Chapitre 2 : Classe et objet

1- Introduction :

En programmation orientée objet, un programme est vu comme un ensemble d'objets qui fonctionnent ensemble pour résoudre un problème donné (objets en interaction). On répond d'abord à la question de quoi s'agit-il en déterminant les différents objets inclus dans ce problème. On peut considérer ce type de programmation comme une projection du monde réel sur le monde de la programmation.

2- Classe et Objet

L'objet

Un objet représente une entité du monde réel que ce soit physique (chaise, voiture, maison, personne …etc.) Ou bien virtuelle (Compte bancaire, commande, flux de donnée …).

Chaque objet comporte deux aspect un aspect statique qui représente ses caractéristiques et les différentes informations liées à ce dernier et l'aspect dynamique qui représente le comportement de ce dernier exprimer par les différents actions et méthodes d'utilisées pour manipuler ce dernier.

Exemple :

- **ahmed** et **said** sont deux objet (deux étudiants), le numéro d'inscription de **ali** est 2003155 son âge est 19 ans il peut assister au cours, participer dans des manifestation scientifique ….

- l'objet v1 représente la voiture ayant le matricule 5462.108.21 avec un couleur noir elle peut klaxonner, accélérer, arrêter , clignoter, tourner,…

La classe

Une classe est une description abstraite d'un objet où on définit les caractéristiques et le comportement communs à un ensemble d'objets. Elle est considérée comme un modèle ou la moule à partir du quelle l'objet va être créé. On peut la voir aussi comme un guide d'emploi d'un objet.

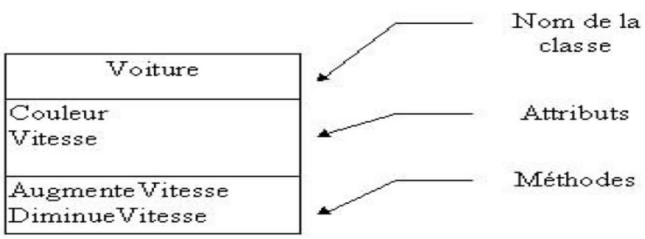
Une classe c'est une génération de notion de type. La notion d'objet est une génération de la notion de la variable d'un type.

La classe est donc constituée de deux parties (deux descriptions) :

- Partie statique qui définit les différentes caractéristiques ou informations liées à ce type d'objets qu'on l'appelle **attributs**.
- Partie dynamique où on décrit le comportement de ce type d'objet exprimer par les différents actions et fonctionnalités fournis par ce type d'objets qu'on l'appelle **méthodes**.

Exemple :

- Tous les voiture ont un couleur et ont une vitesse qui se change pendant la vie de cette voiture. Ils ont le même type de comportement qui est par exemple l'augmentation et la diminution de la vitesse.



- Tous les comptes bancaires ce caractérisent par un numéro, les coordonnées du propriétaire du compte, et un solde. Ils ont tous les mêmes fonctionnalités créditer, débiter, transférer….
- **Par convention** le nom de la classe commence toujours par une majuscule. Si le nom est un mot composé les autres mots doivent commencer par une majuscule (pas une obligation).

Syntaxe de définition d'une classe

Voici la syntaxe de définition d'une classe de base en Java :

Class Nom_De_La_Classe
{
// attribut
// méthodes
}

La syntaxe complète de déclaration d'une classe est comme suit :

```
Modificateurs class nom_de_classe [extends nom_classe_mere] [implemenents Interfaces]
{
// insérer ici les champs et les méthodes
      }
```

Un modificateur est un mot clé écrit au début de la déclaration d'une classe (attribut ou une méthode) qui a une signification spéciale lors de la déclaration (niveau de visibilité, abstraction, etc.). Par exemple : public, private, protected, static, final, abstract…

Les attributs

Les attributs sont les caractéristiques ou les informations liées à un objet. Elle décrit l'aspect statique (partie données) de l'objet. Ils représentent généralement soit :

- Des caractéristiques d'un objet (rarement changées) par exemple : la couleur et matricule d'une voiture, le nom et l'adresse d'un étudiant.
- Des variables qui expriment L'état d'un objet par exemple : vitesse d'une voiture, allumage d'une lampe.
- Des composants d'un objet complexe Par exemple : une voiture est composée des roues, un moteur …
- Des objets en relation avec l'objet lui-même par exemple : le propriétaire de la voiture (objets de la classe Personne).

