TPs JAVA & POO

TP 1

Écrire un programme Java permet de déclarer un tableau d'entiers en Java, et qui ensuite va afficher le plus grand nombre, sa position; le plus petit nombre, et sa position.

Dans l'affichage, il faut afficher le tableau en format [element1, element2,...,elementN], et dans l'affiche du plus grand nombre faire afficher le text(PG) et pour le plus petit(PP):

Ex :

Tb : [12,3,5,6,-3]

Résultat:

Tb : [12**(PG)** , 3, 5, 6, -3**(PP)**]
Plus grand : 12
Plus petit : -3

TP2

Écrire un programme Java demande à l'utilisateur de saisir un nombre, et qui ensuite affiche le nombre inverse.

Ex :

Votre nombre : 17

Résultat : 71

Votre nombre : -89
Résultat : -98

Votre nombre : -20
Résultat : -2(Car le zéro devant un nombre n’est pas pris en compte)

TP 3

Ecrire un programme Java demande à l’utilisateur de saisir 2 chaines de caractères et qui ensuite renseigne si ce sont des anagrammes.

Qu'est-ce qu'une anagramme ?

Une anagramme est un mot ou un groupe de mots obtenu en changeant de place les lettres d'un autre mot ou groupe de mots.
Par exemple, "**GARE**" est une anagramme de "**RAGE**", ou "**GARE MAMAN**" est une anagramme de "**ANAGRAMME**".

TP 4

Écrire un programme Java qui demande à l’utilisateur de saisir une chaîne de caractères et va mettre en majuscule toutes les premières lettres de chaque mot.

Ex :

Input : “je suis dans la joie”
Résultat : Je Suis Dans La Joie

TP 5

Écrire un programme Java qui demande à l’utilisateur d’entrer une phase à partir du clavier.

Ensuite, le programme va retirer tous les doublons de la phase, puis à la fin le programme va afficher:

- ❏ La chaîne de départ fournie par l’utilisateur
- ❏ La chaîne après suppression des doublons
- ❏ Une liste contenant tous les doublons et le nombre de fois que

ceux-ci figuraient dans l'ancienne chaîne

Après essayer le même exercice en supprimant les mots qui se répètent.

TP 6

Écrire une méthode Java qui prend en paramètre un tableau 2D, et un nombre.

La méthode devrait nous retourner la valeur correspond au nombre de fois que ce nombre se trouve dans le tableau et ses différentes positions.

Ex :

```
maMethode({
  {1, 4, 2, 1},
  {6, 3, 8, 9},
  {1, 5, 1, 0}
},1)
> 1 se retrouve 4 fois dans les emplacements suivants :
(0,0),(0,3),(2,0),(2,2)
```

TP 7

Écrire une méthode Java qui prend en paramètre un tableau de caractères.

Et qui va ensuite retourner un nouveau tableau avec une alternance entre les lettres en Maj et Min.

Ex :

```
maMethode(['a', 'b', 'c', 'd', 'e'])
> ['a', 'B', 'c', 'D', 'e']
```

TP 8

On vous donne comme argument un tableau contenant des chaînes de directions (haut, bas, gauche, droite). Imaginez une personne debout sur

une grille au point 0, 0. Pour chaque direction dans le tableau de chaînes, déplacez votre personne dans cette direction sur la grille. Retournez le point final X,Y où se trouve la personne sous la forme d'un tableau de deux entiers.

Exigences
Doit retourner un tableau de deux entiers

Exemple:

maMethode(["haut", "haut", "bas", "gauche", "gauche", "droite", "haut"])

> [-1, 2]

TP 9

Recherche de caractères identiques dans une rangée
On vous donne une chaîne de caractères d'un seul mot comme argument. Vous retournerez true si elle contient deux caractères identiques dans une rangée,c'est-à-dire qui se suivent, sinon elle retournera false.

Exigences
- ❏ Doit retourner true ou false
- ❏ Doit également fonctionner avec les caractères spéciaux

Exemple n° 1
maMethode("te**rr**ific")
> true
Exemple n°2
maMethode("chats")
> false
Exemple n°3
maMethode("chats **!!**")
> true

TP 10

Le but de cet exercice est de simuler une tirelire dans laquelle on stocke et retire de l'argent et que l'on souhaite utiliser pour payer un certain budget (de vacances,par exemple).
Voici les détails de l'exercice :

Les traitements qui lui sont spécifiques sont :

- ☐ une méthode **getMontant** retournant le montant de la tirelire;

- ☐ une méthode **afficher** affichant les données de la tirelire sous le format suivant :
  - Vous êtes sans le sou,si la tirelire ne contient pas d'argent
  - Vous avez : <montant> € dans votre tirelire.

