Support de cours :

Programmation Orientée Objet en Java

Enseignant : Ali ZIDANII

- ali.zidani@isaeg.u-gafsa.tn
- http://ali-zidanii.e-monsite.com/

Auditoire : BC2

2ème Licence Business Computing

A.U : 2024-2025

Références bibliographiques

- Initiation à la programmation orientée objet en Java : Rappels de cours et exercices corrigés, Ben Othmane Chiraz Zribi, Centre de Publication Universitaire, 2003.
- La programmation objet en java, Michel Divay, Dunod, 2006
- Exercices en java, Claude Delannoy, Eyrolles, 2ème édition, 2006
- Cours programmation orientée objet en Java Wassim Messaoudi, 2012.
- Cours programmation orientée objet en Java Walid SAAD, 2015.

Plan du cours

- Chapitre 1 : L'approche Orienté Objet (5)
  1. Programmation traditionnelle vs orientée objet
  2. P.O.O : L'Abstraction
  3. Apports de l'approche objet
  4. Caractéristiques de l'approche objet
- Chapitre 2 : Introduction au langage Java (27)
  1. Historique du langage Java
  2. Caractéristiques du langage Java
  3. Exemple d'une application Java
  4. Les bases du langage Java
- Chapitre 3 : Syntaxe du langage (44)
  1. Les types de données
  2. Les structures de contrôle
  3. Les structures de répétition
- Chapitre 4 : Éléments de Programmation Java (76)
  1. Variables?
  2. Méthodes
  3. Classes

Plan du cours (2)

- Chapitre 5 : Héritage
  1. Redéfinition
  2. Polymorphisme
  3. Interfaces
  4. classes abstraites
- Chapitr6 : Gestion des exceptions (113)
  1. Déclaration
  2. Interception et traitements.
  3. Classes d'exception
- Chapitre 7 : Les entrées/ sorties simple
  1. Flux d'entrée
  2. Flux de sortie
- Chapitre 8 : Les interfaces graphiques en Java
  1. AWT
  2. Swing
  3. GUI

Chapitre 1 :

L'approche Orientée Objet

1. Programmation traditionnelle vs orientée objet
2. P.O.O : L'Abstraction
3. Apports de l'approche objet
4. Caractéristiques de l'approche objet

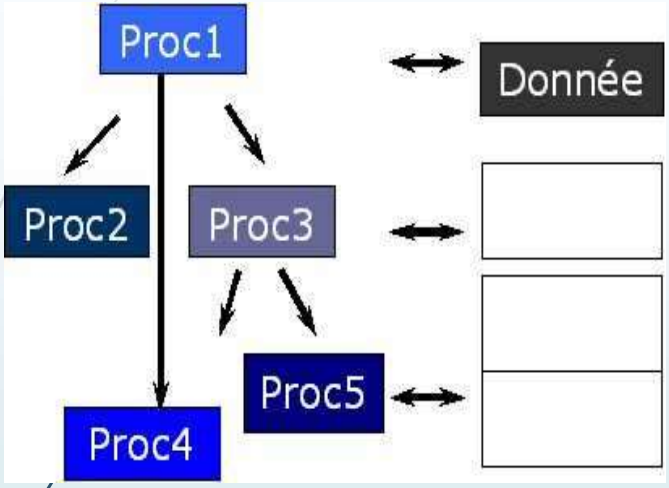
L'approche Orienté Objet

Introduction

- Les langages de programmation ont connu une évolution historique remarquable caractérisée par une volonté continue d'améliorer leur niveau expressif, leur niveau d'abstraction et la qualité des logiciels qu'ils permettent d'obtenir.
- La programmation orientée objet existe depuis l'arrivée du langage Simula en 1967. Cependant, elle n'est vraiment devenue un des paradigmes de la programmation qu'au milieu des années 1980.
- Au contraire de la programmation structurée traditionnelle, la programmation orientée objet met dans une même et unique structure les données et les opérations qui leurs sont associées.

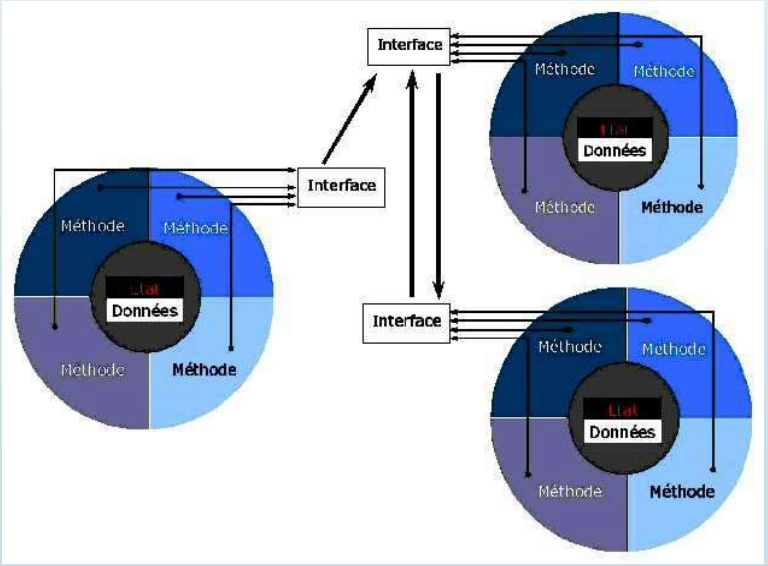
L'approche Orienté Objet

1. Programmation traditionnelle VS Orientée Objet



- **En programmation traditionnelle,** les données et les opérations sur les données sont séparées, les structures de données sont donc envoyées aux procédures et fonctions qui les utilisent.

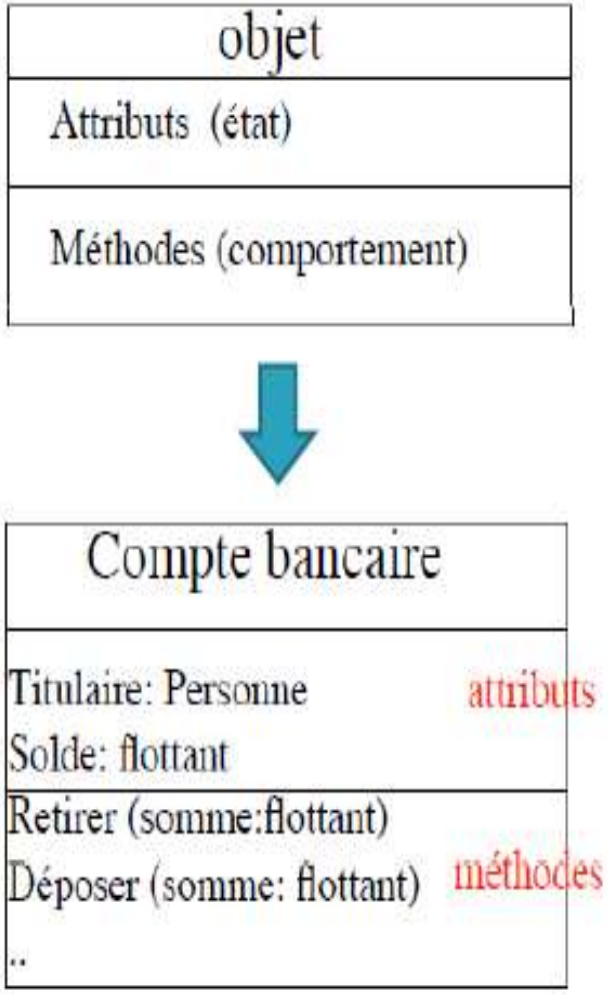
- **En programmation orientée objet :**
  - Pas de séparation des données et des actions.
  - Chaque objet peut invoquer une méthode d'un autre objet qui coopère en répondant à cette demande.



L'approche Orienté Objet

2. P.O.O : Notion d'Objet (1)

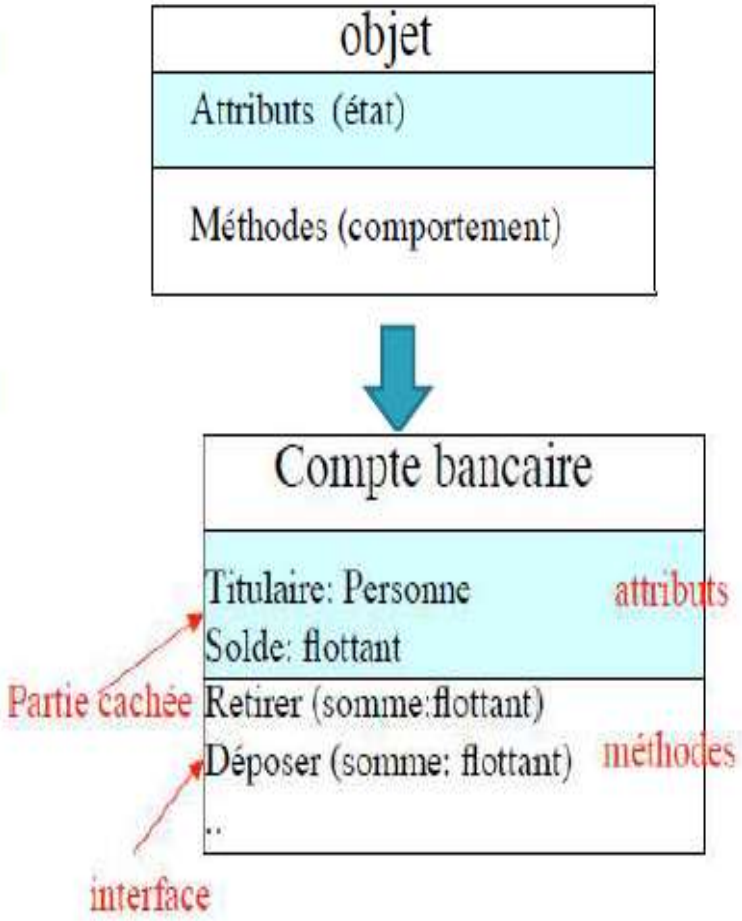
- Un objet est un modèle de données avec:
  - une partie déclarative contenant des variables internes à l'objet, exprimant son état, appelés : champs ou attributs
  - une partie fonctionnelle contenant un ensemble d'opérations appelées méthodes, exprimant le comportement de l'objet.



