Programmazione Orientata agli Oggetti in Linguaggio Java

Tecniche di Programmazione: C#

versione 1.1

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

Sommario

- Le API di .NET
- Date
- Collezioni
- Ordinamento
- Clonazione e Serializzazione
- Classi Nidificate
- Delegati
- Thread

Le API di .NET

- Nel seguito
  - ⇨alcuni aspetti delle applicazioni di riferimento
  - ⇨simili a quelli già discussi per Java
- La buona notizia
  - ⇨le API di .NET sono più recenti di quelle di Java
  - ⇨è stato fatto uno sforzo per semplificare alcuni aspetti della programmazione

Date

- Un esempio
  - ⇨la gestione delle Date
  - ⇨tutte le operazioni principali si possono fare con la classe System.DateTime
- In effetti, però...
  - ⇨il meccanismo è identico a quello di Java
  - ⇨rappresentazione astratta (millisecondi) + internazionalizzazione
  - ⇨calendari e formattazione

Date

- System.DateTime
  - ⇨si tratta in effetti di una struttura e non di una classe
- Differenza tra classi e strutture
  - ⇨la principale è che le istanze delle strutture sono allocate nello stack e non nello heap
  - ⇨non si manipolano attraverso i riferimenti ma come variabili dei tipi primitivi
  - ⇨estendono System.ValueObject

Date

- In effetti
  - ⇨dal punto di vista dell'utilizzo la differenza non si nota
- I metodi di DateTime
  - ⇨permettono di fare tutto quello che in Java è possibile fare con Date + GregorianCalendar + DateFormat
  - ⇨creare date sulla base del calendario, fare i conti, formattare diversamente le date

Date

>> System.DateTime

- In particolare
  - ⇨esistono diversi costruttori
  - ⇨esistono proprietà per ottenere giorno, mese, anno, ore, minuti e secondi
  - ⇨nota: i mesi vanno da 1 a 12
  - ⇨esistono varianti del metodo ToString() che accettano un parametro che indica il formato per stampare la data
  - ⇨esistono metodi per i conti (Add(), Subtract())

```
namespace Unibas.Appuntamenti.Modello {

public class Giorno : System.IComparable {

   private System.DateTime dateTime;
   private System.Collections.ArrayList listaImpegni = new System.Collections.ArrayList();

   public Giorno(int gg, int mese, int anno) {
      if (gg < 0 || gg > 31) { throw new System.ArgumentException("Giorno scorretto: " + gg); }
      if (mese < 1 || mese > 12) { throw new System.ArgumentException("Mese scorretto: " + mese); }
      if (anno < 0) { throw new System.ArgumentException("Anno scorretto: " + anno); }
      this.dateTime = new System.DateTime(anno, mese, gg);
   }
   public int NumGiorno {
     get { return this.dateTime.Day; }
   }
   public int Mese {
     get { return this.dateTime.Month; }
   }
   public int Anno {
     get { return this.dateTime.Year; }
   }
   public string ToShortString() {
      return this.dateTime.ToString("dd/MM/yy");
   }
   ...
```

```
namespace Unibas.Appuntamenti.Modello {

public class Orario : System.IComparable {

   private System.DateTime dateTime;

   public Orario(int ora, int minuti) {
      if (ora < 0 || ora > 23) {  throw new System.ArgumentException("Ora scorretta: " + ora); }
      if (minuti < 0 || minuti > 59) { throw new System.ArgumentException("Minuti scorretti: " + minuti);
     }
      this.dateTime = new System.DateTime(1970, 1, 1, ora, minuti, 0);
   }

   public int Ora {
      get { return this.dateTime.Hour; }
   }

   public int Minuti {
      get { return this.dateTime.Minute; }
   }

   public int CompareTo(System.Object orario) {
      return System.DateTime.Compare(this.dateTime, ((Orario)orario).dateTime);
   }
   ...
}
}
```


Collezioni

- Le collezioni di .NET
  - ⇨System.Collections.ArrayList implementa l'interfaccia System.Collections.IList
  - ⇨System.Collections.Hashtable
- Hashtable
  - ⇨implementazione di un dizionario associativo basata su hash

