UNIVERSITÉ de la NOUVELLE-CALÉDONIE

Programmation Orientée Objets 1 - Java

Frédéric Flouvat

Université de la Nouvelle-Calédonie

frederic.flouvat@univ-nc.nc

Présentation de l'UE

- Volume horaire :
  - 12h de cours / 16h de TD / 24h de TP

- Evaluation :
  - 4 contrôles continus (4 CC, 25% par CC)
    - CC1: évaluation "papier" (2h, documents autorisés, 75% de la note) + TP1 à rendre (25% de la note)
    - CC2: projet 1 noté (TP à finir à la maison)
    - CC3: évaluation en salle machine (TP noté de 2h)
    - CC4: projet 2 noté (TP à finir à la maison)
    - Pour les TP / projets
      un rapport décrivant (avec des captures d'écran) les fonctionnalités développées
      code source commenté
      documentation technique (avec diagramme de classes)
  - 2ème chance: plus mauvaise note de CC enlevée

- Objectif : apprentissage des concepts objets et du langage de programmation Java

Quelques références bibliographiques en ligne

- https://docs.oracle.com/javase/tutorial/java/index.html : la documentation officielle
- https://beginnersbook.com/java-tutorial-for-beginners-with-examples/
- http://www.jguru.com, …

- https://java.developpez.com : tutoriels, FAQ, ressources ...
- https://openclassrooms.com/courses/apprenez-a-programmer-en-java

- cours de Philippe Genoud, maître de Conférences, Université Joseph Fourier, Grenoble, http://lig-membres.imag.fr/genoud/ENSJAVA/M2CCI/cours.html
- cours de Sébastien Combéfis, Ecole Centrale des Arts et Métiers, Bruxelles https://www.ukonline.be/cours/java/apprendre-java

- Evan Jones, Adam Marcus, and Eugene Wu. 6.092 Introduction to Programming in Java. January IAP 2010. Massachusetts Institute of Technology: MIT OpenCourseWare, https://ocw.mit.edu License: Creative Commons BY-NC-SA.

Plan

- Introduction au Java
  - **Généralités**
  - Syntaxe de base
- Concepts et modélisation orientée objets
  - Objet, classe et modélisation UML
  - Les principes fondamentaux: encapsulation, abstraction, héritage et polymorphisme
- Programmation Orientée Objets en Java
  - Classes, objets et bonnes pratiques
  - Héritage, interfaces, agrégation, composition et association
  - Packages
  - Généricité
  - Exceptions
  - Flux d'Entrée/Sortie et fichiers
  - Empaqueter et déployer son programme

Java c'est quoi ?

- Une technologie lancée par SUN Microsystems en 1995, puis rachetée par Oracle en 2009
  - un langage de programmation
  - un environnement logiciel multiplateforme (Java Virtual Machine ou JVM)
  - mais aussi un système d'exploitation (JavaOS), des environnements de développement (p.ex. Eclipse), une plateforme pour les périphériques mobiles/embarqués (Java ME), des technologies Web et orientées services (Java EE), etc

- Utilisée dans de très nombreux domaines d'application : des sites Web aux téléphones portables (p.ex. Android) en passant par les cartes à puces

