

Programmation orientée-objet: Java

Présenté par:

Pr. Samba DIAW

Département Génie Informatique /ESP /UCAD

Page web: <https://sites.google.com/a/esp.sn/sdiaw>



Sommaire

1. Présentation de l'approche objet

- Histoire de la POO
- Notion d'objet
- Notion de classe

2. Codage en Java

- Introduction
- Variable et Types primitifs
- La méthode principale (main)
- Les opérateurs
- Les classes
- Structures de contrôle
- Chaines de caractères
- Le type Date
- Tableaux (vecteurs et matrices)
- Les énumérations
- Interface et classe d'implémentation
- Les exceptions
- Connexion JDBC

Introduction

- Ecrire un **programme informatique** revient à **coder dans un langage de programmation**.
- Les programmes s'appuient sur des données et sur la logique pour fonctionner.
- Pour cela, le programmeur doit dire à l'ordinateur ce qu'il doit faire et comment il doit le faire.



Paradigme Objet

■ Histoire de la POO

- 1967 : Simula fut le premier langage de programmation à implémenter le concept de type abstrait à l'aide de classes
- 1976 : Smalltalk implémente les concepts fondateurs de l'approche objet (encapsulation, agrégation, héritage) à l'aide de :
 - classes
 - associations entre classes
 - hiérarchies de classes
 - messages entre objets
- 1980 : le 1^{er} compilateur C++ est normalisé par l'ANSI et depuis lors de nombreux langages orientés objets académiques implémentent les concepts objets : Eiffel, Objective C, Loops, Java, Python, Ruby, C#,



Concepts de base

■ Notion d'objet

- Représentation abstraite d'une entité du monde réel ou virtuel
- Caractéristique fondamentales d'un objet (informatique)

Objet = État + Comportement + Identité

■ État

Un étudiant

INE = 2019HT

Prénom = Adama

Nom = SECK

Age = 22

- Regroupe les valeurs instantanées de tous les attributs d'un objet : Exemple : Un objet Etudiant regroupe les valeurs des attributs INE, Prénom, Nom et Age

■ Comportement:

- Regroupe les compétences d'un objet (service proposés par l'objet)

■ Identité

- Elle permet d'identifier sans ambiguïté deux objets qui ont le même état

■ Notion de Classe

□ Définition

- Une classe décrit une abstraction d'objets ayant
 - Des propriétés similaires
 - Un comportement commun
 - Des relations identiques avec les autres objets
 - Une sémantique commune
- Par exemple Etudiant (resp. Filière, resp. Cours) est la classe de tous les étudiants (resp. filières, resp. cours)

■ Notion de Classe

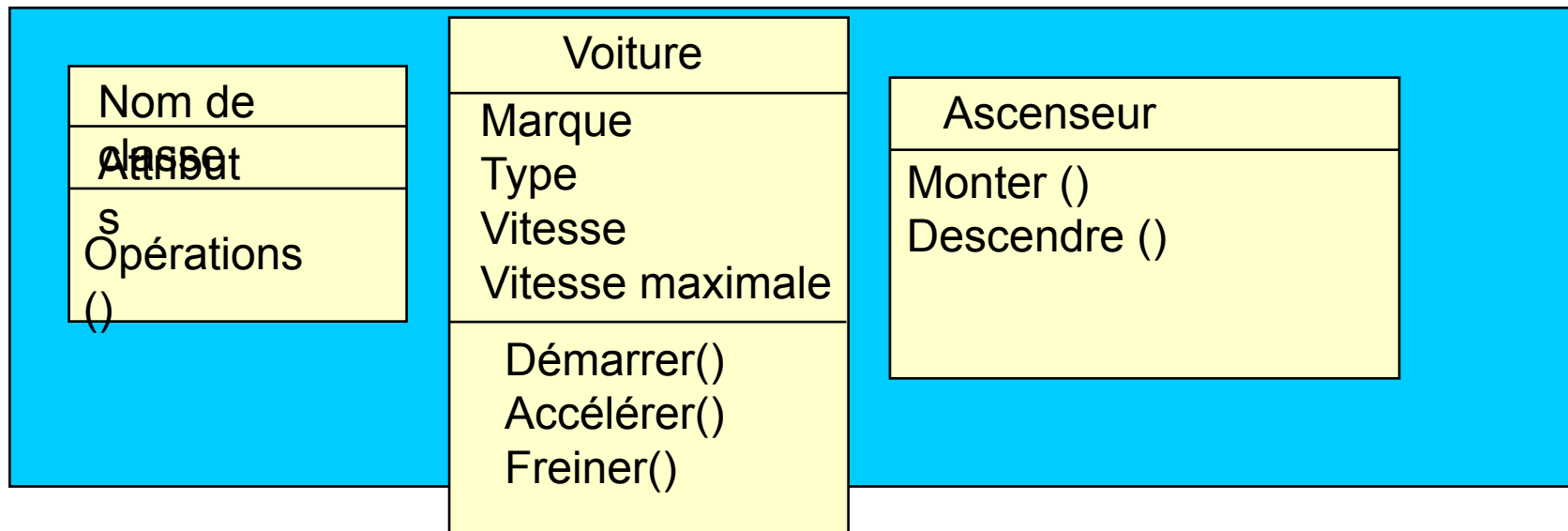
□ Caractéristiques d'une classe

- Un objet créé par (on dit également appartenant à) une classe sera appelé une *instance de cette classe* ou *variables d'instances*
- Les généralités sont contenues dans la classe et les particularité sont contenues dans les objets
- Les objets sont construits à partir des classes, par un processus appelé instantiation : tout objet est une instance de classe
- Nous distinguons deux types de classes
 - **Classe concrète** : peut être instanciée
 - **Classe abstraite** : est une classe non instanciable

