TP POO C++ : Introduction

TP POO C++ : Introduction

Les objectifs de ce TP sont de découvrir la programmation orientée objet. Certains exercices sont extraits du site www.france-ioi.org.

Les critères d'évaluation de ce TP sont :

- le minimum attendu : on doit pouvoir fabriquer un exécutable et le lancer !
- le programme doit énoncer ce qu'il fait et faire ce qu'il énonce !
- le respect des noms de fichiers
- le nommage des classes, objets, attributs et méthodes
- une "chose" doit énoncer ce qu'elle fait et faire seulement ce qu'elle énonce !
- le corps des méthodes ne dépassent pas 15 lignes
- le code est fonctionnel
- la simplicité du code

Exercices : un long voyage

Votre voyage se passe bien mais il vous reste encore de longs jours de marche avant d'arriver à votre destination. Courage !

☞ Objectif : vous allez découvrir les notions de classes et d'objets en C++.

Recette secrète

Vous voici arrivé(e) tout en haut de la montagne. Vous allez enfin pouvoir libérer le chef du village ! Vous tombez des nues lorsque vous l'apercevez en train de discuter tranquillement avec le Grand Sorcier. Loin de s'être fait kidnapper par ce dernier, il l'a rejoint pour préparer une mixture en vue de la célébration qui a lieu dans quelques jours.

La mixture en question est composée de trois ingrédients à mélanger en proportions parfaitement exactes : 5 volumes d'huile, 4 volumes d'eau, et 3 volumes d'un ingrédient secret. Le chef et le Grand Sorcier disposent de deux tonneaux non gradués de contenances **5 litres** et **3 litres**, avec lesquels ils pourront facilement doser l'huile et l'ingrédient secret. Mais il leur manque le tonneau de 4 litres car le chef l'a oublié au village !

Si l'on transfère le contenu d'un tonneau dans l'autre, jusqu'à avoir vidé le premier ou rempli le second, par le calcul, on peut savoir précisément combien d'eau se trouve dans chacun des deux tonneaux. Ainsi, vous vous dites qu'il doit bien y avoir un moyen d'utiliser les tonneaux disponibles pour mesurer exactement 4 litres d'eau. Vous utilisez votre robot pour chercher la solution.

☞ Ce que doit faire votre programme :

Vous vous trouvez devant une source d'eau qui jaillit de la montagne, et **vous disposez de deux tonneaux vides de capacités 5 litres et 3 litres**. Écrivez un programme qui effectue une série de transvasements permettant d'obtenir exactement 4 litres d'eau dans le plus grand tonneau.

Vous disposez d'une classe `Tonneau`. Quand on instancie un objet de cette classe, il faut préciser en paramètre du constructeur sa contenance en litres. Une fois créé, le tonneau est vide.

```
// Ce programme instancie un tonneau de 2 litres
#include <iostream>
#include "tonneau.h"

using namespace std;

int main()
{
    Tonneau tonneau(2); // un tonneau de 2 L

    cout << "Tonneau " << tonneau.contenance() << "L : " << tonneau3.quantite() << endl;
    // Affiche : Tonneau 2L : 0

    return 0;
}
```

Instancier un tonneau

Pour doser l'eau dans les tonneaux, vous disposez de ces trois instructions :

- Remplir tonneau
- Transférer tonneauSource → tonneauDestination
- Vider tonneau

Quand on transvase un tonneau dans l'autre, on s'arrête lorsque le tonneau source est vide ou lorsque le tonneau destination est plein à ras bord. Ainsi, après chaque opération, on peut savoir exactement combien de litres d'eau se trouvent dans les deux tonneaux.

En C++, les trois instructions s'écrivent comme suit :

```
// Ce programme manipule des objets Tonneau
#include <iostream>
#include "tonneau.h"

using namespace std;

int main()
{
    Tonneau tonneau1(2); // un tonneau de 2 L
    Tonneau tonneau2(2); // un tonneau de 2 L

    tonneau1.remplir(); // on remplit le tonneau1

    tonneau2.remplir(tonneau1); // on transvase le tonneau1 dans le tonneau2

    tonneau2.vider(); // on vide le tonneau2

    return 0;
}
```

☞ Exemples :

Arrêtez-vous dès que le grand tonneau de 5 litres contient exactement 4 litres.

```
$ make

$ ./recette-secrete
Tonneau 5L : 4 l
```

Question 1. Écrire le programme `main.cpp`. Tester et valider.

