EXERCICES JAVA
INTRODUCTION

Préambule

Cet exercice doit être fait SANS utiliser d'IDE (IntellIJ/NetBeans ou Eclipse), l'objectif étant de tester les commandes rattachées à l'environnement Java.

a) Construire à l'aide d'un éditeur comme SublimeText, Atom ou n'importe quel éditeur un programme Example qui affiche la liste numérotée des arguments de la ligne de commande. Définir cette classe dans le package labs.one.parta.

b) Utiliser la ligne de commande et compiler avec : javac labs.one.parta.Example.java (aller regarder l'arborescence construite)

c) Toujours sur la ligne de commande, exécuter le programme avec :

java labs.one.Example value1 "value2 with spaces" value3

Remarque : en cas de nécessité, il est possible de rajouter aussi bien à javac qu'à java un paramètre -cp "chemins" pour définir les répertoires de recherche de package et de classes.

Exercice 1 : lectures de programmes étranges

Pour se (re)mettre Java en tête, voici quelques problèmes de lecture de code.

1°) Indiquer ce qu'affichent les programmes suivants ?

int output = 10; *boolean result = false;* *if((result) && ((output += 10) == 20))* *System.out.println("True " + output);* *else System.out.println("False " + output);*	
int counter = 0; *for (int i = 1; i < 010; i++) {* *counter = counter + i;* *}* *System.out.println("Result : " + counter);*	
public void check(boolean isOk) { *if (isOk = true) System.out.println("OK");* *else System.out.println("KO");*	
Boolean ignore = null; *if (ignore == false) {* *System.out.println("Do not ignore!");* *}*	
static int mystery(int x, int y) { // x et y positifs *if (y == 0) return 0;* *else if (y % 2==0) return 2 * mystery(x, y / 2);* *else return x + (2 * mystery(x, (y - 1) / 2));* *}* Avec dans main *: System.out.println(mystery(3,4));*	

2°) Qu'affiche le code suivant ?

```
public class Test {
   String string = "i";
   void f() {
      try {
         string += "a";
         g();
      } catch (Exception e) {
         string += "e";
      }
   }
   void g() throws Exception {
      try {
         string += "b";
         h();
      } catch (Exception e) {
         throw new Exception();
      } finally {
         string += "d";
      }
      string += "3";
   }
   void h() throws Exception {
      throw new Exception();
   }
   void display() {
      System.out.println(string);
   }
   public static void main(String[] args) {
      Test test = new Test();
      test.f();
      test.display();
   }
}
```

3°) Soit la classe suivante :

```
public class Test {
   public static void f(String s) {
      s += "rajout";
   }
   public static String g(String s) {
      s += " texte a concatener en string";
      return s;
   }
   public static void g(StringBuffer sb) {
      sb.append(" texte a concatener en sbuffer");
   }
   public static StringBuffer h(StringBuffer sb) {
      return (sb = new StringBuffer("simple construction"));
   }
   public static void main(String[] args) {
      String string = "un premier texte";
      f(string);
      System.out.println(string);
      string = "un deuxieme texte";
      g(string);
      System.out.println(string);
```

```
        string = g(string);
        System.out.println(string);
        StringBuffer stringBuffer = new StringBuffer("construction sbuffer");
        g(stringBuffer);
        System.out.println(stringBuffer);
        stringBuffer = new StringBuffer("construction sbuffer 2");
        h(stringBuffer);
        System.out.println(stringBuffer);
        stringBuffer = h(stringBuffer);
        System.out.println(stringBuffer);
    }
}
```

Après avoir essayé de comprendre les manipulations mémoire de cet exercice, indiquer ce que le programme affiche.

4°) Essayer de déterminer ce qu'affiche le programme suivant sans le taper; le vérifier ensuite et le corriger si nécessaire.

```
public class Test {
    static void f(int [] array, int nb) {
        array = new int[nb];
        for(int value: array)
            System.out.println(++value + " ");
    }

    public static void main(String[] args) {
        int size = 3;
        int [] array = null;
        f(array, size);
        for(int value: array)
            System.out.println(value + " ");
    }
}
```

Exercice 2

La classe BufferedReader (java.io) peut être utilisée pour lire un fichier texte, de la manière suivante :

```
reader = new BufferedReader(new FileReader("truc.xtx"));
String line = reader.readLine();
```

a) Construire un fichier texte contenant des paires de valeurs séparées par des sauts de ligne et l'enregistrer à la racine d'un projet. Supposer que ces paires de valeurs désignent les résultats obtenus sur HackerRank (comme par exemple arnd.mia@gmail.com:3482)

b) Construire une classe Java, permettant d'effectuer les travaux suivants:

- lecture des lignes de ce fichier et enregistrement dans un HashMap (tableau associatif Java, définie dans java.util).
- calcul de la moyenne, grâce à un ArrayList .
- retrouver le nombre de points d'un étudiant d'adresse mail fournie.

