Un moule pour préparer des cakes


SAHLA MAHLA
المصدر الاول للطالب الجزائري

Poids:200g
Qualité:passable
Durée de la cuisson: 35mn

Découper cake
Décorer cake
Congeler cake

Deuxième cake

Poids:300g
Qualité:catastrophique
Durée de la cuisson: 55mn

Découper cake
Décorer cake
Congeler cake

Troisièmecake

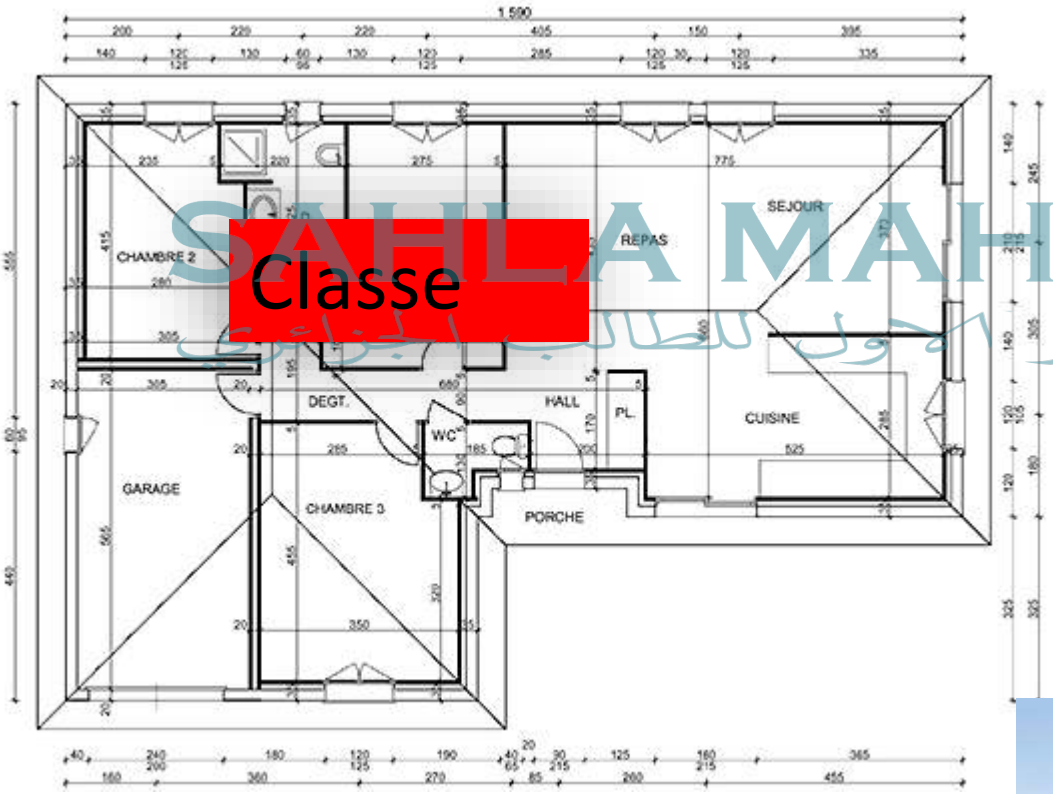
Poids:100g
Qualité:Excellente
Durée de la cuisson: 30mn

Découper cake
Décorer cake
Congeler cake

Classe

Objets ou instances

- Premier cake
- Deuxième cake
- Troisième cake

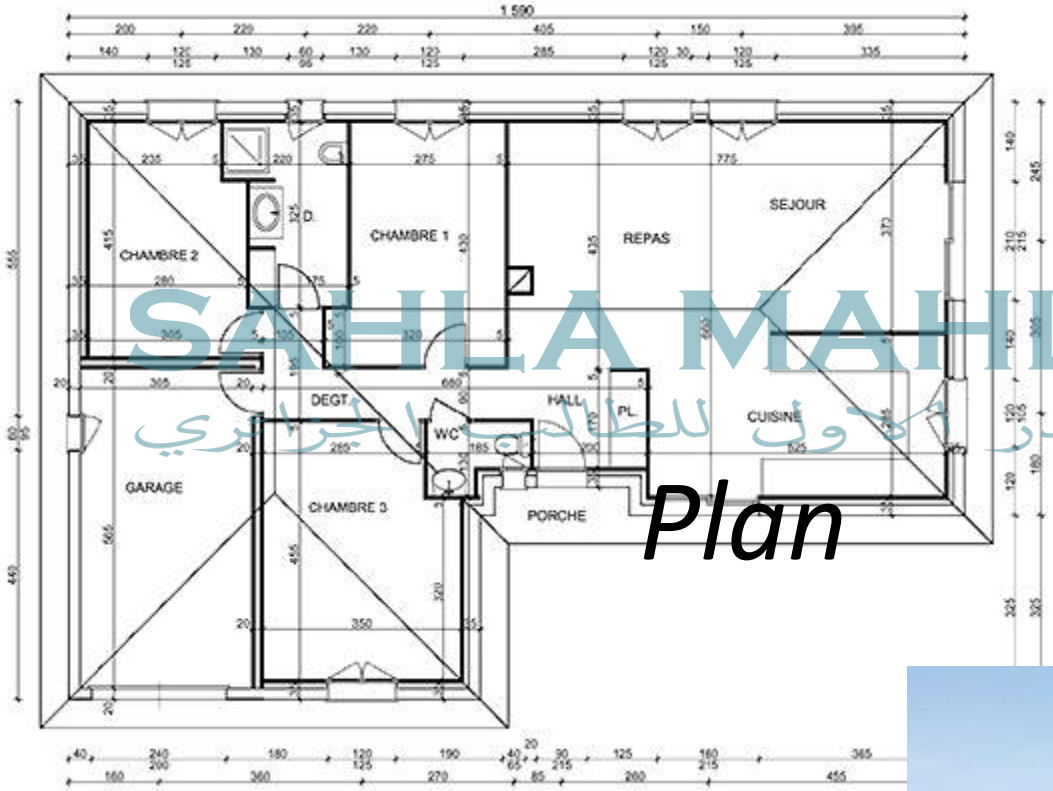


Classe

Objet ou instance

Objet ou instance

Objet ou instance



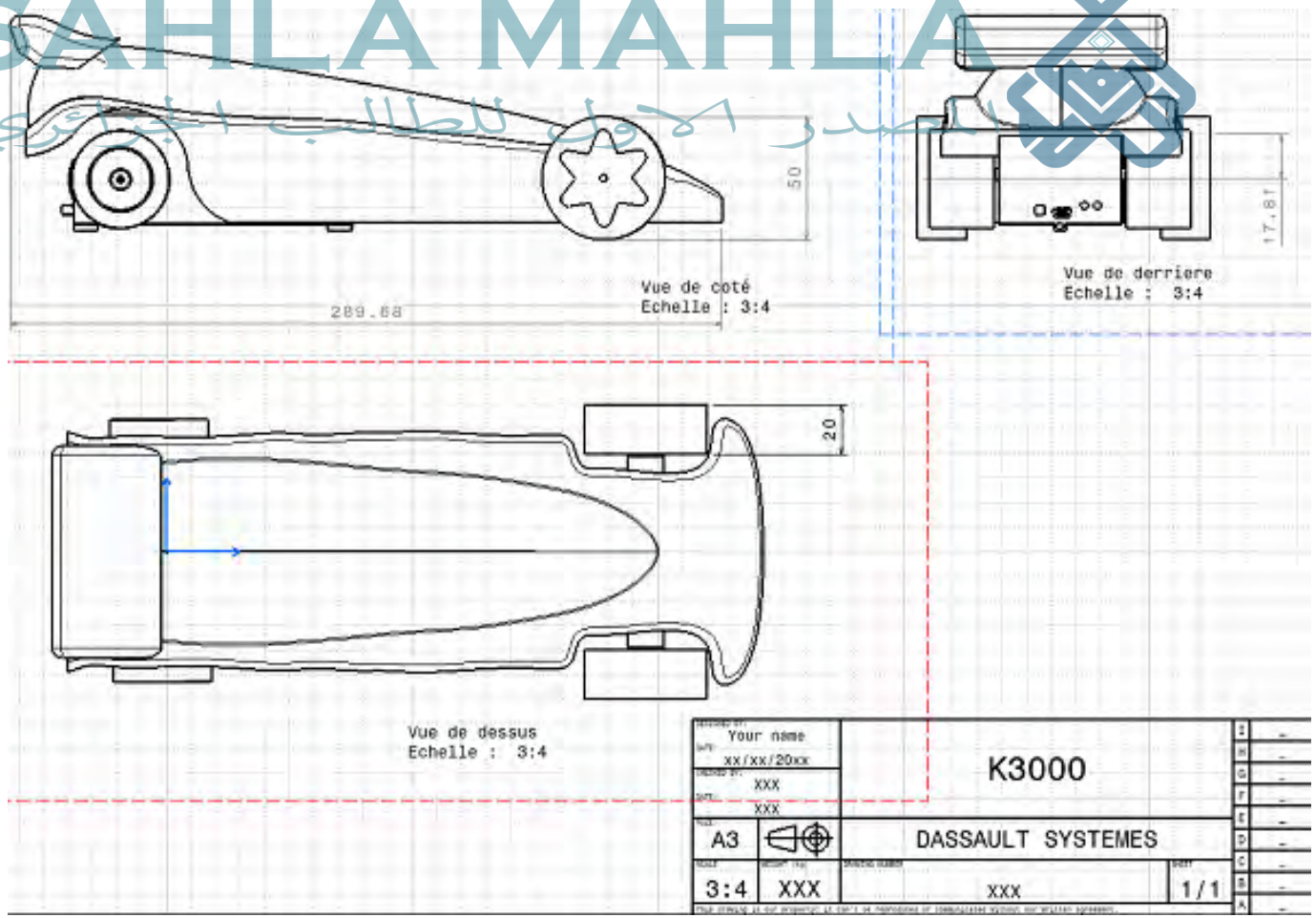
Plan

Réalisation (1)

Réalisation (2)

Réalisation (3)

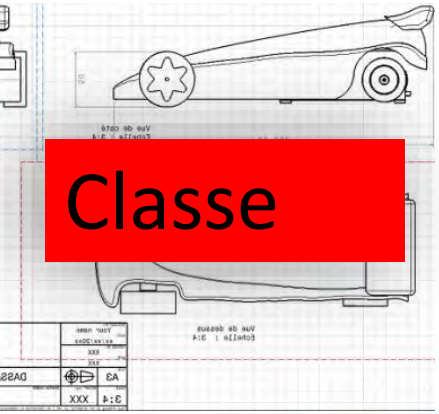
Plan de construction de la Toyota yaris 2016



Yaris 2

Objet ou instance

Yaris 1



Classe

Yaris 3

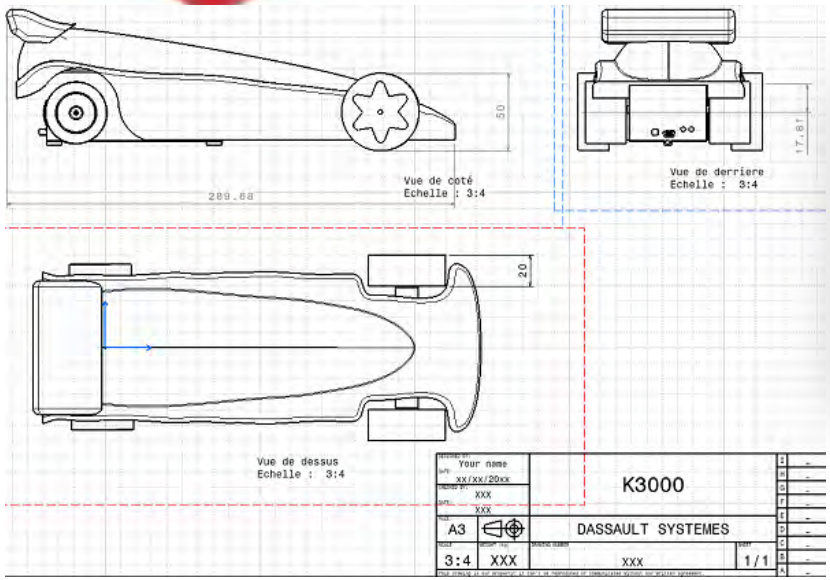
Objets ou instances

Yaris 4

Yaris 5

Yaris 6

Dans la terminologie POO, le moule est une classe. Les cakes sont des objets, produits par le moule.

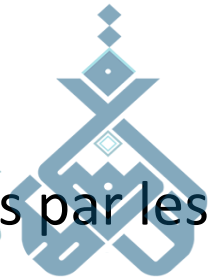


Dans la terminologie POO, ce plan est une classe. Les voitures sont des objets, fabriqués selon le plan.

✓ Tous les cakes sont caractérisés par les mêmes propriétés (attributs, champs ou caractéristiques):

- Poids (integer) ,
- Qualité(String),
- DuréeCuisson(integer).

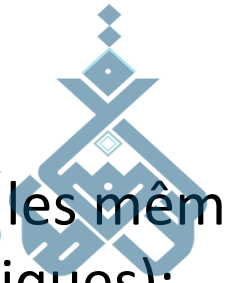
✓Toutefois, les valeurs de ces attributs peuvent différer d'un cake à un autre.



✓ Toutes les voitures yaris sont caractérisées par les mêmes propriétés (attributs, champs ou caractéristiques):

- Matricule (integer).
- Puissance(String).
- Couleur(String).
- Marque (String).
-

✓Toutefois, les valeurs de ces attributs peuvent différer d'une voiture à une autre.



✓ Toutes les maisons sont caractérisées par les mêmes propriétés (attributs, champs ou caractéristiques):

- Adresse.
- Superficie(String).
- propriétaire (String).
- Nombre de niveaux (integer).

✓Toutefois, les valeurs de ces attributs peuvent différer d'une maison à une autre.

✓Les mêmes traitements peuvent être appliqués à n'importe quel cake:

- congeler,
- décorer.
- cuire

✓Les mêmes traitements peuvent être appliqués à n'importe quel cake:

- congeler,
- décorer.
- cuire

✓Les mêmes traitements peuvent être appliqués à n'importe quelle voiture:

- démarrer,
- réparer,
- conduire,…..

✓Les mêmes traitements peuvent être appliqués à n'importe quelle maison:

- vendre,
- peindre,
- louer,…..

- ✓ La Programmation Orientée Objet (abrégée en POO) désigne une approche de résolution algorithmique ou programmation avec laquelle un logiciel est organisé sous forme d'une collection d'éléments (objets).

- ✓ Ces éléments sont représentés tels qu'ils sont dans le monde réel.

- ✓ Cette approche se base essentiellement sur deux éléments clés : *l'objet* et *la classe*.

Notion de classe

Une classe est la représentation de la structure d'une famille d'objets partageant des propriétés et méthodes communes. Elle permet la **déclaration** des attributs (propriétés ou attributs) ainsi que la **définition** de l'ensemble de méthodes (traitements).

- Une classe est considérée comme un ***moule*** de création d'objets, et un objet serait donc une instance (exemplaire) d'une classe.

Notion de L'objet

Un objet est une entité caractérisée par trois éléments fondamentaux :

1. Une identité : c'est un nom qu'on lui attribue,

Exemple: g1 et g2 sont des noms d'objets correspondant à deux gâteaux

2. Des propriétés : c'est l'état de l'objet défini par un ensemble de valeurs d'attributs,

Exemples: MaToyota: Matricule:111111
couleur:verte
puissance: 5 chevaux

Toyota de Mr Dupont:
Matricule:2222222
couleur:blanche
puissance: 5 chevaux

Des comportements

✓ C’est l’ensemble d’actions (opérations, méthodes) pouvant être réalisées sur l’ob et
permettant de traiter des données,

❑exemple : décorer, congeler, trouer, cuire pour un objet gâteau

❑ calcul de moyenne, le changement de section et l’affichage de des informations pour un objet étudiant .

✓ Cependant, les objets sont créés à partir d’**une classe**

La structure d’une classe en java

La définition d’une classe consiste à lui attribuer un nom, déclarer ses attributs et à implémenter (écrire le code) ses méthodes. Ainsi, le squelette d’une classe est composé de trois parties :

A. Entête de la classe

B. La déclaration des attributs

C. La définition des méthodes

A. **L'entête de la classe**

L'entête d'une classe est décrite par le mot réservé **class** suivi de nom de la classe ainsi que d'autres éléments (seront vus plus loin).

Conventions :

1. La première lettre de nom de classe est une majuscule (e.g. Point).

2. Si le nom de la classe est composé de plusieurs mots, la première lettre de chaque mot est en majuscule et le reste en minuscules (e.g. PointColoré)

La classe Voiture ressemble à ceci :

```
class Voiture {
// Déclaration des attributs
 ……..
// Déclaration des méthodes
  ……….
          }
```

La classe Maison ressemble à ceci :

```
class Maison{
    // Déclaration des attributs
     ……..
    // Déclaration des méthodes
      ……….
              }
```

Déclaration des attributs

- ✓ La déclaration des attributs permet de spécifier, le nom, le type et éventuellement une valeur initiale.

- ✓ Par convention, la déclaration des attributs est donnée au début de la classe.

- ✓ Le type d'un attribut peut être :
  - Primitif tel que *double, int, float, double, boolean*.
  - Nom d'une classe tel que Point, String, Etudiant.

Définition des méthodes

- ✓ La définition d'une méthode est composée de sa signature suivie de son corps. La signature comporte généralement le nom de la méthode, son type de retour et un ensemble de paramètres.

- ✓ La syntaxe utilisée pour définir le corps d'une méthode java est identique à celle utilisée pour les fonctions en C. Ainsi, nous retrouvons les mêmes :
  - Instructions conditionnelles, itératives, etc.
  - Opérateurs de comparaisons, logiques, etc.

La classe Point ressemble à ceci :


```
class Point {
 // déclaration des attributs
double x;
double y;
 // Déclaration des méthodes
 void deplace(double xx,
double yy){x+=xx;
            y+=yy;
             }
……
             }
```

La classe Etudiant ressemble à ceci :

```
class Etudiant{
 // déclaration des attributs
   int mat;
    String nom,prenom;
    ……
     ……..
    // Déclaration des méthodes
      ……….
             }
```

- ✓La définition d'une classe **consiste à créer un nouveau type**. Une fois créée, la classe devient un type et des objets peuvent être associés à ce type.

- ✓Une classe généralise la notion de type

- ✓et l'objet celle d'une variable

Exemples de déclarations

✓ Type primitif

```
int x,y;      // int est un type primitif
 float z;     // on peut déclarer autant de variables que l'on veut
```


✓ Type prédéfini (classe)

```
Point p1,p2

Etudiant et1,et2,et3,et4;

Voiture maToyota, maPolo;

Amphi amph1, amph2;

Pays  pay1, pay2,pay3;
```

Exemple 1

Définir une classe Point. Chaque point d'un plan est caractérisé par une abscisse et une ordonnée. (Traitement après)


```
class Point {
// déclaration des attributs
double   x, y;

// Définition des méthodes
après
}
```

Exemple 2

Définir la classe Segment.
Un segment est caractérisé par deux extrémités de type Point. (Traitement après).


```
class Segment {
// Attributs
double  xe1, ye1,
xe2,ye2 ;
double épaisseur;
// Définition des
méthodes après
}
```

- Attention, ce n'est pas intéressant
- Profiter du type Point.

```
class  Segment {
// attributs
Point  p1, p2 ;
double
épaisseur;

// Définition des
méthodes après
}
```



```
class  Segment {
// attributs
double xe1, ye1,
xe2,ye2 ;
double épaisseur;

// Définition des
méthodes après
}
```

Les méthodes de traitements

Les méthodes de traitement correspondent aux actions possibles pouvant être invoquées par des objets de la classe telles que déplacer un point, afficher les coordonnées d'un point.

```
class Point {
// Déclaration des attributs
      double x,y ;

// Définition des méthodes
 void déplacer( double a, double b) {x+=a ;y+=b ;  }

 void afficher(){
 System.out.println (" l'abscisse =" +x+ "ordonnée = "+y );}

}
```

```
class Etudiant {
// Déclaration des attributs
……….
// Définition des méthodes
String getNom() {return nom ;  }
Void setNom(String n){nom=n;}

 void afficher(){
…….

}

}
```

2. La méthode d'exécution: main

Une classe définie par ses attributs et ses méthodes n'est pas exécutable même si elle est compilée sans erreur, Pour exécuter une classe, les concepteurs java ont choisi pour cela de particulariser une méthode, appelée la méthode ***main***. La méthode ***main*** a la signature suivante :

public static void main(String[] args).

Remarque :

1. Toute classe possédant la méthode main est appelée classe exécutable.
2. Dans une application java il suffit d'avoir une seule classe exécutable.
3. Les termes static et public seront définis plus loin.

Exemple d'une classe exécutable


```
class Application
{
public static void main(String[] ars)
{
System.out.println("On dit que ISIL A 2017 est une bonne section!!!" );
System.out.println(" Prouvez le moi!! " );
}
}
```

Remarque: De préférence mettre ma méthode main dans une classe à part, elle est considérée la classe qui lance l'application ou la classe de test.

Notion de constructeur

A la création d'un objet, ses attributs **sont initialisés** par une méthode particulière appelée, **constructeur**. Un constructeur est une méthode qui :

- n'a pas de type de retour,

- porte *le même nom* que la classe,

- ne peut être appelé en dehors de la création d'un objet,

- il permet d'initialiser les attributs de l'objet à créer.

Si une classe ne définit pas de constructeur (explicitement), un constructeur par défaut est créé automatiquement (implicitement) par le compilateur.
Le constructeur implicite permet d'initialiser les attributs par des valeurs par défaut.

Par exemple, les numériques sont initialisés à zéro, le booléen à faux, une référence d'objet à null.

En revanche, le constructeur explicite initialise les attributs par des valeurs passées en paramètre.

Classe point sans constructeur explicite


```
class Point {
        // Déclaration des attributs
          double x,y ;

        // Définition des méthodes de traitement
        void déplacer( double a, double b) {x+=a ;y+=b ;  }

        void afficher(){ System.out.println (" l'abscisse ="
+x+ "ordonnée = "+y );}
}
```

Classe Point avec un constructeur explicite


```
class Point {
        // Déclaration des attributs
        double x,y ;
         // Constructeur
         Point(double a, double b){x=a ; y=b ;}

       // Définition des méthodes de traitement
       void déplacer( double a, double b) {x+=a ;y+=b ;  }

       void afficher(){ System.out.println (" l'abscisse =" +x+
"ordonnée = "+y );}
}
```

- Si une classe ne définit pas de constructeur (explicitement), un constructeur par défaut est créé automatiquement (implicitement) par le compilateur. Ce constructeur est sans paramètres.

  Exemple: On le voit pas dans la classe, exemple la classe Point bleue.

- Le constructeur implicite permet d'initialiser les attributs par des valeurs par défaut. Par exemple, les numériques sont initialisés à zéro, le booléen à faux, une référence d'objet à null.

  Les objets de la classe Point bleue sont créés avec des coordonnées nulles.

- Le constructeur explicite initialise les attributs par des valeurs passées en paramètre.

Exemple : Les objets de la classe Point verte sont créés avec des coordonnées choisies par l'utilisateurs.

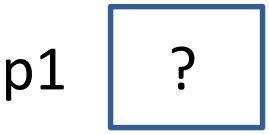
Remarque: Dès qu'un constructeur avec paramètre est mis dans la classe, le constructeur sans paramètres ne peut être utilisé sauf si on rajoute un constructeur explicite sans paramètre à la classe.

Instanciation d'une classe ou création d'objets

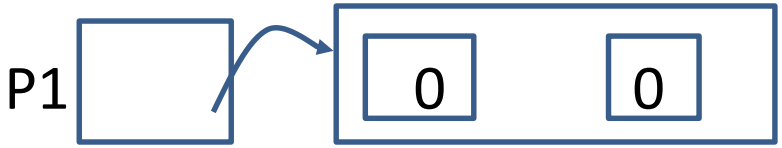
Pour créer une instance (ou objet) d'une classe, on utilise l'opérateur ***new*** suivi d'un constructeur de la classe. Bien évidemment, l'objet à créer doit être préalablement déclaré avec le type de la classe adéquate.

Exemple 1 (classe Point sans constructeur explicite, la bleu)

1. Point p1 ; // déclaration de l'objet



2. p1 = new Point ( ) ; //Création de l'objet



Instanciation d'une classe ou création d'objets

Exemple 2 (classe Point avec constructeur explicite, la verte)

1. Point p2,p3,p4,p5 ; // déclaration de l'objet

2. p2 = new Point (3, 4) ; //Création de l'objet p2
3. p3 = new Point (-3, -4) ; //Création de l'objet p3
4. p4 = new Point (-3, -4) ; //Création de l'objet p4

Questions: - représentez en mémoire ces objets.

- p5 a-il-été créé?
- P3 et p4 correspondent ils à un même objet?

Accès aux membres d'un objet

L'accès à un membre (attribut ou méthode) d'un objet donné se fait à l'aide de la notation à point. Cette notation possède deux parties séparées par un point: à gauche du point, on trouve la référence de l'objet et à sa droite le membre:

NomDeObjet.NomAttribut

ou

NomDeObjet.NomMethode( )

Exemples

Point p1,p2 ; Voiture v1,v2 ; Oiseau o1 ; Femme f1 ; Homme h1 ;

p1.affiche() ; p2.deplace(3,4) ; bool= p1.egal(p2) ;

v1.affiche() ; bool= v2.plusPuissante(v1) ; v2.demarrer() ;

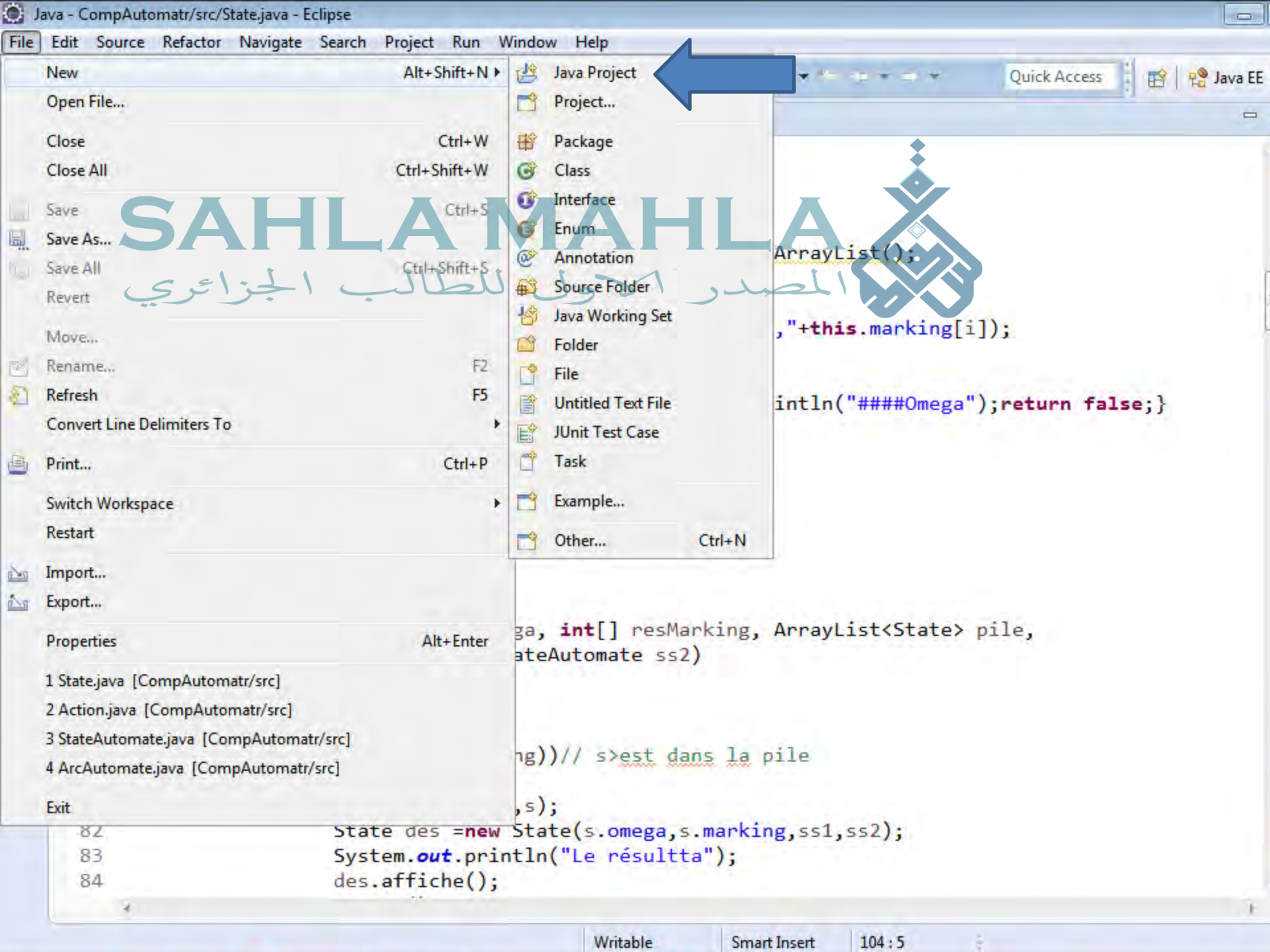
o1.affiche() ; o1.voler() ; esp= o1.espece() ;

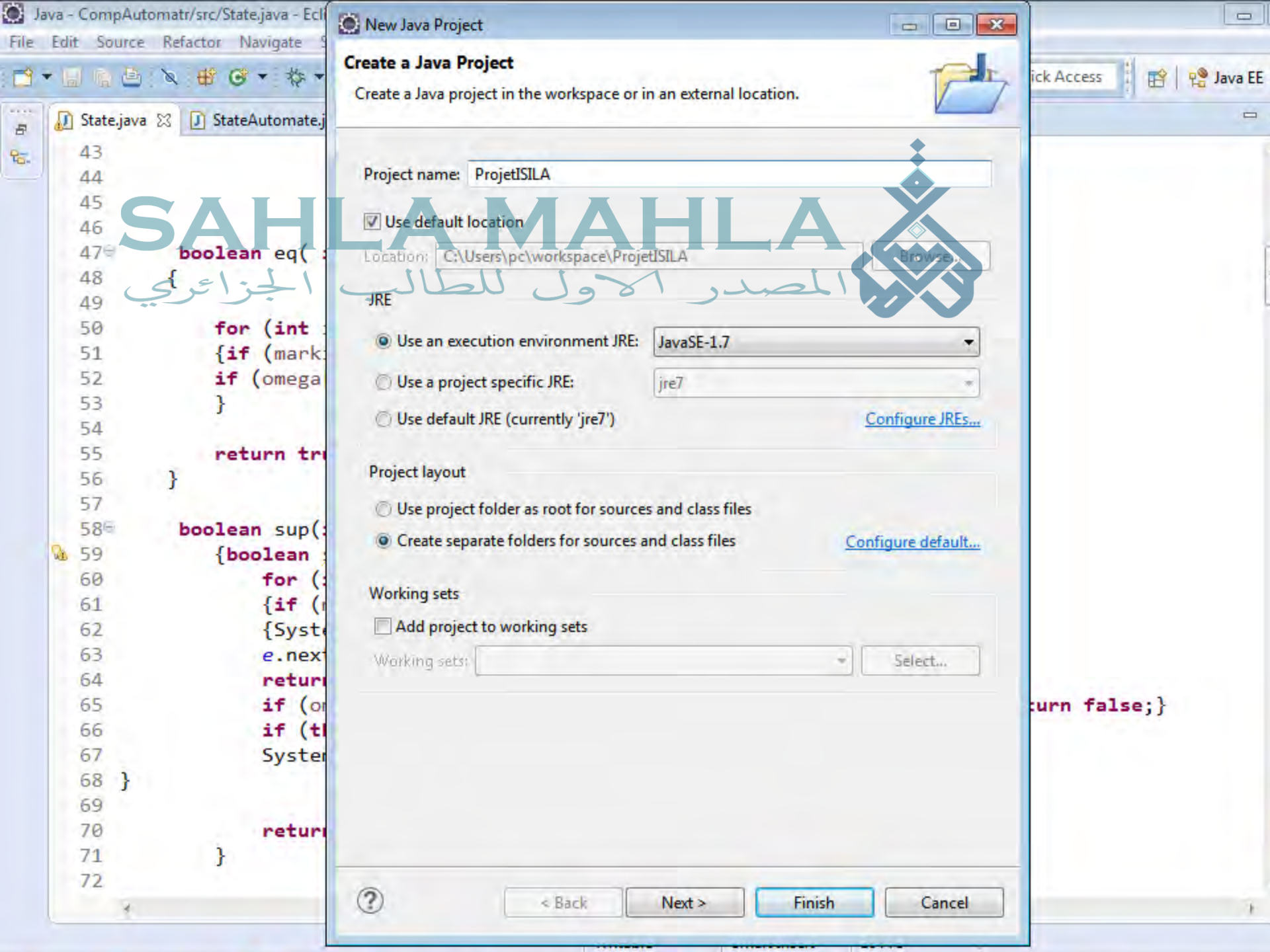
f1.affiche() ; h1.marie(f1) ;
p5.affiche(); ?????????

