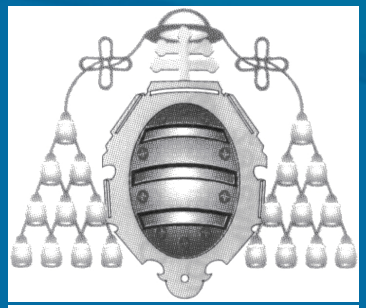


Cursos de Extensión Universitaria
Enero 2003
UNIVERSIDAD DE OVIEDO
Cod. 0

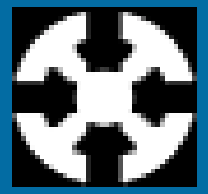
PLATAFORMA .NET Y SERVICIOS WEB

Diseño de Aplicaciones con C# y .NET Framework

Aquilino A. Juan Fuente
aquilino@lsi.uniovi.es

Benjamín López Pérez
benja@lsi.uniovi.es

Dpto. de Informática
OOTLab - Laboratorio de Tecnologías de Orientación a Objetos
http://www.ootlab.uniovi.es

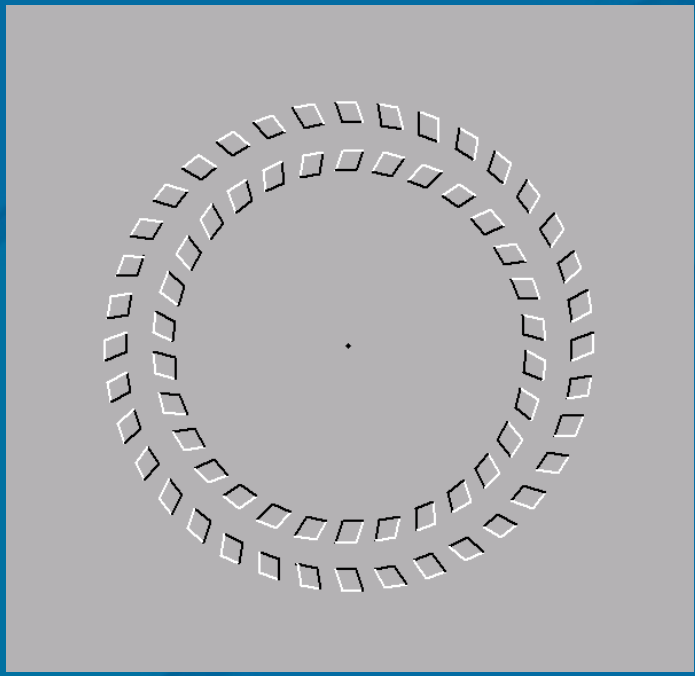


Contenidos

- **Lunes 20:** Diseño de Aplicaciones (I).
  - Visión general del framework de .NET
  - Desarrollo de aplicaciones (Tipos, depuración, desarrollo, y empaquetado de entregables –deployment)
- **Martes 21:** Diseño de Aplicaciones (II).
  - Visión general de la biblioteca de clases del framework
  - Realización de varios ejemplos prácticos
- **Miércoles 22:** Diseño de Aplicaciones (III).
  - Ficheros en C#
  - Acceso a Base de Datos
  - Implementación de varios ejemplos prácticos

Diseño de Aplicaciones (I)
Contenidos

- Visión General del Framework
- Ejemplo de Implementación (Conversor €)
- Deployment

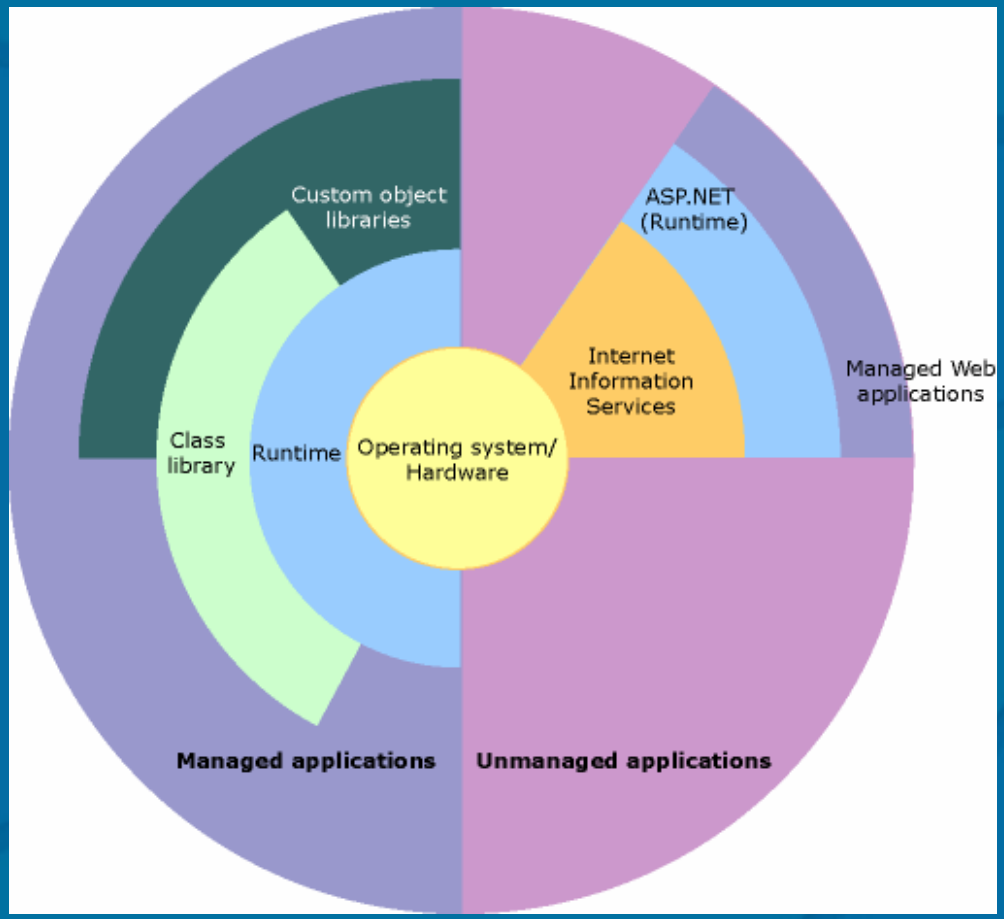


Diseño de Aplicaciones (I)
Visión General del Framework

- El **.NET Framework** es una nueva plataforma de computación que simplifica el desarrollo de aplicaciones en el entorno altamente distribuido de Internet.
- **.NET Framework** ha sido diseñado para lograr los siguientes objetivos:
  - Para proveer un entorno de programación orientada a objetos en el que el código puede ser almacenado y ejecutado en local, ejecutado en local pero distribuido por Internet o ejecutado en remoto.
  - Para proveer un entorno de ejecución de código que minimice la fase de “deployment” del código y los conflictos de versiones.
  - Para proveer un entorno de ejecución de código que garantice la ejecución segura de código, incluyendo el código creado por terceras partes desconocidas o no confiables.
  - Para proveer un entorno de ejecución de código que elimine los problemas de realización (performance) de los entornos de scripting o interpretados.
  - Para conseguir que la experiencia del desarrollador en diferentes tuipos de aplicaciones sea reaprovechable. (Aplicaciones Windows, aplicaciones Web,...)
  - Construri todas las comunicaciones en estándares de la industria para asegurar que el código basado en el **.NET Framework** pueda integrarse fácilmente.

Diseño de Aplicaciones (I)
Visión General del Framework

- CLR (Common Language Runtime)
  - Gestión de memoria
  - Ejecución de hilos (threads)
  - Ejecución de Código
  - Verificación de código seguro
  - Compilación
  - Otros servicios del sistema
- .NET Framework Class Library



Diseño de Aplicaciones (I)
Visión General del Framework

- **C# Type.NET Framework type**
  - Bool — System.Boolean
  - Byte — System.Byte
  - Sbyte — System.SByte
  - Char — System.Char
  - Decimal — System.Decimal
  - Double — System.Double
  - Float — System.Single
  - Int — System.Int32
  - Uint — System.UInt32
  - Ong — System.Int64
  - Ulong — System.UInt64
  - Object — System.Object
  - Short — System.Int16
  - Ushort — System.UInt16
  - String — System.String

Diseño de Aplicaciones (I)
Visión General del Framework

- Tipos de Proyectos con C# y Visual Studio .NET

Windows Application	Aplicación que comienza con una Form vacía que responde a eventos en entorno Windows
Class Library	Una clase .NET que puede ser llamada por otro código
Windows Control Library	Una clase .NET que puede ser llamada por otro código y que tiene un interfaz de usuario (Al estilo de los controles ActiveX)
ASP.NET Web Application	Un site basado en páginas ASP.NET y C# clases que genera respuestas HTML para ser enviadas a los browsers
ASP.NET Web Service	Una clase C# que actua como Web Service
Web Control Library	Un control que puede ser llamado por páginas ASP.NET para generar el código HTML que da apariencia de un control cuando se muestra en un Browser
Console Application	Una aplicación que corre en una ventana de línea de comandos
Windows Service	Una servicio que corre en el background en un Windows NT o Windows 2000
Empty Project	Proyecto vacío. Este tipo de proyecto debe ser comenzado desde el principio.
Empty Web Project	Un proyecto vacío, pero los parámetros de compilación están colocados para que el compilador genere código para páginas ASP.NET.
New Project in Existing Folder	Ficheros de proyecto nuevos para un proyecto vacío. Se usa si se desea introducir en Visual Studio .NET un proyecto que ha sido escrito con un editor de texto independiente.

Diseño de Aplicaciones (I)
Ejemplo de Implementación (Conversor €)

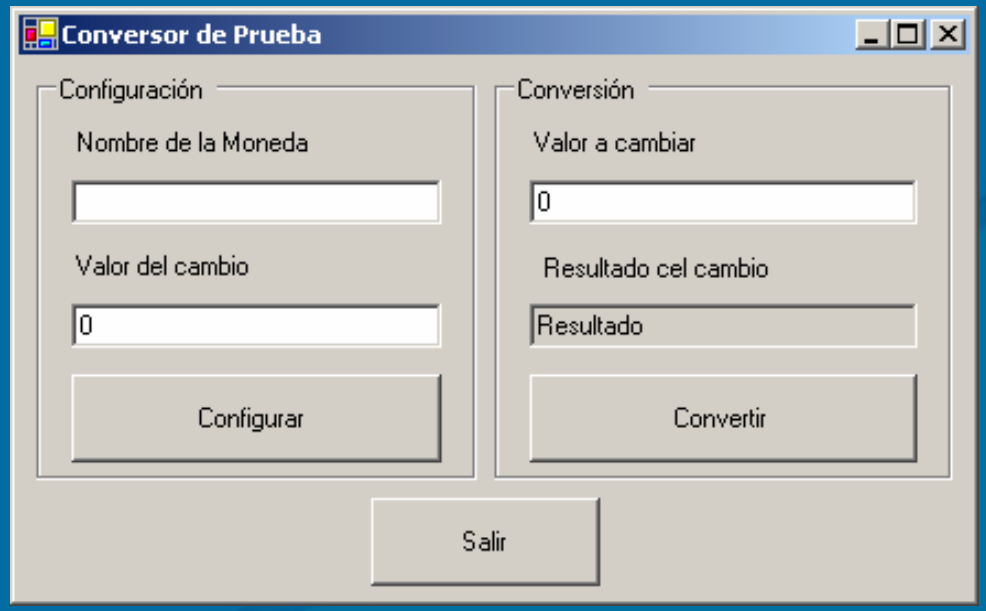
- Implementar un conversor de € a otra moneda cualquiera que se pueda configurar.
- Debe ser posible cambiar el nombre de la moneda y su valor frente al € y, una vez elegida, hacer tantas conversiones como se desee.

