Programación Orientada a Objetos

2

Programación II y Laboratorio de Computación II

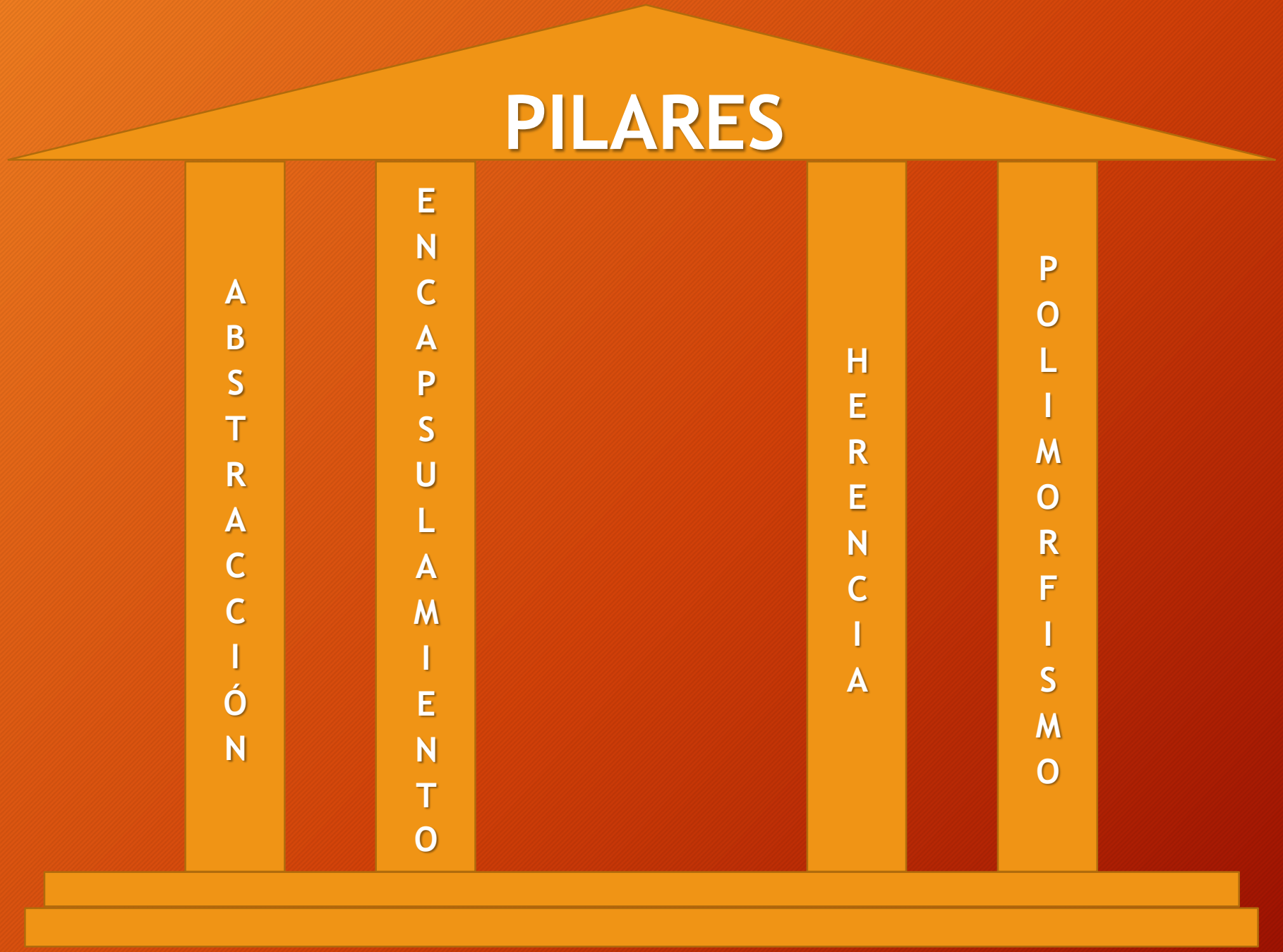
Programación Orientada a Objetos (P.O.O.)

- Es un **paradigma de programación.**
  - Un **paradigma** es una teoría o conjunto de teorías cuyo núcleo central se acepta sin cuestionar y que suministra la base y modelo para resolver problemas y avanzar en el conocimiento.

- **Es decir, un enfoque o modelo a seguir para resolver ciertos problemas.**

- **Propone resolver problemas de la realidad a través de identificar objetos y relaciones de colaboración entre ellos.**

PILARES



Abstracción

- **Abstraer** es aislar conceptualmente las propiedades o funciones concretas de algo (entidad, objeto, concepto) ignorando otras cualidades.

- Decide qué es importante y qué no lo es.
  - Se enfoca en lo que es importante.
  - Ignora lo que no es importante.

- En programación orientada a objetos consiste en seleccionar las características relevantes y comportamiento **en común** dentro de un conjunto de objetos, definiendo nuevos tipos de entidades.

- La abstracción es clave en el proceso de análisis y diseño orientado a objetos, ya que mediante ella podemos llegar a armar un conjunto de clases que permitan modelar la realidad o el problema que se quiere resolver.

Clases: Definición

- Una clase en el paradigma orientado a objetos es un tipo de clasificador.
- Representan entidades o conceptos resultantes del proceso de abstracción.
  - Clasificamos en base a comportamientos y atributos esenciales y en común de un grupo de objetos.
- Funciona como plantilla o molde para la creación de objetos.

