



**Framework .NET**

Ingegneria del software L-A

- È un ambiente di esecuzione (*runtime environment*)
- Semplifica lo sviluppo e il *deployment*
- Aumenta l'affidabilità del codice
- Unifica il modello di programmazione
- È completamente indipendente da COM (*Component Object Model*)
- È fortemente integrato con COM

2

**Sviluppo semplificato**

Ingegneria del software L-A

- Ambiente *object-oriented*
  - Qualsiasi entità è un oggetto
  - Classi ed ereditarietà pienamente supportati
  - Anche tra linguaggi diversi
- Riduzione errori comuni di programmazione
  - Linguaggi fortemente tipizzati – **Type Checker**
  - Errori non gestiti – generazione di eccezioni
  - Meno *memory leak* – **Garbage Collector**

3

**Indipendenza dalla piattaforma**

Ingegneria del software L-A

- .NET è un'implementazione di CLI
  - *Common Language Infrastructure*
- CLI e il linguaggio C# sono standard ECMA
  - ECMA-334 (C#), ECMA-335 (CLI)
- Esistono altre implementazioni di CLI:
  - **SSCLI** (Shared Source CLI by Microsoft, per Windows, FreeBSD e Macintosh) - **Rotor**
  - **Mono** (per Linux)
  - **DotGNU**
  - **Intel OCL** (Open CLI Library)
  - ...

4

### Standard ECMA-335

Ingegneria del software L-A

- Defines the Common Language Infrastructure (CLI) in which applications written in **multiple high level languages** may be executed in **different system environments** without the need to rewrite the application to take into consideration the unique characteristics of those environments
- CLI is a **runtime environment**, with:
  - a file format
  - a common type system
  - an extensible metadata system
  - an intermediate language
  - access to the underlying platform
  - a factored base class library

5

### Piattaforma multi-linguaggio

Ingegneria del software L-A

- Libertà di scelta del linguaggio
  - Tutte le funzionalità del *framework* .NET sono disponibili a tutti i linguaggi .NET
  - I componenti di un'applicazione possono essere scritti con diversi linguaggi
- Impatto sui tool
  - Tool disponibili per tutti i linguaggi: Debugger, Profiler, ecc.