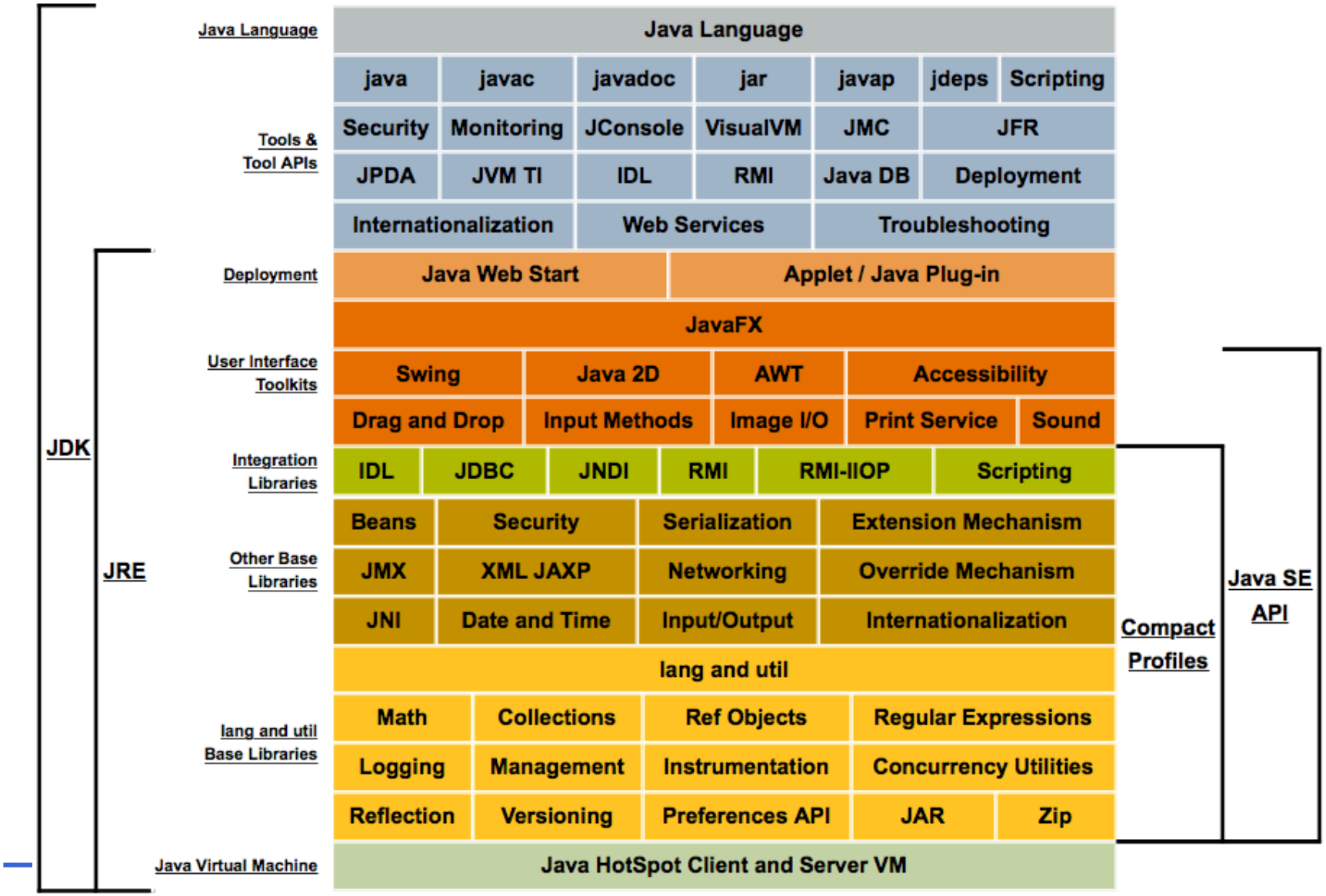
- Des millions de développeurs et des centaines de millions d'applications

L'environnement de développement

- Les environnements Java :
  - **Java SE** (Standard Edition)
  - Java ME (Mobile Edition)
  - Java EE (Enterprise Edition)

- L'environnement JSE :
  - **JDK (Java Development Kit)**
    - Comprend de nombreux outils : le compilateur (javac), l'interpréteur d'application (java), le débogueur (jdb), le générateur de documentation (javadoc), etc.
  - Des Environnements de Développement Intégrés (EDI en français ou IDE en anglais) :
    - Libres: **Eclipse**, Netbeans, *Android Studio (Google)*, etc
    - Commerciaux : **IntelliJ IDEA** (JetBrains), JBuilder (Borland/Inprise), etc

