

**Rappels de cours :**

- Syntaxe pour déclarer une classe B héritant d'une classe A :

```
class B : public A {}
```

- Le constructeur de B doit alors faire appel au constructeur de A :

```
B::B(...) : A {...}, ... {  
    ...  
}
```

- Les champs/méthodes privés de A sont invisibles dans B. Les champs/méthodes **protected** de A sont visibles dans B et toutes les autres sous-classes de A.
- La liaison n'est pas dynamique par défaut : la méthode appelée correspond au type déclaré. Pour obtenir une liaison dynamique : la méthode redéfinie doit avoir été déclarée **virtual** dans la classe mère.
- Si B **redéfinit** une méthode f de A alors il est possible d'accéder à la méthode f de A via : `A::f()`
- Attention aux copies/recopies. Dans la séquence `A a; B b; a=b`, a n'est pas un B : il est modifié par appel de l'opérateur d'affectation.

**Exercice 1** Implémentez les classes suivantes.

1. Créez une classe **Article** qui contient 2 champs, son nom (**string**) et son prix (**double**), ainsi que les accesseurs utiles. Pensez à mettre **const** là où c'est utile et à proposer une méthode d'affichage couplée à une méthode **string toString() const**.
2. Testez votre classe dans un main qui devra afficher : « Parapluie, 5e ».
3. La classe **ArticleEnSolde** hérite d'article et contient en plus une remise (en pourcentage).
  - Ecrivez un constructeur **ArticleEnSolde(nom, prix, remise)**
  - Ecrivez un autre constructeur qui prenne en entrée un article et une remise. Testez!
  - Ajoutez une valeur de remise par défaut au constructeur précédent. Remarquez que vous avez maintenant un constructeur avec une signature compatible avec celle qu'on utilise d'habitude pour un constructeur de copie. Que pouvez-vous faire pour le tester ? Est ce une surcharge ? Pouvez-vous neutraliser le constructeur par copie standard ?<sup>1</sup>
  - Réécrivez l'accesseur **getPrix()** qui devra renvoyer le prix en tenant compte de la remise.
  - Vos articles soldés s'affichent-ils maintenant avec le bon prix ? (Si non, peut-être que votre méthode d'affichage d'article n'utilisait pas **getPrix()** ? N'oubliez pas également que **getPrix** doit être **virtual** pour que la liaison dynamique fonctionne!)
4. Surchargez les destructeurs pour qu'ils affichent "destruction d'article" (ou d'article en solde). Remplacez votre **main** par le suivant :

---

1. le rendre abstrait serait une approche, le déclarer delete en serait une autre

```

int main() {
    Article a1("Parapluie", 12);
    cout << a1 << endl;

    ArticleEnSolde a2("Botte", 12, 5);
    cout << a2 << endl;

    ArticleEnSolde a3(a1);

    cout << a3 << endl;

    ArticleEnSolde a4 = a1;
    cout << a4 << endl;

    return 0;
}

```

Essayez de prévoir les affichages, puis d'exécuter le code. Tout se passe-t'il comme prévu ? Si non, pourquoi ?

5. Définissez la classe `Caddie`, destinée à gérer un vecteur d'articles.
  - Munissez la classe `Caddie` d'un constructeur par défaut.
  - Ecrivez une méthode `void ajoute(Article &a)` qui prenne en argument une référence et ajoute cet article au `Caddie`.
  - Complétez par une méthode d'affichage.
  - Écrivez un test du `Caddie` dans votre `main`.
  - Une façon de définir une politique de création et destruction d'objet consiste à faire en sorte que la méthode `ajoute` fasse un clone de son argument. Comment faire pour préserver le polymorphisme ?

**Exercice 2** On suppose que dans le code suivant chaque fonction `f()` écrite pour une classe `X` affichera `X::f()` lors de son exécution.

Proposez une hiérarchie des classes pour qu'on obtienne 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 3** On considère les classes suivantes :

```
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 déclarées se résume à un affichage sommaire, sur le modèle :

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

Avec le `main` ci-dessous :

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

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

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

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

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

    cout << "Appels de l(--)" << 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;
}
```

1. indiquez quelles lignes ne compilent pas et les affichages que produisent les autres ;
2. vérifiez ensuite sur machine.

**Exercice 4** En 2112, un voyage sur Alpha du Centaure a permis de découvrir la société Centaurienne. A la différence des êtres humains les Centauriens peuvent être possiblement de trois sexes différents : Truc, Bidule et Machin. D'autre part, les Centauriens ont un nom qui ne peut être modifié. Il ont aussi un âge.

- Définissez la hiérarchie de classes correspondante. Réfléchissez à l'ensemble des attributs et opérations que l'on peut associer à un Centauren. Pensez à déclarer les opérations permettant d'afficher le nom et l'âge.
- Utilisez le modifieur `const` à chaque fois que c'est utile.
- Créez l'ensemble des constructeurs adéquats dans les différentes classes de la hiérarchie. Peut-on construire un centauren ?
- On souhaite écrire une méthode `string getSexe()` dans la classe `Centauren`. Comment la déclarez vous ?

- L'âge des Centauriens évolue bizarrement : un Centaurien ne vieillit que lorsqu'on interagit avec lui (vous lui demandez son âge et hop il prend un petit coup de vieux, vous lui demandez son nom et hop il prend un autre coup de vieux, etc)... Implémentez le mécanisme de vieillissement des Centauriens sans enlever les modifieurs **const** !