Corrigés des exercices sur les fonctions récursives

Exercice 7.1.1 *sous-programmes récursifs*

Pour chacun des sous-programmes, nous donnerons les paramètres en précisant le paramètre sur lequel porte la récurrence, le cas de base (valeur de ce paramètre pour lequel le calcul s'arrête) et la variation qui affecte le paramètre à chaque appel récursif.

1. Ecrire un sous-programme récursif qui calcule la somme des n premiers carrés. Par exemple, si n vaut 3, ce sous-programme calculera $1^2 + 2^2 + 3^2$. Ce sous programme n'est défini que pour un n supérieur à 0.
   - **Un seul paramètre n, qui doit être positif.**
   - **cas de base : n=1.**
   - **variation de n à chaque appel : -1**
2. Ecrire un sous-programme récursif qui calcule la somme des éléments positifs d'un tableau.
   - **Deux paramètres : un tableau d'entiers tab et un indice ind. Le but de la fonction est de renvoyer la somme des entiers positifs du tableau compris entre ind et la fin du tableau. Pour avoir le résultat pour tout le tableau, il faut appeler la fonction avec pour indice 0.**
   - **cas de base : ind=tab.length.**
   - **variation de ind à chaque appel : +1**
3. Ecrire un sous-programme récursif qui vérifie si une chaîne de caractère est un palindrôme. Pour cela vous utiliserez les méthodes `charAt` et `length` de la classe `String`. `s.charAt(i)` renvoie le ième caractère de la chaîne `s` et `s.length()` renvoie la longueur de `s`.
   - **Deux paramètres : une chaîne s et un indice ind**
   - **cas de base : ind=s.length()/2. En effet, à chaque appel, on va vérifier la correspondance de 2 caractères. Il est inutile de parcourir le tableau en entier.**
   - **variation de ind à chaque appel : +1**
4. Ecrire un sous-programme récursif qui réarrange les éléments d'un tableau en ordre inverse.
   - **Deux paramètres : une tableau d'entiers tab et un indice ind**
   - **cas de base : ind=tab.length/2. En effet, à chaque appel, on va inverser 2 caractères. Il ne faut surtout pas faire un parcours complet du tableau, sinon chaque élément est changé de place deux fois et revient à sa position d'origine.**
   - **variation de ind à chaque appel : +1**
5. Ecrire un sous-programme récursif qui calcule la valeur numérique d'une chaîne de caractères composée de chiffres. **Ici encore, deux paramètres : la chaîne et un indice. Cette fois, nous parcourons la chaîne de droite à gauche, ce qui simplifie la tâche. Cas de base : 0. Pas de calcul : -1.**

Si cela vous aide, vous pouvez commencer par chercher une formule qui exprime le calcul récursif à effectuer.

Pour les opérations sur les tableaux, vous pourrez vous inspirer de l'exemple `affichageInverse` de la section 13.3.1 du cours.

Remarquez comment est fait le traitement d'exception dans cet exemple ; c'est un peu différent de ce que nous avons vu jusqu'ici.

```
class Exo20_1{
    static int sommePremiersCarres(int n) throws HorsDomaine{
        if (n==1){
            return 1;
        }else if (n>0){
            return (n*n)+sommePremiersCarres(n-1);
        }
        throw HorsDomaine.typique;
    }
    static int sommePositifs(int[] tab, int indice) throws HorsDomaine{
        if (indice >= tab.length){
            return 0;
        }else if (indice>=0){
            if (tab[indice]>0){
                return tab[indice]+sommePositifs(tab, indice+1);
            }else{
                return sommePositifs(tab, indice+1);
            }
        }
        throw HorsDomaine.typique;
    }
    static boolean palindrome(String s, int nieme) throws HorsDomaine{
        if (nieme > s.length() /2){
            return true;
        }
        if ((nieme<0) || (nieme > s.length() /2)){
            throw HorsDomaine.typique;
        }
        return (s.charAt(nieme) == s.charAt(s.length()-nieme-1))
            && palindrome(s, nieme+1);
    }
    static void ordreInverse(int[] tab, int indice) throws HorsDomaine{
        if(indice<0){
            throw HorsDomaine.typique;
        }else if (indice < tab.length /2){
            int tampon;
            tampon = tab[indice];
            tab[indice] = tab[tab.length-indice-1];
            tab[tab.length-indice-1]=tampon;
            ordreInverse(tab,indice+1);
        }
    }
    static int valeurDeChar(char c) throws HorsDomaine{
        if (c == '0'){
            return 0;
        }else if (c == '1'){
```

```
            return 1;
        }else if (c == '2'){
            return 2;
        }else if (c == '3'){
            return 3;
        }else if (c == '4'){
            return 4;
        }else if (c == '5'){
            return 5;
        }else if (c == '6'){
            return 6;
        }else if (c == '7'){
            return 7;
        }else if (c == '8'){
            return 8;
        }else if (c == '9'){
            return 9;
        }
        throw HorsDomaine.typique;
    }
    static int valeurNumerique(String s, int indice) throws HorsDomaine{
        char c = s.charAt(indice);
        if (indice == 0){
            return valeurDeChar(c);
        }else{
            return (valeurNumerique(s, indice-1) *10 + valeurDeChar(c));
        }
    }
    public static void main(String[] args) throws HorsDomaine{
        int[] test = {1, 5, -5, 10, -10, 3};
        Terminal.ecrireIntln(sommePremiersCarres(3));
        Terminal.ecrireIntln(sommePremiersCarres(4));
        Terminal.ecrireIntln(sommePositifs(test,0));
        Terminal.ecrireBooleanln(palindrome("ada",0));
        Terminal.ecrireBooleanln(palindrome("callac",0));
        Terminal.ecrireBooleanln(palindrome("calioplac",0));
        for (int i=0; i<test.length; i++){
            Terminal.ecrireInt(test[i]);
            Terminal.ecrireChar('␣');
        }
        Terminal.sautDeLigne();
        ordreInverse(test,0);
        for (int i=0; i<test.length; i++){
            Terminal.ecrireInt(test[i]);
            Terminal.ecrireChar('␣');
        }
        Terminal.sautDeLigne();
        Terminal.ecrireIntln(valeurNumerique("1234",3));
    }
}
class HorsDomaine extends Exception{
    static HorsDomaine typique = new HorsDomaine();
}
```

