

# Filière Analytique des Données et Intelligence Artificielle

## Rapport de TP numéro 3

Réalisé par : chaymae belfaik

Encadré par : M.Oukdach

### La classe agence contient six attributs :

numero: Le numéro de l'agence.

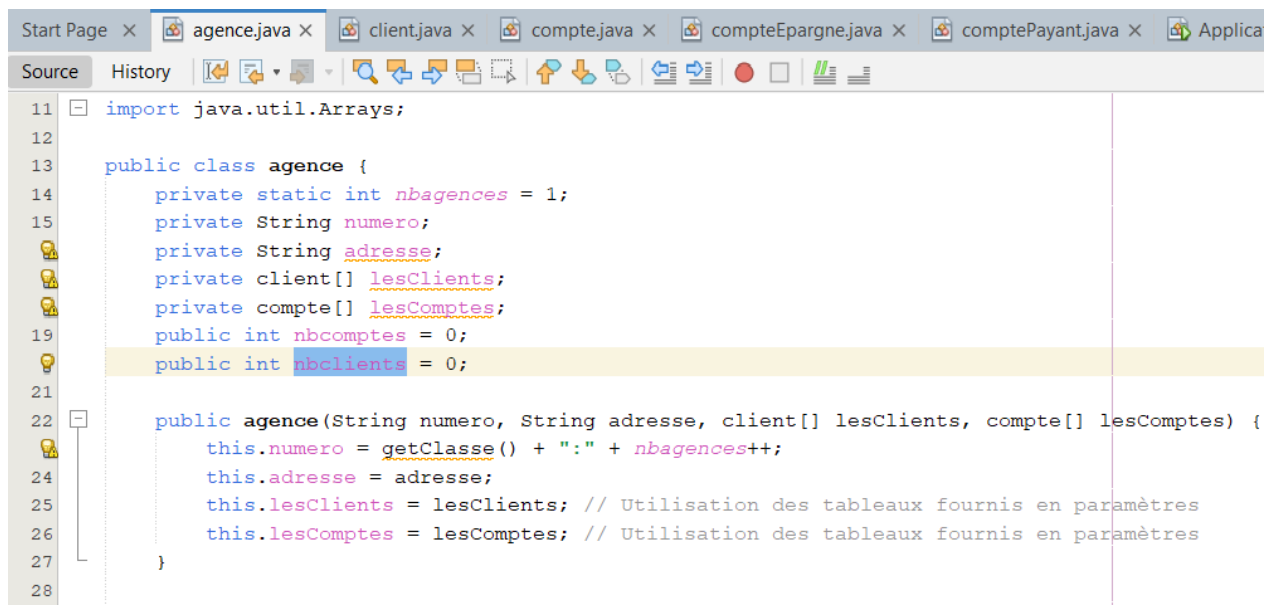
Adresse : L'adresse de l'agence.

lesClients : Un tableau de clients associés à l'agence.

lesComptes : Un tableau de comptes associés à l'agence.

nbcomptes : Le nombre de comptes associés à l'agence.

nbclients : Le nombre de clients associés à l'agence.

The screenshot shows an IDE window with several tabs: 'agence.java', 'client.java', 'compte.java', 'compteEpargne.java', 'comptePayant.java', and 'Applica...'. The 'agence.java' tab is active, showing the source code. The code includes an import statement for 'java.util.Arrays', a class declaration 'public class agence', and several private and public attributes: 'nbagences' (static int), 'numero' (String), 'adresse' (String), 'lesClients' (client[]), 'lesComptes' (compte[]), 'nbcomptes' (int), and 'nbclients' (int). A constructor 'public agence(String numero, String adresse, client[] lesClients, compte[] lesComptes)' is also shown, initializing the attributes and incrementing 'nbagences'.

```
11 import java.util.Arrays;
12
13 public class agence {
14     private static int nbagences = 1;
15     private String numero;
16     private String adresse;
17     private client[] lesClients;
18     private compte[] lesComptes;
19     public int nbcomptes = 0;
20     public int nbclients = 0;
21
22     public agence(String numero, String adresse, client[] lesClients, compte[] lesComptes) {
23         this.numero = getClasse() + ":" + nbagences++;
24         this.adresse = adresse;
25         this.lesClients = lesClients; // Utilisation des tableaux fournis en paramètres
26         this.lesComptes = lesComptes; // Utilisation des tableaux fournis en paramètres
27     }
28 }
```

