Tutoriel et évaluation du niveau de connaissance en programmation Java

ensIIE - FISA
MOdélisation OBjet & MEmoires IMperatives
2025-2026

Cette première séance du cours de Modélisation Orientée OBjet (MOOB) est un tutoriel de programmation en Java. Durant ce tutoriel, vous allez survoler les concepts qui seront abordés dans les cours de MOOB, tout en vous préparant pour le tutoriel de C pour lequel de nombreux aspects syntaxiques seront identiques. L'objectif est, premièrement, de vous donner un support dans lequel vous trouverez une base pour programmer en Java ; et, deuxièmement, d'évaluer le niveau de chacun. À l'issue de cette séance, si vous n'aviez jamais codé en Java avant, vous aurez normalement une bonne intuition de ce qu'est un programme et des bases de l'algorithmique et de la programmation dans ce langage.

Le langage Java étant un langage orienté objet (nous verrons plus tard ce que cela veut dire), nous ne pourrons éviter quelques allusions à cet aspect. Nous nous contenterons du minimum dans ce document.

Ce TP se fera sous linux, sur une machine de l'école. Si vous avez décidé d'installer un autre système d'exploitation (Mac ou Windows), vous serez livrés à vous même pour tout ce qui concerne l'exécution du programme.

1 Obtenir de l'aide

Durant ce TP, vous aurez peut être besoin d'aide. Votre chargé de TP sera peut être occupé à aider une autre personne. Vous êtes bien entendu autorisés à utiliser votre moteur de recherche préféré pour chercher des réponses. Cependant :

- restez critiques quant aux réponses que vous trouverez en ligne. Attention à ne pas tomber sur une source obsolète ou fausse.
- la documentation de l'API Java peut être trouvée ici : `https://docs.oracle.com/en/java/javase/21/docs/api/index.html`. Contrairement au C, Java n'est pas le langage de programmation natif d'Ubuntu. La commande `man` ne vous servira donc pas pour trouver des informations. Néanmoins, la document Java est très complète, n'hésitez pas à vous y référer.
- vous avez, en théorie, toutes les explications nécessaires pour résoudre un exercice en début de chaque section. Si la solution que vous trouvez implique d'utiliser un outil plus évolué, de télécharger une librairie tierce, ce n'est probablement pas ce qui est attendu. Plus généralement, évitez de copier coller du code trouvé en ligne sans comprendre de quoi il s'agit.
- il est **fortement découragé** d'utiliser ChatGPT ou toute autre IA générationnelle pour ce tutoriel visant à acquérir les bases d'un langage. Cela n'aide pas à acquérir les fondamentaux minimum à la bonne utilisation de ces outils.

2 Rapide introduction au Shell.

Avant de passer à l'initiation Java, il est important que vous sachiez utiliser une console sous linux.

Lorsque vous utilisez votre ordinateur, vous avez deux modes : graphique et console. On va s'intéresser ici à l'exploration des fichiers. En mode graphique, vous utilisez l'explorateur de fichiers.

Vous disposez de fichiers, qui sont rangés dans des dossiers, qui sont eux-mêmes rangés dans des dossiers, ... Généralement, on parle d'arborescence de fichiers pour désigner la manière dont les fichiers sont rangés dans une machine. On parle d'arborescence car on peut redessiner les fichiers sous la forme de la figure 1, comme les branches d'un arbres qui se séparent et se ramifient jusqu'à atteindre les feuilles.

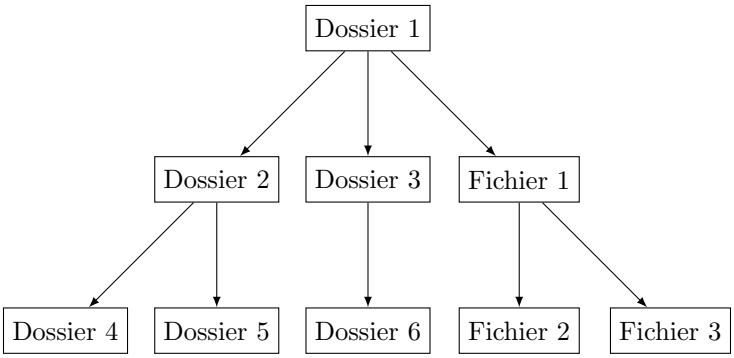

FIGURE 1 – Arborescence de fichiers

Il est possible, en mode graphique de parcourir ces fichiers et ces dossiers en cliquant sur les dossiers. Vous pouvez faire la même chose en mode console : c'est à dire en tapant des commandes dans une console (aussi appelé terminal). Pour vous déplacer et visualiser les contenus de répertoire, vous allez utiliser 2 commandes, `ls` et `cd`.

2.1 Tutoriel

Commencez par ouvrir un terminal. Sous Ubuntu, il suffit d'ouvrir le menu (icone Ubuntu de mémoire en haut à gauche, ou touche windows) et de chercher Terminal dans la barre de recherche. Vous pouvez aussi appuyer sur Ctrl+Alt+T. Lorsque vous ouvrez ce terminal, vous êtes dans un explorateur de fichiers textuel (*i.e.* on ne peut pas cliquer dedans) et dans lequel vous pouvez faire plein d'opérations dont la portée dépassent le cadre de ce cours (ce sera pour le cours de SYEX !). On ne va s'intéresser qu'à 2 choses dans ce cours : reproduire ce qu'on sait faire en graphique et compiler/exécuter du Java (et du C de la même manière dans l'autre tutoriel).

En ouvrant le terminal, vous devriez voir quelque chose comme

```
> machin:~$
```

Sous linux, l'arborescence est constituée généralement de deux parties distinctes : le dossier de l'utilisateur et le reste. Le dossier de l'utilisateur est ce qu'on appelle son dossier `home` ; généralement représenté par une tilde `~`. Quand vous ouvre votre terminal, vous êtes donc dans votre dossier `home`.

Vous pouvez afficher ce qui se trouve dans votre dossier home avec la commande `ls`.

```
> machin:~$ ls
> Bureau Images Documents Enregistrements Musique Telechargements text.txt
> machin:~$
```

Vous pouvez vous déplacer dans un dossier de `~` en utilisant la commande `cd`.

```
> machin:~$ cd Documents
> machin:~/Documents$ ls
> Autres Personnel Travail img.png poeme.txt
```

Vous vous trouvez maintenant dans le dossier `~/Documents` c'est-à-dire le dossier qui se nomme `Documents` et qui se trouve dans `~`.