■ Classe

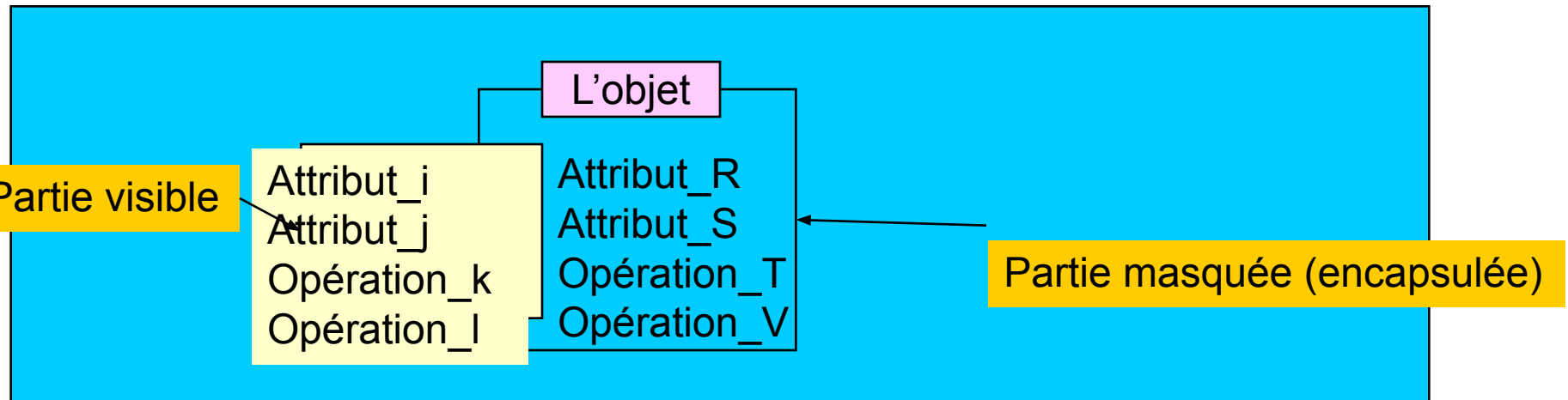
□ Représentation graphique d'une classe en UML

- Chaque classe est représentée sous la forme d'un rectangle divisé en trois compartiments
- Les compartiments peuvent être supprimés pour alléger les diagrammes



■ Encapsulation

- Consiste à masquer les détails d'implémentation d'un objet en définissant une interface
- L'interface est la vue externe d'un objet (spécification), elle définit les services accessibles (offerts) aux utilisateurs de l'objet.
- Elle permet de garantir l'intégrité des données en restreignant l'accès direct aux attributs des objets



Règle de visibilité

+ Attribut public
Attribut protégé
- Attribut privé
Attribut de niveau package

+ Opération publique()
Opération protégée()
- Opération privée()

Salarié

+ nom
age
- salaire

+ donnerSalaire()
changerSalaire()
- calculerPrime()

Encapsulation

- Il est possible d'assouplir le degré d'encapsulation en définissant des niveaux de visibilité
- Les niveaux de visibilité sont:
 - **Niveau privé (-)** : c'est le niveau le plus fort ; la partie privée de la classe est totalement opaque et n'est pas visible aux autres objets. Un attribut défini dans la partie privée d'une classe n'est visible que dans cette classe. Pour toutes les autres classes, l'attribut reste invisible
 - **Niveau Paquetage**: Un attribut défini dans ce niveau est visible dans toutes les classes appartenant au même paquetage

■ Encapsulation

Règle de visibilité

+ Attribut public
Attribut protégé
- Attribut privé
Attribut de niveau package

+ Opération publique()
Opération protégée()
- Opération privée()

- **Niveau protégé (#)** : les attributs placés dans la partie protégée sont visibles
 1. dans toutes les classes appartenant au même paquetage
 2. dans toutes les classe filles de la classe mère qui contient l'attribut
- **Niveau public (+)** : Ce niveau revient à se passer de la notion d'encapsulation et de rendre visibles les attributs pour toutes les classes

Employe

+ nom
prénom
- salaire

+calculerSalaire()
changerSalaire()
- calculerPrime()



Codage en Java

Sommaire

1. Introduction
2. Variable et Types primitifs
3. La méthode principale (main)
4. Les opérateurs
5. Les classes
6. Structures de contrôle
7. Chaines de caractères
8. Les Dates
9. Tableaux (vecteurs et matrices)
10. Les énumérations
11. Interface et classe d'implémentation
12. Les exceptions
13. Connexion JDBC

