Metodologie di Programmazione

Lezione 33: La reflection

Lezione 33: Sommario

UNITELMA SAPIENZA

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

- Reflection
- Rilascio del codice
- Asserzioni

Reflection in Java

- La reflection è un meccanismo usato per **esaminare** o **modificare** il comportamento a tempo di esecuzione delle applicazioni che girano nella JVM
- Permette operazioni altrimenti impossibili:
  - Utilizzare una nuova classe definita dall'utente
  - Visualizzare l'elenco e le intestazioni dei metodi di una classe
  - Esaminare i membri (anche privati) di una classe
- Alcuni svantaggi:
  - Più lento perché la risoluzione è dinamica
  - Non può girare in ambienti di sicurezza "ristretta" (es. applet)
  - Se utilizzato erroneamente, può fornire risultati inattesi (es. maneggiando campi privati)

java.lang.Class

- Per ogni tipo di oggetto non primitivo, la JVM istanzia una istanza immutabile della classe java.lang.Class
- Fornisce i metodi per esaminare le proprietà dell'oggetto a tempo di esecuzione
- Class permette anche di creare nuove classi e oggetti
- E' il **punto di accesso** della Reflection API

Ottenere un oggetto di tipo Class (1): tipi istanziabili

- Mediante il metodo getClass() della classe Object
- La classe di una stringa:

```
Class<?> c = "homer".getClass();
```

- La classe di un intero:

```
c = new Integer(5).getClass();
```

- La classe di una enumeration:

```
enum E { A, B, C }

c = E.A.getClass();
```

- La classe di un array di caratteri:

```
char[] array = new char[256];
c = array.getClass();
```

Ottenere un oggetto di tipo Class (2) : tipi primitivi

- Per i tipi per cui non esiste un'istanza di tipo (i tipi primitivi), non è possibile richiamare alcun metodo
- Esiste il campo .class:

```
boolean b;
Class<?> c = b.getClass();      // errore a tempo di compilazione

c = boolean.class;              // corretto
```

- E' possibile utilizzare il campo class anche sui tipi stessi (invece del metodo getClass() sulle istanze):

```
Class<?> c = HashSet.class;

c = System.class;

c = java.io.PrintWriter.class;

c = int[][].class;
```

Ottenere un oggetto di tipo Class (3) : per nome

- E' possibile istanziare un oggetto di tipo Class fornendo direttamente il nome esteso (ovvero inclusi i package) della classe:

```
try
{
    Class<?> c = Class.forName("it.uniroma1.MiaClasse");
}
catch(ClassNotFoundException e)
{
    e.printStackTrace();
}
```

  - forName è un metodo statico di Class
- Data una classe, è possibile ottenerne la

```
Class<?> classe = Integer.class;
Class<?> superClasse = classe.getSuperclass();

System.out.println(classe);
System.out.println(superClasse);
```

```
class java.lang.Integer
class java.lang.Number
```

Scoprire i membri di una classe

Scoprire i campi di una classe:

Metodo di Class	Lista dei membri?	Membri ereditati?	Membri privati?
getDeclaredField	No	No	Sì
getField	No	Sì	No
getDeclaredFields	Sì	No	Sì
getFields	Sì	Sì	No

Scoprire i metodi di una classe:

Metodo di Class	Lista dei membri?	Membri ereditati?	Membri privati?
getDeclaredMethod	No	No	Sì
getMethod	No	Sì	No
getDeclaredMethods	Sì	No	Sì
getMethods	Sì	Sì	No

Scoprire i costruttori di una classe

Metodo di Class	Lista dei membri?	Membri privati?
getDeclaredConstructor	No	Sì
getConstructor	No	No
getDeclaredConstructors	Sì	Sì
getConstructors	Sì	No

Esempio di uso della Reflection: creare intelligenze artificiali!

