Sommaire

[Notion générale sur le langage de programmation java 4](#_Toc41009527)

[Définition 4](#_Toc41009528)

[Comment fonctionne JAVA 4](#_Toc41009529)

[Quel avantage apporte le fait que Java soit compilé et interprété ? 4](#_Toc41009530)

[JDK : Java développement kit 5](#_Toc41009531)

[Travailler avec java 6](#_Toc41009532)

[Outils en relation avec java 6](#_Toc41009533)

[Installation java 11 open JDK 7](#_Toc41009534)

[Modifier l’environnement de l’exécution d’un ordinateur JAVA\_HOME JRE\_HOME 7](#_Toc41009535)

[Déclaration des variables primitif commencent par minuscule 8](#_Toc41009536)

[L’operateur d’affectation 8](#_Toc41009537)

[Les opérateurs arithmétiques 9](#_Toc41009538)

[Les opérateurs Relationnels 9](#_Toc41009539)

[Les opérateurs Logiques 9](#_Toc41009540)

[Les opérateurs d’incrémentation / Décrémentation 9](#_Toc41009541)

[L’opérateur de concaténation + 10](#_Toc41009542)

[Les structure de contrôle 10](#_Toc41009543)

[Le bloc d’instruction 10](#_Toc41009544)

[La structure conditionnelle if else 10](#_Toc41009545)

[Résoudre les erreurs : le débuggeurs 11](#_Toc41009546)

[La structure conditionnelle switch - case 11](#_Toc41009547)

[La structure itérative for 12](#_Toc41009548)

[La structure itérative while 12](#_Toc41009549)

[Les classes et les objets 14](#_Toc41009550)

[L’objet ou l’instance 14](#_Toc41009551)

[Les méthodes 15](#_Toc41009552)

[méthode avec return 15](#_Toc41009553)

[Le mot clé this 16](#_Toc41009554)

[Propriété sous forme d’objet 16](#_Toc41009555)

[Les Propriété / attribue static ou variable de classe 18](#_Toc41009556)

[méthode static 19](#_Toc41009557)

[Constructeur sans paramètre et par défaut 20](#_Toc41009558)

[Ajouter constructeur avec parametre 20](#_Toc41009559)

[L’héritage et les interfaces 21](#_Toc41009560)

[Le mot clé extends 21](#_Toc41009561)

[Le transtypage 23](#_Toc41009562)

[L’héritage et constructeur le mot clé super 23](#_Toc41009563)

[Covariance des méthodes 24](#_Toc41009564)

[L’interface : le mot clé implement 25](#_Toc41009565)

[Le Mot clé instanceof 26](#_Toc41009566)

[Méthode et classe abstraite(abstract) 28](#_Toc41009567)

[Classes et techniques utilitaires 28](#_Toc41009568)

[La classe scanner 29](#_Toc41009569)

[Classes conteneur 30](#_Toc41009570)

[Conversions de type(parseInt, parseString …) 31](#_Toc41009571)

[Les types énumérer 31](#_Toc41009572)

[Les tableaux et les collections 33](#_Toc41009573)

[Tableau unidimensionnel 33](#_Toc41009574)

[Les elips ou varags 34](#_Toc41009575)

[La méthode main 34](#_Toc41009576)

[Tableau Multidimensionnel 34](#_Toc41009577)

[Les collections 35](#_Toc41009578)

[Les List 36](#_Toc41009579)

[Les Set (HashSet) 36](#_Toc41009580)

[Foreach : Itérer sur une collection 37](#_Toc41009581)

[Itération 37](#_Toc41009582)

[Map 37](#_Toc41009583)

[Collection et collections 39](#_Toc41009584)

[Trier les collections par ordre naturel de ces éléments. 40](#_Toc41009585)

[les classes (TreeSet et TreeMap) 41](#_Toc41009586)

[L’équivalence de l'objets : equals() et HashCode() 42](#_Toc41009587)

[Organiser et gérer l’accès aux classes 43](#_Toc41009588)

[Notion de package et d’import 43](#_Toc41009589)

[L’import static 44](#_Toc41009590)

[les modificateur d’accès (public, private, protected, package) 44](#_Toc41009591)

[L’encapsulation des propriétés : premiere approche des composants réutilisable 44](#_Toc41009592)

[Tenir compte des scenario alternatifs été des cas d’erreurs 45](#_Toc41009593)

[La classe exception 45](#_Toc41009594)

[le mot clé throws 45](#_Toc41009595)

[les mot clés try catch et  finally 45](#_Toc41009596)

[Cas des exceptions multiple 46](#_Toc41009597)

# Notion générale sur le langage de programmation java

## Définition

Java et langage de programmation orienté objet, c’est l’intermédiaire entre le langage humain et le langage binaire de la machine, il a été inventé dans les années 1990 par (sun microsoft) ensuite ORACLE a acheté les droit de JAVA en 2010.

## Comment fonctionne JAVA

**La COMPULATION,** Au premier lieu java transforme le code java au bite \_code, ensuite **l’INTERPRETATION** la transformation du bite\_code au langage machine(Binaire)

**Le langage C** utilise juste la compulation(c’est de convertir le code au binaire directement)

**Javascripte** fais juste l’interpretaion(c’est de laisser la machine pour convertire le code)

**Lien(Quel version de JAVA choisi)** <https://www.youtube.com/watch?v=OVb7vNaxtYo&t=5s>

JAVA 8 et JAVA 11 sont des LTS (Long Term Support)

La compulation se fait avec javac.exe / sh

L’interprétation se fait avec java.exe / sh

**NB : C plus rapide que JAVA . java plus rapide que javascripte**

**Avantage :** à cause la transformation du code java au bitecode c’est de pourvoir communiquer avec plusieurs systèmes d’exploitation, java donc considérer comme un langage de programmation portable

## Quel avantage apporte le fait que Java soit compilé et interprété ?

Le code compilé et portable et plus rapide à exécuter que du code simplement interprété

Les environements du developpement

Les environnements de développement en ligne dans le cloud tel que (jdoodle, codeanvy, cloude 9)

**Avantage :** ils sont mobile

**Inconvenient :** connexion internet nécessaire, moins de performance, moins de fiabilité

**Eclips :** est l’environnement le plus populaire, gratuit, open source

**Netbean**

open source, il y a 2 forme une gratuite supportait par la fondation APATCH depuis 2018, et une propriétaire supporter par oracle, facile à utiliser par rapport à Eclips .

**Inconvenien Netbean :** la lenteur du système quand le projet devient plus gros, et l’évolution est très long ca provoque le retard par rapport la version de java

**Intellij IDEA :** compatible à JAVA, version professionnel est payante, compatible avec la dernière version de JAVA, i l’est plus puissant que les autres langages

**Lien Pour faire la comparaison entre ces 3 langages** <https://www.javaworld.com/article/3114167/choosing-your-java-ide.html>

* **C4 code java ‘Hello world’**

**System.out.println(‘Hello world’) ; permet d’écrire Hello world sur le console.**

**Site web jdoodle de java pour exécuter le code java sans installer des outils a l’ordinateur**

**NB : java sensible à la case (minuscule et majuscule)**

## JDK : Java développement kit

Le **Java Development Kit** (**JDK**) désigne un ensemble de [bibliothèques logicielles](https://fr.wikipedia.org/wiki/Biblioth%C3%A8que_logicielle) de base du [langage de programmation](https://fr.wikipedia.org/wiki/Langage_de_programmation) [Java](https://fr.wikipedia.org/wiki/Java_(langage)), ainsi que les outils avec lesquels le code Java peut être compilé, transformé en [bytecode](https://fr.wikipedia.org/wiki/Bytecode) destiné à la [machine virtuelle Java](https://fr.wikipedia.org/wiki/Machine_virtuelle_Java).

