Programmazione Orientata agli Oggetti in Linguaggio Java

Qualità del Codice: Conclusioni

versione 1.2

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

Sommario

- Riepilogo
- Utilizzo del CLASSPATH
- Le API di Java
  - ⇨Il Package java.io
  - ⇨La Classe Console
  - ⇨La Classe Record
- Un Altro Esempio

Riepilogo

- Qualità del codice
  - ⇨a breve termine: correttezza e robustezza
  - ⇨a lungo termine: manutenibilità
- Correttezza e robustezza
  - ⇨effettuare i test per la correttezza
  - ⇨e gestire le eccezioni
  - ⇨adottare uno stile di “programmazione difensiva”

Riepilogo

- Manutenibilità
  - ⇨è la caratteristica fondamentale
- I fattori essenziali
  - ⇨buona organizzazione del codice
  - ⇨semplicità e leggibilità
  - ⇨automatizzazione dei test (regressione)
- Il metodo di sviluppo suggerito
  - ⇨è finalizzato ad avere queste caratteristiche

Riepilogo

ATTENZIONE
a come cambia
il processo di sviluppo

- Il processo di sviluppo discusso finora
  - ⇨è centrato sulla corretta attribuzione delle responsabilità che garantisce buona organizzazione
- Le nuove linee guida
  - ⇨progettare e gestire le eccezioni
  - ⇨automatizzare i test
  - ⇨adottare convenzioni di stile

Riepilogo

- Progettare e gestire le eccezioni
  - ⇨adottare uno stile di programmazione difensiva
  - ⇨per ciascun metodo, valutare le possibili condizioni eccezionali e lanciare le corrispondenti eccezioni
  - ⇨negli schermi, effettuare sistematicamente la convalida dei dati dell'utente e le verifiche necessarie prima di chiamare i metodi

Riepilogo

- Automatizzare i test
  - ⇨i test sono parte integrante dell'applicazione
  - ⇨il processo di sviluppo deve alternare due attività
  - ⇨sviluppo del codice e sviluppo dei test
  - ⇨per ciascun metodo dell'applicazione scritto è opportuno scrivere uno o più metodi di test
  - ⇨è opportuno compilare frequentemente ed eseguire l'intera suite di test ogni volta

La Variabile CLASSPATH

- Una novità introdotta con JUnit
  - ⇨la gestione esplicita del CLASSPATH
  - ⇨serve ad aggiungere ulteriori punti di ricerca delle classi a quello standard (.)
- Attenzione
  - ⇨in teoria non è indispensabile definire la variabile di ambiente CLASSPATH
  - ⇨ma lo è in pratica per utilizzare applicazioni e framework diversi

La Variabile CLASSPATH

ATTENZIONE alla procedura tipica di installazione di un'applicazione Java

- La procedura tipica di installazione
  - ⇨di un'applicazione scritta in Java
- Operazione n. 1
  - ⇨scaricare il pacchetto (tipicamente in forma di archivio compresso: zip o jar)
- Operazione n. 2
  - ⇨decomprimere il pacchetto
- Operazione n. 3
  - ⇨aggiornare il CLASSPATH aggiungendo eventuali jar o cartelle contenenti il codice dell'applicazione

Le API di Java

- A questo punto
  - ⇨sono state introdotte praticamente tutte le nozioni necessarie per interpretare il codice dei progetti utilizzati in queste lezioni
  - ⇨con una eccezione
- La gestione dei flussi
  - ⇨la classe it.unibas.utilita.Console
  - ⇨la classe it.unibas.indovinasemplice.Record

Le API di Java

- La gestione dei flussi in Java
  - ⇨estremamente complessa e travagliata
- Java 1.0
  - ⇨prima versione del package java.io
- Java 1.1
  - ⇨completa revisione del package java.io
- Java 1.4
  - ⇨introduzione del package java.nio (new io)

Le API di Java

- Il package java.io
  - ⇨79 tra classi e interfacce (!)
- Il package java.nio
  - ⇨82 tra classi e interfacce (!!)
- Nel seguito
  - ⇨introduciamo alcuni elementi di base
  - ⇨riservandoci di tornarci nel seguito per discutere gli aspetti avanzati

