Chapitre 2 :

Classes et Objets

La syntaxe d'une classe

```
[public] class  NomClasse
    {
          [private/public] typeAttribut1  nomAttribut1 [= valeur1];
          …
          [private/public] typeAttributN  nomAttributN [= valeurN];

          [private/public] typeRetour  nomMéthode1 ([type param11,…,
                                                    type param1M] ){…}
          …
          [private/public] typeRetour  nomMéthodeN ([type paramN1,…,
                                                    type paramNM] ){…}
    }
```

Exemple 1

```
public class Rectangle {
 private int longueur ;
 private int largeur ;

 public int surface ( ) {
     return (longueur*largeur) ;
 }
 public void initialise (int x, int y) {
     longueur =x ; largeur = y ;}
```

```
public static void main (String [] args) {
            Rectangle rec=new Rectangle();
            rec.initialize(3,4);
             int s = rec.surface();
            System.out.print(s);
    }

}//fin classe Rectangle
```

Exemple

```
public class Etudiant
{
    private int matricule;
    private String nom;
    private String prenom;
    private  String filiere;
    private int niveau;
    private int groupe;
    public void changerFiliere(String f)
          {filiere = f;}
    public void changerNiveau (int n)
          {niveau = n;}
    public void changerGroupe (int g)
          {groupe = g;}
}   //Fin classe Etudiant
```

Attributs

Méthodes

Exemples d'objets

Exemple : l'objet Etudiant1

Etudiant1 = (matricule =10, nom= ''Ayedi'', prenom= ''Ahmed'', filiere= ''Mécanique'', niveau= 1, groupe= 3)

Exemple : l'objet Etudiant2

Etudiant2 = (matricule =11, nom= ''Affes'', prenom= ''Mariem'', filiere= ''Informatique'', niveau= 2, groupe= 1)

Exemple

Public static void main (String [] args) {

// Création de 2 objets E1 et E2

```
Etudiant E1=new Etudiant (10, ''Ayedi'', ''Ahmed'', ''Mécanique'', 1, 3);
Etudiant E2=new Etudiant (11, 'Affes'', ''Mariem'', ''Informatique'', 2, 1);
```

// Appel des méthodes

.....

}

La notion de Constructeur

- Le constructeur d'une classe est une méthode appelée une seule fois par objet et ce au moment de sa création. Le constructeur porte le même nom que la classe
- Le rôle du constructeur est d'initialiser les attributs de l'objet
- En l'absence d'un constructeur défini par le programmeur, JAVA offre un constructeur non paramétré qui initialise les attributs de l'objet crée par les nulls correspondant à leurs types (0 pour *int*, *null* pour les références aux objets, …)
- Le constructeur est toujours publique et ne possède pas de type de retour (même pas *void*).

Exemple

```
public class Etudiant
{  private int matricule;
   private String nom;
   private String prenom;
   private  String filiere;
   private int niveau;
   private int groupe;
  public Etudiant (int m, String n, String p, String fil, int niv, int gr) {
     matricule  = m ;
     nom  = n ;
     prenom = p;
     filiere = fil;
     niveau = niv;
     groupe = gr;
}

……….
}   //Fin classe Etudiant
```

Attributs

Constructeur

```
Public static void main (String [] args) {

// Création de 2 objets E1 et E2

    Etudiant E1=new Etudiant (10, ''Ayedi'', ''Ahmed'', ''Mécanique'', 1, 3);
    Etudiant E2=new Etudiant (11, 'Affes'', ''Mariem'', ''Informatique'', 2, 1);

// Appel des méthodes
    E1.changerGroupe (4);
    E2.changerGroupe (4);
}
```

La notion du Constructeur

Personne.java

```
public class Personne
{
 private String nom;
 private String prenom;
 private int age;
  Public Personne(String unNom,
            String unPrenom, int unAge)
  {
    nom=unNom;
    prenom=unPrenom;
    age = unAge;
  }
 public Personne()
   {
     nom=null; prenom=null;
     age = 0;
   }
}
```

Constructeur avec 3 paramètres

Constructeur sans paramètres

Remarque:
On peut définir plusieurs constructeurs qui se différencient uniquement par leurs paramètres (on parle de leurs signatures),

La notion du Constructeur

Personne.java

```
public class Personne
{
  private String nom;
  private String prenom;
  private int age;
public Personne(String unNom,
                String unPrenom, int unAge)
  {
    nom=unNom;
    prenom=unPrenom;
    age = unAge;
  }
 public Personne()
   {
     nom=null; prenom=null;
     age = 0;
   }
}
```

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

   Personne P1 = new Personne (''Tounsi'',
''Ahmed','21'');

  Personne P2 = new Personne();

  }
}
```

L'opérateur d'instanciation en Java est **new**. new va réserver l'espace mémoire nécessaire pour créer l'objet P1 de la classe Personne,

La notion du Constructeur

Personne.java

```
public class Personne
{
 private String nom;
 private String prenom;
 private int age;
  public Personne(String unNom,
                  String unPrenom,
                  int unAge)
  {
    nom=unNom;
    prenom=unPrenom;
    age = unAge;
  }
 }
```

Définition d'un Constructeur. Le constructeur par défaut (Personne())n'existe plus.

