Université Paris Diderot **Langages à objets avancés**
M1 **Année 2016-2017**

TP n° 7 - Héritage

Exercice 1 On suppose que dans le code suivant chaque fonction $f()$ écrite pour une classe X affichera `X::f()` lors de son exécution.

Proposez une hierarchie des classes pour que le code suivant ait le comportement décrit en commentaire. Ecrivez les, puis vérifiez.

```
cout << "---- 1 ----" << endl;
A *a=new A();
a->f(); // A::f()
a->g(); // A::g()
cout << "---- 2 ----" << endl;
A *b=new B();
b->f(); // B::f()
b->g(); // A::g()
cout << "---- 3 ----" << endl;
... *c=new C(); // le type de la variable est à compléter
c->f(); // B::f()
c->g(); // B::g()
cout << "---- 4 ----" << endl;
B *d=new D();
d->f(); // D::f()
d->g(); // D::g()
cout << "---- 5 ----" << endl;
A *e=new E(); // avec E hérite de C
e->f(); // B::f()
...e... -> g(); // ajoutez un cast de e vers B pour obtenir E::g()
```

Exercice 2

1. Créez une classe de base `Article` qui contient 2 champs, son nom et son prix, ainsi que les accesseurs idoines.
2. Créez ensuite la classe `ArticleEnSolde` qui hérite d'Article et qui contient en plus un champ remise (un pourcentage). La méthode `getPrix()` devra renvoyer le prix en tenant compte de la remise.
3. Finalement, créez la classe `Caddy` qui aura pour vocation de gérer un tableau d'Articles. Définissez entre autre la fonction `prixTotal()` qui renverra la somme des prix des articles du caddy.

Exercice 3 On considère le code suivant :

```
class A{
 public:
  void f();
  void g();
  virtual void h();
  void k(int i);
  virtual void l(A *a);
  virtual void l(B *a);
};
```

```
class B: public A{
  public:
  void f();
  virtual void h();
  void k(char c);
  virtual void l(B *a);
};
```

On suppose que le code de chacune des fonctions citées se résume à une présentation sommaire, sur le modèle :

```
void A::k(int i){
 cout << "A::k(int)"<<endl;
 }
```

Dans le `main` ci-dessous, quelles lignes ne compilent pas, et que produisent les autres. Vérifiez ensuite sur machine.

```
int main(){
  A* a = new A;
  B* b = new B;
  A* ab = new B;

  cout<< "appels de f()";
  cout <<endl;
  a->f();
  b->f();
  ab->f();

  cout<< "appels de g()";
  cout <<endl;
  a->g();
  b->g();
  ab->g();

  cout<< "appels de h()";
  cout <<endl;
  a->h();
  b->h();
  ab->h();

  cout<< "appels de k(--)";
  cout <<endl;
  a->k('a');
  b->k(2);
  ab->k('a');

  cout<< "appels de l(--)";
  cout<<endl;
  a->l(a);
  a->l(b);
  a->l(ab);
  b->l(a);
  b->l(b);
  b->l(ab);
  ab->l(a);
  ab->l(b);
  ab->l(ab);
  return 0;
}
```

Exercice 4 Soit une classe X qui contient des champs entiers x_1, x_2, x_3 qui sont respectivement public, privé et protégé. On considère également les classes Y, Z, ayant respectivement leurs trois champs y_i, et z_i publics, privés et protégés.

X est classe mère de Y, elle même mère de Z. Ces héritages sont publics.

Dans chacune des classes on redéfinit une fonction `void f(X,Y,Z)` pour tester les droits d'accès sur différents objets comme indiqué dans le `main` suivant :

```
int main(int argc, char** argv) {
 X x;
 Y y;
 Z z;
 x.f(x,y,z);
 cout << "-------" << endl;
 y.f(x,y,z);
 cout << "-------" << endl;
 z.f(x,y,z);
}
```

Le début du code des fonctions `f` est inspiré du modèle :

```
void X::f(X x,Y y,Z z){
    cout << "X::f" << endl;
    // à compléter
}
```

1. Quelles sont parmi les lignes suivantes celles que l'on peut écrire pour compléter les différentes fonctions f de X, Y et Z ? Vous les choisirez d'abord à la main. Pour simplifier la vérification sur machine utilisez netbeans qui vous indique en un clin d'oeil toutes les erreurs de compilations à l'aides d'annotations sur votre code.

```
cout << "x.x1 =" << x.x1 << endl;
cout << "y.x1 =" << y.x1 << endl;
cout << "z.x1 =" << z.x1 << endl;

cout << "x.x2 =" << x.x2 << endl;
cout << "y.x2 =" << y.x2 << endl;
cout << "z.x2 =" << z.x2 << endl;

cout << "x.x3 =" << x.x3 << endl;
cout << "y.x3 =" << y.x3 << endl;
cout << "z.x3 =" << z.x3 << endl;

cout << "x.y1 =" << x.y1 << endl;
cout << "y.y1 =" << y.y1 << endl;
cout << "z.y1 =" << z.y1 << endl;

cout << "x.y2 =" << x.y2 << endl;
cout << "y.y2 =" << y.y2 << endl;
cout << "z.y2 =" << z.y2 << endl;
```

```
cout << "x.y3 =" << x.y3 << endl;
cout << "y.y3 =" << y.y3 << endl;
cout << "z.y3 =" << z.y3 << endl;

cout << "x.z1 =" << x.z1 << endl;
cout << "y.z1 =" << y.z1 << endl;
cout << "z.z1 =" << z.z1 << endl;

cout << "x.z2 =" << x.z2 << endl;
cout << "y.z2 =" << y.z2 << endl;
cout << "z.z2 =" << z.z2 << endl;

cout << "x.z3 =" << x.z3 << endl;
cout << "y.z3 =" << y.z3 << endl;
cout << "z.z3 =" << z.z3 << endl;
```

2. Que se passe t'il si l'héritage Y de X est un héritage protected et que celui Z de Y est public ?
3. Reprenez la première question, en modifiant le graphe d'héritage : Y,Z sont deux fils de X sans que Y et Z n'héritent l'un de l'autre.