Programmazione Orientata agli Oggetti in Linguaggio Java

Ereditarietà e Polimorfismo: C#

versione 1.0

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

Sommario

- Introduzione
- Ereditarietà
- Polimorfismo
- API di .NET
  - ⇨File di Risorse
  - ⇨File di Configurazione

Introduzione

>> appuntamenti
>> volpiEConigli

- Nel seguito
  - ⇨ereditarietà e polimorfismo in C#
- In particolare
  - ⇨illustreremo le differenze tra i due linguaggi sulla base dei progetti di riferimento per questo modulo

Ereditarietà

- In C#
  - ⇨l'ereditarietà ha esattamente la stessa semantica di quella di Java
  - ⇨cambia la sintassi
- In particolare
  - ⇨invece di "super" si usa "base"
  - ⇨non esiste la parola chiave extends
  - ⇨il costruttore della superclasse viene chiamato con una sintassi speciale

```
namespace Unibas.Appuntamenti.Modello {

public class Riunione : ImpegnoAstratto {

   private string descrizione;
   private string partecipanti;

   public Riunione() {}

   public Riunione(string descrizione, string partecipanti,
                   string luogo, string note, Orario orario) : base(luogo, note, orario) {
      this.descrizione = descrizione;
      this.partecipanti = partecipanti;
   }

   public override string ToString() {
      return (base.orario + " -- " + this.descrizione + " - luogo: " + base.luogo
                   + " - partecipanti: " + this.partecipanti
                   + " - note: " + base.note);
   }

   ...

}
}
```

i due punti (:) sostituiscono extends

la chiamata al costruttore della superclasse viene indicata nell'intestazione del costruttore della sottoclasse

il riferimento al padre è base e non super

Ereditarietà

- Nota
  - ⇨anche in C# vale il vincolo di ereditarietà singola
  - ⇨tutte le classi estendono System.Object
  - ⇨le regole per la costruzione degli oggetti sono identiche a quelle di Java
  - ⇨vale anche in questo caso il meccanismo dell'associazione gerarchica di oggetti per l'esecuzione dei metodi

Polimorfismo

- In C#
  - ⇨l'utilizzo del polimorfismo è più articolato che in Java
- In particolare
  - ⇨C# utilizza come semantica ordinaria il binding statico e non quello dinamico
  - ⇨il programmatore ha un controllo molto completo sul tipo di binding da utilizzare

Polimorfismo

- In Java, riassumendo
  - ⇨i metodi ridefiniti vengono eseguiti utilizzando l'algoritmo di binding dinamico
  - ⇨per evitarlo, il programmatore può solo impedire la ridefinizione usando final
  - ⇨ci sono due eccezioni: per i metodi privati e per i metodi statici viene usato il binding statico

Polimorfismo

- In C#
  - ⇨l'algoritmo standard di binding è quello statico (eredità del C++)
  - ⇨in altri termini, normalmente i metodi delle sottoclassi nascondono quelli delle superclassi e non li ridefiniscono
  - ⇨il compilatore fornisce messaggi di avvertimento se un metodo della sottoclasse ne nasconde uno della superclasse

Polimorfismo

```
using System;
class Superclasse {

   public void stampa() {
       Console.WriteLine("superclasse");
   }

}
```

```
using System;
class Sottoclasse : Superclasse {

   public void stampa() {
       Console.WriteLine("sottoclasse");
   }

}
```

```
// in Principale
 public static void Main(string[] args) {
       Superclasse s = new Sottoclasse();
       s.stampa();
 }
```

```
Prova.cs(11,17): warning CS0108:
    La parola chiave new è
    necessaria in
    'Sottoclasse.stampa()' perchè
    nasconde il membro ereditato
    Superclasse.stampa()'
```

```
risultato:
superclasse
```

Polimorfismo

- La parola chiave new
  - ⇨consente di specificare esplicitamente che il metodo della sottoclasse deve nascondere il corrispondente metodo della superclasse
  - ⇨in questo modo si elimina il warning

```
using System;
class Sottoclasse : Superclasse {
   public new void stampa() {
        Console.WriteLine("sottoclasse");
   }
}
```