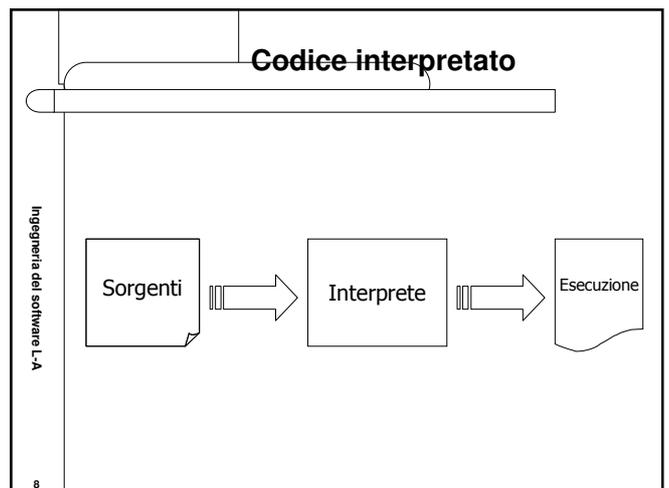
6

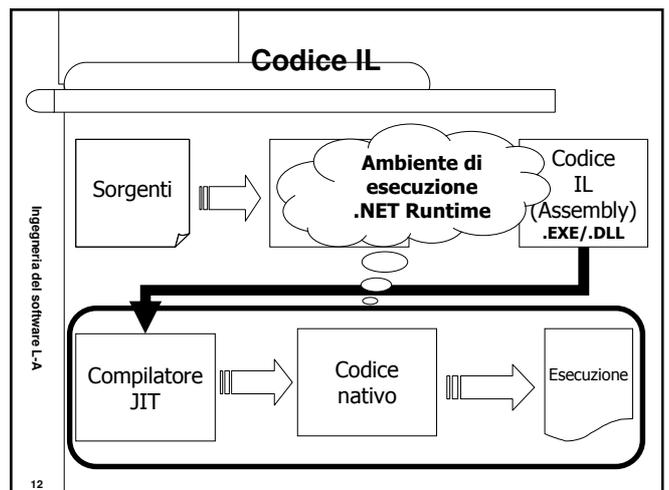
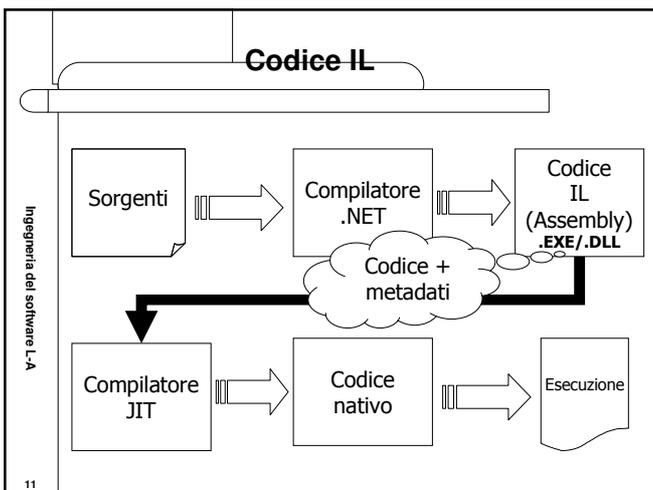
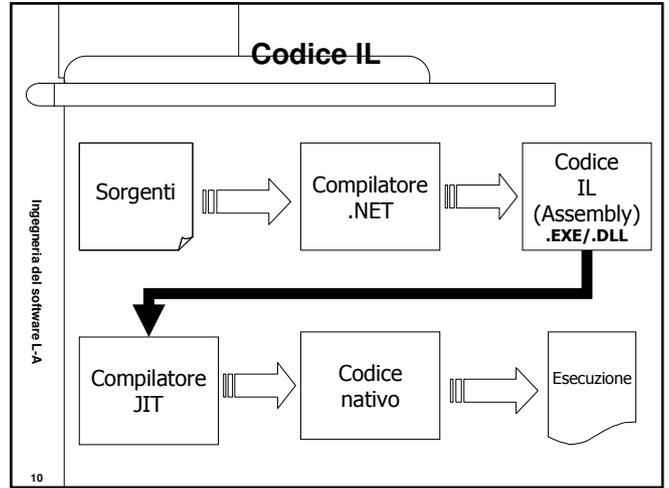
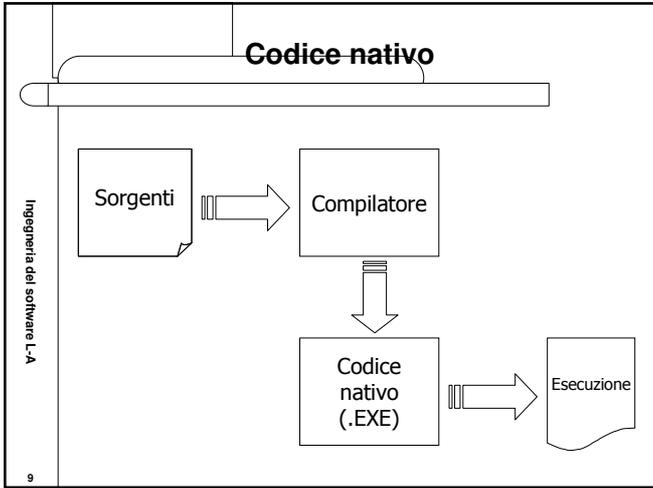
### Framework .NET

Ingegneria del software L-A

- Concetti chiave:
  - (Microsoft) **Intermediate Language** - (MS)IL
  - **Common Language Runtime** - CLR  
ambiente di esecuzione *runtime* per le applicazioni .NET  
il codice che viene eseguito sotto il suo controllo si dice **codice gestito** (*managed*)
  - **Common Type System** - CTS  
tipi di dato supportati dal *framework* .NET  
consente di fornire un modello di programmazione unificato
  - **Common Language Specification** - CLS  
regole che i linguaggi di programmazione devono seguire per essere interoperabili all'interno del *framework* .NET  
sottoinsieme di CTS

