**Document de travail pour le TP1**

**NE PAS IMPRIMER CE DOCUMENT**
il va être modifié et complété

# Document de travail pour le TP1

L'objectif de ce premier TP est double :

- prendre en main un environnement Java sur lequel vous pourrez travailler. Il est vivement conseillé de choisir l'environnement que vous connaissez le mieux (Windows ou Linux, JavaBeans ou compilateur Java) ;
- écrire et tester les programmes correspondants au travail fait en TD. Sauf mention contraire en TD, vous pouvez travailler en utilisant ou non les capacités Orienté Objet de Java[1].

Prenez l'habitude :

- de mettre des commentaires dans vos programmes (indiqués par //) ;
- d'évaluer vos programmes : compteurs du nombre d'itérations ou d'appels récursifs, temps d'exécution, etc.
- de réfléchir au type des variables pour les exercices des premiers TD (en particulier entre `int` et `long`, respectivement limités à environ $2^{31}$ et $2^{63}$) ;
- de tester dans un premier temps sur de petites valeurs vos programmes récursifs.

**Travail à faire**

- Ecrivez et testez sur de petites valeurs les **deux calculs de pcgd** (Fibonacci et Bézout) vus en TD. Les entiers dont on doit calculer le pgcd seront passés en paramètre au programme à l'exécution.

- Ajoutez à vos programmes des compteurs d'évaluation et faites-les tourner sur de grandes valeurs.

- Préparez une classe tableau d'entiers avec des fonctions permettant d'**afficher** un tableau, d'**initialiser** aléatoirement un **tableau quelconque**, d'initialiser aléatoirement un **tableau croissant**, d'initialiser aléatoirement un **tableau décroissant**. Le nombre effectif d'éléments du tableau sera passé en paramètre au programme à l'exécution. Vous devez tester cette classe avant de passer à la suite.

  ----------pour le tableau croissant
  - choisir au hasard le premier élément (indice *0*),
  - dans une boucle croissante (*i de 1 à N*), choisir à chaque étape un incrément croissant (*inc*) et initialiser *t[i]* avec *t[i-1]+inc*.

  ----------pour le tableau décroissant
  - choisir au hasard le dernier élément (indice *N*),
  - dans une boucle décroissante (*i de N-1 à 0*), choisir à chaque étape un incrément croissant (*inc*) et initialiser *t[i]* avec *t[i+1]+inc*.

- Faites une copie de votre classe tableau d'entiers pour construire une **classe tableau de booléens**, avec une fonction qui initialise systématiquement à *true* ou à *false* tout le tableau, une fonction qui affiche tout le tableau, une fonction qui n'affiche que les éléments à *true*. Le nombre effectif d'éléments du tableau sera passé en paramètre au programme à l'exécution. Vous devez tester cette classe avant de passer à la suite.

- Ajoutez à cette classe la fonction vue en TD qui utilise le **crible d'Eratosthène** pour déterminer les entiers premiers inférieurs à une valeur donnée, passée en paramètre au programme à l'exécution.

- Préparez une **classe matrice et vecteurs d'entiers** avec des fonctions permettant d'afficher des matrices *2 lignes-2 colonnes* et des vecteurs (*1 ligne-2 colonnes*), de calculer le produit de deux matrices ainsi que le produit d'un vecteur par une matrice.

- Ecrivez et testez une solution récursive au problème des tours de Hanoi. Le nombre de galets à déplacer sera passé en paramètre au programme à l'exécution. Cet exercice (vu en L1) ne sera pas préparé en TD.

1 Voir l'annexe 2...

# Annexe 1- Java

Vous pouvez avoir besoin des éléments suivants :

### Générateur aléatoire

http://docs.oracle.com/javase/6/docs/api/java/util/Random.html/

déclaration d'importation
```
import java.util.Random;
```
déclaration de variable
```
Random generateur = new Random();
```
appel
```
generateur.nextInt(1000); // entier int dans [0,1000[
```

### Caractères unicode

http://www.unicode.org/charts/

### Détection du temps

Il ne semble pas exister de solution générale pour mesurer de façon précise le temps d'exécution d'une partie de programme (toutes les mesures de temps dépendent fortement du contexte d'exécution : ordinateur, système et son paramétrage, etc.). Vous pouvez utiliser la fonction `System.nanoTime();` qui renvoie sous forme d'entier long le temps écoulé depuis une origine non/mal connue[2]. Pour obtenir une durée en secondes, vous pouvez utiliser :

déclaration de variable
```
long startTime;
float duree;
```
appels
```
startTime = System.nanoTime();
ICI LA PARTIE DE PROGRAMME A EVALUER
duree=(System.nanoTime()-startTime) / 1000000.0f;
```
la notation `1000000.0f` est utilisée pour indiquer que la valeur 1000000 doit être en format `float`

### Fonctions mathématiques

Voir http://docs.oracle.com/javase/7/docs/api/java/lang/Math.html

racine carrée définie avec
```
paramètre et résultat : double
```
appel
```
Math.sqrt(x)
```