```
public class Application
{
  public static void main(String args[])
  {
    Personne P = new Personne()
} }
```

Va donner une erreur à la compilation

Les objets

- La déclaration des objets de type classe est :

nomClasse nomObjet;

- La déclaration ne crée pas un objet, mais uniquement une référence nulle du type mentionné.
- Pour créer un objet, on utilise l'opérateur *new* suivi de l'appel à un constructeur de la classe de l'objet :

new nomClasse(expr1, expr2, ...)

Les objets

Construction des objets

L'appel de new pour créer un nouvel objet déclenche, dans l'ordre :

- L'allocation mémoire nécessaire au stockage de ce nouvel objet et l'initialisation par défaut de ces attributs,
- L’initialisation explicite des attributs, s'il y a lieu,
- L'exécution d'un constructeur,
- Retourner une référence sur l'objet.

Les objets

Cercle.java

```
public class Cercle
{
  public double x, y; // Coordonnées du centre
  public double r; // rayon du cercle
  public Cercle(double r1) {
        r = r1;
   }
   public double surface() {
        return 3.14159 * r * r;
    }
}
```

MonProgramme.java

```
public class MonProgramme
{
  public static void main(String[] args)
   {
      Cercle c; // c est une référence sur un objet Cercle, pas un objet
      c = new Cercle(5); // c référence maintenant sur un objet alloué en mémoire
      c.x = c.y = 10;
      System.out.println("Aire de c :" + c.surface());
    }
}
```

Les objets

Détermination de la nature d'un objet

- L'opérateur *instanceof* permet de tester si l'objet référencé est une instance d'une classe donnée (ou d'une de ses sous-classes).

Exemple

Cercle c = new Cercle(2.0) ;

c instanceof Cercle retourne true

c instanceof Personne retourne false

Les objets

Destruction d'objets

- Java utilise **le ramasse-miettes** (ou Garbage Collector - GC en anglais) qui s'occupe de collecter les objets qui ne sont plus référencés. Il fonctionne en permanence dans un thread de faible priorité.
- Il est possible d'indiquer ce qu'il faut faire juste avant de détruire un objet en utilisant la méthode ***finalize()*** de l'objet (pour fermer une base de données, fermer un fichier, couper une connexion réseau,…)

Les objets

- ***Accès aux attributs***: les attributs d'objet sont désignés selon la syntaxe suivante: *nomObjet.nomAttribut*
- ***Invocation de méthodes:*** les méthodes sont invoquées selon la syntaxe suivante: *nomObjet.nomMethode()*
- ***Remarque:*** Au sein de la classe propriétaire, on peut désigner un attribut ou une méthode uniquement par son nom.
- On ne peut accéder à un attribut de l'extérieur de la classe là où il a été défini que si on est autorisé à le faire
- On ne peut invoquer une méthode de l'extérieur de la classe là où elle a été définie que si on est autorisé à le faire

Accès au attributs

- ***Accès aux attributs***: les attributs d'objet sont désignés selon la syntaxe suivante: *nomObjet.nomAttribut*

```
public class Personne
{  private String nom;
   private String prenom;
  public Personne (String n, String p) {
     nom  = n ;
     prenom = p;        }

.....................
Public static void main (String [] args) {
 // Création de l'objet P1
   Personne P1=new Personne (''Ayedi'',''Ahmed'');
// Accès à l'attribut
    System.out.println(''Le nom de la personne est ''+ P1.prenom);
    P1.prenom = ''Ali'' ;
}
}    //Fin classe Personne
```

Les accesseurs

- Ce sont les méthodes d'accès aux attributs d'une classes. On distingue :
  - Les accesseurs en lecture (getters) de la forme *getNomAttribut( ). Ils* fournissent des informations relatives à l'état d'un objet, c. à d. des valeurs de certains attributs sans les modifier
  - les accesseurs en écriture (setters) de la forme *setNomAttribut( ).* Ils modifient l'état d'un objet, c. à d. les valeurs de certains attributs

Accès au attributs

