RTTI - Run-Time Type Identification

RTTI …

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 …

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.

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

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

L'output del programma è :

```
inside main
Loading Candy
After creating Candy
Loading Gum
After Class.forName("Gum")
Loading Cookie
After creating Cookie
```

Esempio di funzionamento del Class Loader di Java

…RTTI …

In questo esempio, ognuna delle classi *Candy*, *Gum* e *Cookie* ha una ***clausola statica*** che viene eseguita quando la classe è caricata la prima volta.

…RTTI …

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"*.

```
class Cat{
void print(){System.out.println(«cat»);}
}
class Dog{
void print(){System.out.println(«dog»);}
}

class MainClass{
public static void main (String args[]){
ArrayList a=new ArrayList();
for (int i=0;i<7;i++)
        a.add(new Cat());
a.add(new Dog());
for(int i=0;i<a.size();i++)
{
        Object o=a.get(i);
        Class c=o.getClass();
        if (c.getName().equals("Cat") ((Cat)o).print();
        if (c.getName().equals("Dog") ((Dog)o).print();
}
}
```

…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*.

```
//esempio di uso di istanceof()
// Pets.java
class Pet {}
class Dog extends Pet {}
class Pug extends Dog {}
class Counter { int i; }
```

```
import java.util.*;
public class PetCount {
static String[] typenames = {"Pet", "Dog",
"Pug"};
// Eccezioni evidenziate su console:
public static void main(String[] args)
{
 ArrayList pets = new ArrayList();
 try {
       Class[] petTypes = {
             Class.forName("Dog"),
             Class.forName("Pug"),
             };
       for(int i = 0; i < 5; i++)
             pets.add(
             petTypes[(int)(Math.random()*
             petTypes.length)].newInstance());
       }
 catch(InstantiationException e) {
       System.err.println("Cannot instantiate");
       return;
 }
 catch(IllegalAccessException e) {
       System.err.println("Cannot access");
       return;
 }
 catch(ClassNotFoundException e) {
       System.err.println("Cannot find class");
       return;
 }
 …
```

```
 …
 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++;
 }
 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);
 }
}
```

```
Output:
class Dog
class Pug
class Pug
class Dog
class Pug
Pet quantity: 5
Dog quantity: 2
Pug quantity: 3
```

…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*.

Esempio

```
Class[] petTypes = {Dog.class, Pug.class};
```

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*.

```
Class[] petTypes = {Dog.class, Pug.class};
…
```

```
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++;
}
```

versus

```
    for (int j = 0; j < petTypes.length; ++j)
        if (petTypes[j].isInstance(o)) {
        String key = petTypes[j].toString();
        ((Counter)h.get(key)).i++;
    }
```