Programmazione Orientata agli Oggetti in Linguaggio Java

Test e Correzione: Debugging e Logging

versione 1.1

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

Sommario

- Debugging
- Utilizzo del Debugger
- Stampe di Debugging Tradizionali
- Sistema di Logging
- La Classe it.unibas.utilita.Logger

Debugging

- Debugging
  - ⇨ricerca della causa e correzione di un errore individuato attraverso un test (manuale o automatizzato)
- Il problema fondamentale
  - ⇨è la ricerca dell'errore
- Due tecniche fondamentali
  - ⇨stampe di debugging
  - ⇨utilizzo del debugger

Debugging

- Stampe di debugging
  - ⇨istruzioni di stampa sullo standard output dei valori delle variabili
  - ⇨possono essere aggiunte al codice del componente da correggere o anche al codice dei test
  - ⇨per capire meglio le cause reali dell'errore
  - ⇨metodo rapido e molto diffuso

Debugging

- Utilizzo del debugger
  - ⇨il debugger consente di eseguire il codice dell'applicazione
  - ⇨impostando "breakpoint", linee di codice sorgente in corrispondenza delle quali l'esecuzione si interrompe
  - ⇨in modo da ispezionare la pila di attivazione e poter procedere passo passo

Uso del Debugger

- Debugger ("Correttore")
  - ⇨strumento che consente di eseguire passo passo il codice
  - ⇨per ispezionare lo stack e lo heap durante il funzionamento di un programma
  - ⇨verificando i valori delle variabili
  - ⇨i valori dei parametri
  - ⇨lo stato dei componenti

Uso del Debugger

- Le operazioni tipiche del debugger
  - ⇨definire i punti di interruzione
  - ⇨comandare l'esecuzione del codice
- Definire i punti di interruzione
  - ⇨"breakpoint", ovvero righe di codice
  - ⇨"guardie" (watch), ovvero espressioni che rappresentano condizioni sullo stato del programma in cui fermarsi

Uso del Debugger

- Comandare l'esecuzione del codice
  - ⇨avviare l'esecuzione fino al punto di interruzione successivo
  - ⇨continuare sorpassando un punto di esecuzione
  - ⇨procedere passo passo
  - ⇨esplorare la pila
  - ⇨stampare i valori di espressioni complesse

Uso del Debugger

- Compilazione del codice
  - ⇨al solito, perchè le informazioni di debugging siano disponibili, è necessario compilare opportunamente il codice
- In Java
  - ⇨il compilatore javac prevede l'opzione –g per specificare le opzioni relative alle informazioni di debug

Uso del Debugger

- Varie opzioni
  - ⇨javac –g – produce nel file .class tutte le informazioni di debug
  - ⇨javac –g:none – nessuna info di debug
  - ⇨javac –g:{lines, vars, source} – produce informazioni solo su numeri di linea, variabili nella pila e file di codice sorgente
  - ⇨es: javac –g:lines,vars Prova.java
  - ⇨il valore standard è –g:lines, source

Uso del Debugger

- Il debugger fornito con l'SDK
  - ⇨jdb.exe
  - ⇨debugger puramente testuale e non grafico
  - ⇨eseguento jdb si accede ad un prompt attraverso il quale è possibile eseguire vari comandi
  - ⇨per un elenco: help

Uso del Debugger

```
>> javac –g circonferenze\*.java
>> jdb
```

- I principali comandi di jdb
  - ⇨run <classe>
  - ⇨stop in <classe>.<metodo>
  - ⇨stop in <classe>:<numeroLinea>
  - ⇨print <espressione>
  - ⇨locals (visualizza il contenuto dello stack frame corrente)

Uso del Debugger

- In effetti
  - ⇨si tratta di uno strumento molto poco usabile
  - ⇨esistono molti altri debugger open source per Java (es: JSwat)
- Ma, in generale...
  - ⇨per quanto il debugger resti uno strumento utile nel caso di bug particolarmente annidati e complessi da scovare, è in generale macchinoso

Uso del Debugger

- Una citazione dal libro “The Practice of Programming” di Brian Kernighan e Rob Pike

“As personal choice, we tend not to use debuggers beyond getting a stack trace or the value of a variable or two. One reason is that it is easy to get lost in details of complicated data structures and control flow; we find stepping through a program less productive than thinking harder and adding output statements and self-checking code at critical places. Clicking over statements takes longer than scanning the output of judiciously-placed displays. It takes less time to decide where to put print statements than to single-step to the critical section of code, even assuming we know where that is.
More important, debugging statements stay with the program; debugging sessions are transient.”

