TP 1 - Prise en main de java en mode console

Ce premier TP a pour but de se familiariser avec Java et de comprendre l'utilité de son compilateur et de la machine virtuelle.
Pour cela, nous allons nous limiter à l'utilisation de la console et d'un simple éditeur de texte pour écrire du code Java. Vous pouvez utiliser votre éditeur de texte préféré, tant qu'il reconnait bien les mots-clefs du langage et vous permet une identation automatique : par exemple gedit, kate, emacs,... Nous verrons dans les TP suivants un IDE (environnement de travail) nommé Eclipse pour permettre de gérer des projets de programmation plus important.

Un peu d'organisation : en Java le répertoire est l'organe clef du rangement des programmes.
Créer un répertoire `TP1` avec en sous-répertoire `Exo1` etc.

Si vous codez sous Linux (ce qui est recommandé), les lignes de commande `javac` et `java` sont à exécuter en ligne de commande dans un terminal/console.
Si vous codez sous Windows : vous pouvez utiliser une 'invite de commande" avec les mêmes fonctionnalités : pour se déplacer dans l'invite de commande Windows, les commandes dites "DOS" sont similaires à celles de Linux sauf pour lire le contenu d'un fichier "dir" et les slachs / sont remplacées par des anti-slachs.

Exercice 1 – Premier Programme

On rappelle qu'en Java chaque classe publique doit être définie dans un fichier qui lui est propre. Le nom du fichier doit être le nom de la classe qu'il contient, auquel on ajoute le suffixe .java. Les noms des classes doivent être constitués d'un seul mot dont la première lettre est une majuscule : pour pouvoir utiliser plusieurs mots en un seul, on concatène deux mots en utilisant le jeu des majuscules par exemple `MotComposeDeCinqMots` ou `Mot_compose_de_cinq_mots`.

Q 1.1 Écrire un premier programme affichant à l'écran 10 fois "C'est mon premier programme en Java !"dans un fichier .java de même nom que la classe utilisée dans le pogramme.

Q 1.2 Compiler le programme en utilisant le commande `javac` puis vérifier que le fichier `.class` correspondant a bien été créé. Exécuter le programme en utilisant la machine virtuelle `java`. (on ne met pas l'extension .java en paramètre).
La machine virtuelle va éxécuter séquentiellement (on dit interpréter) une à une les commandes du code compilé dans un fichier .class contenant une méthode `main`.

Q 1.3 Si vous codez sous Linux (resp. Windows) et que vous disposez [1] du système Windows (resp. Linux), que donne l'exécution de la même ligne `java PremierProgramme` à partir du code compilé sous Linux (resp. Windows). Que constatez-vous ? Pourquoi ?

Exercice 2 – Arguments en ligne de commande

1. en accès rapide... ne perdez pas de temps, vous pourrez le faire une autre fois !

On veut écrire une classe `PrintArgs` qui affiche les arguments de la ligne de commande. La ligne de commande pour le programme Java `PrintArgs.class` est obtenue en ajoutant des mots ou des nombres, séparées par des espaces après l'appel au programme, par exemple

```
java PrintArgs bonjour 4 ah bon
```

On rappelle que les arguments de la ligne de commande sont récupérés dans le tableau `String args[]` en paramètre du main.

Q 2.1 Dans un premier temps, afficher le premier argument de la ligne de commande. Que se passe-t-il si l'on ne passe pas d'argument lors de l'exécution du programme ?

Q 2.2 Écrire une boucle affichant tous les arguments avec une boucle utilisant les indices du tableau.

Q 2.3 Même question avec une boucle de type "for each".

Exercice 3 – Calculette simple

On souhaite écrire un programme affichant la somme d'entiers pris en paramètres sur la ligne de commande.

Voici un exemple d'exécution :

```
java Calculette 15 5 231
La somme des entiers: 15 5 231 est 251.
```

Q 3.1 Réaliser cette calculatrice en réalisant autant d'appel à `System.out.print()` que vous voulez.

