Travaille à rendre

Réaliser par :IZIKKI Hajar

Prof encadrant :M.OUKDACH

Gestion des comptes bancaires

Dans cet exercice, nous allons créer un système de gestion de comptes bancaires. Pour ce faire, nous allons implémenter différentes classes telles que **Agence**, **Client**, **Compte**, **CompteEpargne** et **ComptePayant**. Chaque classe aura ses propres attributs et méthodes pour représenter les caractéristiques des entités correspondantes.

Voici un aperçu de ce que nous allons réaliser :

1. Créer les différentes classes avec leurs attributs et méthodes nécessaires.
2. Chaque classe doit avoir un constructeur d'initialisation pour instancier les objets avec un identifiant automatiquement incrémenté.
3. Redéfinir la méthode **toString()** dans chaque classe pour permettre l'affichage des informations d'objet de manière lisible.
4. Développer une classe **ApplicationBancaire** avec plusieurs fonctionnalités :

    - Créer une seule agence bancaire.
    - Créer des clients avec différents types de comptes.
    - Permettre aux clients de déposer et retirer des montants sur leurs comptes.
    - Ajouter les clients et leurs comptes à l'agence.
    - Appliquer le calcul des intérêts sur les comptes d'épargne.
    - Afficher diverses informations telles que la liste des clients avec leurs comptes, les comptes d'épargne et payants de l'agence, le solde total des comptes d'un client, et classer les clients selon leur solde total.

1. Création de la classe Agence :

Ce code représente la classe **Agence** qui modélise une agence bancaire. Voici une explication de ses attributs et méthodes :

Attributs :

- **count** : Il s'agit d'une variable statique qui est utilisée pour attribuer un numéro unique à chaque agence créée. Cette variable est incrémentée à chaque fois qu'une nouvelle agence est créée.
- **numero** : C'est l'identifiant de l'agence, il est généré automatiquement et unique pour chaque instance de l'agence.
- **adresse** : C'est l'adresse de l'agence.
- **lesClients** : C'est une liste qui stocke les clients associés à cette agence.
- **lesComptes** : C'est une liste qui stocke les comptes associés à cette agence.

Méthodes :

- **Agence(String adresse)** : C'est le constructeur de la classe **Agence**. Il prend en paramètre l'adresse de l'agence, initialise les listes de clients et de comptes, puis attribue un numéro à l'agence en utilisant la variable statique **count**.
- **getCompte(int index)** : Cette méthode renvoie le compte correspondant à l'indice donné dans la liste **lesComptes**.
- **getClient(int index)** : Cette méthode renvoie le client correspondant à l'indice donné dans la liste **lesClients**.
- **addCompte(Compte compte)** : Cette méthode ajoute un compte à la liste **lesComptes**.
- **addClient(Client client)** : Cette méthode ajoute un client à la liste **lesClients**.
- **getNbClients()** : Cette méthode renvoie le nombre de clients associés à cette agence.
- **getNbComptes()** : Cette méthode renvoie le nombre de comptes associés à cette agence.
- **toString()** : Redéfinition de la méthode **toString()** pour obtenir une représentation sous forme de chaîne de caractères de l'objet agence, affichant son numéro et son adresse.

Agence.java > Agence > Agence(String)

```java
import java.util.ArrayList;
import java.util.List;

public class Agence {
    private static int count=0;
    private  String numero;
    private  String adresse;
    private List<Client> lesClients;
    private List<Compte> lesComptes;

    // Constructeur
    public Agence(String adresse) {
        this.adresse = adresse;
        this.lesClients = new ArrayList<>();
        this.lesComptes = new ArrayList<>();

        this.numero = "Agence:" +(++count);
    }
```

Agence.java > Agence > Agence(String)

```java
        this.lesComptes = new ArrayList<>();

        this.numero = "Agence:" +(++count);
    }

    public Compte getCompte(int index) {
        return lesComptes.get(index);
    }

    public Client getClient(int index) {
        return lesClients.get(index);
    }

    public void addCompte(Compte compte) {
        lesComptes.add(compte);
    }

    public void addClient(Client client) {
        lesClients.add(client);
    }

    public int getNbClients() {
        return lesClients.size();
    }

    public int getNbComptes() {
        return lesComptes.size();
    }

    public String toString() {
        return "Agence [numero=" + numero + ", adresse=" + adresse + "]";
    }
}
```

2. Creation de la classe Compte :

La classe **Compte** représente un compte bancaire. Voici une explication de ses attributs et méthodes :
Attributs :

- **count** : Il s'agit d'une variable statique utilisée pour attribuer un identifiant unique à chaque compte créé.
- **code** : C'est l'identifiant du compte, qui est généré automatiquement et est unique pour chaque instance de compte.
- **solde** : C'est le solde du compte, représentant la quantité d'argent présente sur ce compte.
- **agence** : C'est l'agence bancaire à laquelle le compte est associé.
- **proprietaire** : C'est le client propriétaire du compte.