Ils ressemblent aux champs d'un enregistrement. Ils sont reconnus dans tous les méthodes constituants la classe (leur porté ressemble au porté des variables globales qui sont reconnus dans toutes les fonctions et les procédures du programme).

La syntaxe de déclaration d'un attribut :

La syntaxe de base pour déclarer un attribut est comme suit :

type_attribut nom_attribut;

On peut ajouter des modificateurs à la déclaration et éventuellement faire une initialisation lors de la déclaration. Donc la syntaxe détaillée pour déclarer un attribut est la suivante :

Modificateurs type_attribut nom_attribut = valeur;

Exemple :

Class Voiture {

String matricule;

String couleur;

Int vitesse ;

Moteur moteur; // Moteur est une autre classe d'objet (c'est une composant de la voiture)

Personne propriétaire; // Personne est une autre classe

}

3- Les méthodes :

A chaque classe sont associées des méthodes, elles permettent de décrire le comportement (aspect dynamique) de ce type d'objets en décrivant les opérations ou les traitements qu'on peut effectuer sur les objets de cette classe. En d'autres termes Elles expriment les différentes fonctionnalités fournis par ce type d'objets. Par exemple pour une voiture on peut augmenter ou diminuer sa vitesse. Pour un rectangle on peut calculer sa surface, calculer son périmètre ou l'agrandir.

Une méthode ressemble à une fonction ou procédure dans la programmation classique.

Syntaxe de définition d'une méthode :

Une méthode est une portion de code à laquelle on associe : un nom, une liste de paramètres éventuelle , un type de retour (plus que l'implémentation).

Voici la syntaxe de base d'une méthode :

Type_de_retour nom_Methode (liste_arguments) {// le traitement}

- Toutes les méthodes sauf les « constructeurs » doivent avoir un type de retour. Les méthodes qui ne renvoient rien, utilisent « void » comme type de retour (considérer comme procédure)

- Les paramètres (les arguments) sont indiqués entre parenthèses après le nom de la méthode. On doit spécifier le nom et le type de chaque paramètre s'il y en a plusieurs, on les sépare par une virgule. S'il n'y aucun paramètre, la liste est vide et les parenthèses **()** sont obligatoire.

- On peut déclarer dans une méthodes des variables locales (reconnus que dans la méthode où elles sont déclarées) afin d'accomplir la tâche fournis par cette dernière.

- La valeur de retour de la méthode (s'il s'agit d'une fonction) doit être transmise (retournée) par l'instruction **return**. Elle termine celle-ci , Donc l'instruction return doit être la dernière instruction à exécuter dans la méthode (toutes les instructions qui suivent return sont ignorées).

- l'implémentation d'une méthode est l'ensemble des instructions écrites entre les deux accolades de la méthode elle exprime le traitement effectuer par cette méthode.

- l'entête de déclaration d'une méthode peut être précéder par des modificateurs (pas obligatoire)

- une méthode peut agisse (manipuler) les attributs de la classe où elle est définie.

Exemple 1 : la méthode afficherVitesse() qui nous renvoi rien et n' a aucun paramètre permet d'afficher la vitesse courante :

void afficherVitesse ()

{

System.out.println("la vitesse est " + vitesse);

}

exp2 : la méthode augmenterVitesse() qui nous renvoi rien et a comme paramètre la valeur d'augmentation est définit comme suit :

void augmenterVitesse (int v)

{

vitesse = vitesse + v;

}

Membre d'une classe :

On considère comme membre d'une classe les différents attributs et méthodes de cette classe.