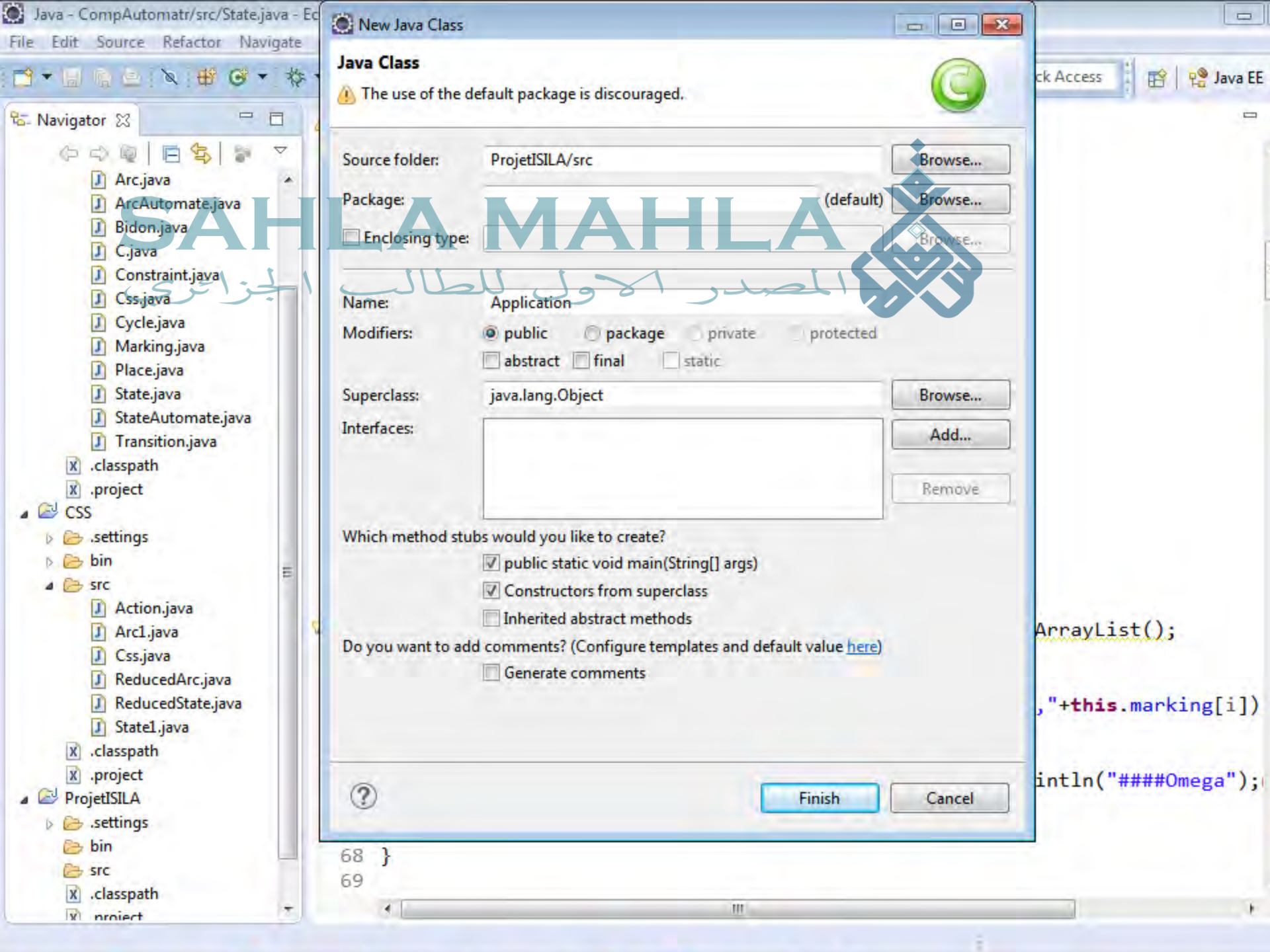
Exemples:

```
Class PointApplication {
public static void main(String[] ars) {
    Point p1 = new Point (12,23) ;// classe Point verte
                System.out.println(p1) ; p1.affiche();
    p1. déplacer(15,-4) ;
          p1.afficher() ;
 Point p2 = new Point (2,3) ;// classe Point verte

          p2. déplacer(15,-4) ;
          p2.afficher() ;
Point p3 = new Point();
Point p4 ;
p4.déplacer(2,6);   // sont elles correctes?
          }
     }
```

Donner les messages affichés suite à l'exécution de cette classe.




Java - CompAutomatr/src/State.java - Ecli
File Edit Source Refactor Navigate
State.java
StateAutomate.j
43
44
45
46
47 boolean eq(:
48 {
49
50 for (int
51 {if (mark:
52 if (omega
53 }
54
55 return tru
56 }
57
58 boolean sup(:
59 {boolean
60 for (:
61 {if (
62 {Syste
63 e.next
64 returr
65 if (or
66 if (t
67 Syster
68 }
69
70 returr
71 }
72
New Java Project
Create a Java Project
Create a Java project in the workspace or in an external location.
Project name: ProjetISILA
Use default location
Location: C:\Users\pc\workspace\ProjetISILA
Browse...
JRE
Use an execution environment JRE: JavaSE-1.7
Use a project specific JRE: jre7
Use default JRE (currently 'jre7')
Configure JREs...
Project layout
Use project folder as root for sources and class files
Create separate folders for sources and class files
Configure default...
Working sets
Add project to working sets
Working sets:
Select...
< Back
Next >
Finish
Cancel
SAHLA MAHLA
المصدر الاول للطالب الجزائري
urn false;}


Java - CompAutomatr/src/State.java - Ec
File Edit Source Refactor Navigate
Navigator
Arc.java
ArcAutomate.java
Bidon.java
C.java
Constraint.java
Css.java
Cycle.java
Marking.java
Place.java
State.java
StateAutomate.java
Transition.java
.classpath
.project
CSS
.settings
bin
src
Action.java
Arc1.java
Css.java
ReducedArc.java
ReducedState.java
State1.java
.classpath
.project
ProjetISILA
.settings
bin
src
.classpath
.project
New Java Class
Java Class
The use of the default package is discouraged.
Source folder: ProjetISILA/src
Browse...
Package: (default)
Browse...
Enclosing type:
Browse...
Name: Application
Modifiers: public package private protected
abstract final static
Superclass: java.lang.Object
Browse...
Interfaces:
Add...
Remove
Which method stubs would you like to create?
public static void main(String[] args)
Constructors from superclass
Inherited abstract methods
Do you want to add comments? (Configure templates and default value here)
Generate comments
Finish
Cancel
ArrayList();
","+this.marking[i])
intln("####Omega");
68 }
69

Java - ProjetISILA/src/Application.java - Eclipse

File Edit Source Refactor Navigate Search Project Run Window Help

Quick Access | Java EE

Navigator

- Arc.java
- ArcAutomate.java
- Bidon.java
- C.java
- Constraint.java
- Css.java
- Cycle.java
- Marking.java
- Place.java
- State.java
- StateAutomate.java
- Transition.java
- .classpath
- .project
- CSS
  - .settings
  - bin
  - src
    - Action.java
    - Arc1.java
    - Css.java
    - ReducedArc.java
    - ReducedState.java
    - State1.java
  - .classpath
  - .project
- ProjetISILA
  - .settings
  - bin
  - src
    - Application.java
  - .classpath

State.java | StateAutomate.java | ArcAutomate.java | Action.java | *Application.java

```
1
2  public class Application {
3
4
5      public static void main(String[] args) {
6          System.out.println("Hello!!!");
7
8      }
9
10 }
11
```


Writable | Smart Insert | 6 : 37

Testons nos connaissances

1. Mettez *C* pour classe ou *O* pour objet:

moule, instance, plan de construction, description, a un état et des comportements, déclaration des attributs et définition des méthodes, type, ressemble à une variable, création, vie, destruction.

2. Répondre par oui ou non.

-une méthode correspond à une fonction dans C,

- toute classe est exécutable.
- dans un projet on peut créer une infinité d'objets
- dans un projet on peut avoir une infinité de classes
- un constructeur a toujours void comme type de retour
- toute classe doit avoir un constructeur explicite
- le compilateur rajoute un constructeur avec ou sans paramètres
- un constructeur explicite peut être défini sans paramètres
- le new permet de créer un seul objet à la fois
- la méthode main est obligatoire dans toute classe

Détecter les erreurs et dérouler


```
class Test
Int x , y =3;
x= 2;
 void Test(){}
int  f(){ System.out.println(‘’BienVenue ‘’+y);}

void g(){
Test t = new Test(); t.y++;
t.f();}


}
```

Gestion des objets

La déclaration *NomcClasse nomObjet permet :*

- ✓ De déclarer un objet sans le créer.

- ✓ D'allouer une case mémoire destinée à recevoir la référence de l'objet (l'adresse mémoire de l'espace qui sera alloué à ses attributs).

Exemple:
Point p1, p2;

Gestion des objets

new NomClasse() ou
new NomClasse(par$_1$,…,par$_n$)

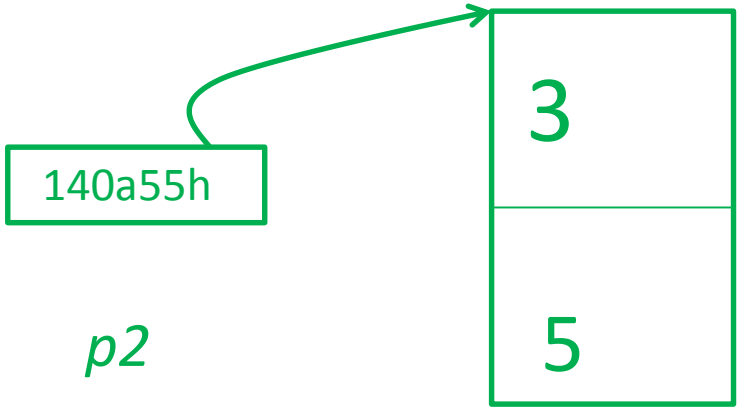
1. Ces instructions permettent d'allouer un espace mémoire à l'objet créé.

2. Les attributs de l'objet créé sont initialisés soit par des valeurs par défaut (constructeur sans paramètres) ou des valeurs passées comme paramètres (constructeur avec paramètres).

Gestion des objets

Exemple (classe Point verte):

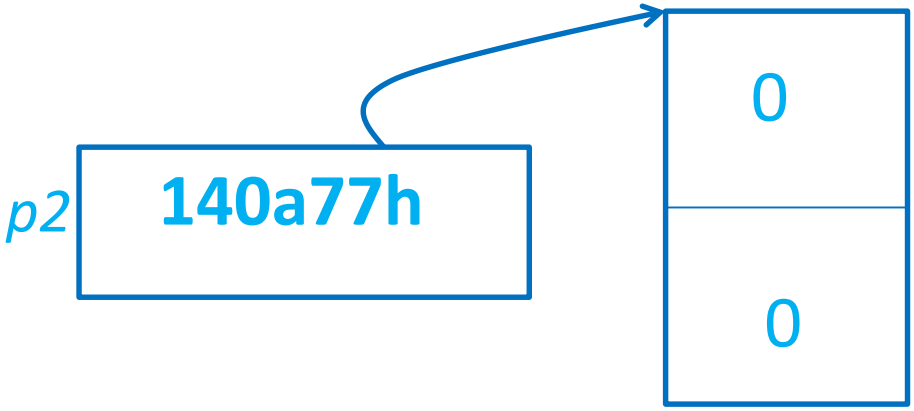
Point p2=new Point(3,5);



Gestion des objets

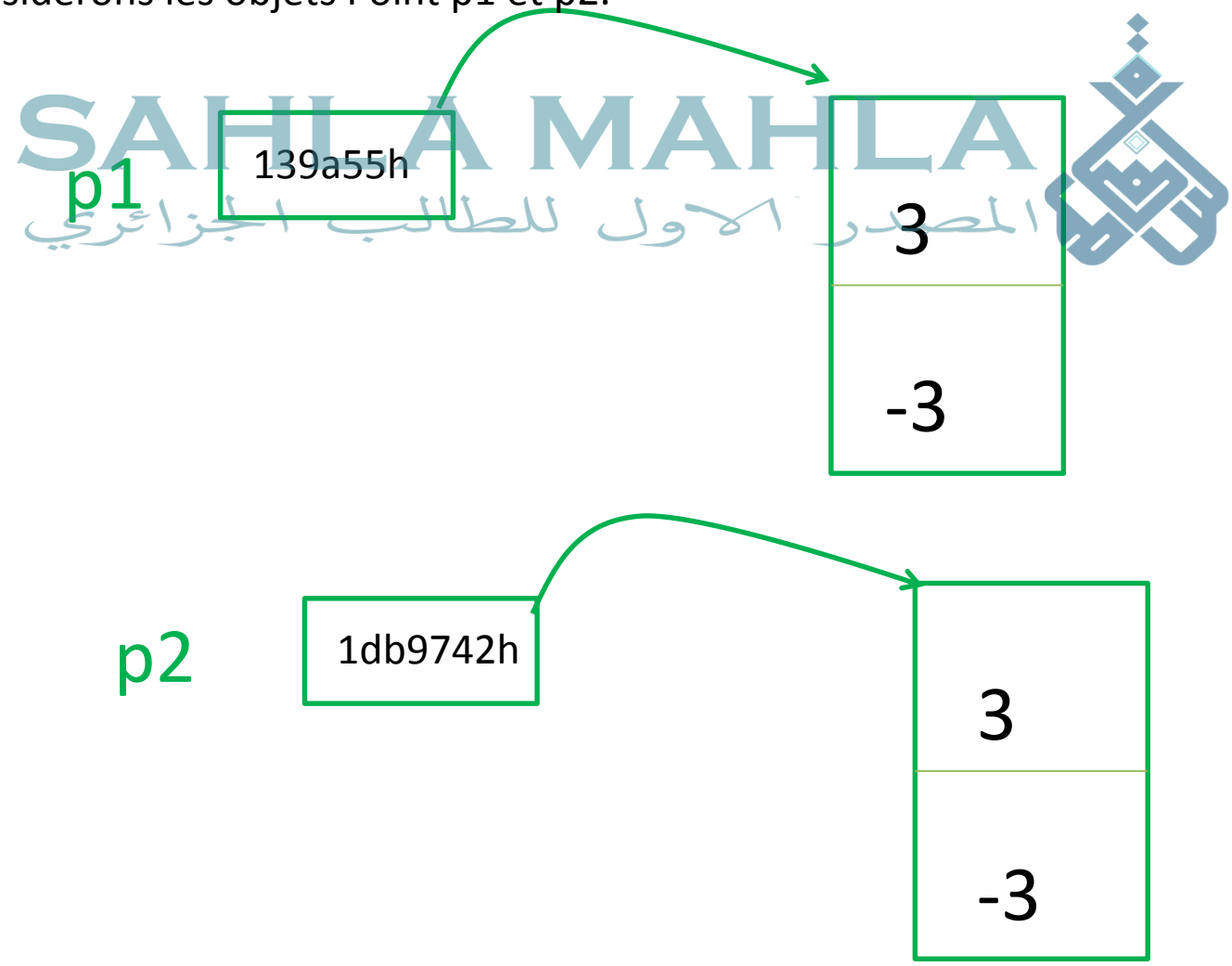
Exemple (classe Point bleue):

Point p2=new Point();



Affecter un objet à un autre

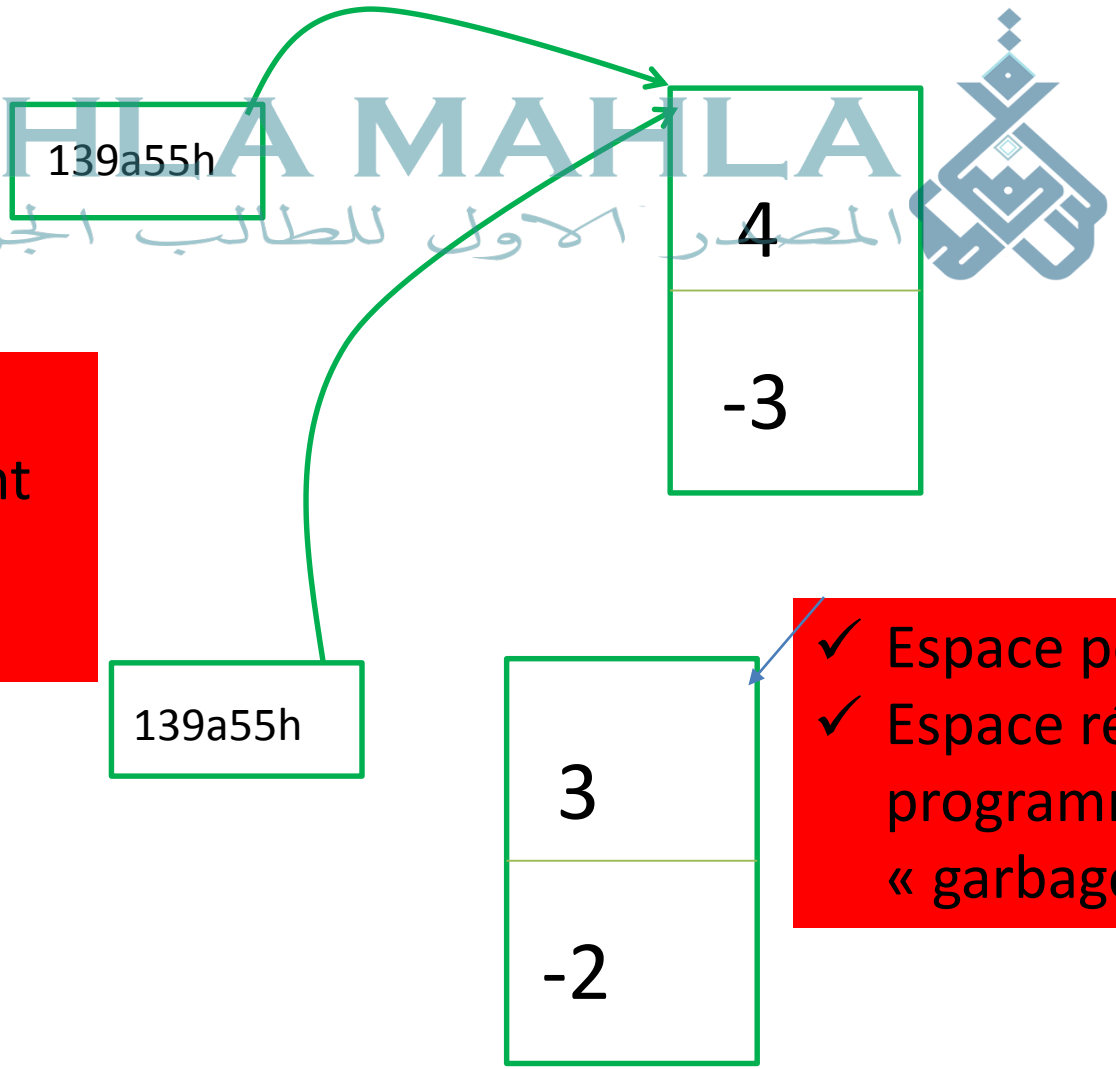
Considérons les objets Point p1 et p2.



Dérouler p1.x++; p2.y++; p1.affiche(); p2.affiche();

Affecter un objet à un autre

1. p2=p1

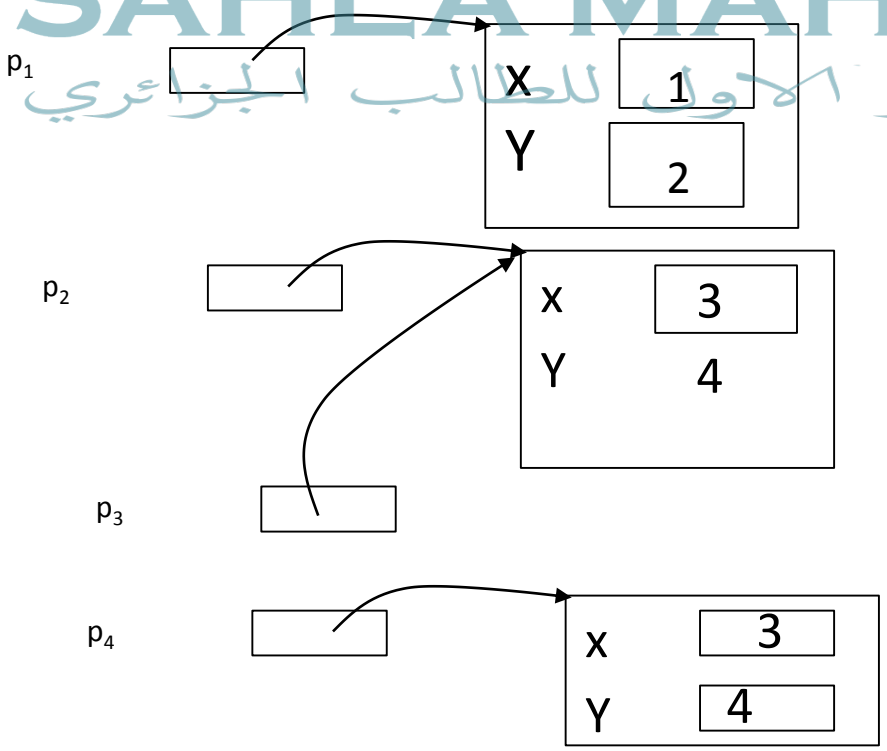


p1 et p2 représentent le même objet.

- ✓ Espace perdu
- ✓ Espace récupéré par un programme spécial « garbage collector »

2. Dérouler p1.x++; p2.y++; p1.affiche(); p2.affiche();

Tas des objets



La surcharge des méthodes

La surcharge (overloading en anglais) est également appelée surdéfinition. C'est l'un des points forts des langages orienté objets. Elle permet de définir dans une même classe plusieurs fonctions avec le même nom. La différence entre les méthodes surchargées réside dans leurs signatures : le nombre et/ ou type de paramètres diffèrent. C'est le compilateur qui sélectionne la méthode appelée à un niveau donné. La surcharge concerne aussi bien les constructeurs que les méthodes de traitements.

```
class Point {
double x,y ;
// 1er  constructeur
Point(){}
// 2 eme constructeur
Point (double x, double y ){ this.x= x; this.y = y}

// Première version
public void deplacer( double xv, double yv)  {
x+=xv ;y+=yv ;}

// Deuxième version
public void deplacer( double d)  {x+=d ; y+=d ;}

// Troisième version
public void deplacer(Point p){
x+= p.x ;y+=p.y ;}     }
```

Remarque : Attention au problème d' ambigüité

Exemple 1
void deplacer(double x,double y) ;
int deplacer(double xx, double yy) ;

Exemple 2
Int f(int , double), void f(double, int);
si on appelle avec f(2, 4.5)
il invoque la première fonction f

si on appelle avec f(3.5, 5)
il invoque la seconde fonction f

Mais si on invoque avec f(4,7)
il y a un problème ambigüité

L'instance courante : this

Dans le corps d'une méthode d'un objet, le mot clé **this** désigne l'instance sur laquelle est invoquée la méthode. Ainsi **this** est l'objet en cours d'utilisation. En d'autres termes, **this** représente l'objet auquel s'applique la méthode.

Exemple:

Point p1,p2,p3;

p1=new Point(3,4);

p2=new Point(-3,-4);

p3=new Point(0,4);

Invocation	Message affiché	L'instance courante lors de l'exécution de la méthode Affiche()
p1.affiche()	x=3 et y=4	p1 représente this
p2.affiche()	x=-3 et y=-4	p2 représente this
p3.affiche()	x=0 et y=4	p3 représente this
Affiche()	erreur	This n'est pas défini

Dérouler et interpréter

```
class  Test{
void  identifier(){
System.out.println(this); // affiche  @ de l'objet
}
public static void main(String args[])
{// déclaration de 3 objets XXX
Test x1,x2,x3;

// création des objets
x1=new Test(); x2=new Test();  x3=new Test();

// 3 invocations de la méthode   identifier;
x1. identifier();   x2. identifier();  x3. identifier();
}
 }
```

L'utilisation de l'instance courante this

Le mot clé this est utilisé :

1. Pour distinguer entre un nom d'attribut et un nom de paramètre dans le cas où ils sont identiques

```
class Point{
  double x, y;
  // constructeur

Point(double x , double y){
  x = x; y = y;}
.....}
```

Le compilateur ne déclare pas d'erreurs Les valeurs des attributs ne changent pas

```
class Point{
  double x, y;
  // constructeur
Point(double a , double b){
  x = a; y = b;}.....}
```


```
class Point{
  double x, y;
  // constructeur
Point(double x , double y){
  this.x = x; this.y = y;}
.....}
```

L'attribut x de l'instance courante

Le paramètre d'appel x

Instance courante

2. La méthode a besoin de manipuler la référence de l'objet courant.

Exemple:

Si on veut écrire une méthode qui compare deux points et rend celui qui a l'abscisse minimale.

```
class Point{
double x; double y;
Point(double x, double y){
this.x = x;
this.y = y;}

Point   minAbs(Point p ){
if ( x < p.x ) return this ;
else return p ;}
```

Il s'agit d'un objet Point

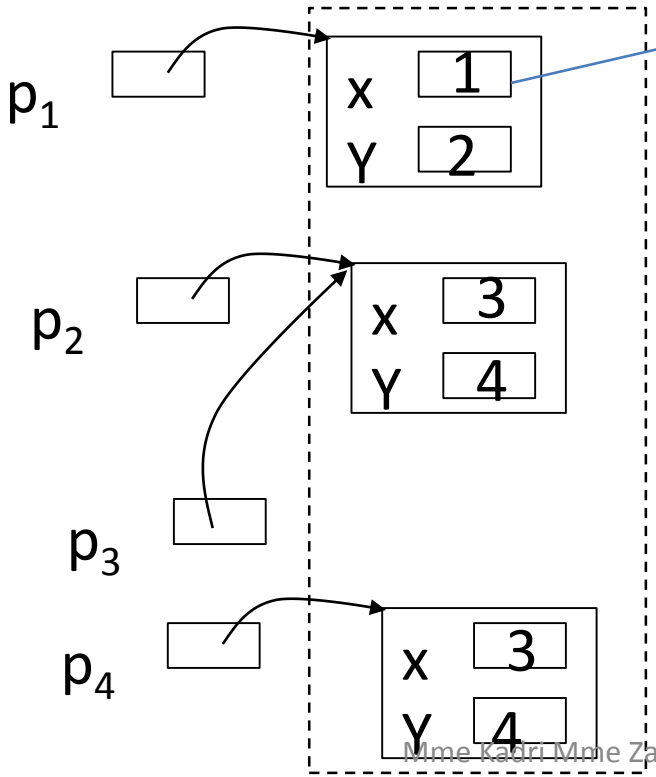
2. Dérouler Point p1 =new Point (3,4); Point p2= new Point(4,7);

p1.minAbs(p2).affiche(); p2.minAbs(p1).affiche();

Attributs d'instance

✓ Les attributs des objets que nous venons de voir jusqu'à présent s'appellent des attributs d'instance.

✓ Un attribut d'instance est propre à chaque objet



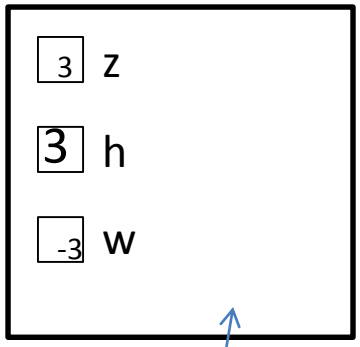
✓ x et y sont des attributs d'instance.

✓ Espace local à chaque objet.

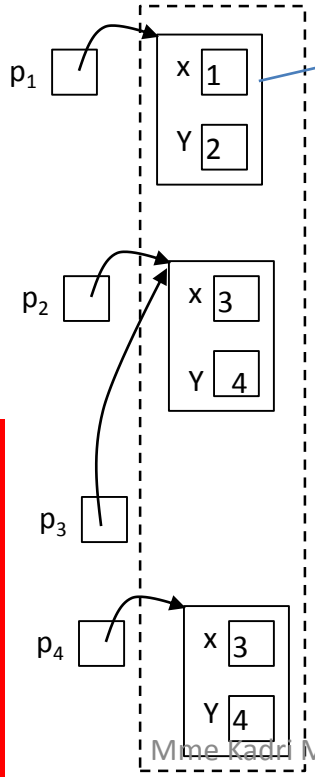
✓ Aucun partage entre les objets.

Attributs de classe

✓Un attribut de classe est commun à tous les objets de la classe. Il est introduit par le mot clé **static**.
✓Un seul espace est alloué à un attribut de classe.



✓ z, h, w sont des attributs de classe.
✓ Espace partagé par tous les objets.



✓ x et y sont des attributs d'instance.
✓ Espace local à chaque objet.
✓Aucun partage entre les objets.

Application (attribut d'instance et de classe)


```
class Point{
double x,y;
static  int nbPoint =0;
Point (double x, double y)
{this.x=x; this.y=y; nbPoint++;}
void affiche(){
System.out.println( ”x=‘’+x+’’ y=‘’+y+ ‘’le nombre de points = ‘’+ nbPoint)}
}
public static void main(String[]args )
{ point p1,p2,p3;
p1=new Point(1,2); x2=new Point(3,4);  x3=new Point(5,6);
p1.affiche(); p2.affiche();p3.affiche();
System.out.println(nbPoint);
System.out.println(x, y);// erreur ????
}}
```

Attribut constant (présentation informelle)

On veut créer une classe Segment comme suit:

- ✓ Chaque segment est caractérisé par deux extrémités.
- ✓ Une épaisseur.
- ✓ Et une couleur **verte** qui ne change jamais.

