Introduction à la programmation par objets avec CLOS

(motivation)

La programmation ascendente est plus facile si les modules sont organisés par types de données.

Pour améliorer la maintenabilité du code, il faut éviter la duplication de code. Il faut donc factoriser le code.

Le langages de programmation traditionnels ne peuvent pas représenter cette organisation du code.

Introduction à la programmation orientée objets

La programmation orientée objets à été conçue pour résoudre certains problèmes de modularité et de réutilisabilité dus à l'utilisation de langages traditionnels (comme le C) pour l'écriture d'applications.

Par conséquent, l'utilité de la programmation orientée objets est difficile à comprendre pour des programmeurs non expérimentés.

Pour éviter l'influence de la terminologie d'un langage particulier, nous allons commencer par une discussion générale des notions de la conception et de la programmation orientées objets.

Puis, nos montrerons la correspondance entre ces notions et l'utilisation de CLOS pour la programmation orientée objets.

Définition: objet

Un *objet* est une valeur qui peut être affectée à une variable, fournie comme argument à une fonction, ou renvoyée par l'appel à une fonction.

Un objet a une *identité*. Deux objets peuvent être comparés concernant leurs identités.

Si la *sémantique par références uniforme* est utilisée, alors cette identité est préservée par l'affectation.

Souvent, un objet est implémenté par un pointeur sur de l'espace en mémoire, mais des objets simples peuvent ne pas avoir de la mémoire allouée. L'identité et la sémantique peuvent quand même être préservées, à condition que des effets de bord ne soient pas possibles sur ces objets.

Définition: type (abstrait)

Un *type abstrait* (ou simplement *type*) est un ensemble d'objets.

Par exemple, un objet de type voiture est aussi de type véhicule.

En particulier, un type peut être un sous type d'un autre type.

Un objet peut donc être de plusieurs types.

Les types ne sont pas forcement mutuellement exclusifs.

Opération

Un *opération* est une fonction ou une procédure dont les arguments sont des objets.

Chaque argument est d'un type particulier.

Par exemple, monter-moteur peut être une opération de deux arguments, un objet de type véhicule à moteur et un objet de type moteur. L'effet de bord de cette opération est de changer le moteur du véhicule.

Protocole

Un *protocole* est une collection d'opérations utilisées ensemble.

Typiquement, au moins un type est partagé entre toutes les opérations du protocole.

Exemple: Dessiner des objets graphiques. Un tel protocole peut par exemple contenir les opérations dessiner, effacer et déplacer, utilisant des objets de type window, pixmap, cercle, rectangle, etc.

Classe

Une *classe* est utilisée pour décrire les détails d'implémentation d'objets, tels que le nombre et les noms des champs (ou créneaux).

Un type est typiquement constitué par un *ensemble de classe* qui peuvent être relatées par l'héritage (mais qui ne le sont pas forcement).

Un objet est l'*instance directe* d'exactement une classe. C'est *la classe de l'objet*.

Remarque: on peut donc parler de la classe d'un objet mais pas du type d'une objet.

Héritage

Les classes sont organisées dans un *graphe sans cycles* (*héritage multiple*) ou dans un *arbre* (*héritage simple*).

Un objet est instance de la classe C si et seulement si il est soit l'instance directe de C ou instance d'une classe qui hérite de C (ou équivalent: qui est dérivée de C).

Si un type contient la classe C, il contient aussi toutes les classe dérivées de C.

Méthode

Une *méthode* est une *implémentation partielle d'une opération*. L'implémentation est partielle, car la méthode l'implémente uniquement pour des arguments instances de certaines classes.

Pour que l'opération soit complète, il faut que les méthodes couvrent l'ensemble des classes des types des arguments.

Les méthodes d'une opération peuvent être *physiquement réparties*.

Toutes les méthodes d'une seule opération ont le même nom.

Fonctions génériques

Une *fonction générique* est une implémentation complète d'une opération.

Elle est utilisée comme une fonction ordinaire.

Une implémentation partielle (une méthode) est choisie selon la classe des arguments.

Sélection

Le mécanisme de *sélection* est responsable du choix d'une méthode particulière d'une fonction générique selon la classe des arguments.

En CLOS, la sélection est *multiple*, à savoir, plusieurs arguments sont utilisés pour déterminer la méthode choisie.

Héritage multiple

C'est la possibilité en CLOS pour une classe d'être dérivée de plusieurs autres classes.