Q 3.2 Réaliser cette calculatrice en obtenant à la fin du programme une chaîne de caractères contenant la liste de tous les entiers. Vous n'utiliserez donc qu'une seule fois à la fois `System.out.println()` pour l'affichage.

Exercice 4 – Et si le cours ne donne pas toutes les commandes Java...

On veut écrire le jeu "DevineChiffre". Le jeu demande à l'utilisateur de deviner un nombre entier entre 1 et 100. Le joueur propose un nombre et le jeu lui répond si le nombre mystère est plus grand ou moins grand. A la fin du jeu, le nombre d'essais est indiqués : si le joueur a trouvé en plus de 5 essais, il a perdu, sinon il est dit gagnant.

Pour construire ce jeu, il vous manque par rapport au cours :
- une méthode permettant de saisir des entiers au clavier.
- une méthode pour déterminer un entier au hasard entre 1 et 100

Pour la saisie au clavier, il est possible d'utiliser un objet de la classe `Scanner` (et particulièrement sa méthode `nextInt()`). Pour comprendre son fonctionnement, il est donc nécessaire de regarder la documentation disponible sur internet (pensez à mettre des liens en bookmark...).
`https://docs.oracle.com/javase/10/docs/api/java/util/Scanner.html`

Pour un nombre aléatoire : à vous de chercher une solution en utilisant un browser internet.

Exercice 5 – Jeu du bonneteau

Le jeu du bonneteau est un jeu classique de hasard... souvent utilisé comme une arnaque : trois cartes sont posées à la suite sur une table (typiquement 2 valets et une dame de coeur), un joueur, appelé le maître du jeu, mélange les cartes en échangeant leurs positions et le joueur adverse doit deviner où se trouve la dame de coeur.
Nous allons modéliser le jeu en utilisant 4 classes : `Carte` pour représenter une carte, `Emplacement` pour représenter là où peuvent être posées les cartes, `MaitreDuJeu` pour représenter cette personne, et `MainJeu` pour gérer le déroulé du jeu. Dans chaque classe, pensez à protéger les données par des champs `private` si besoin. Noter qu'il n'est pas demandé d'ajouter des méthodes ou des attributs non prévus dans l'énoncé.

Q 5.1 Donner la classe `Carte` qui contient uniquement
- un attribut `String nom`
- deux constructeurs : le premier sans paramètre (qui nomme la carte par défaut "Valet"), et le deuxième avec un seul paramètre : le nom
- la méthode `String toString()` qui permet de renvoyer le nom de la carte

Q 5.2 Donner la classe `MainJeu` qui contient une fonction main où est créé un objet Carte `Carte` stockée dans une variable cible (ce sera la carte à retrouver) et affichez-la.

Q 5.3 Donner une classe `Emplacement`. Tester **AU FUR ET A MESURE** votre classe dans la fonction `main` de `MainJeu` en créant trois variables `Emplacement gauche, milieu, droite` avec deux valets et la carte cible ; puis, pour un des emplacements contenant un valet, afficher sa description, enlever sa carte, afficher à nouveau l'emplacement, remettre la même carte et afficher.
La classe `Emplacement` contient :
- deux attributs `Carte carte` et `String nom`. La variable carte permet de stocker la carte qui sera posée sur cet emplacement
- deux constructeurs : `Emplacement(String nom)` qui initialise juste le nom, et `Emplacement(String nom, Carte carte)` qui permet de passer également une carte à poser sur l'emplacement en paramètre.
- la méthode `String toString()` qui renvoie "nom de l'emplacement : nom de la carte"
- une méthode `boolean estVide()` qui permet de savoir si l'emplacement est disponible ;
- les méthodes `boolean poser(Carte carte)` qui permet de poser une carte sur l'emplacement (renvoie faux si l'emplacement n'est pas vide, vrai si l'emplacement était vide et la carte a été posée), et `Carte enlever()` qui permet de retirer la carte de l'emplacement (renvoie null s'il n'y a pas de carte).