Mastermind

Vous vous octroyez un petit moment de détente et décidez de programmer une partie de *Mastermind* avec votre robot.

Le *Mastermind* est un jeu de société pour deux joueurs dont le but est de trouver un code. C'est un jeu de réflexion, et de déduction, inventé par Mordecai Meirowitz dans les années 1970 alors qu'il travaillait comme expert en télécommunications. Il se présente généralement sous la forme d'un plateau perforé de 10 rangées de quatre trous pouvant accueillir des pions de couleurs.

Le nombre de pions de couleurs différentes est de 8 et les huit couleurs sont généralement : rouge ; jaune ; vert ; bleu ; orange ; blanc ; violet ; fuchsia. Il y a également des pions blancs et rouges (ou noirs) utilisés pour donner des indications à chaque étape du jeu.

Un joueur commence par placer son choix de pions sans qu'ils soient vus de l'autre joueur à l'arrière d'un cache qui les masquera à la vue de celui-ci jusqu'à la fin de la manche. Le joueur qui n'a pas sélectionné les pions doit trouver quels sont les quatre pions, c'est-à-dire leurs couleurs et positions. Pour cela, à chaque tour, le joueur doit se servir de pions pour remplir une rangée selon l'idée qu'il se fait des pions dissimulés.

Une fois les pions placés, l'autre joueur indique :

- le nombre de pions de la bonne couleur bien placés en utilisant le même nombre de pions rouges ;
- le nombre de pions de la bonne couleur, mais mal placés, avec les pions blancs.

Il arrive donc surtout en début de partie qu'il ne fasse rien concrètement et qu'il n'ait à dire qu'aucun pion ne correspond, en couleur ou en couleur et position. La tactique du joueur actif consiste à sélectionner en fonction des coups précédents, couleurs et positions, de manière à obtenir le maximum d'informations de la réponse du partenaire puisque le nombre de propositions est limité par le nombre de rangées de trous du jeu.

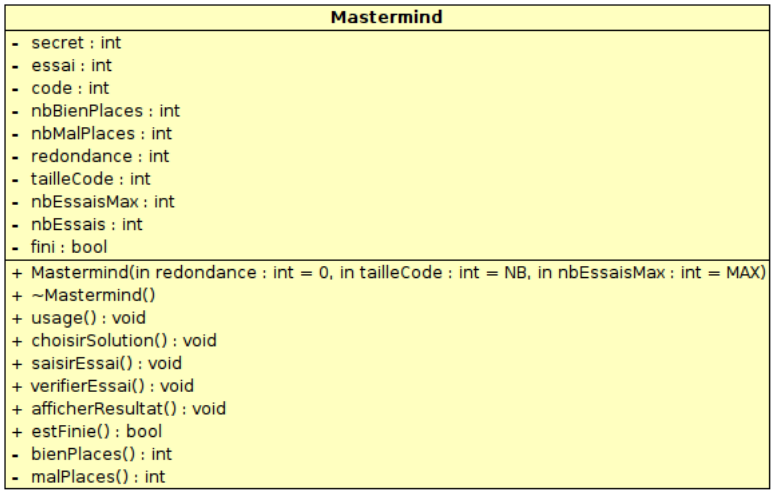
☞ Ce que doit faire votre programme :

Les couleurs seront remplacées par des chiffres de 1 à 8. Dans ce jeu de société pour deux joueurs, on distinguera les rôles :

- le joueur qui devra deviner la combinaison secrète (couleur et position des pions) ;

– la “machine” qui assurera logiciellement le déroulement d’une manche en établissant la combinaison secrète puis en déterminant le nombre de pions de la bonne couleur bien placés et mal placés par rapport à la proposition indiquée par le joueur “humain” à chaque tour.

On vous fournit la classe `Mastermind` qui implémente le jeu :



La déclaration de cette classe se trouve dans le fichier en-tête (*header*) `Mastermind.h`. Cette classe respecte le principe d’encapsulation : l’ensemble des attributs sont déclarés privés (`private`, indiquées par un `-` dans le diagramme ci-dessus). Les méthodes publiques sont repérées par un `+`.