Pour remonter d'un cran, utilisez la commande `cd ..`

```
> machin:~/Documents$ cd ..
> machin:~$
```

Vous pouvez enchaîner pour aller plus vite.

```
> machin:~$ cd Documents/Personnel
> machin:~/Documents/Personnel$ cd ../../Bureau
> machin:~/Bureau$
```

Enfin, vous pouvez créer un nouveau dossier avec la commande `mkdir` et un nouveau fichier avec la commande `touch` :

```
> machin:~/Bureau$ ls
>
> machin:~/Bureau$ mkdir Jeux
> machin:~/Bureau$ ls
> Jeux
> machin:~/Bureau$ touch fichier.txt
> machin:~/Bureau$ ls
> Jeux fichier.txt
> machin:~/Bureau$ cd Jeux
> machin:~/Bureau/Jeux$
```

Manipulez ces commandes tant que possible si vous n'y êtes pas habitué, c'est très pratique.

Vous êtes perdus ? : utilisez la commande `cd` seule depuis n'importe où pour retourner dans votre dossier `home`.

```
>   machin:~/Bureau/Jeux$ cd
>   machin:~
```

3 Votre premier programme en Java

Vous pouvez sauter cette première partie si vous avez déjà codé en Java et déjà compilé un programme avec la machine virtuelle Java. Sinon, ce tutoriel a pour but de repasser sur les transparents de la présentation et de vous initier à la programmation Java. Utilisez les pour vous aider à coder !

3.1 Récupération des fichiers nécessaires au tutoriel

Nous allons tout d'abord récupérer toutes les ressources de ce TP, qui sont sur un répertoire partagé appelé `/pub`. En faisant dans un terminal `ls /pub/`, vous verrez que le dossier est organisé entre FISA et FISE, et que chacun contiennent une répertoire pour chaque UE et module de chaque année.

Nous allons faire les manipulations suivantes :

1. Ouvrez un terminal, et déplacez vous sur le bureau : `cd Bureau`.
2. Créez un répertoire `MOOB` que vous utiliserez pour l'ensemble des TP du cours de MOOB avec la commande `mkdir MOOB`.
3. Dans ce répertoire, créer un sous-dossier pour cette séance nommé `TP0_java` avec la commande `mkdir -p MOOB/TP0_java`.
4. Déplacez vous dans ce dossier avec la commande `cd MOOB/TP0_java`.
5. Copiez l'ensemble des ressources du TP fournies dans le répertoire `/pub/FISA_INFO11/MEIM/tp/tp_intro_java`. Ainsi, vous aurez fait toutes les commandes suivantes :

```
> machin:~$ cp /pub/FISA\_INFO11/MEIM/tp/tp\_intro\_java/* ~/Bureau/MOOB/TP0\_java
>
```

En faisant la commande `ls ~/Bureau/MOOB/TP0_java`, vous devriez pouvoir lister tous les fichiers nécessaires à ce tp (fichiers .java, le pdf de ce sujet etc).

3.2 Un premier programme Java

Attaquons dans le concret, et faisons un premier programme Java que nous allons compiler puis lancer.

1. Créez un nouveau fichier `HelloWorld.java` (avec la commande `touch HelloWorld.java`) dans votre répertoire `TP0_java`
2. Ouvrez ce fichier avec l'éditeur de votre choix (`vim`, `emacs`, `nano`, `code`, ...). Si vous n'avez pas l'habitude du mode console, nous vous conseillons la commande `code HelloWorld.java` pour ce TP.
3. Remplissez le fichier avec le contenu suivant et sauvegardez :

```
1     public class HelloWorld {
2       public static void main ( String [] args ) {
3         // Prints " Hello , World " in the terminal window
4         System.out.println("Hello, World!");
5       }
6     }
7
```

4. Ouvrez un second terminal ou un second onglet dans le premier terminal et placez vous dans le même sous-dossier.
5. Tapez la commande suivante : `javac HelloWorld.java`.
6. A l'aide de la commande `ls`, vous pourrez voir qu'il y a un nouveau fichier `HelloWorld.class` qui vient d'apparaître.

7. Enfin tapez la commande `java HelloWorld`. Votre terminal devrait ressembler à ceci :

```
> machin:~$ cd ~/IAP/TP1_java
> machin:~/IAP/TP1_java$ javac HelloWorld.java
> machin:~/IAP/TP1_java$ ls
> HelloWorld.javalass HelloWorld.java
> machin:~/IAP/TP1_java$ java HelloWorld
> Hello,  World!
```

Félicitations, vous avez écrit votre premier programme en Java, le *classique Hello World*. Quelques explications concernant ce que nous venons de faire :

— Dans la ligne 1, nous avons le mot clé `class` qui apparaît comme nous l'avons présenté dans les transparents. Une classe correspond à un ensemble de données encapsulé pour raison sémantique.

— Très important en Java : il n'y a **qu'une seule** classe par fichier. De plus, le nom de la classe doit commencer par une majuscule et correspondre exactement au nom du fichier ! C'est à dire qu'une classe `HelloWorld` devra être contenue dans un fichier `HelloWorld.java`

— Dans le programme (c'est à dire la méthode `main` dans la classe), nous avons indiqué à la ligne 4 d'afficher "Hello, World !" à l'aide de la *méthode*[1] `println` du module `System.out`. `System` est une classe qui est automatiquement importée, donc pas besoin d'ajouter un `import`, qui correspondrait au `#include <stdio.h>` en C. Remarquez que la méthode `println` ajoute automatiquement un retour à la ligne (`ln` équivaut à `line` en anglais). Si vous ne le souhaitez pas, vous pouvez utiliser la méthode `print` du même module `System.out`. Vous pouvez chercher dans la documentation Java les spécificités de la classe `System.out` : `https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/lang/System.html#out`

— La ligne 3 est un *commentaire*. Un commentaire est toujours ignoré par la machine. Il sert à donner des informations aux personnes qui reliront le code. La syntaxe est la même en C et en Java.

— La ligne 2 définit le programme, délimité avec des accolades. En Java, un programme est une classe contenant une méthode `public static void main`. Lorsque vous exécutez un programme, la machine cherche cette méthode `main` et exécute son contenu. Tout programme Java démarre son exécution par la méthode `main` de la classe exécutée. Comme en C, on éxécute toutes les instructions écrite entre les deux caractères { et }. La méthode prend éventuellement un tableau de chaîne de caractères (`String [] args`), nous verrons plus tard à quoi cela correspond.

Le fichier `HelloWorld.java` n'est qu'un fichier texte. Il ne peut être exécuté tel quel par la machine virtuelle Java. Nous avons donc, à l'étape 9, *compilé* ce programme avec le compilateur Java qui s'appelle `javac`. Cependant, le langage Java est à la fois un langage compilé et intreprété. `javac` génère à partir d'un fichier `.java` un équivalent *bytecode* sous la forme d'un fichier `.class`. Pour exécuter ce *bytecode*, il faut utiliser la machine virtuelle Java à l'aide de la commande `java`. Comme les binaires C (cf cet après-midi), si vous l'ouvrez vous ne verrez que des symboles illisibles.