Collezioni

- Hashing
  - ⇨vengono utilizzati i metodi GetHashCode() ed Equals() ereditati da Object
- I metodi di Hashtable
  - ⇨Add(Object chiave, Object valore)
  - ⇨Count
  - ⇨indicizzatore per l'accesso ai valori data la chiave: <table>[<chiave>]

```
namespace Unibas.Appuntamenti.Modello {

public class Agenda {

   private string utente;
   private System.Collections.Hashtable giorni = new System.Collections.Hashtable();

   public string Utente {
      get { return this.utente; }
      set { this.utente = value; }
   }

   public void AddGiorno(Giorno giorno) {
       this.giorni.Add(giorno.ToShortString(), giorno);
   }
   public void RemoveGiorno(string data) {
       this.giorni.Remove(data);
   }
   public Giorno GetGiorno(string data) {
       return (Giorno)(this.giorni[data]);
   }
   public int NumeroGiorni {
      get { return this.giorni.Count; }
   }
   ...
```

Collezioni

- Nota
  - ⇨il namespace System.Collections è per molti versi diverso da java.util
  - ⇨implementa tipi di collezioni diversi
  - ⇨non esiste un'implementazione analoga a Set o a LinkedList
  - ⇨esistono implementazioni mancanti in Java: es: SortedList
  - ⇨la radice della gerarchia è ICollection

Collezioni

- Iteratori
  - ⇨anche in .NET esiste il concetto di iteratore per le collezioni, chiamati enumeratori
  - ⇨System.Collections.IEnumerator
- Due membri principali
  - ⇨boolean MoveNext()
  - ⇨Object Current

Collezioni

- Esempio: la stampa degli impegni

```
public string StringaImpegni() {
   OrdinaImpegni();
   System.Text.StringBuilder stringaImpegni =
                              new System.Text.StringBuilder();
   System.Collections.IEnumerator enumerator =
                              listaImpegni.GetEnumerator();
   while (enumerator.MoveNext()) {
       stringaImpegni.Append(enumerator.Current.ToString());
       stringaImpegni.Append("\n");
   }
   return stringaImpegni.ToString();
}
```


Collezioni

- Ma...
  - ⇨in C# c'è un modo molto più semplice per scrivere il ciclo
- Ciclo foreach
  - ⇨è una forma di ciclo for migliorato
  - ⇨scandisce tutti gli elementi di una collezione utilizzando (in modo nascosto) un iteratore adeguato alla collezione
  - ⇨sintassi: foreach (<*tipo*> <*rifer*> in <*collez*>)

Collezioni

- Esempio: stampa impegni con foreach

```
public string StringaImpegni() {
    OrdinaImpegni();
    System.Text.StringBuilder stringaImpegni =
                        new System.Text.StringBuilder();
    foreach (IImpegno impegnoIesimo in this.listaImpegni) {
        stringaImpegni.Append(impegnoIesimo.ToString());
        stringaImpegni.Append("\n");
    }
    return stringaImpegni.ToString();
}
```


Collezioni

- Esempio: passo di simulazione

```
public void Simula() {
    System.Collections.IList listaAnimali = this.scacchiera.ListaAnimali;
    foreach (Animale animale in listaAnimali) {
        if (animale != null) {
            animale.Agisci(this.scacchiera);
        }
    }
}
```

Ordinamento

```
// in Giorno
public void OrdinaImpegni() {
      this.listaImpegni.Sort();
}
```

- Analogo a Java
  - ⇨per usare i metodi forniti dalla piattaforma è necessario implementare System.IComparable
  - ⇨un unico metodo: int CompareTo(Object o)
  - ⇨con la stessa semantica dell'analogo Java
- Nota
  - ⇨tutte le collezioni di .NET hanno un metodo Sort() per ordinarle

