FACULTE DES SCIENCES AIN CHOCK
UNIVERSITE HASSAN II DE CASABLANCA

Département: Mathématiques & Informatique

Introduction à Java

Licence : Physique Chimie
Filières : Physique

Pr: Y.Ouassit

Plan Chapitre 1: Introduction à Java

1. **Introduction au langage Java**
   a. Quelques notions historiques
   b. Qu'est-ce que Java ?
   c. Java comme langage de programmation
   d. La plateforme Java
   e. La Java Virtual Machine
   f. Les interfaces de programmation d'application (API)
   g. Déploiement d'un programme
2. **Syntaxe du langage**
   a. Variables ,constantes et types de données
   b. Opérateurs et expressions
   c. Structures de contrôles
   d. Fonctions et méthodes

Introduction

Histoire de Java (1/7)

1991 - The Birth of Java Project ("Green Project"):

Java a été développé à partir de décembre 1990 par une équipe de **Sun Microsystems** dirigée par James Gosling

Au départ, il s'agissait de développer un langage de programmation pour permettre le dialogue entre de futurs ustensiles domestiques (Green Project)

Or, les langages existants tels que C++ ne sont pas à la hauteur : recompilation dès qu'une nouvelle puce arrive, complexité de programmation pour l'écriture de logiciels fiables...

Slogan : **WRITE ONCE RUN ANYWHERE**

Histoire de Java (2/7)

1992 - First Working Demo of Oak:

L'équipe a développé le premier produit de démonstration utilisant Oak pour un petit appareil portable appelé **Star7**.

Histoire de Java (3/7)

1994 - Pivot to the Web:

Avec la montée en puissance du World Wide Web, l'équipe réalise que l'indépendance de plateforme d'Oak pourrait être utile pour le développement web. Oak est adapté pour fonctionner sur les navigateurs web sous forme d'applets, le rendant adapté à du contenu dynamique et interactif sur le web.

1995 - Oak devient Java:

Oak est renommé Java car le nom était déjà déposé. Le nouveau nom est inspiré par le café Java, très apprécié par les développeurs.

Java 1.0 est officiellement lancé lors de la conférence SunWorld. Sun Microsystems rend Java disponible gratuitement, ce qui favorise son adoption rapide.

Netscape Navigator, le navigateur web le plus populaire à l'époque, annonce son support de Java, ce qui lui donne une grande visibilité.

Histoire de Java (4/7)

1997 - Processus de la communauté Java (JCP)

Sun Microsystems lance le Java Community Process (JCP) pour impliquer d'autres entreprises et organisations dans l'évolution de la plateforme Java.

1998 - Java 2 et introduction des éditions (J2SE, J2EE, J2ME)

La sortie de Java 2 (J2SE 1.2) marque une mise à jour importante avec l'introduction de Swing, des Java Foundation Classes (JFC) et des JavaBeans.

Histoire de Java (5/7)

2004 - Java 5 (J2SE 5.0 / Java SE 5.0):

Java 5 introduit des améliorations majeures, comme :

- **Génériques** pour plus de sécurité de type.
- **Annotations** pour simplifier la configuration.
- **Boucle for-each**, **autoboxing/unboxing**, **varargs** et l'**API de concurrence**.

La dénomination est simplifiée, abandonnant le "2", et **J2SE** devient **Java SE** (Edition Standard).

2006 - Java devient open source:

Sun Microsystems annonce que Java sera rendu open source sous la licence GNU General Public License (GPL).

Le projet OpenJDK est lancé, qui deviendra la référence pour l'implémentation de Java.

Histoire de Java (6/7)

2010 - Oracle rachète Sun Microsystems:

Oracle finalise le rachat de **Sun Microsystems**, prenant le contrôle de Java et de son développement.

2011 - Java SE 7 (Première version majeure d'Oracle):

Parmi les nouvelles fonctionnalités, on trouve **try-with-resources** pour la gestion automatique des ressources, l'**opérateur diamant**, et le **framework Fork/Join** pour le traitement parallèle.

2014 - Java 8 et l'introduction des expressions Lambda

Java 8 est lancé, introduisant les **expressions lambda** et l'**API des Streams**, révolutionnant la gestion des collections et la programmation fonctionnelle en Java.

Histoire de Java (7/7)

2017 - Java 9 et le système de modules (Project Jigsaw):

Java 9 introduit le **Java Platform Module System (JPMS)**, également connu sous le nom de **Project Jigsaw**, permettant la modularisation de la JDK et des applications Java.

2018 - Adoption d'un nouveau cycle de publication:

Oracle met en place un **nouveau cycle de publication semestriel** pour Java, avec une **version de support à long terme (LTS)** tous les quelques années.

2023-2024 - Évolution continue de Java:

Les différente versions de Java

- De nombreuses versions de Java depuis 1995
  - Java 1.0 en 1995
  - Java 1.1 en 1996
  - Java 1.2 en 1999 (Java 2, version 1.2)
  - Java 1.3 en 2001 (Java 2, version 1.3)
  - Java 1.4 en 2002 (Java 2, version 1.4)
  - Java 5 en 2004
  - Java 6 en 2006
  - Java 7 en 2011
  - Java 8 en 2014
  - Java 9 en 2018
  - …
  - Java 20 en 2023
- Évolution très rapide et succès du langage

Les différentes versions de Java

Résumé des versions LTS :

- **Java 8 (2014)** : LTS.
- **Java 11 (2018)** : LTS.
- **Java 17 (2021)** : LTS.
- **Java 21 (2023)** : LTS.

Java évolue rapidement, avec une cadence de sortie de nouvelles versions tous les six mois. Cependant, les versions LTS (Long-Term Support) comme Java 8, Java 11, Java 17 et Java 21 sont privilégiées par les entreprises pour leur stabilité et leur support à long terme.

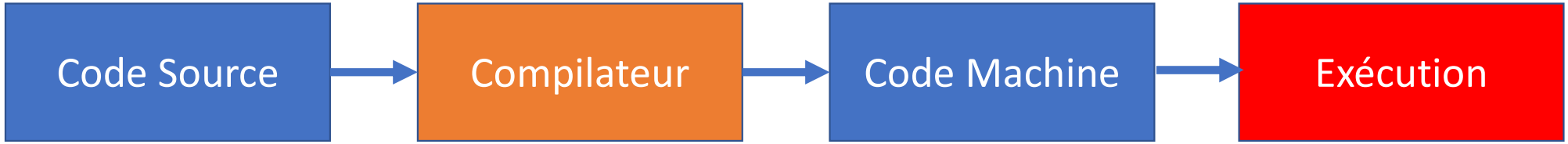
Qu'est-ce que Java ?

- Java est un langage de programmation:
  - Un programme Java est compilé et interprété (pseudo-compilé)

- Java est une plateforme :
  - La plateforme Java, uniquement software, est exécutée sur la plateforme du système d'exploitation
  - La « Java Platform » est constituée de :
    - La « Java Virtual Machine » (JVM)
    - Kit de développement
    - Des interfaces de programmation d'application (Java API)

Compilé vs Pseudo-Compilé vs interprété

Un langage compilé:



- Le code source est directement traduit en code machine natif spécifique à une plateforme.
- Exécution rapide car tout est compilé en amont.
- Nécessite une recompilation pour chaque type de machine.
- Exemple : C, C++

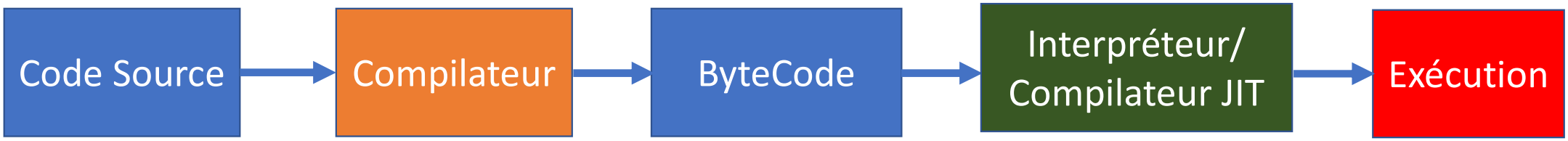
Compilé vs Pseudo-Compilé vs interprété

Un langage interprété:

- Le code source est interprété ligne par ligne à l'exécution.
- Facilité de développement et de debugging, mais exécution plus lente.
- Aucune compilation préalable en code machine.

