

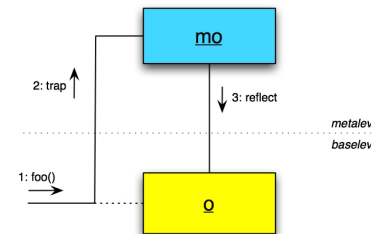
Riflessione Computazionale

- La riflessione è la capacità di un sistema (detto riflessivo) di contenere strutture che rappresentano *aspetti* di sé stesso (*metadata*) che gli permettono di supportare azioni su sé stesso
- Un sistema riflessivo è rappresentabile come un sistema a livelli in cui il livello sottostante, il *livello base*, non conosce che esistono livelli superiori, *livelli meta*
- Un livello superiore è in grado di ridefinire alcune operazioni del livello sottostante
 - Per cambiarne la natura
 - Per inserire nuove funzionalità
- Il livello superiore è capace di *intercettare* le operazioni del livello inferiore ed *ispezionare* il livello inferiore

E. Tramontana - Riflessione Computazionale - 7 Mag 10 1

Riflessione Computazionale

- La riflessione permette di costruire programmi che
 - Introspezionano strutture e dati di programmi
 - Prendono decisioni in base ai risultati dell'introspezione
 - Cambiano il comportamento, la struttura o i dati del programma in base alle decisioni prese



E. Tramontana - Riflessione Computazionale - 7 Mag 10 2

Riflessione in Java

- Java fornisce un supporto parziale per la riflessione (solo introspezione)
 - attraverso le librerie `java.lang` e `java.lang.reflect`
- Tramite tali librerie di Java possiamo
 - scoprire campi, metodi e costruttori di una classe (non noti a compile time)
 - istanziare una classe (il cui nome non è noto a compile time)
 - invocare metodi
 - ispezionare e cambiare il contenuto di campi (conosciuti solo a run-time)
 - scoprire la superclasse e le interfacce di una classe
- La classe `Object` tramite il metodo `getClass()` ritorna una istanza `c` di `Class`. Tale istanza `c` rappresenta la classe dell'oggetto su cui è invocato `getClass()`
- Tramite l'istanza `c` di `Class` possiamo avere tutte le informazioni, attraverso introspezione, della classe rappresentata

E. Tramontana - Riflessione Computazionale - 7 Mag 10 3

Esempio 1: `invoke()`

- Vogliamo invocare un metodo `show()` su una istanza di cui non conosciamo la classe (perché sviluppata da terze parti)
- Senza riflessione, dobbiamo usare un costrutto per l'invocazione per ogni specifica classe. Conseguenze:
 - Codice di invocazione dipendente dalla classe
 - Codice da modificare ogni volta che è prodotta una classe
- Con la riflessione, l'invocazione è resa indipendente dalla classe a cui il metodo `show()` appartiene

```
// cls rappresenta la classe dell'istanza o
Class cls = o.getClass();
// m rappresenta il metodo show
Method m = cls.getMethod("show", new Class[] {String.class});
// invocazione dinamica del metodo
m.invoke(o, new Object[] {msg});
...
```

E. Tramontana - Riflessione Computazionale - 7 Mag 10 4

Invocazione Dinamica

- Tramite `invoke()` riusciamo a chiamare un metodo di un oggetto a runtime
 - senza specificare a design time di quale metodo si tratti
 - Ovvero, a design time non conosciamo il nome metodo né il nome classe
- `getMethods()` (con la variante `getDeclaredMethods()`)
 - restituisce tutti i metodi pubblici di una classe
 - il valore di ritorno è di tipo `java.lang.reflect.Method`
- `getMethod()`
 - restituisce il metodo il cui nome è passato come argomento

Esempio 2: `invoke()`

- Vogliamo invocare i metodi `get*()` su una istanza di cui non conosciamo la classe e scrivere (su disco) i valori di ritorno del metodo chiamato

```
Class cls = o.getClass();

Method mt[] = cls.getDeclaredMethods();

for (int i = 0; i < mt.length; i++) {
    if (mt[i].getName().startsWith("get") )
        try {
            Object v = mt[i].invoke(o, null);
            ...
        }
}
```