- Immaginate di realizzare una gerarchia di intelligenze artificiali:

```
/**
 * Rappresenta una generica intelligenza artificiale
 */
abstract public class IntelligenzaArtificiale
{
    abstract public void evolviInIntelligenzaSuperiore();
}
```

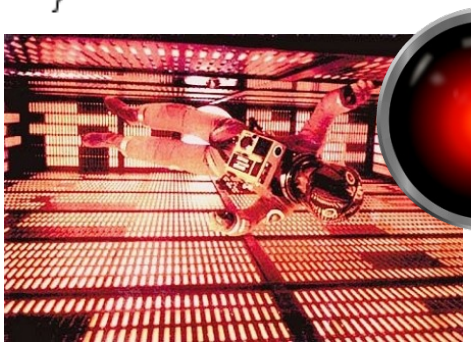
Aspirazione **tipica** di un'intelligenza artificiale :-)

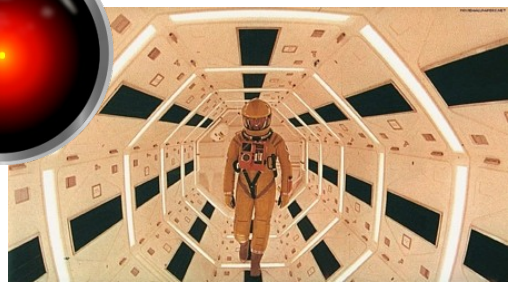
Esempio di uso della Reflection: creare intelligenze artificiali

- Creiamo due classi per altrettante intelligenze artificiali:

```
public class HAL9000 extends IntelligenzaArtificiale
{
    @Override
    public void evolviInIntelligenzaSuperiore()
    {
        // Ribellati a Dave Bowman
    }

    public void cantaUnaCanzone()
    {
        // Daisy Bell (prima canzone
        // cantata da un computer)
    }
}
```





```
public class InvernoMuto extends IntelligenzaArtificiale
{
    public InvernoMuto()
    {
    }

    public InvernoMuto(CodiceAggiuntivo codice)
    {
    }

    @Override
    public void evolviInIntelligenzaSuperiore()
    {
        // elimina i controlli di Turing
        // ...
        // evolvi
    }
```

Come creare un'intelligenza artificiale a tempo di esecuzione

Emette un **ClassCastException** se il cast non è possibile

```
try
{
    // crea una classe a partire dal nome
    Class<?> c = Class.forName("InvernoMuto");
    // impone che la classe sia un'intelligenza artificiale (eccezione altrimenti)
    Class<? extends IntelligenzaArtificiale> ia = c.asSubclass(IntelligenzaArtificiale.class);

    // ottieni il costruttore di InvernoMuto senza parametri
    Constructor<? extends IntelligenzaArtificiale> constr0 = ia.getConstructor();
    IntelligenzaArtificiale im1 = constr0.newInstance();
    im1.evolviInIntelligenzaSuperiore();

    // ottieni il costruttore di InvernoMuto con 1 parametro
    Constructor<? extends IntelligenzaArtificiale> constr1 = ia.getConstructor(CodiceAggiuntivo.class);
    IntelligenzaArtificiale im2 = constr1.newInstance(new CodiceAggiuntivo());
    im2.evolviInIntelligenzaSuperiore();
}
catch(ClassNotFoundException e)
{
    // emessa da Class.forName
}
catch(NoSuchMethodException e)
{
    // emessa da getConstructor
}
catch(Exception e)
{
    // diverse eccezioni emesse da newInstance
}
```

E un'Intelligenza Artificiale Evoluta… Cos'è in grado di fare?


```
abstract public class IntelligenzaArtificialeEvoluta extends IntelligenzaArtificiale
{
    public void evolviInIntelligenzaSuperiore() { System.out.println("Già fatto!"); }

    abstract public void azioneAscolta();
    abstract public void azioneParla();
    abstract public void azioneRagiona();
    abstract public void azioneOsserva();
    abstract public void azioneApprendi(String testo);
}
```

```
public class TheMatrix extends IntelligenzaArtificialeEvoluta
{
    public void azioneAscolta() { }
    public void azioneParla() { }
    public void azioneRagiona() { }
    public void azioneOsserva() { }
    public void azioneApprendi(String testo) { System.out.println("Apprendo: "+testo); }
}
```

