Tema 5: POO

Asignatura: Desarrollo de interfaces

CS Desarrollo de Aplicaciones Multiplataforma

Introducción

- En este capítulo veremos aspectos como:
  - Objetos
  - Clases
  - Métodos

Concepto de POO

- POO es un conjunto de reglas a seguir para hacernos la tarea de programar más fácil.
- Estas reglas son independientes del lenguaje en que trabajemos, que es un conjunto de instrucciones entendibles directamente o traducibles al lenguaje del ordenador con el que trabajemos

Concepto de POO

- Para comprender mejor el concepto de POO, pensemos en el proceso de conducción de un vehículo.
- En un coche podemos identificar **entidades** como puede ser el chasis, el motor, las ruedas, etc.

Concepto de POO

- Un vehículo puede verse como un objeto que tiene unos **atributos**:

  fabricante, modelo, color, potencia, velocidad, número de la marcha, etc.;

- Tiene un **conjunto de acciones** que puede realizar como arrancar, acelerar, frenar, parar el motor, cambiar de marcha, girar, etc.

Concepto de POO

- Por otro lado, pensemos en el conductor del vehículo.
- Desde el punto de vista de la conducción, el conductor tiene unos **atributos**: nombre, edad, antigüedad del carnet, etc.
- Las **acciones o métodos** que puede realizar un conductor serían pisar el freno, pisar el acelerador, pisar el embrague, encender las luces, etc.

Concepto de POO

- Si con el vehículo ya en marcha queremos disminuir la velocidad, el **objeto conductor** enviaría el **mensaje frenar** al **objeto vehículo**.

- Este mensaje lo realiza mediante la **acción pisar el freno**. La respuesta a este mensaje sería la ejecución por parte del vehículo de la **acción o método frenar.**

Concepto de POO

- Es el vehículo el que realiza las acciones necesarias usando el sistema de frenado para disminuir la velocidad.

- En este proceso, **al conductor** no le interesa cómo está construido y cómo actúan los frenos, **sólo quiere que el objeto vehículo responda** de forma adecuada al mensaje, **ejecutando el método frenar**.

Concepto de POO

- Como vimos en este ejemplo, todos los elementos son objetos que se relacionan entre sí o se componen de otros objetos.
- **Este conjunto de objetos dialogan entre sí a través de ‘mensajes’ para realizar tareas.**

Clases y Objetos

- Como hemos visto, en la POO debemos definir objetos y sus relaciones. El mecanismo básico de este esquema se basa en el concepto de **clase**.

Clases y Objetos. Clases

- Una **clase define una plantilla o molde para aquellos objetos que comparten características comunes**.
- Como una clase es un concepto abstracto, hay que definirlo abstrayendo las cualidades comunes de una serie de objetos.
- Por ejemplo, en la clase vehículo ¿**qué es lo común en los vehículos**?

Clases y Objetos. Clases

- Un objeto es una **instancia de la clase** con sus cualidades particulares.
- Por ejemplo: Citroen C4 rojo 115 CV, Renault Megane blanco 100 CV etc., serían objetos de la clase vehículo.
- Cada uno de ellos es diferente a otro; sus atributos son distintos, pero todos ellos realizan las mismas acciones: frenar, acelerar, girar, etc.

Clases y Objetos. Clases

- En este ejemplo, definiríamos las clases **vehículo y conductor**.

- Luego crearíamos objetos concretos, de dichas clases:
  - (Jaguar 300, blanco, 180 CV),
  - (Martín, 30, 12).

Clases y Objetos. Clases

- Usando clases, nos aseguramos la reusabilidad de nuestro código, es decir, las clases que hoy escribimos, si están bien diseñadas, nos servirán para ‘siempre’.

Clases y Objetos. Clases

- Con la POO, si queremos construir un objeto que comparte ciertas cualidades con otro que ya tenemos creado, no tenemos que volver a crearlo desde el principio; simplemente, decimos qué queremos usar del antiguo en el nuevo y qué nuevas características debe tener nuestro nuevo objeto.

- A definir objetos a partir de otros existentes, **se conoce como herencia**

FORMACIÓN PROFESIONAL MONTECASTELO

Actividad 1

Crea la clase Persona, define sus atributos y sus métodos.

