# Programmazione Orientata agli Oggetti in Linguaggio Java

## Test e Correzione:

## C#

## Parte a

versione 1.5



**G. Mecca – Università della Basilicata – mecca@unibas.it**



## Sommario

- Introduzione
- Test di Regressione con NUnit
- Debugging
- Logging
- Costruttore Statico

# Introduzione

- Anche in questo caso
  - ⇨i principi metodologici visti per Java valgono identicamente per C#
- Inoltre
  - ⇨come al solito le due piattaforme presentano molti aspetti simili
  - ⇨la gestione dei test in C# è semplificata





# Test di Regressione

- NUnit
  - ⇨un framework per test di regressione in.NET
  - ⇨nasce come trasformazione di JUnit
  - ⇨con alcune forme di personalizzazione basate sulle funzionalità di C#
- Le principali differenze
  - ⇨NUnit è fortemente basato su attributi di C#
  - ⇨NUnit è orientato agli assembly

# Test di Regressione

- Utilizzo degli attributi
  - ⇨una classe di test deve essere annotata con l'attributo [TestFixture] – non deve necessariamente estendere TestCase
  - ⇨un metodo di test deve essere annotata con l'attributo [Test] – non deve necessariamente chiamarsi TestXXX
  - ⇨un metodo di setup deve essere annotato con l'attributo [SetUp] – non deve necessariamente chiamarsi SetUp





```
namespace Unibas.Indovina {
using NUnit.Framework;

[TestFixture]
public class TestPartita {

   private Partita partita;

   [SetUp]
   protected void SetUp() {
      partita = new Partita();
   }

   [Test]
   public void TestGestisciTentativoBasso() {
      int tentativi = partita.GetNumeroDiTentativi();
      int numeroDaIndovinare = partita.GetNumeroDaIndovinare();
      if (numeroDaIndovinare > 1) {
         partita.GestisciTentativo(numeroDaIndovinare - 1);
         string suggerimento = partita.GetSuggerimento();
         Assert.IsTrue(suggerimento.Equals("Prova con un numero piu' alto"), "suggerimento");
         Assert.IsTrue(partita.GetTrovato() == false, "trovato");
         Assert.IsTrue(partita.GetNumeroDiTentativi() == tentativi + 1, "tentativi");
      }
   }
```

il namespace del framework è NUnit.Framework

per le asserzioni è possibile utilizzare il metodo Assert.IsTrue
ATTENZIONE: prima l'asserzione e poi la stringa

# Test di Regressione

- L'installazione di NUnit
  - ⇨il pacchetto viene distribuito sotto forma di installabile dal sito http://www.nunit.org
  - ⇨per completare l'installazione basta eseguire l'installabile
- I test runner
  - ⇨C:\Programmi\NUnit V2.2\bin\nunit-console.exe
  - ⇨C:\Programmi\NUnit V2.2\bin\nunit-gui.exe





# Test di Regressione

>> TestRecord.cs
>> TestPartita.cs
>> nunit-gui.exe

- NUnit è orientato agli assembly
  - ⇨JUnit è orientato alle classi (unità di compilazione per Java)
  - ⇨viceversa NUnit è orientato agli assembly (unità di compilazione di .NET)
  - ⇨grazie agli attributi, NUnit è in grado di selezionare, tra le classi di un assembly, quelle che rappresentano classi di test
  - ⇨quindi non è necessario creare suite di test

# Test di Regressione

- Due strategie tipiche per i test
- Un unico assembly per codice e test
  - ⇨test e codice vengono compilati assieme
  - ⇨e il test runner viene eseguito sull'assembly
  - ⇨è necessario compilare l'assembly con riferimento all'assembly del framework
  - ⇨C:\Programmi\NUnit V2.2\bin\nunit-framework.dll

Esempio: csc /out:Indovina.exe /r:Unibas.Utilita.dll;"c:\Programmi\NUnit V2.2\bin\nunit.framework.dll" *.cs





# Test di Regressione

- Assembly separato per i test
  - ⇨nel caso in cui non si voglia distribuire il codice dei test con quello dell'applicazione
  - ⇨in questo caso si compila il codice
  - ⇨e poi si compilano i test con un riferimento all'assembly del codice

csc /out:Indovina.exe /r:Unibas.Utilita.dll Principale.cs Partita.cs Record.cs

csc /out:TestIndovina.dll /t:library /r:"c:\Programmi\NUnit V2.1\bin\nunit.framework.dll";Indovina.exe TestPartita.cs TestRecord.cs

