

Bases de Java

II Premières méthodes

Rappel : On peut définir des « fonctions classiques » grâce au mot-clé **static**, c'est-à-dire des méthodes qui ne dépendent pas d'une instance de classe.

Pensez à écrire une méthode main pour vérifier le bon fonctionnement de votre code.

Exercice I (UtilMath)

Définir la classe UtilMath possédant les méthodes suivantes :

- **public static int** somme3(**int** a, **int** b, **int** c) qui fait la somme de trois entiers;
- **public static long** fact(**int** n) qui calcule la factorielle $n!$ d'un entier positif n ;
- **public static long** comb(**int** n, **int** p) qui calcule la combinaison $\binom{n}{p}$ de deux entiers positifs p et n ; ¹
- **public static long** puissance(**int** n, **int** m) qui calcule la puissance m -ème de n : n^m , avec $m \geq 0$.

$$\text{Rappel : } \binom{n}{p} = \frac{n!}{p! \times (n-p)!}, \text{ avec } p \leq n.$$

UtilMath.Java

```
package up.mi.jgm.td01;

public class UtilMath { // Exercice1

    // fait la somme de trois entiers
    public static int somme3(int a, int b, int c)
    {
        return (a + b + c);
    }

    // calcule la factorielle n! d'un entier positif n
    public static long fact(int n)
    {
        long result = 1;
        for(int i = 1 ; i <= n ; i++)
            result *= i;

        return result;
    }

    // calcule la combinaison de deux entiers positifs p et n
    public static long comb(int n, int p)
    {
        return ( ( fact(n) ) / ( fact(p) * fact(n - p) ) );
    }

    // calcule la puissance m-ème de n: n^m, avec m >= 0
    public static long puissance(int n, int m)
    {
        long result = 1;

        for(int i = 1 ; i <= m ; i++)
            result *= n;

        //Si m == 0 , alors result = n^0 = 1
        return result;
    }
}
```

Exercice II (Maximum)

1. Définir une méthode **public static int max2(int a, int b)** qui retourne le maximum des deux entiers donnés en paramètre.
2. Définir une méthode qui retourne le maximum de trois entiers donnés en paramètre, avec deux version différentes :
 - (a) **public static int max3v1(int a, int b, int c)** qui n'utilise pas max2;
 - (b) **public static int max3v2(int a, int b, int c)** qui utilise max2.

```
package up.mi.jgm.td01;

public class Maximum {

    // Retourne le maximum des deux entiers donnés en paramètre.
    public static int max2(int a, int b){
        return ( a > b ) ? a : b ;
    }

    public static int max3v1(int a, int b, int c) // n'utilise pas max2
    {
        int max;

        if( a > b )
            max = a;
        else
            max = b;

        if( c > max )
            max = c;

        return max;
    }

    public static int max3v2(int a, int b, int c) // utilise max2
    {
        return( max2( max2(a,b) , c ) );
    }
}
```

Maximum.Java

Exercice III (Manipulation de tableaux)

Commencez par reprendre la classe UtilTab vue en cours.

Une classe utilitaire pour la manipulation de tableaux

```
package up.mi.jgm.util;  
public class UtilTab {  
    public static void affichageTableau(  
        double [] tab, boolean enLigne) { ... }  
  
    public static boolean appartient(double val,  
        double [] tab) { ... }  
  
    public static double min(double [] tab) { ... }  
  
    public static double max(double [] tab) { ... }  
  
    public static double somme(double [] tab) { ... }  
  
    public static void triParSelection(double [] tab)  
        { ... }  
}
```

01_intro_java.pdf

105

1. Définissez la méthode max.

```

package up.mi.jgm.td01;

public class UtilTab {

    public static double max(double[] tab)
    {
        double tmpMax = tab[0];
        for(int i = 1 ; i < tab.length ; i++){
            if(tab[i] > tmpMax)
                tmpMax = tab[i];
        }

        return tmpMax;
    }
}

```

UtilTab.Java

2. Définir une méthode qui calcule la moyenne des éléments d'un tableau de **double**.

```

package up.mi.jgm.td01;

public class UtilTab {

    // Plus grand élément d'un tableau
    public static double max(double[] tab)
    {
        double tmpMax = tab[0];
        for(int i = 1 ; i < tab.length ; i++){
            if(tab[i] > tmpMax)
                tmpMax = tab[i];
        }

        return tmpMax;
    }

    // Somme des éléments d'un tableau
    public static double somme(double[] tab){
        double tmpSomme = 0;
        for(int i = 0 ; i < tab.length ; i++)
            tmpSomme += tab[i];

        return tmpSomme;
    }

    // Moyenne des éléments d'un tableau de double
    public static double moyenne(double[] tab)
    {
        return( somme(tab) / (double) tab.length );
    }
}

