Cours Java

2002-2003

Le langage java possède des types prédéfinis qui permettent de manipuler

- les nombres entiers
- les nombres réels
- les booléens
- les caractères
- les chaînes de caractères

Les types entiers

nom	taille[(1)]	intervalle
byte	1	-2^7 à $2^7 - 1$
short	2	-2^{15} à $2^{15} - 1$
int	4	-2^{31} à $2^{31} - 1$
long	8	-2^{63} à $2^{63} - 1$

(1) la taille est donnée en octets

Les types réels

nom	taille(1)
float	4
double	8

(1) la taille est donnée en octets

Le type booléen

nom	taille(1)
boolean	1

Une variable de type **boolean** ne peut prendre que deux valeurs **true** et **false**

(1) la taille est donnée en octets

Le type caractère

nom	taille$^{(1)}$
char	2

Les caractères sont codés sur 16 bits (non signés) en **unicode**

Les littéraux caractères

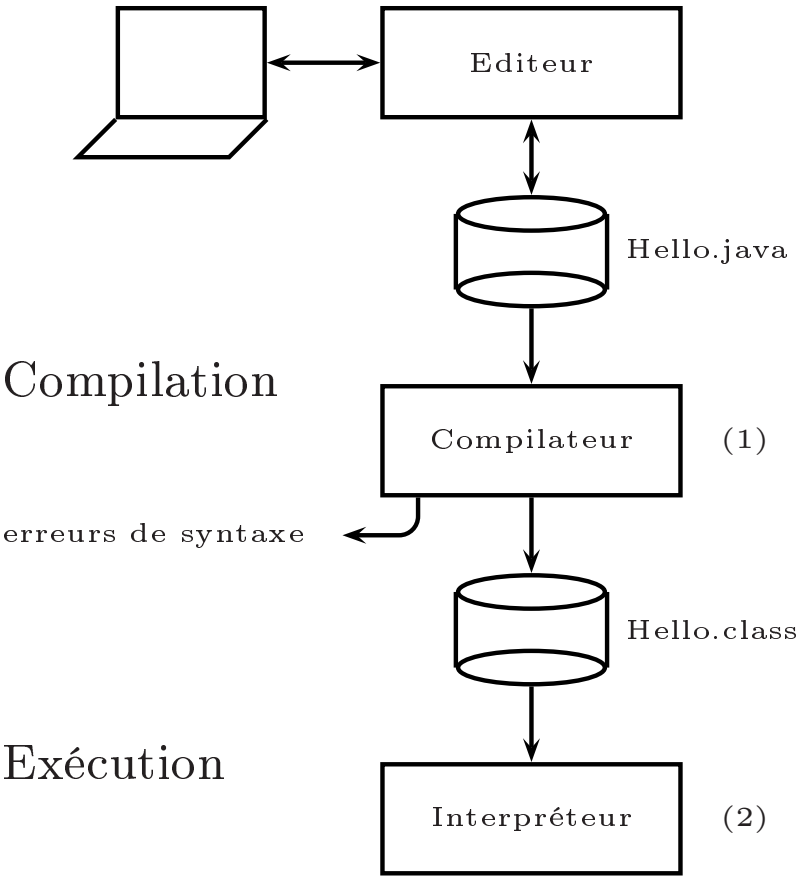
Le type chaîne de caractères

nom	taille(1)
String	

Les littéraux chaîne caractères

(1) La taille dépend du nombre de caractères

Edition



(1) `javac Hello.java`

(2) `java Hello`

Classes
et
variables d'instance

Classe

Une classe est un modèle qui permet de créer des objets

Objet
ou
instance d'une classe

Les objets crées suivant le modèle défini par une classe sont appelés instance de cette classe

Exemples de classes

- la classe **Personne**
- la classe **Cylindre**
- la classe **Point2D**
- la classe **Trinome**

Variables et méthodes d'instance

Une classe comporte des données appelées **variables d'instance**

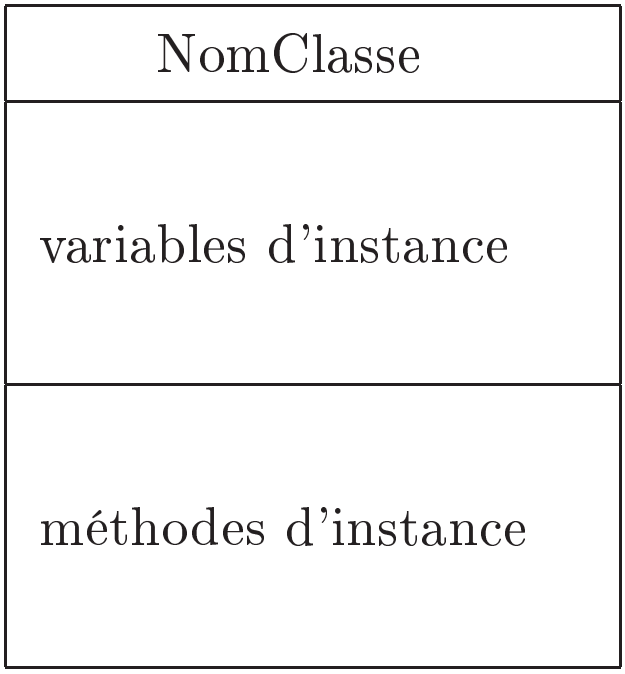
Une classe comporte des méthodes appelées **méthodes d'instance**

Les variables d'instance d'un objet définissent son **état**

Les méthodes d'instance d'un objet définissent son **comportement**

Diagramme de classe

Une classe est représentée par un diagramme de la forme :

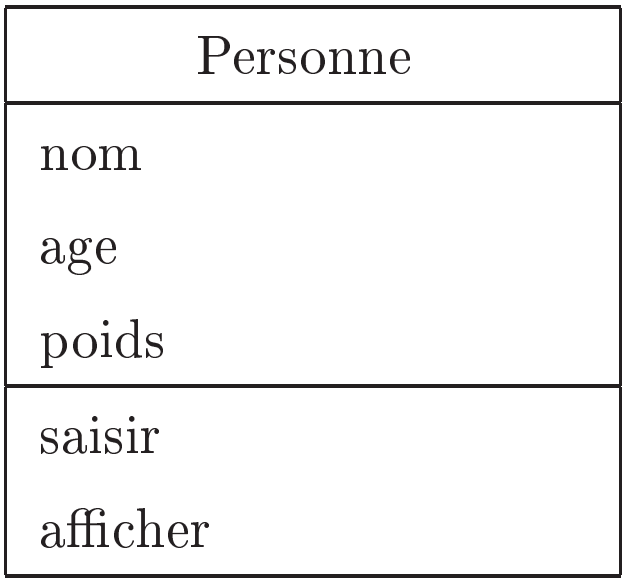


Le tiroir supérieur indique le nom de la classe

Le tiroir intermédiaire donne les variables d'instance

Le tiroir inférieur donne les méthodes d'instance

Exemple : la classe Personne



Exemple : la classe Cylindre

Cylindre
rayon hauteur
saisir afficher retourner le volume retourner la masse

Exemple : la classe Point2D

Point2D
x y
saisir afficher déplacer en (x1,y1) distance à une autre point p est égal à une autre point p symetrique par rapport à l'origine O

Déclaration d'une classe

Une classe doit être déclarée dans un fichier qui porte le même nom que la classe et a pour suffixe **.java**

```
// NomClasse.java

public class NomClasse {

    // variables d'instance

    // méthodes d'instance

}
```

Déclaration des variables d'instance

Pour déclarer une variable d'instance on indique :

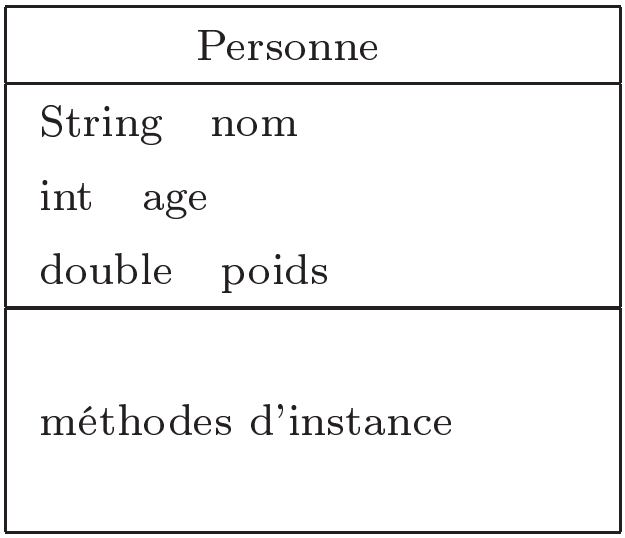
- son type
- son nom

NomClasse	
T1	varInst1
T2	varInst2,varInst3
méthodes d'instance	

où la variable d'instance **varInst1** est type **T1** et les variables d'instance **varInst2** et **varInst3** sont de type **T2**

```
// NomClasse.java

public class NomClasse {

    T1 varInst1 ;

    T2 varInst2 , varInst3 ;

    // méthodes d'instance

}
```

Exemple : la classe Personne



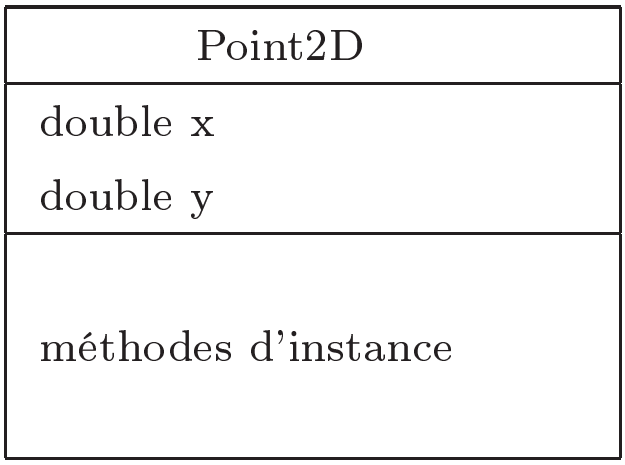

```
// Personne.java

public class Personne {
    String nom ;
    int age ;
    double poids ;

    // méthodes d'instance
}
```

Exemple : la classe Cylindre

Cylindre
double rayon double hauteur
méthodes d'instance

```
// Cylindre.java

public class Personne {
    double rayon ;
    double hauteur ;

    // méthodes d'instance
}
```

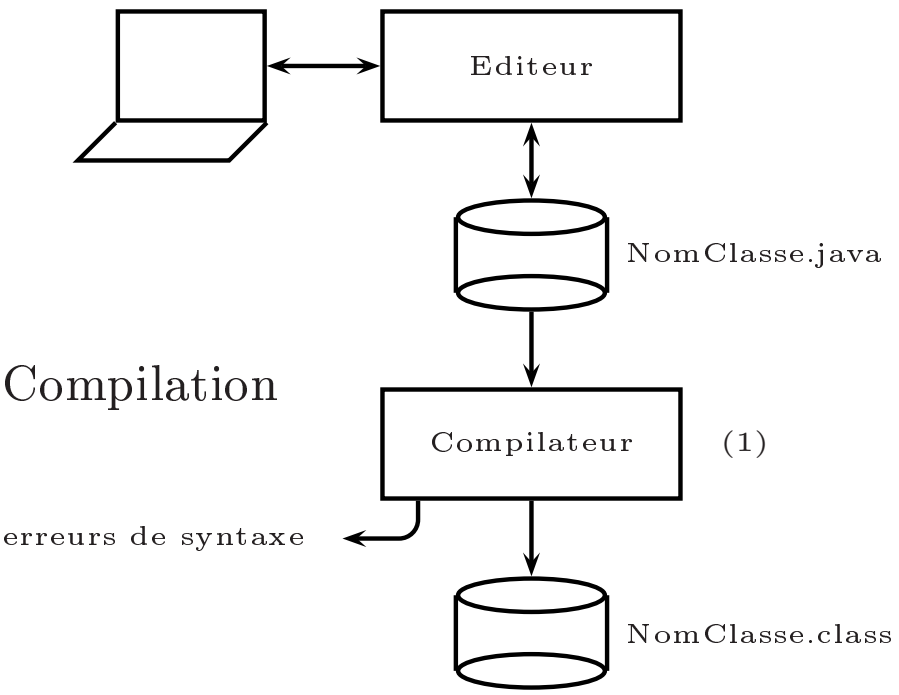
Exemple : la classe Point2D




```
// Point2D.java

public class Point2D {
    double x ;
    double y ;

    // méthodes d'instance
}
```

Compilation du fichier NomClasse.java

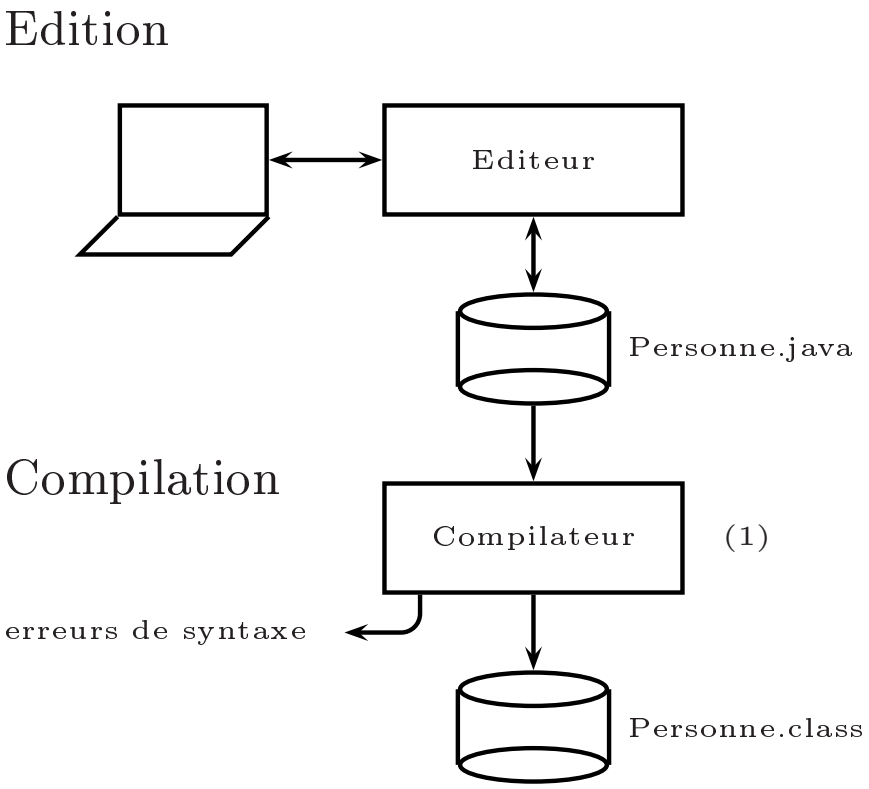
Edition



(1) `javac NomClasse.java`

Le fichier **NomClasse.class** ne pas pas être exécuté car il ne comporte pas de méthode **main**

Compilation du fichier Personne.java



(1) `javac Personne.java`

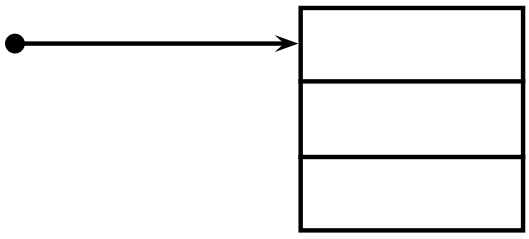
Le fichier **Personne.class** ne pas pas être exécuté car il ne comporte pas de méthode **main**

Création d'une instance
Appel du constructeur par défaut

Pour créer une instance d'une classe **NomClasse** on appelle son constructeur par défaut à l'aide de l'opérateur **new**

```
new NomClasse ( ) ;
```

cet appel alloue dans le tas une zone mémoire destinée aux variables d'instance **varInst1, varInst2** et **varInst3**

