Programmazione Orientata agli Oggetti in Linguaggio Java

Eccezioni: C#

versione 2.1

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

Sommario

- Introduzione
- Gestione delle Eccezioni
- Asserzioni
- Gestione dei Flussi
- Tokenizzazione

Introduzione

- I principi metodologici visti
  - ⇨valgono identicamente per C#
  - ⇨in particolare, la valutazione di qualità si basa sugli stessi principi
- In questa lezione
  - ⇨gestione delle eccezioni in C#
  - ⇨utilizzo dei flussi
  - ⇨convenzioni di stile di C#

Introduzione

- Le buone notizie
  - ⇨è uno dei casi in cui la presentazione di C# è semplificata rispetto a quella di Java
- Infatti
  - ⇨la gestione delle eccezioni è semplificata
  - ⇨la gestione dei flussi è semplificata

Gestione delle Eccezioni

- In C#
  - ⇨la gestione delle eccezioni avviene secondo modalità molto simili a Java
  - ⇨esiste una istruzione throw
  - ⇨esistono blocchi try/catch/finally
- Ma c'è una differenza fondamentale
  - ⇨in C# NON esistono eccezioni controllate
  - ⇨la gestione delle eccezioni è del tutto facoltativa e dipende dal programmatore

Gestione delle Eccezioni

- Eccezioni di C#
  - ⇨sono tutte NON controllate
  - ⇨il compilatore NON impone di catturarle/rilanciarle
  - ⇨il programmatore può disinteressarsi delle istruzioni
- Di conseguenza
  - ⇨in C# NON esiste la clausola throws

Gestione delle Eccezioni

- Secondo alcuni
  - ⇨questo sancirà di fatto la pratica di NON gestire le eccezioni in C# e .NET
- Per il resto
  - ⇨la sintassi e la semantica è la stessa
- Un'unica annotazione
  - ⇨c'è una piccola differenza nella sintassi della regione catch
  - ⇨è possibile omettere il riferimento

Gestione delle Eccezioni

- La ragione: una regola sintattica di C#
  - ⇨se nel codice compaiono variabili dichiarate e non utilizzate viene segnalato un "warning"
- Esempio

```
try {
   file = new
   System.IO.StreamReader(this.nomeFileRecord);
   valoreRecord = System.Int32.Parse(file.ReadLine());
 } catch (System.IO.IOException se) {
        valoreRecord = 101;
} finally { ... }
return valoreRecord;
```

```
Record.cs(44,11): warning CS0168:
La variabile 'se' è dichiarata ma non
è mai stata utilizzata.
```

```
public int GetValoreRecord() {
      int valoreRecord;
      // ATTENZIONE: il codice puo' essere scritto
      // senza la gestione delle eccezioni
      System.IO.StreamReader file = null;
      try {
         file = new System.IO.StreamReader(this.nomeFileRecord);
         string linea = file.ReadLine();
         valoreRecord = System.Int32.Parse(linea);
      } catch (System.IO.IOException) {
         valoreRecord = 101;
      } finally {
         if (file != null) {
            file.Close();
         }
      }
      return valoreRecord;
   }
```


```
private void SetValoreRecord(int valoreRecord) {
      if (valoreRecord < 1) {
         throw new System.ArgumentException("Numero di tentativi scorretto");
      }
      // ATTENZIONE: il codice puo' essere scritto
      // senza la gestione delle eccezioni
      System.IO.StreamWriter file = null;
      try {
         file = new System.IO.StreamWriter(this.nomeFileRecord);
         file.WriteLine(valoreRecord + "");
      } catch (System.IO.IOException ioe) {
         System.Console.WriteLine("ERRORE : " + ioe);
      } finally {
         if (file != null) {
            file.Close();
         }
      }
   }
```

Gestione delle Eccezioni

ATTENZIONE
alla gestione delle eccezioni in C#

- Linee guida
  - è consigliabile utilizzare uno stile di programmazione difensiva anche in C#
  - anche se la gestione delle eccezioni non è obbligatoria
- Rispetto a Java
  - cambiano i nomi delle eccezioni
  - cambiano le eccezioni lanciate dai metodi

Gestione delle Eccezioni

- Per sapere quali eccezioni
  - consultare la documentazione di .NET
- Un buon approccio
  - conviene trattare come virtualmente "controllate" le eccezioni di C# che corrispondono a eccezioni controllate di Java
  - es: System.IO.IOException, System.IO.FileNotFoundException ecc.

