TP N°2

Classes, attributs, méthodes, constructeurs, …

- Ouvrez **Eclipse IDE** et créez un projet Java, que vous nommerez **TP2**
- Créez les deux packages suivants : **ma.emsi.projets.magasin** et **ma.emsi.projets.banque**

Exercice 1 :

a) Dans le package **ma.emsi.projets.magasin**, créez une classe **Article** pour représenter les articles vendus dans un magasin.

   - Un article est caractérisé par une référence (long), une description (String), un prix hors taxe (double) et le nombre d'unités disponibles en stock (int).

b) Complétez la classe par :

   - Un constructeur avec paramètres.
   - Une méthode **`public void approvisionner(int nombreUnites)`** pour augmenter la quantité disponible de l'article.
   - Une méthode **`public boolean vendre(int nombreUnites)`** qui enregistre la vente d'un certain nombre d'unités de l'article.
     - **Remarque** : Si **nombreUnites** est supérieur à la quantité disponible alors le stock n'est pas modifié et la méthode renvoie **false** ; autrement elle renvoie **true**.
   - Une méthode **`public double prixTTC()`** qui calcule le prix TTC d'un article. Le magasin applique la même taxe de 10% sur le prix des articles.
   - Une méthode **`public double prixVenteTTC(int nombreUnites)`** qui calcule le prix de vente TTC d'un nombre d'unités d'un article.
   - Une redéfinition de la méthode **`toString()`** qui retourne une chaîne de caractères exprimant la référence, la description et le prix de l'article. (**Lire la remarque ci-dessous**)
   - Une redéfinition de la méthode **`equals`** qui vérifie si deux articles ont la même référence **(Lire la remarque ci-dessous**)

Remarque :

Les deux méthodes **toString()** et **equals** sont héritées de la super-classe **Object.** Pour générer automatiquement ces deux méthodes suivez les étapes suivantes :

➔ La méthode **toString() :**

1. Dans l'emplacement où vous voulez insérer la méthode toString() :

   Click droit -> Source -> Generate ToString()

2. Sélectionnez les attributs à afficher puis appuyez sur **Generate**.

➔ La méthode **equals :**

1. ***Click droit -> Source -> Generate HachCode() and equals()***
2. Sélectionnez les attributs à utiliser pour la comparaison. Dans notre cas, nous utilisons l'attribut **reference**.
3. Appuyez sur **Generate .** Vous obtiendrez le code des deux méthodes **equals** et **hachCode.**

c) Testez cette classe en ajoutant la méthode **main.** Pour cela :

- Tapez **main** (en dehors des méthodes) et appuyez sur **Ctrl + Espace**
- Sélectionnez la suggestion ***main - main method*** et appuyez sur **Entrée**.
- La méthode main sera insérée automatiquement.

▪ Dans la méthode ***main***, créez un tableau comportant 10 articles (au moins) et essayez toutes les méthodes de la classe **Article**.

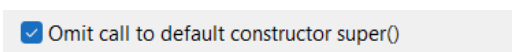
Exercice 2 :

A. Une banque souhaite gérer les comptes bancaires de ses clients.

a) Dans le package **ma.emsi.projets.banque,** créez la classe **Personne** pour représenter des personnes physiques. Cette classe contient :

➔ Deux attributs privés => nom (String) et prenom (String).

➔ Un constructeur avec paramètres. Pour générer ce constructeur dans Eclipse :

- ✓ ***Click droit -> Source -> Generate Constructor using Fields***
- ✓ ***Sélectionnez les deux attributs => nom et prenom***
- ✓ ***Cochez l'option*** 

- ✓ ***Appuyez sur Generate.***

➔ Deux **getters** qui retournent respectivement le nom et le prénom. Pour générer les deux getters seulement (sans setters) :

- ✓ ***Click droit -> Source -> Generate Getters and Setters***
- ✓ ***Appuyez sur "Deselect All" si tout est sélectionné.***
- ✓ ***Appuyez sur "Select Getters ", puis sur Generate.***

b) Dans le package **ma.emsi.projets.banque ,** créez une classe **CompteBancaire** pour représenter les comptes bancaires de la banque.