Mensajes y métodos

- Un programa orientado a objetos se compone básicamente de objetos.
- Cada **objeto** es una **entidad con unas características particulares** (atributos o campos), y una **forma de operar sobre los atributos** (métodos).

Mensajes y métodos

- Por ejemplo, una ventana de una aplicación Windows es un objeto.
  - Las **propiedades** serían el color de fondo, ancho, largo,tamaño, posición, etc.
  - Los **métodos** serían los trozos de código que permiten cerrar la ventana, maximizarla, moverla, minimizarla, etc.

Mensajes y métodos

- Los **atributos** que describen las características de un objeto **tienen valores distintos para cada objeto** (cada instancia).

- En cambio, los **métodos son comunes a la clase**, ya que todos los objetos de la clase pueden realizar las mismas tareas.

Mensajes y métodos

- En la ejecución de un programa los objetos reciben mensajes y responden a éstos ejecutando los métodos apropiados.

- Así, un mensaje se asocia a un método, de tal forma que cuando un objeto recibe un mensaje, la respuesta al mismo consiste en ejecutar el método asociado: se ejecuta como respuesta.

FORMACIÓN PROFESIONAL MONTECASTELO

Actividad 2

Utiliza la clase Persona para crear varios objetos y muestra cómo interactúan entre ellos.

Diseño de clases

- Las clases en C# se organizan en Namespaces (paquetes en Java)

```
namespace Space { class Class1 {...} .... }
```

- Para acceder a una clase podemos usar:

```
Space.Class1
```

```
using Space;
```

Diseño de clases

- A continuación, vemos un ejemplo de clase:

```
class Cuenta
{
    private string nombre; // Nombre del titular
    private string cuenta; // Número de cuenta
    private double saldo; // Saldo actual de la cuenta
    private double tipoDeInterés; // Tipo de interés en tanto por cien.
    // . . .
}
```

Diseño de clases

- Normalmente los atributos de un objeto de una clase se ocultan a los usuarios del mismo.
- Por ejemplo, un usuario que utilice la clase Cuenta **no podrá escribir código que manipule directamente estos atributos**.
- Tendrá que acceder a ellos a través de los métodos.

Diseño de clases

- **Esta protección se consigue usando el modificador private** (cuando se omite el modificador se supone private).

- **Un miembro que se declare privado es accesible solamente por los métodos de su propia clase.** Esto implica que no se puede acceder al miembro a través de métodos de cualquier otra clase.

Diseño de clases

- **Los modificadores de acceso son:**

public	La clase o miembro es accesible en cualquier ámbito.
protected	Se aplica sólo a miembros de la clase. Indica que sólo es accesible desde la propia clase y desde las clases derivadas.
private	Se aplica a miembros de la clase. Un miembro privado sólo puede utilizarse en el interior de la clase donde se define, por lo que no es visible tampoco en clases derivadas.
internal	La clase o miembro sólo es visible en el proyecto (ensamblado) actual.
internal protected	Visible en el proyecto (ensamblado) actual y también visible en las clases derivadas.

FORMACIÓN PROFESIONAL MONTECASTELO

Actividad 3

Define los métodos de acceso a las variables de la clase Cuenta

Diseño de clases

- Algunas acciones que el objeto de clase Cuenta puede realizar serían:
  - asignar el nombre de un cliente a una cuenta
  - obtener el nombre del cliente de una cuenta
  - asignar el número de cuenta
  - obtener el número de cuenta
  - realizar un ingreso
  - realizar un reintegro, etc.

Diseño de clases

- Por ejemplo, agregamos a la clase Cuenta un método que responda a la acción de asignar el nombre de un cliente del banco a una cuenta:

```
public void asignarNombre(string nom)
{
    if (nom == null || nom.Length == 0)
    {
    System.Console.Writeline("Error. Sin nombre");
    return;
    }
    nombre = nom;
}
```

Diseño de clases

- **El método ha sido declarado public.** Un miembro público es accesible para cualquier otra clase que necesite utilizarlo.

- **El método tiene tipo de retorno void** (no devuelve nada) y **usa un parámetro de tipo string** con el nombre del titular que se va a asignar.

