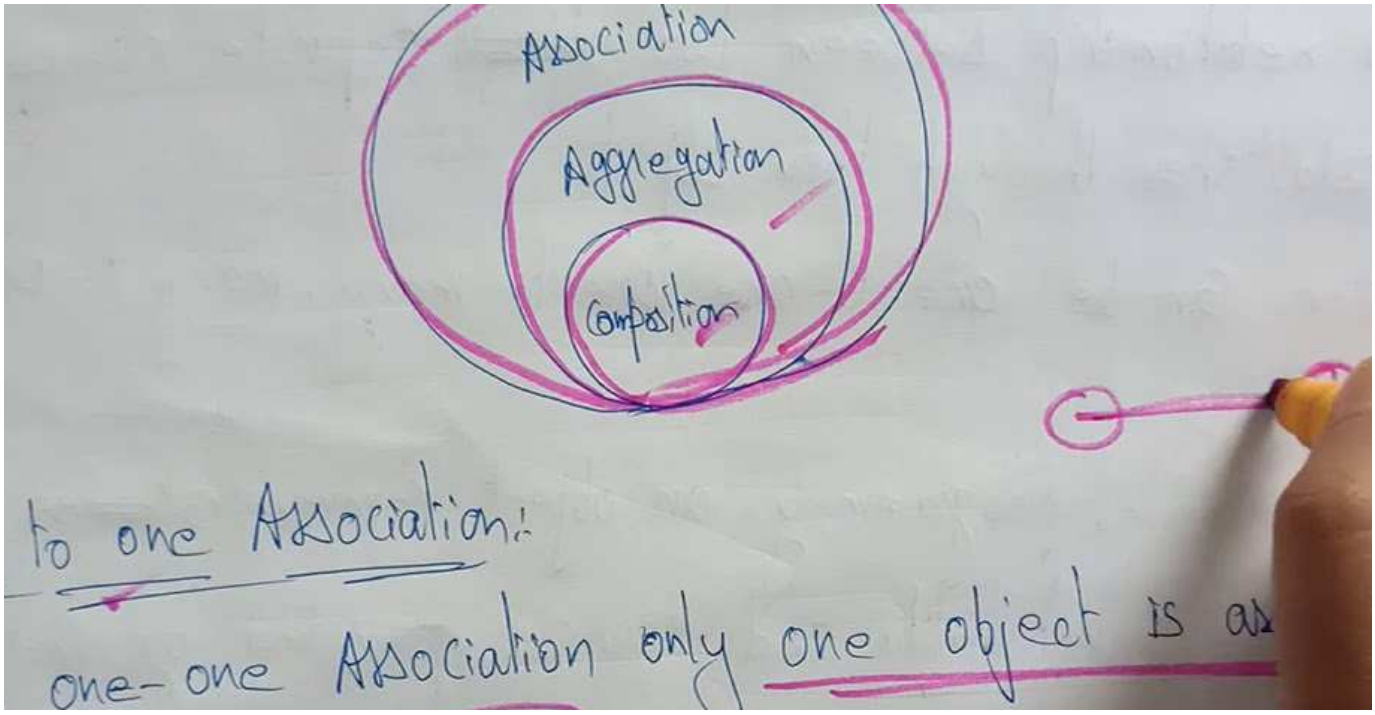
Niveau 4

TP
S4.6.Programmation orientée objet
Mécanismes d'agrégation et de composition

Liaisons entre classes

TABLE DES MATIÈRES



1 - PRÉSENTATION

Le but de ce TP est d'arriver à maîtriser le codage des diverses liaisons entre classes.

Chacune d'entre elle a une technique de codage particulière qui lui est adapté.

Chaque exercice devra faire l'objet d'un nouveau projet.

Le grand nombre de question ne doit pas vous effrayer : la plupart d'entre elles sont des redites (par exemple, la 3, la 11, la 18...), et vous donne la marche à suivre pour coder de A à Z les classes et les liaisons.

