JAVA

Introduction à la programmation objet avec java – 1/4

SOMMAIRE

- Caractéristiques de java (40mn)
- Eléments du langage (40mn)
- Les *core* API java (20mn)

Partie 1

Caractéristiques de java

Qu'est-ce que Java ?

- Un langage de programmation orienté objets
- Une architecture de *Virtual Machine*
- Un ensemble d'API variées
- Un ensemble d'outils (le JDK)

Note : les API (=bibliothèque de fonctions) sont équivalentes des librairies standards de C/C++, en plus riche : pour les interfaces graphiques, le son, le multithreading, la programmation réseau, ...

Bref historique

- **1993** : projet Oak (langage pour l'électronique grand public)
- **1995** : Java / HotJava à WWW3
- **Mai 95** : Netscape prend la licence
- **Sept. 95** : JDK 1.0 b1
- **Déc. 95** : Microsoft se dit intéressé
- **Janv. 96** : JDK 1.0.1
- **Eté 96** : Java Study Group ISO/IEC JTC 1/SC22
- **Fin 96** : RMI, JDBC, JavaBeans, ...
- **Fév. 97** : JDK 1.1

Un exemple de code (Hello World!)

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

- La compilation: le programme javac

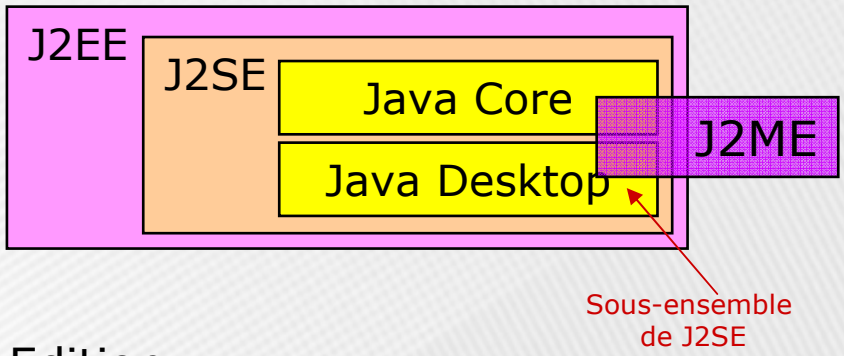
- Execution :on interprète le fichier byte code
  - > java Classe
    - où « Classe » est le nom de la classe contenant main()
  - par exemple: >java HelloWorld

Java : Le JDK

- JDK = Java Development Kit
- JDK = L'environnement minimal pour écrire des programmes Java
- le JDK est gratuit et disponible en ligne sur le site de SUN (http://java.sun.com)
- Il existe une version du JDK pour chaque plateforme : Solaris, Windows, Linux, Mac OS,IBM AIX, HP-UX...)
  - donc télécharger la bonne version sur le site de SUN
- Le JDK contient principalement:
  - un compilateur de sources java : **javac**
  - un interpréteur de *byte code* : **java (la JVM)**
  - des API pour gérer le son, interfaces graphiques, le multithreading, le réseau, … : Les API **Java *core***

Java: les différentes versions

- JDK ou JRE?
  - JDK = Java Development Kit
  - JRE = Java Runtime Environment
  - JDK = JRE + compilateur
  - JRE = JVM + API Java Core
    - (JVM=Java Virtual Machine)



- J2SE ou J2EE ou J2ME?
  - J2SE = Java 2 Platform, Standard Edition
    - Environment pour des applications classiques (de bureau) sous sur les OS classiques Windows/Linux/Mac OS
    - Sert de fondation à J2EE
  - J2EE = Java 2 Platform, Enterprise Edition
    - Environment pour des applications multi-tier d'enterprise
  - J2ME = Java 2 Platform, Micro Edition
    - Environment pour des applications embarquées, en particulier téléphones mobiles et PDA
    - Une version reduite et limitée de J2SE

- A noter qu'il existe une version de java pour la carte à puces (JavaCard) avec un status un peu particulier

Les caractéristiques du langage Java

- Orienté objets
- Interprété
- Portable
- Simple
- Robuste
- Sécurisé
- Multi-threads
- Distribué