L'approche Orienté Objet

2. P.O.O : Notion d'Objet (2)

- L'ensemble des attributs de l'objet ne sont accessibles que via les méthodes de celui-ci
  - Encapsulation et protection de données.
- L'ensemble des méthodes définit l'interface à travers laquelle on peut agir sur l'objet (ou le manipuler).



L'approche Orienté Objet

3. Caractéristiques de l'approche objet (1)

Alan Kay résume ainsi cinq caractéristiques principales de l'approche objet :

- **Toute chose est un objet**, il faut penser à un objet comme à une variable améliorée : il stocke des données, mais on peut « effectuer des requêtes » sur cet objet, lui demander de faire des opérations sur lui-même.
- **Un programme** est un ensemble d'objets qui communiquent entre eux en s'envoyant des messages. Pour qu'un objet effectue une requête, on «envoie un message» à cet objet. Plus concrètement, un message est un appel à une fonction appartenant à un objet particulier.
- **Chaque objet** possède son propre espace de mémoire composé d'autres objets. On peut ainsi **cacher la complexité** d'un programme par la simplicité des objets mis en œuvre.
- **Chaque objet est d'un type précis**, chaque objet est une instance (ou une réalisation) d'une classe.
- **Tous les objets d'un type particulier** peuvent recevoir le même message.

L'approche Orienté Objet

3. Caractéristiques de l'approche objet (2)

L'approche Objet est donc fondée sur :

- La notion **d'encapsulation** qui permet de définir des classes qui contiennent des attributs et des méthodes :
  - **L'encapsulation** des données et du code dans une même entité permet de garantir la cohérence des objets. Cette cohérence est indispensable pour réutiliser un objet dans un autre contexte.
  - Il est possible de **changer le fonctionnement interne d'un objet particulier**, **sans modifier la manière de l'utiliser** (c'est à dire le reste du programme)
- La notion d'**héritage qui permet de réutiliser les propriétés partagées entre** les objets afin de : Diminuer le volume de travail à faire, réduire les sources d'erreurs, Ne pas refaire ce qui a été réalisé, réduire la complexité, etc.
- La notion de **polymorphisme qui permet de manipuler de manière identique et la plus** naturelle possible des objets ayant des comportements totalement différents.

L'approche Orienté Objet

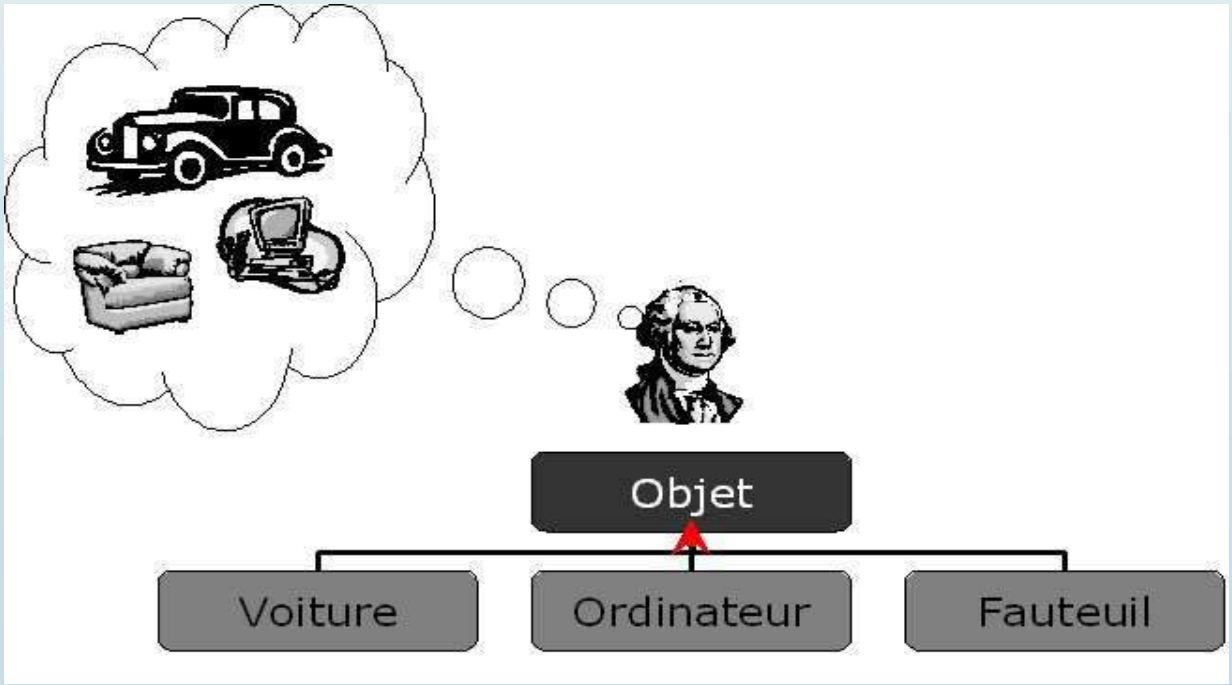
4. Apports de l'approche Objet (1)

- **L'approche objet offre :**
  - Un niveau **d'abstraction** facilitant la modélisation du monde réel et la réduction de la complexité des problèmes à résoudre.
  - Des outils d'analyse et de conception permettant de promouvoir la réutilisation et le partage des codes et des conceptions
  - Des techniques de programmation augmentant la productivité des logiciels et facilitant leur maintenance..
  - Sécuriser les programmes en interdisant ou autorisant l'accès aux objets et aux autres parties du programme,

L'approche Orienté Objet

5. P.O.O : L'Abstraction (1)

- **L'abstraction** dans l'approche objet permet la représentation des entités du monde réel sous forme d'entités informatiques de la manière la plus naturelle.
- Etablir une association entre le modèle du problème à résoudre et le modèle de la machine :



L'approche Orienté Objet

5. P.O.O : L'Abstraction (2)

- L'abstraction est une représentation des éléments du monde réel **«objets réels»** dans l'espace du problème (la machine) en tant qu'**«objets informatiques».**

  ➔Décrire le problème avec les termes mêmes du problème plutôt qu'avec les termes de la machine.

  - **un programme traitant des images doit manipuler des structures de données représentant des images, et non leur traduction sous forme de suite de 0 et de 1.**
  - **un programme de gestion de personnel doit représenter des personnes avec toutes les informations pertinentes, qu'il s'agisse de texte, de date, de nombres ou autre.**

- **L'idée est d'adapter le programme à l'esprit du problème réel en ajoutant de nouveaux types « objets ».**

Quand on lit le code décrivant la solution, on lit aussi quelque chose qui décrit le problème.

L'approche Orienté Objet

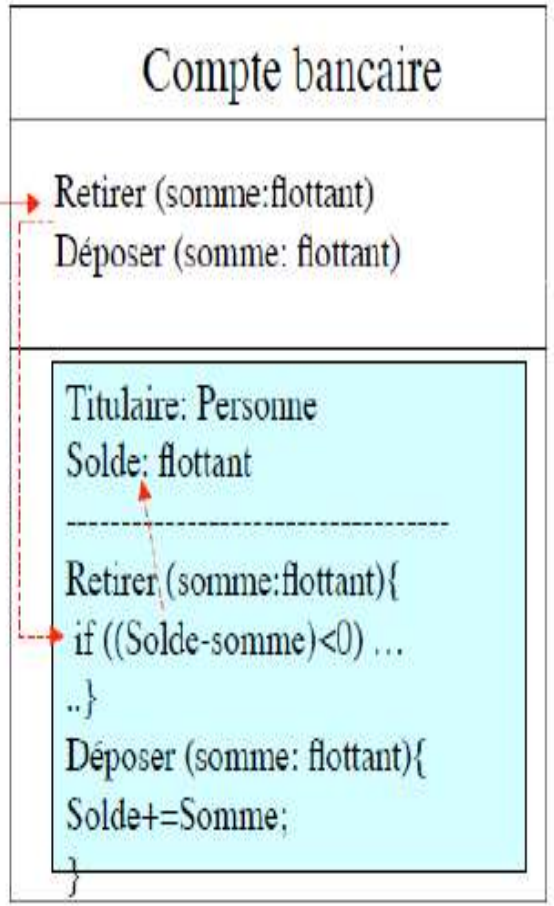
5. P.O.O : L'Abstraction (3)

- La complexité des problèmes et la capacité à les résoudre sont directement proportionnelles au type et à la qualité de nos **capacités d'abstraction**.
- **Plusieurs niveaux d'abstraction pour un objet:**
  - de point de vue concepteur, un ordinateur est un objet formé d'un ensemble d'éléments physiques appelés matériels (hardware).
  - de point de vue informaticien, un ordinateur est un objet résultant d'un assemblage hardware et d'un ensemble de programmes appelé logiciels (software).
  - de point de vue utilisateur, un ordinateur est une boite noire qui offre un certain nombre de fonctions ou de services qui permettent d'interagir avec elle.

