Advanced Programming

Federico Bruzzone

4 ottobre 2022

Indice

1 Informazioni generali

Scopo del corso

- Scoprire il concetto di separazione dei compiti;
- Imparare a programmare decomponendo le funzionalità del SW;
- Imparare ad ottimizzare il SW separandone le funzionalità;

Materiale di riferimento

- i licidi del corso;
- Ira R. Forman and Note B. Forman. Java Reflection in Action Manning Publications, October 2004;
- Ramnivas Laddad. AspectJ in Action: Pratical Aspect-Oriented Programming. Manning Pubblications Company, 2003;

2 Computational Reflection

2.1 Computational Reflection

2.1.1 A first definition

Computational reflection can be intuitively defined as:

"The activity done by a SW system to represent and manipulate its own structure and behavior"

The reflective activity is done analogously to the usual system activity

2.2 Reflection

2.2.1 Historical Overview

In the sisties

- Research field: artificial intelligence;
- First approaches to relection: intelligent behavior;

In the eighties

- Research filed: programming languages;

- Brian C. Smith, he introduces the reflection in Lisp (1982 and 1984), the reflective tower has been defined;
- Several reflective list-oriented languages have been defined (they exploit the quoting machanism);

In the meanwhile

- Research field: logic programming;
- the meta-programming takes place in PROLOG;

Between the eighties and the nineties

- Research fild: object-oriented programming languages;
- Pattie maes defines the computational reflection in OOPL (1987);
- Several people move from Lisp to OO:
  - P. Coite, ObjVLips (1987)
  - A. Yonezawa, ABCL-R (1988)
  - J. des Rivières e G. Kiczales MOP for CLOS (1991)
- SmallTalk is elected as the best reflective programming language

In te nineties

- Research field: typed and/or compiled object-oriented programming languages;
- Shigeru Chiba realizes OpenC++ (1993-1995), OpenJava (1999);

In the 1997

- Gregor Kiczales et al. defined the aspect-oriented programming and the story ends;

2.3 Computational Reflection

2.3.1 Reflection à la Pattie Maes

Pattie Maes has pioneered the filed

- a **computational system** is a system that can reason about and act on its applicative somain;

- a computational system is **causally connected** to its domain if and only if a change to its domain is reflected on it and vice versa;
- a **meta-system** is a computational system whose applicative domain in another computational system;
- **reflection** is the property of reasoning about and acting on itself;

therefore

- a **reflective system** is a meta system causally connected to itself;

2.3.2 Reflective system

From the definition, we can evince that a reflective system is:

- a software system logically layered into two or more levels respectively called base-level and meta-levels;
- the system running in a meta-level observes and manipulates the system running in the underlying level (reflective tower);

Characteristics

- the system running in the base-level is unaware of the existence and of the work of the systems running in the overlying levels;
- a meta-level system acts on a representation (called the system running in the underlying levels; and
- a system and its reification are causally connected and therefore, they are kept mutually consistent

2.3.3 Reflective system: Base- and Meta-levels

A meta-level system refies what it is implicit (e.g. mechanisms and structure) of the underlying base- or meta-level

2.3.4 How to Characterize a Reflective System

The reflective systems can be classified based on:

- what and when

What kind of reflective actions the system can carry out:

- structural and behavioral reflection;

- introspection (just to observe) and intercession (to alter)

When the meta-level entities exist:

- compile-time
- load-time; and
- run-time

2.3.5 Behavioral and structural reflection

The behavioral reflection allows the program of monitoring and manipulating its own computation, e.g.:

- to trap a method call and activating a different method instead;
- to monitor the object state;
- to create new objects, and so on

These activities can take place at run-time without a specific support

The structural reflection allows the program of inspecting and altering its own structure, e.g.:

- the code of a method can be modified or removed from the class;
- new methods and field can be added to a class, and so on;

These activities need a specific support by the execution environment (from the VM, RTE, ...) to be carried out at run-time

2.3.6 Reification

The base-level entities (referents) are reified into the metalevel, i.e., they have a representative into the meta-level

Such a representative, called reification, has to:

- support all the operations and have the same characteristics of the corresponding referent;
- be kept consistent to its referent (causal connection);
- be subjected to the manipulations of the meta-level entities to protect the base-level entities from potential inconsistency

Any change carried out on the reification has to be reflected on the corresponding referent.

2.4 To Develop a Reflective System

Jacques Ferber [2] has raised some issues that the developers must take in consideration:

- which kind of entities should be reified?
- what and how it is implemented the causal connection?
- when does the execution shift to the meta-level?

2.5 Which Kind of Entities Should Be Reified?

It depends on the programming language:

- functional: lambda expression/closures, environment, continuations, and so on ...;
- object-oriented: objects, methods, classes, messages and so on ...;
- concurrent and object-oriented: threads, processes, schedulers, monitors, and so on ...;
- distribution: namespaces, proxies, mailers, and so on ...

2.6 What and How It Is Implemented the Causal Connection?

It depends on when the reflective activities take place:

- atrun-time: the causal connection is explicit and must be maintained by an entities super-parties, e.g., by the virtual machine or by the run-time environment;
- at compile-time: the causal connection is implicit, base-level and meta-levels are merged together during a preprocessing phase;
- at load-time: in this case the causal connection behaves as in the case, reflection takes place at compile-time;

Most of the times, the supported reflective activity is related to observe (introspection) the base-level system so the causal connection become unilateral and can be managed by the metaentities.