```

UtilTab.Java

3. Définir une méthode qui calcule la médiane des éléments d'un tableau de **double**.

Rappel : la médiane d'une série de nombres est un nombre x tel que la moitié des éléments de la série sont inférieurs à x , et la moitié sont supérieurs à x .

```
package up.mi.jgm.td01;
```

```
public class UtilTab {
```

```
    // ..... *
```

```
/* **** Exercice 3 **** */
```

```
/**
```

```
 * Determine la médiane des éléments d'un tableau de nombres décimaux
```

```
 *
```

```
 * @param tab le tableau dont on veut la médiane
```

```
 * @return la médiane des éléments de tab
```

```
 */
```

```
public static double mediane(double[] tab) {
```

```
    double[] copie = copieTab(tab);
```

```
    triParSelection(copie);
```

```
    int milieu = copie.length / 2;
```

```
    if( (copie.length % 2) == 0 ){
```

```
        return ( (copie[milieu - 1] + copie[milieu]) / 2);
```

```
    } else {
```

```
        return copie[milieu];
```

```
    }
```

```
}
```

```
/**
```

```
 * Cree une copie d'un tableau de double
```

```
 *
```

```
 * @param tab le tableau a copier
```

```
 * @return un nouvel objet equivalent a tab
```

```
 */
```

```
private static double[] copieTab(double[] tab) {
```

```
    double[] copie = new double[tab.length];
```

```
    for(int i = 0; i < tab.length ; i++)
```

```
        copie[i] = tab[i];
```

```
    return copie;
```

```
}
```

UtilTab.Java

```

/**
 * Methode qui trie un tableau par selection. Le tri par selection consiste a
 * chercher l'element le plus petit du tableau, et a l'echanger avec le premier
 * element. Le processus est reitere pour le deuxieme plus petit qui est echange
 * avec le deuxieme element du tableau, etc, jusqu'a ce que le tableau soit
 * trie.
 *
 * @param tab le tableau a trier.
 */
public static void triParSelection(double[] tab) {
    for(int i = 0 ; i < tab.length - 1 ; i++){
        int indiceMin = rechercheIndicePlusPetit(tab, i);

        if(indiceMin != i){
            echanger(tab, i, indiceMin);
        }
    }
}

/**
 * Methode qui permet d'obtenir l'indice du plus petit element d'un tableau a
 * partir d'une position donnee
 *
 * @param tab le tableau dont on cherche le plus petit element
 * @param indiceMin la position a partir de laquelle on recherche le plus petit
 * element
 * @return l'indice du plus petit element de tab situe apres la position
 * indiceMin
 */
private static int rechercheIndicePlusPetit(double[] tab, int indiceMin) {
    for(int j = indiceMin + 1 ; j < tab.length ; j++){
        if(tab[j] < tab[indiceMin]){
            indiceMin = j;
        }
    }

    return indiceMin;
}

/**
 * Methode qui echange deux elements d'un tableau, donnees par leur position
 *
 * @param tab le tableau dans lequel on echange deux elements
 * @param i l'indice du premier element a echanger
 * @param j l'indice du second element a echanger
 */
private static void echanger(double[] tab, int i, int j){
    double tmpVal = tab[i];
    tab[i] = tab[j];
    tab[j] = tmpVal;
}

} // class

```

4. Définir une méthode qui, étant donné un tableau de **double** et un tableau d'**int** de même longueur, calcule la moyenne des éléments du premier tableau pondérée par les éléments du second tableau.

```
package up.mi.jgm.td01;

public class UtilTab {

    /* **** Exercice 1 **** */

    // . . .

    /* **** Exercice 2 **** */

    // ...

    /* **** Exercice 3 **** */

    // . . .

    /* **** Exercice 4 **** */

    /**
     * Méthode qui, étant donné un tableau de double et un tableau d'int de même
longueur,
     * calcule la moyenne des éléments du premier tableau pondérée par les éléments du
second tableau
     */
    private static double moyennePonderee(double[] tab, int[] coeff)
    {
        double result = 0;
        int sumCoeff = 0;

        if(tab.length != coeff.length)
            return -1;

        for(int i = 0 ; i < tab.length ; i++){
            result += ( tab[i] * coeff[i] );
            sumCoeff += coeff[i];
        }

        return ( result / (double) sumCoeff );
    }

} // Class
```


Exercice IV (Modalités de contrôle de connaissance)

Dans de nombreuses unités d'enseignement, la note finale d'un étudiant est la moyenne d'une note de contrôle continu et d'une note d'examen, sauf si la note d'examen est supérieure à celle du contrôle continu. Dans ce cas, seule la note d'examen compte.