L'approche Orienté Objet

6. Encapsulation

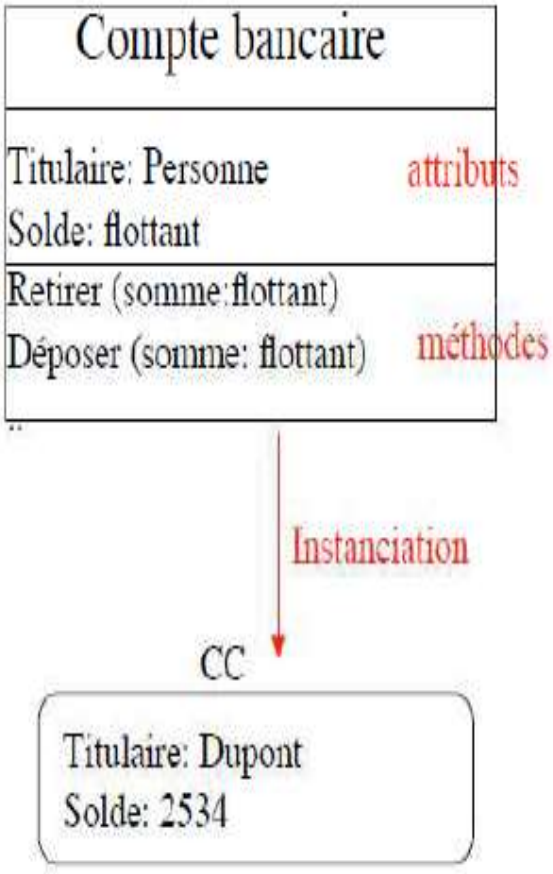
- Partie visible de l'extérieur d'un objet
  - Opérations.
- Partie invisible de l'extérieur d'un objet
  - Données (Attributs)
  - Description du comportement des opérations



L'approche Orienté Objet

7. Notion de classe

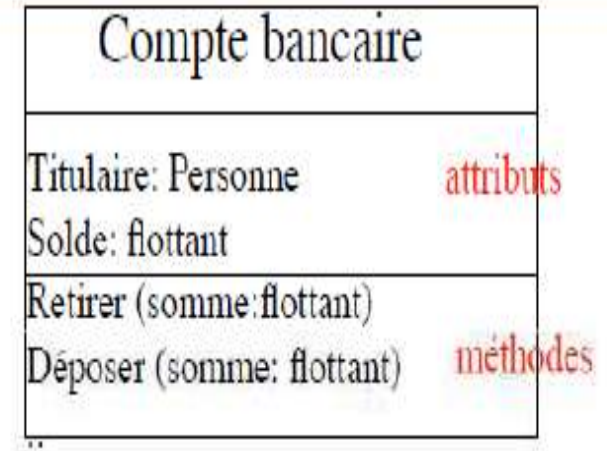
- Une classe définit la description d'une famille d'objets ayant la même structure (partie déclarative) et le même comportement (partie procédurale)
  - Description d'un concept
- Une instance correspond à un exemplaire concret de la classe, avec des valeurs d'attributs définis(l'objet)
  - Objet concret créé à partir du recette fourni par la définition de classe (instanciation)

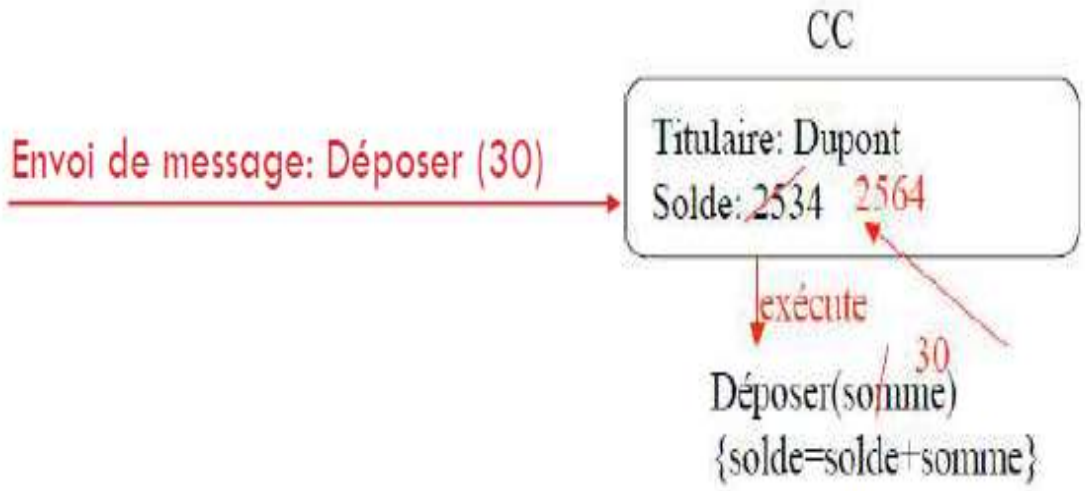


L'approche Orienté Objet

7. Notion de classe : Mode de fonctionnement

- Envoi de messages entre objets:
  - Un message contient le nom d'une méthode, et la liste des arguments
  - La réponse à la réception d'un message consiste à exécuter la méthode associée





L'approche Orienté Objet

7. Notion de Classes: sous classes et instances

- Programmer une bibliothèque de manipulation de figures géométriques: les polygones

La classe CARRE
Nombre de coté: 4 Nombre de sommets: 4 Liste de sommets Longueur cote:
Périmètre: 4* Longueur cote Surface: Longueur cote^2 Affichage Rotation (sommet) Rotation (Axe) ..

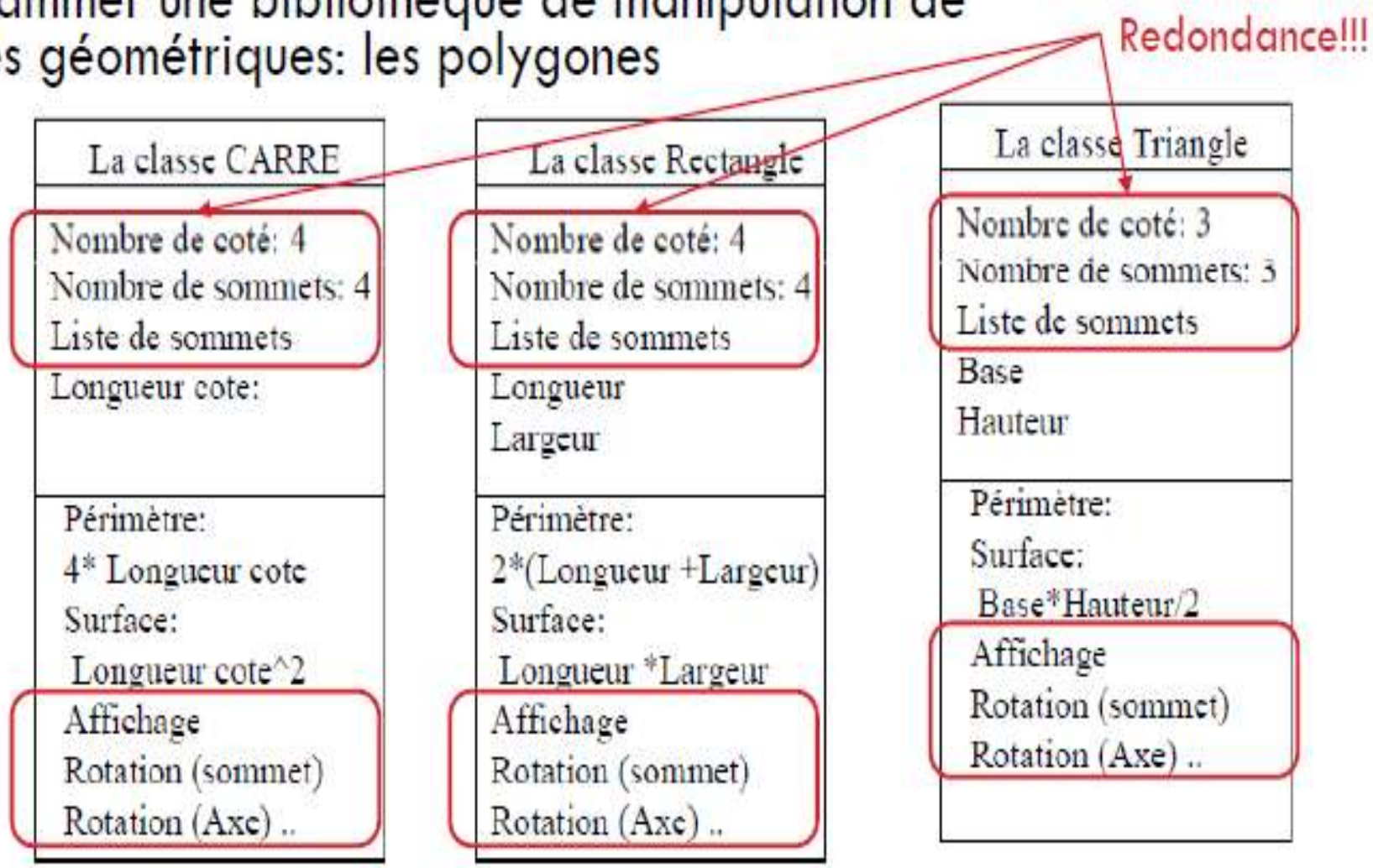
La classe Rectangle
Nombre de coté: 4 Nombre de sommets: 4 Liste de sommets Longueur Largeur
Périmètre: 2*(Longueur +Largeur) Surface: Longueur *Largeur Affichage Rotation (sommet) Rotation (Axe) ..