Il Package java.io

- I flussi di Java 1.0
  - ⇨orientati ai byte
  - ⇨flussi di lettura (InputStream)
  - ⇨flussi di scrittura (OutputStream)
  - ⇨byte read(), void write(byte b) (oppure int)
- Attenzione
  - ⇨a differenza di altri linguaggi, in cui esistono pochi tipi di flussi, in Java esistono molti tipi diversi di flussi

Il Package java.io

- Principali tipi di flussi
  - ⇨flussi destinati ai file: FileInputStream e FileOutputStream
  - ⇨flussi tamponati (in cui i byte sono memorizzati in un buffer) BufferedInputStream, BufferedOutputStream
  - ⇨flussi per stampare valori di tipi diversi (int, float, double ecc.): PrintStream
  - ⇨molte altre categorie (es: flussi filtrati)

Il Package java.io

- Problemi dei flussi di Java 1.0
  - ⇨essendo orientati ai byte consentivano di gestire file di testo in formato ASCII
  - ⇨ma non in formato Unicode
  - ⇨caratteri Unicode: coppie di byte
- Viceversa
  - ⇨un InputStream legge un byte alla volta
  - ⇨un OutputStream scrive un byte alla volta

Il Package java.io

- I flussi di Java 1.1
  - ⇨vengono definiti flussi orientati a Unicode
  - ⇨flussi di lettura (Reader)
  - ⇨flussi di scrittura (Writer)
  - ⇨metodi char read(), void write(char c) //o int
  - ⇨e relativi derivati
  - ⇨FileReader, FileWriter
  - ⇨BufferedReader
  - ⇨PrintWriter

Il Package java.io

- L'importanza di BufferedReader
  - si tratta di un flusso tamponato orientato ai caratteri
  - i caratteri immessi sono tenuti in una zona temporanea di memoria prima di immetterli nel flusso
  - offre il metodo String readLine() – i dati possono essere letti a linee
- Esempio: lettura dalla tastiera
  - leggendo con un flusso non tamponato, non è possibile correggere gli errori
  - con un flusso tamponato, viceversa, sì

Il Package java.io

- Ma, per ragioni di compatibilità...
  - non era possibile eliminare i precedenti flussi
  - quindi sopravvivono entrambe le categorie
- Inoltre
  - Java 1.1 introduce classi per trasformare i vecchi flussi nei nuovi
  - InputStreamReader: trasforma un InputStream in un Reader
  - OutputStreamWriter: trasforma un OutputStream in un Writer

Il Package java.io

- Riassumendo, il package java.io
  - ⇨tante classi e interfacce diverse

Flussi orientati ai byte	Flussi orientati a Unicode	Traduttori
InputStream	Reader	InputStreamReader
OutputStream	Writer	OutputStreamWriter
FileInputStream	FileReader	
FileOutputStream	FileWriter	
BufferedeInputStream	BufferedReader	
BufferedOutputStream	BufferedWriter	
PrintStream	PrintWriter	

La Classe Console

- Obiettivo
  - ⇨trasformare System.in di tipo InputStream in un un flusso su cui eseguire input formattato e non formattato
- Per l'input non formattato
  - ⇨è sufficiente trasformare System.in in un Reader tamponato
  - ⇨per potere eseguire il metodo readLine() che restituisce una stringa di caratteri Unicode

La Classe Console

- Per l'input formattato
  - ⇨sfruttiamo le classi wrapper per i tipi di base che consentono di estrarre da una stringa un valore del tipo opportuno (metodi "parse")
- Di conseguenza
  - ⇨il primo passo è trasformare il flusso System.in in un Reader tamponato
  - ⇨e poi implementare i metodi per l'input non formattato e formattato

