TP 9 : Cocktail

notions : Création de classes et d'objets simples, attributs et méthodes, encapsulation

1 La classe Verre

Dans un fichier d'en tête `verre.h` (fourni sur le site WWW du module) et un fichier source `verre.cpp`, écrire une classe `Verre` ayant deux attributs entiers contenance et quantité ainsi que les méthodes :

- `int remplir(int q)` qui ajoute `q` cl de liquide à la quantité contenue dans le verre en saturant le verre à sa contenance maximale, et retourne la quantité effectivement ajoutée.
- `int boire(int q=-1)` qui extrait au maximum `q` cl du verre et retourne la quantité de liquide qui a pu être bue. Si le paramètre est négatif, la totalité du contenu est bue. Si le verre est vide, un message est affiché.
- `int getVolumeVide()` qui retourne le volume vide restant disponible.
- `std::string toString()` qui retourne une description textuelle de l'objet. On utilisera la classe `stringstream` de la bibliothèque `sstream`.

Le fichier header verre.h vous est donné ; à vous de créer le fichier source .cpp. Notez qu'en C++ l'inclusion de <`algorithm` >(#`include`<`algorithm`>) vous donne accès à une fonction `min(int, int)`.

2 Un programme qui utilise un Verre

Dans un fichier `testverre.cpp`, écrire un programme principal qui :

- crée un verre de contenance 20cl ;
- remplit le verre avec 10 cl, puis affiche le verre ;
- boit le verre, puis affiche le verre ;
- remplit le verre avec 15 cl, puis affiche le verre ;
- remplit le verre avec 10 cl supplémentaires, puis affiche le verre ;
- boit 10 cl du verre, puis affiche le verre ;
- boit 15 cl du verre, puis affiche le verre.

3 La classe Bouteille

Dans un fichier d'en tête `bouteille.h` et un fichier `bouteille.cpp`, écrire une classe `Bouteille` ayant un attribut chaine de caractères nom (`std::string`, qui décrit le liquide contenu dans la bouteille), un attribut entier quantité et un attribut booléen indiquant si la bouteille est ouverte, ainsi que les méthodes :

- `void ouvrir()` et `void fermer()`.
- `int verser(int q)` qui vide la bouteille de `q` cl — ou moins si la bouteille contient moins de q cl, auquel cas un message est affiché — et retourne la quantité effectivement versée. Si la bouteille est vide, un message est affiché. Attention, la bouteille doit être ouverte ; on affichera un message sinon.

— `int verser(Verre * pv, int q)` qui verse la quantité `q` de liquide de la bouteille dans le verre pointé par pv. Comme on ne veut pas perdre du précieux liquide, on vérifiera quelle quantité le verre est capable de recevoir avant de verser. Attention, la bouteille doit être ouverte ; on affichera un message sinon.
— `std::string toString()` qui retourne une description textuelle de l'objet (cette méthode est automatiquement appelée lorsque l'objet est affiché).

4 Un programme avec un verre et une bouteille

Dans un fichier `test.cpp` écrire un programme principal qui :
— crée une bouteille de jus d'orange, fermée, de 1 litre ;
— crée une bouteille de vodka, fermée, de 1 litre ;
— crée un verre de contenance 20cl ;
— essaye de verser 8cl de vodka dans le verre ;
— affiche les bouteilles et le verre.
— ouvre les bouteilles ;
— verse 6cl de vodka ;
— verse 14cl de jus d'orange dans le verre ;
— affiche les bouteilles et le verre.
— boit le verre en 2 fois ;
— affiche les bouteilles et le verre.

5 Compléments

Le programme ainsi écrit fonctionne, mais ne répond pas encore aux normes de développement en C++.

— ajouter au moins un constructeur à la classe `Verre`. Le verre est initialement vide.
— ajouter un ou plusieurs constructeur(s) à la classe `Bouteille`. La bouteille est initialement fermée. La quantité contenue vaut 100 par défaut, ou est précisée lors de la construction.
— est-il nécessaire de définir un destructeur dans ces classes ? En ajouter un malgré tout, ne serait-ce que pour écrire quelque chose dans le Terminal. Observez à quel moment les objets sont détruits.
— rendre les attributs des classes privés (encapsulation). Ceci permet que les classes garantissent les invariants de classe.
— ajouter des accesseurs permettant de consulter l'état de l'objet. Penser à utiliser le qualificateur `const` pour ces méthodes (un accesseur ne modifiant pas l'état de l'objet).
— ajouter divers modifieurs.

Viser la complétude de l'interface. Par exemple :
— il est utile de pouvoir accéder aux quantités contenues en cl, mais aussi en pourcentage de remplissage, etc.
— il est utile de pouvoir modifier les objets de différentes manière (e.g. : surcharge de la méthode verser() de Bouteille pour la verser "par terre", ou pour la verser dans un verre en le remplissant au maximum possible, etc.)

6 Pour ceux qui ont pris de l'avance : une classe `Bar`

Dans notre petit exercice, une bar sera défini par son nom et une collection de de 10 emplacements pouvant chacun contenir un pointeurs sur bouteille. Les 10 emplacements seront représentés par un tableau de 10 pointeurs-sur-bouteilles.

Ecrire la classe `Bar` correspondant aux spécifications suivantes.

6.1 Attributs

La classe `Bar` sera dotée des attributs suivants (privés bien sûr...) :
— `std::string nom` : le nom du bar
— un tableau `tab` de 10 pointeurs sur bouteilles. L'élément d'indice `i` de ce tableau représente le i-ième emplacement du bar. Si `tab[i]` vaut `NULL`, cela signifie que l'emplacement est vide. Sinon (c'est à dire : si tab[i] pointe un objet de type bouteille) c'est qu'il contient une bouteille. En C++, cet attribut sera donc déclaré ainsi : `Bouteille*  tab [10] ; .`

6.2 Méthodes

Définissez les méthodes suivantes :
— Le contructeur public `Bar(std::string nom)` qui initialise l'objet `bar`. Le bar est initialement vide (pas de bouteille, toutes les cases du tableau doivent être initialisées à `NULL`).
— Les accesseurs publics `std::string getNom()` qui retourne le nom du bar et `int getNbEmplacements()` (qui retourne donc 10 dans notre cas).
— L'accesseurs public `int nbBouteilles()` qui retourne le nombre de bouteilles contenues dans le bar (nombre de cases non `NULL` du tableau).
— la méthode `void ranger(Bouteille * b, int numEmplacement)` qui range la bouteille `b` dans l'emplacement numéro `numEmplacement`. Si l'emplacement n'existe pas ou contient déjà une bouteille, il s'agit d'une erreur...
— la méthode publique `Bouteille * prendre(int numEmplacement)` qui retourne la bouteille de l'emplacement `numEmplacement` et vide l'emplacement (sa valeur passe à `NULL`). Si l'emplacement n'existe pas ou ne contient pas de bouteille, la méthode retourne `NULL`.
— la méthode publique `Bouteille * prendre(std::string nom)` qui recherche et "prend" dans le bar une bouteille nommée `nom` (une bouteille de "coca" par exemple). La méthode retourne NULL si le bar ne contient pas de bouteille du `nom` voulu.
— `public std::string toString()` qui retourne une représentation textuelle du bar, en listant tous les emplacements et le détail des bouteilles.

6.3 Le programme de test, la suite

Modifier le programme de test pour :
— créer le bar nommé "OMaitre" ;
— ranger la bouteille de coca et la bouteille de whisky dans des emplacement ;
— prendre les bouteilles dans leurs emplacements avant de remplir le verre, puis les ranger à nouveau dans le bar ;
— afficher le bar à différentes étapes.