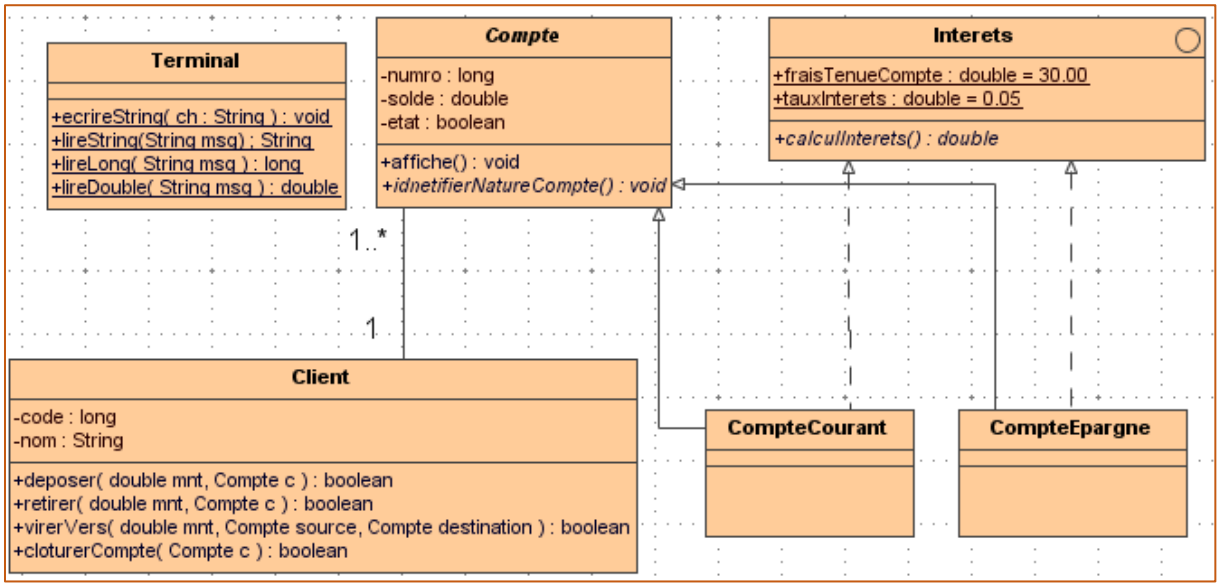
	Direction Générale des Etudes Technologiques **Institut Supérieur des Etudes Technologiques de Djerba** **Département Technologies de l'Informatique**
Classes : L2-MDW, L2-DSI	**Date :** Mardi 11/01/2022
Durée : 1h30 (08h30 – 10h00)	**Année universitaire :** 2020-2021 (Semestre 1)
Enseignant : Anis ASSAS	**Documents et calculatrice :** non autorisés.

Devoir de synthèse : « Programmation JAVA »

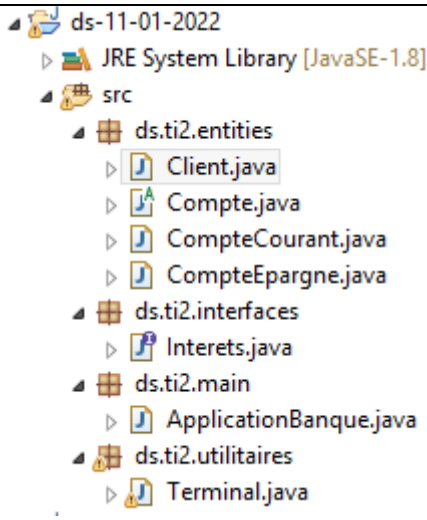
Problème :

On se propose dans ce problème d'implémenter une application Java illustrant un exemple simplifié de gestion de comptes bancaires conformément au diagramme de classes suivant :



Indications importantes :

- *La structure de l'application devrait avoir l'allure ci-jointe.*
- *Pour pouvez utiliser directement les méthodes statiques de* la classe Terminal sans les implémenter afin d'afficher un texte sur la console ou encore lire à partir du terminal une chaine, long ou un réel.
- Vous pouvez également utiliser les getters, les setters ainsi que la méthode toString() relatifs aux différents attributs de toutes les classes sans les implémenter.



1. Créer la classe abstraite « Compte » disposant des propriétés suivantes :
   a. des attributs privés
   b. un constructeur avec 2 paramètres numero et solde (mais aussi avec initialisation de etat à true pour rendre le compte ouvert et activé)
   c. la méthode *'affiche()'* permettant d'afficher les propriétés d'un compte.

d. La méthode *'identifierNatureCompte()'* permettant d'afficher la nature du compte (compte courant ou compte épargne).