1. Définir une méthode qui calcule la note finale d'un étudiant à partir de deux **double** correspondant à sa note de contrôle continu et sa note d'examen.
2. On peut représenter un groupe d'étudiants par un tableau de **double**, tel que si i est un nombre pair, le i -ème élément du tableau est la note de contrôle continu d'un élément, et le $(i+1)$ -ème est sa note d'examen. Définir une méthode qui calcule la moyenne d'un groupe d'étudiants.

On considère pour l'instant que le tableau contient un nombre pair > 0 d'éléments.

On ne gère donc pas les erreurs si ce n'est pas le cas.

```
package up.mi.jgm.td01;

public class Mcc { // Modalités de contrôle de connaissance

    public static double noteUe(double cc, double examen)
    {
        return ( (cc < examen) ? examen : ( (cc + examen)/2 ) );
    }

    public static double moyenneEtudiants(double[] tab)
    {
        double somme = 0;

        for( int i = 0 ; i < (tab.length - 1) ; i+=2 )
            somme += noteUe(tab[i], tab[i + 1]);

        return ( somme / (tab.length / 2) );
    }
}
```

Exercice V (String et formatage de nombres)

Définir une méthode qui prend en entrée un nombre entier < 10000 et qui le formate sous forme de String sur cinq caractères, avec des espaces à gauche. Par exemple,

- `formatage(5)` retourne `" 5"`;
- `formatage(25)` retourne `" 25"`;
- `formatage(325)` retourne `" 325"`.

⚠ Cette méthode retourne un String, mais elle n'affiche rien!

```
package up.mi.jgm.td01;

public class UtilString { // Exercice 5

    public static String formatageDeNombres(int n) {

        if (n >= 10000)
            return null;

        if(n < 10)
            return "    " + n;
        if(n < 100)
            return "   " + n;
        if(n < 1000)
            return "  " + n;

        return " " + n;
    }
}
```

Exercice VI (Interface textuelle pour UtilMath)

1. Écrire un programme qui demande à l'utilisateur s'il souhaite calculer la somme de trois entiers, la factorielle d'un entier, la combinaison de deux entiers, ou la puissance m -ème d'un entier. En fonction de sa réponse, le programme demande ensuite à l'utilisateur le (ou les) nombre(s) nécessaire(s) pour le calcul, et affiche le résultat.
2. Écrire une variante du programme dans laquelle le programme revient au menu après avoir affiché le résultat du calcul. Dans ce cas, il faut prévoir une option "Quitter" en plus du choix des opérations.

```
package up.mi.jgm.td01;
import java.util.Scanner;

public class TestClass {
    static Scanner sc = new Scanner(System.in);

    public static void main(String[] args) {

        int menu;

        do{
            System.out.println("***** MENU *****");
            System.out.println("1 - interface UtilMath");
            System.out.println("2 - quitter");
            System.out.print("Votre choix ? ");
            menu = sc.nextInt();

            switch(menu){
                case 1 : interfaceMath();
            }

        }while(menu != 2);

    }
}
```

```

public static void interfaceMath()
{
    int choix;
    int a, b, c;

    do{
        System.out.println("***** UtilMath - interface *****");
        System.out.println("Menu");
        System.out.println("1 - somme de trois entiers");
        System.out.println("2 - factorielle d'un entier");
        System.out.println("3 - combinaison de deux entiers");
        System.out.println("4 - la puissance m-ème d'un entier");
        System.out.println("5 - quitter");

        System.out.print("Votre choix ? ");
        choix = sc.nextInt();

        switch(choix) {
            case 1 : System.out.print("numero 1 = ? ");
                    a = sc.nextInt();

                    System.out.print("numero 2 = ? ");
                    b = sc.nextInt();

                    System.out.print("numero 3 = ? ");
                    c = sc.nextInt();

                    System.out.println( "Resultat = " + UtilMath.somme3(a, b, c) );

                    break;

            case 2 : System.out.print("numero = ? ");
                    a = sc.nextInt();

                    System.out.println( "Resultat = " + UtilMath.fact(a) );
                    break;

            case 3 : System.out.print("n = ? ");
                    a = sc.nextInt();

                    System.out.print("p = ? ");
                    b = sc.nextInt();

                    System.out.println( "Resultat = " + UtilMath.comb(a, b) );

                    break;

            case 4 : System.out.print("n = ? ");
                    a = sc.nextInt();

                    System.out.print("m = ? ");
                    b = sc.nextInt();

                    System.out.println( "Resultat = " + UtilMath.puissance(a, b) );

        }

    }while( choix != 5);
} // interfaceMath()

} // Class

