Programmation Orientée Objet en JAVA

Hamid LADJAL

Hamid.ladjal@univ-lyon1.fr

Master CCI

https://perso.univ-lyon1.fr/hamid.ladjal/M2CCI_JAVA/

Plan

- Bases et les concepts de JAVA
- Concepts de l'orienté objet; classes, méthodes…..
- Héritage et Polymorphisme
- Classes Abstract, Interfaces, classes génériques
- Enumération haines de caractères
- Gestions des erreurs avec des exceptions
- Interfaces Graphiques
- Applets, entrées/sorties….

Présentation de JAVA

Bases de Java

Historique

- 1991, chez SUN est créé le langage OAK (pour interface entre appareil ménagers et ordinateurs)
- 1994, Oak se tourne vers Internet et devient Java
- 1995, Java est intégré à Netscape
- 1998, Version 2 de Java
- 2000, Version 1.3 de Java
- ……. Version 1.4 de Java ….
- 2010 java JDK 7
- ….
- 2021 java 17

https://www.oracle.com/java/technologies/downloads/#jdk17-windows

Présentation

- Java est un langage récent qui s'est répandu de façon étonnamment rapide

- Car : Java est portable, non dépendant de l'environnement d'implémentation (de la machine)

- Car : la syntaxe de Java est très proche du langage C++ (et donc de C)…

Présentation

- Car : Java est robuste
  - Gestion de la mémoire par un système de ramasse - miettes (Garbage Collector)
  - Typage fort
- Car : Il existe un grand nombre de classes sûres et commentées
- Car Java est gratuit …

Présentation

- Java est "vraiment" Orienté Objet :

  tout est objet

- Java fournit un code (ByteCode) interprété par une machine virtuelle
  - (+) Garantit la portabilité
  - (-) Ralentit l'exécution

Télécharger Java

- Sur le site (www.oracle.com), Oracle propose les différents Kits de Développement Java (JDK)
- https://www.oracle.com/technetwork/java/javase/downloads/index.html
- Sur Internet, grand nombre de sources et de documentation
  [Ex : "Thinking in Java" de B. Eckel]
  http://sd.blackball.lv/library/Thinking_in_Java_4th_edition.pdf

- Programmer en Java (2ème édition)
  - Auteur : Claude Delannoy
  - Éditeur : Eyrolles
  - Edition : 2002 - 661 pages - ISBN : 2212111193

- Java en action
  - Auteur : Ian F. Darwin
  - Éditeur : O'Reilly
  - Edition : 2002 - 836 pages - ISBN : 2841772039

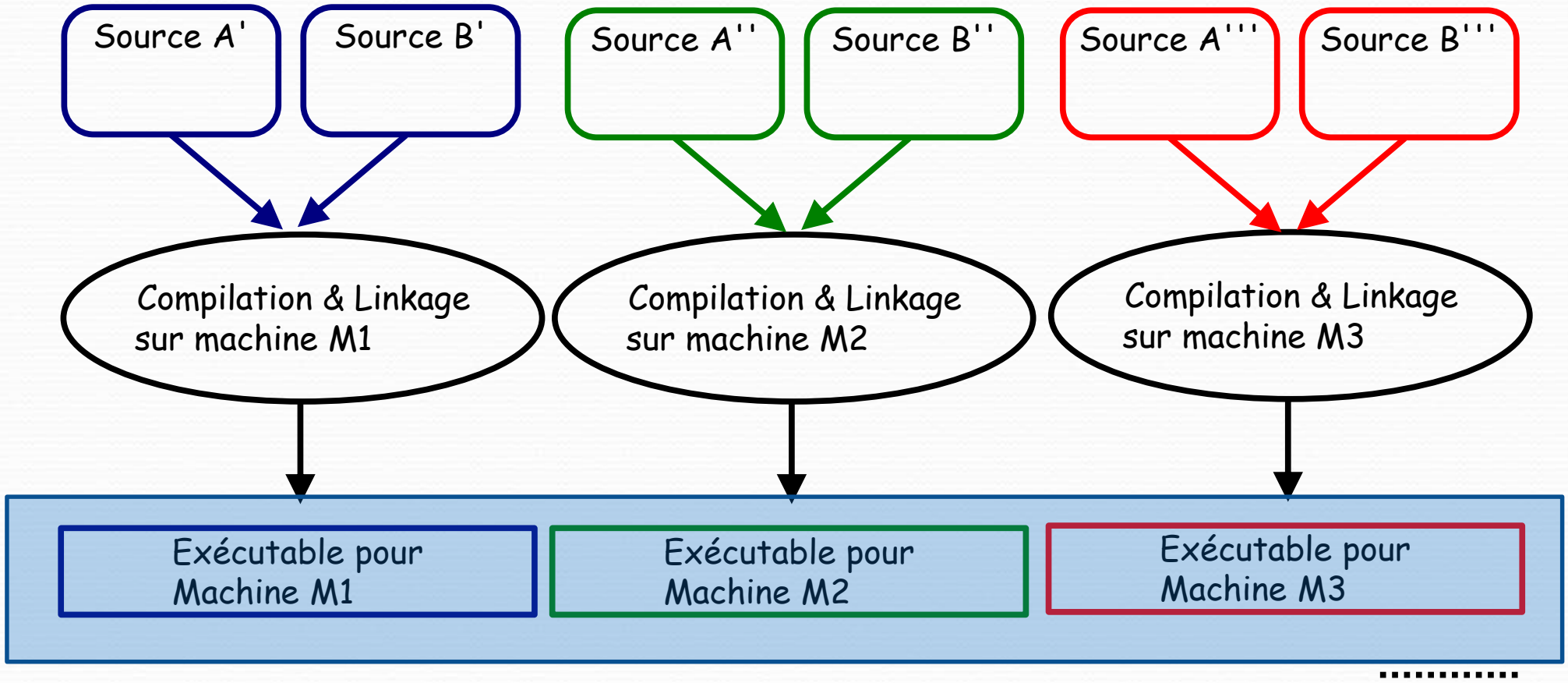
- Apprendre Java et C++ en parallèle
  - Auteur : Jean-Bernard Boichat
  - Éditeur : Eyrolles
  - Edition : 2003 - 742 pages - ISBN : 2212113277

