



Notion de cours abordés :

- **Classes** : Création et utilisation de classes en C++.
- **Attributs privés et protégés** : Gestion de la visibilité et de l'accès aux attributs des classes.
- **Constructeur et Destructeur** : Initialisation et nettoyage des objets.
- **Méthodes de la classe** : Définition et utilisation des fonctions membres.
- **Instance dynamique** : Création d'objets dynamiquement avec des pointeurs.
- **Flux d'entrée/sortie standard** : Utilisation des entrées/sorties standard (cin, cout).
- **Relation d'héritage** : Création de classes dérivées.
- **Relation de composition** : Inclusion d'objets d'une classe dans une autre classe.
- **Gestionnaire de version (GitHub)** : Utilisation de Git pour versionner le code.
- **Documentation (Doxygen)** : Génération de la documentation du code à partir des commentaires.

A) CONSIGNES :

Vous devez déposer votre travail dans un repository privé GitHub, dédié à l'apprentissage du C++. Ce repository servira à stocker et versionner votre code tout au long des différentes étapes du TD. Assurez-vous de nommer chaque commit en fonction de l'étape réalisée, par exemple : *compte bancaire*, *compte épargne*, ou *compte client*.

Ressources utilisées :

Matériel

Un ordinateur sous Linux

Logiciel

Git
Doxygen
Qt Creator
Bibliothèque Menu en C++

B) CRÉATION DU PROJET

1. **Créez un dossier ProjetBanque** : Placez ce dossier dans votre repository GitHub dédié à l'apprentissage du C++. Ce dossier servira de base pour votre projet sous Qt et contiendra tous les fichiers du projet ainsi que la documentation.
2. **Créez un sous-dossier Documentation** : Ce dossier, situé dans ProjetBanque, sera utilisé pour stocker la documentation générée par Doxygen.
3. **Sous QtCreator, créez un projet nommé LaBanque** : Ce projet doit être de type "Application console C++" sans utiliser la bibliothèque Qt. Le projet sera suivi par Git et doit être placé dans un sous-dossier de ProjetBanque.
4. **Ajoutez la classe Menu au projet** : La classe Menu vous est fournie en ressource. Assurez-vous de l'intégrer correctement au projet LaBanque.

C) ÉTAPE N°1 : COMPTE BANCAIRE

Description du Compte Bancaire :

- Un compte bancaire possède à tout moment une donnée : son **solde**, qui peut être positif (compte créditeur) ou négatif (compte débiteur).
- À sa création, un compte bancaire a un solde par défaut égal à **0**. Il est toutefois possible de créer un compte avec un solde initial différent de 0.
- Les opérations possibles sur un compte sont :
 - **Dépôt** : ajouter un montant au solde.
 - **Retrait** : retirer un montant du solde, à condition que le compte soit suffisamment approvisionné.
 - Il est aussi possible de **consulter le solde** du compte.

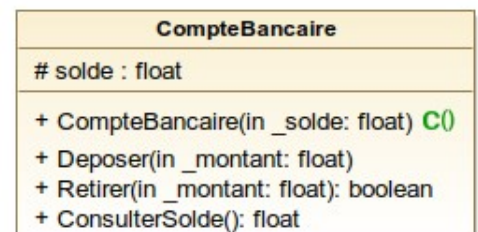
La modélisation du compte bancaire est représentée par la figure ci-contre :

Instructions pour le Codage :

5. **Réalisez le codage de la classe CompteBancaire** en respectant la modélisation UML et les descriptions fournies. Utilisez le type d'attribut approprié pour représenter le solde et ajoutez les méthodes pour gérer les dépôts, retraits et consultations.
6. **Création d'un fichier texte compteBancaire.txt** : Ce fichier doit contenir un menu interactif pour tester les fonctionnalités de la classe CompteBancaire. Les options du menu pourraient inclure : Dépôt, Retrait, Consultation du solde, etc.

Programme Principal :

7. **Réalisez un programme principal** (main.cpp) permettant de tester toutes les fonctionnalités de CompteBancaire à partir du menu défini dans compteBancaire.txt. Assurez-vous que tous les affichages sont effectués dans ce programme principal pour une meilleure séparation du code.



```
Terminal
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
+-----+
0 | Consulter le solde |
1 | Effectuer un depot |
2 | Effectuer un retrait |
3 | Retour |
+-----+
votre choix svp : |
```

8. Documentation avec Doxygen :

Documentez votre classe **CompteBancaire** ainsi que les fichiers associés en ajoutant des commentaires Doxygen. Générez la documentation **Doxygen** et placez-la dans le sous-dossier Documentation de ProjetBanque.