Q 5.4 Donner une classe `MaitreDuJeu` qui modélise le maître du jeu. Elle dispose :
- 3 attributs `Emplacement` pour stocker les références de 3 emplacements
- un constructeur avec 3 paramètres `Emplacement`
- la méthode `String toString()` qui permet de renvoyer la description de l'état du jeu (les 3 emplacements avec la carte qu'ils contiennent).

Tester l'affichage de cette classe.

Q 5.5 Ajouter à la classe `MaitreDuJeu` la méthode `void echanger(Emplacement a, Emplacement b)` qui permet d'échanger les cartes entre les deux emplacements en paramètres (en utilisant les méthodes `poser` et `enlever` de Carte). Ajouter également la méthode `void echanger(int i, int`

`j)` qui permet d'échanger les deux cartes entre des emplacements indiqués par des entiers `i` et `j` (en considérant que gauche est 0, milieu 1 et droit 2. Cela permet d'utiliser l'astuce suivante : si `i+j==1` alors c'est gauche et milieu qui sont échangés ; si `i+j==2` alors c'est gauche et droit ; sinon c'est milieu et droit).

Q 5.6 Ajouter la méthode `boolean reveler(Emplacement a)` et `boolean reveler(int i)` qui renvoie true si l'emplacement a (ou correspondant à l'entier i) contient la carte cible, false sinon. Elle permet de modéliser la demande faite au maitre du jeu de révéler une carte.

Q 5.7 Ajouter la méthode `void melanger()` qui permet d'échanger au hasard deux emplacements ; la méthode `void melanger(int n)` qui permet de faire `n` mélanges au hasard.

Q 5.8 Ajouter dans `MainJeu` une fonction static qui mélange 100 fois aléatoirement, tire au sort un emplacement parmi les 3, et affiche si le joueur adverse a gagné ou perdu. Tester la fonction dans le main. Répéter dans le main l'expérience 1000 fois. Quelle est la moyenne du nombre de fois où le programme gagne ?

Q 5.9 Quel est l'intérêt de déclarer en private (sans mettre de méthodes de modification du nom) pour les variables d'instances de la classe `Carte` ? Le maitre du jeu a-t-il un moyen de tricher dans ce cas ?

Q 5.10 Si la classe `Carte` posséde une méthode permettant de changer le nom de la carte, est-ce la classe `Emplacement` pourrait alors être truquée : que faire pour protéger davantage ?

Exercice 6 – C ou Java

Cet exemple a pour but de montrer les différences de performance entre un algorithme codé en langage C et le même codé en langage Java.

Cet algorithme reprend le principe algorithmique issu de la formule du triange de Pascale pour calculer le nombre C_p^n de p combinaisons parmi n éléments (avec $1 \leq p \leq n$).

Q 6.1 Télécharger le programme `triangle_de_Pascale.c` en langage C. Compiler avec
(`gcc -o triangle_de_Pascale triangle_de_Pascale.c`)
et testez-le pour des valeurs de p et n pour lesquels vous connaissez le résultat (Pour rappel $C_p^n = \frac{n!}{p!(n-p)!}$). Quelle est la complexité de l'algorithme ?

Q 6.2 Exécuter le programme obtenu en demandant au système le temps d'exécution du programme par la commande `time triange_de_Pascale`. Faites-le pour des valeurs importantes (par exemple $n = 30000, p = 250$. Notez que l'affichage n'est plus significatif : en effet, les nombres entiers obtenus sont tellement grands qu'ils ne sont pas stockable dans un type classique de C ou de Java.
Important : compiler le programme C avec et sans l'option "-O3" pour bénéficier d'une optimisation intéressante du code.

Q 6.3 Écrire le programme `Pascal.java` équivalent en Java (n'hésitez pas à copier/coller). Compiler le programme puis l'exécuter en mesurant le temps (toujours avec `time`). Comment peut-on expliquer la différence de vitesse ?