Les concepts de JAVA

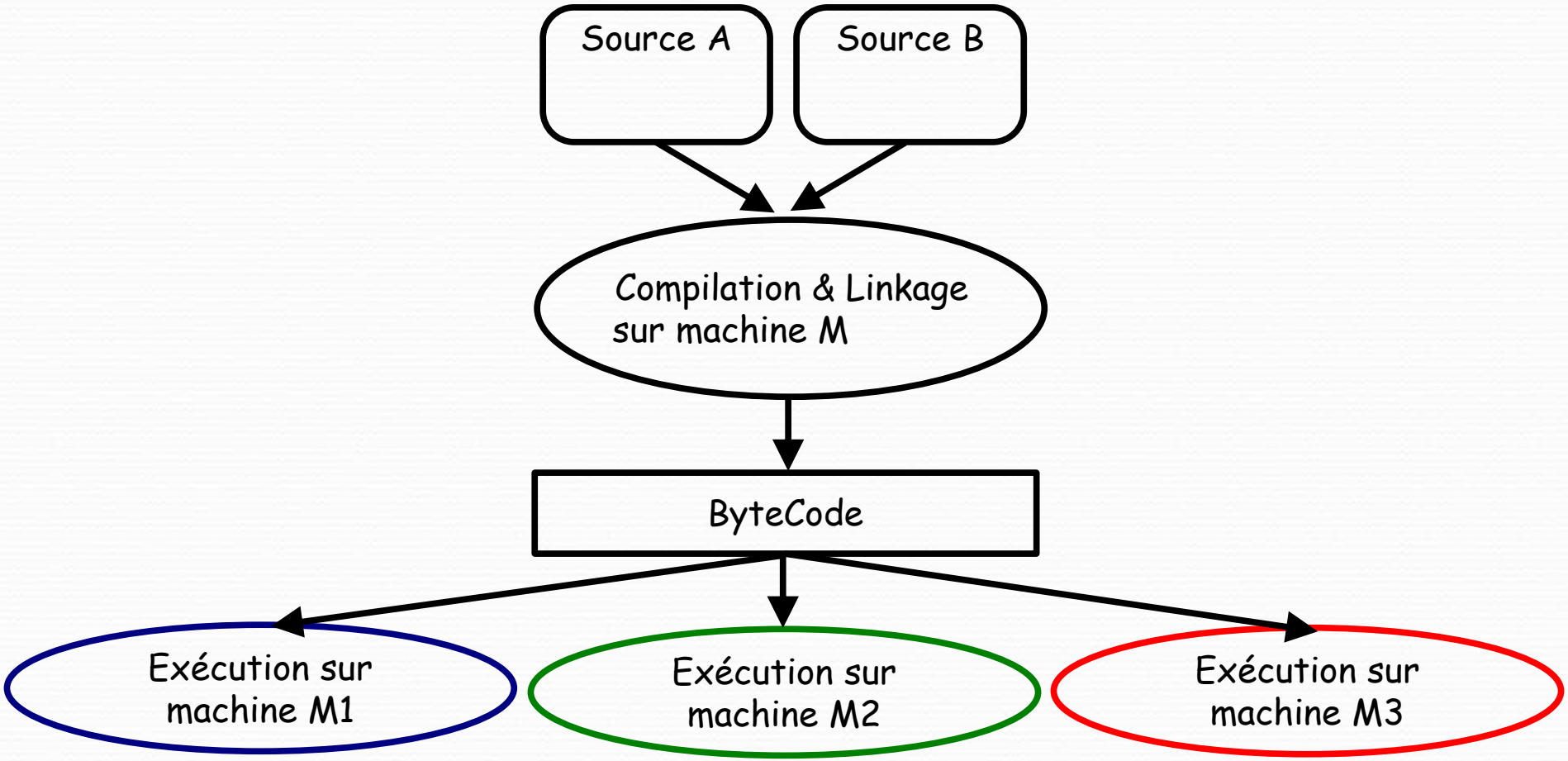
La Compilation & JAVA

- Le principal atout de JAVA concerne sa portabilité.
- Celle-ci est possible par la notion de ByteCode.
- JAVA ne produit pas un code natif, directement exécutable.
- JAVA produit un code intermédiaire, le ByteCode, interprétable par une machine virtuelle.

La Compilation "classique"



La Compilation Java



Interprétation par machine virtuelle (Java Virtual Machine)
spécifique au système d'exploitation

Code natif vs ByteCode

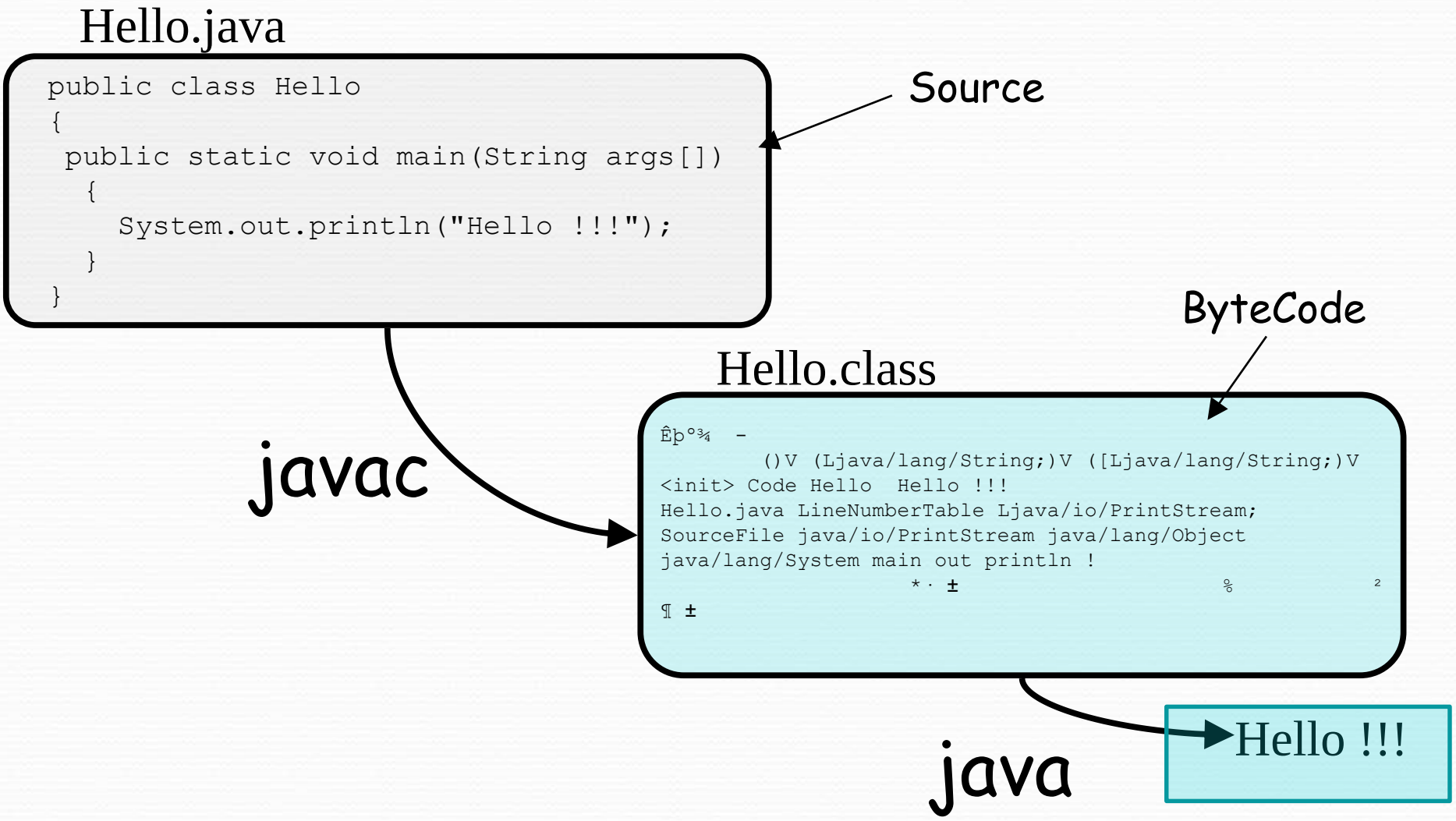
- (+) rapidité d'exécution
- (-) source pas toujours portable
- (-) recompiler pour changer de système