### Et les méthodes suivantes :

public String getClasse(): Une méthode qui renvoie le type de classe, ici "Agence".

public String getName(): Une méthode qui renvoie le nom de l'agence en combinant le type de classe et le numéro d'agence.

public client getClients(int numclient): Une méthode pour obtenir un client en fonction de son numéro.

public compte getCompte(int numcompte): Une méthode pour obtenir un compte en fonction de son numéro.

public void setNumero(String numero): Une méthode pour définir le numéro de l'agence.

public void addClient(client client): Une méthode pour ajouter un client à l'agence si l'espace est disponible dans le tableau.

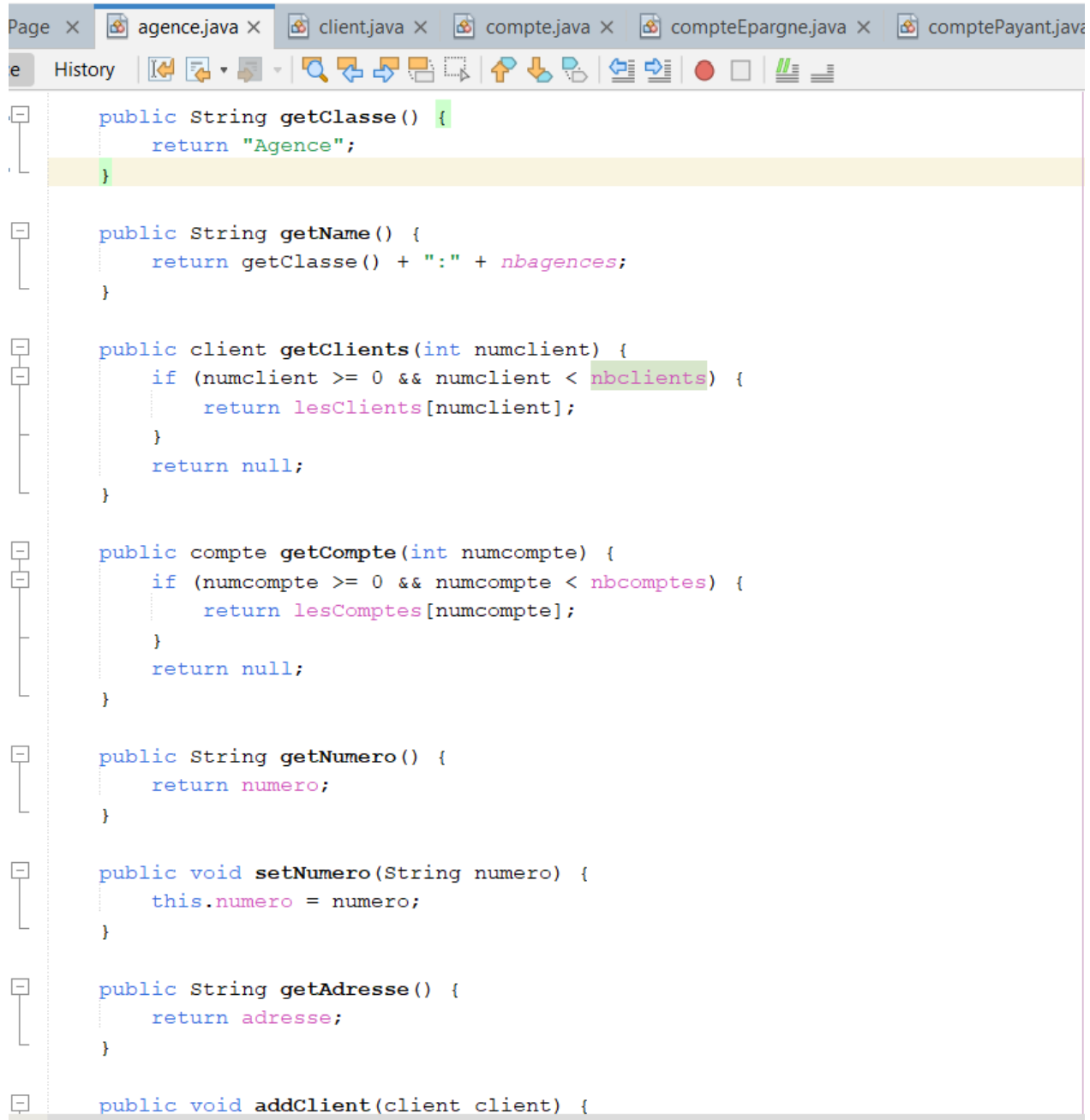
public void addCompte(compte compte): Une méthode pour ajouter un compte à l'agence si l'espace est disponible dans le tableau.

