



Piscine C++ - d14a

Kluedoala

Alexandre “Bibi” BOURLON bourlo_a@epitech.eu
Pierre-Yves “Belga” LEFEUVRE lefeuv_p@epitech.eu

Abstract: Ce document est le sujet du d14a (Bibi, tu fais chier)

Table des matières

I	REMARQUES GÉNÉRALES	2
II	Exercice 0	4
III	Exercice 1	7
IV	Exercice 2	9
V	Exercice 3	12
VI	Exercice 4	13

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 (*.hh)`.
 - 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_d14a-promo-login_x)/exN` (N étant bien sur le numéro de l'exercice).

Chapitre II

Exercice 0

	Exercice : 00	points : 5
Exercice 0		
Répertoire de rendu: (DÉPOT SVN - piscine_cpp_d14a-promo-login_x)/ex00		
Compilateur : g++	Flags de compilation: -W -Wall -Werror	
Makefile : Non	Règles : n/a	
Fichiers a rendre : ex00.cpp, ex00.hh		
Remarques : On vous fournit le fichier "ex00-partial.hh", a completer et rendre comme "ex00.hh"		
Fonctions Interdites : malloc - free		

Vous allez devoir compléter la déclaration des classes `Weapon` , `Suspect` et `Room` . Et implémenter les classes `Card` , `Weapon` , `Suspect` et `Room` .

Pour information, les classes `Weapon` , `Suspect` et `Room` sont à la base des classes `Card` . Donc ce sont des cartes.

Les cartes faisant parties du crime sont celles stockées dans le

```
1 static const Name PartOfTheCrime
```

de chaque classe (une par classe soit 3 au total - une arme, un suspect et une salle).

Ces statics seront settées par nos soins, pour les tests lors de la correction.

Maintenant vous allez devoir, selon les méthodes déclarées dans les classes, rajouter un ou des attributs à chaque classe.



La classe `Weapon` possède la méthode "`bool BearsFingerprints() const`", la classe `Suspect` la méthode "`bool IsLying() const`" et la classe `Room` la méthode "`bool HasSecretPassage() const`". Ce qui devrait vous aiguiller sur les attributs à mettre et ou les mettre.



Pour mettre la propriété `isPartOfTheCrime` à la bonne valeur, Il suffit de voir si le `static const name PartOfTheCrime` correspond avec celui que vous êtes en train d'initialiser.



Les capacités `BearsFingerprints`, `IsLying` et `HasSecretPassage` ne font pas référence au fait d'être `PartOfTheCrime`. Elles sont spécifiques à la classe et donc indépendantes de toute classe de base.

Implémentez la classe `Game` ou plus précisément la fonction membre statique

```
1 bool CaseSolved(Weapon* w[], Suspect* s[], Room* r[])
```

Cette fonction membre retourne `true` si dans les tableaux passés en paramètre se trouvent l'arme, le suspect et la salle du crime.



les tableaux sont terminés par un pointeur `NULL`.

Les 3 éléments doivent être présents pour résoudre l'affaire.

Vous créerez et implémenterez également les classes suivantes :

- `WeaponException`
- `SuspectException`
- `RoomException`

en dehors de tout namespace.

Elles seront levées si (et seulement si) le tableau correspondant à la classe ne contient pas l'élément du crime TANDIS que les deux autres contiennent bien l'élément.

Exemple :

- `Weapon *w` → contient l'arme `PartOfTheCrime`.
- `Suspect *s` → contient le suspect `PartOfTheCrime`.
- `Room *r` → ne contient pas la salle `PartOfTheCrime`.

Une exception `RoomException` est levée.

Chacune des classes possèdera en attributs privés :

- `std::string _message;` qui contiendra “[CLASS] ne contient pas la bonne carte!”
- une copie du tableau qui a fait lever l’exception.



[CLASS] sera bien entendu remplacé par `Weapon`, `Suspect` ou `Room` selon le type.

Le tableau sera passé au constructeur de la classe.
et une fonction membre permettant de lire chacun des attributs :


- `std::string const & getMessage() const;`
- `Weapon **getCards() const;` (Remplacer `Weapon` par `Suspect` ou `Room` selon la classe, bien entendu).