Dans toutes les classes, les constructeurs et destructeurs affichent des messages du style :
`Je suis le constructeur de l'objet 0x22fe10.`

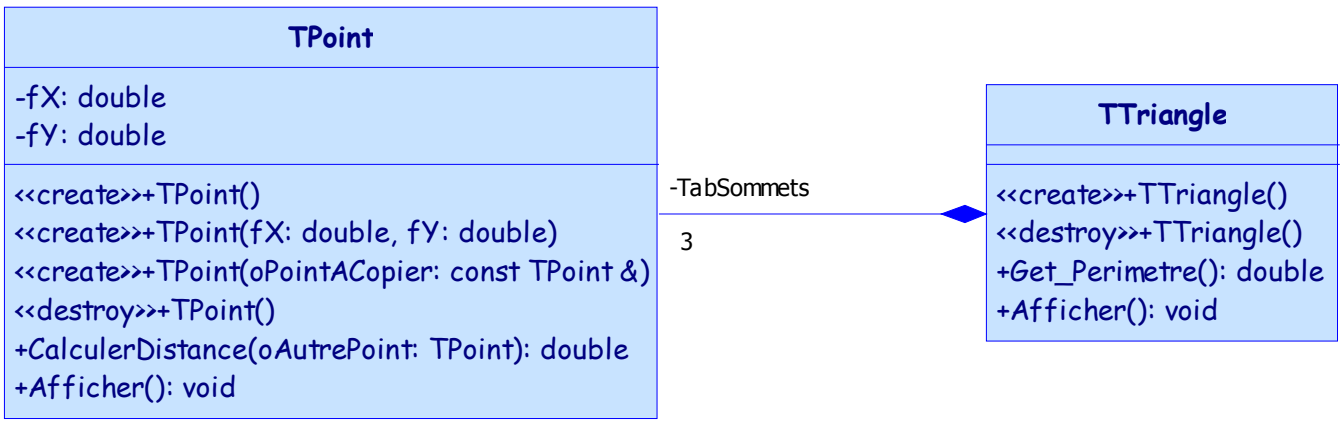
2 - COMPOSITION : TRIANGLE

On se souvient du jeune Éli Coptère qui n'aime vraiment pas les problèmes de maths... et c'est toujours le cas avec la géométrie !

En ce moment, il apprend à calculer le périmètre d'un triangle... Mais il n'y arrive pas !

Vous devez l'aider !

Une rapide analyse de la structure d'un triangle vous a amené au diagramme de classe suivant : les sommets sont définis par trois points et les côtés par des segments de droite les reliant deux à deux successivement. De plus, vous, vous savez que le périmètre est la somme des longueurs de chaque côté. Il ne vous reste plus qu'à lui faire le programme associé !



1) Créez un nouveau projet C++ console avec Qt, nommé `Composition_Triangle`.

2) Ajoutez dans le dossier les sous-dossiers nécessaires !

2.1 - TPOINT

3) Ajoutez à votre projet une nouvelle classe `TPoint`, déplacez les fichiers dans leur sous-dossier respectif et adaptez le `.pro` en conséquence.

4) Selon le diagramme UML proposé, codez la déclaration de la classe `TPoint`.

5) Définissez le constructeur par défaut. Il doit initialiser les attributs de la classe, `fX` et `fY`, avec des valeurs aléatoires comprises entre `1.0` et `19.9`.
Instanciez un objet nommé `oPoint` dans le `main()` et validez les valeurs de ses attributs grâce au débugueur.
Les valeurs des exemples suivants sont données avec une graine fixée à `2023` !

6) Codez le constructeur paramétré qui utilise les paramètres formels qu'il reçoit pour initialiser les attributs de la classe. Instanciez un objet nommé `oAutrePoint` dans le `main()` et validez les valeurs de ses attributs grâce au débugueur.

7) Codez le constructeur de recopie qui va recopier les valeurs des attributs du paramètre `oPointACopier` dans les attributs de la classe.
Remarque : il s'agit ici d'un cas particulier d'accessibilité ! Nous avons un passage de paramètre par référence (ça marche aussi par valeur !), et le type du paramètre est identique à la classe `TPoint` : les attributs du paramètre sont alors accessibles dans ce constructeur comme s'ils étaient déclarés en public.
Instanciez un objet nommé `oPointCopie` dans le `main()` qui recopie l'objet `oPoint` et validez les valeurs des attributs grâce au débugueur.

8) Définissez le destructeur qui réinitialise les attributs à `0.0`.

9) Codez la méthode Afficher() qui doit afficher un message du style :
```
Je suis le point 0x22fe10 aux coordonnees [ 19.4 , 16.1 ]
```
(`0x22fe10` étant l'adresse de l'objet qui s'affiche).
Testez cette méthode dans le `main()` en affichant les informations de l'objet `oPoint`.

10) Finalement écrivez le code de la méthode `CalculerDistance()`, qui retourne la distance entre les deux points, celui qui exécute la méthode et le paramètre. Affichez et validez cette distance dans le `main()`.