Introduction

- Java est un langage de programmation orientée-objet multi-plateforme
- Syntaxe de Java est inspirée de celle du langage C
- Java est sensible à la casse
- Les blocs de code sont encadrés par des accolades
- Chaque instruction se termine par un ;
- Une instruction peut se tenir sur plusieurs lignes



Variables , Types et Opérateurs en Java

Variable

- Variable est assimilable à une boîte qui contient une donnée (valeur)
- L'adresse de la variable en mémoire peut être assimilée à l'emplacement d'une boîte dans un entrepôt de donnée
- Variable est décrite par son identificateur appelé aussi nom de la variable et son type
- Le nom d'une variable doit être le plus explicite possible

Variable

- Le premier caractère doit être une lettre, le caractère de soulignement ou le signe dollar
- Un identificateur ne peut pas appartenir à la liste des mots réservés en Java
- Exemple


```
1  int nombre;  
2  long _x1;  
3  String $test;
```

Les types primitifs

- boolean : Valeur logique true or false
- byte: octet signé (8 bits): -128 à 127
- short: entier court signé (16 bits)
- char: caractère unicode (16 bits)
- int: entier signé (32 bits)
- long: entier long (64 bits)
- float: virgule flottante simple précision
- double: virgule flottante double précision

Mon premier Programme

 Variable.java >  Variable

```
1  public class Variable {  
    Run | Debug  
2  |  public static void main(String[] args) {  
3  |   | int nombre = 2;  
4  |   | System.out.println(nombre);  
5  |   | }  
6  |  
7  | }  
    |
```

Paramètres de la méthode principale

Principale.java > Principale

```
1 public class Principale
2 {
    Run | Debug
3 public static void main (String [] args)
4 {
5     //Affichage des deux paramètres de la méthode principale
6     System.out.println (args [0] + "\t" + args[1]);
7 }
8 }
```

Les opérateurs

1. Opérateurs arithmétiques (+, *, /, -, %)
2. Opérateurs d'assignation (=; +=, -=)
3. Opérateurs de comparaison (<, >, <=, >=, ==, !=)
4. Opérateurs bit à bit: (&, ^, |)
5. Opérateurs logiques: (&&, ||, !)
6. Opérateurs d'incrémentement et de décrémentement (++ , --)
7. ?: Opérateur ternaire condition ? b:c
 - Renvoie b si condition est vraie et c sinon

Priorité des opérateurs

1. Parenthèses ()
2. opérateur d'incrémentement: ++
3. opérateur de décrémentement: --
4. Opérateurs arithmétiques: *,/,%
5. Opérateurs de comparaison ou relationnels: <, >, <=, >=, ==, !=
6. Opérateurs logiques bit à bit: ^, &, |
7. Opérateurs logiques booléens: &&, ||, !
8. Opérateurs d'assignement ou d'affectation composée (=; +=, -=)



Définition d'une classe Java

Etudiant

- INE: String
- Prénom: String
- Nom: String

Classe en JAVA

```
public class Etudiant {  
    private String Ine;  
    private String Prénom;  
    private String Nom;  
}
```

Définition des méthodes

```
public class Etudiant {  
    //définition des attributs  
    private String Ine;  
    private String Prénom;  
    private String Nom;  
    //définition des méthodes  
    public String getIne () { return Ine; }  
    public void setIne (String vIne)  
    { Ine=vIne; }  
}
```

Etudiant

-INE: String
-Prénom: String
-Nom: String

+getINE (): String
+setINE (String): void



Constructeur d'une classe Java

Instanciation

- On appelle instance d'une classe, un objet avec un comportement et un état, tous deux définis par sa classe.
- L'instanciation est l'opération qui consiste à créer un objet à partir d'une classe En Java, le mot-clé **new** provoque une instanciation en faisant appel à un constructeur de la classe instanciée

Constructeur

- Un constructeur est une méthode qui a le même nom que la classe et qui n'a pas de valeur de retour
- Plusieurs constructeurs peuvent exister dans une même classe (avec des arguments différents)
- Il faut au moins un constructeur dans une classe pour en instancier des objets
- L'appel au constructeur affecte une nouvelle adresse en mémoire pour le nouvel objet créé

Constructeur sans paramètre

```
package ecole;

public class Etudiant {
    //définition des attributs
    private String Ine;
    private String Prénom;
    private String Nom;
    //définition du constructeur
    public Etudiant () {}
    //définition des méthodes
    public String getIne () { return Ine; }
    public void setIne (String vIne)
    {Ine=vIne;}
}
```

Etudiant

-INE: String
-Prénom: String
-Nom: String

+getINE (): String
+setINE (String): void

Constructeur avec paramètres

```
package ecole;

public class Etudiant {
    //définition des attributs
    private String Ine;
    private String Prénom;
    private String Nom;
    //définition du constructeur sans paramètre
    public Etudiant () {}
    //définition du constructeur avec paramètre
    public Etudiant( String vIne, String vPrénom, String vNom)
    {
        Ine=vIne;
        Prénom=vPrénom;
        Nom=vNom;
    }
    //définition des accesseurs
}
```