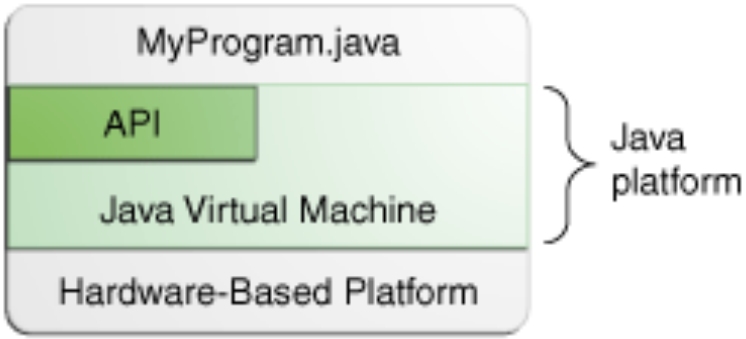
L'environnement de développement



La plateforme Java

- Plateforme = environnement matériel et/ou logiciel dans lequel un programme s'exécute

- Plateforme Java entièrement logicielle s'exécutant au dessus des plateformes matérielles
  - API (Application Programming Interface) Java : bibliothèques Java standards sur lesquelles le programmeur peut s'appuyer pour écrire son code
  - Machine virtuelle Java (JVM) : programme chargé d'exécuté les programmes Java



Remarque sur les variables d'environnements

- Variables d'environnement = variables dynamiques du système d'exploitation
  - Accès sous Windows : via l'objet système du panneau de configuration
  - Accès sous Unix : echo $VARIABLE / export VARIABLE=value
    - p.ex. export PATH =. : /home/me/classes:/products/java/lib/classes.zip

- Nécessaire au bon fonctionnement d'un programme Java à la compilation et à l'exécution
  - Le chemin d'accès aux différentes classes et outils Java

- **JAVA_HOME** : répertoire de base du JDK
- **PATH** : répertoire contenant le compilateur et l'interpréteur

- **CLASSPATH** : répertoire contenant les classes utilisée par le programme
  - Modifiable aussi via l'option -classpath du compilateur (javac) et de la JVM (java)

Exercice: Mon premier programme Java (non objet)

Le code doit être enregistré dans un fichier de même nom (casse comprise)

HelloWorld.java

Tour code Java doit être défini à l'intérieur d'une **classe**

```
public class HelloWorld {

    public static void main( String[] args ) {

        int i = 0 ;

        while ( i < 3 ) {
            System.out.println("Hello World") ;
            i++ ;
        }

    }

}
```

La description de la classe est effectuée à l'intérieur du bloc **{ }**

Le point d'entrée à l'exécution est la méthode **main**( )