7



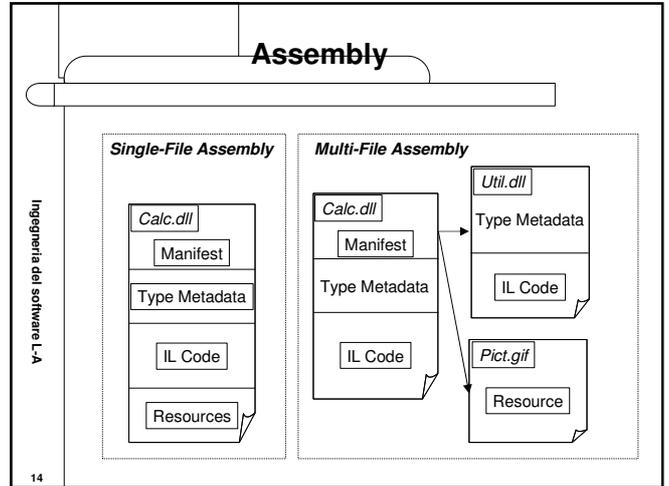


## Assembly

- Unità minima per la distribuzione, il versioning e la security
  - 1+ file
- **Manifest**
  - Metadati che descrivono l'assembly stesso
- **Type metadata**
  - Metadati che descrivono completamente tutti i tipi contenuti nell'assembly
- **Codice in Intermediate Language**
  - Ottenuto da un qualsiasi linguaggio di programmazione
- **Risorse**
  - Immagini, icone, ...

Ingegneria del software L-A

13



## Assembly

```

.assembly Hello { }
.assembly extern mscorlib { }
.method public static void main()
{
    .entrypoint
    ldstr "Hello IL World!"
    call void [mscorlib]System.Console::WriteLine
    (class System.String)
    ret
}

ilasm helloil.il
  
```

Ingegneria del software L-A

15

## Assembly

- **Assembly privati**
  - Utilizzati da un'applicazione specifica
  - *Directory* applicazione (e *sub-directory*)
- **Assembly condivisi**
  - Utilizzati da più applicazioni
  - **Global Assembly Cache (GAC)**
  - c:\windows\assembly
- **Assembly scaricati da URL**
  - *Download cache*
  - c:\windows\assembly\download
- **GACUTIL.EXE**
  - *Tool* per esaminare GAC e *download cache*

Ingegneria del software L-A

16

## Deployment semplificato

- **Installazione senza effetti collaterali**
  - Applicazioni e componenti possono essere
    - condivisi o
    - privati
- **Esecuzione *side-by-side***
  - Diverse versioni dello stesso componente possono coesistere, anche nello stesso processo

Ingegneria del software L-A

17

## Metadati

- **Descrizione dell'assembly - Manifest**
  - Identità: nome, versione, cultura [, public key]
  - Lista dei file che compongono l'assembly
  - Riferimenti ad altri assembly da cui si dipende
  - Permessi necessari per l'esecuzione
- **Descrizione dei tipi contenuti nell'assembly**
  - Nome, visibilità, classe base, interfacce
  - Campi, metodi, proprietà, eventi, ...
- **Attributi**
  - Definiti dal compilatore
  - Definiti dal framework
  - Definiti dall'utente

Ingegneria del software L-A

18

## Tool che usano i metadati

- **Compilatori**
  - Compilazione condizionale
- **Ambienti RAD**
  - Informazioni sulle proprietà dei componenti
    - Categoria
    - Descrizione
  - Editor personalizzati di tipi di proprietà
- **Analisi dei tipi e del codice**
  - Intellisense
  - ILDASM
  - Anakrino, Reflector