Diseño de Aplicaciones (I)
Ejemplo de Implementación (Conversor €)

- Se van a realizar dos soluciones:
  - Solución a la que lleva intuitivamente el framework
    - Ventajas y desventajas
  - Solución usando un patrón MVC
    - Ventajas y desventajas

Diseño de Aplicaciones (I)

- Solución en base al framework...
- Diseño de la interfaz
- Implementación

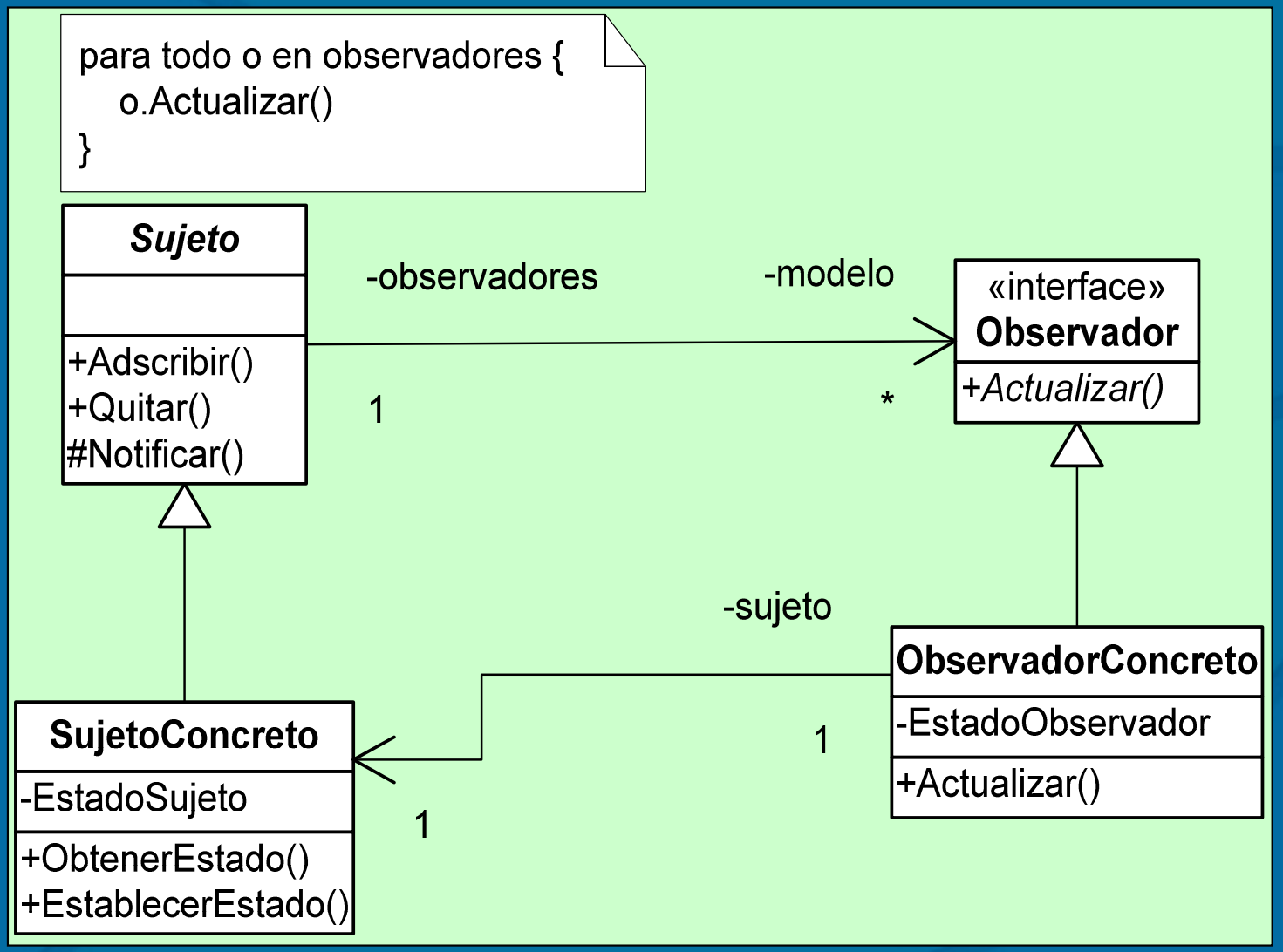


Diseño de Aplicaciones (I)
Ejemplo de Implementación (Conversor €)

- Implementación en base al patrón MVC...
- Previos necesarios
  - Patrones de diseño (GOF)
    - Patrón Observer
    - Patrón State
  - Patrones Arquitectónicos (Buschman)
    - Patrón MVC

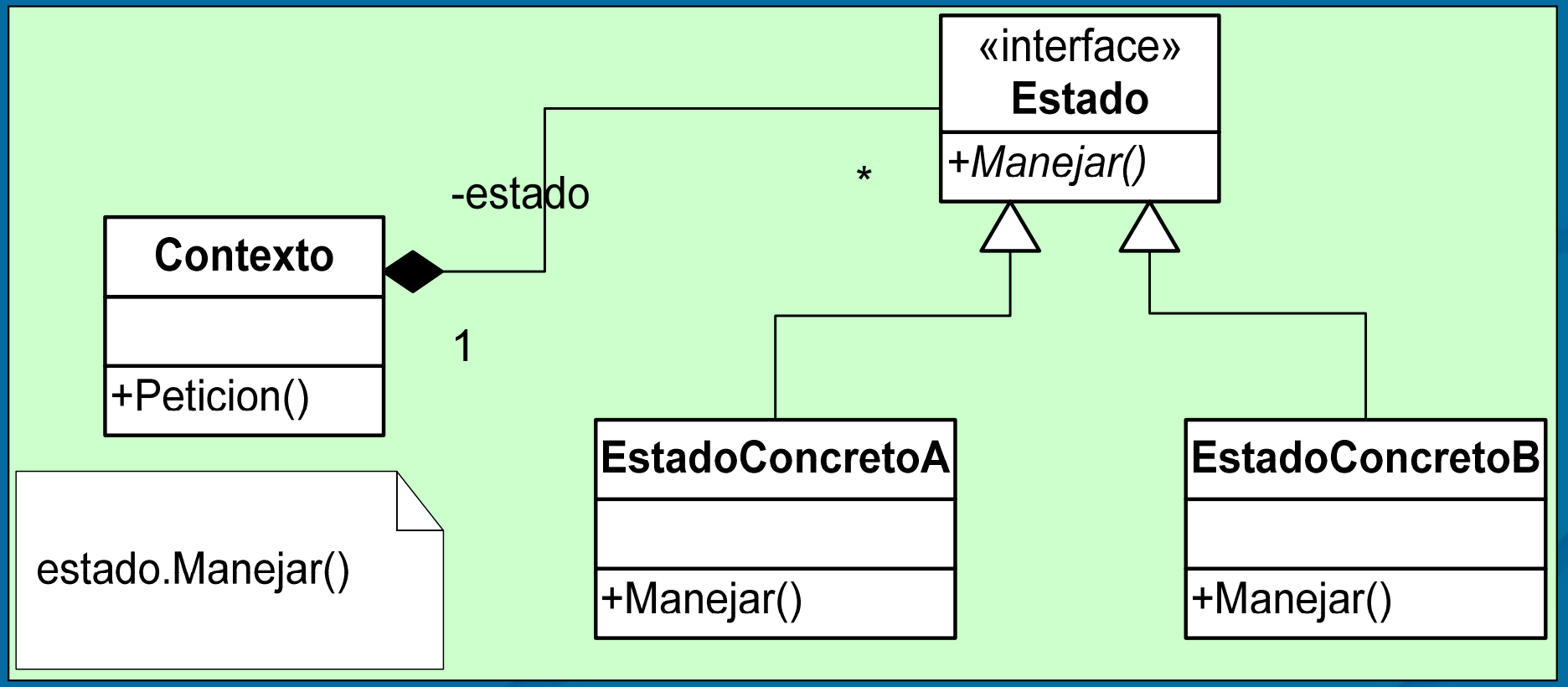
Diseño de Aplicaciones (I)
Ejemplo de Implementación (Conversor €)