Attribut = donnée membre.

Méthode = fonction membre.

- **Par convention** les noms des attributs et des méthodes commencent par une minuscule. Si le nom est un mot composé les autres mots doivent commencer par une majuscule (convention pas une obligation).

Des exemples :

Exemple 1 : soit la classe Rectangle permet de représenter un rectangle

Chaque Rectangle se caractérise par sa longueur et sa largeur, on peut calculer la surface et calculer le périmètre de ce rectangle

```
public class Rectangle {

	double longueur ;
	double largeur ;

	double calculerSurface()
	{
		double s ;
		 s = longueur * largeur;
		 return s ;
	}

	double calculerPerimetre ()
	{
		return (longueur + largeur) * 2 ;
	}
	}
```

Exemple 2 :

soit la classe voiture permet de représenter une voiture

Chaque voiture se caractérise par son matricule, sa vitesse et son nombre de places et sa couleur

Pour une voiture on peut :

- Augmenter ou diminuer la vitesse de la voiture par une valeur donnée.
- Afficher la vitesse de la voiture
- Colorer la voiture par un nouveau couleur

```
public class Voiture {
	String matricule ;
	int vitesse  ;
	int nbrPlaces ;
	String couleur  ;
	void afficherVitesse()
	{
		System.out.println("la vitesse de la voiture est " + vitesse);
	}
	void augmenterVitesse (int val)
	{
		vitesse  = vitesse  + val ;
	}
	void diminuerVitesse (int val)
	{
		vitesse  = vitesse  - val ;
	}

	void colorer (String nouveaucouleur )
	{
		couleur  = nouveaucouleur ;
	}

}
```

2- La création et l'utilisation d'objet :

La référence :

Les objets sont construits à partir de la classe, par un processus appelé l'instanciation, et donc, tout objet est une instance d'une classe (C'est la concrétisation d'une classe). Chaque instance commence à un emplacement mémoire unique (l'adresse de l'objet). C'est la référence de l'objet.

Quand le développeur a besoin d'un objet pendant un traitement, il doit :

- Déclarer et nommer une variable du type de la classe à utiliser ; pour pouvoir le manipuler.
- Instancier l'objet et enregistrer la référence de l'objet dans cette variable.

Chaque instance est unique. Par contre, plusieurs variables peuvent « pointer » sur une même instance.

C'est d'ailleurs quand plus aucune variable ne pointe sur une instance donnée que le ramasse-miettes enregistre cette instance comme devant être détruite.

Les constructeurs

Un constructeur est une méthode particulière qui est appelée automatiquement à la création d'un objet et qui permet d'initialiser correctement cet objet.

Elle permet d'exprimer le traitement nécessaire lors de la création d'un objet.

En java, les objets ne peuvent être crées que par allocation dynamique. Le constructeur est appelé lors de l'allocation de l'objet par new (et non lors de la déclaration de la variable). Il sert à initialiser quelque attribut d'objet lors de sa création et/ou pour faire quelques traitements lors de la création.

Un constructeur se reconnaît facilement en Java:

- il porte le même nom que la classe

- il n'a pas de type de retour (même pas void)

Si aucun constructeur n'est défini, la machine virtuelle appelle un constructeur par défaut vide sans paramètre créé implicitement.

Les destructeurs

Un destructeur permet d'exécuter du code lors de la libération de la place mémoire occupée par l'objet. En java, les destructeurs appelés finaliseurs (finalizers), sont automatiquement appelés par le garbage collector (ramasse-miettes). Pour créer un finaliseur, il faut redéfinir la méthode finalize() héritée de la classe Object.

L'instanciation

L'opération permettant de créer un objet à partir d'une classe s'appelle l'instanciation.

On dit qu'on a instancie (on a créé) un objet.