Ingegneria del software L-A

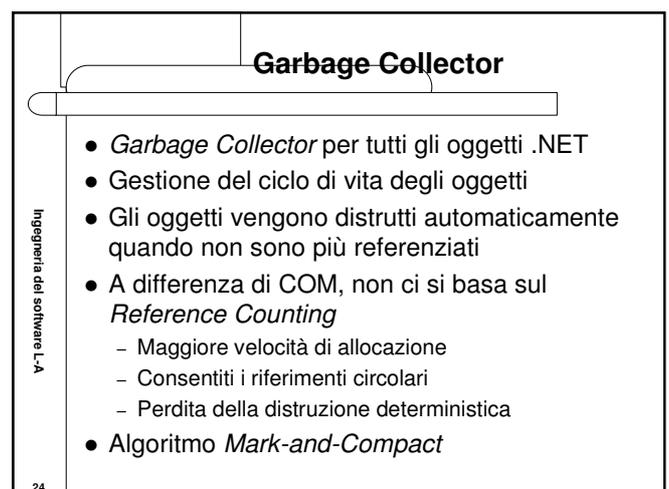
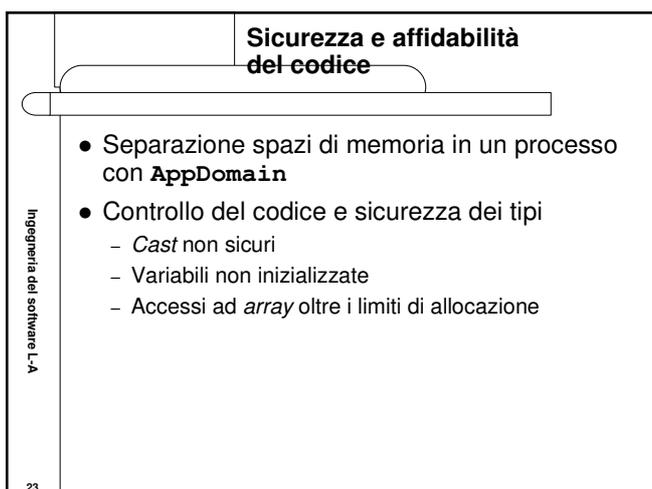
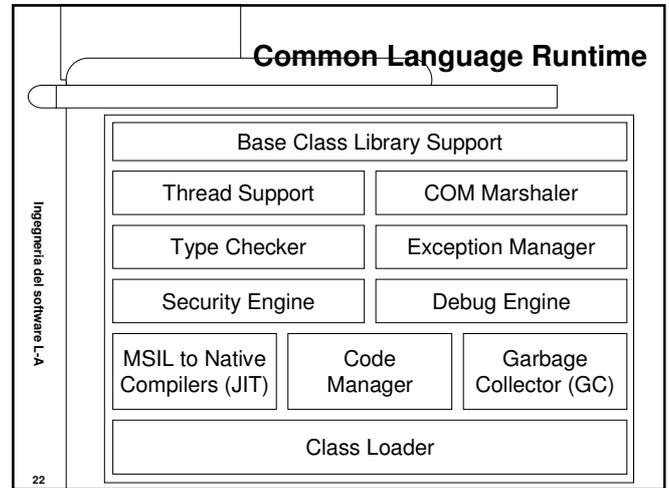
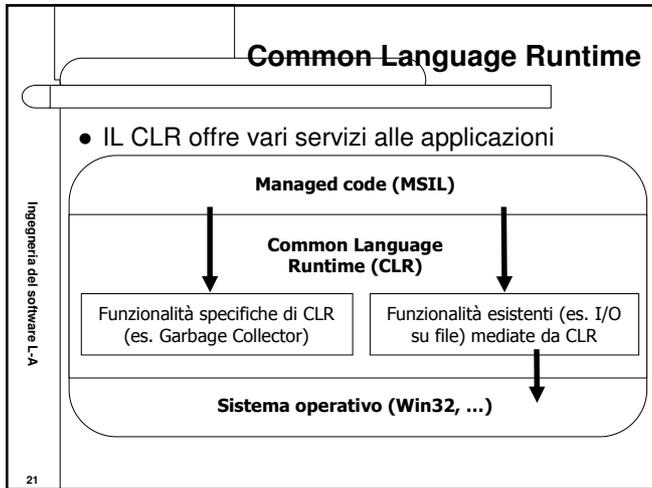
19

## Common Language Runtime

The diagram illustrates the Common Language Runtime (CLR) architecture. At the top, there are boxes for source languages: VB, C++, C#, JScript, and an ellipsis (...). These languages feed into the Common Language Specification. Below this, there are two boxes for Web Services and User Interface. These components then feed into Data and XML, which in turn feeds into the Base Class Library. Finally, the Base Class Library feeds into the Common Language Runtime.

Ingegneria del software L-A

20



## Garbage Collector e distruzione deterministica

- In alcuni casi serve un comportamento di finalizzazione deterministica
  - Riferimenti a oggetti non gestiti
  - Utilizzo di risorse che devono essere rilasciate appena termina il loro utilizzo
- Non è possibile utilizzare il metodo **Finalize** (in C# il distruttore), in quanto non è richiamabile direttamente
- È necessario implementare l'interfaccia **IDisposable**

## Gestione delle eccezioni

- Un'eccezione è
  - Una condizione di errore
  - Un comportamento inaspettatoincontrato durante l'esecuzione del programma
- Un'eccezione può essere generata dal
  - Codice del programma in esecuzione
  - Ambiente di *runtime*
- In CLR, un'eccezione è un oggetto che eredita dalla classe **System.Exception**
- Gestione uniforme - elimina
  - Codici **HRESULT** di COM
  - Codici di errore Win32
  - ...