- Patrón Observer



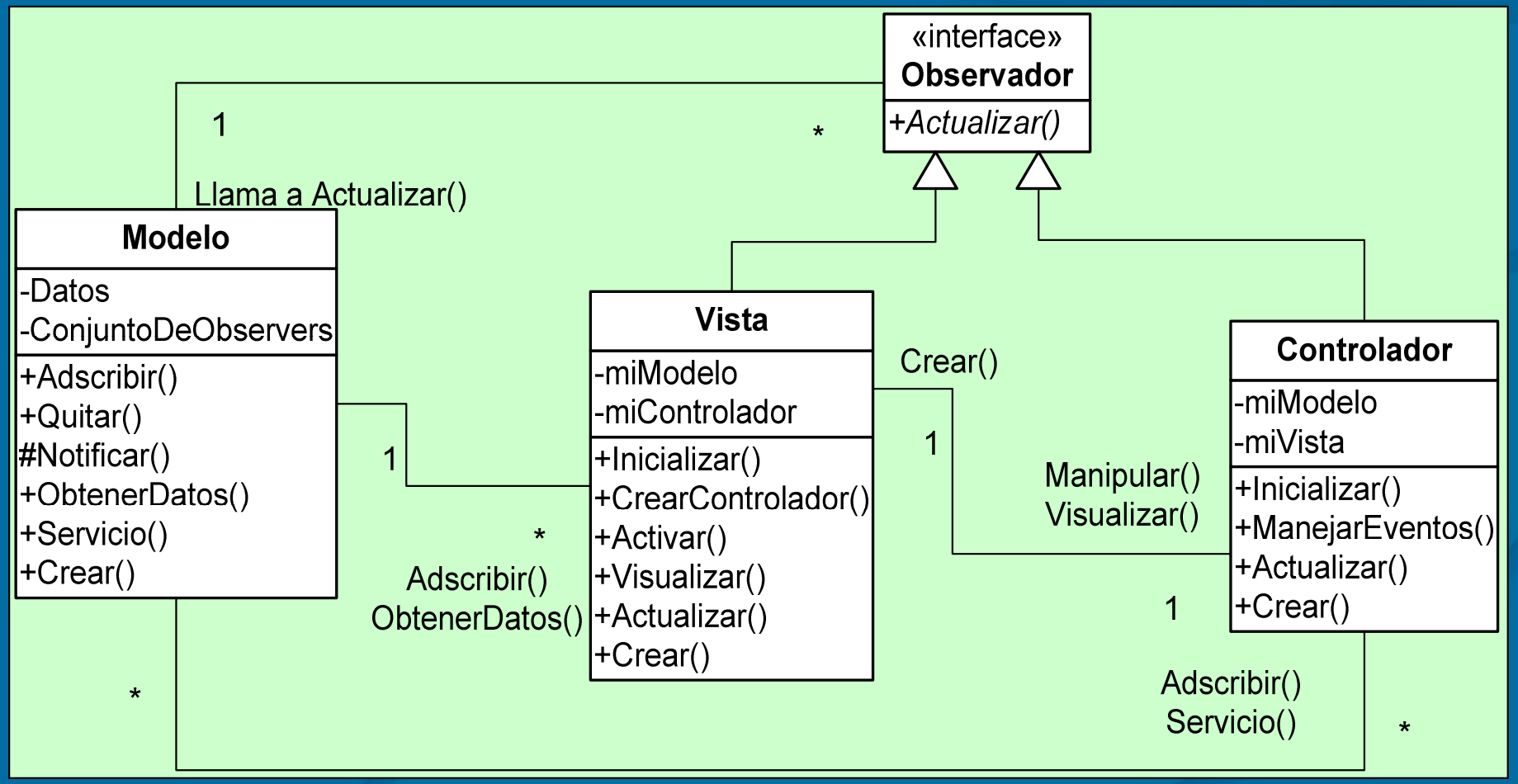
Diseño de Aplicaciones (I)
Ejemplo de Implementación (Conversor €)

- Patrón State



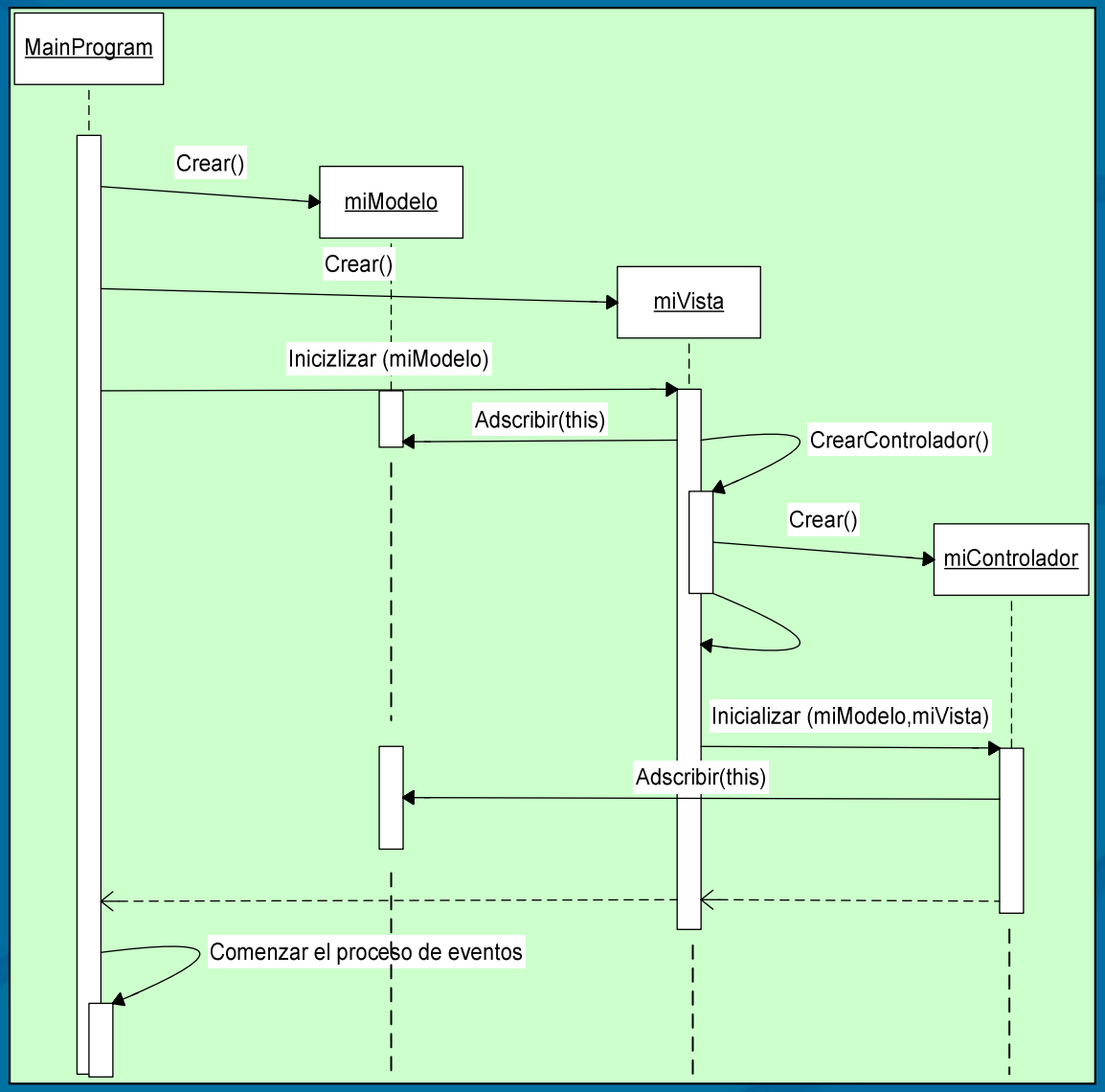
Diseño de Aplicaciones (I)
Ejemplo de Implementación (Conversor €)

- Patrón MVC



Diseño de Aplicaciones (I)
Ejemplo de Implementación (Conversor €)

- Patrón MVC: Creación



Diseño de Aplicaciones (I)
Ejemplo de Implementación (Conversor €)

- Requerimientos:

R1.- Configurar la conversión

R1.1.- Configurar nombre de la moneda

R1.2.- Configurar el valor de cambio

R1.3.- Cancelar modificaciones (Añadido en análisis)

R2.- Realizar conversiones

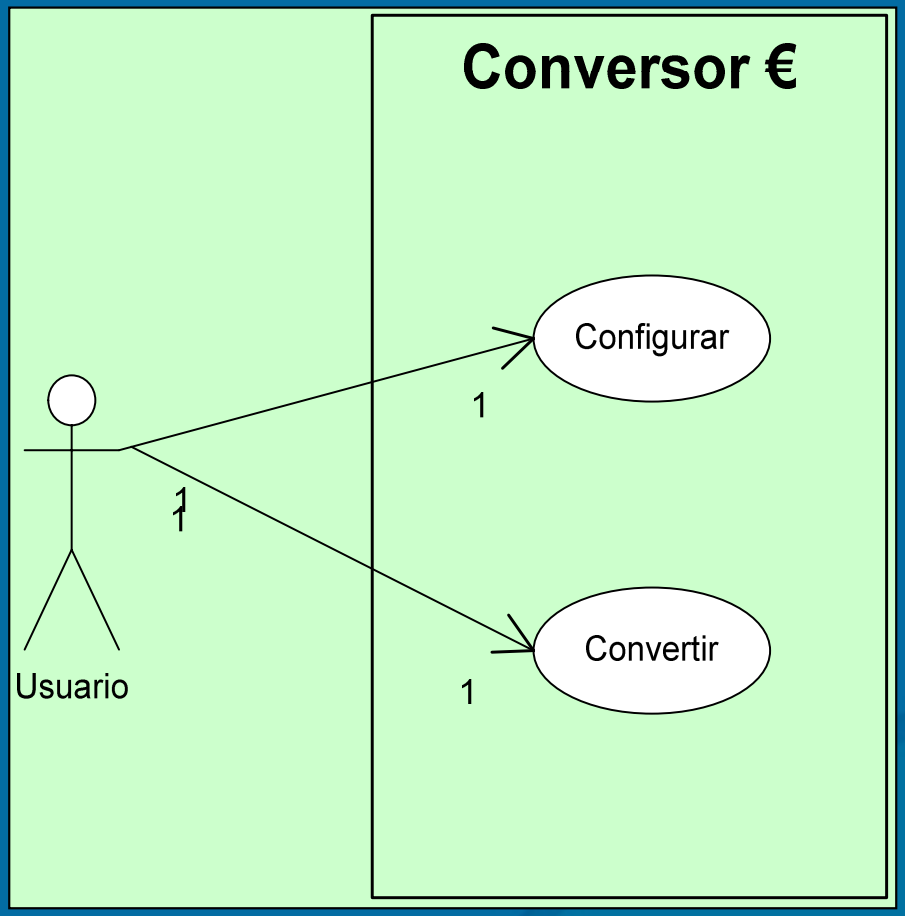
R2.1.- Introducir valor a convertir

R2.2.- Obtener conversión

R2.3.- Limpiar valores a convertir (Añadido en análisis)

Diseño de Aplicaciones (I)
Ejemplo de Implementación (Conversor €)

- Casos de Uso:



Caso 1:

Configurar la conversión: Introducción del nombre de la moneda y del valor de cambio. Poder cancelar el cambio o aceptarlo.