Méthodes :

- **Compte(Client proprietaire, Agence agence, double solde)** : C'est le constructeur de la classe **Compte**. Il initialise les attributs **proprietaire**, **agence**, **solde** et génère un identifiant **code** pour le compte.
- **getCode()** : Cette méthode renvoie l'identifiant du compte.
- **setCode(String code)** : Cette méthode permet de définir manuellement l'identifiant du compte.
- **deposer(double montant)** : Cette méthode permet d'ajouter un montant au solde du compte.
- **retirer(double montant)** : Cette méthode permet de retirer un montant du solde du compte.
- **toString()** : Redéfinition de la méthode **toString()** pour obtenir une représentation sous forme de chaîne de caractères de l'objet compte, affichant son code, son solde, l'agence associée et le propriétaire du compte.

Compte.java > Compte

```java
public class Compte {
    private static int count=0;
    protected String code;
    protected double solde;
    protected Agence agence;
    protected Client proprietaire;

    // Constructeur
    public Compte(Client proprietaire, Agence agence, double solde) {
        this.proprietaire = proprietaire;
        this.agence = agence;
        this.solde = solde;
        this.code = "Compte:" + (count ++);
    }


    public String getCode() {
        return code;
    }

    public void setCode(String code) {
        this.code = code;
    }

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

    public void retirer(double montant) {
        this.solde -= montant;
    }
    public String toString() {
        return "Compte [code=" + code + ", solde=" + solde + ", agence=" + agence + ", proprietaire=" + proprietaire + "]";
    }

}

```

3. Création de la classe Compte Epargne :

La classe **CompteEpargne** étend la classe **Compte** et représente un compte d'épargne, qui possède un taux d'intérêt spécifique et permet de calculer les intérêts sur le solde.

Attribut :

- **taux** : C'est le taux d'intérêt du compte d'épargne, initialisé à 0.06 (6%) dans le constructeur.

Constructeur :

- **CompteEpargne(Client proprietaire, Agence agence, double solde)** : Le constructeur initialise un compte d'épargne avec le propriétaire, l'agence et le solde donnés et initialise le taux d'intérêt à 6%.

Méthodes :

- **getTaux()** : Cette méthode renvoie le taux d'intérêt du compte d'épargne.
- **setTaux(double taux)** : Cette méthode permet de modifier le taux d'intérêt du compte d'épargne.
- **calculInteret()** : Cette méthode calcule les intérêts sur le solde du compte d'épargne en fonction du taux d'intérêt et les ajoute au solde actuel du compte.
- **toString()** : Redéfinition de la méthode **toString()** pour obtenir une représentation sous forme de chaîne de caractères de l'objet compte d'épargne, affichant son code, son solde, l'agence associée, le propriétaire du compte et le taux d'intérêt.

Cette classe permet donc de gérer les comptes d'épargne en définissant un taux d'intérêt, permettant de calculer et d'ajouter les intérêts au solde du compte, et en

offrant des méthodes pour obtenir et modifier le taux d'intérêt ainsi que pour obtenir une représentation textuelle de l'objet.

4. Création de la classe ComptePayant :

La classe **ComptePayant** étend la classe **Compte** et représente un type spécial de compte bancaire où chaque opération de dépôt et de retrait est associée à des frais fixes.

Attribut :

- **TAUX_OPERATION** : C'est une constante qui détermine le taux fixe pour chaque opération (dépôt ou retrait). Cette valeur est définie comme une constante entière (**final static**) et est initialisée à 5.

Constructeur :

- **ComptePayant(Client proprietaire, Agence agence, double solde)** : Le constructeur initialise un compte payant avec le propriétaire, l'agence et le solde donnés.

Méthodes :

- **deposer(double montant)** : Cette méthode redéfinit la méthode **deposer** de la classe mère (**Compte**). Elle permet de déposer un montant sur le compte en ajoutant le montant spécifié et en déduisant les frais **TAUX_OPERATION**.
- **retirer(double montant)** : Cette méthode redéfinit la méthode **retirer** de la classe mère (**Compte**). Elle permet de retirer un montant du compte en soustrayant le montant spécifié et en ajoutant les frais **TAUX_OPERATION**.
- **toString()** : Redéfinition de la méthode **toString()** pour obtenir une représentation sous forme de chaîne de caractères de l'objet compte payant, affichant son code, son solde, l'agence associée et le propriétaire du compte.