Utiliser le mécanisme des exceptions pour l'ouverture du fichier soit standard, soit par gestion de ressources.

c) En utilisant la nouvelle API Time de Java, calculer le temps d'exécution en millisecondes du programme précédent:

```
//Heure actuelle
Instant start = Instant.now() ;
// calcul d'une durée
Duration duration = Duration.between(start, Instant.now()) ;
```

Remarque : utilisation d'un HashMap

```
HashMap<String, String> map = new HashMap<>();
map.put("France","Paris");
map.put("UK","Londres");
map.put("Italie","Rome");
map.put("Allemagne","Berlin");
System.out.println(map.size());
System.out.println(map.get("UK"));
for (String i : map.keySet()) {
   System.out.println("Ville : " + i + " capitale : " + map.get(i));
}
```

Exercice 3

Voici quelques cas de programmation à construire pour tester les exceptions:

- construction d'une fonction statique recevant un tableau de chaînes de caractères contenant normalement des nombres entiers ("123" par exemple) : cette fonction doit ne calculer la moyenne que des nombres présents dans les chaînes du tableau (elle émet une exception si il y a une chaîne non numérique et ne prend pas celle-ci en compte)
- même question mais avec une variante : si une exception est levée, on affiche un message à l'utilisateur lui indiquant que les valeurs sont incorrectes; dans un deuxième temps, l'exception doit être propagée à la fonction main, qui doit afficher le nom de la fonction erronée; dans un troisième temps, l'exception est propagée à la machine virtuelle.
- essayer de tester une exception quelconque de nature RuntimeException

Exercice 4

De nombreux codages sont construits en informatique à l'aide d'un alphabet restreint; par exemple, le français est basé sur l'alphabet {A, ..., Z }, le Morse est basé sur l'utilisation de {.,-}, les couleurs sur un code hexadécimal compris entre 0 et 9 et A et F, l'ADN sur l'alphabet {A, C, G, T} (molécules adénine, cytosine, guanine et thymine). A partir de cet alphabet, des chaînes de longueur quelconque peuvent être construites.

Construire un programme, recevant en paramètres de la ligne de commande deux chaînes de caractères, un alphabet et une chaîne normalement formé des symboles de cet alphabet, qui :

- vérifie si cette chaîne est bien formée (i.e. formée des bons symboles, qu'elles soient écrites en majuscules, ou en minuscules); le programme affiche un message d'erreur dans le cas contraire.
- opère une rotation de n caractères vers la gauche d'une chaîne (par exemple leftRotate(3,"abcdef") renvoie "defabc),
- même fonction vers la droite,
- si le paramètre est /a, recherche si une autre chaîne également fournie sur la ligne de commande (exemple : java prog abcd bac /a cab) est un anagramme du mot fourni
- si le paramètre est /o, recherche une suite également fournie sur la ligne de commande (exemple : java prog abcd bbac /o ac) dans le mot en question, la supprime et affiche le résultat.

Essayer de construire quelques fonctions et de n'agir que sur la classe String. Aller ensuite explorer la classe StringBuffer.

Bonus : quelques fonctions supplémentaires pour manipuler les chaînes

- rajouter avec /f le nombre d'occurrences du mot cherché
- rajouter /an pour afficher tous les anagrammes d'une chaîne reçue en paramètre

Exercice 5

Le but de ce programme est de pouvoir déterminer un ensemble de mots ne différant les uns des autres que d'une seule lettre, à partir d'un tableau de mots, en sachant que le mécanisme est récursif. Un mot ne différant que d'une lettre d'une autre mot est appelé voisin. Par exemple : dey bey deb des bec bel ben boy ...

