Programmazione Orientata agli Oggetti in Linguaggio Java

Test e Correzione: Conclusioni - Parte b Riflessione

versione 1.2

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

Sommario

- Riflessione
- La Classe java.lang.Class
- Il Package java.lang.Reflect
- Decompilatori e Offuscatori
- Classloader

Riflessione

- JUnit, un aspetto misterioso
  - ⇨come fa il testRunner a sapere quali sono i metodi di test da eseguire ?
  - ⇨che cosa vuol dire TestXXX.class nel metodo testSuite() ?
- In effetti
  - ⇨si tratta di tecniche di programmazione avanzate, basate sulla “riflessione”

Riflessione

ATTENZIONE
al concetto di riflessione

- Riflessione
  - ⇨funzionalità per cui è possibile scrivere codice di un linguaggio la cui funzione è analizzare codice dello stesso linguaggio
  - ⇨“il codice riflette su sè stesso”
- Nel caso di JUnit
  - ⇨il testRunner “carica” classi Java (ovvero bytecode) e lo analizza per capire quali i metodi di test da eseguire

Riflessione

- Perchè esiste la riflessione in Java ?
  - ⇨perchè la filosofia della piattaforma è che tutti gli strumenti collegati al codice sono scritti essi stessi in Java
  - ⇨ovvero sono componenti Java che hanno bisogno di manipolare altri componenti Java
  - ⇨es: compilatore, caricatore (“classloader”)
  - ⇨analogamente per gli IDE

La Classe java.lang.Class

- L’idea alla base della riflessione
  - ⇨esiste una classe java.lang.Class
  - ⇨ad ogni classe Java corrisponde un oggetto della classe java.lang.Class
  - ⇨attraverso i metodi della classe è possibile analizzare tutte le caratteristiche della classe
- Analogo
  - ⇨tutti gli oggetti sono anche istanze di java.lang.Object

La Classe java.lang.Class

- In effetti
  - ⇨java.lang.Class si può definire una "metaclasse", ovvero una classe le cui istanze (oggetti) rappresentano altre classi
- Per ogni classe del linguaggio
  - ⇨esiste un oggetto di tipo java.lang.Class
  - ⇨che descrive il contenuto del file .class contenente il codice oggetto della classe

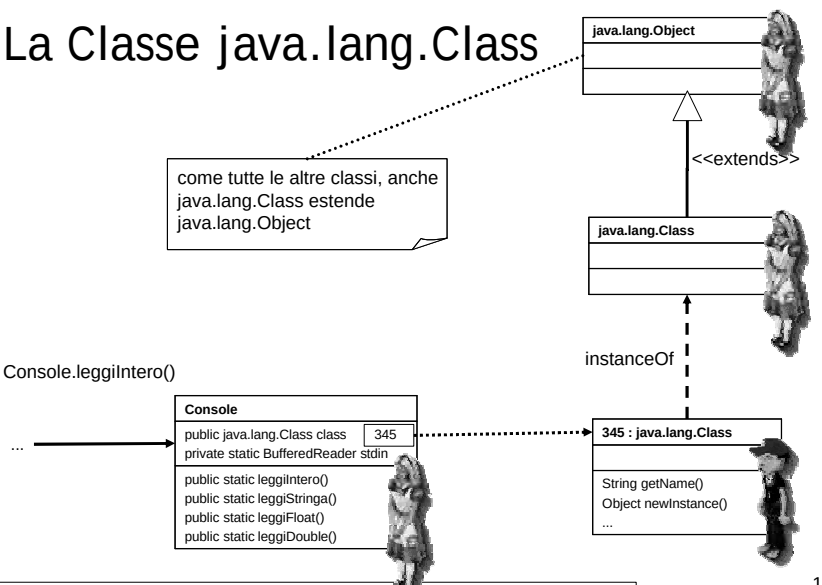
La Classe java.lang.Class

- Il riferimento all'oggetto di tipo Class
  - ⇨è accessibile in vari modi
- Proprietà statica class
  - ⇨tutte le classi hanno una proprietà pubblica e statica chiamata class che mantiene un riferimento all'oggetto di tipo java.lang.Class
  - ⇨es: java.lang.Class classe = it.unibas.utilita.Console.class;

La Classe java.lang.Class

- Inizializzazione della proprietà class
  - ⇨viene effettuata automaticamente dalla macchina virtuale al caricamento della classe
- In particolare
  - ⇨viene caricato il bytecode di java.lang.Class
  - ⇨viene creato un oggetto di tipo Class
  - ⇨viene caricato il bytecode della classe
  - ⇨viene inizializzata la proprietà class

La Classe java.lang.Class



La Classe java.lang.Class

- Il metodo getClass() di Object
  - ⇨alternativa alla proprietà statica class
  - ⇨ereditato da tutti gli oggetti
  - ⇨consente di ottenere il riferimento all'oggetto di tipo java.lang.Class a partire da un oggetto invece che dalla classse
  - ⇨es: Integer integer = new Integer();
    java.lang.Class classe = integer.getClass();