Compilé vs Pseudo-Compilé vs interprété

Un langage Pseudo-Compilé:



- Le code source est d'abord compilé en bytecode indépendant de la plateforme.
- Le bytecode est ensuite interprété ou compilé en code machine par la VM lors de l'exécution.
- Portabilité élevée car le bytecode peut être exécuté sur toute machine avec une VM.

Java comme langage de programmation

Simple et orienté objet:

- Java est un langage de programmation simple
  - Langage de programmation au même titre que C/C++/Perl/Smalltalk/Fortran mais plus simple
  - Les aspects fondamentaux du langage sont rapidement assimilés
- Java est orienté objet :
  - La technologie OO après un moment de gestation est maintenant complètement intégrée
  - En java, tout est un objet (à la différence du C++ par ex.)
- Simple aussi parce qu'il comporte un grand nombre d'objets prédéfinis pour l'utilisateur
- Java est familier pour les programmeurs C++

Java comme langage de programmation

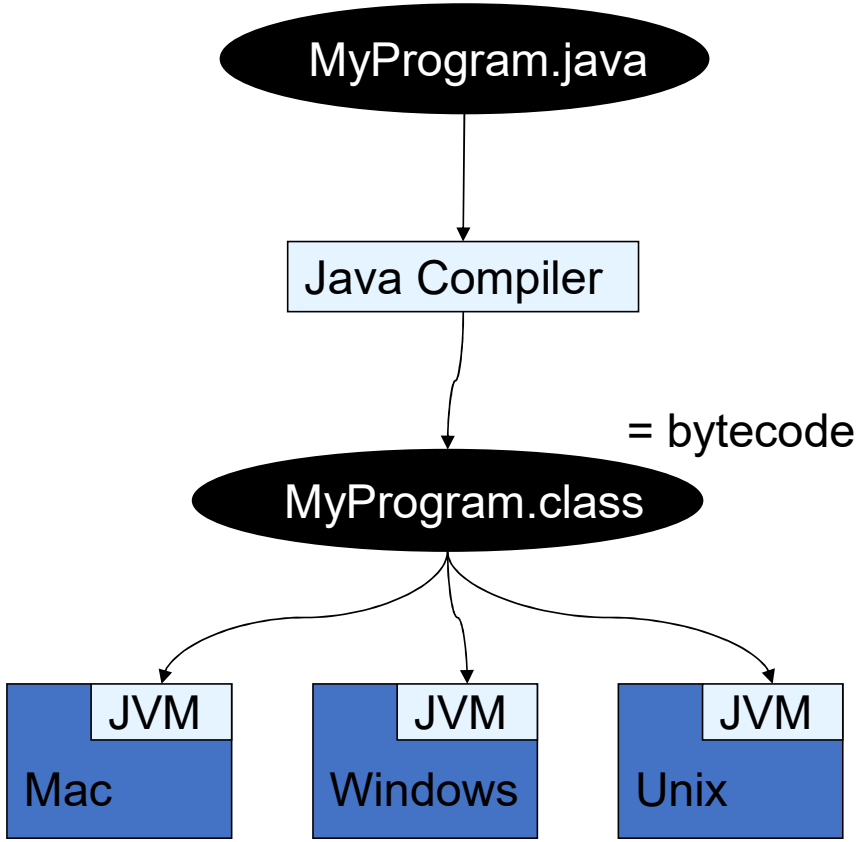
Robuste et sécurisé :

- Conçu pour créer des logiciels hautement fiables
- Oblige le programmeur à garder à l'esprit les erreurs hardware et software
- Vérifications complètes à l'exécution et à la compilation
- Existence d'un « garbage collector » qui permet d'éviter les erreurs de gestion de la mémoire

Java comme langage de programmation

Neutre architecturalement :

- Il existe une grande diversité de systèmes d'exploitation.
- Le compilateur Java génère un bytecode, c'est à dire un format intermédiaire, neutre architecturalement, conçu pour faire transiter efficacement le code vers des hardware différents et/ou plateformes différentes
- Le bytecode ne peut-être interprété que par le processeur de la JVM



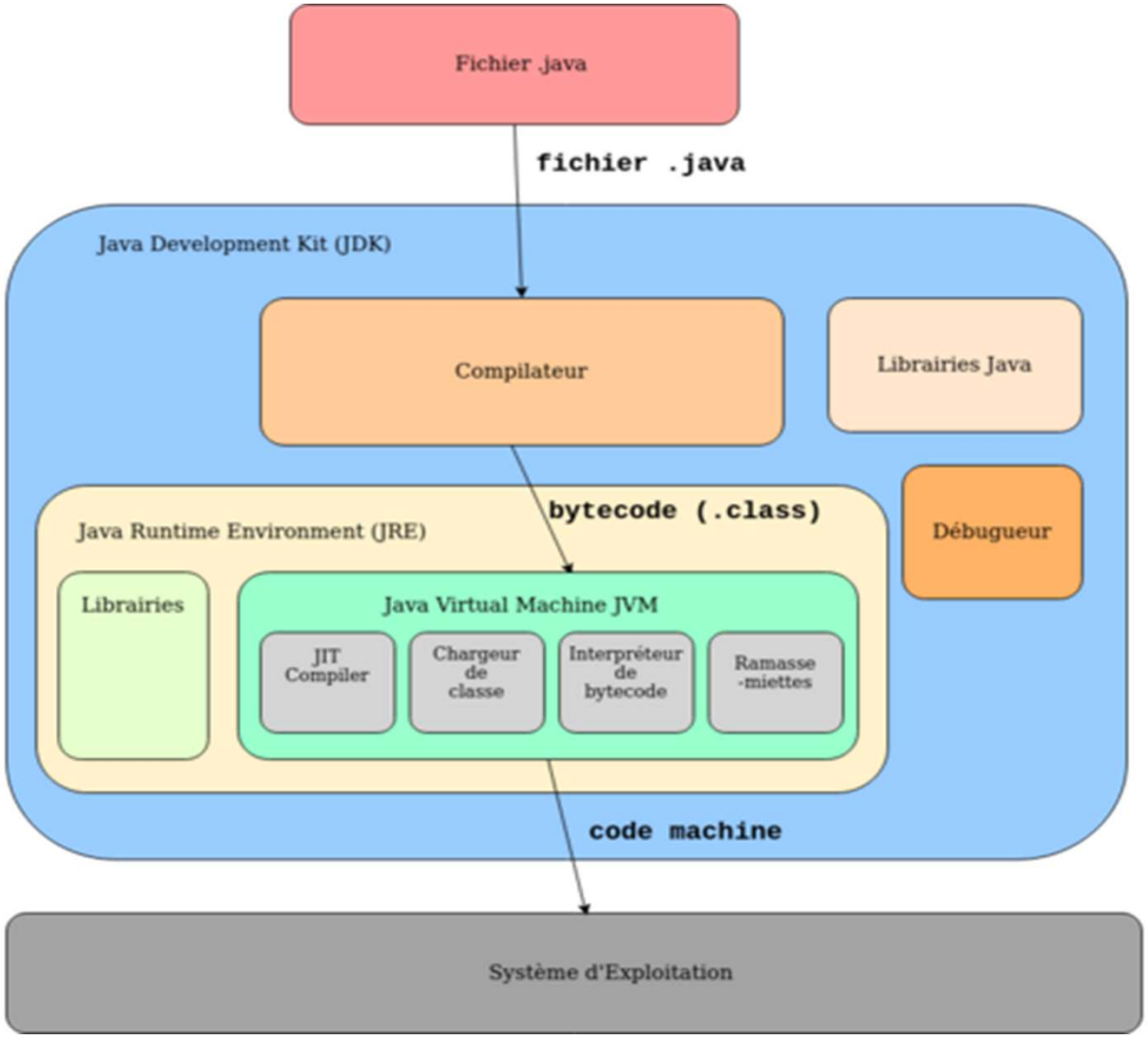
Java comme langage de programmation

Ouvert et distribué :

- Support intégré d'Internet
  - La Class URL
  - Communication réseaux TCP et UDP
  - RMI, CORBA, Servlets
- Connectivité aux bases de données
  - JDBC: Java DataBase Connectivity
  - Offre des facilités de connexions à la plupart des BD du marché
  - Offre un pont vers ODBC
- Support des caractères internationaux
  - Java utilise le jeu de caractères UNICODE
  - JVM équipée de tables de conversion pour la plupart des caractères
  - JVM adapte automatiquement les paramètres régionaux en fonction de ceux de la machine sur laquelle elle tourne

Java comme Plateforme

- Plateforme = environnement hardware ou software sur lequel le programme est exécuté.



