



Piscine C++ - d13

Maxime “Hush-hush” Mouial-Baes mouial_m@epitech.eu

Abstract: Ce document est le sujet du d13

Table des matières

I	REMARQUES GÉNÉRALES	2
II	Exercice 0	4
III	Exercice 1	7
IV	Exercice 2	8
V	Exercice 3	9
VI	Exercice 4	12
VII	Exercice 5	14
VIII	Exercice 6	17
	VIII.1 Le fonctionnement de tellMeAStory	18

Chapitre I

REMARQUES GÉNÉRALES


- LISEZ LES REMARQUES GÉNÉRALES ATTENTIVEMENT!!!!
 - Vous n'aurez aucune excuse si vous avez 0 parce que vous avez oublié une consigne générale ...
- REMARQUES GÉNÉRALES :
 - Si vous faites la moitié des exercices car vous avez du mal, c'est normal. Par contre, si vous faites la moitié des exercices par flemme et vous tirez à 14h, vous AUREZ des surprises. Ne tentez pas le diable.
 - Toute fonction implémentée dans un header ou header non protégé signifie 0 à l'exercice.
 - Toutes les classes doivent posséder un constructeur et un destructeur.
 - Toutes les sorties se font sur la sortie standard et sont terminées par un retour à la ligne sauf si le contraire est précisé explicitement.
 - Les noms de fichiers qui vous sont imposés doivent être respectés À LA LETTRE, de même que les noms de classes et de fonctions membres / méthodes.
 - Rappelez-vous que vous faites du C++ et non plus du C. Par conséquent, les fonctions suivantes sont INTERDITES, et leur utilisation sera sanctionnée par un -42 :
 - `*alloc`
 - `*printf`
 - `free`
 - `open`, `fopen`, etc ...
 - De façon générale, les fichiers associés à une classe seront toujours `NOM_DE_LA_CLASSE.h`

et `NOM_DE_LA_CLASSE.cpp` (s'il y a lieu).

- Les répertoires de rendus sont `ex00`, `ex01`, ..., `exN`
 - Toute utilisation de `friend` se soldera par un -42, **no questions asked** .
 - Lisez attentivement les exemples, ils peuvent requérir des choses que le sujet ne dit pas...
 - Ces exercices vous demandent de rendre beaucoup de classes, mais la plupart sont TRÈS courtes si vous faites ça intelligemment. Donc, halte à la flemme !
 - Lisez ENTièrement le sujet d'un exercice avant de le commencer !
 - REFLÉCHISSEZ. Par pitié.
 - REFLÉCHISSEZ. Par Odin !
 - R.E.F.L.É.C.H.I.S.S.E.Z. Nom d'une pipe.
- **COMPILATION DES EXERCICES :**
 - La moulinette compile votre code avec les flags : `-W -Wall -Werror`
 - Pour éviter les problèmes de compilation de la moulinette, incluez les fichiers nécessaires dans vos fichiers `include (*.h)`.
 - Notez bien qu'aucun de vos fichiers ne doit contenir de fonction `main` . Nous utiliserons notre propre fonction `main` pour compiler et tester votre code.
 - Ce sujet peut être modifié jusqu'à 4h avant le rendu. Rafraichissez-le régulièrement !
 - Le répertoire de rendu est : `(DÉPOT SVN - piscine_cpp_d13-promo-login_x)/exN` (N étant bien sur le numéro de l'exercice).

Chapitre II

Exercice 0

	Exercice : 00	points : 4
Encapsulation		
Répertoire de rendu: (DÉPOT SVN - piscine_cpp_d13-promo-login_x)/ex00		
Compilateur : g++	Flags de compilation: -W -Wall -Werror	
Makefile : Non	Règles : n/a	
Fichiers a rendre : Picture.h, Picture.cpp, Toy.h, Toy.cpp		
Remarques : n/a		
Fonctions Interdites : Aucune		

Nous allons créer des jouets de base, chaque jouet aura une représentation. Nous verrons dans les exercices suivant pour d'autre fonctionnalités.

En premier lieu créez une classe “Picture” qui sera l'illustration de notre jouet.