2. Créer l'interface « Interets » comportant les deux constantes « fraisTenueCompte » et « tauxInterets » et la méthode « calculInterets() » qui retourne :
   a. Dans le cas où il s'agit d'un compte courant : Frais de la tenue de compte + solde du compte * Taux d'intérêts
   b. Dans le cas où il s'agit d'un compte d'épargne : solde du compte * Taux d'intérêts - Frais de la tenue de compte

3. Créer les deux sous classes « CompteCourant » et « CompteEpargne » qui héritent de « Compte » et implémentent l'interface « Interets » :
   a. La classe «CompteCourant», qui sert aux opérations quotidiennes.
   b. La classe « CompteEpargne », vers lequel on effectue des opérations de transfert d'un montant fixe à partir d'un compte courant.
      Ceci se traduit par :
      - un attribut spécifique « comptesTransferts » du type HashMap spécifiant comme clé le compte courant source et comme valeur le montant du transfert.
      - la méthode booléenne 'transfert ()' qui crédite le compte d'épargne et débite le compte courant en utilisant le montant de transfert (retourne true dans ce cas).
        - Si le compte courant passé comme paramètre ne figure pas au niveau du HashMap, le message d'erreur (compte erroné !) s'affiche.
        - Si le montant du transfert ne correspond pas à la valeur correspondante au compte, le message (Montant erroné !) s'affiche.

4. En supposant que le client peut avoir plusieurs comptes, créer maintenant la classe « Client » disposant des caractéristiques suivantes :

   a. Les attributs privés : code (entier long), nom (chaine) et la liste des comptes qu'il dispose (du type ArrayList)

   b. la méthode booléenne *deposer()* qui étant donné un montant « mnt » permet d'ajouter « mnt » au solde du compte donné en paramètre.
      NB : On devrait vérifier si le compte donné fait partie des comptes du client (dans ce cas, la fonction retourne true et affiche la valeur du solde courant après dépôt, sinon un message d'erreur (Compte erroné !) s'affiche et la fonction retourne dans ce cas false.

   c. la méthode booléenne *retirer ()* qui étant donné un montant « mnt » permet de retirer le montant « mnt » du solde du compte.
      NB : On devrait afficher le message (Compte erroné !) dans le cas où le compte ne figure pas dans la liste ou encore dans le cas où on ne peut pas retirer du compte (Solde insuffisant !).

   d. La méthode booléenne *virerVers()* qui étant donné un montant « mnt » et un compte destination permet de virer le montant « mnt » du compte « source » (débiter ce compte) vers le compte « destination » (créditer ce compte).
      NB : Un message d'erreur (Compte erroné !) si le compte ne figure pas, sinon le message (Solde insuffisant !) s'affiche dans le cas où le solde du compte source est insuffisant pour faire le virement

e. La méthode booléenne cloturerCompte() permettant de mettre à jour le solde en fonction du calcul des intérêts avant de clôturer l'un des comptes du client passé comme paramètre.
   i. Si le compte figure dans la liste des comptes du client, on calcule les intérêts relatifs au compte, qu'on les retranchera du solde dans le cas d'un compte courant sinon on les ajoutera au solde s'il s'agit d'un compte d'épargne.
   On met à jour le solde et on affichera sa valeur avant clôture puis mettre le solde à zéro et changer l'état du compte à false.
   ii. Si le compte ne figure pas dans la liste, le message d'erreur (Compté erroné !) devrait s'afficher.

5. Créer la classe 'ApplicationBanque' ayant le squelette suivant en implémentant toutes les fonctions définies ci-après de sorte de pouvoir :
   a. Saisir les informations nécessaires à partir de la console du terminal afin de créer des objets du type 'Client' et les insérer dans le tableau '*clients*'.
   b. Saisir les informations pour la création des comptes courants et des comptes épargnes et les insérer dans les tableaux dédiés.
   c. On suppose affecter pour chaque client, un compte courant et un compte d'épargne (on suppose faire l'affectation en fonction du même indice).
   d. Simuler les opérations suivantes :
      i. Le client d'indice 0 fait un dépôt de 100.00 DT au niveau de son compte courant puis un retrait de 50 DT
      ii. Le même client fait un virement de son compte courant vers le compte d'épargne du client d'indice 1.
      iii. Le client fait un transfert de 200 dt de son compte courant vers son compte d'épargne.
      iv. Clôturer le compte courant du même client.

```
public class ApplicationBanque {
        private final static int NBRE_CLIENTS = 2;
        private final static int NBRE_COMPTES_COURANTS = 2;
        private final static int NBRE_COMPTES_EPARGNES = 2;
        private static Client[] clients = new Client[NBRE_CLIENTS];
        private static Compte[] comptesCourants=new
                                        CompteCourant[NBRE_COMPTES_COURANTS];
        private static Compte [] comptesEpargnes = new
                                        CompteEpargne[NBRE_COMPTES_EPARGNES];
        public static void creerClients() {
        }

        public static void creerComptesCourants() {
        }

        public static void creerComptesEpargnes() {
        }

        public static void main(String[] args) {
        }
```

Bon travail ☺

Barème provisoire :

Q1 (3 points)				Q2	Q3 (6 points)		Q4 (6 points)					Q5 (4 points)			
a	b	c	d		a	b	a	b	C	d	e	a	b	c	d
1 pt	1 pt	0.5 pt	0.5 pt	1 pt	2 pts	4 pts	1 pt	1 pt	1 pt	1 pt	2 pts	1 pt	1 pt	1 pt	1 pt