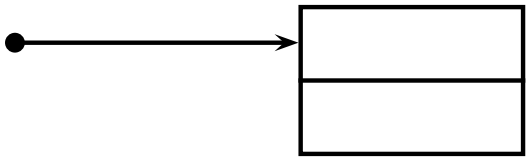


Exemple

Pour créer une instance de la classe **Cylindre** on appelle son constructeur par défaut à l'aide de l'opérateur **new**

```
new Cylindre ( ) ;
```

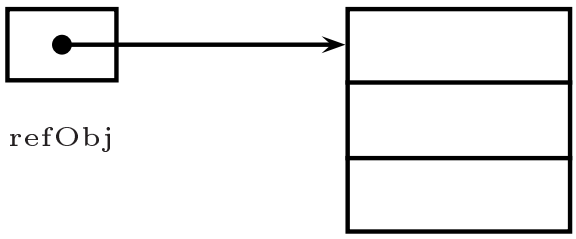
cet appel alloue dans le tas une zone mémoire destinée aux variables d'instance **rayon** et **hauteur**



Référencement d'une instance

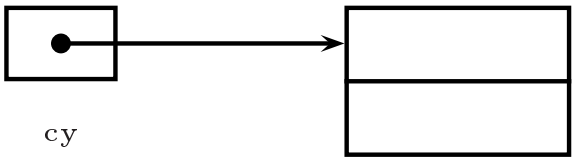
La référence d'une instance créée par l'appel du constructeur doit être mémorisée dans une variable **refObj** de type **NomClasse**

```
NomClasse refObj = new NomClasse ( )  ;
```



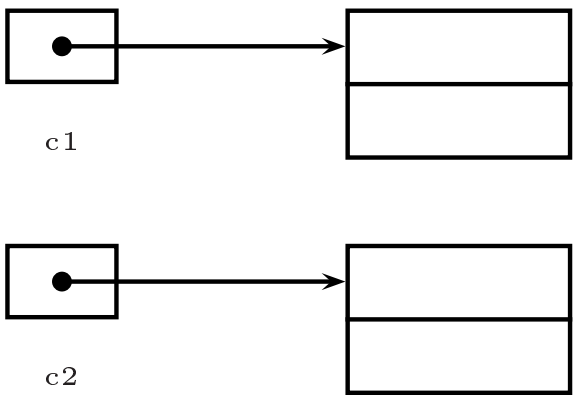
Exemple

```
Cylindre  cy = new Cylindre (  ) ;
```



Il est bien sûr possible de créer plusieurs instances d'une même classe

```
Cylindre  c1 = new Cylindre (  ) ;

Cylindre  c2 = new Cylindre (  ) ;
```



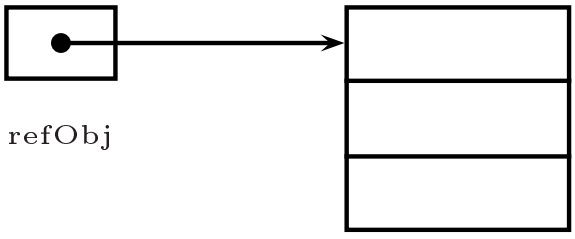
On ainsi construit deux instances de la classe **Cylindre** référencées respectivement par **c1** et **c2**

Ne pas confondre déclaration d'une référence

```
NomClasse refObj ;
```

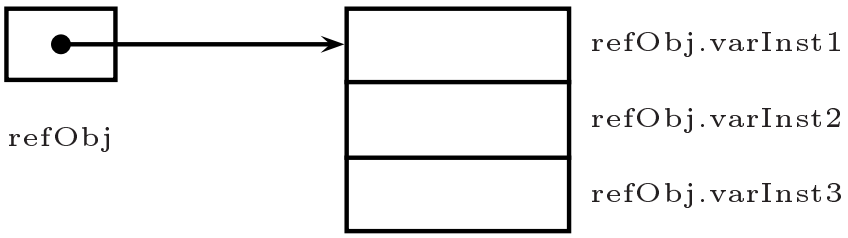
et construction d'une instance
puis affectation d'une référence

```
refObj = new nomClasse (  ) ;
```

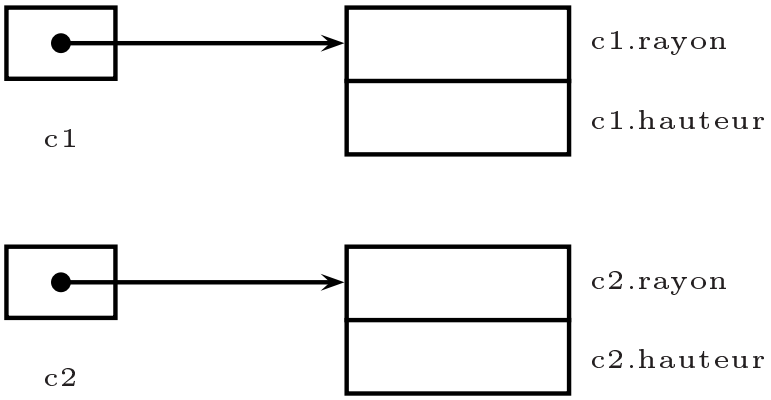


Accès aux variables d'instance
L'opérateur .

L'opérateur . permet d'accèder aux variables d'instance à partir de la référence



Exemple

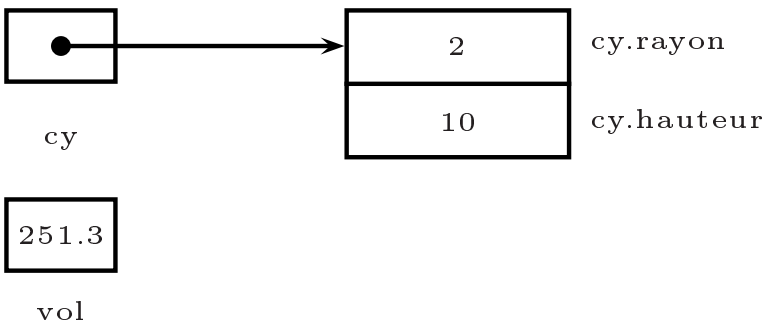


Programme de test

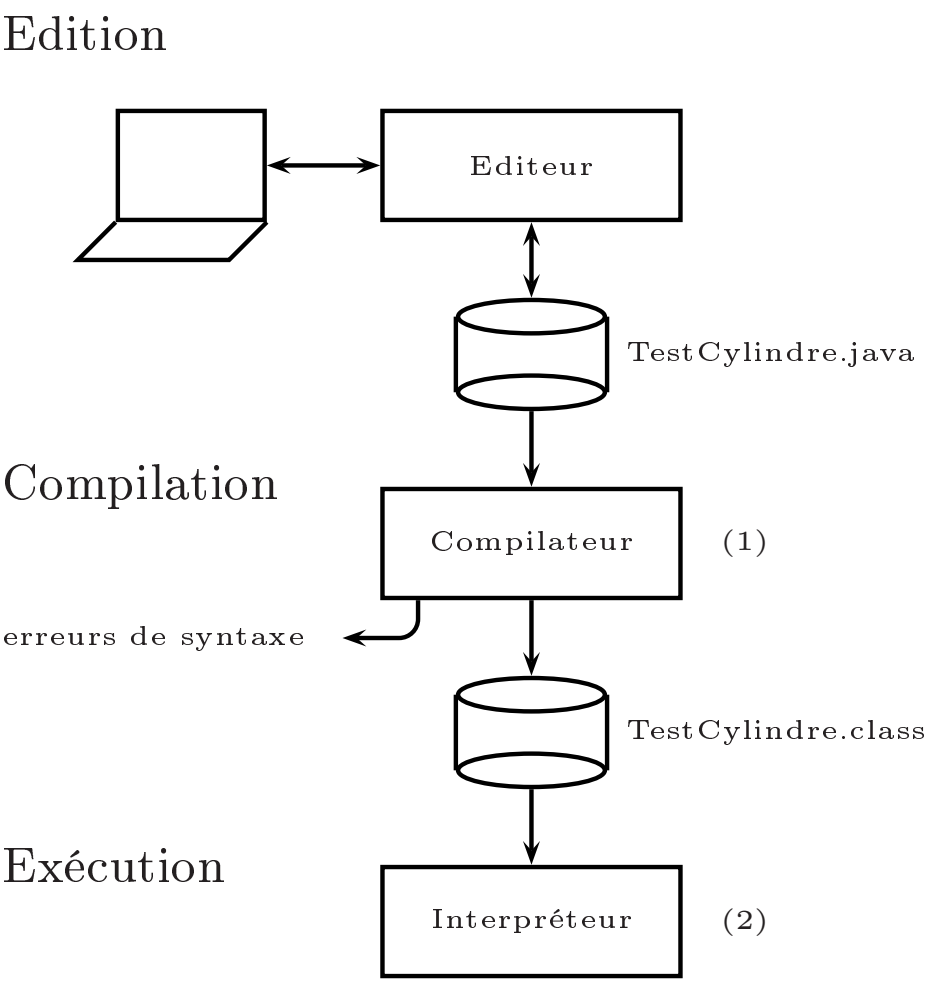
Les objets instances d'une classe sont crées et référencés dans un programme de test

```
public class TestCylindre {

    public static void main ( String [ ] args ) {

        Cylindre cy = new Cylindre ( ) ;

        cy . rayon = 2 ;
        cy . hauteur = 10 ;

        // calcul du volume

        double vol = 2 * Math . PI * cy . rayon * cy . rayon * cy . hauteur ;

        System.out.println ( "volume = " + vol ) ;

    }
}
```

Ce programme de test a permis de créer



Ce programme de test peut être compilé puis exécuté



(1) `javac TestCylindre.java`

(2) `java TestCylindre`

Classes et méthodes d'instance

Déclaration d'une méthode d'instance
en-tête + corps

Une méthode d'instance comporte :

- une en-tête
- et un corps.

En-tête d'une méthode

L'en-tête ou prototype donne :

- le type de la valeur retournée par la méthode
- le nom de la méthode
- les types et noms des paramètres de la méthode

T meth (T1 pf1 , ... , Tn pfn)

Si une méthode ne retourne pas de valeur **T** est égal à **void**

Les parenthèses sont obligatoires même si la méthode n'admet pas d'argument.

Corps d'une méthode

Le corps indique le traitement effectué par la méthode.

```
T   meth ( T1 pf1 , ... , Tn pfn ) {

      // corps de la méthode

}
```

Exemple : la classe Cylindre

Cylindre
double rayon double hauteur
void saisir( ) void afficher ( ) double volume ( ) double masse (double rho)

Le fichier Cylindre.java

```
// Cylinde.java
// ------------

public class Cylindre {

    double rayon ;
    double hauteur ;

    void saisir ( ) {
        rayon = EntreeClavier.readDouble ( "rayon=" ) ;
        hauteur = EntreeClavier . readDouble ( "hauteur=" ) ;
    }

    void afficher ( ) {
        System . out . println ( "rayon=" + rayon + " hauteur=" + hauteur ) ;
    }

    double volume ( ) {
        double v = 4 * Math.PI * rayon * rayon  * hauteur ;
        return v ;
    }

    double masse ( double rho ) {
        double m = masse ( rho ) ;
        return m ;
    }

}
```

Appel d'une méthode

Pour appliquer la méthode **meth** à la référence d'objet **refObj** on donne

- la référence de l'objet sur lequel est appliqué la méthode
- l'opérateur ●
- la liste des paramètres passés à la méthode

refObj ● meth (pe1 , . . . , pen)

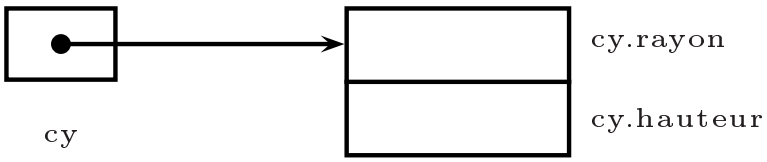
Les parenthèses sont obligatoires, même si la méthode ne comporte pas de paramètre.

Les paramètres sont séparés par des virgules.

La valeur éventuelle retournée par une méthode doit être utilisée :

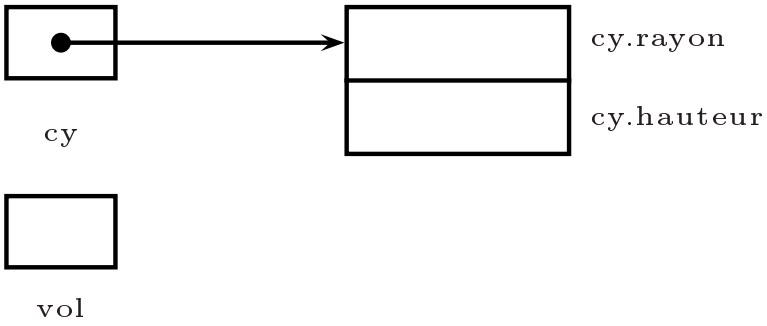
- soit en l'affectant à une autre variable
- soit en l'affichant à l'écran
- soit en la passant comme paramètre à une autre méthode

La méthode ne retourne pas de valeur




```
cy . saisir ( ) ;
```

La méthode retourne une valeur




```
vol = cy . volume ( ) ;
```

ou

```
System.out.println(cy.volume( ));
```

Le programme de TestCylindre.java

```
// TestCylindre.java
// -----------------

public class TestCylindre {

    public static void main ( String [ ] args ) {

        Cylindre cy = new Cylindre ( ) ;

        cy . rayon = 2 ;
        cy . hauteur = 10 :

        double v1 = cy . volume ( ) ;

        System . out . println ( "vol=" + v1 ) ;

        System.out.println ( "masse = " + cy.masse ( 1.5e3 ) ) ;

       // on augmente le rayon et la hauteur de 10%

        cy . rayon = 1.1 * cy . rayon  ;
        cy . hauteur = 1.1 * cy . hauteur ;

        cy . afficher ( ) ;

        double v2 = cy . volume ( ) ;

        System . out . println ( "vol=" + v2 ) ;

        double d = v2 - v1 ;

        System . out . println ( "augmentation = " + d ) ;

    }

}
```

Exemple : la classe Personne

Personne
String nom int age double poids
void saisir ( ) void afficher ( ) boolean aMemeAge (Personne p)

Le fichier Personne.java

```
// Personne.java
// -------------

public class Personne {

    String nom ;
    int age ;
    double poids ;

    void saisir ( ) {
        nom=EntreeClavier.readString ( "nom=" ) ;
        age=EntreeClavier.readInt ( "age=" ) ;
        poids=EntreeClavier.readDouble ( "poids=" ) ;
    }

    void afficher ( ) {
        System.out.println("nom=" + nom + " age=" + age + " poids=" + poids ) ;
    }

    boolean aMemeAge ( Personne p ) {
        return age == p . age  ;
    }

}
```

Le programme de TestPersonne.java

```
// TestPersonne.java
// -----------------

public class TestPersonne {

    public static void main ( String [ ] args ) {

        Personne pers1 = new Personne ( ) ;
        pers1 . nom = "Dupont" ;
        pers1 . age = 10;
        pers1 . poids = 60 ;

        Personne pers2 = new Personne ( ) ;
        pers2 . nom = "Duval" ;
        pers2 . age = 11;
        pers2 . poids = 65 ;

        Personne pers3 = new Personne ( ) ;
        pers3 . nom = "Durand" ;
        pers3 . age = 10;
        pers3 . poids = 62 ;

        pers1 . afficher ( ) ;
        pers2 . afficher ( ) ;
        pers3 . afficher ( ) ;


        System.out.println ( pers1.aMemeAge (pers2 ) ) ;
        System.out.println ( pers1.aMemeAge (pers3 ) ) ;

    }

}
```

Exemple : la classe Point2D

Point2D
double x double y
void saisir ( ) void afficher ( ) void deplacer (double x1, double y1) double dist (Point2D p) boolean estEgal (Point2D p) Point2D symO ( )

Le fichier Point2D.java

```
// Point2D.java
// ------------

public class Point2D {

    double x , y ;

    void saisir ( ) {
        x=EntreeClavier.readDouble ( "x=" ) ;
        y=EntreeClavier.readDouble ( "y=" ) ;
    }

    void afficher ( ) {
        System.out.println("(" + x + "," + y + ")" ) ;
    }

    void deplacer ( double x1 , double y1 ) {
        x = x1 ; y = y1 ;
    }

    double dist ( Point2D p ) {
        double dx = p . x - x ;
        double dy = p . y - y ;
        return Math . sqrt ( dx * dx + dy * dy ) ;
    }

    boolean estEgal ( Point2D p ) {
        return x  == p.x  && y == p.y   ;
    }

    Point2D sym0 ( ) {
         Point2D sy = new Point2D ( ) ;
         sy.x = -x ; sy.y=-y;
         return sy ;
    }

}
```

Le programme de TestPoint2D.java

```
// TestPoint2D.java
// ----------------

public class TestPoint2D {

    public static void main ( String [ ] args ) {

        Point2D a  = new Point2D ( ) ;
        a . x = 1 ; a . y = 2 ;

        Point2D orig  = new Point2D ( ) ;
        orig . x = 1 ; orig . y = 2 ;

        System.out.println ( orig . dist ( a ) ) ;

        Point2D a1  = a . sym0 ( ) ;

        System.out.print ( "a1 = " ) ; a1 . afficher ( ) ;

        Point2D b  = a1 . sym0 ( ) ;

        System.out.print ( "b = " ) ; b . afficher ( ) ;

        System.out.println ( a . estEgal ( a1 ) ) ;
        System.out.println ( a . estEgal ( b  ) ) ;

    }

}
```

Principe d'encapsulation méthodes d'accès

Les variables d'instances d'une classe ne doivent être accessibles qu'aux méthodes de la classe.