La Classe java.lang.Class

- Un esempio di metodi di java.lang.Class
  - ⇨String getName(): restituisce il nome della classe
- Due metodi interessanti
  - ⇨static java.lang.Class forName(String nome): cerca ed eventualmente carica l'oggetto Class di una classe dato il nome sotto forma di stringa
    es: Class.forName("it.unibas.utilita.Logger");
  - ⇨Object newInstance(): crea un nuovo oggetto della classe e ne restituisce l'identificatore

La Classe java.lang.Class

- Creare gli oggetti
  - ⇨due modi diversi
- Modo tradizionale
  - ⇨chiamare il costruttore
  - ⇨richiede di conoscere staticamente (a tempo di compilazione) il nome della classe
- Modo basato sulla riflessione
  - ⇨utilizzando forName e newInstance sono in grado di creare oggetti di classi arbitrarie

Esempio

```
public void creaOggetto() {
      System.out.println("Immetti il nome della classe: ");
      String nomeClasse = it.unibas.utilita.Console.leggiStringa();
      try {
         java.lang.Class classe = java.lang.Class.forName(nomeClasse);
         Object oggetto = classe.newInstance();
         System.out.println(oggetto);
      } catch (ClassNotFoundException e) {
         System.out.println(e);
      } catch (InstantiationException e) {
         System.out.println(e);
      } catch (IllegalAccessException e) {
         System.out.println(e);
      }
   }
```

Il Package java.lang.Reflect

- In realtà
  - ⇨il sistema di classi che è possibile usare per la riflessione è molto più ampio
  - ⇨esempio: java.lang.Package
- Il package java.lang.reflect
  - ⇨fornisce tutte le classi necessarie allo studio di una classe Java attraverso la riflessione

Il Package java.lang.Reflect

- Principali classi
  - ⇨java.lang.reflect.Field per le proprietà
  - ⇨java.lang.reflect.Method per i metodi
  - ⇨java.lang.reflect.Constructor per i costruttori
- Inoltre
  - ⇨java.lang.reflect.Modifier
  - ⇨java.lang.reflect.Array

Il Package java.lang.Reflect

- Attraverso l'oggetto class
  - ⇨è possibile ottenere per una classe i riferimenti agli oggetti di tipo Field, Method, Constructor ecc.
  - ⇨analizzarne le caratteristiche (anche se sono privati !)
  - ⇨e utilizzarli dinamicamente (cambiare una proprietà, eseguire un metodo ecc.)

Il Package java.lang.Reflect

- Un esempio
  - ⇨la classe it.unibas.utilita.Ispezionatore
  - ⇨riceve in ingresso il nome di una classe raggiungibile attraverso il classpath
  - ⇨e ne studia il contenuto utilizzando la riflessione
  - ⇨NOTA: lavora sul bytecode, non richiede la presenza del sorgente

Il Package java.lang.Reflect

>> esecuzione di it.unibas.utilita.Ispezionatore

- Utilizzo
  - ⇨java it.unibas.utilita.Ispezionatore <nomeClasse>
  - ⇨il package it.unibas.utilita deve essere raggiungibile attraverso il classpath
- Esempi
  - ⇨java –cp e:\codice\lib\utilita.jar it.unibas.utilita.Ispezionatore it.unibas.utilita.Console
  - ⇨java –cp e:\codice\lib\utilita.jar it.unibas.utilita.Ispezionatore java.lang.String

Decompilatori e Offuscatori

- Riassumendo
  - ⇨a partire dal bytecode è possibile risalire a moltissime delle caratteristiche del codice sorgente originale
  - ⇨anche senza avere a disposizione il .java
- In effetti
  - ⇨nei linguaggi basati sulla riflessione, questo processo può essere spinto molto avanti

Decompilatori e Offuscatori

- Decompilatore
  - ⇨applicazione che ispeziona il codice oggetto di una classe
  - ⇨utilizza la riflessione per estrarne le caratteristiche
  - ⇨e disassembla il codice oggetto per ricostruire la struttura dei metodi
  - ⇨in altri termini, ricostruisce integralmente il codice sorgente originale dal codice oggetto

Decompilatori e Offuscatori

>> DJ su Ispezionatore.class

- Un esempio
  - ⇨DJ, Java Decompiler
  - ⇨applicazione gratuita per il disassemblaggio di codice Java
- Chiaramente
  - ⇨l'utilizzo dei decompilatori provoca gravi problemi di tutela della proprietà del software commerciale

Decompilatori e Offuscatori

>> ProGuard su utilita.jar: da progrard/lib
java –jar proguard.jar @../utilita.pro

