JAVA – Identificazione di tipo a run-time

Metodi Avanzati di Programmazione

Laurea Triennale in Informatica

Università degli Studi di Bari Aldo Moro

Docente: Pierpaolo Basile

Introduzione

- L'idea di avere l'identificazione dei tipi di oggetti in fase di esecuzione (Run-Time Type Identification, **RTTI**) sembra in apparenza abbastanza semplice e di sicuro interesse nella progettazione object-oriented
- Java permette di scoprire delle informazioni sugli oggetti e sulle classi al run-time, basandosi essenzialmente su due approcci diversi:
  - RTTI "tradizionale": in cui si presuppone che le informazioni su tutti i tipi sono accessibili sia in fase di compilazione e sia in fase di esecuzione
  - il meccanismo di riflessione (reflection): che permette di scoprire informazioni sulle classi esclusivamente al run-time

RTTI tradizionale…

- Per comprendere il funzionamento di RTTI in Java bisogna capire come sono rappresentate al run-time le informazioni sul tipo (cioè sulla classe)
- Ciò è realizzato attraverso un tipo speciale di oggetto chiamato **Class** object che contiene le informazioni sulla classe (per questo talvolta è chiamato *meta-classe*)
  - Quali informazioni? Attributi, metodi, modalità di accesso, etc., cioè tutte le informazioni presenti nel .class
  - Durante la compilazione, viene creato un oggetto Class per ogni classe che costituisce il programma.

…RTTI tradizionale…

- Gli oggetti di Class relativi alle varie classi che compongono un programma non sono caricati tutti in memoria prima di iniziare l'esecuzione
- Quando al run-time si istanzia una classe, la Java Virtual Machine (JVM), su cui sta girando il programma, prima verifica se l'oggetto Class corrispondente è caricato. In caso negativo la JVM lo carica ricercando il file .class con quel nome
  - In questo esempio, ognuna delle classi Candy e Cookie ha una *clausola statica* che viene eseguita quando la classe è caricata la prima volta

```
class Candy {
   static {  // questa è una clausola statica
      System.out.println("Loading Candy");
   }
}
class Cookie {
   static {  // questa è una clausola statica
      System.out.println("Loading Cookie");
   }
}

public class SweetShop {
   public static void main(String[] args) {
      System.out.println("inside main");
      new Candy();
      System.out.println("After creating Candy");
      try {
         Class.forName("Gum");
      } catch(ClassNotFoundException e) {
         e.printStackTrace(System.err);
      }
      System.out.println("After Class.forName(\"Gum\")");
      new Cookie();
      System.out.println("After creating Cookie");
   }
}
```

…RTTI tradizionale…

- Il metodo forName() è un metodo statico di Class che serve per ottenere un riferimento a un oggetto Class. Esso prende un oggetto di tipo String contenente il nome testuale della classe di cui si vuole il riferimento e restituisce un riferimento a Class
- Si può notare come ogni oggetto Class è stato caricato solo quando era necessario

…RTTI tradizionale.

- Alternativamente, per ottenere un riferimento a un oggetto Class si può anche ricorrere al letterale di classe (class literal), dato dal nome della classe seguito da .class (esempio: **Gum.class**)
- I vantaggi di questa notazione sono:
  - Semplicità
  - Efficienza (non si invoca il metodo forName)
  - Controllo di esistenza della classe durante la compilazione.
- Il letterale di classe funziona, oltre che con le classi, con gli array, con i tipi primitivi (e.g., boolean.class) e con le interfacce

…RTTI tradizionale.

- Per i “wrapper” dei tipi primitivi c’è anche un campo standard chiamato TYPE. Questo campo produce un riferimento all’oggetto Class per il tipo primitivo associato tale che si hanno le seguenti equivalenze

… is equivalent to …	
boolean.class	Boolean.TYPE
char.class	Character.TYPE
byte.class	Byte.TYPE
short.class	Short.TYPE
int.class	Integer.TYPE
long.class	Long.TYPE
float.class	Float.TYPE
double.class	Double.TYPE
void.class	Void.TYPE

RTTI in Java…

Le forme di RTTI viste finora, includono:

- il classico **cast** che usa RTTI per assicurarsi che il cast è corretto e solleva una eccezione ClassCastException se è stato ottenuto un cast non corretto
- l'**oggetto Class** rappresentante il tipo dell'oggetto. L'oggetto Class può essere interrogato per ottenere utili informazioni al run-time
- In C++ il classico cast non compie una RTTI. Dice semplicemente al compilatore di trattare l'oggetto come di un altro tipo
- In Java, che esegue il controllo di tipo, questo tipo di cast è spesso chiamato "type safe downcast"

…RTTI in Java…

Un'altra forma di RTTI in Java è ottenuta attraverso l'uso della parola chiave **instanceof** che indica se un oggetto è istanza di un particolare tipo e restituisce un boolean

```
if (m instanceof Dog) ((Dog)m).bark();
```