Première solution

```
class Segment{
Point e1,e2;
double epaisseur;
String couleur=« Verte »;
Segment(Point e1,Point e2, double
epaisseur)
{this.e1=e1;
this.e2=e2;this.epaisseur=epaisseur;}

void setCouleur(String c){ couleur=c;}

}
```

OU

```
class Segment{
Point e1,e2;
String couleur;
double epaisseur;
Segment(Point e1,Point e2 ,double
epaisseur)
{this.e1=e1;
this.e2=e2;couleur=« verte »;
this.epaisseur=epaisseur;}

void setCouleur(String c){
couleur=c;}}
```


Il faut un moyen pour interdire les modifications

Attention

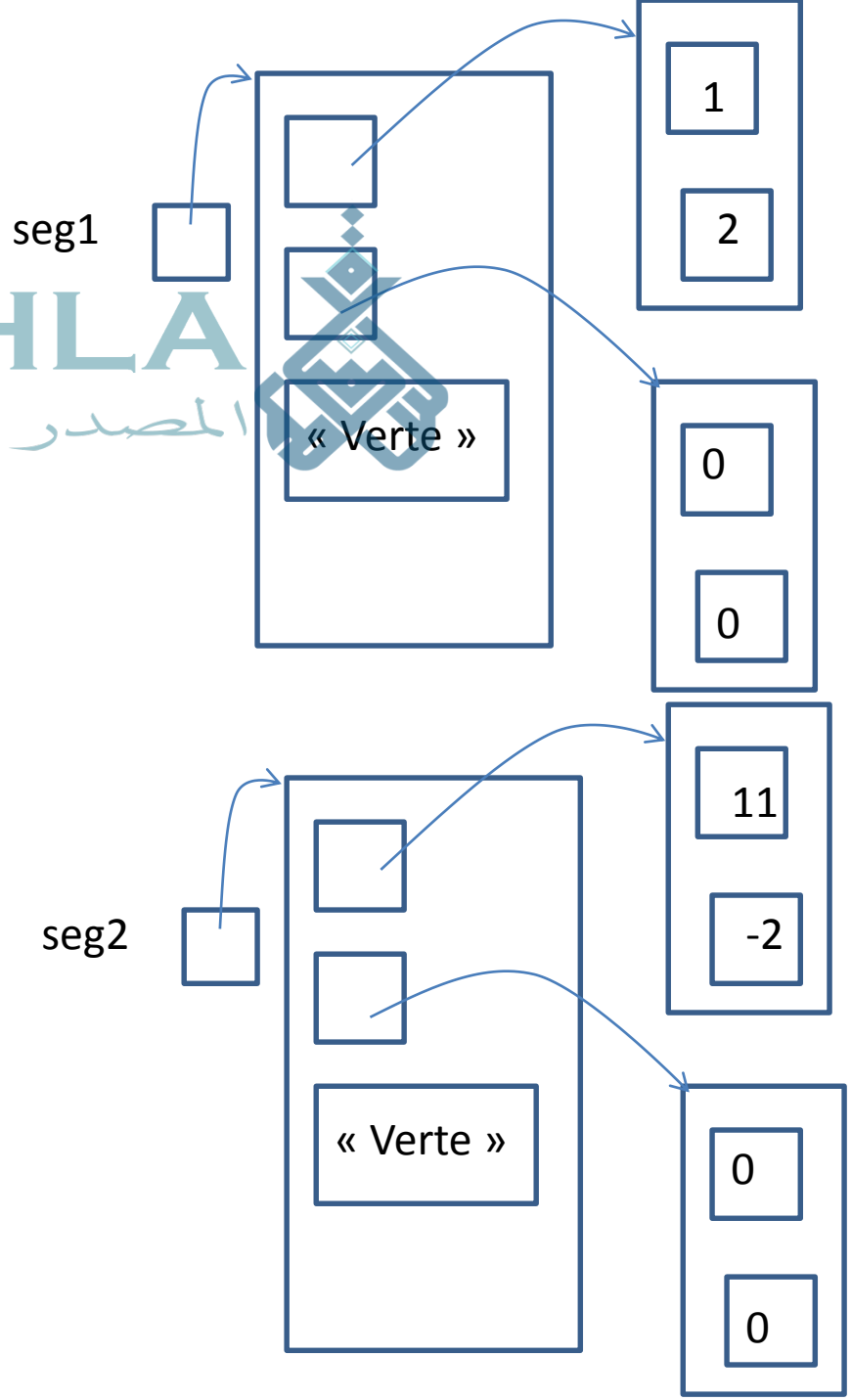
```
Class Application
{public static void main(String args[])
{Segment seg1;
seg1=new Segment(new Point(1,2),new Point(3,6));
Seg1.setCouleur(« Bleu»);
}}
```

Le mot clé final

Le mot clé final précède des attributs pour les rendre immuables ou constants.

Deuxième solution

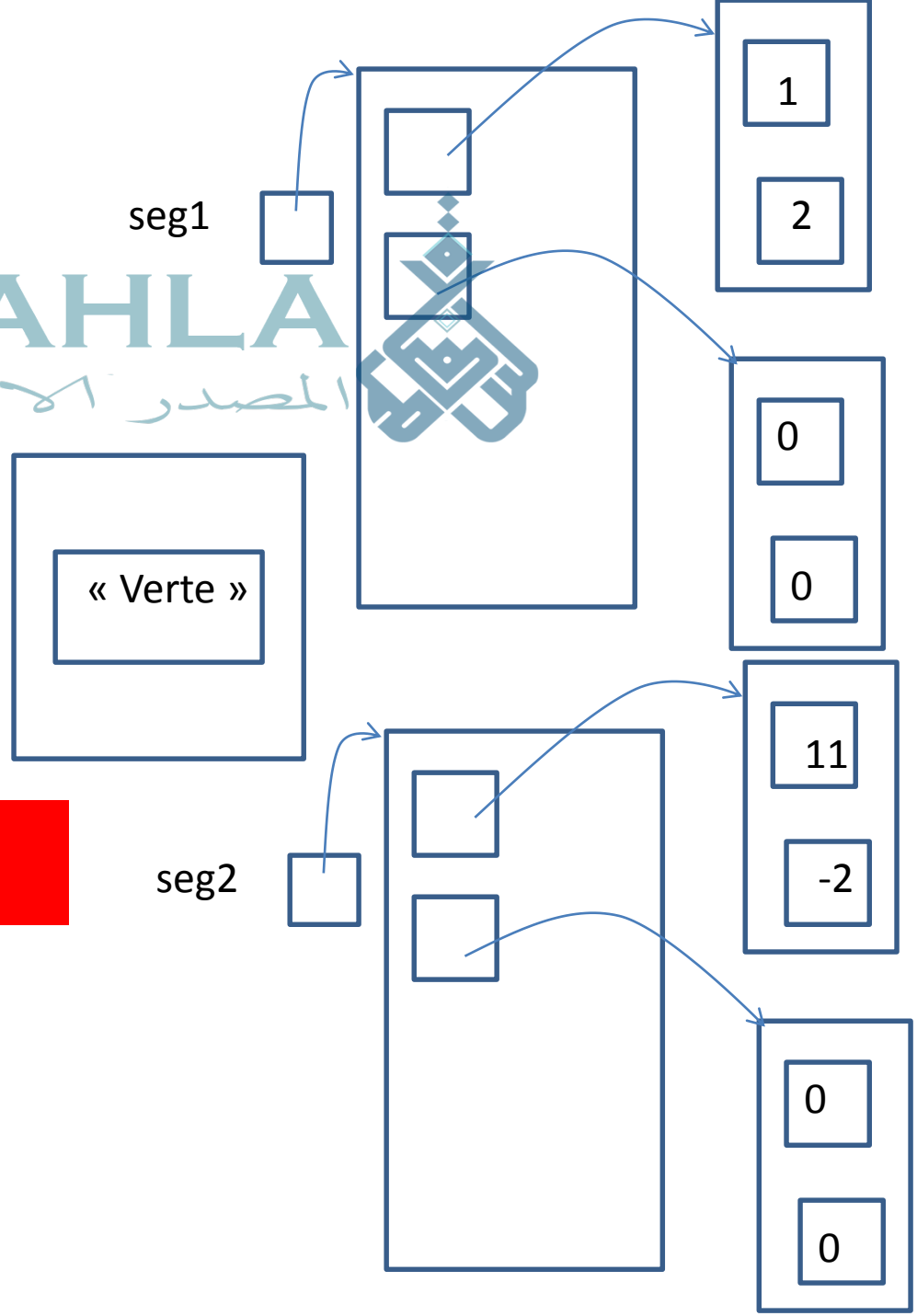
```
class Segment{
Point e1,e2;
final String couleur=« Verte »;
Segment(Point e1,Point e2)
{this.e1=e1; this.e2=e2;}

//interdite
//void setCouleur(String c){ couleur=c;}

}
```

✓Perte d'espace mémoire
✓ Optez plutôt pour un attribut couleur statique, partagé par tous les objets.

Troisième solution

```
class Segment{
Point e1,e2;
final static String
couleur=« Verte »;
Segment(Point e1,Point e2)
{this.e1=e1; this.e2=e2;}

//interdite
//void setCouleur(String c){ couleur=c;}

}
```

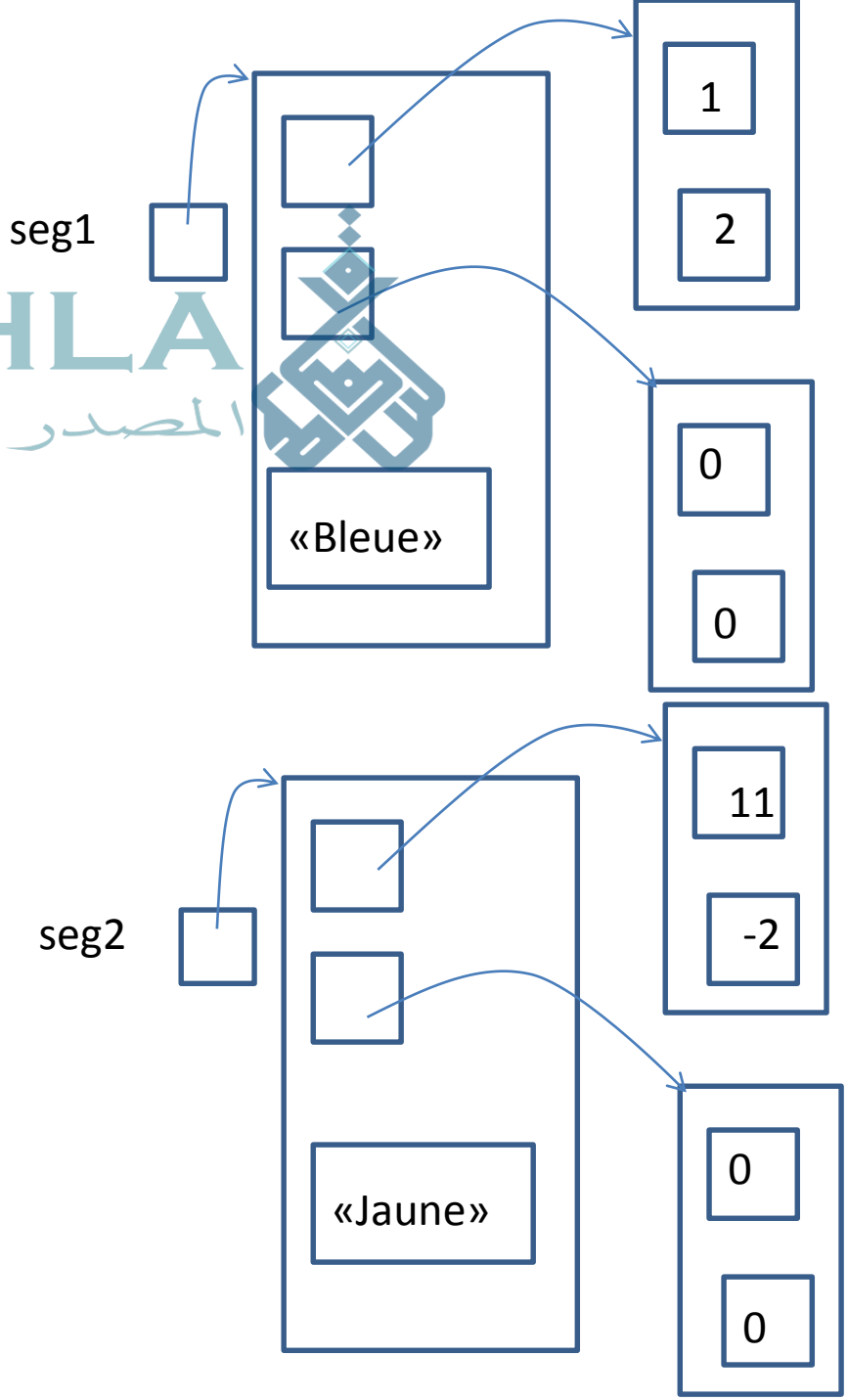
Espace partagé par tous les objets.

Attribut constant

Redéfinir la classe Segment de telle façon à ce que chaque segment ait une couleur donnée qui ne change pas.

```
class Segment{
Point e1,e2;
final String couleur;
Segment(Point e1,Point e2, String c)
{this.e1=e1; this.e2=e2;couleur=c;}

//interdite
//void setCouleur(String c){ couleur=c;}

}
```

- ✓ Constante non spécifiée dans la classe.
- ✓ Elle doit être passée en paramètre dans le constructeur.
- ✓Constante connue à l'exécution.



Types de constantes

✓ Connues à la compilation.

✓ Valeurs connues avant d'exécuter la classe.

✓ Exemple:

final static String couleur=« verte»

✓ Connues à l'exécution.

✓ Les affectations sont faites dans le constructeur. Elles sont obligatoires.

✓ Exemple:

final String couleur;

Segment(Point a,Point b,String c){e1=a; e2=b; couleur=c;}

Attributs constants(présentation formelle)

✓ Un attribut constant est une donnée qui une fois initialisée ne peut plus être modifié. Java utilise le mot clé **final** pour définir des constantes.

✓Une constante peut être initialisée de deux manières :

a. A la déclaration, cette valeur sera la même pour toute instance créée. La constante est connue à la compilation.

b. A la création d'objets (dans le constructeur), dans ce cas, chaque objet pourra avoir sa propre valeur constante. La constante ne sera connue qu'à l'exécution.

Remarque: si un attribut est constant et statique en même temps, il doit être obligatoirement initialisé à la déclaration vu qu'il n'est lié à aucune instance.

Objet constant

✓ Quand l'attribut constant est une référence d'un objet donné, cette référence sera liée définitivement à cet objet et ne peut référencer un autre objet.

✓ Cependant, l'objet référencé peut subir des *modifications*.

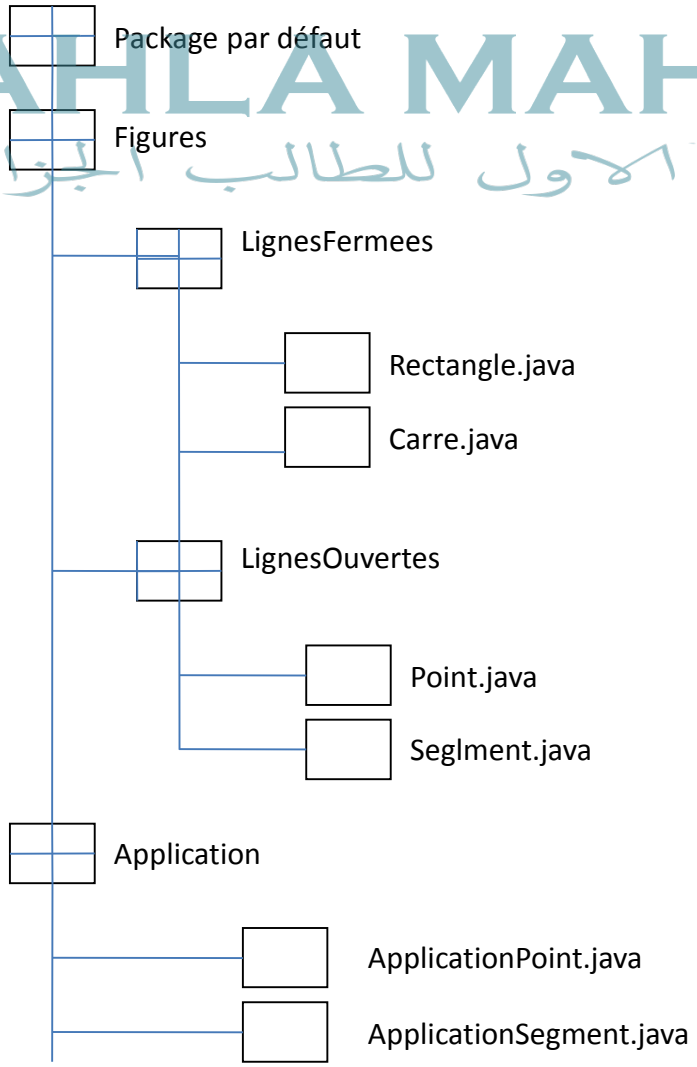
Objet constant (exemple)


```
final Point p1=new Point(3,4);
 Point p2=new  Point(6,7);
  // interdit
 // la référence de p1 est constante
  // p1=p2;
// permis
// Les attributs de p1 ne sont pas constants
p1.x++; p1.y++;
```

Package

Java permet de structurer un programme Java en plusieurs modules, appelés package. Un package est un répertoire regroupant plusieurs classes conceptuellement cohérentes par rapport à leurs fonctionnalités. De plus, il peut être organisé de manière hiérarchique en plusieurs (sous) packages.

Package (exemple)



✓ Pour spécifier qu'une classe appartient à un package, il suffit de rajouter au début du fichier l'instruction

package nomDePackage

où nomDePackage est l'identifiant donné au package.

✓Pour utiliser les services offerts des classes d'un package préalablement défini, l'instruction « import » doit être rajoutée au début du fichier comme suit :

a. *import cheminDePackage.** permet d'importer toutes les classes du package dont le chemin est spécifié.

b. *import cheminDePackage.NomClasse* permet d'importer la classe désignée.

Le concept d'encapsulation

✓ La définition des classes vues jusqu'à maintenant autorise l'accès directe à tous les attributs et méthodes d'un objet. Ainsi, les données de l'objet ne sont pas protégées car elles peuvent être manipulées directement et sans contrôle.

✓ Le mécanisme d'encapsulation, premier concept fondamental de la programmation orientée objet, permet de pallier à ce problème. Ce mécanisme permet de contrôler l'accès aux membres des objets en offrant des niveaux de visibilité.

✓ Il existe plusieurs niveaux de visibilité mais on se limite à présenter trois niveaux dans cette section :

Niveau privé : tout membre privé n'est visible seulement qu'à l'intérieur de sa classe. Le modificateur private doit précédé la définition de ce membre.

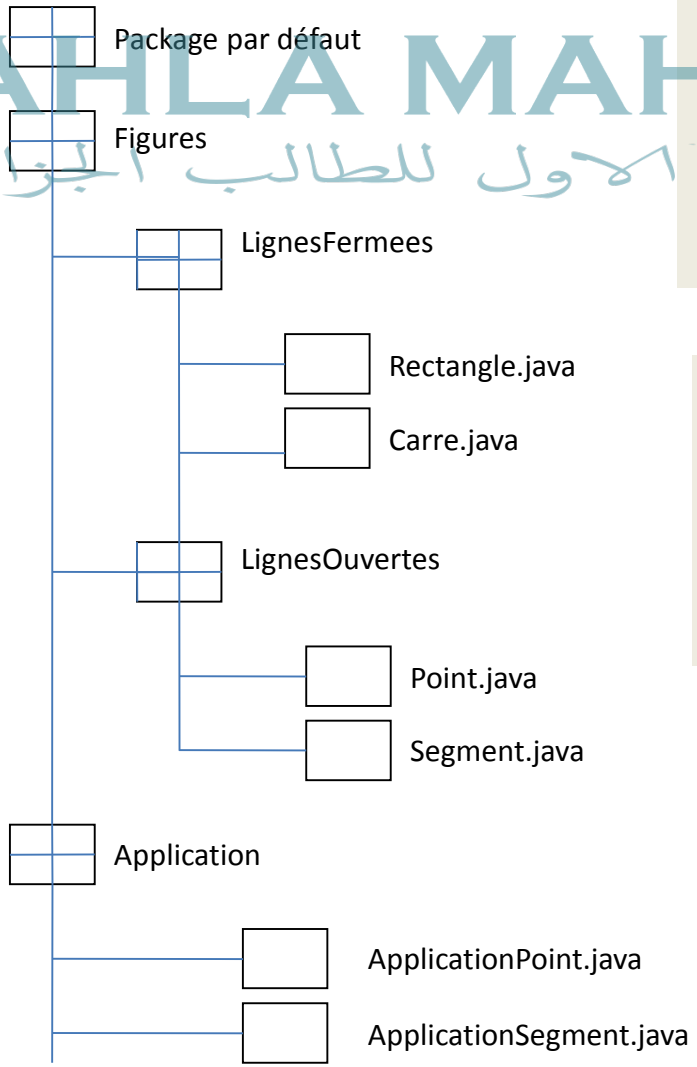
Niveau public : les membres publics d'un objet sont visibles de l'extérieur et sont définis avec le modificateur public.

Niveau package (par défaut) : Tout membre d'un objet non précédé par un modificateur de visibilité est accessible seulement dans le package de sa classe.

Règles d’encapsulation

Visibilité\membre	Sa classe	Une classe de son package	Une classe d’un autre package
private	oui	non	non
public	oui	oui	oui
packag(par défaut)	oui	oui	non

Package (exemple)



Private

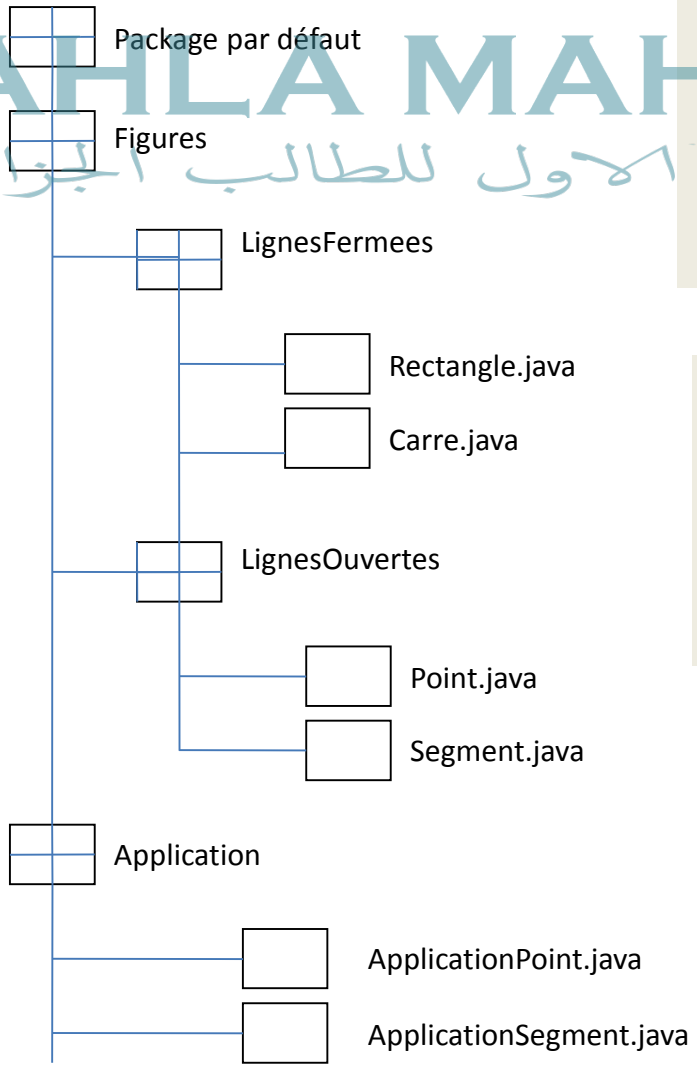
```
package Figures.LignesFermees;
class Point {
private double x,y;
double  getX() {return x;}
}
```

```
package Figures.LignesFermees;
class Segment {
Point e1,e2;
double  getE1x() {return e1.x;}
}
```

```
package Application;
Import Figures.LignesFermees.Point;
class ApplicationPoint
{public static void main(String args[])
{Point p=new Point(3,4);
p.x++;

}}
```

Package (exemple)

Par défaut

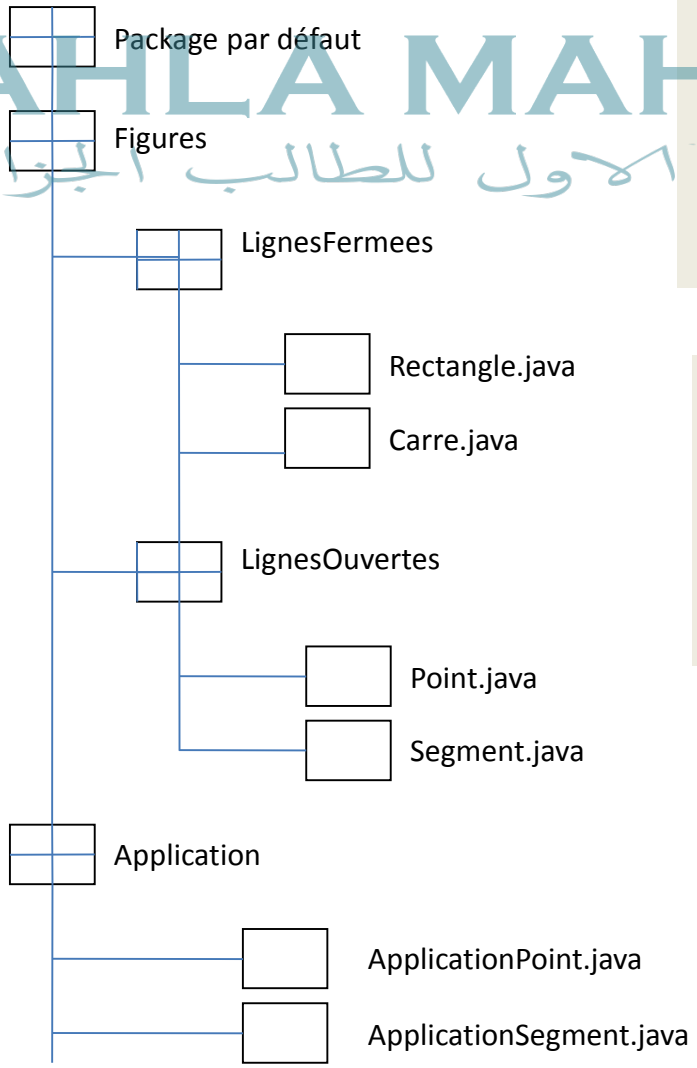



```
package Figures.LignesFermees;
class Point {
double x,y;
double  getX() {return x;}
}
```

```
package Figures.LignesFermees;
class Segment {
Point e1,e2;
double  getE1x() {return e1.x;}
}
```

```
package Application;
Import Figures.LignesFermees.Point;
class ApplicationPoint
{public static void main(String args[])
{Point p=new Point(3,4);
p.x++;

}}
```

Package (exemple)



Public

```
package Figures.LignesFermees;
class Point {
public double x,y;
double  getX() {return x;}
}
```

```
package Figures.LignesFermees;
class Segment {
Point e1,e2;
double  getE1x() {return e1.x;}
}
```

```
package Application;
Import Figures.LignesFermees.Point;
class ApplicationPoint
{public static void main(String args[])
{Point p=new Point(3,4);
p.x++;

}}
```

Les Accesseurs : Getteurs et Setteurs

✓ La conception orienté objet recommande vivement de protéger les attributs d'une classe à l'aide des modificateurs de visibilité mais le concepteur peut autoriser l'accès et/ou modifications de certains attributs en utilisant des méthodes spécifiques appelées accesseurs.

✓ On distingue deux types d'accesseurs : guetteurs et setteurs. Le rôle d'un getteur est de fournir une donnée alors que celui d'un setteur est de modifier des valeurs d'attributs.

Convention :
On utilise les mots get ou set, suivi du nom de l'attribut commençant par une majuscule.

Exemple (getteur et setteur)

```
public class PointColore  {
// Attributs
private double x,y ;
String couleur ;

// Constructeur
public PointColore(double x, double y, String couleur)
{this.x =x ; this.y=y ;this.couleur=couleur ;}

// getteurs et setteurs
public double getX(){return x ;}
public double getY(){return y ;}
public String getCouleur(){return couleur ;}

public void setX(double xx){ x=xx ;}
public void setY(double yy){ y =yy;}
public void setCouleur(String cc){couleur=cc ;}
// Les méthodes de traitement
// …..
}
```

Représentation d'une classe en UML

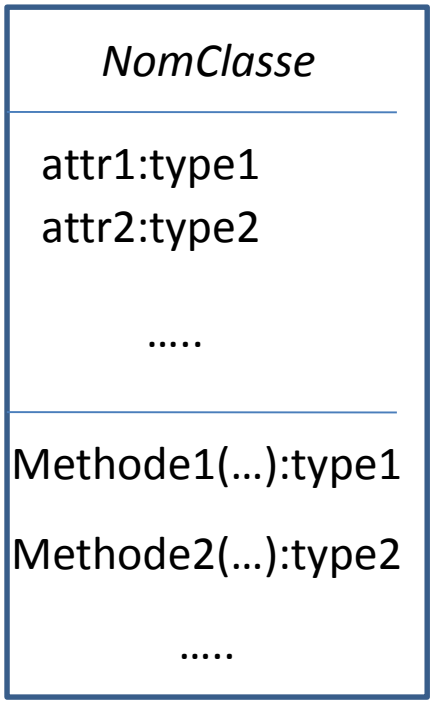
✓ UML, c'est l'acronyme anglais pour « Unified Modeling Language ». On le traduit par « Langage de modélisation unifié ». La notation UML est un **langage visuel** constitué d'un ensemble de schémas, appelés des **diagrammes**, qui donnent chacun une vision différente du projet à traiter. UML nous fournit donc des diagrammes pour **représenter** le logiciel à développer : son fonctionnement, sa mise en route, les actions susceptibles d'être effectuées par le logiciel, etc.

✓ Dans ce module, nous utiliserons le diagramme de classe.

Représentation UML des classes

Rectangle composé de compartiments :

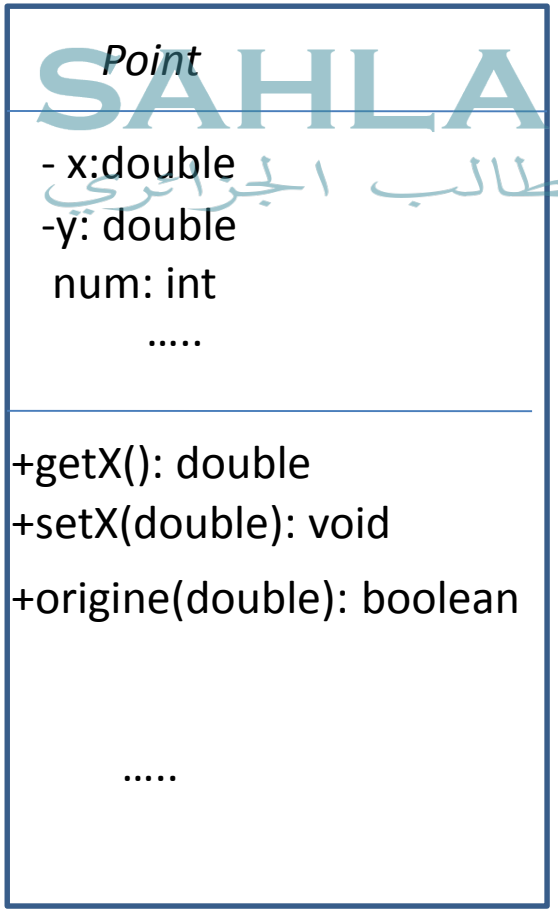
1. Compartiment 1 : Nom de la classe (commence par une majuscule, en gras)
2. Compartiment 2 : Attributs
3. Compartiment 3 :Méthodes



Les modificateurs de visibilité

- ✓***Public*** **ou +**
- ✓ ***Private*** **ou -**

Représentation UML des classes



Les modificateurs de visibilité

- ✓ ***Public*, +**
- ✓ ***Private*, -**
- ✓ **Par défaut ,rien**

Conclusion

✓ Deux notions importantes ont été introduites à savoir : objet et classe.

✓ Un objet est une entité caractérisé par un ensemble de données (propriétés) et de comportements.

✓ Une classe donne la description des propriétés (attributs) et comportements (méthodes) d'une famille d'objets.

✓**L'encapsulation** est le premier concept de l'orienté objet, il permet au concepteur de protéger ses données et méthodes, ainsi, les détails d'implémentation sont cachés à l'utilisateur. Une classe est vue donc comme une boite noire qui offre un ensemble de services.

✓L'encapsulation est gérée par les niveaux de visibilité (public, private, package). Le modificateur de visibilité private est recommandé pour les attributs d'une classe.