## Gestione delle eccezioni

- Concetti universali
  - Lanciare un'eccezione (**throw**)
  - Catturare un'eccezione (**catch**)
  - Eseguire codice di uscita da un blocco controllato (**finally**)
- Disponibile in tutti i linguaggi .NET con sintassi diverse

## Altri servizi del CLR

- **Reflection**
  - Analisi dei metadati di un assembly
  - Generazione di un assembly dinamico
- **Remoting**
  - Chiamata di componenti remoti (.NET)
- **Interoperabilità** (COM, *Platform Invoke*)

## Reflection

- È possibile interrogare un assembly caricato in memoria
  - Tipi (classi, interfacce, enumeratori, etc.)
  - Membri (attributi, proprietà, metodi, etc.)
  - Parametri
- È possibile forzare il caricamento in memoria di un assembly con i metodi **Load/LoadFrom**

## Common Type System

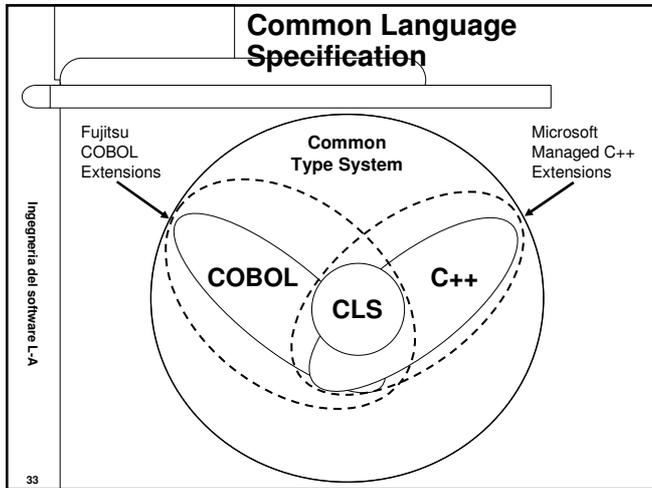
- Tipi di dato supportati dal *framework* .NET
  - Alla base di tutti i linguaggi .NET
- Consente di fornire un modello di programmazione unificato
- Progettato per linguaggi object-oriented, procedurali e funzionali
  - Esamine caratteristiche di 20 linguaggi
  - Tutte le funzionalità disponibili con IL
  - Ogni linguaggio utilizza alcune caratteristiche
- Common Language Specification
  - Regole di compatibilità tra linguaggi
  - Sottoinsieme di CTS

## Common Type System

- Alla base di tutto ci sono i tipi: **classi, strutture, interfacce, enumerativi, delegati**
- Fortemente tipizzato (*compile-time*)
- *Object-oriented*
  - Campi, metodi, tipi nidificati, proprietà, ...
- *Overload* di funzioni (*compile-time*)
- Invocazione metodi virtuali risolta a *run-time*
- Ereditarietà singola di implementazione
- Ereditarietà multipla di interfacce
- Gestione strutturata delle eccezioni

## Common Language Specification

- Regole per gli identificatori
  - Unicode, *case-sensitivity*
  - *Keyword*
- Regole di *overload* più restrittive
- Nessun puntatore *unmanaged*
- Devono essere supportate interfacce multiple con metodi dello stesso nome
- Regole per costruttori degli oggetti
- Regole per denominazione proprietà ed eventi



### Common Type System Tipi nativi

| CTS            | C#      |
|----------------|---------|
| System.Object  | object  |
| System.String  | string  |
| System.Boolean | bool    |
| System.Char    | char    |
| System.Single  | float   |
| System.Double  | double  |
| System.Decimal | decimal |
| System.SByte   | sbyte   |
| System.Byte    | byte    |
| System.Int16   | short   |
| System.UInt16  | ushort  |
| System.Int32   | int     |
| System.UInt32  | uint    |
| System.Int64   | long    |
| System.UInt64  | ulong   |