Etudiant

-INE: String
-Prénom: String
-Nom: String

+getINE (): String
+setINE (String): void

Exemple d'instanciation

```
1 //import ecole.Etudiant; //importation d'une classe
2 public class TestEtudiant {
3     public static void main (String [] args){
4         Etudiant E1; // déclaration
5         E1 = new Etudiant (); //allocation (réservation en mémoire)
6         //Etudiant E = new Etudiant (); déclaration et allocation
7         //initialisation
8         E1.setIne("2019FH01"); ///E1.Ine="2019FH01" si l'attribut est public
9         E1.setNom ("Fall");
10        E1.setPrenom ("Moussa");
11        //affichage
12        System.out.println ("Données de E1");
13        System.out.println ("L'INE est :\t"+ E1.getIne());
14        System.out.println ("Le nom est :\t"+ E1.getNom());
15        System.out.println ("Le prénom est :\t"+ E1.getPrenom());
16        //déclaration, allocation et initialisation
17        Etudiant E2 = new Etudiant ("2019TH01", "Fatou", "Seck");
18        //affichage
19        System.out.println ("Données de E2");
20        System.out.println ("L'INE est :\t"+ E2.getIne());
21        System.out.println ("Le nom est :\t"+ E2.getNom());
22        System.out.println ("Le prénom est :\t"+ E2.getPrenom());
23    }
24 }
```



Les structures de contrôle

Structures de contrôle

■ Les boucles

- Boucle for
- Boucle while
- Boucle do while

■ Les branchements conditionnels

- L'instruction if
- L'instruction switch

Exemple d'utilisation de for

J Factoriel.java >  Factoriel

```
1  public class Factoriel {  
    Run | Debug  
2  public static void main (String [] args)  
3  {  
4      //Affichage des deux paramètres de la méthode principale  
5      int fact =1;  
6      int i;  
7      for ( i= 1; i<= 4; i++)  
8          fact = fact * i;  
9      System.out.println("Le Factoriel de 4 est : "+ fact);  
10  
11  
12 }  
13 }
```

Exemple d'utilisation de while

```
1  public class Reste {  
    Run | Debug  
2  public static void main (String [] args)  
3  {  
4      //reste de la division de 10 par 3  
5      int reste =10 ;  
6      while ( reste >= 3)  
7      |     reste =reste -3;  
8      System.out.println("Le reste de la division de 10 par 3 est : "+ reste);  
9  
10 }  
11 }
```

Exemple d'utilisation de do..while

```
1 public class Somme {  
  Run | Debug  
2 public static void main (String [] args)  
3 {  
4     // Somme des 5 premiers entiers;  
5  
6     int somme=0;  
7     int i= 1;  
8     do {  
9  
10        somme = somme +i;  
11        i++;  
12  
13    } while (i<= 5);  
14    System.out.println("La somme des 5 premiers entiers est : "+ somme);  
15 }  
16 }
```

Exemple d'utilisation du branchement if

```
1 public class TestIf
2 {
  Run | Debug
3 public static void main (String [] args)
4 {
5     //Test de la condition if
6     int b = -4;
7     if ( b > 0 )
8         System.out.println("La variable b a une valeur strictement positive ");
9     else if ( b <= -5)
10        System.out.println("La variable b est comprise entre - l'infini et -5 fermé ");
11    else
12        System.out.println("La variable b est comprise entre -5 ouvert et 0 fermé ");
13
14 }
15 }
```


Exemple d'utilisation du branchement switch

TestSwitch.java / TestSwitch

```
1  public class TestSwitch
2  {
    Run | Debug
3  public static void main (String [] args)
4  {
5      //Test de la condition if
6  char sexe = 'F';
7  switch (sexe)
8  {
9      case 'M' : System.out.println("Masculin "); break;
10     case 'F' : System.out.println("Féminin"); break;
11     default: System.out.println("Erreur ");
12 }
13 }
14 }
```




Classe Scanner (gestion des entrées)

Classe Scanner

Test.java > Test > main(String[])

```
1  import java.util.Scanner;
2  public class Test
3  {
    Run | Debug
4  public static void main (String [] args)
5  {
6  Scanner sc= new Scanner (System.in);
7  //next () lit seulement la chaine précédant le caractère espace
8  // next () place le curseur sur la même ligne après lecture
9  System.out.println ("Saisir un nom: ");
10 String nom = sc.next();
11 System.out.println ("Le nom saisi est : " + nom);
12 //nextline () lit la chaine comportant des espaces
13 //nextline () positionne le curseur à la ligne suivante après lecture
14 System.out.println ("Saisir un prénom: ");
15 sc.nextLine(); //Il faut provoquer obligatoirement le retour de ligne avant d'utiliser nextLine ()
16 String prenom = sc.nextLine();
17 System.out.println ("Le prénom saisi est : " + prenom);
18 //Saisie d'un entier
19 System.out.println ("Saisir un entier ");
20 int i = sc.nextInt();
21 System.out.println ("L'entier saisi est : " + i);
22 sc.close();
23 }
24 }
```



