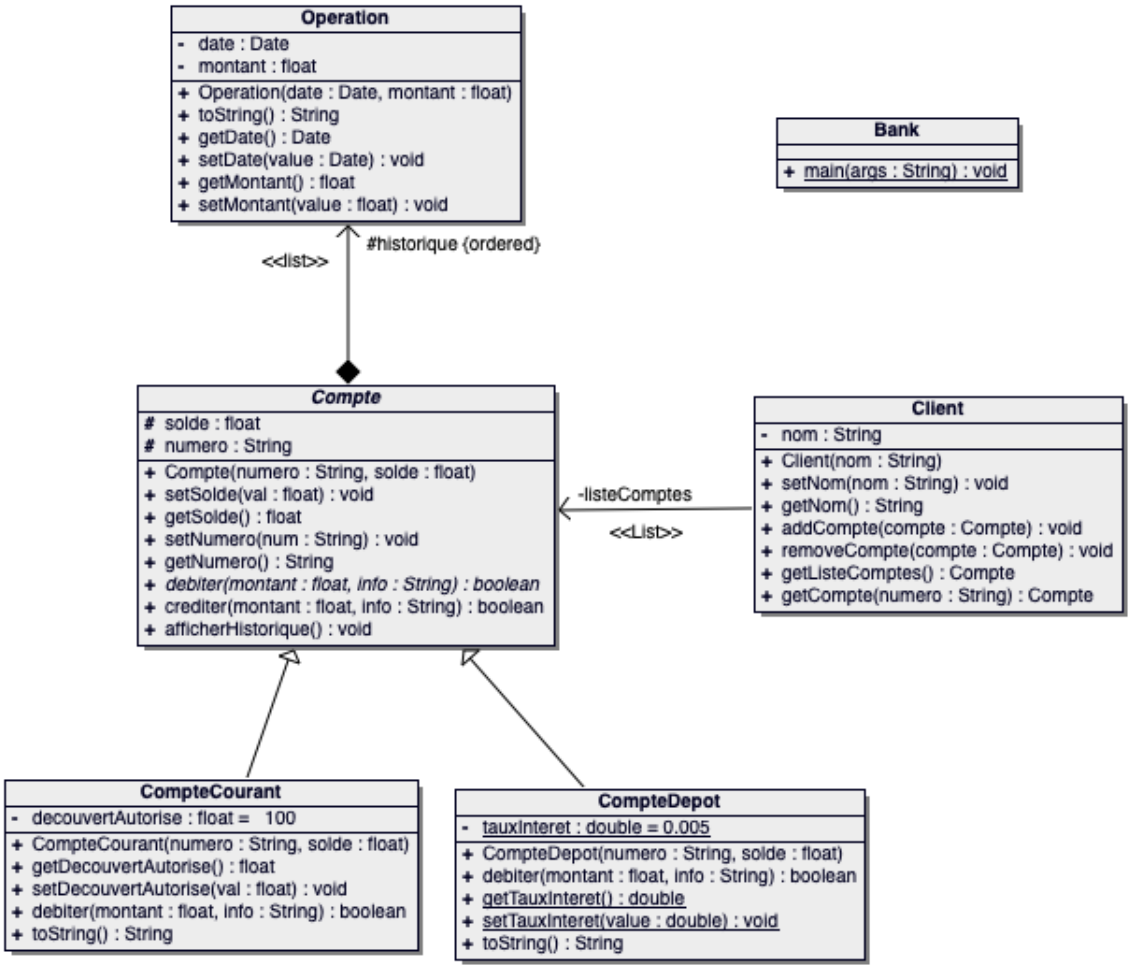
FEUILLE TP N°3 *AGL et Pattern Observateur*

Objectifs

- Pratiquer un développement synchronisé (conception/implémentation) à l'aide d'un AGL
- Proposer une conception mettant en œuvre le pattern Observateur
- Implémenter le pattern Observateur en Java