- ☐ une méthode **secouer** affichant sur le terminal le message Bing bing,suivi d'un saut de ligne, dans le cas où la tirelire contient de l'argent, et qui n'affiche rien sinon;

- ☐ la méthode **remplir** mettant un montant donné en paramètre (double) dans la tirelire. Seuls les montant positifs seront acceptés (dans le cas contraire on ne fait rien);

- ☐ une méthode **vider** (réinitialisant le montant de la tirelire à zéro);
- ☐ une méthode **puiser** permettant de puiser dans la tirelire un montant donné en paramètre. Si le montant est négatif il sera ignoré. Si le montant en argument est plus grand que le montant disponible, la tirelire est alors vidée. La méthode puiser ne retourne rien.

- ☐ une méthode **calculerSolde** qui retourne la différence entre le montant de la tirelire et le budget que l'on souhaite dépenser (un double). Si le budget est négatif (ou nul), la méthode **calculerSolde** doit retourner le montant de la tirelire.

Ces méthodes feront partie de l'interface d'utilisation de la classe.

Exemple d'exécution

Vous êtes sans le sou.
Vous êtes sans le sou.
Bing bing
Vous avez : 550.0 euros dans votre tirelire.
Vous avez : 535.0 euros dans votre tirelire.
Donnez le budget de vos vacances : 450
Vous êtes assez riche pour partir en vacances ! il vous restera 85.0 euros à la rentrée

ou

Vous etes sans le sou.
Vous êtes sans le sou.
Bing bing Vous avez : 550.0 euros dans votre tirelire.
Vous avez : 535.0 euros dans votre tirelire.
Donnez le budget de vos vacances : 1250.0
Il vous manque 715.0 euros pour partir en vacances !

TP 11

Le but de cet exercice est de simuler de façon très basique la gestion d'une bibliothèque. La bibliothèque contient des exemplaires d'œuvres écrites par des auteurs.
Il s'agira de modéliser chacun de ces éléments dans votre programme.

Le code à fournir doit pouvoir **créer des auteurs**, **des œuvres de ces auteurs**, **stocke dans la bibliothèque des exemplaires de ces œuvres**, puis :

- ☐ liste tous les exemplaires de la bibliothèque ;

- ☐ liste tous les exemplaires écrits en anglais ;
- ☐ affiche le nom de tous les auteurs à succès ayant écrit une œuvre dont la bibliothèque stocke un exemplaire ;
- ☐ et affiche le nombre d'exemplaires d'une œuvre donnée ;

Les définitions des classes Auteur, Oeuvre, Exemplaire et Bibliothèque, décrites ci-dessous, devront être fournies.

La classe Auteur
Un auteur est caractérisé par **son nom (une String)** ainsi qu'une **indication permettant de savoir s'il a été primé**.

Les méthodes qui sont spécifiques à cette classe et font partie de son interface d'utilisation sont :

- ☐ des **constructeurs** conformes à la méthode main fournie, avec l'ordre suivant pour les paramètres : le nom et l'indication permettant de savoir si l'auteur a été primé ;
- ☐ une méthode **getNom** retournant le nom de l'auteur ;
- ☐ une méthode getPrix retournant true si l'auteur a été primé.

La classe Oeuvre
Une Oeuvre est caractérisée **par son titre (de type String)**, **(une référence constante à) l'auteur qui l'a rédigée** et la **langue** dans laquelle elle a été rédigée (de type String).

Les méthodes qui sont spécifiques à cette classe et font partie de son interface d'utilisation sont :

- ☐ des **constructeurs** conformes à la méthode main fournie, avec l'ordre suivant pour les paramètres : le titre, l'auteur et la langue. Si la langue n'est pas fournie elle vaudra par défaut "francais" ;
- ☐ une méthode **getTitre** retournant le titre de l'œuvre ;
- ☐ une méthode **getAuteur** retournant l'auteur (il est toléré ici de retourner directement la référence à l'auteur) ;

- ☐ une méthode **getLangue** retournant la langue de l'œuvre ;
- ☐ et une méthode **afficher** affichant les caractéristiques de l'œuvre en respectant strictement le format suivant : <titre>, <nom de l'auteur>, en <langue> où <titre> est à remplacer par le titre de l'œuvre, <nom de l'auteur>, par le nom de son auteur et <langue> par sa langue ;

La classe Exemplaire

La classe **Exemplaire** modélise un exemplaire d'une œuvre. Une instance de cette classe est caractérisée par (une référence à) l'œuvre dont elle constitue un exemplaire.