public int getNbrClients(): Une méthode pour obtenir le nombre de clients associés à l'agence.

public int getNbrComptes(): Une méthode pour obtenir le nombre de comptes associés à l'agence.

public void afficherComptesPayants(): Une méthode qui affiche les comptes payants de l'agence en parcourant le tableau de comptes et vérifiant s'ils sont instances de comptePayant.

@Override public String toString(): Une méthode qui génère une représentation sous forme de chaîne de caractères de l'objet agence. Cette méthode utilise la classe Arrays pour afficher les tableaux de clients et de comptes.



```
Page x agence.java x client.java x compte.java x compteEpargne.java x comptePayant.java
e History

public String getClasse() {
    return "Agence";
}

public String getName() {
    return getClasse() + ":" + nbagences;
}

public client getClients(int numclient) {
    if (numclient >= 0 && numclient < nbclients) {
        return lesClients[numclient];
    }
    return null;
}

public compte getCompte(int numcompte) {
    if (numcompte >= 0 && numcompte < nbcomptes) {
        return lesComptes[numcompte];
    }
    return null;
}

public String getNumero() {
    return numero;
}

public void setNumero(String numero) {
    this.numero = numero;
}

public String getAdresse() {
    return adresse;
}

public void addClient(client client) {
```

```

Start Page x agence.java x client.java x compte.java x compteEpargne.java x comptePayant.java x ApplicationBancaire.java x
Source History
63 public void addClient(client client) {
64     if (nbclients < lesClients.length) {
65         lesClients[nbclients] = client;
66         nbclients++;
67     }
68 }
69
70 public void addCompte(compte compte) {
71     if (nbcomptes < lesComptes.length) {
72         lesComptes[nbcomptes] = compte;
73         nbcomptes++;
74     }
75 }
76
77 public int getNbrClients() {
78     return nbclients;
79 }
80
81 public int getNbrComptes() {
82     return nbcomptes;
83 }
84
85 @Override
86 public String toString() {
87     return "Agence [numero=" + numero + ", adresse=" + adresse + ", lesClients=" + Arrays.toString(a: lesClients)
88         + ", lesComptes=" + Arrays.toString(a: lesComptes) + ", nbcomptes=" + nbcomptes + ", nbclients="
89         + nbclients + "]\n";
90 }
91
92 public void afficherComptesPayants() {
93     System.out.println(x: "Liste des comptes payants de l'agence:");
94     for (compte compte : lesComptes) {
95         if (compte instanceof comptePayant) {
96             System.out.println(x: compte);
97         }
98     }
99 }

```

**La classe client contient cinq attributs :**

code: Le code du client.

nom: Le nom du client.

adresse: L'adresse du client.

agence monAgence: L'agence à laquelle le client est associé.

compte[] mesComptes: Un tableau de comptes associés au client.

```

10 import java.util.Arrays;
11
12 public class client {
13     private String code;
14     private String nom;
15     private String adresse;
16     private agence monAgence;
17     private compte[] mesComptes;
18
19     public client(String code, String nom, String adresse, agence monAgence, compte[] mesComptes) {
20         this.code = code;
21         this.nom = nom;
22         this.adresse = adresse;
23         this.monAgence = monAgence;
24         this.mesComptes = mesComptes;
25     }
26 }

```

## Et les méthodes suivantes :

`public String getCode():` Une méthode pour obtenir le code du client.

