MOB2

Didier VERNA

21 mai 2018

Table des matières

1 Introduction **1**
1.1 Origine . . . 1
1.2 Caracteristiques . . . 1
1.3 Portee de l'inforamtion . . . 2
1.4 Accessibilite . . . 2
1.5 Specificite . . . 2
1.6 Rappels . . . 2

2 Ambivalence de l'heritage **2**
2.1 Polymorphisme . . . 2
2.2 Contravariance/Covariance . . . 2

1 Introduction

(Voir les slides du cours pour les infos manquantes)

Le but de ce cours est de resituer l'approche classique de mob1 en le considerant comme un cas particulier de mob2. C'est un contexte plus general.
CLOS : Common Lisp Object System
Typage dynamique + multimethodes.
Il est implemenete en lui meme → homoicomicite du langage.
La couche oriente objet de CLOS est implemente en lui meme.

1.1 Origine

Small talk est le langage oriente objet dans l'idee.
l'origine de CLOS a la meme origine. Lisp etait presque uniquement utilise dans les labos.
CLOS est a l'origine d'idees novatrices.

En 1986, l'ACM (organisation qui met en place des conferences) organise une conference Lisp. Un groupe informel met en place le debut de la standardisation de la couche objet de LISP.
CLISP a ete formalise en 94. 3 entreprises avec des dialectes de LISP differents.
Lucid - Richard Gabriel : https://en.wikipedia.org/wiki/Richard_P._Gabriel

1.2 Caracteristiques

L'approche classique d'oriente objet est l'envoi de messages.
Nous allons voir les multimethodes, a l'exterieur des classes.
Dans CLOS, il n'y a pas d'envoi de message. C'est plus harmonieux, car pas de syntaxe particuliere.
Il y a des tas d'ambiguite, liees a l'heritage multiple. Il existe dans CLOS mais d'une maniere differente de c++,

il a ete considere que le gain d'expressivite obtenue grace a l'heritage multiple, compense les ambiguites.
En Lisp, on peut reprogrammer l'algorithme de dispatch dynamique.

CLOS est implemente en lui-meme, (Meta-objet et 1ere classe).
Le typage est dynamique.
C'est une source d'expressivite en plus.

1.3 Portee de l'inforamtion

Il est tres complique quand un objet meurt, de savoir qu'il est mort a cause du garbage collector.
En Lisp, pas d'acces standard, aux slots partages entre les classes.

1.4 Accessibilite

On essaie d'avoir plus d'abstraction, on ne souhaite pas forcement connaitre les details d'implementation.
On a envie d'utiliser des accesseurs (getter/setter) pour que le jour ou l'on souhaite modifier les attributs, on puisse le faire sans impacter le code deja existant.

1.5 Specificite

C'est le fait que le typage soit dynamique, qui fait que beaucoup de chose changent, par rapport au c++ par exemple.
On retrouve des paradigmes semblables en ruby ou python.

1.6 Rappels

Copier coller c'est mal, a moins que ca soit de la generation de code.
Revoir aggregation/composition/heritage. Notamment, heritage en LISP.

2 Ambivalence de l'heritage

On herite de l'interface et de l'implementation.
Heriter de la structure et du comportement n'est pas toujours ce que l'on souhaite.

2.1 Polymorphisme

— overload (surcharge) : changer signature des methodes
— override (reecriture) : changer corps de fonction
— (masquage) : methode en cache une autre non virtuelle

Au sens de CLOS, une methode n'est pas executable en tant que tel, il faut faire appel a la classe parente pour qu'elle specialise l'appel.

2.2 Contravariance/Covariance

— Contravariance : Vous permet d'utiliser un type plus générique (moins dérivé) que celui spécifié à l'origine.
— Covariance : Vous permet d'utiliser un type plus dérivé que celui spécifié à l'origine.