Esempio `forName()`

- Vogliamo caricare una classe di cui conosciamo il nome solo a runtime (non a design time) e creare una sua istanza
 - Al fine di invocare su tale istanza un metodo

```
//carica unaa classe e ritorna la corrispondente rappresentazione
Class cls = Class.forName( nomeClasse );
```

```
// crea una istanza della classe cls
Object o = cls.newInstance();
```

```
...
```

```
// invoca un metodo dell'istanza creata
m.invoke(o, null);
```

Caricamento Dinamico

- Tramite `forName(..)` riusciamo a caricare una classe a runtime senza specificare a design time di quale classe si tratti
- Tramite `newInstance(..)` riusciamo a creare un oggetto della classe che conosciamo solo a runtime

Pro e contro

- Vantaggi nell'uso della riflessione in Java
 - Fornisce un modo per collegare ad un programma nuove classi, non conosciute a compile time
 - Permette di manipolare oggetti di una qualsiasi classe senza inserire nel codice la classe, quindi rinviando il binding fino a run-time
- Svantaggi
 - Invocare metodi o accedere a campi con i meccanismi riflessivi è molto più lento che col codice diretto

E. Tramontana - Riflessione Computazionale - 7 Mag 10 9

Metaoggetto VerboseMO

```

...
// cattura l'invocazione di un metodo
public Object trapMethodcall(int identifier, Object[] args)
    throws Throwable {
    System.out.println("* Catturato metodo: " + getMethodName(identifier) +
        "() della classe baselevel "+getClassMetaobject().getName());

    // chiama il metodo del baselevel
    Object ob = super.trapMethodcall(identifier, args);
    // il metodo a livello base è terminato

    System.out.println("* Il baselevel ha terminato l'esecuzione");
    // restituisce il parametro di ritorno
    return ob;
}
}

```

E. Tramontana - Riflessione Computazionale - 7 Mag 10 11

Metaoggetto VerboseMO

Esempio estratto dalla libreria Javassist

```

// VerboseMO è in grado di catturare le operazioni di altri oggetti

import javassist.tools.reflect.*;

public class VerboseMO extends Metaobject {
    // cattura l'istanziamento
    public VerboseMO(Object self, Object[] args) {
        super(self, args); // costruisce l'oggetto del baselevel
        System.out.println("* Costruito: " + self.getClass().getName());
    }

    // cattura la lettura di un campo
    public Object trapFieldRead(String name) {
        System.out.println("* Catturata lettura campo " + name);
        return super.trapFieldRead(name);
    }

    // cattura la scrittura di un campo
    public void trapFieldWrite(String name, Object value) {
        System.out.println("* Catturata scrittura " + name+ " valore: "+value);
        super.trapFieldWrite(name, value);
    }
}
...

```

E. Tramontana - Riflessione Computazionale - 7 Mag 10 10

Commenti al codice

- La classe VerboseMO deve essere una sottoclasse di `javassist.reflect.Metaobject` ed implementare il costruttore, ed i metodi `trapFieldRead()`, `trapFieldWrite()` e `trapMethodcall()`
- Il costruttore di VerboseMO viene invocato quando l'oggetto associato deve essere creato
- I metodi di VerboseMO ricevono il controllo dall'applicazione (ovvero catturano l'esecuzione) quando viene fatta una lettura di un campo, una scrittura o un'invocazione di metodo, rispettivamente.
- Alcuni vantaggi offerti dalla riflessione Javassist
 - Fornisce il modo di comporre vari aspetti di un'applicazione (funzionalità, sincronizzazione, distribuzione, etc.), mantenendo i codici sorgenti separati
 - La separazione tra vari aspetti di un'applicazione rende più semplice lo sviluppo, il riuso e l'evoluzione dei singoli aspetti

E. Tramontana - Riflessione Computazionale - 7 Mag 10 12