`public void setCode(String code):` Une méthode pour définir le code du client.

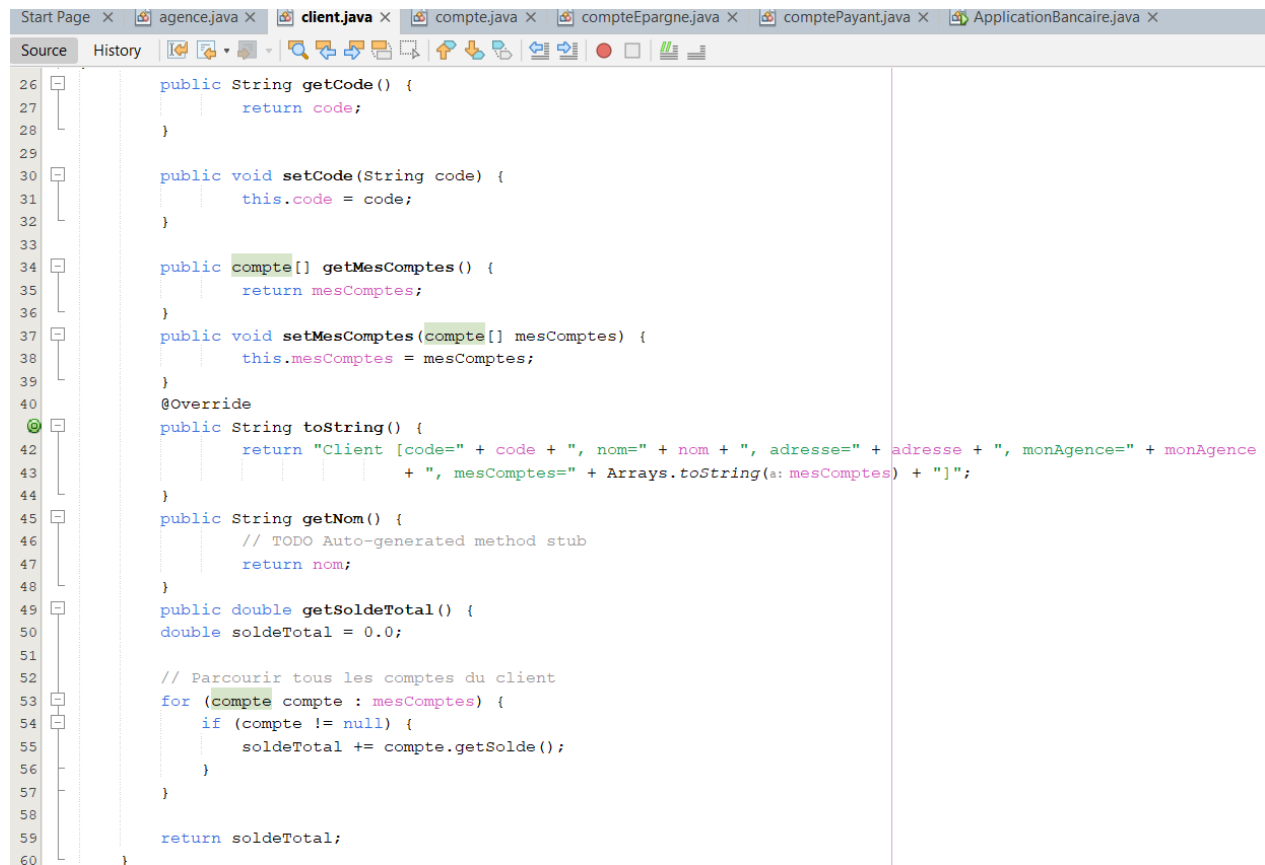
`public compte[] getMesComptes():` Une méthode pour obtenir le tableau de comptes associés au client.

`public void setMesComptes(compte[] mesComptes):` Une méthode pour définir le tableau de comptes associés au client.

`public String toString():` Une méthode qui génère une représentation sous forme de chaîne de caractères de l'objet client. Elle utilise la classe Arrays pour afficher le tableau de comptes.