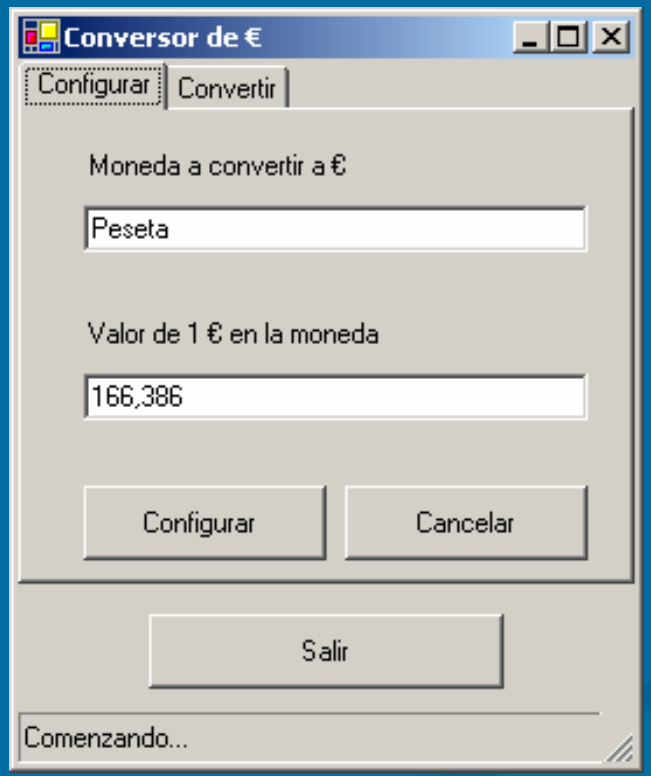
Caso 2:

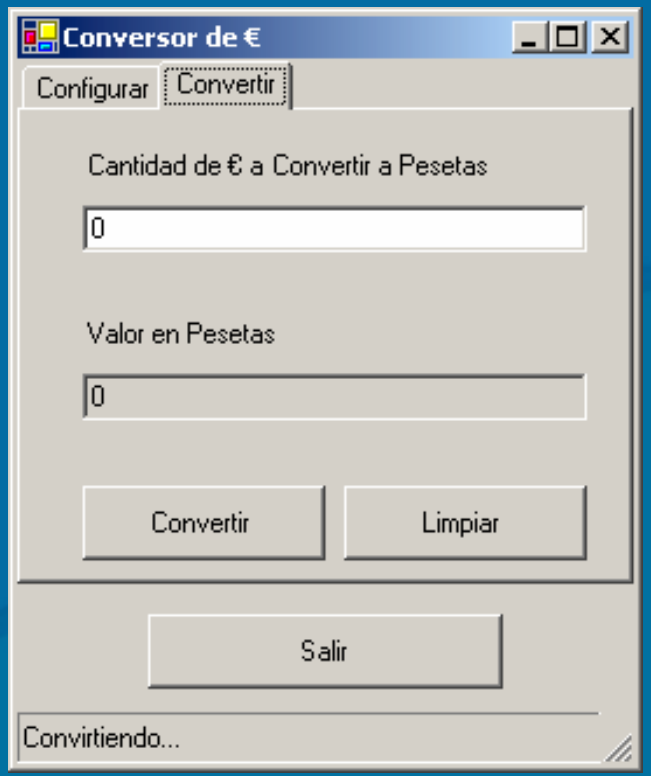
Convertir: Introducir el valor a convertir y realizar la conversión.

Poder limpiar los valores a convertir.

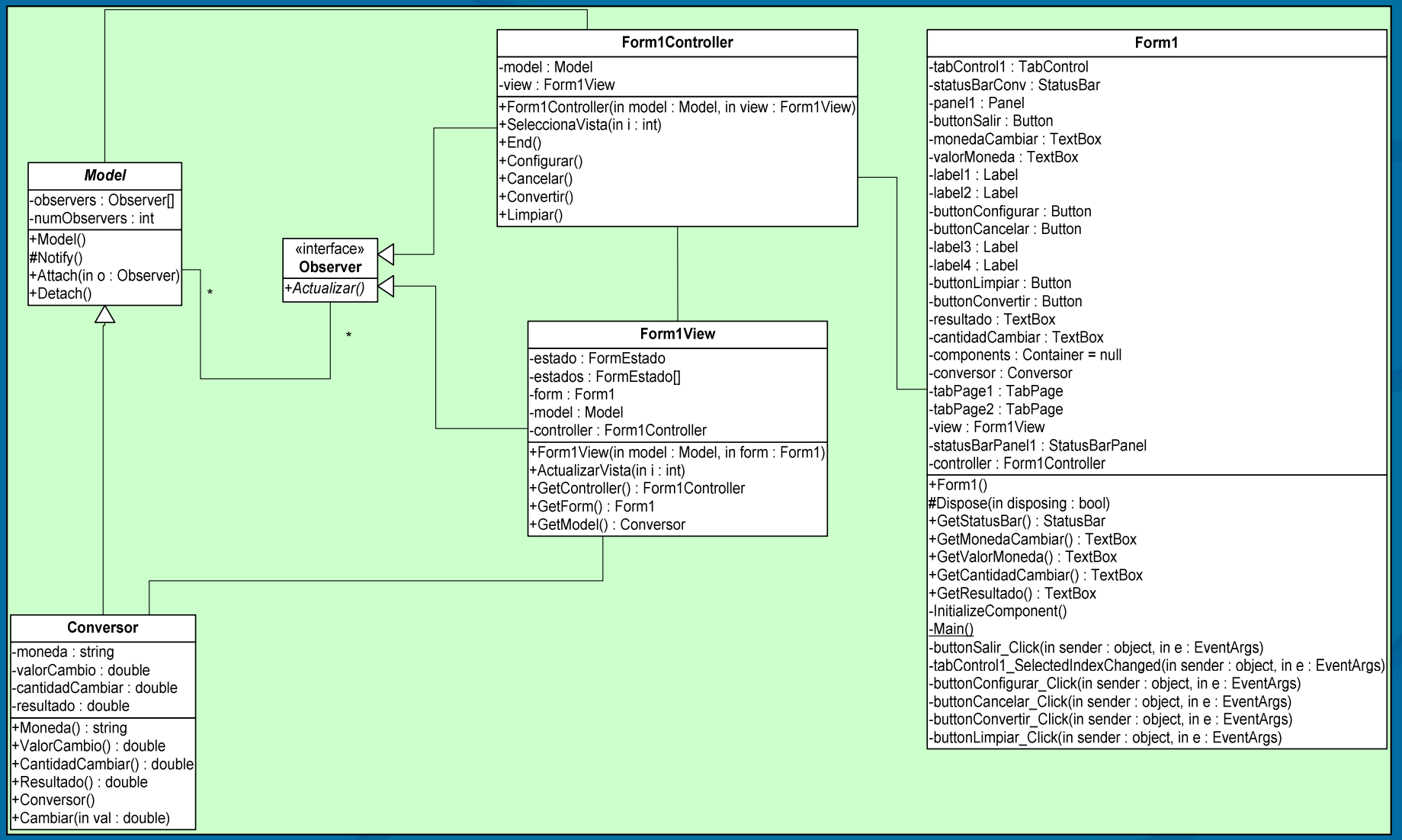
Diseño de Aplicaciones (I)
Ejemplo de Implementación (Conversor €)

- Diseño de la Interfaz...





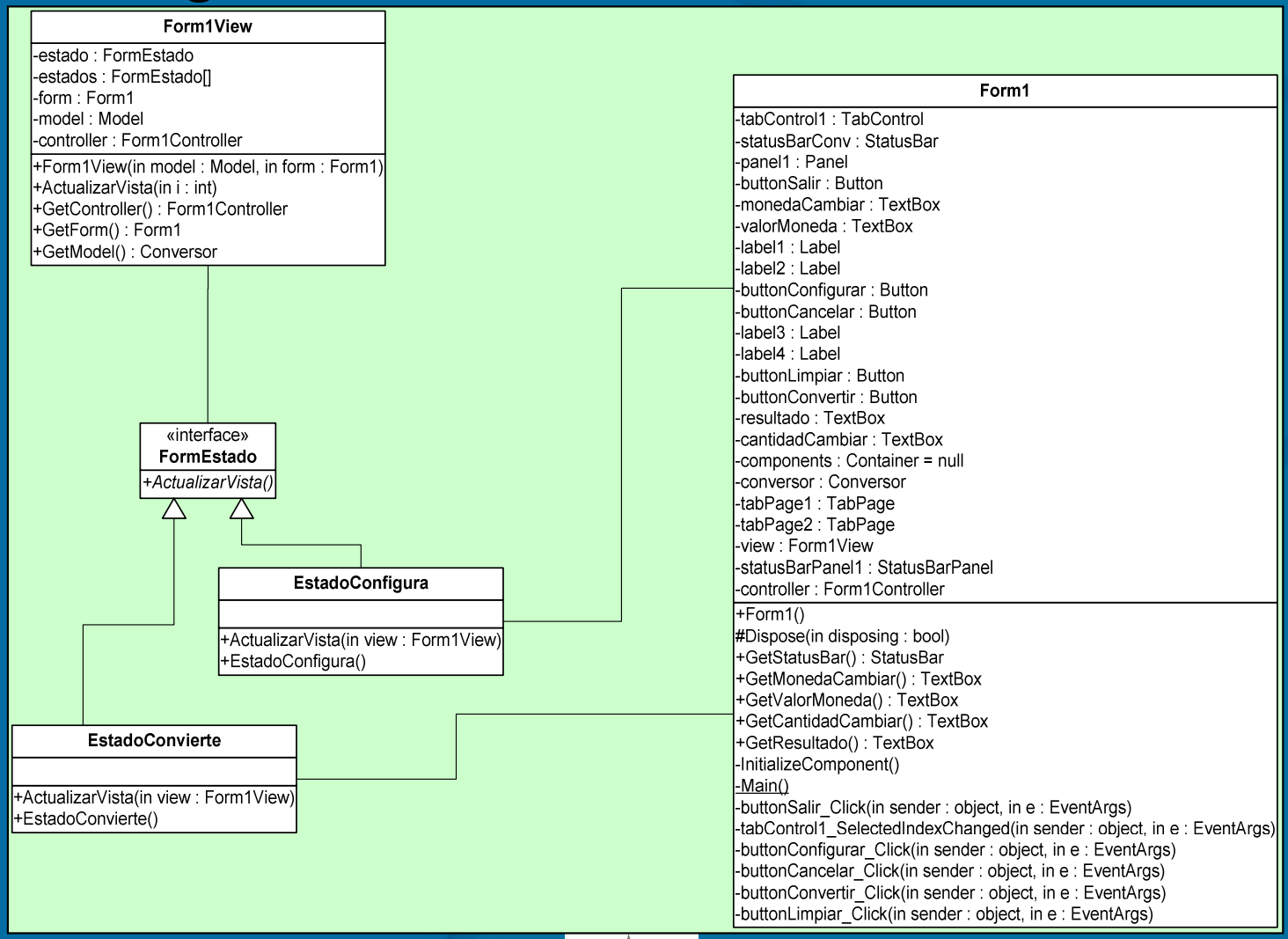
Diseño de Aplicaciones (I)
Ejemplo de Implementación (Conversor €)