TESTONS NOS CONNAISSANCES

Soient les classes java suivantes :

```
package P1 ;
public class C1{
private int x ;
public int y ;
int z ;
C1(){……}
void f( ){
x++ ;//
y++ ;//
z++ ;//
}
}
```

CORRECTE

CORRECTE

CORRECTE

```
package P1 ;
 class C2{ public int x ;
void f( ){
C1 obj   //
 obj=new C1( ) ; //
obj.x++ ; //
obj.y++ ; //
obj.z++ ;//
x++;//
}
.....
}
```

CORRECTE

CORRECTE

FAUX

CORRECTE

CORRECTE

CORRECTE

```
package P2 ;
import P1.* ;
class C3{
void f( ){
C1 obj //
obj =new C1( ) ;
obj.x++ ;//
obj.y++ ;//
obj.z++ ;//
}
.....
}
```

CORRECTE

FAUX

FAUX

CORRECTE

FAUX

```
package P2 ;
class C4{
void f( ){C1 obj
obj =new C1( ) ; //
C2 obj2 = new C2() ;//
obj.x++ ;//
obj.y++ ;//
obj.z++ ;//
obj2.x++ ;//
}
…….}
```

Chapitre 2:

Eléments de Java pour démarrer nos projets

Affichage des informations

✓L'affichage à l'écran en java utilise l'instruction **System.out.print** (ou **System.out.println**)

✓ **System.out.print est** définie dans le package standard java.lang.

Sa syntaxe:

System.out.print ("message"); *// pour afficher le texte //d'un message à l'écran*

System.out.print (var); *// pour afficher le contenu de la //variable var*

Pour afficher plusieurs éléments, on utilise l'opérateur +.

Exemple :

int x=3 ;

System.out.print ("Valeur de x= "+ x) ; // pour afficher Valeur de x= 3

Affichage d'un objet

L'affichage d'un point permet d'afficher sa référence.

```
Point P1=new Point ();
System.out.println(P1) ;
```

On aura à l'écran **Point @ 12a34f**

Saisie de données

La saisie de données en java peut se faire en utilisant plusieurs classes. Dans ce module, nous nous contentons de deux classes Java: Scanner et JOptionPane.

✓ **Saisie avec La classe Scanner**

La classe Scanner est définie dans le package java.util. La saisie se fait en créant d'abord un objet de la classe Scanner.

Création d'un objet Scanner

```
Scanner e = new Scanner (System.in);
```


Classe Java

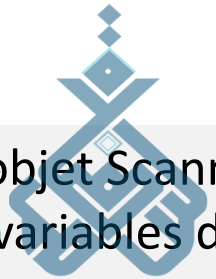
Avec e, je lit mes variables de types primitifs.

Lecture à partir du clavier

L'objet **e** est utilisé par la suite pour lire des entiers, réels, chaines de caractères, … en invoquant une méthode appropriée au type de la donnée à lire. Par exemple, **nextInt()** pour lire un entier, **nextFloat()** pour lire un réel, **next()** pour une chaine de caractères.

Exemple Scanner

Import obligatoire

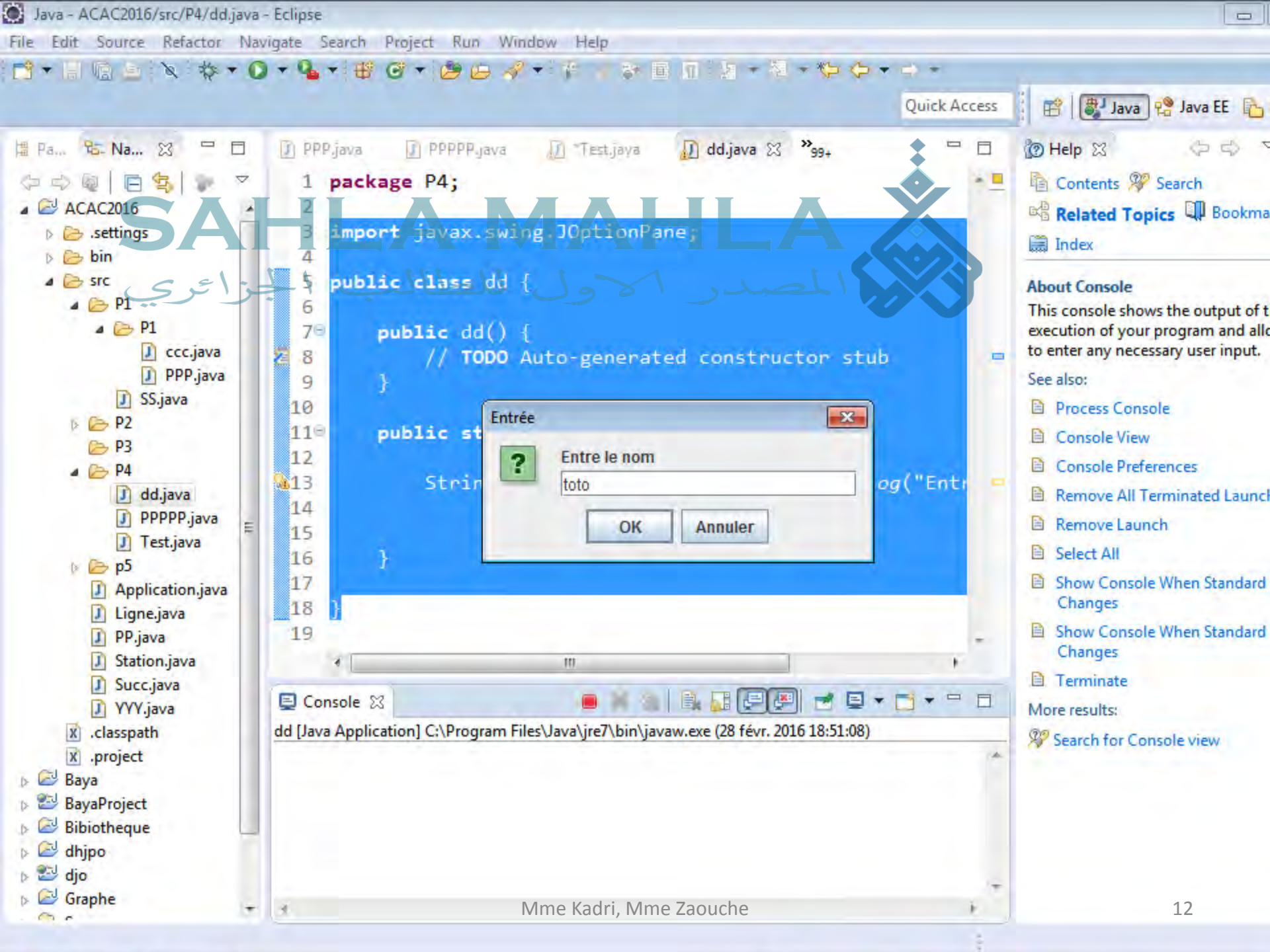


Un seul objet Scanner pour lire mes variables de types primitifs.

```
import java.util.Scanner;
class Test{

public static void main(String args[]{
Scanner e=new Scanner(System.in);
int x=e.nextInt();
String ch=e.next();
double y = e.nextDouble();
}}
```

Lecture avec boites de dialogues


```
import javax.swing.JOptionPane;

public class Test {


public static void main(String[] args) {

String Nom = JOptionPane.showInputDialog("Entre le nom");


}

}
```


Java - ACAC2016/src/P4/dd.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java
Java EE
ACAC2016
.settings
bin
src
P1
P1
ccc.java
PPP.java
SS.java
P2
P3
P4
dd.java
PPPPP.java
Test.java
p5
Application.java
Ligne.java
PP.java
Station.java
Succ.java
YYY.java
.classpath
.project
Baya
BayaProject
Bibiotheque
dhjpo
djo
Graphe
PPP.java
PPPPP.java
Test.java
dd.java
1 package P4;
3 import javax.swing.JOptionPane;
5 public class dd {
7 public dd() {
8 // TODO Auto-generated constructor stub
9 }
11 public st
13 Strin
16 }
18 }
Entrée
Entre le nom
toto
OK
Annuler
Console
dd [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (28 févr. 2016 18:51:08)
Help
Contents
Search
Related Topics
Index
About Console
This console shows the output of t
execution of your program and allo
to enter any necessary user input.
See also:
Process Console
Console View
Console Preferences
Remove All Terminated Launch
Remove Launch
Select All
Show Console When Standard
Changes
Show Console When Standard
Changes
Terminate
More results:
Search for Console view

Les tableaux à une dimension

Un tableau en Java est considéré comme un objet permettant de rassembler sous un même identificateur **un nombre fixe** de données de **même type**.

Déclaration d'un tableau

La taille n'est pas précisée

La déclaration d'un tableau se fait comme suit :

Type nomTab[ ] ; ou Type[ ] nomTab;

Exemple:

int[] tab1; int tab2[]; Point[] TP;

Création d'un tableau

La création d'un tableau se fait comme suit :

tab=new Type[taille];

Exemple:

tab= new int[10];

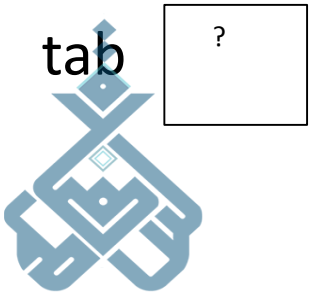
TP=new Point[20];

// ATTENTION on a crée seulement les références les points n'ont pas encore été créés

Déclaration et création d'un tableau de 3 entiers

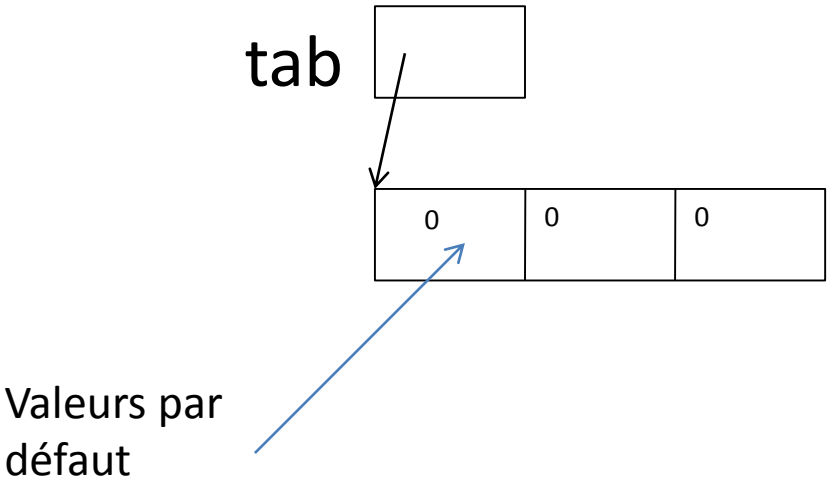
- Déclarer un tableau d'entiers

  int [ ] tab;



- Fixer la taille du tableau et lui allouer de l'espace.

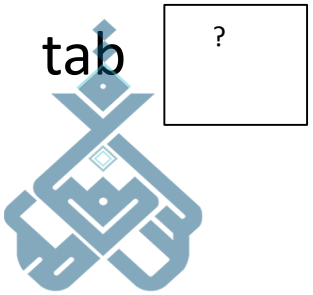
  tab= new int [3];



Déclaration et création d'un tableau de 3 Points

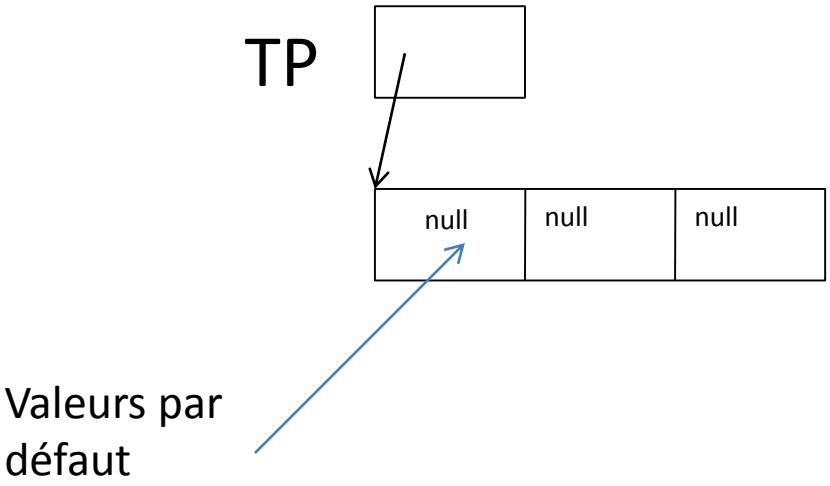
- Déclarer un tableau de Point

  Point [ ] TP;



- Fixer la taille du tableau et lui allouer de l'espace.

  TP= new Point [3];



Lecture d'un tableau

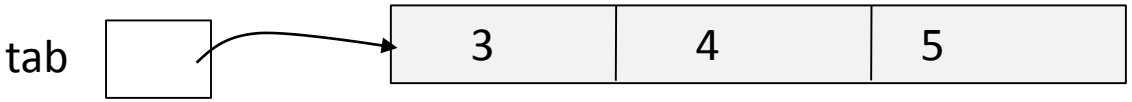
Tout tableau est caractérisé par un **attribut length** qui donne sa taille. Les éléments du tableau sont indicés à partir de 0 et sont initialisés par défaut (un booléen est initialisé à false, un numérique à 0, une référence à null…).

Exemple:

```
for (int i=0;tab.length;i++)
tab[i]=e.nextInt();// e est un objet scanner déjà créé.
```

Avant la saisie

Après la saisie



J'ai saisi:
3, 4, 5

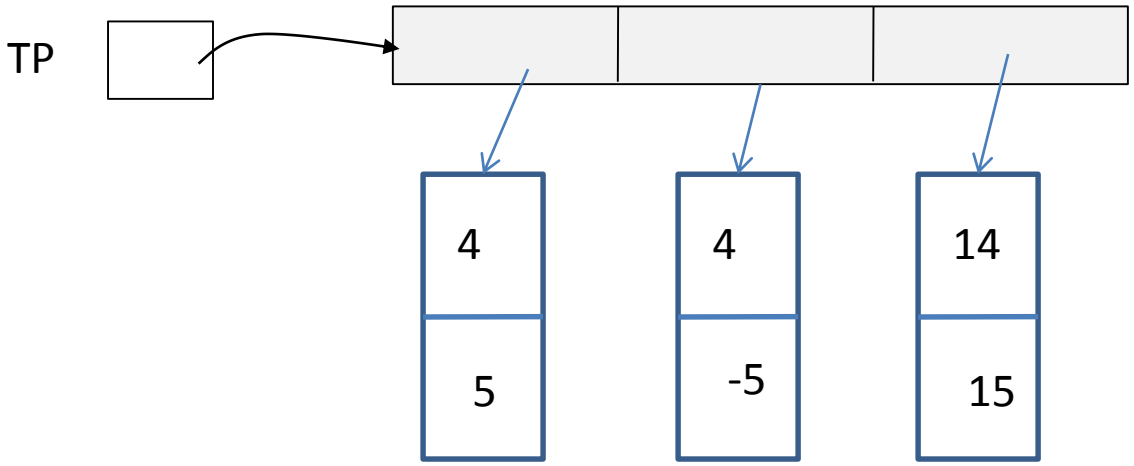
Lecture d'un tableau d'objets

Exemple 2:

```
for (int i=0;tab.length;i++)
TP[i]=new Point(nextDouble(), nextDouble());
```

Avant la saisie

Après la saisie



Donnez les valeurs saisies.

Affichage d'un tableau à l'aide d'un for classique


```
for (int i=0; i<tab.length;i++)
System.out.println(" "+tab[i]);
```

```
for (int i=0; i<TP.length;i++)
points[i].affiche();
```

For étendu

```
for (Type   nomElement :nomTableau )

{
   // traitement
}
```

REMARQUE : le for étendu ne permet pas de modifier le tableau

Affichage d'un tableau à l'aide d'un for étendu


```
for (int   el:   tab)
System.out.println(" "+el);
```

Type du tableau

Nom d'un élement

Nom du tableau

```
for (Point p: TP)
p.affiche();
```

```
for (int   el : tab)
System.out.println(el);
```

```
import java.util.Scanner;
 public class Application {
 public static void main (String [] args){
 // créer un  objet Scanner
```


```
// déclarer un tableau de Point, appelé points

// lisez un entier N
// créer le tableau  points avec la taille N
/ remplir points
// afficher les points.

…..}}
```

```
import java.util.Scanner;
 public class Application {
 public static void main (String [] args){
 Scanner e=new Scanner(System.in) ;
Point []TP;
System.out.println("Donnez le nombre de points") ;
int N=e.nextInt() ;
TP=new Point [N] ;

// lecture du tableau
for (int i=0 ; i<N ;i++ ){
 System.out.println ("Introduisez les cordonnées du point n° "+
i) ;
 TP[i]=new Point (e.nextInt(), e.nextInt()) ;
}
// Affichage du tableau for classique
for(int i=0 ;i<TP.length ;i++ )
TP[i].affiche () ;}}
```

Initialisation sans new

On peut déclarer un tableau et l'initialiser directement sans utiliser l'opérateur de création new.

Exemples : int[] tab = {3,5,8} permet de créer un tableau de trois entiers

Point [ ] TP= {new Point(2,3), new Point (-3,8), null}; permet de créer un tableau de points

Tableau à deux dimensions

Une matrice est une généralisation des tableaux à une dimension et considérée comme un tableau de tableaux,

l'attribut length associé au nom de la matrice ne permet pas de donner la taille de la matrice mais seulement **le nombre de ses lignes.**

Pour déclarer un tableau à deux dimensions, on peut procéder de deux manières :

***La première manière* :** Allouer la matrice en une seule fois en précisant les deux
dimensions. Voici un exemple d'illustration :

```
int [ ] [ ] mat =  new  int [2] [3];    // crée un tableau de six éléments entiers
```

La deuxième manière : Allouer d'abord un vecteur de références vers des tableaux à une dimension ; ainsi on aura défini un tableau pour référencer les lignes de la matrice.
Puis on alloue (crée) une par une les lignes de la matrice.
Les lignes ne doivent pas avoir nécessairement la même dimension.

Le même exemple pourrait se réécrire comme suit :

```
int [ ] [ ] mat;
 mat =  new  int [2] [ ];    // crée deux éléments qui sont des références
for (i = 0 ; i < mat.length ; i ++)
 mat [i] =  new  int [3] ;      // on crée les lignes une par une
```

Cette technique est intéressante quand on veut créer des matrices ayant des lignes de tailles différentes.
Ainsi pour avoir la matrice avec 5 colonnes sur la première ligne et 8 colonnes sur la 2ème ligne,
on écrira :

```
int [ ] [ ] mat;
        mat =  new  int [2] [ ];
// crée deux éléments    qui sont des références
            mat[0] =  new  int [5] ;
          mat[1] =  new  int [8] ;
```

Les chaines de caractères

La classe String

Cette classe possède deux constructeurs, l'un sans arguments créant une chaine vide, l'autre avec un argument de type String qui en crée une copie.

Exemple :

```
String ch1=new String () ;
            Ou
String   ch1=""
```

La classe String

String ch2=new String ("Java (") ;

Ou

String ch2="Java "

String ch3=new String (ch2) ;

ou

ch3=ch2 ;

Concaténation de ceux chaines

La concaténation deux chaines peut se faire soit avec la méthode concat () ou l'opérateur + ;

T = T+H ; ou T= T.*concat*(H) ;

Toutefois, bien que la méthode *concat* ne s'applique que sur des chaines de caractères, l'opérateur + peut opérer aussi bien sur des chaines que sur des numériques.

L'opérateur + entre deux numériques donne un numérique, mais il produit une chaine dés que l'un de ses arguments est une chaine.

Détecter les 7 erreurs


```
class Application {
Scanner sc = new Scanner(System.in);
Féminin saisie( ){
private  Féminin  f = new Féminin(sc.next(),…….);
return f;
}
public static void main( String [} A){
Saisie();
Int T [] = new int [20];
Int i = sc.nextInt();
T[i] = 100;
Point [ 10] TP  = new Point[10];
for( Point el : TP) el = new Point(sc.nextInt(), sc.nextInt());
// Déterminer le nombre de Points égaux à (3,3);
Point P1 = new Point(3); int nb=0;
for(int i = 0; i< 10; i++) if(  TP[i] == P1) nb++;
……..
}}
```

Les chaines de caractères

Création de chaines de caractères

Cette classe possède deux constructeurs, l'un sans arguments créant une chaine vide, l'autre avec un argument de type String qui en crée une copie.

Exemple :

```
String ch1=new String () ;
            Ou
String   ch1=""
```

Création de chaines de caractères

String ch2=new String ("Java (") ;

Ou

String ch2="Java"

String ch3=new String (ch2) ;

ou

ch3=ch2 ;

Manipulation de chaines de caractères

Concaténation de ceux chaines

La concaténation deux chaines peut se faire soit avec la méthode concat () ou l'opérateur + ;

T = T+H ; ou T= T.*concat*(H) ;

Toutefois, bien que la méthode *concat* ne s'applique que sur des chaines de caractères, l'opérateur + peut opérer aussi bien sur des chaines que sur des numériques.

L'opérateur + entre deux numériques donne un numérique, mais il produit une chaine dés que l'un de ses arguments est une chaine.

Longueur d'une chaine

La classe String ne possède aucun attribut. Parmi ses méthodes, la méthode length() donne la longueur de la chaine est spécifiée.

Exemple :

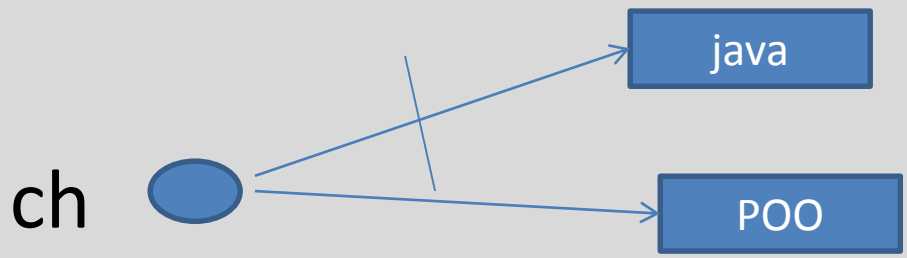
```
String ch2="Java"

int lg=ch2.length();
```

Caractéristique des objets de String

Les objets de la classe String sont immuables : ils ne peuvent changer de valeur, On a la possibilité de lire un caractère et non le modifier. Toute opération permettant de changer le contenu de la chaine invoque la création d'une nouvelle chaine.

EXEMPLE : ch = " java “;……..

ch = “POO "



Test d'égalité

Le test d'égalité entre deux chaines avec l'opérateur == ne permet pas toujours de comparer les contenus des chaines de caractères. Pour comparer l'égalité de deux chaines, on doit utiliser la méthode **equals**.


```
String ch1=e.next();// e est un objet scanner
boolean bool= (ch1 == ''POO''); // je saisis POO mais bool est à false!!!!
```

```
String ch1=e.next();// e est un objet scanner
boolean bool=ch1.equals(''POO''); // je saisis POO et bool est à true!!!!
```

```
String ch1=e.next();// e est un objet scanner
boolean bool=ch1.equalsIgnoreCase (‘’ POO’’ )// je saisis pOo et bool est à true!!!!
```

```
int x=7;
int y=3;
String z=y+x; // z=10  // ici il s'agit de l'addition
```



```
int x=7;
int y=3;
String  z="'"+y+x; // z="37"  // ici il s'agit de
concaténation de caractères
```

Comparaison de chaines

La méthode compareTo permet de comparer deux chaine - comparaison léxicographique –
Elle retourne une valeur = 0 si égalité,
>0 si la chaine >
et <0 si la chaine est <0

Conversion d'une chaine à un numérique

Pour convertir une chaine de caractères en un entier, on utilise la méthode statique *parseInt* de la classe enveloppe Integer.

Exemple

```
String ch = "12" ; int val = Integer.parseInt ( ch);
System.out.println(val+8);
```

De même, pour convertir une chaine en long, float ou double, on utilise respectivement les méthodes parseLong, parseFloat ou parseDouble des classes Long, Float et Double.

Conversion d'un numérique en une chaine de caractères

Pour convertir un entier en une chaine de caractères, on utilise trois possibilités:

1. On utilise la méthode valueOf de String comme suit :
String ch = String.valueOf(valeur); // c'est valable pour tout type de la valeur car cette méthode est surchargée pour chacun des types.
2. On utilise la méthode toString des classes Integer, Long, Float ou Double selon la valeur à convertir comme suit :
String ch = Integer.toString(valeur); // de même pour les autres classes
3. On concatène une chaine (même si elle est vide) au nombre. String ch = ""+valeur ;

Conversion entre tableau de caractères et chaine

1. construire une chaine de caractère à partir d'un tableau de caractèresil suffit d'instancier une chaine avec comme paramètre ce tableau.

Exemple char mot [] = {'R','E','U','S','S','I','T','E') ;

String ch = new String (mot) ; // ch est maintenant la chaine "REUSSITE".

2. De façon symétrique, On peut transformer un objet String en un tableau de caractères grâce à la méthode toCharArray.

Exemple

String ch="JOIE" ;

char mot[] = ch.toCharArray() ;

REMARQUE:

Pour appliquer le for étendu sur une chaine on doit utiliser la méthode toChar Array

L’accès à un élément d’une chaine

L’accès à un élément d’indice i d’une chaine se fait grâce à la méthode charAt

Exemple

```
String s = new String(‘’On avance bien ‘’;
char c = s.charAt(6);
```

Parcours d'une chaine de caractère

Le parcours d'une chaine peut se faire de deux manières

1. En utilisant la méthode charAt

Exemple si on veut afficher tous les caractères numériques d'une chaine, on aura le code suivant :

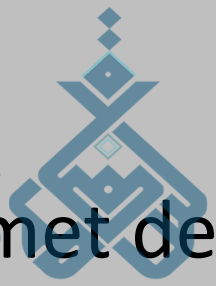
```
for( int i=0 ; i<ch.length() ;i++)
if((ch.charAt(i)<='9')&& (ch.charAt(i)>='0'))
      System.out.print( ch.charAt(i) ) ;
```

2. En transformant la chaine en un tableau de caractère

Le code suivant répond à l'énoncé précédent :

```
      for(char e :ch.toCharArray())
if( e<='9' && e>='0')
System.out.print(e);
```

Exercice



1. Ecrire un programme qui permet de lire N chaines et les imprime en déterminant pour chacune le nombre de caractère $.

```
class Programme {
public static void main (String [] args) {
Scanner sc = new Scanner (System.in) ;
String H ; int cpt,
int N = sc.nextInt() ;
for(int i =0;i<N;i++)
  {  H= sc.next() ; cpt = 0;

      for (int  j=0 ; j< H.length();j++)
       if (H.charAt[j] =='$' ) cpt++ ;

System.out.println( ''La chaine est ''+ H + '' Le nb de $
= '' + cpt);    } }}
```

2. Modifier le programme si l'arrêt se fait à la rencontre d'une chaine spéciale ”fin”, l'utilisateur peut donner toute chaine en majuscule ou minuscule.

```
import java.util.Scanner
class Programme {
public static void main (String [] args) {
Scanner e = new Scanner (System.in) ;
String H ; int cpt,
while (true)  {
 H= e.next() ;
     if ( H.lowercase().equals("fin") ) break;
     cpt = 0;
     for ( int j=0 ; j< H.length();j++)
     if (H.charAt[j] =='$') cpt++;
     System.out.println (“La chaine est "+ H + " Le nb de $ = " + cpt);
  }}}
```

Signature de la méthode	Description
public **String**()	Constructeur
public **String**(String s)	Constructeur
public int **length**()	Longueur de la chaîne
public char **charAt**(int index)	Retourne le caractère à la position index
public String **substring**(int dbt,int fin)	Extrait la chaîne entre les positions dbt et fin
public boolean **equals**(Object o)	Test d'égalité
public boolean equalsIgnoreCase(String s)	Test d'égalité en ignorant la casse
public boolean **startsWith**(String s)	Test si chaîne s est égale au début de la chaîne
public boolean **endsWith**(String s)	Test si chaîne s est égale à la fin de la chaine
public int **compareTo**(String s)	Compare les 2 chaînes,(0 si égalité, <0 si elle est inférieure, >0 sinon)

Signature de la méthode	Description
public int **indexOf**(char c) public int **lastIndexOf**(char c)	Retourne la position du caractère c dans la chaîne Retourne la dernière position du caractère c
public int **indexOf**(char c, int i)	Retourne la position de c à partir de i
public int **indexOf**(String s)	Retourne la position de la sous-chaîne s
public String **replace**(char c,char d)	Remplace toute occurrence de c par d
public String **toLowerCase**()	Convertit la chaine en minuscules
public String **toUpperCase**()	Convertit la chaine en majuscules
public char[] **toCharArray**()	Convertit la chaîne en tableau de caractères
public String **trim**()	Supprime les espaces en début et fin de la chaîne
public static String **valueOf**(char t[])	Convertit un tableau de caractères en String

Chaîne formatée

On veut définir des matricules sur 12 caractères on utilise la méthode format de String

String mat = ""+ **new Date().getYear()+**

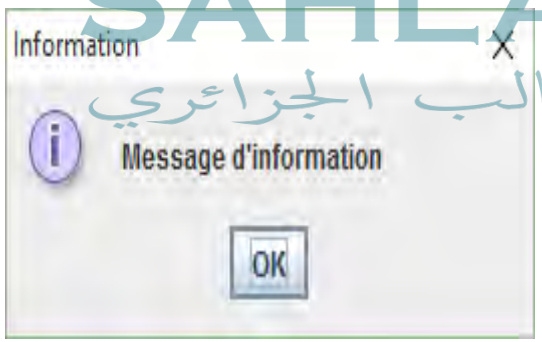
String.*format("%08d", num) ;*

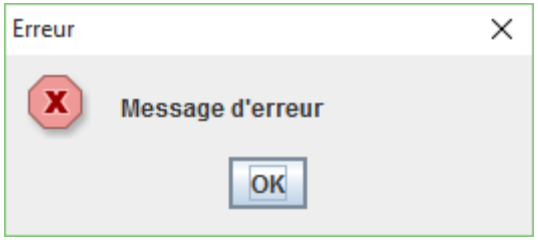
Exemple 201400000001

201400002222

201500000001

Les Boites de dialogue





```
JOptionPane.showMessageDialog(null, "Message d'information",
 "Information", JOptionPane.INFORMATION_MESSAGE);
```

Le premier argument est le ***JComponent*** parent. C'est à dire le composant au dessus du quel devra s'ouvrir la fenêtre.
S'il y en a pas comme ici, on met **null**.
On aurait cependant pu mettre **this** pour centrer la boite de dialogue au dessus du content pane,
même si la fenêtre principale avait été déplacée.

.

Le second argument est le message que l'utilisateur verra dans la **boite de dialogue**.

Le troisième est le titre de la **boite de dialogue**,
affiché dans la barre d'action
Et le dernier est celui qui définira s'il s'agit d'un message d'information, d'erreur ou d'avertissement.
D'ailleurs il est possible de simplement mettre un message, sans icône, à l'aide de ***JOptionPane.PLAIN_MESSAGE***,
ou alors une question avec
JOptionPane.QUESTION_MESSAGE.