Les principaux composants du JDK sont une sélection d'outils de programmation, incluant :

* java : le [chargeur](https://fr.wikipedia.org/wiki/Chargeur_(informatique)) d'application Java ;
* javac : le compilateur, qui convertit le code source en [fichier .class](https://fr.wikipedia.org/wiki/Class_(format_de_fichier)) (contenant le [bytecode Java](https://fr.wikipedia.org/wiki/Bytecode_Java)) ;
* appletviewer : cet outil peut être utilisé pour exécuter et déboguer des [applets Java](https://fr.wikipedia.org/wiki/Applet_Java) sans navigateur ;
* apt : l'outil de traitement des annotations ;
* extcheck : un outil détectant les conflits de fichiers JAR ;
* idlj : compilateur IDL vers Java. Cet outil génère les bindings Java d'un fichier Java IDL donné ;
* [javadoc](https://fr.wikipedia.org/wiki/Javadoc) : le générateur de documentation, qui génère automatiquement de la documentation à partir des commentaires du code source ;
* jar : l'archiveur, qui met sous forme d'un paquetage unique l'ensemble des [fichiers class](https://fr.wikipedia.org/wiki/Class_(format_de_fichier)) en un [fichier JAR](https://fr.wikipedia.org/wiki/JAR_(format_de_fichier)) ;
* javah : le générateur de fichiers headers C, utilisé pour écrire les méthodes natives ;
* javap : le désassembleur de fichier .class ;
* javaws : le lanceur [Java Web Start](https://fr.wikipedia.org/wiki/Java_Web_Start) pour les applications JNLP ;
* jconsole : Java Monitoring and Management Console ;
* jdb : le débogueur ;
* jhat : outil expérimental d'analyse du [tas](https://fr.wikipedia.org/wiki/Tas_(allocation_dynamique)) ;
* jrunscript : script shell Java ;
* policytool : outil de création et de gestion de la vie privée, déterminant le niveau de confidentialité utilisé par Java en fonction de la source du code ;
* VisualVM : outil de visualisation intégrant plusieurs des outils présentés ci-dessus et permettant de faire du [profiling](https://fr.wikipedia.org/wiki/Profiling).

Le JDK est également founi avec l'[environnement d'exécution Java](https://fr.wikipedia.org/wiki/Environnement_d%27ex%C3%A9cution_Java) complet, contenant la [Java Virtual Machine](https://fr.wikipedia.org/wiki/Java_Virtual_Machine) ainsi que toutes les bibliothèques de classes présentes dans l'environnement de production.

NB on peut telecharger seulement java.exe pas javac.exe lor ce qu’on est pas des développeur juste pour executer bytecode fourni par quelqu’un

# Travailler avec java

## Outils en relation avec java

**JRE :** Java Runtime Environment

L'**environnement d'exécution Java** JRE, parfois nommé simplement « Java », est une famille de [logiciels](https://fr.wikipedia.org/wiki/Logiciel) qui permet l'exécution des programmes écrits en [langage de programmation](https://fr.wikipedia.org/wiki/Langage_de_programmation) [Java](https://fr.wikipedia.org/wiki/Java_(langage))[1](https://fr.wikipedia.org/wiki/Environnement_d%27ex%C3%A9cution_Java#cite_note-whatis-1), sur différentes [plateformes informatiques](https://fr.wikipedia.org/wiki/Plate-forme_(informatique)). Il est souvent considéré comme une [plateforme informatique](https://fr.wikipedia.org/wiki/Plate-forme_(informatique)) au même titre qu'un [système d'exploitation](https://fr.wikipedia.org/wiki/Syst%C3%A8me_d%27exploitation) (OS). En effet, s'il n'est pas un OS, il offre les mêmes fonctionnalités par l'intermédiaire de ses bibliothèques[3](https://fr.wikipedia.org/wiki/Environnement_d%27ex%C3%A9cution_Java#cite_note-puybaret-3) et permet ainsi l'exécution des programmes écrits en langage Java sur de nombreux types d'appareils — ordinateurs personnels, [mainframes](https://fr.wikipedia.org/wiki/Ordinateur_central), téléphones mobiles

JRE se compose d'une [machine virtuelle](https://fr.wikipedia.org/wiki/Machine_virtuelle_Java), de [bibliothèques logicielles](https://fr.wikipedia.org/wiki/Biblioth%C3%A8que_logicielle) utilisées par les programmes Java et d'un [plugin](https://fr.wikipedia.org/wiki/Plugin) pour permettre l'exécution de ces programmes depuis les [navigateurs web](https://fr.wikipedia.org/wiki/Navigateur_web)[1](https://fr.wikipedia.org/wiki/Environnement_d%27ex%C3%A9cution_Java#cite_note-whatis-1).Selon les principes de la [technologie Java](https://fr.wikipedia.org/wiki/Java_(technique)), lancée par [Sun Microsystems](https://fr.wikipedia.org/wiki/Sun_Microsystems) en 1995, JRE simule la présence de la [machine virtuelle java](https://fr.wikipedia.org/wiki/Machine_virtuelle_java) - un ordinateur fictif. Il joue le rôle d'un [émulateur](https://fr.wikipedia.org/wiki/%C3%89mulateur) et imite le comportement de cette machine fictive qui exécute des programmes Java[4](https://fr.wikipedia.org/wiki/Environnement_d%27ex%C3%A9cution_Java#cite_note-all-in-one-4).

[**Android**](https://fr.wikipedia.org/wiki/Android)**:** la plateforme pour les téléphones mobiles créée par [Google](https://fr.wikipedia.org/wiki/Google), comporte une [machine virtuelle](https://fr.wikipedia.org/wiki/Dalvik_(machine_virtuelle)) et une bibliothèque logicielle similaire à JRE[6](https://fr.wikipedia.org/wiki/Environnement_d%27ex%C3%A9cution_Java#cite_note-6), qui est incorporée dans les appareils par les fabricants[2](https://fr.wikipedia.org/wiki/Environnement_d%27ex%C3%A9cution_Java#cite_note-required-2).

## Installation java 11 open JDK

Téléchargement JDK 11 depuis le site de oracle 🡪 téléchargement de l’exécutable ou le ZIP🡪L’installation de l’exécutable ou la décompression du ZIP dans un dossier qu’on l’accès complet

NB : L’installation de java 11 juste la décompression car on peut configurer l’installation dans l’environnement de développement

## Modifier l’environnement de l’exécution d’un ordinateur JAVA\_HOME JRE\_HOME

Poste de travail 🡪paramètre avancés 🡪variable d’environnement🡪 ajouter : variable = java\_home ; valeur = chemin de dossier JDK (c:\programe\java\jdk\_11))

Notification : message d’erreur CMD : java n’est pas reconnu dans l’interne ou l’externe

Instalation de l’environement de travail intellij IDEA la version gratuite

La création du premier projet java/intellij\_IDEA **HELLOWORL**

Src 🡪new🡪classe.

public class Helloworld {

public static void main(String args[]) {

System.out.println("Hello world");

}

}

**NB** : un message automatique pour ajouter le projet au GIT automatiquement Intellij\_IDEA fait le sauvegarde automatiquement

Le code crées dans la classe helloworld.java est compiler dans le fichier helloworld.class qui sera exécuter dans c fichier il ya le Bytecode et au niveau de l’interpretation pour la machine c’est le role de JVM(Java Virtuel Machine) qui est inclus dans JRE, se trouve dans out…

NB : IntelIj offre des racourcis exp :A la place d’ecrire system.out.println()  on peut ecrire sout

VIDEO 11 pour créer le projet sous NETBEAN

**Quiz**

**Où allez-vous trouver le compilateur Java ?** A l'intérieur de la JRE (Java Runtime Environment)

La déclaration d’un variable c’est une réservation d’un espace vide avec une largeur qui dépend le type de variable, le contenue de cette espace vide peut changer

Le type de l’écriture **camel\_case** c’est d’écrire le premier caractère on minuscule et le premier caractère des autre mot on majuscule exp : nouvelleVoiture