Ingegneria del software L-A

34

- ### Common Type System
- Tutto è un oggetto
    - `System.Object` è la classe radice
  - Due categorie di tipi
    - **Tipi riferimento**
      - Riferimenti a oggetti allocati sull'*heap* gestito
      - Indirizzi di memoria
    - **Tipi valore**
      - Allocati sullo *stack* o parte di altri oggetti
      - Sequenza di byte
- Ingegneria del software L-A
- 35

- ### Common Type System Tipi valore
- I tipi valore comprendono:
    - Tipi primitivi (*built-in*)
      - Int32, ...
      - Single, Double
      - Decimal
      - Boolean
      - Char
    - Tipi definiti dall'utente
      - Strutture (`struct`)
      - Enumerativi (`enum`)
- Ingegneria del software L-A
- 36

**Common Type System**  
Tipi valore vs tipi riferimento

```

public struct Size
{
    public int height;
    public int width;
}
public class CSize
{
    public int height;
    public int width;
}
public static void Main()
{
    Size v;           // v istanza di Size
    v.height = 100;  // ok
    CSize r;         // r è un reference
    r.height = 100;  // NO, r non assegnato
    r = new CSize(); // r fa riferimento a un CSize
    r.height = 100;  // ok, r inizializzata
}
    
```

37

**Common Type System**  
Tipi valore vs tipi riferimento

```

public struct Point
{
    private int _x, _y;
    public Point(int x, int y)
    {
        _x = x;
        _y = y;
    }
    public int X
    {
        get { return _x; }
        set { _x = value; }
    }
    public int Y
    {
        get { return _y; }
        set { _y = value; }
    }
}
    
```

38

**Common Type System**  
Tipi valore vs tipi riferimento

```

public class Rectangle
{
    Point v1;
    Point v2;
    ...
}
...
Rectangle r = new Rectangle();
    
```

39

**Common Type System**  
Tipi valore vs tipi riferimento

```

...
Point[] points = new Point[100];
for (int i = 0; i < 100; i++)
    points[i] = new Point(i, i);
...
    
```

- Alla fine, rimane 1 solo oggetto nell'heap (l'array di Point)

```

...
Point[] points = new Point[100];
for (int i = 0; i < 100; i++)
{
    points[i].X = i;
    points[i].Y = i;
}
...
    
```

40

## Common Type System Tipi valore vs tipi riferimento

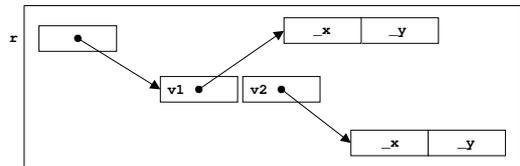
```
public class Point
{
    private int _x, _y;
    public Point(int x, int y)
    {
        _x = x;
        _y = y;
    }
    public int X
    {
        get { return _x; }
        set { _x = value; }
    }
    public int Y
    {
        get { return _y; }
        set { _y = value; }
    }
}
```

Ingegneria del software L-A

41

## Common Type System Tipi valore vs tipi riferimento

```
public class Rectangle
{
    Point v1;
    Point v2;
    ...
}
...
Rectangle r = new Rectangle();
```



Ingegneria del software L-A

42

## Common Type System Tipi valore vs tipi riferimento

```
...
Point[] points = new Point[100];
for (int i = 0; i < 100; i++)
    points[i] = new Point(i, i);
...
```

- Alla fine, rimangono 101 oggetti nell'*heap* (1 array di Point + 100 Point)

```
...
Point[] points = new Point[100];
for (int i = 0; i < 100; i++)
{
    points[i].X = i;
    points[i].Y = i;
}
...

```

**NO!**

Ingegneria del software L-A

43

## Boxing / Unboxing

- Un qualsiasi tipo valore può essere automaticamente convertito in un tipo riferimento (**boxing**) mediante un *up cast* implicito a `System.Object`

```
int i = 123;
object o = i;
```