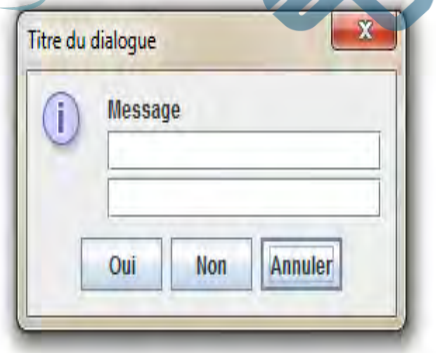
Java - yakoub/src/etatCivil/dedans/A.java - Eclipse

File Edit Source Refactor Navigate Search Project Run Window Help

Quick Access | Java

Test.java | A.java

```
1
2  package etatCivil.dedans;
3
4  import javax.swing.JOptionPane;
9
10 public  class A {
11 public static int x;
12     /**
13      * @param args
14      */
15 public static void g(){
16 // Données
17     Object[] message = new Object[3];
18     message[0] = "Message"; //Message apparaîssant dans le corps du dialog
19     message[1] = new JTextField();
20     message[2] = new JTextField();
21
22     //Options (nom des boutons)
23     String option[] = {"Oui", "Non", "Annuler"};
24
25     int result = JOptionPane.showOptionDialog(
26     null, // fenêtre parente
27     message, // corps du dialogue
28     "Titre du dialogue",
29     JOptionPane.DEFAULT_OPTION, // type de dialogue
30     JOptionPane.INFORMATION_MESSAGE, // type icone
31     null, // icône optionnelle
32     option, // boutons
33     option[2] // bouton par défaut
34     );
35
36     if(result == 0){
37     String saisie = "";
38     for(int i = 1; i < 3; i++)
39     saisie = saisie + ((JTextField)message[i]).getText();
40     }
41
42 }
```

Java - yakoub/src/etatCivil/dedans/A.java - Eclipse

File Edit Source Refactor Navigate Search Project Run Window Help

Quick Access | Java

Navigator

- abcds
- TP1
  - bin
  - src
    - Appli.java
    - Date.java
    - Féminin.java
  - .classpath
  - .project
- yakoub
  - .settings
  - bin
  - src
    - dedans
    - etatCivil
      - dedans
        - A.java
      - Test.java
  - .classpath
  - .project

Test.java | A.java

```
38      for(int i = 1; i < 3; i++)
39      saisie = saisie + ((JTextField)message[i]).getText();
40      }
41
42  }
43  public static  void f() {
44      Object[] message = new Object[ 4 ];
45      message[ 0 ] = "login :";
46      message[ 1 ] = new JTextField();
47      message[ 2 ] = "mot de passe";
48      message[ 3 ] = new JPasswordField();
49
50      String option[] = { "OK", "Annuler" };
51
52      int result = JOptionPane.showOptionDialog(
53        null,
54        message,
55        "connexion",
56        JOptionPane.DEFAULT_OPTION,
57        JOptionPane.QUESTION_MESSAGE,
58        null,
59        option,
60        message[1] );
61
62      if( result == 0 )
63      {
64        System.out.println(
65          "login : " + ( (JTextField)message[ 1 ] ).getText() +
66          "\npasse : " + new String( ( (JPasswordField)message[ 3 ] ).getPassword() ) );
67      }
68  }
69      public static void main(String[] args) {
70          // TODO Auto-generated method stub
71  f();
72      }
73
74  }
75
```

Writable | Smart Insert | 71 : 2

FR 20:39 06/03/2016

La classe StringBuffer

Cette classe a été introduite pour palier aux inconvénients de la classe ***String*** pour donner la possibilité de modifier le contenu d'une chaîne sans en créer d'autres instances,

la classe **StringBuffer** appartient au paquetage **java.lang**

ainsi, cette classe possède des méthodes permettant de modifier la ch aine sans créer de zones temporaires.

Cependant, dans cette classe on ne peut concaténer les chaines avec l'opérateur + ni avec concat mais une nouvelle méthode **append qui permet de** de réaliser les ajouts.

De plus on ne peut créer une chaine directement mais on doit passer explicitement par l'opérateur new.

StringBuffer H = "java"; **ERREUR**

StringBuffer H= new StringBuffer ("java"); **CORRECT**

Remarque : L'utilisation de cette classe permet un gain considérable de temps quand il s'agit de modifications répétitives de la chaine

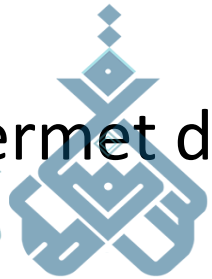
Cependant, les méthodes de cette classes sont synchronisées afin d'être utilisées par des processus parallèle (plusieurs thread sur une même chaine).

Ceci influe sur le temps d'exécution (perte de temps) dans le cas d'absence de parallélisme
(un seul processus ou thread)

La classe StringBuilder

La classe StringBuilder est comme la classe StringBuffer mais a été conçue pour un accès séquentiel à une chaine.
Elle présente les meilleurs temps quand il s'agit de modifier une chaine dans le cas de mono-threding.

Les Types Enumérés



✓On sait jusqu'à présent qu'une classe permet de créer un nombre infini d'objets.

✓Il existe des classes qui doivent avoir qu'un nombre fini et constant d'objets

Exemple : les jours de semaine, les mois de l'année, les spécialités d'une formation…

La question qui s'impose : Comment représenter ces éléments?

1. En représentant les mois par une classe ordinaire Année : c'est faux car on pourrait créer plus de 12 mois
Mois m = new Mois (….); on peut créer une infinité de mois.

2. En représentant les mois par un String : String mois
c'est faux car on ne pourrait faire rentrer un nom de mois non correct ‘’BONJOUR ‘’;

3. En représentant les mois par 12 constantes de type String : c'est correct mais lourd, on devrait comparer chaque chaine correspondante à un mois par les différentes constantes jusqu'à retrouver la bonne chaine, de plus si on veut rajouter des informations supplémentaires aux mois, on va encombrer notre base d'information de chaines répétées (consommation inutile d'espace)

•A partir de la version 5 de java, une solution a été proposée par la communauté scientifique,

•il s'agit de définir une classe spéciale composée d'un nombre fini et fixe d'objets appelée énumération.

•Un type énuméré permet de définir un ensemble objets constants

•définissant ainsi des types de données personnalisés tels que les jours de la semaine, les mois de l'année, le statut social d'une personne…

•Chaque élément d'une énumération est un objet à part entière.

Cette structure permet de contenir une série de données constantes ayant un type sûr (safe type)
Le contrôle par rapport aux valeurs de l'énumération est implicite.

Dans le langage java, une énumération est introduite par le mot **enum** au lieu de **class**.

A. Définition d'un type énuméré:
(forme minimale d'une énumération)

enum NomEnumération
{ objet1, objet2, …., objetn }

Exemples

enum Animal { KANGOUROU, TIGRE, CHIEN, SERPENT, CHAT}

enum Saison { HIVER, PRINTEMPS, ETE, AUTOMNE }

enum Spécialité_Master { MIL, RSD, SII, SSI,CP}

enum Mois {JANVIER, FÉVRIER, MARS, AVRIL, MAI, Juin, JUILLET, AOUT, SEPTEMBRE, OCTOBRE, NOVEMBRE, DECEMBRE}

Remarque : les éléments de l'énumération sont des objets spécifiques et non chaines de caractères.

2. Méthodes pour la manipulation de types énumérés

Il est possible de comparer des valeurs d'une énumération entre elles à l'aide des opérateurs de comparaison == et aussi l'utilisation de l'instruction switch.

De plus les quatre méthodes suivantes sont liées à toute énumération créée :

• **Premiere méthode : valueOf** est une méthode statique dont
• la signature est nomEnum valueOf(String)
exemple Mois.valueOf(String),
Cette méthode retourne l'objet correspondant à la chaine de caractère donnée en paramètre,

si aucun objet ne correspond à cette chaine,
une exception est générée.

Il s'agit de faire correspondre la chaine de caractère introduite par l'utilisateur
et l'objet énumération correspondant.
String ch= sc.next();
Exemple Mois m = Mois.valueOf(ch);

Si on introduit les chaines hhh ou janvier ou Janvier une exception sera déclenchée .
Par contre si on introduit JANVIER le code est correct.

Pour pallier au problème de la sensibilité à la casse,

on propose de donner des noms majuscules aux objets d'une énumération
et réaliser la conversion de la chaine quand on invoque
La méthode valueOf
M = Mois.valueOf(ch. toUpperCase()) ;

• **Seconde méthode : toString ()** cette méthode fait le travail inverse, elle transforme l'objet d'une énumération en chaine de caractères

•**Troisième méthode : values()** : elle regroupe tous les objets de l'énumération dans un tableau de valeur afin qu'on puisse réaliser un parcours sur ses éléments.

Exemple :

a/ for **(Mois m: Mois.values())** *// parcourir les éléments de l'énumération*

System.out.println (**m**) ;

b/ String ch = sc.next() ;
boolean chaineAccptée = false ;
for (Mois m:Mois.values())
{ if (ch. equalsIgnoreCase (m.toString())
{chaineAacceptée = true; break;}

•**Quatrième méthode : ordinal ()**: renvoie le numéro d'ordre d'un objet dans l'énumération,
•La numérotation commence à partir de 0, ainsi Mois.FEVRIER donne la valeur 1.
•Mois.DECEMBRE donne la valeur 11

```
enum SpécialitéMaster { MIL, RSD, SII, SSI,CP;
void présRequis(){
System.out.print("Prérequis : ");
switch ( this){
case MIL :
System.out.println( " Java+UML ") ; break ;
case RSD :
System.out.println( " Système + Réseau  ");
 break;
case SII :
System.out.println("Math + algorithmique“);
break ;
case SSI  :
System.out.println( "Réseau + Math ");
}}
```

Remarque : on aurait pu le faire comme suit

```
void présRequis(){

System.out.print("Prérequis : ");
switch ( ordinal()){
case 0 :System.out.println( " Java+UML ") ;
 break ;
case 1 :
System.out.println( " Système + Réseau  ");
break;
case 2 :
System.out.println("Math + algorithmique");
break ;
case 3  :
System.out.println( "Réseau + Math ");
                    }}
```

B/ La forme énumération avec constructeur

Dans une classe déclarée **enum**, il est possible des rajouter des informations aux constantes de l'énumération, il s'agit des caractéristique de chacun des objets

Dans ce cas, la définition des attributs et constructeur devient obligatoire. Le constructeur est obligatoirement privé car aucune nouvelle instance ne peut être créée.

On a aussi la possibilité de définir d'autres méthodes

```
enum NomEnumération
  {   obj1(v1, x1,…), obj2(v2,x2..), ….,
objn(vn,xn)…
      // Définition des attributs éventuellement
      Type1 attribut1 ;
      ….
      // Constructeur
      NomEnumération(paramètres) { … }


      // Définition des méthodes éventuellement
      Type1 méthode1 (paramètres) { …}     ….  } ;
```

```
enum SpécialitéMaster{ MIL(12, “GénieLogicie”),
RSD ( 15,“Réseaux et System”), SII ( 16, “Intelligence
Artificielle”), SSI ( 10, “ Sécurité”, CP ( 12, “calcul parallèl”);
private int nbMod;
private String “ intitulé”;

SpécialitéMaster( int nbMod, String intitulé) {
this.nbMod = nbMod;
this.intitulé = intitulé; }

int  nbModules() { return nbMod;}
}
```

```
class Etudiant {
private String nom, prénom, adresse;
private SpécialitéMaster spec;
private float note1, note2

public Etudiant ( String n, String p String a , String s){
nom = n; prénom = p ; adresse = a;
spec = SpécialitéMaster.valueOf(s.toUpperCase());
 // Attention on risque une exception si la chaîne ne correspond pas
// on traitera ce type d'erreur plus tard
……..
}
```

Concept d'héritage

Féminin
-code: int -nom:String -prénom:String -date:Date -email:Email -Epoux: Masculin
+getCode():int ……. +getEpoux():iMasculin +setEpoux(….):void

Masculin
-code: int -nom:String -prénom:String -date:Date -email:Email -Epouse:Feminin
+getCode():int ……. +getEpouse():Féminin +setEpouse(….):void

Redondant

Spécifique

Concept d'héritage

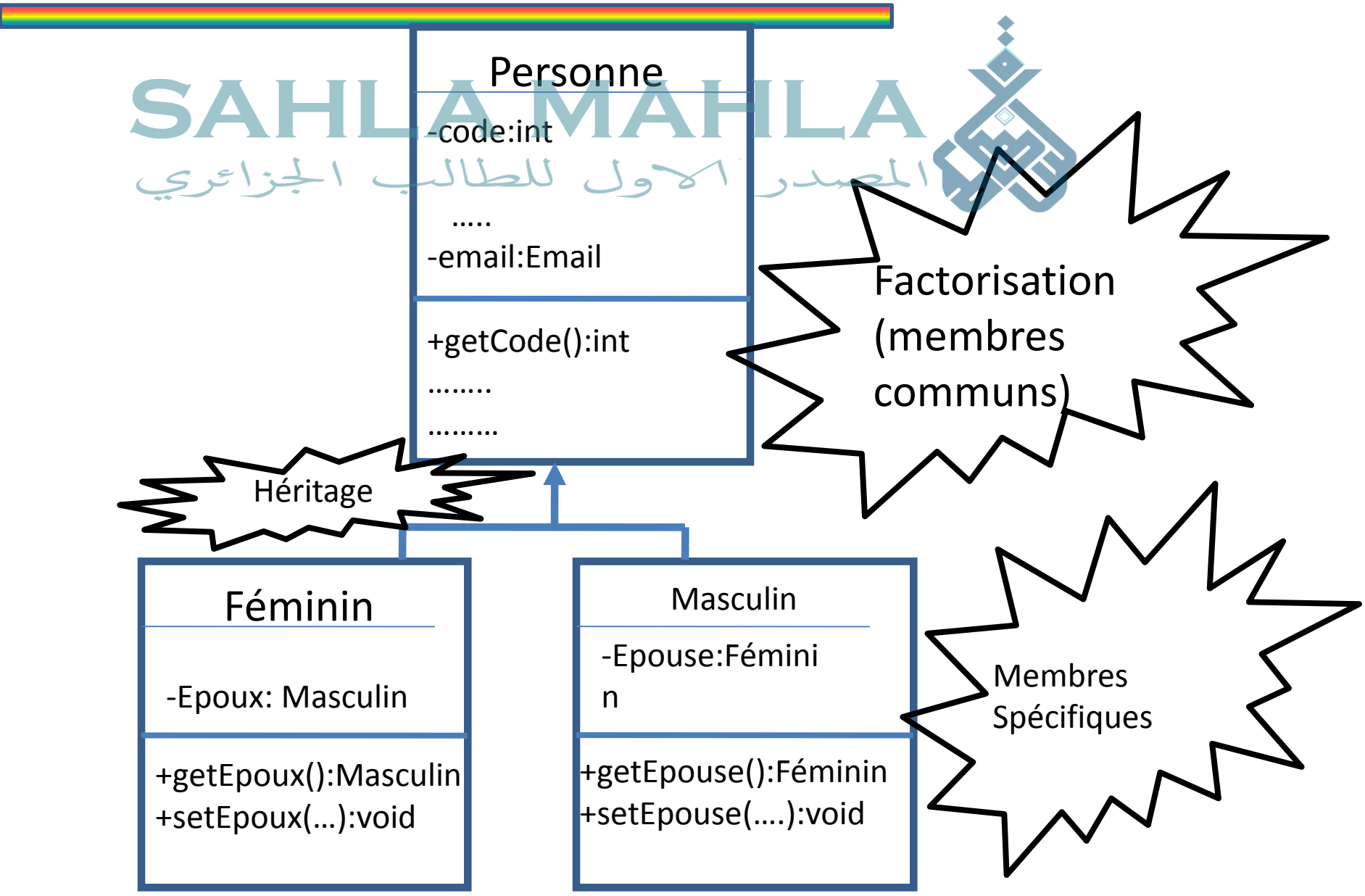
- Factoriser
- Créer une troisième classe

Spécifique

Féminin
-code: int -nom:String -prénom:String -date:Date -email:Email -Epoux: Masculin
+getCode():int ……. +getEpoux():iMasculin +setEpoux(….):void

Masculin
-code: int -nom:String -prénom:String -date:Date -email:Email -Epouse:Feminin
+getCode():int ……. +getEpouse():Féminin +setEpouse(….):void

Concept d'héritage



Concept d'héritage

L'héritage est **le second concept** de la programmation orientée objets.

Le terme héritage exprime le principe selon lequel une classe peut hériter des caractéristiques (attributs et méthodes) d'autres classes.
Elle consiste de créer de nouvelles classes à partir d'une classe existante,

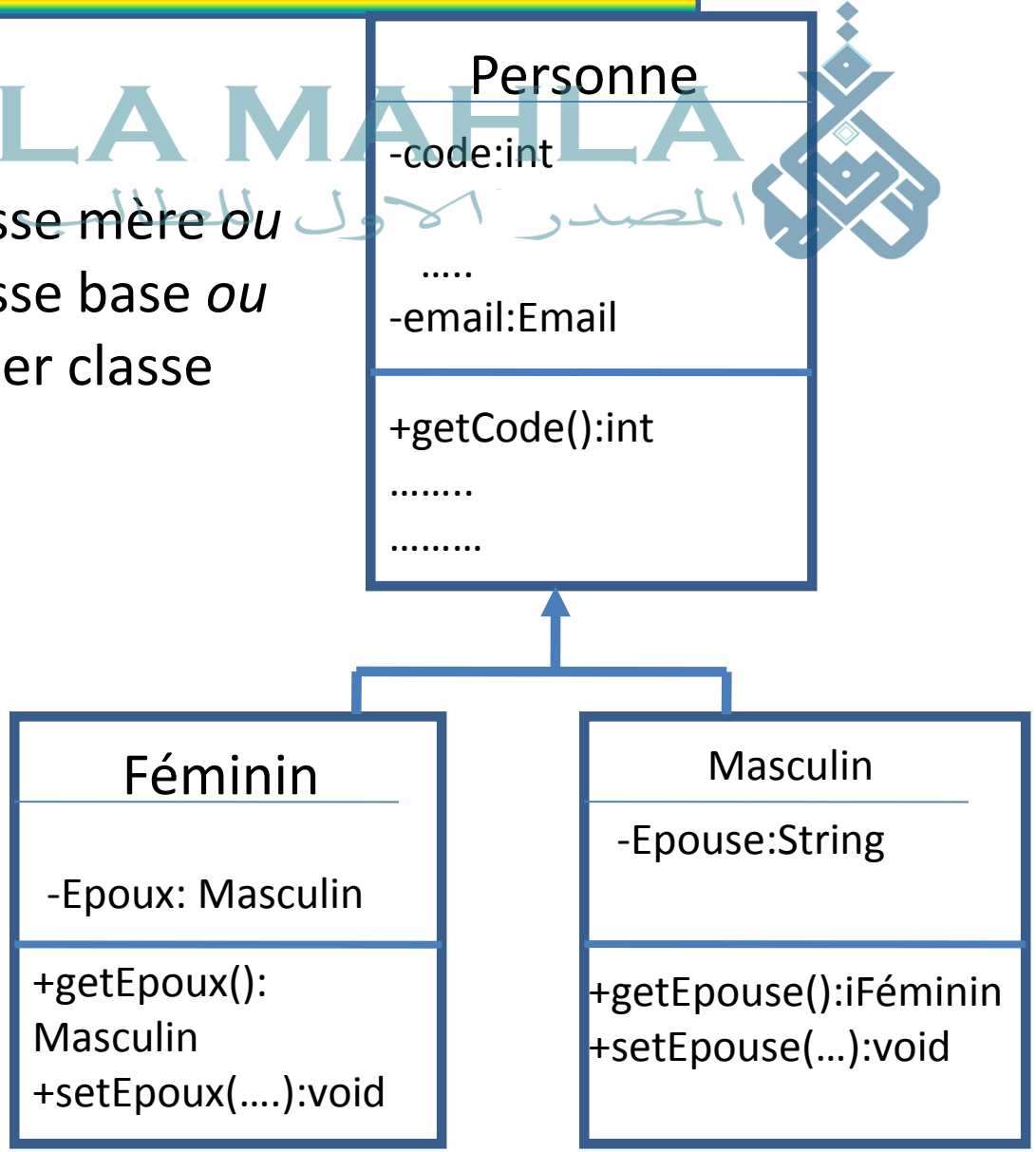
les classes nouvellement définies sont considérées des sous types de la classe d'origine.

L'héritage est exprimé à l'aide du mot réservé « **extends** ».

Un objet de type Personne est caractérisé par:…….
Un objet de type Masculin est caractérisé par: …….
Un objet de type Féminin est caractérisé par:…….

Concept d'héritage

- Classe mère *ou*
- Classe base *ou*
- Super classe

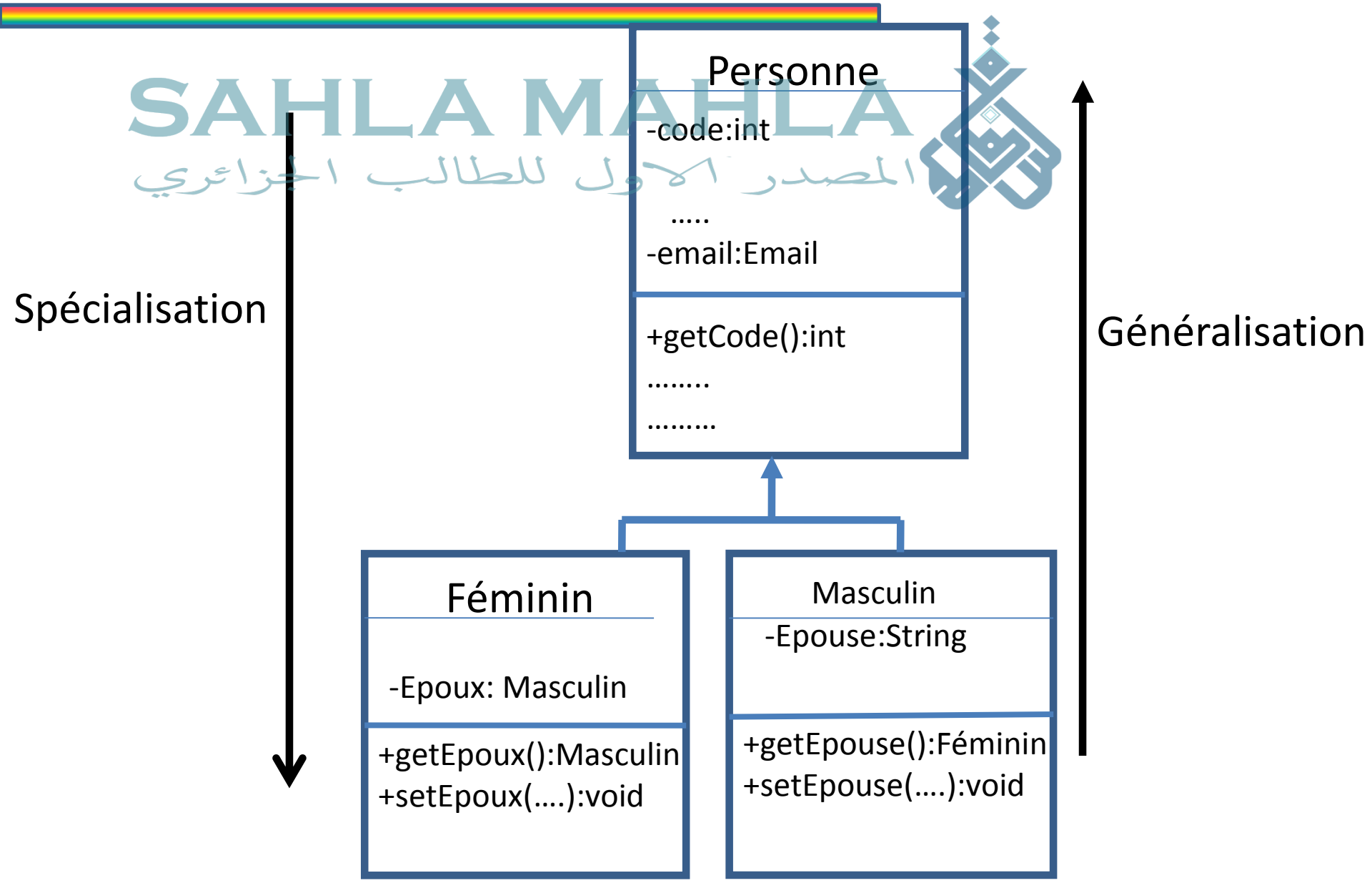


- Classe fille *ou*
- Classe dérivée

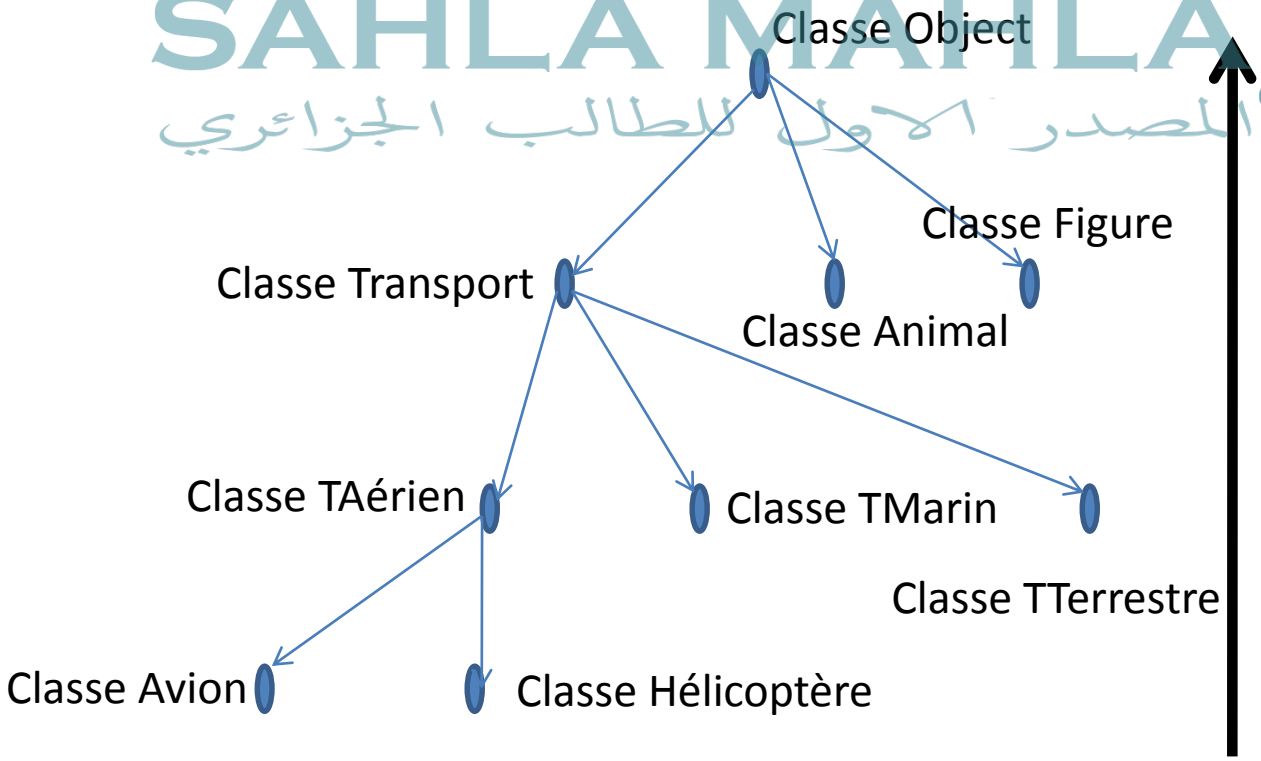
ou

- Sous classe

Héritage



Arbre d'héritage



- Chaque objet a le type de sa classe et ceux ses classes ascendantes.
- Il n' a pas les types de ses classes descendantes

Avion avion=new Avion (‘’ Airbus ’’, C123);
avion a les types: Avion, TAérien, Transport et Object

Membres d'un objet d'une classe fille

- Les membres d'un objet de la classe fille sont les membres définies dans sa classe et les membres de toutes ses supers classe jusqu'à la classe Objet. Ainsi, de point de vu espace mémoire d'un objet de la classe dérivée, un espace est réservé pour chaque attribut propre à la classe fille puis un espace pour chaque attribut de toutes les classes qui lui sont ascendantes.

- Une classe dérivée possède toutes les méthodes définies dans son corps et toutes les méthodes (sauf les constructeurs) de toutes ses super classes jusqu'à la classe Object.

Redéfinition des méthodes

Une classe fille peut:

Utiliser intégralement une méthode de sa classe mère.

Utiliser partiellement le code d'une méthode de sa classe mère.

Utiliser un code nouveau

Héritage des méthodes

Redéfinition des méthodes

Définition fraiche des méthodes

Redéfinition des méthodes

La redéfinition (overriding) est la possibilité d'utiliser exactement la même signature pour définir un service dans un type et dans un sous type (entre une classe et sa classe dérivée directement ou indirectement).

Le type de retour du service doit être le même, mais **la visibilité** peut changer (**mais doit être >= visibilité initiale**). On dit dans ce cas que la classe fille a spécialisé la méthode.

Redéfinition des méthodes


```
puclic class Mere{

void f() {s.o.p(«Je suis  f de la
mère  »;}
}
```

```
puclic class Fille extends Mere{

public void f() {s.o.p(«Je suis  f
de la fille  »;}
}
```

- **Même signature**
- **Visibilité supérieure**
- Cas de redéfinition

Redéfinition des méthodes


```
puclic class Mere{

public void f() {s.o.p(«Je suis le
f de la mère  »;}
}
```

```
puclic class Fille extends Mere{

 void f() {s.o.p(«Je suis le f de
la fille  »;}
}
```

Erreur: visibilité inférieure

Différence entre surcharge et Redéfinition


```
puclic class Mere{

void f() {s.o.p(«Je suis le f de la mère  »;}
}
```

```
puclic class Fille extends Mere{

 void f(int x) {s.o.p(«Je suis le f de la fille  »;}

}
```

cas de **Surcharge** et **non redéfinition**

Instance this et super

Chaque instance d'une classe dérivée est munie de deux références particulières :

- this réfère l'instance elle-même.

- super réfère la partie héritée de l'instance.

L'utilisation de l'un de ses mots clés est nécessaire quand on doit distinguer entre deux membres qui portent le même nom et se trouve dans plus d'une classe dans la hiérarchie de la dérivation.

Instance this et super

La référence this associé à un membre (exp this.m où m est un membre) permet de désigner le membre le plus proche à partir de la classe fille jusqu'à la classe Object.

La référence super associé à un membre permet de désigner le membre le plus proche à partir de la mère jusqu'à Object.

Instances this et Super


```
public class A {
int x=1, y=-1, w=-11;

}
class B extends A {
int x=2, y=-2, z=-22;

}
class C extends B {

int x=3, h=-3;
public void f1(){
System.out.println(this.x);
System.out.println(this.h);
System.out.println(this.y);
System.out.println(this.z);
} }
```

```
public void f2(){
System.out.println(super.x);
System.out.println(super.h);
System.out.println(super.y);
System.out.println(super.z);
}
                    }
```

Redéfinition de la méthode afficher de la classe Point


```
class PointEspace extends Point{
private double x, y,z…….

public void afficher() {
super.afficher(); System.out.print( ‘’, ‘’
+ z;)
}
```

On définit dans la classe point la méthode symetrie comme suit

```
public Point symetrie() {
return new Point(-x,-y) ;}
```


Cette methode peut être redéfinie dans la classe PointEspace comme suit :

```
public PointEspace symatrie(){
return new PointEspace(-getX(),-getY(),-z) ;}
```

Malgré que les deux signatures sont différentes
mais ça correspond à une redéfinition car PointEspace est un sous type de Point.

•**Remarque :**
Il est conseillé de rajouter l'annotation @**Override** avant toute méthode redéfinie, ceci est utile pour que le compilateur réalise le contrôle et envoie une erreur si la méthode ne redéfinie aucune autre méthode.

Mot clé final

Si une classe est précédée par le mot clé **final**, elle ne peut être dérivée.
Elle est considérée obligatoirement comme feuille dans la hiérarchie de l'héritage.
Si une méthode est précédée par le mot clé final, elle ne peut être redéfinie.

•Si, dans une classe fille, un constructeur est défini sans commencer par un appel de constructeur, Java insère un appel du constructeur par défaut (sans paramètre) de la super classe par l'instruction super().

•La classe dérivée n'hérite pas des constructeurs présents dans les supers classes.

•Le constructeur de la classe dérivée doit s'occuper entièrement de l'initialisation de l'objet

• Il peut toutefois faire appel au constructeur de la classe mère, à l'aide de l'instruction super.

•Si la classe mère possède (explicitement) un constructeur sans paramètres, la classe fille peut ne pas faire appel au constructeur par le mot clé super.

•En revanche, si la classe mère possède seulement des constructeurs avec paramètres,

la classe fille doit faire appel explicitement à l'un de ses constructeurs au moyen du mot clé super.

•Si utilisée, super doit toujours être la première instruction du constructeur de la classe dérivée.

