TD COMPILATION — FEUILLE 6

Exercice 1. Soit les déclarations Java suivantes :

```
class A {
   int ca ;
   public int f () {ca=ca+1; return 1  ;}
   public int m () {return f() ;} }

class B extends A {
   int cb ;
   public int f () {cb=cb+1; return 2 ;} }

class C extends B {
  public int f () {return 3 ;} }
```

▷ **Question 1** *Donner les résultats produits par le main suivant :*

```
public static void main (String args[]) {
   A a = new A();
   B b = new B();
   C c = new C();
   System.out.println(a.m());
   a=c;
   System.out.println(a.m());
   a=(A)c;
   System.out.println(a.m());
   b=(B)a;
   System.out.println(b.m()); }
```

▷ **Question 2** *Dessiner la représentation des objets de classes A, B et C. Donner le code des méthodes f et m. On supposera que l'adresse de l'objet est contenu dans le registre $R1$ et que le résultat des fonctions est placé dans le registre $R0$.*

▷ **Question 3** *L'opérateur cast nécessite de disposer à l'exécution de la hiérarchie des types. Proposer une représentation à l'exécution de cette hiérarchie. Donner le schéma de traduction de l'opérateur cast (X)o, avec o déclaré de type statique O.*

Exercice 2. Soit les déclarations suivantes :

```
class A {
   public  void f() {System.out.print("1");}}
class B extends A {
   public  void g() {f() ; System.out.println("2");}}
class C extends B {
   {public  void f() {g() ; System.out.println("3");}}}
class D extends C {
   {public  void g() {f() ; System.out.println("4");}}}
class E extends A {
   {public void g() {f() ; System.out.println("5");}}}
class F extends E {
   {public void g() {f() ; System.out.println("6");}}}
```

On veut déterminer s'il est possible de décider statiquement quelle méthode est appelée. De manière générale ceci permet d'optimiser le code en remplaçant un branchement indirect par un branchement direct. Cette optimisation est particulièrement intéressante lorsqu'on cherche à paralléliser les instructions (architecture RISC avec pipeline).

▷ **Question 1** *Déterminer pour quelles méthodes il est possible de déterminer statiquement les appels pour les cas suivants :*
- *Les définitions constituent le programme en son entier (pas d'autres sous-classes)*
- *Les définitions font partie d'un programme plus large qui peut étendre toutes les classes*

▷ **Question 2** *Une optimisation complémentaire consiste à répliquer certaines méthodes. On peut par exemple construire deux instances de la méthode g définie dans la classe B : une qui appelle la méthode f de la classe A et une qui appelle la méthode f de la classe C. Utiliser cette possibilité pour affiner les résultats de la question précédente.*

Exercice 3. Soit les définitions Java suivantes :

```
class A {
   public void m (B bb, A aa) {System.out.println (" 1 ") ; }
   public A g (A aa) {... } }

class B extends A  {
   public void m (B bb, A aa) {System.out.println (" 2 ") ; }
   public void m (A aa, B bb) {System.out.println (" 3 ") ; }
   public A g (B aa) {... }
   public B g (A aa) {... } }
```

▷ **Question 1** *Lister, pour chaque déclaration de méthode, le profil de ces méthodes. Préciser pour chaque cas si on se trouve en présence d'une définition, d'une redéfinition ou d'une surcharge.*

▷ **Question 2** *Soit le main suivant :*

```
class C {
   public static void main  (String args []) {
       A a = new A() ;
       B b = new B() ;
       a.m(b, a) ;
       a.m(b, b) ;
       b.m(b, a) ;
       b.m(a, b) ;
       b.m(b, b) ; }}
```

Donner, pour chaque appel, quelle méthode est associée statiquement et dynamiquement.

▷ **Question 3** *Même question en remplaçant la classe C par :*

```
class C {
   public static void main  (String args[]) {
       A a = new A() ;
       B b = new B() ;
       A aa = b ;
       aa.m(b, a) ;
       ((B) aa).m(a, b) ;
       ((B) aa).m(aa, b) ;
       aa.m(b, aa) ; } }
```