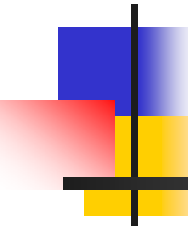


Características Avanzadas en C#

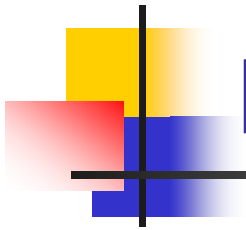
Programación Orientada a Objetos

Curso 2008/2009

Begoña Moros Valle

Características avanzadas

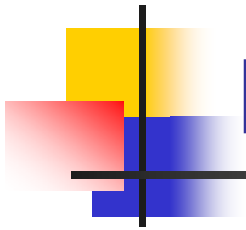
- C# incluye características destacadas que no se incluyen en otros lenguajes como Java o C++:
  - **Eventos.**
  - **Métodos de extensión.**
  - **Tipos valor anulables.**
  - **Inferencia de tipos en variables.**
  - **Objetos anónimos.**
  - **Lenguaje de consultas LINQ.**



Eventos

- **Motivación**:
  - Queremos avisar al titular de la cuenta y al banco cuando se realice un reintegro superior a una determinada cantidad.

- **Solución**:
  - Los eventos en C# son utilizados para notificar cambios de estado que son de interés para otros objetos.



Eventos

- Un evento es un componente en la definición de una clase al mismo nivel que las propiedades, los métodos y constructores.

- Se declaran con la palabra clave **`event`** y su tipo debe ser el de un delegado:
  - Los avisos se realizan aplicando un método sobre los objetos interesados en el evento.
  - Los métodos que van a recibir el aviso deben ser compatibles con el delegado.

Eventos – Tipo del evento

- **Declaración del tipo delegado**:
  - Los parámetros del tipo delegado contienen la información del evento: la cuenta que ha producido el evento y la cantidad del reintegro.

```
public delegate void NotificacionReintegro(Cuenta cuenta,
                                            double cantidad);
```

Eventos – Declaración

- **Declaración del evento** en la clase **`Cuenta`**.

```
public class Cuenta
{
    public const double LIMITE_NOTIFICACION = 5000;

    public event NotificacionReintegro ReintegroEvento;
    …
}
```

- Se declara también una constante que establece el límite de reintegro para realizar el aviso.

Evento – Clase Receptora

- La clase **`Persona`** declara un método para recibir las notificaciones del evento:

```
public class Persona
{
  …
  public void AvisoReintegro(Cuenta cuenta, double cantidad)
  {
     Console.WriteLine("Persona - Reintegro: " + cantidad);
  }
}
```

- El método es compatible con el tipo del evento.

Eventos – Registro de Objetos

- En el constructor de la clase **`Cuenta`** registramos al titular de la cuenta como objeto interesado en el evento:

```
public Cuenta(Persona titular, double saldo) {
    …
    ReintegroEvento += titular.AvisoReintegro;
}
```

- Se utiliza += para el **registro** del método que recibe la notificación del evento.
- Por el contrario, para la **eliminación** se utiliza -=.

Eventos – Notificación del evento

- El método **`Reintegro`**( ) lanza el evento si la cantidad supera el límite y hay objetos

```
public void Reintegro(double cantidad)
{
    if (cantidad <= saldo)
    {
        saldo = saldo - cantidad;

        if (cantidad > LIMITE_NOTIFICACION
            && ReintegroEvento != null)
        {
            ReintegroEvento(this, cantidad);
        }
    }
}
```

Evento – Clase Receptora

- Cualquier otra clase que esté interesada en los eventos de **`Cuenta`** debe declarar un método que cumpla el tipo del delegado.
- La clase **`Banco`** declara un método compatible con el tipo del evento para recibir los avisos:

```
public void Reintegros(Cuenta cuenta, double cantidad)
{
    Console.WriteLine("Banco - Reintegro : " + cantidad);
}
```

Eventos – Ejemplo

- Una cuenta tiene asignado al menos un objeto interesado en el evento: el titular de la cuenta.
- El banco también se interesa en el evento. Se aplica un reintegro que provoca el evento:

```
Cuenta cuentaEvt = new Cuenta(persona, 10000);
cuentaEvt.ReintegroEvento += banco.Reintegros;
cuentaEvt.Reintegro(5001);
```

- Tras la ejecución son notificados el titular y el banco. Se muestra por la consola:
  - `Persona - Reintegro: 5001`
  - `Banco - Reintegro : 5001`

Métodos de extensión

- **Motivación**:
  - Añadir un método en una clase existente, sin crear un subtipo ni editar el código de la clase.
- Los métodos de extensión se declaran **en clases "estáticas"**.
- El primer parámetro del método es la referencia al **objeto receptor** y se utiliza el modificador **`this`**.
- Dentro del código del método de extensión sólo podemos acceder a las declaraciones visibles desde la clase estática.
- Para hacer uso de un método de extensión hay que tener visibilidad sobre la clase estática en la que se define.
- **En ningún caso un método de extensión reemplaza a un método de la clase**.

