جامعة ابن زهر
ⵜⴰⵙⴷⴰⵡⵉⵜ ⵏ ⵖⴱⵔ ⵅⵓⴱⵓⵇ
UNIVERSITÉ IBN ZOHR

Filière d'excellence ADIA

TRAVAUX PRATIQUE DE JAVA

Réaliser par :

TARZOUKT Abderrahim

Encadré par :

M.OUKDACH Yassine

Gestion des comptes bancaires

- # Classe Agence

```java
import java.util.Arrays;


public class Agence {
   private int numero ;
   private String adresse ;
   private Client[] lesClients;
   private Compte[] lesComptes;

   private static int nbClients = 0;
   private static int nbAgences = 0;
   private static int nbComptes = 0;


   public final static int NB_MAX_COMPTES = 100;
   public final static int NB_MAX_CLIENTS = 100;

   public Agence (String adresse) {
    numero = ++nbAgences;
    this.adresse = adresse;
    lesClients = new Client[NB_MAX_CLIENTS];
    lesComptes = new Compte[NB_MAX_COMPTES];
}
public void addCompte(Compte c) {
    if(numero < 100)
        lesComptes[nbComptes++] = c;
    else System.out.println(x:"Vous avez atteindre le nombre max du comptes que vous pouvez avoir");
}
public void addClient(Client c) {
    if(nbClients < NB_MAX_CLIENTS)
        lesClients[nbClients++] = c;
    else System.out.println(x:"Vous avez atteindre le nombre max du clients que vous pouvez avoir");
}
```

```java
public int getNumero() {
    return numero ;
}
public void setNumero(int numero) {
    this.numero = numero;
}
public String getAdresse() {
    return adresse ;
}
public void setAdresse(String adresse){
    this.adresse=adresse ;
}
public Compte getCompte(int n) {
    return lesComptes[n];
}
public Client getClient(int n) {
    return lesClients[n] ;
}
public static int getnbClients(){
      return nbClients ;
}
public static int getnbCompte(){
    return nbComptes ;
}
public static int getnbAgence(){
    return nbAgences ;
}
@Override
public String toString() {
    return "Agence{" +"numAgence=" + numero
    +", adresse='" + adresse + '\'' +
     ", lesClients=" + Arrays.toString(lesClients) +
     ", lesComptes=" + Arrays.toString(lesComptes) + '}';
}
```

Attributs de classe:

NB_MAX_COMPTES et NB_MAX_CLIENTS: Ce sont des constantes qui définissent la capacité maximale d'une agence en termes de comptes bancaires et de clients respectivement.

Attributs d'instance:

numero: Un identifiant unique attribué à chaque instance de la classe, représentant le numéro de l'agence.

adresse: Une chaîne de caractères représentant l'adresse de l'agence.

lesClients: Un tableau de clients qui stocke les clients associés à cette agence.

lesComptes: Un tableau de comptes bancaires qui stocke les comptes associés à cette agence.

Attributs de classe statiques:

nbAgences, nbComptes, nbClients: Ce sont des variables statiques qui comptent le nombre total d'agences, de comptes et de clients dans toutes les instances de la classe Agence.

Constructeur:

Il s'agit d'un constructeur qui prend l'adresse de l'agence en tant que paramètre. L'identifiant de l'agence est incrémenté automatiquement à chaque nouvelle instance. Les tableaux lesClients et lesComptes sont initialisés avec la capacité maximale définie par les constantes.

Méthodes addCompte et addClient:

Ces méthodes permettent d'ajouter des comptes et des clients à l'agence respectivement. Avant d'ajouter, elles vérifient si la capacité maximale n'a pas été atteinte.

Méthodes Getters et Setters:

Il existe des méthodes pour récupérer et définir les valeurs des attributs (getNumAgence, getAdresse, setAdresse, etc.).

Méthodes getCompte et getClient:

Ces méthodes permettent de récupérer un compte ou un client spécifique à partir de leur position dans les tableaux.

Méthodes statiques getnbAgences, getnbComptes, getnbClients:

Ces méthodes statiques fournissent le nombre total d'agences, de comptes et de clients dans toutes les instances de la classe Agence.

Méthode toString:

Cette méthode est héritée de la classe Object et est redéfinie pour retourner une représentation sous forme de chaîne de caractères de l'objet Agence. Elle affiche l'identifiant de l'agence, son adresse, la liste des clients et la liste des comptes.

