Exercices JAVA

S.Gounane

Exercice1

1- On passe un argument à la méthode main. La méthode main affiche cet argument. Supposons que la méthode main soit écrite dans la classe AfficheArguments.

Pour l'exécution, Utilisez deux méthodes:

- Dans un terminal
- Sous Eclipse

2- Cas de plusieurs arguments

Exercice2

Ecrire un programme qui calcule la factorielle d'un entier ***n*** et indique à l'écran le résultat. Le nombre ***n*** doit être lu sur la ligne de commande.

Indication:

Utiliser la méthode static parseInt() de la classe Integer pour transformer une chaine de caractère en entier.

Exercice3

1- créez un nouveau projet sous Eclipse que vous nommez par exemple TP1A
2- créez un nouveau paquetage dans le projet TP1A que vous nommez etudiant
3- créez une nouvelle classe dans le paquetage etudiant que vous nommez Etudiant
La classe Etudiant a :

- un attribut privé de type String nommé nom ;
- un constructeur publique qui a un paramètre de type String servant à initiliser le nom de l'étudiant ;
- une méthode publique sans paramètre et qui ne renvoie rien, nommée travailler, qui écrit à l'écran, si le nom de l'étudiant a pour nom toto :
- toto se met au travail !
- une méthode publique sans paramètre et qui ne renvoie rien, nommée seReposer, qui écrit à l'écran : **toto** se repose

4- Ensuite créer une classe TestEtudiant (dans le paquetage etudiant) contenant une méthode main qui :

- crée un étudiant (instance de la classe Etudiant);
- invoque la méthode travailler de l'étudiant créé ;
- invoque la méthode seReposer de l'étudiant créé .

5- Le de l'étudiant comme argument

Exercice4

Il s'agit de modéliser un segment de droite dont les valeurs des abscisses des deux extrémités sont entières. Les opérations que l'on souhaite faire sur ce segment sont :

- calculer sa longueur ;
- savoir si un point d'abscisse donné se trouve sur le segment (c'est-à-dire si son abscisse est comprise entre la plus petite et la plus grande valeurs des abscisses des extrémités du segment).

1- Écrire classe publique Segment se trouvant dans un paquetage segment comportant :

deux attributs privés de type **int**, extr1 < extr2

Les methodes:

- **private void** ordonne(int , int); //échange les valeurs des extrémités
- **public boolean** appartient(**int** x);
- **public int** longueur()
- Les getter et les setter
- **public** String toString() (example: "segment [-35, 44]")

2- Vous définirez aussi dans le paquetage segment une classe TestSegment pour tester la classe Segment.

3- Passez trois argument au main : ext1 ext2 et x

Execution avec les parametres -35 44 8:

Longueur du segment [-35, 44] : 79

8 appartient au segment [-35, 44]

Exercice5

Un élève sera modélisé par la classe Eleve du paquetage gestionEleves. Cette classe posséde trois attributs privés :

- son nom type String,
- un ensemble de notes dans un ArrayList<Integer>
- une moyenne de type **double**. Un élève sans aucune note sera considéré comme ayant une moyenne nulle.

La classe Eleve possède un constructeur permettant uniquement d'initialiser le nom de l'élève et possède:

- **public double** getMoyenne()
- **public void** ajouterNote(**int** note)
- **public** String getNom()
- **public** ArrayList<Integer> getListeNotes()
- **public String** toString() //retourne le nom les notes et la moyenne

Exercice6

Un groupe d'Eleve(s) sera modélisé par la classe GroupeEleves du paquetage gestionEleves de la façon suivante.
La classe GroupeEleves possède un attribut privé : une collection d'Eleve(s) nommée listeEleves, de type ArrayList<Eleve>.
La classe GroupeEleves ne possède pas de constructeur explicite.
La classe GroupeEleves possède aussi cinq méthodes publiques :

- **public int** nombre() //renvoie le nombre d'Eleve(s) contenus dans listeEleves
- **public** ArrayList<Eleve> getListe() //renvoie listeEleves.
- **public void** ajouterEleve(Eleve eleve) //ajoute l'Eleve reçu en paramètre à listeEleves.
- **public** Eleve chercher(String nom) //renvoie l'Eleve dont le nom est indiqué par le paramètre
- **public void** lister() //écrit à l'écran la liste des Eleve(s). Elle utilise une ligne par Eleve ; elle utilise la méthode toString de la classe Eleve.

Exercice7

Complétez la classe Eleve pour faire en sorte d'avoir une classe qui implémente l'interface java.lang.comparable<T>. C'est une interface générique, comme l'indique le <T>.

L'interface Comparable<T> déclare une seule méthode : **public int** compareTo(T o);

Quand cette méthode est implémentée, elle doit retourner une valeur strictement négative, nulle ou strictement positive selon que l'objet concerné est plus petit que l'objet o, égal à l'objet o ou plus grand que l'objet o. (On comparera les élèves selon leur moyenne)

1. Reprendre la classe Eleve écrite précédemment pour la transformer en une classe qui implémente l'interface java.lang.Comparable<Eleve>.
2. Définir la méthode compareTo déclarée par l'interface Comparable. Cette méthode est à nouveau générique ; si la classe implémente Comparable<Eleve>, le paramètre de la méthode doit être de type Eleve : **public int** compareTo(Eleve autreEleve)
3. Modifiez la méthode main de la classe TestEleve pour tester la méthode compareTo.

Exercice8

Il s'agit de modéliser un groupe d'élèves comparables entre eux selon leurs moyennes. On souhaite ajouter à la classe ***GroupeEleves*** deux méthodes :

- une méthode, nommée ***meilleurEleve***, qui retourne l'élève de meilleure moyenne de la liste listeEleves ;
- une méthode, nommée ***trierEleves***, qui trie la liste listeEleves selon l'ordre croissant des moyennes des élèves.

Note:

Utiliser *java.util.Collections.max(listeEleves)*

Et

java.util.Collections.sort(listeEleves)

Exercice9

On définit quatre classes.

- La classe **Animal** est abstraite et déclare uniquement une méthode abstraite nommée ***action***, sans paramètre et qui ne retourne rien.
- La classe **Chien** hérite de Animal et définit la méthode action qui écrit à l'écran "J'aboie".
- La classe **Chat** hérite de Animal et définit la méthode action qui écrit à l'écran "Je miaule".
- La classe **EssaiChat** contient trois champs statiques :
  - un champ statique pour un attribut de type ***java.util.Random*** qui est initialisé dès sa définition
  - une méthode statique nommée ***tirage*** sans paramètre qui retourne un Animal qui a une chance sur deux d'être un Chat et une chance sur deux d'être un Chien.
  - une méthode main qui utilise la méthode tirage et invoque la méthode action sur l'animal obtenu par cette méthode.

Exercice10

On souhaite définir un ensemble de classes pour "modéliser" des oiseaux. On veut munir chaque ***Oiseau*** d'une méthode nommée ***decrire()***. On veut pouvoir mettre un mélange d'"***Oiseaux***" dans un même tableau puis, dans une boucle, pouvoir appliquer successivement la méthode ***decrire()*** à tous les "***Oiseaux***" du tableau.

Exemple d'execution:

Famille des oiseaux : je suis un merle
Famille des oiseaux : je suis une pie
Famille des oiseaux : je suis une pie
Famille des oiseaux : je suis un merle
Famille des oiseaux : je suis une pie