Objets et Classes
Idée très naive

Un objet est un enregistrement dont certains champs sont des attribut et d'autres sont des pointeurs vers le code de procédures (méthodes)

(Une méthode est une procédure (ou fonction) particulière parce qu'elle a un paramètre caché, un pointeur vers l'objet)

Inacceptable parce qu'il y a énormément de duplication inutile de pointeurs vers les méthodes

Traiter les appels de méthodes comme les autres appels?

C'est-à-dire compiler un appel comme un appel d'une procédure normale.

Quand une méthode d'une classe C est appelée le compilateur sait de quelle procédure il s'agit?

Pas toujours! dans le cas d'héritage avec possibilité de (re)définir une méthode connue au niveau supérieur, un pointeur qui est (textuellement) un pointeur vers la classe initiale peut être (dynamiquement) un pointeur vers un objet de l'extension. C'est la méthode de l'extension qu'il faut appeler. Et le même code doit marcher dans les deux cas.

Y compris le pointeur implicite qui est le paramètre caché.

Solution partielle pour héritage simple un tableau des méthodes

Chaque objet contient (dans son premier mot) un pointeur vers un tableau (constant) qui dépend du type de l'objet. Ce tableau contient les pointeurs vers les méthodes valables pour ce type. Le tableau d'une extension contient des pointeurs vers les méthodes de la classe mère sauf quand ces méthodes ont une définition locale dans l'extension.

Dans une extension les nouveaux attributs [méthodes] sont placés après les anciens dans l'enregistrement [tableau des méthodes]

Coût supplémentaire mais pas très important:
espace: un petit tableau pour chaque classe et

un pointeur pour chaque objet
temps: un niveau d'indirection pour chaque appel d'une méthode (sauf optimisation...)

Mais ce système ne marche toujours pas dans le cas d'héritage multiple!
ni pour les méthodes ni pour les champs.
Parce que si une classe C est une extension de A et de B, un objet de classe C ne peut pas avoir les attributs de A et ceux de B au début.

Une solution générale

Pour chaque méthode décider une position dans le tableau qui sera valable pour toutes les classes:

inconvénients

- des trous dans les tableaux (pas très grave)
- impossible de générer du code indépendamment pour toutes les classes
- il faudra faire la même chose pour les champs: un niveau d'indirection (parce qu'il faut éviter des trous dans l'objet)
- algorithme nécessaire pour décider les positions
  (algorithme de coloration de graphes; deux

attributs/méthodes ne peuvent pas avoir la même position s'ils sont présents dans une même classe)

Une deuxième solution générale

Un objet et le tableau d'une classe sont divisé en sections, une pour chaque classe ancêtre

L'ordre est: d'abord les nouveaux attributs/méthodes et ensuite ceux hérités.

la section d'un objet contient un pointeur vers la section correspondante du tableau

Un pointeur vers un objet pointe vers la section du type du pointeur

Le tableau contient, pour chaque méthode, l'adresse de son code et un δ, qui sert à mettre à jour un pointeur

Autres opérations

- "Casts": conversion d'un pointeur d'une classe à une autre
  Opération nulle sauf dans le dernier système
  d'une extension vers une classe mère: ajout d'une constante connue au compilateur
  d'une classe vers une extension: (dangéreux) soustraction
  (l'appel d'une méthode peut contenir les deux

- Test de classe: `instanceof`
  un pointeur (textuel) vers un objet d'une classe, est-il (dynamiquement) un pointeur vers une extension?
  Associer au tableau de chaque classe un vecteur de pointeurs vers les tableaux de chaque ancêtre (selon niveau d'extension)
  le test consiste à comparer un élément de ce vecteur avec le pointeur (connu au compilateur) vers l'extension souhaitée.