# Test di Regressione

- Nota
  - una differenza tra NUnit e JUnit
  - se le asserzioni falliscono, il framework lancia nunit.framework.AssertionException, un'eccezione ordinaria (come tutte in C#)
- Invece, in JUnit
  - junit.framework.AssertionFailedError è un'eccezione di tipo error, chiaramente distinguibile da quelle applicative





# Test di Regressione

- Test e asserzioni
  - tipicamente in C# NON vengono scritti test sulle guardie
  - infatti, la semantica di Debug.Assert() prevede che, in caso l'asserzione fallisca, NON sia sollevata un'eccezione ma semplicemente visualizzata la finestra di dialogo
  - questo non consente di scrivere facilmente test sulla guardia

# Debugging

- Il debugger del framework
  - ⇨dbgCLR.exe, debugger grafico
  - ⇨disponibile nella cartella C:\Programmi\Microsoft.NET\SDK\v1.1\GuiDebug
- Per eseguire il debugger su un assembly
  - ⇨è necessario compilare l'assembly con le opzioni di debug: /optimize- /debug+

Esempio: csc /out:Indovina.exe /r:Unibas.Utilita.dll /optimize- /debug+ Principale.cs Partita.cs Record.cs





# Debugging

- L'opzione /debug
  - ⇨valore standard: /debug-
  - ⇨esclude tutte le informazioni di debug
- Per abilitare le informazioni di debug
  - ⇨/debug oppure /debug+ oppure /debug:full
  - ⇨viene creato un file con estensione .pdb utilizzabile dal debugger
  - ⇨si dice in questo caso che l'assembly è un assembly di debug

# Debugging

- Attenzione alle opzioni di debug
  - a differenza di Java (in cui i numeri di linea e il riferimento al file sorgente sono inclusi di default), in C# l'opzione standard è /debug-
  - di conseguenza, per esempio, non è possibile avere informazione sui numeri di linea quando un test fallisce
  - è opportuno, durante l'esecuzione dei test, specificare sempre /debug in fase di compil.





# Debugging

- Per usare il debugger
  - caricare un assembly eseguibile attraverso il comando Debug >> Programma per Debug
  - caricare il codice sorgente con il comando File >> Apri
  - a questo punto è possibile fissare punti di interruzione
  - ed avviare l'esecuzione con Debug >> Avvia

# Logging

- Come alternativa al debugger
  - ⇨anche in .NET è possibile utilizzare un sistema di logging
- Il sistema di logging di riferimento
  - ⇨Log4Net, versione .NET di Log4J
- Altrimenti
  - ⇨esiste anche un sistema di logging integrato nelle API, ma è molto povero





# Logging

- Il sistema di logging standard
  - ⇨basato su System.Diagnostics.Debug
  - ⇨e su System.Diagnostics.Trace
  - ⇨due classi con comportamento molto simile
  - ⇨entrambe forniscono un metodo statico Write e uno WriteLine
  - ⇨l'esecuzione di entrambi i metodi può essere attivata o disattivata utilizzando un parametro di compilazione

# Logging

- Nel caso dela classe Debug
  - ⇨compilando gli assembly con /d:debug, il codice delle chiamate al metodo Debug.WriteLine viene incluso nell'assembly
  - ⇨senza l'opzione, il codice delle chiamate al metodo Debug.WriteLine viene completamente escluso dall'assembly
- Analogamente per Trace
  - ⇨opzione /d:trace (abilitata da Visual Studio)





# Logging

- A ciascuna delle classi
  - ⇨è possibile associare un handler
- Di conseguenza, rispetto a Java
  - ⇨è possibile abilitare e disabilitare le istruzioni di logging utilizzando l'opzione del compilat.
  - ⇨è possibile effettuare il logging su dispositivi diversi
  - ⇨ma esiste un unico livello di logging

# Logging

- Per abilitare il debugging
  - ⇨è necessario specificare un handler per la classe Debug
  - ⇨tipicamente un handler per la console
- Esempio
  - ⇨Debug.Listeners.Add(new TextWriterTraceListener(Console.Out));
  - ⇨Debug.AutoFlush = true;
  - ⇨Debug.Indent();





# Logging

- La classe Unibas.Utilita.Logger
  - ⇨semplificata rispetto al Logger per Java
  - ⇨inizializza la classe Debug
  - ⇨fornisce due soli metodi statici
  - ⇨Logger.Log(string messaggio)
  - ⇨Logger.Log(string classe, string metodo, string messaggio)

