**Les tableaux et les collections**

Tableau unidimensionnel

Les tableaux Sont des regroupements ordonné de données du même type

Un tableau est caractérisé par son type et sa taille

int[] tablInt=new int[5];  
char[] tablChar=new char[5];  
Voiture[] tablVoiture=new Voiture[5];  
  
Voiture peugeot=new Voiture();  
peugeot.nomVoiture="voiture1";  
peugeot.nbrPort=2;  
  
tablInt[0]=15;  
tablInt[4]=17;  
tablVoiture[0]=peugeot;  
tablVoiture[1]=new Voiture("reunot",4);  
 System.*out*.println("le premier eleme du tableau est : "+tablInt[0]);  
 System.*out*.println("le nom de la voiture N°1 "+peugeot.nomVoiture);  
 System.*out*.println("le nom de la voiture N°2 "+tablVoiture[1].nomVoiture);  
 System.*out*.println("la taille du dableau de voistures est : "+tablVoiture.length); pour avoir la taille d’un tableau

**Résulta**

le premier eleme du tableau est : 15

le nom de la voiture N°1 voiture1

le nom de la voiture N°2 reunot

la taille du dableau de voistures est : 5

lenght : permet de donner la taille d’un tableau

Exercice

Crée une méthode dans la classe planeteTelerique pour verifier la disponibilité de places

public class PlaneteTellurique extends Planete implements Habitable{

Vaisseau[] bailAccoste;

boolean restePlaceDispo(){  
 for (int index=0;index<bailAccoste.length;index++){  
 if (bailAccoste[index] == null){  
 return true;  
 }  
 }  
 return false;  
}

**Les elips ou varags**

Dans le cas qu’on crée une methode ou constructeur qui a un argument qu’on sait pas combien de fois on vas l’ulisliser exp :

Constructeur voiture qui recoit en argument un le nom de voiture est les villes qu’il vas visiter

On sait pas combien de ville dans ce cas on crée un argument d’une forme de table

Ville transporter(Passager passagera,Ville villeDeppart,Ville... villeEtapes){   
System.*out*.println("la première vile etape est : "+villeEtapes[0].nomVille);

ici on donne ville de départ ensuite n’import quels nombre des autre villes étape

public class origin {  
 public static void main(String[] args){  
  
 Voiture peugeot=new Voiture();  
 peugeot.typB=TypeBoitVitesse.*auto*;  
 Passager passager1=new Passager();  
 passager1.nomPassager="reda";  
 Ville depart=new Ville();  
 depart.nomVille="rabat";  
 Ville etape1=new Ville();  
 etape1.nomVille="casablanca";  
 Ville etape2=new Ville();  
 etape2.nomVille="gladbach";  
 Ville etape3=new Ville();  
 etape3.nomVille="NewYork";  
  
 peugeot.transporter(passager1,depart,etape1,etape2,etape3);

}  
}

**La mèthode main**

La classe qui contient la methode main est l’acces principale du programme c’est une classe principale executable qui peut contenir des attribue des méthodes et des constructeurs

On peut declarer la méthode main dans une autre classe 🡪 la classe deviendra executable.

NB : on peut changer la classe d’exécution par défaut dans les propriétés du projet courant.

**Tableau Mulidentionnel**

Pour manupuler le tableau il faut deux boucle imbriquer

Exp

int[][] tableauMilti=new int[5][3];  
for (int col=0;col<tableauMilti.length;col++){  
 for (int ligne=0;ligne<tableauMilti[col].length;ligne++){  
 System.*out*.println("l'element contenu à la position "+col+" , "+ligne+"vaut "+tableauMilti[col][ligne]);  
 }  
}

resulta

l'element contenu à la position 0 , 0vaut 0

l'element contenu à la position 0 , 1vaut 0

l'element contenu à la position 0 , 2vaut 0

l'element contenu à la position 1 , 0vaut 0

l'element contenu à la position 1 , 1vaut 0

// // // // // // // // // // // // // // // // //

Exercice 35 a refaire

**Les collections**

Introduction

Une collection est un ensemble d’éléments sous forme d’objets, on peut présenter une collection comme un tableau unidimensionnel à la difference les collections ne peuvent contenir des types premitif.