- Un compte bancaire est caractérisé par un code (**String**), un titulaire (**Personne**) et un solde (**BigDecimal**).

➔ **BigDecimal** qui est une classe du package **java.math,** est le type recommandé pour représenter des montants monétaires. Il permet :

- ✓ D'éviter les erreurs d'arrondi liées au type double.
- ✓ D'utiliser des opérations monétaires prédéfinies (add, subtract, multiply, etc.)
- ✓ De formater proprement l'affichage des montants.

c) Complétez la classe **CompteBancaire** par :

- Un constructeur avec paramètres qui initialise un compte en précisant son code et son titulaire et dont le solde initial est égal à 0.
  - **Pour initialiser un BigDecimal à 0, utilisez la constante statique `BigDecimal.ZERO`**
- Un Constructeur avec paramètres qui initialise un compte en précisant son code, son titulaire et son solde initial qui doit être supérieur à 0. Sinon le solde doit être initialisé à 0.
  - **Réutilisez le premier constructeur en utilisant le mot-clé this.**
  - **Pour pour vérifier si un solde est supérieur à 0 utilisez la condition suivante :**

    if (solde.compareTo(BigDecimal.ZERO) > 0) …
- Une méthode ***public void deposer(BigDecimal montant)*** qui effectue l'opération de dépôt d'un montant d'argent dans le compte.
  - Utilisez la méthode **add** de la classe **BigDecimal**. **(Voir le tableau ci-dessous)**
- Une méthode ***public boolean retirer(BigDecimal montant)*** qui effectue l'opération de retrait d'un montant d'argent du compte. Cette opération n'est effectuée que si le montant à retirer est positif.
  - Utilisez la méthode **subtract** de la classe **BigDecimal**. **(Voir le tableau ci-dessous)**
- Un getter pour consulter le solde du compte.

Syntaxe	Description
`a.add(b)`	Additionne l'argument b à l'instance a et retourne le résultat sous forme de BigDecimal.
`a.subtract(b)`	Soustrait l'argument b de l'instance a et retourne le résultat sous forme de BigDecimal.
`a.multiply(b)`	Multiplie l'argument b par l'instance a et retourne le résultat sous forme de BigDecimal.
`a.divide(b)`	Divise a par b et retourne le résultat sous forme de BigDecimal.
`a.setScale(2)`	Retourne un BigDecimal arrondi à 2 chiffres après la virgule.
`a.pow(n)`	Elève le BigDecimal a à la puissance n qui est un entier. Le résultat est un BigDecimal.
`BigDecimal.ZERO`	Constante pour initialiser un BigDecimal à 0
`BigDecimal.One`	Constante pour initialiser un BigDecimal à 1

B. La banque souhaite autoriser pour certains clients un découvert. Par défaut, ce découvert autorisé est égal à 0.

d) Ajouter à la classe **CompteBancaire**, l'attribut **decouvert (BigDecimal)** et modifiez le constructeur adéquat en conséquence.

e) Ajouter une méthode **public void decouvertAutorise(BigDecimal montant)** qui permet de spécifier le montant du découvert autorisé. Le montant du découvert autorisé doit être supérieur à 0, sinon le découvert gardera sa valeur initiale qui est 0.

f) Modifiez la méthode de retrait. Un retrait est possible tant que le client ne dépasse pas le découvert autorisé c'est-à-dire que le montant à retirer est inférieur ou égal à solde + découvert. Le cas échéant, le retrait est refusé.

g) Ajoutez une méthode **public boolean estDebiteur()** qui vérifie si un compte est débiteur.

- **Remarque** : un compte est débiteur si le solde du compte est négatif.

C. La banque souhaite conserver le nombre de comptes bancaires débiteurs. Proposez une solution pour cela.

- ✓ **Idée : utilisez une variable statique.**

D. Tester la classe " CompteBancaire" dans une méthode main.