Les chaines de caractères

Chaines de caractères

- Les variables de Type String sont des objets
- Si une chaine de caractères est déclarée avec une constante: le compilateur génère un objet de type String avec le contenu spécifié
- `boolean equals(String)` : Comparaison de deux chaines
- L'égalité (`==`) compare les adresses et non les contenus

Fonctions sur les chaines

- Concaténation de deux chaines (+)
- Remplacement
 - **String replace (char c, char x)** remplace le caractère c par le caractère x dans la chaine
- Caractère à une position :
 - **char charAt(j)** retourne le caractère à la position j
- Sous-chaine
 - **String subString(int i, int j)** retourne la sous-chaine de la position i à la position j-1
 - **String subString(int i)** retourne la sous-chaine de la position i à la position fin

Méthodes de la classe String

- Transformer une chaîne en majuscule
 - **String toUpperCase()**
- Transformer une chaîne en minuscule
 - **String toLowerCase()**
- Fusion de deux chaînes
 - **String concat (String)**
- Covertir une variable en String
 - **String valueOf (Type)**
 - *Type: int, long, char, boolean, double, float, Object, char [], etc.)*



Conversion entre types

Conversation implicite

- byte vers short
- short vers int
- int vers long
- float vers double

```
16  byte  a =2;  
17  short b=a;  
18  int   c=b;  
19  long  d=c;  
20  
21  float x;  
22  double z= x;  
23
```


Formatage et conversion

Conversion.java > Conversion > main(String[])

```
1  import java.text.DecimalFormat;
2  import java.text.NumberFormat;
3  public class Conversion
4  {
    Run | Debug
5  public static void main (String [] args)
6  {
7  //définition d'un objet formater avec deux chiffres après la virgule
8  //Nombre de #
9  NumberFormat formatter = new DecimalFormat("#.##");
10 //formatage d'un nombre réel pour deux chiffres après la virgule
11 String ch = formatter.format (12.54622222f);
12 System.out.println ("Après formatage on la chaine suivante : "+ch);
13 ch= ch.replace ('.', '#'); //remplacer la virgule par un point
14 ///Pour Transformer une chaine en float il faut
15 //veiller à ce que les caractères de la chaine soient des caractères
16 //reconnus par un float pas de virgule ni de lettres
17 float reel = Float.parseFloat(ch);
18 System.out.println ("Transformation de String en float donne: "+ reel);
19 //Transformer un float en une chaine
20 String s =String.valueOf(reel);
21 System.out.println ("Transformation de float vers String donne: "+s);
22 //transformer une chaine en entier
23 int entier= Integer.parseInt("1936");
24 System.out.println ("Transformation de String en int donne: "+ entier);
```



Le type Date

Les Dates

- `Java.util.Date` (pour les dates et heures)
- `Java.sql.Date` (pour les dates uniquement sans les informations de l'heure)

Formatage des dates

```
1  import java.text.SimpleDateFormat;
2  import java.util.Date;
3  //import java.sql.Date;
4  public class TestDate
5  {
6      Run | Debug
7      public static void main (String [] args){
8          SimpleDateFormat formater = null;
9          Date aujourd'hui = new Date();
10         formater = new SimpleDateFormat("dd-MM-yy");
11         System.out.println(formater.format(aujourd'hui));
12         formater = new SimpleDateFormat("EEEE, d MMM yyyy");
13         System.out.println(formater.format(aujourd'hui));
14         formater = new SimpleDateFormat("'le' dd/MM/yyyy 'à' hh:mm:ss");
15         System.out.println(formater.format(aujourd'hui));
16         formater = new SimpleDateFormat("'le' dd MMMM yyyy 'à' hh:mm:ss");
17         System.out.println(formater.format(aujourd'hui));
18         formater = new SimpleDateFormat("dd MMMMM yyyy GGG, hh:mm aaa");
19         System.out.println(formater.format(aujourd'hui));
20     }
```



Les tableaux (Vecteurs et Matrices)

Les tableaux

- Ce sont des objets: ils sont dérivés la classe Object. Il est possible d'utiliser les méthodes héritées telles que equals () ou getClass().
- Le premier élément d'un tableau possède l'indice 0
- length détermine la taille d'un tableau

Vecteur ou Tableau

- Java permet de placer les crochets avant ou après le nom du tableau
- Déclaration puis allocation

```
int T1[];  
T1= new int [20];
```

- Déclaration et allocation

```
int T1[]= new int [20];
```


Initialisation explicite

```
//initialisation explicite
int T1[]={4,5,1,2};
//Affichage verticale
for (int i=0; i<T1.length; i++)
System.out.println(T1[i]);
//Affichage horizontale
for (int i=0; i<T1.length; i++)
System.out.print(T1[i]+ " \t");
```