Java comme Plateforme

Java Application Programming Interface (API) :

- L'API Java est structuré en libraires (packages). Les packages comprennent des ensembles fonctionnels de composants (classes)..
- Le noyau (core) de l'API Java (incluse dans toute implémentation complète de la plateforme Java) comprend notamment :
  - Essentials (data types, objects, string, array, vector, I/O,date,…)
  - Applet
  - Abstract Windowing Toolkit (AWT)
  - Basic Networking (URL, Socket –TCP or UDP-,IP)
  - Evolved Networking (Remote Method Invocation)
  - Internationalization
  - Security
  - …..

Java comme Plateforme

Java Virtual Machine :

- « An imaginery machine that is implemented by emulating it in software on a real machine. Code for the JVM is stored in .class files, each of which contains code for at most one public class ».
- Définit les spécifications hardware de la plateforme.
- Lit le bytecode compilé (indépendant de la plateforme).
- Implémentée en software ou hardware.
- Implémentée dans des environnements de développement ou dans les navigateurs Web.

Java comme Plateforme

Java Runtime Environment:

Trois tâches principales :

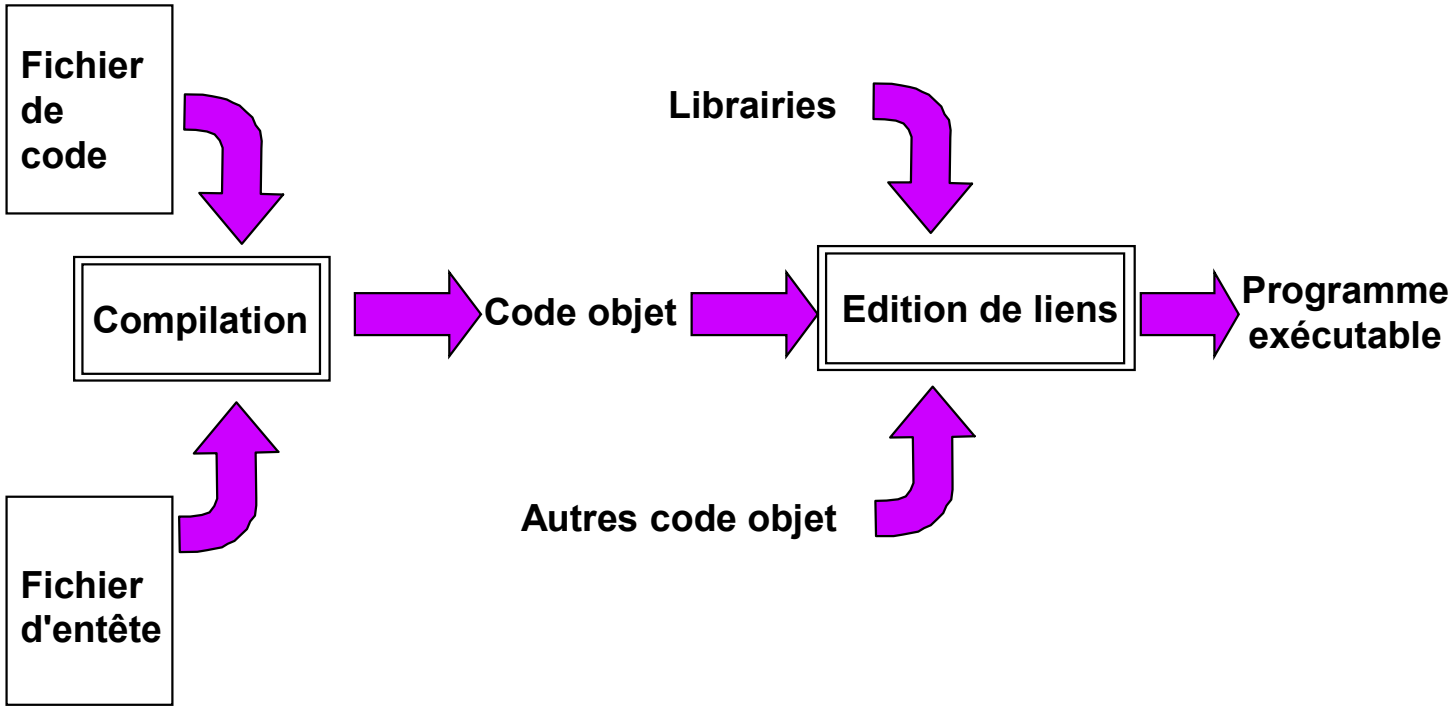
- Charger le code (class loader)
- Vérifier le code (bytecode verifier)
- Exécuter le code (runtime interpreter)

D’autres THREAD s’exécutent :

- Garbage collector

Déploiement d'un programme (1/2)

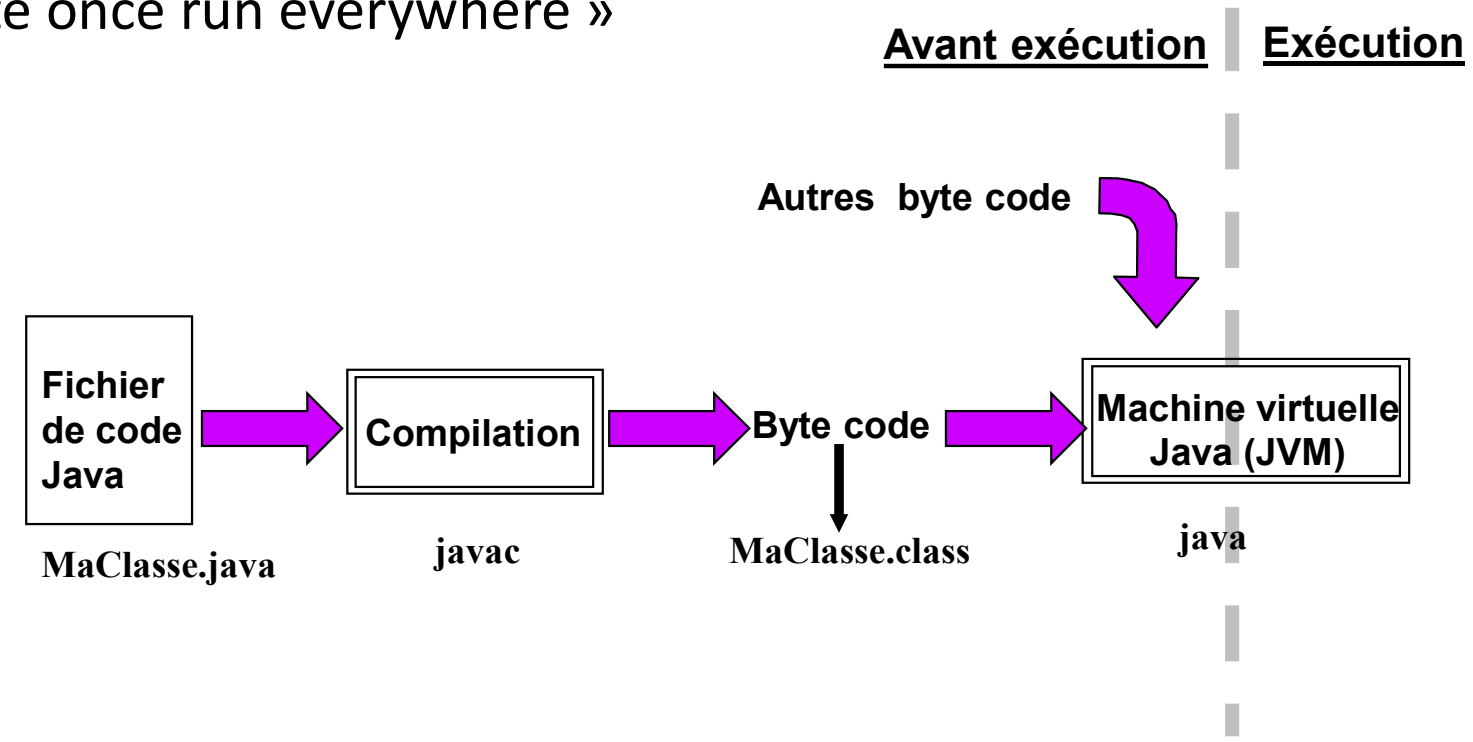
Paradigme classique de la compilation



Déploiement d'un programme (2/2)