Avantage : on n’indique pas sa taille il’est dynamique

On java on peut stocké des collections d’objet grace a différente sorte de classes ces classes sont regroupé sous 4 grand type de collections(list, Set, Map, Queue)

Map

**Collection**

Set

SortedMap

Queue

List

SortedSet

Les classe Set List et Queue Hérite de la classe collection

Dans une collection on peut ajouter des éléments ou les supprimer les ordonnées …

List

Les listes sont des collections qui peuvent contenir plusieurs fois le meme élément chacun de ces element est numeroté c’est l’index

List est une interface, pour interfacer un objet sous forme de list on appel une classe qui implémente list , les plus courronte (ArrayList et LinkList)

ArrayList est la classe la plus rapide en générale

Link plus rapide qu’on effectue des insertion au debut de liste ou suppression d’element pendent une itération

Exemple ArrayList

List list= new ArrayList();creation arraylist  
list.add(peugeot); index 0 va avoir la voiture peugo comme valeur  
list.add(3);  
list.add(peugeot); on peut affecter le meme objet à deux index different

list.remove(peugeot); ici on suoorime peugeot du premier index  
Object o=list.get(0); affectation la valeur de l’index 0 à l’objet o  
System.*out*.println("l'objet à l'index 0 est " +o);

Voiture o2=(Voiture)list.get(0); pour affecter la valeur de l’objet Voiture

System.*out*.println("l'objet à l'index 0 est " + o2.nom); afficher le nom de la voiture

A partir de JAVA 5 on peut preciser ce qui va deposer dans une collectiongrace a l’operateur diamant <>

Dans lequel on indique le type qu’on veut utiliser EXP :

List<Voiture> list= new ArrayList(); ici on a indiquer le type qu’on vas stocker  
list.add(peugeot);  
list.add(peugeot2);  
Voiture o=list.get(0);  
Voiture o2=list.get(1);  
System.*out*.println("le nom de la voiture de l'index 1 est " + o.nom);  
System.*out*.println("le nom de la voiture de l'index 2 est " + o2.nom);

Set

Que l’on peut traduire par (des ensembles) sont descollections qui ne peuvent pas contenir 2 fois le meme élément

Set<Voiture> set= new HashSet();  
set.add(peugeot);  
set.add(peugeot2);  
set.add(peugeot); cette ligne est ignorer par java car l’objet peugeot existe dans l’index 0  
System.*out*.println("le nombre d'index dans le set est : "+set.size()); Résulta🡪 2

**NB :** L’ordre des index dont le set stock des éléments est imprévisible

Foreach : Itérer sur une collection

Foreach est très couramment effectuer sur une collection, c’est de parcourir sont contenu

for (Voiture voiture:set) Set<Voiture> set= new HashSet();  
set.add(peugeot);  
set.add(peugeot2);  
set.add(peugeot);  
System.*out*.println("le nombre d'index dans le set est : "+set.size());

}

for (Voiture voiture:set) {  
 System.*out*.println("le nom de voiture "+voiture.nom);  
  
}

resulta

le nom de voiture BMW

le nom de voiture mercedes

{ entre les parenthese a droite des points la collection qu’on va parcourir et à gauche des 2 points la variable qui contenir les élément de collection

NB : on peut pas supprimer un élément de la collection en parcourant la collection

for (Voiture voiture:set) {  
 System.*out*.println("le nom de voiture "+voiture.nom);  
 set.remove(voiture) 🡪 message d’erreur  
}

Itération

Il y a une deuxieme manier pour parcourir une collection c’est d’utiliser un iterateur (interface)

Iterator<Voiture> it=set.iterator();  
while (it.hasNext()) {  
 Voiture v = it.next();  
 System.*out*.println(v.nom);  
} Resulta Voiture1 Voiture2 …

NB : ArrayList vs HashSet. HashSet ne permet pas de stocker les doublons

**Map**

Est une collection qui référence chaque élément par une clé, on ne peut pas insert deux fois la même clé dans une map mais on peut insérer deux fois la même valeur l’implémentation la plus courant pour Map est HashMap.

