Examen VARI1 (NFP135)

Les documents imprimés sur papier sont autorisés, smart-phone/tablettes/ordinateurs interdits.
Note : chaque question vaut 1 point sur 20, pour un total de 9/20 pour l'examen (1pt bonus)

Exercice 1 Déclarer un tableau de taille 5 et initialiser ce tableau avec les valeurs : `0` ; `2` ; `5.3` ; `3` ; `1`. Dans une deuxième étape, multiplier chaque élément du tableau par 2 (à l'aide d'une boucle `for`) et afficher le tableau résultant avec les valeurs doublées (`java` ou `processing`). Vous pouvez utiliser la classe `Arrays`.

Exercice 2 Écrire (en `Java` ou `Processing`) une fonction

float noteFinale(**float** exam, **float** projet, **float** devoir1, **float** devoir2)

qui calcule et renvoie la note finale d'une UE en fonction de : une note d'examen, une note de projet, et deux notes de devoirs. Les règles sont les suivantes :

- Une note d'examen inférieure à 4 est éliminatoire et `noteFinale(...)` renvoie 0 dans ce cas.
- La plus petite des deux notes de devoirs et ignorée.
- La note finale est la moyenne des trois notes suivantes : la note d'examen, la note de projet et la note non-ignorée de devoir.

Exemple : Pour exam=12, projet=16, devoir1=4 et devoir2=9, la fonction renvoie la note finale $\frac{12+16+9}{3} = 12.33$.

Exercice 3 Écrire le morceau de code (`Java` ou `Processing`) qui affiche les $n = 50$ premiers entiers naturels multiples de 5 qui ne sont pas des multiples de 20. Par exemple, s'il avait fallu résoudre cela pour $n = 7$, le code aurait du afficher : 5, 10, 15, 25, 30, 35, 45

Note 1 *: vous pouvez utiliser* `(x%20==0)` *pour vérifier si x est un multiple de 20.*
Note 2 *: le placement correct des virgules dans l'affichage compte pour 0.2/1 dans la notation.*

Exercice 4 Écrire (en `Java` ou `Processing`) une fonction

boolean nombreParfait(**int** n)

qui détermine si n est un nombre parfait. Un entier naturel n est parfait s'il est égal à la somme de ses diviseurs inférieurs à n. *Exemples :*

- pour $n = 6$ on obtient les diviseurs 1, 2 et 3. Donc $6 = 1 + 2 + 3$ est parfait.
- pour $n = 28$ on obtient les diviseurs 1, 2, 4, 7 et 14. Donc $28 = 1 + 2 + 4 + 7 + 14$ est parfait.

Exercice 5 Écrire le morceau de code (`Java` ou `Processing`) qui permet de calculer la valeur approximative de π à l'aide de la formule suivante

$$\frac{\pi^3}{32} \approx \frac{1}{1^3} - \frac{1}{3^3} + \frac{1}{5^3} - \frac{1}{7^3} + \cdots + \frac{1}{97^3} - \frac{1}{99^3} + \frac{1}{101^3}$$

Note : Vous avez ci-après un « copier-coller » de la documentation officielle de la fonction `Math.pow(...)`. On peut traduire la dernière phrase par «renvoie la valeur du premier argument élevée à la puissance du second argument». Par exemple, `Math.pow(2,3)=8` et `Math.pow(2,0.5)=`$\sqrt{2} \approx$ `1.4142`; rappel : $x^{\frac{1}{3}} = \sqrt[3]{x}$.

static double pow(double a, double b)

Returns the value of the first argument raised to the power of the second argument.

Exercice 6 Écrire un programme complet `Java` qui permet de lire 7 entiers naturels à partir d'un fichier texte `in.txt` (utiliser la classe `BufferedReader`). On suppose que ce fichier comporte un entier par ligne, donc un total de 7 lignes. Stocker dans un tableau `int[] tabSup10` tous les entiers strictement supérieurs à 10. Afficher ces entiers en ordre inverse.