✍ *Les méthodes peuvent aussi être déclarées privées (c’est le cas de* `bienPlaces()` *et* `malPlaces()` *qui sont à usage interne à la classe).*

Voici une brève description de ses méthodes publiques :

– `usage()` : affiche un texte informatif sur le jeu ;
– `choisirSolution()` : détermine aléatoirement la combinaison secrète et initialise une nouvelle manche ;
– `saisirEssai()` : assure la saisie d’une combinaison proposée par le joueur ;
– `verifierEssai()` : vérifie si la combinaison proposée par le joueur correspond à la combinaison secrète et détermine le nombre de pions de la bonne couleur bien placés et mal placés ;
– `afficherResultat()` : affiche l’état du tour (le numéro de tour, le nombre de tours restant et le nombre de pions de la bonne couleur bien placés et mal placés) et de la manche si elle est finie (en dévoilant la combinaison secrète en cas de défaite du joueur) ;
– `estFinie()` : retourne vrai (`true`) si la manche est finie sinon faux (`false`).

La classe `Mastermind` possède un **constructeur** qui permet de paramètrer le type de manche à jouer en indiquant :

– si la redondance des couleurs est admise dans le code ;
– le nombre de pions dans le code ;
– le nombre d’essais maximum pour deviner la combinaison secrète en une manche.

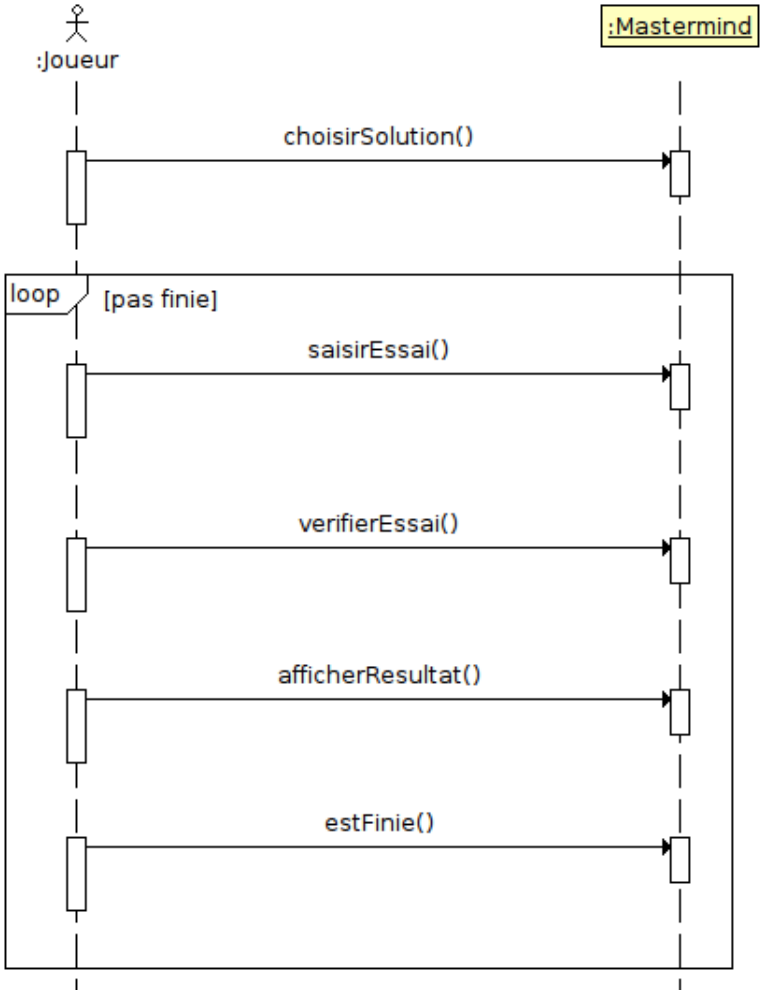
```
// Valeurs par défaut
#define NB 4   // nombre de pions dans le code secret
#define MAX 10 // nombre d'essais maximum
#define REDONDANCE 1 // la redondance des couleurs dans le code est admise

Mastermind(int redondance=0, int tailleCode=NB, int nbEssaisMax=MAX);
```

La déclaration du constructeur de la classe Mastermind

Le constructeur par défaut initialisera donc une manche avec 4 pions de couleurs uniques à deviner en 10 tours maximum.