Il faudra donc trouver les voisins d'un mot, puis les voisins des voisins et ainsi de suite jusqu'au moment où il n'y aura plus de nouveaux voisins.

a) Ecrire un programme modélisant ce traitement (il est possible de commencer par trouver les voisins d'un mot, puis tous les voisins des voisins et ainsi de suite). Tester les fonctions sur la liste des mots suivante :

```
String[] words = { "aas", "ace", "alu", "are", "api", "ays", "bec",
                   "bel", "bey", "ben", "boy", "deb", "des", "dom",
                   "dot", "daw", "fax", "fan", "faq", "fob", "hem",
                   "hop", "man", "mao", "mug","mus", "mie", "sur",
                   "dey", "mur"
                 };
```

Trouver tous les mots accessibles depuis bel et aussi depuis dey.

b) Rajouter une fonction permettant de savoir s'il est possible d'aller d'un mot à un autre mot en n coups exactement.

Exercice 6 et point de cours

Il est important d'apprendre à manipuler un utilitaire de java : l'outil jar.
Cet outil sert à fabriquer des fichiers appelé JAR (Java Archive) qui servent à regrouper en une seule entité tous les fichiers classes, touts les images, ... en bref toutes les ressources nécessaires au fonctionnement d'un composant logiciel, d'une librairie ou d'une application (fichiers compilés .class, images, documentation, …).
Le rôle de ces fichiers est fondamental car ils permettent :

- de distribuer une bibliothèque java (n'importe quelle bibliothèque comme par exemple celle du DOM ou de Xerces pour XML)
- de distribuer un composant ou une brique logicielle importante (comme un driver de base de données en Java)
- de distribuer une application lourde (ce qui est beaucoup plus simple que de distribuer un dossier formé de centaines ou de milliers de petits fichiers).

Il faut savoir que dans les fichiers jar, il y a principalement:

- des fichiers classes et des fichiers de ressources structurés en répertoires
- un fichier texte descriptif du contenu du jar appelle MANIFEST.MF; par exemple si le jar contient une application lourde, le fichier manifeste doit inclure une ligne donnant le nom de la classe qui contient la fonction main. Le fichier MANIFEST.MF doit se terminer par un retour chariot.
  Exemple :
  Manifest-Version: 1.0
  Main-class: nompack.nomClasse (sans .class)

Il est donc nécessaire d'apprendre à utiliser cette commande. deux techniques sont proposées :

- l'utilisation de l'IDE de travail, en particulier IntelliJ : il suffit de construire un Artifact (un modèle) à l'aide du menu File/Project Structure, puis dans le menu Build, de choisir l'option BuildArtifacts (jar, From Module dependency)
  Il sera nécessaire de choisir les ressources à inclure dans le jar, choisir le nom du jar, de demander à générer le fichier manifest.mf et d'indiquer la classe qui contient le main.

- l'habituelle technique de la commande en ligne : jar options nomfichierjar
  Exemple :
  - voir le contenu d'un jar : jar tf fichier.jar
  - créer un fichier jar : jar cv fichier.jar liste_fichiers_à_inclure (peut être * ou un répertoire)
  - créer un fichier jar exécutable : jar cfe fichier.jar liste_fichiers_à_inclure (peut être * ou un répertoire)
  - créer un fichier jar avec son manifeste : jar cmf fichiermanifeste fichier.jar liste_fichiers_à_inclure (peut être * ou un répertoire)
  - extraire le contenu d'un jar : jar xf fichier.jar
  - exécuter une application contenue dans un jar : java -jar fichier.jar

  Note : si un jar a besoin d'un autre jar, rajouter dans le manifeste :
  Class-path: . autrefic.jar

ATTENTION : Un dernier point délicat est le fait suivant : si l'on place des ressources (images, logos, sons, etc.) dans un fichier JAR, l'extraction de ces ressources s'effectue en utilisant la méthode **getResource()** de la classe class. Cette méthode renvoie l'URL de la ressource (c'est le chargeur de classes qui se charge de localiser la ressource demandée en fonction de la situation qui prévaut).
Exemple :

```
URL fUrl = getClass().getResource("Logo.gif");
```

ou, dans une méthode statique :

```
URL fUrl = MyClass.class.getResource("Logo.gif");
```

Cette technique fonctionne si la ressource se trouve soit dans un répertoire défini comme *Classpath*, soit dans le fichier JAR. On peut naturellement placer la ressource dans un sous-répertoire ("images/Logo.gif").
Travail à faire :

1. construire deux fichiers classes, l'un avec une ou deux fonctions statiques, l'autre avec un main utilisant ces fonctions.
2. construire un jar d'application à l'aide d'IntelliJ
3. le tester avec : java -jar fic.jar