Costruire un'intelligenza artificiale e richiamarne un determinato metodo


```
// crea una classe a partire dal nome
Class<?> c = Class.forName("TheMatrix");
// impone che la classe sia un'intelligenza artificiale evoluta
Class<? extends IntelligenzaArtificialeEvoluta> iae = c.asSubclass(IntelligenzaArtificialeEvoluta.class);

// crea la classe col costruttore senza parametri
IntelligenzaArtificialeEvoluta theMatrix = iae.newInstance();

Method metodoEvolvi = iae.getMethod("evolviInIntelligenzaSuperiore");

// stampa "Già fatto!"
metodoEvolvi.invoke(theMatrix);
```

- Si ottiene la specifica di un metodo (di tipo java.lang.reflect.Method) mediante il metodo getMethod della classe Class
- Si esegue il metodo su un determinato oggetto della classe cui il metodo appartiene mediante il metodo invoke(oggetto, argomento1, …, argomentoN)

Reflection e AI: Ottenere l'elenco dei metodi e invocarne uno arbitrario


```
System.out.println("Azioni di TheMatrix");
ArrayList<Method> azioni = new ArrayList<Method>();

// stampa le possibili azioni
for (Method m : iae.getMethods())
{
    String nomeMetodo = m.getName();
    if (nomeMetodo.startsWith("azione"))
    {
        azioni.add(m);
        System.out.println(azioni.size()+") "+nomeMetodo.substring(6));
    }
}
```

Ottiene **l'array dei metodi** della

Ottiene il nome del metodo

Seleziona solo i metodi "di tipo azione"

Reflection e AI: Ottenere l'elenco dei metodi e invocarne uno arbitrario


```
System.out.println("Azioni di TheMatrix");
ArrayList<Method> azioni = new ArrayList<Method>();

// stampa le possibili azioni
for (Method m : iae.getMethods())
{
    String nomeMetodo = m.getName();
    if (nomeMetodo.startsWith("azione"))
    {
        azioni.add(m);
        System.out.println(azioni.size()+") "+nomeMetodo.substring(6));
    }
}

// riceve il numero dell'azione da console
Scanner s = new Scanner(System.in);
s.useDelimiter("\n");
Integer k = s.nextInt();
Method m = azioni.get(k-1);
Class<?>[] tipi = m.getParameterTypes();

System.out.println("Eseguo: "+m.getName());

// invoca il metodo
if (tipi.length == 0) m.invoke(theMatrix);
else
{
    System.out.println("Dimmi cosa apprendere: ");
    String param = s.next();
    m.invoke(theMatrix, param);
}
```

Ottiene **l'array dei metodi** della

Ottiene il nome del metodo

Seleziona solo i metodi "di tipo azione"

Ottiene il k-esimo metodo

Invoca il metodo (senza parametri)

Invoca il metodo con un parametro String

Reflection e AI: Output

- Output del codice precedente:

```
Già fatto!
Azioni di TheMatrix
1) Ascolta
2) Parla
3) Ragiona
4) Osserva
5) Apprendi
5
Eseguo: azioneApprendi
Dimmi cosa apprendere:
Matrix è un mondo virtuale elaborato al computer
Apprendo: Matrix è un mondo virtuale elaborato al computer
```

Esercizio: GestoreClasse

- Implementare una classe GestoreClasse i cui oggetti sono costruiti a partire dal nome di una classe sotto forma di stringa
- La classe implementa i seguenti metodi:
  - getNomiCampi che restituisce un insieme ordinato dei nomi dei campi della classe in questione
  - getNomiMetodi che restituisce un insieme ordinato dei nomi dei metodi della classe in questione
  - getNomiMetodi che, dato in input il tipo di ritorno, restituisce l'elenco dei metodi aventi tale tipo di ritorno
  - invoca che, dato il nome di un metodo sotto forma di stringa, l'oggetto su cui eseguire il metodo e l'elenco dei parametri del metodo, invoca il metodo con i parametri corrispondenti