Constructeur & Héritage

Quand une classe Mère admet un constructeur avec paramètre(s), toutes ses classes filles doivent avoir au moins un constructeur. Et chaque constructeur d'une classe fille doit appeler le constructeur de la mère avec super comme première instruction.


```
public  Class A{
int x;
}
```

```
public  Class B extends
A{
int y;
}
```

```
public  Class A{
int x;
public A(int
x){this.x=x;}
}
```

```
public  Class B extends
A{
int y;
}
```

```
public  Class A{
int x;
public A(int
x){this.x=x;}
}
```

```
public  Class B extends
A{
int y;

public B(int x,int y){
Super(x); this.y=y;
}}
```

Exemple

```
class Point{
private double x,y ;
Point (double x, double y){
this.x=x ; this.y=y;}
public double getX(){ return
x);
public double getY(){ return
y);
public void deplacer( double
xx, double yy){
x+=xx ; y+=yy ;}
```

```
class PointEspace  extends
Point{ private double z;
PointEspace(double x, double
y, double z ){super(x,y);
this.z=z;}
public double setZ(double z){
this.z=z;);
public double getZ(){ return
z;};
} public void deplacer( double
xx, double yy, double zz){
super.deplace();z+=zz  ;}    }
```

Visibilité (protected)

- Protégé « protected », il est accessible de toute classe du même package et de toutes ses classes dérivées même si elles appartiennent à d'autres packages (à condition que la classe de base soit public).

Modificateur	private	Défaut	protected	public
La classe elle-même	OUI	OUI	OUI	OUI
Sous classe et même package	NON	OUI	OUI	OUI
Non sous classe et même package	NON	OUI	OUI	OUI
Sous classe et package ≠	NON	NON	OUI	OUI
Non Sous classe et package≠	NON	NON	NON	OUI

Méthode toString de la Classe Object

La méthode toString est définie dans la classe Object ; en conséquence toutes les classes Java en hérite.
La méthode toString définie dans la classeObject ne fait pas grand-chose :
elle renvoie le nom de la classe de l'objet concerné suivi de l'adresse de cet objet.
Lorsqu'on définit une classe, il peut être très utile de redéfinir la méthode toString afin de donner une description satisfaisante des objets de cette classe.
Beaucoup de classes de l'API redéfinissent la méthode toString.

Redéfinition des certaines méthodes de Object

Dans la classe Personne, on peut redéfinir toString de la manière suivante:

```
public String toString{
return  code+’’,’’+  nom+’’,’’+ prénom;
}
```

Avant la redéfinition de toString()

```
Personne p=new
Personne(‘’Akli’’,  ’’Amel ’’);

System.out.println(p); affiche
Personne@1234
```

Après la redéfinition de toString()

```
Personne p=new
Personne(‘’Akli’’, ‘’Amel’’);

System.out.println(p); affiche 45
Akli Amel
```

Dans toute classe où on définit la méthode toString(), la méthode afficher devient comme suit ;

```
public void afficher() {
System.out.println( this);
}
```

Opérateur instanceof

```
class B{ …}
class D extends B{…}
 class C {…}
```



```
 B b = new B();
D d = new D();
 C c = new C();
 b instanceof B // true
b instanceof D // false
 d instanceof B // true
d instanceof D // true
b = d;
b instanceof B // true
b instanceof D // true
c instanceof B //erreur de compilation
```

La redéfinition de la méthode equals de Object

On la redéfinit pour la classe Point

```
public boolean equals( Object  O){
if( O == nul) return false;
If ( ! (O instanceOf(Point)) return false;
Point P = (Point) O;
 return x==  P.x && y == P.y;
}
```

Les Interfaces Graphiques

- Le package Java AWT (Abstract Window Toolkit) est un des premiers package java dédié aux interfaces graphiques (GUIs).
- Le package Swing est une version améliorée de AWT
  - Toutefois, il ne remplace pas entièrement AWT.
  - Certaines classes AWT sont nécessaires quand on utilise Swing.

- Une *GUI (graphical user interface)* est un système de fenêtres, intéragissant avec l'utilisateur.

- Swing GUIs sont basés sur une forme de programmation orientées objets, dirigée par évènement.

Une Fenêtre Simple

- Elle peut être un objet de la classe **JFrame**
  - Un objet **JFrame** comporte une bordure et trois boutons pour réduire, changer la taille de la fenêtre ou fermer la fenêtre.
  - La classe **JFrame** se trouve dans le package **javax.swing**

    JFrame Fenetre = new JFrame();

- On peut ajouter à un objet **JFrame** des composants, tels que des boutons, menus, texte ou des étiquettes
  - Ces composants sont ajoutés à l'objet **JFrame :**

    Fenetre.add(……);
  - L'objet **JFrame** est rendu visible en ajoutant la méthode **setVisible**

    firstWindow.setVisible(true);

Quelques Méthodes de la classe JFrame

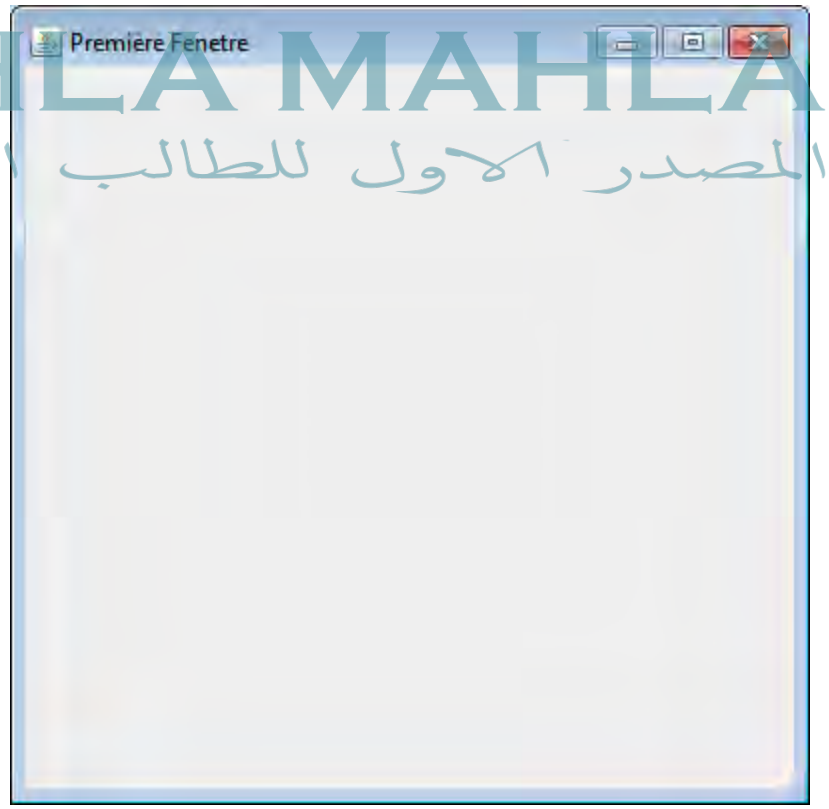
La classe JFrame possède deux constructeurs

public JFrame()
Constructeur qui crée un objet de la classe JFrame

public JFrame(String Title)
Constructeur qui crée un objet de la classe Jframe avec Title comme titre

```
Import  javax.swing.JFrame;
public class  Fenetre extends JFrame
{


public static void main(String[] args) {
Fenetre f=new Fenetre();
f.setSize(400, 400);
}}

}}
```

```
import  javax.swing.JFrame;


public class Fenetre extends JFrame {

public Fenetre(String titre)
    {     super(titre); }


public static void main(String[] args) {
Fenetre f=new Fenetre("Première Fenetre");
f.setSize(400, 400);
  f.setVisible(true);


 }}
```



Boutons

- Un objet bouton est crée à l'aide de la classe **JButton** et peut être ajouté à un objet **JFrame**

- L'argument du constructeur de **JButton** est une chaine de caractères qui apparait sur le bouton.

```
JButton B = new JButton("Quitter");
add(B);
```


```
import javax.swing.JButton;
import javax.swing.JFrame;


publiic class Fenetre extends JFrame {
JButton bt;
public Fenetre(String titre)
   {    super(titre);
bt=new JButton("Quittez");
     add(bt);




   }

public static void main(String[] args) {
Fenetre f=new Fenetre("Première Fenetre");
f.setSize(400, 400);
f.setVisible(true);

}}
```

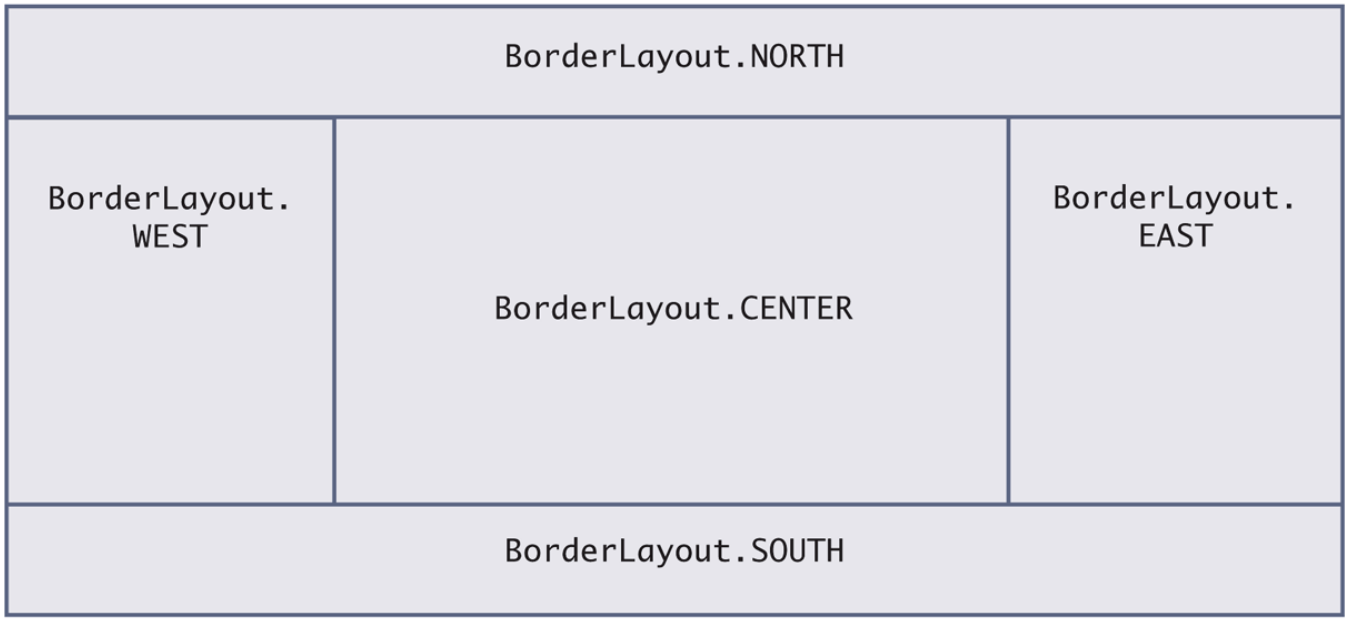

```
import javax.swing.JButton;
import javax.swing.JFrame;


public class Fenetre extends JFrame {
JButton bt1,bt2;
public Fenetre(String titre)
   {    super(titre);

bt1 =new JButton("Quittez");
bt2=new JButton("Validez");
add(bt1);
add(bt2);



   }

public static void main(String[] args) {
Fenetre f=new Fenetre("Première Fenetre");
f.setSize(400, 400);
f.setVisible(true);

}}
```

Containers et Layout Managers

- Plusieurs composants peuvent être ajoutés à **JFrame** utilisant la méthode **add.**
  - Toutefois, la méthode **add** ne spécifie comment ces composants sont placés sur la fenêtre
- Pour décrire comment plusieurs composants peuvent être placés, un gestionnaire (*layout manager*) est utilisé.
- Il y a plusieurs classes de layout manager classes telles que **BorderLayout**, **FlowLayout**, et **GridLayout**
  - Si aucun layout manager n'est spécifié, un gestionnaire par défaut est utilisé.

BorderLayout Regions

Display 17.8 BorderLayout Regions



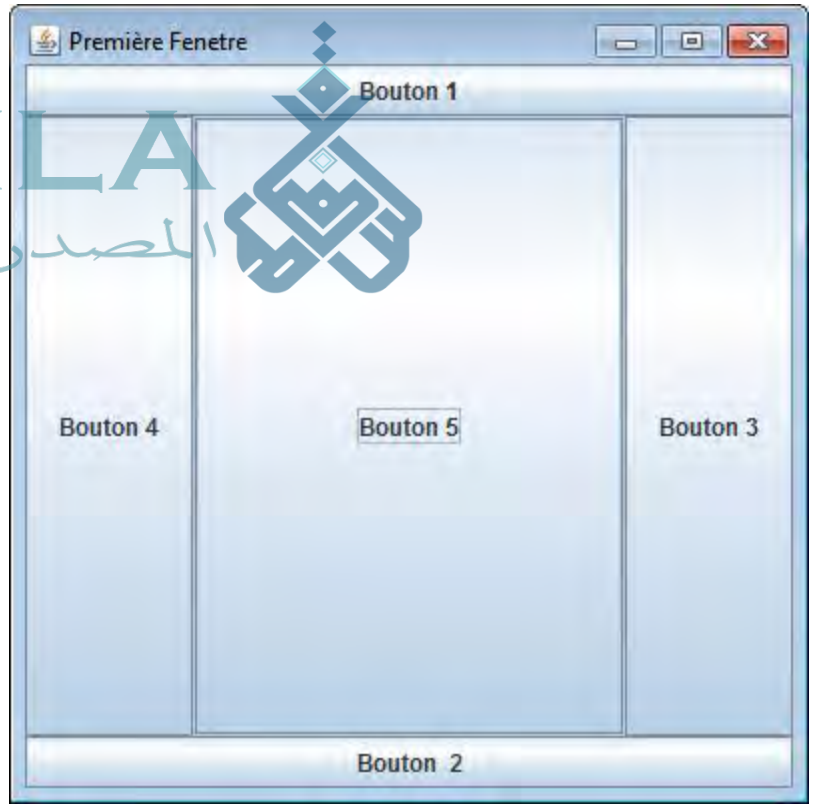
- Un **BorderLayout** gestionnaire place les composants qui peuvent être ajoutés à un objet **JFrame** en 5 régions:

  - Ces régions sont: **BorderLayout.NORTH**, **BorderLayout.SOUTH**, **BorderLayout.EAST**, **BorderLayout.WEST**, et **BorderLayout.Center**

- Un **BorderLayout** gestionnaire est ajouté à un objet **JFrame** en utilsant la méthode **setLayout**
  - Par exemple:

```
setLayout(new BorderLayout());
```

```
import javax.swing.JButton;
import javax.swing.JFrame;

public class Fenetre extends JFrame {
JButton bt1,bt2,bt3,bt4,bt5;
public Fenetre (String titre)
   {    super(titre);
 setLayout(new BorderLayout());
bt1=new JButton("Bouton 1");
bt2=new JButton("Bouton  2");
bt3=new JButton("Bouton 3");
bt4=new JButton("Bouton 4");
bt5=new JButton("Bouton 5");

add(bt1, BorderLayout.NORTH);
add(bt2, BorderLayout.SOUTH);
add(bt3, BorderLayout.EAST);
add(bt4, BorderLayout.WEST);
add(bt5, BorderLayout.CENTER);
  }

public static void main(String[] args) {
Fenetre f=new Fenetre("Première
Fenetre");
 f.setSize(400, 400);
f.setVisible(true);
}}
```

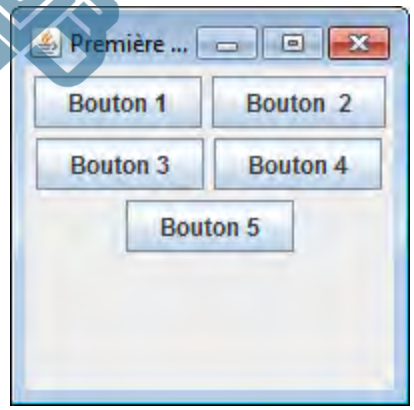
- Un **FlowLayout** gestionnaire est le plus simple layout manager

  setLayout(new FlowLayout());

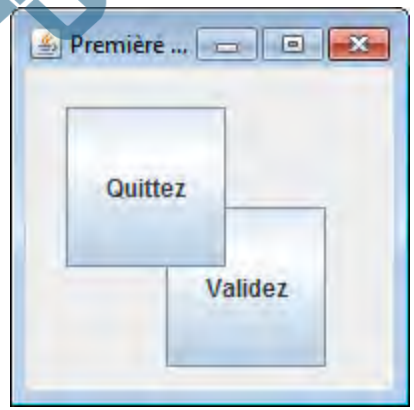
  - Il place les composants un après à la suite de l'autre, partant de la gauche vers la droite.
  - Composants sont placés dans l'ordre dans le lequel ils sont ajoutés.
  - Puisqu'aucune location n'est spécifiée, la méthode **add** n'a qu'à un seul argument quand on utilse **FlowLayoutManager**

    add(…);

```
import java.awt.FlowLayout;
import javax.swing.JButton;
import javax.swing.JFrame;

public class Fenetre extends JFrame {
JButton bt1,bt2,bt3,bt4,bt5;
public Fenetre(String titre)
    {    super(titre);
         setLayout(new FlowLayout());
bt1=new JButton("Bouton 1");
bt2=new JButton("Bouton  2");
bt3=new JButton("Bouton 3");
bt4=new JButton("Bouton 4");
bt5=new JButton("Bouton 5");
add(bt1);
add(bt2);add(bt3);add(bt4);add(bt5);

    }
public static void main(String[] args) {
Fenetre f=new Fenetre("Première Fenetre");
f.setSize(200, 200);
f.setVisible(true);}}
```

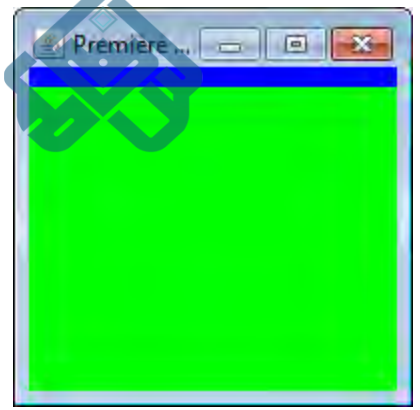
```
import javax.swing.JButton;
import javax.swing.JFrame;


public class Fenetre extends JFrame {
Jbutton bt1, bt2;
public Fenetre(String t)
   {   super(t);
       setLayout(null);
bt1=new JButton("Quittez");
bt1.setBounds(20, 20, 80, 80);
bt2=new JButton("Validez");
bt2.setBounds(70, 70, 80, 80);
add(bt1);   add(bt2);

   }



public static void main(String[] args) {
Fenetre f=new Fenetre(‘’ "Première Fenetre");
f.setSize(200, 200);
   f.setVisible(true);}}
```

Panels

- Un panel est un objet de la classe **`JPanel`** classe qui est un simple conteneur
  - Il permet de regrouper plusieurs composants
  - Il permet de diviser une JFrame en plusieurs sous-régions, d'associer un gestionnaire distinct à chaque panel, une apparence graphique différente, ….

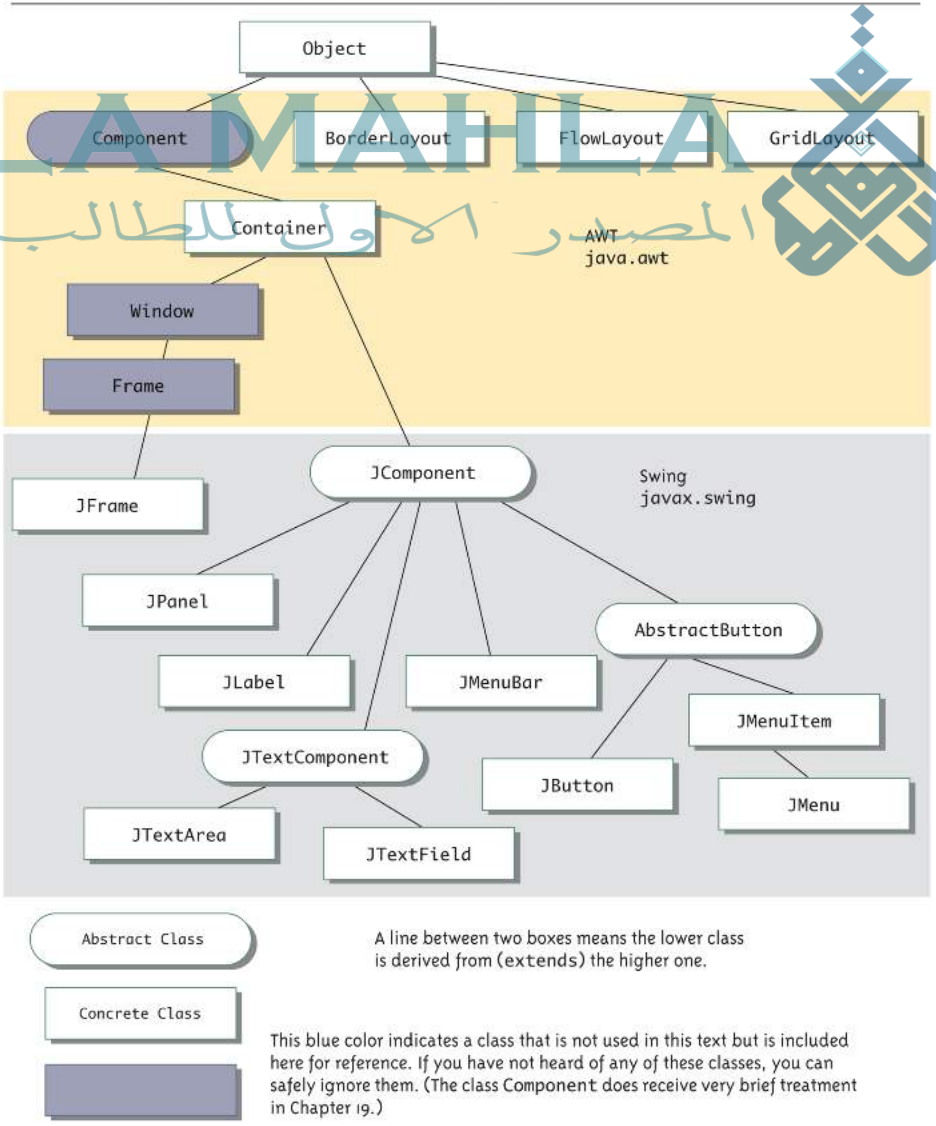
```
import java.awt.BorderLayout;
import java.awt.Color;


import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;


public class Fenetre extends JFrame {
Jpanel panel1,panel2;
public Fenetre(String titre)
    {     super(titre);
          setLayout(new BorderLayout());
 panel1 =new JPanel();
 panel2=new  JPanel();

  add(panel1, BorderLayout.NORTH);
  add(panel2, BorderLayout.CENTER);

  panel1.setBackground(Color.BLUE);
  panel2.setBackground(Color.GREEN);

}

public static void main(String[] args) {

Fenetre f=new Fenetre("Première Fenetre";

f.setSize(200, 200);
f.setVisible(true);
                    }

}
```

```
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.FlowLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;

public class Fenetre extends JFrame {
JPanel panel1,panel2;
public Fenetre (String titre)
   {    super(titre);
      setLayout(new BorderLayout());

panel1 =new JPanel();
 panel2=new  JPanel();
      add(panel1, BorderLayout.NORTH);
      add(panel2, BorderLayout.CENTER);

      panel1.setBackground(Color.BLUE);
      panel2.setBackground(Color.GREEN);

panel1.setLayout(new FlowLayout());
 panel2.setLayout(new FlowLayout());

     panel1.add(new JButton("Un"));
      panel1.add(new JButton("Deux"));
      panel2.add(new JButton("Trois"));
      panel2.add(new JButton("Quatre"));
                  }
public static void main(String[] args) {
PremiereWindow f=new PremiereWindow("Première Fenetre");
setSize(400,120);
   setVisible(true);}}
```

Hierarchy of Swing and AWT Classes

Display 17.12 **Hierarchy of Swing and AWT Classes**



Fenetre

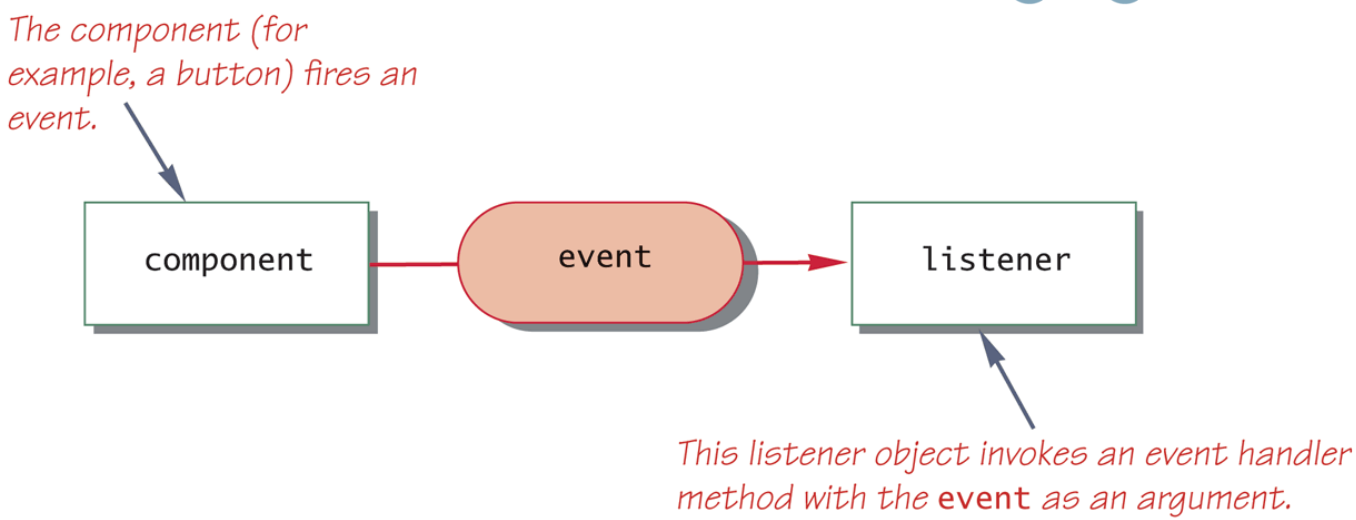
Gestion des évènements

Un évènement peut être généré par:

- Une action de l'utilisateur (pression d'une touche, clic sur un JButton)
- Par le programme (ex : horloge, compteur).

Gestion des évènements

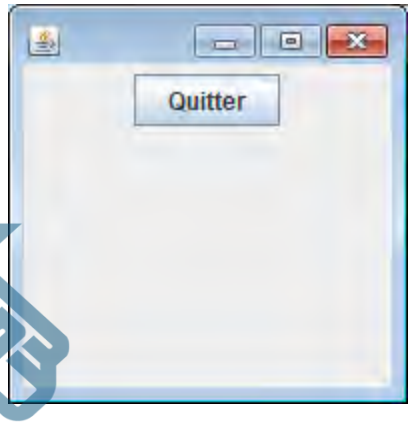
Display 17.1 Event Firing and an Event Listener



```
public interface ActionListener{
// interface java prédéfinie
public void actionPerformed(ActionEvent e);
}
```

```
// Ecouteur est une classe d'objets listener
// ils permettent d'intercepter les évènements
// tels que click sur un bouton.

class Ecouteur implements ActionListener{
public void actionPerformed(ActionEvent e) {
                    System.exit(0);}}

public class Example1 extends JFrame{
JButton b=new JButton("Quitter");
public Example1()
{setVisible(true);setSize(200,200);
setLayout(new FlowLayout());
add(b);

Ecouteur c=new Ecouteur();
b.addActionListener(c);}

public static void main(String[] arg){
Example1 f=new Example1();}}
```

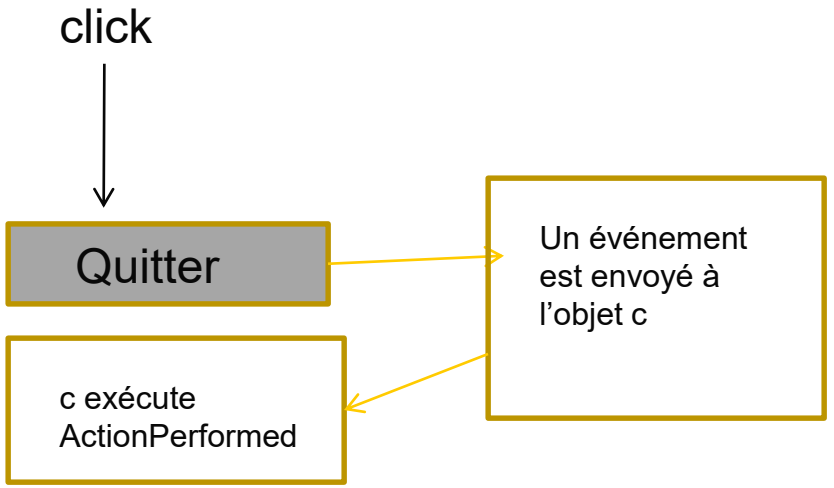

Première Solution




```
....
class Ecouteur implements ActionListener{
public void actionPerformed(ActionEvent e) {
                System.exit(0);}}
```

```
public class Example1 extends JFrame{
JButton b=new JButton("Quitter");
public Example1()
{setVisible(true);setSize(200,200);
setLayout(new FlowLayout());
add(b);

Ecouteur c=new Ecouteur();
b.addActionListener(c);}

public static void main(String[] arg){
Example1 f=new Example1();}}
```

c est
un objet Ecouteur
Enregistré au
sein du bouton b

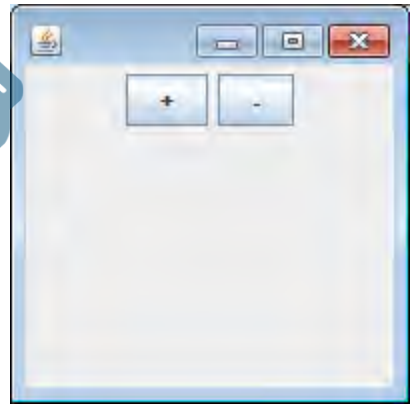


Deuxième Solution


```
....
public class Example1 extends JFrame implements
ActionListener{
JButton b=new JButton("Quitter");


Example1()

{setVisible(true);setSize(200,200);
setLayout(new FlowLayout());

add(b);
;
b.addActionListener(this);}

public static void main(String[] arg)
{Example1 f=new Example1();
                }

@Override
public void actionPerformed(ActionEvent e) {
System.exit(0);
}
}
```

….

