Programmation orientée objet

Série 2

Joel Cavat / 2020

Exercices pratiques

2.1 Exercice

A l'aide de Maven (cf. exercice du support de cours) Réalisez ensuite une méthode statique `askInt()` qui demande un entier à l'utilisateur et le retourne. Tant que l'utilisateur n'entre pas une valeur entière, la demande lui est reformulée. Produisez deux versions:

- une première avec une boucle et qui utilise `Scanner.nextInt()` (doc)
- une deuxième sans boucle et qui utilise `Integer.parseInt()` (doc) en combinaison avec le `Scanner`

Exemple d'utilisation dans le terminal:

```
Entrez une valeur: 34ead
Valeur erronée
Entrez une valeur: asdf
Valeur erronée
Entrez une valeur: 12
La méthode a retourné 12
```

Exemple d'utilisation dans le code:

```
public static void main(String[] args) {

   System.out.println("La méthode a retourné " + askInt() )

}
```

2.2 Exercice

A l'aide de la classe utilitaire `Math`, écrivez plusieurs fonctions :

- `double minMaxRange(double value, double minValue, double maxValue)` qui retourne une valeur bornée entre une valeur min et une valeur max. Cette fonction ne doit pas contenir de `if` et doit tenir sur une ligne.

  - `minMaxRange(1.0, 0.0, 3.0) -> 1.0`
  - `minMaxRange(5.0, 0.0, 3.0) -> 3.0`
  - `minMaxRange(-4.3, 0.0, 3.0) -> 0.0`
- `double normalize(double value, double minSource, double maxSource, double minTarget, double maxTarger)` qui transforme une valeur comprise dans une fenêtre de valeurs sur une autre fenêtre de valeurs.
  - `normlize(0.0, 0.0, 1.0, 500.0, 600.0) -> 500.0`
  - `normlize(1.0, 0.0, 1.0, 500.0, 600.0) -> 600.0`
  - `normlize(0.5, 0.0, 1.0, 500.0, 600.0) -> 550.0`
  - `normlize(0.0, 0.0, 1.0, 500.0, 600.0) -> 500.0`
- `double random(double minValue, double maxValue)` qui retourne une valeur aléatoire comprise entre `minValue` (comprise) et `maxValue` (non comprise). Utilisez `Math.random()`
- `int random(int minValue, int maxValue)` identique à la précédente, mais pour les entiers.
- `List<Double> random(int nb, double minValue, double maxValue)` qui retourne une liste de `nb` valeurs comprises entre `minValue` et `maxValue`.
- `List<Integer> random(int nb, int minValue, int maxValue)` identique à la précédente, mais pour les entiers.
- `List<List<Double>> altitudeToShadesOfGray(List<List<Double>> altitudes)` qui retourne une matrice d'altitudes en une matrice de nuances de gris comprises entre 0 et 255 (valeurs comprises).
  - la matrice en entrée, représentée à l'aide d'une liste de listes, peut être irrégulière.

$$\begin{bmatrix} 500.0 & 550.0 & 600.0 \\ 570.0 & 510.0 & \end{bmatrix} \xrightarrow{altitudeToShadesOfGray} \begin{bmatrix} 0.0 & 127.5 & 255.0 \\ 178.5 & 25.5 & \end{bmatrix}$$

2.3 Exercice *(Triangle)*

A l'aide d'un paramètre n indiquant la hauteur, écrivez une fonction `printTriangle(int n)` qui affiche un triangle sous cette forme (pour $n = 4$) :

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

2.4 Exercice *(Sapin)*

A l'aide d'un paramètre n indiquant la hauteur, écrivez une fonction `printFir(int n)` qui affiche un triangle sous cette forme (pour $n = 4$):

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

2.5 Exercice *(Factorielle)*

Réalisez une fonction permettant de calculer la valeur factorielle d'un nombre n :

$$n! = \prod_{i=1}^{n} i$$

2.6 Exercice *(Pi)*

Réalisez une fonction permettant de calculer une approximation de π :

$$\sum_{n=1}^{N} \frac{1}{n^4} = \frac{\pi^4}{90}$$

Cette fonction prend N en paramètre et retourne π.

