Tema 7: Programación Orientada a Objetos

- Definición
- Clasificación
- Encapsulación
- Clases
- Nomenclatura
- Constructores
- Sobrecarga

Definición

- La programación orientada a objetos (POO) permite cambiar el paradigma de progamación, prestando atención en el objeto y sus relaciones con el resto.
- Cambia la filosofía respecto a ala programación tradicional en la que el foco de desarrollo era la función.
- Ejemple POO – Tradicional (imprimir tabla)

Clasificación

- Organización de elementos con significado común.
- Produciendo una relación directa con los objetos con los que nos rodean. En P.O.O. permite definir clases que contienen:
  - Comportamiento: métodos (funciones)
  - Atributos: datos (variables)

Encapsulación

- Organizar diferentes elementos y dotarles de la misma estructura.
- Se emplea para:
  - Combinar métodos y datos en una clase → Definición (class)
  - Controlar el acceso los métodos y datos en una clase → Control de acceso (public, private)

Definición clases

```
class nombre
{
    Métodos → Métodos.
    Variables→Campos
}

class Circulo
  {
    double radio;

    double Area()
    {
      return 3.1416 * radio * radio;
    }
  }
```


Control de acceso

- Public: Accesible por exterior e interior de la clase.
- Private: Accesible por interior de la clase.
- Protected: Accesible por interior de la clase y los hijos.
  - Omisión: private

Nomenclatura

- public: Notación camello, comenzando por mayúscula.Ej: Circulo, Area.
- private: Notación camello, comenzando por minúscula. Ej: radio, colorBorde, tramaRelleno

Contructores

- Clase: Clasificación de elementos que comparten métodos y atributos. Ej: Todas las personas tienen edad, nombre y pueden escribir su nombre.
- Objeto: Instanciación de un elemento de la clase. Ej: Puede haber 20 objetos de la clase persona con edades y nombres diferentes. Todos podrán escribir su nombre.

Constructores

- Para crear una variable en C#, hay que llamar al constructor. Según tipo de datos:
  - Primitivos: int,long,float, decimal, doucle, char, string.
  - Datos de clase (objetos): llamar a un método constructor.

```
<NombreClase>  <identidicador>;
<identidicador> = new Constructor;
```


Constructor

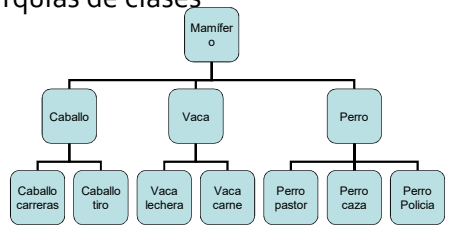
- Constructor por defecto:
  - Numeros a 0.
  - Ref a null.
  - Bool a false.

Sobrecarga

- Defínción de varios métodos que tengan el mismo identificador pero se diferencian por distintos parámetros: por número o por tipo diferente.
- Cuando se realza la llamada se determina el método según los parámetros.
- Cuando un mismo método tiene varias versiones con distintos parámetros se dice que está ***sobrecargado.***

Herencia

- Clasificación de clases con relaciones de tipo padre hija.
- Mediante la herencia se consiguen establecer jerarquías de clases



Herencia

- class <claseHija> : <clasePadre>
- La clase Hija **hereda** de la clase Padre, sólo se puede heredar de una única clase.
- Cuando una clase hija hereda de una clase padre, puede acceder a:
  - Sí puede acceder a atributos y métodos public.
  - No puede acceder a atributos y métodos private.
  - Sí puede acceder a atributos y métodos protected.

Herencia

```
Class Mamifero
{
   Int extremidades;
   String nombreCientifico;
   String nombreComun;
}
Class Caballo
{
   String colorCrin;
   String tipoBocado;

}
Class Vaca
{
   String tipoPelaje;
}
Class CaballoCarreras
{
   Int añosCorriendo;
   Int campeonatosGanado;
   Int velocidadMaxima;
}
Class CaballoTiro
{
   Int cargaMaxima;
   Int cargaHabitual;
}
el
```


Herencia

- Cuando se crea un objeto de la clase derivada, se llama primero al constructor de la clase padre, si no se indica el constructor explícitamente se llamará al constructor por defecto.
- Se puede escoger a qué constructor de la clase padre se quiere llamar con la palabra reservada:

  base(parámetros del constructor)

Herencia

- La sintaxis de la llamada al constructor padre será:

```
class ClaseHija : ClasePadre
{
  public ClaseHija(par1,par2,par3) : base(par1,par2)
  {
    // Constructor hija
  }
```

Polimorfismo

- Ocurre cuando una clase base y una clase derivada declaran el mismo método con el mismo nombre y los mismos parámetros
- La clase hija genera un mensaje indicando que el método hijo oculta al método padre. Como ambos métodos comparten el nombre, la llamada se liga al método hijo.
- No es posible al hijo poder acceder al método público del padre.

Polimorfismo

- Solución 1: palabra *new*
  - Indicar la palabra new en el método hijo.
  - Esta indicación no modifica el hecho de que un método oculta al otro, pero elimina el aviso del compilador.

Polimorfismo

- Solución 2: *virtual / overrride*
  - Permite conectar dos métodos de una clase padre y una clase hija y declarar dos implementaciones para el mismo método.
  - Padre → Indicar la palabra *virtual*
  - Hijo/s → Indicar la palabra *override*

```
Clase padre
{
  virtual public NombreMetodo()
}
// Indica que es la primera implementación    del método
Clase hija
{
  override public NombreMetodo()
}
```