Le contôle de leurs valeurs peut ainsi être centralisé au niveau de la classe.

Pour cela on les déclare avec le modificateur d'accès **privé**

Au contraire les méthodes d'une classe sont accessibles à l'extérieur de la classe, elles sont donc déclarées avec le modificateur d'accès **public**.

Comme les variables d'une classe sont privées il est nécessaire de prévoir des méthodes d'instance qui vont permettre

- d'obtenir la valeur d'une variable d'instance
- de modifier la valeur d'une variable d'instance

Ces méthodes sont appelées accesseurs

- en lecture
- en écriture

Le concepteur de la classe peut restreindre l'accès uniquement en lecture

Exemple : la classe Cylindre

Cylindre
− double rayon − double hauteur
+ void saisir ( ) + void afficher ( ) + double volume ( ) + double masse (double rho) + doublegetRayon ( ) + doublegetHauteur ( ) + voidsetRayon (double r) + voidsetHauteur (double h)

Le fichier Cylindre.java

```
// Cylinde.java
// ------------
public class Cylindre {

    private double rayon ;
    private double hauteur ;

        ...

    public double getRayon ( ) {
        retutn rayon ;
    }

    public double getHauteur ( ) {
        retutn hauteur ;
    }

    public void setRayon ( double r ) {
        rayon = r ;
    }

    public void setHauteur ( double h ) {
        hauteur = h ;
    }
}
```

Le fichier Cylindre.java

```
// Cylinde.java
// ------------
public class Cylindre {

    private double rayon ;
    private double hauteur ;

    public void saisir ( ) {
        rayon = EntreeClavier.readDouble ( "rayon=" ) ;
        hauteur = EntreeClavier . readDouble ( "hauteur=" ) ;
    }

    public void afficher ( ) {
        System . out . println ( "rayon=" + rayon + " hauteur=" + hauteur ) ;
    }

    public double volume ( ) {
        double v = 4 * Math.PI * rayon * rayon  * hauteur ;
        return v ;
    }

    public double masse ( double rho ) {
        double m = masse ( rho ) ;
        return m ;
    }

    public double getRayon ( ) {
        retutn rayon ;
    }

    public double getHauteur ( ) {
        retutn hauteur ;
    }

    public void setRayon ( double r ) {
        rayon = r ;
    }

    public void setHauteur ( double h ) {
        hauteur = h ;
    }
}
```

Le programme de TestCylindre.java

```
// TestCylindre.java
// -----------------

public class TestCylindre {

    public static void main ( String [ ] args ) {

        Cylindre cy = new Cylindre ( ) ;

        cy . setRayon ( 2 ) ;
        cy . setHauteur  ( 10 ) :

        double v1 = cy . volume ( ) ;

        System . out . println ( "vol=" + v1 ) ;

        System.out.println ( "masse = " + cy.masse ( 1.5e3 ) ) ;


       // on augmente le rayon et la hauteur de 10%

        cy . setRayon ( 1.1 * cy . getRayon ( ) )  ;
        cy . setHauteur ( 1.1 * cy . getHauteur ( ) ) ;

        cy . afficher ( ) ;

        double v2 = cy . volume ( ) ;

        System . out . println ( "vol=" + v2 ) ;

        double d = v2 - v1 ;

        System . out . println ( "augmentation = " + d ) ;

    }
}
```

Exemple : la classe Personne

Personne
− String nom − int age − double poids
+ void saisir ( ) + void afficher ( ) + boolean aMemeAge (Personne p) + String getNom ( ) + int getAge ( ) + double getPoids( ) + void setNom (String n) + void setAge (int a) + void setPoids (double p)

Le fichier Personne.java

```
// Personne.java
// -------------

public class Personne {

    private String nom ;
    private int age ;
    private double poids ;

    public void saisir ( ) {
        nom=EntreeClavier.readString ( "nom=" ) ;
        age=EntreeClavier.readInt ( "age=" ) ;
        poids=EntreeClavier.readDouble ( "poids=" ) ;
    }
    public void afficher ( ) {
        System.out.println("nom=" + nom + " age=" + age + " poids=" + poids ) ;
    }
    public boolean aMemeAge ( Personne p ) {
        return age == p . age  ;
    }
    public String getNom ( ) {
        return nom ;
    }
    public int getAge ( ) {
        return age ;
    }
    public double getPoids ( ) {
        return poids ;
    }
    public void setNom ( String n ) {
        nom = n ;
    }
    public void setAge ( int a ) {
        age = a
    }
    public void setPoids ( double p ) {
        poids = p ;
    }
}
```

Le programme de TestPersonne.java

```
// TestPersonne.java
// -----------------

public class TestPersonne {

    public static void main ( String [ ] args ) {

        Personne pers1 = new Personne ( ) ;
        pers1 . setNom ( "Dupont" ) ;
        pers1 . setAge (  10 );
        pers1 . setPoids ( 60 ) ;

        Personne pers2 = new Personne ( ) ;
        pers2 . setNom  ( "Duval" ) ;
        pers2 . setAge ( 11 ) ;
        pers2 . setPoids ( 65 ) ;

        Personne pers3 = new Personne ( ) ;
        pers3 . setNom  ( "Durand" ) ;
        pers3 . setAge ( 10 ) ;
        pers3 . setPoids ( 62 ) ;

        pers1 . afficher ( ) ;
        pers2 . afficher ( ) ;
        pers3 . afficher ( ) ;

        System.out.println ( pers1.aMemeAge (pers2 ) ) ;
        System.out.println ( pers1.aMemeAge (pers3 ) ) ;

    }

}
```

Exemple : la classe Point2D

Point2D
− double x − double y
+ void saisir ( ) + void afficher ( ) + void deplacer (double x1, double y1) + double dist (Point2D p) + boolean estEgal (Point2D p) + Point2D symO ( ) + double getX ( ) + double getX ( ) + void setX (double x0) + void setY (double y0)

Le fichier Point2D.java

```
// Point2D.java
// ------------
public class Point2D {
    private double x , y ;

    public void saisir ( ) {
        x=EntreeClavier.readDouble ( "x=" ) ;
        y=EntreeClavier.readDouble ( "y=" ) ;
    }
    public void afficher ( ) {
        System.out.println("(" + x + "," + y + ")" ) ;
    }
    public void deplacer ( double x1 , double y1 ) {
        x = x1 ; y = y1 ;
    }
    public double dist ( Point2D p ) {
        double dx = p . x - x ;
        double dy = p . y - y ;
        return Math . sqrt ( dx * dx + dy * dy ) ;
    }
    public boolean estEgal ( Point2D p ) {
        return x  == p.x  && y == p.y   ;
    }
    public Point2D sym0 ( ) {
         Point2D sy = new Point2D ( ) ;
         sy.x = -x ; sy.y=-y;
         return sy ;
    }
    public double getX (  ) {
        return x ;
    }
    public double getY  (  ) {
        return y ;
    }
    public void setX ( double x0  ) {
        x = x0 ;
    }
    public void  setY ( double y0  ) {
        y = y0 ;
    }
}
```

Le programme de TestPoint2D.java

```
// TestPoint2D.java
// ----------------

public class TestPoint2D {

    public static void main ( String [ ] args ) {

        Point2D a  = new Point2D ( ) ;
        a . setX ( 1 )  ; a . setY ( 2 ) ;

        Point2D orig  = new Point2D ( ) ;
        orig . setX ( 0 ) ; orig . setY ( 0 )  ;

        System.out.println ( orig . dist ( a ) ) ;

        Point2D a1  = a . sym0 ( ) ;

        System.out.print ( "a1 = " ) ; a1 . afficher ( ) ;

        Point2D b  = a1 . sym0 ( ) ;

        System.out.print ( "b = " ) ; b . afficher ( ) ;

        System.out.println ( a . estEgal ( a1 ) ) ;
        System.out.println ( a . estEgal ( b  ) ) ;

    }

}
```

Constructeurs

Intérêt d'un constructeur

```
// TestCylindreUn.java
// -------------------

public class TestCylindreUn {

    public static void main ( String [ ] args ) {

        Cylindre cy = new Cylindre ( ) ;

        double v = cy . volume ( ) ;

        System . out . println ( "vol=" + v ) ;

    }

}
```

Ce programme peut être compilé et exécuté mais on obtient la valeur 0 car le rayon et la hauteur du cylindre **cy** n'ont pas été donnés

La version correcte du programme est

```
// TestCylindreDeux.java
// ---------------------

public class TestCylindreDeux {

    public static void main ( String [ ] args ) {

        Cylindre cy = new Cylindre ( ) ;

        cy . setRayon ( 2 ) ;
        cy . setHauteur ( 10 ) ;

        double v = cy . volume ( ) ;

        System . out . println ( "vol=" + v ) ;

    }

}
```

Si on dispose d'un constructeur qui permet de donner une valeur au rayon et à la hauteur à partir de ses deux paramètres

le programme précédent peut s'écrire sous la forme

```
// TestCylindreTrois.java
// ----------------------

public class TestCylindreTrois {

    public static void main ( String [ ] args ) {

        Cylindre cy = new Cylindre ( 2 , 10 ) ;

        double v = cy . volume ( ) ;

        System . out . println ( "vol=" + v ) ;

    }

}
```

Dans ce cas on n'utilise pas le constructeur par défaut qui ne possède pas paramètre

On doit définir dans la classe Cylindre un constructeur avec deux paramètres

Déclaration d'un constructeur

Un constructeur est une méthode qui porte le même nom que la classe et qui n'indique pas le type de la valeur retournée

NomClasse
variables d'instance
+ NomClasse (T1 pf1 , T2 pf2 , . . . , Tn pfn) méthodes d'instance

Le fichier NomClasse.java

```
// NomClasse.java

public class NomClasse {

    // variables d'instance

    public NomClasse ( T1 pf1 , T2 pf2 , ... , Tn Pfn ) {


    }

    // méthodes d'instance

}
```

Création d'une instance

```
NomClasse refObj = new NomClasse ( pe1 , pe2 , ... , pen )  ;
```

ou

```
NomClasse refObj ;

refObj = new NomClasse ( pe1 , pe2 , ... , pen )  ;
```

Exemple la classe Cylindre

```
// Cylinde.java
// ------------

public class Cylindre {

    private double rayon ;
    private double hauteur ;


    public Cylindre ( double r , double h )  {
        rayon = r ;
        hauteur = h ;
    }


    ...


}
```

Le fichier Cylindre.java

```
// Cylinde.java
// ------------
public class Cylindre {

    private double rayon ;
    private double hauteur ;

    public Cylindre ( double r , double h ) {
        rayon = r ;
        hauteur = h ;
    }

    public void saisir ( ) {
        rayon = EntreeClavier.readDouble ( "rayon=" ) ;
        hauteur = EntreeClavier . readDouble ( "hauteur=" ) ;
    }

    public void afficher ( ) {
        System . out . println ( "rayon=" + rayon + " hauteur=" + hauteur ) ;
    }

    public double volume ( ) {
        double v = 4 * Math.PI * rayon * rayon  * hauteur ;
        return v ;
    }

    public double getRayon ( ) {
        return rayon ;
    }

    public double getHauteur ( ) {
        retutn hauteur ;
    }

    public void setRayon ( double r ) {
        rayon = r ;
    }

    public void setHauteur ( double h ) {
        hauteur = h ;
    }
}
```

Constructeur par défaut et constructeur déclaré

Dés qu'on a déclaré dans une classe un constructeur (avec paramètres), le constructeur par défaut (sans paramètre) n'existe plus

Il interdit d'écrire

```
NomClasse refObj = new NomClasse (  )  ;
```

ou

```
Cylindre cy = new Cylindre (  )  ;
```

Utilisation d'un constructeur pour retourner un objet

Point2D
− double x − double y
+ Point2D (double x0 , double y0)   + Point2D symO ( )

```
// Point2D.java
// ------------

public class Point2D {

    private double x , y ;

    public Point2D ( double x0 ,double y0 ) {
        x = x0 ;
        y = y0 ;
    }

    ...

    public Point2D sym0 ( ) {
         Point2D sy = new Point2D ( -x , -y ) ;
         return sy ;
    }



}
```

on peut aussi retourner directement la référence renvoyée par le constructeur

```
// Point2D.java
// ------------

public class Point2D {

    private double x , y ;

    public Point2D ( double x0 ,double y0 ) {
        x = x0 ;
        y = y0 ;
    }


    public Point2D sym0 ( ) {
         return new Point2D ( -x , -y ) ;
    }



}
```

La classe Complexe

Complexe
− double x − double y
+ Complexe (double x0 , double y0) + Complexe add (Complexe z) + void afficher ( )

```
// Complexe.java
// -------------

public class Complexe {

    private double x , y ;

    public Complexe ( double x0 ,double y0 ) {
        x = x0 ;
        y = y0 ;
    }


    public Complexe add ( Complexe z ) {
         Complexe zs = new Complexe ( x + z. x , y + z.y ) ;
         return zs
    }

    public void affciher ( ) {
        System.out.println ( "(" + x + "," + y + ")" ) ) ;
    }

}
```

```
// Complexe.java
// -------------

public class Complexe {

    private double x , y ;

    public Complexe ( double x0 ,double y0 ) {
        x = x0 ;
        y = y0 ;
    }


    public Complexe add ( Complexe z ) {
         return new Complexe ( x + z. x , y + z.y ) ;
    }

    public void affciher ( ) {
        System.out.println ( "(" + x + "," + y + ")" ) ) ;
    }

}
```

```
// TestComplexe.java
// -----------------

public class TestComplexe {

    public static void main ( String [  ] args ) {

        Complexe z1 = new Complexe ( 1 , 2 ) ;

        Complexe z2 = new Complexe ( 3 , 4 ) ;

        Complexe z3 = z1.add(z2) ;

        z3 . afficher ( ) ;

    }

}
```

