جامعة ابن زهر
ⵜⴰⵙⴷⴰⵡⵉⵜ ⵏ ⵉⴱⵏ ⵣⵀⵔ
UNIVERSITÉ IBN ZOHR

Filière d'excellence ADIA

TRAVAUX PRATIQUE DE JAVA

Réaliser par :
EL RHYATI Aymane

Encadré par :
M.OUKDACH Yassine

Gestion des comptes bancaires

CLASS AGENCE :

```java
import java.io.Serializable;
import java.util.Arrays;

public class Agence implements Serializable{

    // Attributs
    public final static int COMPTES = 80;
    public final static int CLIENTS = 30;


    private int numAg;
    private String adresse;
    private Client[] lClients;
    private Compte[] lComptes;

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

    // Constructors
    public Agence (String adresse) {
        numAg = ++nbAgences;
        this.adresse = adresse;
        lClients = new Client[CLIENTS];
        lComptes = new Compte[COMPTES];
    }

    // Methods
    public void addCompte(Compte nb) {
        if(nbComptes < COMPTES)
            lComptes[nbComptes++] = nb;
        else System.out.println("Vous avez atteindre le nombre max du comptes que vous pouvez avoir");
    }

    public void addClient(Client c) {
        if(nbClients < CLIENTS)
            lClients[nbClients++] = c;
        else System.out.println("Vous avez atteindre le nombre max du clients que vous pouvez avoir");
    }

    // Getters and Setters
    public int getNumAgence() {
        return numAg;
    }

    public void setNumAgence(int numAg) {
        this.numAg = numAg;
    }
```

```java

    public String getAdresse() {
        return adresse;
    }

    public void setAdresse(String adresse) {
        this.adresse = adresse;
    }

    public Compte getCompte(int n) {
        if(n>=0 && n<lComptes.length){return lComptes[n];}
        else return null;
    }

    public Client getClient(int n) {
        return lClients[n];
    }

    public static int getNbAgences() {
        return nbAgences;
    }

    public static int getNbComptes() {
        return nbComptes;
    }

    public static int getNbClients() {
        return nbClients;
    }

    @Override
    public String toString() {
        return "Agence{" +
                "numAgence=" + numAg +
                ", adresse='" + adresse + '\'' +
                ", lesClients=" + Arrays.toString(lClients) +
                ", lesComptes=" + Arrays.toString(lComptes) +
                '}';
    }

}
```

Attributs de classe:

COMPTES et 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:

numAg: 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.

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

lComptes: 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 lClients et lComptes 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.io.Serializable;
import java.util.Arrays;

public class Client implements Serializable {

    // Attributes
    public final int MAX_COMPTE = 4;

    private int code;
    private final String nom;
    private final String prenom;
    private String adresse;
    private Agence monAgence;
    Compte[] mesComptes;
    private int nbCompte = 0;

    private static int nbClient = 0;

    // Constructor

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

    // Methods
    public void addCompte(Compte c) {
        if(nbCompte < MAX_COMPTE)
            mesComptes[nbCompte++] = c;
        else System.out.println("Vous avez atteindre le nombre max du comptes que vous pouvez avoir");
    }

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

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

    // Getters and Setters
    public Compte getCompte(int n) {
        return mesComptes[n];
    }
```

```java

    public int getCode() {
        return code;
    }

    public String getNom() {
        return nom;
    }

    public String getPrenom() {
        return prenom;
    }

    public String getAdresse() {
        return adresse;
    }

    public void setAdresse(String adresse) {
        this.adresse = adresse;
    }

    public Agence getMonAgence() {
        return monAgence;
    }

    public void setMonAgence(Agence monAgence) {
        this.monAgence = monAgence;
    }

    public int getNbCompte() {
        return nbCompte;
    }

    public static int getNbClient() {
        return nbClient;
    }

    @Override
    public String toString() {
        return "Client{" +
                "code=" + code +
                ", nom='" + nom + '\'' +
                ", prenom='" + prenom + '\'' +
                ", adresse='" + adresse + '\'' +
                ", mesComptes=" + Arrays.toString(mesComptes) +
                '}';
    }
```

Attributs de classe:

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:

nbClient: Nombre total de clients dans toutes les instances.

Méthodes:

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

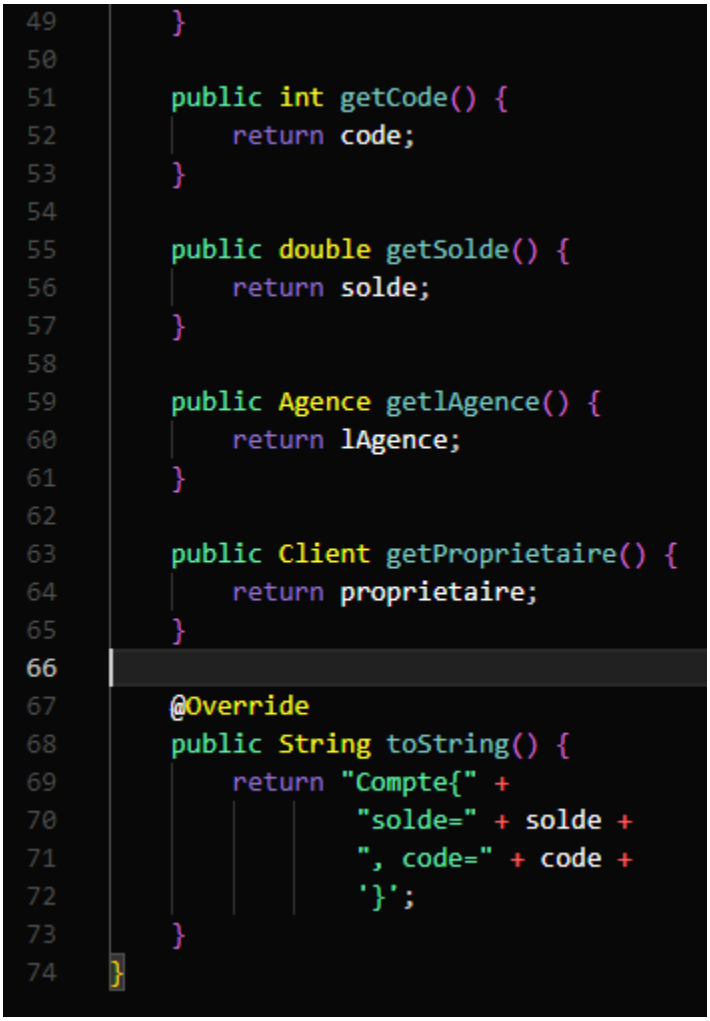
deposer et retirer: Opérations de dépôt et de retrait déléguées aux comptes du client.

Méthodes statiques:

getNbClient: Retourne le nombre total de clients.

CLASS COMPTE :

