Programmation Orientée Objet (Java)
*****

TD 1 : instructions de base

Exercice 1 : Nombre Mystérieux
Le but de cet exercice est l'écriture d'un programme qui permet à l'utilisateur de deviner un nombre entier choisi au hasard par l'ordinateur. **Les instructions qui permettent le tirage aléatoire d'un entier appartenant à l'intervalle [min, max] sont dans la bibliothèque « java.util ». Ces instructions sont les suivantes :**

```
Random rand = new Random();
int r= rand.nextInt(max - min) + min;
```

Version 1 : on est obligé de gagner ! Écrire un programme qui permet :
• le tirage aléatoire d'un entier appartenant à l'intervalle [1, 50],
• permet à l'utilisateur de faire des propositions pour deviner ce nombre ; à chaque proposition le joueur sera informé que le nombre qu'il a donné est «trop grand » ou « trop petit »,
• À la fin du jeu, le nombre d'essais nécessaires pour trouver le nombre mystérieux est affiché.
Version 2 : on peut perdre ! Modifier le programme pour n'autoriser qu'un maximum de 10 propositions. Dans le cas où le joueur ne découvre pas le nombre mystérieux au bout des 10 essais, la solution lui sera donnée.

Exercice 2 : Moyenne olympique
Ecrire un programme qui lit au clavier une suite de nombres réels positifs ou nuls (correspondant à des notes), terminée par la valeur -1, et calcule la moyenne olympique de ces valeurs, c'est à dire la moyenne des notes sans prendre en compte la note la plus élevée ni la note la moins élevée.
Exemple de trace d'exécution (en gras les valeurs introduites par l'utilisateur):

```
donnez une note ( >=0 ou -1 pour arrêter): 9.6
donnez une note ( >=0 ou -1 pour arrêter): 9.7
donnez une note ( >=0 ou -1 pour arrêter): 10.0
donnez une note ( >=0 ou -1 pour arrêter): 9.8
donnez une note ( >=0 ou -1 pour arrêter): 9.2
donnez une note ( >=0 ou -1 pour arrêter): 9.9
donnez une note ( >=0 ou -1 pour arrêter): -1

La note la plus élevée (10.0) et la note plus basse (9.2) ont été retirées
La moyenne olympique est : 9.75
```

Exercice 3 : Affichage de motifs
Écrire un programme affiche un motif triangulaire sous forme d'une pyramide (voir ci-dessous) dont la taille est fixée par une valeur lue au clavier.

Exemple de trace d'exécution (en gras les valeurs introduites par l'utilisateur):

```
donnez la taille du motif : 7

       *
      ***
     *****
    *******
   *********
  ***********
*************
```

Les Wrappers

Classes Integer, **Float**, **Boolean** etc. du package **java.lang**. Ce sont des sous-classes de la classe **Number**. Ces classes détiennent principalement des méthodes pour convertir entre elles des données numériques, surtout vers (ou à partir de) leur forme chaîne de caractères.

a) On considérera le cas de la classe **Integer**, les autres sont semblables.

String vers **Integer**. Fonction **valueOf().** Correspondance entre la chaîne "123" et l'entier **Integer** de valeur 123.

```
Integer I;
I = Integer.valueOf("123");
```

int vers **Integer**. Correspondance inverse de **int** vers **Integer**

```
Integer I = new Integer (i);    // Par constructeur
```

- Discuter (The constructor Integer(int) has been deprecated since version 9).
- Essayer maintenant de convertir l'entier **123** en un objet Integer de valeur **123** :

Integer I = 123 ; // Autoboxing

N.B. L'autoboxing fait référence à la conversion d'une valeur primitive en un objet de la classe wrapper correspondante.

String vers **int**. Fonction parseInt**()**. Correspondance entre la chaîne "456" et l'entier de valeur 456.

```
int i;
i = Integer.parseInt("456");
```

Integer vers **int**. Fonction **intValue().** Correspondance entre objet **Integer** et entier **int**

```
int i; Integer I;
i =  I.intValue();
```

- Remarquer que c'est une méthode d'instance ici (fonction d'accès).
- Essayer maintenant une affectation directe : **i = I ;** // Auto-unboxing

N.B. L'auto-unboxing fait référence à la conversion d'un objet d'un type wrapper en sa valeur primitive correspondante.

Integer ou **int** vers **String**. Passage réciproque de **Integer** ou **int** vers une chaîne **String**.

Méthode générale **valueOf()** de la classe **String** s'appliquant (par surcharge) à tous les types primitifs.

```
int i = 34;
s = String.valueOf(i);       // s devient "34"
```

Méthode **toString()** de la classe **Integer** ici.

```
String s;
Integer I = 345;
s = I.toString();       // s devient "345"
```

Cette méthode est intéressante car elle est héritée de la classe **Object** et peut donc s'appliquer à tout objet si elle est redéfinie. On peut l'utiliser à profit pour imprimer un

objet **println(objet.toString);.**

Comparaison entre deux objets **Integer**. Résultat **int**

```
int i = I.compareTo(J);     // 0 si I = J, négatif si I<J, positif si I>J
```

On peut aussi utiliser les opérateurs de comparaison comme **I < J**.

b) Le cas des autres classes est analogue. Testez-les (**Float, Double**, …)