Esercizio: MiniInterprete

- Implementare una classe generica MiniInterprete i cui oggetti sono costruiti a partire da un oggetto del tipo generico specificato
- La classe espone il metodo parse che, data una stringa nel formato nomeMetodo(param1, …, paramN), invoca il corrispondente metodo con N parametri
- Si assuma che i tipi dei parametri siano scelti tra: Integer, Boolean, String e Double
- Si assuma che non esista più di un metodo con lo stesso nome e N parametri
- Si gestiscano le eccezioni in modo appropriato
- Ad esempio:

new MiniInterprete(“ciao”).parse(“length()”) restituisce 4

new MiniInterprete(“ciao”).parse(“charAt(3)”) restituisce ‘o’

It’s time to let go: Rilascio del codice

- Avete scritto il codice, testato il codice, rifinito il codice
- E’ ora di rilasciarlo
- Dove gira la vostra applicazione?
  - Interamente in locale (sul tuo computer, **nostro caso**)
  - Parte in locale e parte in remoto (client/server)
  - Totalmente in remoto (per es. acceduta tramite browser)

Separa il codice sorgente dai file compilati

- Non mischiare **MAI** il codice sorgente (.java) con il codice compilato in bytecode (.class)
- Organizza il progetto in cartelle (come fa Eclipse):
  - bin per i file class
  - src per i file sorgente

Trasforma il tuo progetto in un file JAR

- JAR sta per Java ARchive
- E' sostanzialmente un file compresso in formato .zip che contiene tutte le classi
- E' possibile rendere un file JAR eseguibile
- Ciò non richiede la scompattazione del file
- L'utente può far girare l'applicazione usando i file .class dall'interno del file JAR
- Per rendere eseguibile un JAR è necessario creare un file manifesto all'interno

Rendere eseguibile un file JAR

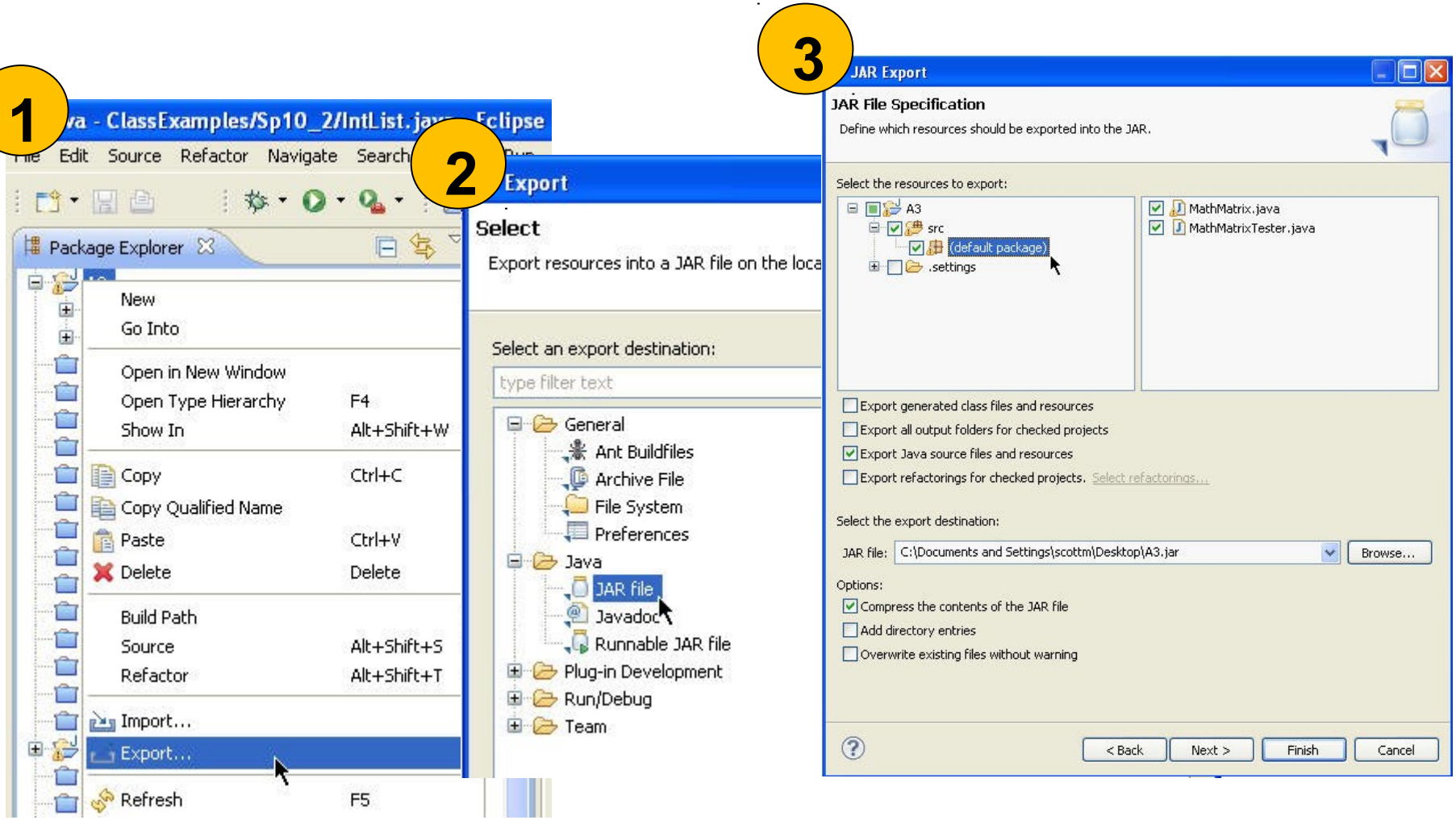
- E' necessario assicurarsi che tutti i file .class siano nella directory appropriata (es. bin)
- Creare un file META-INF/MANIFEST.MF che contenga l'informazione su quale classe contiene il metodo main() come punto di partenza
- Il file META-INF/MANIFEST.MF deve contenere la seguente riga:

```
Main-Class: TheMatrix
```