Codage des structures de contrôles

Séquence

Alternative

Répétition

Séquence

a1

a2

a3

Alternative

si c **alors**	`if ( c ) {`
a1	`a1`
sinon	`} else {`
a2	`a2`
fsi	`}`

La condition **c** doit être une expression de type boolean

La partie sinon peut être vide

```
si  c  alors          if ( c ) {
  a                       a1
fsi                   }
```

La partie alors peut être vide

si c **alors**	`if ( c ) {`
	`;`
sinon	`} else {`
a	`a`
fsi	`}`

dans ce cas il est préférable de transformer l'alternative sous la forme suivante

si non c **alors**	`if ( ! c ) {`
a	`a`
fsi	`}`

Le sinonsi

si c1 **alors**	`if ( c1 ) {`
a1	`a1`
sinonsi c2 **alors**	`} else if ( c2 ) {`
a2	`a2`
sinonsi c3 **alors**	`} else if ( c3 ) {`
a3	`a3`
sinon	`} else {`
a4	`a4`
fsi	`}`

L' alternative suivante ne peut pas utiliser le **sinonsi**

```
si c1 alors
  a1
sinon
  si c2 alors
    a2
  sinon
    a3
  fsi
  a4
fsi
```

```
if ( c1 ) {
    a1
} else  {
    if ( c2 ) {
        a2
    } else {
        a3
    }
    a4
}
```

L' alternative suivante ne peut également pas utiliser le **sinonsi**

```
si c1 alors                    if ( c1 ) {
  a1                               a1
sinon                          } else  {
  a2                               a2
  si c2 alors                      if ( c2 )  {
    a3                                 a3
  sinon                            } else {
    a4                                 a4
  fsi                              }
fsi                            }
```

Répétitions dont le nombre est connu : la boucle pour

Pas positif

pour i **de** e1 **à** e2 **pas** p **faire**
  a
fpour

```
for ( int i = e1 ; i <= e2 ; i = i + p ) {
    a
}
```

Répétitions dont le nombre est connu : la boucle pour

Cas particulier **p** égal à 1

pour i **de** e1 **à** e2 **faire**
  a
fpour

```
for ( int i = e1 ; i <= e2 ; i++ ) {
    a
}
```

Répétitions dont le nombre est connu : la boucle pour

Pas négatif

pour i **de** e1 **à** e2 **pas** −p **faire**
  a
fpour

```
for ( int i = e1 ; i >= e2 ; i = i - p ) {
    a
}
```

Répétitions dont le nombre est connu : la boucle pour

Cas particulier **p** égal à -1

pour i **de** e1 **à** e2 **pas** -1 **faire**
  a
fpour

```
for ( int i = e1 ; i >= e2 ; i-- ) {
    a
}
```

Répétitions dont le nombre n'est pas connu :
la boucle tant que faire

```
tq c faire
  a
ftq
```

```
while ( c )  {
    a
}
```

Répétitions dont le nombre n'est pas connu : la boucle faire tant que

faire
  a
tq c

```
do {
    a
} while ( c ) ;
```

Répétitions dont le nombre n'est pas connu : la boucle itérer

itérer
  a1
  sortirsi c
  a2
fin itérer

```
wile ( true ) {
    a1
    if ( c ) break ;
    a2
}
```

Tableaux à une dimension ou vecteurs

Un tableau est un groupement d'éléments de même type

Un tableau permet de représenter plusieurs variables désignées sous le même non

Les éléments d'un même tableau seront distingués à partir de leur rang

Déclaration et construction

Un tableau doit être déclaré puis construit

Déclaration

Pour déclarer un tableau **a** dont les éléments sont de type **T**, on écrit :

```
T [ ] a ;
```

par exemple

```
int [ ] u ;
float [ ] v ;
char [ ] c ;
```

Construction

Pour construire le tableau **a** précédemment déclaré, on précise son nombre d'éléments **nbElts** :

```
a = new T [ nbElts ];
```

nbElts doit être une expression entière dont la valeur est connue lors de la construction.

par exemple

```
u = new int [ 8 ] ;
v = new float [ 3 ] ;
int n=5;
c = new char [ 2 * n + 1 ] ;
```

Déclaration et construction

On peut regrouper les deux étapes précédentes en une seule :

```
T [ ] a = new T [ nbElts] ;
```

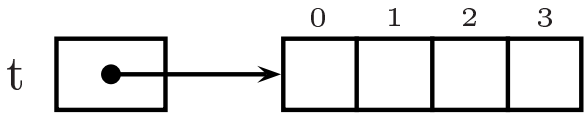
Exemples

```
int [ ] u  = new int [ 8 ] ;
float [ ] v = new float v [ 6 ];
int n = 5 ;
char [ ] c = new char [ 2 * n + 1 ] ;
```

Ne pas confondre déclaration et construction

```
T [ ] t ;
```


```
t = new T [ 4 ] ;
```



L'attribut length

a . length donnne le nombre d'éléments du tableau **a**

Exemple

```
u . length est égal à 8
```

Accès à un élément

L'élément d'indice **i** du tableau **a** est noté **a** [**i**]

Les éléments sont indicés à partir de zéro.

Le premier élément a pour indice 0

Le dernier élément a pour indice **a . length - 1**

Si la valeur de l'indice **i** n'est pas comprise entre 0 et **a.length-1** une exception de type **ArrayIndexOutOfBoundsException** est déclenchée et l'exécution du programme est stoppée.

Initialisation

Un tableau peut être initialisé

- lors de sa déclaration
- après sa déclaration

Initialisation lors de la déclaration

On peut initialiser un tableau lors de sa déclaration

```
T [ ] a = { val1 , val2 , ... , valn} ;
```

Les valeurs **val1**, **val2** ... et **valn** sont affectées aux éléments d'indice **0**,**1** et **n-1** du tableau

```
T [ ] a = new float [ n ] ;
a [ 0 ] = val1 ;
a [ 2 ] = val2 ;
   ...
a [ n - 1 ] = valn ;
```

Les valeurs **val1**, **val2** ... et **valn** doivent être du type **T**.

Exemples

```
int [ ] nbres  = { 10 , 20 , 40 , 50 , 60 , 70 , 80 , 100 } ;

float [ ] vals = { 0.1f , 2 , 10.5f , -3 } ;

char [ ] voyelles = { 'a' , 'e' , 'i' , 'o' , 'u' } ;

String nomMois [ ] = { "Janvier" , "Février" , "Mars" , "Avril" , "Mai" ,
                       "Juin" , "Juillet" , "Août" , "Septembre" , "Octobre"
                       "Décembre" } ;
```

Initialisation après de la déclaration

Si le tableau **a** a déjà été déclaré, on doit écrire :

```
a = new T [ ] { val1 , val2 , ... , valn } ;
```

Exemples

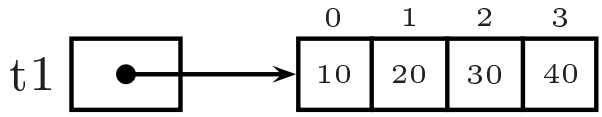
```
int [ ] nbres ;
nbres = new int [ ] { 10 , 20 , 40 , 50 , 60 , 70 , 80 , 100 } ;
float [ ] vals ;
vals = new float [ ] { 0.1f , 2 , 10.5f , -3 } ;
char [ ] voyelles ;
voyelles = new char [ ] { 'a' , 'e' , 'i' , 'o' , 'u' } ;
String nomMois [ ] ;
nomMois [ ] = { "Janvier" , "Février" , "Mars" , "Avril" , "Mai" ,
                "Juin" , "Juillet" , "Août" , "Septembre" , "Octobre" ,
                "Décembre" } ;
```

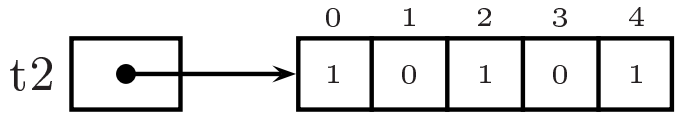
Cette possibilité sera utilisé pour passer en paramètre un tableau initialisé.

Affectation

Soient les tableaux **t1** et **t2**

```
int [ ] t1 = { 10 , 20 , 30 , 40 };

int [ ] t2 = { 1 , 0 , 1 , 0 , 1 } ;
```

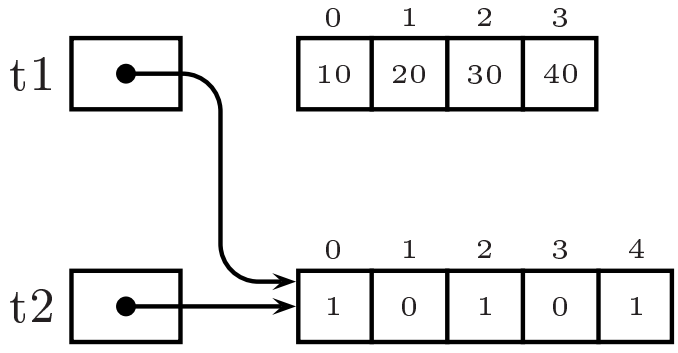




l'affectation

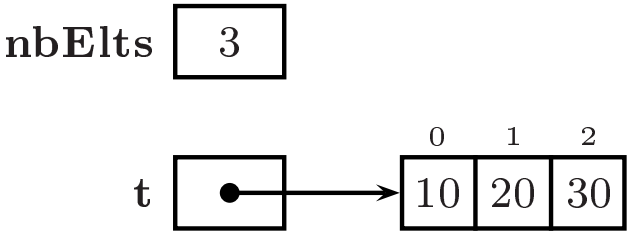
```
t1 = t2 ;
```

copie la référence du tableau **t2** dans **t1**

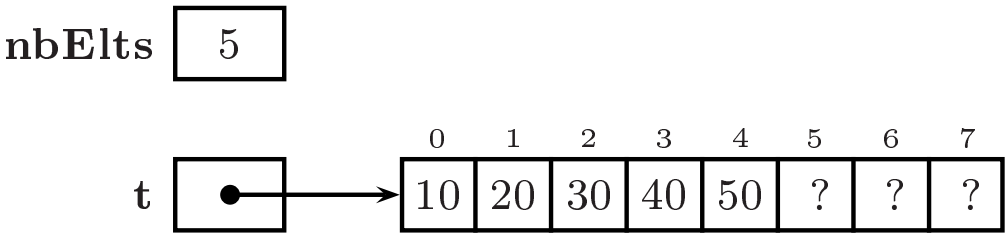


Agrandir un tableau

On a le tableau **t** et son nombre d'éléments **nbElts**

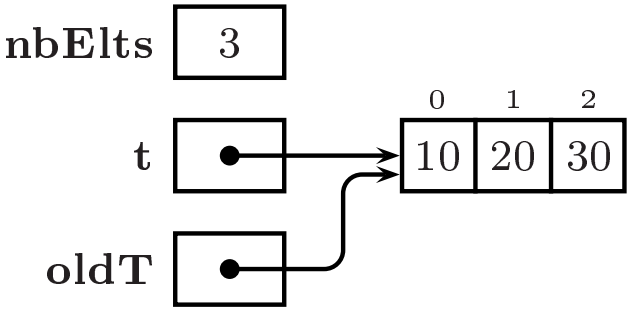


On veut obtenir



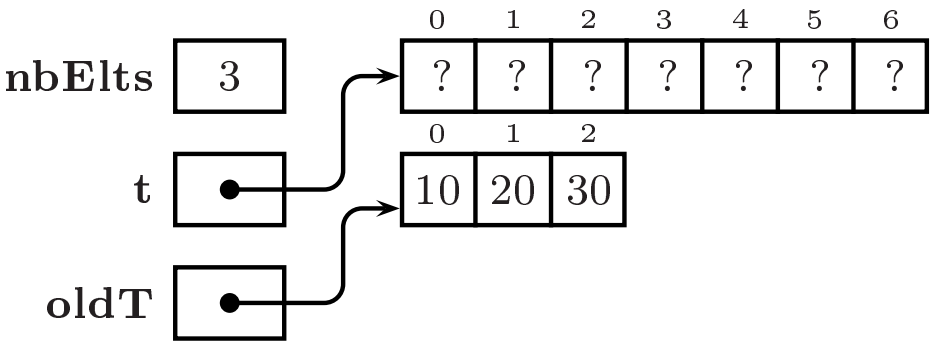
On doit d'abord sauvegarder la référence sur l'ancien tableau **t**

```
int [ ] oldT = t ;
```



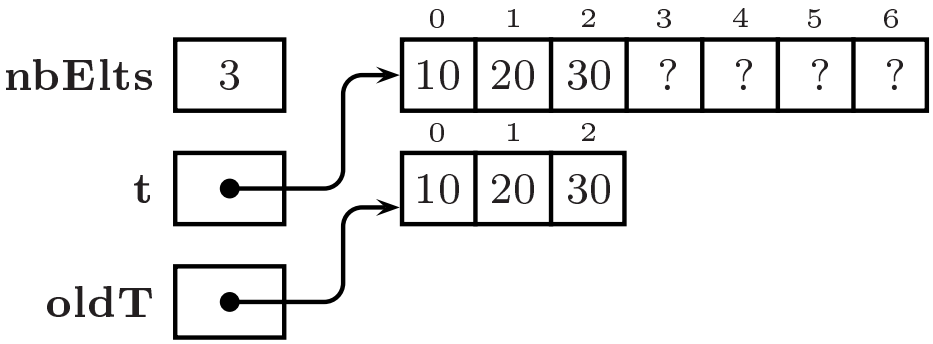
On recrée le tableau **t** en augmentant sa capacité

```
t = new int [ t . length + 4 ] ;
```



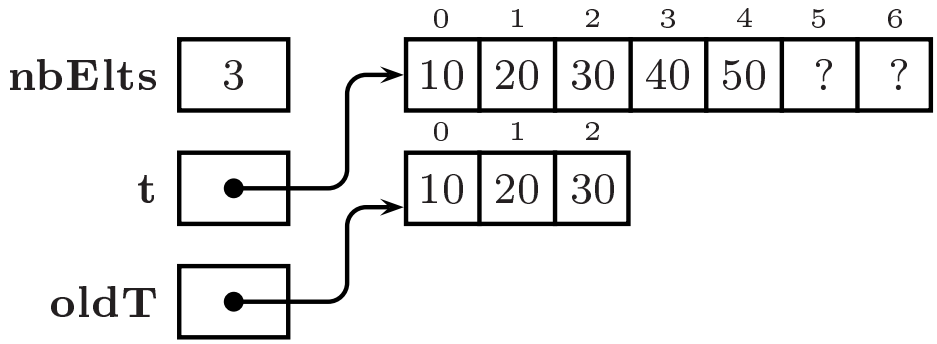
On recopie dans **t** les anciennes valeurs

```
for ( int i = 0 ; i < nbElts ; i ++ ) {
    t [ i ] = old [ i ] ;
}
```



On ajoute les valeurs 40 et 50

```
t [ nbElts ] = 40 ;
nbElts ++ ;

t [ nbElts ] = 50 ;
nbElts ++ ;
```



On pourra ajouter deux autres valeurs sans être obligé d'agrandir le tableau **t**

Exemple : la classe Tableau

Tableau
− String nom
− int [ ] a ;
+ Tableau (String n, int taille)
+ String getNom ( )
+ String getNbElts ( )
+ void saisir ( )
+ void afficher ( )
+ int somme ( )
+ double moyenne ( )

```
// Tableau.java
// ------------

public class Tableau {

    private String nom ;
    private int [ ] a ;

    public Tableau ( String n , int taille ) {
        nom = n ;
        // on suppose que taille est >= 0
        a = new int [ taille ] ;
    }

    public String getNom ( ) {
        return nom ;
    }

    public int getNbElts ( ) {
        return a . length ;
    }
```

```
public void saisir ( ) {
    System.out.println ( nom ) ;
    for ( int i = 0 ; i < a . length ; i ++ ) {
       a [ i ] = EntreeClavier.readInt(i + ":") ;
    }
}

public void afficher ( ) {
    System.out.print ( nom  + "[" ) ;
    if ( a . length != 0 ) {
       System.out.print ( a [ 0 ] ) ;
    }
    for ( int i = 1 ; i < a . length ; i++ ) {
       System.out.print ( "," + a [ i ] ) ;
    }
    System.out.println ( "]" ) ;
}
```

```
    public int somme ( ) {
        int som = 0 ;
        for ( int i = 0 ; i < a . length ; i ++ ) {
           som = som + a [ i ] ;
        }
        return som;
    }

    public double moyenne ( ) {
        return ( double ) somme ( ) / a . length ;
    }

} // fin Tableau.java
```

Utilisation de la classe Tableau

```
// TestTableau.java
// ------------

public class TestTableau {

    public static void main ( String [ ] args ) {

        Tableau a = new Tableau ( "a" , 5 ) ;

        Tableau b = new Tableau ( "b" , 3 ) ;

        a . saisir ( ) ;
        b . saisir ( ) ;

        a . afficher ( ) ;
        b . afficher ( ) ;

        System . out . println ( a . moyenne (  ) ) ;

    }
}
```

Compléments

La classe Point2D

Point2D
− double x , y
+ Point2D (double x , double x) + void setX (double x) + void setY (double y)

On doit distinguer les variables d'instance x , y des paramètres passés au constructeur ou aux méthodes x , y

Pour cela on utilise this qui est une référence sur l'instance courante

```
public class Point2D {

    private double x , x  ;

    public Point2D ( double x , double y  ) {
            ...
    }

    public void setX ( double x ) {
            ...
    }

    public void setY ( double y ) {
            ...
    }


}
```

this est une référence sur l'instance courante

```
public class Point2D {

    private double x , x  ;

    public Point2D ( double x , double y  ) {
        this . x = x ;
        this . y = y ;
    }

}
```

Tableaux d'objets

Variables et méthodes de classe

Variables de classe

Une variable de classe existe en un seul exemplaire.