Ordinamento

```
// in Agenda
using System.Collections;
public override string ToString() {
      System.Text.StringBuilder stringa = new System.Text.StringBuilder();
      stringa.Append("Utente: " + this.utente + "\n");
      System.Collections.ICollection collGiorni = this.giorni.Values;
      System.Collections.ArrayList listaGiorni = Ordina(collGiorni);
      foreach (Giorno giornoIesimo in listaGiorni) {
          stringa.Append(giornoIesimo);
      }
      return stringa.ToString();
}

private ArrayList Ordina(System.Collections.ICollection coll) {
    ArrayList lista =  new System.Collections.ArrayList(coll);
    lista.Sort();
    return lista;
}
```

Clonazione e Serializzazione

- Il processo di clonazione
  - ⇨è più razionale che in Java
- Oggetti clonabili
  - ⇨devono implementare System.ICloneable
  - ⇨che prevede che sia definito il metodo Object Clone()
  - ⇨quindi ICloneable NON è un'interfaccia vuota

Clonazione e Serializzazione

- Il supporto alla clonazione superficiale
  - ⇨fornito dal metodo MemberWiseClone() ereditato da Object
  - ⇨metodo protetto
- Quindi, tipicamente
  - ⇨nella sottoclasse implemento ICloneable
  - ⇨definisco public Object Clone()
  - ⇨utilizzo super.MemberWiseClone() per ottenere una copia superficiale

Clonazione e Serializzazione

- Serializzazione
  - .NET supporta vari tipi di serializzazione
- Serializzazione binaria
  - namespace System.Runtime.Serializazion
- Serializzazione XML
  - namespace System.Xml.Serialization

Classi Nidificate

- Le classi interne
  - uno strumento decisamente meno importante ed evoluto in C# rispetto a Java
  - considerato probabilmente troppo complesso per il programmatore
- In particolare
  - esistono solo classi nidificate (statiche)
  - non è necessario specificare il qualificatore static

Classi Nidificate

- Una virtuale LinkedList per C#
  - ⇨non esiste una implementazione di LinkedList nel namespace System.Collections
  - ⇨di conseguenza il codice successivo è puramente riportato a titolo di esempio
  - ⇨esistono però varie implementazioni disponibili in rete


```
namespace System.Collections {

public class LinkedList : IList {

   // classe interna statica Entry
   private class Entry {
      Object element;
      Entry next;
      Entry previous;

      Entry(Object element, Entry next, Entry previous) {
         this.element = element;
         this.next = next;
         this.previous = previous;
      }
   }

   // proprietà
   private Entry header = new Entry(null, null, null);
   private int size = 0;

   // costruttori
   public LinkedList() {
      header.next = header.previous = header;
   }
   ...
```

non è necessario specificare static perchè tutte le classi interne sono automaticamente statiche

Classi Nidificate

- Una conseguenza
  - ⇨dal momento che in C# non esistono classi interne, non esistono classi anonime
  - ⇨di conseguenza non è possibile utilizzarle al modo di Java, per “passare metodi come parametri”
- La soluzione di C#
  - ⇨uno strumento dedicato del linguaggio: i delegati

Delegati

- Delegato
  - ⇨oggetto utilizzato per eseguire metodi diversi
  - ⇨il delegato implementa un’interfaccia fatta di un unico metodo
  - ⇨alla creazione dell’oggetto è necessario specificare un’implementazione per il metodo
  - ⇨ovvero “passare un metodo concreto come argomento per il metodo astratto (parametro)”

Delegati

- Il termine delegato
  - ⇨richiama la funzione dell'oggetto
  - ⇨ovvero il fatto che l'oggetto NON ha propri metodi da eseguire
  - ⇨di volta in volta gli viene associato un metodo concreto
  - ⇨e l'oggetto delega le invocazioni che riceve al metodo specificato

Delegati

- I passi per utilizzare un delegato
  - ⇨sono vari
- I passo: definizione dell'interfaccia
  - ⇨descrive l'interfaccia del delegato, ovvero il prototipo del metodo che la compone
- II passo: creazione del delegato
  - ⇨specifica una implementazione per il metodo previsto dal delegato
- III passo: chiamata

Delegati

>> Calcolatrice.Delegati

- Esempio
  - ⇨la calcolatrice con i delegati
  - ⇨definisco un delegato Operazione
  - ⇨definisco in Principale i tre metodi (somma(), sottrai(), moltiplica())
  - ⇨di volta in volta creo un oggetto delegato di tipo Operazione associandogli uno dei tre metodi
  - ⇨poi eseguo il delegato

