Exercice 1.

Q1) Quels résultats fournit le programme suivant ?

```
class A {
        public void affiche() {
                System.out.print("Je suis un A ");
        }
}

class B extends A {
}

class C extends A {
        public void affiche() {
                System.out.print("Je suis un C ");
        }
}

class D extends C {
        public void affiche() {
                System.out.print("Je suis un D ");
        }
}

class E extends B {
}

class F extends C {
}

class Poly {
        public static void main(String arg[]) {
                A a = new A(); a.affiche(); System.out.println();

                B b = new B(); b.affiche(); a = b; a.affiche(); System.out.println();

                C c = new C(); c.affiche(); a = c; a.affiche(); System.out.println();

                D d = new D(); d.affiche(); a = d; a.affiche(); c = d; c.affiche();
System.out.println();

                E e = new E(); e.affiche(); a = e; a.affiche(); b = e; b.affiche();
System.out.println();

                F f = new F(); f.affiche(); a = f; a.affiche(); c = f; c.affiche();
        }
}
```

Q2) Certaines possibilités d'affectation entre objets des types classes A, B, C,D, E et F ne figurent pas dans le programme ci-dessus. Pourquoi ?

Solution:

En Java, l'une des propriétés du "polymorphisme" est que l'appel d'une méthode est déterminé au moment de l'exécution, suivant la nature de l'objet effectivement référencé (et non seulement suivant le type de la référence). C'est pourquoi ici tous les appels de affiche concernant un même objet fournissent le même message, quel que soit le type de référence utilisé :

Je suis un A
Je suis un A Je suis un A
Je suis un C Je suis un C
Je suis un D Je suis un D Je suis un D
Je suis un A Je suis un A Je suis un A
Je suis un C Je suis un C Je suis un C

//Néanmoins, une référence de type T ne peut se voir affecter qu'une référence d'un type T ou dérivé de T. C'est ce qui se passait effectivement dans notre programme. Mais (en supposant les mêmes déclarations), ces affectations seraient incorrectes :

b=a ; e=a ; e=b ; c=a ; d=c ; d=a ; f=c ; f=a ; b=c ; b=d ; b=f ; e=c ; e=d ; e=f ; c=b ; c=e ; d=b ; d=e ; f=b ; f=e ;

Exercice 3

Quels résultats fournit le programme suivant?

```
class A {
        public void f(double x) {
                System.out.print("A.f(double=" + x + ") ");
        }
}

class B extends A {
}

class C extends A {
        public void f(long q) {
                System.out.print("C.f(long=" + q + ") ");
        }
}

class D extends C {
        public void f(int n) {
                System.out.print("D.f(int=" + n + ") ");
        }
}

class F extends C {
        public void f(float x) {
                System.out.print("F.f(float=" + x + ") ");
        }

        public void f(int n) {
                System.out.print("F.f(int=" + n + ") ");
        }
}

class PolySur {
        public static void main(String arg[]) {
                byte bb = 1; short p = 2; int n = 3; long q = 4;
                float x = 5.f; double y = 6.;

                System.out.println("** A ** ");
                A a = new A(); a.f(bb); a.f(x);       System.out.println();

                System.out.println("** B ** ");
                B b = new B(); b.f(bb); b.f(x); System.out.println();
                a = b; a.f(bb); a.f(x); System.out.println();

                System.out.println("** C ** ");
                C c = new C(); c.f(bb); c.f(q); c.f(x); System.out.println();
                a = c; a.f(bb); a.f(q); a.f(x); System.out.println();

                System.out.println("** D ** ");
                D d = new D(); d.f(bb); c.f(q); c.f(y); System.out.println();
                a = c; a.f(bb); a.f(q); a.f(y); System.out.println();

                System.out.println("** F ** ");
                F f = new F(); f.f(bb); f.f(n); f.f(x); f.f(y); System.out.println();
                a = f; a.f(bb); a.f(n); a.f(x); a.f(y); System.out.println();
                c = f; c.f(bb); c.f(n); c.f(x); c.f(y);
```

```
        }
}
```

Solution:

Rappel: la signature d'une fonction est: son nom et ses paramètres.

Certaines possibilités d'affectation entre objets des types classes A, B, C, D, E et F ne figurent pas dans le programme ci-dessus. Pourquoi ?
Ici, on combine :
Les possibilités qu'offre le polymorphisme de choisir une méthode suivant la nature de l'objet effectivement référence, les possibilités de surdéfinition qui permettent de déterminer une méthode suivant le type de ses arguments.
Mais il faut bien voir que le choix d'une méthode surdéfinie est réalisé par le compilateur, alors que la ligature dynamique induite par le polymorphisme ne s'effectue qu'à l'exécution.
Plus précisément, lors d'un appel du type o.f(...), la signature de la méthode f est définie à la compilation au vu de son appel, en utilisant le type de la variable o (et non le type de l'objet référencé, non encore connu) et en appliquant éventuellement les règles de choix d'une méthode surdéfinie. Ce choix ne peut alors se faire que dans la classe de o ou ses ascendantes (et en aucun cas dans ses descendantes éventuelles, comme le permettra la ligature dynamique).
Au moment de l'exécution, on cherchera parmi la classe de l'objet effectivement référencé par o (qui peut donc éventuellement être une classe descendante de celle de o), une méthode ayant la signature précédemment déterminée. Mais, on ne reviendra plus sur le choix de la meilleure méthode.
Par exemple, dans le troisième groupe d'instructions (** C **), les appels de la forme c.f(...) sont traités en considérant les méthodes f de C et de son ascendante A. En revanche, malgré l'affectation a=c, ceux de la forme a.f(...) sont traités en ne considérant que les méthodes f de A. Ainsi, l'appel c.f(bb) utilise C.f(long) tandis que l'appel a.f(bb) utilise A.f(double).
Finalement, le programme fournit les résultats suivants :

```
** A **
A.f(double=1.0) A.f(double=5.0)
** B **
A.f(double=1.0) A.f(double=5.0)
A.f(double=1.0) A.f(double=5.0)
** C **
C.f(long=1) C.f(long=4) A.f(double=5.0)
A.f(double=1.0) A.f(double=4.0) A.f(double=5.0)
** D **
D.f(int=1) C.f(long=4) A.f(double=6.0)
A.f(double=1.0) A.f(double=4.0) A.f(double=6.0)
** F **
F.f(int=1) F.f(int=3) F.f(float=5.0) A.f(double=6.0)
A.f(double=1.0) A.f(double=3.0) A.f(double=5.0) A.f(double=6.0)
C.f(long=1) C.f(long=3) A.f(double=5.0) A.f(double=6.0)
```