L'opérateur **new** est un opérateur de haute priorité qui permet d'instancier des objets et d'appeler le constructeur. Il fait appel à la machine virtuelle pour obtenir l'espace mémoire nécessaire à la représentation de l'objet puis appeler le constructeur pour initialiser l'objet dans l'emplacement obtenu. Il renvoie une valeur qui référence l'objet instancié.

Une instanciation consiste à :

1. Création et initialisation des attributs en mémoire (réservation d'espace mémoire).

2. Appel du constructeur.

3. Renvoi d'une référence sur l'objet (son identité) maintenant créé et initialisé.

Généralement (pas forcément) la référence de l'objet instancié (crée) doit être stockée dans une variable qui réfère (pointe sur) cet objet pour pouvoir le manipuler dans un programme plusieurs fois qu'on veut. Sinon on perde la trace de cet objet juste après l'instruction qui contient l'instanciation de cet objet.

Le type de la variable déclarée doit être la classe de cet objet.

Exemple : on déclare la variable r1 qui représente un objet de la classe Rectangle comme suit : Rectangle r1 ;

Syntaxe d'instanciation :

```
Nom_Classe nom_Variable_d'objet;
nom_Variable_d'objet = new nom_constructeur();
// les constructeur ont le même nom que la classe et peuvent avoir des arguments.
```

Exemple :

```
Rectangle r1 ;
r1 = new Rectangle (5,3);
```

- Il est possible de tout réunir en une seule déclaration :

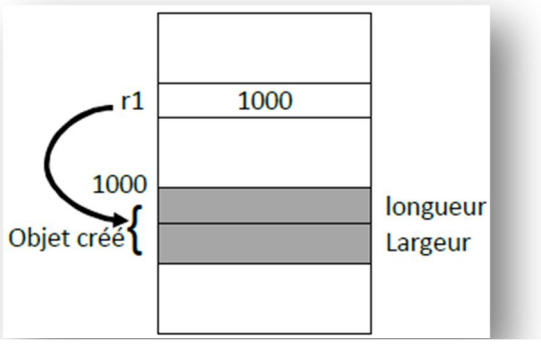
```
Nom_Classenom_Variable_d'objet = new nom_constructeur();
//les constructeur ont le même nom que la classe et peuvent avoir des arguments.
```

Exemple :

```
Rectangle r1 = new Rectangle (5,3) ;
```

r1 est la variable qui va pointer sur l'objet créé (elle réfère l'objet créé)

L'objet est créé à l'emplacement mémoire 1000. Cette référence est sauvegardée dans r1



L'accès aux attributs dans les méthodes de la même classe

Pour accéder à un attribut (utiliser un attribut) de **l'intérieur** de la classe (dans une méthode de la même classe), il suffit d'indiquer le nom de l'attribut que l'on veut y accéder.

Exemple : vitesse = 5;

L'appel d'une méthode dans les méthodes de la même classe

- pour invoquer une méthode dans l'une des méthodes de la même classe il suffit de faire l'appel classique en indiquant le nom de la méthode avec la liste des paramètres effectifs (Appel d'une fonction ou procédure classique).

Exemple : on suppose que le comportement d'une voiture est exprimé par les deux méthodes tourner () et clignoter () ;

Pour tourner il faut clignoter et diminuer la vitesse donc on doit appeler ces deux méthodes dans la méthode tourner.

```
Class Voiture {
Void clignoter () {
     System.out.println (˝la voiture clignote  ˝) ;
      }
void tourner ()   {
Clignoter () ;
System.out.println (˝la voiture  tourne ˝) ;
 }
}
```

L'émission de messages

Les objets se communiquent à travers l'envoi de message.

Un message est émis lorsqu'on demande à un objet d'exécuter l'une de ses méthodes.

On ne peut pas envoyer un message à un objet qui n'existe (l'objet doit être créé d'abord)

Il faut spécifier l'objet concerné en indiquant généralement le nom de la variable qui le réfère et préciser la méthode voulue en fournissant l'ensemble des paramètres nécessaires.