- (+) source portable
- (+) Bytecode portable
- (-) moins rapide car interprété

Source portable

- Les types primitifs sont totalement spécifiés.
- Ils ne dépendent pas de l'architecture de la machine :
  - Les entiers, réels, etc… sont toujours définis sur le même nombre de bits.
- Génération du ByteCode par le compilateur javac

Exemple de compilation



Ramasse Miettes

- En JAVA, tout est objet. Il faut donc créer explicitement chaque objet manipulé.
- Cependant, il est inutile de les détruire. Le Garbage Collector (ramasse miettes) s'en charge.
- Dès qu'un objet n'est plus utilisé (sortie de boucle, ...), il est détruit.

Eléments de base

Tout est Objet (ou presque...)

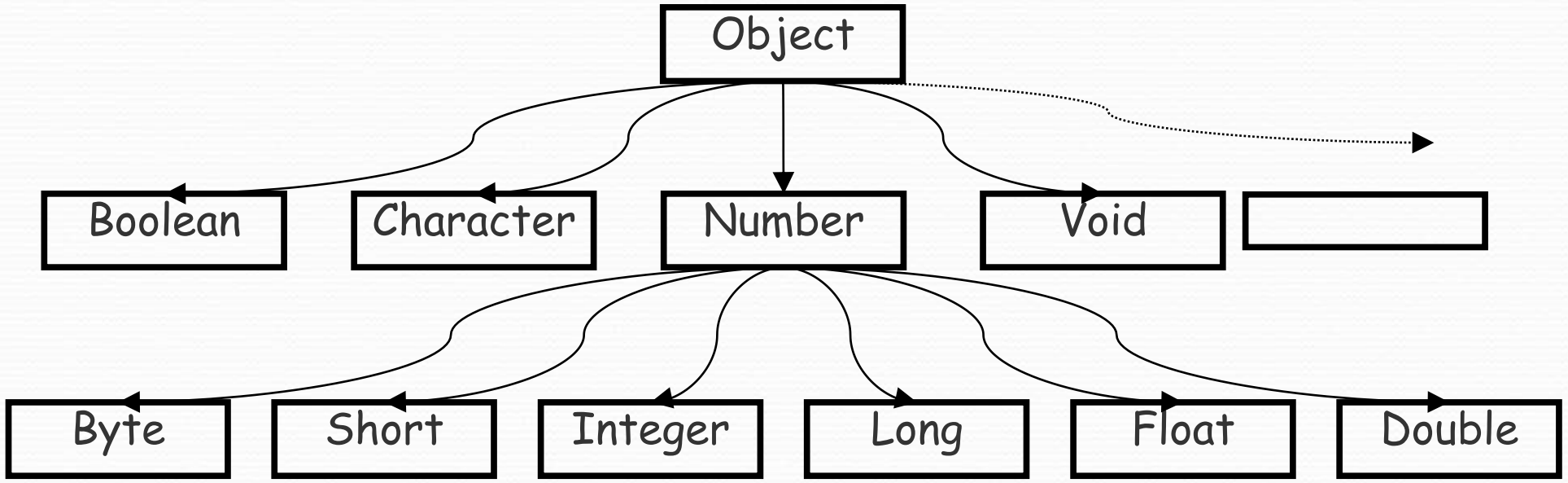
- Tous les éléments en JAVA dérivent de la classe Object.
- Tous sauf les primitives (8 en tout) représentant des types simples.
- A chaque type de primitive correspond une classe d'objet.

Les primitives : des types simples

Nom	*Taille*	*Minimum*	*Maximum*	*Classe*
boolean	1 bit	false	true	Boolean
char	16 bits	Unicode 0	Unicode $2^{16}-1$	Character
byte	8 bits	-128	127	Byte
short	16 bits	-2^{15}	$2^{15}-1$	Short
int	32 bits	-2^{31}	$2^{31}-1$	Integer
long	64 bits	-2^{63}	$2^{63}-1$	Long
float	32 bits	$(+/-)1.4\ 10^{-45}$	$(+/-)3.4\ 10^{38}$	Float
double	64 bits	$(+/-)4.9\ 10^{-324}$	$(+/-)1.8\ 10^{308}$	Double
void	-	-	-	Void

Les objets simples

- Tout objet Java hérite de la classe Object :



Primitives <-> Objets

- Passer d'une primitive à l'objet correspondant :

```
int i = 5;
Integer obj = new Integer (i);
```

- Dans la classe Number sont définies les méthodes de conversion intValue, doubleValue, floatValue, ...

```
int j = obj.intValue();
```

- Dans la classe Object est définie la méthode toString retournant la valeur d'un objet sous forme de chaîne.