Cette classe **ComptePayant** permet de gérer des comptes bancaires spéciaux où des frais fixes sont appliqués à chaque opération de dépôt ou de retrait, en redéfinissant les méthodes de dépôt et de retrait héritées de la classe **Compte** pour inclure ces

frais. Elle offre également une méthode pour obtenir une représentation textuelle de l'objet.

5. Creation de la classe Client :

Cette classe **Client** représente un client d'une agence bancaire. Voici une explication de ses attributs et méthodes :

Attributs :

- **count** : Variable statique utilisée pour générer des identifiants uniques pour chaque client.
- **code** : Identifiant unique du client.
- **nom** : Nom du client.
- **prenom** : Prénom du client.
- **adresse** : Adresse du client.
- **monAgence** : Référence vers l'agence à laquelle le client est associé.
- **mesComptes** : Liste des comptes détenus par le client.

Constructeur :

- **Client(String nom, String prenom, String adresse, Agence agence)** : Le constructeur initialise un client avec son nom, prénom, adresse et l'agence à laquelle il est associé. Il incrémente également le compteur **count** pour générer un identifiant unique pour ce client.

Méthodes :

- **getCompte(int index)** : Renvoie le compte à l'indice spécifié dans la liste **mesComptes**.
- **addCompte(Compte compte)** : Ajoute un compte à la liste **mesComptes**.
- **deposer(int index, double montant)** : Appelle la méthode **deposer()** du compte à l'indice spécifié avec le montant spécifié.
- **retirer(int index, double montant)** : Appelle la méthode **retirer()** du compte à l'indice spécifié avec le montant spécifié.
- **toString()** : Redéfinition de la méthode **toString()** pour obtenir une représentation textuelle de l'objet client, affichant son code, son nom, son prénom, son adresse, l'agence associée et la liste de ses comptes.

Cette classe permet de représenter un client de la banque, lui permettant d'effectuer des opérations sur ses comptes associés et de fournir une représentation textuelle de l'objet.

```java
import java.util.ArrayList;
    this.adresse = adresse;
    this.monAgence = agence;
    this.mesComptes = new ArrayList<>();
    this.code = "Client:" + (++ count);
}

public Compte getCompte(int index) {
    return mesComptes.get(index);
}

public void addCompte(Compte compte) {
    mesComptes.add(compte);
}

public void deposer(int index, double montant) {
    mesComptes.get(index).deposer(montant);
}

public void retirer(int index, double montant) {
    mesComptes.get(index).retirer(montant);
}
public String toString() {
    return "Client [code=" + code + ", nom=" + nom + ", prenom=" + prenom + ", adresse=" + adresse + ", monAgence=" + monAgence + ", mesComptes
}
}
```

6. Création de la classe ApplicationBancaire :

1. **Création de l'agence bancaire** : Vous créez une instance de l'agence bancaire en spécifiant son adresse.
2. **Création de clients et de leurs comptes** : Pour chaque client, vous créez un ou plusieurs comptes associés à l'agence bancaire.
    - **Client client1** avec un **CompteEpargne**.
    - **Client client2** avec un **ComptePayant**.
    - **Client client3** avec deux **ComptePayant**.
    - **Client client4** avec un **CompteEpargne** et un **ComptePayant**.
3. **Ajout des clients à l'agence** : Chaque client est ajouté à l'agence bancaire avec leurs comptes associés.
4. **Exemples de dépôts et retraits** : Vous effectuez des opérations de dépôt et de retrait sur certains comptes pour certains clients, pour illustrer le fonctionnement des méthodes.
5. **Application de la méthode de calcul des intérêts sur les comptes d'épargne** : Vous parcourez tous les comptes de l'agence pour appliquer la méthode de calcul d'intérêts sur les comptes d'épargne.
6. **Affichage des informations** :
    - Liste des clients avec leurs comptes.
    - Liste des comptes d'épargne de l'agence.
    - Liste des comptes payants de l'agence.