La classe Triangle
Nombre de coté: 3 Nombre de sommets: 3 Liste de sommets Base Hauteur
Périmètre: Surface: Base*Hauteur/2 Affichage Rotation (sommet) Rotation (Axe) ..

L'approche Orienté Objet

7. Notion de Classes: sous classes et instances

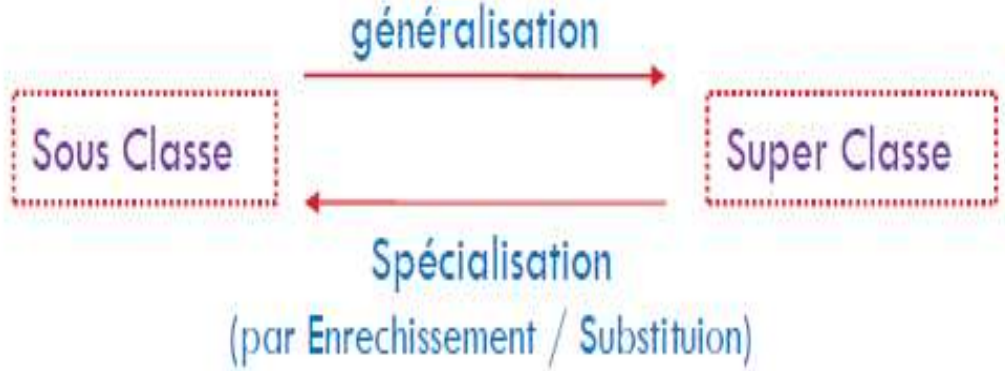
- Programmer une bibliothèque de manipulation de figures géométriques: les polygones



L'approche Orienté Objet

7. Notion de Classes: sous classes et instances

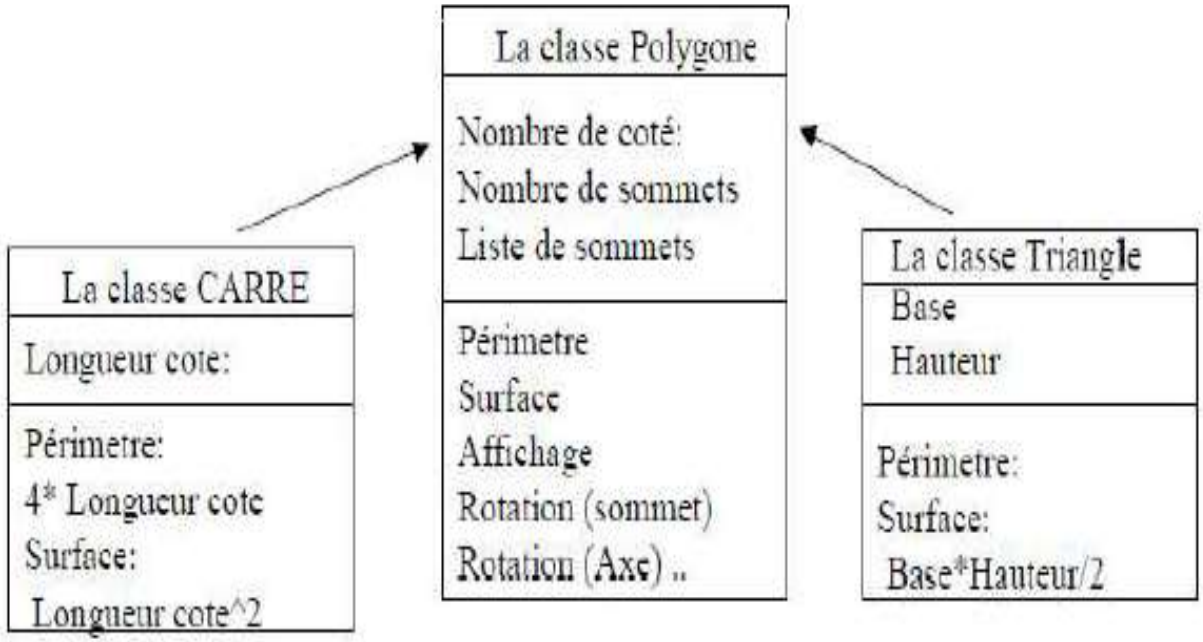
- Organisation des objets en hiérarchies
  - A la racine une classe regroupant les propriétés communes à toutes les autres classes: classe mère (super classe)
  - Descendre dans la hiérarchie par spécialisation (plus de propriétés communes): création d'une sous classe
- La hiérarchie des classes définit le graphe d'héritage.
  - Aux niveaux supérieurs, on dispose des classes abstraites
  - Au niveau le plus bas : les classes concrètes utilisées pour l'instanciation
  - Récupération des propriétés (attributs, méthodes) au niveau le plus bas grâce à l'héritage



L'approche Orienté Objet

8. Héritage (1)

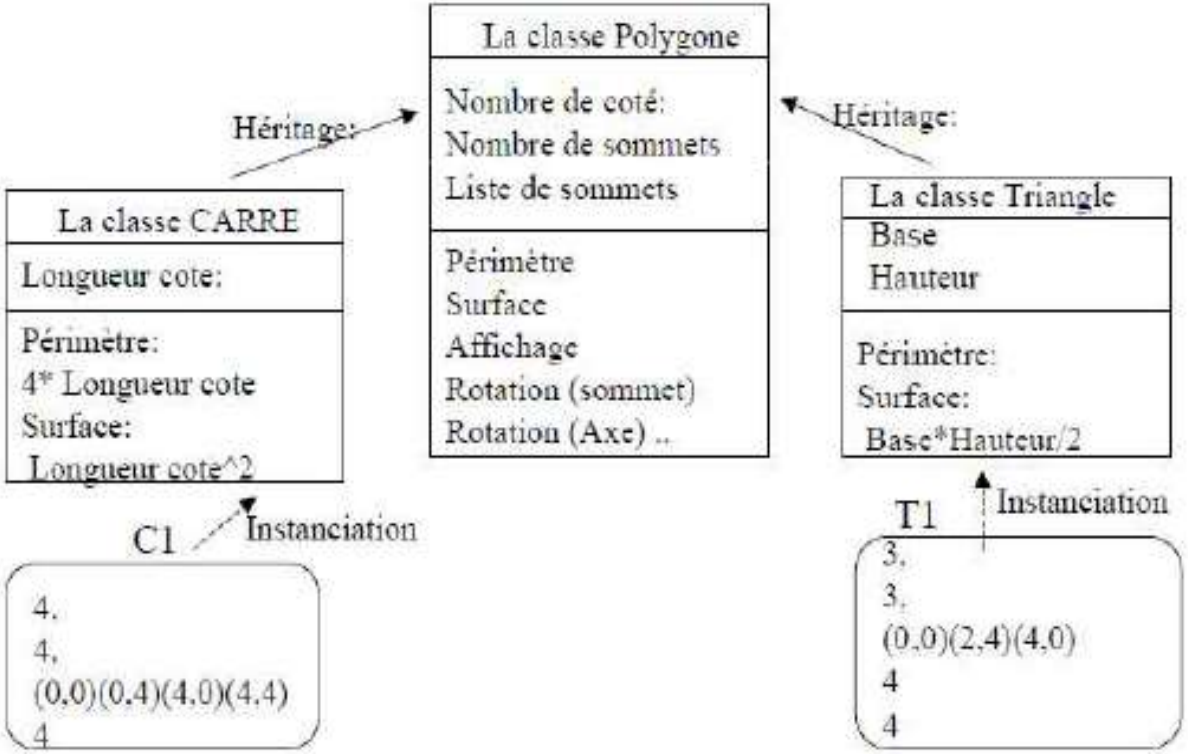
- Programmer une bibliothèque de manipulation de figures géométriques: les polygones
  - Les classes CARRE et Triangle dérivent de la classe Polygone



L'approche Orienté Objet

8. Héritage (2)

- Programmer une bibliothèque de manipulation de figures géométriques: les polygones



L'approche Orienté Objet

9. Polymorphisme (1)

- Objets Polymorphiques
  - Les objets sont dits polymorphes car ils possèdent plusieurs types: le type de leurs classes et les types des classes ascendantes.
- On fait A ← B ; A possède un type initial, le type que l'on a donné à l'instanciation
  - Cette affectation est valide si et seulement si B est de même type initial que A ou d'un type dérivé de celui de A.