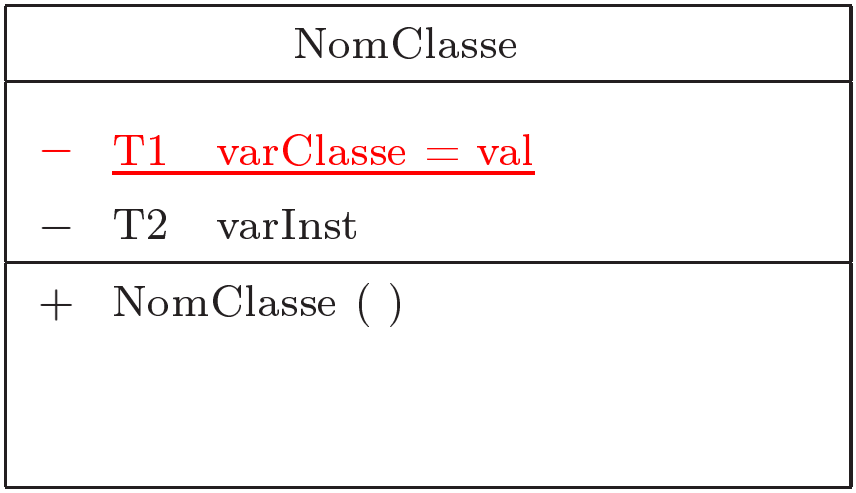
Elle est partagée par toutes les instances de la classe.

Une variable de classe existe même si aucune instance de la classe n'a été créée.

Une variable de classe doit être initialisée lors de sa déclaration

Diagramme de classe UML

Dans un diagramme une variable de classe est soulignée pour la distinguer d'une variable d'instance.



Déclaration d'une variable de classe en java

En java la déclaration d'une variable de classe est précédée du qualificatif **static**

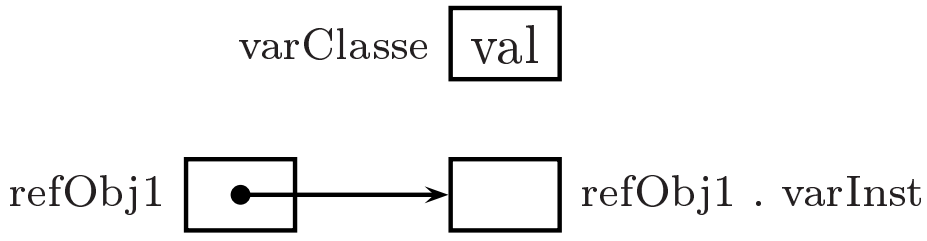
```
public class NomClasse {

    private static T1 varClasse = val ;

    private T2 varInst ;

    public NomClasse (  ) {
        ...
    }

      ...
}
```

Une variable de classe est initialisée lors de sa déclaration

Avant toute création d'instance de la classe NomClasse, on a

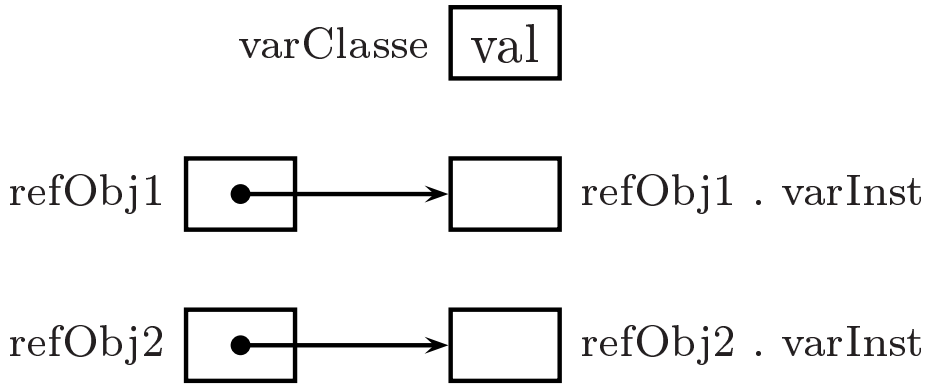
Après la création d'une instance de NomClasse, on a

```
NomClasse refObj1 = new NomClasse (  ) ;
```



Après la création d'une deuxième instance de NomClasse, on a

```
NomClasse refObj2 = new NomClasse (  ) ;
```



Méthodes de classe

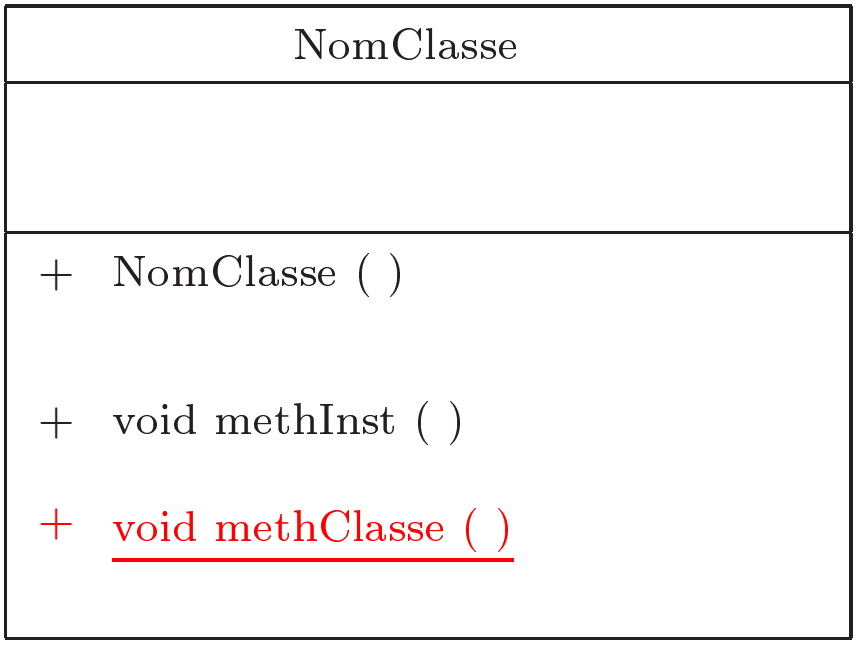
Une méthode de classe peut être appelée sans faire référence à une instance de la classe, contrairement à une méthode d'instance.

Une méthode de classe ne peut pas utiliser les variables d'instance de sa classe.

Une méthode de classe peut utiliser les variables de classe de sa classe.

Diagramme de classe UML

Dans un diagramme une méthode de classe est soulignée pour la distinguer d'une méthode d'instance.



Déclaration d'une méthode de classe en java

En java la déclaration d'une méthode de classe est précédée du qualificatif **static**

```
public class NomClasse {
     ...

    public void methInst (  ) {
        ...
    }

    public static void methClasse (  ) {
        ...
    }
      ...
}
```

Appel d'une méthode de classe

Pour appeler une méthode de classe on fait précéder l'appel du nom de la classe suivi d'un point

```
// appel d'une méthode de classe

NomClasse . methClasse (  ) ;

// appel d'une méthode d'instance

NomClasse refObj = new NomClasse ( ) ;

refObj . methInst ( ) ;
```

Numérotation des points

Pour numéroter les points successivement créés, on utilise

- une variable d'instance **numPoint** pour stocker le numéro du point
- une variable de classe **numPointSuivant** pour stocker le numéro du prochain point crée

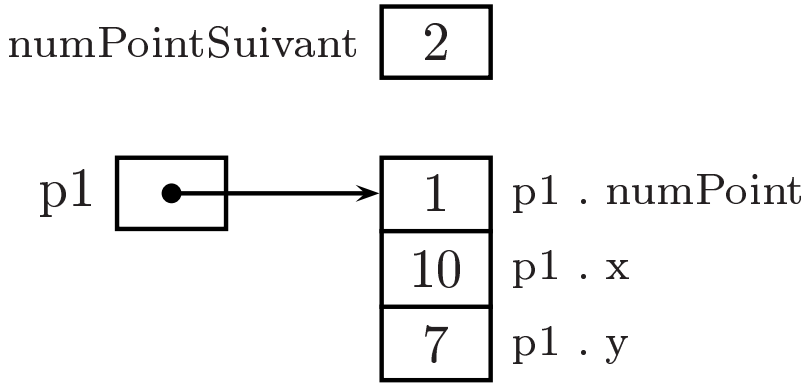
Chaque point possède sa propre variable d'instance **numPoint**.

La variable **numPointSuivant** qui permet de numéroter les points n'existe qu'en un seul exemplaire

Avant toute création de point, on a

On crée le point **p1**

```
Point2D p1 = new Point2D ( 10 , 7 ) ;
```



On crée ensuite le point **p2**

```
Point2D p2 = new Point2D ( -3 , 8 ) ;
```

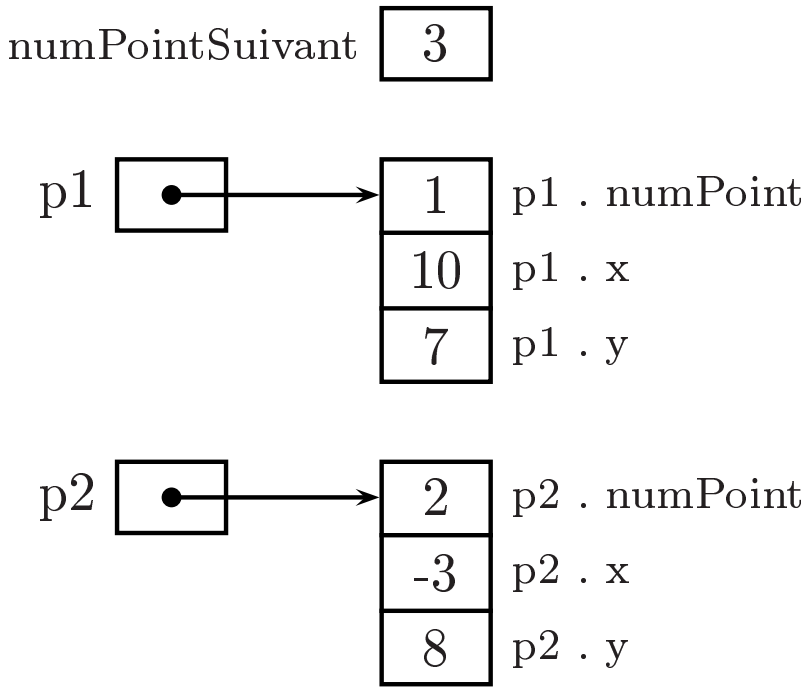
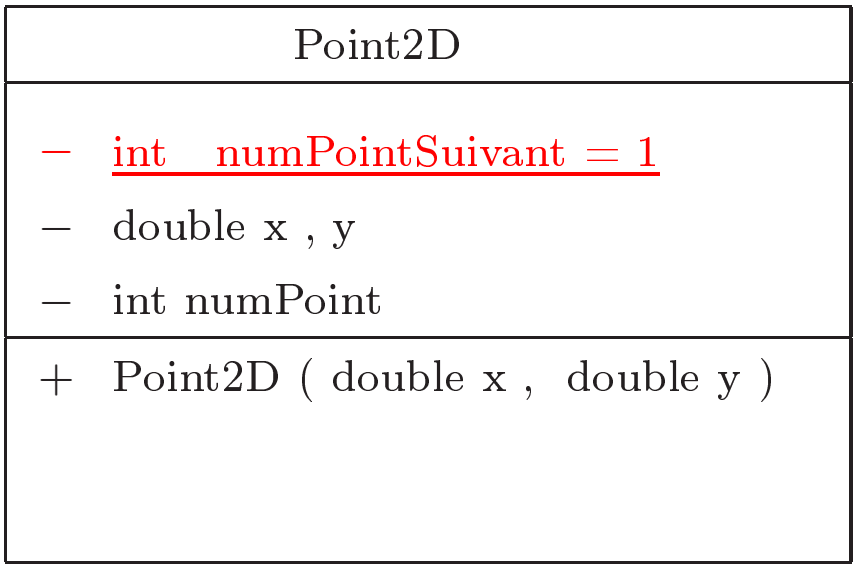


Diagramme de la classe Point2D



Le fichier Point2D.java

```
public class Point2D {

    private static int numPointSuivant = 1 ;

    private double x , y ;

    private int numPoint ;

    public Point2D ( double x , double y ) {
        this . x = x ; this . y = y ;
        numPoint = numPointSuivant  ;
        numPointSuivant ++ ;
    }

      ...
}
```

La variable de classe NumPointSuivant est initialisée lors de sa déclaration

Pour connaître le numéro du point qui va être crée, on doit prévoir une méthode de classe **getNumPointSuivant**

Le diagramme de classe devient :

Point2D
− int numPointSuivant = 1 − double x , y − int numPoint
+ Point2D (double x , double y) + int getNumPointSuivant ( )

Le fichier Point2D.java

```
public class Point2D {

    private static int numPointSuivant = 1 ;

    private double x , y ;

    private int numPoint ;

    public Point2D ( double x , double y ) {
        this . x = x ; this . y = y ;
        numPoint = numPointSuivant  ;
        numPointSuivant ++ ;
    }

    public static int getNumPointSuivant ( ) {
        return numPointSuivant ;
    }
      ...
}
```

Le programme de test

```
public class TestPoint2D {

    public static void main ( String [ ] args ) {

        System.out.println
             ( Point2D.getNumPointSuivant ( ) );

        Point2D p1 = new Point2D ( 10 , 7 ) ;
        Point2D p2 = new Point2D ( -3 , 8 ) ;

        System.out.println
             ( Point2D.getNumPointSuivant ( ) );
    }
}
```

affiche les valeurs **1** et **3**

Les constantes et fonctions mathématiques de la classe Math

Les constantes mathématiques **PI** et **E** sont des variables de classe de la classe **Math**.

Pour être accessibles à l'extérieur elles sont déclarées publiques.