- Offuscatore
  - ⇨applicazione che riorganizza il codice oggetto di una classe per renderlo più difficilmente manipolabile dal decompilatore
  - ⇨in sostanza tende a confondere nomi e identificatori
- Un esempio
  - ⇨ProGuard, progetto open source

Il Classloader

- A questo punto
  - ⇨è possibile entrare maggiormente nel dettaglio dei meccanismi di esecuzione della macchina virtuale
- Un principio essenziale
  - ⇨la macchina virtuale in realtà utilizza le classi esclusivamente attraverso l'oggetto Class corrispondente

Il Classloader

- L'operazione fondamentale
  - ⇨è caricare il bytecode della classe dal disco e creare l'oggetto class
  - ⇨con tutti gli oggetti correlati
- Il classloader
  - ⇨è il componente incaricato di questa operazione
  - ⇨un oggetto della classe java.lang.ClassLoader

Il Classloader

- Classloader usati dalla macchina virtuale
  - ⇨sono normalmente più di uno
- System ClassLoader
  - ⇨l'oggetto di tipo classloader utilizzato per caricare la prima classe (quella da cui viene avviata l'esecuzione)
- Successivamente
  - ⇨possono essere utilizzati altri classloader figli

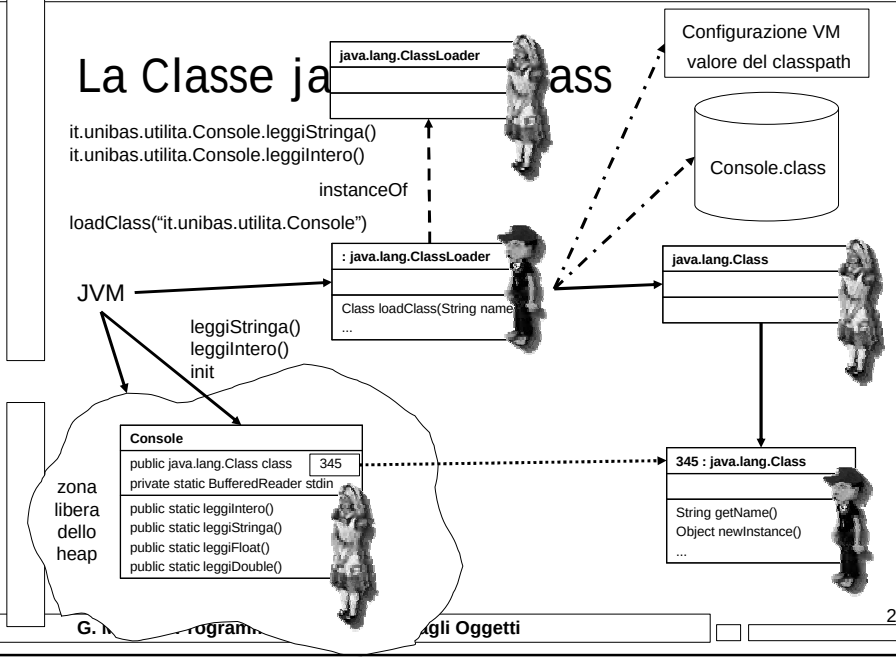
Il Classloader

- Il metodo principale di ClassLoader
  - ⇨public Class loadClass(String nome)
  - ⇨dato un nome di classe, carica dal disco il bytecode (byte), li analizza e costruisce l'oggetto class corrispondente
  - ⇨a quel punto la macchina virtuale può provvedere all'inizializzazione del componente e all'utilizzo

Il Classloader

- La preparazione di una classe all'uso
  - ⇨caricamento: il classloader cerca il file .class su disco attraverso il classpath e carica il bytecode dal file .class su disco
  - ⇨collegamento: il bytecode dei metodi viene verificato e viene assegnato lo spazio nello heap al componente (proprietà statiche)
  - ⇨inizializzazione: la macchina virtuale inizializza le proprietà statiche ed esegue i blocchi di inizializzazione statici

La Classe ja ass



Il Classloader

- Nota
  - ⇨ lo stesso principio viene utilizzato per il caricamento di risorse correlate (file che non contengono bytecode)
- I metodi di ClassLoader
  - ⇨ InputStream getResourceAsStream(String nome)
  - ⇨ java.net.URL getResource(String nome): restituisce l'URI di una risorsa localizzandola sul classpath (dall'URI è poi necessario creare lo stream per caricarla)

Il Classloader

- Utilizzo tipico
  - ⇨caricamento di file contenenti dati da manipolare nell'applicazione
- Esempio
  - ⇨immagini in formato .gif o .jpg da visualizzare negli schermi
  - ⇨file di properties con parametri di configurazione

Riassumendo

- Introduzione
- La Classe java.lang.Class
- Il Package java.lang.Reflect
- Decompilatori e Offuscatori
- Classloader

Termini della Licenza