## Déclaration des variables primitif commencent par minuscule

Int nom variable ; pour les entier entre -2147483648 et + 2147483647

long nomVariable ; pour les entier plus petit que -2147483648 et + 2147483647

short nomVariable ; pour les entier entre –32768 et + 32767

byte nomVariable ; pour les entier entre –128 et + 127

char nomVariable pour stocker caractère

float nomVariable nombre avec virgule 32 bit

double nomVariable grand nombre avec virgule 64 bit

boolean nomVariable vrai , faux

exemple de Déclaration du variable

Int nbr1 ;

nbr1=10 ; ici on a donnée une valeur à nbr1

nbr=11 ; ici on a changé la valeur de nbr 1

ou

Int nbr1 =10 ;

long nbr2 = 23579887655433L ici on a utilisé le caractère (L ou l) pour indiquer au java que ce nombre et trop grand pour que java réserve l’espace suffisant

float nbr2 = 2.34F ;

char c1 = ‘c’ ; c1=\uFF27 pour un caractère encodé en forma unicode

boolean javaEstFacile = true ; javaEstFacile = false ;

stacktrace = pule d’exécution détecte le problème et de trouver solution

**NB : les Variables non premitif commencent par majiscule**

String S1 ; s1= ‘Hello world’ ;

## L’operateur d’affectation

String c2= c1= ‘hello ‘ ; le programme java fait la lecture de droite à gauche

## Les opérateurs arithmétiques

Int nbr1 = 5 + 2 ; Int nbr1 = 5 - 2 ; Int nbr1 = 5 \* 2 ;

Int nb1 = 5/2 ; 🡪(2).

Int nb1 = 5%2 ; 🡪(1). Ca donne le reste de la division

float nbr1=5/2 ; 🡪(2,0) float nbr1=5f/2 ; 🡪(2,5)

## Les opérateurs Relationnels

< et > inferieur et suppérieur

Egale : (==) ; different : ( !=) ; inférieur : (<) ; supérieure : (>)

supérieure ou égale : (>=) ; inférieure ou égale : (>=)

Exp : boolean nbr = 20<30 ; 🡪 (true)

NB : on peut faire la comparaison entre les mots et les caractères et les boolean

## Les opérateurs Logiques

Ils permettent de faire des opération sur des valeurs boolean

Et : (**& ou &&** (s’il trouve le résulta au début les autre expressions ne serons pas évaluer )) ;

Ou : (**|ou ||** (s’il trouve le résulta au début les autre expressions ne serons pas évaluer)) ;

différente : ( **^** ) ; inverse le résulta : ( **!** )

Boolean b1=true ; boolean b2=false ;

Boolean b3= b1 **&** b2 ; 🡪 false ; Boolean b3= b1 **|** b2 ; 🡪 true ;

Boolean b3= b1 **^** b2 ; 🡪 true ; Boolean b3= **!**b2 **&** b1 ; 🡪 true ;

Les operateurs

## Les opérateurs d’incrémentation / Décrémentation

incrémentation : ( **++** ) ; Décrémentation : ( **--** ) ;

Int nbr1= 5 ; nbr**++** ; 🡪6 Int nbr1= 5 ; **++**nbr ; 🡪6

Int nbr1= 5 ; nbr **--** ; 🡪4 Int nbr1= 5 ; **--** nbr; 🡪5

La différence de placer le + ou le – avant ou derrière la valeur c’est au niveau la lecture le la fonction

EXP :

Int nbr1 = 5 ; int nbr2 = 2 ; nbr2 = nbr1++ 🡪(nbr2 = nbr1 = 5) et (nbr1 = 5+1 = 6)

Ici java interprète la lecture comme suite : nbr2 = nbr1 ensuite nbr1 = nbr1 + 1

Int nbr1 = 5 ; int nbr2 = 2 ; nbr2 = ++nbr1 🡪 (nbr1 = 5+1 = 6) et (nbr2 = nbr1 = 6)

Ici java interprète la lecture comme suite : nbr1 = nbr1 + 1 ensuite nbr2 = nbr1

Int nbr1 = 5 ; int nbr2 = 2 ; nbr2 = nbr1 -- 🡪(nbr2 = nbr1 = 5) et (nbr1 = 5 – 1 = 4)

Ici java interprète la lecture comme suite : nbr2 = nbr1 ensuite nbr1 = nbr1 - 1

Int nbr1 = 5 ; int nbr2 = 2 ; nbr2 = -- nbr1 🡪 (nbr1 = 5 - 1 = 4) et (nbr2 = nbr1 = 4)

Ici java interprète la lecture comme suite : nbr1= nbr1 - 1 ensuite nbr2=nbr1

## L’opérateur de concaténation +

string chaine1 = ‘Hello’ + ‘World’ ; 🡪 ‘Hello world’

string chaine1 = ‘Hello’ ; string chaine2 = ‘World’ ;

chaine1 + chaine 2 🡪 ‘Hello world’

string age = ‘mon âge est : ’ ; printLn(âge + ‘ 3’ +’ 4’) ; 🡪 mon âge est : 34

NB :java considère le 3 et 4 comme des chaine de caractère pas des nombre

string age = ‘mon âge est : ’ ; printLn(âge + (3 + 4)) ; 🡪 mon âge est : 7

NB :java considère le 3 et 4 comme des nombre pas des chaines de caractère

# Les structure de contrôle

## Le bloc d’instruction

Le bloc d’instruction est un ensemble d’instructions délimité par 2 accolades {}

Il permet de regrouper les instructions.

NB : toute variable déclaré à l’intérieur d’un bloc d’instruction est illisible à l’extérieur du bloc

Toute variable déclaré a l’extérieur du bloc d’instruction est lisible dans le bloc

## La structure conditionnelle if else

Les structure qu’on a vue s’exécute d’une manier séquentiel l’une après l’autre

Les bloc conditionnelle s’exécute si la condition est respecté

L’operateur ternaire pour réduire le code de if else ; l’idé c’est de crée un boolean de lui affecter la condition et l’utiliser dans la condition if

EXP

int age3=20;  
boolean B4 =true ;  
boolean B5 = age3==20 && B4 ;  
System.*out*.println(B5?"l'age du capitaine est :" + age3 : "l'age du capitaine different de :"+age3);  
System.*out*.println(age3==20 && B4?"l'age du chaf est :" + age3 : "l'age du chaf different de :"+age3);

EXP if else if

int age4 = 43;  
  
if (age4 == 40){  
 System.*out*.println("l'age du capitain est égale à : 40" );  
}  
else if (age4>40){  
 System.*out*.println("l'age du capitain est superieur à : 40" );  
}  
else {  
 System.*out*.println("l'age du capitaine est inférieur à : 40" );  
}

## Résoudre les erreurs : le débuggeurs

On utilise la fonctionnalité de debugage offerte par les environnements de développement comme intellij\_IDEA.