Uso del Debugger

- In altri termini, secondo gli autori
  - ⇨il debugger è uno strumento complesso
  - ⇨le sessioni di lavoro con il debugger richiedono tipicamente molto tempo
  - ⇨le sessioni del debugger vanno riprodotte ogni volta, mentre le stampe di debug restano nel codice e sono riutilizzabili
- Inoltre
  - ⇨non sempre il debugger è disponibile

Stampe di Debugging Tradizionali

- L'approccio tradizionale alle stampe
  - ⇨inserire nel codice istruzioni di stampa; es: System.out.println()
  - ⇨per visualizzare sullo schermo durante l'esecuzione il valore delle variabili, dei parametri e l'evoluzione del flusso di controllo
  - ⇨le istruzioni vengono commentate dopo aver trovato l'errore
  - ⇨ed eventualmente ripristinate per altri errori

Stampe di Debugging Tradizionali

- Un esempio
  - ⇨la classe Partita di indovinaIlNumero
  - ⇨supponendo che sia stato riscontrato un errore nello svolgimento del gioco
- Possibili stampe
  - ⇨stampa del valore del numero da indovinare
  - ⇨stampa dei tentativi effettuati
  - ⇨stampa dei risultati prodotti


```
package it.unibas.indovinailnumero;
public class Partita {
   public Partita() {
      this.numeroDaIndovinare = Math.abs(Partita.generatore.nextInt(100) + 1);
      System.out.println("*** Partita() : numeroDaIndovinare = " + numeroDaIndovinare);
      ...
   }
   public void gestisciTentativo(int tentativo) {
      System.out.println("*** gestisciTentativo() : tentativo = " + tentativo);
      this.numeroDiTentativi++;
      System.out.println("*** gestisciTentativo() : numTentativi = " + numTentativi);
      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";
      }
      System.out.println("*** gestisciTentativo() : suggerimento = " + suggerimento);
   }
   ...
}
```

Stampe di Debugging Tradizionali

- Il vantaggio di questo approccio
  - ⇨è un approccio semplice e immediato (rispetto all'uso del debugger)
- Gli svantaggi di questo approccio
  - ⇨sono però numerosi
- Svantaggio n. 1
  - ⇨è necessario modificare il codice per commentare – decommentare le stampe di debug (introduce potenzialmente errori)

Stampe di Debugging Tradizionali

- Svantaggio n. 2
  - ⇨le stampe vengono prodotte tutte sullo standard output
  - ⇨possono facilmente produrre lo scorrimento veloce dello schermo pregiudicandone la leggibilità
  - ⇨d'altro canto produrre le stampe in un flusso diverso (es: file) complicherebbe decisamente le istruzioni di stampa

Stampe di Debugging Tradizionali

- La soluzione ideale
  - ⇨dovrebbe essere basata su istruzioni e stringhe, in modo da essere semplice
  - ⇨dovrebbe consentire di abilitare e disabilitare le stampe senza modificare il codice
  - ⇨dovrebbe consentire di produrre le stampe su dispositivi diversi e in formati diversi
- Un esempio: indovina il numero

Sistema di Logging

- Sistema di logging
  - ⇨libreria di classi per la registrazione di informazioni sull'esecuzione del codice
- Caratteristiche fondamentali
  - ⇨consente registrazioni a diversi livelli di "gravità"
  - ⇨consente di registrare su dispositivi diversi
  - ⇨consente di registrare in formati diversi
  - ⇨e altre caratteristiche avanzate

Sistema di Logging

- Il sistema di logging standard di Java
  - ⇨il package java.util.logging (da Jdk1.4)
- Un sistema di logging molto diffuso
  - ⇨Log4j (org.apache.log4j)
  - ⇨open source (Apache Software Foundation)
  - ⇨scaricabile da http://logging.apache.org
  - ⇨utilizzato da moltissimi progetti industriali
- Si tratta in realtà di due framework

Sistema di Logging

- Idee principali di un sistema di logging
  - ⇨il concetto di “messaggio di logging”
  - ⇨il concetto di “logger”
  - ⇨il concetto di “handler”
  - ⇨il concetto di “livello di logging”
- Messaggio di logging (“log record”)
  - ⇨stringa registrata durante l’esecuzione di un metodo