Delegati

- La definizione del delegato
  - ⇨viene riportata al livello di una dichiarazione di classe o di interfaccia

```
public delegate double Operazione(double a, double b);
```

definisce un nuovo tipo di delegato chiamato Operazione
gli oggetti di questo tipo hanno un'interfaccia fatta
da un unico metodo (anonimo)
che riceve due argomenti di tipo double e restituisce
un risultato di tipo double

```
namespace Calcolatrice.Delegati {

public delegate double Operazione(double a, double b);

public class Principale {

   private int SchermoMenu() {
      System.Console.WriteLine("--------------------------------");
      System.Console.WriteLine("        Calcolatrice           ");
      System.Console.WriteLine("--------------------------------");
      System.Console.WriteLine("   1. Esegui somma");
      System.Console.WriteLine("   2. Esegui differenza");
      System.Console.WriteLine("   3. Esegui moltiplicazione");
      System.Console.WriteLine("   0. Esci");
      System.Console.Write("   Scelta ----> ");
      int scelta = Unibas.Utilita.Console.LeggiIntero();
      while ((scelta < 0) || (scelta > 3)) {
         System.Console.WriteLine("   Errore. Ripeti ----> ");
         scelta = Unibas.Utilita.Console.LeggiIntero();
      }
      return scelta;
   }

   public static void Main(string[] args) {
      new Principale().EseguiOperazioni();
   }
   ...
```



```
   public double Somma(double a, double b) { return a + b; }

   public double Sottrai(double a, double b) { return a - b; }

   public double Moltiplica(double a, double b) { return a * b; }

   public void EseguiOperazioni() {
      Operazione operazione = null;
      bool continua = true;
      while (continua) {
         double a, b;  int scelta = SchermoMenu();
         if (scelta == 0) {
            continua = false;
         } else {
            System.Console.Write("Primo argomento:   --> "); a = Unibas.Utilita.Console.LeggiDouble();
            System.Console.Write("Secondo argomento: --> "); b = Unibas.Utilita.Console.LeggiDouble();
            if (scelta == 1) {
               operazione = new Operazione(this.Somma);
            } else if (scelta == 2) {
               operazione = new Operazione(this.Sottrai);
            } else if (scelta == 3) {
               operazione = new Operazione(this.Moltiplica);
            }
            System.Console.WriteLine("\n   Risultato: " + operazione(a, b) + "\n");
         }
      }
   }

} // fine Principale
```

creazione del delegato;
viene specificato come argomento
del costruttore il riferimento
al metodo concreto

Delegati

- Sintassi della dichiarazione
  - ⇨delegate <tipo> <nome> ([<parametri>]);
- Esempio
  - ⇨ public delegate double Operazione(double a, double b);
- Semantica della dichiarazione
  - ⇨definisce un nuovo tipo di delegato
  - ⇨il cui metodo anonimo ha il tipo di ritorno e i parametri specificati

Delegati

- Sintassi della creazione
  - ⇨<rif> = new <NomeDelegato>(<metodo>);
- dove
  - ⇨<rif> è un riferimento
  - ⇨<NomeDelegato> è un tipo di delegato
  - ⇨<metodo> è il nome del metodo da associare al metodo anonimo; forma <rif>.<nome>
  - ⇨deve avere lo stesso prototipo del delegato
  - ⇨es: operazione = new Operazione(this.somma);

Delegati

- Anatomia della creazione
  - ⇨operazione = new Operazione(this.Somma);
- this.Somma
  - ⇨una strana sintassi
  - ⇨viene interpretato come un riferimento all'oggetto System.Reflection.MethodInfo che descrive il metodo Somma dell'oggetto Principale
  - ⇨è una sintassi utilizzabile solo in questo caso

Delegati

- Sintassi della chiamata
  - ⇨<rif>([<argomenti>]);
  - ⇨es: operazione(a, b);
- Semantica
  - ⇨viene eseguito il metodo specificato alla creazione dell'oggetto delegato sugli argomenti specificati