`public String getNom():` Une méthode pour obtenir le nom du client.

`public double getSoldeTotal():` Une méthode pour obtenir le solde total de tous les comptes du client. Elle parcourt le tableau de comptes et accumule les soldes.



```
Start Page X agence.java X client.java X compte.java X compteEpargne.java X comptePayant.java X ApplicationBancaire.java X
Source History
26 public String getCode() {
27     return code;
28 }
29
30 public void setCode(String code) {
31     this.code = code;
32 }
33
34 public compte[] getMesComptes() {
35     return mesComptes;
36 }
37 public void setMesComptes(compte[] mesComptes) {
38     this.mesComptes = mesComptes;
39 }
40 @Override
41 public String toString() {
42     return "Client [code=" + code + ", nom=" + nom + ", adresse=" + adresse + ", monAgence=" + monAgence
43         + ", mesComptes=" + Arrays.toString(mesComptes) + "]";
44 }
45 public String getNom() {
46     // TODO Auto-generated method stub
47     return nom;
48 }
49 public double getSoldeTotal() {
50     double soldeTotal = 0.0;
51
52     // Parcourir tous les comptes du client
53     for (compte compte : mesComptes) {
54         if (compte != null) {
55             soldeTotal += compte.getSolde();
56         }
57     }
58
59     return soldeTotal;
60 }
```

### La classe compte contient quatre attributs :

code: Le code du compte.

solde: Le solde du compte.

agence agence: L'agence associée au compte.

client proprietaire: Le propriétaire (client) du compte.

```
12 public class compte {  
13     private String code;  
14     protected double solde;  
15     protected agence agence;  
16     protected client proprietaire;  
17  
18     public compte(String code ) {  
19         this.code=code;  
20     }  
21 }
```

### Et les méthodes suivantes :

public String getCode(): Une méthode pour obtenir le code du compte.

public void setCode(String code): Une méthode pour définir le code du compte.

public void déposer(double somme): Une méthode pour déposer une somme sur le compte. Elle ajoute la somme au solde actuel.

public void retirer(double somme): Une méthode pour retirer une somme du compte. Elle soustrait la somme du solde actuel.

public double getSolde(): Une méthode pour obtenir le solde du compte.

public void setSolde(double solde): Une méthode pour définir le solde du compte.

@Override public String toString(): Une méthode qui génère une représentation sous forme de chaîne de caractères de l'objet compte. Elle inclut le code, le solde, l'agence et le propriétaire.

```

Start Page x agence.java x client.java x compte.java x compteEpargne.java x comptePayant.java x ApplicationBancaire.java x
Source History
20 public String getCode() {
21     return code;
22 }
23
24 public void setCode(String code) {
25     this.code = code;
26 }
27
28 public void deposter(double somme) {
29     solde += somme;
30 }
31
32 public void retirer(double somme) {
33     solde -= somme; // Modification ici : retire la somme du solde
34 }
35
36 public double getSolde() {
37     return solde;
38 }
39
40 public void setSolde(double solde) {
41     this.solde = solde;
42 }
43
44 @Override
45 public String toString() {
46     return "Compte [code=" + code + ", solde=" + solde + ", lagence=" + lagence + ", proprietaire=" + proprietaire
47         + "]\n";
48 }

```

Cette classe `compteEpargne`, hérite de la classe `compte` et introduit des fonctionnalités spécifiques aux comptes épargne.

### Les Attributs de la classe `compteEpargne` :

`public double taux = 0.06;` : Un attribut qui représente le taux d'intérêt associé au compte épargne, initialisé à 6%.

```

10 public class compteEpargne extends compte {
11     public double taux = 0.06; // 6%
12
13     public compteEpargne(String code, client proprietaire, agence agence, double solde) {
14         super(code);
15         this.proprietaire = proprietaire;
16         this.lagence = agence;
17         this.solde = solde;
18     }
19
20     public void calculInteret() {
21         double apresInteret = getSolde() * taux; // Utilisation de getSolde() au lieu de solde
22         super.deposer(somme: apresInteret);
23     }
24
25     @Override
26     public String toString() {
27         return "compteEpargne{" +
28             " code='" + getCode() + '\'' +
29             ", solde=" + getSolde() +
30             // Ajoutez d'autres informations spécifiques au CompteEpargne ici
31             ", proprietaire=" + (proprietaire != null ? proprietaire.getNom() : "Aucun") +
32             '\n';
33     }
34 }