- CLASS CLIENT :

```java
import java.util.Arrays;

public class Client {
   public final int NB_MAX_COMPTE = 4;
   private int code ;
   private String nom ;
   private String prenom ;
   private String adresse ;
   private Agence monAgence ;
   private Compte mesComptes[] ;

   private int nbComptes = 0 ;
   private static int nbClients = 0;
   private static double soldeTotal = 0.0;


   public Client(String nom, String prenom , String adresse , Agence agence) {
      code = ++ nbClients ;
      this.nom = nom ;
      this.prenom = prenom ;
      this.adresse = adresse ;
      this.monAgence = agence ;
      mesComptes = new Compte[NB_MAX_COMPTE] ;
   }


    public void addCompte(Compte c){
        if(nbComptes < NB_MAX_COMPTE)
            mesComptes[nbComptes++]=c;
        else System.out.println(x:"Vous avez atteindre le nombre max du comptes que vous pouvez avoir");
    }
```

```java
public void deposer(int nCompte , double montant){
    mesComptes[nCompte].deposer(montant) ;
}
public void retirer(int nCompte  , double montant){
    mesComptes[nCompte].retirer(montant);
}

public Compte getCompte(int n ){
    return mesComptes[n];
}
public int getCode(){
    return code ;
}
public String getNom(){
    return nom ;
}
public String getPrenom(){
    return prenom ;
}
public String getAdresse(){
    return adresse ;
}
public Agence getMonAgence(){
    return monAgence ;
}
public int getnbClients(){
    return nbClients;
}
public double getSoldeTotal() {
    for (int i = 0; i < getnbCompte(); i++) {
        soldeTotal += getCompte(i).getSolde();
    }
    return soldeTotal;
}
```

```java
public void setAdresse(String adresse) {
    this.adresse=adresse;
}
public void setMonAgence(Agence monAgence){
    this.monAgence=monAgence;
}
public int getCompte(){
    return nbComptes;
}
public int getnbCompte(){
    return nbComptes;
}
@Override
public String toString (){
   return "Client{" +
   "code=" + code +
   ", nom='" + nom + '\'' +
   ", prenom='" + prenom + '\'' +
   ", adresse='" + adresse + '\'' +
   ", mesComptes=" + Arrays.toString(mesComptes) +
   '}';
}
```

Attributs de classe:

NB_MAX_COMPTE: Nombre maximal de comptes qu'un client peut avoir.

Attributs d'instance:

code: Identifiant unique du client.

nom et prenom: Nom et prénom du client.

adresse: Adresse du client.

monAgence: Référence à l'agence à laquelle le client est associé.

mesComptes: Tableau des comptes du client.

nbCompte: Nombre de comptes que le client possède.

Attributs de classe statiques:

soldeTotal: solde total de clients de cescomptes .

Méthodes:

addCompte: Ajoute un compte au tableau si le nombre maximal n'est pas atteint.

deposer et retirer: Opérations de dépôt et de retrait déléguées aux comptes du client.

Méthodes statiques:

getNbClients: Retourne le nombre total de clients.

getsoldeTotal : qui parcourt tous les comptes du client et retourne le solde total .

- **CLASS COMPTE :**

```java
public class Compte {
    private int code ;
    protected double solde;
    protected Agence lAgence ;
    protected Client Proprietaire ;

    private static int nbCompte = 0 ;
    public Compte() {}

    public Compte(double solde ){
        this.solde = solde ;
        code = ++nbCompte ;
    }
    public Compte(Client Proprietaire, Agence lAgence) {
        this(solde:100.00, Proprietaire, lAgence);
    }
    public Compte(double solde , Client client , Agence agence){
        code = ++nbCompte;
        this.solde = solde;
        this.Proprietaire = client;
        this.lAgence = agence;
    }

    public void deposer(double montant) {
        solde += montant ;
    }

    public void retirer(double montant) {
        if (montant < solde)
            solde -= montant ;
        else System.out.println(x:"votre solde est insufisant");
    }
```

```java
public int getCode(){
    return code ;
}
 public double getSolde(){
    return solde ;
}
public static void setnbCompte (int nbCompte){
    Compte.nbCompte = nbCompte ;
}
public Agence getlAgence (){
    return lAgence  ;
}
public Client getProprietaire (){
    return Proprietaire  ;
}
@Override
public String toString () {
    return "Le compte "+code+" a un solde de :"+solde +" et il est déposé dans l'agence " + lAgence + '}' ;
}
```

Attributs de classe:

nbCompte: Nombre total de comptes.