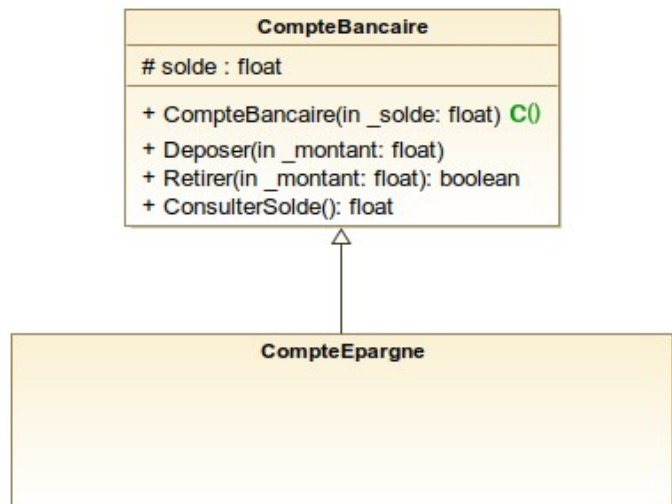
D) ÉTAPE N°2 : COMPTE ÉPARGNE

Cette étape consiste à étendre les fonctionnalités d'un compte bancaire pour modéliser un compte épargne. Le compte épargne possède les mêmes caractéristiques qu'un compte bancaire, mais inclut également un taux d'intérêt et des fonctionnalités supplémentaires pour gérer ce taux.

Description du Compte Épargne :

Comme le montre la modélisation UML ci-contre, le compte épargne est une extension du compte bancaire et possède les mêmes fonctionnalités, avec un attribut supplémentaire : **tauxInterets**, représentant le taux d'intérêt actuel (3 % depuis le 1er février 2023). Attention, ce taux pourrait être modifié à l'avenir (peut-être 2,5 % en 2025).

La classe doit inclure une méthode **CalculerInterets** pour calculer le gain apporté par les intérêts du compte de manière périodique.



À la création du compte épargne, le taux d'intérêt est fixé à la valeur actuelle. Il doit également être possible de modifier ce taux via la méthode **ModifierTaux**

9. Création de la Classe CompteEpargne :

Sous QtCreator, ajoutez une nouvelle classe nommée **CompteEpargne** dans le projet **LaBanque**. Implémentez cette classe en héritant de la classe **CompteBancaire** tout en respectant la description UML fournie.

Lors de la création d'un compte épargne, permettez l'ouverture du compte avec un solde initial ou, par défaut, avec un solde de 0.

10. Codage des Méthodes de CompteEpargne :

Implémentez les méthodes nécessaires, y compris **CalculerInterets** pour calculer les gains en fonction du solde et du taux d'intérêt, et **ModifierTaux** pour mettre à jour le taux d'intérêt.

Assurez-vous que le taux d'intérêt est toujours exprimé en pourcentage (par exemple, 3 %).

11. Documentation avec Doxygen :

Documentez la classe **CompteEpargne**, ses méthodes, et tous les fichiers associés à cette classe. Utilisez des commentaires Doxygen pour générer une documentation claire et complète.

Générez cette documentation et placez-la dans le sous-dossier Documentation de **ProjetBanque**.

12. Création d'un Fichier Texte compteEpargne.txt :

Créez un fichier texte nommé compteEpargne.txt qui servira à tester les fonctionnalités de la classe CompteEpargne.

Ce fichier devrait proposer un menu interactif pour permettre les actions telles que l'ouverture d'un compte, le calcul des intérêts, la modification du taux, le dépôt, et le retrait.

```
Terminal
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
+-----+
| 1 | Consulter le solde
| 2 | Effectuer un depot
| 3 | Effectuer un retrait
| 4 | Calculer les interets
| 5 | Retour
+-----+
Votre choix : entre 1 et 5
|
```

13. Test du Compte Épargne :

Commentez le premier programme principal que vous avez créé pour le compte bancaire. **Créez une nouvelle fonction main** afin de tester les fonctionnalités de la classe CompteEpargne en utilisant les options du menu défini dans compteEpargne.txt.

E) ÉTAPE N°3 : COMPTE CLIENT

Dans cette étape, vous allez créer un compte client qui regroupe un compte bancaire et éventuellement un compte épargne. Cette classe permet de gérer les opérations pour un client de manière centralisée et dynamique.