Diseño de clases

- El método asegura que el nombre a asignar no sea una referencia nula o una cadena vacía chequeándose esta condición mediante la propiedad Length de la clase string.

- Si la condición se evaluara como cierta, se visualizaría el mensaje de error y se acabaría la ejecución del método

Diseño de clases

- En otro caso, **se asigna la cadena nom pasada como argumento al atributo nombre del objeto que recibe el mensaje.**

Diseño de clases

- El método se ejecuta desde un objeto de la clase.
- Por ejemplo, desde la clase Cuenta, utilizaríamos un objeto “micuenta” para ejecutar el método AsignarNombre:

```
micuenta.asignarNombre("Fulanito");
```

Diseño de clases

- En general, para acceder a un miembro de un objeto, ya sea a un atributo o a un método, se utiliza la sintaxis **objeto.miembro**.
- El punto es el operador llamado operador de acceso a miembro. De esta forma queda claro cual es el objeto y cual es el miembro al que hacemos referencia.

Diseño de clases

- Se permite el acceso al miembro a través del punto si el miembro es público.
- Si fuera un miembro privado, el compilador daría un mensaje de error advirtiéndonos de este hecho.
- Evidentemente, **un objeto de una clase sólo puede invocar a métodos de su clase**.

Diseño de clases

- Una vez definida una clase, podremos crear objetos de dicha clase y trabajar con ellos.
- En un programa C# debe haber una clase con un método Main() que se utiliza como punto de entrada.
- **Este requerimiento se puede cumplir de tres formas diferentes:**

Diseño de clases

- **OPCIÓN 1: Añadir a la clase Cuenta un método Main() que incluya el código del programa.**

```
class Cuenta
{
    campos . . .
    metodos() . . .
    public static void Main(strig[] args)
    {
        Cuenta c1 = new Cuenta(); // Se crea el objeto c1 de tipo Cuenta
        c1.asignarNombre("Martin"); // Se le envía el mensaje asignarNombre()
    }
}
```

Diseño de clases

- Esta forma de operar es técnicamente correcta, pero mezclamos la definición de una clase, como es el caso de Cuenta, con el código para ejecutar un programa.
- Es un poco liosa, por lo que es poco recomendada.

Diseño de clases

- **OPCIÓN 2: Añadir, en el mismo fichero fuente en el que se almacena la clase Cuenta, otra clase que incluya el método Main() .**

```
class Cuenta
{
    campos . . .
    metodos() . . .
}
class AplicacionCuenta
{
    public static void Main(strig[] args)
    {
        Cuenta c1 = new Cuenta();
        c1.asignarNombre("Martin");
    }
}
```

Diseño de clases

- **Esta clase se suele llamar clase aplicación**
- Permite separar las clases que definen objetos de la clase que contiene el código que lanza la aplicación.

Diseño de clases

- **OPCIÓN 3: Añadir al proyecto otro archivo fuente que contenga la clase aplicación con el método Main().**
- De esta forma tenemos dos archivos conteniendo cada uno la definición de una clase.

Diseño de clases

- Es importante que los dos archivos estén en la misma carpeta para que el compilador pueda encontrar todos los archivos del proyecto.
- **Esto es lo más adecuado y por ello es la MEJOR FORMA de trabajar.**

Diseño de clases. Ejemplo completo

- El típico ejemplo para instanciar una clase es:

```
Cuenta c1 = new Cuenta();
```

- **Sin embargo, ¿qué significa esta sentencia? Es una forma simple de:**

```
Cuenta c1; // Se declara c1 como referencia de tipo Cuenta
c1 = new Cuenta(); // c1 referencia al objeto tipo Cuenta recién creado
```

Diseño de clases. Ejemplo completo

- La primera línea **declara que c1 es una referencia de tipo Cuenta**.
- La segunda línea, **a través del operador new crea una instancia o ejemplar de la clase Cuenta.**

Diseño de clases. Ejemplo completo

- **La sentencia, usando el operador = asigna dicha referencia a c1.**
- Así, **cuando decimos que c1 es un objeto de tipo Cuenta, la realidad es que c1 es una referencia a un objeto de tipo Cuenta**.