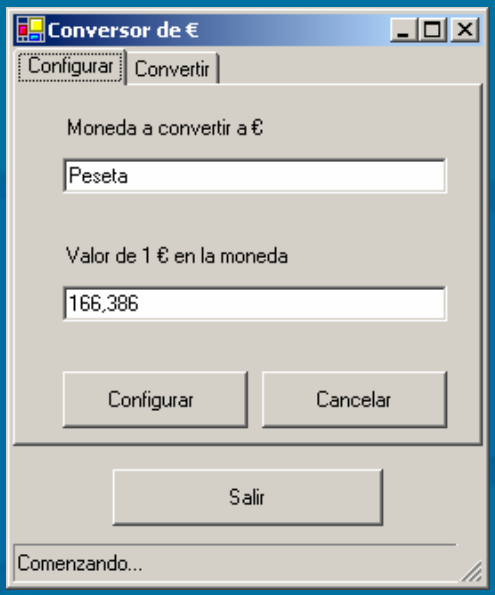


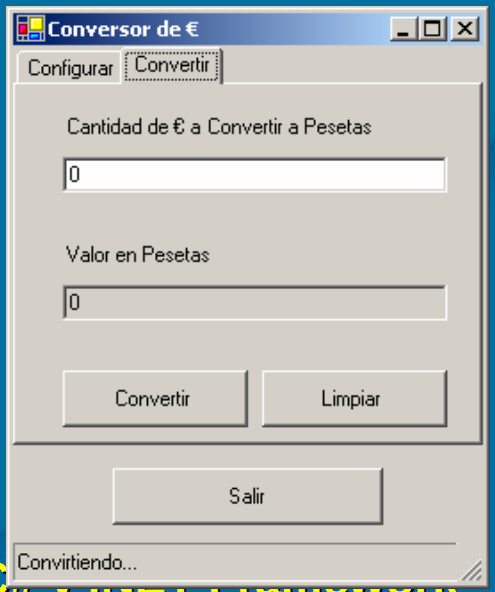
- Diagrama de clases:

Diseño de Aplicaciones (I)
Ejemplo de Implementación (Conversor €)

- Diagrama de clases:

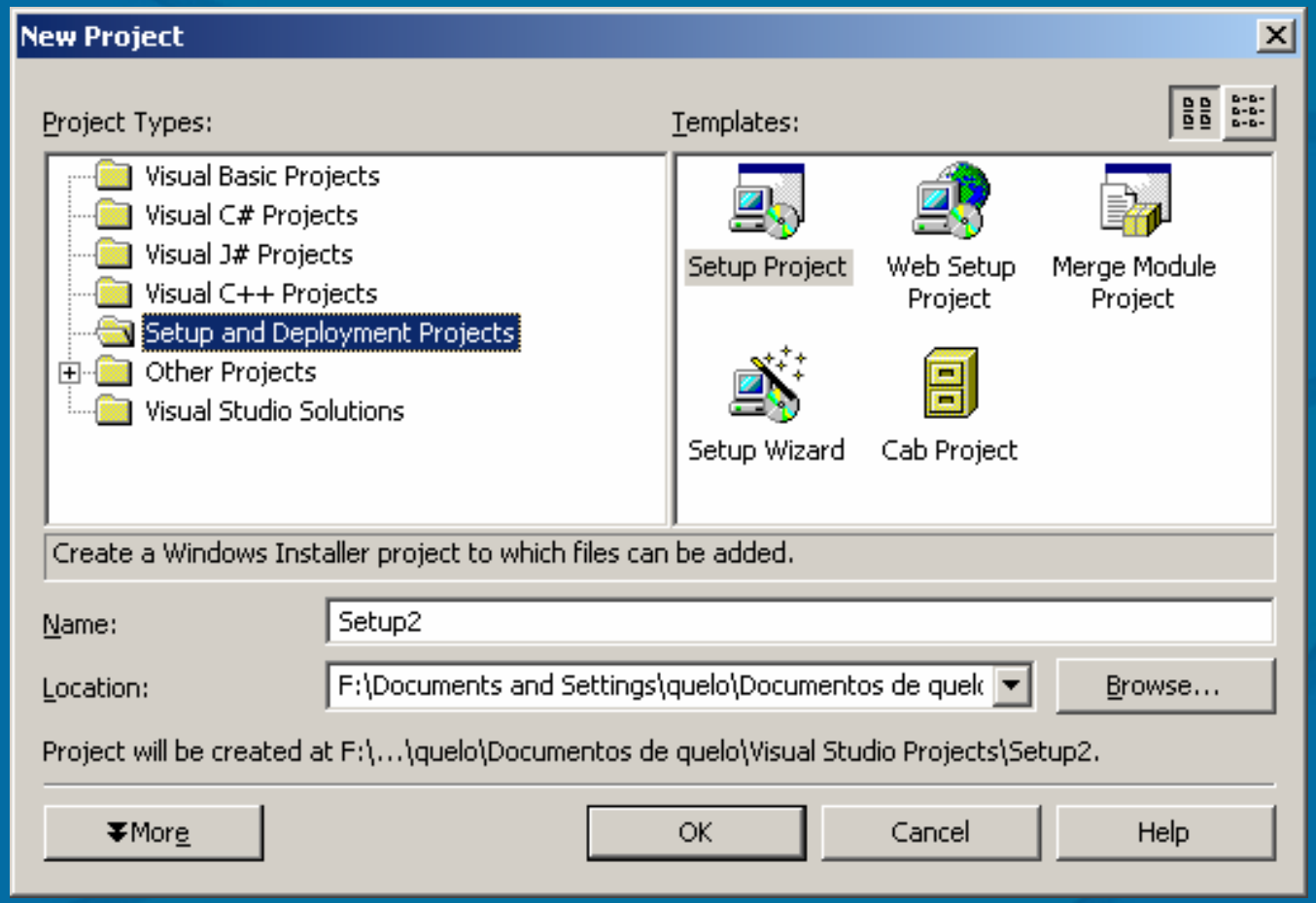






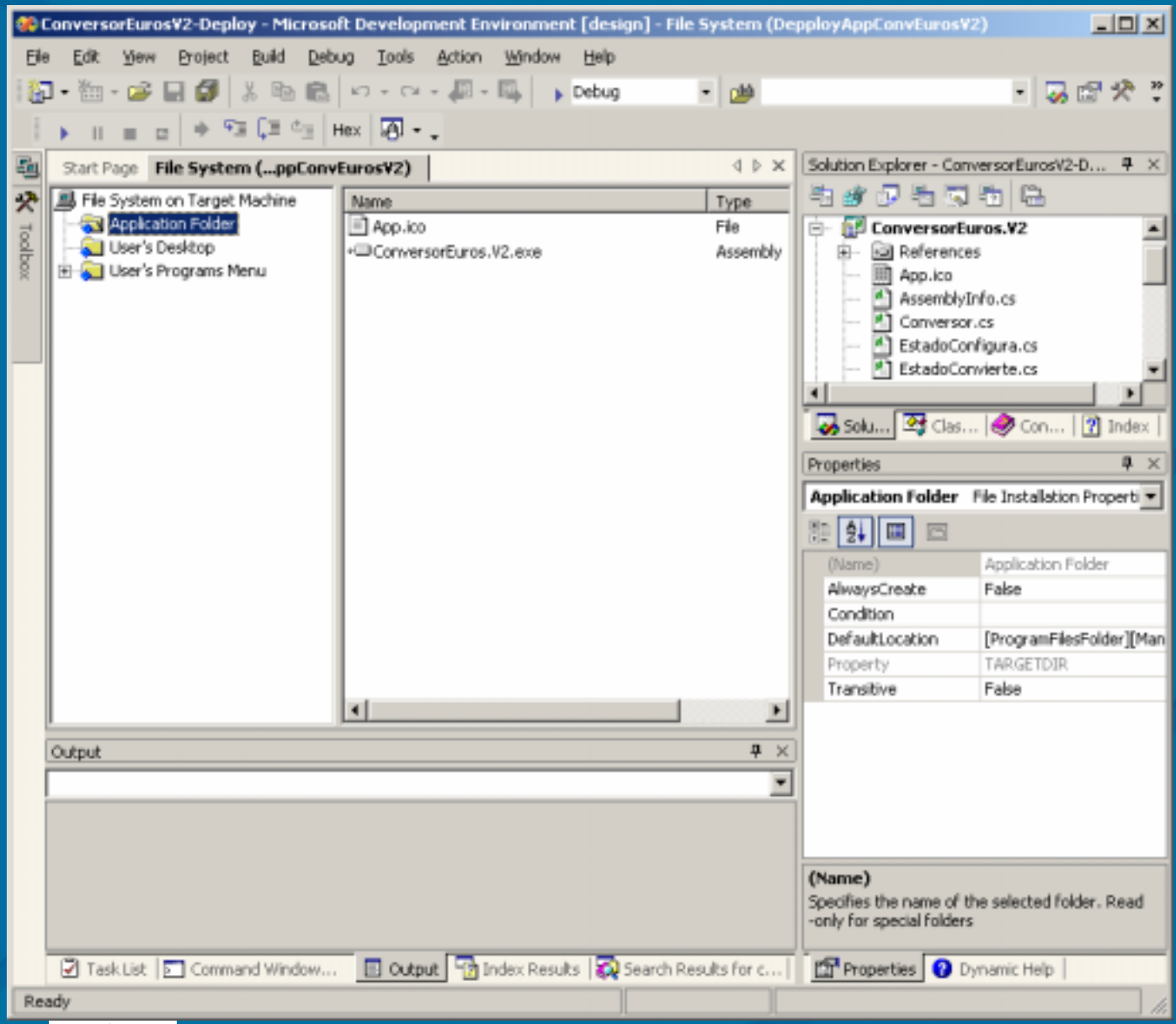
Diseño de Aplicaciones (I)
Deployment

- Para realizar el deployment de una aplicación hay que crear un nuevo proyecto del tipo “Setup & Deployment Projects”



Diseño de Aplicaciones (I)
Deployment

- Después hay que colocar los elementos que se deseen que formen parte del deployment del proyecto.
- Por último hay que construir (Build) el nuevo proyecto.



Diseño de Aplicaciones (II)
Contenidos

- Visión general de la biblioteca de clases del framework
- Realización de varios ejemplos prácticos

Diseño de Aplicaciones (II)
Biblioteca de clases .NET

- La biblioteca de clases es la base fundamental del desarrollo de aplicaciones en .NET.
- Cada grupo de utilidad está dentro de un namespace diferente.
- La convención para navegar por la jerarquía de namespaces es mediante “.” entre los diferentes niveles, ej.: “System.Convert”

Diseño de Aplicaciones (II)
Biblioteca de clases .NET – Lista (1)

Component model	**System.CodeDom**	Representation of the elements and structure of a source code document, and compilation and handling of such code.
	System.ComponentModel	Implementation of components, including licensing and design-time adaptation.
Configuration	**System.Configuration**	Retrieval of application configuration data.
Data	**System.Data**	Access and management of data and data sources.
	System.Xml	Standards-based support for processing XML.
	System.Xml.Serialization	Bidirectional object–to-XML mapping.
Framework services	**System.Diagnostics**	Application instrumentation and diagnostics.
	System.DirectoryServices	Access to Active Directory. The classes in this namespace can be used with any Active Directory service provider, such as Internet Information Services (IIS).
	System.Management	Services and application management tools that work with the Web-Based Enterprise Management (WBEM) standards.
	System.Messaging	Microsoft Message Queuing (MSMQ) access and management, and the sending and receiving of messages.
	System.ServiceProcess	Installation and execution of Windows-based service applications. Does not access specific services, such as Active Directory or Web Services.
	System.Timers	Event raising on an interval or more complex schedule.