Pour qu'elles ne soient pas modifiables, on ajoute le qualificatif **final**

Les fonctions mathématiques sont des méthodes de classe, on peut ainsi les appeler sans être obligé de créer une instance de la classe Math.

Les constantes de la classe Math

```
public class Math {

    public static final double PI = 3.1415... ;

    public static final double E = 2.718... ;

}
```

Les fonctions mathématiques de la classe Math

```
public class Math {
   public static int abs (  int x ){
        ...
    }
    public static long abs ( long x ) {
        ...
    }
    public static float abs ( float x ) {
        ...
    }
    public static double abs ( double x ) {
        ...
    }
    public static int round ( float x ) {
        ...
    }
    public static long round ( double x ) {
        ...
    }
    public static double sqrt ( double x ) {
        ...
    }
    public static double pow(double x,double y) {
        ...
    }
}
```

Les méthodes d'instance et de classe de la classe Complexe

Complexe
− double x , y
+ Complexe (double x , double y)
+ Complexe add (Complexe z)
+ Complexe sub (Complexe z)
+ Complexe mul (Complexe z)
+ Complexe div (Complexe z)
+ Complexe add (Complexe z1, Complexe z2)
+ Complexe sub (Complexe z1, Complexe z2)
+ Complexe mul (Complexe z1, Complexe z2)
+ Complexe div (Complexe z1, Complexe z2)

Calcul de la somme de deux complexes

```
Complexe a = new Complexe ( 1 , 2 ) ;
Complexe b = new Complexe ( 3 , 4 ) ;
```

en utilisant la méthode d'instance **add**

```
Complexe u = a . add ( b ) ;
```

en utilisant la méthode de classe **add**

```
Complexe v = Complexe . add ( a , b ) ;
```

Le fichier **Complexe.java**

```
public class Complexe {

    private double x , y ;

    public Complexe ( double x , double y ) {
        this . x = x ; this . y = y ;
    }

    public Complexe  add ( Complexe z ) {
        return new Complexe (x + z.x , y + z.y ) ;
    }

    public static Complexe  add
                  ( Complexe z1 , Complexe z2 ) {
        return
          new Complexe( z1.x + z2.x , z1.y + z2.y ) ;
    }
          ...
}
```

Les chaînes de caractères
La classe String

Les objets chaînes de caractères sont des instance de la classe **String**

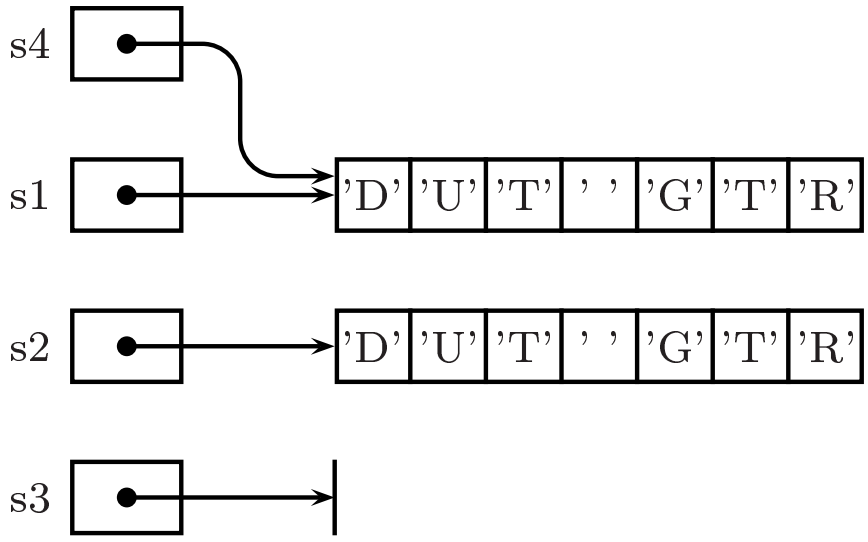
Une instance de la class **String** ne peut pas être modifiée

Les chaînes de caractères littérales (encadrées par des guillemets) sont des instances anonymes de la classe **String**

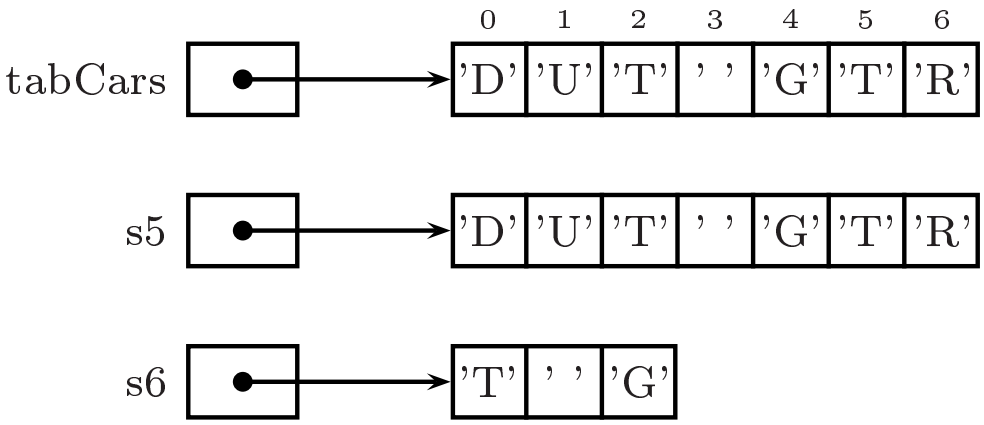
Les constructeurs de la classe **String**

```
public String ( )

public String ( String s )

public String ( char [ ] cars )

public String ( char [ ] cars ,
                int offset , int count )

public String ( byte [ ] octets )

public String ( byte [ ] octets ,
                int offset , int count )

public String ( StringBuffer sb )
```

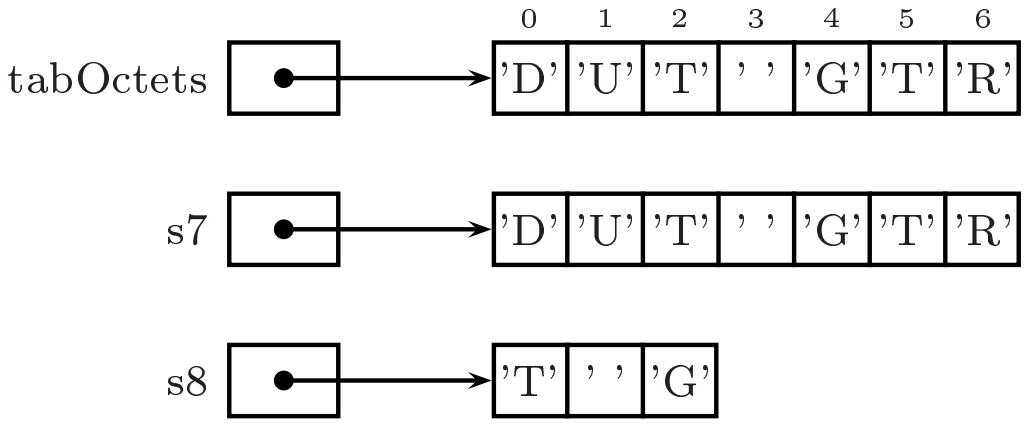
```
String s1 = "DUT GTR" ;

String s2 = new String ( s1 ) ;

String s3 = new String (  ) ;

String s4 = s1 ;
```



```
char [ ] tabCars = { 'D', 'U' , 'T' , ' ' ,
                     'G' , 'T' , 'R'
} ;

String s5 = new String ( tabCars ) ;

String s6 = new String ( tabCars , 2 , 3 ) ;
```



```
byte [ ] tabOctets = { (byte)'D', ( byte)'U' ,
            (byte)'T' , (byte)' ' , (byte)'G' ,
            (byte)'T' , (byte)'R'
} ;

String s7 = new String ( tabOctets ) ;

String s8 = new String ( tabOctets , 2 , 3 ) ;
```



Longueur d'une chaîne de caractères

La méthode **length**

```
public int length ( )
```

retourne la longueur de la chaîne de caractères

`s1 . length ( )` est égal à 7

`s3 . length ( )` est égal à 0

`s6 . length ( )` est égal à 3

Ne pas confondre la méthode length d'une chaîne de caractères avec l'attribut length d'un tableau

`tabCars.length` donne le nombre d'éléments du tableau tabCars

`s1.length ( )` donne la longueur de la chaîne de caractères s1

Accès au caractère de rang i

La méthode **charAt**

```
public char charAt ( int index )
```

retourne le caractère de rang index.

`s . charAt ( i )` retourne la caractère d'indice `i` de la chaîne `s`

Si `i` n'est pas compris entre 0 à `s . length() - 1` une exception instance de **StringIndexOutOfBoundsException** est déclenchée.

Il n'existe pas de méthode **setCharAt**

Afficher la chaîne de caractères **s** à l'envers

```
for ( int i = s . length ( ) - 1 ; i >= 0 ; i -- ) {
    System . out . print ( s . charAt ( i ) ) ;
}
System . out . println ( ) ;
```

Compter le nombre d'espaces de la chaîne **s**

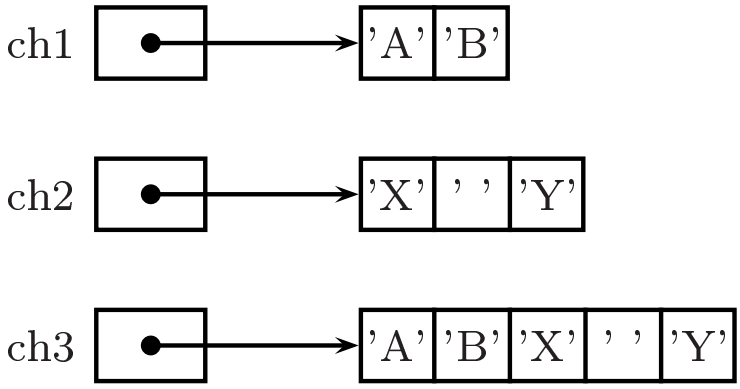
```
int nbEspaces = 0 ;

for ( int i = 0 ; i < s . length ( ) ; i ++ ) {
    if ( s . charAt ( i ) == ' ' ) {
        nbEspaces ++ ;
    }
}
```

Concaténation

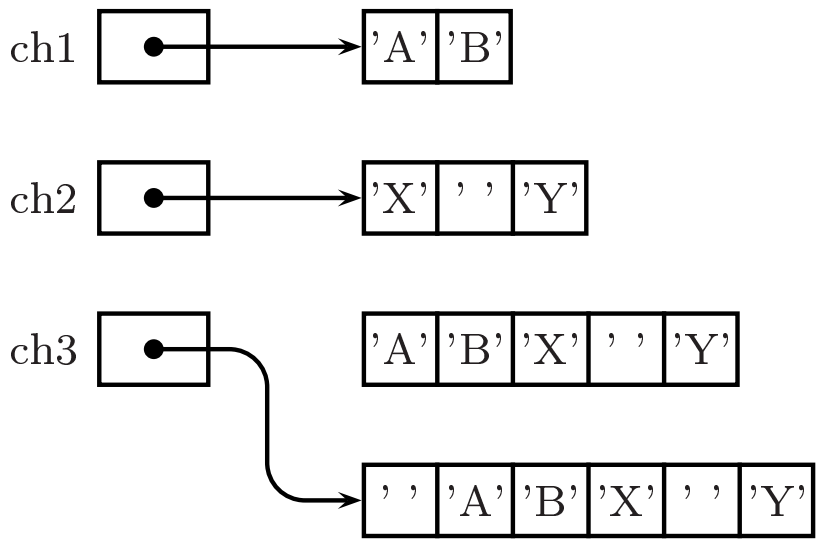
L'opérateur + permet de concaténer deux chaînes de caractères.

Le résultat est une nouvelle chaîne de caractères

```
String ch1 = "AB" ;

String ch2 = "X Y" ;

String ch3 = ch1 + ch2 ;
```



```
ch3 = " " + ch3 ;
```



Egalité de deux chaînes

Les méthodes **equals** et **equalsIgnoreCase** permettent de tester l'égalité de deux chaînes

```
public boolean equals ( Object obj )
```

```
public boolean equalsIgnoreCase ( String s )
```

La deuxième méthode ne tient pas compte des majusules et minuscules

```
String x = "ABC" ;
```

```
String y = "abc" ;
```

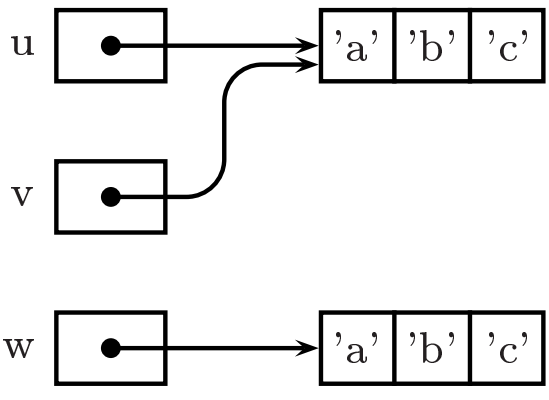
`x . equals ( y )` est égal à false

`x . equalsIgnoreCase ( y )` est égal à true

Ne pas utiliser l'opérateur == pour comparer deux chaînes de caractères, plus généralement deux objets

```
String u = "abc" ;

String v = u ;

String w = new String ( u ) ;
```

u == v	true
u == w	false
u . equals (v)	true
u . equals (w)	true



Comparaison de deux caractères

Dans une expression arithmétique ou logique un caractère est considéré comme un entier dont la valeur est celle de son code **unicode**

Ces instructions

```
int code = 'A' ;

System . out . println ( code ) ;
```

permettent d'afficher le code **Unicode** du caractère **A**

Pour comparer deux caractères on utilise l'opérateur $<$

```
' ' < ...  < '0' < '1' < ... < '9'< ... <

'A' < 'B' ... < 'Z' < ... < 'a' < 'b' < ... < 'z'
```

Comparaison de deux chaînes caractères
L'ordre lexicographique

s1 précède **s2** qu'on écrit **s1** $\prec$ **s2** **ssi**

1. s1 est un préfixe de s2
   bal $\prec$ ballon
2. Le premier caractère de **s1** qui ne coïncide pas avec le caractère de même rang de **s2** est inférieur à ce dernier
   ballon $\prec$ bas

s1 est un préfixe de s2

- `s1 . length ( ) == s2 . length ( )`

- `s1 . charAt ( i ) == s2 . charAt ( i )`
  i = 0 , ... , s1 . length ( ) - 1

Le premier caractère de **s1** qui ne coïncide pas avec le caractère de même rang de **s2** est inférieur à ce dernier

$\exists$ k >= 0 et inférieur aux longueurs de s1 et s2

– `s1 . charAt ( i ) == s2 . charAt ( i )`
i = 0 , ... , k - 1

– `s1 . chartAt ( k ) < s2 . charAt ( k )`

La méthode compareTo

La méthode

```
public int compareTo ( Object obj )
```

permet de comparer la chaîne courante avec la chaîne passée en paramètre

```
s1 . compareTo ( s2 )
```

retourne une valeur $\begin{cases} < 0 & \text{si s1} \prec \text{s2} \\ 0 & \text{si s1 est égal à s2} \\ > 0 & \text{si s1} \succ \text{s2} \end{cases}$

La méthode compareToIgnoreCase

La méthode

```
public int compareToIgnoreCase ( String s )
```

permet de comparer la chaîne courante avec la chaîne passée en paramètre, sans tenir compte des majuscules et miniscules

Recherche d'un caractère (à partir du début)

La méthode

```
public int indexOf ( int car )
```

permet de rechercher le caractère passé en paramètre dans la chaîne courante

Elle retourne l'indice de la première occurrence du caractère **car** dans la chaîne courante, -1 si le caractère **car** n'a pas été trouvé

Recherche d'un caractère (à partir du début)

La méthode

```
public int indexOf ( int car , int ideb )
```

permet de rechercher le caractère **car** dans la chaîne courante à partir de l'indice **ideb**.