FORMACIÓN PROFESIONAL MONTECASTELO

Actividad 4

Utiliza las tres formas vistas anteriormente para crear un programa con la clase Persona y ponlo a funcionar.

¿Cuál de las tres te parece la más sencilla?

Constructores

- Un constructor es un método especial de una clase que es llamado automáticamente siempre que se crea un objeto de dicha clase.

- **Su función es inicializar el objeto**

Constructores

- El constructor se identifica con un nombre que coincide con el nombre de la clase a la que pertenece y no tiene valor de retorno, ni siquiera void.
- Cuando en una clase no se define ningún constructor, C# asume uno por omisión.

Constructores

- Por ejemplo, en nuestra clase Cuenta, no hemos definido ningún constructor, por lo tanto se asume uno cuya definición sería:

```
public Cuenta()
{ }
```

Constructores

- **El constructor por omisión no tiene parámetros y no hace nada.**
- Sin embargo es necesario que esté definido ya que será invocado cada vez que se construya un objeto sin especificar ningún argumento.
- En este caso el objeto será iniciado con los valores predeterminados: los atributos de tipo numérico a cero, y las referencias a null.

Constructores

- Por ejemplo, en esta sentencia:

```
Cuenta c1 = new Cuenta();
```

- El operador new es quien ‘crea’ un nuevo objeto (reserva la memoria y retorna su referencia); a continuación, se invoca al constructor que ‘inicia’ los atributos. En este caso, con los valores por defecto.

Constructores

- En general, se define un constructor con los atributos de la clase:

```
public Cuenta(string nom, string cue, double sal, double tipo)
{
    nombre = nom;
    cuenta = cue;
    saldo = sal;
    tipoDeInteres = tipo;
}
```

Constructores

- A continuación, usamos el operador *new* para llamar al constructor:

```
Cuenta c2 = new Cuenta("Martin", "12345678Z", 100, 7.5);
```

Constructores

- Los constructores normalmente se definen públicos para que se puedan invocar desde cualquier parte.
- Cuando se define un constructor, el constructor por omisión es reemplazado por éste.

Constructores

- ¿Podemos tener dos constructores?
- **La respuesta es SÍ**. Esto se conoce como **sobrecarga de métodos**.
- A continuación veremos una sobrecarga del constructor de la clase Cuenta.

Constructores

```
public Cuenta(string nom, string cue, double sal, double tipo)
{
    nombre = nom;
    cuenta = cue;
    saldo = sal;
    tipoDeInteres = tipo;
}

public Cuenta(string nom, string cue, double sal)
{
    nombre = nom;
    cuenta = cue;
    saldo = sal;
    tipoDeInteres = 0;
}
```

Constructores

- Utilizando estos constructores, crearemos los objetos:

```
Cuenta c2 = new Cuenta("Martin", "12345678Z", 14, 2);

Cuenta c3 = new Cuenta("Marcos", "12345785A", 7);
```

- En cada una de las sentencias utilizamos un constructor diferente, según el número de parámetros que le pasemos.

FORMACIÓN PROFESIONAL MONTECASTELO

Actividad 5

Define un constructor para la clase Persona pasándole todos los atributos

¿Puedes definir otro constructor? Crea objetos con los dos constructores.

¿Qué pasa con el constructor por omisión?

Destructor

- Cuando se crea un objeto con el operador new, se asigna la cantidad de memoria necesaria para ubicar dicho objeto.
- Si no existiera memoria suficiente, se provocaría un error (en realidad el operador new lanzaría una excepción).

FORMACIÓN PROFESIONAL MONTECASTELO

Destructor

- Cuando para un objeto no existe ninguna referencia hacia él, queda inaccesible, pero sigue ocupando memoria.
- Esta memoria debe liberarse quedando disponible para poder crear otros objetos.
- Cada clase sólo puede contar como máximo con **UN DESTRUCTOR**

Destructor

- El destructor no retorna ningún valor y no tiene argumentos.

```
~Cuenta()
{
    System.Console.WriteLine("Destruyendo objetos...");
}
```

- Cada vez que un objeto Cuenta sea destruido, se mostrará el mensaje.

FORMACIÓN PROFESIONAL MONTECASTELO

Actividad 6

Define un destructor para la clase Persona

¿Puedes sobrecargar este destructor?