La syntaxe d'appel d'une méthode est : objet.nom_méthode(parametre1, …) ;

Exemple 1 : maVoiture.diminuerVitesse (20) ;

On dit que l'objet mavoiture a reçu le message diminuerVitesse ()

Exemple 2 : r1.calculerSurface() ;

On dit que l'objet r1 (objet représenté par r1) a reçu le message calculerSurface ()

Si la méthode appelée ne contient aucun paramètre, il faut laisser les parenthèses vides.

On peut aussi manipuler les attributs d'un objet de la même façon (s'ils sont visibles) en spécifiant d'abord l'objet concerné ensuite en indique l'attribut voulu comme suit :

Objet.non_attribut

Exemple :

System.out.println(‘’ la longueur du premier rectangle est ‘’ + **r1.longueur**) ;

Le mot clé « this »

Cette variable sert à référencer dans une méthode l'instance de l'objet en cours d'utilisation (l'objet courant). **this** est un objet qui est égale à l'instance de l'objet dans lequel il est utilisé.

- Lors de l'exécution d'une méthode Il désigne exactement l'objet qui a reçu le message.
- Lors de la création d'un objet il désigne l'objet créé.

La référence « this » peut être utile :

- Lorsqu'une variable locale ou un paramètre porte le même nom d'un attribut de la classe. Dans ce cas l'utilisation de this est obligatoire pour enlever l'ambigüité et distinguer entre l'attribut et l'autre variable.

- Pour appeler un constructeur depuis un autre constructeur.

Exemple :

```
Classe Nombre  {
int valeur;
Nombre (int valeur)
 {
this.valeur = valeur;
}
}
```

L'objet null

En Java, "**null**" est une valeur spéciale qui indique l'absence de référence à un objet. Cela signifie qu'elle ne pointe vers aucun objet en mémoire. On peut l'affecter à n'importe quelle variable de n'importe quelle classe ou on peut la donner comme paramètre.

Si on essaye d'utiliser un attribut ou une méthode d'une variable qui vaut **null** , on obtient une erreur.

Les références et la comparaison d'objets

Une variable qui appartient à une classe sert représenter un objet, elle le réfère (elle pointe sur cet objet). En d'autres termes elle contient l'adresse où l'objet est stocké (la référence de l'objet).

Si nous comparons deux variables appartenant à la même classe à l'aide des opérateurs de comparaison tels que l'opérateur d'égalité ˝== ˝, alors dans ce cas nous comparons les références (les adresses) des deux objets pas les deux objets eux mème car chaque variable contient la référence d'objet pas les valeurs de attributs.
Exemple :

```
Rectangle r1 = new Rectangle(100,50);
Rectangle r2 = new Rectangle(100,50);
Rectangle r3 ;
r3 = r1 ;
```

Dans cet Exemple on a créé deux objets pointés successivement par r1 et r2.
Si on suppose que la référence de l'objet pointé par r1 est 1000 et l'autre est 2000.
Donc :
r1== r2 vaut **faux** car on a comparé entre les contenus de ces deux variable : entre 1000 et 2000.
r1== r3 vaut **vrai** car les deux variables contiennent la même référence car on a copie le contenu de r1 (la référence du premier objet : 1000) dans r3 donc les deux variables r1 et r3 pointent sur le même objet (le premier objet).

Donc **pour faire la comparaison** entre les objets il faut définir une méthode qui garantit ça en comparant quelques attributs de ces objets (pas forcément tous les attributs).

Exemple :

```
boolean estPlusLong (Rectangle r2)
{       Return (longueur > r2.longueur ) ;       }
```

Généralement en Java pour faire la comparaison d'égalité entre deux objets il suffit de redéfinir la méthode **equals** hérité de la classe mère Object (ce n'est pas la peine de définir une autre méthode). C'est elle qui est utilisé pour comparer l'égalité entre les chaines de caractères.