Asserzioni

- Le asserzioni in C#
  - ⇨non esiste un'istruzione apposita
  - ⇨ma viene chiamato un metodo della classe System.Diagnostics.Debug
- Il metodo Assert
  - ⇨due parametri
  - ⇨l'asserzione
  - ⇨la stringa descrittiva

Asserzioni

- A differenza che in Java
  - ⇨dove le asserzioni vengono abilitate/disabilitate dalla macchina virtuale
  - ⇨in C# l'abilitazione viene fatta dal compilatore
- In particolare
  - ⇨normalmente le chiamate ai metodi della classe Debug sono escluse dal bytecode
  - ⇨vengono incluse in corrispondenza di direttive di compilazione

Asserzioni

- Primo modo
  - ⇨utilizzare direttive per il preprocessore nel codice
- Direttive del preprocessore
  - ⇨analoghe alle direttive del C/C++
  - ⇨devono essere specificate all'inizio del sorgente
  - ⇨servono a dare indicazioni al compilatore o alla macchina virtuale

Asserzioni

- La direttiva per la classe Debug
  - ⇨#define DEBUG
  - ⇨definisce il simbolo speciale DEBUG e indica al compilatore di abilitare i metodi della classe System.Diagnostics.Debug
- Esempio

```
#define DEBUG
namespace Unibas.Indovina {

    public class Partita {
        ...
    }
}
```

Asserzioni

- Un modo più flessibile
  - ⇨utilizzare l'opzione /define o /d del compilatore
  - ⇨l'opzione /d serve a fornire direttive al preprocessore dei sorgenti
  - ⇨/d:DEBUG equivale a specificare #define DEBUG per tutti i sorgenti compilati

Asserzioni

- In altri termini
  - ⇨in C# le asserzioni vengono effettuate con chiamate ad un metodo della classe Debug
  - ⇨a differenza di Java, vengono sempre compilate senza errori dal compilatore
  - ⇨ma sono abilitate solo nel caso in cui sia definito il simbolo DEBUG durante la compilazione (o nel codice o attraverso l'opzione /d)

Asserzioni

>> morraCinese – Partita

- Comportamento standard dell'asserzione
  - ⇨se ha successo non produce effetti
  - ⇨se fallisce, genera una finestra di dialogo con i dettagli sull'asserzione fallita
- Di conseguenza, in questo approccio
  - ⇨il fallimento deve essere gestito interattivamente (l'utente deve chiudere la finestra di dialogo)
  - ⇨non viene sollevata eccezione

Gestione dei Flussi

- Il namespace per la gestione dei flussi
  - ⇨System.IO
- Principali classi
  - ⇨File, FileInfo, Directory, DirectoryInfo
  - ⇨StreamReader, StreamWriter: classi per leggere e scrivere flussi Unicode
  - ⇨BinaryReader, BinaryWriter: flussi binari
  - ⇨BufferedStream, CryptoStream... altri flussi

Gestione dei Flussi

- In generale, in C#
  - ⇨la gestione dei flussi è decisamente semplificata rispetto a quella di Java
  - ⇨è stato fatto uno sforzo per semplificare la vita del programmatore
  - ⇨es: lo standard input (System.Console.In)
  - ⇨es: i flussi verso i file

Gestione dei Flussi

- La classe Console
  - ⇨la scrittura di Unibas.Utilita.Console è decisamente semplificata dall'esistenza di System.Console
  - ⇨e del metodo ReadLine() che effettua input non formattato dallo standard input
  - ⇨di conseguenza non è necessario creare altri flussi oltre al flusso System.Console.In