Les opérateurs

- Arithmétiques :
  pour tout nombre x, on peut écrire :
  x++; x--; ++x; --x; x+=2; x-=2; x*=2; x/=2;
  x%=2 (x = x modulo 2)

  <, <=, >, >=, == égalité

  != : différence; ! : négation

- Logiques : pour tout booléen x et y,

non x	⇔ !x			
x et y	⇔ x & y	ou	x && y	*(test économe)*
x ou y	⇔ x | y	ou	x || y	*(test économe)*
x ou exclusif y	⇔ x ^ y			

Structures de contrôle

- Les structures de contrôle utilisées en Java sont fortement inspirées de celles du langage C ou C++ :
  - if ; else ; while ; do ; for ; break ; goto

Branchement conditionnel (1/2)

- Si … Alors … Sinon

```
if (test)
      {
       action1_si_test_vrai;
       action2_si_test_vrai;
      }
else
      {
       action1_si_test_faux;
       action2_si_test_faux;
      }
```

- Le test est de la forme :

  a ; !a ; a & b ; a & !b ;

  x<y ; x == y; x != y ; …

Branchement conditionnel (2/2)

- Branchement selon un choix multiple :

```
switch (expression)
{
    case valeur_1 : actions; break;
    case valeur_2 : actions; break;
    case valeur_3 : actions; break;
    default : actions si valeur inconnue;
}
```

Boucle "Tant que"

- **while** (test)
  {
  actions;
  }

- **do**
  {
  actions;
  }
  while (test);

Boucle "Pour"

- int i;
  for(i=0; i<5; i++) // *ou* ***for***(*int i=0; i<5; i++*)
  {
  actions;
  }

Sorties de boucles

- L'instruction **break** permet de sortir d'une boucle avant sa fin "normale".
- L'instruction **continue** passe le reste de la boucle, mais n'en sort pas.
- L'instruction **goto label** subsiste encore. Le label doit alors être défini (et suivi de **:**).

Concepts de l’orienté objet avec java

Exemple de programme

- Programme affichant « Bonjour !!!». Ce programme contient une classe principale Bonjour. Il doit donc se nommer Bonjour.java

```
public class Bonjour
{
        public static void main(String []  args)
        {
                System.out.println(“Bonjour !!! “);
        }
}
```

Exemple de programme

- Créer un fichier source Bonjour.java

  Un fichier source contient du texte, écrit en Java

  Compiler le source en fichier bytecode Bonjour.class

  Le compilateur **javac**, traduit le texte source en instructions compréhensibles par la Machine Virtuelle Java (JVM)

  Exécuter le programme contenu dans le fichier **bytecode**

  L'interprète java implémente la **JVM**

  L'interprète traduit le bytecode en instructions exécutables par la machine

Exemple de programme

Fichier Bonjour.java

```
Public class Bonjour {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

Compilation

javac Bonjour.java

Bonjour.class

Exécution

java Bonjour

Hello World !

La compilation

- La compilation d'un programme, ne génère pas d'instructions spécifiques à votre plate-forme
- Mais du **bytecode** Java, qui sont des instructions de la **Machine Virtuelle Java** (JVM)
- Si votre plate-forme (Windows, UNIX, MacOS, un browser Internet) dispose d'une JVM, elle peut comprendre le bytecode

Exemple de programme

Définir la classe Bonjour

```
public class Bonjour {
  public static void main(String[] args) {
      System.out.println("Bonjour !");
  }
}
```

La méthode `main`

```
public class Bonjour {
  public static void main(String[] args) {
      System.out.println("Bonjour !");
  }
}
```

Une application Java doit contenir une méthode main

- Appelée en premier par l'interprète
- main appelle les autres méthodes nécessaires pour exécuter l'application

Classes et de Méthodes

« class Bonjour » est la déclaration d'une classe (obligatoire en Java).
« public » signifie que main est visible de l'extérieure de la classe Bonjour.
« static » signifie que main n'est pas associée a une instance mais à une classe…
« void » main ne retourne rien

« main » est le nom de la méthode…
« argv » est le tableau de chaînes de caractères habituels du C.
« String » est le nom de la classe chaîne de caractères
« System » est le nom de la classe système
« out » est le nom de l'instance pour effectuer des sorties
« println » est le nom de la méthode qui imprime une ligne avec un retour chariot.

Classes et de Méthodes

Dans une fichier texte **Nomdelaclasse.java** on écrira

```
class Nomdelaclasse {
// déclaration des attributs
Type nom ;
//méthodes
Type retour Nom (type arg1, type arg2) {
}... ;

//fonction main
```

S'il n'y a pas de type retour (par exemple si on fait un affichage ou si on travaille sur les attributs) on met **void.**

Notion de Classe

Concept

- Une classe est un support d'encapsulation : c'est un ensemble de données et de fonctions regroupées dans une même entité.
- Une classe est une description abstraite d'un objet.
- Les fonctions qui opèrent sur les données sont appelées des méthodes.

Notion de Classe

- Instancier une classe consiste à créer un objet sur son modèle. Entre classe et objet il y a, en quelque sorte, le même rapport qu'entre type et variable.

- En Java : tout appartient à une classe sauf les variables de types primitifs (int, float…).

- Pour accéder à une classe il faut en déclarer une instance de cette classe (ou un objet).

Par convention:

- Les noms de classe commencent par une Majuscule.
- Les noms d'instances ou de variables commencent par une minuscule

Notion de Classe

Une classe comporte sa déclaration, des variables et la définition de ses méthodes

public : La classe est accessible partout

Private : La classe n'est accessible qu'à partir du fichier où elle est définie

protected : Est accessible aux sous-classes et aux classes voisines,

private protected : Est accessible également aux sous-classes,

Notion de Classe

final : La classe ne peut pas être modifiée, sa redéfinition grâce à l'héritage est interdite. Les classes déclarées final ne peuvent donc pas avoir de classes filles.

abstract : La classe contient une ou des méthodes abstraites, qui n'ont pas de définition explicite.

Une classe déclarée abstract ne peut pas être instanciée : il faut définir une classe qui hérite de cette classe et qui implémente les méthodes nécessaires pour ne plus être abstraite.

Autres mots-clés : final

- **final** définit :
  - un attribut invariable,
  - une méthode qui ne peut être surchargée,
  - une classe qui ne peut être héritée.

Autres mots-clés : static

- **static** définit un membre commun à toutes les instances de la classe
- On parle de membre de classe
  par opposition à membre d'instance
- Une variable de classe est indépendante de l'objet, elle existe dès le premier appel à la classe.

Autres mots-clés : static

- Seule une méthode de classe peut utiliser des variables de classe
- Une méthode de classe ne peut être surchargée dans les sous-classes
- Utilisation des membres de classe : compteur d'objet, optimisation (définition de constante)

Autres mots-clés : abstract

- abstract associé à une méthode permet de ne pas lui donner de corps. Elle devra donc être définie dans les sous-classes.
- Si une méthode est abstraite, sa classe doit être définie comme abstraite.
- Une classe abstraite ne peut être instanciée.

Autres mots-clés : abstract

- L'abstraction : définir un moule général pour les classes dérivées.
- Une **interface** est une classe :
  - ne contenant que les entêtes de méthodes
  - ne contenant pas d'attributs variables.
  - Notation :
    interface Etre_vivant {…}
    class Animal **implements** Etre_vivant {…}

Attributs et Méthodes

Les Attributs

Les données d'une classe sont contenues dans les propriétés ou attributs. Les attributs sont les données d'une classe, ils sont tous visibles à l'intérieur de la classe.

Syntaxe générale:

[< modificateur de visibilité>**]** <type> <nom_attribut> [=<expression>] ;

Exemple