2.7 Exercice *(Jeu du Serpent)*

Ecrivez un programme qui affiche un serpent. A chaque itération, le serpent est affiché à l'aide d'une suite d'étoiles puis le programme demande d'augmenter ou de diminuer la taille du serpent. Le programme s'arrête lorsque la taille du serpent est nulle.

Exemple d'utilisation:

```
*
Direction: +
**
Direction: +
***
Direction: -
**
Direction: +
***
Direction: -
**
Direction: -
*
Direction: -
Au revoir!
```

L'algorithme est relativement simple et peut être modélisé ainsi :

```
taille initiale du serpent est de 1
tant que la taille est supérieur à 0 alors :
- afficher le serpent en fonction de sa taille
- demander à l'utilisateur d'augmenter ou de diminuer la taille
  - s'il souhaite l'augmenter, incrémentez la taille du serpent de 1
  - s'il souhaite la diminuer, décrémentez la taille du serpent de 1
```

Le programme se termine lorsque la taille est de zéro.

Considérations:

- A vous de définir (= simplifiez-vous la vie) ce qu'il se passe si l'utilisateur entre une valeur différente de `+` ou `-` (quitter le programme, ne rien faire, afficher autre chose qu'un serpent, ...)
- Vous êtes libre d'afficher le serpent sur une ou plusieurs lignes
- Un **bonus** est accordé si vous affichez une tête au serpent, par exemple, pour une taille de 4 le serpent pourrait être affiché ainsi : `***0`, `***X` ou `***(^^)`

2.8 Exercice *(Vecteurs)*

Réalisez un ensemble de fonctionnalités permettant le calcul vectoriel sur des vecteurs. Utilisez un tableau de `double`s pour représenter un vecteur.

Ecrivez les méthodes statiques suivantes:

- la méthode `add` . Elle prend deux vecteurs et retourne leur addition. Par exemple `add(new double[]{1.0,2.0,2.0}, new double[]{2.0,1.0,-5.0})` retourne `double[3]{3.0, 3.0, -3.0}`.
- `mul` multiplie un vecteur par une valeur numérique.
- à l'aide des deux premières fonctions, écrivez la méthode statique `sub` qui soustrait le deuxième vecteur du premier.
- `len` qui retourne le nombre de composantes d'un vecteur.
- `norm` qui retourne la norme (ou la longueur) d'un vecteur. Par exemple, `norm(new double[]{1.0, 2.0, 2.0})` retourne 3.0. Calcul de la norme d'un vecteur:

$$\vec{v} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix}$$

  est $\|\vec{v}\| = \sqrt{v_1^2 + v_2^2 + v_3^2}$.
- `sum` qui prend une liste de vecteurs en paramètre et les additionne.
- `norms` qui prend une liste de vecteurs en paramètre et retourne la norme de leur addition.
- `concat` qui concatène deux vecteurs.
- `sliceFrom` qui retourne un sous-ensemble du vecteur à partir d'un indice.
- `sliceTo` qui retourne un sous-ensemble du vecteur jusqu'à un indice (non compris).
- `slice` qui est la combinaison des deux précédentes (avec un index de début et un de fin).

Considérations:

- Respectez le nommage.
- Respectez les conventions de nommage du langage oracle: code conventions.
- Ecrivez des méthodes courtes et concises.
- Retournez des nouvelles copie de tableaux. Ne modifiez pas vos arguments.
- Utilisez les fonctionnalités offertes par la classe `Arrays`: oracle: Arrays.
- Ecrivez vos méthodes statiques dans un fichier `Vector.java` (sans méthode `main`).
- La méthode principale avec vos tests doivent se trouver dans un fichier `App.java`.

Gestion d'erreurs:

- Les fonctions qui retournent des vecteurs doivent retourner un **vecteur vide** si les vecteurs entrés en arguments ne sont pas conformes.