On exécute le projet avec le bouton debug a la place de l’exécution normale ca permet d’executer le code line par line et de détecter si l ya quelque chose illogique méme si le code est correcte

## La structure conditionnelle switch - case

La condition swich – case peut remplacer if else pour éviter la redondance

EXP

int age5 = 80 ;  
switch (age5){  
 case 20 :  
 System.*out*.println("le capitaine à 20 ans");  
 break;  
 case 10 :  
 System.*out*.println("le capitaine à 10 ans");  
 break;  
 default:  
 System.*out*.println("cette valeur ne correspond pas");

**Break** : pour arreter les la lecture des autre case le cas de réaliser la condition

**Default** : pour donner une instruction le cas de ne pas réaliser la condition dans tous les cases

**NB :** avant la version 7 de java on peut utiliser la structure switch-case seulement sur les type int et char, depuis la version java 7 on peut l’utiliser sur le type string

## La structure itérative for

Permet de répéter un bloc d’instruction autant de fois souhaiter tant que ce qui est donnée en condition est vrai

for (int nbr7=0;nbr7<=3;nbr7++){  
 System.*out*.println("le nombre est : " + nbr7);  
  
}

Résultat

le nombre est : 0

le nombre est : 1

le nombre est : 2

le nombre est : 3

Exp utilisation switch-case dans la boucle for

for (int nbr7=0;nbr7<=3;nbr7++){  
 switch (nbr7){  
 case 0 :  
 System.*out*.println("le nombre est : " + nbr7);  
 break;  
 case 1 :  
 System.*out*.println("le nombre est : " + nbr7);  
 break;  
 case 2 :  
 System.*out*.println("le nombre est : " + nbr7);  
 break;  
 case 3 :  
 System.*out*.println("le nombre est : " + nbr7);  
 break;  
 default :  
 System.*out*.println("aucun nombre ne correspond");  
 }

## La structure itérative while

Le fonctionnement du mot clés **WHILE** est proche de **FOR**

Lors qu’on connaît dès le départ le nombre d’itération qu’on va réaliser on utilise FOR sinon on utilise **while** ca sera plus pratique.

boolean b9=true;  
 while (b9){  
 System.*out*.println("l'itération while active");  
 b9=false;  
 }

On peut utiliser l’instruction inverse **do-while** c’est d’exécuter des instructions avant de commencer dans la boucle while

boolean b10=true;  
 do {  
 System.*out*.println("l'itération DO\_while n'es pas active");  
 b10=false;  
 }while (b10);

utiliser swich-case dans l’instruction while

while (b<3){  
 switch (b){  
 case 0 :  
 System.*out*.println("le nombre b est : " + b);  
 break;  
 case 1 :  
 System.*out*.println("le nombre b est : " + b);  
 break;  
 case 2 :  
 System.*out*.println("le nombre b est : " + b);  
 break;  
 case 3 :  
 System.*out*.println("le nombre b est : " + b);  
 break;  
 default :  
 System.*out*.println("aucun nombre b ne correspond");  
  
 }  
 b++;  
}

Résulta

le nombre b est : 0

le nombre b est : 1

le nombre b est : 2

L’utilisation du mot clés break dans l’instruction while permet d’éviter la redondance

Dans cette exemple si on utilise pas break ca donnera une boucle infinie.

boolean bo1 =true;  
while (bo1){  
 System.*out*.println("la boucle est active pour une autre seconde");  
 break;  
}

# Les classes et les objets

Un langage orienté objet permet d’offrir une vision plus proche de la réalité physique

La différence entre classe et objet :

La classe définie les caractéristique du concept en générale par exemple du voiture

Et un objet qu’on parle d’une voiture en particulier avec cette caractéristique

Une classe dispose d’un nom qui commence par un majuscule suivi par des caractère minuscule et sans accent lord ce que la classe ce compose de plusieurs mot chaque mot commence par un majiscule Exemple classe Voiture java

public class Planete {  
 String nomP ;  
 String mariereP ;  
 long diametreP ;  
}

**NB :** String est une classe

A la création d’une classe il faut indiquer qu’elles sont ces propriétés

Sin on donne pas des valeur par default pour les propriétés crées ils vent prendre des valeurs par defaut ca dépend le type du propriété

EXP des valeurs par defaut

Int 🡪 0 ; float 🡪0,0 ; String 🡪Null ; boolean 🡪 false

char 🡪 un caractère sans valeur

## L’objet ou l’instance

C’est un exemplaire de la classe on l’appel aussi instance de la classe

A la création d’un objet on dit instancier la classe par exemple a la classe Voiture on va crée un objet avec la commande new voiture() ici il vas crée l’objet et le mettre dans la mémoire

**NB :** l’association de l’objet a une variable qui va référencer l’emplacement de cet objet dans la mémoire

public class helloUniverse {  
  
 public static void main(String... args){  
  
 Planete soleil = new Planete();  
 soleil.nomP="solei";  
 soleil.mariereP="gazeuse";  
 soleil.diametreP=1391000l;

System.*out*.println("le nom du planete 1 est : " + soleil.nomP + "matire du planete est ; " + soleil.mariereP + " le diametre du planete est : " + soleil.diametreP);

}

}

Dans cette exemple on a instancié l’objet soleil de la classe Planete

NULL signifie que le variable n’a aucune valeur

## Les méthodes

Une classe est définie par des attribues également par des méthodes …

Une méthode représente une opération ou une activité qui caractérise la classe

void NomMethode() : ici a la déclaration d’une méthode void(vide) signifie la méthode ne retourne pa de résulta

Déclaration d’une méthode dans la classe planète

void revolution(){  
 System.*out*.println("je suis la planete " + nomP + "je tourne autour de mon étoile");  
}

l’exécution du méthode dans la classe main pour la planete(objet) mars déjà crées dans la classe main

mars.revolution();

## méthode avec return

int accelerer(){  
 System.*out*.println("j'accelère");  
 return 5;  
}

voitureR.accelerer();

Résulta : j'accelère ne donne pas ce qui est en return  
System.*out*.println(voitureR.accelerer());

Resulta : j'accelère

5 il donne aussi resulta du return

Ou

int nvv = voitureR.accelerer();  
System.*out*.println("la nouvelle vitesse est : " + nvv + "KM/h");

quand on déclare un objet java le déclare dans un emplacement

A la déclaration d’un String java stocke en mémoire un emplacement particulier pour une chaine de caractère et a la déclaration de la même manier un deuxième String java va utiliser le premier.

String Chaine1= ‘phrase 1 ‘ ; String chaine 2=’phrase 1’ ;

Boolean b = chaine1=chaine2 🡪 true ;

String Chaine1= new String(‘phrase 1 ‘) ; String chaine 2=new String(’phrase 1’) ;

Boolean b = chaine1=chaine2 🡪 false ;

La fonction equals compare la chaine caractère par caractère ;

equalsEgnorecase ne fait pas la différence entre le majuscule et le minuscule

Boolean b= chaine1.equals(chaine2) ;

Boolean b= chaine1.equalsEgnorecase(chaine2) ;

La surcharge des méthodes c’est de crée deux méthodes avec le même nom une avec des paramètre l’autre sans paramètre.

## Le mot clé this

Permet de faire référence a un objet en coure d’utilisation

Qu’on un paramètre qui a exactement le même nom que l’attribue de la classe on écrie this.attribue\_Classe pour que java puisse faire la différence

Exemple

int vitesse ;  
int accelerer (int vitesse){  
 this.vitesse=this.vitesse+vitesse;  
  
 return this.vitesse ;  
}

## Propriété sous forme d’objet

Encapsuler un objet c’est de faire déclarer un variable de type classe\_B dans classe\_A

Exp : crée la classe moteur et on déclare un variable dans classe voiture de type moteur

public class Moteur {  
 String carbuant ;  
 int cylindre ;  
}

public class Voiture {  
 Moteur moteur ;

]

public static void main(String[] args){

Moteur moteur1= new Moteur();

mohcine.moteur=moteur1; on peut affecter les attribue de la classe moteur a plusieur objet voiture  
reda.moteur=moteur;

NB : les méthodes peuvent prendre un argument de type objet

**Exemple**

public class Passager {  
  
 String nomPassager;  
}

Création d’une méthode qui prend un argument de type Passager crée dans la classe précédente

public class Voiture {

void transporter(Passager passagera){ /\* l’objet passager dans les argument de la méthode \*/  
 System.*out*.println("je transporte un passager qui s'appel : "+pass.nomPassager);  
}

L’utilisation de la méthode crée dans la classe main

public static void main(String[] args){

Voiture reda = new Voiture();  
reda.nom="Reda";

reda.nbrPort= 2 ;

Passager pass1=new Passager();  
pass1.nomPassager="yazid";  
reda.pass=pass1;  
reda.transporter(pass1);/\*l’utilisation de la méthode avec un passager en argument

Résulta : je transporte un passager qui s'appel : yazid

**Exemple 2**

Création d’une classe Ville + les 2 autres Passager et Voiture

public class Ville {  
 String nomVille;  
}

public class Voiture {

Passager pass ;  
Ville vil;

Ville transporter(Passager passagera,Ville villeDeppart){  
 System.*out*.println("je transporte un passager qui s'appel : "+pass.nomPassager);  
 System.*out*.println("le passager est parti de la ville de : "+vil.nomVille);  
  
 Ville villeDest=new Ville();  
 villeDest.nomVille="rabat";  
 return villeDest ;  
}

public static void main(String[] args){

Voiture reda = new Voiture(); Création d’un objet voiture  
reda.nom="Reda";

Passager pass1=new Passager(); création d’un objet passager  
pass1.nomPassager="yazid"; affectation des donnée aux attribues de l’objet passager crée  
reda.pass=pass1; affectation des valeurs de l’objet passager (passe1) a la voiture reda

Ville rabat=new Ville();   
rabat.nomVille="casablanca";  
reda.vil=rabat;

Ville destination= reda.transporter(pass1,rabat); création un variable de type Ville pour recevoire le return de la methode transpport.