```

Exercice VII (Interface textuelle pour les MCC)

Écrire un programme qui demande à l'utilisateur un nombre d'étudiants dont on doit entrer les notes, puis pour chaque étudiant sa note de contrôle continu et sa note d'examen. Le programme doit ensuite afficher à l'écran la moyenne du groupe d'étudiants.

```
package up.mi.jgm.td01;
import java.util.Scanner;

public class TestClass {
    static Scanner sc = new Scanner(System.in);

    public static void main(String[] args) {

        int menu;

        do{
            System.out.println("***** MENU *****");
            System.out.println("1 - interface UtilMath");
            System.out.println("2 - interface MCC");
            System.out.println("3 - quitter");
            System.out.print("Votre choix ? ");
            menu = sc.nextInt();

            switch(menu){
                case 1 : interfaceMath();
                        break;
                case 2 : interfaceMcc();
            }

        }while(menu != 3);

    }
}
```

```

public static void interfaceMcc()
{
    int nb;
    double[] notes;
    int j;

    System.out.print("nb etudiants ?");
    nb = sc.nextInt();
    notes = new double[nb * 2];

    j = 1;
    for(int i = 0 ; i < (notes.length - 1) ; i+=2)
    {
        System.out.println("Etudiant numero " + j++);
        System.out.print("Note CC : ");
        notes[i] = sc.nextDouble();
        System.out.print("Note examen : ");
        notes[i + 1] = sc.nextDouble();
    }

    System.out.println("Moyenne = " + Mcc.moyenneEtudiants(notes));
}

public static void interfaceMath()
{

} // interfaceMath()

} // Class

```

Exercice VIII (Nombres premiers)

Un nombre entier est premier s'il a exactement deux diviseurs : 1 et lui-même.

De fait 1 n'est pas premier car il n'a qu'un seul diviseur.

Pour savoir si un nombre n est premier,

il suffit d'essayer de le diviser par tous les entiers $2 \leq k < n$,

et si un d'entre eux est un diviseur de n , alors n n'est pas premier. Dans le cas contraire, n est premier.

On rappelle que n'importe quel nombre entier (> 1)

peut être décomposé de manière unique en un produit de puissances de nombres premiers.

Par exemple, $360 = 8 \times 9 \times 5 = 2^3 \times 3^2 \times 5$.

1. Écrire une méthode qui détermine si un entier n est premier. Plusieurs versions sont possibles.

(a) Définir d'abord une méthode qui vérifie tous les entiers $2 \leq k < n$.

(b) On peut améliorer l'algorithme en vérifiant uniquement pour $k = 2$ et pour les entiers impairs compris entre 2 et n .⁵ Définissez cette variante de la méthode.

(c) On peut encore améliorer l'algorithme en arrêtant les tests si $k > \sqrt{n}$.⁶ Écrivez cette nouvelle version de la méthode.

5. En effet, si n est divisible par un entier pair, il est forcément divisible par 2...

6. Si $n = k \times k'$ avec $k > \sqrt{n}$, alors nécessairement on a déjà trouvé k' plus tôt dans les itérations de la boucle...

```

package up.mi.jgm.td01;

public class Premiers {
    public static boolean isPremierV1(int n) {

        boolean result = true;

        for(int i = 2 ; ( (i < n) && (result == true) ) ; i++) {
            if(n % i == 0)
                result = false;
        }

        return result;
    }

    public static boolean isPremierV2(int n) {
        boolean result = true;

        if(n % 2 == 0)
            result = false;

        for( int i = 3 ; ( (i < n) && (result == true) ) ; i+=2 ) {
            if(n % i == 0)
                result = false;
        }

        return result;
    }

    public static boolean isPremierV3(int n) {
        boolean result = true;

        if(n % 2 == 0)
            result = false;

        for( int i = 3 ; ( (i < n) && (result == true) ) ; i+=2 ) {
            if(n % i == 0)
                result = false;

            if(i > Math.sqrt(n))
                result = false;
        }

        return result;
    }
}

```


2. Définir une méthode qui prend en entrée un nombre entier n , et qui retourne un String correspondant aux n premiers nombres premiers, séparés par un espace.

```
package up.mi.jgm.td01;

public class Premiers {
    public static boolean isPremierV1(int n) {

    }

    public static boolean isPremierV2(int n) {

    }

    public static boolean isPremierV3(int n) {

    }

    public static String nombresPremiers(int n){
        String nombres = "";

        for(int i = 2 ; i <= n ; i++) {
            if(isPremierV3(i))
                nombres += i + " ";
        }

        return nombres;
    }
}
```