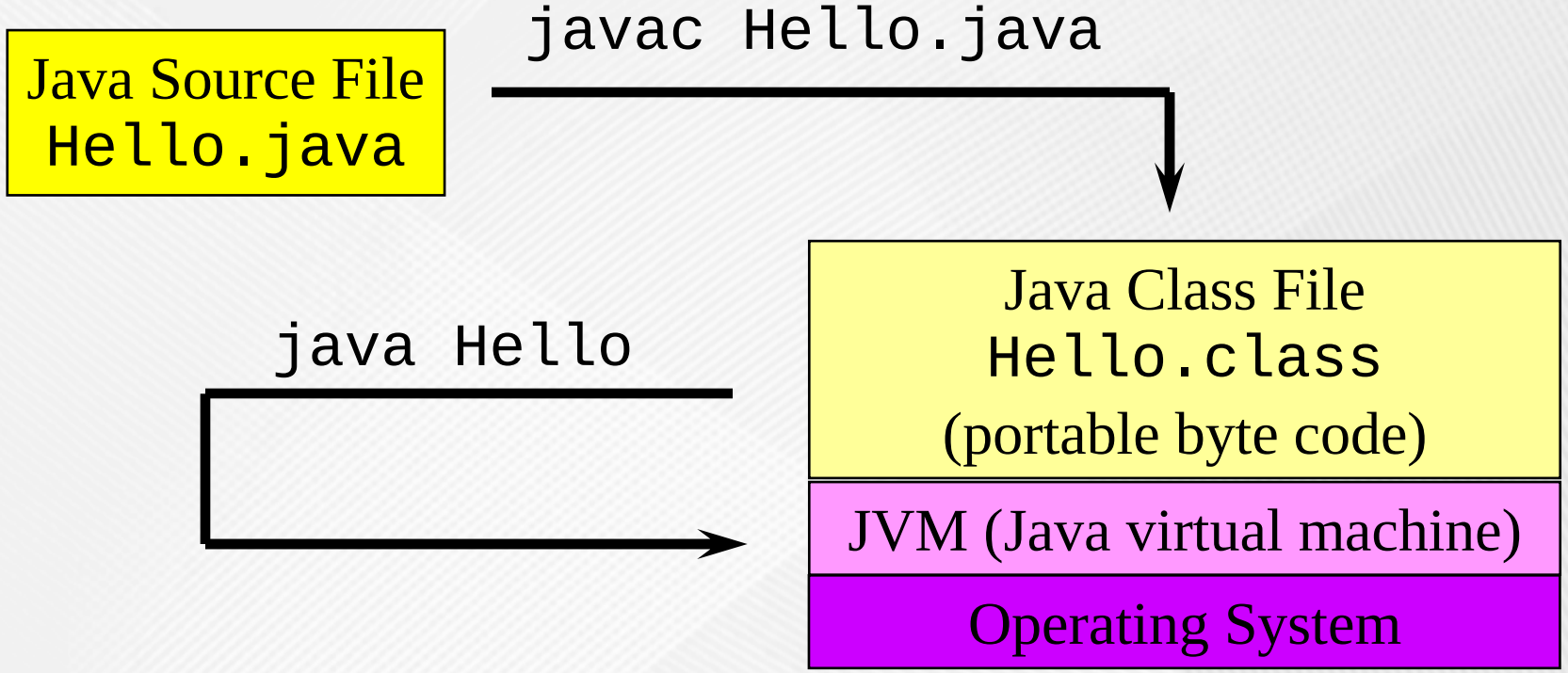
Java est un langage orienté objets

- Tout est classe (pas de fonctions ni de variables globales) sauf les types primitifs (`int`, `float`, `double`, ...)
- Toutes les classes dérivent de `java.lang.Object`
- Héritage simple pour les classes
- Héritage multiple interdit pour les classes (notion d'interfaces en remplacement)
- Les objets se manipulent via des références
- Une API objet standard est fournie (core API)
- La syntaxe est proche de celle de C