Cependant, veuillez noter qu'il y a une petite erreur dans la méthode d'affichage des clients avec leurs comptes. Il semble que vous parcourez les comptes de l'agence pour chaque client, mais cela affichera tous les comptes de l'agence pour chaque client, plutôt que les comptes spécifiques de chaque client. Il faudrait plutôt parcourir les comptes associés à chaque client pour obtenir le bon résultat.

```java
public class ApplicationBancaire {
    public static void main(String[] args) {
        System.out.println("Hellooooooooooooooooooo");

        // Créer une seule agence bancaire
        Agence agence = new Agence("Adresse de l'agence");

        // Créer des clients et leurs comptes
        Client client1 = new Client("izikki", "Hajar", "Tamraght", agence);
        CompteEpargne compteEpargne = new CompteEpargne(client1, agence, 1000);
        client1.addCompte(compteEpargne);

        Client client2 = new Client("Fifa", "Afif", "Agadir", agence);
        ComptePayant comptePayant = new ComptePayant(client2, agence, 2500);
        client2.addCompte(comptePayant);

        Client client3 = new Client("Bisaka", "Ahmed", "Casa", agence);
        ComptePayant comptePayant2 = new ComptePayant(client3, agence, 1000);
        ComptePayant comptePayant3 = new ComptePayant(client3, agence, 2500);
        client3.addCompte(comptePayant2);
        client3.addCompte(comptePayant3);


        Client client4 = new Client("Sardi", "Karima", "Marrakech", agence);
        CompteEpargne compteEpargne2 = new CompteEpargne(client4, agence, 0);
        ComptePayant comptePayant4 = new ComptePayant(client4, agence, 3000);
        client4.addCompte(compteEpargne2);
        client4.addCompte(comptePayant4);
```

```java
    //  Ajouter les clients à l'agence
    agence.addClient(client1);
    agence.addClient(client2);
    agence.addClient(client3);
    agence.addClient(client4);

    // Exemple de dépôt et retrait pour un client
    client1.deposer(0, 500); // Exemple : client1 dépose 500 dans son compte d'épargne
    client2.retirer(0, 100); // Exemple : client2 retire 100 de son compte payant
```

```java
    // Affichage ***/// Liste des différents clients avec leurs différents comptes
    System.out.println("--- Liste des differents clients avec leurs differents comptes ---");
    Client client;
    for(int i=0; i<agence.getNbClients(); i++){
    client = agence.getClient(i);
    System.out.println(client.toString());
    for(int j=0; j<agence.getNbComptes(); j++)
    System.out.println(client.getCompte(j).toString());

    }

    // Liste des comptes d'épargne de l'agence
    System.out.println("\n--- Liste des comptes d epargne de l agence ---");
    for(int i=0; i<agence.getNbComptes(); i++)
    if(agence.getCompte(i) instanceof CompteEpargne)
    System.out.println(agence.getCompte(i).toString());

    // Liste des comptes payants de l'agence
    System.out.println("\n--- Liste des comptes payants de l agence ---");
    for(int i=0; i<agence.getNbComptes(); i++)
    if(agence.getCompte(i) instanceof ComptePayant)
    System.out.println(agence.getCompte(i).toString());
    }

    }
```

Résumer :

Agence :

- Représente une agence bancaire avec une adresse et un numéro d'agence incrémenté automatiquement.
- Gère une liste de clients et de comptes associés à cette agence.
- Méthodes permettant d'ajouter des comptes et des clients, de récupérer le nombre de clients et de comptes, et d'accéder à un compte ou un client spécifique.

Client :

- Modèle de client avec des attributs tels que le code, le nom, le prénom, l'adresse, et une agence associée.
- Possède une liste de comptes bancaires.

Compte :

- Classe de base pour différents types de comptes bancaires avec un code, un solde, une agence et un propriétaire associés.
- Méthodes pour déposer et retirer de l'argent du compte.

CompteEpargne :

- Un type spécifique de compte bancaire avec un taux d'intérêt.
- Hérite de la classe Compte avec des fonctionnalités supplémentaires telles que le calcul d'intérêts.

ComptePayant :

- Un autre type spécifique de compte bancaire avec des frais pour les opérations de dépôt et de retrait.
- Hérite également de la classe Compte avec des méthodes redéfinies pour tenir compte des frais.

ApplicationBancaire :

- Classe principale pour tester les fonctionnalités des classes précédentes.
- Crée une agence bancaire, des clients, et leurs différents types de comptes associés.
- Effectue des opérations de dépôt et de retrait sur certains comptes.
- Calcule les intérêts pour les comptes d'épargne.
- Affiche les informations sur les clients, leurs comptes, ainsi que les comptes d'épargne et payants de l'agence.