Revenons au code du programme. Pourquoi la méthode `main` est-elle déclarée avec `static` ? Comprendre ceci nécessite de rentrer dans les détails de la programmation orientée objet, que nous verrons en MOOB.
Sachez simplement qu'en programmation objet, on manipule des objets (ayant pour type leur classe). Les classes sont à voir comme des moules, à partir desquels on peut fabriquer plusieurs objets qui auront les caractéristiques du moule. Par exemple, avec une classe `Personne`, je peux créer une variable `Personne p1 = new Personne();` et `Personne p2 = new Personne();`. Ici, `p1` et `p2` seront des instances de la classe `Personne`, avec toutes ses caractéristiques. Nous verrons tout cela en détail en MOOB. Pour l'instant, nous n'aurons besoin que d'une classe contenant notre méthode `main` représentant notre programme. Toute classe contenant une méthode `main` est une **classe exécutable**.

1. Une méthode est une fonction appartement à une classe. Ainsi, `main` est une méthode de la classe `HelloWorld`.

Nous verrons dans le module MOOB comment compiler plusieurs classes dans un même programme, ainsi que les détails de la programmation Java. Pour le moment, on va se contenter d'avoir une seule classe, avec une méthode `main` correspondant à notre programme. Nous verrons dans les sections suivantes comment ajouter d'autres méthodes que `main` à notre classe pour lui donner des fonctionnalités supplémentaires.

3.3 Exercices

Exercice 1 — *Affichage simple*

Créer, compilez et exécutez un programme `Lorem.java` qui affiche le texte suivant (en respectant les sauts de ligne) Quel doit donc être le nom de la classe apparaissant dans ce fichier ?

```
> Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut
    labore et dolore magna aliqua.
> Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea
    commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum
    dolore eu fugiat nulla pariatur.
>
> Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit
    anim id est laborum.
```

Exercice 2 — *Erreur de compilation*

Copiez le programme `HelloWorld.java` du tutoriel dans un fichier `HelloWorldError.java` avec la commande
`cp HelloWorld.java HelloWorldError.java`. Supprimez la première ligne et la dernière ligne (c'est à dire la déclaration de la classe), et compilez pour constater que le programme ne compile pas.

4 Création d'objets, variables, attributs et types de base en Java (A lire)

En programmation objet, on sépare la notion d'attribut et de variables. On s'attache dans cette section à présenter les bases de ces principes. Revenons tout d'abord sur comment les objets sont créés.

4.1 Notion de constructeur

Précédemment, nous avons pris l'exemple d'une classe `Personne`. A partir de cette classe, nous pouvons créer deux instances `Personne p1 = new Personne();` et `Personne p2 = new Personne();` de cette classe. Dans ces deux codes, la méthode `Personne()` est un constructeur. Cette méthode porte le même nom que la classe qui la contient et ne doit pas indiquer de valeur de retour ; un constructeur renvoie implicitement la référence de l'instance construite. Toute classe possède au moins un constructeur (celui par défaut). Les constructeurs sert en particulier à initialiser les attributs de la classe. La première ligne d'un constructeur est :

- soit un appel au constructeur de la superclasse, appel effectué à l'aide du mot `super` suivi de parenthèses contenant éventuellement des paramètres (ces aspects seront vus plus tard dans le cours) ;
- soit un appel à un autre constructeur de la même classe ayant un jeu différent de paramètres, appel effectué à l'aide du mot `this` suivi de parenthèses contenant éventuellement des paramètres.

Lorsqu'une classe ne contient pas de constructeur, le compilateur en ajoute automatiquement un qui ne comporte qu'un appel au constructeur sans paramètre de la superclasse. En effet, en Java, toute classe est une classe fille d'une superclasse. Nous verrons cela plus tard. Ici, on se contentera de dire que s'il n'y a pas de constructeur, un constructeur par défaut (sans paramètre) est automatiquement ajouté et permet de créer des objets de notre classe.

Voici un exemple d'une classe possédant trois constructeurs :

```
1  public class Essai
2  {
3    public int milieu;
4    public int largeur;
5
6    Essai() {
7    }
8
9    Essai(int i) {
10     this.milieu = i;
11   }
12
13   Essai(int a, int b)
14   {
15     this(a); // appel au constructeur Essai(int) et effectuera :
      milieu = a
16     this.largeur=b;
17   }
18
19
20   public static void main (String[] args)
21   {
22     Essai e1 = new Essai();
23     Essai e2 = new Essai(2);
24     Essai e3 = new Essai(2,4);
25
26     System.out.println(e1.milieu + " , " + e1.largeur);
27     System.out.println(e2.milieu + " , " + e2.largeur);
28     System.out.println(e3.milieu + " , " + e3.largeur);
29   }
30 }
```

Exécutez le code ci-dessus. Que remarquez vous quand les attributs `milieu` et `largeur` ne sont pas explicitement initialisés (ligne 22) ?

Essayez de commenter la ligne 6 et 7 concernant le constructeur sans paramètre, puis recompilez votre code. Que remarquez vous ?

Notez l'utilisation de `this` dans le code des constructeurs. Ici, on indique que ce sont les attributs de l'instance courante (c'est à dire l'objet qui est construit) qui doivent être modifiés. Essayez de compilet et exécuter le code sans ce `this`, voyez vous une différence ? Qu'en déduisez vous ?

4.2 Attribut

On appelle ainsi une donnée définie dans une classe, une variable non locale à une méthode. Un attribut est lié au moule d'une classe. Ainsi, chaque instance d'une classe donnée possèdera tous ses attributs. On revient à la notion de "moule à gateau" pour la classe. Si c'est un moule circulaire, le gateau le sera aussi. C'est le même principe pour les classes et leurs instances. Si une classe a un attribut `a`, toute instance de cette classe possèdera un tel attribut propre à chaque instance de cette classe.

Par exemple, dans le code ci-dessous, `a, b, c` sont trois attributs de la classe `Attribut` :

```
1    public class Attribut {
2      int a;
3      int b;
4      int c;
5    }
```

Les attributs sont initialisés par défaut lors de la construction de l'objet par le **constructeur**. Un attribut est dit statique si le modificateur `static` est appliqué lors de sa déclaration. On dit aussi qu'il s'agit d'un champ de classe. Les deux qualificatifs sont équivalents. Un attribut déclaré `static` existe dès que sa classe est chargée en mémoire. Quel que soit le nombre d'instanciations de la classe (0, 1 ou plus), un attribut de classe, i.e. statique, existe en un et un seul exemplaire. Un tel attribut joue un peu le rôle d'une variable globale d'un programme non-objet ; sa classe lui sert de "boîte de rangement". En MOOB, nous n'utiliserons a priori jamais d'attribut statique.

Souvenez vous que les méthodes peuvent aussi être statiques (souvenez vous du `main`). Ce sont dans ce cas des méthodes de classe. Ces méthodes peuvent être appelées sans avoir besoin de créer une instance de classe, mais elles ne peuvent manipuler que des attributs statiques. Pourquoi ?