2.9 Exercice *(Matrices)*

Réalisez des fonctionnalités sur les matrices :

- l'affichage d'une matrice
- la multiplication de deux matrices
- l'addition de deux matrices

Retournez une matrice vide en cas d'erreur

2.10 Exercice *(Jeu du pendu - itération 1)*

Réalisez une fonctionnalité de vérification pour le jeu du pendu (`Hangman.java`) :

- une méthode `check(String currentWord, String guessWord, char letter)`

Exemple d'utilisation

- `check("_____", "amies", 'm')` retourne `"_m___"`
- `check("_______", "caramba", 'a')` retourne `"_a_a__a"`
- `check("_____", "frite", 'a')` retourne `"_____"`

Réalisez une fonctionnalité d'affichage du pendu en fonction du nombre d'essais manqués:

- Première erreur, affichage d'une potence

```
----------
|/
|
|
|
|
|
|
```

- Exemple de 5e erreur

```
----------
|/       |
|       (_)
|       /|
|
|
|
|
```

- Exemple à la 9e et dernière erreur

```
----------
|/       |
|       (_)
|       /|\
|        |
|       / \
|
|
```

2.11 Exercice *(Listes)*

Réalisez des fonctionnalités sur des listes. Complétez le code ci-dessous:

```
public class ListHelper {

  /* Retourne une nouvelle liste où toutes les valeurs sont doublées */
  public static List<Integer> doubleThat(List<Integer> is) {









  }

  /* Retourne une nouvelle liste où toutes les valeurs sont plus grandes ou
     égale à un seuil */
  public static List<Integer> filterGreaterOrEqual(List<Integer> is, int value) {









  }

  /* Retourne une nouvelle liste où toutes les valeurs positives sont
     multipliées par deux. Les valeurs négatives sont omises */
  public static List<Integer> filterPositiveAndThenDoubleThem(List<Integer> is) {
    // Faites simple, ne cherchez pas l'efficacité


  }

  public static void main(String[] args) {

    /* Vos tests éventuels */









  }
}
```

2.12 Exercice *(Test 2018/2019)*

Ecrivez une méthode qui compte le nombre d'éléments positifs dans un tableau de tableaux. Un exemple d'utilisation est fourni dans la méthode principale `main`.

Complétez le code ci-dessous :

```
public class Test1 {









  public static void main(String[] args) {
    double[][] example = {
      {1.0, -2.0},
      {3.0},
      {1.5, -2.5, 3.0}
    };
    int res = countPositive( example );
    System.out.prinln("Count: " + res); /* afficherait "Count: 4" */
  }
}
```

2.13 Exercice *(Test 2018/2019)*

Ecrivez une méthode qui additionne tous les éléments d'un tableau de tableaux. Un exemple d'utilisation est fourni dans la méthode principale `main`.

Complétez le code ci-dessous :

```
public class Test1 {









```

```
  public static void main(String[] args) {
    double[][] example = {
      {1.0, 2.0},
      {3.0},
      {1.5, 2.5, 3.0}
    };
    double s = sum( example );
    System.out.prinln("Sum: " + s); /* afficherait "Sum: 13.0" */
  }
}
```

2.14 Exercice *(Test 2019/2020)*

Réalisez la méthode `removeAll()` qui permet de supprimer toutes les occurences d'une lettre dans une chaine de caractères.

Aidez-vous des méthodes `removeFirstOccurrence()` et `contains()` qui vous sont fournies et qui permettent respectivement de supprimer la première occurrence d'une lettre d'une chaîne de caractères et de préciser si une lettre se trouve dans une chaîne.