System.*out*.println("le passager arrive à :"+destination.nomVille);

}

Résulta

je transporte un passager qui s'appel : yazid

le passager est parti de la ville de : casablanca

le passager arrive à :rabat

## Les Propriété / attribue static ou variable de classe

Une propriété ou un attribue static garde sa valeur ou la référence à un objet et la même quelque soit l’instance (l’objet) sauf si on l’a mofidié

Exemple : crées une propriete static forme sui sera fixe pour tous les planètes crée et de la tester avec la planète mars

public class Planete {  
 public static String *forme*="SPHERE";

}

public static void main(String... args){

Planete mars = new Planete();  
mars.nomP="mars";  
mars.mariereP="tellurique";

System.*out*.println("la forme de tous les planetes est : "+Planete.*forme*);  
System.*out*.println("la forme de mars est :"+ mars.*forme*);

}

**Résulta**

la forme de tous les planetes est : SPHERE

la forme de mars est :SPHERE

## méthode static

Une méthode static c’est une méthode qui n’applique aucun attribue d’instance

Exemple pour la méthodes qu’on utilisé précédemment

static void klaxonner(){ /\*ici on a ajouté static cette methode ne depend pas des attribue de l’objet \*/  
 System.*out*.println("tuuuuuuutuuuuuutuuuuu!!!!!");  
}

Exemple 2

static tourner(boolean droite,int anglet){  
 String direction = null;  
 if(droite){  
 direction="droite"; cette modification est accepter car on n’implique pas les attribue de l’instance  
 }  
 else {  
 direction="gauche";  
 }  
 System.*out*.println("la voiture est tourné à " + direction + " d'une anglet de : " + anglet);  
}

Exemple 3 NB cette methode ne peut pas etre static car il modifie une proprieté de l’instance (vitesse)

int rapportC(boolean b){  
 if (b){  
 vitesse++ ;  
 }  
 else {  
 vitesse-- ;  
 }  
 return vitesse;  
}

NB : problème de compilation j’ai changé au code ajouter un 2 éme return dans la méthode pour pouvoir exécuter le code

## Constructeur sans paramètre et par défaut

On peut instancier la classe pour donner des instructions à chaque création d’un objet de cette classe

Exemple la classe voiture

Voiture(){  
 System.*out*.println("la voiture est encoure de construction");  
}

Ce message sera affecter a tous les voitures crée

public class Planete {  
 static int *nombreplanette*; le type static pour pouvoire utilister directement dans la classe main  
 Planete(){  
 *nombreplanette*++;  
 }

public static void main(String... args){  
  
 System.*out*.println(Planete.*expansion*(10.2d));  
 System.*out*.println(Planete.*expansion*(14.2d));  
 Planete soleil = new Planete();

Planete venus = new Planete();  
 soleil.nomP="venus";

Planete mars = new Planete();  
 soleil.nomP="mars";

Planete uranu = new Planete();  
 soleil.nomP="uranus";

System.*out*.println("le nombre des planette est"+ Planete.nombreplanette);

## Ajouter constructeur avec parametre

Voiture(){  
 System.*out*.println("la voiture est encoure de construction");  
}  
Voiture(String coleur){  
 this.coleur=coleur;  
 System.*out*.println("la vcouleur de la voiture est :"+this.coleur);  
}  
Voiture(int nbrou2){  
 this.nbrou2=nbrou2;  
 System.*out*.println("le nombre de roues est : "+nbrou2);  
}

Voiture voitureR5 = new Voiture();  
Voiture voitureR6 = new Voiture("jaune");  
Voiture voitureR7 = new Voiture(5);

Dans cette exemple java choisi le constructeur adéquat au type de la déclaration de l’objet voiture

NB : On peut ordonné même autre classe tel que la classe moteur dans les arguments du constructeur

Voiture(Moteur moteur){  
 this.moteur=moteur;  
 }

# L’héritage et les interfaces

## Le mot clé extends

L’héritage c’est l’ord ce qu’il une classe hérite d’une autre classe les méthode les même attribues pour pouvoir les utiliser, le mot extends permet de faire l’héritage au langage JAVA.

Une classe fils peut hériter d’une seul classe mère ;

La classe mère par défaut de chaque classe est la classe objet et à la création d’une classe mère la classe objet il ne le sera plus

On peut modifier les méthodes de la classe mère dans la classe fils

public class Voiture extends Vehicule{

}

public class Vehicule {  
 String matiere;  
 void tourner(boolean droite,int anglet){  
   
 System.*out*.println("la voiture est tourné à " + direction + " d'une anglet de : " + anglet);  
 }  
}

Dans cette exemple la classe mère hérite la proprieté matiere et la méthode tourner

NB : la classe voiture est la classe fils ; la classe Vehicule est la classe mère

Exercice :

Création d’une classe fils

public class VesseauDeGuerre extends Vesseau{  
  
 void attaque(Vesseau vesseauattaque,String materierle, int duree){  
 System.*out*.println("un vesseau de type : "+typV+" attque un vesseau de type"+vesseauattaque.typV+"en utilisant l'arme "+materierle+" pendant "+duree+" Minute");  
 vesseauattaque.resistanceDeBuclier=0;  
 vesseauattaque.blindage=vesseauattaque.blindage/2;  
 }  
}

La classe mère

public class Vesseau {  
 String typV;  
 int nbrPassager;  
 int blindage;  
 int resistanceDeBuclier;  
  
 void activerBouclier(){  
 System.*out*.println("le vesseau " +typV+" à activé le bouclier");  
  
 }  
 void desactiverBouclier(){  
  
 System.*out*.println("le vesseau " +typV+" à désactivé le bouclier");  
  
 }  
}

Classe Main

public static void main(String... args){

VesseauDeGuerre chasseur=new VesseauDeGuerre();  
 chasseur.blindage=156;  
 chasseur.resistanceDeBuclier=2;  
 chasseur.typV="CHASSEUR";  
  
 VesseauCivil vesseauMonde=new VesseauCivil();  
 vesseauMonde.blindage=4784;  
 vesseauMonde.resistanceDeBuclier=30;  
 vesseauMonde.typV="Vesseau-Monde";

vesseauMonde.activerBouclier();  
 chasseur.activerBouclier();  
 chasseur.attaque(vesseauMonde,"lasére photomique",3);  
 vesseauMonde.desactiverBouclier();  
  
 System.*out*.println("la résistance d'une bouclier de VM est "+vesseauMonde.resistanceDeBuclier);  
 System.*out*.println("le blindage de VM est "+vesseauMonde.blindage);  
  