Ci-dessous un exemple d'appel d'une méthode de classe :

```
1   public class Test {
2
3     public static void test()
4     {
5       System.out.println("Je suis une methode statique!");
6     }
7     public static void main (String [] args ) {
8       Test.test();
9       test();
10    }
11  }
```

Pourquoi les deux appels ligne 8 et 9 sont équivalents ? Dans le cours de MOOB, **nous ne devrions pas utiliser de méthode de classe hormis la méthode exécutable** `main`.

4.3 Variables et types de base

Une variable est comme un attribut mais dont la portée est locale à une méthode. Comme en C, c'est en quelque sorte une boîte dans laquelle on peut mettre une valeur. Un peu à l'image des variables en mathématiques, on peut facilement accéder à cette valeur et l'utiliser pour faire des calculs. Il y a toutefois une différence fondamentale entre une variable mathématique et une variable informatique. La première se nomme variable car il s'agit d'un symbole capable de représenter une valeur constante inconnue parmi de nombreuses valeurs constantes (d'ailleurs on parle aussi d'*inconnue* plutôt que de variable). En informatique (plus exactement dans les langages de programmation impérative comme le C), on nomme cet objet variable car sa valeur varie au cours du temps.

On peut créer une variable et l'utiliser très simplement en Java :

```
1   public class Variable {
2     public static void main (String [] args ) {
3       int a = 2;
4       int b = 3;
5       int c = a+b;
6     }
7   }
```

Nous venons de créer 3 variables de type entier (`int`), les variables `a`, `b` et `c` ayant pour valeurs respectives 2, 3 et $2 + 3 = 5$.

Le ligne `int x;` permet de *déclarer* la variable `x`, c'est-à-dire indiquer au programme qu'on souhaite utiliser une variable portant le nom `x` étant de type `int`. Puisqu'elle est de type `int`, elle ne pourra avoir que des valeurs entières (positives ou négatives) et non des valeurs telles que `3.14`.

La seconde ligne `x = 2;` est une *affectation* de la variable, c'est à dire qu'on lui affecte la valeur 2. Tant qu'on ne lui affecte pas une autre valeur, elle aura pour valeur 2.

La troisième ligne `int y = 3;` fusionne les deux concepts précédents. Il peut être conventionnel de déclarer toutes ses variables avant de commencer les calculs et donc d'éviter les lignes de ce

genre. Cela permet de mieux organiser son programme et de le rendre plus lisible. Certains langages, comme PASCAL, forcent cette convention.

Le programme précédent n'affiche rien. Si vous souhaitez afficher une variable, il faut utiliser un peu plus en profondeur la méthode `System.out.println`.

```
1    public class Variable {
2      public static void main ( String [] args ) {
3        int a = 2;
4        int b = 3;
5        int c = a+b;
6        System.out.println("L'addition de " + a +" et " + b + " est
     : " + c);
7      }
8    }
9  }
```

Ce programme affiche :

```
>L'addition de 2 et 3 est : 5
```

`println` est donc une méthode qui peut prendre autant de paramètres qu'on le souhaite. L'opérateur `+` permet la concaténation de chaînes de caractère. Ainsi, les valeurs des variables `a`, `b` et `c` sont automatiquement converties d'un entier vers une chaîne de caractères (conversion autorisée !). Ainsi, la méthode `println` génère une chaîne de caractère résultant de la concaténation de chaque sous-chaîne passée en paramètre, et affiche cette concaténation sur la sortie standard de la machine virtuelle.

Il existe d'autres types de variables comme présenté dans les transparents, vous pouvez vous y référer pour vous entraîner.

Nous avons dit ci-dessus que toute variable souhaitant être affichée sera convertie en `String` (chaîne de caractères). Est-ce toujours possible pour les types primitifs (types de base) ? Pourquoi ?

4.4 Opérations arithmétiques

Pour faire des calculs avec les variables, nous avons vu qu'il est possible d'additionner des variables. On peut évidemment faire plus d'opérations. Ces opérateurs sont globalement les mêmes qu'en C.

On suppose ici que `x` et `y` sont de type `int`.

— `x + y` renvoie un `int` de valeur égale à la somme des valeurs de x et y

— `x * y` renvoie un `int` de valeur égale au produit des valeurs de x et y

— `x - y` renvoie un `int` de valeur égale à la différence des valeurs de x et y

— `x / y` renvoie un `int` de valeur égale au quotient de la division euclidienne de x par y. Autrement dit $\frac{x}{y}$ arrondi à l'entier inférieur (sa partie entière).

— `x % y` renvoie un `int` de valeur égale au reste de la division euclidienne de x par y. Autrement dit $x - (\frac{x}{y}) * x$. **En particulier**, si la valeur de `x` est divisible par celle de `y`, alors `x % y` a pour valeur 0.

Si `x` ou `y` est de type `double`.

— `x + y` renvoie un `double` de valeur égale à la somme des valeurs de x et y

— `x * y` renvoie un `double` de valeur égale au produit des valeurs de x et y

— `x - y` renvoie un `double` de valeur égale à la différence des valeurs de x et y

— `x / y` renvoie un `double` de valeur égale à la division des valeurs de x et y.

Ci-dessous un exemple d'opération de division avec des doubles.

```java
public class Variable {
  public static void main ( String [] args ) {
    double a = 2;
    double b = 3;
    System.out.println(a/b);
    double c = a/b;
    System.out.println(c);
  }
}
```

4.5 Exercices

Exercice 3 — *Somme et produit*

Écrire un programme `SommeProduit.java` qui déclare et défini trois entiers x, y et z de votre choix et affiche leur somme et leur produit sur la même ligne.

Exercice 4 — *Les `int` ne sont pas des `double`.*

1. Que se passe-t-il si vous additionnez une variable `x` de type `int` et une variable `b` de type `double` ?

```java
int a = 2;
double b = 3;
System.out.println(a+b);
```

2. Que se passe-t-il si vous divisez une variable `x` de type `int` et une variable `b` de type `int` ?

```java
int a = 2;
int b = 3;
System.out.println(a/b);
```

   Que remarquez vous avec le typage des opérations ? Si les types des variables concernées par l'opération est différent, alors une conversion de la variable du type le plus faible est effectuée vers celle du type le plus fort ! Relisez le transparent de cours pour reprendre ce point.

3. Créer un programme `Inverse.java` qui déclare, défini et affiche un `int a` > 1 de votre choix et le flottant (donc de type `double`) ayant pour valeur $\texttt{a}^{-1}$.

Exercice 5 — *Milieu*

Écrire un programme `Milieu.java` qui déclare et défini quatre flottants x_1, x_2, y_1 et y_2 de votre choix et affiche sous la forme suivante les coordonnées du point (x_m, y_m) situé au milieu des points (x_1, y_1) et (x_2, y_2).

```
>Point p1 : (x1, y1)
>Point p2 : (x2, y2)
>Milieu de p1 et p2 : (xm, ym)
```

Exercice 6 — *Sphère*

Écrire un programme `Sphere.java` qui déclare et défini un entier `r` et affiche le volume d'une sphère de rayon r égal à $\frac{4}{3} \cdot \pi r^3$. Vous utiliserez la macro `Math.PI` de la bibliothèque Math de Java pour la valeur de π

5 Conditions

5.1 Tutoriel

Dans la plupart des langages de programmation, il est possible d'exécuter une instruction si une condition est vérifiée et d'en exécuter une autre sinon.