- Exemple:
  - La classe Voiture hérite de la classe Véhicule
  - monVehicule est une instance de type Véhicule;
  - maVoiture est une instance de type Voiture;

(1) maVoiture ← monVehicule : est-ce une affectation valide?

(2) monVehicule ← maVoiture : est-ce une affectation valide?

  - (1) est fausse, (2) est vraie

L'approche Orienté Objet

9. Polymorphisme (2)

- Méthodes Polymorphiques
  - Une classe peut comporter plusieurs méthodes de même nom et possédant des arguments différents
  - Une classe dérivée peut comporter plusieurs méthodes de même nom et meme arguments mais possédant un code différent selon la classe qui reçoit le message
- Exemple:
  - Voiture.rouler(chemin) la façon dont la voiture roule ne dépend que du chemin.
  - Voiture.rouler(chemin,meteo) la façon dont la voiture roule dépend du chemin et de la météo.
  - CARRE.surface() et Triangle.surface(): même nom de méthode (surface), mais code différent

L'approche Orienté Objet

Conclusion

- Ce cours se propose d'introduire et de clarifier la terminologie, les concepts et les techniques de base associées à l'approche objet et de les appliquer au langage de programmation Java.
- Ce cours présente d'abord le langage Java, ses concepts de base, son syntaxe, etc. Puis, le troisième chapitre invoque la Programmation Orientée Objet en Java.

Chapitre 2 :

Introduction au langage Java

1. Historique du langage Java
2. Caractéristiques du langage Java
3. Exemple d'une application Java
4. Les bases du langage

Introduction au langage Java

1. Historique du langage Java

- **1990 :** Sun Développe un nouveau langage plus adapté à la réalisation de logiciels embarqués, appelé OAK par :
- Petit, fiable et indépendant de l'architecture
- Destiné à la télévision interactive
- Non rentable sous sa forme initiale
- **1993 :** le WEB « décolle », Sun redirige ce langage vers Internet : les qualités de portabilité et de compacité du langage OAK en ont fait un candidat parfait à une utilisation sur le réseau. Cette réadaptation prit près de 2 ans.
- **1995 :** Sun changeOAK en Java (*nom de la machine à café autour de laquelle se réunissait James Gosling et ses collaborateurs)*
- **1996 :** Les Java Développement Kits (JDK) sont disponibles gratuitement pour la plupart des machines du marché.

Introduction au langage Java

2. Caractéristiques du langage Java (1)

➨ **Le langage Java est familier :**

Java est un langage familier très proche du langage C , C++. Par exemple:

- Même types de base que C++ (int, float, double, etc.),

  Même formes de déclarations que C++,

  Même structure de contrôle que C++ (if, while, for, etc.).

➨ **Le langage Java est simple :**

Java est un langage simple par rapport au langage C et C++.

- Il n'y a plus de pointeurs et ses manipulations
- Java se charge (presque) de restituer au système les zones mémoire inaccessibles et ce sans l'intervention du programmeur.

Introduction au langage Java

2. Caractéristiques du langage Java (2)

- **Le langage Java est orienté objet :**

Paquetage pour la réutilisation :

- java.lang : classes de base
- java.io : entrée/sortie
- java.awt : interfaces graphiques
- java.net : communication réseaux (socket et URL)
- java.applet : API Applet
- java.util : classes outils

- **Le langage Java est distribué :**

Supporte des applications réseaux (protocoles de communication java.net)

- URL : permet l 'accès à des objets distants
- RMI : Remote Method Invocation
- Programmation d'applications Client/Serveur : classe Socket
- Manipulation de fichier local ou fichier distant identique : indifférence à la localisation.

Introduction au langage Java

2. Caractéristiques du langage Java (3)

- **Le langage Java est interprété :**

Un programme Java n'est pas compilé en code machine ;

- Il sera compilé en code intermédiaire interprété nommé ByteCode.
- Lors de l'exécution le ByteCode sera interprété à l'aide d'une machine dite virtuelle JVM (Java Virtual Machine).

- **Le langage Java est portable et indépendant des plates-formes :**

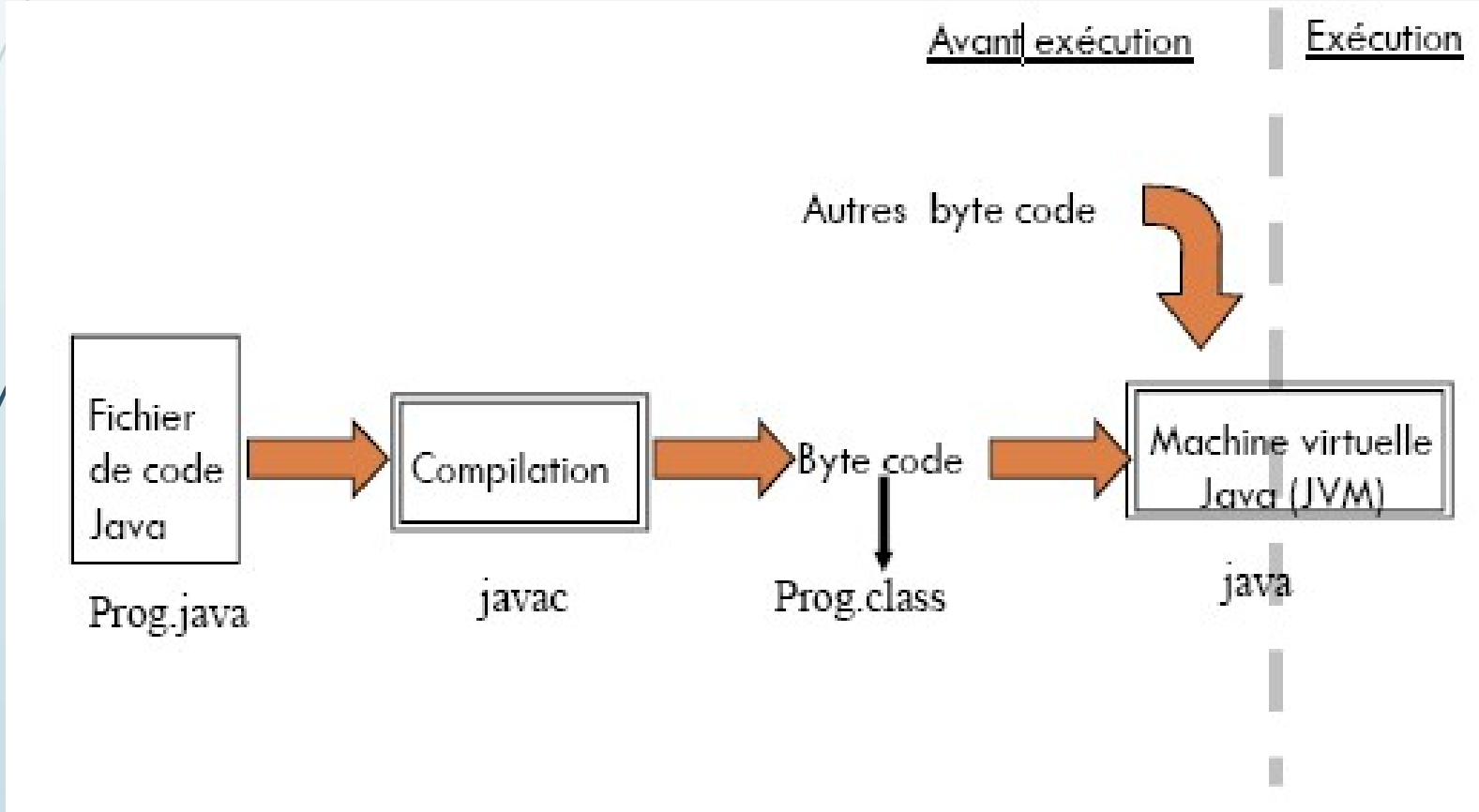
Le code intermédiaire produit « ByteCode » est indépendant des plates-formes.

- Il pourra être exécuté sur tous types de machines et systèmes pour peu qu'ils possèdent l'interpréteur de code Java « JVM ».