2.7 When Does the Execution Shift to the Meta-Level?

Switching among levels depends on:

- which entities are reified;
- when such entities are reified; and
- how the causal connection is managed

The shift-up and-down actions

- the shift-up and-down actions.

When

- an observed element changes; or
- an action is going to be done;

the computational flow passes into the meta-level (shift-up)

Instead

- the computational flow goes back (shift-down) on the meta-level program decision

Usually, the shift-up action is managed by call-backs

3 Reflection in OO Programming Languages

3.1 Structural and Behavioral Reflection

Structural Reflection

- Object creation and init
  - constructor
  - prototype
  - meta-classes
- Class manipulation
  - to add or remove fields
  - to add or remove methods
  - to change the super class

Behavioral Reflection

- message sending
  - classes and inheritance
  - prototypes and delegation
  - errors
  - encapsulations
  - proxies
  - meta-objects

3.2 Structural Reflection

The objects running in the meta-level, called **meta-objects** are associated to all (or just to some of) the objects running in the base-level, called **referents**.

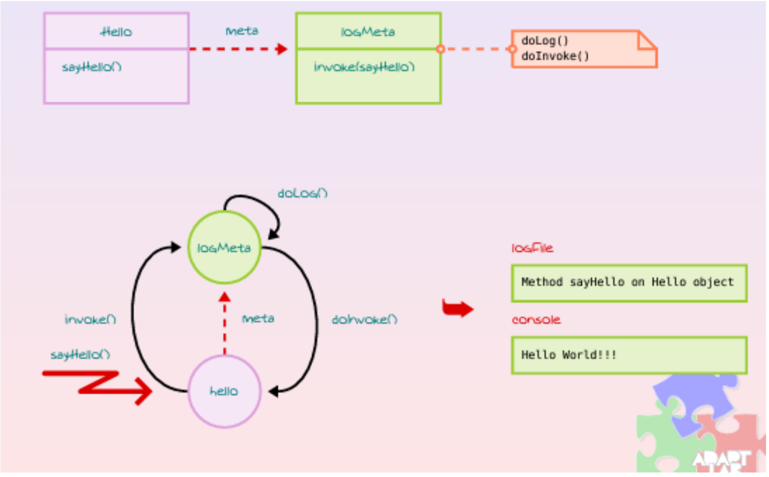
The connection among referents and meta-objects is called **causal connection** when it is a two-way link or **meta-connection** when it is a one-way link.

The meta-objects exist at run-time and extend or modify the semantics of some mechanisms:

- method invocation, field access, object creation, and so on

The **MOP** is the set of messages that a meta-object can understand

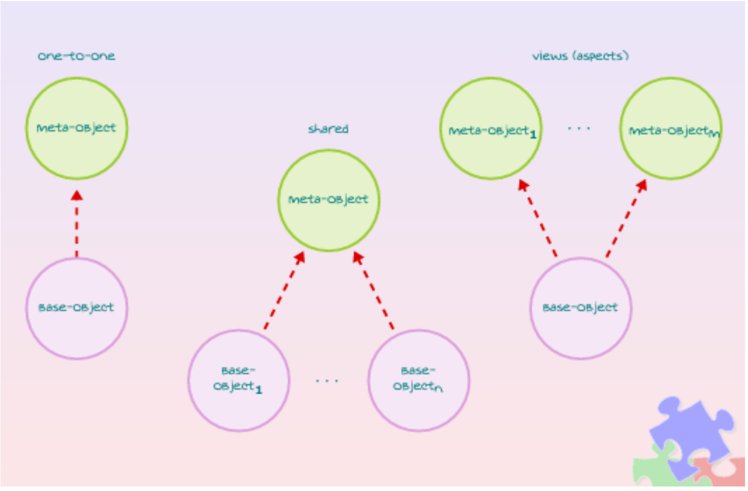
3.2.1 Es.To Enrich the Behavior of a Method Call



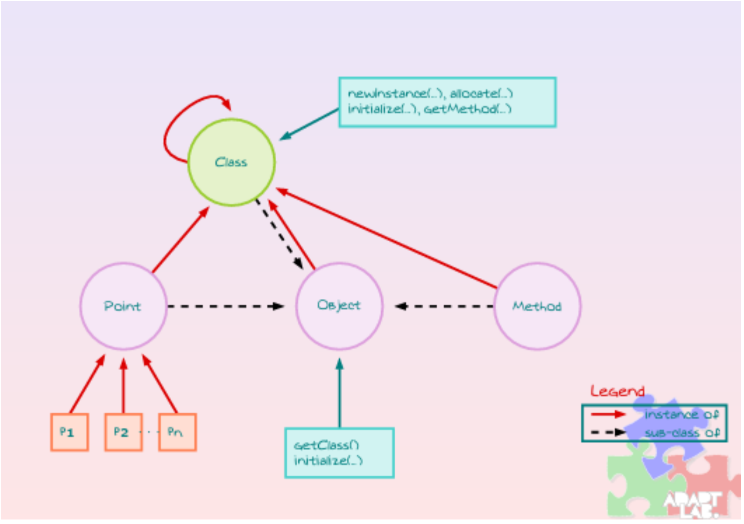
```
Method sayHello on Hello object
```

```
Hello World!!!
```

3.2.2 Different views



3.2.3 Different views


newInstance(..), allocate(..)
initialize(..), getMethod(..)
Class
Point
Object
Method
P1
P2
. . .
Pn
getClass()
initialize(..)
Legend
instance of
sub-class of