C'est le mot-clef `if` qui permet cela. Par exemple le code suivant permet d'afficher `x est PAIR` si la variable entière `x` est paire et `x est IMPAIR` sinon. On rappelle que `x` est divisible par 2 si `x % 2` vaut 0.

```
1  if(x % 2 == 0){
2   System.out.println(x + " est PAIR");
3  }
4  else{
5   System.out.println(x + " est IMPAIR");
6  }
```

Tout ce qui est entre les premiers symboles { } est exécuté si `x` est pair et tout ce qui est entre les symboles { } après le `else` est exécuté sinon.

`else` n'est pas obligatoire. Il n'est pas nécessaire d'avoir un bloc `else`. Si la condition est fausse, alors il ne se passe rien. Par exemple si `x` est impair ici, il ne se passe rien.

Enchaîner les if. S'il y a plus que 2 possibilités vous pouvez enchaîner les conditions.

```
1   if(x % 3 == 0){
2     System.out.println(x + " est DIVISIBLE PAR 3");
3   }
4   else if(x % 3 == 1){
5     System.out.println(x + " est DIVISIBLE PAR 3");
6   }
7   else{
8     System.out.println(x + " 2 est DIVISIBLE PAR 3", x);
9   }
```

Les accolades { } ne sont pas (toujours) obligatoires. Lorsqu'il n'y a qu'une ligne entre les symboles { }, on peut les enlever. Par exemple le programme suivant fait la même chose que le premier :

```
1  if(x % 2 == 0)
2   System.out.println(x + " est PAIR");
3  else
4   System.out.println(x + " est IMPAIR");
5
```

Attention, c'est une source classique d'erreurs. Il est recommandé de toujours mettre les symboles { } pour éviter tout problème bête.

Et ce == alors ? Il existe en C ce qu'on appelle des opérateurs de comparaison qui permettent de comparer deux variables et de renvoyer une valeur (valant vrai ou faux). Ces opérateurs sont les suivants et fonctionnent aussi bien sur des `int` que sur des `double`.

- `x == y` renvoie vrai si `x` et `y` ont la même valeur.
- `x < y` renvoie vrai si `x` est strictement plus petit que `y`.
- `x > y` renvoie vrai si `x` est strictement plus grand que `y`.
- `x <= y` renvoie vrai si `x` est plus petit ou égal à `y`.
- `x >= y` renvoie vrai si `x` est plus grand ou égal à `y`.

Et et Ou. À ces opérateurs s'ajoute la possibilité de les associer. Par exemple

— `x == y && x == z` renvoie vrai si `x` et `y` ont la même valeur **et** si `x` et `z` ont la même valeur.

— `x == y || x == z` renvoie vrai si `x` et `y` ont la même valeur **ou** si `x` et `z` ont la même valeur.

5.2 Exercices

Exercice 7 — *Positivité*

Créer un programme `Positif.java` qui affiche un entier `x` de votre choix puis affiche `POSITIF` si `x` est positif ou nul et `NEGATIF` sinon.

Exercice 8 — *Comparaison de `double`*

Attention à la comparaison de `double`,

1. Créez deux `double` `x` et `y` avec le programme suivant

```
1       double x, y;
2       x = 340282346638528859811704183484516925440.0 - 1;
3       y = 340282346638528859811704183484516925440.0 - 10;
```

   Complétez le programme pour vérifier si `x` et `y` sont égaux. Que constatez vous ?

2. Créez deux `double` `x` et `y` avec le programme suivant

```
1       double x, y;
2       x = 1/34028234.0;
3       y = ((1/34028234.0) + 3) - 3;
```

   Complétez le programme pour vérifier si `x` et `y` sont égaux. Que constatez vous ?

Autant on ne peut éviter le premier problème, autant on peut éviter le second : au lieu de comparer `x` et `y` avec `x == y`, on peut vérifier si `|x - y|` n'est pas plus petit que 0.001.

3. Écrire un programme `CompareDoubles.java` qui déclare et défini deux doubles `x` et `y` et vérifie si `|x - y|` est plut petit que 0.001.
4. Vous avez peut-être utilisé deux `if` dans le programme précédent. Si c'est le cas, refaite le avec un seul `if` et en utilisant les opérateurs `&&` et `||`.

6 Boucles

6.1 Tutoriel

Il existe en Java, et dans la plupart des langages de programmation, deux manières de répéter un certain nombre de fois les mêmes opérations : les boucles `for` et `while`.

6.1.1 Boucle `for`

Un exemple classique de programme avec une boucle `for` est le suivant :

```
1
2   for(int i = 0; i < 5; i = i + 1){
3     System.out.println("2 x " + i + " = "+2*i);
4   }
```

Ce programme affiche la sortie suivante :

```
>Table de 2
>2 x 0 = 0
>2 x* 1 = 2
>2 x 2 = 4
>2 x 3 = 6
>2 x 4 = 8
```

On peut lire la ligne 1 du programme comme : *Pour tout i allant de 0 à 4, exécuter les instructions suivantes ...*

Remarquez la présence des accolades pour délimiter les instructions de la boucle. Vous pouvez mettre plusieurs instructions entre ces accolades sous une boucle `for`. Comme pour le mot-clef `if`, les accolades sont facultatives s'il n'y a qu'une instruction à l'intérieur. Mais il est recommandé de toujours les mettre.

Vous pouvez constater la présence du type `int` dans la boucle. C'est parce qu'on déclare la variable `i` au sein de la boucle. Il est possible de déclarer la variable avant la boucle. Cela est nécessaire si vous souhaitez déclarer toutes vos variables avant le début de votre programme.

```
1   int i;
2   for(i = 0; i < 5; i = i + 1){
3     System.out.println("2 x " + i + " = "+2*i);
4   }
```

Les valeurs chiffrées ne sont pas nécessairement des constantes, vous pouvez les définir comme des variables :

```
1   int i, n, m;
2   n = 0;
3   m = 5;
4   for(i = n; i < m; i = i + 1){
5     System.out.println("2 x " + i + " = "+2*i);
6   }
```

Cela a bien entendu peu d'intérêt dans cet exemple, mais en aura dans les sections ultérieures.

Dernier détail, la variable `i` est ici incrémentée de 1 à chaque itération grâce à la partie `i = i + 1`. Il est possible d'augmenter `i` de plus que 1 ou de le diminuer en changeant ce morceau :

```
1   for(i = 0; i < 12; i = i + 3){
```

```
1   for(i = 12; i >= 0; i = i - 1){
```

Généralement, `i = i + 1` est écrit de manière plus compacte sous la forme `i++` ou `i += 1`. Choisissez la forme qui vous convient le mieux.