3. On souhaite déterminer la décomposition en facteurs premiers d'un nombre entier n .

- (a) Écrire une méthode qui prend en entrée un entier n et un nombre premier⁷ p et qui retourne la puissance à laquelle p apparaît dans la décomposition de n .
Par exemple, si on appelle cette méthode avec $n = 360$ et $p = 2$, on obtient le résultat 3. Avec $p = 7$, on obtient 0.
- (b) Écrire une méthode qui retourne un String correspondant à la décomposition en facteurs premiers d'un entier n .

7. On ne gère pas les erreurs si la méthode est appelée avec un nombre non premier.

```

package up.mi.jgm.td01;

public class Premiers {
    //...

    /**
     * Determine a quelle puissance le nombre premier p apparait dans la
     * decomposition en facteurs premiers de n.
     *
     * @param n un nombre entier
     * @param p un nombre premier
     * @return la puissance a laquelle p apparait dans la decomposition en facteurs
     *         premiers de n.
     */
    public static int puissanceDecomposition(int n, int p) {
        int puissance = 0;
        while (((n / p) > 0) && (n % p) == 0) {
            puissance++;
            n /= p;
        }
        return puissance;
    }

    /**
     * Retourne la decomposition en facteurs premiers d'un nombre sous forme de
     * String
     *
     * @param n le nombre a decomposer
     * @return la decomposition en facteurs premiers de n, sous forme de String
     */
    public static String decompositionFacteursPremiers(int n) {
        StringBuilder build = new StringBuilder();
        if (n < 2) {
            System.err.println("Decomposition impossible");
        }

        int p = 2;
        while (n > 1) {
            if ((n % p) == 0) {
                int k = puissanceDecomposition(n, p);
                build.append("(" + p + ", " + k + " ");
                n /= UtilMath.puissance(p, k);
            }
            p = prochainNombrePremier(p);
        }

        return build.toString();
    }

    /**
     * Determine le prochain nombre premier apres un nombre premier donne
     *
     * @param p le nombre premier de depart
     * @return le plus petit nombre premier strictement plus grand que p
     */
    private static int prochainNombrePremier(int p) {
        for (int i = p + 1; i++; ) {
            if (isPremierV3(i))
                return i;
        }
    }
}

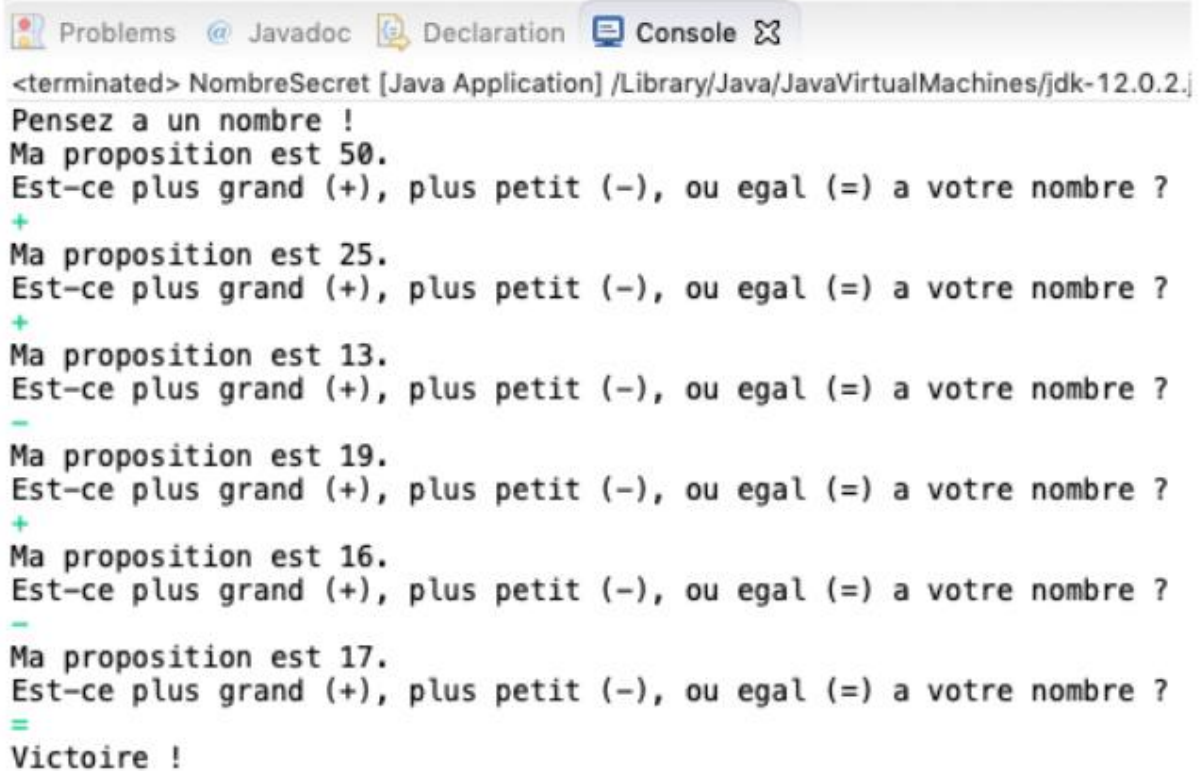
```