```
class Rectangle{
private int longueur ;
private int largeur ;
public Point Centre ;
}
```

Attributs et Méthodes

Les variables d'instances

Une variable d'instance nécessite simplement une déclaration de la variable dans le corps de la classe.

Exemple :

```
public class MaClasse {
public int valeur1 ;
int valeur2 ;
protected int valeur3 ;
private int valeur4 ;
}
```

Chaque instance de la classe a accès à sa propre occurrence de la variable.

Attributs et Méthodes

Les variables de classes (static)

Les variables de classes sont définies avec le mot clé **static**

Exemple :

```
public class MaClasse {
static int compteur ;
}
```

Chaque instance de la classe partage la même variable.

Attributs et Méthodes

Les constantes

Les constantes sont définies avec le mot clé **final** : leur valeur ne peut pas être modifiée.

Exemple :

```
public class MaClasse {
final double pi=3.14 ;
}
```

Attributs et Méthodes

Les méthodes

Les méthodes sont des fonctions qui implémentent les traitements de la classe.

La syntaxe de la déclaration

La syntaxe de la déclaration d'une méthode est :

```
modificateurs type_retourné nom_méthode ( arg1, ... ) {...
// Définition des variables locales et du bloc d'instructions
// Les instructions
}
```

Le type retourné peut être élémentaire ou correspondre à un objet. Si la méthode ne retourne rien, alorson utilise **void.**

Attributs et Méthodes

Une méthode représente l'équivalent d'une procédure ou fonction dans les langages classiques de programmation.

Exemple :

```
class Rectangle {
private int longueur ;
int largeur ;
public int calcul_surface ( ) {
return (longueur*largeur) ;
}
public void initialise (int x, int y) {
longueur =x ; largeur = y ;
}
}
```

Attributs et Méthodes

Toutes les méthodes sauf les «constructeurs » doivent avoir un type de retour.

- Les méthodes qui ne renvoient rien, utilisent « **void** » comme type de retour.
- Une méthode peut ne pas avoir besoin d'arguments, ni de variables à déclarer.
- Les méthodes agissent sur les attributs définis à l'intérieur de la classe.
- Sans modificateur, la méthode peut être appelée par toutes autres méthodes des classes du package auquel appartient la classe.

- La valeur de retour de la méthode doit être transmise par l'instruction **return.**

Exemple :

```
int add(int a, int b) {
    return a + b;
}
```

La transmission de paramètres

La transmission de paramètres

- Lorsqu'un objet est passé en paramètre, ce n'est pas l'objet lui même qui est passé mais une référence sur l'objet.

- La référence est bien transmise par valeur et ne peut pas être modifiée mais l'objet peut être modifié via un message (appel d'une méthode).

- Pour transmettre des arguments par référence à une méthode, il faut les encapsuler dans un objet qui prévoit les méthodes nécessaires pour les mises à jour.

Constructeurs

Un constructeur d'objet est une méthode particulière d'une classe qui est appelée lors de la création d'un objet, quel qu'il soit. Son but est :

- d'allouer un emplacement mémoire pour l'objet,
- d'initialiser les attributs de l'objet avec de bonnes valeurs de départ
- de retourner l'adresse de l'emplacement mémoire choisi.

Destructeur

Destructeur

Il n'y a pas de destructeur en Java. Cependant, un destructeur d'objet est une méthode très commune en POO qui :

(d) remet l'objet dans un état terminal et

(e) libère l'emplacement mémoire associé à l'objet.

Ramasse Miettes

- En JAVA, tout est objet. Il faut donc créer explicitement chaque objet manipulé.
- Java n'a pas de destructeur d'objet. Java possède un « ramasse-miettes » qui libère les emplacements mémoire occupés par les objets non référencés
- Cependant, il est inutile de les détruire. Le Garbage Collector (ramasse miettes) s'en charge.
- Dès qu'un objet n'est plus utilisé (sortie de boucle, ...), il est détruit.

le mot-clé *this*

- Quand le programme est dans une méthode d'instance, le programmeur peut manipuler La référence de l'instance courante avec le mot-clef '**this**'.
- Le champ **this** d'une classe se réfère à l'instance de l'objet en cours de traitement.
- Il peut être utilisé : Dans un constructeur
- Lors d'un appel à une fonction d'une autre classe requérant l'objet
  Pour retourner une référence à l'objet

```
class Essai { ....
    Essai changeNom(String nom2)
    {
       nom = nom2;    return this;
    }
    }
```

Exemples

Exemple : une classe qui permet de faire la somme du contenu de deux entiers stockes dans des variables pour la stocker dans une troisième variable

```
class Somme {
// attributs
int x ;
int y ;
int s ;
// methode qui somme le contenu de x et de y et
//  qui renvoie leur somme , le type retour est donc int
        int somme ( ) {
        return x+y;
        }
// methode qui somme le contenu de x et de  y . Le résultat n'est pas
retourne //mais il est stocke dans la troisième variable.
        void addition ( ) {
        s=x+y ;
        }
}
```

Exemple 2 : La portée des variables (qu'affiche le programme?)
Une variable existe dans le bloc dans lequel elle a été créée.