2.2 - TTRIANGLE

11) Ajoutez à votre projet une nouvelle classe `TTriangle`, déplacez les fichiers dans leur sous-dossier respectif et adaptez le `.pro` en conséquence.

12) Selon le diagramme UML proposé, codez la déclaration de la classe `TTriangle`.
Pensez à déclarer correctement l'attribut permettant de faire la liaison avec la classe `TPoint`.

13) Définissez le constructeur par défaut et le destructeur.
Implémentez dans le `main()` un objet de cette classe qui sera nommé `oTriangle`.

14) Définissez la méthode `Get_Perimetre()` qui retourne la longueur du périmètre du triangle. il est rappelé que le périmètre est la somme des distances entre les trois sommets du triangle. Pensez à utiliser toutes les possibilités proposées par la classe `TPoint` !

15) Codez la méthode Afficher() qui doit afficher un message du style :

```
Je suis le triangle 0x22fde0 et voici la liste de mes sommets :
        Je suis le point 0x22fde0 aux coordonnees [ 7.3 , 1 ]
        Je suis le point 0x22fdf0 aux coordonnees [ 13.3 , 7.5 ]
        Je suis le point 0x22fe00 aux coordonnees [ 10.7 , 10.8 ]
et j'ai un perimetre de : 23.4201
```

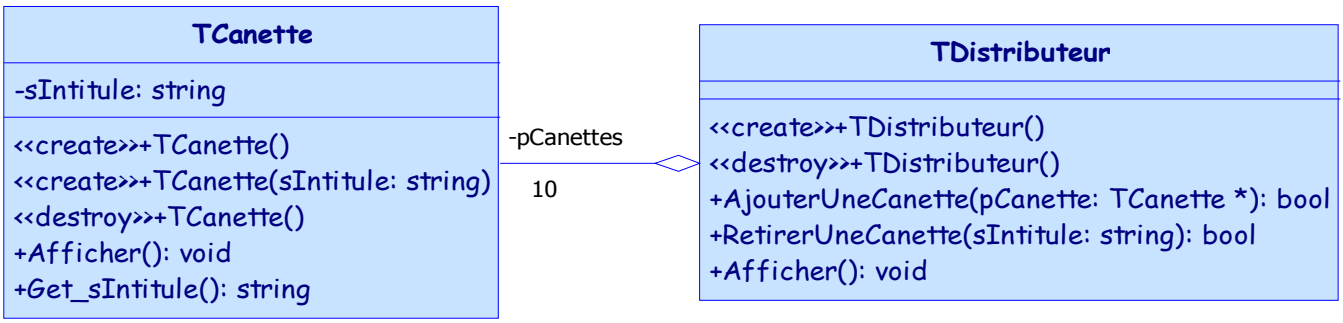
Testez cette méthode dans le `main()` en affichant les informations de l'objet `oTriangle`.

3 - AGRÉGATION : DISTRIBUTEUR

Monsieur Timdeuro est réparateur de distributeur de boisson. Malheureusement, il est confronté à l'obsolescence programmée du logiciel de ses machines qui n'est plus compatible depuis la dernière mise à jour du SE des appareils.

Vincent vous demande de lui réaliser un module pour gérer les canettes de boissons dans les distributeurs !

Une petite analyse vous a conduit au diagramme de classe suivant :



16) Créez un nouveau projet C++ console avec Qt, nommé `Association_Distributeur`.

17) Ajoutez dans le dossier les sous-dossiers nécessaires !

3.1 - TCANETTE

18) Ajoutez à votre projet une nouvelle classe `TCanette`, déplacez les fichiers dans leur sous-dossier respectif et adaptez le `.pro` en conséquence.

19) Selon le diagramme UML proposé, codez la déclaration de la classe `TCanette`.

20) Définissez le constructeur par défaut. Il doit initialiser l'attribut `sIntitule` à la valeur `"Boisson Mystere"`.
Instanciez un objet nommé `oCanetteMystere` dans le `main()` et validez les valeurs de ses attributs grâce au débugueur.

21) Codez le constructeur paramétré. Testez-le dans le `main()` en créant un nouveau objet nommé `oCanette` ayant l'intitulé `"Super Bulles"`. Validez l'instanciation grâce au débugueur.

22) Définissez le destructeur.

23) Codez l'accesseur qui retourne l'intitulé de la cannette. Testez-le dans le `main()` en affichant l'intitulé de l'objet `oCanette`.

