Corrigés des premiers exercices sur les classes

Exercice 2.1.1 *utilisation d'une classe*

Voici le texte d'une classe représentant de façon sommaire un compte bancaire et les opérations bancaires courantes.

```
class Compte{
    int solde = 0;
    void deposer(int montant){
        solde = solde + montant;
    }
    void retirer(int montant){
        solde = solde -montant;
    }
    void virerVers(int montant, Compte destination){
        this.retirer(montant);
        destination.deposer(montant);
    }
    void afficher(){
        Terminal.ecrireString("solde: "+ solde);
    }
}
```

Question 1

Comment fonctionne la méthode virement ? Combien de comptes fait-elle intervenir ?

Réponse : la méthode virement fait intervenir deux objets de type `Compte` : this, l'objet sur lequel la méthode est appélée et `destination`, le paramètre de la méthode. Le virement s'effectue de this vers le paramètre de la méthode. L'argent est retiré d'un compte et déposé sur l'autre.

Question 2

Créez deux comptes que vous affecterez à deux variables. Ecrivez le code correspondant aux opérations suivantes :

- dépot de 500 euros sur le premier compte.
- dépot de 1000 euros sur le second compte.
- retrait de 10 euros sur le second compte.

– virement de 75 euros du premier compte vers le second.
– affichage des soldes des deux comptes.

Vous mettrez le code java correspondant à cette question dans la méthode `main` d'une nouvelle classe appelée `TesteCompte`. Vous compilerez et testerez ce programme.

```
public class TesteCompte {
  public static void main(String[] args) {
    Compte martin, jean;

    martin = new Compte();
    jean = new Compte();

    // dépôt de 500 euros sur le premier compte.
    martin.deposer(500);

    // dépôt de 1000 euros sur le second compte.
    jean.deposer(1000);

    // retrait de 10 euros sur le second compte.
    jean.retirer(10);

     // virement de 75 euros du premier compte vers le second.
    martin.virerVers(75, jean);

    // affichage des soldes des deux comptes.
    Terminal.ecrireString("Compte de martin, ");
    martin.afficher();
    Terminal.sautDeLigne();

    Terminal.ecrireString("Compte de jean, ");
    jean.afficher();
    Terminal.sautDeLigne();

  }
}
```

Exécution du programme :

```
> run TesteCompte
Compte de martin, solde: 425
Compte de jean, solde: 1065
>
```

Question 3

Créez un tableau de dix comptes. Pour cela, notez bien qu'il faut d'abord créer le tableau puis créer successivement les dix comptes à mettre dans les dix cases de ce tableau.

Dans chaque case, faites un dépot de 200 euros plus une somme égale à 100 fois l'indice du compte dans le tableau.

Ensuite, vous ferez un virement de 20 euros de chaque compte vers chacun des comptes qui le suivent dans le tableau (par exemple, du compte d'indice 5, il faut faire des virements vers les comptes d'indice 6, 7, 8 et 9).

Enfin, vous afficherez les soldes de tous les comptes.
Ici encore, vous testerez et compilerez le code proposé.

```
public class TesteCompte {
  public static void main(String[] args) {
    Compte martin, jean;
    Compte[] table = new Compte[10];

    martin = new Compte();
    jean = new Compte();

    // dépôt de 500 euros sur le premier compte.
    martin.deposer(500);

    // dépôt de 1000 euros sur le second compte.
    jean.deposer(1000);

    // retrait de 10 euros sur le second compte.
    jean.retirer(10);

     // virement de 75 euros du premier compte vers le second.
    martin.virerVers(75, jean);

    // affichage des soldes des deux comptes.
    Terminal.ecrireString("Compte de martin, ");
    martin.afficher();
    Terminal.sautDeLigne();

    Terminal.ecrireString("Compte de jean, ");
    jean.afficher();
    Terminal.sautDeLigne();

    for (int i=0; i<table.length; i++){
      table[i] = new Compte();
      table[i].deposer(200 + i*100);
    }

    for (int i=0; i<table.length; i++){
      for (int j=i+1; j<table.length; j++){
        table[i].virerVers(20, table[j]);
      }
    }

    for (int i=0; i<table.length; i++){
      Terminal.ecrireString("Compte numero "+ i + ", ");
      table[i].afficher();
      Terminal.sautDeLigne();
    }
  }
}
```

Exécution du programme :

```
> run TesteCompte
```

```
Compte de martin, solde: 425
Compte de jean, solde: 1065
Compte numero 0, solde: 20
Compte numero 1, solde: 160
Compte numero 2, solde: 300
Compte numero 3, solde: 440
Compte numero 4, solde: 580
Compte numero 5, solde: 720
Compte numero 6, solde: 860
Compte numero 7, solde: 1000
Compte numero 8, solde: 1140
Compte numero 9, solde: 1280
>
```

Exercice 2.1.2 *constructeurs*

Cet exercice reprend la classe `Compte` de l'exercice précédent.

