Héritage, polymorphisme, "dynamic cast" et gestion des exceptions

Les exercices sont présentés grossièrement dans un ordre croissant de difficulté. Comme vous le constaterez très souvent en Programmation, un problème peut être résolu de différentes manières. Il peut donc plusieurs solutions à un exercice. Cependant, les meilleures solutions sont souvent les plus simples, et définitivement celles qui suivent la logique du cours et des travaux dirigés.

Les exercices marqués d'une étoile (*) sont réservés aux plus avancés sur les notions de base.

Objectifs

- Savoir définir une fonction polymorphe.
- Saisir la différence entre surdéfinition et redéfinition.
- Comprendre l'intérêt du dynamic cast.
- Utiliser la gestion d'erreur en c++.

1 Polymorphisme et Héritage

EXERCICE 1 :

Une entreprise souhaite gérer les salaires de ces employés. Pour cela, elle crée une classe `Employe` ayant les caractéristiques suivantes :

- Une donnée membre string nom
- Une donnée membre string prénom
- Une donnée membre int age
- Une donnée membre int ancienneté (le nombre d'années de la personne dans l'entreprise)
- Une méthode **double calculer_base_salaire()** qui calcule la base du salaire de la personne.
- Une méthode **void afficher()** qui affiche la personne avec sa profession
- Une méthode **double calculer_salaire()** qui calcule le salaire de la personne (base de salaire plus 10% par année d'ancienneté)
- Un constructeur
- Un destructeur

  On souhaite pouvoir définir les classes dérivées d'`Employe` suivantes :
- Une classe `Commercial` ayant une donnée membre double chiffre_affaire et deux données statiques constantes double BASE et double PART. La base de salaire est donnée par la formule suivante :

$$\mathbf{BASE + chiffre\_affaire * PART}$$

— Une classe `Technicien` ayant une donnée membre int nb_unité_produite et trois données statiques constantes double BASE, double PART et double GAIN_UNITE. La base de salaire est donnée par la formule suivante :

$$\mathbf{BASE + nb\_unit\_produite * PART * GAIN\_UNITE}$$

— Une classe `Manutentionnaire` ayant une donnée membre int nb_heures et une donnée sta- tique constante double BASE_SALAIRE_HORAIRE. La base de salaire est donnée par la formule suivante :

$$\mathbf{BASE\_SALAIRE\_HORAIRE * nb\_heures}$$

1. Quelles sont les fonctions qui doivent être déclarées en tant que fonctions virtuelles dans la classe `Employe` ?
2. Quelles sont les fonctions qui doivent être déclarées en tant que fonctions virtuelles pures dans la classe `Employe` ? La classe `Employe` est-elle une classe abstraite ?
3. Implémenter les différentes classes et les tester.

EXERCICE 2 :

On souhaite créer de nouvelles classes dérivées de `Commercial` :

— `Vendeur` dont le salaire est calculé comme un `Commercial`
— `Representant` dont le salaire est calculé comme un `Manutentionnaire`

1. Que faut-il changer à la classe `Commercial` pour implémenter ces deux classes ?
2. Implémenter ces deux classes et tester-les.
3. Il-y-t-il de la duplication de code entre `Manutentionnaire` et `Representant` ? Si oui, comment y remédier ?

EXERCICE 3 :

On souhaite créer de nouvelles classes dérivées d'`Employe` correspondant aux employés avec prime de risque (un montant différent pour chaque employé qui est rajouté à la base de salaire) : `ManutentionnaireARisque` et `TechnicienARisque`.

1. Que faut-il changer aux classes `Manutentionnaire` et `Technicien` pour implémenter ces deux classes ?
2. Comment éviter la duplication de code pour la gestion des primes de risque dans ces deux classes ?
3. Implémenter ces deux classes et les tester.

*EXERCICE 4 :

On souhaite créer une classe `Personnel` contenant :

— Un vecteur extensible de pointeurs d'`Employe` (de type *vector*)
— Une méthode **void embaucher(Employe* newbie)** qui ajoute un employé
— Une méthode **void licencie(Employe* e)** qui supprime l'employé e
— Une méthode **void licencie()** qui supprime tous les employés
— Une méthode **void afficher_salaires()** qui affiche le salaire de tous les employés
— Une méthode **double salaire_moyen()** qui calcule le salaire moyen des employés

— Un constructeur
— Un destructeur

1. De quelles façons peut-on déterminer si un employé est dans le vecteur extensible de `Personnel` ? Faut-il rajouter une méthode à la classe `employe` ?
2. Implémenter cette classe et la tester.

*EXERCICE 5 :

On souhaite modifier la classe `Employe` en ajoutant des référence à un compte en banque à la classe. Une `ReferenceBanquaire` contiendra les données suivantes :
— le numéro du client qui est unique pour chaque banque.
— le nom et prénom du titulaire du compte.
— le nom de la banque.