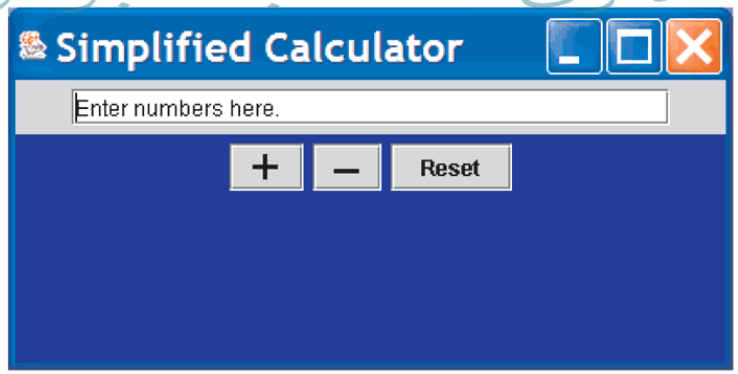
```
public class Example1 extends JFrame implements
ActionListener{
JButton b1,b2;

Example1(){
setLayout(new FlowLayout());
b1=new JButton("+");
b2=new JButton("-");

add(b1); add(b2);

;
b1.addActionListener(this);
b2.addActionListener(this);}

public static void main(String[] arg)
{Example1 f=new Example1();
setSize(200,200); setVisible(true);}

@Override
public void actionPerformed(ActionEvent e) {
Object source =      e.getSource();
If( source.equals(b1)) System.out.println("+");
else System.out.println("-");
}
}
```

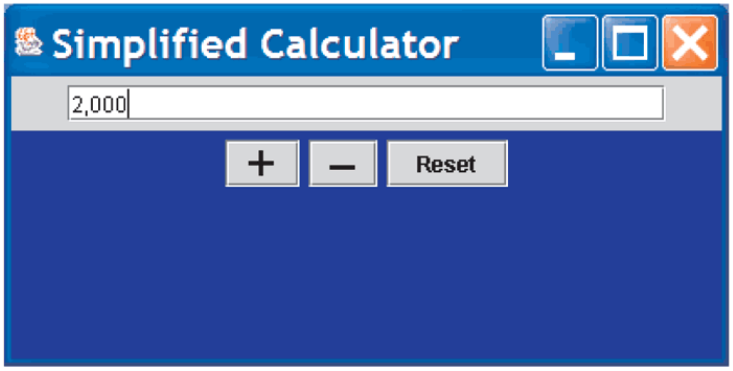
Rélaiser une calculatrice simple

Display 17.19 **A Simple Calculator**

RESULTING GUI (When started)



RESULTING GUI (After entering 2,000)

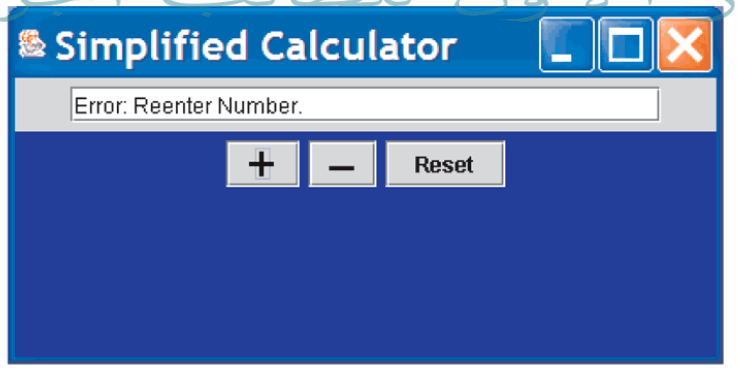


(continued)

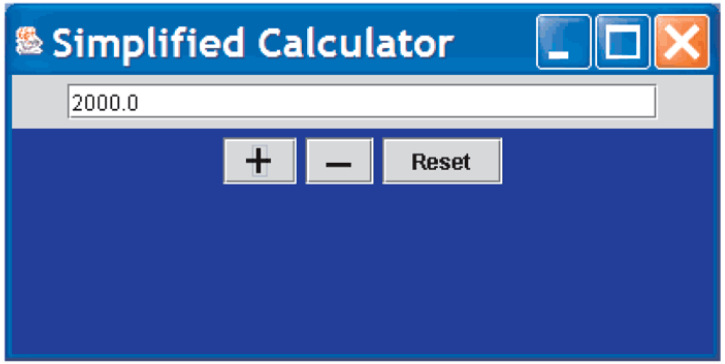
A Simple Calculator (Part 10 of 11)

Display 17.19 A Simple Calculator

RESULTING GUI (After clicking +)



RESULTING GUI (After entering 2000 and clicking +)

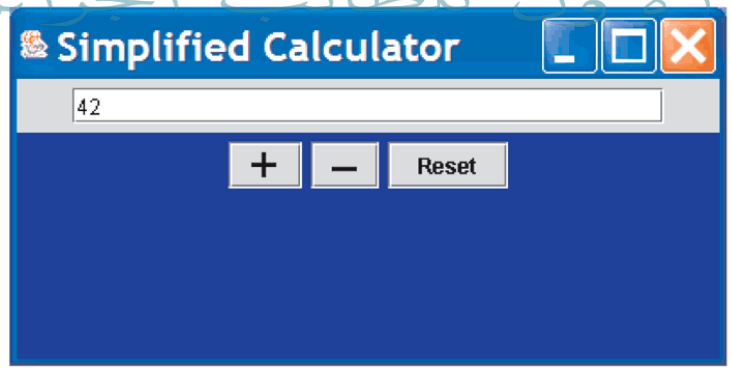


(continued)

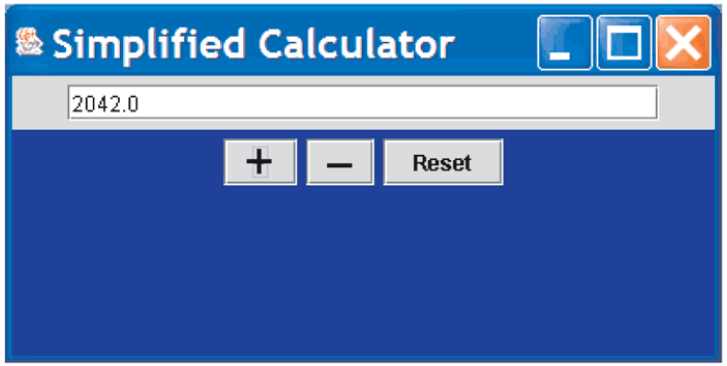
A Simple Calculator (Part 11 of 11)

Display 17.19 **A Simple Calculator**

RESULTING GUI (After entering 42)



RESULTING GUI (After clicking +)



Révision Héritage

Exercice 1

```
public class A{
  int x,y,z ;
      public A(int x,int y,int z){this.x=x; this.x=y; this.x=z;}
      public incX(){x++ ;}
      public incY(){y++ ;}
      public incZ(){z++ ;}
```

```
public class B  extends A  {
 int x,y,z ;
public B(int x,int y,int z)
{
Super(x, y ,z);
}
  public incYZ(){y++ ;z++;}
```

```
public class C extends A {
int w,h ;
 public C(int x,int y,int z,int
w,int h)
{Super(x, y ,z);
This.w=w;
This.h=h}
public incYZ(){y++ ;w++;}
public incH(){h++ ;}
```

Révision Héritage

- **Redéfinir les classes A, B et C en appliquant les principes d'héritage et d'encapsulation.**

- **Considérons les objets A a , B b et C c. Créer ces objets.**

```
a = new A (1,2,3);
b = new B (-1,-2,-3);
c = new C (0,1,2,3,4);
```

- **Quels sont les membres des objets a, b et c.**

- **Quels sont les types de chacun de ces objets.**

a Object et A.
b Object , A et B.
c Object , A et C.

Révision Héritage

Exercice n°2

Ecrire la classe Point de l'espace. Prévoir les attributs x, y, z et couleur.

Ecrire les getteurs et setteurs et les méthodes affiche, symetrie et egal.

Révision Héritage

Exercice n°3 **Dérouler ce code:**

```
public class A {
 int f(){return 0;}

public static void
main(String[] args)
 {

B b=new B();
System.out.println(b.g(4));
}}
```

```
class B extends A{
int f(){return 1;}

int g(int x)
{if (x==0) return super.f();
if (x==1) return this.f();
return g(x-1)+g(x-2);}

}
```

Déroulez en enlevant le this et super.

Types statique et dynamique d'un objet

- Un objet a deux liaisons, statique et dynamique. La liaison statique (type statique) correspond au type de la classe de la déclaration d'objet. Une fois définie, elle ne peut pas changer.

- Par contre la liaison dynamique (type dynamique) peut changer au cours du programme et correspond au type de la classe d'instanciation

Types statique et dynamique d'un objet

Exemple:

p1 a le type statique **Personne**

Type de déclaration ou statique (immuable)

Personne p1;

p1= new *Féminin* (« c1», « Amina», ……);

p1 a le type **dynamique** *Féminin*

Type d' instanciation ou dynamique (variable)

Types statique et dynamique d'un objet

Exemple:

Personne p2;

p2 a le type statique **Personne**

Type de déclaration ou statique (immuable)

p2= new *Masculin* (« c1», « Amine», ……);

p2 a le type **dynamique Masculin**

Type d' instanciation ou dynamique (variable)

Compilation & Type statique

p1.

Seules les membres de la classe Personne (type statique) (et bien évidemment de ses classes ascendantes) apparaissent.

Compilation & Type statique

p1.

Getnom()

GetAdresse()

~~GetEpoux()~~

Exécution & Type dynamique

p1.sexe()

La liaison dynamique est utilisée à l'exécution pour déterminer le comportement effectif à exécuter.

.

ATTENTION: la méthode sexe() doit être définie dans la classe Personne.

Exécution & Type dynamique

Personne p1=new Féminin(….);

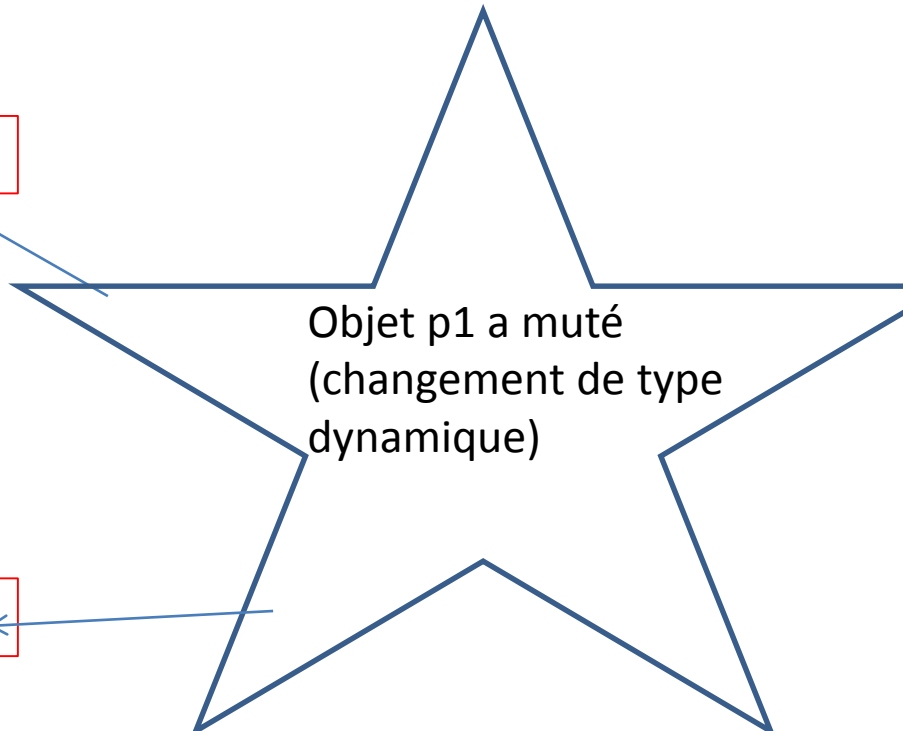
Personne p2=new Masculin(….);

p2.sexe() retoune ‘m’

p1.sexe() retoune ‘f’

p1= new Masulin(….);

p1.sexe() retoune ‘m’



```
public class A{
public voidf(){ System.out.println("Je suis dans A");}
}
public class B extends A { int x,y;
public voidf(){ System.out.println("Je suis dans B");}
}
public class C extends B { int z;
public voidf(){ System.out.println("Je suis dans C");}
}

/*
 tab est un tableau de Type statique A.
 tab[0] a le Type dynamique A.
 tab[1] a le Type dynamique B.
 tab[2] a le Type dynamique C.  */

 tab[1].
```

F
(one ne voit que les membres de la classe A)

```
public class Test{
public static void main(String[]
args)
{A [ ] tab =new A[3];
 tab[0]=new A();
 tab [1]=new B();
 tab[2]=new C();
 tab [0].f();
//  affiche Je suis dans A
 tab[1].f();
// affiche Je suis dans B
 tab[2].f();
// affiche Je suis dans C
tab[0]=tab[1];
tab[0].f() ;
// affiche  Je suis dans B
} }
```

Polymorphisme

- Le nom de *polymorphisme* vient du grec et signifie *qui peut prendre plusieurs formes*. Cette caractéristique est un des concepts essentiels de la programmation orientée objet. Alors que l'héritage concerne les classes (et leur hiérarchie), le polymorphisme est relatif aux méthodes des objets.

- le choix du code à exécuter (pour une méthode polymorphe) ne se fait pas statiquement à la compilation mais dynamiquement à l'exécution.

- Le polymorphisme peut être vu comme la capacité de choisir dynamiquement la méthode qui correspond au type réel de l'objet.

- Une méthode est polymorphe si:

Polymorphisme

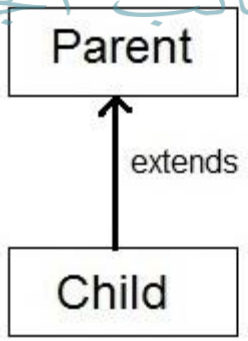
Le polymorphisme concerne :

- L'héritage
- Les méthodes seulement (pas les attributs).
- Les méthodes d'instances seulement (pas les statiques)
- Les méthodes redéfinies seulement (pas les surchargés)

L’opérateur *instanceOf*

In Java, instanceof operator is used to check the type of an object at runtime. It is the means by which your program can obtain run-time type information about an object. instanceof operator is also important in case of casting object at runtime. instanceof operator return boolean value, if an object reference is of specified type then it return **true** otherwise **false**.

Types statique et dynamique




```
Parent p = new Child( );
```

Upcasting

```
Child c = new Parent( );
```

Compile time error

```
Child c = ( Child ) new Parent( );
```

Downcasting but throws ClassCastException at runtime.

```
class Parent{}

class Child1 extends Parent{}

class Child2 extends Parent{}

class Test{  public static void main(String[] args)
 {     Parent p =new Parent();     Child1 c1 = new Child1();
 Child2 c2 = new Child2();

 System.out.println(c1 instanceof Parent);              //true
System.out.println(c2 instanceof Parent);               //true
System.out.println(p instanceof Child1);                //false
System.out.println(p instanceof Child2);                //false
    p = c1;      System.out.println(p instanceof Child1);          //true
System.out.println(p instanceof Child2);                //false
  p = c2;      System.out.println(p instanceof Child1); //false
System.out.println(p instanceof Child2);                //true      }   }
```

```
class Test{  public static void main(String[] args)
 {     Parent p =new Parent();      Child1 c1 = new Child1();
 Child2 c2 = new Child2();

c1=p; // erreur
c1=(Child1) p //correct mais une erreur d'exécution peut apparaitre
si p n'a  pas un type dynamique Child1.
c1=c2// erreur


}   }
```

Classes Abstraites

- Une classe est dite abstraite si elle ne peut être instanciée (ne peut produire d'instance). En java elle est définie par le mot réservé abstract.

- Soient les classes **Masculin** et **Féminin**. On peut factoriser les attributs et méthodes communs des deux classes, on obtient donc la super classe **Personne**.

- Comme On ne peut avoir une personne qui ne soit ni masculin ni féminin, la classe mère Personne ne doit pas donner la possibilité de créer des objets et doit donc être abstraite

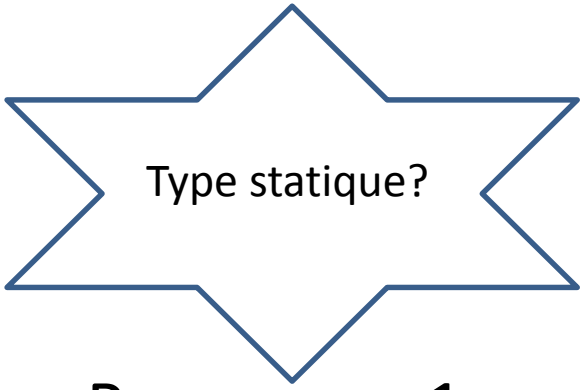
Exemple de classe abstraite

```
public abstract class Personne{
private String nom, prenom, adresse, lieuNais ;
Date dateNais ;
……..
public Personne(String n, String p, String
ad,Date dN String lN){
   nom = n ; prenom= p ; adresse = ad,
dateNais= dN, lieuNais = lN ;}

……..}

}
```

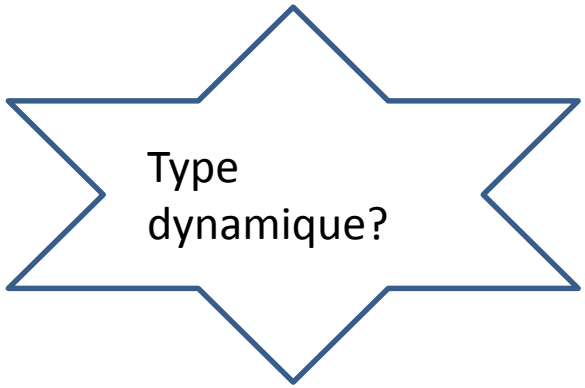
```
public class Masculin extends Personne {
…….
public Masculin (String n, String p,
String ad,Date dN String lN)){
super (n, p, ad, dN, lN) ;}
}

public class Feminin extends Personne {
…….
public Feminin (String n, String p,
String ad,Date dN String lN)){
super (n, p, ad, dN, lN) ;}

}
```

Exemple de classe abstraite

Personne p1 = new Personne(….);



Personne p1 = new Feminin(….); oui



Méthode abstraite

- On peut aussi définir des méthodes abstraites. Elles sont définies seulement par leurs signatures.

- Une méthode abstraite est appelée à être redéfinie.

Vers les Méthodes abstraites


```
public abstract class Personne {
....
// pas de méthode sexe
}
```

```
public Feminin extends Personne
{....
public char sexe(){
return 'f';
}}
```

```
public Masculin extends Personne
{....
public char sexe(){
return 'm';
}}
```

Dans le main, on considère:

```
Personne enfants =new Personne[2];
enfants[0]=new Feminin(.....); enfants[1]=new Masculin(.....);
for (int i=0; i<enfants.length;i++)
s.o.p(enfants[i].sexe());
```

Erreur de compilation

Vers les Méthodes abstraites

```
public abstract class Personne {

....
Public char sexe(){
return '?';
}
}
```

Pas très élégant


```
public Feminin extends Personne
{....
public char sexe(){
return 'f';
}}
```

```
public Masculin extends Personne
{....
public char sexe(){
return 'm';
}}
```

Dans le main, on considère:

```
Personne enfants =new Personne[2];
enfants[0]=new Feminin(.....); enfants[1]=new Masculin(.....);
for (int i=0; i<enfants.length;i++)
s.o.p(enfants[0].sexe());
```

correct

Méthode abstraite

```
public abstract class Personne {

....
public abstract char sexe();

}
```

Enfin la bonne soultion


```
public Feminin extends Personne
{....
public char sexe(){
return 'f';
}}
```

```
public Masculin extends Personne
{....
public char sexe(){
return 'm';
}}
```

Dans le main, on considère:

```
Personne enfants =new Personne[2];
enfants[0]=new Feminin(.....); enfants[1]=new Masculin(.....);
for (int i=0; i<enfants.length;i++)
s.o.p(enfants[0].sexe());
```

correct

Interface Java

Les interfaces

- les *interfaces* sont des « sorte de classes'' qui ne possèdent que des **champs static final** (autant dire des constantes) et que des **méthodes abstraites.**

Elles ne possèden**t** donc ni variables d'instances ni méthodes concrètes (elles sont toutes abstraites).

- En fait, les interfaces sont un moyen de préciser les services qu'une classe peut rendre.
- On dira qu'une classe **implante une interface XX** si cette classe fournit les implantations des méthodes déclarées dans l'interface XX (elle donne une existence réelle à ces méthodes)

Déclaration des interfaces

Comme les classes, les interfaces sont constituées de champs et de méthodes ; mais :

- **Toutes les méthodes** qui sont déclarées dans cette interface sont **implicitement abstraites** (le mot clé abstact est facultatif) ;

- **Toutes les méthodes** d'une interface sont implicitement **publiques et non statiques (** le modificateur public est facultatif**).**

- **Tous les champs** d'une interface sont **implicitement public, static et final**. Ils sont là pour définir des constantes. Les mots clés static et final ne figurent pas dans la définition des champs ; ils sont implicites.

Déclaration d'une interface

```
public interface XX{
public final static int MAX=1024;
public abstract void f();

}
```

Ou

```
 interface XX{
  int MAX=1024;
  void f();
}
```

Implanter une interface

```
public class C implements XX{
……
public void  f(){

for (int i=0; i<MAX;i++)…….
……
}
…….
}
```

- **Les interfaces définissent des**

``*promesses de services*'' mais seule Les classes peuvent rendre effectivement les services qu'une interface promet.

- **La classe C doit donc redéfinir toutes les méthodes précisées dans l'interface**

Utiliser les interfaces

Une référence de type interface peut contenir:

- ✓ null
- ✓ une instance d'une classe qui doit impérativement implanter l'interface.

Exemple

```
class C implements XX{}
class D  implements XX{}
XX s1, s2, s3;
s1=null;
S2=new C();
s3= new D();
```

Ainsi Le polymorphisme peut être appliqué

Application

```
interface Représentable {
 void dessiner();
 void tourner(int angle);
 void translater(int depl);    }
```



```
public class abstract Figure implements
Représentable{
public void tourner(int angle){….}
}

public  Carre extends Figure{…….
public void dessiner(){ ……..}
public void translater(int depl){ ……..}
```

```
public  Cercle extends Figure{…….
public void dessiner(){ ……..}
public void translater(int depl){ ……..}
}

// main

// déclarer un tableau de formes
géométriques
Représentable[] tab= new Représentable[3];
Tab[0]= new Cercle(….);
Tab[1]= new Carre(….);  int i = sc.nextInt();
……Tab[i]. Dessiner();
```

Remarque :

On peut implémenter plusieurs interfaces

class C implements I1,I2,I3{…..}

Il existe des Interfaces prédéfinies de java tel que Clonable, Serialiseable, Comparable, Obdserver, ActionListener…

La classe Arrarys est associé aux tableaux

Elle possède une méthode statique sort permettant de trier les éléments du tableau.

Int [] T = { 4,2,7,1};

Arrays.sort(T); / le tableau T sera trié 1, 2, 4, 7

Pour pouvoir trier un tableau d'objet il faut que la classe correspondante implémente l'interface Comparable

```
Interface   Comparable {
int  CompareTo (Object O) ;
 }
```


Pour trier un tableau d'objets

1. Il faut que la classe d'objets implémente l'interface Comparable
2. Il faut que la classe d'objets implémente la méthode compareTo
3. On invoque la méthode static sort de la classe Arrays

```
class  Couple implements Comparable{
private int x,y
@Override
public int compareTo(Object o) {
Point Couple   C= (Couple)o;

int a= (int) (x-C.x) ;
 int b =(int) (y- C.y);

 if(a!= 0) return a; return b;
}}
```

```
// main
int n = sc.nextInt();
Couple [] tabC = new  Couple[n];
// Remplissage du tableau
for ( int i = 0; i++)
tabC[i] = new Couple(sc.nextInt(),
sc.nextInt());
// tri du tableau
Arrays.sort(tabC);
```

L’interface ActionListener


```
public Interface ActionListener{
// interface java prédéfinie
public void actionPerformed(ActionEvent e);}
```

```
class Ecouteur implements ActionListener{
public void actionPerformed(ActionEvent e) {
                    System.exit(0);}}
```

```
public class Fenetre extends JFrame{
JButton b=new JButton("Quitter");
add(b);

Ecouteur c=new Ecouteur();
b.addActionListener(c);}
```

Remarque : On peut implémenter directement la classe Fenetre

```
public class Fenetre extends JFrame implements
ActionListener{
JButton b=new JButton("Quitter");
b.addActionListener(this);
@Override
public void actionPerformed(ActionEvent e) {
System.exit(0);
}}
```

L’interface Observer


```
class Capteur implements Observer {

@Override
public void update(Observable O, Object arg1) {
Machine m=(Machine)O;
if (m.getTemperature() > 40 ){
System.out.println("La machine "+m.getNom()+"
a chauffé");
}}}
```

```
public class Machine extends Observable {
private int temperature=25;
private String nom;
public String getNom() { return nom; }

public void setNom(String nom) { this.nom = nom; }

public Machine(String nom) { super(); this.nom = nom;
}
public int getTemperature() { return temperature; }

public void setTemperature(int temperature) {
this.temperature = temperature;
setChanged();
notifyObservers();
}
```

```
public static void main(String[] args) {
Machine m1, m2;
m1=new Machine("Astra");
m2=new Machine("Alpha");
m1.setTemperature(50);
m1.setTemperature(60);
// jusqu'à présent il ne se passe rien car on n'a pas de capteur
Capteur capt=new Capteur();

m1.addObserver(capt);
m2.addObserver(capt);
m1.setTemperature(50); // il affiche un message pour m1
m2.setTemperature(30); // il n'affiche rien pour m2

m2.setTemperature(50); // il affiche un messa ge pour m2
}
```

Chapitre IV
Collection et Généricité

Généricité

Classe paramétrée

Java offre (comme les langages orienté objets c++, ..) la possibilité de créer des classes paramétrées par des types, elle apporte souplesse et robustesse dans le code vu qu'elle permet de créer une seule classe pour un ensemble de types. Il suffit de rajouter une variable type entre < >

Classe paramétrée

```
Class  Paire  <T> {
private T x ,y;      // T n'est pas connu à la compilation
 Paire  (T x, T y)   { this.x= x ;this.y=y ;}

void afficher( ){System.out.println( ''paire  :''+
x.toString()+ '' ,'' + y.toString()) ;}

T getX(){ return x;}  T getY(){ return y;}

void setX(T xx){ x=xx ;}

void setY(T yy){y=yy ;}
}
```

- Instantiation avec le type Integer

  Paire<**Integer**> p1= new Paire (1,4);

  Affecter l'entier 3 à l'attribut x de p1

  p1.setX(3);

- Instantiation avec le type String

  Paire <**String**> p2= new Paire(''ISILA'','' 2014'');

  Affecter la chaine "ISILB" à l'attribut x de p2

  p2.setX(''SILB'');

- Instantiation avec le type Etudiant

  Etudiant E1 = new Etudiant(…….);

  Etudiant E2 = new Etudiant(…….);

  Paire<Etudiant> binome = new Paire(E1,E2)

Classe paramétrées avec plus d'un type

On peut avoir plusieurs types paramétrées

Class Test < T,P>

{ T x ; P y ;….}

Exemple:

```
public class Pair<T1,T2> { p rivate T1   x; private T2  y;
public T1 getX() {return x;}
public T2 getY() {return y;}
public void setX(T1 a) {x= a;}
public void setY(T2 b) {y= b;}}
```

Instanciations

```
Pair <String, Integer> obj1=
 new Pair();
String i1=obj1.getX();
  int i2=obj1.getY();
```

```
Pair <Integer, String> obj2=
   new Pair()
    int i1=obj2.getX();
   String i2=obj2.getY();
```

Généricité et héritage

On peut étendre une classe générique par une autre classe générique, à condition que les variables de type soient les mêmes.

```
public class B<T> {
T x;
…
}
public class D<T> extends B <T> {
T y; ……
}
```

La généricité a les caractéristiques suivantes :

- A la compilation, lors de l'instanciation d'une classe générique, les paramètres de type sont remplacés par le type spécifié.

- Une seule classe compilée est produite, quelque soit le nombre d'instanciations.

- Les types paramètres ne peuvent être utilisés pour créer des objets,

  T o = new T() **est faux**

Sous classes

- D<String> est une sous classe de B<String>

- D<Integer> est une sous classe de B<Integer>

Mais

- D<Integer> n'est pas une sous classe de B<Object>, bien que Integer soit une classe fille de la classe Object.

Méthode Générique

Une méthode peut être paramétrée par un type, qu'elle soit dans une classe générique ou non. On veut permuter entre deux éléments d un tableau typé

Exemple:

```
public class C{
public static <T> void permute(T[]
tab, int i, int j) {
T temp = tab[i];
tab[i] = tab[j];
tab[j] = temp; }}
```

```
//main
String[] tab1 = new String[]{"toto","titi","tutu"};
Float[] tab2 = new Float[]{3.14f,27.12f};
C.permute(tab1,0,2);  C.permute(tab2,0,1) ;
```

La généricité avec les types contraintes

- Sur les types parametrés on ne peut réaliser les opérateurs ou méthodes de comparaison
- Pour régler ce problème on doit préciser que ce type hérite d'une classe ou d'une ou plusieurs interfaces qui définissent ces méthodes.

public class C <T extends OtherClass & Interface1 & Interface2 >

Exemple : Si on veut comparer entre les attributs de Paire on est obligé d'utiliser la méthode compareTo de Comparable vu qu'elle est valable pour la majorité de classes définies.