Vous n’avez pas à vous préoccuper de comment est fait le `try/catch` pour le moment... Mis à part qu’il prend un pointeur sur les exceptions!

Chapitre III

Exercice 1

	Exercice : 01	points : 5
Exercice 1		
Répertoire de rendu: (DÉPOT SVN - piscine_cpp_d14a-promo-login_x)/ex01		
Compilateur : g++	Flags de compilation: -Wextra -Werror -Wall	
Makefile : Non	Règles : n/a	
Fichiers a rendre : ex01.cpp, ex01.hh		
Remarques : On vous fournit le fichier "ex01-partial.hh", a completer et rendre comme "ex01.hh"		
Fonctions Interdites : malloc, free		

La classe `Card` possède maintenant une nouvelle propriété `CardType`. Cette propriété doit être configurée correctement lors de l'instanciation d'un `Weapon`, d'un `Suspect` ou d'une `Room`.

Implémentez également la surcharge

```
1 bool CaseSolved(Card* c[])
```

de la classe `Game` qui retournera `true` uniquement si le tableau passé en paramètre contient les 3 `PartOfTheCrime`.

Vous allez maintenant créer la classe `CardException`. Cette classe héritera de `std::exception`.

Vous remplacerez la fonction membre `getMessage` par l'implémentation de `what()`.
Le message sera le suivant :


```
1 Aucune des cartes ne correspond!
```


Cette exception sera levée si le tableau passe en paramètre à `CaseSolved` ne contient aucune carte ne faisant partie du crime. Ce tableau sera stocké dans l'instance de `CardException`, passé en paramètre au constructeur. Encore une fois, il faudra `throw` un pointeur sur l'exception. Aucune autre exception ne devra être levée dans cet exercice.

Vous implémenterez également la fonction membre `Card** getCards() const` qui renverra le tableau stocké dans l'instance.

Chapitre IV

Exercice 2

	Exercice : 02	points : 5
Exercice 2		
Répertoire de rendu: (DÉPOT SVN - piscine_cpp_d14a-promo-login_x)/ex02		
Compilateur : g++	Flags de compilation: -Wextra -Werror -Wall	
Makefile : Non	Règles : n/a	
Fichiers a rendre : ex02.cpp, ex02.hh		
Remarques : On vous fournit le fichier "ex02-partial.hh", a completer et rendre comme "ex02.hh"		
Fonctions Interdites : malloc - free		

Au Kluedoala, les armes sont représentées par une carte et une figurine. Les suspects sont représentés également par une carte mais aussi par une pièce (un pion) représentant le joueur.

Faites donc en sorte que les classes `Weapon` et `Suspect` héritent également des classes correspondantes à leurs représentations, soit respectivement `Figurine` et `Piece`.

- La classe `Piece` possède une position sur le plateau de jeu.
 - Cette position est comprise entre 1 et 100.
 - Elle doit être accessible via son getter `int Position() const`
 - et son setter `void Position(int position)`.
- Elle est également configurée lors de son instantiation via son constructeur.
- La position de base est 1 et doit être définie en interne par la classe héritante.



la liste d'initialisation est votre amie. :p

Comme il est courant de perdre des figurines dans un jeu de plateau comme le Cluedo, la classe `Figurine` se contente d'implémenter la méthode `bool IsLost() const` permettant de savoir si la figurine a été perdue au cours des années par votre grand frère :).



`bool IsLost()` est un Getter.

Cette propriété est configurée via le constructeur et doit être précisée en externe via le constructeur de la classe héritante.

Vous allez maintenant créer la classe `GameException`. Cette classe héritera de la classe `std::exception`.

Vous créerez ensuite les classes

- `FigurineException`
- `PieceException`

qui implémenteront toutes les deux la classe `GameException`.