Compilation :
javac HelloWorld.java

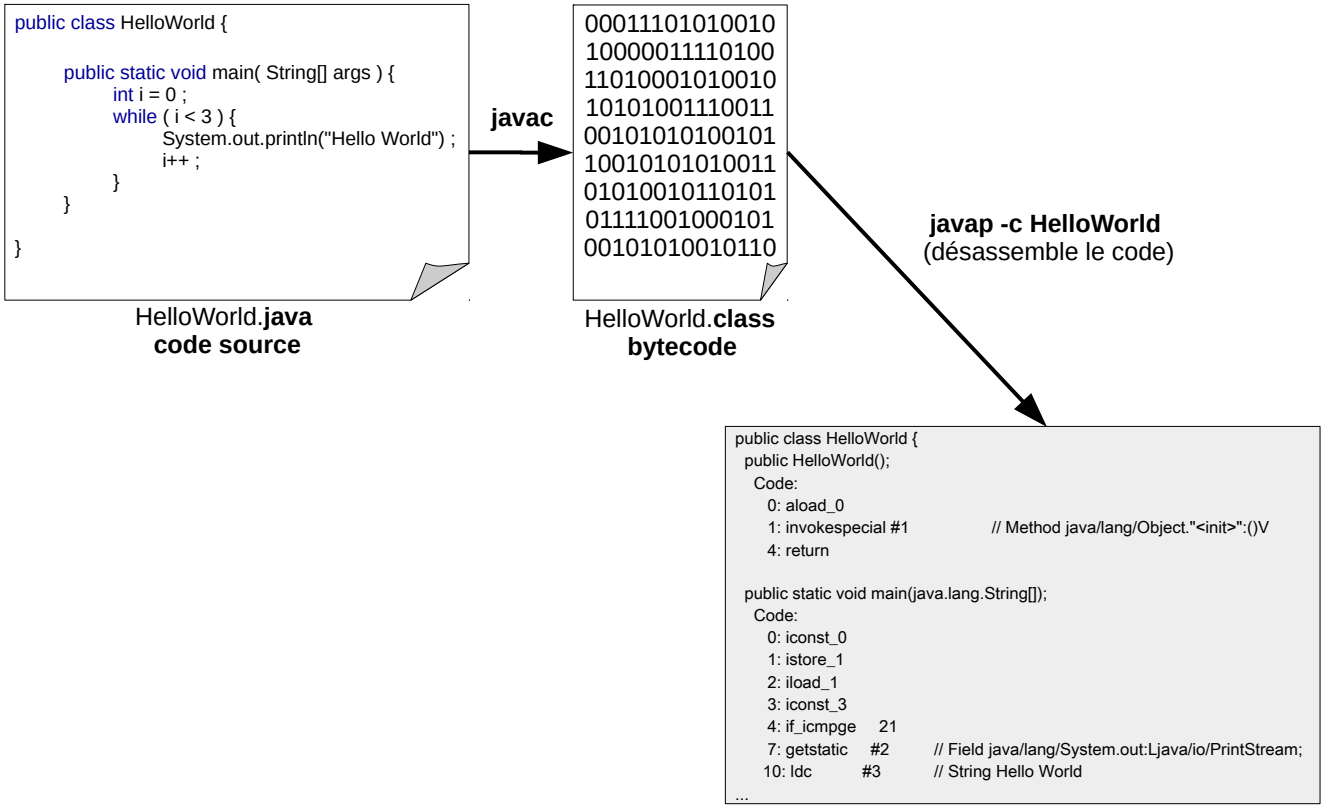
Exécution :
java HelloWorld

HelloWorld.**java**

HelloWorld.**class**

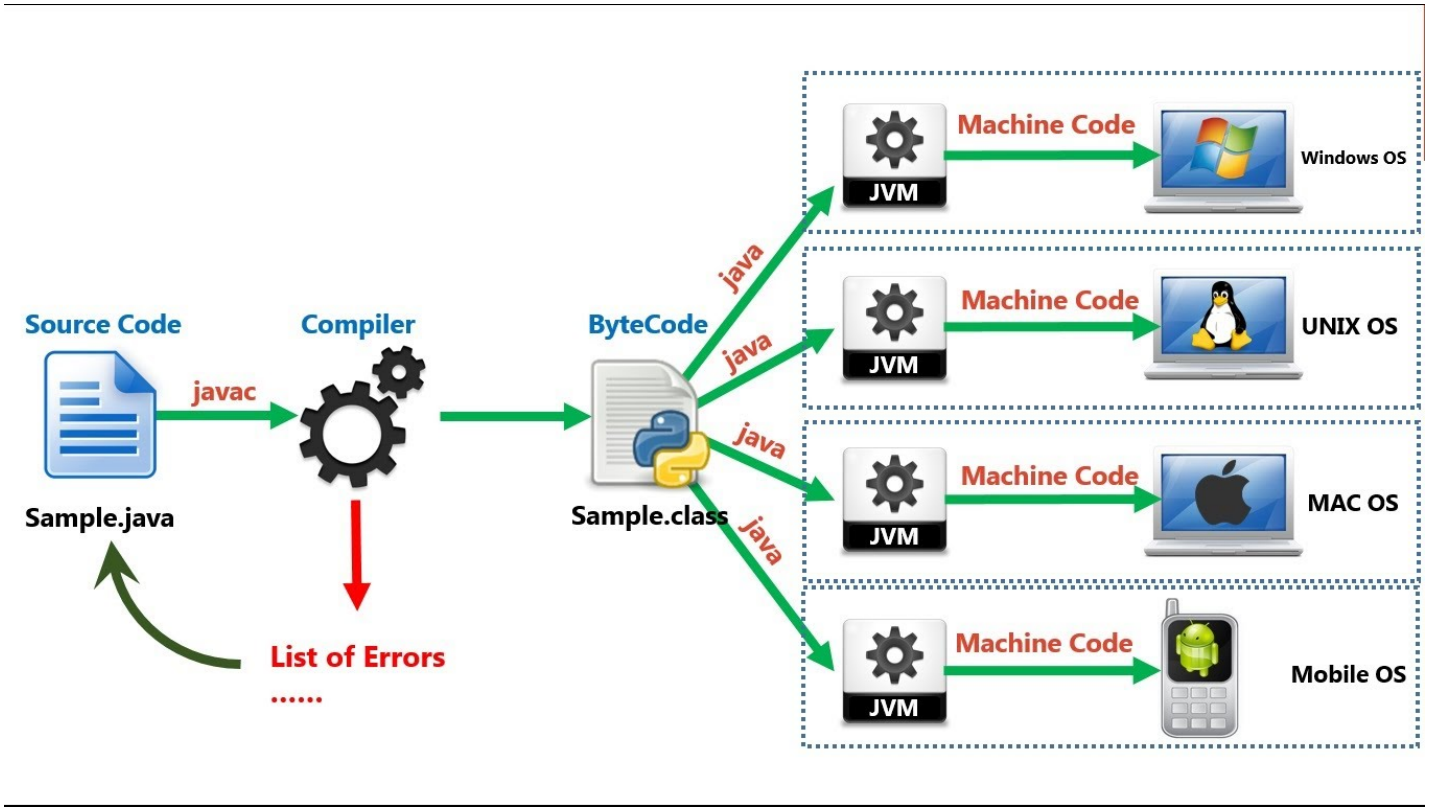
Construction d'un programme Java

- Compilation d'un programme Java génération de bytecode (.class)
  - Vérification syntaxique
  - Vérification sémantique (typage)
  - Production de code dans un langage plus proche de la machine



Exécution d'un programme Java

- Bytecode exécuté par une machine virtuelle Java (JVM = l'interpréteur) spécifique à chaque plateforme (machine/système d'exploitation)



*Java Tutorial by Rajinikanth: Execution Of Java Program

- "compile once, run everywhere" (slogan Sun)

Avantages et inconvénients de l'architecture Java

- Avantages :
  - **Portabilité** du code Java
  - Compacité du code Java
  - Chargement sélectif et dynamique des classes nécessaires
  - Une API très importante
  - Open source

- Inconvénient :
  - Java interprété -> impact sur les performances mais machine virtuelle de plus en plus optimisée

- Pour plus d'informations sur les performances de Java
  - https://benchmarksgame-team.pages.debian.net/benchmarksgame
  - https://jrebel.com/rebellabs/getting-cc-performance-in-java-by-john-davies/

Plan

- Introduction au Java
  - Généralités
  - **Syntaxe de base**
- Concepts et modélisation orientée objets
  - Objet, classe et modélisation UML
  - Les principes fondamentaux: encapsulation, abstraction, héritage et polymorphisme
- Programmation Orientée Objets en Java
  - Classes, objets et bonnes pratiques
  - Héritage, interfaces, agrégation, composition et association
  - Packages
  - Généricité
  - Exceptions
  - Flux d'Entrée/Sortie et fichiers
  - Empaqueter et déployer son programme

Le langage et ses spécificités

- Java offre les mêmes possibilités que des langages tel que le C
  - des blocs d'instructions
  - des variables (locales/globales, portée liée au bloc, etc) et un typage fort
  - des tableaux
  - des structures de contrôle (**while**, **for**, **do**, **if**, **switch**)
  - des fonctions/procédures (passage par recopie des paramètres dont le type est primitif)
  - …

