Compilation des langages à objets

Représentation à l'exécution des langages orientés objets.

Pour certains langages, on doit disposer à l'exécution d'informations supplémentaires. Dans le cas des langages à objets, on doit disposer du descriptif des objets que l'on peut allouer dynamiquement (pointeur sur des structures dynamiques) et du descriptif des classes. Dans la pile, il n'y a rien de "spécial", mais il faudra penser à avoir en mémoire les informations pour le *descriptif des classes*.
La mémoire à l'exécution est organisée comme l'indique la figure suivante 1.

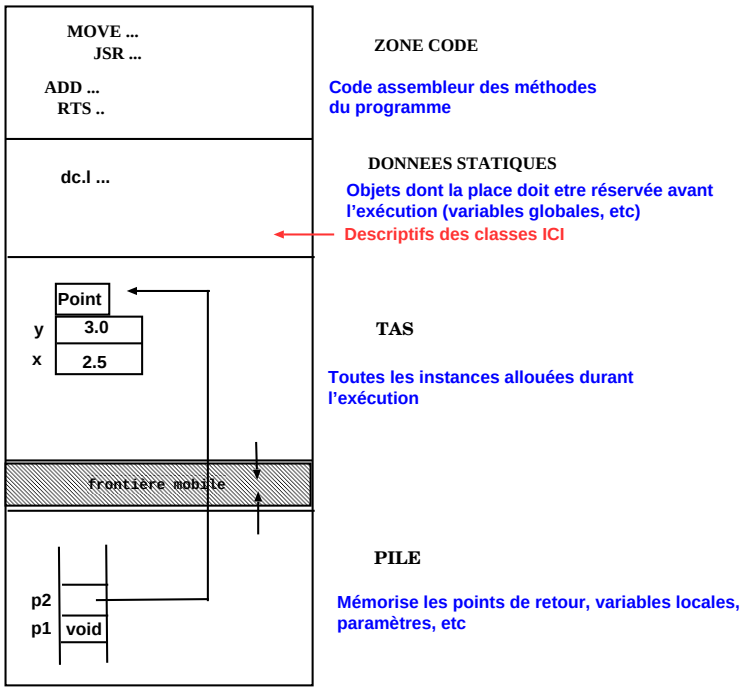

Figure 1: Mémoire à l'exécution.

Représentation des objets.

- chaque objet est une "structure" contenant autant de champs que l'objet comporte d'attributs. Il faut donc définir un déplacement associé à chaque attribut.
- il faut disposer de la taille (en octets) de l'objet pour les instanciations.

Cette information est la même pour tous les objets de la classe, elle sera stockée dans un *descripteur de classe* dont l'adresse sera rangée en début de la structure.

Descripteur de classe.

Le descripteur de classe contient toutes les informations concernant la classe :

- taille des objets
- nombre de méthodes (non obligatoire)
- adresse dans la zone de code des différentes méthodes.

Remarque : il faut définir des déplacements pour atteindre l'adresse de chaque méthode à partir du début du descripteur. La figure suivante 2 illustre la représentation d'un objet x de la classe $ARTICLE$.

```
classe ARTICLE
   // 2 attributs
   ref: INT
   prixHT : INT

   // 2 méthodes
   prixTTC() : INT
   etiqueter(ref, prix : INT)

   ...
```

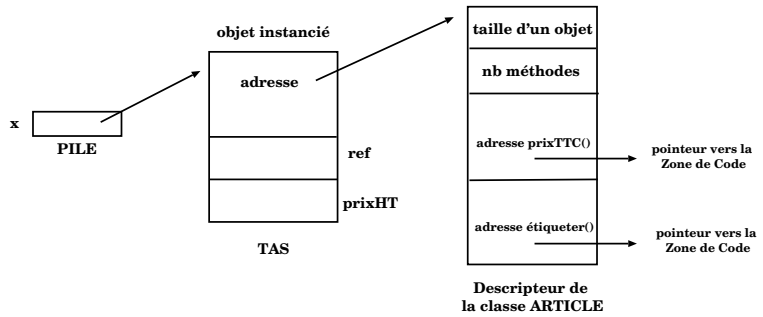

Figure 2: Représentation mémoire d'un objet de la classe ARTICLE.

Les descripteurs de classe seront donc stockés dans la zone de données statiques à une adresse globale. Ils contiennent l'information sur les classes à l'exécution.

Implantation de l'héritage simple avec/sans redéfinition de méthodes.

Il est nécessaire que :

- un attribut hérité corresponde au même déplacement par rapport au début du descripteur de l'objet, quelle que soit la classe de l'objet,
- l'adresse d'une méthode corresponde au même déplacement par rapport au début du descripteur de classe, quelle que soit la classe.

Soit la nouvelle classe $LIVRE$ suivante (avec redéfinition de méthodes) :

```
classe LIVRE herite_de ARTICLE
   // 3 attributs propres à  LIVRE
   titre: STRING
   auteur : STRING
   pages : INT

   //  méthode redéfinie
   prixTTC() : INT
   // méthode supplémentaire
   editer()

   ...
```

Ainsi, la représentation à l'exécution de $y : LIVRE$ est donnée figure 3 :

Accès à une méthode.