```
Class Exemple {
// attribut
int x=10 ;
// methode qui crée une variable y stocke 5
//dedans et retourne la valeur.
        int methode () {
        int y=5 ;              //y est une variable locale
        return y ;
        }
// methode qui affiche
        void affichage ( ) {          // on affiche x car x est une variable globale
        System.out.println (x) ;      //??????
         System.out.println(y);}}
```

Concepts de l'orienté objet avec java (suite)

Constructeurs

Un constructeur d'objet est une méthode particulière d'une classe qui est appelée lors de la création d'un objet, quel qu'il soit. Son but est :

- d'allouer un emplacement mémoire pour l'objet,
- d'initialiser les attributs de l'objet avec de bonnes valeurs de départ
- de retourner l'adresse de l'emplacement mémoire choisi.

Constructeurs d'une classe

Chaque classe a un ou plusieurs constructeurs

qui servent à:

- créer les instances
- initialiser l'état de ces instances

Un constructeur

- a le même nom que la classe
- n'a pas de type retour

Constructeur, Ex: 1 Class Point

```
/** Modélise un point de coordonnées x, y */
public class Points {
private int x, y;
    public Points(int x1, int y1) { // un constructeur
    x = x1;
    y = y1;
    }
public double distance(Points p) { // une méthode
    return Math.sqrt((x-p.x)*(x-p.x) + (y-p.y)*(y-p.y));
    }

public static void main(String[] args) {
    Points p1 = new Points(1, 2); // Création deux objets
    Points p2 = new Points(5, 1);
    System.out.println("Distance : " + p1.distance(p2));
    }
}
```

2 classes et 1 fichier

```
/** Modélise un point de coordonnées x, y */
public class Points {
private int x, y;
   public Points(int x1, int y1) {
   x = x1; y = y1;
   }
   public double distance(Points p) {
   return Math.sqrt((x-p.x)*(x-p.x) + (y-p.y)*(y-p.y));
   }
}
/** Teste la classe Points */
class TestPoints {
   public static void main(String[] args) {
   Points p1 = new Points(1, 2);
   Points p2 = new Points(5, 1);
   System.out.println("Distance : " + p1.distance(p2));
   }
}
```

Compilation et exécution

- Programme source Java = ensemble de fichiers « **.java** »

- Chaque fichier « **.java** » contient une ou *plusieurs* définitions de classes

- Au plus une définition de classe **public** par fichier « .java » avec nom du fichier = nom de la classe publique

Compilation et exécution

- Durant l'exécution d'un code Java, les classes

(leur *bytecode*) sont chargées dans la **JVM** au fur et à mesure des besoins

- Une classe peut être chargée:

– depuis la machine locale (le cas le plus fréquent)

– depuis une autre machine, par le réseau

– par tout autre moyen (base de données,…)

2 classes dans 2 fichiers

Fichier **Point.java**

```
/** Modélise un point de coordonnées x, y */
public class Points {
private int x, y;
    public Points(int x1, int y1) {
    x = x1; y = y1;
    }
    public double distance(Points p) {
    return Math.sqrt((x-p.x)*(x-p.x) + (y-p.y)*(y-p.y));
    }
}
```

Fichier **TestPoints.java**

```
/** Pour tester la classe Points */
class TestPoint {
    public static void main(String[] args) {
    Points p1 = new Points(1, 2);
    Points p2 = new Points(5, 1);
    System.out.println("Distance : " + p1.distance(p2));
    }
}
```

2 classes et 1 fichier

```
/** Modélise un point de coordonnées x, y */
public class Points {
private int x, y;
   public Points(int x1, int y1) {
   x = x1; y = y1;
   }
   public double distance(Points p) {
   return Math.sqrt((x-p.x)*(x-p.x) + (y-p.y)*(y-p.y));
   }
}
/** Teste la classe Points */
class TestPoints {
   public static void main(String[] args) {
   Points p1 = new Points(1, 2);
   Points p2 = new Points(5, 1);
   System.out.println("Distance : " + p1.distance(p2));
   }
}
```

Plusieurs constructeurs (surcharge)

```
public class Employe {
private String nom, prenom;
private double salaire;
// 3 Constructeurs
    public Employe() {}
    public Employe(String n, String p) {
    nom = n;
    prenom = p;
    }
    public Employe(String n, String p, double  s) {
    nom = n;
    prenom = p;
    salaire = s;
    }
. . .
Em1 = new Employe("Dupond", "Pascal");
Em2 = new Employe("Durand", "Jacques", 12000);
```

Constructeur par défaut

Lorsque le code d'une classe ne comporte pas de constructeur, un constructeur sera automatiquement ajouté par Java

Pour une classe **Classe**, ce constructeur sera par défaut défini par Java

[public] Classe() { }

Désigner l’instance qui reçoit le message, « **this** »

le mot-clé *this*

- Quand le programme est dans une méthode d'instance, le programmeur peut manipuler La référence de l'instance courante avec le mot-clef '**this**'.
- Le champ **this** d'une classe se réfère à l'instance de l'objet en cours de traitement.
- Il peut être utilisé : Dans un constructeur
- Lors d'un appel à une fonction d'une autre classe requérant l'objet
  Pour retourner une référence à l'objet

```
class Essai { ....
    Essai changeNom(String nom2)
        {nom = nom2;    return
this;}
}
```

le mot-clé *this*

Le code d'une méthode d'instance désigne
– l'instance qui a reçu le message (l'instance courante), par le mot-clé **this**

– donc, les membres de l'instance courante
en les préfixant par « **this.** »

Lorsqu'il n'y a pas d'ambiguïté, **this** est optionnel pour désigner un membre de l'instance courante

Exemple de **this** implicite

```
public class Employe {
private double salaire;
. . .
public void setSalaire(double unSalaire) {
salaire = unSalaire;
}
public double getSalaire() {
return salaire;
}
. . .
}
```

Implicitement
this.salaire

Implicitement
this.salaire

this explicite

- **this** est utilisé surtout dans 2 occasions :

– pour distinguer une variable d'instance et un paramètre qui ont le même nom :

```
public void setSalaire(double salaire)
this.salaire = salaire;
}
```

– un objet passe une référence de lui-même à un autre objet :