Polimorfismo

risultato:
sottoclasse
superclasse

- Attenzione
  - ⇨il metodo nascosto è comunque accessibile attraverso il riferimento base

```
using System;
class Superclasse {
   public void stampa() {
        Console.WriteLine("superclasse");
   }
}
// in Principale
 public static void Main(string[] args) {
        Superclasse s = new Sottoclasse();
        s.stampa();
}
```

```
using System;
class Sottoclasse : Superclasse {
   public void esegui() {
        this.stampa();
        super.stampa();
   }
   public new void stampa() {
        Console.WriteLine("sottoclasse");
   }
}
```

Polimorfismo

- Il significato di “nascondere”
  - ⇨come per le proprietà in Java, il metodo della superclasse non appare più come ereditato dalla sottoclasse
  - ⇨accessibile attraverso this
  - ⇨ma non è realmente nascosto (è accessibile attraverso base)
  - ⇨l’oggetto ha due “versioni” diverse del metodo a disposizione

Polimorfismo

- Le ragioni di questa scelta
  - ⇨si tratta di una variante della semantica del C++, a cui C# è in parte legato
  - ⇨ragioni di prestazioni (il binding statico è più veloce)
  - ⇨ragioni di sicurezza (il programmatore non può nascondere per errore un metodo della superclasse)

Polimorfismo

- Di conseguenza
  - ⇨per rendere un metodo polimorfo ed applicargli il binding dinamico bisogna indicarlo esplicitamente come virtuale
  - ⇨con la parola chiave virtual
  - ⇨e sovrascriverlo esplicitamente con la parola chiave override

Polimorfismo

```
using System;
class Superclasse {

   public virtual void stampa() {
       Console.WriteLine("superclasse");
   }

}
```

```
using System;
class Sottoclasse : Superclasse {

   public override void stampa() {
       Console.WriteLine("sottoclasse");
   }

}
```

```
// in Principale
 public static void Main(string[] args) {
       Superclasse s = new Sottoclasse();
       s.stampa();
 }
```

risultato:
sottoclasse

Polimorfismo

- Il meccanismo, quindi
  - ⇨è più complesso
  - ⇨ci sono tre parole chiave nuove: new, virtual, override
  - ⇨diverse combinazioni possibili
  - ⇨con diverse semantiche
  - ⇨le riassumiamo nella trasparenza successiva

Polimorfismo

Metodo della superclasse	Metodo della sottoclasse	Messaggi del compilatore	Semantica del binding
nessuna p. chiave	nessuna p. chiave	Warning: il metodo è nascosto; usare new	binding statico
nessuna p. chiave	new	nessuno	binding statico
virtual	override	nessuno	binding dinamico
virtual	nessuna p. chiave	Warning: il metodo è nascosto; usare override	binding statico
virtual	new	nessuno	binding statico
nessuna p. chiave	override	Errore sintattico: il metodo non può essere ridefinito	-
new (????)	nessuna p. chiave	2 Warning: new è inutile perchè non nasconde; usare new nella sottoclasse	binding statico

Polimorfismo

- In concreto
  - ⇨tre casi significativi
  - ⇨nella superclasse non viene specificato nessun modificatore, nella sottoclasse new: binding statico
  - ⇨nella superclasse virtual, nella sottoclasse override: binding dinamico
  - ⇨nella superclasse virtual, nella sottoclasse new: binding statico

Polimorfismo

- Una nota importante
  - ⇨i metodi astratti (abstract) sono considerati automaticamente virtuali (virtual)
  - ⇨questo vale anche per le interfacce
  - ⇨i metodi ridefiniti (che contengono override nel prototipo) sono automaticamente virtuali
- Inoltre
  - ⇨i metodi privati e i metodi statici, come in Java, sono sempre oggetto di binding statico

Ereditarietà e Polimorfismo: C# >> Ereditarietà

```
namespace Unibas.Appuntamenti.Modello {

public interface IImpegno : System.IComparable {

   Giorno Giorno {
      get;
   }

   Orario Orario {
      get;
   }

   string Luogo {
      get;
      set;
   }

   string Note {
      get;
      set;
   }

   string ToString();
   string SaveString();
}
}
```

convenzione di stile di C#: i nomi delle interfacce cominciano con I

anche le proprietà oltre ai metodi possono essere astratte, secondo la sintassi illustrata

