Corrigés des exercices sur les fonctions

Exercice 5.1.1 *adaptation de programme*

Question 1

Modifiez le programme pour que le calcul de la division soit réalisé par une fonction à deux paramètres, x et y.

Question 2

Que se passe-t-il dans le cas où le deuxième nombre entré vaut 0 ?

Le programme risque de boucler éternellement si x est positif : en effet, soustraire 0 à chaque tour de boucle ne change en rien la valeur de la condition.

Modifiez le programme pour qu'une erreur soit déclenchée quand le diviseur y vaut 0.

Le programme suivant répond aux deux premières questions.

```
public class Exo8_1bis{
    static int division(int x, int y){
        int res = 0;
        int cour = x;
        if (y == 0){
            throw new DiviseParZero();
        }
        while (cour>=y){
            cour = cour -y;
            res = res+1;
        }
        return res;
    }
    public static void main(String[] args){
        int a, b;
        Terminal.ecrireString("Entrez␣le␣nombre:␣");
        a = Terminal.lireInt();
        Terminal.ecrireString("Entrez␣l'exposant:␣");
        b = Terminal.lireInt();
        Terminal.ecrireString("" + a + "/" + b + "␣=␣");
        Terminal.ecrireIntln(division(a,b));
    }
}
class DiviseParZero extends Error{}
```

Question 3

Modifiez le programme pour qu'il fonctionne également pour x ou y négatifs.

```
public class Exo8_1ter{
    static int division(int x, int y){
        int res = 0;
        int signe = 1;
        int cour, moins;
        if (x<0){
            cour = -x;
            signe = -1;
        }else{
            cour = x;
        }
        if (y<0){
            moins = -y;
            signe = -1 *signe;
        }else{
            moins = y;
        }
        if (y == 0){
            throw new DiviseParZero();
        }
        while (cour>=moins){
            cour = cour -moins;
            res = res+1;
        }
        return res*signe;
    }
    public static void main(String[] args){
        int a, b;
        Terminal.ecrireString("Entrez␣le␣nombre:␣");
        a = Terminal.lireInt();
        Terminal.ecrireString("Entrez␣l'exposant:␣");
        b = Terminal.lireInt();
        Terminal.ecrireString("" + a + "/" + b + "␣=␣");
        Terminal.ecrireIntln(division(a,b));
    }
}
class DiviseParZero extends Error{}
```

Exercice 5.1.2 *fonctions mathématiques*

Ecrire un programme avec les fonctions `carre` et `cube` qui calculent respectivement le carré et le cube (ou puissance 3) d'un nombre de type `double`. La méthode `main` doit tester ces deux fonctions sur plusieurs exemples.

```
class Exo8_2{
    static double carre(double x){
        return x *x;
    }
```

```
    static double cube(double x){
        return x*x*x;
    }
    public static void main(String[] argv){
        double n;
        Terminal.ecrireString("Entrez un nombre: ");
        n = Terminal.lireDouble();
        Terminal.ecrireString("" + n + " au carre = ");
        Terminal.ecrireDoubleln(carre(n));
        Terminal.ecrireString("" + n + " au cube = ");
        Terminal.ecrireDoubleln(cube(n));
    }
}
```

Quelques remarques sur ce programme. Les deux fonctions que nous avons là sont très simples. Il est aussi rapide d'écrire dans le programme `n*n` que l'appel à la fonction `carre(n)`, pour un résultat équivalent. Le seul intérêt éventuel d'une fonction dans un pareil cas est d'améliorer la lisibilité du programme. Ici, c'est relativement discutable.

Autre remarque : voyez comme on évite de faire les entrées-sorties dans les fonctions. Les fonctions servent seulement à calculer. Les entrées-sorties sont effectuées dans la méthode main. On pourrait également utiliser une méthode spécialisée dans les entrées-sorties, mais il faut toujours éviter de mêler des calculs et des entrées-sorties.

Exercice 5.1.3 *égalité de tableaux*

Bien souvent, on veut comparer deux tableaux selon une égalité selon laquelle les deux tableaux `t1` et `t2` sont égaux si et seulement si ils ont la même longueur et les éléments de même indice sont égaux, c'est à dire que `t1[i] == t2[i]` pour tout indice `i`.