24) Finalement, codez la méthode `Afficher()` qui affiche un message du style :
```
Je suis la canette 0x22fdf0 qui contient : Boisson Mystere.
```
Appelez cette méthode pour les deux objets dans le `main()`.

3.2 - TDISTRIBUTEUR

25) Ajoutez à votre projet une nouvelle classe `TDistributeur`, déplacez les fichiers dans leur sous-dossier respectif et adaptez le `.pro` en conséquence.

26) Selon le diagramme UML proposé, codez la déclaration de la classe `TDistributeur`.
Pensez à déclarer correctement l'attribut permettant de faire la liaison avec la classe `TCanette`.

27) Définissez le constructeur par défaut. Il va initialiser chaque case de l'attribut `pCanettes` à la valeur `nullptr`.
Instanciez un objet nommé `oDistributeur` dans le `main()` et validez les valeurs de ses attributs grâce au débugueur.

28) Définissez le destructeur qui réinitialise toutes les cases de l'attribut `pCanettes` à `nullptr`.

29) Codez la méthode `AjouterUneCanette()` qui recherche le premier emplacement libre dans le distributeur (`pCanettes[]`) et qui lui attribue la canette passée en paramètre. Si le distributeur est plein (pas de case disponible dans `pCanettes`) la méthode retourne `false`, sinon elle renvoie `true`.
Ajoutez les canettes `oCanetteMystere` et oCanette au `oDistributeur` dans le `main()` et validez grâce au débugueur.
Pour chaque ajout, vous validerez l'action et afficherez, dans le `main()` l'un des messages selon la réussite :
```
La canette a pu etre ajoutee au distributeur !
La canette n'a pu etre ajoutee : le distributeur est plein !
```

30) Définissez la méthode `Afficher()` qui compte le nombre de canettes disponibles et affiche en conséquence un message du style :
```
Je suis le distributeur 0x22fd80 et je n'ai pas de canette disponible !
```
ou
```
Je suis le distributeur 0x22fd80 et j'ai 2 canette(s) disponible(s) :
    Je suis la canette 0x22fdf0 qui contient : Boisson Mystere
    Je suis la canette 0x22fdd0 qui contient : Super Bulle
```

31) Finalement, codez la méthode `RetirerUneCanette()` qui recherche dans le distributeur (`pCanettes[]`) la première canette ayant l'intitulé passé en paramètre. Si on a trouvé une canette portant cet intitulé, la méthode réinitialise la case de `pCanette` correspondante et retourne `true`, elle ne fait rien et renvoie `false` sinon.
Testez cette méthode dans le `main()` en supprimant une canette intitulée `"Super Bulle"` puis une autre intitulée `"Giga Geyser"`. Affichez l'un des messages suivant en conséquence :
```
Vous pouvez prendre votre boisson !
```
ou
```
Desole, cette boisson est en rupture de stock...
```

4 - UTILISATION : COURS

Madame Praufesseur donne des cours particuliers de français à des étudiants en difficulté.

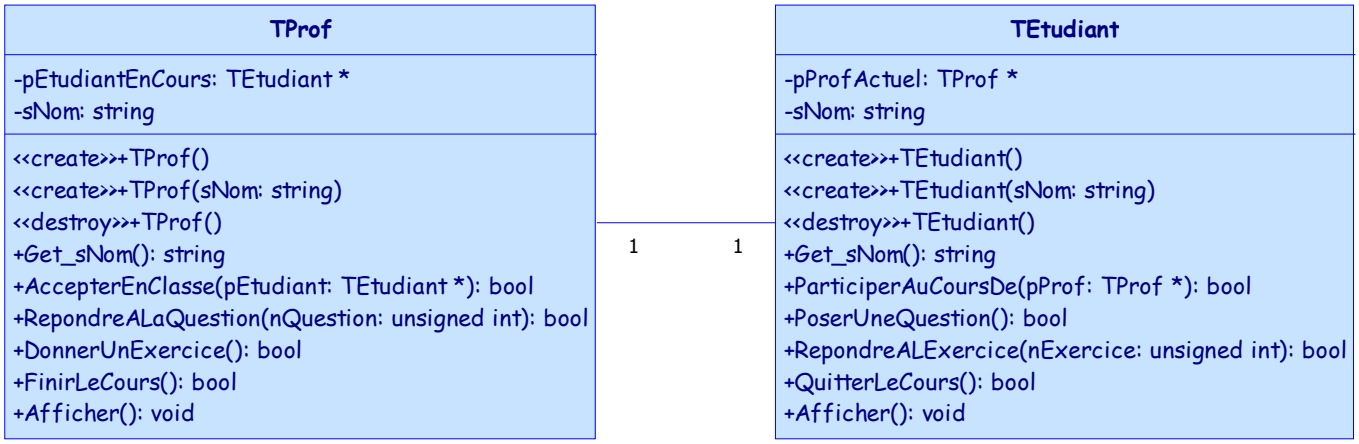
Elle vous demande de lui faire un petit programme pour lui permettre de faire ses cours à distance.

