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

Ecrire 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"
Resultat : Je Suis Dans La Joie

TP 3

Ecrire 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

TP 4

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 5

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 5

Écrire une méthode qui devra avoir comme input un **tableau** et la **taille du morceau**, divisez le tableau en plusieurs sous tableaux ou morceaux.

Exigences
Chaque sous-réseau a une longueur égale à la taille du morceau passé en paramètre.

Exemples
maMthode([1, 2, 3, 4], 2) --> [[1, 2] , [3, 4]]
maMthode([1, 2, 3, 4, 5], 2) --> [[1, 2], [3, 4], [5]]
maMthode([1, 2, 3, 4, 5, 6, 7, 8], 3) --> [[1, 2, 3], [4, 5, 6], [7, 8]]
maMthode([1, 2, 3, 4, 5], 4) --> [[1, 2, 3, 4], [5]]
maMthode([1, 2, 3, 4, 5], 10) --> [[1, 2, 3, 4, 5]]

TP 6

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 7
Écrire une méthode Java qui prend en paramètre un tableau d'entiers et un entier qui représente une somme nommée **S**.

Retourner un tableau contenant tous les couples des deux nombres tels que leur somme soit égale à **S**.

Vous pouvez supposer que chaque entrée a exactement une solution, et vous ne pouvez pas utiliser deux fois le même élément.

Exemple 1 :

Entrée : nombres = [2,7,11,15], **S** = 9
Sortie : [2,7]
Sortie : Parce que **(2)nums[0] + (7)nums[1]** == **9**, nous retournons [2, 7].
Exemple 2 :

Entrée : nums = [3,2,4,7,5,-1], **S** = 6
Sortie : [[2,4] , [7,-1]]

TP 8

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 9

Soit à développer une application pour la gestion d'un stock.

Un article est caractérisé par son numéro de référence, son nom, son prix de vente et une quantité en stock.

Le stock est représenté par une collection d'articles.

Travail à faire:

Créer la classe article contenant les éléments suivants :

- Les attributs/propriétés.
- Un constructeur d'initialisation.
- La méthode ToString().

Dans la classe Program créer :

Le stock sous forme d'une collection d'articles de votre choix.

Un menu présentant les fonctionnalités suivantes :

- Rechercher un article par référence.
- Ajouter un article au stock en vérifiant l'unicité de la référence.
- Supprimer un article par référence.
- Modifier un article par référence.
- Rechercher un article par nom.
- Rechercher un article par intervalle de prix de vente.
- Afficher tous les articles.

- Quitter

TP 11

Énoncé:
Un parc auto se compose de voitures et des camions qui ont des caractéristiques communes regroupées dans la classe Véhicule.

- Chaque véhicule est caractérisé par son matricule, l'année de son modèle, son prix.

- Lors de la création d'un véhicule, son matricule est incrémenté selon le nombre de véhicules créés.

- Tous les attributs de la classe véhicule sont supposés privés. ce qui oblige la création des accesseurs (get...) et des mutateurs (set....) ou les propriétés.

- La classe Véhicule possède également deux méthodes abstraites démarrer() et accélérer() qui seront définies dans les classes dérivées et qui affichent des messages personnalisés.

- La méthode ToString() de la classe Véhicule retourne une chaîne de caractères qui contient les valeurs du matricule, de l'année du modèle et du prix.

- Les classes Voiture et Camion étendent la classe Véhicule en définissant concrètement les méthodes accélérer() et démarrer() en affichant des messages personnalisés.

Travail à faire:
- Créer la classe abstraite Véhicule.
- Créer les classes Camion et Voiture.

- Créer une classe Test qui permet de tester la classe Voiture et la classe Camion

TP 10

Énonce:
Ecrivez une classe Bâtiment avec les attributs suivants:

- adresse

La classe Bâtiment doit disposer des constructeurs suivants:

- Batiment(),
- Batiment (adresse).

La classe Bâtiment doit contenir des accesseurs et mutateurs pour les différents attributs. La classe Bâtiment doit contenir une méthode **ToString ()** donnant une représentation du Bâtiment.

Ecrivez une classe Maison héritant de Bâtiment avec les attributs suivants:

- NbPieces: Le nombre de pièces de la maison.

La classe Maison doit disposer des constructeurs suivants:

- Maison(),
- Maison(adresse, nbPieces).

La classe Maison doit contenir des accesseurs et mutateurs (ou des propriétés) pour les différents attributs. La classe Maison doit contenir une méthode ToString () donnant une représentation de la Maison.

Écrivez aussi un programme afin de tester ces deux classes .

TP 11

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 12

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