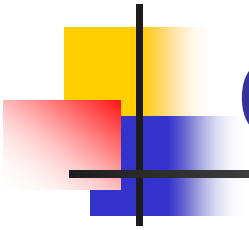
Tema 4: Corrección y Robustez en C#

Programación Orientada a Objetos

Curso 2008/2009

Begoña Moros Valle

DIS
Departamento de
Informática y Sistemas



Contenido

- Soporte para corrección y robustez en C#:
  - Asertos
  - Mecanismo de excepciones:
    - Declaración de excepciones
    - Lanzar excepciones
    - Manejo de excepciones
    - Definición de excepciones

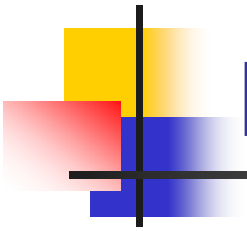
Corrección y Robustez

- **Corrección**:
  - Es la capacidad de los productos software de realizar con exactitud su tarea (**cumplir su especificación**).
- **Robustez**:
  - Es la capacidad de los productos software de reaccionar adecuadamente ante **situaciones excepcionales**.
- Al igual que Java, el lenguaje ofrece **asertos** y **excepciones** como soporte de la corrección y robustez del código.
- La **verificación** del código se realiza con pruebas unitarias

Asertos

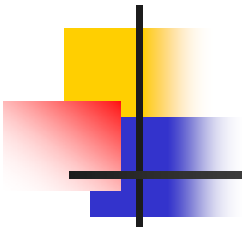
- La clase **`System.Diagnostics.Debug`** declara el **método** de clase **`Assert`** para evaluar asertos.
- La evaluación de asertos sólo se realiza en la ejecución de la aplicación de **depuración**.
- Los asertos de C# tienen las mismas limitaciones que en C++ y Java. Simplemente son una utilidad de depuración.

```
Debug.Assert(valores != null,
             “Lista de valores no debe ser nula");
```



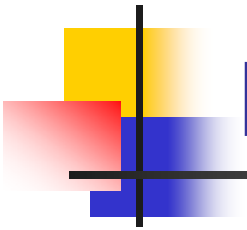
Pruebas Unitarias

- Al igual que en Java y C++, las pruebas unitarias no forman parte de la biblioteca del lenguaje.
- Existen **herramientas externas** para realizar pruebas unitarias.
- El entorno de desarrollo de **Microsoft Visual Studio** incluye en el espacio de nombres `Microsoft.VisualStudio.TestTools.UnitTesting` soporte para realizar pruebas unitarias.



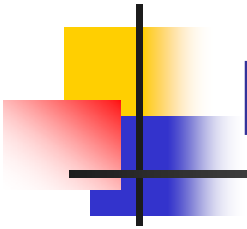
Excepciones

- El modelo de excepciones de Java comparte aspectos en común con Java y C++:
  - **Las excepciones son objetos** (= Java).
  - La raíz de todas las excepciones es la clase **`System.Exception`** (= `Throwable` de Java).
  - Todas las excepciones son **no comprobadas** (= C++).
- Sin embargo, se diferencia de Java y C++:
  - **En la declaración de un método no se puede indicar las excepciones que lanza**. Sólo podemos indicarlo en la documentación.



Excepciones

- La clase **`System.Exception`** tiene las características comunes a todas las excepciones:
  - `string` **`Message`** `{ get; }`: mensaje de error.
  - `string` **`StackTrace`** `{ get; }`: pila de llamadas en la que se ha producido la excepción.
- Las excepciones se lanzan con **`throw`** (= Java y C++).



Excepciones

- Una excepción se puede construir de tres formas (= Java):
  - Llamando al **constructor sin parámetros**.
  - Llamando al **constructor con la cadena de error**.
  - Llamando al constructor con la **cadena de error** y la **excepción** que ha causado el error.

Excepciones – Clasificación

- Se definen dos grupos de excepciones con el propósito de distinguir excepciones predefinidas y de usuario:
  - **`System.SystemException`**: predefinidas en .NET
  - **`System.ApplicationException`**: excepciones de usuario.
- A diferencia de Java, estos dos grupos sólo representan una clasificación de excepciones.

  ➔ No tiene relación con el concepto comprobada/no comprobada de Java.

Control de precondiciones

- Las excepciones predefinidas incluyen excepciones para el tratamiento de precondiciones:
  - **`ArgumentException`**: precondiciones de argumentos.
  - **`InvalidOperationException`**: precondiciones de estado.

```
public void Ingreso(double cantidad){
  if (cantidad < 0)
    throw new ArgumentException("Cantidad negativa");
  if (estado != EstadoCuenta.OPERATIVA)
    throw new InvalidOperationException("Estado incorrecto");

  saldo = saldo + cantidad;
}
```

Excepciones de usuario

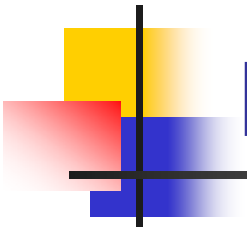
- Las excepciones de usuario utilizadas para notificar el fallo en las postcondiciones heredan de `System.`**`ApplicationException`**.

```
namespace Navegador {

  public class RedNoDisponible : ApplicationException
  {
    public RedNoDisponible() { }
    public RedNoDisponible(string msg): base(msg) { }
    public RedNoDisponible(string msg, Exception causante)
        : base(msg, causante) { }
  }

}
```

Declaración de excepciones

- **Las excepciones que lanza un método no son declaradas en su signatura**.
- Se aconseja documentarlas en la declaración del método

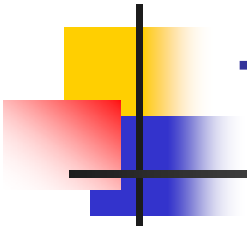
```
/// <summary>
/// Obtiene una nueva línea del fichero
/// </summary>
/// <returns>Una línea del fichero
///          o null si no hay disponibles</returns>
/// <exception cref="Navegador.RedNoDisponible">
/// Error producido por un fallo en la red
/// </exception>
public String leerLinea() { …  }
```



Excepciones y herencia

- En C# no se controlan las excepciones que lanza un método.
- Por tanto, no hay restricción en el lanzamiento de excepciones en la redefinición de un método en un subtipo.
- Es responsabilidad del programador el correcto lanzamiento de excepciones.

Tratamiento de excepciones



- Al igual que Java y C++, las excepciones pueden ser tratadas en bloques **try-catch**.
- Cuando ocurre una excepción se evalúan los tipos definidos en los manejadores y se ejecuta el primero cuyo tipo sea compatible (= Java y C++)
- Se puede definir un **manejador para cualquier tipo de excepción**: `catch(Exception e)`
- Es posible **relanzar una misma excepción** que se está manejando (= C++): `throw;`
- Las excepciones no tratadas en un método se propagan al método que hace la llamada (= Java y C++).

Tratamiento de excepciones

```
public class Navegador
{
    public void Visualiza(String url)
    {
        Conexion conexion;
        int intentos = 0;
        while (intentos < 20) {
            try {
                conexion = new Conexion(url);
                break;
            }
            catch (RedNoDisponible e) {
                System.Threading.Thread.Sleep(1000);
                intentos++;
                if (intentos == 20) throw; // relanza
            }
        }
            …
```