tutti i componenti dell'interfaccia sono implicitamente pubblici e virtuali

Ereditarietà e Polimorfismo: C# >> Ereditarietà

```
namespace Unibas.Appuntamenti.Modello {

public abstract class ImpegnoAstratto : IImpegno {

   protected string luogo;
   protected string note;
   protected Orario orario;
   protected Giorno giorno;

   protected ImpegnoAstratto() {}

   protected ImpegnoAstratto(string luogo, string note, Orario orario) {
      this.luogo = luogo;
      this.note = note;
      this.orario = orario;
   }

   public string Luogo {
      get { return this.luogo; }
      set { this.luogo = value; }
   }
   ...
   public abstract override string ToString();
   public abstract string SaveString();
}
}
```

i due punti (:) sostituiscono anche implements

attenzione alla parola chiave override

```
namespace Unibas.VolpiEConigli.Modello {

public interface Animale {

    void Agisci (Scacchiera scacchiera);

    bool Morto ();

    string ToString();

    bool IsConiglio();

}

}
```


```
namespace Unibas.VolpiEConigli.Modello {
public class Coniglio : Animale {

    private int eta = 0;
    private bool mangiato = false;

    public Coniglio() {}
    public Coniglio(int eta) { this.eta = eta; }

    public int Eta {
        get { return this.eta; }
    }

    public bool IsConiglio() {
        return true;
    }

    public bool Morto() {
        return (this.eta >= Configurazione.EtaLimiteConigli || this.mangiato);
    }

    public override string ToString() {
        return "c";
    }
    ...
```

API di .NET

- Operazioni significative con le API
  - ⇨tokenizzazione
  - ⇨utilizzo di file di configurazione
- Tokenizzazione
  - ⇨il metodo Split() di System.String
  - ⇨divide una stringa in token sulla base di un separatore
  - ⇨e restituisce un array dei token estratti

```
private static IImpegno EstraiImpegno(string lineaImpegno) {
    string[] tokens = lineaImpegno.Split("|".ToCharArray());
    string tipo = tokens[0].Trim();
    IImpegno impegno;
    if (tipo.Equals("riunione")) {
       impegno = EstraiRiunione(tokens);
    } else {
       impegno = EstraiLezione(tokens);
    }
    return impegno;
}

private static IImpegno EstraiRiunione(string[] tokens) {
    string stringaOrario = tokens[1];
    Orario orario = EstraiOrario(stringaOrario);
    string descrizione = tokens[2].Trim();
    string luogo = tokens[3].Trim();
    string partecipanti = tokens[4].Trim();
    string note = tokens[5].Trim();
    Riunione riunione = new Riunione(descrizione, partecipanti, luogo, note, orario);
    return riunione;
}
```

File di Risorse

- Un ultimo aspetto
  - ⇨.NET non prevede un analogo dei file di properties
- Filosofia
  - ⇨i file di properties sono basati su un formato non standard deciso dagli sviluppatori di Java
  - ⇨viceversa, .NET è costruito sul formato standard di XML

File di Risorse

- File di risorse
  - ⇨file che contengono dati utilizzati in un assembly (stringhe, oggetti, immagini)
  - ⇨possono essere file binari con estensione .resources
  - ⇨oppure file XML con estensione .resX
  - ⇨possono essere inclusi all'interno dell'assembly utilizzando l'opzione /res del compilatore csc

File di Risorse

- Utilizzo dei file di risorse
  - ⇨possono servire ad acquisire rapidamente proprietà per la configurazione dell'applicazione (come i file di properties)
  - ⇨servono a rendere l'assembly autocontenuto, distribuendo l'informazione aggiuntiva (configurazioni, messaggi in una o più lingue, immagini) all'interno dello stesso assembly

File di Risorse

- I file .resources
  - ⇨devono essere costruiti con un'opportuna applicazione detta Resource Creator (disponibile nella cartella del framework)
  - ⇨oppure con Visual Studio
- I file .resX
  - ⇨sono documenti XML ordinari
  - ⇨ma devono obbedire ad uno schema XML fissato, che viene immerso nel file resX

File di Risorse

>> configurazione.resX

- Un esempio di file di risorse
  - ⇨configurazione.resX
  - ⇨di fatto contiene un insieme di coppie nome, valore del tutto analogo ad un file .properties
- Acquisire i valori da un file di risorse
  - ⇨classi del namespace System.Resources
  - ⇨esistono classi per il formato .resources e classi per il formato .resX