Ecrire une fonction appelée `estEgal` qui réalise ce test d'égalité pour des tableaux de type `int[]`.

```
class Exo8_3{
    static boolean estEgal(int[] ti1, int[] ti2){
        boolean res = true;
        if (ti1.length != ti2.length){
            res = false;
        }else{ // ti1.length == ti2.length
            for (int i=0; i<ti1.length; i++){
                res = res && (ti1[i] == ti2[i]);
            }
        }
        return res;
    }
    public static void main(String[] argv){
        int[] t1 = {4, 5, 6};
        int[] t2 = {4, 5, 6};
        int[] t3 = {4, 5};
        int[] t4 = {6, 5, 4};
        Terminal.ecrireStringln("t1 = t2 ? " + estEgal(t1,t2));
        Terminal.ecrireStringln("t1 = t3 ? " + estEgal(t1,t3));
        Terminal.ecrireStringln("t1 = t4 ? " + estEgal(t1,t4));
```

```
    }
}
```

Le principe de la méthode de test est d'abord, traiter le cas des tableaux de longueurs différentes qui ne peuvent être égaux. Ensuite, pour le cas où les deux tableaux sont de même longueur, on fait une boucle `for` dont le compteur `i` va prendre successivement toutes les valeurs des indices des deux tableaux, en commençant par 0. Pour chaque indice, on compare les valeurs des cases de tableaux à cet indice (`ti1[i] == ti2[i]`). Il suffit qu'un seul des tests successifs soit faux pour que la variable `res` contiennent la valeur `false`. C'est là l'intérêt du connecteur logique `&&`. Même si le test faux est suivi de tests vrais, la variable `res` reste à `false`.

On pourrait améliorer quelque peu la fonction en arrêtant le parcours dès qu'on a trouvé un résultat faux. En effet, dès qu'un test échoue, `res` passe à `false` et ne peut plus jamais changer de valeur. On peut interrompre la boucle `for` soit avec une instruction `break`, soit en utilisant un `return` qui met fin à l'exécution de la fonction. Voilà ce que cela donne :

```
static boolean estEgal(int[] ti1, int[] ti2){
    if (ti1.length != ti2.length){
        return false;
    }else{ // ti1.length == ti2.length
        for (int i=0; i<ti1.length; i++){
            if (ti1[i] != ti2[i]){
                return false;
            }
        }
    }
    return true;
}
```

On voit sur cette fonction qu'il faut attendre d'avoir parcouru tout le tableau, c'est à dire attendre d'être arrivé à la fin de la boucle sans avoir fait de `return`, pour pouvoir conclure positivement. Ce serait une erreur grave de faire un `else` au `if` de la boucle : en effet, avoir un test vrai ne permet pas de conclure. Il faut encore voir les éléments suivants, d'où la boucle.

Exercice 5.1.4 *fonctions sur les tableaux*

Question 1

Ecrire une fonction qui cherche si un élément appartient à un tableau de `char`. Le caractère recherché et le tableau seront les deux paramètres de la fonction.

Question 2

Ecrire une fonction qui compte le nombre d'occurrences d'un caractère dans un tableau, c'est à dire le nombre de fois où un élément apparaît dans un tableau de caractères. Le caractère recherché et le tableau seront les deux paramètres de la fonction.

Question 3

Ecrire une fonction qui prend deux tableaux en paramètres et qui teste si tous les éléments du premier tableau apparaissent au moins une fois dans le deuxième tableau. Il est possible d'utiliser

dans le corps de cette fonction la fonction écrite pour la réponse à la question 1.

Voici un programme qui répond aux trois questions.

```
class Exo8_4{
    static boolean appartient(char elem, char[] tab){
        boolean res = false;
        for (int i = 0; i<tab.length; i++){
            if (tab[i] == elem){
                res = true;
            }
        }
        return res;
    }
    static int nbOccurrences(char elem, char[] tab){
        int res = 0;
        for (int i = 0; i<tab.length; i++){
            if (tab[i] == elem){
                res++;
            }
        }
        return res;
    }
    static boolean inclus(char[] t1, char[] t2){
        boolean res = true;
        for (int i = 0; i<t1.length; i++){
            if (!appartient(t1[i],t2)){
                res = false;
            }
        }
        return res;
    }
    public static void main(String[] argv){
        char[] x = {'a','b','a','b','c','a','d'};
        char[] y = {'a','b','a','d'};
        char[] z = {'a', 'z', 'b'};
        Terminal.ecrireStringln("'z' appartient à x ? " +
                                appartient('z',x));
        Terminal.ecrireStringln("'c' appartient à x ? " +
                                appartient('c',x));
        Terminal.ecrireStringln("nombre de 'a' dans x ? " +
                                nbOccurrences('a',x));
        Terminal.ecrireStringln("nombre de 'b' dans x ? " +
                                nbOccurrences('b',x));
        Terminal.ecrireStringln("nombre de 'z' dans x ? " +
                                nbOccurrences('z',x));
        Terminal.ecrireStringln("y inclus dans x ? " +
                                inclus(y,x));
        Terminal.ecrireStringln("z inclus dans x ? " +
                                inclus(z,x));
    }
}
```

La remarque de l'exercice précédent sur la possibilité d'interrompre la boucle peut s'appliquer

aux méthodes `appartient` et `inclus`.