Notre représentation aura :

- en public :
 - `std::string data;` // l'ascii art de notre représentation
 - `bool getPictureFromFile(const std::string& file);`
Prend un nom de fichier en paramètre et set le contenu du fichier en tant que “data”. Renvoie true en cas de succes, en cas d'erreur renvoie false et set “data” a “ERROR”.
 - `Picture(const std::string& file);` Crée un objet `Picture` et load le contenu du fichier “file” comme illustration. Si une erreur arrive avec l'illustration, mettre “ERROR” dans “data” (même gestion d'erreur que dans “getPictureFromFile”).

Si on crée une “Picture” sans fichier en paramètre, “data” sera une chaine de caractère vide.

Maintenant créez une classe “Toy”.

La classe contiendra un enum “ToyType” avec deux champs : “BASIC_TOY” et “ALIEN”.

Votre classe “Toy” aura un type, un nom et une illustration.

De plus elle aura les fonctions membres suivantes :

- **getType** un getter pour le type du jouet, pas de setter: un jouet ne change pas de type !
- **getName** un getter pour le nom.
- **setName** un setter pour le nom.
- **setAscii** prend un nom de fichier en paramètre et set l’illustration du jouet avec. Renvoie **true** en cas de succes et **false** en cas d’erreur.
- **getAscii** renvoie l’illustration du jouet, sous forme de string.
- un constructeur par défaut set le type du jouet à “BASIC_TOY”, le nom à “toy”, et crée une illustration avec une chaine vide pour “data”.
- un constructeur qui prend un ToyType et 2 strings en paramètre et les set comme “type”, “name” du jouet (avec la première string) et load le contenu du fichier (deuxième string) comme illustration.

```

1 #include <iostream>
2 #include 'Toy.h'
3
4 int main()
5 {
6     Toy toto;
7     Toy ET(Toy::ALIEN, 'green', './alien.txt');
8
9     toto.setName('TOTO !');
10
11     if (toto.getType() == Toy::BASIC_TOY)
12         std::cout << 'basic toy: ' << toto.getName() << std::endl
13             << toto.getAscii() << std::endl;
14     if (ET.getType() == Toy::ALIEN)
15         std::cout << 'this alien is: ' << ET.getName() << std::endl
16             << ET.getAscii() << std::endl;
17     return 1337;
18 }

```

\$>./a.out

basic toy: TOTO !

this alien is: green

```


      _|_
    ,-. _ ,-. _
    \ (.) (.) (.) /
    _ , ' \ _ - - - - _ / ' , _
> |-----"-----| <
'""'--/_ _-@-\--'""'
    |===L_I===|
    \ _ _ /
    _ \ _ _ / _
    '""""'""""'

```

\$>

Chapitre III


Exercice 1

	Exercice : 01	points : 1
Forme de Coplien		
Répertoire de rendu: (DÉPOT SVN - piscine_cpp_d13-promo-login_x)/ex01		
Compilateur : g++	Flags de compilation: -Wextra -Werror -Wall	
Makefile : Non	Règles : n/a	
Fichiers a rendre : Picture.h, Picture.cpp, Toy.h, Toy.cpp		
Remarques : n/a		
Fonctions Interdites : Aucune		

Reprenez les deux classes de l'exercice précédent et modifiez les pour en faire des classes de Coplien (pensez à tout ce que ca implique). Si vous avez des doutes reportez-vous aux vidéos de cours et aux koalas. Attention, TOUS les attributs seront copiés.

Chapitre IV

Exercice 2

	Exercice : 02	points : 2
Héritage simple		
Répertoire de rendu: (DÉPOT SVN - piscine_cpp_d13-promo-login_x)/ex02		
Compilateur : g++	Flags de compilation: -Wextra -Werror -Wall	
Makefile : Non	Règles : n/a	
Fichiers a rendre : Picture.h, Picture.cpp, Toy.h, Toy.cpp, Buzz.h, Buzz.cpp, Woody.h, Woody.cpp		
Remarques : n/a		
Fonctions Interdites : Aucune		

Ajoutez à l'enum "ToyType" deux nouveaux jouets : "BUZZ" et "WOODY". Puis créez deux classes : Buzz et Woody.