6.1.2 Boucle `while`

La boucle `while` permet de répéter des instructions comme le fait la boucle `for`.

Combien de fois pouvez-vous diviser un entier positif par 2 (arrondi à l'inférieur) avant d'atteindre 1 ? C'est la définition du log en base 2.

```
1  int n, m, log;
2
3  log = 0;
4
5  n = 28;
6  m = n;
7  while(n > 1){
8    n = n / 2;
9    log = log + 1;
10 }
11 System.out.println("Le log en base 2 de " + m + " est " + log);
```

Ce programme affiche

```
>Le log en base 2 de 28 est 4.
```

Que fait ce programme ?

La ligne 7 indique *Tant que* `n` *est strictement plus grand que 1, faire ...* Tant que c'est vrai, on va diviser `n` par 2 et augmenter `log` de 1. La variable `log` compte donc le nombre de fois où on a divisé `n` par 2. Et on s'arrête quand on atteint 1. Ainsi `log` suit bien la définition indiquée plus haut.

Comme pour la boucle `for`, s'il n'y a qu'une seule instruction sous la boucle `while`, on peut enlever les accolades, mais il est conseillé de les garder.

Vous pouvez mettre dans une boucle `while` n'importe quelle condition que vous auriez mis avec le mot-clef `if`. Vous pouvez donc vous servir de tous les opérateurs de comparaisons et des opérateurs `&&` et `||`.

6.1.3 Différence entre une boucle `while` et une boucle `for`

Techniquement, en Java et en C, il n'y a pas de différence. Tout ce que vous pouvez faire avec une boucle `for`, vous pouvez le faire avec une boucle `while` et inversement.

Cependant, ces deux boucles existent dans pratiquement tous les langages de programmation, quand bien même elles sont redondantes. La raison est une question de lisibilité, on utilise chacune des boucles dans des situations différentes et il est conventionnel de ne pas inverser ces usages.

— Une boucle `for` est une boucle **bornée** : on connait exactement le nombre d'itérations d'une telle boucle et ce nombre ne doit pas être infini.

— Une boucle `while` est une boucle **non bornée** : on ne connait pas aisément le nombre d'itérations d'une telle boucle et ce nombre peut être infini.

Dans les exemples plus haut, on peut déduire de la ligne `for(i = 0; i < 5; i = i + 1){` qu'il y aura 5 itérations dans la boucle `for`. À l'inverse, la ligne `while(n > 1){` ne nous permet pas de savoir combien de fois on va diviser par 2 avant de s'arrêter.

Il vous est demandé dans ce cours de respecter ce principe.

Attention pour les boucles `while` : il faut bien gérer d'incrémenter/décrémenter/changer la variable de la boucle ! ! (`n` dans le calcul du log ci-dessus). Car sinon, vous aurez possiblement une boucle infinie voir un comportement vraiment incongru de vos programmes !

6.2 Exercices

N'utilisez pas une boucle `while` si la boucle `for` est plus adaptée.

Exercice 9 — *Table de m*

Créer un programme `TableMultiplication.java` qui affiche un entier `m` de votre choix et affiche la table de multiplication de cet entier.

Exercice 10 — log *en base m*

Quelle est la définition du log en base m d'un entier n ? Créer un programme `Logm.java` qui affiche deux entiers `n` et `m` de votre choix et affiche le log en base `m` de `n`.

Exercice 11 — *Somme des entiers de 1 à n*

Créer un programme `Somme.java` qui affiche un entier `n` > 10 de votre choix et la somme des entiers de 1 à `n`.

Exercice 12 — *Puissances de 3*

Créer un programme qui affiche l'entier `i` tel que $\texttt{i}^3 \leq 1548 < (\texttt{i}+1)^3$.

Exercice 13 — *int*

On a vu plus dans les transparents que le type `int` représentait les entiers entre -2147483648 et 2147483647, que se passe-t-il si vous compilez et exécutez le programme suivant ? Est-ce différent du C ?

```
1     System.out.println(2147483647 + 1);
```

Créer un programme `MaxPow2.java` qui, à l'aide d'une boucle, affiche l'entier 2^i tel que $2^i \leq MI < 2^{i+1}$ où MI est la plus grande valeur qu'une variable de type `int` puisse avoir. N'utilisez pas la méthode `Math.pow`.

Exercice 14 — *Rectangle*

Créer un programme `Carre.java` qui connaissant un entier `n` de votre choix, affiche en console des caractères `*` organisés sous forme d'un carré de largeur et longueur n. Par exemple pour $n = 4$:

```
> ****
> *  *
> *  *
> ****
```

7 Fonctions

7.1 Tutoriel (A LIRE)

7.1.1 Bases

Toute personne qui souhaite coder professionnellement et partager son code cherche avant tout à éviter la répétition de code. Le code suivant par exemple semble un peu rébarbatif :

```
int n, s;

n = 3;
s = 0;
for(int i = 1; i <= n; i++){
  s += i * i;
}
System.out.println("Somme des carres de 1 a 3 : " +  s);

n = 10;
s = 0;
for(int i = 1; i <= n; i++){
  s += i * i;
}
System.out.println("Somme des carres de 1 a 10 : " + s);

n = 5;
s = 0;
for(int i = 1; i <= n; i++){
  s += i * i;
}
System.out.println("Somme des carres de 1 a 5 : " + s);
```

Il existe en programmation la notion de *fonction*, qui a plus ou moins le même objectif que les fonctions en mathématiques : définir un objet qui prend en entrée un ou plusieurs paramètres, qui effectue un ensemble d'instructions ou de calculs et qui renvoie une ou plusieurs valeurs en sortie.

Par exemple en mathématiques, la fonction log prend en entrée un réel et renvoie un réel. Attention, comme pour les variables, les fonctions en mathématiques et en informatiques sont différentes. Mais on y reviendra une autre fois (voire dans un autre cours). Pour être exact, on devrait plus souvent parler en informatique de *Procédure* plutôt que de *Fonction*, car la procédure fait mention d'un effet mécanique de calcul par une machine alors que la fonction fait plus penser à un objet abstrait.