Delegati

- Una categoria particolare di delegati
  - ⇨i delegati “multicast”
- Delegato multicast
  - ⇨delegato capace di eseguire più di un metodo contemporaneamente
  - ⇨in effetti si tratta di un delegato risultato della combinazione di più delegati semplici
  - ⇨l’esecuzione del delegato produce l’esecuzione di tutti i delegati combinati

Delegati

- Vincolo sintattico
  - ⇨possono essere combinati solo delegati il cui tipo di ritorno è void
- In particolare
  - ⇨qualsiasi delegato con tipo di ritorno void è considerato un delegato multicast
  - ⇨e può essere utilizzato come delegato ordinario (“singlecast”)
  - ⇨oppure per combinare altri delegati multicast

Delegati

- In concreto
  - un delegato multicast mantiene una lista di riferimenti ad altri delegati multicast
  - quando viene eseguito esegue tutti i delegati della lista
- Per aggiungere un delegato alla lista
  - operatore +
- Per rimuovere un delegato dalla lista
  - operatore -

Delegati

>> Calcolatrice.Delegati

- Esempio
  - la calcolatrice con i delegati multicast
  - vengono utilizzati tre delegati per stampare il risultato di somma, sottrazione e prodotto
  - viene creato un delegato multicast che combina i tre per eseguire tutti e tre le stampe

```
namespace Calcolatrice.Delegati {

public delegate void Risultato(double a, double b);

public class PrincipaleMulticast {

   public void EseguiOperazioni() {
      double a, b;
      System.Console.Write("Primo argomento:   --> "); a = Unibas.Utilita.Console.LeggiDouble();
      System.Console.Write("Secondo argomento: --> ");  b = Unibas.Utilita.Console.LeggiDouble();
      Risultato somma = new Risultato(this.StampaSomma);
      Risultato differenza = new Risultato(this.StampaDifferenza);
      Risultato prodotto = new Risultato(this.StampaProdotto);
      Risultato risultato = somma + differenza;
      risultato += prodotto;
      risultato(a, b);
   }

   public void StampaSomma(double a, double b) { System.Console.WriteLine("Somma: " + (a + b)); }

   public void StampaDifferenza(double a, double b) { System.Console.WriteLine("Differenza: " + (a - b)); }

   public void StampaProdotto(double a, double b) { System.Console.WriteLine("Somma: " + (a * b)); }

   public static void Main(string[] args) {
      new PrincipaleMulticast().EseguiOperazioni();
   }
}
}
```

può essere usato come delegato multicast il tipo di ritorno è void

creazione del delegato multicast come combinazione di somma, differenza e poi anche prodotto

Delegati

- Nota
  - ⇨tutti i tipi delegati ereditano dalla classe System.Delegate
  - ⇨estesa da System.MulticastDelegate
- Alcuni metodi
  - ⇨static void Combine(Delegate d1, Delegate d2)
  - ⇨static void Remove(Delegate d1, Delegate d2)
  - ⇨Delegate[] GetInvocationList()

Delegati

- I membri di una classe C#
  - possono essere di vari tipi
  - campi
  - costruttori
  - metodi
  - proprietà .NET (get/set)
  - classi nidificate, enumerazioni, strutture
  - delegati, eventi (>>)

Thread

- Un esempio di uso dei delegati
  - i thread di C#
- Il namespace per i thread
  - System.Threading
- Le classi fondamentali
  - System.Threading.Thread
  - System.Threading.ThreadStart
  - System.Threading.Monitor

Thread

- Per creare un nuovo thread
  - ⇨bisogna creare un oggetto di tipo Thread
  - ⇨e poi chiamare il metodo Start() sull'oggetto
- Creare l'oggetto di tipo Thread
  - ⇨bisogna specificare il metodo da eseguire
  - ⇨per farlo è necessario creare un delegato di tipo ThreadStart
  - ⇨public delegate void ThreadStart()
  - ⇨e passare il delegato al costruttore di Thread