Candice sait bien ce qu'elle souhaite : que les étudiants puissent s'inscrire à un cours si elle n'est pas occupée, qu'ils puissent lui poser une question à laquelle elle répondra immédiatement, qu'elle puisse leur donner un exercice qu'il devront solutionner dans la foulée, et finalement qu'elle déclare le cours terminé !

À vous de jouer !

Ici, nous allons avoir une utilisation bidirectionnelle : il faut que les fichiers `.h` soient complets et les `.cpp` des deux classes soit définis, avant de pouvoir coder les méthodes en elles-mêmes.

Du fait de la liaison et de son codage par pointeur, il faudra obligatoirement tester que le pointeur ne soit pas `nullptr` avant de faire quoi que ce soit.



32) Créez un nouveau projet C++ console avec Qt, nommé `Utilisation_Cours`.

33) Ajoutez dans le dossier les sous-dossiers nécessaires !

34) Ajoutez à votre projet une nouvelle classe `TEtudiant`, déplacez les fichiers dans leur sous-dossier respectif et adaptez le `.pro` en conséquence.

35) Selon le diagramme UML proposé, codez la déclaration de la classe `TEtudiant`.
Pensez à déclarer correctement l'attribut permettant de faire la liaison avec la classe `TProf`.

36) Définissez toutes les méthodes de la classe `TEtudiant` mais laissez-les vides, sauf si elles retournent quelque chose. Dans ce cas, codez les `return`...

37) Ajoutez à votre projet une nouvelle classe `TProf`, déplacez les fichiers dans leur sous-dossier respectif et adaptez le `.pro` en conséquence.

38) Selon le diagramme UML proposé, codez la déclaration de la classe `TProf`.
Pensez à déclarer correctement l'attribut permettant de faire la liaison avec la classe `TEtudiant`.

39) Définissez toutes les méthodes de la classe `TProf` mais laissez-les vides...

4.1 - TETUDIANT

40) Codez le constructeur par défaut qui initialise les attributs avec leur valeur par défaut. Un étudiant instancié par ce constructeur portera le nom `"Lambda"`.
Instanciez un objet `oUnEtudiant` dans le `main()` et validez les valeurs de ses attributs avec le débugueur.

41) Codez le constructeur paramétré qui initialise les attributs avec leur valeur par défaut et le `sNom` avec le paramètre passé s'il n'est pas vide. Dans le cas contraire le nom devra être `"Lambda"`.
Instanciez un objet `oUnAutreEtudiant` dans le `main()` et validez les valeurs de ses attributs avec le débugueur.

42) Remplissez maintenant le destructeur.

43) Ajoutez le code de `Afficher()`, qui va afficher les messages suivants selon le cas :
```
Je m'appelle Lambda (0x22fca0) et je ne suis aucun cours en ce moment...
```
ou
```
Je m'appelle Lambda (0x22fca0) et je suis en cours avec Monsieur Nimbus...
```

44) Complétez la méthode `ParticiperAuCoursDe()` qui permet à l'étudiant de s'inscrire au cours du `pProf` passé en paramètre. Cette méthode va instaurer la liaison `TEtudiant` ⇨ `TProf`, en initialisant l'attribut `pProfActuel` avec le paramètre reçu.
On ne peut s'inscrire au cours que si on ne participe pas déjà à un cours (`pProfActuel` à `nullptr`) et qu'on a indiqué un professeur existant (`pProf` non égal à `nullptr`).
On doit retourner `true` si le professeur nous accepte en classe (appel de `AccepterEnClasse()` du `pProfActuel`) et `false` sinon.
Créez un objet `oLeProf` dans le `main()` en utilisant son constructeur par défaut.
Faites participer `oUnEtudiant` au cours de ce `oLeProf`. Validez avec le débugueur le fait que l'attribut pointe bien sur l'objet `oLeProf`.