3- Membres statiques et mot clés (final)

Les attributs statiques

Un attribut statique est un attribut marqué par le modificateur (mot clé) « static », il désigne un attribut commun entre toutes les instances de la classe.

Contrairement à un attribut normal ; un attribut statique est placé dans un espace mémoire commun à tous les objets de la classe. Cela signifie que toutes les instances de la classe partagent la même valeur pour cet attribut. Si un seul objet le modifie, sa valeur sera modifiée pour tous les objets de la classe.

Un attribut déclaré **`static`** existe dès que sa classe est chargée en mémoire, avant et indépendamment de toute instanciation.

En dehors de sa classe, une variable statique est accessible à l'aide du nom de la classe sans instancie la classe dans laquelle elle est définie En utilisant la notation suivante : nom_classe.nom_ariable_statique.

Au sein d'une même classe, ils sont accessibles directement.

Les attributs déclarés statique sont dite **Variable de classe**. Par contre les attributs déclarés non statique s'appellent **variable d'instance**.

Exemple :

Voici l'exemple de classe rectangle en ajoutant (autre que ses attribut longueur et largeur) un attribut statique appelé **nombre** qui va indiquer le nombre de rectangle créés.

```
class Rectangle
{
static int nombre = 0  ;  // l'initialisation par 0 est facultative car par défaut sa valeur initial est 0
private int longueur, largeur ;
Rectangle (int  longueur, int largeur) {
nombre++ ;
this. longueur = longueur ;
this.largeur = largeur ;
}
//…….Les autres méthodes
}
classTest_Rec
{
public static void main (String [] args)
{System.out.println ("Nombre des rectangles  = " + Rectangle.nombre) ; // affiche 0
Rectangle r1 = new Rectangle (10,5);
System.out.println ("Nombre des rectangles = " + Rectangle.nombre) ; // affiche 1
Rectangle r2 =  new Rectangle (14);
System.out.println ("Nombre des rectangles = " + Rectangle.nombre) ; //affiche 2
}     }
```

Les méthodes statiques

S'appellent aussi **méthodes de classe**, sont des méthodes marqué par le modificateur « static ».

Une méthode statique peut être appelée sans créer une instance de sa classe. Elle n'agit que ses paramètres et ses variables locales ainsi que les variables de la classe (attributs statiques) mais pas sur les variables d'instance.

Elle est appelée en dehors de sa classe en précisant le nom de sa classe en donnant les paramètres nécessaires. Selon la notation suivante : **nom_classe.nom_méthode_statique(paramètre1, …)** ; c'est la notation la plus recommandée.

On peut l'appeler aussi (en dehors de sa classe) en utilisant une instance de cette classe mais c'est rarement utilisé (pas recommandée). Cependant cette méthode n'agit pas sur les variables d'instance de cette dernière (l'objet qui a reçu l'appel).

Le code d'une méthode statique ne doit ni manipuler des attributs non statiques de sa la classe (variable d'instance), ni appeler directement d'autres méthodes non statiques de la même classe, ni utiliser la référence this.

Exemple :

La classe Math a plusieurs méthodes statiques tel que la méthode sqrt() , la méthode sin() , …

Il suffit pour les invoquer d'utiliser le nom de la classe accompagné par le nom de la méthode voulue en fournissant les paramètres nécessaires par exemple : **Math.sqrt(4) ;**

Les constantes et le mot clé « final »

Le mot clé final s'applique aux variables et aux méthodes et aux classes.

Une variable déclarée final, est une variable dont la valeur ne peut pas être modifiée une fois qu'elle a été initialisée. En d'autres termes, une fois qu'une variable finale a été attribuée une valeur, cette valeur ne peut plus être changée.

Il est également possible de déclarer une constante comme une variable de classe (**constante de la classe**)en utilisant les deux modificateurs **static** (pour le rendre partagé entre toute les instance) et **final** (pour le rendre immodifiable une fois il est initialisé).