Sistema di Logging

- Logger
  - componente responsabile della creazione dei messaggi di logging
  - è possibile utilizzare più di un Logger all'interno della stessa applicazione; es: un logger per package o per classe
  - ogni logger ha un nome
  - i logger sono organizzati in una gerarchia, con un logger principale, detto "root" logger

Sistema di Logging

- Handler (o Appender)
  - componente responsabile di registrare un messaggio di logging prodotto da un logger su un flusso o dispositivo
  - con un formato specificato
  - esempio: console, file, dbms ecc.
  - a ciascuna categoria di logger può essere associato uno o più handler

Sistema di Logging

- Livello di logging
  - ⇨descrive la gravità e l'urgenza di un messaggio
- Alcuni livelli tipici, in ordine decrescente
  - ⇨SEVERE: problema molto grave
  - ⇨WARNING: avvertimento potenz. grave
  - ⇨INFO: messaggio informativo sul funzionam.
  - ⇨FINE: messaggio di "tracing"
  - ⇨FINEST: messaggio di "tracing" dettagliato

Sistema di Logging

- Una caratteristica fondamentale
  - ⇨è possibile associare un livello a ciascun logger, in modo da abilitare o disabilitare i messaggi prodotti dal logger
  - ⇨il livello standard è tipicamente INFO, ma è configurabile attraverso un file di configuraz.
  - ⇨questo consente di controllare quali messaggi vengono effettivamente prodotti, eventualmente disabilitandoli completamente

Sistema di Logging

- Attenzione
  - ⇨le istruzioni di creazione dei messaggi vengono introdotte e restano nel codice
  - ⇨non c'è bisogno di commentarle
  - ⇨a seconda del livello abilitato, le registrazioni vengono effettuate o meno senza dover modificare il codice
- Esempio: indovina il numero

```
package it.unibas.indovinailnumero;
public class Partita {
   ...
   public Partita() {
      this.numeroDaIndovinare = Math.abs(Partita.generatore.nextInt(100) + 1);
      Logger.logFine("Numero scelto: " + numeroDaIndovinare);
      ...
   }

   public void gestisciTentativo(int tentativo) {
      if ((tentativo < 1) || (tentativo > 100)) {
         throw new IllegalArgumentException("Il valore deve essere compreso tra 1 e 100");
      }
      Logger.logFine("Partita", "gestisciTentativo", "Tentativo: " + tentativo);
      this.numeroDiTentativi++;
      Logger.logFine("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.logFine("Partita", "gestisciTentativo", "Suggerimento: " + suggerimento);
   }
   ...
```

Sistema di Logging

- In generale
  - ⇨si tratta di sistemi complessi
  - ⇨con molti aspetti di carattere tecnico
  - ⇨e alcune caratteristiche delicate
- Caratteristiche essenziali del sistema
  - ⇨efficienza
  - ⇨robustezza

Sistema di Logging

- Efficienza
  - ⇨le registrazioni devono essere prodotte rapidamente per non rallentare il codice
  - ⇨e il test per decidere se escludere o meno una registrazione deve essere eseguito molto rapidamente
- Esempio: log4j su un AMD 800Mhz
  - ⇨una registrazione costa da 21 a 37 micros.
  - ⇨decidere se effettuarla costa 5 nanosecondi

Sistema di Logging

- Robustezza
  - ⇨il codice del sistema di logging viene eseguito assieme al codice dell'applicazione
  - ⇨il sistema di logging non deve lanciare eccezioni
  - ⇨altrimenti interromperebbe il funzionamento dell'applicazione
  - ⇨sistemi di questo tipo vengono detti sistemi di tipo "fail-stop"

La Classe it.unibas.utilita.Logger

- In questo corso
  - ⇨adottiamo per ora il sistema di logging standard di Java
  - ⇨java.util.logging
- Il package java.util.logging
  - ⇨15 classi
  - ⇨2 interfacce
  - ⇨molti dettagli tecnici

La Classe it.unibas.utilita.Logger

- La classe it.unibas.utilita.Logger
  - ⇨una interfaccia semplice verso il package java.util.logging
- In particolare
  - ⇨crea e mette a disposizione un logger
  - ⇨consente di gestire i principali livelli
  - ⇨consente di effettuare registrazioni su console e su file