Exercice IX (Jeu du nombre secret)

On souhaite implémenter un programme qui joue au jeu du nombre secret avec l'utilisateur :

- Le joueur (humain) pense à un nombre entre 1 et 100;
- L'ordinateur fait une proposition au joueur;
- Le joueur indique au programme si le nombre proposé est plus grand, plus petit, ou égal au nombre secret;
- L'ordinateur continue à chercher jusqu'à ce qu'il trouve.

Écrivez un programme qui permet de jouer à ce jeu. Voici un exemple d'exécution du programme :



```

Problems  Javadoc  Declaration  Console  ✕
<terminated> NombreSecret [Java Application] /Library/Java/JavaVirtualMachines/jdk-12.0.2.j
Pensez a un nombre !
Ma proposition est 50.
Est-ce plus grand (+), plus petit (-), ou egal (=) a votre nombre ?
+
Ma proposition est 25.
Est-ce plus grand (+), plus petit (-), ou egal (=) a votre nombre ?
+
Ma proposition est 13.
Est-ce plus grand (+), plus petit (-), ou egal (=) a votre nombre ?
-
Ma proposition est 19.
Est-ce plus grand (+), plus petit (-), ou egal (=) a votre nombre ?
+
Ma proposition est 16.
Est-ce plus grand (+), plus petit (-), ou egal (=) a votre nombre ?
-
Ma proposition est 17.
Est-ce plus grand (+), plus petit (-), ou egal (=) a votre nombre ?
=
Victoire !
  
```

Jeu.Java

```

package up.mi.jgm.td01;
import java.util.Scanner;

public class Jeu {
    static Scanner sc = new Scanner(System.in);

    public static void startJeu(){
        int min = 1;
        int max = 100;
        char reponse;
        int proposition;

        System.out.println("Pensez a un nombre entre 1 et 100 !");

        do{
            proposition = (int) (Math.random() * (max - min + 1)) + min;
            System.out.println("Ma proposition est " + proposition );
            System.out.println("Est-ce plus grand (+), plus petit (-), ou egal a
votre nombre ?");
            reponse = sc.nextLine().charAt(0);

            switch(reponse)
            {
                case '-' : min = proposition + 1;
                           break;
                case '+' : max = proposition - 1;

            }

        }while(reponse != '=');

        System.out.println("Victoire !");
    }
}

```

TestClass.Java

```

package up.mi.jgm.td01;
import java.util.Scanner;

public class TestClass {
    static Scanner sc = new Scanner(System.in);

    public static void main(String[] args) {

        int menu;

        do{
            System.out.println("***** MENU *****");
            System.out.println("1 - interface UtilMath");
            System.out.println("2 - interface MCC");
            System.out.println("3 - interface jeu");
            System.out.println("4 - quitter");
            System.out.print("Votre choix ? ");
            menu = sc.nextInt();

            switch(menu){
                case 1 : interfaceMath();
                           break;
                case 2 : interfaceMcc();
                           break;
                case 3 : Jeu.startJeu();
            }

        }while(menu != 4);

    }

    // ...
} // Class

```

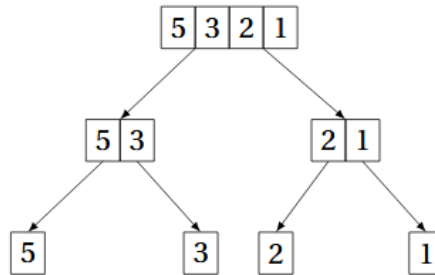
Exercice X (Tri fusion)

On peut trier efficacement des données grâce au principe du tri fusion.