Elle retourne l'indice de la première occurrence du caractère **car** dans la chaîne courante à partir de l'indice **ideb**, -1 si le caractère **car** n'a pas été trouvé

Compter le nombre d'espaces de la chaîne **s**

```
int nbEspaces = 0 ;

int i = s . indexOf ( ' ' ) ;

while ( i != -1 ) {
    nbEspaces ++ ;
    i = s . indexOf ( ' ' , i + 1 ) ;
}
```

Recherche d'un caractère à partir de la fin

La méthode

```
public int lastIndexOf ( int car )
```

permet de rechercher le caractère passé en paramètre dans la chaîne courante à partir de la fin.

Elle retourne l'indice de la dernière occurrence du caractère **car** dans la chaîne courante, -1 si le caractère **car** n'a pas été trouvé

Recherche d'un caractère à partir de la fin

La méthode

```
public int lastIndexOf ( int car , int ifin )
```

permet de rechercher le caractère **car** dans la chaîne courante à partir de l'indice **ifin**.

Elle retourne l'indice de la dernière occurrence du caractère **car** dans la chaîne courante à partir de l'indice **ifin**, -1 si le caractère **car** n'a pas été trouvé

Test si une chaîne est un préfixe

La méthode

```
public boolean startsWith ( String s )
```

retourne **true** si la chaîne courante commence par **s**, **false** sinon.

Test si une chaîne est un suffixe

La méthode

```
public boolean endsWith ( String s )
```

retourne **true** si la chaîne courante se termine par **s**, **false** sinon.

Recherche d'une sous-chaîne (à partir du début)

La méthode

```
public int indexOf ( String s )
```

permet de rechercher la chaîne passée en paramètre dans la chaîne courante

Elle retourne l'indice de la première occurrence de la chaîne **s** dans la chaîne courante, -1 si la chaîne **s** n'a pas été trouvée

Recherche d'une sous-chaîne (à partir du début)

La méthode

```
public int indexOf ( String , int ideb )
```

permet de rechercher la chaîne **s** dans la chaîne courante à partir de l'indice **ideb**.

Elle retourne l'indice de la première occurrence de la chaîne **s** dans la chaîne courante à partir de l'indice **ideb**, -1 si la chaîne **s** n'a pas été trouvée

Recherche d'une sous-chaîne à partir de la fin

La méthode

```
public int lastIndexOf ( String s )
```

permet de rechercher la chaîne **s** dans la chaîne courante à partir de la fin

Elle retourne l'indice de la dernière occurrence de la chaîne **s** dans la chaîne courante, -1 si la chaîne **s** n'a pas été trouvée

Recherche d'une sous-chaîne à partir de la fin

La méthode

```
public int lastIndexOf ( String s , int ifin )
```

permet de rechercher la dernière occurrence de la chaîne **s** dans la chaîne courante à partir de l'indice **ifin**.

Elle retourne l'indice de la dernière occurrence de la chaîne **s** dans la chaîne courante à partir de l'indice **ifin**, -1 si la chaîne **s** n'a pas été trouvé

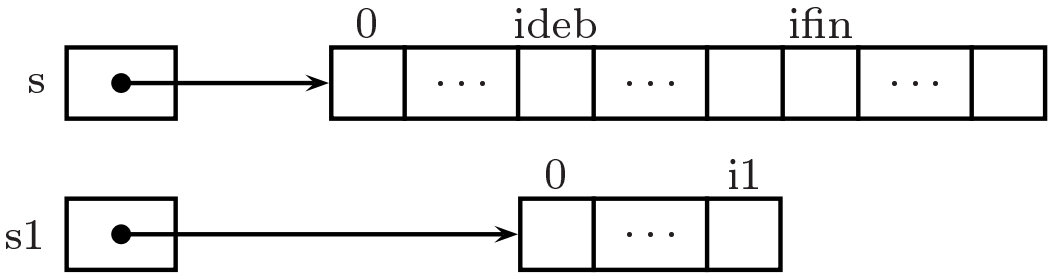
Extraction d'une sous-chaîne

La méthode

```
public String substring ( int ideb , int ifin )
```

retourne une nouvelle chaîne constituée des caractères de la chaîne courante dont les indices sont compris entre **ideb** et **ifin - 1**

```
s1 = s . substring ( ideb , idfin ) ;
```




```
i1 = ifin - ideb - 1
```

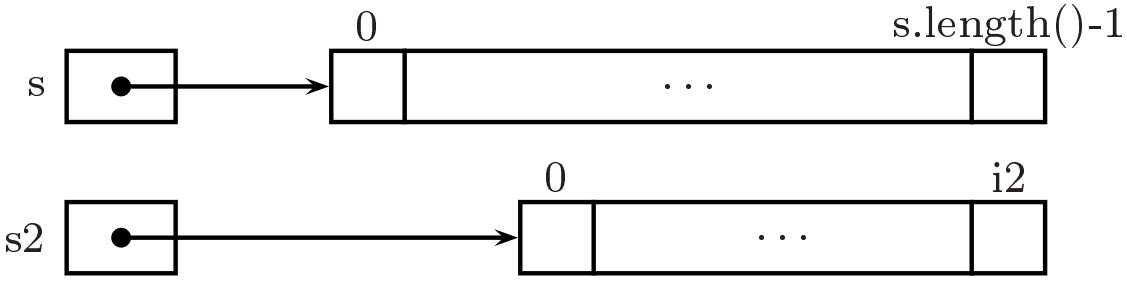
Extraction d'une sous-chaîne (suite)

La méthode

```
public String substring ( int ideb  )
```

retourne une nouvelle chaîne constituée des caractères de la chaîne courante dont les indices sont compris entre **ideb** et **s . length ( ) - 1**

```
s2 = s . substring ( ideb ) ;
```




```
i2 = s . length ( ) - ideb - 1
```

Transformation en majuscules ou en minuscules

La méthode

```
public String toUpperCase ( )
```

crée une nouvelle chaîne obtenue à partir de la chaîne courante en transformant les minuscules en majuscules.

La chaîne courante n'est pas modifiée

La méthode

```
public String toLowerCase ( )
```

crée une nouvelle chaîne obtenue à partir de la chaîne courante en transformant les majuscules en minuscules

La chaîne courante n'est pas modifiée

Suppression des blancs de début et de fin

La méthode

```
public String trim (  )
```

La méthode toString retourne une nouvelle chaîne obtenue à partir de la chaîne courante en supprimant les blancs (white spaces) de début et de fin

Les caractères blancs sont les caractères suivants

- ' ' (espace)
- '\n' LF
- '\r' CR
- '\t' HT
- '\f' FF

Conversion en chaîne de caractères

Les méthodes de classes

```
public static String valueOf ( boolean b )

public static String valueOf ( char c )

public static String valueOf ( int i )

public static String valueOf ( long l )

public static String valueOf ( float f )
```

permettent de convertir leur paramètre en chaîne de caractères

Les tableaux de chaînes de caractères

Les paramètres de la ligne de commande

Les classes enveloppes

Les types primitifs (int, boolean, char ...) ne sont pas des classes

Pour chaque type primitif il existe une classe enveloppe qui porte le même nom que le type primitif et dont la première lettre est en majuscules.

Type primitif	Classe enveloppe
byte	Byte
short	Short
int	Integer
long	Long
char	Character
float	Float
double	Double
boolean	Boolean

La classe Integer

Elle possède deux variables de classe finales

- **MAX_VALUE**
- **MIN_VALUE**

qui donnent le plus grand et le plus petit entier

Integer
+ int MAX_VALUE = 2147486647 { final } + int MIN_VALUE = -2147486648 { final }

```
public class Test1 {

    public satic void main ( String [ ] args ) {

        System.out.println ( Integer . MAX_VALUE ) ;

        System.out.println ( Integer . MIN_VALUE ) ;

    }
}
```

Elle possède deux constructeurs

Integer
+ Integer (int i) + Integer (String s)

qui permettent de construire un **Integer** à partir d'un **int** ou d'une **String**

La chaîne de caractères passée au constructeur doit bien sur représenter un nombre entier.

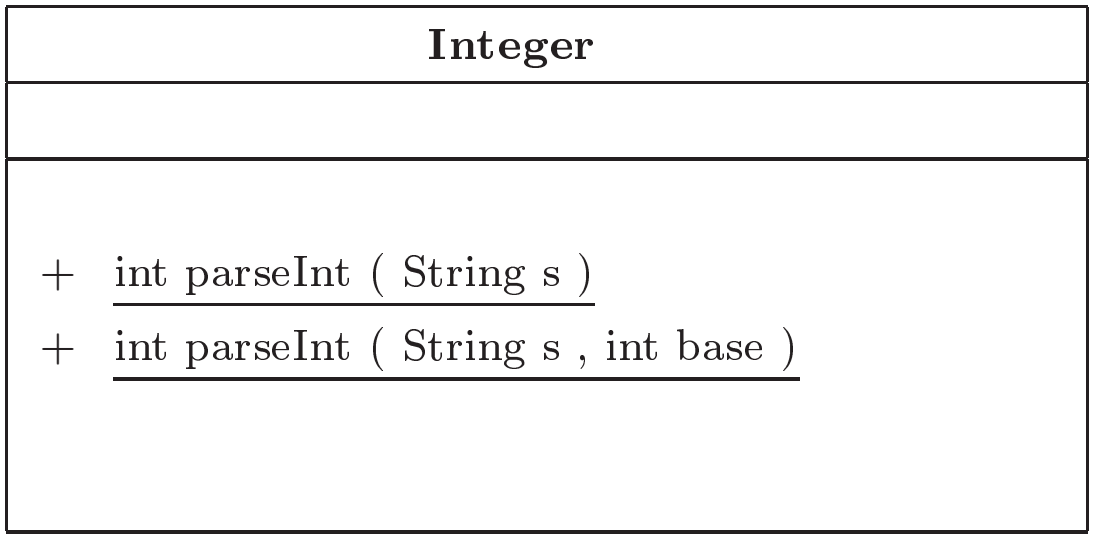
Les méthodes suivantes :

Integer
+ int intValue ( ) + byte byteValue ( ) + short shortValue ( ) + long longValue ( ) + float floatValue ( ) + double doubleValue ( )

retourne la valeur d'un **Integer** sous forme d'un **int**, **byte** ...

```
public class Test2 {

    public static void main ( String [ ] args ) {

      // Création de deux Integer i1 et i2
      // dont les valeurs sont 100 et 50

        Integer i1 = new Integer ( 100 ) ;
        Integer i2 = new Integer (  50 ) ;

      // calcul de la somme sous forme d'un Integer

         int som = i1.intValue( ) + i2.intValue( ) ;
         Integer i3 = new Integer ( som ) ;


    }
}
```

Les méthodes de classe suivantes



permettent de convertir la chaîne de caractères **s** passée en paramètre en **int**

la première dans le système décimal

la seconde dans une base quelconque

La chaîne de caractères **s** passée en paramètre doit représenter un nombre sinon une exception **NumberFormatException** est levée

Exercice

Ecrire la méthode **Add2** qui permet de calculer la somme des deux argument de la ligne de commande

Par exemple la commande

```
java Add2 100 50
```

fournit la résultat

```
100+50=150
```

Les deux arguments de la ligne de commande sont des chaînes de caractères

```
public class Add2 {

    public static void main ( String [ ] args ) {

        int n1 = Integer . parseInt ( args [ 0 ] ) ;
        int n2 = Integer . parseInt ( args [ 1 ] ) ;

        int som = n1 + n2 ;

        System.out.println
                ( n1 + "+" + n2 +"=" + som ) ;

    }
}
```

Les méthodes de classe suivantes

Integer
+ String toString (int i) + String toString (int i , int base)

permettent de convertir l'entier passé en paramètre sous forme d'une chaîne de caractères

la première dans le système décimal

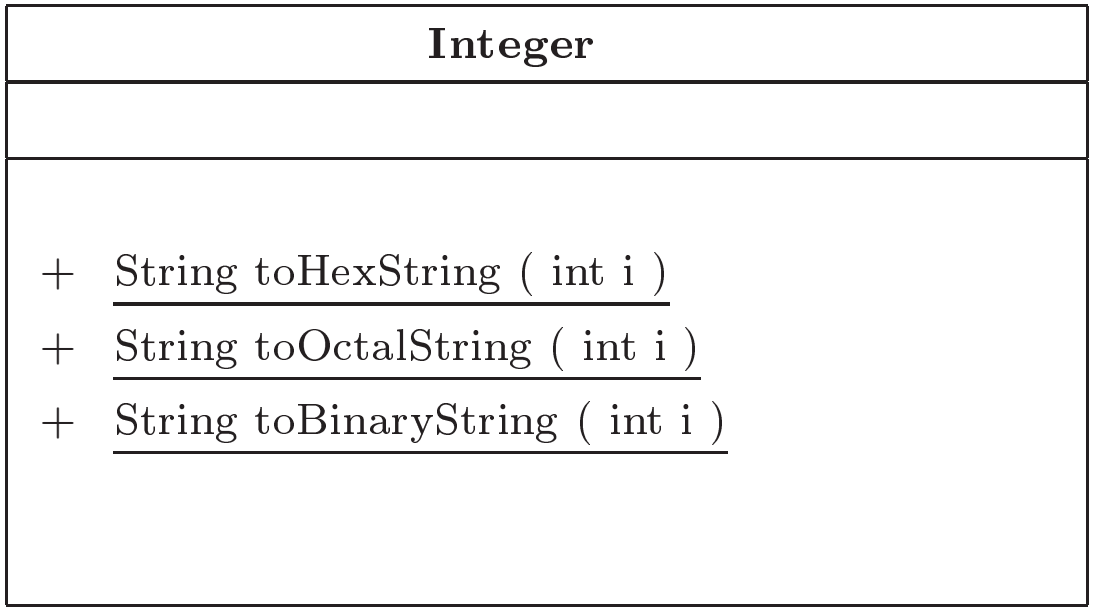
la seconde dans une base quelconque

Le programme suivant :

```
public class TestBases {

    public static void main ( String [ ] args ) {

        int n = 20 ;

        String hex = Integer . toString ( n , 16 ) ;
        String oct = Integer . toString ( n ,  8 ) ;
        String bin = Integer . toString ( n ,  2 ) ;

        System.out.println ( hex ) ;
        System.out.println ( oct ) ;
        System.out.println ( bin ) ;

    }
}
```

affiche les représentations hexadécimale, octale et binaire de 20

On peut aussi utiliser les méthodes de classe suivantes



Les classes StringBuffer et StringTokenizer

Utilisation des exceptions

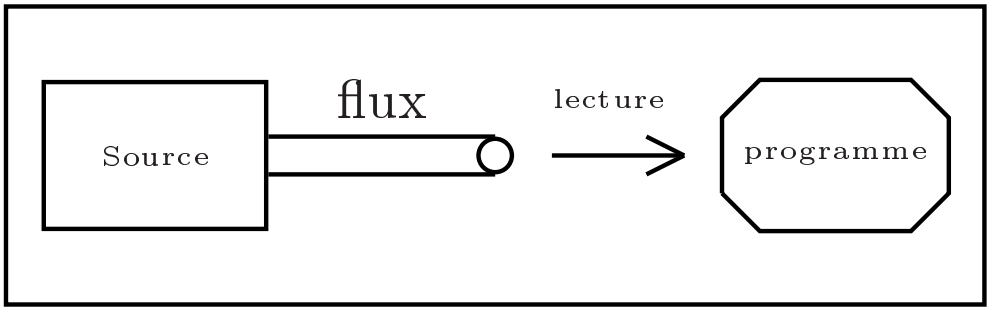
Les tableaux à deux dimensions en Java

Les opérateurs bit à bit

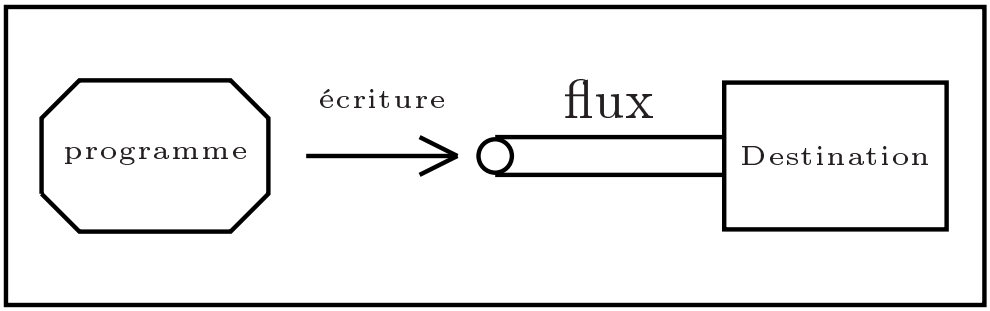
Les flux

Les flux

1. Flux d'entrée



2. Flux de sortie



Java distingue

1. les flux d'octets
2. et les flux de caractères

Les flux d'octets sont gérés par les classes
dont le suffixe est

InputStream
ou **OutputStream**

Les flux de caractères sont gérés par les classes
dont le suffixe est

Reader
ou **Writer**

Les différentes sources et destinations possibles sont données par le tableau suivant :