```

## Méthodes :

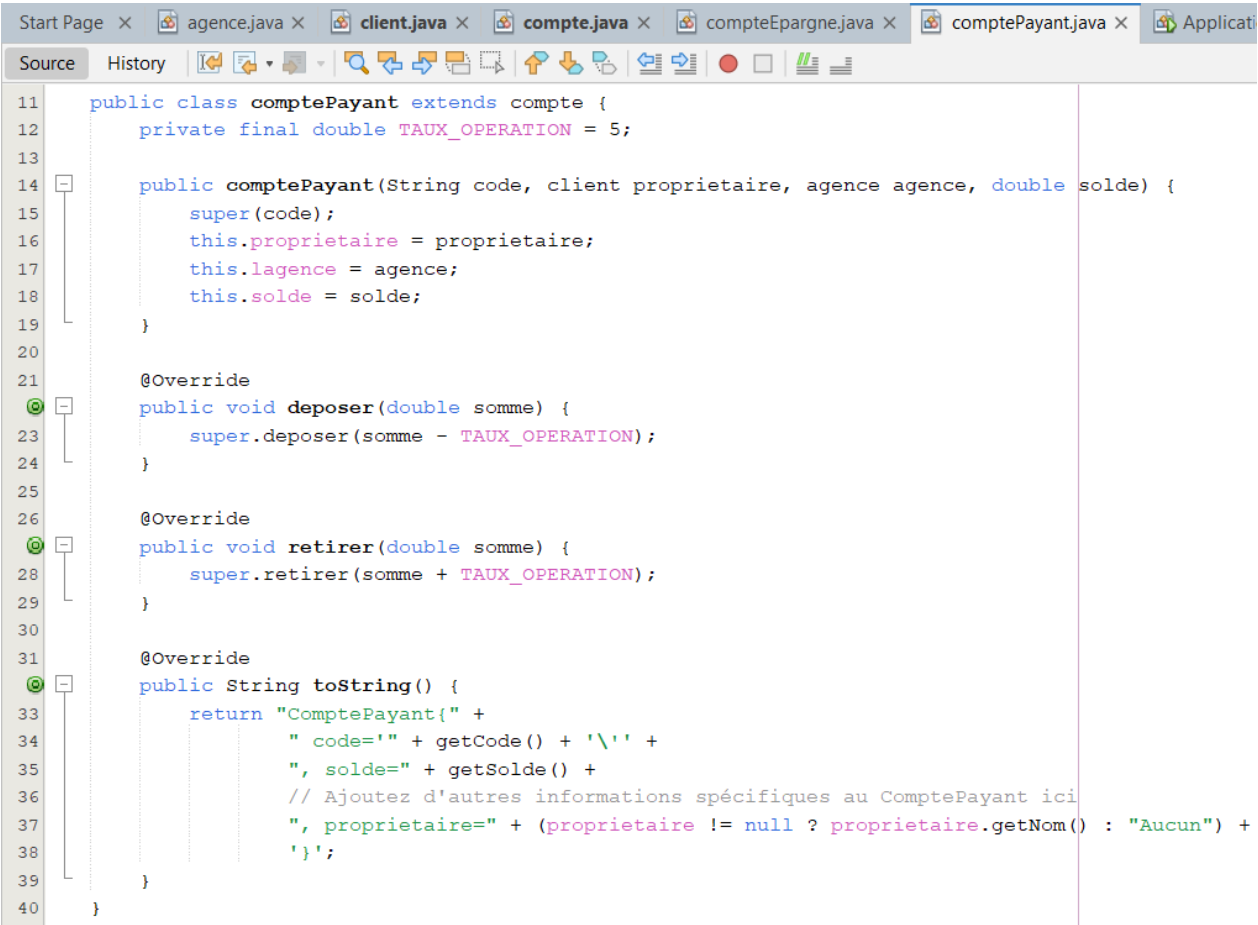
`public void calculInteret()`: Une méthode spécifique aux comptes épargne qui calcule les intérêts en multipliant le solde actuel par le taux d'intérêt. Les intérêts sont ensuite déposés sur le compte en utilisant la méthode `deposer` de la classe mère.

