Examen Licence Informatique - L3

Programmation par objets 2 (ULIN 606)

Session 1 - mai 2007

Tous documents autorisés

1 Généricité paramétrique

Soit la classe partielle `Armoire` suivante dans une version spécifique destinée à ranger des vêtements.

```
class Armoire{
private:
vector<Vetement*> contenu;
public:
Armoire();
virtual bool contient(Vetement* v);
};

Armoire::Armoire(){}
bool Armoire::contient(Vetement* v)
{return find(contenu.begin(),contenu.end(),v)!=contenu.end();}
```

Question 1.1 *Proposez une version générique de la classe* `Armoire` *paramétrée par le type des objets rangés à l'intérieur. Instanciez-la dans un programme pour obtenir une armoire contenant de la vaisselle et appelez la méthode* `contient`*.*

2 Héritage multiple

Question 2.1 *Faites un schéma de la hiérarchie d'héritage correspondant au programme ci-dessous. Dans ce schéma il vous est demandé de faire figurer les liens d'héritage (y compris d'éventuels liens transitifs si l'exemple le veut), mais également l'ordre de déclaration des superclasses de chaque classe afin de faciliter l'application de l'algorithme.*

```
class Vin{
 private:
  string couleur;
  string nom;
 public:
  Vin(){cout << "Vin " << endl;}
  virtual ~Vin(){}
  virtual void copieCaracteristiques(string&)const{cout <<  "copie caracteristiques " << endl;}
};

class Vin_Invention : virtual public Vin{
 public:
 Vin_Invention(){cout << "Vin_Invention " ;saisieCaracteristiques(cin);}
 virtual ~Vin_Invention(){}
 virtual void saisieCaracteristiques(istream& is){cout <<  "saisie caracteristiques " << endl;}
};

class Vin_Supervise : virtual public Vin{
 private:
static string Historique;
 public:
 Vin_Supervise(){cout << "Vin_Supervise " ;copieCaracteristiques(Historique);}
 virtual ~Vin_Supervise(){}
};

string Vin_Supervise::Historique="-";
```

```
class Bourgogne  : virtual public Vin_Invention, virtual public Vin_Supervise {
public:
Bourgogne(){cout << "Bourgogne " << endl;}
virtual ~Bourgogne(){}
};

class Cotes_De_Beaune : virtual public Vin_Supervise, virtual public Bourgogne{
public:
Cotes_De_Beaune(){cout << "Cotes_De_Beaune " << endl;}
virtual ~Cotes_De_Beaune(){}
};

int main(){
 cout << " ------------------------- " << endl;
  Bourgogne instanceOfBourgogne;
  cout << endl;
  cout << " ------------------------- " << endl;
  Cotes_De_Beaune instanceOfCotes_De_Beaune;
  cout << endl;
}
```

Question 2.2 *Montrez ce qu'affiche ce programme en expliquant l'algorithme que vous avez déroulé pour connaître l'ordre d'appel des constructeurs. Le graphe des liens d'héritage peut contenir des arcs de transitivité qui peuvent être empruntés lorsque vous appliquez l'algorithme.*

Question 2.3 *Comparer les ordres de construction de* `instanceOfBourgogne` *et* `instanceOfCotes_De_Beaune`*. Quelle différence y voyez-vous et quelles peuvent en être les conséquences ?*

Question 2.4 *Faites un schéma mémoire des deux instances dans le cas de l'héritage virtuel (correspondant au programme actuel)*

Question 2.5 *Que changeriez-vous à ce programme pour faire de l'héritage « répété » ? Quel est alors le schéma mémoire de* `instanceOfBourgogne` *et qu'en pensez-vous ?*

3 Cave vinicole

Nous étudions quelques éléments simplifiés de modélisation et de programmation pour un système de gestion de cave vinicole.

Les *cépages* sont décrits par un nom, une région et le fait d'être agréés pour leur utilisation dans les vins de qualité AOC. Ils disposent de deux méthodes :

- `string couleur()`, abstraite dans le cas général, qui retourne la couleur du cépage,
- `string toString()` retournant une chaîne contenant le nom, la région et la couleur du cépage.

Les *cépages* sont classés dans deux catégories suivant leur couleur :

- *cépages rouges* décrits en plus par leur taux en tanins, qui est ajouté à la chaîne retournée par la méthode `toString`,
- *cépages blancs*.

La couleur doit être stockée sous forme d'une variable `static` dans les classes représentant les cépages rouges et les cépages blancs. Chacune de ces deux classes contient sa propre implémentation de la méthode `string couleur()`.

Un vin est un assemblage de plusieurs cépages, il a une dénomination et une qualité (AOC, AOVDQS, pays, table). On dispose pour les vins de deux méthodes :

- `string toString()` retournant une chaîne contenant le nom et la liste des cépages,
- `ajoute(c :Cepage)` qui ajoute un cépage à un vin.