- L'istruzione precedente, verifica se l'oggetto *m* appartiene alla classe Dog prima di effettuare il casting, altrimenti si potrebbe sollevare una ClassCastException

```
class Pet {}
class Dog extends Pet {}
class Pug extends Dog {}
class Cat extends Pet {}
class Rodent extends Pet {}
class Gerbil extends Rodent {}
class Hamster extends Rodent {}
class Counter { int i; }

import java.util.*;
public class PetCount {
   static String[] typenames = {"Pet", "Dog", "Pug", "Cat","Rodent",
   "Gerbil", "Hamster", };
   public static void main(String[] args) throws Exception {
      ArrayList pets = new ArrayList();
      Class[] petTypes = {Class.forName("Dog"), Class.forName("Pug"),
      Class.forName("Cat"), Class.forName("Rodent"), Class.forName("Gerbil"),
      Class.forName("Hamster")};
      for(int i = 0; i < 15; i++)
         pets.add(petTypes[(int)(Math.random()*petTypes.length)]
         .newInstance());
```

```
HashMap h = new HashMap();
for(int i = 0; i < typenames.length; i++)
   h.put(typenames[i], new Counter());
for(int i = 0; i < pets.size(); i++) {
   Object o = pets.get(i);
   if(o instanceof Pet)
      ((Counter)h.get("Pet")).i++;
   if(o instanceof Dog)
      ((Counter)h.get("Dog")).i++;
   if(o instanceof Pug)
      ((Counter)h.get("Pug")).i++;
   if(o instanceof Cat)
      ((Counter)h.get("Cat")).i++;
   if(o instanceof Rodent)
      ((Counter)h.get("Rodent")).i++;
   if(o instanceof Gerbil)
      ((Counter)h.get("Gerbil")).i++;
   if(o instanceof Hamster)
      ((Counter)h.get("Hamster")).i++;
}
```

```
    for(int i = 0; i < pets.size(); i++)
       System.out.println(pets.get(i).getClass());
    for(int i = 0; i < typenames.length; i++)
       System.out.println(typenames[i] + " quantity: " +
       ((Counter)h.get(typenames[i])).i);
  }
}
```

…RTTI in Java.

- Quando si dispone di un oggetto, si può estrarre il riferimento all'oggetto Class relativo alla sua classe richiamando un metodo che è implementato in Object: getClass( ).
- Nel precedente esempio, alternativamente all'uso di Class.forName si possono usare i letterali class
  - Es. `Cat.class.newInstance()`
  - In questo caso la creazione di petTypes non deve essere inclusa in un blocco try, perché viene valutata al compile-time, diversamente dal metodo Class.forName().

…RTTI in Java.

- L’uso dell’operatore instanceof potrebbe risultare spesso molto noioso perché lo si deve specificare per il confronto di ogni tipo di oggetto distinto
- La classe Class mette a disposizione il metodo isInstance che fornisce un modo per invocare dinamicamente l’operatore instanceof

```
Object o = …
if (Dog.class.isInstance(o)) …
```

Il meccanismo di riflessione…

- Nel meccanismo di RTTI tradizionale, il tipo dell'oggetto deve essere noto al compilatore ovvero il compilatore deve avere una conoscenza completa delle classi utilizzate
- Talvolta le informazioni sulla classe dell'oggetto non sono accessibili a tempo di compilazione. In tal caso risulta molto utile poter usufruire di un meccanismo che ricava le informazioni relative alla classe al run-time
- La classe Class supporta il concetto di riflessione e c'è una libreria aggiuntiva java.lang.reflect che contiene delle classi utili allo scopo: Field, Method, Constructor (ognuno dei quali implementa una interfaccia Member).

…Il meccanismo di riflessione…

- Questo tipo di oggetti sono creati dalla JVM al run-time per rappresentare il corrispondente membro della classe sconosciuta
- Consultando la documentazione on-line della classe Class si nota che le informazioni relative alla classe di oggetti anonimi, possono essere completamente ricavate al run-time
- Quando si usa il meccanismo di riflessione, la JVM tratta l'oggetto come appartenente ad una classe particolare
- Il file class per questa classe particolare deve essere ancora accessibile alla JVM sia sulla macchina locale che sulla rete

…ll meccanismo di riflessione…

- La differenza tra RTTI tradizionale e riflessione è che nella RTTI, il compilatore accede ed esamina il file .class a tempo di compilazione mentre con la riflessione esso è accessibile solo dall'ambiente run-time
- Non capiterà spesso di ricorrere al meccanismo di riflessione per quanto riguarda le applicazioni tradizionali
- Gli strumenti di riflessione sono stati implementati per supportare caratteristiche avanzate di Java come l'invocazione remota di metodi (RMI)

…Il meccanismo di riflessione.

- Tuttavia in alcuni casi tradizionali, risulta molto utile estrarre dinamicamente le informazioni relative ad una classe
- Uno strumento molto utile è l'estrattore del metodo della classe

```
import java.lang.reflect.*;

public class ShowMethods {
  static final String usage = "usage: \n" +
   "ShowMethods qualified.class.name\n" +
   "To show all methods in class or: \n" +
   "ShowMethods qualified.class.name word\n" +
   "To search for methods involving 'word'";
  public static void main(String[] args) {
     if(args.length < 1) {
        System.out.println(usage);
        System.exit(1);
     }
     try {//args[0] è il nome della classe
        Class c = Class.forName(args[0]);
        Method[] m = c.getMethods();
        Constructor[] ctor = c.getConstructors();
        if(args.length == 1) {
           for (int i = 0; i < m.length; i++)
              System.out.println(m[i]);
           for (int i = 0; i < ctor.length; i++)
              System.out.println(ctor[i]);
        } else { //args[1] specifica il metodo o costruttore che sto cercando
```

```
      for (int i = 0; i < m.length; i++)
         if(m[i].toString().indexOf(args[1])!=-1)
            System.out.println(m[i]);
      for (int i = 0; i < ctor.length; i++)
         if(ctor[i].toString().indexOf(args[1])!=-1)
            System.out.println(ctor[i]);
    }
  } catch(ClassNotFoundException e) {
    System.err.println("No such class: " + e);
  }
 }
}
```


MWAHAHAHA
I'M USING JAVA REFLECTION API
memegenerator.net