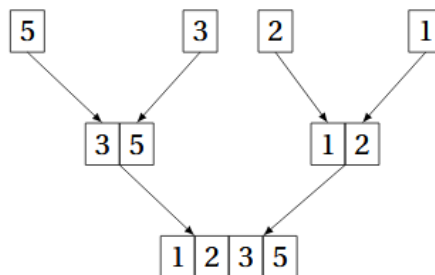
Le principe de l'algorithme est assez simple :

- Un tableau qui contient un seul élément est déjà trié.
- Si un tableau contient plus qu'un élément, on peut le diviser en deux moitiés, trier séparément chaque moitié, puis fusionner les deux moitiés.
- La fusion consiste à inter-classer les éléments de chaque moitié en un seul tableau, ce qui est simple car les deux moitiés sont déjà triées à ce moment là.

Par exemple, on divise récursivement le tableau [5,3,2,1] :



Le processus de fusion consiste à comparer le premier élément de la première moitié, le premier élément de la deuxième moitié, et insérer le plus petit des deux en premier dans le résultat. Puis on réitère avec chaque élément des deux moitiés du tableau.





Tri récursif

- Tri récursif :
 - 1 diviser les données en deux "presque moitiés" ;
 - 2 appeler sur chaque moitié.
- Nous allons voir deux tris récursifs :
 - Tri par fusion (Merge Sort)
 - Tri rapide (Quick Sort)
- Paradigme **Divide-and-Conquer** (diviser pour régner)



Divide and Conquer =

- 1 **Diviser** :

Si les données sont trop grandes pour être traitées de façon directe, alors les diviser en deux ou plusieurs sous-ensemble disjoints ;
- 2 **Appliquer récursivement** le principe "diviser pour régner" sur chaque sous-ensemble ;
- 3 **Conquérir** : Fusionner les solutions des sous-ensembles pour obtenir la solution au problème initial.

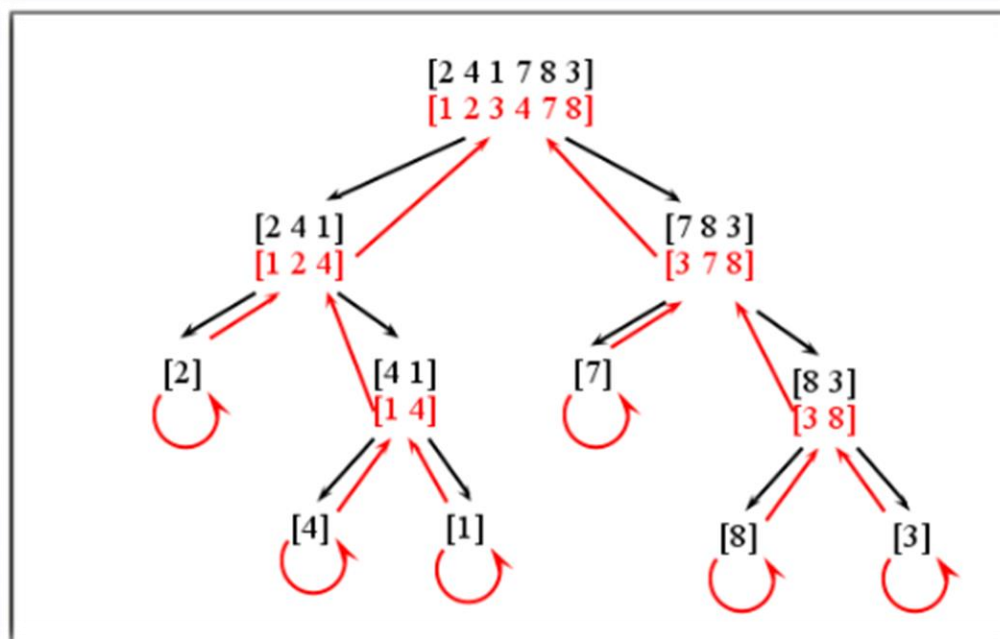
Soit $V : [1, n] \rightarrow E$ un vecteur non trié.

Tri par fusion = 3 étapes :

- ❶ **Diviser** V en 2 sous-vecteurs V_1 et V_2 d'environ $\frac{n}{2}$ éléments chacun
- ❷ **Appliquer récursivement** le tri par fusion sur V_1 et V_2
- ❸ **Conquérir** : fusionner V_1 et V_2 dans un vecteur trié :
 - V_1 et V_2 sont triés
 - Répéter :
comparer le plus petit élément de V_1 avec le plus petit élément de V_2 ,
et insérer le plus petit des deux dans V

L'exécution du tri par fusion est représentée par un arbre binaire :