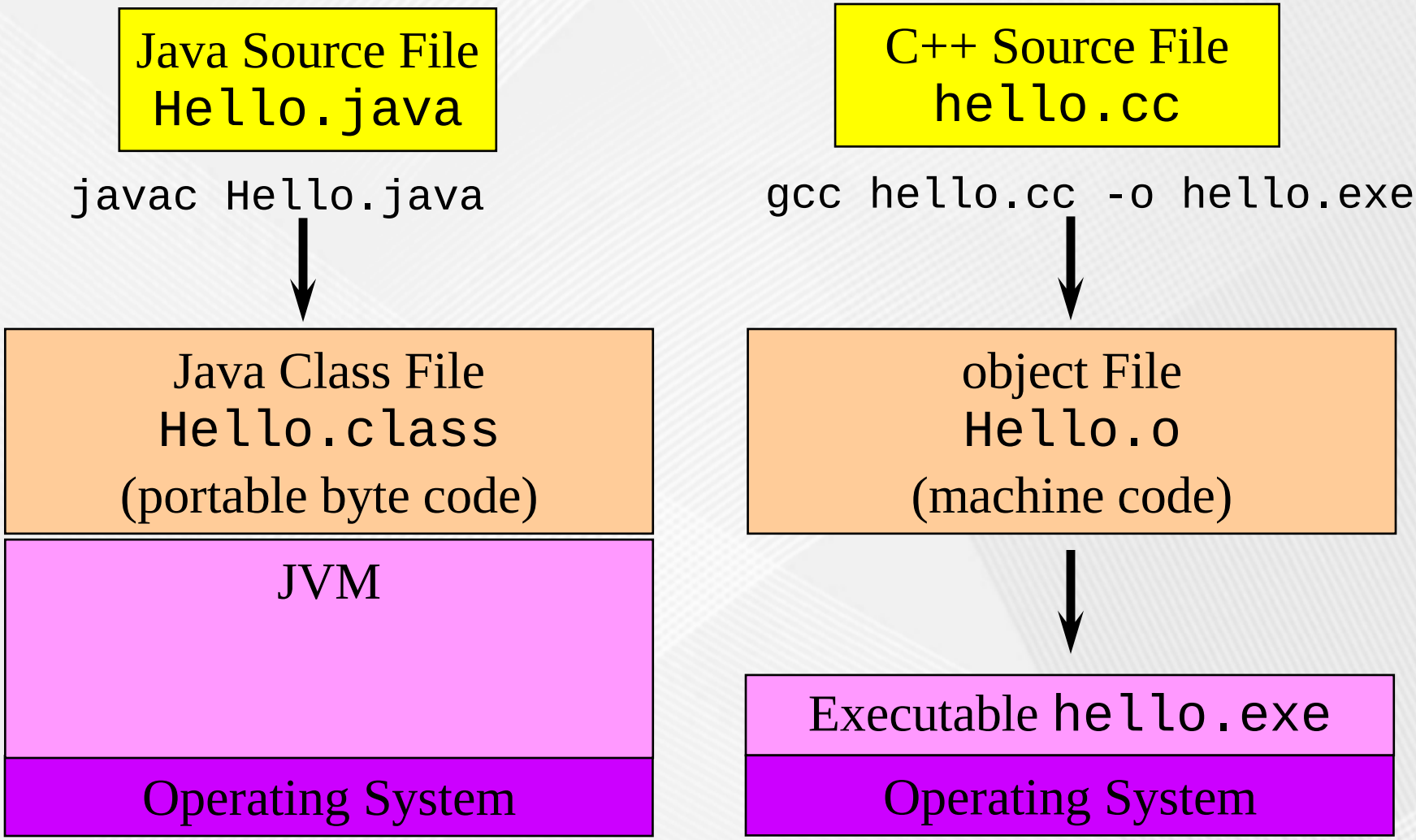
Java est portable

- Le compilateur Java génère du *byte code*
- ce *byte code* est interprété par une JVM
- La *Java Virtual Machine* (JVM) est présente sur Unix, Win32, Mac, OS/2, Netscape, IE, ...
- La taille des types primitifs est indépendante de la plate-forme.
- Java supporte un code source écrit en Unicode
- Java est accompagné d'une librairie standard qui evite tout appel systeme natif