}

**Résulta**

le vesseau Vesseau-Monde à activé le bouclier

le vesseau CHASSEUR à activé le bouclier

un vesseau de type : CHASSEUR attque un vesseau de typeVesseau-Mondeen utilisant l'arme lasére photomique pendant 3 Minute

le vesseau Vesseau-Monde à désactivé le bouclier

la résistance d'une bouclier de VM est 0

le blindage de VM est 2392

## Le transtypage

Qu’on nous voulons qu’un variable soit déclaré comme un type parent

Exmple

Moteur reda = new Voiture();  
reda.cylindre=6;

((Voiture)reda).nom="red";"; ici on a utilité les propriété de la classe fils sans changer le typre de la voiture reda (le transtypage a la volé)

Dans cette exemple on a déclarer une voiture(classe fils) reda de type moteur(classe mère) on peut utiliser seulement le propriétés qui existent dans la classe mère

Il y a la possibilité de changer le type de voiture déclaré de la classe mère au classe fils le cas de besoin d’utilisation des méthodes ou attribues de la classe fils, c’es en ajouttant le code suivant avant d’utiliser les propriétés

Voiture redaVersionVoiture = (Voiture)reda;

## L’héritage et constructeur le mot clé super

Une classe fille hérite les propriétés et les méthodes d’une classe mère, il n’hérite pas les constructeurs crées, pour faire cela il faut utiliser le mot clé super

Dans cette exemple on va crée un constructeur dans la classe VehiculeAmoteur (mère) ce constructeur vas rappeler le constructeur du classe moteur pou pouvoir l’utiliser dans la classe voiture

Classe mère

public class VehiculeAmoteur {  
  
 Moteur moteur ;  
 int vitesse ;  
  
 VehiculeAmoteur(){  
  
 }  
 VehiculeAmoteur(Moteur moteur){  
 this.moteur=moteur;  
 System.*out*.println("une voiture est construite avec le moteur ");  
 }  
  
}

Classe fille

public class Voiture extends VehiculeAmoteur{

Voiture(){  
 super();  
}

Ou

Voiture(){  
 super(new Moteur()); dans ce cas il faut ajouter super(new Moteur()); ou this ; dans tous les constructeur de la classe voiture Voiture

}

/////////////////////////////////////////////////  
Voiture(Moteur moteur){  
 super(moteur);  
}  
public Voiture() {  
  
}

Dans la classe main pour la création avec ce constructeur de la classe mère

Moteur moteur1= new Moteur();

Voiture reda1 = new Voiture(moteur1);

## Covariance des méthodes

Création d’une classe UsineDassemblage dans laquelle on déclare une méthode qui initialise VehiculeAmoteur

public class UsineDassemblage {  
  
 VehiculeAmoteur assemblageM(){  
 Moteur moteur=new Moteur();  
 VehiculeAmoteur vam=new VehiculeAmoteur(moteur);  
 return vam;  
 }  
  
}

création d’une classe qui hérite de la classe UsineDassemblage et redifinir la méthode assemblageM

public class UsineDassemblageVoiture extends UsineDassemblage{  
  
 Voiture assemblageM(){  
 Voiture v =new Voiture();  
 return v;  
 }  
}

L’utilisation de la méthode assemblageM avant et après la redéfinition (la modification du type de retour)

public class origin {  
 public static void main(String[] args){

**avans la redifinition**

UsineDassemblage ua=new UsineDassemblage();  
 VehiculeAmoteur vam=ua.assemblageM();

**Après la redifinition**

UsineDassemblageVoiture u=new UsineDassemblageVoiture();  
 Voiture v=u.assemblageM();  
  
 }  
}

## L’interface : le mot clé implement

Quand il s’agit d’un héritage multiple c’est l’utilisation des interfaces

Une classe peut hériter une seul classe et implémenter plusieurs interfaces

Une interface peut hériter une autre interface

Exemple

Ici on va crée une interface vidanger qu’on peut utiliser dans plusieurs classes ca dépend le besoin (on vas faire le vidange pour une voiture ou pour un lavabo … )

Interface avec méthode

public interface Amarrable {  
  
// methode Nombre de corde  
 public int combienDecorde(int vitesseduvent);  
}

**Classe bateau utilise l’interface** Amarrable **et redéfinie la méthode**

public class Bateau implements Amarrable{  
 int masse;  
 @Override  
 public int combienDecorde(int vitesseduvent) {  
 int nbcord=masse/10;  
 nbcord=nbcord+vitesseduvent/100;  
 return nbcord;  
 }  
}

dans la classe main

public class origin {  
 public static void main(String[] args){  
  
 Bateau b=new Bateau();  
 b.masse=60;

Port p=new Port();  
 p.acceuilleEngin(b);  
  
 }  
}

**Résulta**

**le nombre de corde necessaire est : 6**

NB : une interface peut hériter un ou plusieurs interfaces

public interface Amarrable extends Videngeable,interface2{

Interface Drappeau et serializable : Des interfaces vide fondamentale ??

**NB** : une interface peut avoir des propriétés avec des valeurs fixe

## Le Mot clé instanceof

Permet de tester la nature d’un objet crée

public class Port {  
  
 void acceuilleEngin(Amarrable amarable){  
 if (amarable instanceof Bateau){  
 System.*out*.println("c'ette amarable est un bateau");  
 }  
 int nbcorde=amarable.combienDecorde(50);  
 System.*out*.println("le nombre de corde necessaire est : "+nbcorde);  
 }  
}

dans cette exemple on a tester si le amarable crée est un bateau on ecrit le message de printLn

**Exemple exercice video 81**

public class PlaneteTellurique extends Planete implements Habitable{  
  
  
 public PlaneteTellurique(String nom){  
 super(nom);  
 }  
  
 public Vesseau acceuillirVesseau(Vesseau vesseau) {  
 if (vesseau instanceof VesseauDeGuerre){  
 ((VesseauDeGuerre)vesseau).desactiverArme();  
 }  
 nbrH = nbrH + vesseau.nbrPassager;  
 vesseauA = vesseau;  
 if (vesseauA == null) {  
 System.*out*.println("aucun vesseau ne s'en va");  
 }  
 else {  
 System.*out*.println("un vesseau de type :" + vesseauA.typV + " doit s'en aller");  
 }  
 return vesseau;  
  
 }  
}

public class VesseauDeGuerre extends Vesseau{  
 boolean armedesactiver;  
 void attaque(Vesseau vesseauattaque,String materierle, int duree){  
 if (armedesactiver){  
 System.*out*.println("l'attaque impossible arme désactiver");  
 }  
 System.*out*.println("un vesseau de type : "+typV+" attque un vesseau de type"+vesseauattaque.typV+"en utilisant l'arme "+materierle+" pendant "+duree+" Minute");  
 vesseauattaque.resistanceDeBuclier=0;  
 vesseauattaque.blindage=vesseauattaque.blindage/2;  
 }  
 void desactiverArme(){  
 armedesactiver=true;  
 System.*out*.println("l'arme est désactivé "+typV);  
 }  
}

le polymorphisme (de sous-typage)

Permet de considérer un objet comme un autre compatible

Une méthode peut accepte un objet sous la forme de son type parent

Exercice Video 83 a refaire

@Override  
 void activerBouclier(){  
 this.desactiverArme();  
 super.activerBouclier();  
 System.*out*.println("le vesseau " +typV+" à activé le bouclier");  
  
 }  
}

Main

Vesseau chasseur=new VesseauDeGuerre();  
chasseur.blindage=156;  
chasseur.resistanceDeBuclier=2;  
chasseur.typV="CHASSEUR";

chasseur.activerBouclier();  
((VesseauDeGuerre) chasseur).attaque(vesseauMonde,"lasére photomique",3);  
vesseauMonde.desactiverBouclier();

## Méthode et classe abstraite(abstract)

abstraite est une classe qui ne peut pas être instancier, il a le but de généraliser d’autre classes

La classe abstraite peut avoir de propriétés des méthodes aussi des méthodes spéciale que l’on appel abstrait et n’on pas un bloc d’instruction

exemple classe abstract

public abstract class Vehicule {  
 abstract void klaxoner();  
}

une classe abtract peut être hérité par une autre classe et de redifinir les méthodes déclarés ça dépend le besoin ça donne plus de flexibilité à l’utilisation des méthodes

public class Voiture extends Vehicule implements Videngeable {

@Override  
 void klaxoner() {  
 System.*out*.println("tuuutuuu!!!!!");  
 }  
}

public class Moteur extends Vehicule implements Videngeable {

@Override  
 void klaxoner() {  
 System.*out*.println("TooooooTOOOOOOOOOOOOPPOOOOOOO!!!!!");  
  
 }  
}

**NB** : si on veut redéfinir les méthodes dans les classe sans crée une classe abstract il faut les rendre aussi abstract, l’inconvénient c’est qu’on peut pas instancier ces classe(crée des objets)

Exercice a refaire

# Classes et techniques utilitaires

**La classe système**

Est une classe qui offres des propriete et des methodes, cette classe ne peut pas etre instancier car il n’a pas de constructeur visible.