Exemple d’utilisation

Map<String,Voiture> map=new HashMap();  
map.put("206",peugeot);// prend en arguments objet et valeur  
map.put("207",peugeot2);  
Voiture v=map.get("207");  
System.*out*.println(v.nom);  
// parcourir une Map  
for (Map.Entry<String,Voiture> entry:map.entrySet()) {  
 String key=entry.getKey();  
 Voiture value=entry.getValue();  
 System.*out*.println("l'lement suisvant à pour clé : "+key+" il s'agit de : "+value.nom);

**keySet()** a la place de **entrySet()** le cas de boisin d’utiliser juste les clés de la collection Map

for (String key :map.keySet()) {  
 System.*out*.println("la liste des clés : "+key);  
 }

Résulta

la liste des clés : 206

la liste des clés : 207

**values()** a la place de **entrySet()** le cas de boisin d’utiliser juste les valeur des objets non pas les clés de la collection Map

for (Voiture value :map.values()) {  
 System.*out*.println("parmi les voitures : "+value.nom);  
 }

Résulta

parmi les voitures : mercedes

parmi les voitures : BMW

exempled’utilisation

public class Atmosphere {  
 Float tauxHelium;  
 Float tauxHydrogene;  
 Float tauxAzote;  
 Float tauxArgon;  
 Float tauxDioxydeDeCarbone;  
 Float tauxSodium;  
 Float tauxMethane;  
}

à la place d’écrire le code au dessus on vas utilise le Map et l’objet HashMap

public class Atmosphere {

Map<String,Float> constituants=new HashMap();

}

Et dans le main on fait l’appel comme suit

Atmosphere atmostherUranus=new Atmosphere();  
atmostherUranus.constituants.put("CO2",95.3f);  
atmostherUranus.constituants.put("H2O",2.7f);  
  
uranus.atmosphere=atmostherUranus;  
System.*out*.println("l'atmosphere de uranus est : ");  
for (Map.Entry<String,Float> constituant: uranus.atmosphere.constituants.entrySet()){  
 System.*out*.println(constituant.getValue()+ " % de "+constituant.getKey());

**Résulta**

l'atmosphere de uranus est :

2.7 % de H2O

95.3 % de CO2

**Collection et collections**

size : permet d’avoir combien d’éléments se constitue la collection

isEmpty : pour voir si la collection est vide ou non. Return true or false

containsKey : pour voir si un clé donnée existe. Return true or false

System.*out*.println("207 existe ? "+map.containsKey("207"));

Résulta

207 existe ? true

containValue : pour voir si une valeur existe

map.clear() ; pour vider la map

java dispose d’une classe collections permettant de faire nombreuse operation classic sur les collections par le biais de méthode static

exemple de fonction replaceAll qui permet de remplcaer une valeur rechercher par une autre donné

List<String> list1=new ArrayList<>();  
list1.add("one");  
list1.add("two");  
list1.add("tree");  
Collections.*replaceAll*(list1,"one","four");  
for (String string : list1){  
 System.*out*.println(string);  
}

Résulta

four

two

tree

NB : replaceAll permettant de remplacer meme les objets

Collections.*replaceAll*(list1,"one",4)  
for (Object o : list1){  
 System.*out*.println(o);  
}

Résulta

4

two

tree

<https://docs.oracle.com/javase/8/docs/api/java/util/Comparator.html> définition des méthodes et des fonctions

**Trier les collection par l’ordre naturel de ces éléments.**

**sort :** permet de trier une collection exemple d’utilisation

List<Integer> list1=new ArrayList<>();  
list1.add(9);  
list1.add(2);  
list1.add(5);  
Collections.*sort*(list1);  
  
for (Integer entier : list1){  
 System.*out*.println(entier);  
}

Résulta

2

5

9

Ce trie est possible car les entier dispose d’un ordre naturel, une classe qui dispose d’un ordre naturel est une classe qui implement l’interface **Comparable, et pour** comparer deux objet qui dispose pas de l’interface comparable il faut l’ajouter au classe est configurer la méthode comarTo pour donnée le critère de comparaison

Exemple de d’utilisation de la classe **Comparable**

public class Caree implements Comparable{  
  
 long cote;  
  