Java est portable : Modèle VM



Byte Code vs. Machine Code



Java est robuste

- A l'origine, c'est un langage pour les applications embarquées
- Gestion de la mémoire par un *garbage collector*
- Pas d'accès direct à la mémoire (pas de pointeurs)
- Mécanisme d'exception
- Accès à une référence `null` → exception
- compilateur contraignant (erreur si exception non gérée, si utilisation d'une variable non affectée, ...)
- Tableaux = objets (taille connue, débordement → exception)
- Seules les conversions sûres sont automatiques
- Contrôle des *cast* à l'execution

Java est sécurisé

- Indispensable avec le code mobile
- Prise en charge dans l'interpréteur
- Trois couches de sécurité :
  - Verifier : vérifie le byte code
  - Class Loader : responsable du chargement des classes
  - Security Manager : accès aux ressources
- c'est d'autant plus important que java permet :
  - le chargement de "sous programmes" (classes) à la volée
  - y compris le téléchargement à travers le réseau! (par exemples applets, RMI..)

Java est multi-thread

- Threads Intégré dans le langage
  - garbage collector dans un thread de basse priorité
- Accès concurrents à objet gérés (par un monitor)
- Mais: Implémentation propre à chaque JVM
  - Difficultés pour la mise au point et le portage

Java est distribué

- API réseau (java.net.Socket, java.net.URL, ...)
- Chargement / génération de code dynamique (classloader)
- Applet (programmes intégré dans les navigateurs web)
- Servlet (programmes sachant répondre aux navigateurs web coté serveur)
- *Remote Method Invocation*: appels de méthodes vers des objets à travers le réseau
- JavaIDL (CORBA) : idem mais communication standard : permet de dialoguer avec des programmes C / C++ et autres.

Partie 2

Eléments du langage

Les différences avec C/C++

- Pas de structures ni d'unions
- Pas de types énumérés (sauf en 1.5)
- Pas de *typedef*
- Pas de préprocesseur (#define, #ifdef,#include …)
- Pas de variables ni de fonctions en dehors des classes
- Pas de fonctions à nombre variable d'arguments (sauf en 1.5)
- Pas d'héritage multiple de classes
- Pas de types paramétriques (*template*)
- Pas de surcharge d'opérateurs (redefinition de "+", "+=", etc)
- Pas de passage par copie pour les objets
- Pas de pointeurs, seulement des références

Structure du langage

- La notation Java est similaire à du C :
  - Les lignes se terminent par un « ; »
  - La déclaration des variables est identique
    - mais pas forcément en début de bloc
  - Les tableaux peuvent se déclarer de 2 manières dont l'une correspond à celle du C
  - L'affectation est le symbole « = »
  - La comparaison est le symbole « == »
- Commentaires : `//` ou `/* */` (ou `/** */`)
  - `// Ceci est un commentaire`
  - `/* Ceci est un commentaire */`
  - `/** Ceci est un commentaire « javadoc » */`

Les types primitifs

- Nouveauté par rapport à C: boolean(true/false), byte (1 octet)
- tailles figées pour les types “standard”: char (2 octets), short (2 octets), int (4 octets), long (8 octets), float (4 octets), double (8 octets)
- Les variables peuvent être déclarées n'importe où dans un bloc (pas forcément au debut)
- Les affectations non implicites doivent être *castées* (sinon erreur à la compilation).

```
int i = 258;
long l = i;   // ok
byte b = i;   // error: Explicit cast needed to convert int to byte
byte b = 258; // error: Explicit cast needed to convert int to byte
byte b = (byte)i; // ok mais b = 2
```

structures de contrôle

- Essentiellement les mêmes qu'en C
  - `if`, `switch`, `for`, `while`, `do while`
  - `++`, `+=`, `&&`, `&`, `<<`, `>>`, `?:`
  - `Continue, break`
- Notion de bloc de code :
  - matérialisés par des accollades { ...}
  - les blocs peuvent démarrer n'importe où
  - les blocs sont imbricables à l'infini
- Définition de variable interne « à la volée »:

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

Ici I n'est visible qu'a l'intérieur de la boucle (mais plus après la boucle).