1. Quelle sont les deux manières possibles de rajouter RéférenceBanquaire à la classe Employé ? Quelles sont les avantages et les inconvénients de chaque méthode ?
2. Implémenter la classe `ReferenceBanquaire` et l'intégrer dans la classe `Employe`.

Dynamic cast

EXERCICE 6 :

Le code suivant est à :
`http://www.labri.fr/perso/bissyand/POO_2010_2011/dynamic_cast.cpp`

```
1 #include <iostream>
2 #include <string>
3
4 class Polygone
5 {
6     public:
7         virtual std::string get_type() = 0;
8 };
9
10 class Triangle : public Polygone
11 {
12     public:
13         virtual std::string get_type() { return "Triangle"; }
14 };
15
16 class Carre : public Polygone
17 {
18     public:
19         virtual std::string get_type() { return "Carre"; };
20 };
21
22 Polygone * creer_carre_ou_triangle()
23 {
24     static int nb = 0;
25     // si nb est pair, on cree un Triangle, sinon un Carre
26     ++nb;
27     if ( nb % 2 == 0 ) { return new Triangle; }
```

```
28      return new Carre;
29  }
30
31  int main()
32  {
33      for ( int i = 0; i < 5; ++i )
34      {
35          Polygone *a = creer_carre_ou_triangle();
36          std::cout << "Test sur un " << a->get_type() << " : ";
37          Triangle *b = dynamic_cast<Triangle*>( a );
38          if ( b == 0 )
39          {
40              try
41              {
42                  Carre & c = dynamic_cast<Carre&>( *a );
43                  std::cout << "il s'agit d'un Carre.\n";
44              }
45              catch ( const std::bad_cast & )
46              {
47                  std::cout << "Oups!\n";
48              }
49          }
50          else
51          {
52              std::cout << "il s'agit d'un Triangle.\n";
53          }
54      }
55  }
```

1. Que fait l'opérateur dynamic_cast ? Quelle est la différence avec le cast statique ?
2. À quoi servent les mots-clés try et catch ?

EXERCICE 7 :

Le code suivant est à `http://www.labri.fr/perso/bissyand/POO_2010_2011/division.cpp`

```
1  #include <iostream>
2  using namespace std;
3
4  int division(int a,int b) // Calcule a divise par b.
5  {
6      if(b==0){
7          string s = "DIVISION par 0";
8          throw (s);
9      }
10     return a/b;
11 }
12
13 int main()
14 {
15     int a,b;
16     std::cout << "Valeur pour a : "<< std::flush;
17     std::cin >> a;
18     std::cout << "Valeur pour b : "<< std::flush;
19     std::cin >> b;
20     int c;
21     try{
```

```
22      c = division(a, b);
23      std::cout << a << " / "<<b << " = " << c << std::endl;
24      }
25      catch (const string &s ){
26          cout << "Exception recue :" << s << endl;
27      }
28      return 0;
29  }
```

1. Que se passe-t-il si l'utilisateur entre 0 pour b ?
2. On souhaite gérer la division par zero avec une exception :
   — Lancer une exception avec le mot-clé throw de type chaîne de caractère std :string : :("ERREUR : Division par zéro !")
   — Utiliser le mot-clé try pour le code de la fonction main correspondant à la division.
   — Récupérer l'erreur en utilisant le mot-clé catch et traiter-là.

*EXERCICE 8 :

On souhaite pouvoir définir des classes `SequenceAcidesAmines`, SequenceADN, SequenceARN. Pour cela, on a besoin d'implémenter une classe `Alphabet` qui va nous permettre de définir proprement les lettres autorisées dans les séquences (A, C, G et T pour SequenceADN et A, C, G et U pour SequenceARN). La classe Alphabet contient :

— Un constructeur auquel on passe une chaine de caractères. Chaque lettre différente dans la chaine de caractère appartient à l'alphabet.
— Une méthode **bool is_in_alpha(char c)** qui renvoie true si c est dans l'alphabet
— Une méthode **int nb_of_elt** qui renvoie le nombre d'éléments de l'alphabet

1. De quelles données membres a-t-on besoin dans la classe `Alphabet` ?
2. Implémenter la classe `Alphabet` et la tester.
3. On souhaite maintenant que le constructeur d'`Alphabet` lève une exception quand la chaine de caractère passée en paramètre contient deux fois la même lettre. Ajouter cette exception à la classe.
4. Implémenter les trois classes `SequenceAcidesAmines`, `SequenceADN` et `SequenceARN`. Les constructeurs de ces classes prennent une chaine de caractère en paramètre. Si la chaine contient des caractères non-autorisés par la séquence, le constructeur doit lever une exception.
5. Implémenter des fonctions de traduction entre les différents type de séquences. Dans le cas de la traduction d'ARN en acides aminés, la fonctions doit les lever deux exceptions suivantes :
   — no_gene si aucun codon start n'est trouvé,
   — incomplete_gene si un codon start est trouvé mais pas de codon stop.