Corrigés des exercices sur les classes (suite)

Exercice 2.2.1 *comptes (suite)*

On veut améliorer et compléter la classe `Compte`.

1. modifier la méthode `retrait` pour empêcher le retrait quand le compte n'est pas suffisament approvisionné.
2. modifier la classe pour ajouter à chaque compte un numéro de compte.

```
class Compte{
    int solde = 0;
    String titulaire;
    int numero;
    Compte(String nom, int num){
        titulaire = nom;
        numero = num;
    }
    Compte depot(int montant){
        solde = solde + montant;
        return this;
    }
    boolean retrait(int montant){
        if (solde-montant > 0){
            solde = solde -montant;
            return true;
        }else{
            return false;
        }
    }
    Compte virement(int montant, Compte autre){
        if (autre.retrait(montant)){
          this.depot(montant);
        }
        return this;
    }
    void afficher(){
        Terminal.ecrireString("Compte␣numero␣" + numero + "␣de␣" + titulaire);
        Terminal.ecrireString(",␣solde:␣" + solde);
    }
}
```

```
class Exo_14_1{
    public static void main(String[] argv){
        Compte unCompte = new Compte("Jean␣Delacroix",10052);
        unCompte.depot(700);
        unCompte.afficher();
        Terminal.sautDeLigne();
    }
}
```

Exercice 2.2.2 *bibliothèque*

```
class Livre2{
    String titre, auteur, editeur;
    int annee, nbPages;
    void ecrireLivre(){
        Terminal.ecrireStringln("Titre:␣" + this.titre);
        Terminal.ecrireStringln("Auteur:␣" + this.auteur);
        Terminal.ecrireStringln("Editeur:␣" + this.editeur);
        Terminal.ecrireStringln("Annee:␣" + this.annee);
        Terminal.ecrireStringln("Nombre␣de␣pages:␣" + this.nbPages);
    }
    static Livre2 lireLivre(){
        Livre2 res = new Livre2();
        Terminal.ecrireString("Entrez␣le␣nom␣de␣l'auteur:␣");
        res.auteur = Terminal.lireString();
        Terminal.ecrireString("Entrez␣le␣titre:␣");
        res.titre = Terminal.lireString();
        Terminal.ecrireString("Entrez␣l'editeur:␣");
        res.editeur = Terminal.lireString();
        Terminal.ecrireString("Entrez␣le␣nombre␣de␣page:␣");
        res.nbPages = Terminal.lireInt();
        Terminal.ecrireString("Entrez␣l'annee␣de␣publication:␣");
        res.annee = Terminal.lireInt();
        return res;
    }
}
class Bibliotheque2{
    Livre2[] tabLiv = new Livre2[25];
    int nbLivre = 0;
    void ecrireBibliotheque(){
        for (int i = 0; i<this.nbLivre; i++){
            Terminal.ecrireStringln("************************");
            (this.tabLiv[i]).ecrireLivre();
        }
    }
    void ajouter(Livre2 liv){
        this.tabLiv[this.nbLivre] = liv;
        this.nbLivre++;
    }
}
class Exo_14_2{
```

```
    public static void main(String[] args){
        Bibliotheque2 laBib = new Bibliotheque2();
        Livre2 chartreuse = new Livre2();
        chartreuse.titre = "La␣chartreuse␣de␣Parme";
        chartreuse.auteur = "Stendhal";
        chartreuse.editeur = "Gallimard";
        chartreuse.annee = 1987;
        chartreuse.nbPages = 683;
        laBib.ajouter(chartreuse);
        laBib.ajouter(Livre2.lireLivre());
        laBib.ecrireBibliotheque();
    }
}
```

La méthode `lireLivre` a vocation à être statique parce qu'il s'agit de créer un nouveau livre. Il n'y a pas d'accès aux informations d'un livre particulier. Les autres méthodes sont appelées sur un objet particulier.

La tableau `tabLiv` de la classe `Bibliotheque2` est un tableau qui contient des objets de type livre. On peut donc appeler les méthodes de la classe `Livre2` sur les cases de ce tableau, comme dans l'appel : `(this.tabLiv[i]).ecrireLivre();`.

Exercice 2.2.3 *tableau trié*

Nous allons voir dans cet exercice comment on peut avoir un tableau trié sans jamais faire une opération de tri.

On va utiliser des produits comportant un nom, une référence et un prix hors taxe. La référence sera un nombre entier. On veut écrire une classe `Stock` qui contient une liste de produits représentée par un tableau de produits trié par ordre croissant de la référence.

Un stock sera initialement vide, puis rempli petit à petit au moyen d'une opération d'ajout. Ce sera à cette opération d'ajout d'insérer le nouvel objet à la bonne place du tableau, compte tenu de l'ordre des références.