¿Qué pasaría si no tenemos destructor?

This

- ¿Cómo sabe un método de una clase sobre qué objeto está trabajando si en el cuerpo de dicho método no se indica nada de forma explícita?
- Para hacer referencia explícita dentro de la clase al objeto que llama al método usaremos **this.**

This

- Por ejemplo, ¿qué diferencia hay entre estos métodos de la clase Cuenta?

```
public string obtenerNombre()
{
    return nombre;
}
```

```
public string obtenerNombre()
{
    return this.nombre;
}
```

This

- Por ejemplo, ¿qué diferencia hay entre estos métodos de la clase Cuenta?

```
public string obtenerNombre()
{
    return nombre;
}
```

```
public string obtenerNombre()
{
    return this.nombre;
}
```

- **NO HAY DIFERENCIA,** pues C# utiliza **this** de forma implícita

FORMACIÓN PROFESIONAL MONTECASTELO

Actividad 7

*Modifica los métodos de la clase Persona asegurándote que usas correctamente la palabra reservada **this***

Propiedades

- **Aparecen en la clase como campos de datos.**
- Las propiedades tienen un tipo, pero la asignación y lectura de las mismas se realiza a través de métodos de lectura y escritura: **get y set**
- Con las propiedades, **se puede limitar el acceso a un campo permitiendo sólo la lectura o sólo la escritura**

Propiedades

- Para definir una propiedad, se usa la siguiente sintaxis:

```
<tipoPropiedad> <nombrePropiedad>
{
    set
    {
        <códigoEscritura>
    }
    get
    {
        <códigoLectura>
    }
}
```

Propiedades

- Por ejemplo:

```
public int edad
{
    set
    {
        this.edad = edad;
    }
    get
    {
        return this.edad;
    }
}
```

Propiedades. Ejemplo completo

- Se va a redefinir la clase Cuenta pero usando propiedades que sustituyan a los métodos de acceso a los campos:

Propiedades.

```
class Cuenta
{
    // Atributos
    private string nombre;
    private string cuenta;
    private double saldo;
    private double tipoDeInterés;

    // Propiedades
    public double Saldo
    {
        get {return saldo;} // Propiedad de sólo lectura
    }

    public string Nombre
    {
        get {return nombre;}

        set
        {
            if (value == null || value.Length == 0)
            {
            System.Console.WriteLine("Error: cadena vacía");
            return;
            }
            nombre = value;
        }
    }
    public string Cuenta
    {
        get {return cuenta;}
        set
        {
            if (value == null || value.Length == 0)
            {
            System.Console.WriteLine("Error: cuenta no válida");
            return;
            }
            cuenta = value;
        }
    }
}
```

FORMACIÓN PROFESIONAL MONTECASTELO

Propiedades.

- El **acceso get** contiene el código de la lectura de la propiedad y por tanto del atributo que se maneja. **Debe terminar siempre con la sentencia return o throw.**

- El **acceso set** contiene el código para escribir un valor en la propiedad y por tanto en el atributo que se maneja. **El parámetro value es el valor que se le da a la propiedad.**

Propiedades.

- Un ejemplo de uso de la clase Cuenta usando propiedades podría ser:

```
class PruebaPropiedades
{
    public static void Main(string[] args)
    {
        Cuenta c6 = new Cuenta();
        c05.Nombre = "Martin";
        c05.Cuenta = "12345";
        c05.TipoDeInterés = 2.5;
        c05.ingreso(250);
        c05.reintegro(50);
        System.Console.WriteLine(c05.Nombre);
        System.Console.WriteLine(c05.Cuenta);
        System.Console.WriteLine(c05.Saldo);
        System.Console.WriteLine(c05.TipoDeInterés);
    }
}
```

Ejemplo completo

- **A continuación, se expone el código completo de un programa que crea objetos de tipo Cuenta y que usa sus métodos para probar la funcionalidad de dicha clase de objetos.**

{Fichero adjunto EjemploCompleto.cs}

KEEP CALM IT'S KAHOOT TIME

Tema 5: POO

Asignatura: Desarrollo de interfaces

CS Desarrollo de Aplicaciones Multiplataforma