Initialisation implicite

```
//initialisation implicite
```

```
import java.util.Scanner;  
Scanner sc=new Scanner(System.in);  
System.out.println("Taille du tableau ");  
int n =sc.nextInt();  
int T2[]= new int [n];  
for (int i=0; i<T2.length; i++)  
{  
System.out.println("Elément position "+i);  
T2[i]=sc.nextInt();  
}
```

```
//Affichage horizontale
```

```
for (int i=0; i<T2.length; i++)  
System.out.print(T2[i]+ " \t");
```

Déclaration (matrice)

- Java permet de placer les crochets avant ou après le nom du tableau
- Déclaration puis allocation

```
int m1[] [];  
m1= new int [10] [10];
```

- Déclaration et allocation

```
int m1[] []= new int [10] [10];
```

Déclaration

- matrice dont les vecteurs n'ont pas le même nombre
- Déclaration et allocation

```
14  int m1[] [] = new int [3] [];  
15  m1[0]=new int [4];  
16  m1[1]=new int [5];  
17  m1[2]=new int [2];
```

- Initialisation explicite d'une matrice

```
int mat[][]={{4,5,2}, {9,10,11}, {12,13,15}};  
int mat[][]={{4,5}, {9,10,11}, {12,13,15,18}};
```

Initialisation d'une matrice carrée

```
public static void main(String[] args) {  
    int mat [][];  
    Scanner in= new Scanner (System.in);  
    System.out.print("Donner le nombre de vecteurs ");  
    int n=in.nextInt();  
    mat=new int[n] [n]; //matrice carrée  
    for (int i=0; i<n; i++)  
    {  
        for (int j=0; j<n; j++)  
        {  
            System.out.print("Donner la valeur de l'éléement "+ i+ ", "+j+" ");  
            int val=in.nextInt();  
            mat[i][j]=val;  
        }  
    }  
}
```

Initialisation d'une matrice non carrée

```
public static void main(String[] args) {  
    int mat [][];  
    Scanner in= new Scanner (System.in);  
    System.out.print("Donner le nombre de vecteurs ");  
    int n=in.nextInt();  
    mat=new int[n] [];  
    for (int i=0; i<n; i++)  
    {  
        System.out.print("Donner le nombre d'elements du vecteur numéro "+ i+ " ");  
        int m=in.nextInt();  
        mat[i]=new int [m];  
        for (int j=0; j<m; j++)  
        {  
            System.out.print("Donner la valeur de l'éléement "+ i+ ", "+j+ " ");  
            int val=in.nextInt();  
            mat[i][j]=val;  
        }  
    }  
}
```


Palindrome

```
public static void main(String[] args) {  
    // TODO Auto-generated method stub  
    //String T="rotor";  
    Scanner entree=new Scanner(System.in);  
    String T;  
    System.out.print(" Saisissez une chaine de caractère: ");  
    T=entree.next();  
    T=T.toUpperCase();  
    int i=0;  
    int j=T.length()-1;  
    while(i<j && T.charAt(i)==T.charAt(j))  
    {i++; j--;}  
    if (T.charAt(i)!=T.charAt(j))  
        System.out.println(T +" n'est pas un palindrome");  
    else  
        System.out.println(T +" est un palindrome");  
} // Fin Main
```

Fonction Palindrome

```
public static boolean estunPalindrome(String ch)
{
    int i=0;
    int j=ch.length()-1;
    while(i<j && ch.charAt(i)==ch.charAt(j))
    {
        i+=1;
        j-=1;
    }
    if (ch.charAt(i)!=ch.charAt(j))
        return false;
    else
        return true;
}
```



Les énumérations

Les énumérations

- Les énumération sont des types dont l'ensemble des valeurs n'est pas indéfini

```
package enumeration;
//définition d'une énumération
public enum EnumSexe {
    M, F;

    //test de l'énumération
public class TestEnumSexe{
    public static void main (String [] args)
    {
        for (EnumSexe sexe:EnumSexe.values())
            System.out.println(sexe+"\t"+sexe.ordinal());
    }
}
```

Tester une énumération

```
package Enumeration;
import java.util.Scanner;
//tester l'énumération EnumSexe
public class EssaiEnumSexe {
public static void main(String[] args) {
    System.out.print("Donner le Sexe: ");
    String vsex=Scanner.next();
    EnumSexe sexe=EnumSexe.valueOf(vsex);
    if (sexe==EnumSexe.M)
        System.out.print("Masculin");
    else if (sexe==EnumSexe.F)
        System.out.print("Feminin");
    else
        System.out.print("Erreur");
    }//fin main
} // fin classe
```

Définition d'une énumération en JAVA

```
package enumeration;  
//définition de l'énumération EnumJour  
public enum EnumJour{  
    LUNDI, MARDI, MEcredi, JEUDI,  
    VENDREDI, SAMEDI, DIMANCHE;  
}
```

Tester une énumération

```
package Enumeration;
import java.util.Scanner;
public class EssaiEnumJour {
    /**
     * @param args
     */
    //Affiche les objets de l'énumération EnumJour
    private static void afficherEnumJourSem()
    {
        for (EnumJour j:EnumJour.values())
            System.out.print(j+"(" + j.ordinal()+")"+"", " ");
        System.out.println();
    }
}
```