Diseño de Aplicaciones (II)
Biblioteca de clases .NET – Lista (2)

Globalization and localization	**System.Globalization**	Support for internationalization and globalization of code and resources.
	System.Resources	Resource management and access, including support for localization.
Net	**System.Net**	Support for sending and receiving data over a network, including simple programming interfaces for common network protocols.
Common tasks	**System.Collections**	Collections of objects, such as lists, queues, arrays, hash tables, and dictionaries.
	System.IO	Basic data stream access and management, including file I/O, memory I/O, and isolated storage.
	System.Text	Character encoding, character conversion, and string manipulation.
	System.Text.RegularExpressions	Full regular expression support.
	System.Threading	Multithreaded programming support, including locking and synchronization.
Reflection	**System.Reflection**	Access to type metadata and dynamic creation and invocation of types.
Rich, client-side GUI	**System.Drawing**	Rich 2-D graphics functionality, and access to GDI+.
	System.Windows.Forms	Rich user interface features for Windows-based applications.

Diseño de Aplicaciones (II)
Biblioteca de clases .NET – Lista (3)

Runtime infrastructure services	**System.Runtime.CompilerServices**	Support for compilers that target the runtime.
	System.Runtime.InteropServices	Support for interoperability with COM and other unmanaged code.
	System.Runtime.Remoting	Support for creating tightly or loosely coupled distributed applications.
	System.Runtime.Serialization	Object serialization and deserialization, including binary and SOAP encoding support.
.NET Framework security	**System.Security**	Access to the underlying mechanisms of the .NET Framework security system, including policy resolution, stack walks, and permissions.
	System.Security.Cryptography	Cryptographic services, including encoding and decoding of data, hashing, random number generation, message authentication, and formation of digital signatures.
Web Services	**System.Web**	Support for Web server and client management, communication, and design. Provides core infrastructure for ASP.NET, including Web Forms support.
	System.Web.Services	Client- and server-side support for SOAP-based Web services.

Diseño de Aplicaciones (II)
Biblioteca de clases .NET - Colecciones

- **ArrayList**
  - Implementa la interfaz **IList** utilizando un array cuyo tamaño se incrementa dinámicamente.
- **BitArray**
  - Maneja un array compacto de bits que representan valores booleanos (0 – F, 1 – V).
- **CollectionBase**
  - Provee la clase base abstracta para una colección fuertemente tipada.
- **DictionaryBase**
  - Provee la clase base abstracta para una colección fuertemente tipada de pares clave-valor.
- **Hashtable**
  - Representa una colección de pares clave-valor organizados en base a una clave hash.
- **Queue**
  - Representa una colección FIFO (first-in, first-out) de objetos.
- **ReadOnlyCollectionBase**
  - Provee la clase base abstracta para una colección fuertemente tipada de sólo lectura.
- **SortedList**
  - Representa una colección de pares clave-valor ordenados por clave y accesibles mediante un índice de la clave.
- **Stack**
  - Representa una simple colección LIFO (last-in, first-out) de objetos.

Diseño de Aplicaciones (II)
Biblioteca de clases .NET - Características

- Hay un par de características comunes a todas las colecciones:
  - Los miembros estáticos y públicos de estas colecciones son seguros para operaciones multihilo. Para los miembros de las instancias no está garantizada la seguridad.
  - La enumeración a través de una collection no es un procedimiento seguro para el multihilo. Incluso cuando la collection está sincronizada, otros hilos pueden estar modificando la collection. Para garantizar la seguridad durante la enumeración se puede bloquear la collection o capturar las excepciones que resulten de los cambios realizados por otros hilos.

Diseño de Aplicaciones (II)
Biblioteca de clases .NET - Abstractas

- **Las colecciones abstractas no pueden crear elementos y simplemente dan implementaciones por defecto para algunos elementos.**
- **Pueden ser heredadas por los implementadores para crear sus propias colecciones.**
- **Es importante comprobar el nivel de seguridad en acceso a estas colecciones, ya que puede no tener seguridad de acceso concurrente.**
  - **CollectionBase**
    - Provee la clase base abstracta para una colección fuertemente tipada.
  - **DictionaryBase**
    - Provee la clase base abstracta para una colección fuertemente tipada de pares clave-valor.
  - **ReadOnlyCollectionBase**
    - Provee la clase base abstracta para una colección fuertemente tipada de sólo lectura.

Diseño de Aplicaciones (II)
Biblioteca de clases .NET - Interfaces

- **ICollection**
  - Define métodos para enumeradores, tamaño y sincronización para todas las colecciones.
- **IComparer**
  - Contiene la definición para un método que compara dos objetos.
- **IDictionary**
  - Representa una colección de pares clave-valor.
- **IDictionaryEnumerator**
  - Enumera los elementos de un diccionario.
- **IEnumerable**
  - Define el enumerador que soporta una simple iteración sobre una colección.
- **IEnumerator**
  - Soporta una simple iteración sobre una colección.
- **IHashCodeProvider**
  - Suministra un código hash para un objeto usando una función hash de usuario.
- **IList**
  - Representa una colección de objetos que pueden ser accedidos individualmente mediante índice.

Diseño de Aplicaciones (II)
Biblioteca de clases .NET - ArrayList

- **ArrayList**
  - Implementa la interfaz **IList** utilizando un array cuyo tamaño se incrementa dinámicamente.
  - Un **ArrayList** puede soportar de manera segura varios lectores concurrentes mientras la lista no se modifique. Para garantizar el acceso seguro, las operaciones deben ser realizadas mediante el **wrapper** retornado por el método **Synchronized**.
  - La capacidad de un **ArrayList** es el número de elementos que la lista puede contener. Como la capacidad aumenta a medida que se necesita para contener nuevos elementos, se puede hacer que decrezca mediante el método **TrimToSize** o colocando en **Capacity** el valor adecuado.
  - Los índices de esta colección comienzan a contar en cero (zero-based).

Diseño de Aplicaciones (II)
Biblioteca de clases .NET - BitArray

- **BitArray**
  - Maneja un array compacto de bits que representan valores booleanos (0 – F, 1 – V).
  - Esta implementación no provee un **wrapper** sincronizado (**thread-safe**) para un **BitArray**.
  - El tamaño de un BitArray es controlado por el cliente, la indexación de elementos más allá del final del **BitArray** lanza una **ArgumentException**.
  - Los índices de esta colección comienzan a contar en cero (zero-based).

Diseño de Aplicaciones (II)
Biblioteca de clases .NET - Hashtable

- **Hashtable**
  - Representa una colección de pares clave-valor organizados en base a una clave hash.
  - Un **Hashtable** puede soportar de forma segura un escritor y múltiples lectores concurrentes. Para poder soportar múltiples lectores, todas las operaciones deben realizar mediante el wrapper devuelto por el método **Synchronized**.
  - Cada elemento de la **Hashtable** es un par clave-valor almacenado en un objeto **DictionaryEntry**.
  - Los objetos usados como clave en una **Hashtable** debe implementar o heredar los métodos **Object.GetHashCode** y **Object.Equals**.
  - Si la igualdad de los objetos fuera simplemente la igualdad de las referencias, no es necesaria la implementación de esos métodos.
  - Los objetos clave debe permanecer inalterados mientras sean usados como clave en un **Hashtable**.

Diseño de Aplicaciones (II)
Biblioteca de clases .NET - Queue

- **Queue**
  - Representa una colección FIFO (first-in, first-out) de objetos.
  - Para poder soportar múltiples lectores, todas las operaciones en la **Queue** deben realizarse mediante el **wrapper** devuelto por el método **Synchronized**.
  - Las colas son útiles para almacenar mensajes en el orden en que se reciban para el subsecuente procesamiento.
  - Esta clase implementa la cola como un **array** circular.
  - Si el número de elementos añadidos supera la capacidad de la cola, ésta se aumenta de tamaño automáticamente.
  - La capacidad se puede decrementar llamando a **TrimToSize**.

Diseño de Aplicaciones (II)
Biblioteca de clases .NET - SortedList

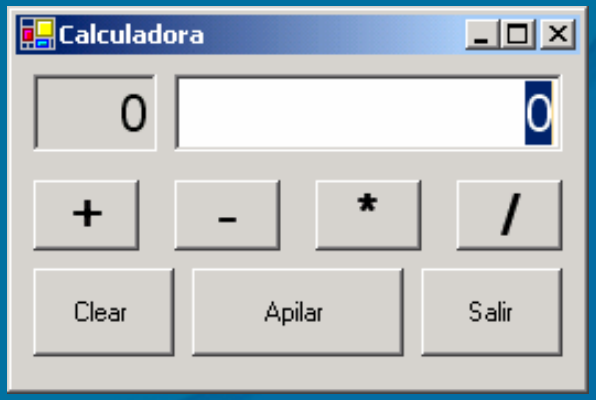
- **SortedList**
  - Representa una colección de pares clave-valor ordenados por clave y accesibles mediante un índice de la clave.
  - Para poder soportar múltiples lectores, todas las operaciones en la **SortedList** deben realizarse mediante el wrapper devuelto por el método **Synchronized**.
  - Una SortedList es un híbrido entre un **Array** y una **Hashtable**. Cuando un elemento se accede mediante su clave usando el indexador **Item**, se comporta como una Hashtable, pero cuando se accede por su índice mediante **GetByIndex** o **SetByIndex**, se comporta como un **Array** (Internamente mantiene dos arrays).

Diseño de Aplicaciones (II)
Biblioteca de clases .NET - Stack

- **Stack**
  - Representa una simple colección LIFO (last-in, first-out) de objetos.
  - **Stack** se implementa internamente como un buffer circular.
  - Si **Count** es menor que la capacidad del **Stack**, **Push** es una operación de complejidad O(1). Si la capacidad necesita incrementarse para poder introducir nuevos elementos, **Push** tiene una complejidad de O(n).
  - **Pop** siempre tiene de complejidad O(1).

Diseño de Aplicaciones (II)
Ejercicio 1

- Implementar una calculadora que haga las operaciones en notación polaca inversa.
- En este tipo de notación no son necesarios los paréntesis. La calculadora debe tener el interfaz que se muestra.
- Ejemplos:
  - 12 + 3 * (2 + 5) → 12 ↵3 ↵2 ↵5 ↵ + * +
  - 3 + 6 → 3 ↵ 6 +
  - 1 + 2 + 3 → 1 ↵ 2 + 3 +
  - 1 + 2 * 3 + 4 → 1 ↵ 2 ↵ 3 * + 4 +



Diseño de Aplicaciones (II)
Ejercicio 2

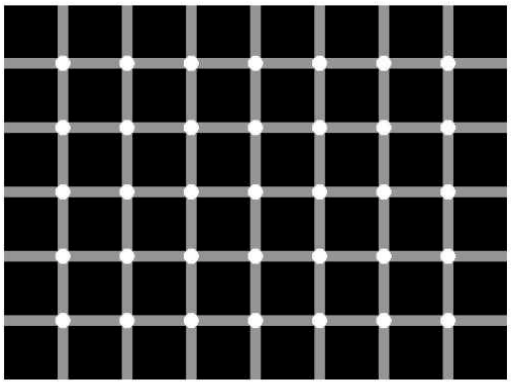
- Realizar una aplicación que dada una colección de números haga cáculos estadísticos.
- Debe tener el interfaz que se muestra.
- Los calculos son:
  - Número de elementos (siempre)
  - Máximo
  - Mínimo
  - Media

Diseño de Aplicaciones (II)
Ejemplo 3

- Hacer un diccionario que, buscando una palabra devuelva su traducción al inglés.