Les exceptions (1)

- Elles permettent de séparer un bloc d'instructions de la gestion des erreurs pouvant survenir dans ce bloc.
- Format:

```
try {
    // Code pouvant lever des IOException ou des SecurityException
}
catch (IOException e) {
    // Gestion des IOException et des sous-classes de IOException
}
catch (Exception e){
    // Gestion de toutes les autres exceptions
}
```

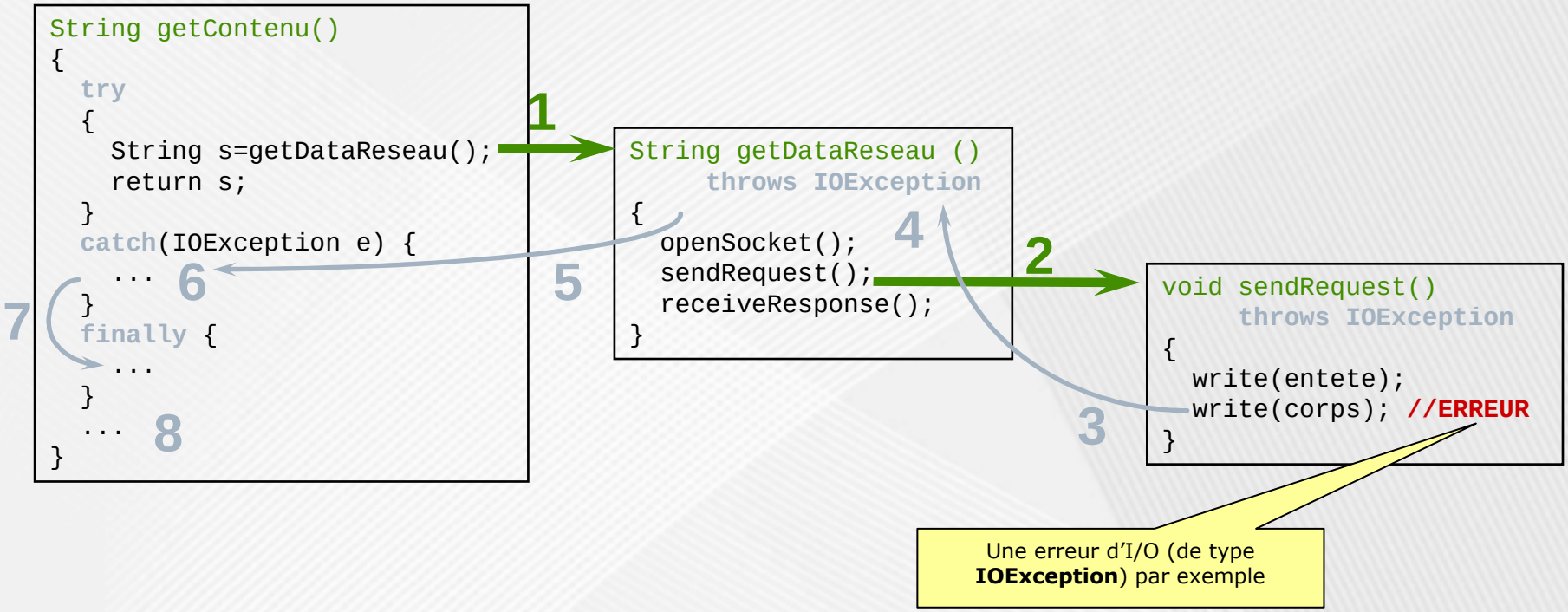
Les exceptions (2)

- Ce sont des instances de classes dérivant de java.lang.Exception
- La levée d'une exception provoque une remontée dans l'appel des méthodes jusqu'à ce qu'un bloc catch acceptant cette exception soit trouvé
- Si aucun bloc catch n'est trouvé, l'exception est capturée par l'interpréteur (JVM) qui l'affiche et le programme s'arrête.