Exercice 5.1.5 *concaténation de tableaux*

On appelle *concaténation* l'opération qui prend deux tableaux et calcule un tableau contenant les éléments du premier tableau, puis, à leur suite, les éléments du second tableau, dans le même ordre, mais avec un indice différent.

Ecrire une fonction qui calcule la concaténation de deux tableaux d'entiers. Indice : il faut créer le tableau résultat dans le corps de la fonction.

```
class Exo8_5{
    static int[] concatene(int[] t1, int[] t2){
        int[] res = new int[t1.length+t2.length];
        for (int i = 0; i<t1.length; i++){
            res[i] = t1[i];
        }
        for (int i = 0; i<t2.length; i++){
            res[t1.length + i] = t2[i];
        }
        return res;
    }
    public static void main(String[] args){
        int[] t1 = {12, 17, 15, 10};
        int[] t2 = {13, 14, 11};
        int[] t3 = concatene(t1,t2);
        for (int i = 0; i<t3.length; i++){
            Terminal.ecrireString("␣" + t3[i]);
        }
        Terminal.sautDeLigne();
    }
}
```

Exercice 5.1.6 *affichage de tableau*

On reprend le programme `AfficheTable` donné dans le cours du chapitre 8.

Question 1

Dans un premier temps, on ne s'intéressera qu'aux entiers strictement positifs. Le nombre de caractères `-` à afficher à chaque tour de boucle dépend du nombre de chiffre de la valeur contenue dans le tableau à l'indice `i`.

```
class Exo8_6_1{
    static void afficheTable(int[] t){
        int nb;
        Terminal.ecrireChar('+');
        for (int i=0; i<t.length; i++){
            nb = t[i];
            Terminal.ecrireChar('-');
            while (nb != 0){
```

```
                Terminal.ecrireChar('-');
                nb = nb / 10;
            }
            Terminal.ecrireString("-+");
        }
        Terminal.sautDeLigne();
        Terminal.ecrireChar('|');
        for (int i=0; i<t.length; i++){
            Terminal.ecrireString(" " + t[i] + "|");
        }
        Terminal.sautDeLigne();
        Terminal.ecrireChar('+');
        for (int i=0; i<t.length; i++){
            nb = t[i];
            Terminal.ecrireChar('-');
            while (nb != 0){
                Terminal.ecrireChar('-');
                nb = nb / 10;
            }
            Terminal.ecrireString("-+");
        }
        Terminal.sautDeLigne();
    }
    public static void main(String[] args){
        int[] ex = {1,5,8,9,7};
        int[] ex2 = {12, 5, 8, 123};
        afficheTable(ex);
        afficheTable(ex2);
    }
}
```

Le principe de cette méthode : on élargit les cases du tableau pour les nombres plus grands. La boucle `while` fait autant de tours qu'il y a de chiffres dans le nombre.

Question 2

Améliorez encore le programme pour qu'il fonctionne avec des entiers négatifs, positifs ou nuls.

Le programme que l'on a fait fonctionne déjà pour les nombres positifs. Quand le nombre est 0, il faut tirer un trait alors que l'on n'entre pas dans la boucle `while` (la condition est fausse dès le départ). Pour les nombres négatifs, il faut ajouter un tiret au nombre de chiffres pour pouvoir tracer le signe `-` devant le chiffre. Dans ces deux cas, il faut ajouter un tiret. C'est ce que fait le programme suivant avec un `if`.

```
class Exo8_6_2{
    static void afficheTable(int[] t){
        int nb;
        Terminal.ecrireChar('+');
        for (int i=0; i<t.length; i++){
            nb = t[i];
            Terminal.ecrireChar('-');
            if (nb <= 0){
                Terminal.ecrireChar('-');
```

```
            }
            while (nb != 0){
                Terminal.ecrireChar('-');
                nb = nb / 10;
            }
            Terminal.ecrireString("-+");
        }
        Terminal.sautDeLigne();
        Terminal.ecrireChar('|');
        for (int i=0; i<t.length; i++){
            Terminal.ecrireString("␣" + t[i] + "␣|");
        }
        Terminal.sautDeLigne();
        Terminal.ecrireChar('+');
        for (int i=0; i<t.length; i++){
            nb = t[i];
            Terminal.ecrireChar('-');
            if (nb <= 0){
                Terminal.ecrireChar('-');
            }
            while (nb != 0){
                Terminal.ecrireChar('-');
                nb = nb / 10;
            }
            Terminal.ecrireString("-+");
        }
        Terminal.sautDeLigne();
    }
    public static void main(String[] args){
        int[] ex = {1,5,8,9,7};
        int[] ex2 = {12, 5, -67, 8, 123, 0, -12};
        afficheTable(ex);
        afficheTable(ex2);
    }
}
```