`@Override public String toString()`: Une méthode qui génère une représentation sous forme de chaîne de caractères de l'objet `compteEpargne`. Elle inclut le code, le solde, le propriétaire et d'autres informations spécifiques.

## Utilisation de méthodes héritées :

`super.deposer(apresInteret)`: Appel à la méthode `deposer` de la classe mère (`compte`) pour déposer les intérêts calculés sur le compte.

## Pour la classe `comptePayant` :



```
11 public class comptePayant extends compte {
12     private final double TAUX_OPERATION = 5;
13
14     public comptePayant(String code, client proprietaire, agence agence, double solde) {
15         super(code);
16         this.proprietaire = proprietaire;
17         this.lagence = agence;
18         this.solde = solde;
19     }
20
21     @Override
22     public void deposer(double somme) {
23         super.deposer(somme - TAUX_OPERATION);
24     }
25
26     @Override
27     public void retirer(double somme) {
28         super.retirer(somme + TAUX_OPERATION);
29     }
30
31     @Override
32     public String toString() {
33         return "ComptePayant{" +
34             " code='" + getCode() + '\'' +
35             ", solde=" + getSolde() +
36             // Ajoutez d'autres informations spécifiques au ComptePayant ici
37             ", proprietaire=" + (proprietaire != null ? proprietaire.getNom() : "Aucun") +
38             "'}";
39     }
40 }
```

## Attributs de la classe `comptePayant` :

`private final double TAUX_OPERATION = 5;`: Un attribut final qui représente le taux d'opération associé au compte payant, fixé à 5%.



## Méthodes supplémentaires :

@Override public void déposer(double somme): Une méthode qui surcharge la méthode déposer de la classe mère. Elle dépose une somme sur le compte en déduisant le taux d'opération du montant déposé.

@Override public void retirer(double somme): Une méthode qui surcharge la méthode retirer de la classe mère. Elle retire une somme du compte en ajoutant le taux d'opération au montant retiré.

@Override public String toString(): Une méthode qui génère une représentation sous forme de chaîne de caractères de l'objet comptePayant. Elle inclut le code, le solde, le propriétaire et d'autres informations spécifiques.

## Utilisation de méthodes héritées :

super.dépenser(somme - TAUX\_OPERATION): Appel à la méthode dépenser de la classe mère (compte) en ajustant le montant déposé en fonction du taux d'opération.

super.retirer(somme + TAUX\_OPERATION): Appel à la méthode retirer de la classe mère (compte) en ajustant le montant retiré en fonction du taux d'opération.

## La class ApplicationBancaire :



```
public class ApplicationBancaire {  
  
    public static void main(String[] args) {  
        // Créer des objets Client  
        compte[] lesComptes = new compte[50];  
        client[] clients = {  
            new client(code: "b12", nom: "aya", adresse: "hana2", monAgence:null, mesComptes: null),  
            new client(code: "b13", nom: "chaymae", adresse: "hana1", monAgence:null, mesComptes: null),  
            new client(code: "b14", nom: "yousra", adresse: "hana1", monAgence:null, mesComptes: null),  
            new client(code: "b15", nom: "amine", adresse: "hana1", monAgence:null, mesComptes: null)  
        };  
  
        agence agence1 = new agence(numero: "B11", adresse: "123 Rue de la Banque", lesClients: clients, lesComptes);  
    }  
}
```

## Création d'objets Client et Agence :

Un tableau clients est créé, contenant plusieurs objets client.

Un tableau lesComptes est créé pour stocker les comptes (même s'il n'est actuellement pas utilisé dans le code).

Une instance de la classe agence (agence1) est créée avec un numéro, une adresse, la liste des clients et des comptes.