Ces deux classes sont des spécialisations de "Toy", elles hériteront donc de "Toy". Chaque classe mettra les attributs de Toy à la bonne valeur lors de la construction de l'objet :

- **type** : respectivement "BUZZ" et "WOODY"
- **name** : sera pris en paramètre.
- **ascii** pourra optionnellement être pris en paramètre, si aucun nom de fichier n'est donné alors les objets chargeront leurs illustrations depuis les fichiers "buzz.txt" et "woody.txt" dans le dossier courant.

On ne doit pas pouvoir créer de "Buzz" ou de "Woody" sans nom.

Chapitre V

Exercice 3

	Exercice : 03	points : 2
Polymorphisme par héritage		
Répertoire de rendu: (DÉPOT SVN - piscine_cpp_d13-promo-login_x)/ex03		
Compilateur : g++	Flags de compilation: -Wextra -Werror -Wall	
Makefile : Non	Règles : n/a	
Fichiers a rendre : Picture.h, Picture.cpp, Toy.h, Toy.cpp, Buzz.h, Buzz.cpp, Woody.h, Woody.cpp		
Remarques : n/a		
Fonctions Interdites : Aucune		

Nous voudrions maintenant faire parler nos jouets, vous allez donc rajouter une méthode “speak” à “Toy” qui prendra une phrase à dire en paramètre



beware const, ref, cycle lunaire, lapin blanc, toussa, toussa

Cette méthode affichera le nom du jouet suivi d’un espace et de la phrase prise en paramètre et un retour à la ligne. La sortie devra etre ainsi formatée :

```
1 name ‘‘sentence’’
```

Vous surchargerez cette méthode dans les classes `Buzz` et `Woody` pour afficher respectivement :

```
1 BUZZ: name ‘‘sentence’’
```

et

```
1 WOODY: name ‘‘sentence’’
```

Dans les trois cas, ‘name’ est le nom du jouet et ‘sentence’ la string passée en paramètre.

Contrairement à ce que l’on peut croire, la méthode `speak` ne devra pas être `const`, vous verrez pourquoi par la suite.

```
1
2 #include <iostream>
3 #include 'Toy.h'
4 #include 'Buzz.h'
5 #include 'Woody.h'
6
7 int main()
8 {
9     Toy *b = new Buzz('buzziiii');
10    Toy *w = new Woody('wood');
11    Toy *t = new Toy(Toy::ALIEN, 'ET', 'alien.txt');
12
13    b->speak('To the code, and beyond !!!!!!!!!');
14    w->speak('There's a snake in my boot.');
```


```
15    t->speak('the claaaaaaw');
16 }
```



```
$>./a.out
BUZZ: buzziiii 'To the code, and beyond !!!!!!!!!'
WOODY: wood 'There's a snake in my boot.'
ET 'the claaaaaaw'
$>
```

Chapitre VI

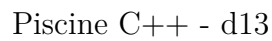
Exercice 4

	Exercice : 04	points : 3
Surcharge d'opérateurs		
Répertoire de rendu: (DÉPOT SVN - piscine_cpp_d13-promo-login_x)/ex04		
Compilateur : g++	Flags de compilation: -Wextra -Werror -Wall	
Makefile : Non	Règles : n/a	
Fichiers a rendre : Picture.h, Picture.cpp, Toy.h, Toy.cpp, Buzz.h, Buzz.cpp, Woody.h, Woody.cpp		
Remarques : n/a		
Fonctions Interdites : Aucune		

Nous allons maintenant mettre en place deux surcharges d'opérateur.

Une première surcharge de l'opérateur “«” entre un `ostream` et un “Toy”. Cela affichera sur la sortie standard le nom du jouet suivi de son illustration. Le nom et l'illustration devront être suivis d'un retour à la ligne.

Une deuxième surcharge de l'opérateur “«” entre un Toy et une string. Cela aura pour effet de remplacer l'illustration du jouet par la string.