Source Destination	Flux de caractères	Flux d'octets
Mémoire	CharArrayReader CharArrayWriter	ByteArrayInputStream ByteArrayOutputStream
	StringReader StringWriter	StringBufferInputStream
Tube	PipedReader	PipedInputStream
	PipedWriter	PipedOutputStream
Fichier	FileReader	FileInputStream
	FileWriter	FileOutputStream

Java distingue

1. les flux sans traitement des données
2. et les flux avec traitement des données

Le tableau suivant donne les différents traitements possibles suivant les flux de caractères ou d'octets

Traitement	Flux de caractères	Flux d'octets
Bufferisation	BufferedReader BufferedWriter	BufferedInputReader BufferedOutputReader
Filtrage	FilterReader FilterWriter	FilterInputStream FilterOutputStream
Conversion entre octets et caractères	InputStreamReader OutputStreamReader	
Concaténation		SequenceInputStream
Sérialisation d'objet		ObjectInputStream ObjectOutputStream
Conversion de données		DataInputStream DataOutputStream
Numérotation des lignes	LineNumberReader	LineNumberInputStream
Lecture avec retour arrière	PushBackReader	PushBackInputStream
Impression	PrintWriter	PrintStream

La classe abstraite InputStream

java . lang . Object

└ java . io . InputStream

Constructeur

```
public InputStream ( )
```

Méthodes

```
public abstract int read ( )  throws IOException
public int read ( byte [ ] b ) throws IOException
public int read ( byte [ ] b , int off , int len )
                              throws IOException
public void close ( ) throws IOException
public void reset ( ) throws IOException
public long skip ( long n ) throws IOException
public int available ( ) throws IOException
```

La methode int **read ( )** retourne
- l'octet lu à partir du flux (valeur comprise entre 0 et 255)
- -1 à la fin du flux

Les autres méthodes **read** retournent
- le nombre d'octets lus
- -1 à la fin du flux

La classe abstraite OutputStream

java . lang . Object

└ java . io . OutputStream

Constructeur

```
public OutputStream ( )
```

Méthodes

```
public abstract void write ( int b )  throws IOException
public void write ( byte [ ] b ) throws IOException
public void write ( byte [ ] b , int off , int len )
                                throws IOException
public void close ( ) throws IOException
public void flush ( ) throws IOException
```

La methode void **write (int b)** écrit un octet dans le flux.

La classe FileInputStream

```
java . lang . Object
  └ java . io . InputStream
      └ java . io . FileInputStream
```

Constructeurs

```
public FileInputStream ( String nomFich )
                                  throws FileNotFoundException
public FileInputStream ( File f ) throws FileNotFoundException
public FileInputStream ( FileDescriptor fd )
```

Méthodes

```
public int read ( )  throws IOException
public int read ( byte [ ] b ) throws IOException
public int read ( byte [ ] b , int off , int len )
                                 throws IOException
public void close ( ) throws IOException
public long skip ( long n ) throws IOException
public int available ( ) throws IOException
public FileDescriptor getFD ( ) throws IOException
```

La classe FileOutputStream

```
java . lang . Object
  └ java . io . OutputStream
      └ java . io . FileOutputStream
```

Constructeurs

```
public FileOutputStream ( String nomFich )
                                    throws FileNotFoundException
public FileOutputStream ( String nomFich , boolean append )
                                    throws FileNotFoundException
public FileOutputStream ( File f ) throws FileNotFoundException
public FileOutputStream ( FileDescriptor fd )
```

Méthodes

```
public void write ( int b ) throws IOException
public void write ( byte [ ] b ) throws IOException
public void write ( byte [ ] b , int off , int len )
                                  throws IOException
public void close ( ) throws IOException
public FileDescriptor getFD ( ) throws IOException
```

Ecrire la classe **NbOCtets** qui permet d'afficher le nombre d'octets du fichier dont le nom est passé en paramètre de la ligne de commande

La commande

```
java NbOctets fic
```

affiche le nombre d'octets du fichier **fic**

Ecrire la classe **NbLignes** qui permet d'afficher le nombre de lignes du fichier dont le nom est passé en paramètre de la ligne de commande

La commande

```
java NbLignes fic
```

affiche le nombre de lignes du fichier **fic**

Ecrire la classe **Copie** qui permet de copier un fichier dans un autre, les noms des fichiers sont donnés par les paramètres de la ligne de commande : fichier source puis fichier destination

La commande

```
java Copie fic1 fic2
```

copie le fichier **fic1** dans le fichier **fic2**.

La classe abstraite Reader

java . lang . Object

└ java . io . Reader

Variables d'instance

```
proctected Object lock
```

Constructeurs

```
protected Reader ( )
protected Reader ( Object lock )
```

Méthodes

```
public abstract int read ( )  throws IOException
public int read ( char [ ] cbuf ) throws IOException
public int read ( char [ ] cbuf , int off , int len )
                                  throws IOException
public void close ( ) throws IOException
public void reset ( ) throws IOException
public long skip ( long n ) throws IOException
public boolean ready ( ) throws IOException
```

La methode int **read ( )** retourne
- le caractère lu à partir du flux (valeur comprise entre 0 et 65535)
- -1 à la fin du flux

Les autres méthodes **read** retournent
- le nombre d'octets lus
- -1 à la fin du flux

La classe abstraite Writer

java . lang . Object

└ java . io . Writer

Variables d'instance

```
proctected Object lock
```

Constructeurs

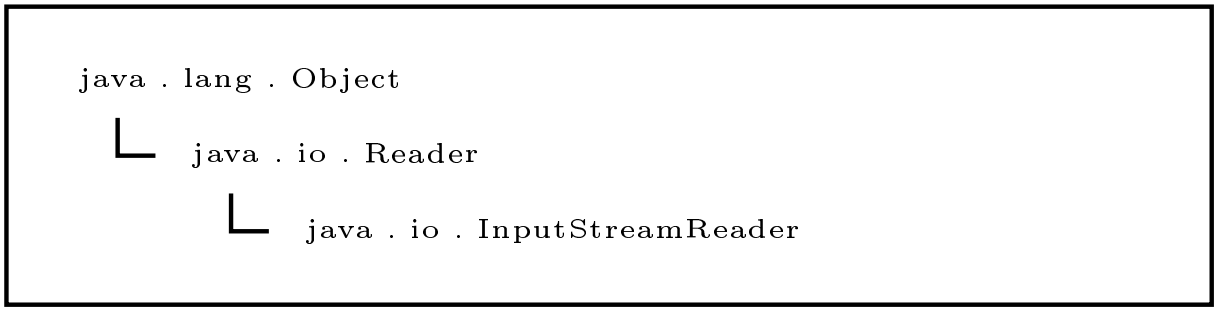
```
protected Writer ( )
protected Writer ( Object lock )
```

Méthodes

```
public void write ( int c)  throws IOException
public abstract void write ( char [ ] cbuf ) throws IOException
public abstract void write ( char [ ] cbuf , int off , int len )
                     throws IOException
public abstract void close ( ) throws IOException
public abstract void flush ( ) throws IOException
```

La methode void **write** (int c) écrit un caractère dans le flux.

La classe abstraite InputStreamReader



Cette classe est un pont entre les flux de caractères et les flux d'octets : elle lit des octets et les transforme en caractères compte-tenu du mode d'encodage des caractères défini.

Constructeurs

```
public InputStreamReader ( InputStream in  )
public InputStreamReader ( InputStreamReader in  , String enc )
                throws UnsupportedEncodingException
```

Méthodes

```
public int read ( )  throws IOException
public int read ( char [ ] cbuf , int off , int len )
                            throws IOException
public void close ( ) throws IOException
public boolean ready ( ) throws IOException
public String getEncoding ( )
```

La classe abstraite OutputStreamReader

java . lang . Object

└ java . io . Writer

└ java . io . OutputStreamWriter

Cette classe est un pont entre les flux de caractères et les flux d'octets : les caractères écrits sont transformés en octets compte-tenu du mode d'encodage des caractères défini.

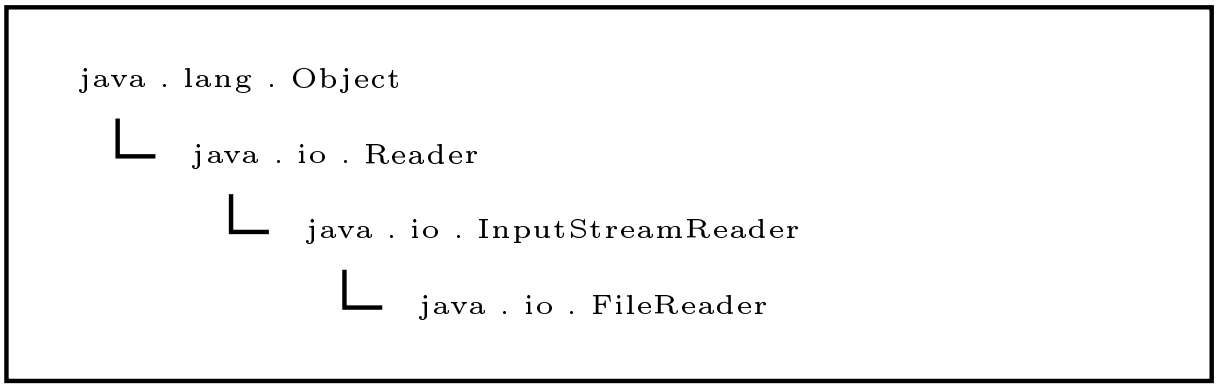
Constructeurs

```
public OutputStreamWriter ( OutputStream out )
public OutputStreamWriter
    ( OutputStreamReader out  , String enc )
                throws UnsupportedEncodingException
```

Méthodes

```
public void write ( int c )  throws IOException
public void write ( char [ ] cbuf , int off , int len )
                                throws IOException
public void write ( String str, int off , int len )
                                throws IOException
public void close ( ) throws IOException
public void flush ( ) throws IOException
public boolean ready ( ) throws IOException
public String getEncoding ( )
```

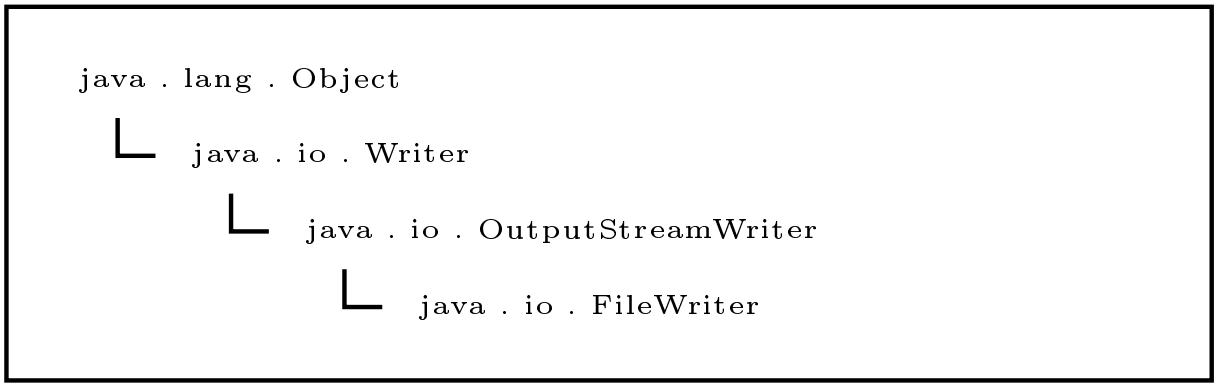
La classe FileReader



Constructeurs

```
public FileReader ( String nomFich ) throws FileNotFoundException
public FileReader ( File f ) throws FileNotFoundException
public FileReader ( FileDescriptor fd )
```

La classe FileWriter



Constructeurs

```
public FileWriter ( String nomFich )
public FileWriter ( String nomFich  , boolean append )
public FileWriter ( File f )
public FileWriter ( FileDescriptor fd )
```

Reprendre les classes

- **NbOctets**
- **NbLignes**
- **Copie**

en utilisant les classes **FileReader** et **FileWriter**

La classe BufferdReader

java . lang . Object
└ java . io . Reader
└ java . io . BufferdReader

Constructeurs

```
public BufferdReader ( Reader in )
public BufferdReader ( Reader int  , int sz )
```

Méthodes

```
public String readLine ( ) throws IOException
```

La méthode readLine ( ) retourne **null** à la fin du flux

Utiliser la classe **BufferedReader** pour effectuer un lecture ligne par ligne du fichier dont le nom est passé en paramètre de la ligne de commande.

Les classes DatatInputStream et DataOutputStream

Ces deux classes permettent de lire et écrire des données en binaire (c'est à dire sans aucune conversion).

La classe DataInputStream

java . lang . Object
└ java . io . InputStream
└ java . io . FilterInputStream
└ java . io . DataInputStream

Constructeur

```
DataInputStream ( InputStream in )
```

Méthodes

```
public boolean readBoolean (  ) throws IOException
public byte readByte (  ) throws IOException
public char readChar (  ) throws IOException
public double readDouble (  ) throws IOException
public float readFloat (  ) throws IOException
public int readInt (  ) throws IOException
public long readLong (  ) throws IOException
public short readShort (  ) throws IOException
public int readUnsignedByte (  ) throws IOException
public int readUnsignedShort (  ) throws IOException
public String readUTF (  ) throws IOException
public static String readUTF ( DataInput in ) throws IOException
public public int skipBytes ( int n ) ;
public float readFloat (  ) throws IOException
public float readFloat (  ) throws IOException
```

Ces méthodes déclenchent une exception de type
- EOFException à la fin du flux
- IOException en cas d'erreur de lecture

La classe DataOutputStream

java . lang . Object
└ java . io . OutputStream
└ java . io . FilterOutputStream
└ java . io . DataOutputStream

Constructeur

```
DataOutputStream ( OutputStream out )
```

Méthodes

```
public void writeBoolean (  boolean b ) throws IOException
public void writeByte (  byte b ) throws IOException
public void writeBytes (  String s ) throws IOException
public void writeChar ( char c ) throws IOException
public void writeChars ( String s ) throws IOException
public void writeDouble ( double d  ) throws IOException
public void writeFloat (  float f ) throws IOException
public void writeInt ( int i ) throws IOException
public void writeLong ( long l  ) throws IOException
public void writeShort ( short s ) throws IOException
public void writeUTF ( String s ) throws IOException
```

Les fichiers texte

Les fichiers binaire