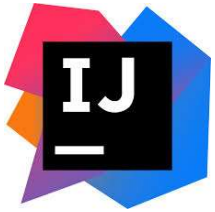
Changement de la vision traditionnelle de la compilation

- Chaque programme est compilé et interprété
- « Write once run everywhere »



Préparation de l'environnement

1. Télécharger le Kit du développement :
   https://www.oracle.com/java/technologies/downloads
2. Installer un éditeur IDE



Les utilitaires Java

- javac
  - Compilateur, traduit fichier source .java en fichier bytecode .class
- java
  - Interpréteur java, lance des programmes
- javadoc
  - Générateur de documentation d'API
- jar
  - Utilitaire d'archivage et de compression

Les utilitaires Java

Javac

- Compile un fichier source .java ou un package entier
- Exemples:
  - `javac MyBankAccount.java`
    compile le fichier mentionné, qui doit se trouver dans le package par défaut
  - `javac com\fsac\*.java -d c:\classes`
    compile tout le package `com.fsac` et génère du code compilé dans `c:\classes`, qui doit exister

Les utilitaires Java

java

Lance un programme principal:

- `java MyProgram`
- `java com.fsac.MyProgram`
  Lance le programme spécifié dans la classe `MyProgram` qui se trouve dans le package `com.fsac`.

Les utilitaires Java

java

A partir de la version Java 11, la commande **java** permet d'exécuter directement le code sans passer par la première étape de compilation.

- `java MyProgram.java`

L'API de Java

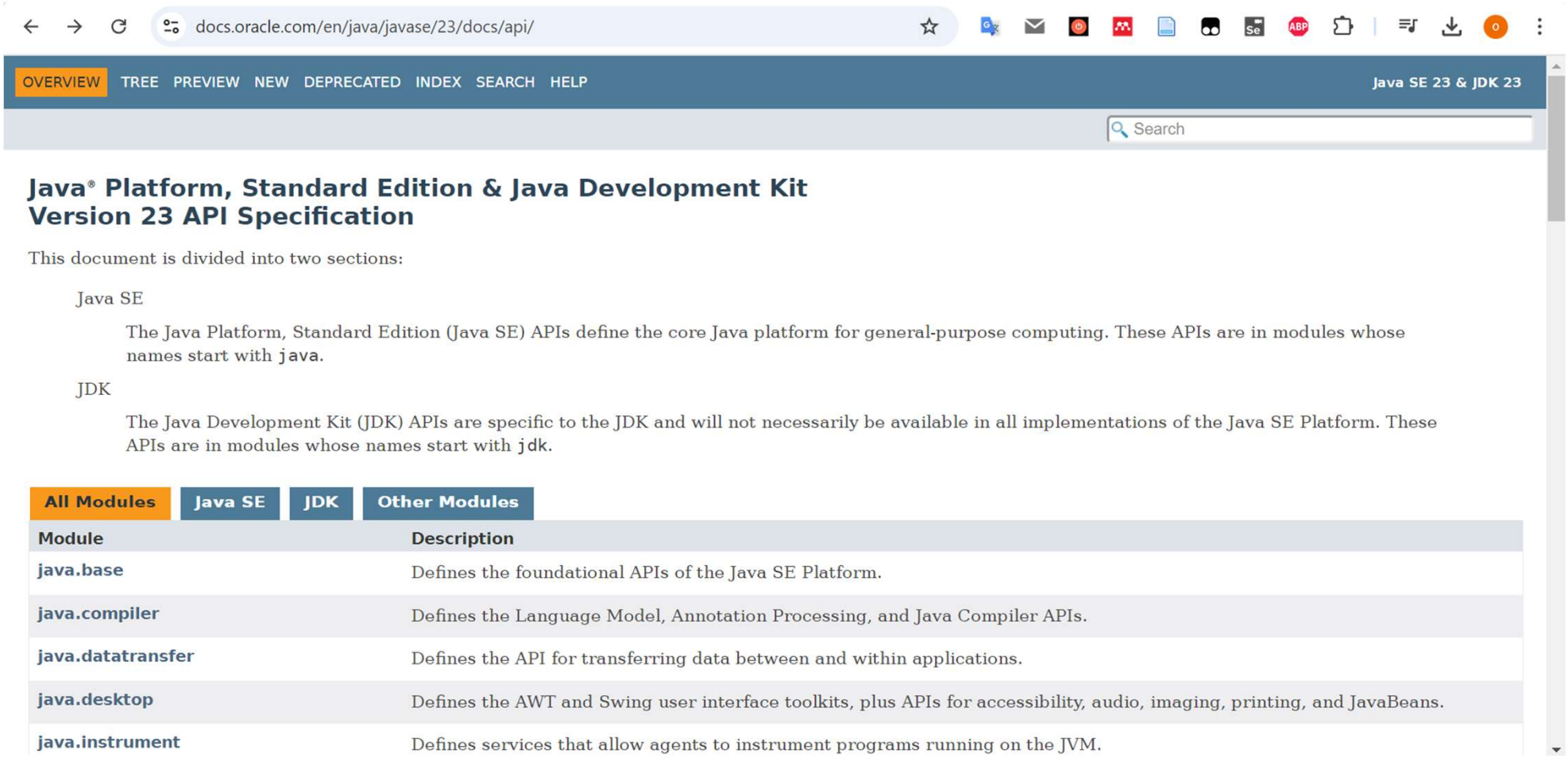
- Java fournit de nombreuses librairies de classes remplissant des fonctionnalités très diverses : c'est l'API Java
  - API (Application and Programming Interface /Interface pour la programmation d'applications) : Ensemble de bibliothèques permettant une programmation plus aisée car les fonctions deviennent indépendantes du matériel.
- Ces classes sont regroupées, par catégories, en paquetages (ou "packages").

L'API de Java

- Les principaux paquetages
  - java.util : structures de données classiques
  - java.io : entrées / sorties
  - java.lang : chaînes de caractères, interaction avec l'OS, threads
  - java.applet : les applets sur le web
  - java.awt : interfaces graphiques, images et dessins
  - javax.swing : package récent proposant des composants « légers » pour la création d'interfaces graphiques
  - java.net : sockets, URL
  - java.rmi : Remote Method Invocation (pas abordé dans ce cours)
  - java.sql : fournit le package JDBC (pas abordé dans ce cours)

L'API de Java

https://docs.oracle.com/en/java/javase/23/docs/api/



Java, un langage novateur ?

- Java n'est pas un langage novateur : il a puisé ses concepts dans d'autres langages existants et sa syntaxe s'inspire de celle du C++.

- Cette philosophie permet à Java
  - De ne pas dérouter ses utilisateurs en faisant "presque comme ... mais pas tout à fait"
  - D'utiliser des idées, concepts et techniques qui ont fait leurs preuves et que les programmeurs savent utiliser

- En fait, Java a su faire une synthèse efficace de bonnes idées issues de sources d'inspiration variées
  - Smalltalk, C++, Ada, etc.