Par convention : le nom des constants d'une classe doit être complètement en majuscules avec les mots séparés par des tirets de soulignement (_).

Exemple :

- Le constant PI de la classe Math vaut 3.14 dans un Programme en utilise comme suit : Math.PI
- Soit la classe Chimie

```
class Chimie
{
        final double AVOGADRO = 6.023e23;
    // les autres attribus et méthodes
}
```

4-Les packages

En java, il existe un moyen de regrouper des classe voisines ou qui couvrent un même domaine : ce sont les packages. Un package peut être considéré comme une bibliothèque des classes : il permet de regrouper un ensemble de classes relativement liées.

Ils permettent de rendre les classes plus faciles à trouver et à comprendre, et pour faciliter la gestion de grands projets Java.

Les packages sont organisés en une hiérarchie. Dans un package, il peut avoir plusieurs sous-packages, et ces derniers peuvent encore contenir d'autres sous-sous-packages… Dans une hiérarchie les noms du packages sont séparés par un point « . » allant de plus haut niveau jusqu'à le plus bas par exemple le package : java.util

Création d'un package

Pour réaliser un package :

1. Ajouter dans chaque fichier ".java" composant le package la ligne : ***package <nom_du_package>;***

2. Enregistrer le fichier dans un répertoire portant le même nom que le package.

Le mot clé package doit être la première instruction dans un fichier source et il ne doit être présent qu'une seule fois dans le fichier source (une classe ne peut pas appartenir à plusieurs packages).

L'utilisation d'une classe qui appartient à un package

On a deux façons pour désigner une classe qui appartient à un autre package :

1- Faire précéder chaque occurrence du nom de la classe par le nom du package (chemin du package) dans lequel elle est définie. Comme suit : **chemin_du_package.nom_classe**
2- importer la classe à partir de sa package en utilisant le mot clé **import**

Pour importer, il y a deux méthodes si le chemin de recherche est correctement renseigné :

Syntaxe	**Rôle**
import nomPackage.*;	toutes les classes du package sont importées
import nomPackage.nomClasse;	appel à une seule classe : l'avantage de cette notation est de réduire le temps de compilation

Attention : l'astérisque (l'étoile) n'importe pas les sous paquetages. Par exemple, il n'est pas possible d'écrire import java.*.

Remarque :

- Le compilateur implémente automatiquement une commande import lors de la compilation d'un programme Java même si elle ne figure pas explicitement (importation implicite) au début du programme :import java.lang.*; Ce package contient entre autre les classes de base de tous les objets java (les classes : String, System, Object, Math etc.).
- Un package par défaut est systématiquement attribué par le compilateur aux classes qui sont définies sans déclarer explicitement une appartenance à un package. Ce package par défaut correspond au répertoire courant qui est le répertoire de travail.

La collision de classes

Deux classes entrent en collision lorsqu'elles portent le même nom mais qu'elles sont définies dans des packages différents. Dans ce cas, il faut qualifier explicitement le nom de la classe avec le nom complet du package.

Les packages et l'environnement système

Les classes Java sont importées par le compilateur (au moment de la compilation) et par la machine virtuelle (au moment de l'exécution). Les techniques de chargement des classes varient en fonction de l'implémentation de la machine virtuelle. Dans la plupart des cas, une variable d'environnement CLASSPATH référence tous les répertoires qui hébergent des packages susceptibles d'être importés.

L'importation des packages ne fonctionne que si le chemin de recherche spécifié dans une variable particulière pointe sur les packages, sinon le nom du package devra refléter la structure du répertoire où il se trouve. Pour déterminer l'endroit où se trouvent les fichiers .class à importer, le compilateur utilise une variable d'environnement dénommée CLASSPATH. Le compilateur peut lire les fichiers .class comme des fichiers indépendants ou comme des fichiers ZIP ou JAR dans lesquels les classes sont réunies et compressées.