Attributs d'instance:

code: Identifiant unique du compte.

solde: Solde du compte.

lAgence: Référence à l'agence associée.

proprietaire: Référence au client propriétaire du compte.

Constructeurs:

Par défaut.

Avec solde initial.

Avec solde initial, client, et agence.

Méthodes:

retirer: Retire un montant du solde.

deposer: Ajoute un montant au solde.

Getters et Setters:

Pour accéder aux attributs du compte.(getCode-getSolde-setnbCompte-getlAgence-getProprietaire)

Méthode toString:

Fournit une représentation textuelle du compte, affichant le solde et le code.

- **CLASS COMPTE EPARGNE :**

```java
public class CompteEpargne extends Compte {

    public final String typeCompteP = "CompteEpargne";
    private double tauxInteret = 6;


   public CompteEpargne(double solde){
    super(solde);
   }

   public CompteEpargne(Client Proprietaire , Agence lAgence){
    super(Proprietaire , lAgence) ;
   }

  public CompteEpargne(double solde , Client client , Agence agence){
    super(solde, client, agence);
  }

  public void CalculInteret(){
    super.deposer(getSolde() * tauxInteret / 100);
  }

  public double getTauxInteret(){
    return tauxInteret;
  }

  public void setTauxInteret(double tauxInteret){
    this.tauxInteret = tauxInteret ;
  }
  public String getTypeCompteP() {
    return typeCompteP;
  }
  @Override
    public String toString() {
        return "CompteEpargne{" +
                "solde=" + super.getSolde() +
                ", code=" + super.getCode() +
                '}';
    }
}
```

La classe Java CompteEpargne étend la classe Compte et représente un type spécifique de compte bancaire, à savoir un compte épargne :

Attributs:

typeCompteP: Une constante indiquant le type de compte comme "CompteEpargne".

tauxInteret: Le taux d'intérêt associé au compte épargne, initialisé à 6%.

Constructeurs:

Avec solde initial.

Avec client et agence associés, avec ou sans solde initial.

Méthodes:

calculInteret: Calcule les intérêts en déposant un montant équivalent à un pourcentage du solde du compte, défini par tauxInteret.

Getters et Setters:

Pour accéder aux attributs du compte épargne.(getTauxInteret-getTypeCompteP)

Méthode toString:

Fournit une représentation textuelle du compte épargne, affichant le solde et le code du compte.

- **CLASS COMPTE PAYANT :**

```java
public class ComptePayant extends Compte {
    private final double TAUX_OPERATION = 5 ;
    public final String typeCompteP="ComptePayant" ;


    public ComptePayant(double solde){
        super(solde);
       }

       public ComptePayant(Client Proprietaire , Agence lAgence){
        super(Proprietaire , lAgence) ;
       }

      public ComptePayant(double solde , Client client , Agence agence){
        super(solde, client, agence);
      }

    public void deposer (double montant){
        super.deposer(montant+TAUX_OPERATION);
    }
     public void retirer (double montant){
        super.retirer(montant-TAUX_OPERATION);
    }
    public String getTypeCompteP() {
        return typeCompteP;
    }
    @Override
    public String toString(){
        return "ComptePayant{" +
        "solde=" + super.getSolde() +
        ", code=" + super.getCode() +
        '}';
    }
}
```

La classe Java ComptePayant étend la classe Compte et représente un type spécifique de compte bancaire, à savoir un compte payant. :

Attributs:

TAUX_OPERATION: Une constante représentant le taux d'opération, fixé à 5%.

typeCompteP: Une constante indiquant le type de compte comme "ComptePayant".

Constructeurs:

Avec solde initial.

Avec client et agence associés, avec ou sans solde initial.

Méthodes:

retirer: Retire un montant du solde du compte en ajoutant le taux d'opération.

deposer: Dépose un montant au solde du compte en déduisant le taux d'opération.

Getters et Setters:

Pour accéder au type de compte.(getTypeCompteP)

Méthode toString:

Fournit une représentation textuelle du compte payant, affichant le solde et le code du compte.