# Logging

- Attenzione
  - ⇨l'opzione /d:debug deve essere specificata in sede di compilazione dell'assembly a cui appartiene Logger.cs
  - ⇨se le classi dell'applicazione sono in un assembly separato, non conta l'opzione relativa a quest'ultimo assembly
  - ⇨di conseguenza o si include Logger.cs nello stesso assembly, o bisogna ricompilare Unibas.Utilita.dll





>> indovinaIlNumero

```
namespace Unibas.Indovina {
using Unibas.Utilita;
public class Partita {
   public Partita() {
      this.numeroDaIndovinare = System.Math.Abs(Partita.generatore.Next(100) + 1);
      Logger.Log("Numero da indovinare: " + numeroDaIndovinare);
      ...
   }
  public void GestisciTentativo(int tentativo) {
      ...
      Logger.Log("Partita", "GestisciTentativo", "Tentativo: " + tentativo);
      this.numeroDiTentativi++;
      Logger.Log("Partita", "GestisciTentativo", "Num. tentativi: " + numeroDiTentativi);
      if (tentativo == this.numeroDaIndovinare) {
         this.trovato = true;
         this.suggerimento = "Numero indovinato";
      } else if (tentativo < numeroDaIndovinare) {
         this.suggerimento = "Prova con un numero piu' alto";
      } else if (tentativo > numeroDaIndovinare) {
         this.suggerimento = "Prova con un numero piu' basso";
      }
      Logger.Log("Partita", "GestisciTentativo", "Suggerimento: " + suggerimento);
   }
```

# Costruttore Statico

- La classe Logger
  - ⇨un aspetto interessante: il blocco di inizializzazione statico
  - ⇨nella terminologia .NET: costruttore statico
  - ⇨public static Logger() {...}
  - ⇨viene eseguito all'atto del caricamento della classe da parte della macchina virtuale e serve ad inizializzare il componente statico





```
namespace Unibas.Utilita {
///  <summary>Questa classe consente di effettuare semplici operazioni di logging...
public class Logger {

      static Logger() {
         System.Diagnostics.TextWriterTraceListener listener =
                  new System.Diagnostics.TextWriterTraceListener(System.Console.Out);
         System.Diagnostics.Debug.Listeners.Add(listener);
         System.Diagnostics.Debug.AutoFlush = true;
         System.Diagnostics.Debug.Indent();
      }

      public static void Log (string messaggio) {
         System.Diagnostics.Debug.WriteLine("**** LOG **** " + messaggio);
      }

      public static void Log (string classe, string metodo, string messaggio) {
         System.Diagnostics.Debug.WriteLine("**** LOG **** " + "Classe: " + classe
                                          + " - Metodo: " + metodo + ": \n" + messaggio);
      }
   }
}
```

# Riassumendo

- Introduzione
- Test di Regressione con NUnit
- Debugging
- Logging
- Costruttore Statico





# Test e Asserzioni

- Comportamento delle asserzioni in C#
  - ⇨il comportamento standard in caso di fallimento dell'eccezione non prevede che siano sollevate eccezioni
  - ⇨ma visualizza una finestra di dialogo
- Rispetto ai test
  - ⇨entrambe le caratteristiche sono negative
  - ⇨la prima non consente di eseguire test sull'asserzione, la seconda richiede interaz.

# Test e Asserzioni

- Funzionamento di Debug.Assert()
  - ⇨viene in realtà utilizzato il sistema di logging
  - ⇨se l'asserzione fallisce, la classe Debug delega il fallimento ad un handler, che è un oggetto di tipo TraceListener
  - ⇨appartenente alla collezione pubblica Listeners
  - ⇨eseguendo il metodo Fail() dell'handler





# Test e Asserzioni

>> Unibas.Utilita.MioTraceListener

- Per risolvere il problema (complesso)
  - ⇨è possibile disabilitare la finestra di dialogo cambiando il comportamento del trace listener predefinito
  - ⇨e aggiungere agli handler un ulteriore TraceListener personalizzato che lancia l'eccezione
- Un esempio
  - ⇨Unibas.Utilita.MioTraceListener

# Test e Asserzioni

>> morraCinese – TestParita

- Per cambiare il comportamento di Assert
  - ⇨è necessario specificare le operazioni nel costruttore del test
  - ⇨modificare il trace listener predefinito per impedire che venga visualizzata la finestra di dialogo
  - ⇨aggiungere alla collezione Listener di Debug il nuovo trace listener





# Termini della Licenza