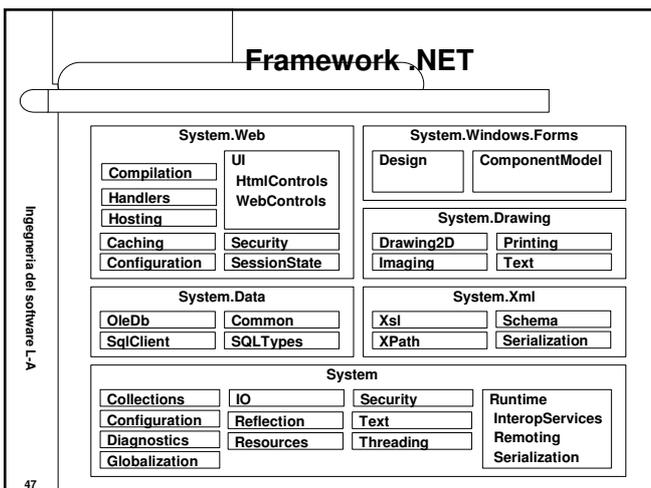
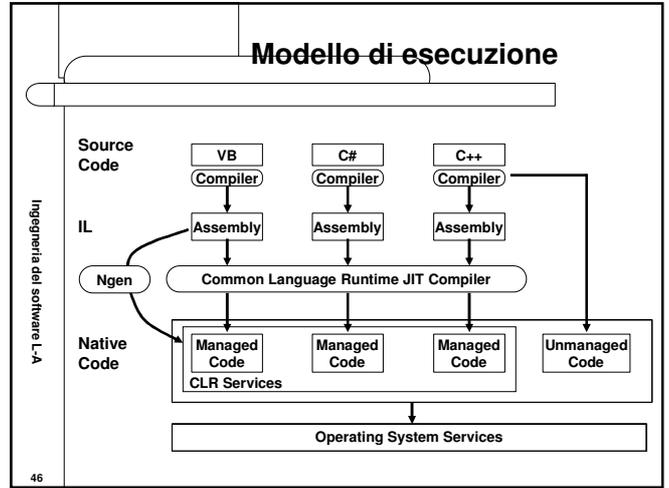
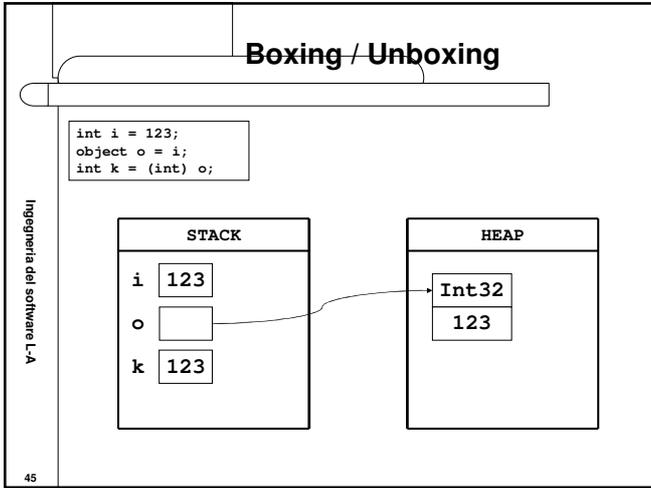
- Un tipo valore "*boxed*" può tornare ad essere un tipo valore standard (**unboxing**) mediante un *down cast* esplicito

```
int k = (int) o;
```

- Un tipo valore "*boxed*" è un **clone indipendente**

Ingegneria del software L-A

44



- ### Framework .NET
- Il *framework* non è una "scatola nera"
  - E' possibile estendere una qualsiasi classe .NET (non *sealed*) mediante ereditarietà Diversamente da COM,
    - si usa e si estende la **classe stessa**
    - non si utilizza uno strato intermedio (*wrapper*)
  - L'ereditarietà è *cross-language*
- Ingegneria del software L-A
- 48

## Bibliografia

Libri di base:

- *D. S. Platt, Introducing Microsoft® .NET*, Second Edition (\*)
- *J. Sharp, J. Jagger, Microsoft® Visual C#™ .NET Step by Step* (\*)
- *T. Archer, A. Whitechapel, Inside C#*, Second Edition (\*)
- *M. J. Young, XML Step by Step*, Second Edition (\*)
- *R. M. Riordan, Microsoft® ADO.NET Step by Step* (\*)

Libri avanzati:

- *J. Richter, Applied Microsoft® .NET Framework Programming*
- *C. Petzold, Programming Microsoft® Windows® with C#* (\*)
- *S. Lidin, Inside Microsoft® .NET IL Assembler*

(\*) Disponibile nella biblioteca del DEIS