- Chaque nœud représente un appel récursif du tri par fusion et emmagasine
 - Le vecteur non trié avant l'exécution de l'algorithme
 - Le vecteur trié à la fin de l'exécution
- La racine est l'appel initial
- Les enfants sont les appels pour les sous-vecteurs
- Les feuilles sont les appels pour les vecteurs de taille 1



Algorithme 17 : TriFusion**début**/* ENTRÉES: Un vecteur V de taille n *//* SORTIE: Le vecteur V trié */**si** $n > 1$ **alors** $p \leftarrow n \text{ div } 2$ **pour** $i = 1$ **à** p **faire** $V_1(i) \leftarrow V(i)$ **pour** $i = p + 1$ **à** n **faire** $V_2(i - p) \leftarrow V(i)$ $V_1 \leftarrow \text{TriFusion}(V_1, p)$ $V_2 \leftarrow \text{TriFusion}(V_2, n - p)$ Retour Fusion(V_1, V_2)**sinon** Retour V **fin****Algorithme 18 : Fusion****début**/* ENTRÉES: Deux vecteurs triés V_1 et V_2 de taille p et q *//* SORTIE: Un vecteur V trié */ $i \leftarrow 1 ; j \leftarrow 1 ; k \leftarrow 1$ **tant que** $j \leq p$ **et** $k \leq q$ **faire** **si** $V_1(j) \leq V_2(k)$ **alors** $V(i) \leftarrow V_1(j) ; j \leftarrow j + 1$ **sinon** $V(i) \leftarrow V_2(k) ; k \leftarrow k + 1$ $i \leftarrow i + 1$ **pour** $t = j$ **à** p **faire** $V(i) \leftarrow V_1(t) ; i \leftarrow i + 1$ **pour** $t = k$ **à** q **faire** $V(i) \leftarrow V_2(t) ; i \leftarrow i + 1$ Retour V **fin**

1. Implémenter la méthode suivante :

```
/**
 * Si les elements du tableau t son tries entre iMin et iMilieu - 1
 * d'une part, et entre iMilieu et iMax d'autre part, la methode
 * modifie t pour qu'il soit trie entre iMin et iMax.
 */
public static void fusion(int[] t, int iMin, int iMilieu, int iMax){
    ...
}
```

Conseil : On peut créer un tableau intermédiaire dans lequel les valeurs sont rangées au fur et à mesure de la fusion, avant de les recopier dans le tableau principal à la fin.

2. Implémenter la méthode suivante :

```
/**
 * Trie par fusion les elements de t entre les indices iMin et iMax
 */
public static void triFusion(int[] t, int iMin, int iMax){
    ...
}
```

```
package up.mi.jgm.td01;

public class Fusion {
    /**
     * Trie un tableau d'entier par fusion entre les indices iMin (inclus) et
     iMax
     * (exclus)
     *
     * @param t le tableau a trier
     * @param iMin l'indice de debut de la zone a trier
     * @param iMax l'indice de fin de la zone a trier
     */
    public static void triFusion(int[] t, int iMin, int iMax) {
        if (iMax - iMin > 1) {
            int iMilieu = (iMin + iMax) / 2;
            triFusion(t, iMin, iMilieu);
            triFusion(t, iMilieu, iMax);
            fusion(t, iMin, iMilieu, iMax);
        }
    }
}
```

```

/**
 * Prend en entree un tableau t tel que les valeurs sont trie'es entre iMin
 * (inclus) et iMilieu (exclu) d'une part, et entre iMilieu (inclus) et iMax
 * (exclus) d'autre part. Modifie le tableau pour qu'il soit trie entre iMin
 * (inclus) et iMax (exclus).
 *
 * @param t      le tableau dans lequel on fusionne les donnees
 * @param iMin    l'indice de debut des zones a fusionner
 * @param iMilieu l'indice du pivot entre les zones a fusionner
 * @param iMax    l'indice de la fin des zones a fusionner
 */
public static void fusion(int[] t, int iMin, int iMilieu, int iMax) {
    int[] tmpTab = new int[iMax - iMin];

    int i, j, cpt;
    for (cpt = 0, i = iMin, j = iMilieu; (i < iMilieu) && (j < iMax);) {
        if (t[i] < t[j]) {
            tmpTab[cpt] = t[i];
            cpt++;
            i++;
        } else {
            tmpTab[cpt] = t[j];
            cpt++;
            j++;
        }
    }

    while (i < iMilieu) {
        tmpTab[cpt] = t[i];
        cpt++;
        i++;
    }

    while (j < iMax) {
        tmpTab[cpt] = t[j];
        cpt++;
        j++;
    }

    for (i = 0; i < tmpTab.length; i++) {
        t[iMin + i] = tmpTab[i];
    }
}

```