Diseño de Aplicaciones (III)
Contenidos

- Ficheros en C#
- Acceso a Base de Datos
- Implementación de varios ejemplos prácticos



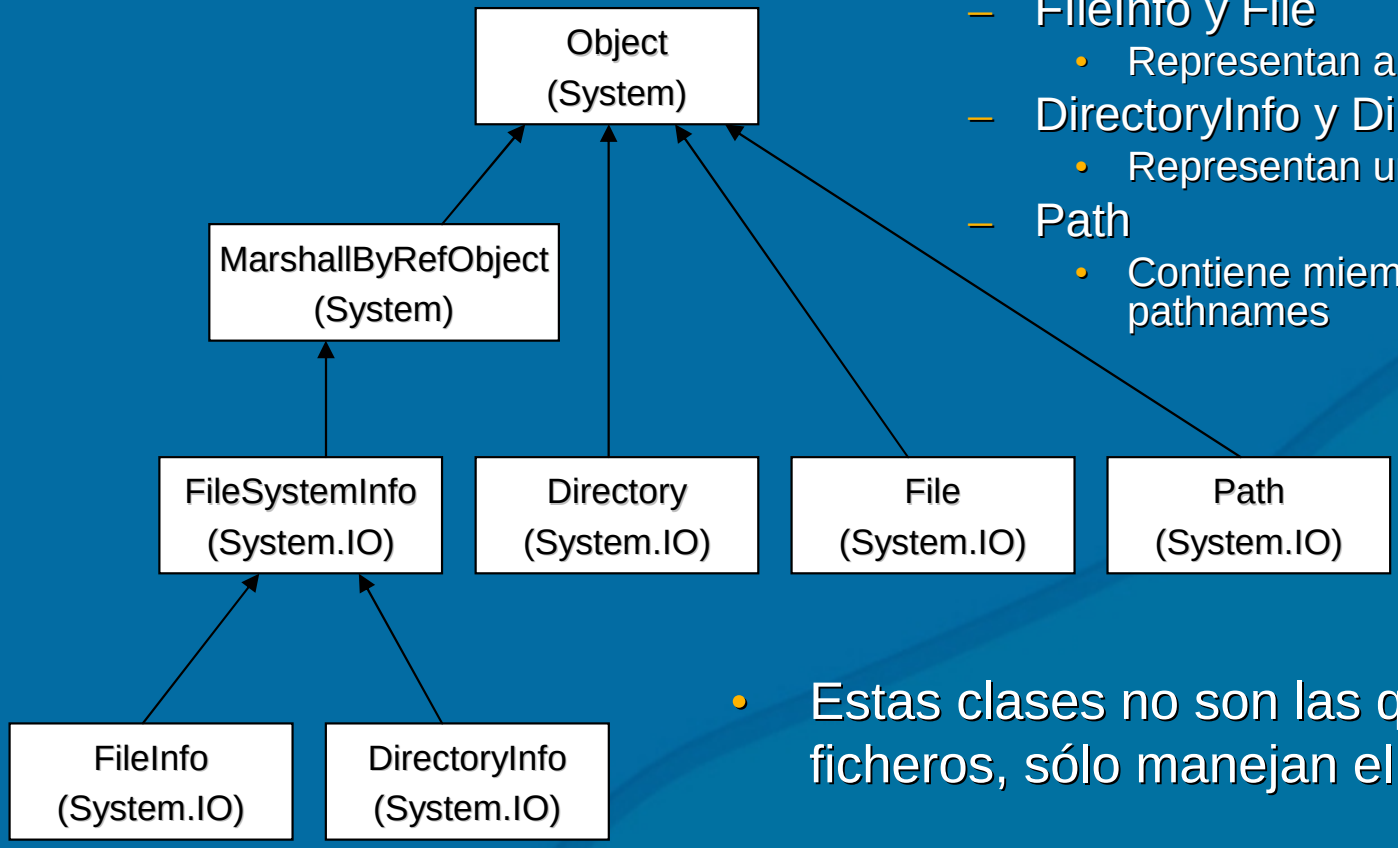
Cuenta los puntos negros :o)

Diseño de Aplicaciones (III)
Ficheros en C#

- Manupulación del Sistema de Ficheros
  - Mediante el framework se puede acceder a ficheros y directorios para modificarlos, copiarlos, crearlos o destruirlos.
  - Para utilizar estas librerías se debe tener acceso al sistema y la aplicación debe incluir el namespace **System.IO**

Diseño de Aplicaciones (III)
Ficheros en C#

- Jerarquía de clases:
  - SystemMarshallByRefObject
    - Clase base para permitir el control de los ficheros y comunicación con el sistema
  - FileSystemInfo
    - Clase base que representa a los ficheros que se manejarán mediante instancias
  - FIleInfo y File
    - Representan a un fichero en el sistema
  - DirectoryInfo y Directory
    - Representan un directorio
  - Path
    - Contiene miembros estáticos para manejar pathnames



- Estas clases no son las que leen y encriben en ficheros, sólo manejan el sistema de ficheros.

Diseño de Aplicaciones (III)
Ficheros en C#

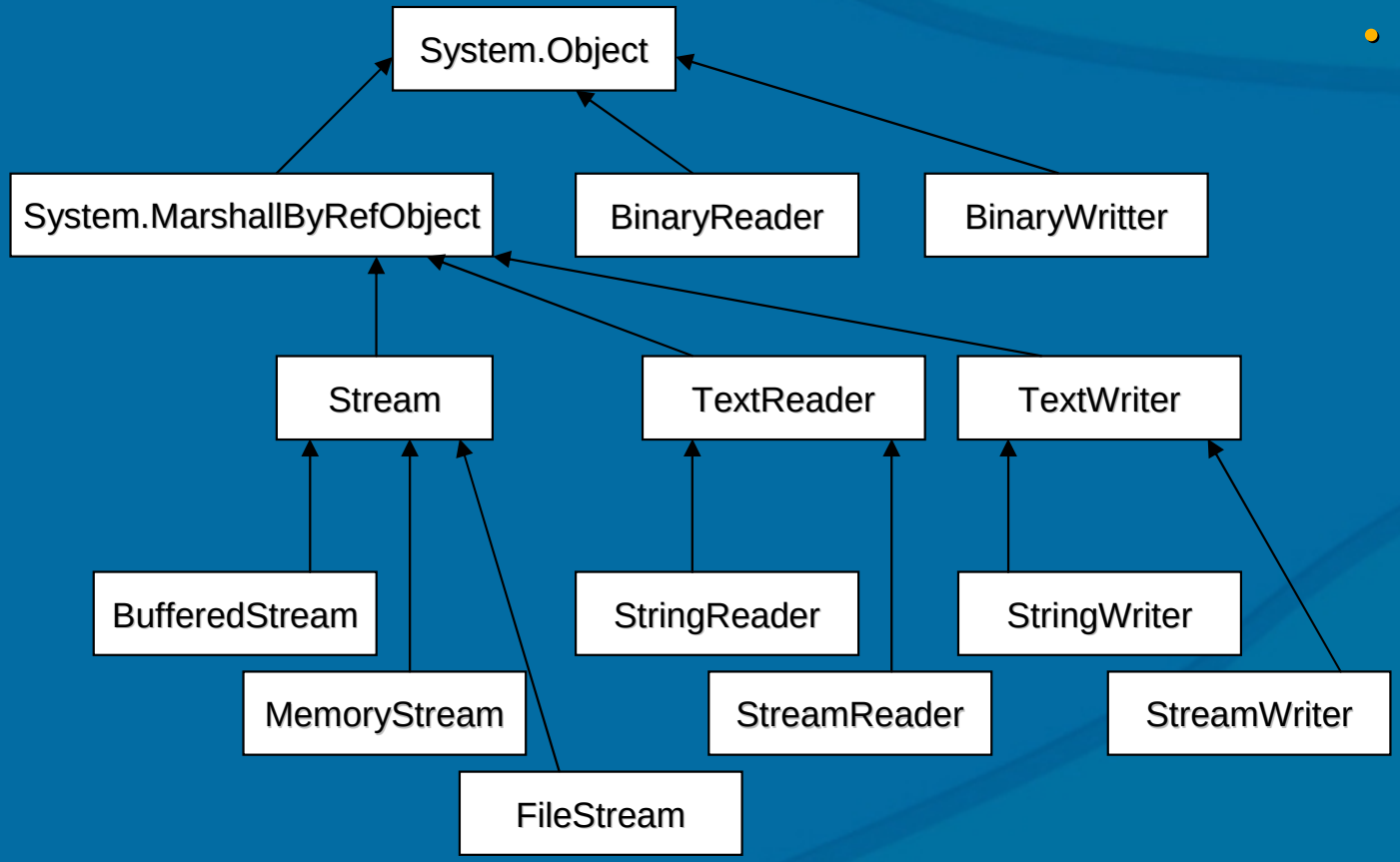
- Las clases FileInfo y DirectoryInfo tienen propiedades y operaciones para mnipular ficheros y acceder a la información de éstos:
  - Propiedades
    - CreationTime, DirectoryName, Parent, Exist, Extension, FullName, LastAccessTime, LastWriteTime, Name, Root, Length
  - Operaciones
    - Create(), Delete(), MoveTo(), CopyTo(), GetDirectories(), GetFiles(), GetFileSystemObjects()
  - No es una lista exahustiva de las operaciones ni de las propiedades, sino sólo algunas de las más importantes.

Diseño de Aplicaciones (III)
Ficheros en C#

- Escritura y lectura de ficheros
  - Se realiza mediante Streams
    - Cuando se envía información al fichero se está ecridiendo la Stream y cuando se recoge información se dice que le lee la Stream.
- Mediante Streams, de la misma manera que se escriben y leen ficheros se pueden leer y escribir direcciones de red, memoria, variables, etc.
- Algunas de estas operacioens ya están sobrecargadas en el .NET Framework, otras pueden ser implementadas por el usuario según sus necesidades heredando de System.IO.Stream

Diseño de Aplicaciones (III)
Ficheros en C#

- Jerarqúia de Clases:



- Las clases que más nos importan para leer y escribir ficheros son:
  - FileStream : Para leer/escribir ficheros binarios.
  - StreamReader y StreamWriter para leer/escribir ficheros de texto.