Introduction au langage Java

2. Caractéristiques du langage Java (4)

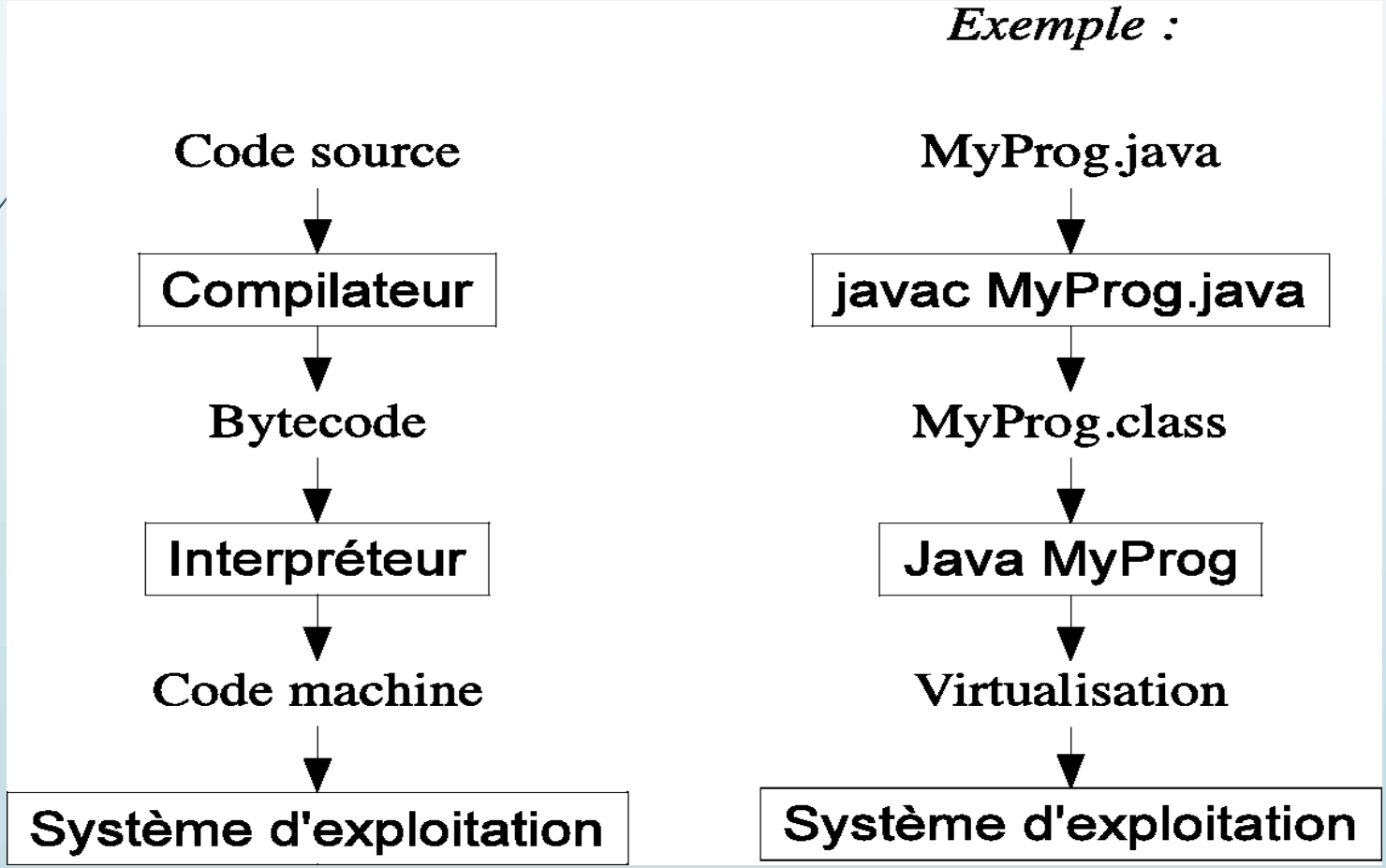
Compilation et exécution d'une application Java :



Introduction au langage Java

2. Caractéristiques du langage Java (5)

Compilation et exécution d'une application Java :



Introduction au langage Java

2. Caractéristiques du langage Java (6)

Compilation et exécution d'une application Java :

- **Byte-code :** fichier crée lors de la compilation d'un programme Java, il ne peut pas être exécuté directement par le processeur.

- **JVM (Java Virtual Machine) :** Programme capable d'interpréter les instructions contenues dans les fichiers ***ByteCode*** Java afin de les exécuter.

Introduction au langage Java

3. Exemple d'une application Java (1)

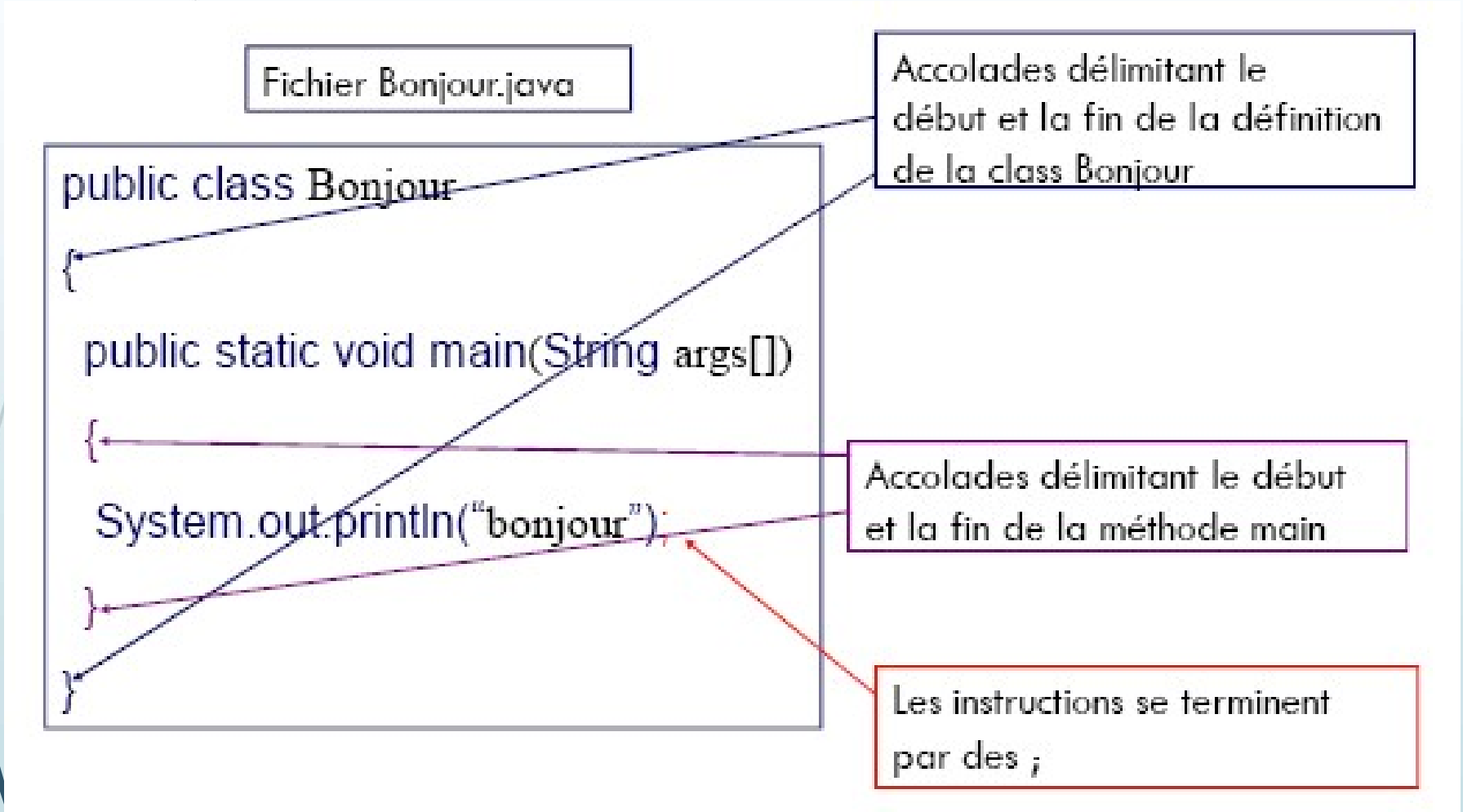
Pour pouvoir faire un programme exécutable il faut toujours une classe qui contienne une méthode particulière, la méthode **«main»**

c'est le point d'entrée dans le programme : le microprocesseur sait qu'il va commencer à exécuter les instructions à partir de cet endroit.

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

Introduction au langage Java

3. Exemple d'une application Java (2)



Introduction au langage Java

3. Exemple d'une application Java (3)

Fichier Bonjour.java

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

Une méthode peut recevoir des paramètres. Ici la méthode main reçoit le paramètre args qui est un tableau de chaîne de caractères.

Introduction au langage Java

3. Exemple d'une application Java (4)

Fichier Bonjour.java

Le nom du fichier est nécessairement celui de la classe avec l'extension .java en plus. Java est sensible à la casse des lettres.