Le diagramme de séquence pour “jouer une manche” est le suivant :



☞ *Un diagramme de séquence est un diagramme d'interaction. Le but est de décrire comment les objets collaborent au cours du temps et quelles responsabilités ils assument. Il décrit un scénario d'un cas d'utilisation ou le cas d'utilisation lui-même. Un diagramme de séquence représente donc les interactions entre objets, en insistant sur la chronologie des envois de message. C'est un diagramme qui représente la structure dynamique d'un système car il utilise une représentation temporelle. Les objets, intervenant dans l'interaction, sont matérialisés par une « ligne de vie », et les messages échangés au cours du temps sont mentionnés sous une forme textuelle. Les messages sont des méthodes d'une classe et l'envoi d'un message correspond donc à l'appel de cette méthode.*

Le diagramme de séquence ci-dessus utilise un **fragment combiné** de type *loop* (une boucle) qui permet de répéter le fragment tant que la condition indiquée entre crochets (`[]`) est vérifiée. Évidemment, il existe d'autres type de fragments combinés comme *alt* (alternatif) qui est équivalent à la structure conditionnelle “`SI ... ALORS ... SINON ... FSI`”.

☞ Exemples :

Avant de jouer la partie, il est conseillé d'afficher les règles du jeu.

```
$ make

$ ./mastermind
Le Mastermind est un jeu de société dont le but est de trouver un code secret.
```

```
C'est un jeu de réflexion et de déduction, inventé par Mordecai Meirowitz dans les années
    1970.
Le nombre de couleurs différentes est de 8 (de 1 à 8).
Vous avez 10 essais pour deviner un code de 4 pions de couleurs strictement différentes.

Entrez votre essai : 1 2 3 4
Essai 1/10 : bien places 1, mal places 1
Entrez votre essai : 1 5 6 2
Essai 2/10 : bien places 0, mal places 3
Entrez votre essai : 5 2 1 7
Essai 3/10 : bien places 2, mal places 1
Entrez votre essai : 5 2 3 4
Essai 4/10 : bien places 2, mal places 0
Entrez votre essai : 5 2 8 1
Essai 5/10 : bien places 4, mal places 0
```

Question 2. Écrire le programme `main-v1.cpp` en respectant les spécifications exprimées précédemment. Tester et valider.

Question 3. Écrire le programme `main-v2.cpp` afin de jouer une manche autorisant la redondance des couleurs pour une combinaison de 5 pions en 12 tours maximum. Tester et valider.

Des histoires pour les enfants

Vous avez raccompagné le chef au village, à la grande joie de ses habitants qui vous sont très reconnaissants. Vous décidez de passer un peu de temps avec ces villageois. Les enfants du village sont très intrigués par votre langue si différente de la leur, à tel point qu'ils insistent pour que vous leur donniez des cours. Pour les y aider, vous souhaitez imprimer des histoires. Les intervenants de vos histoires sont tous des humains. Un humain est caractérisé par son nom, sa boisson favorite et sa popularité (0 pour commencer). La boisson favorite d'un humain est, par défaut, de l'eau. Dans vos histoires, un humain pourra parler de la manière suivante :

```
(nom de l'humain) -- texte
```

Un humain pourra également se présenter : il dira bonjour, son nom, et indiquera sa boisson favorite. Il pourra boire : il dira alors « Ah ! un bon verre de (sa boisson favorite) ! GLOUPS ! ».

☞ Ce que doit faire votre programme :

Le programme doit raconter une histoire sur *Lucky Luke*. Vous devez définir une classe « Humain » pour les personnages de vos histoires. Vous utiliserez le type `string` pour les chaînes de caractères et respecterez le principe d'encapsulation.

☞ Exemples :

```
$ make

$ ./histoire-enfants
Une histoire sur Lucky Luke
(Lucky Luke) -- Bonjour, je suis Lucky Luke et j'aime le coca-cola
(Lucky Luke) -- Ah ! un bon verre de coca-cola ! GLOUPS !
```

Question 4. Écrire la classe `Humain` (`humain.h` et `humain.cpp`) afin de valider le programme fourni.

```
#include <iostream>
#include "humain.h"

using namespace std;

int main()
{
   Humain lucky("Lucky Luke", "coca-cola");

   cout << "Une histoire sur " << lucky.getNom() << endl;

   lucky.sePresente();

   lucky.boit();

   return 0;
}
```

Une histoire sur Lucky Luke

Bilan

Conclusion : les types sont au centre de la plupart des notions de programmes corrects, et certaines des techniques de construction de programmes les plus efficaces reposent sur la conception et l'utilisation des types.

Rob Pike (ancien chercheur des *Laboratoires Bell* et maintenant ingénieur chez *Google*) :

> « Règle n°5 : Les données prévalent sur le code. Si vous avez conçu la structure des données appropriée et bien organisé le tout, les algorithmes viendront d'eux-mêmes. La structure des données est le coeur de la programmation, et non pas les algorithmes. »

Cette règle n°5 est souvent résumée par « Écrivez du code stupide qui utilise des données futées ! »