Système. Out.printLn

Système. err.printLn

Système. in.printLn ; in : pour exploiter ce qui est entrer sur clavier …

Système.exit() ; pour mettre fin au programme

long time=Système.currentTimeMilles() ; pour donner combien de temps est écouler depuis 1970 en millisecondes

système.nanoTime permet de donner le temp en nanoseconde permet de donner les ecart du temp .

**long timeStat=** système.nanoTime() ;

system.out.printLn= (‘hello world’) ;

**long timeFin=** système.nanoTime() ;

system.out.printLn= (**timeStat** - **timeFin**) ;

ce code donne en résulta combien de temp à pris l’affichage de ‘Hello world’ ; en nanotime

system.out.printLn(system.GetProperties()) ;

Permet d’afficher les propriétés du système.

## La classe scanner

Permet de scanner ce qui vas être entre dans le clavier ou …

Exemple d’utilisation

import java.util.Scanner; pour pouvoir utiliser la classe Scanner  
  
public class Helloworld {  
  
 public static void main(String[] args){  
  
 System.*out*.println("Hello World");  
 Scanner sc=new Scanner(System.*in*);  
 String ligneSivant=sc.nextLine(); après l’affichage de hello world le système va nous donner la main pour ecrire au clavier   
 System.*out*.println("la ligne suivante est : "+ligneSivant);l’affichage de ce qu’on a taper par le clavier  
 }  
}

exemple 2 exercice video 88

import java.util.Scanner;  
  
public class HelloUniverse {  
  
 public static void main(String... args) {  
  
  
 Vaisseau fregat = new VaisseauDeGuerre("fregat");  
  
 Vaisseau croiseur = new VaisseauDeGuerre("CROISEUR");  
 Vaisseau chasseur = new VaisseauDeGuerre("CHASSEUR");  
 Vaisseau cargo = new VaisseauDeGuerre("CARGO");  
 Vaisseau veasseauMonde = new VaisseauDeGuerre("Terre");  
  
 Scanner sc=new Scanner(System.*in*);  
 System.*out*.println("quel vaisseau choisissez vous ? ");  
 String vaisseauChoisi=sc.nextLine();  
 System.*out*.println("sur quel planete ? ");  
 String planeteChoisi=sc.nextLine();  
 System.*out*.println("quel tonnage ? ");  
 int tonnageChoisi=sc.nextInt();  
  
 Vaisseau vaisseau=null;  
 switch (vaisseauChoisi){  
 case "fregat":  
 vaisseau=fregat;  
 break;  
 case "chasseur":  
 vaisseau=chasseur;  
 break;  
 case "cargo":  
 vaisseau=cargo;  
 break;  
 }  
 PlaneteTellurique planeteC=null;  
 switch (planeteChoisi){  
 case "mars":  
 vaisseau=cargo;  
 break;  
 case "uranus":  
 vaisseau=cargo;  
 break;  
 }

## Classes conteneur

Encapsule chaque type primitif par exemple pour int 🡪Integer ; string🡪String …

Chaque classe conteneur possède une méthode permettant de définir la valeur similaire d’un variable de string le rendre int par exemple.

L’un des avantage ces de beneficier de tous les caracterestique des classe en génerale par contre au variable de type primitif par exemple les classe onteneur peuvent etre nul

Exemple d’utilisation

Integer i = new Integer(12);  
  
Float f=new Float(12.2f);  
Float f2=new Float("62.8");  
float f3=f.floatValue();  
int i2=i.intValue();  
float f4=Float.*parseFloat*("89.9");

L’auto boxing et auto-unboxing : permet de donner une valeur a une variable de classe conteneur sans instancier le variable déelarer . c’est de changer un variable de type conteneur a un variable de type primitif

Integer a = 5 ; a la place de Integer a = new Integer(5) ;

## Conversions de type(parseInt, parseString …)

Pour transformer une chaine en int 🡪 int i3=Integer.parseInt(’16 ‘) ;

Exp

int i3=Integer.*parseInt*(5); transferer Integer en Int  
String s4=String.*valueOf*(i3); le meme cas pour transferer un long

Long long1 = *i3.longvalueOf*(i3)

## Les types énumérer

Il s’agit de restreins les valeurs d’une variable

Par exemple si nous avons une boite a vitesse qui contient troi type :

Auto , manuel, semi-auto.

A la place de déclarer un boolean il faut un autre type ou de crées une classe ‘exp

public class TypeBoitVitesse {  
 static final int *auto*=1; le mot clé final indique que cette variable est fixe  
 static final int *manuel*=2;  
 static final int semi-*auto*=3;  
}

Voiture peugeot=new Voiture();  
peugeot.typB=TypeBoitVitesse.*auto*; ou peugeot.typB=1;  
if (peugeot.typB==TypeBoitVitesse.*auto*){  
 System.*out*.println("la boite a vitesse est auto");  
}

Résulta

la boite a vitesse est auto

Dans ce cas on va utiliser le système énumération

A la place de déclarer une classe avec des variable static final on changer le type de classe en enum

public enum TypeBoitVitesse {  
 *auto*,  
 *manuel*,  
 *semiAuto*;  
}

public class Voiture extends Vehicule implements Videngeable {  
  
 TypeBoitVitesse typB;

public class origin {  
 public static void main(String[] args){  
  
 Voiture peugeot=new Voiture();  
 peugeot.typB=TypeBoitVitesse.*auto*;  
 switch (peugeot.typB){  
 case *auto*:  
 System.*out*.println("la boite est automatique");  
 break;  
 case *manuel*:  
 System.*out*.println("la boite est manuel");  
 break;  
 case *semiAuto*:  
 System.*out*.println("la boite est semi\_auto");  
 break;  
 }

Résulta

la boite est automatique

On peut ajouter un indication après la déclaration des enums exp

public enum TypeBoitVitesse {  
 *auto*("automatique"),  
 *manuel*("semi-manuelle"),  
 *semiAuto*("Manuelle");  
 String nomTypeBoite;  
 // java essaie de trouver un constructeur avec string en argument: on va l'jouter  
 private TypeBoitVitesse(String nomTypeBoite){  
 this.nomTypeBoite=nomTypeBoite;  
 }  
}

public class Voiture extends Vehicule implements Videngeable {  
  
 TypeBoitVitesse typB;

public class origin {  
 public static void main(String[] args){  
  
 Voiture peugeot=new Voiture();  
 peugeot.typB=TypeBoitVitesse.*auto*;  
 System.*out*.println("le type de ma boite vitesse est : "+peugeot.typB.nomTypeBoite);

System.*out*.println("le type de ma boite vitesse est : "+peugeot.typB.ordinal());cette ligne permet de donner le numero d’ordre du variable typeB

}  
}

Résulta

la boite est automatique

NB : le même résulta que le code précédent

Exercice Final à refaire

QUIZ

Il y a bien une méthode pour chaque type : nextLine() pour les String, nextInt() pour les int etc... pour les char; il faudra faire un nextLine().chartAt(0)

Sytem.out C'est en effet le canal de sortie par défaut qui se trouve être la console en général mais ceci est modifiable.