- **Le code de test ou l'ApplicationBancaire(Main):**

Le programme principal ou l'ApplicationBancaire(Main) démontre l'utilisation des classes Agence, Client, Compte, CompteEpargne, et ComptePayant. Voici un résumé des principales étapes et fonctionnalités du programme :

Création de l'agence:

Une instance de la classe Agence est créée avec une adresse spécifiée.

```
public class ApplicationBancaire {

    public final static int NB_CLIENTS = 4;
    private static double soldeTotalClient ;
    Run | Debug
    public static void main(String[] args) {

        //Creation d'une seule agence bancaire
        Agence lAgence = new Agence(adresse:"Inezegane , Eljihadiya , Immeuble Assalam N15");
```

Création des clients et de leurs comptes:

Quatre instances de la classe Client sont créées avec des informations spécifiées.

Différents types de comptes (CompteEpargne et ComptePayant) sont associés à chaque client avec des soldes initiaux.

```
//Creation d'un tableau composé de 4 clients
Client LesClients[] = new Client[NB_CLIENTS] ;
LesClients[0]=new Client(nom:"Tarzoukt", prenom:"Abderrahim", adresse:"Inezgane", lAgence);
LesClients[1]=new Client(nom:"ELGRHIYATI", prenom:"Ayman", adresse:"Ait Melloul", lAgence);
LesClients[2]=new Client(nom:"Elassali", prenom:"Kawtar", adresse:"Dcheira", lAgence);
LesClients[3]=new Client(nom:"Mjahed", prenom:"Ouael", adresse:"Dakhla", lAgence);
```

Opérations sur les comptes:

Des opérations de dépôt et de retrait sont effectuées sur certains comptes de clients.

```
// Client 1
LesClients[0].addCompte(new CompteEpargne(solde:1000));
// Client 2
LesClients[1].addCompte(new ComptePayant(solde:2500));
// Client 3
LesClients[2].addCompte(new ComptePayant(solde:0));
LesClients[2].addCompte(new ComptePayant(solde:3000));
// Client 4
LesClients[3].addCompte(new CompteEpargne(solde:2300));
LesClients[3].addCompte(new ComptePayant(solde:0));
```

Ajout des clients à l'agence:

Les clients sont ajoutés à l'agence à l'aide de la méthode addClient.

```
// Ajout des clients et ses comptes a l'agence
for(int i = 0; i < LesClients.length; i++) {
    lAgence.addClient(LesClients[i]);
}
```

Calcul des intérêts sur les comptes d'épargne:

La méthode calculInteret est appliquée sur tous les comptes d'épargne de tous les clients de l'agence.

```
// Application de la méthode calculInteret() sur tous les comptes d'épargne
for(int i = 0; i < Agence.getnbClients(); i++) {
    for(int j = 0; j < lAgence.getClient(i).getnbCompte(); j++) {
        if (lAgence.getClient(i).getCompte(j) instanceof CompteEpargne) {
            ((CompteEpargne)lAgence.getClient(i).getCompte(j)).CalculInteret();
        }
    }
}
```

Affichage des clients et de leurs comptes:

Les informations sur tous les clients et leurs comptes sont affichées.

```
// Liste des différents clients avec leurs différents comptes

System.out.println(x:"Liste des differents clients avec leurs differents comptes ");

for(int i = 0; i < Agence.getnbClients(); i++) {
    System.out.println(lAgence.getClient(i).toString());
}
```

Affichage des comptes d'épargne de l'agence:

Les informations sur tous les comptes d'épargne de l'agence sont affichées.

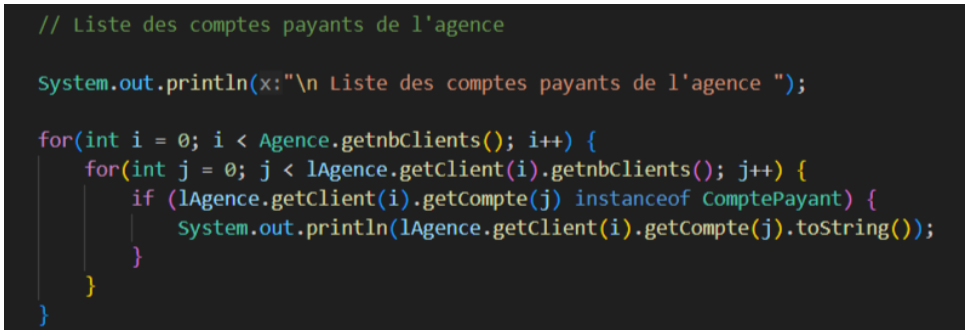
```
// Liste des comptes d'épargne de l'agence

System.out.println(x:"\n Liste des comptes d'epargne de l'agence");

for(int i = 0; i < Agence.getnbClients(); i++) {
    for(int j = 0; j < lAgence.getClient(i).getnbCompte(); j++) {
        if (lAgence.getClient(i).getCompte(j) instanceof CompteEpargne) {
            System.out.println(lAgence.getClient(i).getCompte(j).toString());
        }
    }
}
```