```
public class Personne
{  private String nom;
   private String prenom;
  public Personne (String n, String p)
{
    nom  = n ;
    prenom = p;
}

public String getPrenom( )
 { return  prenom ; }

public void  setPrenom (String s1)
{  prenom   = s1; }

}
```

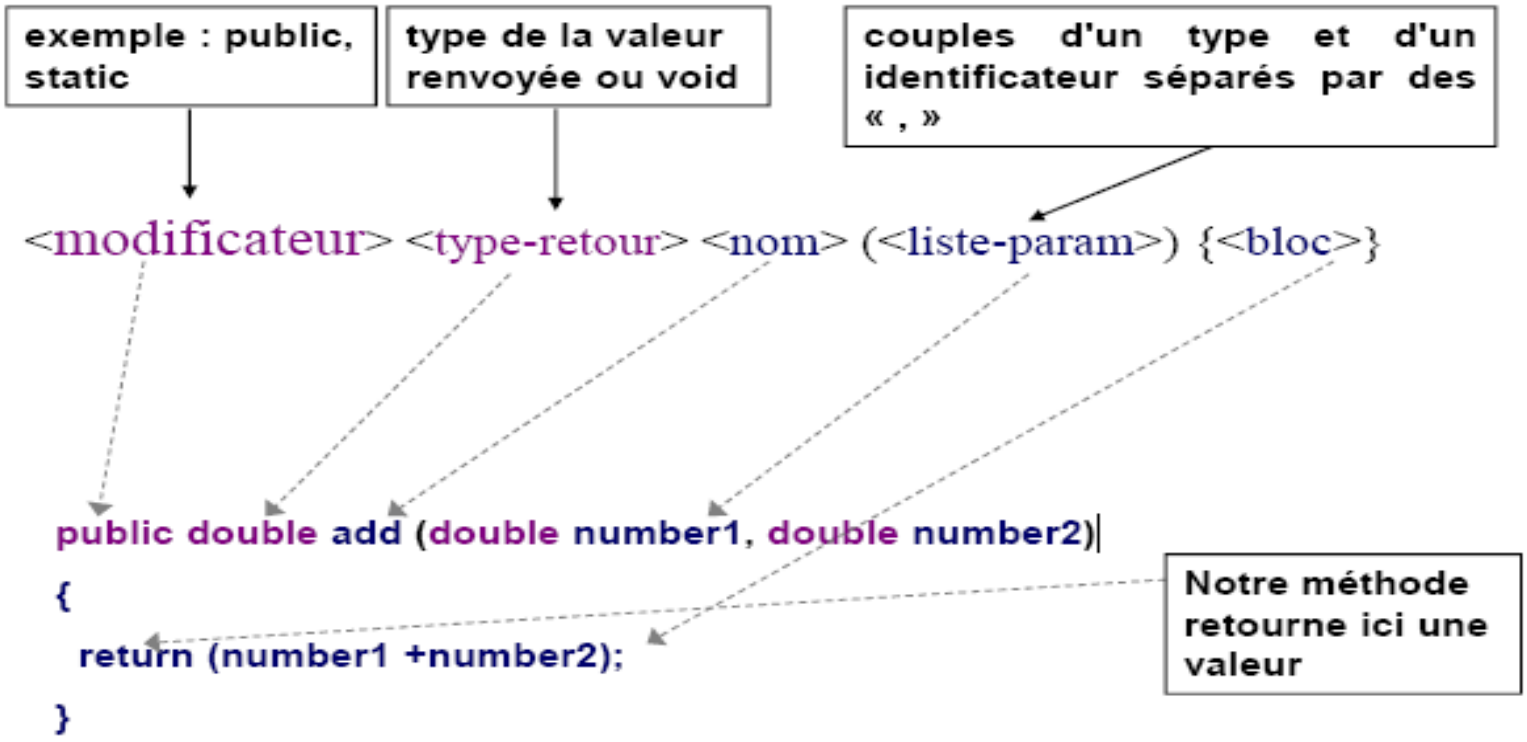
```
public class TestPersonne
{
 Public static void main (String [] args) {
 // Création de l'objet P1
   Personne P1=new Personne (''Ayedi'',
''Ahmed'');
// Accès à l'attribut
    System.out.println(''Le nom de la
personne est ''+ P1.getPrenom());
  P1.setPrenom('Ali'')
}
}
```

Appel des méthodes

Appel des méthodes

- Pour demander à un objet d'effectuer un traitement, il faut lui envoyer un message.
- Un message est composé de 3 parties :
  - Une référence permettant de désigner l'objet à qui le message est envoyé
  - Le nom de la méthode à exécuter
  - Les éventuels paramètres de la méthode

Appel des méthodes



Exemple d'invocation de cette méthode: double a = *nomObjet.add(2,5)*

Appel des méthodes

Mode de passage de paramètres

- Java n'implémente qu'un seul mode de passage des paramètres à une méthode : **le passage par valeur**.
- Pour les types simples :
  - Leurs valeurs sont recopiées
  - Leur modification dans la méthode n’entraîne pas celle de l’original
- Pour les objets :
  - Leurs références sont recopiées et non pas les attributs
  - Leur modification dans la méthode entraîne celle de l'original

Appel des méthodes