- Mais avec
  - une gestion de la mémoire transparente ("pas de pointeurs" et pas de malloc/free)
  - une organisation du code très différente: Programmation Orientée Objets (POO ou OOP en anglais)
    - plus de modularité, plus de réutilisation, plus de flexibilité, etc.
  - une API très fournie (chaînes de caractères, tableaux, listes, etc.)
  - …

Les similarités syntaxiques avec le C

- Instructions terminées par **;**
- Déclaration d'une variable **<type> <nomVariable>;**
- Délimitation des blocs avec **{** … **}**
  - portée d'une variable limitée au bloc où elle a été déclarée
- Mise en commentaires avec *//* ou */** … **/*
- Utilisation de structures de contrôles
  - **while(** *cond* **) {** … **}**
  - **do{** … **} while(** *cond* **);**
  - **for(** *init* **;** *cond* **;** *modif* **){** … **}**
    - possibilité de déclarer l'itérateur dans la boucle
  - **if(** *cond* **){** … **} else if(** *cond* **){** … **} else {** … **}**
  - **switch(** *expr* **){ case** *val_1***:** … **break; case** *val_2***:** … **break; default:** …**}**
    - *expr* de type char, byte, short, ou int

```
for( int i = 0; i<10; i++) {
  // …
}
```

- Définition des opérateurs
  - arithmétiques: +, -, *, /, %, ++, --
  - d'affectation: =, +=, -=, *=, /=, .=, %=
  - de comparaison: ==, <, >,<=, >=, !=
  - logiques: &&, ||, !

Pré- incrémentation / décrémentation	**Post- incrémentation / décrémentation**

j= ++ i ; // i = i+1
//suivi de j = i
	j= i++ ; // j = i
	// suivi de i = i+1

Les types primitifs

Data Types					
Type Name	Minimum Value	Maximum Value	Default	Size	Literal
byte	-128	127	0	8-bit +/-	———
short	-32768	32767	0	16-bit +/-	———
int	-2147483648	2147483647	0	32-bit +/-	3, 077, 0xBAAC
long	-9223372036854775808	9223372036854775807	0	64-bit +/-	3L
float	-1.40239846e-45	3.40282347e+38	0.0	32-bit IEEE float	3.0F, 3.0E2F
double	-4.94065645841246533e-324	1.79769313486231570e+308	0.0	64-bit IEEE float	3.0, 3.0E2, 3.0e2D
boolean	false	true	false	N/A	true, false
char	\u0000	\uffff	\u0000	16-bit Unicode	'3'

*http://java-answers.blogspot.com/2012/01/primitive-data-types-in-java.html

- **Attention aux erreurs de calcul !!!**
  - P.ex. avec des double 1.33 - 1.3 donne 0.030000000000000027
  - Cause : les nombres décimaux sont stockés sous forme binaire dans l'ordinateur ➔ pas une représentation exacte
    - précision des doubles: 15 à 17 chiffres significatifs

- Pour les calculs où la précision est importante (p.ex. finance ou calculs scientifiques), utiliser la classe **java.math.BigDecimal** (attention, assez lent), ou convertir dans un nombre type (p.ex. long) et contrôler les arrondis

Les conversions de types primitifs

- Les conversions **implicites**
  - Par affectation
  - Par promotion arithmétique

```
int anIntegerVariable = 3 ;
long aLongVariable = anIntegerVariable ;
```

Opérateurs unaires: -, +, --, ++	byte, short, char -> int
Opérateurs binaires: +, -, *, /, %, <, <=, >=, ==, !=	Si l'un des opérandes est de type double/float/long, l'autre est converti en double/float/long Sinon les deux sont convertis en int

- Les conversions **explicites** (**cast ou transtypage**)
  - Java langage fortement typé -> erreurs à la compilation (ou conversions implicites non appropriées)
  - ➢ Spécifier dans le code comment convertir

```
int i = 64;
char c = (char) i; // c = @

int l = 2;
int j = 3;
double k = (double) j / (double) l ;  // k = 1.5
```