```java
import java.io.Serializable;

public class Compte implements Serializable {

    // Attributs
    private static int nbCompte = 0;

    protected int code;
    protected double solde;
    protected Agence lAgence;
    protected Client proprietaire;

    // Constructors
    public Compte() {}

    public Compte(double solde) {
        this.solde = solde;
        code = ++nbCompte;
    }

    public Compte(Client client, Agence agence) {
        this(solde:50.00, client, agence);
    }

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

    // Methods
    public void retirer(double montant) {
        if (montant < solde) solde -= montant;
    }

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

    // Getters and Setters

    public static int getNbCompte() {
        return nbCompte;
    }

    public static void setNbCompte(int nbCompte) {
        Compte.nbCompte = nbCompte;
```



```
    }

    public int getCode() {
        return code;
    }

    public double getSolde() {
        return solde;
    }

    public Agence getlAgence() {
        return lAgence;
    }

    public Client getProprietaire() {
        return proprietaire;
    }

    @Override
    public String toString() {
        return "Compte{" +
                "solde=" + solde +
                ", code=" + code +
                '}';
    }
}
```

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.

Méthode toString:

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

CLASS COMPTE EMPARGNE :

```java
public class CompteEpargne extends Compte  {

    // Attributs
    public final String typeCompteP = "CompteEpargne";

    private double tauxInteret = 6;

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

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

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

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

    // Getters and Setters
    public String getTypeCompteP() {
        return typeCompteP;
    }

    public double getTauxInteret() {
        return tauxInteret;
    }

    public void setTauxInteret(double tauxInteret) {
        this.tauxInteret = tauxInteret;
    }

    @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.

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 {

    // Attributs
    private final double TAUX_OPERATION = 5;
    public final String typeCompteP = "ComptePayant";

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

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

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

    // Methods
    public void retirer(double mt) {
        super.retirer(mt + TAUX_OPERATION);
    }

    public void deposer (double mt) {
        super.deposer(mt - TAUX_OPERATION);
    }

    // Getters and Setters
    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.

Méthode toString:

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

Le code de test ou le main :

Le programme principal (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.

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.

Sauvegarde des clients dans un fichier:

Les objets Client sont sauvegardés dans un fichier "comptes" à l'aide de la sérialisation.

Opérations sur les comptes:

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

Ajout des clients à l'agence:

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

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.

Affichage des clients et de leurs comptes:

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

Affichage des comptes d'épargne de l'agence:

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

Affichage des comptes payants de l'agence:

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

L’utilisation de Serializable :

L'interface Serializable en Java est une interface vide qui sert à indiquer au compilateur Java qu'une classe peut être sérialisée. La sérialisation est le processus de conversion d'un objet Java en une séquence d'octets, généralement dans le but de sauvegarder l'état de l'objet ou de le transmettre sur un réseau.

RESULTS FROM CMD :

```
clients
Client{code=1, nom='EL RHYATI', prenom='Aymane', adresse='Ait melloul', mesComptes=[CompteEpargne{solde=13144.0, code=1}, null, null, null]}
Client{code=2, nom='Tarzoukt', prenom='Abrahim', adresse='Inezgane', mesComptes=[ComptePayant{solde=550.0, code=2}, null, null, null]}
Client{code=3, nom='Allal', prenom='hmeed', adresse='Agadir', mesComptes=[ComptePayant{solde=4800.0, code=3}, ComptePayant{solde=895.0, code=4}, null, null]}
Client{code=4, nom='bouhmd', prenom='hamiid', adresse='ouleed tieeema', mesComptes=[CompteEpargne{solde=2459.2, code=5}, ComptePayant{solde=4300.0, code=6}, null, null]}

comptes d'epargne de l'agence
CompteEpargne{solde=13144.0, code=1}
CompteEpargne{solde=2459.2, code=5}

 comptes payants de l'agence
ComptePayant{solde=550.0, code=2}
ComptePayant{solde=4800.0, code=3}
ComptePayant{solde=895.0, code=4}
ComptePayant{solde=4300.0, code=6}
```