La figure 1 vous propose un schéma possible pour ces classes que nous vous invitons à respecter pour le reste de l'examen.

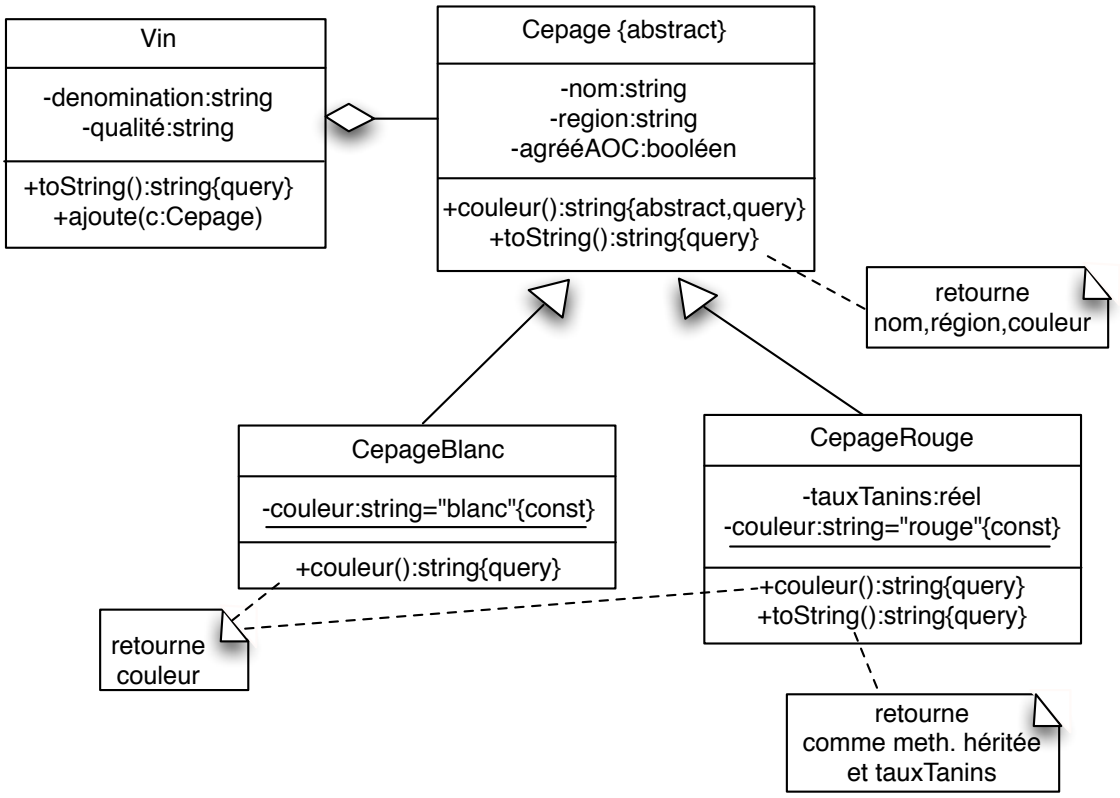

FIG. 1 – Une solution de conception pour la cave vinicole

Question 3.1 (Premiers éléments des classes)
Utilisez la librairie STL pour implémenter les collections qui vous seront nécessaires dans le code C++, `vector` *est un bon candidat pour une implémentation simple.*

Ecrivez partiellement en C++ les classes représentant les cépages (`Cepage`,`CepageRouge` *et* `CepageBlanc`*) et les vins.*

N'écrivez dans cette question QUE :

- *l'entête des classes (dont les noms, liens de spécialisation, etc.),*
- *les attributs,*
- *les constructeurs (justifiez leur implémentation et le choix ou non de faire un constructeur par copie),*
- *les destructeurs (justifiez leur implémentation).*

Question 3.2 (toString)
Ecrivez, pour les classes représentant les cépages et les vins, la méthode `toString`*. Proposez ensuite une surcharge de l'opérateur d'insertion dans un flot pour afficher les cépages.*

Question 3.3 (ajout)
Nous étudions l'écriture de la méthode `ajoute(c :Cepage)` *pour la classe* `Vin`*.*
(a) Proposez une classe d'exceptions pour représenter l'un des problèmes pouvant survenir lors de l'ajout, et qui se produit lorsque le vin est un AOC, mais que le cépage c ajouté n'est pas agréé pour les AOC.
(b) Ecrivez la méthode `ajoute` *en prévoyant le signalement d'exception approprié.*
(c) Ecrivez également un petit programme qui se préoccupe de capturer l'exception liée à un ajout.