Les conversions de types primitifs

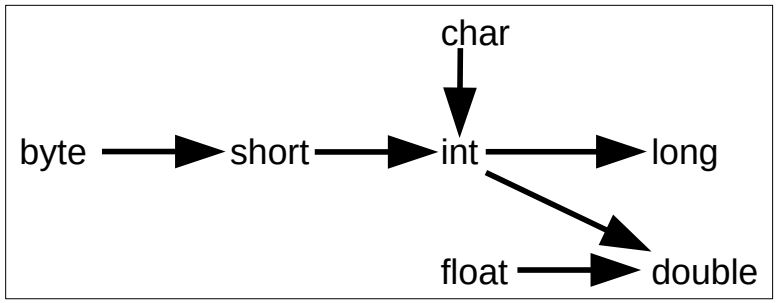
- Dangers des conversions: la perte d'informations
  - Perte de données
  - Perte de précision

```
// Perte de données (int -> short)
int i = 32768
short s = (short) i; // s = -32767

// Perte de précision (int -> float)
int n = 123456789;
float f = n; // f vaut 1.23456792E8

// Perte de précision (int / int = int)
int l = 2;
int j =3;
double k = j / l ; // résultat 1
double k2 = (double) ( j / l ) ; // résultat 1
```

- Conversions sans perte d'informations



Les Entrée/Sortie standards

- Afficher sur la console une chaîne de caractère :
  - **System.out.print(** *myString* **)**
  - **System.out.println(** *myString* **)**

```
System.out.println( "Hello" );

int i = 0 ;
System.out.print( "i = " );
System.out.println( i );

System.out.println( "i = " + i );
```

```
Hello
i = 0
i = 0
```

- Lecture d'un caractère saisi au clavier : (char) **System.in.read()**

```
char c = (char) System.in.read() ;
```

  - Très limité (p.ex. impossible de lire directement un simple entier)
  - ➢ Mise en place à partir de Java 1.5 d'un type **Scanner** offrant beaucoup plus de fonctionnalités

Exercice: GravityCalculator[1]

1: MIT, "Introduction to Programming in Java"

- Ecrire dans Eclipse un programme Java permettant de calculer la position d'un objet en chute libre dans l'atmosphère terrestre. Utiliser la formule suivante:

$$x(t) = 0.5\ a\ t^2 + v_i t + x_i$$

avec l'accélération a=-9,81 m/s^2, la durée de la chute t (s), la vitesse initiale v_i (m/s) et la position initiale x_i.

1. Dans un premier temps, faire un programme permettant de calculer la position d'un objet après une chute de 10 secondes (vitesse et position initiales à zéro).

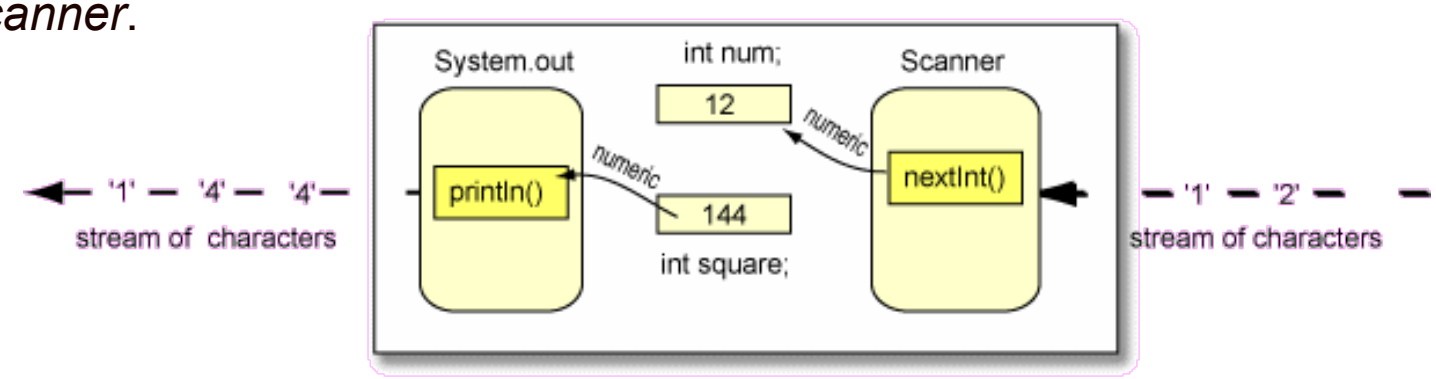
```
class GravityCalculator {
        public static void main(String[] args) {
                double gravity = -9.81;
                double initialVelocity = 0.0;
                double fallingTime = 10.0;
                double initialPosition = 0.0;
                double finalPosition = 0.0;

                System.out.println("An object's position after " + fallingTime + " seconds
is " + finalPosition + " m.");
        }
}
```