File di Risorse

- System.Resources.ResxResourceSet
  - ⇨consente di acquisire valori da un file di risorse in formato .resX specificando il nome del file
  - ⇨il file è considerato al di fuori dell'assembly
  - ⇨i valori vengono prelevati attraverso il metodo string GetString(string nomeRisorsa)

```
public static void CaricaConfigurazione(String nomeFile) {
   System.Resources.ResXResourceSet rs = null;
   try {
      rs = new System.Resources.ResXResourceSet(nomeFile);
      double percentualeVolpi = Double.Parse(rs.GetString("percentualeVolpi"));
      double percentualeConigli = Double.Parse(rs.GetString("percentualeConigli"));
      int numeroRighe = Int32.Parse(rs.GetString("numeroRighe"));
      int numeroColonne = Int32.Parse(rs.GetString("numeroColonne"));
      int etaLimiteConigli = Int32.Parse(rs.GetString("etaLimiteConigli"));
      int fameLimiteVolpi = Int32.Parse(rs.GetString("fameLimiteVolpi"));
      int corsaLimiteConiglio = Int32.Parse(rs.GetString("corsaLimiteConiglio"));
      Configurazione.PercentualeVolpi = percentualeVolpi;
      Configurazione.PercentualeConigli = percentualeConigli;
      Configurazione.NumeroRighe = numeroRighe;
      Configurazione.NumeroColonne = numeroColonne;
      Configurazione.EtaLimiteConigli = etaLimiteConigli;
      Configurazione.FameLimiteVolpi = fameLimiteVolpi;
      Configurazione.CorsaLimiteConiglio = corsaLimiteConiglio;
   } catch (Exception e) {
       throw new DAOException(e.StackTrace);
   }
}
```


File di Configurazione

- Un metodo alternativo
  - ⇨utilizzare il file di configurazione dell'assembly
- File di configurazione dell'assembly
  - ⇨deve chiamarsi come l'assembly con il suffisso .config
  - ⇨es: volpiEConigli.exe.config
  - ⇨deve stare nella stessa cartella dell'assembly

File di Configurazione

- Contenuto del file di configurazione
  - ⇨normalmente parametri di configurazione per l'esecuzione dell'assembly nella macchina virtuale
  - ⇨ma può contenere anche parametri acquisibili all'interno dell'applicazione
  - ⇨all'interno dell'elemento <appSettings>

File di Configurazione

```
<?xml version="1.0" encoding="utf-8" ?>

<!-- File di configurazione per Volpi e Conigli -->

<configuration>
   <appSettings>
      <add key="percentualeConigli" value="0,1" />
      <add key="percentualeVolpi" value="0,05" />
      <add key="numeroRighe" value="10" />
      <add key="numeroColonne" value="15" />
      <add key="etaLimiteConigli" value="10" />
      <add key="fameLimiteVolpi" value="5" />
      <add key="corsaLimiteConiglio" value="4" />
   </appSettings>
</configuration>
```

File di Configurazione

- Prelevare il contenuto del file
  - ⇨è possibile attraverso la classe System.Configuration.ConfigurationSettings
  - ⇨la proprietà AppSettings restituisce un oggetto di tipo NameValueCollections (del namespace System.Collections.Specialized)
  - ⇨il metodo string Get(string nome) consente di acquisire il valore di una proprietà

File di Configurazione

```
public static void CaricaConfigurazione() {
    try {
       double percentualeVolpi = Double.Parse (
         System.Configuration.ConfigurationSettings.AppSettings.Get("percentualeVolpi"));
       double percentualeConigli = Double.Parse(
         System.Configuration.ConfigurationSettings.AppSettings.Get("percentualeConigli"));
       int numeroRighe =  Int32.Parse(
         System.Configuration.ConfigurationSettings.AppSettings.Get("numeroRighe"));
          ...
       Configurazione.PercentualeVolpi = percentualeVolpi;
       Configurazione.PercentualeConigli = percentualeConigli;
       Configurazione.NumeroRighe = numeroRighe;
          ...
    } catch (Exception e) {
         throw new DAOException(e.Message);
    }
}
```

Riassumendo

- Introduzione
- Ereditarietà
- Polimorfismo
- API di .NET
  - ⇨File di Risorse
  - ⇨File di Configurazione

Termini della Licenza