La Classe Console

- da InputStream (Sytem.in) a Reader
  Reader tmp = new InputStreamReader(System.in);
- da Reader a BufferedReader
  BufferedReader stdin = new BufferedReader(tmp);

```
package it.unibas.utilita;
import java.io.*;

public class Console {

   private Console() {}

   private static BufferedReader stdin = new BufferedReader(new InputStreamReader(System.in));

   public static String leggiStringa() {
      while (true) {
         try {
            String valore = stdin.readLine();
            return valore;
         } catch (IOException e) {
            System.out.println(e);
         }
      }
   }
```

sul flusso ottenuto posso eseguire il metodo public String readLine()

```
public static int leggiIntero() {
    while (true) {
        try {
            int valore = Integer.parseInt(stdin.readLine());
            return valore;
        } catch (IOException e) {
            System.out.println(e);
        } catch (NumberFormatException nfe) {
            System.out.println("*** Errore: non si tratta di un numero intero. Riprova.");
        }
    }
}

public static float leggiFloat() {
    while (true) {
        try {
            float valore = Float.parseFloat(stdin.readLine());
            return valore;
        } catch (IOException e) {
            System.out.println(e);
        } catch (NumberFormatException nfe) {
            System.out.println("*** Errore: non si tratta di un numero reale. Riprova.");
        }
    }
}
```

la classe wrapper Integer fornisce il metodo statico static int parseInt(String s) che estrae un intero da una stringa (es: 12 da “12”)

La Classe Console

- Nota
  - ⇨in tutti i metodi deve essere catturata java.io.IOException (eccezione controllata)
  - ⇨es: cosa succede se la tastiera è rotta ?
  - ⇨inoltre, nei metodi che fanno input formattato viene catturata java.lang.NumberFormatException (eccezione non controllata)
  - ⇨per rendere robusti i metodi di lettura

La Classe Record

- Obiettivo
  - ⇨tenere il valore del record in un file su disco
  - ⇨leggerne e scriverne il valore quando necessario
- Idea
  - ⇨la classe Record mantiene una proprietà di tipo String che corrisponde al nome del file
  - ⇨apre un flusso di lettura o scrittura quando necessario per effettuare le operazioni

La Classe Record

ATTENZIONE
ai procedimenti standard per la gestione dei file

- I procedimenti standard
- Per poter leggere dal file
  - ⇨è necessario creare un BufferedReader
  - ⇨e utilizzare readLine()
- Per poter scrivere nel file
  - ⇨è necessario creare un PrintWriter
  - ⇨e utilizzare print e println

La Classe Record

- creo un FileReader specificando il nome del file
- dal FileReader a BufferedReader

```
public int getValoreRecord() {
      int valoreRecord;
      java.io.BufferedReader file = null;
      try {
         file = new java.io.BufferedReader(new java.io.FileReader(this.nomeFileRecord));
         valoreRecord = Integer.parseInt(file.readLine());
      } catch (java.io.FileNotFoundException f) {
         valoreRecord = 101;
      } catch (java.io.IOException e) {
         System.out.println("Errore nella lettura del record: " + e);
         valoreRecord = 101;
      } catch (NumberFormatException nfe) {
         valoreRecord = 101;
      } finally {
         try {
            if (file != null) {file.close();}
         } catch (java.io.IOException ioe) {}
      }
      return valoreRecord;
   }
```


La Classe Record

```
private void setValoreRecord(int valoreRecord) {
      if (valoreRecord < 1) {
         throw new IllegalArgumentException("Numero di tentativi scorretto");
      }
      java.io.PrintWriter file = null;
      try {
         file = new java.io.PrintWriter(new java.io.FileWriter(this.nomeFileRecord)));
         file.println(valoreRecord + "");
      } catch (java.io.IOException e) {
         System.out.println("Impossibile salvare il record: " + e);
      } finally {
         if (file != null) {
            file.close();
         }
      }
   }
```

- creo un FileWriter specificando il nome del file
- da FileWriter a PrintWriter, sui cui posso usare println()

La Classe Record

- La classe java.io.File
  - ⇨la classe per effettuare operazioni direttamente sul file system
  - ⇨creare, cancellare, spostare file e cartelle
- Il metodo reset() di Record

```
public void reset() {
      try {
         java.io.File file = new java.io.File(this.nomeFileRecord);
         file.delete();
      } catch (SecurityException e) {System.out.println(e);}
   }
```