Les méthodes spécifiques à la classe **Exemplaire** et qui doivent faire partie de son interface d'utilisation sont :

- ☐ un **constructeur** prenant en argument une référence à une œuvre et affichant un message respectant strictement le format suivant : Nouvel exemplaire -> <titre>, <nom de l'auteur>, en <langue> suivi d'un saut de ligne ;

- ☐ un **constructeur** de copie affichant un message respectant strictement le format suivant : Copie d'un exemplaire de -> <titre>, <nom de l'auteur>, en <langue> suivi d'un saut de ligne ;
- ☐ une méthode **getOeuvre** retournant l'œuvre ;
- ☐ et une méthode **afficher** affichant une description de l'exemplaire respectant strictement le format suivant :Un exemplaire de <titre>, <nom de l'auteur>, en <langue>sans saut de ligne. La méthode d'affichage de la classe Oeuvre sera utilisée pour réaliser cet affichage.

La classe Bibliotheque

Une bibliothèque est caractérisée par un nom et contient une liste d'exemplaires.

La liste des exemplaires sera modélisée au moyen d'un tableau dynamique (ArrayList). Cet attribut devra obligatoirement s'appeler exemplaires.
Les méthodes spécifiques à la classe Bibliotheque et qui font partie de son interface d'utilisation sont :

- ☐ un **constructeur** conforme à la méthode main fournie et affichant le message :
  La bibliothèque <nom> est ouverte ! suivi d'un saut de ligne, où <nom> est à remplacer par le nom de la bibliothèque ;

- ☐ une méthode **getNom** retournant le nom de la bibliothèque ;

- ☐ une méthode **getNbExemplaires** retournant le nombre d'exemplaires contenus dans la liste ;

- ☐ une méthode **stocker** permettant d'ajouter un ou plusieurs exemplaires d'une œuvre dans la bibliothèque ; elle doit être conforme au main fourni, avec l'ordre suivant des paramètres : la référence à une œuvre et le nombre n d'exemplaires à ajouter ; cette méthode va ajouter à la liste d'exemplaires de la bibliothèque n exemplaires de l'œuvre fournie, qu'il s'agit de construire. Si le nombre d'exemplaires n'est pas fourni, cela signifie que sa valeur par défaut est 1. **Attention, les exemplaires devront impérativement être ajoutés à la fin du tableau dynamique (méthode add des ArrayList)** ;

- ☐ une méthode **listerExemplaires** retournant dans un ArrayList tous les exemplaires d'une œuvre écrite dans une langue donnée ; si aucune langue n'est donnée (chaîne vide), tous les exemplaires de la bibliothèque seront retournés ; Une méthode utilitaire est fournie qui permet ensuite d'afficher le contenu du tableau dynamique retourné par listerExemplaires (voir l'exemple de déroulement fourni plus bas) ;

- ☐ une méthode **compterExemplaires** retournant le nombre d'exemplaires d'une œuvre donnée passée en paramètre ;

- ☐ une méthode **afficherAuteur** avec un paramètre de type booléen ou sans paramètre, qui affiche les noms des auteurs dont un exemplaire est stocké dans la bibliothèque. Si le booléen est fourni et qu'il vaut true, seuls s'afficheront les noms des auteurs avec un prix ; s'il vaut false seuls les auteurs sans prix s'afficheront. Si le booléen n'est pas fourni, la méthode naffichera que les noms des auteurs à prix. Les noms apparaissent autant de fois qu'il y a d'exemplaires d'œuvres écrites par l'auteur. Un saut de ligne sera fait après l'affichage de chaque nom ;

Exemple d'exécution

```
La bibliotheque municipale est ouverte !
Nouvel exemplaire -> Les Miserables, Victor Hugo, en francais
Nouvel exemplaire -> Les Miserables, Victor Hugo, en francais
Nouvel exemplaire -> L'Homme qui rit, Victor Hugo, en francais
Nouvel exemplaire -> Le Comte de Monte-Cristo, Alexandre Dumas, en francais
Nouvel exemplaire -> Le Comte de Monte-Cristo, Alexandre Dumas, en francais
Nouvel exemplaire -> Le Comte de Monte-Cristo, Alexandre Dumas, en francais
Nouvel exemplaire -> Zazie dans le metro, Raymond Queneau, en francais
Nouvel exemplaire -> The count of Monte-Cristo, Alexandre Dumas, en anglais
La bibliotheque municipale offre
Un exemplaire de Les Miserables, Victor Hugo, en francais
Un exemplaire de Les Miserables, Victor Hugo, en francais
Un exemplaire de L'Homme qui rit, Victor Hugo, en francais
```