```
public class Exemple {
    public  void Doubler(int valeur)
    {
       valeur=valeur * 2;
    }

    public static void main(String[] args) {
            int age = 30;
            Exemple e=new Exemple();
            e.Doubler(age);
            System.out.println(age);
    }

}
```

La valeur retourné: 30

```
public class Voiture {
    String Couleur;

    public void Repeindre(Voiture v)
    {
        v.Couleur = "Bleue";
    }

    public static void main(String[] args) {
         Voiture v = new Voiture();
         v.Repeindre(v);
         System.out.println(v.Couleur);
    }

}
```

La valeur retourné: Bleue

Les tableaux

Définition

- Un tableau est composé d'un nombre déterminé de variable d'un même type (primitif ou objet). Les cellules non initialisées contiennent 0 (pour des types primitifs) ou null (pour des objets).
- **Déclaration d'un tableau :** Type [] nomTableau;
- **Création du tableau :** nomTableau = new Type[taille];
- **Initialisation du tableau :** nomTableau[numeroCellule] = UneValeur;

Remarque

- La taille du tableau est définie au moment de sa création, et ne peut plus être changée par la suite. Si on manque de la place dans un tableau, il faut obligatoirement créer un nouveau plus grand.

Les tableaux

- **Déclaration et instanciation d'un tableau :**

```
int monTableau[ ] =new  int[10];
ou
int [ ] monTableau=new  int[10];
```

- L'indexation démarre à partir de 0, ce qui veut dire que pour un tableau de N éléments, la numérotation va de 0 à N-1.
- **Remplissage d'un tableau :**

```
monTableau[5] =23 ;  //La 6ème case contient la valeur 23
```

- L'initialisation d'un tableau peut se faire par une liste statique d'initialisation comme suit:

```
int monTableau []= {5, 8 , 6 , 7 , 0} ; //Ce tableau contient 5 élèments
```

Les tableaux

- **Pour lire ou écrire les valeurs d'un tableau :**

```
int monTableau[ ]= {5, 8 , 6 , 7 , 0, 5 , 1} ;
int nb;
monTableau [3] = 25; // résultat: 5  8  6  25   0   5   1
nb = monTableau [4] ; //  nb = 0
```

- L'attribut *length* d'un tableau donne sa longueur (le nombre d'éléments). Donc, pour un tableau nommé monTableau *l'indice du dernier élément* est *monTableau.length -1*
- **Pour parcourir les éléments d'un tableau :**

```
for ( int i=0 ; i < monTableau.length ; i++)
{
    int element= monTableau [i] ;
    // traitement
 }
```

Les tableaux à plusieurs dimensions

- Les tableaux à plusieurs dimensions sont simplement des tableaux de tableaux.
- **Exemple:** Pour allouer une matrice de 5 lignes et 6 colonnes :

```
int matrice [ ][ ]  = new int[5] [ ];
for ( int i=0 ; i < matrice.length ; i++)
{
    matrice[i] = new int[6] ;
 }
```

- Java permet de résumer l'opération précédente en :

```
int matrice [][]  = new int[5][6] ;
```

- La première version montre qu'il est possible de créer un tableau de tableaux n'ayant pas forcément tous la même dimension

Les tableaux à plusieurs dimensions

- On peut également remplir le tableau à la déclaration et laisser le compilateur déterminer les dimensions des tableaux, en imbriquant les accolades

```
int[][] matrice =
    {
        { 0, 1, 4, 3 } , // tableau [0] de int
        { 5, 7, 9, 11, 13, 15, 17 } // tableau [1] de int
    };
```

- Pour déterminer la longueur des tableaux, on utilise également l'attribut ***length***

```
matrice.length      //  2
matrice[0].length   //  4
matrice[1].length   //  7
```

- Pour parcourir tous les éléments d'un tableau :

```
int i, j;
for(i=0; i<matrice.length; i++) {
    for(j=0; j<matrice[i].length; j++) {
        //Action sur matrice[i][j]
    }
}
```