NOTRE PREMIER PROGRAMME JAVA

Point d'entrée d'un programme Java

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

```
public static void main(String arg[ ])
{
      ……
}
```

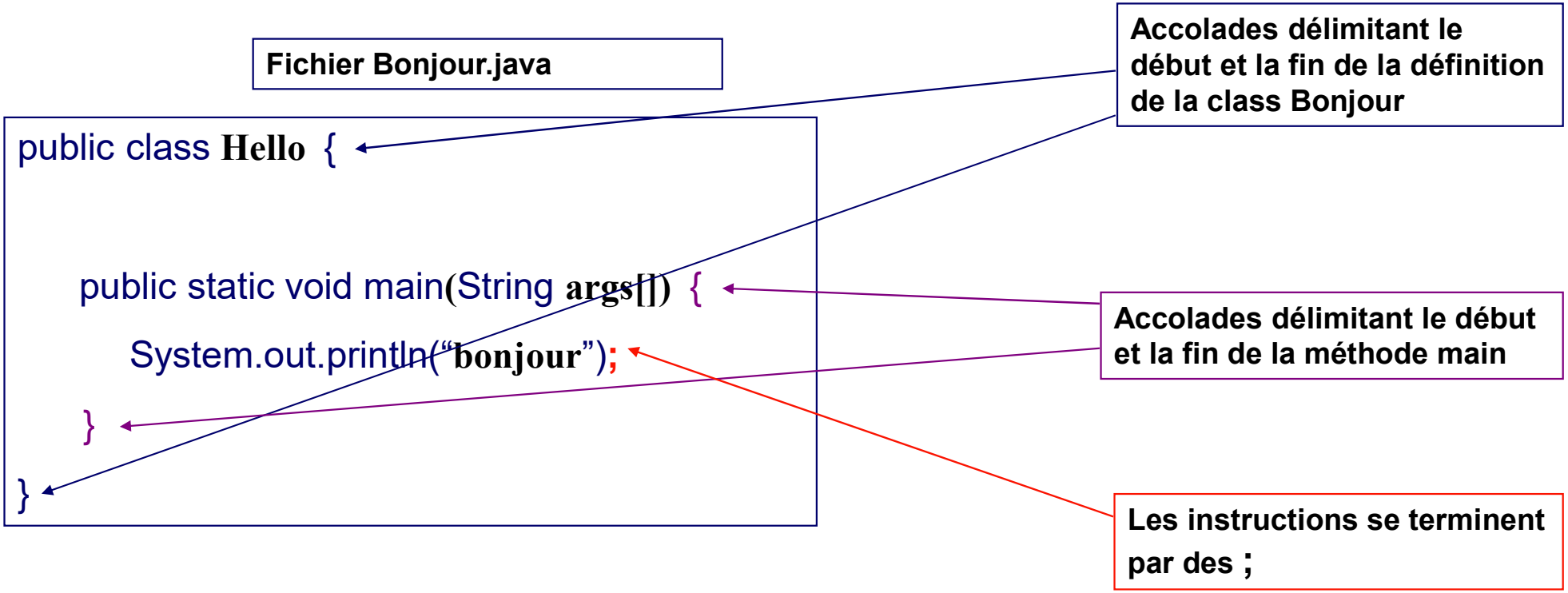
Exemple

Fichier Hello.java

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

La classe est l'unité de base de nos programmes. Le mot clé en Java pour définir une classe est class

Exemple


Fichier Bonjour.java
public class Hello {
public static void main(String args[]) {
System.out.println("bonjour");
}
}
Accolades délimitant le début et la fin de la définition de la class Bonjour
Accolades délimitant le début et la fin de la méthode main
Les instructions se terminent par des ;

Exemple

Fichier Hello.java

```
public class Hello  {
      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.

Compilation et exécution (1)

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 Hello.java
Génère un fichier Bonjour.class

Exécution du programme (toujours depuis la console DOS) sur la JVM :
java Hello
Affichage de « bonjour » dans la console

A partir de la version java 11 :
java Hello.java
Permet de compiler et exécuter

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

Exercice 1:

- Ecrire une première classe avec une fonction main qui affiche "Hello World".
- Compiler et exécuter cette classe.
- Modifier cette classe pour qu'elle accepte un argument de la console.

Syntaxe de base

Syntaxe de base

- Conventions d'écriture
- Commentaires dans le code source
- Identificateurs
- Mots-clé
- Types primitifs et types de références
- Les tableaux (« Array »)
- La classe String
- Arithmétique et opérateurs
- Instructions de contrôle
  - if, then, else
  - for
  - while
  - do… While
  - break et continue
- Packages

Syntaxe de base

Conventions d'écriture

Classes
- class BankAccount
- class RacingBike

Interfaces
- interface Account

Méthodes
- deposit()
- getName()

Packages
- package coursTechnofutur3.bank ;

Variables
- int accountNumber

Constantes
- MAXIMUM_SIZE

Les commentaires

Trois façons d'inclure des commentaires :

- Tout texte entre « // » et la fin de la ligne

```
// Commentaires sur une seule ligne
```

- Tout texte entre « /* » et « */ »

```
/* Commentaires
  sur un nombre important voire très important
  de lignes */
```

- Les textes entre « /** » et « */ » sont utilisés pour créer des commentaires que l'exécutable JAVADOC pourra traiter afin de produire une documentation (cf. documentation de l'API Java)

```
/** Commentaires destinés
à la documentation */
```

Les identificateurs

- Un identificateur (*identifier*) permet de désigner une classe, une méthode, une variable …
- On ne peut utiliser ce que l'on veut :
  - Interdiction d'utiliser les mots-clés
  - Contient des lettres et des chiffres
  - A condition de commencent par :
    - Une lettre
    - Un « $ »
    - Un « _ » (underscore)

Syntaxe de base

Mots clé

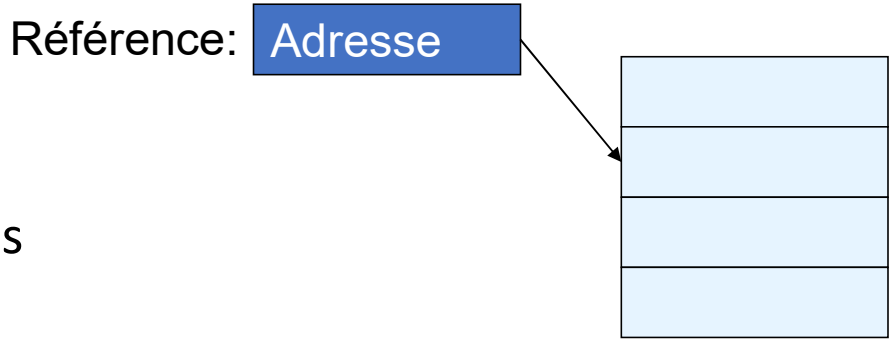
abstract	double	int	strictfp **
boolean	else	interface	super
break	extends	long	switch
byte	final	native	synchronized
case	finally	new	this
catch	float	package	throw
char	for	private	throws
class	goto *	protected	transient
const *	if	public	try
continue	implements	return	void
default	import	short	volatile
do	instanceof	static	while

* Indique un mot clé qui est peu utilisé

** A partir de la plate-forme Java2

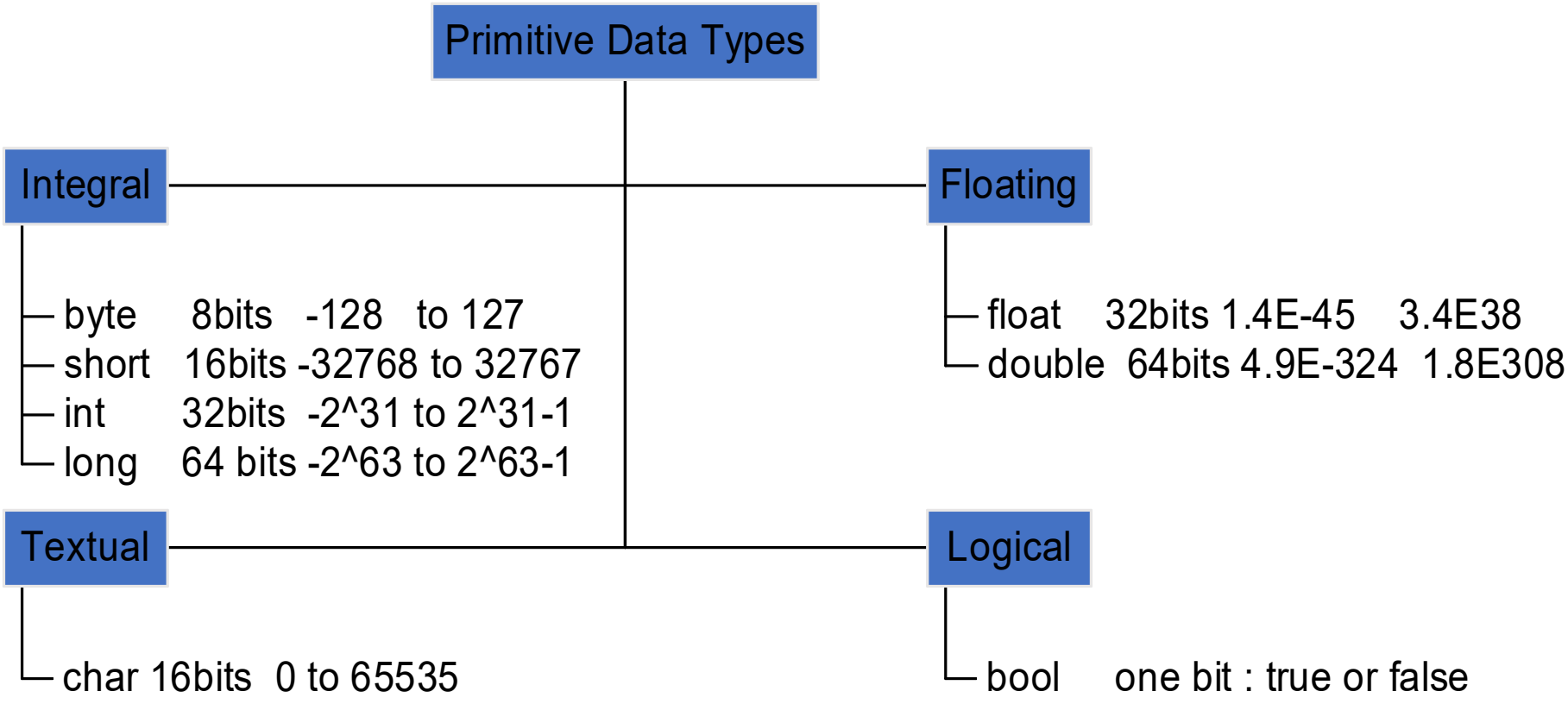
Types primitifs et types de référence

- Java est un langage fortement typé
- Le type de données précise
  - les valeurs que la variable peut contenir
  - les opérations que l'on peut réaliser dessus
- Deux types de données:
  - **Donnée primitive**: contient physiquement la valeur (caractère, nombre, booléen)
  - **Type de référence**: contient l'adresse mémoire où l'information relative à l'objet, l'interface, etc. est réellement stockée



Types primitifs et types de référence

Types de données primitifs (1/3)



Types primitifs et types de référence

Types de données primitifs (2/3)

- Explication:
  - byte : codé sur 8 bits ➔ 2^8 valeurs ➔ (-2^7) to (2^7-1) = -128 à 127
  - int : codé sur 32 bits ➔ 2^{32} valeurs ➔ (-2^{31}) to $(2^{31}-1)$

- Déclaration et initialisation :
  - int `int x=12;`
  - short `short x= 32;` (`short x=33000;` // Hors limite)
  - long `long x= 200L;` // Nombre accolé à un L
  - byte `byte x=012;` // Nombre commençant avec un 0
  - double `double x=23.2323;`
  - float `float x= 23.233F;` // Nombre accolé à un F
  - char `char c='a'; char c='\u0061'; char c=(char)97;`
  - boolean `boolean b=true;`

Types primitifs et types de référence

Types de données primitifs (3/3)

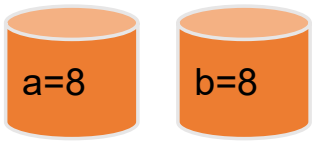
Exemple:

```
int a = 5;
int b = 8;
```

Déclaration et initialisation de 2 entiers: a et b

```
a=b;
```

Affectation de la valeur de b à a

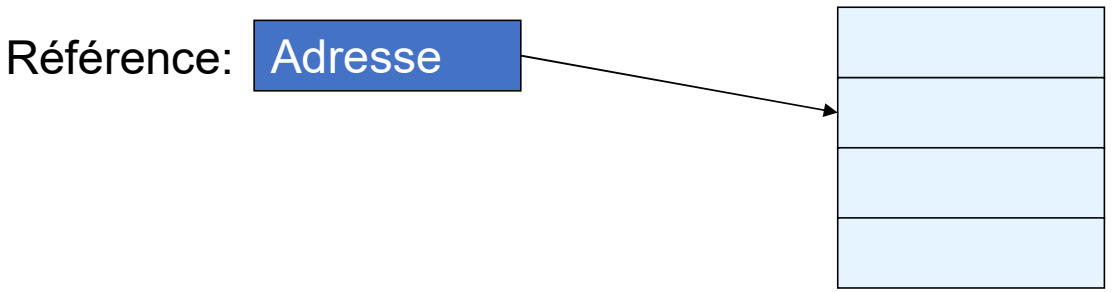


Désormais, il existe deux variables en mémoire qui ont la même valeur

Types primitifs et types de référence

Types de référence

- Tous les types hormis les types primitifs
- « Pointeur implicite » sur un objet



Types primitifs et types de référence

La classe String

- String n'est pas un type primitif, c'est une classe
- Déclaration de deux String:

```
String s1, s2;
```

- Initialisation :

```
s1 = "Hello";
s2 = "le monde";
```

- Déclaration et initialisation :

```
String s3 = "Hello";
```

- Concaténation :

```
String s4 = s1 + " " + s2;
```

Types primitifs et types de référence

Les tableaux "Array" (1/2)

- Un tableau est utilisé pour stocker une collection de variables de même type
- On peut créer des tableaux de types primitifs ou de types de références (cf. argument de la fonction main : `String[] args`)
- Un tableau doit être
  - Déclaré
  - Créé
  - Ses variables initialisées

```
int[] nombres;                  // déclaration
nombres = new int[10];          // création
int[] nombres = new int[10];    // déclaration et création
nombres[0] = 28;
```

Types primitifs et types de référence

Les tableaux "Array" (2/2)

- On peut construire des tableaux à plusieurs dimensions
- Des tableaux à plusieurs dimensions sont en fait des tableaux de tableaux

`int[][] matrice = new int[3][];`	**« matrice » est une référence vers un tableau contenant lui-même 3 tableaux de taille non définie**

`matrice[0] = new int[4];`
`matrice[1] = new int[5];`
`matrice[2] = new int[3];`
	Le premier élément de la matrice est une référence vers un tableau de 4 entiers,…

`matrice[0][0] = 25;`	**Le premier élément du premier tableau de la matrice est un entier de valeur 25**

Exemple:

- **Créer et initialiser une matrice contenant deux tableaux de 2 et 3 points respectivement**
- **Créer 5 objets de type « Point »**
- **Affecter ces 5 objets aux 5 éléments de la matrice**

Arithmétique et opérateurs

Arithmétique élémentaire

- Quelle est la valeur de : 5+3*4+(12/4+1)
- Règles de précédences sur les opérateurs:

Niveau	Symbole	Signification
1	()	Parenthèse
2	*	Produit
	/	Division
	%	Modulo
3	+	Addition ou concaténation
	-	Soustraction

Arithmétique et opérateurs

Opérateurs de comparaison

- Pour comparer deux valeurs:

Opérateur	Exemple	Renvoie TRUE si
>	v1 > v2	v1 plus grand que v2
>=	v1 >= v2	Plus grand ou égal
<	v1 < v2	Plus petit que
<=	v1 <= v2	Plus petit ou égal à
==	v1 == v2	égal
!=	v1 != v2	différent

- Opérateurs logiques:

Opérateur	Usage	Renvoie TRUE si
&&	expr1 && expr2	expr1 et expr2 sont vraies
&	expr1 & expr2	Idem mais évalue toujours les 2 expressions
||	expr1 || expr2	Expr1 ou expr2, ou les deux sont vraies
|	expr1 | expr2	idem mais évalue toujours les 2 expressions
!	! expr1	expr1 est fausse
!=	expr1 != expr2	si expr1 est différent de expr2

Arithmétique et opérateurs

Opérateurs d'assignation(d'affectation)

- L'opérateur de base est '='
- Il existe des opérateurs d'assignation qui réalisent à la fois
  - une opération arithmétique, logique, ou bit à bit
  - et l'assignation proprement dite

Opérateur	Exemple	Équivalent à
+=	expr1 += expr2	expr1 = expr1 + expr2
-=	expr1 -= expr2	expr1 = expr1 – expr2
*=	expr1 *= expr2	expr1 = expr1 * expr2
/=	expr1 /= expr2	expr1 = expr1 / expr2
%=	expr1 %= expr2	expr1 = expr1 % expr2

Instructions et structures de contrôle

Déclarations, instructions, blocs

- Une instruction
  - Réalise un traitement particulier:
  - Renvoie éventuellement le résultat du calcul
  - Est comparable à une phrase du langage naturel
  - Constitue l'unité d'exécution
  - Est toujours suivie de « ; »
  - Instruction d'affectation (d'assignation), instruction de déclaration …
- Un bloc
  - Est une suite d'instructions entre accolades « { » et « } »
  - Délimite la portée des variables qui y sont déclarées
- Une déclaration
  - Constitue la signature d'un élément (classe, variable ou méthode)
  - Annonce la définition de cet élément
  - Est (normalement) toujours suivie d'un bloc d'instructions

Instructions et structures de contrôle

Structures de contrôle

- Les structures de contrôles permettent d'arrêter l'exécution linéaire des instructions (de bas en haut et de gauche à droite)
- Elles permettent d'exécuter conditionnellement une instruction, ou de réaliser une boucle

Type d'instruction	Mots clés utilisés
Décision	if() else – switch() case
Boucle	for(; ;) – while () – do while()
Traitement d'exceptions	try catch finally – throw
Branchement	label : -- break – continue -- return

Instructions et structures de contrôle

Structures de contrôle - IF – ELSE

L'instruction if en Java est une structure de contrôle de flux qui permet d'exécuter un bloc de code en fonction d'une condition. Elle est couramment utilisée pour effectuer des actions uniquement si une certaine condition est vraie.