Exercice: GravityCalculator[1]

1: MIT, "Introduction to Programming in Java"

2. Puis, lire au clavier les valeurs. Pour faire cela, vous utiliserez la classe *Scanner*.



EchoSquare Java Program

- Scanner: permet de lire des valeurs de types différents à partir d'un flot de données.
  1. Importer la bibliothèque au début du programme: import java.util.Scanner;
  2. Dans une méthode, créer un « Scanner »

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

  3. Lire les valeurs grâce à l'une de ces méthodes

```
int entier = NomduScanner.nextInt();
long entierlong = NomduScanner.nextLong();
float flotant= NomduScanner.nextFloat();
double reel = NomduScanner.nextDouble();
String chaine = NomduScanner.nextLine();
…
```

Exercice: GravityCalculator[1]

1: MIT, "Introduction to Programming in Java"

3. Exploiter la variable ***args*** pour récupérer ces valeurs à partir du terminal.

```
…
public static void main(String[] args) {
        double gravity = -9.81;

        if(args.length > 2) {
                System.out.println("Initial velocity:" + args[0]);
                double initialVelocity = Double.parseDouble( args[0] );
…
```

- Paramètre ***args***: tableau de chaines de caractères qui contient les paramètres passés à l'application java sur la ligne de commande.
  - Pour convertir la chaîne de caractères en nombre: utiliser les méthodes ***Integer.parseInt(s), Float.parseFloat(s)***, et ***Double.parseDouble(s),*** où s est une variable de type chaîne de caractères.

Exercice: GravityCalculator[1]

1: MIT, "Introduction to Programming in Java"

4. Utiliser ***BigDecimal*** pour contrôler la précision de vos opérations
   1. Importer les librairies:

```
import java.math.BigDecimal;
import java.math.RoundingMode;
```

   2. Convertir les valeurs en ***BigDecimal***

```
BigDecimal gravity = new BigDecimal("-9.81");
BigDecimal initialVelocity = new BigDecimal(args[0]);
…
```

   3. Faire les calculs en utilisant les méthodes de ***BigDecimal***

```
BigDecimal finalPosition = gravity.multiply( fallingTime.pow(2) ) ;
finalPosition = finalPosition.multiply( BigDecimal.valueOf(0.5) );
finalPosition = finalPosition.add( initialVelocity.multiply(fallingTime) );
```

   4. Indiquer la précision souhaitée pour le résultat

```
finalPosition = finalPosition.setScale(5, RoundingMode.HALF_UP);
```

Exercice: GravityCalculator[1]

1: MIT, "Introduction to Programming in Java"

5. Faire la même chose en jouant sur les conversions de types et en contrôlant les arrondis
   - beaucoup moins coûteux et moins de code

```
double gravity = -9.81;
double initialVelocity = Double.parseDouble( args[0] );
double fallingTime = Double.parseDouble( args[1] );
double initialPosition = Double.parseDouble( args[2] );

double factor = 1e5;
double finalPosition = (long) ( 0.5 * gravity * fallingTime * fallingTime +  initialVelocity * fallingTime + initialPosition ) / factor;
```