Le codage de l'appel à une méthode est identique à celui de l'appel d'une procédure : elle a des arguments déclarés et éventuellement un résultat, mais il faut empiler le receveur (son adresse) après les paramètres. En effet, comme les champs de l'objet sont accessibles dans le corps de la méthode, un *paramètre supplémentaire* doit être ajouté : le pointeur sur l'objet. La méthode accèdera alors aux champs de l'objets grâce à ce pointeur.

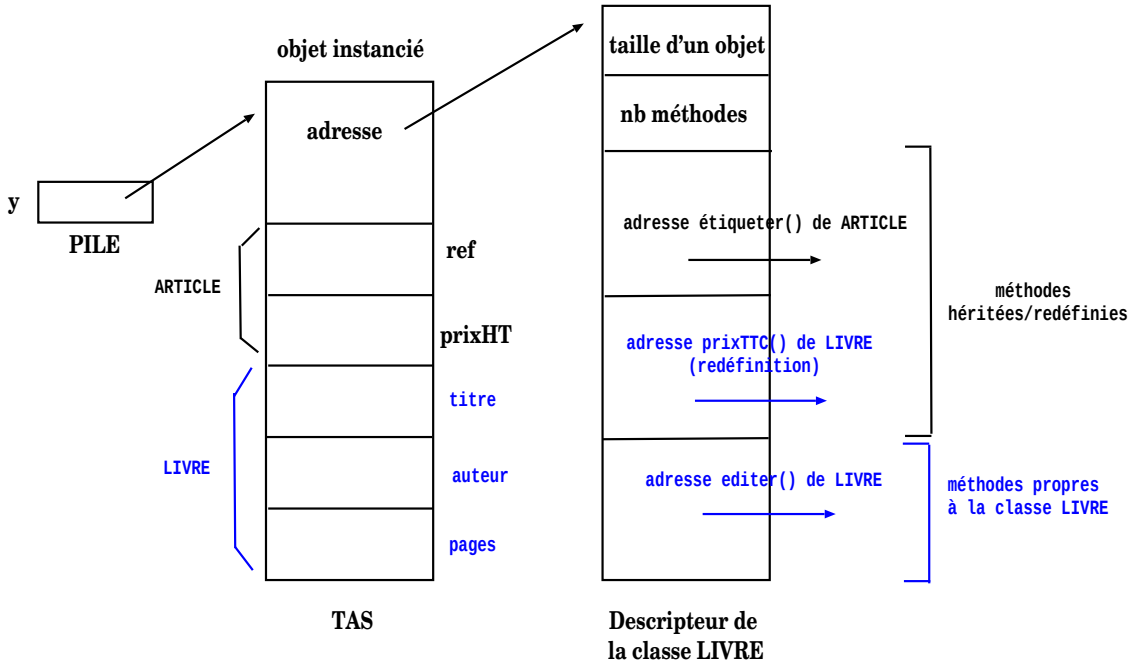

Figure 3: Représentation mémoire d'un objet de la classe LIVRE.

Synthèse : table des symboles, compilation, héritage simple.

La *table des symboles* fournit une information statique, utilisée par le compilateur :

- elle ne sert pas à l'exécution...
- elle peut contenir (entre autres) pour chaque classe déclarée les informations suivantes :
  - le nom de la classe mère (pour le typage statique),
  - le nom des attributs (ainsi que leur type) et les décalages associés,
  - le noms des méthodes applicables et les décalages associés dans le descripteur de classe.

On rappelle que pour trouver la bonne méthode à appliquer, on détermine au *typage* (c'est à dire à la compilation) la classe de l'objet. Ceci permet de connaître *statiquement* la méthode à appliquer, mais c'est la classe de l'objet au moment de *l'exécution* qui détermine dynamiquement la méthode à sélectionner. Les classes des objets jouent un rôle important dans la compilation, les objets manipulés comportent explicitement un pointeur vers le descripteur de leur classe.

Dans le cas de l'héritage simple, une classe B hérite au plus d'une autre classe A. Comme cela été présenté dans les figures précédentes,

- les méthodes compilées pour les objets de la classe A peuvent s'appliquer aux objets de la classe B,
- un objet de la classe B qui hérite de A contient d'abord les attributs de A puis ceux de B.

Pour compiler $c.f()$, il faut :

- vérifier que le receveur n'est pas à Void,
- empiler les paramètres et l'adresse du receveur,
- déterminer l'adresse de la méthode en utilisant le descripteur de la classe de c,
- appeler le code associé à $f()$, c'est à dire se brancher au $i^{ème}$ sous-programme : ceci implique que le compilateur doit connaître la valeur i. Dans le cas de l'héritage simple, pas de problème (cf. sections précédentes) et dans cas d'héritage multiple, il faut utiliser une table de déplacements (non abordé dans ce document).

Pour compiler de manière générale $x := new\ A$, il faut :

- chercher la taille T d'un objet de la classe de A (cette taille se trouve dans le descriptif de la classe A),
- allouer dans le TAS un bloc de $(T+1)$ éléments (la taille du bloc est la taille des attributs + l'adresse du descripteur de la classe A),
- ranger cette adresse obtenue dans la PILE à l'emplacement de la variable x,
- mettre à jour le pointeur vers le descripteur de classe.