo a una plantilla (una clase).

- Ignorancia selectiva. Interesa el contexto, lo importante, datos relevantes.
- Decide qué es importante y qué no lo es.
- Se enfoca en lo importante.
- Ignora lo que no es importante
- Utiliza la encapsulación para reforzar la abstracción.

Encapsulamiento

Determina que acceso le damos a la información de nuestra clase.

- Esta característica es la que denota la capacidad del objeto de responder a peticiones a través de sus métodos o propiedades sin la necesidad de exponer los medios utilizados para llegar a brindar estos resultados.
- El exterior de la clase lo ve como una caja negra.

Herencia

La herencia es uno de los conceptos más cruciales en la POO. La herencia básicamente consiste en que una clase puede heredar sus variables y métodos a varias subclases (la clase que hereda es llamada superclase o clase padre). Esto significa que una subclase, aparte de los atributos y métodos propios, tiene incorporados los atributos y métodos heredados de la superclase. De esta manera se crea una jerarquía de herencia.

Relación “es un” significa que la clase hija (o heredera), es, además, lo mismo que su padre. Es decir, un auto “es un” transporte, un caballo “es un” animal, etc.

Estos pueden compartir (y extender) su comportamiento sin tener que re implementar su comportamiento. Esto suele hacerse habitualmente agrupando los objetos en clases y las clases en árboles o enrejados que reflejan un comportamiento común.

- La relación entre clases es del tipo “es un tipo de”.
- Va de la generalización a la especialización.
- Clase base o padre.
- Clase derivada o hija.
- Hereda la implementación.

Polimorfismo

El término de polimorfismo también define la capacidad de que más de un objeto puedan crearse usando la misma clase de base para lograr dos conceptos de objetos diferentes, en este caso podemos citar el típico ejemplo de los teléfonos, los cuales se basan en un teléfono base, con la capacidad de hacer *ring* y tener un auricular, para luego obtener un teléfono digital, inalámbrico, con botonera de marcado y también, tomando la misma base, construir un teléfono analógico y con disco de marcado.

- La definición del método reside en la clase base o padre.
- La implementación del método reside en la clase derivada o hija.
- La invocación es resuelta al momento de la ejecución.

¿Qué es una clase?

Definición

Las clases son declaraciones de objetos, también se podrían definir como abstracciones de objetos. Esto quiere decir que la definición de un objeto es la clase. Cuando programamos un objeto y definimos sus características y funcionalidades en realidad lo que estamos haciendo es programar una clase.

La clasificación se basa en un comportamiento y atributos comunes. Permite crear un vocabulario estandarizado para comunicarse y pensar dentro del equipo de trabajo.

- Una clase es una Clasificación.
- Clasificamos en base a comportamientos y atributos comunes.
- A partir de la clasificación se crea un vocabulario.
- Es una abstracción de un objeto.

Sintaxis

```
[modificador] class Identificador
{
    // miembros: atributos y métodos
}
```

- Modificador: Determina la accesibilidad que tendrán sobre ella otras clases. Puede estar o no, si no esta por defecto es internal.
- class: Es una palabra reservada que le indica al compilador que el siguiente código es una clase.
- Identificador: Indica el nombre de la clase.

- Los nombres deben ser sustantivos, con la primera letra en mayúscula y el resto en minúscula.
- Si el nombre es compuesto, las primeras letras de cada palabra en mayúsculas, las demás en minúsculas.
- Ejemplo: Mi Clase

Modificadores de Clases

- internal (*): Accesible en todo el proyecto (Assembly)
- public (*): Accesible desde cualquier proyecto.
- private (*): Accessor por defecto.
- sealed: Indica que la clase no podrá heredar. Es una clase que no puede instanciarse.
- abstract: Indica que la clase no podrá instanciarse. Es sellada, no se puede heredar.

(*): Modificador de visibilidad

SE LE PUEDE AGREGAR A OTROS MODIFICADORES.

Atributos

Definición

En la programación orientada a objetos, los atributos son una propiedad o característica que se puede asignar a un objeto (elemento). Mediante el uso de atributos se pueden asignar valores específicos a ciertos elementos.

```
[modificador] tipo identificador;
```

- modificador: Determina la accesibilidad que tendrán sobre él las demás clases. Por defecto son **private**.
- tipo: Representa al tipo de dato. Ejemplo: int, foat, etc.
- Identificador: Indica el nombre del atributo.
  - Los nombres tener todas sus letras en minúsculas.
  - Si el nombre es compuesto, las primera letra de la segunda palabra estará en mayúsculas, las demás en minúsculas.
  - Ejemplo: string miNombre;

Modificadores de Atributos

- private (*): Los miembros de la misma clase.
- protected: Los miembros de la misma clase y clases derivadas o hijas.
- internal: Los miembros del mismo proyecto.
- internal protected: Los miembros del mismo proyecto o clases derivadas.
- public: Cualquier miembro. Accesibilidad abierta.