```
if (expression)
{
   //instructions
}
else
{
   //instructions dans les autres cas
}
```

```
if (expression)
{
    //instructions
}
```

Instructions et structures de contrôle

Structures de contrôle - IF – ELSE

Les instructions if multiples en Java permettent de vérifier plusieurs conditions les unes après les autres, afin d'exécuter différents blocs de code en fonction de chaque condition. Cela se fait souvent en utilisant des combinaisons de if, else if, et else.

```
if (expression 1)
{
   //instructions 1
}
else if(expression 2){
{
   //instructions 2
}
else if(expression 3){
   //instructions 3
}
else {
   //instructions 4
}
```

Instructions et structures de contrôle

Structures de contrôle - SWITCH - CASE

En Java, l'instruction switch permet de contrôler le flux d'exécution en fonction de la valeur d'une expression, souvent une variable. Elle est souvent utilisée comme alternative aux multiples instructions if-else if, offrant une syntaxe plus concise et lisible pour les cas où une variable peut prendre un ensemble limité de valeurs spécifiques.

```
switch (number)
{
      case 1 : instructions;  break;
      case 2 : instructions; break;
      default : instructions;
}
```

Instructions et structures de contrôle

Structures de contrôle - FOR

La boucle **for** en Java est une structure de contrôle qui permet de répéter un bloc de code un certain nombre de fois, en fonction d'une condition. C'est une des boucles les plus couramment utilisées, particulièrement adaptée lorsque le nombre d'itérations est connu à l'avance.

Instructions et structures de contrôle

Structures de contrôle - FOR

```
for (initialisation; condition; mise à jour de valeurs){
  // instructions
}
```

- Initialisation: à exécuter lorsque le programme rentre pour la première fois dans la boucle
- Condition : à remplir pour recommencer le bloc d'instructions
- Mise à jour: instruction exécutée chaque fois que la boucle est terminée

Exemples:

```
for (int i=0 ; i<10 ; i++) {
    System.out.println("The value of i is : " + i);
}
```

Instructions et structures de contrôle

La boucle - WHILE

Définition :

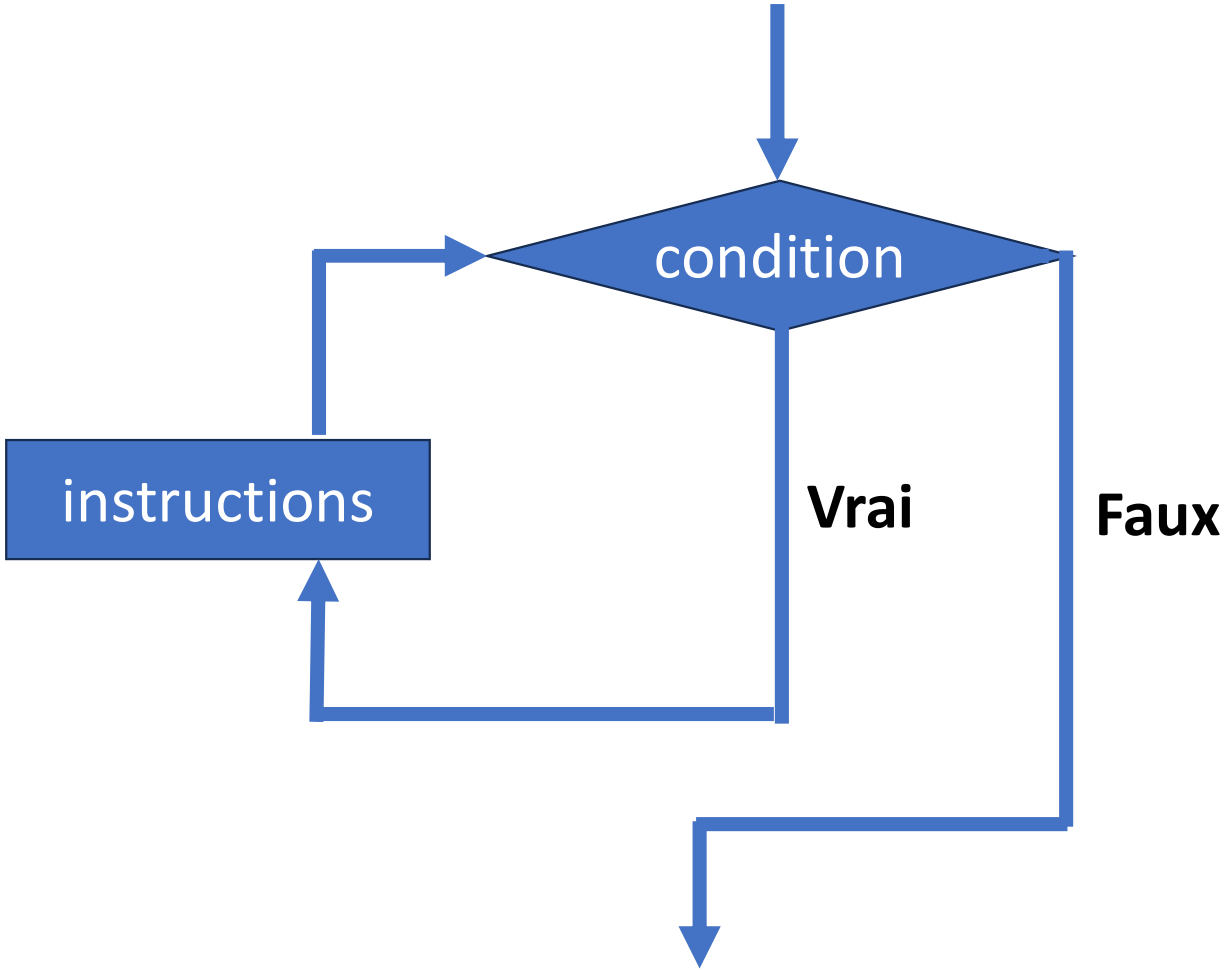
La boucle WHILE est une structure de contrôle en programmation qui permet d'exécuter un bloc de code répétitif tant qu'une condition donnée est vraie. Cela signifie que le bloc de code à l'intérieur de la boucle WHILE est exécuté de manière répétée tant que la condition spécifiée reste vraie. Une fois que la condition devient fausse, l'exécution de la boucle s'arrête et le programme passe à l'instruction suivante après la boucle.

Instructions et structures de contrôle

La boucle - WHILE

Syntaxe :