Outre cette opération d'ajout, vous ferez une opération qui affiche le contenu d'un stock et une opération qui recherche le prix d'une référence donnée dans un stock. Vous structurerez au mieux les deux classes `Produit` et `Stock` pour que chacune d'elle comporte les méthodes concernant leurs données respectives. Vous mettrez la méthode main dans une troisième classe.

```
class Produit2{
    String nom;
    int reference;
    double prixHT;
    Produit2(String n, int ref, double prix){
        nom=n;
        reference=ref;
        prixHT=prix;
    }
    void affiche(){
        Terminal.ecrireString(nom + "␣reference:␣" + reference + "␣prix:␣");
        Terminal.ecrireDoubleln(prixHT);
    }
    boolean precede(Produit2 p){
```

```
        return reference < p.reference;
    }
}
class Stock{
    Produit2[] tab = new Produit2[15];
    int nbProduits = 0;
    void affiche(){
        for (int i=0; i<nbProduits; i++){
            tab[i].affiche();
        }
        Terminal.sautDeLigne();
    }
    Stock ajouter(Produit2 p){
        int i=0;
        while ((i<nbProduits) && (tab[i].precede(p))){
            i++;
        }
        for (int j=nbProduits; j>i; j--){
            tab[j]=tab[j-1];
        }
        tab[i]=p;
        nbProduits++;
        return this;
    }
    double prix(int ref){
        int i=0;
        while ((i<nbProduits) && (tab[i].reference != ref)){
            i++;
        }
        if (i == nbProduits){
            throw new NotFound();
        }
        return tab[i].prixHT;
    }
}
class NotFound extends Error{}
class Exo_14_3{
    public static void main(String[] argv){
        Stock st = new Stock();
        st.ajouter(new Produit2("Brosse",10025,12.50));
        st.affiche();
        st.ajouter(new Produit2("Balai",10012,13.75));
        st.affiche();
        st.ajouter(new Produit2("Seau",10019,7.60));
        st.affiche();
        st.ajouter(new Produit2("Chiffon",10032,2.30));
        st.affiche();
        Terminal.ecrireDoubleln(st.prix(10025));
    }
}
```

Le principe de la méthode `ajout` est le suivant : on cherche d'abord l'indice `i` où le produit `p` doit être inséré. Cet indice est soit celui de la première case libre (c'est à dire `nbProduits`), soit la

case du premier produit qui a une référence supérieure à celle de `p`.

Une fois qu'on a déterminé cet emplacement, il faut décaler d'un cran tous les produits qui dans le tableau sont entre cet indice `i` et le dernier produit (contenu de la case d'indice `nbProduits-1`). Pour réaliser ce décalage, il est plus pratique de commencer par la fin, parce que la copie n'efface pas de valeur utile du tableau.

Une fois ce décalage réalisé, on peut mettre `p` dans la case `i` et incrémenter `nbProduits`.

Exercice 2.2.4 *polynômes*

Un polynome est une fonction qui à une variable numérique x associe la somme de facteurs obtenus en multipliant une puissance de x par un certain coefficient entier. Par exemple $f(x) = 3*x^3 - 5*x^2 + 2$. Noter que le coefficient $+2$ s'applique implicitement à la puissance x^0, qui vaut 1.

L'addition de polynômes se fait en additionnant les coefficients puissance par puissance. Par exemple $(3*x^3 - 5*x^2 + 2) + (2*x^3 - 4*x - 3) = 5*x^3 - 5*x^2 - 4*x - 1$

La multiplication de polynômes se fait en utilisant les propriétés de distributivité de la multiplication par rapport à l'addition. Par exemple
$(3*x^3 - 5*x^2 + 2)*(2*x^3 - 4*x - 3) = (3*x^3)*(2*x^3 - 4*x - 3) + (-5*x^2)*(2*x^3 - 4*x - 3) + 2*(2*x^3 - 4*x - 3)$
$= (6*x^6 - 12*x^4 - 9*x^3) + (-10*x^5 + 20*x^3 + 15*x^2) + (4*x^3 - 8*x - 6)$
$= 6*x^6 - 10*x^5 - 12*x^4 + 15*x^3 + 15*x^2 - 8*x - 6$

Ecrivez une classe des polynômes avec les opérations d'addition, de multiplication et de calcul de la valeur du polynôme pour un x donné.