Compilation en bytecode java dans une console DOS:
javac Bonjour.java
Génère un fichier Bonjour.class
Exécution du programme (toujours depuis la console DOS) sur la JVM :
java Bonjour
Affichage de « bonjour » dans la console

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

Introduction au langage Java

3. Exemple d'une application Java (4)

```
class Rectangle {
int longueur;
int largeur;
int origine_x;
int origine_y;
void deplace(int x, int y) {
this.origine_x = this.origine_x + x;
this.origine_y = this.origine_y + y;
}
int surface() {
return this.longueur * this.largeur;
}
}
```

4. Méthode constructeur

Chaque classe doit définir une ou plusieurs méthodes particulières appelées des **constructeurs**. Un constructeur est une méthode invoquée lors de la création d'un objet. Cette méthode, qui peut être vide, effectue les opérations nécessaires a l'initialisation d'un objet. **Chaque constructeur doit avoir le même nom que la classe où il est défini et n'a aucune valeur de retour** (c'est l'objet créé qui est renvoyé). Dans l'exemple précédent de la classe rectangle, le constructeur initialise la valeur des données encapsulées,

Plusieurs constructeurs peuvent être définis s'ils acceptent des paramètres d'entrée différents.

```
class Rectangle {
    ...
                Rectangle(int lon, int lar) {
                    this.longueur = lon;
                    this.largeur = lar;
                    this.origine_x = 0;
                    this.origine_y = 0;
    }
    ...
}
```

Introduction au langage Java

5. Instanciation

Un objet est une instance (anglicisme signifiant <<cas .> ou << exemple .>) d'une classe et est référencé par une variable ayant un état (ou valeur). Pour créer un objet, il est nécessaire de déclarer une variable dont le type est la classe a instancier, puis de faire appel a un constructeur de cette classe. L'exemple ci-dessous illustre la création d'un objet de classe Cercle en Java

Instanciation sans paramètres d'entrées

```
Cercle mon_rond;
mon_rond = new Cercle();
```

Instanciation avec paramètres d'entrées

```
Rectangle mon_rectangle = new Rectangle(15,5);
```

6. Accès aux variables et aux méthodes (1)

Pour accéder a une variable associée a un objet, il faut préciser l'objet qui la contient. Le symbole '.' sert a séparer l'identificateur de l'objet de l'identificateur de la variable. Une copie de la longueur d'un rectangle dans un entier temp s'écrit ~

```
int temp = mon_rectangle.longueur;
```

La même syntaxe est utilisée pour appeler une méthode d'un objet. Par exemple ~

```
mon_rectangle.deplace(10,-3);
```

Introduction au langage Java

6. Accès aux variables et aux méthodes (2)

Pour référencer l'objet "courant" (celui dans lequel se situe la ligne de code), le langage Java fournit le mot-clé this. Celui-ci n'a pas besoin d'être instancié et s'utilise comme une variable désignant l'objet courant. Le mot-clé this est également utilisé pour faire appel a un constructeur de l'objet courant. Ces deux utilisations possibles de **this** sont illustrées dans l'exemple suivant ~

```
class Carre {
    int cote;
    int origine_x;
    int origine_y;
    Carre(int cote, int x, int y) {
        this.cote = cote;
        this.origine_x = x;
        this.origine_y = y;
    }
    Carre(int cote) {
    // this.cote = cote;  this.origine_x = 0;  this.origine_y = 0;

    this(cote, 0, 0);
    }
}
```

Chapitre 3 :

Syntaxe du langage Java

1. Historique du langage Java
2. Caractéristiques du langage Java
3. Exemple d'une application Java
4. Les bases du langage

Syntaxe du langage Java

1. Les commentaires (1)

- Tout programme (grand ou petit, simple ou complexe) contient (ou devrait contenir) des commentaires.
- Ils ont pour but d'expliquer :
  - Ce qu'est sensé faire le programme,
  - Les conventions adoptées,
  - Tout autre information rendant le programme lisible à soi même et surtout à autrui.
- Java dispose de trois types de commentaires :
  - Les commentaires multi-lignes,
  - Les commentaires lignes,
  - Les commentaires de type documentation.

Syntaxe du langage Java

1. Les commentaires (2)

- **Commentaires en lignes**

Les commentaires lignes débutent avec les symboles « // » et qui se terminent à la fin de la ligne.

```
// Ce programme imprime la chaîne
// de caractères " bonjour " à l'écran
...
```

Ils sont utilisés pour des commentaires courts qui tiennent sur une ligne.

Syntaxe du langage Java

1. Les commentaires (3)

- **Commentaires multilingues :**

Un commentaire multiligne commence par les caractères « /* » et se terminent par « */ »

```
/* Ce programme imprime la chaîne
de caractères "bonjour" à l'écran
*/
```

- A l'intérieur de ces délimiteurs toute suite de caractères est valide (sauf évidemment « */ »).

Syntaxe du langage Java

1. Les commentaires (4)

Commentaires de types documentation :

- Ces commentaires, appelés aussi commentaires javadoc, servent à documenter les classes que l'on définit.
- Ces commentaires sont encadrés entre ``/**'' et ``*/'' .

```
/** Documentation de la classe .
*/
```

- Java exige que ces commentaires figurent avant la définition de la classe, d'un membre de la classe ou d'un constructeur.
- Ces commentaires servirons à produire automatiquement (avec l'outil javadoc) la documentation sous forme HTML à l'image de la documentation officielle de SUN.

Syntaxe du langage Java

2. Les types de données (1)

Les types numériques entiers

	byte	short	int	long
Taille en bits	8	16	32	64
Etendue	-128 .. 127	-32768 .. 32767	-2^{31} .. $2^{31}-1$	-2^{63} .. $2^{63}-1$

- **Le bit y est utilisé pour décrire les tailles.**
  - Un bit ne peut prendre que deux valeurs (0 ou 1).
  - **n** bits ne peuvent définir que 2^n valeurs.
  - Un **octet** ou **byte** en anglais = 8 bits.
  - Un octet peut donc contenir 2^8 (soit 256) valeurs distinctes.

Syntaxe du langage Java

2. Les types de données (2)

Les types numériques flottants

- De même que pour les types numériques, il existe deux types pour les nombres flottants :
  - float,
  - double.
- la seule différence résidant dans la taille utilisée pour stocker une valeur de ce type.

	float	double
Taille en bits	32	64
Exemple de valeur	3.25f	3.25

Syntaxe du langage Java

2. Les types de données (3)

- **Le type booléen**

Ce type accepte seulement deux états :

- l'un est nommé **TRUE** : Symbolise un état d'acceptation,
- l'autre, nommé **FALSE**, Symbolise un état de réfutation.

Syntaxe du langage Java

2. Les types de données (4)

- **Le type Caractère**
- Ce type, introduit par le mot clé **char**, et permet la gestion des caractères.
- **Java** utilise le codage de caractères universel **Unicode** qui est un extension du codage **ASCII**.
- Le codage **ASCII** utilise **8** bits et permet de représenter seulement **128** caractères.
- Le codage **Unicode** permet la portabilité du code produit.

Il utilise **16** bits pour représenter un caractère.

- **65536** caractères possibles.
- Codage des caractères d'alphabets Cyrillique, Hébreux, Arabe, Chinois, Grec, ...

Syntaxe du langage Java

3. Déclaration et initialisation des variables

- **Syntaxe :**

Type identificateur [= constante ou expression];

- **Exemples :**

- int NbredeMois = 12 ;
- int NbredeMois = 4*3 ;
- boolean Unboolean = false ;
- float Unfloatant = 1.3f ;
- double Undouble = 1.3 ;
- char Uncaractère = 'c' ;
- String Unstring= **" bonjour "** ;

- Et éventuellement, un « modificateur d'accès ou de visibilité » :

final float pi=3.14159

Syntaxe du langage Java

4. Les opérateurs (1)

Opérateurs unaires	Action	Exemple
-	négation	i=-j
++	incrémentation de 1	i=j++ ou i=++j
--	décrémentation de 1	i=j-- ou i=--j

➔ **++ et -- peuvent préfixer ou postfixer la variable.**

- i = j++ : post-incrémentation

La valeur en cours de j est affectée à i et ensuite la valeur de j est incrémentée de 1.

- i = ++ j : pré-incrémentation

La valeur en cours de j est incrémentée de 1 et ensuite la valeur de j est affectée à i.

Syntaxe du langage Java

4. Les opérateurs (2)

Opérateurs binaires	Action	Exemple	Syntaxe équivalente
+	addition	i = j+k	
+=		i += 2	i= i + 2
-	soustraction	i = j – k	
-=		i -= j	i= i - j
*	multiplication	x=2*y	
*=		x *=x	x= x * x
/	division (tronque si les arguments sont entiers)	i =j/k	
/=		x /= 10	x= x /10
%	modulo	i = j %k	
%=		i %=2	i= i %2

Syntaxe du langage Java

4. Les opérateurs (3)

- Le résultat d'une comparaison est une valeur booléenne **(vrai ou faux)**
- Dans le langage Java, le résultat d'une comparaison est **True ou False**

Opérateurs relationnels	Action	Exemple
<	plus petit que	x<i
>	plus grand que	i>100
<=	plus petit ou égal que	j<=k
>=	plus grand ou égal que	c>='a'
==	égal à	i==20
!=	différent de	c!='z'

Syntaxe du langage Java

4. Les opérateurs (4)

Opérateurs logiques	Action	Exemple
!	négation	!p
&	ET	p & (i<10)
|	OU	p | q
^	OU exclusif	p ^ false

Syntaxe du langage Java

4. Les opérateurs (5)

- **L'opérateur ternaire**

Cette expression est une sorte de si-alors-sinon sous forme d'expression :

- a = (condition e) ? x : y
- si la condition e est vraie alors a vaut x sinon elle vaut y.