Exemple : pour le fichier à droite, le programme devrait afficher :

12,14,11

Note : *il n'y a pas de virgule après 11*

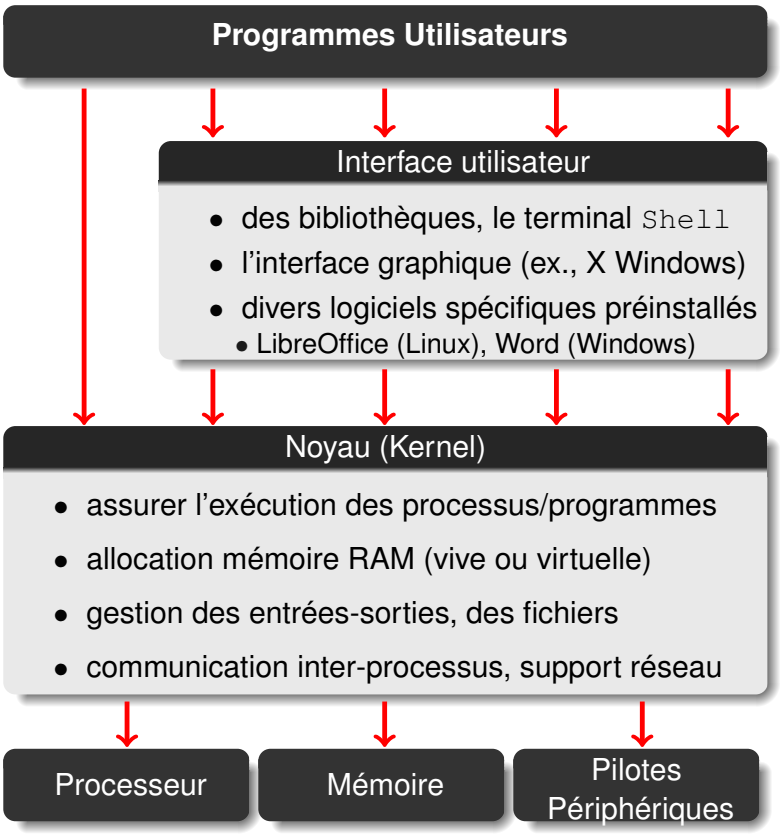
in.txt
———————
3
7
11
14
12
8
−9

Exercice 7 Question de théorie :

A. Que représente la figure à droite ? Décrire en quelques lignes (maximum 10) les interactions et les modules dans cette figure. Par exemple, expliquer ce que représente les flèches des programmes utilisateurs vers l'interface utilisateur, ou détailler l'expression « mémoire RAM (vive ou virtuelle) »
[0.5pt]

B. Donner trois commandes `Linux` (resp.) pour :
   — lister les fichiers du dossier courant.
   — afficher le contenu d'un fichier `Scoop.java`.
   — afficher toutes les lignes du fichier `Scoop.java` qui contiennent le mot «`toto`».
   [0.3pt]

C. Décrire en maximum 2 lignes la signification du mot «constructeur».
[0.2pt]



Exercice 8 Écrire un programme (Java/Processing) qui permet de résoudre par programmation dynamique un problème de sac-à-dos (cours 15) défini par une capacité, des poids et des valeurs/profits saisies en dûr dans le code comme dans l'exemple ci-dessous. Il faut *uniquement afficher le profit optimal*, sans les articles associés.

```
static int capac = 100; //capacité
static int[] poids = {20,14,23,45,54,23,33,43,15,1,23,43,2};
static int[] val   = {40,25,42,80,99,49,76,89,33,3,49,88,5};//profits
```

Exercice 9 Un carré magique (simplifié) est une matrice 3×3 avec 9 entiers strictement positifs (`mat[i][j]`> 0) disposés de sorte que la somme sur chaque ligne et chaque colonne soit la même. Par exemple, la matrice ci-après a une somme de 10 sur chaque ligne et chaque colonne.

$$\texttt{mat} = \begin{bmatrix} 2 & 5 & 3 \\ 3 & 1 & 6 \\ 5 & 4 & 1 \end{bmatrix}$$

Écrire un programme java qui permet de lire à partir d'un fichier texte `in.txt` les valeurs du sous-carré 2×2 en haut à gauche d'une matrice 3×3. Ce fichier comporte donc quatre lignes avec les valeurs suivantes :
`mat[0][0]`
`mat[0][1]`
`mat[1][0]`
`mat[1][1]`
Voici un exemple de fichier `in.txt` pour la matrice `mat` définie ci-dessus :

```
in.txt
------
2
5
3
1
```

Le programme devrait lire ce fichier et essayer de remplir le reste de la matrice avec des valeurs strictement inférieures à 10 pour construire un carré magique 3×3. Pour clarifier : toutes les valeurs du carré doivent être strictement supérieures à 0 et strictement inférieures à 10. Si cela est possible, le carré magique doit être affiché. Sinon, on affiche : «impossible de construire un carré magique à partir de ces quatre nombres». Par exemple, il est impossible de construire un carré magique 3×3 à partir du sous-carré 2×2 suivant :

$$\begin{bmatrix} 5 & 3 & \dots \\ 9 & 8 & \dots \\ \dots & \dots & \dots \end{bmatrix}$$