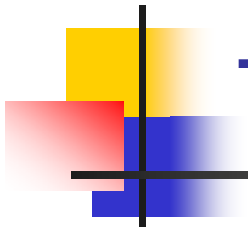
Métodos de extensión

- **Ejemplo**: método que simula el resultado de un ingreso en una cuenta:

```
static class Extensiones {

    // Método de extensión de la clase Cuenta
    public static double SimulaIngreso ( this Cuenta cuenta,
                int cantidad)
    {
        return cuenta.Saldo + cantidad;
    }
}
```

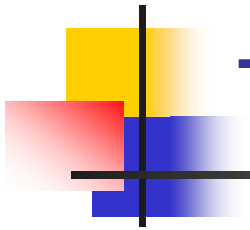
```
cuenta.SimulaIngreso(20);
```



Tipos valor anulables

- **Motivación**:
  - Permitir que un tipo por valor admita el valor especial `null`.
  - Resulta útil cuando se obtienen datos de una consulta a una base de datos relacional. El modelo relacional admite nulos para los tipos primitivos.

- **Declaración**: añadir `?` a un tipo por valor
  - `int? intAnulable;`
  - `Punto? puntoAnulable;`



Tipos valor anulables

- **Métodos disponibles**:
  - **`HasValue`**:
    - indica si almacena un valor `null`.
  - **`GetValueOrDefault`**:
    - para obtener el valor del tipo anulable, o un valor por defecto si almacena `null`.
    - Recibe como argumento el valor por defecto.

Tipos valor anulables

```
int? intAnulable;
int entero;

intAnulable = null;

if (! intAnulable.HasValue)
    Console.WriteLine("Valor nulo");

entero = intAnulable.GetValueOrDefault(-1);

Console.WriteLine("Valor entero: " + entero); // -1

intAnulable = 3;

entero = intAnulable.GetValueOrDefault(-1);

Console.WriteLine("Valor entero: " + entero); // 3
```

Inferencia de tipos en variables

- Declaración de variables sin tipo:

```
var cuenta = new Cuenta(persona, 500);
```

- Se utiliza la palabra clave **`var`** para indicar que debe inferirse el tipo de la variable.

- En el ejemplo, el compilador determina que la variable tiene como tipo estático `Cuenta`.

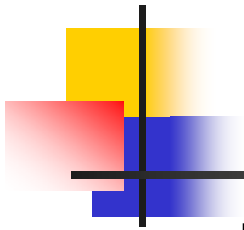
Objetos anónimos

- Permiten declarar objetos en línea estableciendo un conjunto de propiedades con sus valores.
- Tienen implementado automáticamente el método `ToString`.
- Sobre una variable que referencia a un objeto anónimo es posible acceder a las propiedades.

```
var pagina = new {Nombre = "UMU", Web = "http://www.um.es/"};

Console.WriteLine("Página: " + pagina);

Console.WriteLine("Nombre de la página: " + pagina.Nombre);
```



LINQ

- Facilidad de consulta sobre estructuras de datos integrada en el lenguaje (*Language INtegrated Query*).
- Define un modelo de consulta abstracto que es aplicable a cualquier modelo de datos: colecciones, consultas a una base de datos relacional, etc.
- **Esquema básico** de consulta:
  - `from` → colección de objetos iterables
  - `where` → condición definida como expresión lambda
  - `select` → objeto de la colección o anónimo.

LINQ – Ejemplo

- **Ejemplo**: se realiza una consulta sobre una lista de cuentas.
- El resultado de la consulta son objetos anónimos.
- Se aplica inferencia de tipos en las variables.

```
var resultado =
    from c in cuentas
    where c.Saldo > 400
    select new { Saldo = c.Saldo, Titular = c.Titular };

foreach (var c in resultado) {
    Console.WriteLine("Resultado LINQ " + c);
}
```