```
salaire = comptable.calculeSalaire(this);
```

Comptable, calcule le salaire de **moi**

Interdit de modifier **this**

this se comporte comme une variable **final** (mot-clé étudié plus loin), c'est-à-dire qu'on ne
peut le modifier ;

le code suivant est interdit : **this = valeur;**

Il serait, par exemple, interdit d'écrire:

```
static double tripleSalaire() {
return this.salaire * 3;
```

Tableaux

Les tableaux.

Un tableau est un espace mémoire représenté par une grille ayant plusieurs lignes ou colonnes.

En Java, le type des éléments se trouvant dans le tableau est fixé lors de la création. Il est unique.

Par exemple : si on crée un tableau d'entiers, on ne peut y mettre que des entiers. On ne peut pas y mettre un caractère, un double……

LES TABLEAU A UNE DIMENSION.

1 ligne, plusieurs colonnes.
Déclaration :
Type des elements nom du tableau[ ] ;
Types des éléments : primitifs : int, double, char, float

LES TABLEAU A UNE DIMENSION.

Exemple : Déclaration de t un tableau d'entier : **int t[] ;**

Remarque : la taille du tableau n'est pas indiqué lors de la déclaration.

Création : on utilise le mot clef **new.**

Nom du tableau = new type [taille] ;

Exemple : T = new int [11] ;
T est un tableau d'entier de 11 cases.

Lors de la création :
Alloué la mémoire nécessaire au tableau.
Initialise le contenu du tableau.
par exemple pour des **int** on aura des zéro et pour une classe un null.
On peut définir directement un tableau à l'aide de son contenu.

Dans ce cas, la déclaration et la création son dans la même instruction.

Les tableaux.

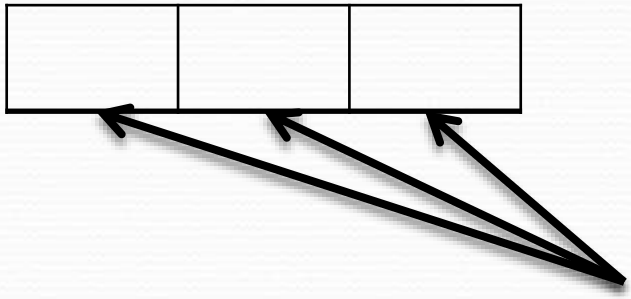
```
int t[] = {3,9,11} ;
```

Déclaration

Création.

3	9	11

```
Point tab[] = {new Point (), new Point (), new Point () }
```



(Dans chaque case du tableau on a des points)

Les tableaux.

Accès aux éléments du tableau.
Nomdutableau [indice]
Les indices commencent à 0.

Exemple :
t[0] ; t[1] ; t[2]

3	9	11

Java vérifie automatiquement que l'indice ne sort pas de l'intervalle [0 ; taille du tableau-1].

Si on sort de l'intervalle on voit le message : « Array Index Out of bounds Exception

L'attribut **length** est un entier qui nous indique la taille du tableau.

Les tableaux.

Exemple :

int t [] = {22, 14, 6} ;

System.out.println (t.length) ;

A l'écran on voit s'afficher 3.

Les indices d'un tableau t vont de 0 à t.length-1

Remarques :

- **length** est un attribut et non une méthode.
- **length** est un attribut non modifiable.
- **length** prend sa valeur après la création du tableau

Les tableaux.

Exercice:

On va créer une classe exemple dont l'attribut est un tableau d'entiers.

- Un constructeur créera un tableau de taille 10 et le complétera aléatoirement.

- Un deuxième constructeur créera un tableau **d'entiers aléatoires** dont **la taille est passée en paramètre**.

- Une méthode affichera le contenu du tableau

Exercice:

```
public class ExempleTab{
        // les attributs
        int t [ ]; // déclaration
        // Conctructeurs  n'a pas de type retour p
         ExempleTab ( ) {
                t= new int[10];
```

t[0]	t[1]	.....	t[9]

Les cases sont initialisées avec des 0 car int est un type primitif

```
                for (int i=0; i<t.length ; i++) {
                t[ i ] = ( int )( Math.random( ) * 10 );
                }
        }
```

// Méthode random de la classe Math (classe de java. Cette méthode retourne un nombre aléatoire appartenant à [0 , 1])

// (int) : partie entière

// les cases de t vont donc contenir un entier compris entre 0 et 9.

Exercice: Suite

```
// Le deuxieme constructeur prend la taille du tableau en parametre
        ExempleTab (int n) {
                t = new int [n];
                for (int i=0; i<t.length ; i++) {
                t[ i ] = ( int )( Math.random( ) * 100 )
                }
        }
// méthode qui affiche le contenu du tableau
// on doit se positionner sur chaque case du tableau
// pour cela on utilise un indice i qui va de 0 a taille -1
// la méthode ne retourne rien car c'est un affichage a l'ecran
// le type retour est donc void
        void afficher( ) {
                for (int i=0; i<t.length ; i++) {
                System.out.println ( t[ i ]);
                }
        }
}
```

Exercice: suite

La classe **ExempleTab** peut être compilée mais elle n'est pas exécutable car elle n'a pas de fonction main. On va écrire une classe contenant un main testant les méthodes de la classe,

Rappel le main a toujours la même signature, car la machine virtuelle doit le retrouver :

le String args []

public static void main (String args [])

Exercice: suite

```
Class Principale {
        public static void main (String args [ ]) {
       // on cree un objet de type ExempleTab , cet objet s'appelle e
       //  on utilise le premier constructeur qui n'a pas de paramètre
      //  il ne faut pas pour autant oublier les parenthèses

        ExempleTab  e=new ExempleTab ( );

       // e est un objet
       // son attribut e.t est un tableau de taille 10
       // remplit aléatoirement
       // pour voir le résultat de ce que l'on a programmé a l'écran il faut afficher
       // le contenu du tableau t.
```

Exercice: suite

```
// pour cela avec l'objet e on lance la méthode afficher
  // un objet peut appeler les méthodes appartenant a sa  classe

   e.afficher ( );   // la méthode afficher n'a pas de paramètres
                          mais on n'oublie  pas   les parenthèses
// on va créer un deuxième objet a l'aide du deuxième constructeur
          // un autre objet implique un nouveau nom
             ExempleTab f=new Exemple (3);
           // f est un objet de type ExempleTab , son attribut f.t est un tableau de
taille 3
            // dont le contenu est aussi rempli aléatoirement
            // pour visualiser ce qu'on a programme on lance la méthode affiche
            // comme on veut le contenu de f.t
           // on appelle afficher avec f
            f.afficher ( );
            // on peut voir que le contenu de e.t n'est pas identique a celui de f.t
            }
}
```