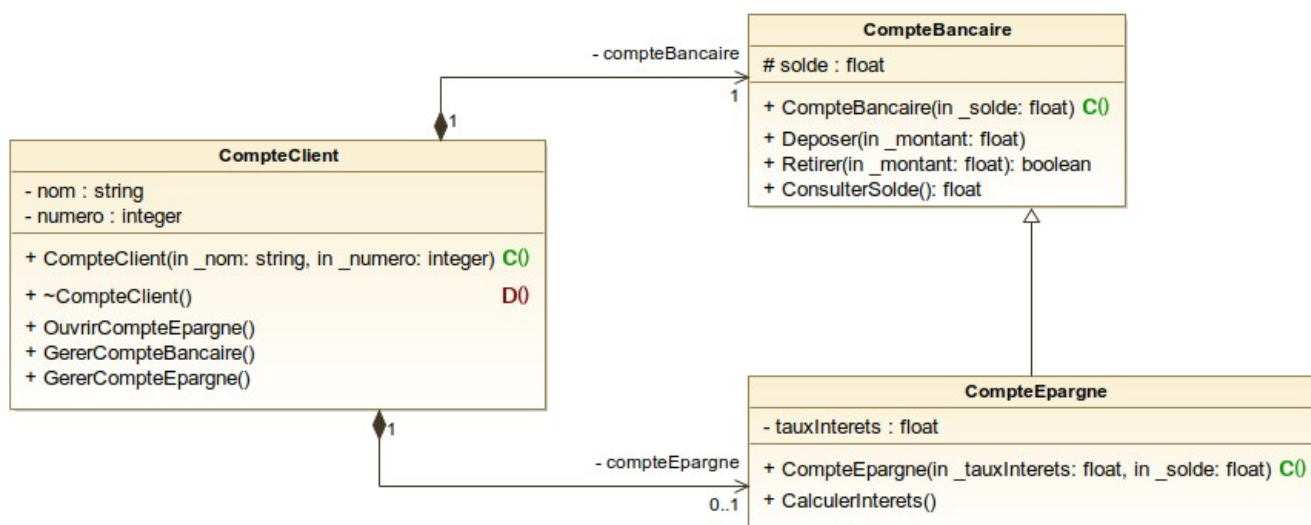
Description du Compte Client :

Lorsqu'un client rejoint la banque, un **compte client** est créé, contenant obligatoirement un **compte bancaire** pour les opérations courantes et éventuellement un **compte épargne** si le client le désire.

Un **compte client** est limité à un seul compte bancaire et, au maximum, un seul compte épargne.

Modélisation UML :

La classe CompteClient complète la modélisation UML de votre application. Assurez-vous de respecter les relations définies précédemment (composition et héritage).



14. Déclaration de la Classe **CompteClient** :

Créez la classe **CompteClient** dans votre projet **LaBanque** sous **QtCreator**. Implémentez les relations dynamiques avec **CompteBancaire** et **CompteEpargne** (utilisation de pointeurs pour gérer les comptes dynamiquement).

15. Constructeur de **CompteClient** :

Le constructeur initialise les attributs **nom** et **numéro de compte** du client.

Le pointeur **compteEpargne** doit être initialisé à **nullptr**.

Allouez dynamiquement un **CompteBancaire** avec un solde initial de zéro.

16. Destructeur de **CompteClient** :

Libérez la mémoire allouée pour le **CompteBancaire**. Si le **CompteEpargne** existe, libérez également la mémoire allouée pour ce compte.

17. Méthode **OuvrirCompteEpargne()** :

Cette méthode doit vérifier si un compte épargne existe déjà. Si ce n'est pas le cas, demandez à l'utilisateur le montant du solde de départ et le taux d'intérêt. Allouez dynamiquement la mémoire pour un **CompteEpargne** avec les valeurs fournies par l'utilisateur. Si un compte épargne existe déjà, affichez un message indiquant que l'ouverture d'un second compte épargne n'est pas autorisée.

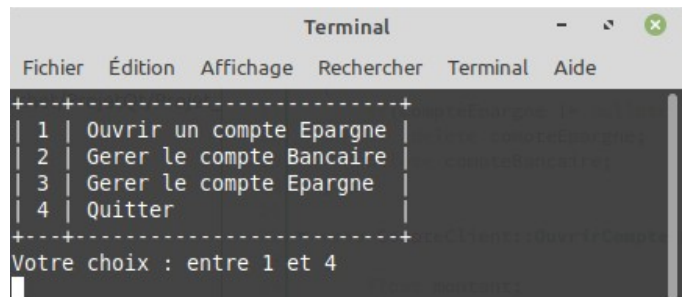
18. Réutilisation des Méthodes de **Compte Bancaire et Épargne** :

Intégrez le code des méthodes déjà implémentées pour les comptes bancaires et épargne dans **CompteClient**. Adaptez ces méthodes pour tenir compte de la gestion dynamique des instances de **CompteBancaire** et **CompteEpargne**.

19. Création d'un Fichier **client.txt** :

Créez un fichier texte nommé **client.txt** qui propose un menu pour gérer les opérations du client, telles que :

- Ouvrir un compte épargne
- Consulter les soldes
- Effectuer des dépôts ou des retraits sur les comptes.



```
Terminal
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
+-----+
| 1 | Ouvrir un compte Epargne |
| 2 | Gerer le compte Bancaire |
| 3 | Gerer le compte Epargne |
| 4 | Quitter |
+-----+
Votre choix : entre 1 et 4
```

20. Programme Principal :

Commentez le programme principal utilisé pour l'étape précédente. Créez un nouveau programme principal pour gérer un client nommé **Albert** avec un numéro de compte **1**.

Ce programme principal doit permettre de tester toutes les fonctionnalités du compte client, y compris la gestion des deux types de comptes.

21. Documentation avec **Doxygen** :

Finalisez le commentaire de votre code puis générez la documentation avec **Doxygen** dans le dossier prévu à cet effet.

Réalisez ensuite le dernier commit et rédigez le fichier **Read.me** précisant le contexte de ce TD.