Question 1

Complétez la classe `Compte` avec une information supplémentaire : le nom du titulaire du compte (type `String`). Vous modifierez la méthode d'affichage pour qu'elle affiche cette information.

Question 2

Créez un constructeur pour la classe `Compte`. Ce constructeur doit prendre en paramètre le nom du titulaire du compte.

Donnez le code de création d'un compte qui appelle ce constructeur.

```
class Compte{
    int solde = 0;
    String titulaire;
    Compte(String n){
        titulaire = n;
    }
    void depot(int montant){
        solde = solde + montant;
    }
    void retrait(int montant){
        solde = solde -montant;
    }
    void virement(int montant, Compte autre){
        autre.retrait(montant);
        this.depot(montant);
    }
    void afficher(){
        Terminal.ecrireString("Compte␣de␣" + titulaire + ",␣solde:␣"+ solde);
    }
}
class Exo13_2{
```

```
    public static void main(String[] argv){
        Compte unCompte = new Compte("Jean␣Delacroix");
        unCompte.depot(700);
        unCompte.afficher();
        Terminal.sautDeLigne();
    }
}
```

Question 3

Faut-il prévoir des méthodes permettant de changer le nom du titulaire du compte ?

Réponse : ce n'est pas facile à dire. Est-ce qu'un compte peut changer de titulaire ? Est-ce qu'une personne peut changer de nom ? En tout état de cause, il peut être utile de prévoir un moyen de corriger une éventuelle faute de frappe à la saisie.

Exercice 2.1.3 *méthodes statiques ou non*

Parmi les méthodes de la classe suivante, lesquelles peuvent être statiques et lesquelles ne peuvent en aucun cas être statiques ?

```
class Exo13_3{
    int x, y;
    String nom;
    void afficher(){
        Terminal.ecrireString(nom + "␣" + x + "␣␣" + y);
    }
    void ajouter(Exo13_3 obj){
        x = x + obj.x;
        y = y + obj.y;
        nom = nom + obj.nom;
    }
    Exo13_3 nouveau(int n){
        Exo13_3 res = new Exo13_3();
        res.x = n;
        res.y = n*2;
        res.nom = "Auto_"+n;
        return res;
    }
    boolean plusGrand(Exo13_3 obj){
        if (obj.x == x){
            return y>obj.y;
        }else{
            return x>obj.x;
        }
    }
    boolean compare(Exo13_3 obj1, Exo13_3 obj2){
        if (obj1.x == obj2.x){
            return obj1.y>obj2.y;
        }else{
            return obj1.x>obj2.x;
        }
```

```
    }
}
```

Les méthodes statiques sont des méthodes qui existent indépendamment de tout objet. Elles ne peuvent pas utiliser les variables d'instances ni `this`. Ici, il y a trois variables d'instance, x, y et nom.

Les méthodes `afficher, ajouter, plusGrand` utilisent des variables d'instance et/ou this. Elle ne peuvent en aucun cas être statiques.

Les deux autres méthodes, `nouveau` et `compare` peuvent être déclarées `static`.

Exercice 2.1.4 *égalité d'objets*

```
class Compteur{
    int x;
    Compteur(int n){
        x=n;
    }
    Compteur incremente(){
        x++;
        return this;
    }
    int value(){
        return x;
    }
}
class Exo13_4{
    public static void main(String[] argv){
        Compteur c1, c2, c3;
        c1 = new Compteur(0);
        c1.incremente();
        c2 = new Compteur(1);
        c3 = c1;
        if (c1 == c3){
            Terminal.ecrireStringln("c1 et c3 sont egaux");
        }else{
            Terminal.ecrireStringln("c1 et c3 ne sont pas egaux");
        }
        if (c1.value() == c2.value()){
            Terminal.ecrireStringln("c1 et c2 ont meme valeur");
        }else{
            Terminal.ecrireStringln("c1 et c2 n'ont pas la meme valeur");
        }
        if (c1 == c2){
            Terminal.ecrireStringln("c1 et c2 sont egaux");
        }else{
            Terminal.ecrireStringln("c1 et c2 ne sont pas egaux");
        }
        if (c1.value() == c1.incremente().value()){
            Terminal.ecrireStringln("c1 et c1 incremente ont meme valeur");
        }else{
            Terminal.ecrireStringln("c1 et c1 incremente n'ont pas la meme valeur");
        }
```

```
        if (c1 == c1.incremente()){
            Terminal.ecrireStringln("c1 et c1 incremente sont egaux");
        }else{
            Terminal.ecrireStringln("c1 et c1 incremente ne sont pas egaux");
        }
    }
}
```

Essayez de prédire le résultat de l'exécution de ce programme. Testez le programme. Que peut-on en déduire sur la notion d'égalité d'objets en java ?