Nous vous proposons deux solutions. Dans la première, nous avons une structure de donnée appelée `Facteur`, qui regroupe un coefficient et un degré. Un polynôme est un tableau de facteurs. Contrairement à beaucoup de programmes que nous avons vus ensemble, le tableau est ici toujours plein, c'est à dire que la taille du tableau est toujours égale au nombre de facteurs dans le polynôme. Nous adoptons une construction progressive des polynômes au moyen de l'opération `ajouteFacteur`, qui permet d'ajouter un facteur à un polynôme. Cette opération joue d'ailleurs un rôle central dans la réalisation des fonctions d'addition et de multiplication.

```
class Facteur{
    int coeff, degre;
    Facteur(int c, int d){
        coeff = c;
        degre = d;
    }
    void affiche(){
        Terminal.ecrireString("" + coeff + "*x^" + degre);
    }
    Facteur multiplie(Facteur f){
        return new Facteur(coeff*f.coeff, degre+f.degre);
    }
    int valeur(int x){
        int res = 1;
        for (int i=1; i<=degre; i++){
            res = res *x;
        }
        return res*coeff;
    }
```

```
}
class Polynome{
    Facteur[] tab=new Facteur[0];
    void affiche(){
        if (tab.length > 0){
            tab[0].affiche();
            for (int i=1; i<tab.length; i++){
                Terminal.ecrireString("+");
                tab[i].affiche();
            }
        }
    }
    void ajouteFacteur(Facteur f){
        boolean ajoute=false;
        for (int i=0; i<tab.length; i++){
            if (tab[i].degre == f.degre){
                tab[i]=new Facteur(tab[i].coeff+f.coeff,f.degre);
                ajoute=true;
            }
        }
        if (!ajoute){
            Facteur[] newTab = new Facteur[tab.length+1];
            for (int i=0; i<tab.length; i++){
                newTab[i]=tab[i];
            }
            newTab[tab.length]=f;
            tab=newTab;
        }
    }
    Polynome ajouter(Polynome p){
        Polynome res = new Polynome();
        for (int i=0; i<tab.length; i++){
            res.ajouteFacteur(tab[i]);
        }
        for (int i=0; i<p.tab.length; i++){
            res.ajouteFacteur(p.tab[i]);
        }
        return res;
    }
    Polynome multiplier(Polynome p){
        Polynome res = new Polynome();
        for (int i=0; i<tab.length; i++){
            for (int j=0; j<p.tab.length;j++){
                res.ajouteFacteur(tab[i].multiplie(p.tab[j]));
            }
        }
        return res;
    }
    int valeur(int x){
        int res = 0;
        for (int i=0; i<tab.length; i++){
            res = res + tab[i].valeur(x);
        }
```

```
        return res;
    }
}
class Exo_14_4{
    public static void main(String[] argv){
        Polynome p1=new Polynome();
        Polynome p2=new Polynome();
        p1.ajouteFacteur(new Facteur(3,3));
        p1.ajouteFacteur(new Facteur(-5,2));
        p1.ajouteFacteur(new Facteur(2,0));
        p1.affiche();
        Terminal.sautDeLigne();
        p2.ajouteFacteur(new Facteur(2,3));
        p2.ajouteFacteur(new Facteur(-4,1));
        p2.ajouteFacteur(new Facteur(-1,0));
        p2.affiche();
        Terminal.sautDeLigne();
        (p1.ajouter(p2)).affiche();
        Terminal.sautDeLigne();
        (p1.multiplier(p2)).affiche();
        Terminal.sautDeLigne();
    }
}
```

La deuxième solution que nous vous proposons est une représentation des polynômes au moyen d'un tableau d'entiers. Ces entiers sont les coefficients, les degrés des facteurs étant représentés par les indices du tableau. Par exemple, le polynôme $(3 * x^3 - 5 * x^2 + 2)$ sera représenté par le tableau suivant :

0	1	2	3
2	0	-5	3

Ici encore, nous avons une opération `ajouteFacteur` qui joue un rôle central.

```
class Polynome2{
    int[] tab=new int[0];
    void affiche(){
        if (tab.length > 0){
            Terminal.ecrireString(""+tab[tab.length-1]+"*x^" + (tab.length-1));
            for (int i=tab.length-2; i>=0; i--){
                if (tab[i]!=0){
                    if (tab[i]>0){
                        Terminal.ecrireString("+");
                    }
                    Terminal.ecrireString(""+tab[i]+"*x^" +i);
                }
            }
        }
    }
    void ajouteFacteur(int coeff,int degre){
        if (degre>= tab.length){
            int[] newTab = new int[degre+1];
            for (int i=0; i<tab.length; i++){
                newTab[i]=tab[i];
            }
```

```
            newTab[degre]=coeff;
            tab=newTab;
        }else{
            tab[degre]=tab[degre]+coeff;
        }
    }
    Polynome2 ajouter(Polynome2 p){
        Polynome2 res = new Polynome2();
        for (int i=0; i<tab.length; i++){
            res.ajouteFacteur(tab[i],i);
        }
        for (int i=0; i<p.tab.length; i++){
            res.ajouteFacteur(p.tab[i],i);
        }
        return res;
    }
    Polynome2 multiplier(Polynome2 p){
        Polynome2 res = new Polynome2();
        for (int i=0; i<tab.length; i++){
            for (int j=0; j<p.tab.length;j++){
                res.ajouteFacteur(tab[i]*p.tab[j],i+j);
            }
        }
        return res;
    }
}
class Exo_14_4_bis{
    public static void main(String[] argv){
        Polynome2 p1=new Polynome2();
        Polynome2 p2=new Polynome2();
        p1.ajouteFacteur(3,3);
        p1.ajouteFacteur(-5,2);
        p1.ajouteFacteur(2,0);
        p1.affiche();
        Terminal.sautDeLigne();
        p2.ajouteFacteur(2,3);
        p2.ajouteFacteur(-4,1);
        p2.ajouteFacteur(-3,0);
        p2.affiche();
        Terminal.sautDeLigne();
        (p1.ajouter(p2)).affiche();
        Terminal.sautDeLigne();
        (p1.multiplier(p2)).affiche();
        Terminal.sautDeLigne();
    }
}
```