```
public class ExempleTab{
                int t [ ];
                ExempleTab ( ) {   t= new int[10];
                for (int i=0; i<t.length ; i++) {
                t[ i ] = ( int )( Math.random( ) * 10 );
                }
        }
 ExempleTab (int n) {
                t = new int [n];
                for (int i=0; i<t.length ; i++) {
                t[ i ] = ( int )( Math.random( ) * 100 )
                }
        }
        void afficher( ) {
                for (int i=0; i<t.length ; i++) {
                System.out.println ( t[ i ]);
                }
        }
}
```

Résumé

```
Class Principale {
public static void main (String args [ ]) {

ExempleTab  e=new ExempleTab ( );
e.afficher ( );
ExempleTab f=new Exemple (3);
f.afficher ( );
}}
```

Exercice: suite

Les tableaux ont une taille fixée lors de la création.
Si on veut ajouter ou enlever un élément il faut recréer un tableau avec le bon nombre de cases et recopier les éléments gardés.

Exemple *: On se replace dans la classe ExempleTable*
on va supprimer la dernière case du tableau t ,

on se replace dans la classe exemple dans laquelle on a déjà,

Ecrivons la méthode ***supprimer ();***

on va rajouter un entier dans le tableau t

Ecrivons la méthode ***ajouter ();***

Méthode supprimer ()

```
void supprimer (){
            int[]temp;  // déclaration d'un tableau avec un contenu de meme type
            temp = new int[t.length-1];  // creation de temp qui a besoin d'une
                                                    case de moins
for(int i=0; i<temp.length;i++){
             temp[i] = t[i];
               }
t= new int[temp.length];
for(int i=0; i<temp.length;i++){
t[i] = temp[i];
}
}
```

Méthode ajouter ()

```
void ajouter (int v){
        int[]temp;  // declaration d'un tableau avec un contenu de même type
        // on peut le rappelr temp car l'autre n'est connu que dans supprimer

temp = new int[t.length+1];  // creation de temp  qui a besoin d'un case de plus
for(int i=0; i<temp.length-1;i++){
temp[i] = t[i];  // on recopie les cases qui nous intéresse
}
temp[temp.length-1]=v; // on rajoute le contenu de la dernière case

// t a toujours le même contenu , il faut recrée t avec la bonne taille
// et recopier les éléments de temp un a un,

t= new int[temp.length]; // on redonne à t la bonne dimension.
for(int i=0; i<temp.length;i++){
t[i] = temp[i];

}}
```

Exemple 2

On va écrire un programme contenant un tableau de String contenant deux cases une contenant droite l'autre gauche.

On va écrire :

Une méthode pour afficher le tableau.

Ainsi on pourra visualiser a l‘écran ce que nous avons programme

Une méthode qui permute le contenu des deux cases

public class Exemple1{

// attribut

String a[]; // on déclare un tableau de String qui s'appelle a

// constructeur même nom que la classe, pas de paramètre car il a deux cases et on connait précisément le contenu.

Exemple1(){

a= new String[2]; // création

a[0]= ”droite”; // une fois crée on peut le remplir

a[1]= ‘’gauche”; }

//méthode qui affiche le tableau a , ne renvoie rien le type retour est un **void**

// ne prend pas de paramètre

```
void affichage(){
        for(int i=0; i<a.length;i++)
System.out.println(a[i]);
}
```

// méthode qui permute le contenu des deux cases
// il faut une variable temporaire pour faire un cycle et ne pas perdre de données
// la méthode ne renvoie rien et ne prend pas de paramètres extérieurs

```
void echange (){
 String temp;   // la variable temporaire doit  avoir le même type que
                   /: les objets du tableau
  temp = a[0]; // on stocke a[0]
                  // maintenant on peut écraser  a[0] sans perdre la valeur
   a[0]=a[1];
    a[1]=temp;
}
}
Public class PrincipaleExemple1{
public static void main(String a[]){
Exemple1 e = new Exemple1();
e.affichage();
e.echange();
}
}
```

II. LES TABLEAUX A DEUX DIMENSIONS.

Les tableaux ont plusieurs lignes et plusieurs colonnes.

On va faire un tableau de tableaux.

On retrouve donc ce qu'on a fait pour les tableaux a une dimension

La déclaration précise le nom et le type des éléments pour la création on donne en plus la taille ici on en a deux les lignes et les colonnes

Un fois que le type et la taille sont fixées on ne peut pas les changer.

1°) DÉCLARATION

type des éléments Nom de la matrice [ ] [ ] ;

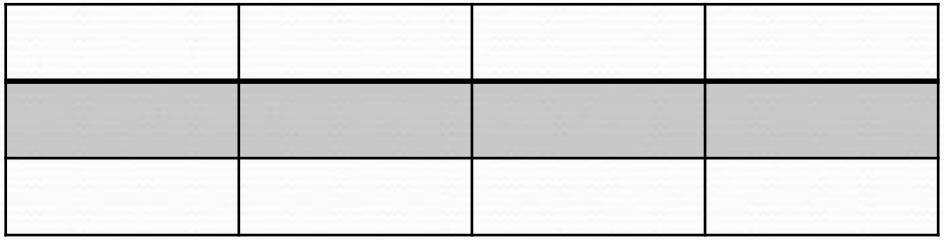
Remarque : La taille n'est pas indiquée lors de la déclaration.
Exemple : int M[ ] [ ];

2°) CRÉATION

On se sert du mot clé new et on indique la taille.

Nom=new type [nombreligne] [nombrecolonne]

Exemple : M= new int [3][4];
Ici, on a un tableau de trois lignes et quatre colonnes, dont les cases sont remplies de 0.



LES TABLEAUX A DEUX DIMENSIONS.

Exemple: **M=new int[2][ ]**
Pour chaque ligne on indique le nombre de colonnes.
M[0]=new int[2];
//// la première ligne a M a 2 colonnes ////
M[1]=new int[1];
//// la deuxième ligne a 1 colonne////

Avant de remplir chaque ligne il faut bien pense à indiquer le nombre de colonnes

Sinon on ne pourra rien stocker sur la ligne

On peut aussi faire des tableaux contenant des objets crées a partir de classes que nous avons programmé.