 Caree (long cote){  
 this.cote=cote;  
 }  
  
  
 @Override  
 public int compareTo(Object o) {  
 Caree autrecaree=(Caree) o;  
 if (cote==autrecaree.cote)return 0;  
 if (cote>autrecaree.cote)return 1;  
 return -1;  
 }  
}

dans la classe main

List<Caree> autrecar=new ArrayList<>();  
autrecar.add(new Caree(7));  
autrecar.add(new Caree(4));  
autrecar.add(new Caree(8));  
autrecar.add(new Caree(2));  
  
Collections.*sort*(autrecar);  
  
for (Caree caree : autrecar){  
 System.*out*.println(caree.cote);  
}

**Résulta**

2

4

7

8

Si on change le type pérémitif long par la classe conteneur Long on vas economiser le code en utilisant la methode de comparaison a la place de faire la comparaison nous meme

public class Caree implements Comparable{  
 Long cote; ici on a changer le type de prémitif long au Long  
 Caree (long cote){  
 this.cote=cote;  
 }  
 @Override  
 public int compareTo(Object o) {  
 Caree autrecaree=(Caree) o;  
 return this.cote.compareTo(autrecaree.cote); la comparaison devient plus simple  
 }  
}

**les classes (TreeSet et TreeMap)**

**TreeSet** est une classe qui implemente set et **TreeMap** qui implemente map peuvent etre ordoné d’une façon natif, chaque ajout d’un élément va automatiquement se placer par ordre

Exemple utilisation TreeSet

Set<Caree> setCaree=new TreeSet<>();  
setCaree.add(new Caree(7));  
setCaree.add(new Caree(4));  
setCaree.add(new Caree(8));  
setCaree.add(new Caree(2));  
  
for (Caree caree : setCaree){  
 System.*out*.println(caree.cote);  
}

Résulta

2

4

7

8

NB : on a pas utiliser la **Collections.sort** pour faire le trie

Exemple d’utilisation TreeMap

Map<Caree,Voiture> mapCaree=new TreeMap();  
mapCaree.put(new Caree(7),peugeot);  
mapCaree.put(new Caree(4),peugeot2);  
mapCaree.put(new Caree(8),peugeot2);  
mapCaree.put(new Caree(2),peugeot);  
  
  
  
for (Map.Entry<Caree,Voiture> entryM: mapCaree.entrySet()){  
 System.*out*.println(entryM.getKey().cote+" "+entryM.getValue().nom);  
}

Résulta

2 mercedes

4 BMW

7 mercedes

8 BMW

**Exercice 39 à refaire**

**L’equivalence de l'objets : equals() et HashCode()**

Equals est utilisé a la collection lord ce qu’on ajoute un élément à la collection pour savoir si cette élément existe déjà

@Override  
public boolean equals(Object obj) {  
 if (!(obj instanceof Caree)){  
 return false;  
 }  
 Caree autreCaree=(Caree) obj;  
 return cote==autreCaree.cote;  
}

compareTo est ulisé dans les collections triable pour savoir si les objets sont plus petit ou plus grand

@Override  
public int compareTo(Object o) {  
 Caree autrecaree=(Caree) o;  
 return this.cote.compareTo(autrecaree.cote);  
}

HashCode() return un entier cette entier doit être unique chaque instance doit montrer un HashCode different sauf a les considerer equivalente, HashCode est plus rapide que equals

@Override  
public int hashCode() {  
 return cote.hashCode();  
}

Il’est conseiller de redifinir la mèthode equals par une clesse qui implimente comparable

Quiz

Question 1:

**Quel est le type de Collection qui peut contenir un ensemble non ordonné d'Objets mais qui n'accepte pas les doublons.**

SET

Question 2:

**En quoi va être converti l'ensemble des valeurs passées à une méthode recevant un paramètre sous forme d'ellipse ?**

En tableau unidimentionnel

Question 3:

**Qu'est ce que l'ordre naturel dans une collection d'objets ?**

Sont des instance de classe qui implemente comparable

Question 4:

**Dans quel ordre sont les objets placés dans une collection de type List?**

L’ordre d’ajout dans la liste