Thread

- I passi
  - ⇨creare il metodo per avviare il thread (analogo al metodo run() in Java)
  - ⇨creare un delegato di tipo ThreadStart per eseguire il metodo selezionato
  - ⇨creare un oggetto di tipo Thread a partire dal delegato
  - ⇨avviare il thread chiamando Start()

Thread

>> EsempioThread

- Per controllare il Thread
  - ⇨static void Sleep(int millis): metodo statico che mette a dormire il thread che lo esegue
  - ⇨void Join(): mette il thread che lo esegue in attesa che termini il thread su cui viene chiamato
  - ⇨void Abort(): solleva ThreadAbortException nel thread su cui viene chiamato; può essere usato per distruggere il thread

```
using System.Threading;

public class PrimoEsempio {

   private int id;

   public PrimoEsempio(int id) { this.id = id; }

   public void Run() {
      for (int i = 0; i < 20; i++) {
         System.Console.WriteLine("In esecuzione il thread " + this.id);
         Thread.Sleep(1000);
      }
   }

   public static void Main(string[] args) {
      PrimoEsempio primo = new PrimoEsempio(0);
      PrimoEsempio secondo = new PrimoEsempio(1);
      ThreadStart primoDelegato = new ThreadStart(primo.Run);
      ThreadStart secondoDelegato = new ThreadStart(secondo.Run);
      Thread thread0 = new Thread(primoDelegato);
      Thread thread1 = new Thread(secondoDelegato);
      thread0.Start();
      thread1.Start();
      System.Console.WriteLine("Main concluso");
   }
}
```

```
using System.Threading;
public class PrimoEsempioJoin {

   private int id;
   public PrimoEsempioJoin(int id) { this.id = id; }

   public void Run() {
      for (int i = 0; i < 20; i++) {
         System.Console.WriteLine("In esecuzione il thread " + this.id);
         Thread.Sleep(1000);
      }
   }

   public static void Main(string[] args) {
      PrimoEsempioJoin primo = new PrimoEsempioJoin(0);
      PrimoEsempioJoin secondo = new PrimoEsempioJoin(1);
      Thread thread0 = new Thread(new ThreadStart(primo.Run));
      Thread thread1 = new Thread(new ThreadStart(secondo.Run));
      thread0.Start();
      thread1.Start();
      try {
        thread0.Join();
        thread1.Join();
      } catch (ThreadInterruptedException) {}
      System.Console.WriteLine("Main concluso");
   }
}
```


Thread

- Sincronizzazione
  - ⇨come per Java ci sono due sintassi
- Sintassi esplicita
  - ⇨basata sulla classe System.Threading.Monitor
- Sintassi esplicita
  - ⇨basata sulla parola chiave lock
  - ⇨consente solo di acquisire e rilasciare blocchi

Thread

- Sincronizzazione esplicita
  - ⇨tutti gli oggetti di C# hanno un monitor implicito (lock implicito + condizione implicita)
  - ⇨manipolabile utilizzando i metodi statici della classe Monitor sul riferimento all'oggetto
  - ⇨Monitor.Enter(<rif>): acquisisce il lock
  - ⇨Monitor.Exit(<rif>): rilascia il lock
  - ⇨Monitor.Wait(<rif>): si mette in attesa della cond.
  - ⇨Monitor.PulseAll(<rif>): risveglia i thread in attesa

Thread

>> Unibas.BancaThread

- Il lock utilizzato
  - ⇨è rientrante
  - ⇨al solito è opportuno chiamare Monitor.Enter(this) prima di un try e Monitor.Exit(this) nel finally

```
Monitor.Enter(this);
try {
   // regione sincronizzata
} finally {
   Monitor.Exit(this);
}
```

Thread

- La sintassi implicita
  - ⇨utilizza la parola chiave lock

```
lock (this) {
   // regione sincronizzata
}
```

```
Monitor.Enter(this);
try {
   // regione sincronizzata
} finally {
   Monitor.Exit(this);
}
```


Riassumendo

- Le API di .NET
- Date
- Collezioni
- Ordinamento
- Clonazione e Serializzazione
- Classi Nidificate
- Delegati
- Thread

Termini della Licenza