# Les tableaux et les collections

## Tableau unidimensionnel

Les tableaux Sont des regroupements ordonnés 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 vérifier 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 méthode main

La classe qui contient la méthode main est l’accès principale du programme c’est une classe principale exécutable qui peut contenir des attribue des méthodes et des constructeurs

On peut déclarer la méthode main dans une autre classe 🡪 la classe deviendra exécutable.

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

## Tableau Multidimensionnel

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 différence les collections ne peuvent contenir des types primitif.

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

On java on peut stocké des collections d’objet grâce aux 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 …

## Les List

Les listes sont des collections qui peuvent contenir plusieurs fois le même élément chacun de ces élément est numéroté 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 courante (ArrayList et LinkList)

ArrayList est la classe la plus rapide en générale

LinkList plus rapide qu’on effectue des insertion au début de liste ou suppression d’élément 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 supprime 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 préciser ce qui va etre dans une collection grâce 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 Voiture 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);

## Les Set (HashSet)

Que l’on peut traduire par (des ensembles) sont des collections qui ne peuvent pas contenir 2 fois le même 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 pour 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

exemple d’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 va 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 opération classique sur les collections par le biais de méthode static

Exemple de fonction replaceAll qui permet de remplacer 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 même 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 collections par 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 implémente 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 primitif long par la classe conteneur Long on va économiser le code en utilisant la méthode de comparaison a la place de faire la comparaison nous même

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 implémente set et **TreeMap** qui implémente map peuvent être ordonné 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 à 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’équivalence 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();  
}

**NB :** Il ‘est conseiller de redéfinir la méthode 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.

# Organiser et gérer l’accès aux classes

## Notion de package et d’import

Les package permettent d’organier les classes on creants des packages comme des dossiers pour rassembler certain classes.

On peut créer des sous package

Les classe du langage java sont situé dans des packages tel-que la classe String est situé dans le package java.long et List dans java.util

NB : Comment nomé un package : on choisi deux prmier niveau, le nom de domain internet de l’entreprise proprietaire de logiciel réalisé et de version inverser

## L’import static

Permet d’utliser des methodes ou foncionalité sans rappeler la classe qu’il est contient exp ;

import java.lang.math.PI;

public class HelloUniverse {  
  
 public static void main(String... args) {

System.*out*.println("la valeur de pi est"+PI);

????? j’arrive pas à importer le package ?????

import java.lang.math.PI;

Cette méthode d’importation n’est pas conseiller

## les modificateur d’accès (public, private, protected, package)

ses modificateur modifie la visibiliter d’attribue de mèthode et constructeur essentiellement

public ; permet de rappeler la classe la proprieté … de n’importe quel package, c’est la visibilté par default qu’on est dans le meme package.

Private : interdit d’utiliser la propriete ou mèthode par d’autre classe

Protected : permet d’utiliser la propriete ou .. par les classe situé dans le meme package et les classe hérittante

## L’encapsulation des propriétés : premiere approche des composants réutilisable

Get ang Set

C’est de déclarer les propriété en private et de mettre (la methode get pour la lecture et la methode set pour la modification) ca depend le besoin

public int getNbPassagers() {  
 return nbPassagers;  
}  
  
public void setNbPassagers(int nbPassagers) {  
 this.nbPassagers = nbPassagers;  
}

e la classe main pour l’utilisation

chasseur.setNbPassagers(3);

Les commentaire

Pour ajouter un commentaire en utilise // pour une seul ligne et /\* et \*/

/\*\* sont des commentaire qu’on appel JavaDoc ce type de commentaire peuvent s’ajouter devant une classe un constructeur methode ou attribue il s’agi de documenter

Pour generer la documentation java en HTML visible dans un navigateur : Run 🡪generatedoc …

Le mot clé Final indique afin on peut pas modifier la valeur d’une propriété une fois valorisé

final int vitesse=150 ; la propriété doit etre initialiser

# Tenir compte des scenario alternatifs été des cas d’erreurs

## La classe exception

En java pour indiquer qu’un traitement représente une exception il faut indiquer une classe qui représente au mieux cette exception en peut utiliser La classe java.lang.Exception ou de créer une classe qui gère l’exception

A la création de la classe exception il faut au moins un constructeur (meme constructeur sans paramétre)

Exemple classe exception

public class NombreCaractereInvalide extends Exception{  
  
 public NombreCaractereInvalide() {  
 }  
 public NombreCaractereInvalide(String message) {  
 super(message);  
 }  
}

## le mot clé throws

lord ce qu’une methode susceptible d’avoir des exceptions on peut sur fixer la méthode par le mot clé throws suivi par les classes d’exceptions que la méthode vas générer

## les mot clés try catch et  finally

le bloc try catch permet de gérer une exception

en exécute le bloc try si tous est bien passé sinon on passe au bloc du catch

le mot clé finally permet d’exécuter un bloc d’instruction a la fin du bloc try catch

import com.udemy.decouvertejava.demenstration.transport.UsineDassemblage;  
  
public class UsineDassemblageVoiture extends UsineDassemblage {  
 public void assembler(){  
 Voiture v=new Voiture();  
 try {  
 v.immtarculer("AA AAZ 54");  
 System.*out*.println("le matricule est valid");  
 } catch (Exception ex) {  
 System.*out*.println("le matricule invalide");  
 System.*out*.println(nombreCaractereInvalide.getMessage());  
 }  
 finally {  
 System.*out*.println("ce bloc s'execute dans tous les cas");  
 }  
 }

## Cas des exceptions multiple

public void immtarculer(String numimmatriculation)throws NombreCaractereInvalide,DejaImmatriculer{  
 if (numimmatriculation.length()!=9){  
 throw new NombreCaractereInvalide("le numero "+numimmatriculation+" n'est pas valide");  
 }  
 if(this.immatriculation!=null){  
 }  
 this.immatriculation=numimmatriculation;  
}

import com.udemy.decouvertejava.demenstration.transport.UsineDassemblage;  
  
public class UsineDassemblageVoiture extends UsineDassemblage {  
 public void assembler(){  
 Voiture v=new Voiture();  
 try {  
 v.immtarculer("AA AAZ 54");  
 System.*out*.println("le matricule est valid");  
 } catch (NombreCaractereInvalide nombreCaractereInvalide) {  
 System.*out*.println("le matricule invalide");  
 System.*out*.println(nombreCaractereInvalide.getMessage());  
 }  
 catch (DejaImmatriculer dejaImmatriculer){  
 System.*out*.println("cette voiture a deja une immatriculation");  
 }  
 finally {  
 System.*out*.println("ce bloc s'execute dans tous les cas");  
 }   
 }  
}

public void assembler(){  
 Voiture v=new Voiture();  
 try {  
 v.immtarculer("AA AAZ 54");  
 System.*out*.println("le matricule est valid");  
 } catch (NombreCaractereInvalide | DejaImmatriculer ex) { ici on fait le test sur deux exception  
 System.*out*.println(ex.getMessage());  
 }  
  
 finally {  
 System.*out*.println("ce bloc s'execute dans tous les cas");  
 }

La classe Exception n’est pas la seul type des classes qu’on peut rejeter

La hiérarchie de java qui concerne les erreurs

La classe throwable est la classe de tous types d’erreurs de java

Java.lang.objet🡪java.lang.Throwable🡪java.lang.Exception

Java.lang.objet🡪java.lang.Throwable🡪java.lang.Error

Concerne les problèmes lie à l’erreur d’environnements

OutOfMemoryError dénote un problème de mémoire sur machine

Error est une exception UNCHEKED n’est pas nécessaire de

Java.lang.objet🡪java.lang.Throwable🡪java.lang.Exception🡪 java.lang.RunTimeException

La classe java.lang.RunTimeException aussi une classe non verifier il n’est nécessaire de les déclarer aux méthodes.