Pour définir une fonction il faut lui donner un nom, définir ce qui est donné en entrée et ce qui est donné en sortie, puis décrire comment calculer la sortie à partir de l'entrée. C'est en fait la même chose qu'en C. Sauf qu'ici, nos fonctions seront encapsulées dans une classe, et viendront s'ajouter à la méthode `main` représentant le programme. Ainsi encapsulées, ces fonctions s'appellent les **méthodes** d'une classe. Par exemple, ci-dessus, on pourrait définir la classe `SommeCarres`, y définir une méthode `public class SommeCarres` et y faire appel 3 fois, ce qui évite de recopier 3 fois le même code.

```java
public class SommeCarres{

  public int sommeCarres(int n){
    int s = 0;
    for(int i = 1; i <= n; i++){
      s += i * i;
    }
    return s;
  }

  public static void main( String [] args){
    System.out.println("Somme Carres 1-3 : " + sommeCarres(3));
    System.out.println("Somme Carres 1-10 : " + sommeCarres(10));
    System.out.println("Somme Carres 1-5 : " + sommeCarres(5));
  }
}
```

Si on décortique la méthode `public int sommeCarres(int n){` :

— `public` est un mot clé signifiant que la méthode peut être utilisée par toute classe, et non seulement par la classe `SommeCarres`. Nous verrons plus en détails dans le module MOOB la signification de ce mot clé. Sachez simplement qu'il existe différentes visibilités pour les éléments de vos programmes, où `public` signifie que l'élèment est accessible par tous les autres de votre programme (classe, fonction etc)

— `sommeCarres` est le nom de la méthode.

— Cette méthode prend en entrée un entier, qui sera, dans la méthode, une variable nommée `n` de type `int`.

— Elle renvoie en sortie un entier. Cette entier est calculé à l'aide de la variable `s`, puis renvoyé avec `return s;`.

Remarquez la présence des accolades. Contrairement aux accolades des conditions et des boucles, celles des fonctions sont obligatoires.

7.1.2 Méthodes de classe et instances d'une classe

Essayez de compiler le code de la classe `SommeCarres`. Que se passe-t-il ?
Vous devriez obtenir le message suivant :

```
>javac SommeCarres.java ; java SommeCarres
>SommeCarres.java:11: error: non-static method sommeCarres(int) cannot be referenced from a
     static context
>    System.out.println("Somme des carres de 1 a 3 : " +  sommeCarres(3));
>                                                        ^
>SommeCarres.java:12: error: non-static method sommeCarres(int) cannot be referenced from a
     static context
>    System.out.println("Somme des carres de 1 a 3 : " +  sommeCarres(10));
>                                                        ^
>SommeCarres.java:13: error: non-static method sommeCarres(int) cannot be referenced from a
     static context
>    System.out.println("Somme des carres de 1 a 3 : " +  sommeCarres(5));
>                                                        ^
>8 errors
```

Que se passe-t-il ici ? Tout cela est dû au modèle Java et la programmation objet. La méthode `main` est `static`, donc de classe comme décrit précédemment. Une méthode, pour être de classe, ne doit pas manipuler, directement ou indirectement, d'attributs non statiques de sa classe. En conséquence, si une méthode de classe utilise dans son code un attribut ou une méthode non statique de sa classe, une erreur est détectée à la compilation. De l'extérieur de sa classe, un

attribut ou une méthode de classe pourront être utilisés précédés du nom de sa classe : `nom_classe.nom_attribut_ou_méthode_de_classe`

On trouve un certain nombre de méthodes statiques dans l'API. C'est le cas de toutes les méthodes de la classe `java.lang.Math` : on pourra ainsi utiliser les méthodes `Math.sin` , `Math.log`, de même que la constante statique `Math.PI` utilisée précédemment .

Pour ne pas avoir ce problème, la méthode doit en fait être appelée par une instance de la classe la contenant. C'est le but de la programmation objet : déléguer aux objets leurs fonctionnalités ! On "crée" un objet, qui va vivre par lui-même et se charger de faire les appels aux méthodes internes (ie, non statiques) de sa classe explicitement. Ainsi, une version fonctionnelle du code précédent pourrait ainsi être :

```
1  public class SommeCarres{
2
3    public int sommeCarres(int n){
4      int s = 0;
5      for(int i = 1; i <= n; i++){
6        s += i * i;
7      }
8      return s;
9    }
10
11   public static void main( String [] args){
12     SommeCarres s = new SommeCarres(); // constructeur (cf section
       4.1)
13     System.out.println("Somme Carres 1-3: " + s.sommeCarres(3));
14     System.out.println("Somme Carres 1-10: " + s.sommeCarres(10));
15     System.out.println("Somme Carres 1-5: " + s.sommeCarres(5));
16   }
17 }
```

avec l'utilisation d'une instance de la classe `SommeCarres` dénomée `s` (`s` est un objet de type `SommeCarres`), et un appel à la méthode `sommeCarres` de cette instance.

A ce point : Les instances d'une classe (= les objets de cette classe) sont comme les gateaux d'un moule. Ces objets ont toutes la même propriété (ie, la forme du moule) mais sont tous différents. On peut construire un objet en appelant un constructeur. Ici, vous remarquerez que ce constructeur n'a jamais été défini dans la classe. En effet, il existe **toujours** un constructeur par défaut, implicite !
On expliquera tout cela en détail pendant le cours de MOOB.

Comment rendre le code initial valide sans utiliser le constructeur `SommeCarres s = new SommeCarres();` pour appeler `sommeCarres` ?

Indice : utiliser le mot clé `static` aussi pour la méthode `SommeCarres`. Ainsi, comme la méthode `main`, cette méthode sera une méthode de classe et pourra donc être directement appelée par une autre méthode de classe.
Testez cette version et assurez vous qu'elle fonctionne.

Tout cela peut sembler un peu compliqué, mais il n'en est rien ! On verra cela dans les futures séances de MOOB !

7.1.3 Paramètres d'une fonction

Une méthode peut prendre plusieurs paramètres, et les types des paramètres et de la sortie peuvent être différents. On peut mettre plusieurs `return` dans la méthode. Dès qu'on rencontre un `return`, la méthode s'arrête et renvoie la valeur indiquée.

```java
public class Sommes1SurI {
   public int sommes1SurI(int n, double d){
    double s = 0.0;
    if(n <= 0)
      return -1;
    for(int i = 1; i <= n; i++)
      s += 1/(double)i;
    if(s < d)
      return 1;
    else
      return 0;
  }


  public static void main( String [] args){
    Sommes1SurI s = new Sommes1SurI();
    System.out.println(30 + " " + 23.45 + " " + s.sommes1SurI(30, 23.45));
    System.out.println(-2 + " " +1000.0 + " " + s.sommes1SurI(-2, 1000.0));
    System.out.println(16 + " " + 2 +" " + s.sommes1SurI(16, 2));
    }
}
```

Ceci renvoie :

```
>30 23.450000 1
>-2 1000.000000 -1
>16 2.000000 0
```

On pourrait imaginer enfin de mettre l'appel de `System.out.println` dans une méthode, pour éviter de recopier cet appel 3 fois :

```java
public class Sommes1SurI {
  public int sommes1SurI(int n, double d){
    double s = 0.0;
    if(n <= 0)
      return -1;
    for(int i = 1; i <= n; i++)
      s += 1/(double)i;
    if(s < d)
      return 1;
    else
      return 0;
  }

  public void compareSommes1SurI(int n, double d){
    System.out.println( n + " " + d + " " + sommes1SurI(n, d));
  }

  public static void main ( String [] args){
    Sommes1SurI s = new Sommes1SurI();
    s.compareSommes1SurI(30, 23.45);
    s.compareSommes1SurI(-2, 1000.0);
    s.compareSommes1SurI(16, 2);
  }
}
```

Remarquez que le type `void` renvoyé par la méthode `compareSommes1SurI`. Ce type signifie que la méthode ne renvoie rien. Elle se contente de faire un calcul et de l'afficher ou de modifier un état.