Les implémentations de `what` renverront :

- "Lost Figurine" //pour la classe `FigurineException`
- "Piece is not on the board" //pour la classe `PieceException`

`FigurineException` sera levée si la figurine est l'arme du crime, est perdue et qu'on tente de s'en servir dans `CaseSolved(Card **)`. Elle contiendra un pointeur sur `Figurine` ainsi que la fonction membre : `Figurine *getFigurine() const`;

`PieceException` sera levée si on tente de fournir à une `Piece` une position dont la valeur n'est pas dans le range 1-100. Elle contiendra un pointeur sur `Piece` et la fonction membre `Piece *getPiece() const`;

Bien entendu toutes les exceptions levées doivent être applicables au bloc `try/catch` suivant:

```
1 try
2 {
3     ....
4 }
```


```
5 catch (std::exception* e)
6 {
7     std::cout << e->what() << std::endl;
8 }
```

Récapitulatif :

- Complétez la déclaration et l'implémentation des classes `Piece` et `Figurine`.
- Implémentez la méthode `int Position() const` de `Piece`.
- Implémentez la méthode `void Position(int position)` de `Piece`.
- Complétez l'implémentation des classes `Weapon` et `Suspect`.
- Modifiez le constructeur de `Suspect` pour configurer la position à la valeur par défaut.
- Modifiez le constructeur de `Weapon` pour configurer la propriété `IsLost` à la valeur définie par l'utilisateur.

Chapitre V

Exercice 3

	Exercice : 03	points : 5
Exercice 03		
Répertoire de rendu: (DÉPOT SVN - piscine_cpp_d14a-promo-login_x)/ex03		
Compilateur : g++	Flags de compilation: -Wextra -Werror -Wall	
Makefile : Non	Règles : n/a	
Fichiers a rendre : ex03.hh, ex03.cpp		
Remarques : On vous fournit le fichier "ex03-partial.hh", a completer et rendre comme "ex03.hh"		
Fonctions Interdites : malloc - free		

Réécrivez la méthode statique `bool CaseSolved(Card* c[])` de l'exercice 2 mais cette fois sans utiliser le `CardType`. De plus cette méthode ne devra plus lever aucune exception.




Ca ne veut pas dire que vous devez juste enlever votre `throw(exception)` du corps de la fonction...

Récapitulatif :

- Ré-implémentez la méthode statique `bool CaseSolved(Card* c[])` de la classe `Cluedo`.

Chapitre VI

Exercice 4

	Exercice : 04	points : 5
Exercice 4		
Répertoire de rendu: (DÉPOT SVN - piscine_cpp_d14a-promo-login_x)/ex04		
Compilateur : g++	Flags de compilation: -Wextra -Werror -Wall	
Makefile : Non	Règles : n/a	
Fichiers a rendre : ex04.cpp, ex04.hh		
Remarques : On vous fournit le fichier "ex04-partial.hh", a completer et rendre comme "ex04.hh"		
Fonctions Interdites : Aucune		

Complétez la déclaration et l'implémentation des classes `Weapon`, `Suspect` et `Room` en fonction de la nouvelle méthode de la classe `Card`.

```
1 virtual bool SpecialAbility() const
```

Cette méthode réagit de la même manière que les méthodes `BearsFingerprints`, `IsLying` et `HasSecretPassage` des classes précédemment définies.



Si elle réagit de la même façon, c'est que vous n'avez pas besoin de coder grand chose.

Vous modifierez également les classes `WeaponException`, `SuspectException` et `RoomException` de façon à ce qu'elles héritent toutes directement de `std::exception`. Vos implémentations de `what` renverront les informations suivantes :

```
1 This Weapon bears finger prints. But it's not a part of the crime!
```

```
1 This Suspect is lying. But he's not part of the crime!
```

```
1 This Room has a secret passage. But it's not part of the crime!
```

Ces exceptions seront levées dans les fonctions `BearsFingerPrints`, `IsLying` et `HasSecretPassage`. Et uniquement dans ces fonctions membres.

Bien entendu, les exceptions levées seront toutes du type `std::exception*`.



Les fonctions `BearsFingerPrints`, `IsLying` et `HasSecretPassage` sont `const` et doivent le rester. Pensez bien à la façon de construire vos exceptions.

Récapitulatif:

- Complétez la déclaration et l'implémentation des classes `Weapon`, `Suspect` et `Room`.
- Modifiez les classes `WeaponException`, `SuspectException` et `RoomException`.