- Exemple1 : Méthode de tri générique

```
static <T extends Comparable > T[]  tri(T[] tab){
T z;
for ( int i=0; i<tab.length-1;i++)
for( int j = i+1;j<tab.length;j++)
if (( tab[i]).compareTo( tab[j])>0){
z=tab[i];
tab[i]=tab[j];
tab[j]=z;

} return tab;
}
```

- Exemple2 : Créer un tableau trié de paires d'éléments

```
public class Couple<T extends Comparable>
implements Comparable {
private T x,y;
Couple(T xx,T yy){ x=xx; y=yy;}

@Override
public int compareTo(Object arg0) {
Couple C =(Couple)arg0;
int a = x.compareTo(C.x);
int b = y.compareTo(C.y);
return a!=0 ? a : b;
}}
```

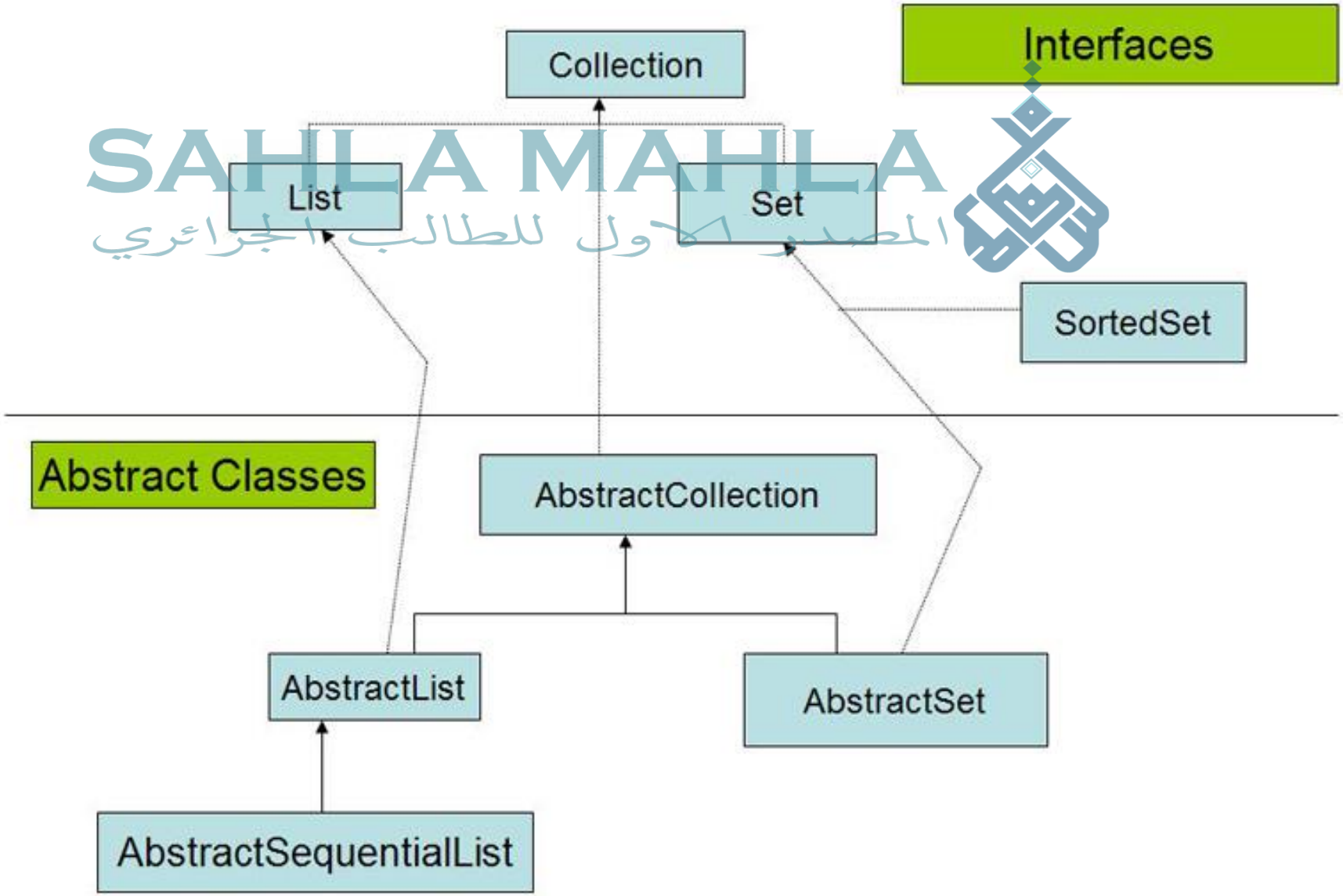
```
public static void main(String[] args){
Scanner e = new Scanner(System.in);
Couple<String>[] Tab = new Couple[5];
for( int i=0;i<5;i++)
Tab[i]= new Couple(e.next(),e.next());
Arrays.sort(Tab);
```

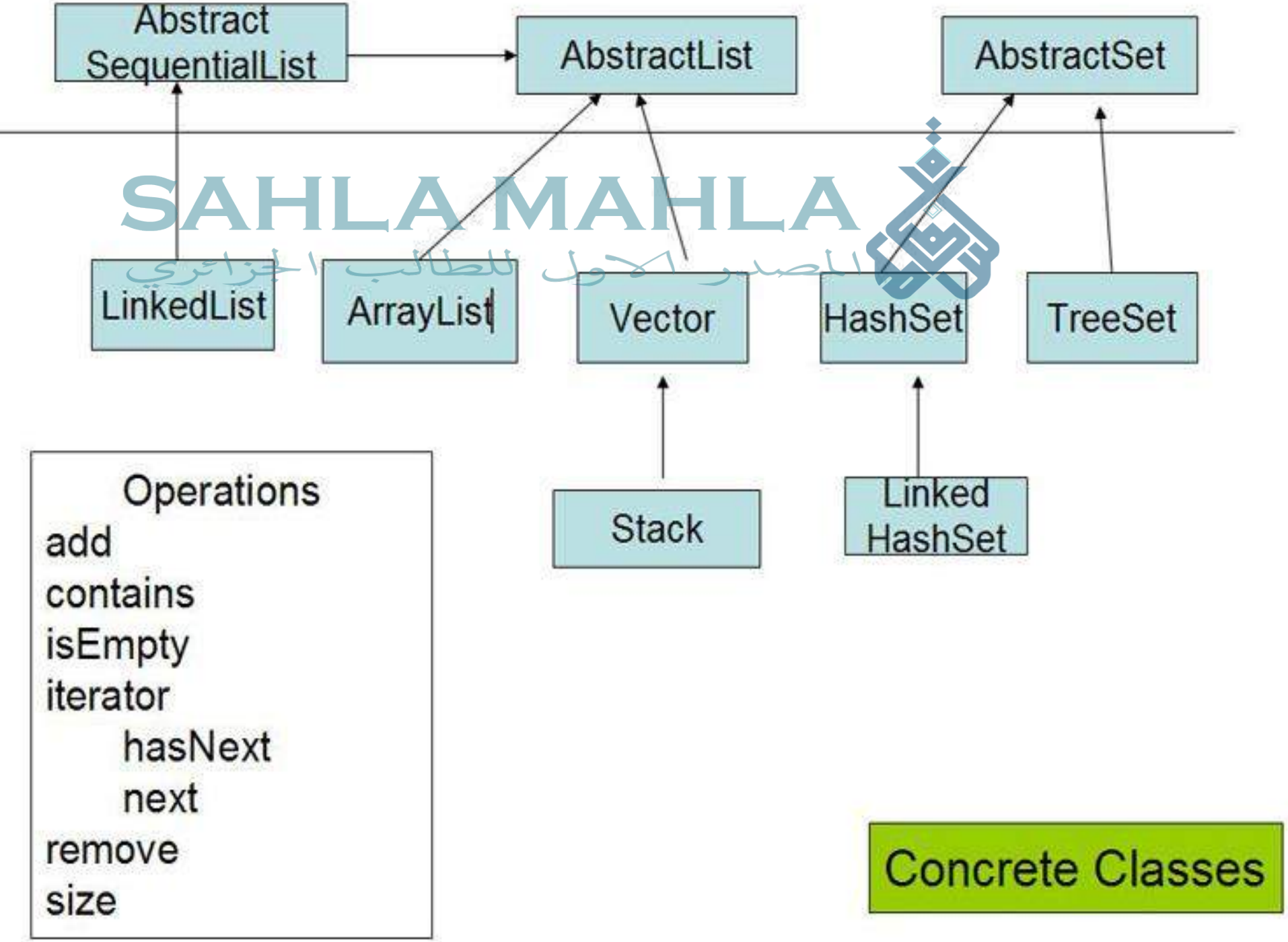

REMARQUE :

Toute classe utilisant cette classe Couple doit implémenter Comparable Exemple un tableau de binôme d'étudiant, la classe étudiant doit implémenter Comparable

Collections

En Java, le terme **collection** est utilisé pour représenter toutes les structures qui regroupent plusieurs objets avec une gestion dynamique de la taille. Il existe plusieurs types de collections. Par exemple list, set map,



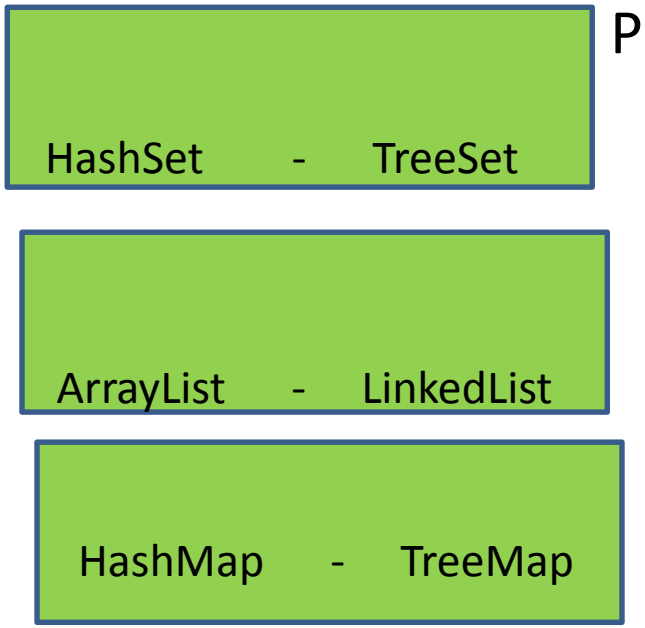


Dans le paquetage java.util, on trouve

– Des interfaces: Collection, List, Set, map

- Des classes abstraites qui implémentent partiellement ces interfaces: AbstractCollection, AbstractList, AbstractMap, AbstractSet, AbstractSequentialList (accès séquentiel)

- Plusieurs implémentations (concrètes)



-

On peut utiliser les classes existantes, ou en concevoir de nouvelles (à intégrer convenablement dans la hiérarchie des classes de collections)

Interface Collection

Principales méthodes

- ✓ boolean add(Object obj)
- ✓ boolean contains(Object obj)
- ✓ boolean containsAll(Collection collection)
- ✓ Iterator()
- ✓ int size();
- ✓……..

- **La classe ArrayList**

Cette classe représente un tableau d'objets dont la taille est dynamique.

- Elle hérite de la classe AbstractList donc elle implémente l'interface List.
- Le fonctionnement de cette classe est identique à celui de la classe Vector.

La différence avec la classe Vector est que cette dernière est multi thread (toutes ses méthodes sont synchronisées).

Pour une utilisation dans un thread unique, la synchronisation des méthodes est inutile et coûteuse. Il est alors préférable d'utiliser un objet de la classe ArrayList.

- On a les méthodes suivantes :
- get pour récupérer un élément
- add pour ajouter un élément (addAll pour rajouter tous les éléments d'une liste
- size pour donner le nombre d'élements
- contains pour tester l'appartenance d'un élément

Exemple 1:

- Déclation et utilisation d'un vecteur dynamique de type ArrayList

- ✓ ArrayList list =new ArrayList();
- ✓ list .add(''Toto'');
- ✓ list .add(3);

On a crée une liste d'objets

String ch=tab.get(0); // faux

- ✓ String ch=(String) list .get(0); // correct

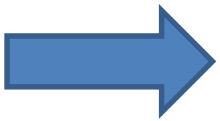
int i= list.get(1); // faux

- ✓ int i= (Integer) list.get(1); // correct

Exemple 2:

- Déclation et utilisation d'un vecteur dynamique de type ArrayList


```
ArrayList<String>  list =new ArrayList();
list.add(''Toto'');
list.add(''Djo'');
 list.add(3);// faux

String ch=list.get(0); // correct
String ch= list.get(1); // correct
```



Le contrôle est automatique

Parcours d’une ArrayList

EXEMPLE D’AFFICHAGE

1. for simple

```
ArrayList<String> list    =   new ArrayList();
……
   for (int i=0; i<list.size();i++)
System.out.println(list.get(i));
```

2. for étendu

```
for (String  e: list)  System.out.println(e);
```

3. Directement Sans le for

System.out.println(list);

A condition que La classe des éléments de list s redéfinisse la méthode toString pour ne pas afficher les adresses

4. En utilisant un itérateur

Interface Iterator

✓ Une interface permettant d'énumérer les éléments contenus dans une collection.

✓ Toutes les collections proposent une méthode itérator renvoyant un itérateur.

Méthodes Iterator

- ✓ boolean hasNext()
- ✓ Object next()

Exemple

```
ArrayList < C> list=new ArrayList();
……
  Iterator  it=list.iterator();
  while  (it.hasnext())
  System.out.println(it.next());
```

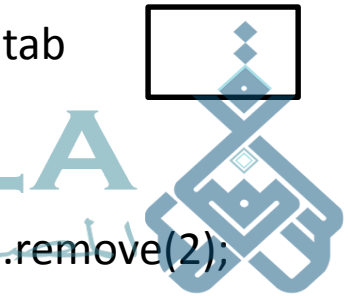
Tableau de personnes

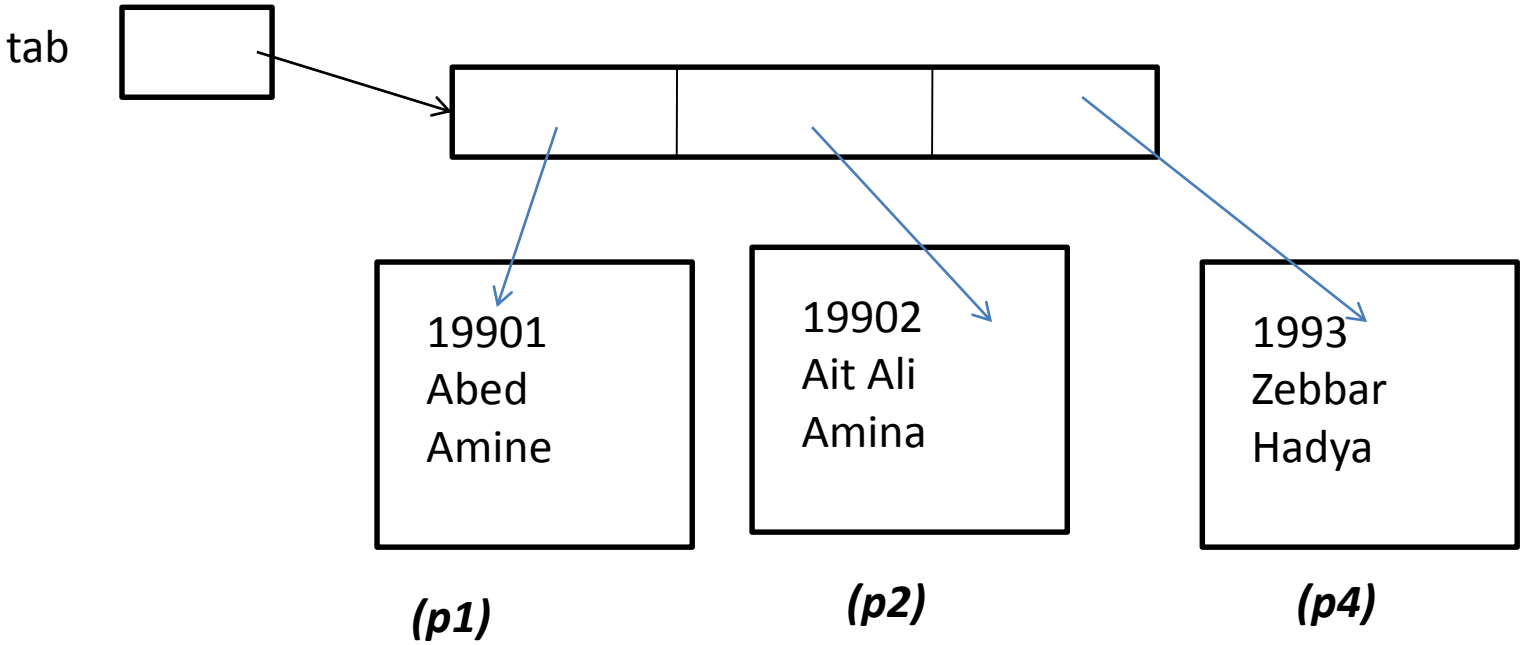
// Déclaration

tab<Personne> list =new ArrayList();

// 4 insertions et une suppression

tab.add(p1); tab.add(p2); tab.add(p3);tab.add(p4);tab.remove(2);





Comment trier une liste

- Pour trier uner liste on opère comme pour un tableau statique, mais au lieu d'utiliser la classe Arrays on utilise la classe Collections
- Collections.sort(list)

Avec la condition que la classe correspondante à list ait implémenté l'interface Comparable

Question:

comment faire pour afficher les informations de la personne qui a un numéro donné?

Les objets Map

❑ Ce type de collection gère les éléments avec deux entités : une clé et une valeur associée. La clé doit être unique donc il ne peut y avoir de doublons. En revanche la même valeur peut être associée à plusieurs clés différentes.

❑ Cette interface est une des deux racines de l'arborescence des collections. Les collections qui implémentent cette interface ne peuvent contenir des doublons. Les collections qui implémentent cette interface utilisent une association entre une clé et une valeur.

❑Java propose deux nouvelles classes qui implémentent cette interface :
HashMap qui stocke les éléments dans une table de hachage
TreeMap qui stocke les éléments dans un arbre

La classe HashMap

Cette classe implémente une table de hachage. La clé et la valeur de chaque élément de la collection peuvent être de n'importe quel objet.

Elle autorise les objets null comme clé ou valeur

Pour créer une map on a besoin d'un élément et sa clé.

HashMap <String , Personne>_table = **new** HashMap();

pour rajouter un élément à une map on utilise la méthode put(key, value)

Exemple

```
public static void main(String args[]) {
        HashMap <String , Personne>   table =
newHashMap();

         table.put(p1.getNum(), p1);
        table.put(p2.getNum(),p2);
        table.put(p4.getNum,p4);
        System.out.println(map);
}}
```

Comment réalise-t-on un accès direct ?

Grace à la méthode get

Personne p=table.get(“20040011”);
//accès direct à la onzième personne né en 2004

Si le code n’existe pas p recevera null

La classe HashSet

• Cette classe est un ensemble sans ordre de tri particulier. Les éléments sont stockés dans une table de hashage. Pour parcourir un HashSet, on est obligé d'utiliser un Iterator.

• Les éléments d’un set ne sont pas doublés, chaque élément est unique

•On utilise cette classe quand on a la manipulation des ensembles (union , intersection, ….)

Exemple:

```
import java.util.*;
public class  TestHashSet {
public static void main(String args[]) {
HashSet set = new HashSet();

set.add("CCCCC");
set.add("BBBBB");
set.add("DDDDD");
set.add("BBBBB");
set.add("AAAAA");
 Iterator it= set.iterator();
while (it.hasNext()) {
System.out.println(it.next());
}  }
```

Chapitre V
Les exceptions

Introduction informelle

- Avion A = new Avion (‘’AirBusA250’’, ‘’ Piste_2’’);

  A.decolle();

  A.vole Jusqu’à(Endroit e);

  A.atterit();

- On a supposé que la situation est normale. Si l’avion est confronté à un accident ou à une attaque terroriste , quoi faire?

- int nombre = e.nextInt();

  float r = 20/nombre;

- Qu'arrive t- il si on introduit un caractère au lieu d'un nombre ?
- Qu'arrive t- il si on introduit la valeur 0 ?
- Le programme se termine avec comme message Exception .

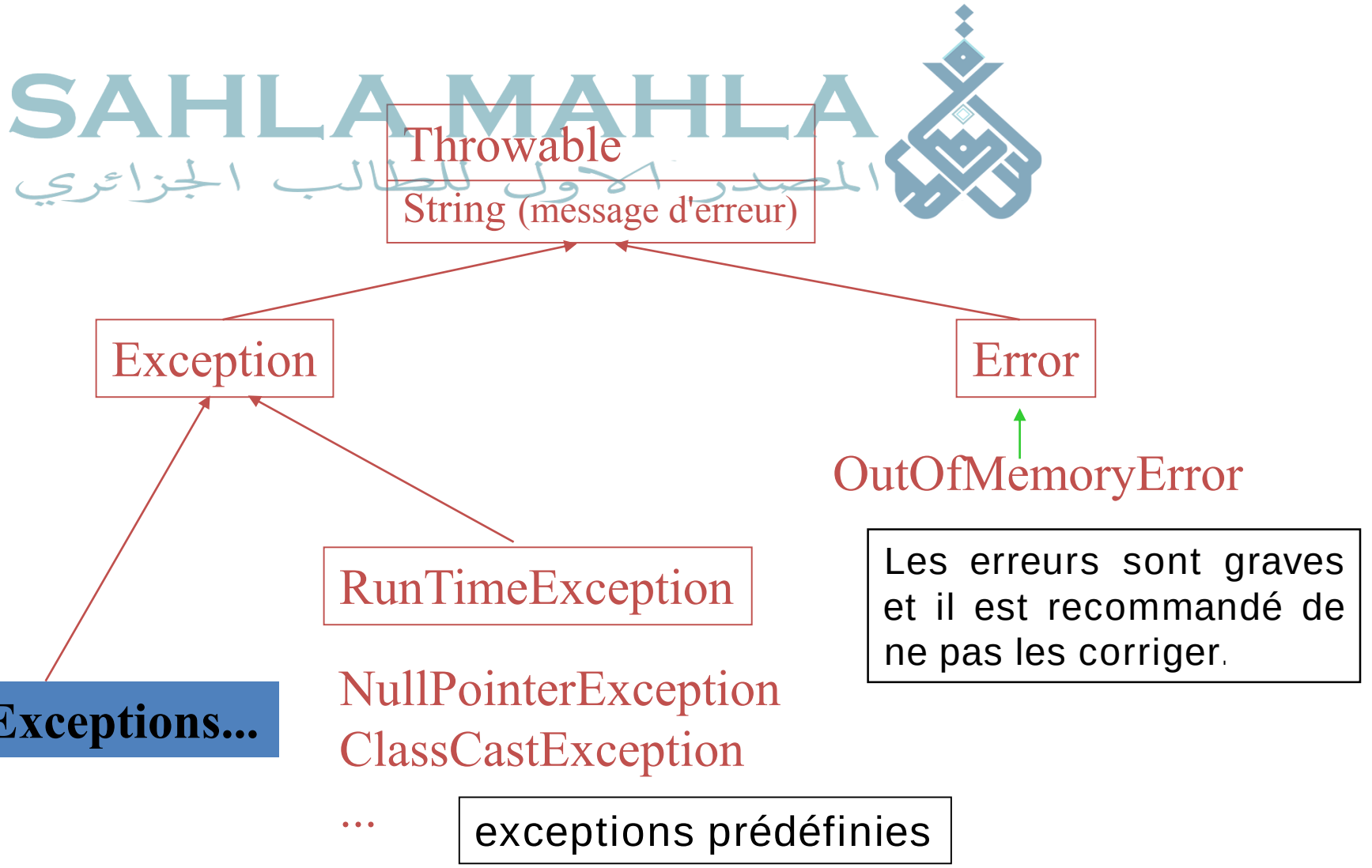
Notion d'exception

- Une exception est un signal qui indique que quelque chose d'exceptionnel est survenu en cours d'exécution. On a deux alternatives :

1. Laisser le programme se terminer avec erreur,
2. Essayer, malgré l'exception, de continuer l'exécution normale

- Une exception crée un objet d'une classe d'exception. Il provoque la sortie d'une méthode, correspond à un type d'erreur et contient les informations concernant l'erreur.

• Arbre des exceptions



Les Exceptions standards

- Il en existe une quarantaine
- Chaque type d'exception est une classe
- De la classe Exception on a
  - ArithmeticException
  - NumberFormatException
  - InputMismatchException
  - IndexOutOfBoundsException
  - NegativeArraySizeException
  - NullPointerException
  - FileNotFounfException

Gestion des exceptions

- On peut empêcher l'arrêt brutal d'une exception en mettant en place un mécanisme de gestion des exceptions

- On utilise les blocs try / catch

- On ajoute un bloc try auquel on rajoute 0 , 1 ou plusieurs blocs catch

Format général d'un gestionnaire d'exception

try { lignes de codes à protéger }
catch (UneException1 ex) **{**
<*lignes de code réagissant à l'exception UneException1* > }
catch (UneException2 e) **{**
<*lignes de code réagissant à l'exception UneException2* >**}**
…
finally
{
Lignes de code s'exécutant quelque soit la situation
même si dans l'un des blocs contient l'instruction return
}

Les types **UneException1, UneException2** sont obligatoirement des classes qui **héritent de la classe Exception**.

```
try {    A.decolle();
         A.voleJusquaProchaineEtape();
     A.atterit(); }
```



```
catch(ProblemeTechniqueException ex)
{    A.atteritDurgence();
  A.getPilote().annonce("Vos Parachutes et
    sautez !! vite !!");
A.getPilote().sauteEnParachute(); }
catch(MenaceTerroristeException ex)
{ A.getSecurite();  }
```

```
int nombre =  e.nextInt();
float r;
try  {r =  20/nombre;}
Catch (Exception ex) { System.out.println(ex);}
```


L'analyse

- Java crée automatiquement un objet de la classe ArithmeticException.
- L'objet créé est capturé par le bloc Catch vu que c'est une instance de la classe exception. Les instructions de ce bloc catch sont exécutées. Le message affiché est : Objet exception java.lang.ArithmeticException

```
try {nombre = e.nextInt();
        result = x / nombre;
        System.out.println( "Résulat:" + result);
    }
catch (InputMismatchException ex ) {
    System.out.print( Sop(‘’ le nombre est erroné
    ‘’);}
catch (ArithmeticException ex) {Sop( ‘’ le
    nombre doit être différent de zéro’’);}
```

```
int v;
try {
v = Integer.parseInt (
   JOptionPane.showInputDialog() )
}
catch (NumberFormatException ex) {
System.out.println( "Ceci n'est pas un entier!!");
   System.exit(0);
```

```
 int saisirEntier(){
int x; while ( true)
 try {x= Integer.parseInt (
   OptionPane.showInputDialog ("Introduire un
   entier")); return x;}


catch (Exception ex){System.out.println("veillez
   donner une valeur entière");}
}
```

- L'exception est levée si l'usager entre une chaine de caractère ou un réel

```
int saisirEntier(){
int x;
while ( true)
    try {x= e.nextInt(); return x;}

catch (Exception  ex ) {
    e.next();System.out.println(ex+ "veillez donner
    une valeur entière");}
}
```


- L'exception est levée si l'usager entre une chaine de caractère ou un réel

(IndexOutOfBoundsException

```
float  T[ ] = new float[5];
    try {
        for (int k=0; k<5; k++)
                    T[ k ] = 100 - k;

        int n = lec.nextInt();
        System.out.println("La valeur est: " + T[n] );
    }
    catch (IndexOutOfBoundsException e)  {
          System.out.println( "Ce numéro ne correspond pas
à aucun élément!");
     }
```

```
try {n = lec.nextInt();
float  notes[ ] = new float[n];      }
catch (NegativeArraySizeException e)  {
        System.out.println( « Taille du tableau
négative");
 }
```

Déclenchement d'une exception existante (prédéfinie)

La Java machine peut déclencher une exception automatiquement comme dans l'exemple de la levée d'une ArithmeticException lors de l'exécution de l'instruction "x = 1/0 ;".

La Java machine peut aussi déclencher une exception à votre demande suite à la rencontre d'une instruction **throw**.

```
public class Trajet{
double distance;
public Trajet (double d ) throws Exception {
{   System.out.println("...Avant Incident  ) ;
 if( d <0) throw new Exception(" Distance ne peut
      être négative !");
  System.out.println("Apres Incident... " ) ;
  }

    public static void main(String[] args) throws Exception  {
          Scanner lec = new Scanner(System.in);
                  Trajet  T = new Trajet(lec.netInt()) ;

                }       }
```


- SI ON Introduit une valeur négative le programme s'arrête.

```
public static void main(String[] args)   {
        Scanner lec = new Scanner(System.in);
    System.out.println("Debut programme:") ;
    double  x= lec.nextInt();
    try {Trajet  O = new Trajet (x);}
    catch( Exception ex) {
System.out.println("Interception exception :"+
e.getMessage()) ;}
```

- **Résultats de l'exécution :**

Debut programme
...Avant incident
Interception exception : " Distance ne peut être négative !"

Création d'une classe exception personnalisée

- On a la possibilité de créer une nouvelle classe d'exception, pour ce faire, il faut qu'elle **hérite obligatoirement de la classe Exception** ou de n'importe laquelle de ses sous-classes.

- Créons une classe d'exception que nous nommerons ArithmeticExceptionPerso héritant de la classe ArithmeticException puis exécutons ce programme :

```
public class   ArithmeticExceptionPerso extends
    ArithmeticException{

int x =8; public ArithmeticExceptionPerso(String  s){ super(s) ;}
}
```

```
class A {
int calcul(int y)
    throws  ArithmeticExceptionPerso
{ System.out.println("…Avant Incident " ) ;

if( y <0) throw new  ArithmeticExceptionPerso
    (“Mauvais Calcul");
   System.out.println("Apres Incident… " ) ;
  return x+y;}
```

```
public static void main(String[] args)   {
        A  O = new A();
        Scanner lec = new Scanner(System.in);
        int  y= lec.nextInt();
      System.out.println("Debut propramme:") ;
        try{ System.out.print( O.calcul(y));}
      catch(ArithmeticExceptionPerso ex) {
    System.out.println("Interception exception :"+
    e.getMessage()) ;}
```


L'exécution de ce programme est identique à celle du programme précédent, notre exception fonctionne bien comme celle de Java.

Exception et manipulation des fichiers

- Certaines exceptions ne sont à vérification obligatoire (***unchecked***) tel que les exceptions standards vues jusqu'à présent. En d'autre terme le compilateur ne nous exige pas de rajouter try catch. Ou un minimum avec throws
- Mais il existe une catégorie d'exceptions qui à sont vérification obligatoire (*checked*) telle que *IOException* celles-ci produisent des erreurs de compilation lorsqu'elles sont non gérées.

1/ Lecture d'un fichier

- La lecture d'un fichier peut se faire par le biais de plusieurs classes, On va se contenter de la classe Scanner qui utilise la classe FileInputStream comme flux d'entré au lieu de System.in de la console.
- Si on veut lire un fichier Etud.txt , on a deux choix :

1. Le placer dans le répertoire du projet de la classe courante, dans ce cas le chemin ne sera pas spécifié

   Scanner Sc = new Scanner (**new** FileInputStream("Etud.txt")) ;

2. Le placer librement dans n'importe quel endroit mais il faut spécifier le chemin correspondant avec des « slash » entre chaque répertoire

Scanner sc = **new** Scanner(**new** FileInputStream("C:/Documents and Settings/Bureau/Etud.txt"));

- La lecture, se fait normalement comme la lecture sur console en utilisant les méthodes next(), nextInt()… la condition d'arrêt correspond au cas où l'objet scanner n'a pas d'éléments suivants.

On le test avec la condition :

while(sc.hasNext).

- **EXEMPLE** On désire lire un fichier, « etud.txt » qui contient, matricule, nom, prenom et la moyenne pour un ensemble d'étudiants.


```
public class AppLecFichier {
public static void main( String [] args){
    String nom,  prenom;     int mat , I = 0;      float moy ;
     Scanner sc = null;
    try   {  sc = new Scanner(new FileInputStream("Etud.txt"));
           }
     catch(FileNotFoundException e)
    {      System.out.println("le fichier  n'existe pas dans le chemin spécifié, ouverture impossible ");
           System.exit(0);
    }
```

Remarque : // le format du fichier doit être connu

```
sc.nextLine() ; // sauter la première ligne qui contient une
    entête
while(sc.hasNext() )
    {    Mat = sc.nextInt();
          nom = sc.next();
          prenom=sc.next();
          moy = sc.nextFloat() ;
          System.out.println ("Etudiant:"+mat+"  "+ nom+"
    "+prenom+" "+moy);
    i++;
    }

if (i==0) System.out.println ("aucun étudiant enregistré
    ");}}
```

On prend trois cas possibles d'affichage :

1er cas : fichier n'existe pas

- Le programme s'arrête en affichant : le fichier n'existe pas dans le chemin spécifié, ouverture impossible

2ème cas : fichier existe et ne contient que l'entête :

Le programme affiche le message : aucun étudiant enregistré

Remarque : si l'entête n'est pas donnée une exception est déclenchée

java.util.NoSuchElementException: No line found
at java.util.Scanner.nextLine(Unknown Source)

2/ Ecriture dans un fichier

L'écriture dans u fichier se fait aussi par le biais de plusieurs classes, on se contente de la classe PrintWriter qui utilise le flux de FileOutputStream et on utilise la méthode print de la classe PrintWritter.

- **A/ créer un nouveau fichier**

Pour créer un nouveau fichier, on utilise l'une des instructions suivantes :

```
new PrintWriter(new FileOutputStream("Info.txt"));
```

ou

```
new PrintWriter(new FileOutputStream("Info.txt", false));
```

- Exemple : On désire créer un fichier texte qui contient les informations des étudiants :

le fichier peut ou non exister

S'il existe il sera **écrasé**, sinon il sera **crée**

```
public class App_ecr_fichier {
    public static void main(String[] args) throws
    IOException  {

    PrintWriter pw = null;

        pw = new PrintWriter(new
    FileOutputStream("Info.txt"));
```

```
pw.print("Matricule: " + 123);          pw.print(" Nom: "
+ "Sounci");
pw.print(" Prénom: " + "Kamel");
```


```
pw.println("================== fin d'enregistrement
==============\n");
pw.print("Matricule: " + 124); pw.print(" Nom: " +
"ABED");
        pw.print(" Prénom: " + "Amine");
pw.println("================== fin d'enregistrement
==========\n");
```

pw.close(); // fermeture du fichier

- **Pourquoi as-t on rajouté la fermeture du fichier dans ce cas ?**

Tout simplement car la sauvegarde du contenu du fichier n'est faite réellement sur disque que si le fichier est fermé,

B/ Compléter un fichier existant

Pour compléter un fichier existant, il suffit de rajouter le paramètre true à l'instruction d'ouverture

new PrintWriter(**new** FileOutputStream("Info.txt", true));

Manipuler des fichiers binaires

- Pour manipuler des fichiers binaires (fichiers d'objets) on doit implémenter l'interface Sérializable qui ne contient aucune méthodes
- Cette interface est utilisée pour sérialiser les objets dans un flux de fichier ou dans une communication de réseau.

```
public class Personne implements Serializable{
private String nom,prenom,….
….
 }
public class Application1 {
public static void main(String[] args){
ObjectOutputStream out;
try{ FileOutputStream fos = new
   FileOutputStream(‘’fichierPersonne’’);
```

```
out = new ObjectOutputStream(fos);
Personne pers = new Masculin(……..);
out.writeObject(pers);
}
Catch( Exception ex) { System.out.println(ex);}
}}
```

Lecture d’un fichier d’objets sérialisé

On utilise la classe ObjectInputStream et la classe pour le flux FileInputStream et la méthode pour la lecture readObject appliquée sur un objet de ObjectInputStream