Tester une énumération

```
private static boolean Test(String param)
{
    boolean trouve=false;
    EnumJour ji=EnumJour.valueOf(param);
    for (EnumJour j:EnumJour.values())
        if (j==ji)
        {
            trouve=true;
            break;
        }
    return trouve;
}
```


Tester une énumération

```
public static void main(String[] args) {  
    // TODO Auto-generated method stub  
    System.out.println("Affichage de l'énumération EnumJour");  
    afficherEnumJourSem();  
    Scanner in=new Scanner(System.in);  
    System.out.print("Donner un Jour de la semaine: ");  
    String jour=in.next();  
    EnumJour j=EnumJour.valueOf(jour);  
    if (j==EnumJour.SAMEDI)  
        System.out.print("Fin de semaine");  
    switch(j)  
    {case SAMEDI:    System.out.print("Fin de semaine");  
      break;  
      case DIMANCHE: System.out.print("Jour de repos");  
      break;  
      default: System.out.print("Jour de travail");  
    } //fin switch
```



Héritage

Définition d'une classe concrète

```
public class Personne
{
    protected String nom;
    protected String prenom;
    public Personne () {}
    public Personne (String vnom, String vprenom)
    {
        nom=vnom;  prenom=vprenom;
    }
    public String getNom()
    { return nom;}
    public String getPrenom()
    {return prenom;}
    public void setNom( String vnom)
    {nom=vnom;}
    public void setPrenom( String vprenom)
    {prenom=vprenom;}
}
```


Extension d'une classe concrète

```
public class Etudiant extends Personne {  
    private String ine;  
    public Etudiant () {}  
    public Etudiant (String vine, String vnom, String vprenom)  
    {  
        ine=vine;  
        nom=vnom;  
        prenom=vprenom;  
        //super (vnom, vprenom);  
    }  
    public String getine() {return ine;}  
    public void setine(String vine) {ine=vine;}  
    //public String getNom() {return nom;}  
}
```

Classe abstraite

- Une classe abstraite contient au moins une méthode abstraite
 - Une méthode abstraite est une méthode dont on connaît juste la signature (pas de corps)

Définition d'une classe abstraite

```
1  public abstract class  Personne {
2
3      protected String nom;
4      protected String prenom;
5      // public Personne () {}
6
7      //accesseurs en lecture
8      //méthodes get sont abstraites
9      public abstract String getNom () ;
10     public abstract String getPrenom ();
11     //accesseurs en écriture
12     // public void setNom (String vnom) {this.nom=vnom;}
13     public abstract void setNom (String vnom);
14     public void setPrenom (String vprenom) {this.prenom =vprenom;}
15 }
```

Extension d'une classe abstraite

```
1  public class Etudiant extends Personne {
2  private String ine;
3      //constructeurs
4      public Etudiant () {}
5      public Etudiant ( String vine, String vnom, String vprenom)
6      { this.ine =vine;
7        this.nom= vnom; // si Nom est privé this.setNom (vnom);
8        this.prenom= vprenom; //si prénom est private this.setPrenom (vpr
9      }
10     //accesseurs
11     public String getIne () { return this.ine;}
12     public void setIne (String vine) {this.ine =vine;}
13
14     //implémentaion des méthodes abstraites héritées de la classe Personne
15     public String getNom () {return this.nom;}
16     public String getPrenom () { return this.prenom;}
17     public void setNom (String vnom) {this.nom= vnom;}
```

Classe abstraite (Suite)

```
18 public static void main(String[] args) {
19     Etudiant e = new Etudiant();
20     e.setIne("2019FG007");
21     e.setNom("DIOP");
22     e.setPrenom ("Moussa");
23     //affichage
24     System.out.println ("L'ine est : "+ e.getIne());
25     System.out.println("Le nom est :"+e.getNom());
26     System.out.println("Le prénom est : "+e.getPrenom ());
27
28 }
29
30 }
```



Interface et classe d'implémentation

Définition d'une interface

- Une interface ne contient que des méthode abstraites et des constantes non modifiables
 - Une méthode abstraite est une méthode dont on connaît juste la signature (pas de corps)

Définition d'une interface

J ICercle.java > ICercle

```
1  public interface ICercle
2  {
3      float pi = 3.14f;
4      public float getRayon ();
5      public void setRayon (float vrayon);
6      public float perimetre ();
7      public float surface ();
8  }
```


Implémentation d'une interface

CercleImpl.java > CercleImpl

```
1 public class CercleImpl implements ICercle
2 {
3     private float rayon;
4     //implémentation des methodes de l'interface ICercle
5     public float getRayon () {return this.rayon;}
6     public void setRayon (float vrayon) {this.rayon=vrayon;}
7     public float perimetre () {
8         float p = (float) (2*ICercle.pi*rayon);
9         return p;}
10    public float surface () {
11        float s= (float) (ICercle.pi*Math.pow (rayon, 2));
12        return s;}
13    //constructeurs
14    public CercleImpl (){}
15
16    Run | Debug
17    public static void main(String[] args) {
18        CercleImpl c = new CercleImpl();
19        c.setRayon(5);
20        System.out.println("Le périmètre du cercle c est : "+ c.perimetre ()+ " mètres");
21        System.out.println("La surface du cercle c est : "+c.surface() +" m2");
22    }
```