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


int main()
{
    Toy a(Toy::BASIC_TOY, "REX", "rex.txt");

    std::cout << a;
    a << "\\o/";
    std::cout << a;
}
```

13

Chapitre VII

Exercice 5

	Exercice : 05	points : 4
Classes Imbriquées		
Répertoire de rendu: (DÉPOT SVN - piscine_cpp_d13-promo-login_x)/ex05		
Compilateur : g++	Flags de compilation: -Wextra -Werror -Wall	
Makefile : Non	Règles : n/a	
Fichiers a rendre : Picture.h, Picture.cpp, Toy.h, Toy.cpp, Buzz.h, Buzz.cpp, Woody.h, Woody.cpp		
Remarques : n/a		
Fonctions Interdites : Aucune		

Nous savons que certains jouets ont plusieurs modes, notre Buzz l'éclair a par exemple un mode espagnol !

Nous allons donc rajouter pour lui, à la classe `Toy`, une méthode `speak_es` qui aura la MEME signature que `speak`.

Dans la classe `Buzz` cette méthode fera la même chose que `speak` mais ajouter "senorita" avant et après la phrase :

```
1 BUZZ: [name] senorita "[sentence]" senorita
```

Avec `[name]` le nom du jouet et `[sentence]` la string passée en paramètre. Vous devez bien afficher les double quotes autour de la phrase comme dit l'exemple.

Le problème est que tous les jouets n'ont pas de mode espagnol, il nous faut donc gérer ce cas d'erreur.

Pour tout jouet n'ayant pas de mode espagnol, la méthode `speak_es` n'affichera rien et renverra false (la méthode renverra donc true en cas de succès).

Profitons en pour améliorer la gestion d'erreur générale de la classe `Toy`. Nous avons actuellement deux sources d'erreur possibles :

- `setAscii`

- `speak_es`

Toutes deux renvoient `false` en cas d'erreur.

Vous allez créer une classe imbriquée `Error` dans `Toy` qui contiendra deux méthodes et un attribut publique:

- `what`
qui renverra le message d'erreur:
 - "bad new illustration" pour une erreur survenant dans `setAscii`
 - "wrong mode" pour une erreur survenant dans `speak_es`
- `where` qui renverra le nom de la fonction où l'erreur est survenue.
- `type` qui contiendra le type de l'erreur.

De plus la classe `Error` contiendra une enum `ErrorType` avec les différents types d'erreurs :

- UNKNOWN
- PICTURE
- SPEAK

Enfin, vous ajouterez à la classe `Toy`, une fonction membre `getLastError` qui renverra une instance de la classe `Error` contenant les informations sur la dernière erreur survenue. Si il n'y a jamais eu d'erreur, `getLastError` renverra une instance de la classe `Error` avec deux chaînes vides pour `what` et `where`, et aura pour type d'erreur UNKNOWN.

Un petit main avec sa sortie pour éclaircir les choses :

```

1 #include <iostream>
2 #include 'Toy.h'
3 #include 'Buzz.h'
4 #include 'Woody.h'
5
6 int main()
7 {
8     Woody w('wood');
9
10    if (w.setAscii('file_who_does_not_exist.txt') == false)
11    {
12        Toy::Error e = w.getLastError();
13        if (e.type == Toy::Error::PICTURE)
14        {
15            std::cout << 'Error in ' << e.where()
16                    << ': ' << e.what() << std::endl;
17        }
18    }
19
20    if (w.speak_es('Woody does not have spanish mode') == false)
21    {
22        Toy::Error e = w.getLastError();
23        if (e.type == Toy::Error::SPEAK)
24        {
25            std::cout << 'Error in ' << e.where()
26                    << ': ' << e.what() << std::endl;
27        }
28    }
29
30    if (w.speak_es('Woody does not have spanish mode') == false)
31    {
32        Toy::Error e = w.getLastError();
33        if (e.type == Toy::Error::SPEAK)
34        {
35            std::cout << 'Error in ' << e.where()
36                    << ': ' << e.what() << std::endl;
37        }
38    }
39 }

```