### Passage de paramètre à un programme Java

Pour récupérer dans un programme java des paramètres fournis lors de l'exécution, vous pouvez utiliser en début de main() :

```
public static void main (String[] args)
  { int n1 = Integer.parseInt(args[0]);
    int n2 = Integer.parseInt(args[1]);
```

### Tableaux

Les tableaux en Java ont des indices variant entre *0* et *n-1* où *n* est le nombre d'éléments qu'ils contiennent. Pour des matrices *2 lignes-2 colonnes*, utilisez toujours la même convention : le premier indice est celui de la ligne, le second indice est celui de la colonne. Vous trouverez ci-dessous un rappel de notations utilisables :

déclaration d'une matrice 2 lignes-2 colonnes
```
long pr[][] = new long[2][2];
```

déclaration et initialisation ligne par ligne d'une matrice 2 lignes-2 colonnes
```
long[][] F = {{1,1},{1,0}};    // pour Fibonacci matrice   1   1
                               //                          1   0
```

2 API QueryPerformanceCounter sous Windows, fonction clock_gettime avec CLOCK_MONOTONIC sous Linux; pour plus de détails voir http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/System.html/ et http://www.javacodegeeks.com/2012/02/what-is-behind-systemnanotime.html/

schéma d'une fonction de calcul d'un produit de matrices

```
public static long[][] produitM ( long[][] A, long[][] B )
   { long res[][] = new long[2][2];     // matrice résultat
     // ICI LES CALCULS
     return res;
    }
```

appel de la fonction produit de deux matrices

```
pr= produitM ( F,F );
```

**Quelques messages d'erreur de javac**

```
► error: Class names, 'hanoi', are only accepted if annotation processing is explicitly
requested
```

manque l'extension .java au lancement du compilateur javac sur hanoi.java

```
► Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 10
  at TD1question12b.initialiser(TD1question12b.java:22)
```

débordement dans un tableau, fonction initialiser(), ligne 22 du programme

```
► TD1question12b.java:20: cannot find symbol
  symbol  : variable i
  location: class TD1question12b
     for (  i=1; i <= n; i++)
            ^
```

variable non déclarée

# Annexe 2- Orienté objet ou non orienté objet ?

Java vous permet de travailler en orienté-objet ou non. C'est principalement le modificateur *static* qui vous permet d'en décider. Vous trouverez ci-dessous un extrait de l'exemple l'exemple détaillé et commenté qui est proposé dans « *Java in a nutshell*[3] », pages 101 à 108 :

```
public class Circle
  {
       public static final double Pi=3.14159 ;              // champ de classe

       public static double radiansToDegrees (double rad)   // méthode de classe
           { return rad*180/Pi ;
           }

       public double r ;                                     // champ d'instance

       public double area ()                                 // méthode d'instance
           { return Pi*r*r ;
           }
   }
```

– le **champ de classe** est accessible sans préfixe depuis la classe `Circle` (voir dans la fonction `area`)
avec en prefixe le nom de la classe depuis l'extérieur de la classe

3 « Java in a Nutshell, Manuel de référence », David Flanagan, traduit par Alexandre Gachet, éditeur O'Reilly, 2002.

- la **méthode de classe** est accessible sans préfixe depuis la classe `Circle`
  avec en prefixe le nom de la classe depuis l'extérieur de la classe (comme `Math.sqrt()` par exemple)
- le **champ d'instance** est accessible avec en préfixe l'objet qui le possède
- la **méthode d'instance** est accessible avec en préfixe l'objet qui l'exécute
- dans la définition d'une méthode d'instance, le terme `this` (qui est implicite) désigne l'objet qui exécute la méthode. Les champs de l'objet qui exécute sont accessibles sans prefixe ou avec le préfixe `this`

Toujours dans « *Java in a nutshell* », pages 131 à 133, l'exemple de la classe *Circle* est repris afin de montrer comment utiliser les modificateurs *public*, *protected* et *private*.

La figure ci-dessous reprend de façon très schématique certaines caractéristiques des langages orienté-objet ou non orienté objet, mises en perspective sur les notions de champs et fonctions de classe ou d'instance. Ce schéma vous propose quelques pistes pour choisir le style de programation à utiliser dans les exercices qui vous sont proposés :

- en **orienté-objet strict** : **champs et méthodes d'instance** (sauf pour les services proposés par certaines classes pré-définies de Java ;
- l'orienté-objet peut être assoupli en utilisant des méthodes et champs de classe pour proposer des services (valeurs ou calculs particuliers) ;
- en **non orienté-objet strict** : **méthodes de classe avec éventuellement des champs de classe constants** (*final*) ;
- le non orienté-objet peut être assoupli en utilisant de façon limitée des champs globaux non constants (voir par exemple la correction des coefficients de Bézout faite en TD et disponible sur le serveur pédagogique *ufrsciencestech.u-bourgogne.fr*).