Clases: Sintaxis

- **modificador:**
  - Determina la accesibilidad que tendrán sobre ella otras clases.
- **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 en UpperCamelCase.
    - 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: *MiClase*

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

Clases: Modificadores

Nombre	Descripción
abstract	Indica que la clase no podrá instanciarse.
internal (*)	Accesible en todo el proyecto (Assembly).
public (*)	Accesible desde cualquier proyecto.
private (*)	Accesor por defecto.
sealed	Indica que la clase no podrá heredar.

(*): Modificadores de visibilidad.

Objetos

Los objetos son entidades que tienen:

- **Identidad:**
  - Es una propiedad que permite identificarlos. Como un nombre. Análogo a los identificadores de variables.
- **Estado o Atributos:**
  - Son los datos del objeto, sus características o propiedades.
- **Comportamiento u Operaciones:**
  - Es lo que el objeto puede hacer, las acciones y funcionalidades.
- **Relaciones:**
  - Se relacionan y comunican de distintas formas con otros objetos.

Objetos

- La definición de un objeto, es una clase. Es decir, los objetos se crean a partir de las clases que funcionan como molde.
- **Instanciar** es leer las definiciones en una clase y crear un objeto a partir de ellas.
- Un **objeto o instancia de una clase** contiene valores en sus atributos independientes de otros objetos, incluso entre objetos de la misma clase.

Atributos

- La clase define los atributos de un tipo de objetos.
- Los atributos contendrán valores específicos de cada instancia de la clase, es decir, de cada objeto.
- Cada objeto tiene su propio conjunto de datos.
- En C# a los atributos se los llama también fields o campos.

Atributos: Sintaxis

[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, float, etc.
- **Identificador**: Indica el nombre del atributo.
  - Los nombres deben estar escritos en lowerCamelCase.
    - Tener todas sus letras en minúsculas. Si el nombre es compuesto, la primera letra de la segunda palabra estará en mayúsculas, las demás en minúsculas.
    - Ejemplo: *string miNombre;*

Atributos: Modificadores

Nombre	Puede ser accedido por...
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

Atributos Estáticos

- Son atributos o propiedades asociadas a la clase y no a una instancia de la misma.
- No se necesita instanciar un objeto para acceder a estos atributos.
- No pueden acceder a los atributos no-estáticos, los cuales son específicos de cada instancia.
- Se declaran utilizando la palabra reservada **static**.
- Se llaman utilizando el nombre de la clase + punto + nombre del atributo o propiedad.
  - Console.Title
  - Math.PI

Métodos

- Para implementar las operaciones de un tipo de objeto se utilizarán **métodos.**
- Los métodos estarán declarados dentro de la clase correspondiente así como su implementación.
- Los métodos contendrán algoritmos que producirán cambios en los atributos del objeto.
- Los métodos se ejecutarán cuando sean llamados a través de un **mensaje.**
- Los métodos pueden llegar a generar un nuevo mensaje para otro objeto a través de **eventos.**

Métodos: Sintaxis

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

- **modificador:**
  - Determina la forma en que los métodos serán usados.
- **retorno:**
  - Es el tipo de valor devuelto por el método (sólo retornan un único valor).
- **Identificador**: Indica el nombre del método.
  - Los nombres deben ser verbos en UpperCamelCase.
    - 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: *AgregarAlumno*

Métodos: Sintaxis

- **args**: Representan 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 usará la palabra reservada **void**.
- Para retornar algún valor del método se utilizará la palabra reservada **return**.

Nombre	Descripción
abstract	Sólo la firma del método, sin implementar.
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 (*)	Sólo accesible desde la clase.
protected (*)	Sólo 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 de visibilidad

Métodos Estáticos

- Son operaciones asociadas a la clase y no a una instancia de la misma.
- Son lo más parecido a las funciones de los lenguajes estructurados.
- Se utilizan para procesar datos de entrada y retornar un resultado sin necesidad de acceder al estado / atributos de un objeto concreto.

Métodos Estáticos

- No se necesita instanciar un objeto para llamar a estos métodos.
- No pueden acceder a los atributos no-estáticos, los cuales son específicos de cada instancia.
- Se declaran utilizando la palabra reservada **static**.
- Se llaman utilizando el nombre de la clase + punto + nombre del método.
  - Console.WriteLine()
  - Math.Pow()

Ejemplo

```
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

- Es una agrupación lógica de clases y otros elementos.
- Toda clase está dentro de un NameSpace.
- Proporcionan un marco de trabajo jerárquico sobre el cuál 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.

Namespace

- System.Console.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 completamente 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");
    }
}
```

Namespace: Sintaxis

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

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

Miembros

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

Tipos de Programación Orientada a Objetos

Basada en Clases:

- Más utilizada.
- Se basa en crear una estructura molde llamada clase donde se especifican los atributos y métodos que tendrán nuestros objetos.
- Cada vez que necesitamos un objeto creamos una instancia (o copia del objeto) usando la clase como molde.
- C#, C++, JAVA

Tipos de Programación Orientada a Objetos

Basada en Prototipos:

- No hay clases, solo hay objetos.
- Se crean directamente objetos y cuando se quiere generar otro con la misma estructura se usa clonación.
  - Una vez clonado si queremos podemos agregar los campos y métodos necesarios.
- Un objeto prototípico es un objeto que se utiliza como una plantilla a partir de la cual se obtiene el conjunto inicial de propiedades de un objeto.
- Cualquier objeto puede ser utilizado como el prototipo de otro objeto, permitiendo al segundo objeto compartir las propiedades del primero.
- Es soportado en Javascript, Python y Ruby.