```
while (condition) {
    //instructions
}
```



Instructions et structures de contrôle

La boucle - DO-WHILE

Définition :

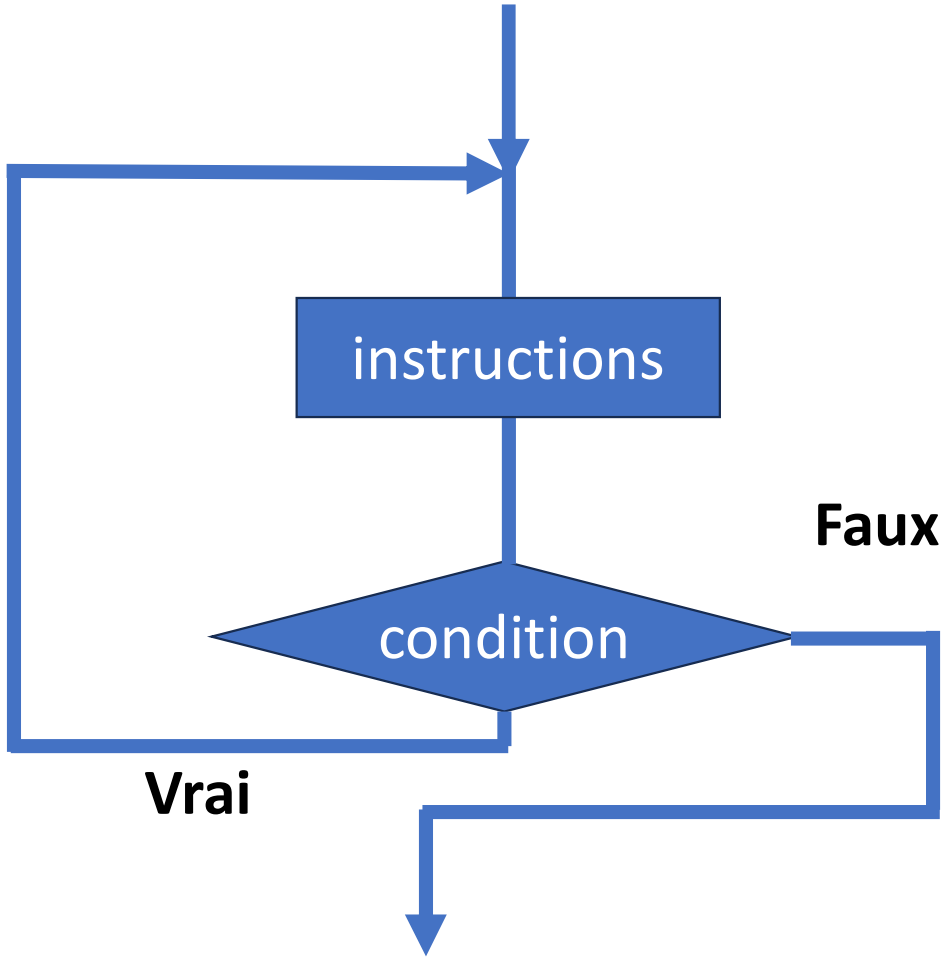
La boucle DO-WHILE est une structure de contrôle en programmation qui permet d'exécuter un bloc de code répétitif jusqu'à une condition donnée est vraie. Le nombre de répétition minimal de cette boucle est 1.

Instructions et structures de contrôle

La boucle - DO-WHILE

Syntaxe :

```
do {
 // instructions
} while (condition)
```



Instructions et structures de contrôle

Structures de contrôle - BREAK / CONTINUE

- BREAK: achève immédiatement la boucle
- CONTINUE: ignore le reste des instructions et recommence au début de la boucle

```
for (int i=0; i<10 ;i++){
    if (i==5) continue; // Si i=5, on recommence au début
    if (i==7) break;    /* Si i=7, on sort de la boucle et
                           les instructions suivantes sont
                           exécutées */
    System.out.println("The value of i is : " + i);
}
```

Instructions et structures de contrôle

Structures de contrôle - BREAK / CONTINUE

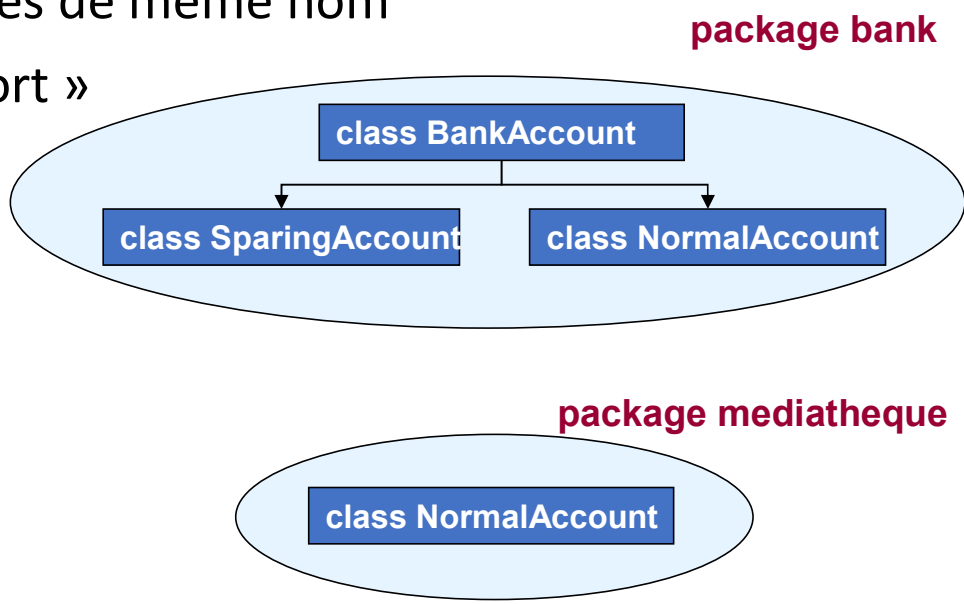
BREAK [LABEL]

CONTINUE [LABEL]

```
outer:
for (int i=0 ; i<10 ; i++)  {
  for(int j=20;j>4;j--){
    if (i==5) continue;       // if i=5, you jump to the beginning of the loop
    if (i==7)  break outer; // if i=7, you jump outside the loop and continue
    System.out.println("The value of i,j is :" + i +  "," + j);
}
```

Les packages et les importations

- Les packages offrent une organisation structurée des classes
- La répartition des packages correspond à l'organisation physique
- Les packages conditionnent les importations de classes
- La variable CLASSPATH indique le chemin d'accès aux packages
- Les packages permettent la coexistence de classes de même nom
- Les mots-clé associés sont « package » et « import »
- Exemple:
  - package technofutur3.bank
    - class NormalAccount
    - class SparingAccount
  - package technofutur3.mediatheque
    - import technofutur3.bank.NormalAccount



Instructions d'écriture

Pour afficher dans la sortie standard :

System.out.println() : Affiche un message et passe à la ligne suivante.

System.out.print() : Affiche un message sans passer à la ligne suivante.

Affichage formaté :

System.out.printf() : Affiche un message formaté.

Exemple : System.out.printf("Le factoriel de %d est %d", n, f);

Instructions de lecture

Pour lire de l'entré standard :

En Java, la classe Scanner permet de lire les données entrées par l'utilisateur à partir de la console.

Étapes pour utiliser Scanner :

1. Importer la classe Scanner : import java.util.Scanner;
2. Créer un objet Scanner : Scanner sc = new Scanner(System.in);
3. Utiliser les méthodes de Scanner pour lire des données.

Voici quelques méthodes utiles :

- nextInt() : Lit un entier.
- nextDouble() : Lit un nombre à virgule flottante.
- nextLine() : Lit une ligne complète de texte.
- next() : Lit un seul mot.

Transtypage (casting):

En Java, le transtypage (ou casting) est l'opération qui permet de convertir une variable d'un type à un autre, notamment lorsqu'on travaille avec des types de données primitifs ou des objets. Il en existe deux types principaux : le **transtypage implicite** et le **transtypage explicite**.

Transtypage (casting):

Transtypage Implicite (Upcasting):

Le transtypage implicite se fait automatiquement lorsque la conversion se fait vers un type plus grand ou plus large, c'est-à-dire lorsque la conversion n'entraîne pas de perte de données. C'est le cas, par exemple, du passage d'un int à un double.

```
int x = 42;
double y = x;  // Implicite : int est converti en double
```

Transtypage (casting):

Transtypage Explicite (Downcasting)

Le transtypage explicite est nécessaire lorsque la conversion se fait vers un type plus petit ou plus précis, ce qui pourrait entraîner une perte de données. Il s'applique aux types primitifs ainsi qu'aux types objets.

```
double x = 42.67;
int y = (int) x;  // double est converti en int, la partie décimale est perdue
```

Transtypage (casting):

Conversion des chaînes vers des types primitifs et vice versa :

```
String str = "42";
int a = Integer.parseInt(ch);  // Chaîne en entier

String str = "42.5";
int a = Double.parseDouble(ch);  // Chaîne en double

String str = "true";
int a = Boolean.parseBoulean(ch);  // Chaîne en booléen

int a = 20;
String str = String.valueOf(a);       // Entier en String
String str = Integer.toString(a);     // Entier en String
```

Exercices