- **Exemple :** a = (v==2) ? 1 : 0;
- Cette expression affecte à la variable a la valeur 1 si v vaut 2, sinon affecte à la variable a la valeur 0.

Syntaxe du langage Java

5. Structure de contrôle: if (1)

Syntaxe :

```
if (expression booléene ) instruction ;
```

Exemple :

Résultat d'affichage

```
public class ifApplication1 {

        public static void main(String[ ] args)
        {
                int i = 4 ;
            if ( i % 2 != 0 ) System.out.print (" i est impair ");
        }
}
```

Attention : L'expression logique attendue est obligatoirement de type boolean.

Syntaxe du langage Java

5. Structure de contrôle: if – else (2)

Syntaxe

```
if (expression booléenne )
{
     instruction 1 ;
     ……
     instruction i ;
}
else
{
     instruction j ;
     ……
     instruction n ;
}
```

Résultat d'affichage

```
i est pair
```

Exemple

```
public class if_elseApplication {

   public static void main(String[ ] args)
   {
      int i = 4 ;
        if (i % 2 == 0 )
        {
            System.out.println (" i est pair " );
        }
        else
        {

          System.out.println (" i est impair " );
        }
   }
}
```

Syntaxe du langage Java

5. Structure de contrôle: if – else – if (3)

Syntaxe

```
if (expression booléene 1)
{
      instruction 1 ;
      ……
      instruction i ;
}
else if (expression booléene 2 )
{
      instruction j;
      ……
      instruction m ;
}
else
{
      instruction m+1 ;
      ……
}
```

Exemple

```
public class if_else_ifApplication {

   public static void main(String[ ] args)
   {
      int i = 1, j =2 ;
      if (i == j )
      {
         System.out.println (" i est égal à j " );
      }
      else if ( i>j)
      {
       System.out.println (" i est supérieur à j" );
      }
      else
      {
        System.out.println (" i est inférieur à j " );
      }
   }
}
```

Résultat d'affichage

```
i est positive
```

Syntaxe du langage Java

5. Structure de contrôle: switch (4)

Syntaxe

```
switch (variable)
{
   case valeur 1 :
      instr1_1; instr1_2; ...
      break;
   ...
   case valeur N :
      instrN_1; instrN_2;
      break;
   default: /* optionnel */
     instrD_1; instrD_2; ...
     break;
}
```

Résultat d'affichage

```
1
```

Exemple

```
public class switchApplication {

   public static void main(String[ ] args)
   {
      int i = 1,
        switch (i)
        {  case 0 :
                   System.out.println (" 0 " );
                   break;
           case 1 :
                   System.out.println (" 1 " );
                   break;
           case 2 :
                   System.out.println (" 2 " );
                   break;
           default :
                   System.out.println (i);
                   break;
        }
   }
}
```

Syntaxe du langage Java

6. Structure Itérative: for (1)

```
public class forApplication {

        public static void main(String[ ] args)
        {
                int somme = 0;
                for ( int i = 1 ; i < 4 ; i++)
                {
                   somme = somme + i ;
                }
                System.out.println (somme );
        }
}
```

Résultat d'affichage

```
6
```

Fonctionnement :

- initialisation du compteur,
- comparaison avec max,
- réalisation des instructions,
- Incrémentation du compteur et on recommence.

Syntaxe du langage Java

6. Structure Itérative: for (continue) (2)

Utilisation :

- Pour ignorer le reste de la boucle et reprendre l'exécution à
- l'itération suivante de la boucle.

```
public class continueApplication {

        public static void main(String[ ] args)
        {
                int compteur =0;

                for(compteur=0 ; compteur< 6 ; compteur++ )
                {
                        if (compteur == 4) continue;
                        System.out.println (compteur );
                }
        }
}
```

Résultat d'affichage

```
0
1
2
3
5
```

Syntaxe du langage Java

6 Structure Itérative: while (3)

```
public class whileApplication {

    public static void main(String[ ] args)
    {
        int compteur =1; int max = 4 ; int somme = 0;

        while (compteur < max )
        {
            somme = somme + compteur ;
            compteur++;
        }

        System.out.println (somme );
    }
}
```

Résultat d'affichage

```
6
```

Syntaxe du langage Java

6 Structure Itérative: do-while (4)

```
public class do_WhileApplication {

    public static void main(String[ ] args)
    {
        int compteur =1; int max = 4 ; int somme = 0;

        do
        {
            somme = somme + compteur ;
            compteur++;
        }
        while (compteur < max ) ;

        System.out.println (somme );
    }
}
```

Résultat d'affichage

```
6
```

Syntaxe du langage Java

6 Structure Itérative: while (break) (5)

Utilisation :

Pour sortir d'une structure de boucle avant que la condition du test soit remplie.

- Quand la boucle rencontre une instruction break, elle se termine immédiatement en ignorant le code restant.

```
public class breakApplication {

    public static void main(String[ ] args)
    {
        int compteur =0;

        while (compteur < 10 )
        {
            System.out.println (compteur );
            compteur++;
            if (compteur == 5) break;
        }
    }
}
```

Résultat d'affichage

```
0
1
2
3
4
```

Syntaxe du langage Java

7 Les conversions des types (1)

Il y a 2 catégories de conversions possibles :

1. Conversions explicites :

celles faites sur une demande explicite par un programmeur.

2. Conversions implicites :

celles faites automatiquement par un compilateur **:**

- lors d'une affectation,
- lors d'une opération arithmétique,
- lors d'un passage de paramètres (lors de l'invocation d'une méthode)

Syntaxe du langage Java

7 Les conversions des types (2)

Conversion explicite :

- ***Objectif :***
  Changer le type d'une donnée si besoin.

- ***Comment ?*** :
  Préfixer l'opérande par le type choisi.
  Encadrer le type choisi par des parenthèses.

- ***Exemple :***
  char c = '5' ;
  int I = **(int) c** ;

Syntaxe du langage Java

8 Les tableaux (1)

- **Déclaration d'un tableau à une dimension :**
  - Syntaxe :

    <Type> <nom_tableau> [ ] ;

ou encore :

<Type> [ ] <nom_tableau> ;

  - **Exemple :**

    int [] Notes ; // Ou : int Notes [] ;

- **Création d'un tableau à une dimension :**
  - Syntaxe :

    <Nom_tableau> = new <type> [<dimension>] ;

  - **Exemple :**

    Notes = new int[50];

    float T[]= new float[20];

Syntaxe du langage Java

8 Les tableaux (2)

- **Utilisation d'un tableau à une dimension :**

L'accès à un élément d'un tableau :

- Le premier élément commence à l'indice 0
- Le dernier élément a pour indice (n-1) (n étant la dimension du tableau)

Exemple :

```
Notes[0] = 15 ;
System.out.println(Notes[3]);
```

La taille d'un tableau est contenue dans une variable prédéfini : « length » :

```
int taille = Notes.length;
```

Syntaxe du langage Java

8 Les tableaux (3)

- **Déclaration d'un tableau bidimensionnel:**
- Syntaxe :

  <Type> <nom_tableau> [ ] [ ];

ou encore :

  <Type> [ ] [ ]<nom_tableau> ;

- **Exemple :**

  int [][] Matrice ;

- **Création d'un tableau à 2 dimensions :**
- Syntaxe :

  <Nom_tableau> = new <type> [<dimension 1>] [<dimension 2>] ;

- **Exemple :**

  Matrice = **new int[3][4];**

Syntaxe du langage Java

8 Les tableaux (4)

- **Exemple d'utilisation :**

```
public class tabeauApplication {

        public static void main(String[ ] args)
        {
                int[ ] notes = new int[50]; // création
                System.out.println(notes.length);
                notes[51]=10;

        }
}
```

Résultat d'affichage

```
50
java.lang.ArrayIndexOutOfBoundsException: 51
```

Syntaxe du langage Java

9 Lecture des valeurs entrées par arguments (1)

```
public class Somme {
    public static void main(String[ ] args)
    {
      int a = Integer.parseInt(args[0]);
      int b = Integer.parseInt(args[1]);
      int S = a + b ;
      System.out.println(‘ La somme est ‘ + S);
    }
}
```

Syntaxe du langage Java

9 Lecture des valeurs entrées au clavier (2)

```
import java.util.*;
public class ExempleScanner{
public static void main (String [ ] args) {
Scanner S = new Scanner(System.in);
System.out.println("Donner deux entiers");
int a = S.nextInt();
int b = S.nextInt(); S.close();
if (a>b)
System.out.println(a + " est plus grand que " + b);
else if(b>a)
System.out.println(b + " est plus grand que " + a);
else
System.out.println(" Egalité " );
}}
```

Syntaxe du langage Java

9 Lecture des valeurs entrées au clavier (3)

```
public class Somme {
    public static void main(String[ ] args)  {
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
System.out.println(« Donner aa»);
String aa=br.readLine();//lire la chaine aa
int a =Integer.parseInt(aa.trim());
System.out.println(« Donner bb»);
String bb=br.readLine();//lire la chaine aa
int b =Integer.parseInt(bb.trim());
    int S = a + b ;
    System.out.println(« La somme est »  + S);
   } }
```