Affichage des comptes payants de l'agence:

Les informations sur tous les comptes payants de l'agence sont affichées.



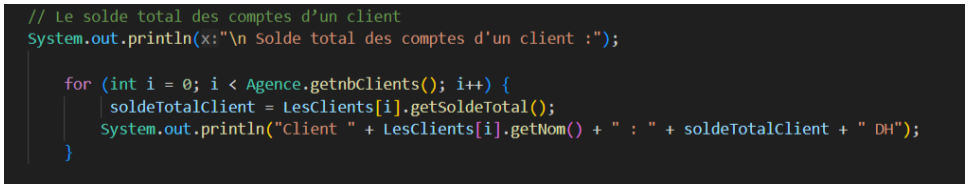
```
// Liste des comptes payants de l'agence

System.out.println(x:"\n Liste des comptes payants de l'agence ");

for(int i = 0; i < Agence.getnbClients(); i++) {
    for(int j = 0; j < lAgence.getClient(i).getnbClients(); j++) {
        if (lAgence.getClient(i).getCompte(j) instanceof ComptePayant) {
            System.out.println(lAgence.getClient(i).getCompte(j).toString());
        }
    }
}
```

Affichage de solde total des comptes d’un client :

Laffichage de solde total des compte d’un client en utilisant une boucle for et la methode getters de la classe Client getSoldeTotal.



```
// Le solde total des comptes d’un client
System.out.println(x:"\n Solde total des comptes d'un client :");

    for (int i = 0; i < Agence.getnbClients(); i++) {
        soldeTotalClient = LesClients[i].getSoldeTotal();
        System.out.println("Client " + LesClients[i].getNom() + " : " + soldeTotalClient + " DH");
    }
```

- **RESULTS a partir du TERMINAL:**

```
PS C:\Gestion Compte bancaire>  c:; cd 'c:\Gestion Compte bancaire'; & 'C:\Program Files\Java\jdk-21\bin\java.exe' '--enable-preview' '-XX:+ShowCodeDetailsInE
xceptionMessages' '-cp' 'C:\Users\abderr\AppData\Roaming\Code\User\workspaceStorage\f724060f7dddbf3b969073c7b1b42ec8\redhat.java\jdt_ws\Gestion Compte bancair
e_ba0e96c0\bin' 'ApplicationBancaire'
votre solde est insufisant
Liste des differents clients avec leurs differents comptes
Client{code=1, nom='Tarzoukt', prenom='Abderrahim', adresse='Inezgane', mesComptes=[CompteEpargne{solde=2120.0, code=1}, null, null, null]}
Client{code=2, nom='ELGRHIYATI', prenom='Ayman', adresse='Ait Melloul', mesComptes=[ComptePayant{solde=2500.0, code=2}, null, null, null]}
Client{code=3, nom='Elassali', prenom='Kawtar', adresse='Dcheira', mesComptes=[ComptePayant{solde=0.0, code=3}, ComptePayant{solde=3000.0, code=4}, null, null
]}
Client{code=4, nom='Mjahed', prenom='Ouael', adresse='Dakhla', mesComptes=[CompteEpargne{solde=2438.0, code=5}, ComptePayant{solde=0.0, code=6}, null, null]}

 Liste des comptes d'epargne de l'agence
CompteEpargne{solde=2120.0, code=1}
CompteEpargne{solde=2438.0, code=5}

 Liste des comptes payants de l'agence
ComptePayant{solde=2500.0, code=2}
ComptePayant{solde=0.0, code=3}
ComptePayant{solde=3000.0, code=4}
ComptePayant{solde=0.0, code=6}

 Solde total des comptes d'un client :
Client Tarzoukt : 2120.0 DH
Client ELGRHIYATI : 4620.0 DH
Client Elassali : 7620.0 DH
Client Mjahed : 10058.0 DH
```