Dans ce corrigé, nous allons constater le résultat affiché et essayer de comprendre ce que cela signifie.

```
> java Exo13_4
c1 et c3 sont égaux
c1 et c2 ont même valeur
c1 et c2 ne sont pas égaux
c1 et c1 incremente n'ont pas la même valeur
c1 et c1 incremente sont égaux
```

On voit que le comparateur == teste l'identité plutôt que l'égalité en terme de valeur. Par exemple, `c1` et `c3` sont considérés comme égaux parce qu'il s'agit d'un même objet, créé par un seul et unique `new`. En revanche, `c1` et `c2` qui sont créés par deux `new` différents ne peuvent en aucun cas être reconnus égaux, même si ils ont le même état (c'est le cas ici, puisque l'état consiste dans la valeur de l'unique variable d'instance `x`.

A contrario, si on compare deux fois le même objet en ayant changé sa valeur entre temps, il sera bien reconnu comme égal. Le test porte donc sur l'identité de l'objet et non pas sur les valeurs de ses variables d'instance.

Il est bien évident que ce test ne correspond pas toujours à l'égalité dont on a besoin, c'est pourquoi dans beaucoup de classes, il existe un autre type de test utilisant la méthode `equals`. Nous avons déjà vu cela pour les `String`. Lorsqu'on écrit une classe, il est parfois utile et nécessaire de définir cette méthode `equals`.

Exercice 2.1.5 *conception*

Cet exercice a pour but de réfléchir sur la conception d'un programme, sa structuration en classes. Il ne s'agit pas pour le moment de réaliser ce programme, mais juste de concevoir son architecture.

On fait des cocktails avec différents liquides (alcools, sodas, jus de fruits). On a un bar avec des bouteilles qui peuvent être pleines ou à moitié vides. On a des shakers qui ont une contenance donnée. Il y a des recettes de cocktails qui indiquent seulement les proportions. Ces recettes peuvent s'appliquer à des quantités plus ou moins grandes selon les besoins du moment.

Les cocktails se font en déversant une partie du contenu des bouteilles dans des shakers. Après, il faut secouer. Les shakers sont ensuite vidés (dans les verres, mais on ne tiendra pas compte des verres dans cette application). Il faut les laver après usage.

Question : quelles classes faut-il créer ? Quelles informations faut-il dans chaque classe ? Quelles méthodes faut-il écrire, et dans quelle classe les mettre ?

Voyons d'abord une première approche :

classe	variables	méthodes
Bouteille	nom du produit quantité restante	verser une certaine quantité
Bar	liste de bouteilles	ajouter une bouteille jeter une bouteille rechercher une bouteille d'un produit
Shaker	contenance liste des ingrédients avec nom et quantité	ajouter un ingrédient vider le shaker laver le shaker
Recette	liste des ingredients avec nom et proportion	vérifier si un shaker respecte la recette

Des questions se posent quant aux types des variables. Par exemple, pour les quantités, allons-nous choisir des entiers ou des nombres à virgule ? Pour les ingrédients, allons-nous créer des enregistrements, donc une nouvelle classe, avec un nom d'ingrédient et une quantité ? Ce sont à ces questions qu'il faut répondre. Il n'y a pas forcément une solution unique, mais un ensemble de solutions possibles, parmi lesquelles il faut choisir.

Reprenons maintenant classe par classe pour préciser les types.

Classe `Ingredient` :
- `nomProduit` : type `String`
- `quantite` : type `int`

Classe `Bouteille` :
- `contenu` : type `Ingredient`
- `verser` :
  - paramètre : quantité à verser, `type int`
  - valeur retournée : aucune, type `void`

Classe `Bar`
- `reserve` : type `Bouteille[]`
- `ajouter` :
  - paramètre : une bouteille, type `Bouteille`
  - valeur retournée : aucune, type `void`
- jeter :
  - paramètre : une bouteille, type `Bouteille`
  - valeur retournée : aucune, type `void`
- chercher :
  - paramètre : un nom d'ingrédient, type `String`
  - valeur retournée : une bouteille, type `Bouteille`

Classe `Shaker`
- `contenance` : type `int`
- `contenu` : type `Ingredient[]`
- `propre` : type `boolean`
- `ajouter` :
  - paramètre : `Ingredient`
  - valeur retournée : aucune, type `void`
- `vider` :
  - paramètre : aucun
  - valeur retournée : aucune, type `void`
- `laver` :

  - paramètre : aucun
  - valeur retournée : aucune, type `void`

Classe `Recette`

- `ingredients` : type `Ingredient[]`
- `verifie` :
  - paramètre : un shaker, type `Shaker`
  - résultat : type `boolean`