45) Codez la méthode `PoserUneQuestion()`. Elle contient un tableau de 4 questions sous forme de `string`. Une des questions est tirée aléatoirement et est affichée immédiatement :
```
Monsieur Nimbus, j'ai cette question pour vous : Quest 3
```
Le numéro de la question est fournie au `pProfActuel` grâce à l'appel de sa méthode `RepondreALaQuestion()` , même si elle est encore vide...
Faites poser une question par `oUnEtudiant` dans le `main()` et validez le fonctionnement actuel.

46) Complétez la méthode `QuitterLeCours()` qui affiche un message d'au-revoir au `pProfActuel` et qui le réinitialise à `nullptr`. Les messages sont les suivants :
```
Au revoir, Monsieur Nimbus
```
ou
```
Je ne suis aucun cours de prof en ce moment !
```
Bien sûr, on ne peut quitter le cours que s'il y a un `pProfActuel` défini ! Dans ce cas, la méthode retourne `true`, sinon c'est `false`.
Faites quitter le cours à `oUnEtudiant` dans le `main()` et affichez s'il a effectivement quitté le cours.

4.2 - TPROF

Sur le même modèle que `TEtudiant`, codez les méthodes de `TProf`...

47) Codez le constructeur par défaut qui donne le nom `"Nimbus"` aux objets qu'il initialise.
Instanciez un objet `oLeProf` dans le `main()` et validez au débugueur.

48) Codez le constructeur paramétré qui initialise le `sNom` avec le paramètre passé s'il n'est pas vide.
Dans le cas contraire le nom devra être `"Nimbus"`.
Instanciez un objet `oLAutreProf` dans le `main()` et validez avec le débugueur.

49) Remplissez maintenant le destructeur.

50) Ajoutez le code de `Afficher()`, qui va afficher les messages suivants selon le cas :
```
    Je m'appelle Professeur Nimbus(0x22fc40) et je ne donne aucun cours en ce moment...
```
ou
```
    Je m'appelle Professeur Nimbus(0x22fc40) et je suis en cours avec Monsieur Lambda...
```

51) Complétez la méthode `AccepterEnClasse()` qui permet au prof d'accepter de donner un cours au `pEtudiant` passé en paramètre. Cette méthode va instaurer la liaison `TProf` ⇨ `TEtudiant`, en initialisant l'attribut `pEtudiantEnCours` avec le paramètre reçu.
On doit retourner `true` si on peut accepter en classe (`pEtudiantEnClasse` à `nullptr` et `pEtudiant` non égal à `nullptr`) et `false` sinon.
Le code de l'appel de cette méthode a déjà été implémenté à la question 44. Validez avec le débugueur le fait que l'attribut `pEtudiantEnCours` de l'objet `oLeProf` pointe bien sur l'objet `oUnEtudiant`.

52) Codez la méthode `RepondreALaQuestion()`. Elle contient un tableau de 4 réponses sous forme de `string`. Le numéro de la question à laquelle répondre est fournie par le paramètre nQuestion.
La méthode va afficher la réponse sous la forme :
```
Voici ma reponse, Monsieur Lambda : Rep 3
```
Le code nécessaire dans le `main()` a déjà été écrit à la question 45. Validez le fonctionnement du système Question-Réponse : la réponse doit correspondre à la question !

53) Complétez la méthode `FinirLeCours()` qui affiche un message pour le `pEtudiantEnCours`, qui lui demande de quitter le cours (appel de la méthode `QuitterLeCours()` de `pEtudiantEnCours`) qui le réinitialise à `nullptr`. Les messages sont les suivants :
```
Le Cours est termine, Monsieur Lambda
```
ou
```
Je ne donne pas de cours en ce moment !
```
Bien sûr, on ne peut finir le cours que s'il y a un `pEtudiantEnCours` défini ! Dans ce cas, la méthode retourne `true`, sinon c'est `false`.
Faites finir le cours par `oLeProf` dans le `main()` ***(mettez en commentaire le code écrit dans le main() pour la question 46)*** et affichez s'il a effectivement terminé le cours.

4.3 - EN ROUE LIBRE...

54) Selon le processus Question-Réponse (questions 45 et 52), écrivez le code des méthodes `TProf::DonnerUnExercice()` et `TEtudiant::RepondreALExercice()`.

55) Il reste un problème de taille : il ne devrait normalement pas être possible d'appeler les méthodes RepondreALaQuestion(), AccepterEnClasse(), RepondreALExercice() et QuitterLecours() depuis le main(). Elles ne doivent être appelées que dans les méthodes associées !
Quelle pourrait être la solution à se problème ?