Un Altro Esempio

- Supponiamo
  - ⇨di volere mantenere una lista di record (es: tutti gli ultimi 10 record)
- In questo caso
  - ⇨cambia il codice della classe
  - ⇨introduciamo la classe ListaRecord
  - ⇨che preleva da un file i record e restituisce una lista
  - ⇨e, data una lista, la salva in un file

Un Altro Esempio

- Il formato del file listaRecord.txt
  - ⇨scelto dal programmatore
  - ⇨deve essere rispettato nel codice

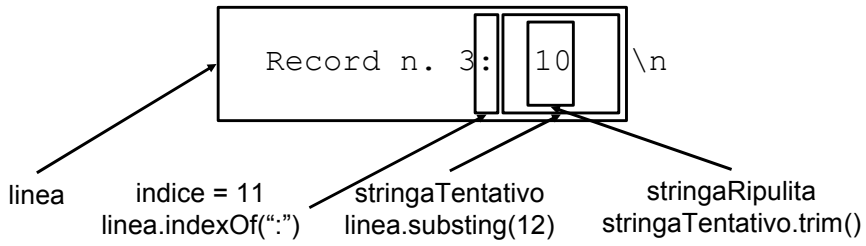
```
Lista dei record
----------------
Record n.1: 2
Record n.2: 5
Record n.3: 7
Record n.4: 10
```


```
package varie;
public class ListaRecord {

   private String nomeFile = "d:\\codice\\listaRecord.txt";

   public void scriviListaRecord(java.util.List lista) {
      java.io.PrintWriter flusso = null;
      try {
         java.io.FileWriter fileWriter = new java.io.FileWriter(nomeFile);
         flusso = new java.io.PrintWriter(fileWriter);
         flusso.println("Lista dei Record");
         flusso.println("----------------------");
         for (int i = 0; i < lista.size(); i++) {
            Integer elementoIesimo = (Integer)lista.get(i);
            int tentativi = elementoIesimo.intValue();
            flusso.println("Record n." + (i + 1) + ": " + tentativi);
         }
      } catch (java.io.IOException ioe) {
         System.out.println("ERRORE: " + ioe);
      } finally {
         if (flusso != null) {  flusso.close(); }
      }
   }
```

```
public java.util.List leggiListaRecord() {
      java.util.List lista = new java.util.ArrayList();
      java.io.BufferedReader flusso = null;
      try {
         java.io.FileReader fileReader = new java.io.FileReader(nomeFile);
         flusso = new java.io.BufferedReader(fileReader);
         String linea;
         flusso.readLine(); flusso.readLine(); // salta le prime due righe
         while ((linea = flusso.readLine()) != null) {
            int tentativi;
            try {
               tentativi = estraiTentativi(linea);
               lista.add(new Integer(tentativi));
            } catch (NumberFormatException nfe) {
               System.out.println("Errore: " + nfe);
            }
         }
      } catch (java.io.FileNotFoundException fnfe) {   System.out.println("ERRORE: " + fnfe);;
      } catch (java.io.IOException ioe) {  System.out.println("ERRORE: " + ioe);
      }
      finally {
         try {  if (flusso != null) { flusso.close(); } } catch (java.io.IOException ioe) {}
      }
      return lista;
   }
```



```
private int estraiTentativi(String linea) throws NumberFormatException {
      int tentativi;
      int indice = linea.indexOf(":");
      String stringaTentativo = linea.substring(indice + 1);
      String stringaRipulita = stringaTentativo.trim();
      tentativi = Integer.parseInt(stringaRipulita);
      return tentativi;
   }
```



Un Altro Esempio

- I metodi utilizzati di java.lang.String
  - ⇨s.indexOf(String s1): restituisce l'indice iniziale della stringa s1 in s oppure -1
  - ⇨s.substring(int index): restituisce la sottostringa di s fatta dei caratteri a partire dalla posizione index fino alla fine
  - ⇨s.trim(): elimina da s eventuali spazi iniziali e finali

Riassumendo

- Riepilogo
- Utilizzo del CLASSPATH
- Le API di Java
  - ⇨Il Package java.io
  - ⇨La Classe Console
  - ⇨La Classe Record
- Un Altro Esempio

Termini della Licenza