```

$> ./a.out
Error in setAscii: bad new illustration
Error in speak_es: wrong mode
Error in speak_es: wrong mode
$>

```

Chapitre VIII

Exercice 6

	Exercice : 06	points : 4
Pointeurs sur membres		
Répertoire de rendu: (DÉPOT SVN - piscine_cpp_d13-promo-login_x)/ex06		
Compilateur : g++	Flags de compilation: -Wextra -Werror -Wall	
Makefile : Non	Règles : n/a	
Fichiers a rendre : Picture.h, Picture.cpp, Toy.h, Toy.cpp, Buzz.h, Buzz.cpp, Woody.h, Woody.cpp, ToyStory.h, ToyStory.cpp		
Remarques : n/a		
Fonctions Interdites : Aucune		

Vous allez créer une classe `ToyStory` , qui aura pour rôle de raconter une histoire avec deux jouets.

`ToyStory` contiendra une fonction de classe `tellMeAStory` qui prendra en paramètre :

- un nom de fichier contenant l'histoire;
- un premier `Toy` , disons `toy1`;
- un pointeur sur méthode de la classe `Toy` prenant une `string` en paramètre et renvoyant un booléen, disons `func1`;
- un deuxième `Toy` , disons `toy2`;
- un pointeur sur méthode de la classe `Toy` prenant une `string` en paramètre et renvoyant un booléen, disons `func2`.

Les instances de `Toy` et les pointeurs sur méthodes recus en paramètre vont par paire. Ainsi, `toy1` va avec `func1` et `toy2` va avec `func2`.

VIII.1 Le fonctionnement de `tellMeAStory`

Elle commence par afficher l'illustration des deux jouets, avec un retour à la ligne après chacune.

Le but de `tellMeAStory` est de lire le fichier et pour chaque ligne d'appeler le pointeur sur méthode passé en paramètre sur le jouet associé. Chaque jouet est appelé chacun son tour :

- la première ligne du fichier est envoyée à `func1` sur `toy1`
- la deuxième sur `toy2` avec `func2`
- la troisième sur `toy1` avec `func1`
- ainsi de suite tant qu'il y a des lignes dans le fichier

De plus, si la ligne commence par “picture:” alors cela change l'illustration du jouet dont c'était le tour et affiche la nouvelle illustration. La nouvelle illustration est le contenu du fichier spécifié après “picture:”.

Pour le fichier suivant:

```
1 $>cat story.txt
2 salut
3 picture:ham.txt
4 coucou
5 a+
6 $>
```

Les actions seront les suivantes :

- affiche l'illustration de `toy1` suivie d'un retour à la ligne.
- affiche l'illustration de `toy2` suivie d'un retour à la ligne.
- `func1` est appelée sur `toy1` avec “salut”.
- on change l'illustration de `toy2` avec le contenu de “ham.txt”.
- on affiche la nouvelle illustration de `toy2`.
- `func2` est appelée sur `toy2` avec “coucou”.
- `func1` est appelée sur `toy1` avec “a+”.

`tellMeAStory` s'arrête à la première erreur (si par exemple elle n'arrive pas à changer l'illustration d'un jouet, ...). Vos fonctions membres “`speak*`” devront donc renvoyer un `bool` . Le tout en n'oubliant pas d'écrire sur la sortie standard ou s'est produite l'erreur suivi de “:” et du message d'erreur. Le format à respecter sera :

```
1 where: what
```

Où ‘where’ est la provenance de l’erreur et ‘what’ le message d’erreur. Et si le fichier pris en paramètre n’est pas valide vous écrirez “Bad Story” sur la sortie standard.

Un petit main pour vous aider à comprendre l'utilisation de `tellMeAStory` :

```
1 #include <iostream>
2 #include 'Toy.h'
3 #include 'ToyStory.h'
4 #include 'Buzz.h'
5 #include 'Woody.h'
6
7 int main()
8 {
9     Buzz b('buzzi');
10    Woody w('wood');
11
12    ToyStory::tellMeAStory('superStory.txt', b, &Toy::speak_es, w, &Toy::
    sp\
13 eak);
14 }
```