Indices : Favorisez la simplicité. Aidez-vous évidemment des deux méthodes fournies.

```
public class Test1 {

  /**
    * Supprime une lettre d'une chaîne de caractères
    * removeFirstOccurrence("abracadabra", 'a') retournerait "bracadabra"
    * removeFirstOccurrence("babar", 'r') retournerait "baba"
    * removeFirstOccurrence("babar", 'z') retournerait "babar"
    * removeFirstOccurrence("", 'r') retournerait ""
    */
  public static String removeFirstOccurrence(String s, char letter) {
    /* méthode fournie */
    /* Rien à faire ici */
  }

  /**
    * Retourne true si la lettre se trouve dans le mot
    * contains("babar", 'z') retournerait false
    * contains("babar", 'a') retournerait true
    * contains("", 'a') retournerait false
    */
  public static boolean contains(String s, char letter) {
    /* méthode fournie */
    /* Rien à faire ici */
  }
```

```java
  /**
    * Supprime toutes les occurrences d'une lettre dans un mot.
    * Retourne la version sans occurrence.
    * removeAll("abracadabra", 'a') retournerait "brcdbr"
    * removeAll("salsa", 's') retournerait "ala"
    * removeAll("biere", 's') retournerait "biere"
    */
  public static String removeAll(String s, char letter) {
    // TODO:









  }

  public static void main(String[] args) {
    /* Rien à faire ici */
  }
}
```

2.15 Exercice *(Manipulation de collections/Strings)*

Ecrivez les fonctionnalités ci-dessous:

```
public class App {

  /* Détermine si un nombre qui se trouve dans une chaîne de caractères contient
   * un entier
   * Note: prend en compte le signe, mais ne prend pas en compte les espaces
   * Exemples:
   * isNumeric("42") -> true
   * isNumeric("  22 ") -> true
   * isNumeric("   -33    ") -> true
   * isNumeric(" 22.0") -> false
   * isNumeric("2f3") -> false */
  public static boolean isNumeric(String term) {
    /* TODO */
  }

  /* Retourne un tableau d'indices où chaque valeur indique la position d'un
   * caractère dans une chaîne
   * Exemples:
   * indexes("maison", 'i') -> {2}
   * isNumeric("tralala", 'a') -> {2,4,6}
   * isNumeric("coucou", 'x') -> {} */
  public static int[] indexes(String term, char c) {

  }

  /* Détermine si une chaîne est en majuscule
   * Conseil: cf. docs String
   * Note: peut tenir sur une ligne
   * Exemples:
   * isUpper("ABC") -> true
   * isUpper("AbC") -> false */
  public static boolean isUpper(String term) {
    /* TODO */
  }

  /* Retourne une version triée d'une chaine de caractères
   * Conseil: cf. docs String + Arrays
   * Exemples:
   * sorted("lkjlkjKJk9") -> "9JKjjkkkll" */
  public static String sorted(String term) {
    /* TODO */
  }

}
```

2.16 Exercice *(Nombre mystérieux)*

Réalisez un jeu `MysteryNumber` invitant un utilisateur à deviner un nombre compris entre 1 et N. Tant que l'utilisateur n'a pas trouvé ce nombre, l'algorithme continue de demander une valeur à l'utilisateur.

La première version tire aléatoirement un nombre que l'utilisateur doit entré

```
java MysteryNumber 100
```

```
Entrez un nombre : 28
Entrez un nombre : 40
Entrez un nombre : asdf
Ceci n'est pas un nombre!
Entrez un nombre : -22
Le nombre n'est pas compris entre 1 et 100
Entrez un nombre : 42
Bravo, vous avez trouvé le nombre en 5 essais
```

Réalisez une seconde version où c'est un algorithme qui doit deviner le nombre que vous avez en tête:

- `guess` indique que c'est à l'ordinateur de deviner le nombre
- 1024 est la valeur max

```
java MysteryNumber guess 1024
```

```
Ici votre ordinateur.
S'agit-il du 512 ? [o/+/-]: -
S'agit-il du 256 ? [o/+/-]: -
S'agit-il du 128 [o/+/-]: +
S'agit-il du 192 [o/+/-]: +
S'agit-il du 224 [o/+/-]: -
S'agit-il du 208 [o/+/-]: +
S'agit-il du 216 [o/+/-]: -
S'agit-il du 212 [o/+/-]: -
S'agit-il du 210 [o/+/-]: o
YOUPIE, l'IA bat définitivement l'humain.
```

Vous remarquez sans doute la stratégie employée : diviser l'espace de recheche par deux. La recherche dychotomique utilise cette technique. Le nombre de coup maximum d'un tel algorithme est donc de `log(N)`.