La Classe it.unibas.utilita.Logger

- I metodi di logging
  - ⇨quattro livelli: severe, info, fine, finer

```
public static void logFiner(String messaggio);
public static void logFiner(String classe, String metodo, String messaggio);

public static void logFine(String messaggio);
public static void logFine(String classe, String metodo, String messaggio);

public static void logInfo(String messaggio);
public static void logInfo(String classe, String metodo, String messaggio);

public static void logSevere(String messaggio);
public static void logSevere(String classe, String metodo, String mess);
```

La Classe it.unibas.utilita.Logger

- Il livello standard di logging
  - ⇨INFO; di conseguenza vengono di norma prodotti tutti i messaggi di livello superiore o uguale ad INFO
- L'handler standard di logging
  - ⇨produce stampe sulla console
- I due parametri sono stabiliti nel file
  - ⇨%JRE_HOME%\lib\logging.properties

La Classe it.unibas.utilita.Logger

```
############################################################
#   Default Logging Configuration File
############################################################

# "handlers" specifies a comma separated list of log Handler.
handlers= java.util.logging.ConsoleHandler

# Default global logging level.
java.util.logging.level= INFO

# default file output is in user's home directory.
java.util.logging.FileHandler.pattern = %h/java%u.log
java.util.logging.FileHandler.limit = 50000
java.util.logging.FileHandler.count = 1
java.util.logging.FileHandler.formatter = java.util.logging.XMLFormatter

# Limit the message that are printed on the console to INFO and above.
java.util.logging.ConsoleHandler.level = INFO
java.util.logging.ConsoleHandler.formatter = java.util.logging.SimpleFormatter
```

La Classe it.unibas.utilita.Logger

- File .properties
  - ⇨file contenente proprietà di sistema
- Proprietà di sistema (“system property”)
  - ⇨coppia nome=valore
  - ⇨serve normalmente a modificare il comportamento di un’applicazione
  - ⇨per esempio a specificare parametri di configurazione

La Classe it.unibas.utilita.Logger

- Acquisizione delle proprietà
  - ⇨caricando un file di properties (utilizzando la classe java.util.Properties >>)
  - ⇨utilizzando l’opzione –D per passare la coppia nome=valore alla macchina virtuale
- La classe it.unibas.utilita.Logger
  - ⇨può essere configurata utilizzando opportune proprietà di sistema

La Classe it.unibas.utilita.Logger

- Le proprietà di sistema rilevanti
  - ⇨it.unibas.utilita.LogLevel
  - ⇨it.unibas.utilita.LogFile
- Esempio
  - ⇨java -Dit.unibas.utilita.LogLevel=finer <classe>
  - ⇨imposta il livello di logging a FINER
  - ⇨java -Dit.unibas.utilita.LogLevel=finer -Dit.unibas.utilita.LogFile=e:\tmp\log.txt <classe>
  - ⇨imposta il livello di logging e il log su file

La Classe it.unibas.utilita.Logger

- Valori per it.unibas.utilita.LogLevel
  - ⇨severe
  - ⇨info
  - ⇨fine
  - ⇨finer
  - ⇨off
  - ⇨all

La Classe it.unibas.utilita.Logger

>> Partita.java
>> Record.java

- Esempio
  - ⇨indovinaIlNumero con logging abilitato
- Messaggi di logging prodotti

Livello: FINE
14-dic-2004 1.19.48 it.unibas.utilita.Logger logFine
FINE: Numero scelto: 55
14-dic-2004 1.19.56 Partita gestisciTentativo
FINE: Tentativo: 50
14-dic-2004 1.19.56 Partita gestisciTentativo
FINE: Num. tentativi: 1
14-dic-2004 1.19.56 Partita gestisciTentativo
FINE: Suggerimento: Prova con un numero più alto

Riassumendo

- Debugging
- Utilizzo del Debugger
- Stampe di Debugging Tradizionali
- Sistema di Logging
- La Classe it.unibas.utilita.Logger

Termini della Licenza

- This work is licensed under the Creative Commons Attribution-ShareAlike License. To view a copy of this license, visit http://creativecommons.org/licenses/by-sa/1.0/ or send a letter to Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305, USA.

- Questo lavoro viene concesso in uso secondo i termini della licenza “Attribution-ShareAlike” di Creative Commons. Per ottenere una copia della licenza, è possibile visitare http://creativecommons.org/licenses/by-sa/1.0/ oppure inviare una lettera all’indirizzo Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305, USA.