Il est aussi possible pour une méthode de ne rien prendre en entrée.

Il n'est pas possible pour une fonction en C ni en Java de renvoyer plusieurs paramètres. Evidemment, il est possible en Java de renvoyer l'instance d'une classe, et donc encapsuler ainsi plusieurs paramètres en un seul.

7.2 Exercices

Exercice 15 — *Gravité*

Dans un fichier `Gravite.java`, créer une méthode qui, connaissant trois `int` $m1$, $m2$ et d, renvoie un `double` égal à la valeur de la force de gravité entre deux corps de masses $m1$ et $m2$ éloignés d'une distance d égale à $G \cdot m1 \cdot m2/d^2$, avec $G = 6.67 \cdot 10^{-11}$. Testez la dans un `main`.

Exercice 16 — *Cylindre*

Dans un fichier `Cylindre.java`, créer 3 méthodes. Une méthode qui, connaissant un `double r`, calcule le périmètre d'un cercle de rayon `r`. Une méthode qui, connaissant un `double r`, calcule l'aire d'un cercle de rayon `r`. Et une méthode qui, connaissant deux **double** `r` et `h` **affiche** l'aire et le volume d'un cylindre de rayon `r` et de hauteur `h`. L'aire vaut $2\mathcal{A}(r) + \mathcal{P}(r) \cdot h$ et le volume vaut $\mathcal{A}(r) \cdot h$ où $\mathcal{A}$ et $\mathcal{P}$ calculent respectivement l'aire et le périmètre d'un cercle. On prendra $\pi =$ `Math.PI`. Testez les dans un `main`.

Exercice 17 — *Retour arrière*

Créez un fichier `Boucles.java` avec 2 méthodes. Une méthode qui, connaissant un entier `n`, affiche la table de multiplication de `n` ; et une méthode qui connaissant deux entiers `n` et `d` renvoie le logarithme en base `d` de `n`. Testez les dans un `main`.

Exercice 18 — *Formules usuelles de mathématiques*

Il existe de nombreuses fonctions mathématiques déjà codées dans le module `Math` : `https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/lang/Math.html`.

Dans un fichier `Nlogn.java`, créer une méthode qui, connaissant un entier `n`, renvoie $\texttt{n}\ln(\texttt{n})$. Testez la dans un `main`.

Exercice 19 — *Triangle*

Créer un fichier `Triangle.java` contenant une fonction qui, connaissant un entier n, affiche en console des caractères `*` organisés sous forme d'un triangle isocèle de hauteur n, de base de taille $2n - 1$ et pointant vers le bas. Par exemple pour $n = 4$, le programme dervait afficher

```
> *******
>  *   *
>   * *
>    *
```

Exercice 20 — *Norme 1*

Créer un fichier `Norme1.java` contenant une méthode qui, connaissant un entier n, affiche en console des caractères `*` organisés sous forme d'un losange de rayon n (affiche toutes les `*` dont la distance en norme 1 au centre est inférieure ou égale à n). Par exemple pour $n = 3$, le programme dervait afficher

```
>     *
>    ***
>   *****
>  *******
>   *****
>    ***
>     *
```

8 Caractère et chaîne de caractères

8.1 Tutoriel

Il existe le type `char`, c'est-à-dire un symbole de texte, lettre ou chiffre ou ponctuation ou autre. Un caractère s'écrit entre guillemets **simples**. Il existe en Java 256 caractères manipulables.

```
1     char c = 'a';
2
```

Concernant les chaînes de caractères, il existe en Java un classe native pour cela : `String`. Cette classe vous offre de très nombreuses fonctionnalités pour manipuler les chaînes de caractères, référez vous à la documentation pour plus de détails : `https://docs.oracle.com/javase/8/docs/api/java/lang/String.html`.

Essentiellement, une chaîne de caractères est un tableau de caractères. L'exemple ci-dessous vous donne une idée des opérations que l'on peut faire dessus (extrait de la doc) :

```
1   String str = "abc";
2
3   // equivalent a la ligne du dessus
4   char data[] = {'a', 'b', 'c'};
5   String str = new String(data); // appel au constructeur de la
     classe String
6
7   System.out.println("abc");
8   String cde = "cde";
9   System.out.println("abc" + cde);
10  String c = "abc".substring(2,3);
11  String d = cde.substring(1, 2);
12
```

Notez qu'en Java, on ne manipule pas de pointeurs vers la mémoire. La gestion de la mémoire est laissée à la machine virtuelle. La création d'objet est gérée en interne, et quand un objet n'est plus référencé (le programme ne l'utilise plus), la mémoire est libérée par ce que l'on appelle le `garbage collector` (ou ramasse-miettes). Vous ne manipulez que des références vers les objets (= classes) que vous utilisez dans vos programmes. Nous verrons cela plus en détails dans le cours de MOOB.

8.2 Exercices

Exercice 21 — *Duplicat*

Créer un programme `Duplicat.java` qui contient une fonction prenant en entrée trois chaînes de caractères `c`, `s` et `t` et modifie `c` pour qu'il contienne `s` puis `t`. Créer ensuite une seconde fonction, utilisant la première, prenant en entrée deux chaînes `c` et `s` et un entier `n` et modifie `c` pour qu'elle contienne `n` fois de suite la chaîne `s`.

Exercice 22 — *affichage*

Créer un programme `Print2.java` qui contient une fonction prenant en entrée une chaîne de caractères `s` et qui, à l'aide d'une boucle `for` et de la méthode `length()` de la classe `String`, affiche chaque caractère de `s` dans l'ordre sur des lignes distinctes. Faire une seconde fonction qui a le même comportement, mais qui utilise une boucle `while`.

Exercice 23 — *palindrome*

Créer un programme `Palindrome.java` qui contient deux fonctions. La première reçoit une chaîne de caractères et vérifie si cette chaîne est un palindrome. Si c'est le cas elle renvoie `True` et sinon elle renvoie `False`. La seconde reçoit une chaîne de caractères `p`. La fonction renvoie une nouvelle chaîne de caractères `s` qui contient un palindrome en mettant à la suite `p` et la chaîne `p` renversée.

Exercice 24 — *Chaîne entière*

Créer un programme `Isnumber.java` qui contient une fonction qui, connaissant une chaîne de caractère, vérifie si cette chaîne est un entier (donc constitué uniquement de numéros). Vous pouvez utiliser la fonction `Character.isDigit(char c)` qui renvoie `True` si `c` est un chiffre et `False` sinon.