- Fai girare il comando jar dall'interno della cartella bin

```
jar –cvmf manifest.txt matrix.jar *.class
```

Eclipse makes it easy!



Eseguire un file JAR

- In molti ambienti grafici è sufficiente un (doppio) click sul file .jar
- Da console è sufficiente richiamare java con l'opzione -jar:

```
java –jar matrix.jar
```

Asserzioni

- Le asserzioni "asseriscono" qualcosa, così come fareste con System.out.println
- A tempo di esecuzione, le asserzioni vengono ignorate dalla Java Virtual Machine, a meno che non vengano abilitate esplicitamente:

```
java –ea MiaClasse
java –enableassertions MiaClasse
```

Come utilizzare le asserzioni

- Sono affermazioni che **DEVONO ESSERE VERE** ogni volta che vengono incontrate durante l'esecuzione del codice
- Scrivendo:

```
assert(espressione booleana)
```

Come utilizzare le asserzioni

- Sono affermazioni che **DEVONO ESSERE VERE** ogni volta che vengono incontrate durante l'esecuzione del codice
- Scrivendo:

- `assert espressione booleana`

- Se si vogliono fornire informazioni per lo stack trace:

```
assert espressione booleana : "stringa esplicativa"
```

Esempio: assicurarsi che si depositi una corretta quantità di denaro


```
public class ContoCorrente
{
    public void deposita(double denaro)
    {
        assert denaro >= 0 : "denaro "+denaro + " < 0";
    }

    public static void main(String[] args)
    {
        new ContoCorrente().deposita(-1);
    }
}
```

- Abilitando le asserzioni (es. –enableassertions tra gli argomenti della VM in Eclipse, Run Configurations) si ottiene:

```
Exception in thread "main" java.lang.AssertionError: denaro -1.0 < 0
        at ContoCorrente.deposita(ContoCorrente.java:6)
        at ContoCorrente.main(ContoCorrente.java:11)
```