Exercice 7.1.2 *Fibonacci*

Ecrire une fonction qui calcule les valeurs de la série de Fibonacci, définie par :

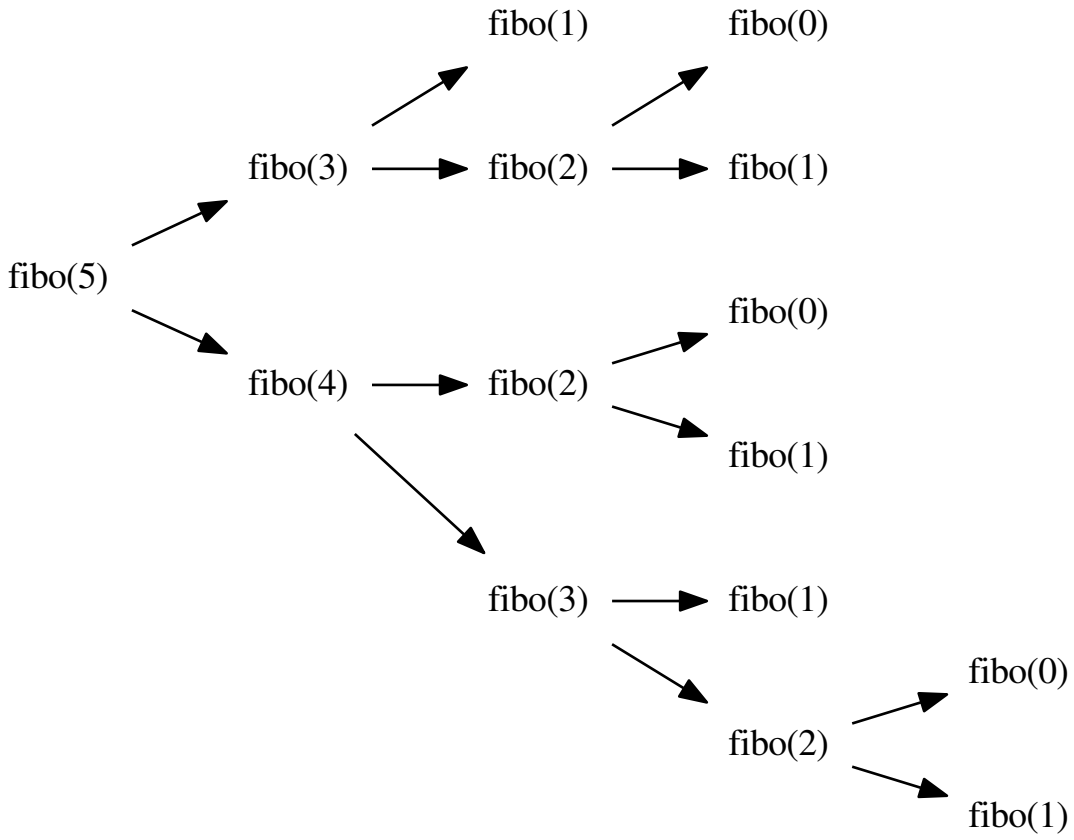
- $u_0 = 0$
- $u_1 = 1$
- $u_n = u_{n-1} + u_{n-2}$

Ecrivez cette fonction sous forme itérative et sous forme récursive. Laquelle des deux variantes est préférable ici ?

```
class Exo20_2{
    static int fiboIteratif(int n){
        if ((n == 0) || (n == 1)){
            return n;
        }else{
            int moinsDeux = 0;
            int moinsUn = 1;
            int nouveau;
            for (int i=2; i<n; i++){
                nouveau = moinsDeux + moinsUn;
                moinsDeux = moinsUn;
                moinsUn = nouveau;
            }
            return moinsDeux + moinsUn;
        }
    }
    static int fiboRecursif(int n){
        if ((n == 0) || (n == 1)){
            return n;
        }else{
            return fiboRecursif(n-2)+fiboRecursif(n-1);
        }
    }
    public static void main(String[] args){
        for (int i=5; i<20; i=i+3){
            Terminal.ecrireStringln("␣" + fiboIteratif(i) + "␣␣" +
                                    fiboRecursif(i));
        }
    }
}
```

La forme récursive est plus facile à écrire et plus proche de la définition de la fonction, mais elle est moins efficace que la version itérative. Dans la version itérative, une valeur de la suite u_n est conservée pendant deux tours de boucles successifs (d'abord dans moinsUn, puis dans moinsDeux), alors que dans la version récursive, comme il n'y a pas de possibilité de conserver une valeur dans une variable entre deux appels récursifs, la valeur est recalculée. Il y a un effet cumulatif.

Voici par exemple les appels effectués pour le calcul de fiboRecursif(5).



On voit que fiboRecursif(2) est appelé trois fois, fiboRecursif(1) quatre fois, etc. La version itérative ne calcule qu'une fois chaque terme de la suite. Elle est donc préférable.