Un exemplaire de Le Comte de Monte-Cristo, Alexandre Dumas, en francais
Un exemplaire de Le Comte de Monte-Cristo, Alexandre Dumas, en francais
Un exemplaire de Le Comte de Monte-Cristo, Alexandre Dumas, en francais
Un exemplaire de Zazie dans le metro, Raymond Queneau, en francais
Un exemplaire de The count of Monte-Cristo, Alexandre Dumas, en anglais
Les exemplaires en anglais sont
Un exemplaire de The count of Monte-Cristo, Alexandre Dumas, en anglais
Les auteurs a succes sont
Raymond Queneau
Il y a 3 exemplaires de Le Comte de Monte-Cristo

TP 12

Pour ce TP, vous devez faire des recherches sur les constructeurs de copie :
https://waytolearnx.com/2020/03/constructeur-de-copie-en-java.html#:~:text=Comme%20C%2B%2B%2C%20Java,n'%C3%A9crivez%20pas%20le%20v%C3%B4tre.

Le votre va beaucoup ressembler à celui-ci

```
Public VotreConstructeur(Souris s){
        this.attribut1 = s.attribut1;
        this.attribut2 = s.attribut2;
        ...
}
```

Le but de cet exercice est de créer des « souris » par différents biais et de les faire « évoluer » au cours du temps.

Le corps de la **classe Souris** manque et c'est ce qu'il vous est demandé d'écrire.
Une souris est caractérisée par **son poids en grammes (un int)**, **sa couleur (une String)**, **son âge en mois (un int)**, son **espérance de vie (un int)** et **une indication sur le fait qu'elle soit clonée ou pas (un booléen)**.
Ces attributs seront nommés respectivement : **poids**, **age**, couleur, **esperanceVie** et **clonee**.

Par ailleurs, les méthodes publiques de la **classe Souris** sont :

- ☐ des **constructeurs** conformes à la méthode main fournie, avec l'ordre suivant pour les paramètres : **le poids**, **la couleur**, **l'âge** et **l'espérance de vie**. Ces deux derniers paramètres ont pour valeur par défaut zéro et 36 respectivement. La valeur 36 est stockée dans une constante fournie que vous utiliserez en écrivant **ESPERANCE_VIE_DEFAUT**. Les constructeurs afficheront le message Une nouvelle souris ! ;

- ☐ un **constructeur de copie** qui doit afficher le message Clonage d'une souris ! ;
  une souris clonée a les mêmes caractéristiques que la souris d'origine, sauf son espérance de vie qui est moindre : les **4 cinquième de celle de la souris d'origine** ;

- ☐ une méthode **toString()** produisant une représentation d'une Souris respectant strictement le format suivant : Une souris <couleur>[, clonée,] de <age> mois et pesant <poids> grammes (sur une seule ligne) où <age> est à remplacer par l'âge de la souris et <poids> par son poids. Le bout de phrase « , clonée, » ne sera affiché que si la souris a été clonée ;

- ☐ une méthode **vieillir** qui augmentera d'une unité l'âge de la souris. **Si la souris est clonée, elle doit devenir verte si elle atteint un âge supérieur à la moitié de son espérance de vie** ; **même si elle**

n’est pas appelée explicitement dans la méthode main(), cette méthode doit être publique ; elle sera testée ;

- ☐ et une méthode **évolue** faisant vieillir la souris depuis son âge courant jusqu’à son espérance de vie.

Tous les affichages demandés se feront sur le terminal et seront terminés par un saut de ligne.

Exemple d'exécution

```
Une nouvelle souris !
Une nouvelle souris !
Clonage d’une souris !
Une souris blanche de 2 mois et pesant 50 grammes
Une souris grise de 0 mois et pesant 45 grammes
Une souris grise, clonée, de 0 mois et pesant 45 grammes
Une souris blanche de 36 mois et pesant 50 grammes
Une souris grise de 36 mois et pesant 45 grammes
Une souris verte, clonée, de 28 mois et pesant 45 grammes
```