Gestione dei Flussi

- La classe Record
  - ⇨anche questa semplificata
- Per leggere da un file
  - ⇨basta creare uno StreamReader sulla base del nome del file e utilizzare ReadLine()
- Per scrivere in un file
  - ⇨basta creare uno StreamWriter sulla base del nome del file e utilizzare WriteLine()

```
public int GetValoreRecord() {
      int valoreRecord;
      // ATTENZIONE: il codice puo' essere scritto
      // senza la gestione delle eccezioni
      System.IO.StreamReader file = null;
      try {
         file = new System.IO.StreamReader(this.nomeFileRecord);
         string linea = file.ReadLine();
         valoreRecord = System.Int32.Parse(linea);
      } catch (System.IO.IOException) {
         valoreRecord = 101;
      } finally {
         if (file != null) {
            file.Close();
         }
      }
      return valoreRecord;
   }
```

```
private void SetValoreRecord(int valoreRecord) {
    if (valoreRecord < 1) {
        throw new System.ArgumentException("Numero di tentativi scorretto");
    }
    // ATTENZIONE: il codice puo' essere scritto
    // senza la gestione delle eccezioni
    System.IO.StreamWriter file = null;
    try {
        file = new System.IO.StreamWriter(this.nomeFileRecord);
        file.WriteLine(valoreRecord + "");
    } catch (System.IO.IOException ioe) {
        System.Console.WriteLine("ERRORE : " + ioe);
    } finally {
        if (file != null) {
            file.Close();
        }
    }
}
```


Tokenizzazione

- Anche in questo caso
  - ⇨è prevista una funzione di tokenizzazione
  - ⇨l'utilizzo è in qualche modo semplificato
- Il metodo Split() di System.String
  - ⇨divide una stringa in token sulla base di un separatore
  - ⇨e restituisce un array dei token estratti

Eccezioni: C# >> Tokenizzazione

```
public static Studente Carica(string nomeFile) {
    Studente studente = new Studente();
    System.IO.StreamReader flusso = null;
    try {
       flusso = new System.IO.StreamReader(nomeFile);
       EstraiDatiStudente(studente, flusso);
       CaricaEsami(studente, flusso);
    } catch (System.Exception e) {
        throw new PersistenceException("ERRORE:" + e.Message);
    } finally {
       try {
          if (flusso != null) {
             flusso.Close();
          }
       } catch (System.Exception) {}
    }
    return studente;
 }
```


Eccezioni: C# >> Tokenizzazione

```
private static void EstraiDatiStudente(Studente studente, System.IO.StreamReader flusso) {
    flusso.ReadLine();
    string lineaStudente = flusso.ReadLine();
    string[] tokens = lineaStudente.Split("|".ToCharArray());
    studente.cognome = tokens[0].Trim();
    studente.nome = tokens[1].Trim();
    studente.matricola = System.Int32.Parse(tokens[2].Trim());
}

private static void CaricaEsami(Studente studente, System.IO.StreamReader flusso) {
    flusso.ReadLine();
    while ((string lineaEsame = flusso.ReadLine()) != null) {
       studente.AddEsame(EstraiDatiEsame(lineaEsame, flusso));
    }
}

private static Esame EstraiDatiEsame(string lineaEsame, System.IO.StreamReader flusso) {
    Esame esame = new Esame();
    string[] tokens = lineaEsame.Split("|".ToCharArray());
    esame.Insegnamento = tokens[0].Trim();
    esame.Crediti = System.Int32.Parse(tokens[1].Trim());
    esame.Voto = System.Int32.Parse(tokens[2].Trim());
    esame.Lode = System.Boolean.Parse(tokens[3].Trim());
    return esame;
}
```

```
namespace Unibas.MediaPesata {

public class PersistenceException : System.Exception {

    public PersistenceException() : base() {
    }

    public PersistenceException(string s) : base(s) {
    }

    public PersistenceException(string s, System.Exception e) : base(s, e) {
    }

}

}
```


Riassumendo

- Introduzione
- Gestione delle Eccezioni
- Asserzioni
- Gestione dei Flussi

Termini della Licenza