L'objectif de ce TP est de mettre en œuvre le design pattern *observateur* sur l'application *Bank* en cours de développement. Pour cela vous pourrez reprendre votre programme obtenu en fin de TP2 ou repartir de l'archive proposée sous Célène de sorte à disposer d'une conception proche du diagramme ci-dessous :



Vous devez offrir aux clients de la banque une fonctionnalité leur permettant d'être notifiés en temps réel des changements de situation de leur compte. Un client devra pour cela installer une extension logicielle sur le ou les terminaux (tablette, smartphone, etc.) sur lesquels il souhaite recevoir ces notifications. Dans la suite du TP nous allons simplifier en considérant que les terminaux à notifier sont de simples fenêtres graphiques.

Exercice 1 *Phase de conception (dans BoUML)*

1.1 Vous devez commencer par ajouter à votre diagramme de classe :

- une classe `JFrame` (prédéfinie dans la librairie *javax.swing*), qui permet l'affichage d'une fenêtre graphique et dans laquelle il sera possible d'ajouter (`.add()`) des composants graphiques, par exemple une étiquette (`JLabel`, également prédéfinie dans *javax.swing*).
- une classe `Terminal` qui hérite de `JFrame`. Un `:Terminal` est composé d'un *message* de type `JLabel`

1.2 Identifiez dans votre diagramme actuel, la classe qui jouera le rôle d'observable et celle qui jouera le rôle d'observateur puis :

- créez une classe (abstraite) `Observable` ainsi qu'une relation de généralisation (héritage) avec la classe qui convient
- créez une interface[1] `IObservateur` ainsi qu'une relation de réalisation (implémentation) avec la classe qui convient
- modélisez le fait qu'un *`:Observable`* dispose d'une liste d'observateurs à notifier

1.3 Ajoutez dans les bonnes classes :

- une opération `notifierObservateurs(String message)` : permettant de notifier tous les *observateurs* d'un *observable*
- une opération `notification(String message)` : pour permettre à un *observateur* de traiter la notification reçue d'un *observable*

⚠ Avant d'aller plus loin vous devez faire valider votre conception par votre enseignant !

1.4 Terminez par la génération du code des nouvelles classes (`Terminal`, `Observable` et `IObservateur`) et des classes modifiées (Compte). En revanche ne générez pas les classes `JFrame` et `JLabel` qui sont prédéfinies dans la librairie *javax.swing*.

Exercice 2 *Implémentation du pattern Observateur*

Pour chacune des classes à finaliser, complétez la définition dans VSCode puis synchronisez avec votre conception BoUML (*roundrip*).

2.1 Complétez la définition de la classe `Terminal` :

- ajoutez un constructeur `Terminal(String titre)` qui initialise un nouveau `:Terminal` ainsi :
  - affichage du message par défaut *"Compte ok !"* (`JLabel`)
  - ajout du titre de la fenêtre (méthode `.setTitle()`)
  - dimensionnement de la fenêtre : par exemple 300x200 (méthode `.setSize()`)
- définition de l'opération `notification(String info)` de sorte à ce que le message affiché par la fenêtre devienne `info`.

2.2 Complétez la définition de la classe `Observable` :

- implémentez la possibilité d'ajouter un observateur
- définissez l'opération `notifierObservateurs(String info)`

1. Dans BoUML une interface est une classe que l'on précise grâce au stéréotype *interface*. Observez au passage la notation UML d'une interface dans BoUML.

2.3 Fonctionnement du pattern :

- modifiez le menu principal de sorte à permettre l'ajout d'un `:Terminal` affichant les informations d'alertes sur un compte
- faites en sorte que le `:Terminal` affiche le solde du compte après chaque crédit ou débit sur le compte
- testez le fonctionnement du `:Terminal` sur un scénario d'utilisation

→ **Déposez votre diagramme de classe final sur Célène** et n'oubliez pas de sauvegarder votre projet (conception BoUML et code Java) !