(*): Acceso por defecto

Métodos

Definición

En la programación, un método es una subrutina cuyo código es definido en una clase y puede pertenecer tanto a una clase, como es el caso de los métodos de clase o estáticos, como a un objeto, como es el caso de los métodos de instancia.

```
[modificador] retorno Identificador ( [args] )
{
    // Sentencias
}
```

- modificador: Determina la forma en que los métodos serán usados.
- retorno: Representa el tipo de dato devuelto por el método (solo retornarán un único valor). Ejemplo: int, foat, etc.
- Identificador: Indica el nombre del método.
    - Los nombres deben ser verbos, con la primera letra en mayúscula y el resto en minúscula .
    - Si el nombre es compuesto, las primeras letras de cada palabra en mayúsculas, el resto en minúsculas .
    - Ejemplo: AgregarAlumno
- **args:** Representa una lista de variables cuyos valores son pasados al método para ser usados por este. Los corchetes indican que los **parámetros son opcionales**
- Los parámetros se definen como:
    - tipo identificador
- Si hay más de un parámetro, serán separados por una coma.
- Si un método no retorna ningún valor se utilizará la palabra reservada **void**
- Para retornar algún valor del método se utilizará la palabra reservada **return**

Modificadores de Métodos

- abstract: Sólo la firma del método, sin implementar. Define una implementación, funciona como un .h en C.
- extern: Firma del método (para métodos externos).
- Internal (*): Accesible desde el mismo proyecto.
- override: Reemplaza la implementación del mismo método declarado como **virtual** en una clase padre.
- public (*): Accesible desde cualquier proyecto.
- private (*): Solo accesible desde la clase.
- protected (*): Solo accesible desde la clase o derivadas.
- static: Indica que es un método de clase.
- virtual: Permite definir métodos, con su implementación, que podrán ser sobrescritos en clases derivadas.

(*): Accesor por visibilidad.

Ejemplo implementación de Atributos y Métodos

```
public class Automovil
{
    // Atributos NO estáticos
    public Single velocidadActual;
    // Atributos estáticos
    public static Byte cantidadRuedas;
    // Métodos estáticos
    public static void MostrarCantidadRuedas()
    {
        Console.Write(Automovil.cantidadRuedas);
    }
    // Métodos NO estáticos
    public void Acelerar(Single velocidad)
    {
        this.velocidadActual += velocidad;
    }
}
```

Namespace

Definición

Los espacios de nombres (namespaces) son una agrupación lógica de clases y otros elementos del código fuente. Así como clasificamos nuestros archivos dentro de distintos directos del sistema operativo, podemos organizar el código fuente de C# y sus componentes en distintos espacios de nombres.

- Es una agrupación lógica de clases y otros elementos.
- Toda clase está dentro de un NameSpace.
- Proporcionan un marco de trabajo jerárquico sobre el cual se construye y organiza todo el código.
- Su función principal es la organización del código para reducir los conflictos entre nombres. Esto hace posible utilizar en un mismo programa componentes de distinta procedencia.
- Ejemplo:
    - System.Control.WriteLine()
    - Siendo:
        - **System** es el NameSpace de la BCL (Base Class Library)
        - **Console** es una clase dentro del NameSpace **System.**
        - **WriteLine** es uno de los métodos de la clase **Console.**

Directivas

- Son elementos que permiten a un programa identificar los **NameSpaces** que se usarán en el mismo.
- Permiten el uso de los miembros de un **NameSpace** sin tener que especificar un nombre completamente cualificado.
- C# posee dos directivas de **NameSpace:**
    - Using.
    - Alias.

Using

- Permite la especificación de una llamada a un método sin el uso obligatorio de un nombre cualificado.

```
using System; //Directiva USING

public class Program
{
    public static void Main()
    {
        Console.WriteLine("Hola");
    }
}
```

Alias

- Permite utilizar un nombre distinto para un **NameSpace**.
- Generalmente se utiliza para abreviar nombres largos.

```
using SC = System.Console; //Directiva ALIAS

public class Program
{
    public static void Main()
    {
        SC.WriteLine("Hola");
    }
}
```

Miembros

Definición

```
namespace Identificador
{
    // Miembros
}
```

- Dónde el identificador representa el nombre del NameSpace.
- Dicho nombre respeta la misma convención que las clases.

Pueden contener ...
Clases
Delegados
Enumeraciones
Interfaces
Estructuras
Namespaces
Directivas using
Directivas Alias