## Création des Comptes :

```

24 // Créer les comptes
25 compteEpargne compteEpargne1 = new compteEpargne(code: "bk1", clients[1], agence:agence1, solde: 1000);
26 comptePayant comptePayant1 = new comptePayant(code: "bk2", clients[2], agence:agence1, solde: 2500);
27 comptePayant comptePayant2a = new comptePayant(code: "belfaik3", clients[3], agence:agence1, solde: 0);
28 comptePayant comptePayant2b = new comptePayant(code: "belfaik4", clients[3], agence:agence1, solde: 3000);
29 compteEpargne compteEpargne3 = new compteEpargne(code: "belfaik5", clients[0], agence:agence1, solde: 2300);
30 comptePayant comptePayant3 = new comptePayant(code: "belfaik6", clients[0], agence:agence1, solde: 0);
31
32 // Associer les comptes aux clients
33 clients[0].setMesComptes(new compte[]{compteEpargne3, comptePayant3});
34 clients[1].setMesComptes(new compte[]{compteEpargne1});
35 clients[2].setMesComptes(new compte[]{comptePayant1});
36 clients[3].setMesComptes(new compte[]{comptePayant2a, comptePayant2b});
37

```

Plusieurs types de comptes sont créés, notamment des comptes épargne et des comptes payants, avec des propriétaires associés, une agence et un solde initial.

### Association des Comptes aux Clients :

Les comptes créés sont associés aux clients correspondants en utilisant la méthode setMesComptes.

### Affichage des Informations :

Une boucle parcourt les clients et leurs comptes, affichant les informations de chaque client et de ses comptes associés.

Une autre boucle affiche la liste des comptes d'épargne de l'agence et utilise la méthode afficherComptesPayants de l'agence pour afficher les comptes payants.

```

37
38 // Afficher les informations
39 System.out.println(x: "Liste des clients et de leurs comptes :");
40 for (client client : clients) {
41     System.out.println("Client : " + client);
42     for (compte compte : client.getMesComptes()) {
43         System.out.println(" - Compte : " + compte);
44     }
45     System.out.println(); // Ligne vide entre chaque client
46 }
47
48 System.out.println(x: "Liste des comptes d'épargne de l'agence:");
49 for (compte compte : lesComptes) {
50     if (compte instanceof compteEpargne && compte != null) {
51         System.out.println(x: compte);
52     }
53 }
54
55 agence1.afficherComptesPayants();

```

### Calcul du Solde Total pour Chaque Client :

Une dernière boucle parcourt les clients pour afficher le solde total de chaque client, calculé en utilisant la méthode getSoldeTotal de la classe client. Cette méthode somme les soldes de tous les comptes associés à un client.

```

56
57 // Afficher le solde total de chaque client
58 for (client client : clients) {
59     System.out.println("Solde total pour le client " + client.getNom() + ": " + client.getSoldeTotal());
60 }
61
62
63

```

**Voilà le résultat de run :**

Liste des clients et de leurs comptes :

Client : Client [code=b12, nom=aya, adresse=hana2, monAgence=null, mesComptes=[CompteEpargne{ solde=2300.0, proprietaire=aya}, ComptePayant{ solde=0.0, proprietaire=aya}]

Client : Client [code=b13, nom=belfaik, adresse=sraghna, monAgence=null, mesComptes=[CompteEpargne{ solde=6000.0, proprietaire=belfaik}, ComptePayant{ solde=0.0, proprietaire=belfaik}]

Client : Client [code=b14, nom=yousra, adresse=el kelaa, monAgence=null, mesComptes=[ComptePayant{ solde=2500.0, proprietaire=yousra}, ComptePayant{ solde=0.0, proprietaire=yousra}]

Client : Client [code=b15, nom=alae, adresse=nakhla, monAgence=null, mesComptes=[ComptePayant{ solde=0.0, proprietaire=alae}, ComptePayant{ solde=3000.0, proprietaire=alae}]

Liste des comptes d'épargne de l'agence:

Liste des comptes payants de l'agence:

Solde total pour le client aya: 2300.0

Solde total pour le client belfaik: 6000.0

Solde total pour le client yousra: 2500.0

Solde total pour le client alae: 3000.0