Les exceptions

Les exceptions

- En Java, les erreurs se produisent lors d'une exécution sous la forme d'**exceptions**
- Une exception :
 - est un objet, instance d'une classe d'exceptions (`java.lang.Exception`)
 - peut provoquer la sortie d'une méthode
 - correspond à un type d'erreur
 - contient des informations sur cette erreur

Déclaration des exceptions

- Une méthode déclare, par le mot-clé `throws`, dans sa signature les classes d'exception qu'elle peut envoyer
- **Exemple de la méthode `substring()` de la classe `String`**
 - `public class String {`
 - `...`
 - `public String substring(int beginIndex, int endIndex)`
 - `throws IndexOutOfBoundsException {`
 - `...`
 - `}`
 - `...`
 - `}`

Traitement des exceptions

- **Propagation**
- L'exception est renvoyée à la méthode ayant invoquée la méthode déclarant l'exception (mots-clés throws et throw)
- **Interception**
- L'exception est traitée dans la méthode appelant la méthode émettant l'exception (mots-clés try et catch)

Example: Propagation

```
public String initialesAuteur(Livre l)
■ throws IndexOutOfBoundsException {
    □ String titre, initiales;
    □ titre = l.getTitre() ;
    □ initiales = titre.substring(0,2) ;
    □ return initiales;
■ }
```

Exemple: Interception

```
public String initialesAuteur(Livre l) {
```

```
■ String titre, initiales;
```

```
■ try {
```

```
    □ titre = l.getTitre() ;
```

```
    □ initiales = titre.substring(0,2) ;
```

```
    □ return initiales;
```

```
    □ } catch (IndexOutOfBoundsException ex) {
```

```
        □ return new String(" Dépassement d'indice") ;
```

```
■ }
```

```
■ }
```

Définition d'une classe d'exception

```
public class ZeroDivision extends
ArithmeticException {


    public ZeroDivision() {
        // TODO Auto-generated constructor stub
    }

    public ZeroDivision(String arg0) {
        super(arg0);
        // TODO Auto-generated constructor stub
    }

}
```


Tester une classe d'exception

```
public class TestException {  
    private static int division (int a, int b) throws ZeroDivision  
    {  
        if (b==0)  
            throw new ZeroDivision("Attention divsion par zero");  
        return a/b;  
    }  
    /**  
     * @param args  
     */  
    public static void main(String[] args) {  
        // TODO Auto-generated method stub  
        int x=0;  
        try {  
            x=division(10,0);  
        } catch (ZeroDivision e) {  
            System.out.println(e.getMessage());  
        }  
        System.out.print(x);  
    }  
}
```



Connexion JDBC pour les bases de données

Fichier de configuration des paramètres

```
1  #Paramétrage de ce fichier pour faciliter la migration vers
2  # une nouvelle BD
3
4  #Paramètres de MYSQL
5  # jdbc.driver.class =com.mysql.jdbc.Driver
6  # jdbc.url=jdbc:mysql://localhost:3306/pedagogie
7  # jdbc.login=root
8  # jdbc.password=
9
10 #Paramètres de PotsgreSQL
11 jdbc.driver.class =org.postgresql.Driver
12 jdbc.url=jdbc:postgresql://localhost:5432/NomBD
13 jdbc.login=postgres
14 jdbc.password=passer
```

Connexion JDBC

1 ConnexionDB.java > 2 ConnexionDB > 3 ConnectDB()

```
1  import java.io.FileInputStream;
2  import java.sql.*;
3  import java.util.Properties;
4  public class ConnexionDB
5  {
6      public static Connection ConnectDB ()
7      {
8          try{ Properties p = new Properties ();
9              try (FileInputStream file = new FileInputStream("Config.properties")){
10                  p.load(file);
11              } //fichier Auto-closable
12              String urlPilote =p.getProperty("jdbc.driver.class");
13              Class.forName(urlPilote);
14              System.out.println ("Le pilote a été bien chargé");
15              String urlBD =p.getProperty("jdbc.url");
16              String user  = p.getProperty("jdbc.login");
17              String password= p.getProperty("jdbc.password");
18              Connection con =DriverManager.getConnection(urlBD, user, password);
19              System.out.println ("Connexion bien établie");
20              return con;
21          } catch (Exception e){
22              e.printStackTrace();
23              return null;
24          }
25      }
```

Test Connexion JDBC

```
26 public static void main(String[] args) throws Exception {
27     Connection con =ConnectDB();
28     Statement s= con.createStatement();
29     //La table banque a trois attributs (ID, Code, Nom)
30     ResultSet res= s.executeQuery("select * from banque");
31     System.out.println("ID "
32         + "\t"+"Code"
33         + "\t"+"Nom ");
34     while (res.next())
35     {
36         System.out.println(res.getInt(1)
37             + "\t"+res.getString(2)
38             + "\t"+res.getString(3));
39     }
40
41 }
42 }
```