Diseño de Aplicaciones (III)
Ficheros en C#

- Ejemplo de FileStream:

```
FileStream fs = new FileStream(@"C:\C# Projects\Projects2.doc",
   FileMode.Create, FileAccess.Write);

...

int NextByte = fs.ReadByte();

...

fs.WriteByte(NextByte);

...

byte [] myByteArray = new byte[100];

int nBytesRead = fs.Read(myByteArray, 0 /*Offset*/, nBytes);

...

Fs.Close()
```

Diseño de Aplicaciones (III)
Ficheros en C#

- Escribir y leer ficheros de texto.

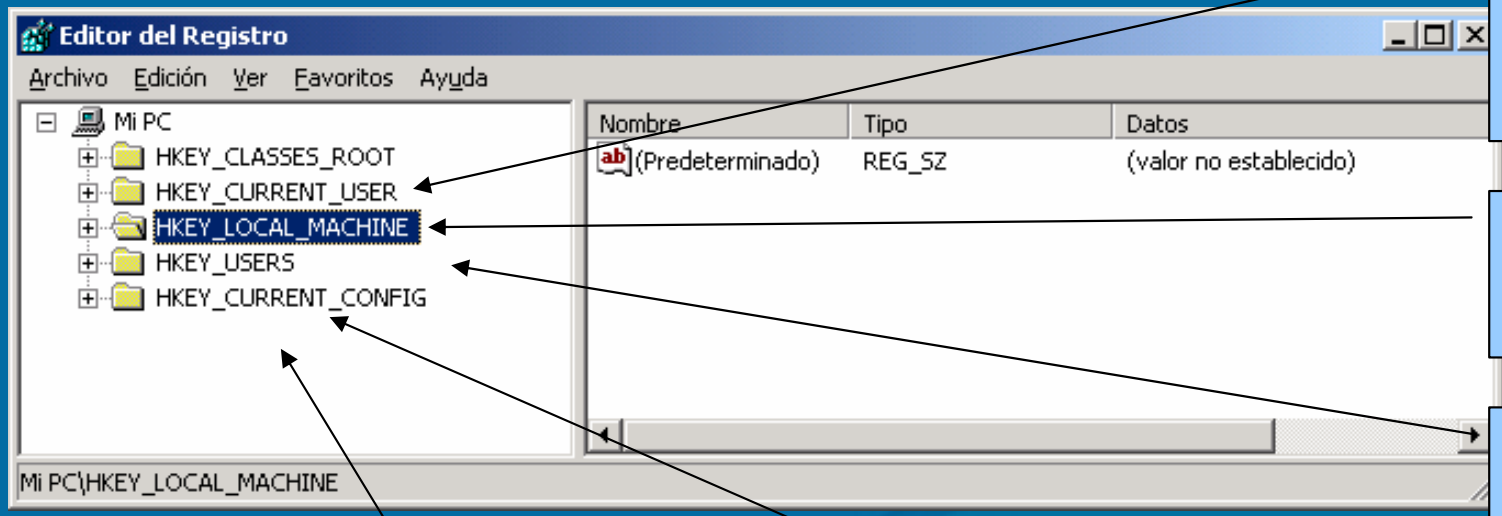
```
StreamReader sr = new StreamReader(@"C:\My Documents\ReadMe.txt");
string nextLine = sr.ReadLine();
string fichero = sr.ReadToEnd();
int nextChar = sr.Read();
sr.Close();
...
StreamWriter sw = new StreamWriter(@"C:\My Documents\ReadMe.txt");
sw.Write();
Sw.Close();
...
FileInfo nyFile = new FileInfo(@"C:\My Documents\ReadMe.txt");
StreamWriter sw = myFile.CreateText();
```

Diseño de Aplicaciones (III)
Ficheros en C#

- Hacer un sencillo editor de texto mediante un TextBox, dialogos de abrir y cerrar ficheros y las clases para menejo de ficheros.
- Debe tener las opciones de cargar, grabar y salir.

Diseño de Aplicaciones (III)
El Registro

- El Registro es el repositorio principal de configuraciones de Windows a partir de la versión 95.



Contiene detalles de la configuración de ficheros en el sistema

Contiene detalles de la configuración del usuario actual

Contiene detalles de la configuración de ficheros en el sistema

Contiene detalles de la configuración del hardware de la máquina

Contiene detalles de la configuración para todos los usuarios de la máquina

Diseño de Aplicaciones (III)
El Registro

- Es necesario usar el namespace Microsoft.Win32
- Registry
  - Provee objetos de RegistryKey y nunca se instancia
- RegistryKey
  - Representa una clave del registro. Tiene métodos para acceder, navegar, crear y borrar claves.

Diseño de Aplicaciones (III)
El Registro

```
RegistryKey hklm = Registry.LocalMachine;

RegistryKey hkSoftware = hklm.OpenSubKey("Software");

RegsitryKey hkMicrosoft = hkSoftware.OpenSubKey("Microsoft" /*, true
   -> para accecer en read-write*/);

RegistryKey hkMine = hkSoftware.CreateSubKey("MyOwnSoftware");

hkMine.SetValue("MyStringValue","HelloWorld");

hkMine.SetValue("MyINtValue", 20);

...

string stringValue = (string) hkMine.GetValue("MyStringValue");

int intValue = (int) hkMine.GetValue("MyIntValue");

...

hkMine.Colse();

...
```

Name, SubKeyCount, ValueCount, Close(), CreateSubKey(), DeleteSubKey(), DeleteSubKeyTrre(), DeleteValue(), GatSubKeyNames(), GetValue(), GetValueNames(), OpenSubKey(), GetValue()

Diseño de Aplicaciones (III)
El Registro

- Poner a la aplicación anterior una entrada en el registro para que recuerde el nombre del último fichero que se abrió y del último que se guardó en fichero.
- Otra entrada que cuente cuantos ficheros se han cargado en la aplicación.

Diseño de Aplicaciones (III)
Acceso a Bases de Datos

- Para utilizar Bases de Datos en ADO.NET es una evolución de ADO para la plataforma .NET.
- Es necesario utilizar los siguientes namespaces:
  - System.Data
    - Todas las clases de acceso genéricas
  - System.Data.Common
    - Clases compartidas o reimplementadas por los proveedores de BD
  - System.Data.SqlClient
    - Clases para los servidores de SQL
  - System.Data.SqlTypes
    - Tipos de datos de los servidores SQL

Diseño de Aplicaciones (III)
Acceso a Bases de Datos

- Clases Compartidas ADO.NET (Shared Classes)
  - DataSet (System.Data)
    - Este objeto puede contener un conjunto de DataTable's, puede incluir relaciones entre esas tablas y está diseñado para uso desconectado
  - DataTable (System.Data)
    - Es un contenedor de datos. Una DataTable está formada por una o más DataColumn's y cuando tiene datos contiene también DataRwo's.
  - DataRow (System.Data)
    - Una serie de valores, una fila de una base de datos o de una hoja de cálculo.
  - DataColumn (System.Data)
    - Contiene la definición de una columna: tipo, nombre, etc.
  - DataRelation (System.Data)
    - Es una conexión entre dos DataTable's en un DataSet. Se usa para claves externas y para relaciones maestro/cliente
  - Constraint (System.Data)
    - Define una restricción o una regla a aplicar a una DataColumn (o a un conjunto de ellas), por ejemplo "Unique Values"
  - DataColumnMapping (System.Data.Common)
    - Mapea el nombre de una columna de la BD con una columna dentro de la DataTable
  - DataTableMapping (System.Data.Common)
    - Mapeael nombre de un tabla de la BD con una DataTable

Diseño de Aplicaciones (III)
Acceso a Bases de Datos

- Clases específicas de BD
  - SqlCommand
  - SqlCommandBuilder
  - SqlConnection
  - SqlDataAdapter
  - SqlDataReader
  - SqlParameter
  - SqlTransaction

Diseño de Aplicaciones (III)
Acceso a Bases de Datos

```
using System.Data.SqlClient;
string source = “server=(local)\\NetSDK;uid=QSUser;pwd=QSPassword;” + “database=Northwind”;
SqlConnection conn = new SqlConnection(source);
try
{
    // Abrir la conexión
    conn.Open();
    // Hacer algo útil...
}
catch (Exception ex)
{
    // Tratar la excepción
}
finally
{
    // Asegurar que se cierra la conexión
    conn.CLose();
}
```

- Conexiones:
  - Se debe asegurar que se cierra la conexión después de su uso

Diseño de Aplicaciones (III)
Acceso a Bases de Datos

- Transacciones
  - Son operaciones que se suponen “atómicas” o protegidas durante al acceso a una BD. Si no se completan se debe restaurar la BD al estado anterior.

```
...
try
{
    conn.Open();
    SqlTransaction tx = conn.BeginTransaction();
    // Hacer operaciones...
    tx.Commit();
}
catch ...
finally
{
    conn.CLose();
}
```

Diseño de Aplicaciones (III)
Acceso a Bases de Datos

- Comandos:
  - Los comandos pueden ser definidos durante la programación o ser procesos almacenados.

```
string select = "SELECT nombre FROM clientes WHERE
   ciudad=\'Oviedo\'"

...

conn.Open();

SqlCommand cmd = new SqlCommand(select,conn);
```

Diseño de Aplicaciones (III)
Acceso a Bases de Datos

- Ejecución de los comandos
  - La clase SqlCommand tiene los métodos:
    - ExecuteNonQuery()
      - Ejecuta un comando pero no devuelve ningun resultado
      - UPDATE, INSERT y DELETE
    - ExecuteReader()
      - Ejecuta el comando un devuelve un IDataReader
      - Se puede iterar sobre el objeto devuelto (SqlDataReader) para leer las tuplas devueltas
      - SELECT (general)
    - ExecuteScalar()
      - Ejecuta el comando y devuleve un valor simple
      - SELECT COUNT (*), ...

Diseño de Aplicaciones (III)
Threads

- Las threads son hilos de ejecución aparentemente en paralelo (multitarea pre-emptiva).
- Se pueden lanzar en paralelo métodos de obejtos o métodos de clase.
- Se precisa el namespace System.Threading
- Se crea un objeto de la clase Thread.

Diseño de Aplicaciones (III)
Ejercicio

- Hacer una aplicación que lea números de un fichero de texto (números separados por retorno de carro, coma decimal) y que los ordene y los escriba ordenados en otro fichero de salida.

Diseño de Aplicaciones (III)
Ejercicio

- Hacer una aplicación que simule la venta de entradas de cine a una sala multicine.
- El usuario es la persona que atiende la taquilla de los clientes. Puede haber varias taquillas.
- Hay varias salas con capacidad concreta (no hace falta controlar las posiciones de las butacas).
- En cada sala se proyectan sesiones de películas.