- Un bloc optionnel finally peut-être posé à la suite des catch. Son contenu sera exécuté qu'il ait eu une exception ou pas.
  - Notamment après un catch ou après un break, un continue ou un return dans le bloc try

Les exceptions (3)

Exemple de séquencement :



Les tableaux

Déclaration

```
int[] array_of_int;          // équivalent à : int array_of_int[];
Color[][][] rgb_cube;
```

Création et initialisation

```
array_of_int = new int[42];
rgb_cube = new Color[256][256][256];
int[] primes = {1, 2, 3, 5, 7, 7+4};     // avec prédicat d'initialisation
String[] colors={"red","blue","green"};
```

Utilisation

- Indexation d'un élément :

```
int[] tableau = new int[4] ;
for (int i = 0 ; i  < 4 ; i ++)
   tableau[i] = i ;
```

- Passage en paramètre :

```
void reverse ( int[] tableau )... ou  void reverse ( int tableau[] )...
```

- Taille et debordement :

```
int l = array_of_int.length; // l = 42
int e = array_of_int[50];    // Lève une ArrayIndexOutOfBoundsException
```

Classes et objets : exemple

```
class Circle{
   double x, y;      // Coordonnée du centre
   double r;         // rayon du cercle

   Circle(double R) {
       r = R;
   }
   double area() {
       return 3.14159 * r * r;
   }
}

class MonPremierProgramme {
   public static void main(String[] args) {
       Circle c;           // c est une référence sur un objet Circle, pas un objet
       c = new Circle(5.0); // c référence maintenant un objet alloué en mémoire
       c.x = c.y = 10;
       System.out.println("Aire de c :" + c.area());
   }
}
```

NOTIONS A ABORDER:
Propriété / méthode
référence d'un objet: c
créer un objet: l'opérateur new
constructeur

Déclaration d'une classe

- Format classique d'un fichier java:

```
import classes;

class NomClasse
{
   Attribut(s) de la classe;
   Méthode(s) de la classe;

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

import: on le voit plus loin dans le cours

main() est optionnel: il permet d'executer la classe via la JVM: >java NomClasse

  - Une classe est un ensemble d'attributs et de méthodes : **les membres**
  - le corps des méthodes sont définies directement à l'intérieur de la classe

La surcharge

- Dans une classe plusieurs méthodes ou constructeurs peuvent coexister avec un même nom si elles ont des signature différentes.
- La signature est constituée de :
  - du nom de la méthode ou du constructeur
  - du type de chacun de ces paramètres.
- Exemple :

```
class Circle {
    double x, y, r;
    void Initialiser(double X, double Y, double R) {
            x = X; y = Y; r = R;
    }
     void Initialiser(Circle c) {
            x = c.x; y=c.y; r=c.r;
    }
     void Initialiser() {
            Initialiser(0.0, 0.0, 0.0);
    }
}
```

NOTE: “Initialiser” peut etre remplacé par le *constructeur*

Mot-clé static

- Placé devant une propriété ou une méthode
  - Exemple : `static int count = 0;`
  - Exemple : `static Circle bigger(Circle c1, Circle c2) {..}`
- Effet: Il n'est pas nécessaire d'instancier la classe pour accéder à ses membres statiques (de l'extérieur d'une classe on peut faire directement `Nom_classe.nom_statique`)
  - Les propriétés **static** sont communes à toutes les instances de la classe
  - Les méthodes **static** sont l'equivalent des fonctions en C
- Les membres statiques sont accessible via la classe (au lieu d'une instance de la classe pour le membre "normaux")

Exemple →

Mot-clé static (2)

```
class Circle {
    static int count = 0;
    double x, y, r;

    Circle(double R) {r = R; count++;)

    boolean plusGrand(Circle c) {
        if (c.r > r) return false; else return true;
    }

    static Circle bigger(Circle c1, Circle c2) {
        if (c1.r > c2.r) return c1; else return c2;
    }
}

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

        int n = Circle.count;                    // n = 0
        Circle c1 = new Circle(10);
        Circle c2 = new Circle(20);
        n = Circle.count;                              // n = 2
        n=Circle.x;                                    // IMPOSSIBLE !!!!
        n=c1.x;                                  // possible ;-)

        boolean sup = c1.plusGrand(c2);           // ok, false car c2 supérieur
        Circle c3 = Circle.plusGrand(c2);        // IMPOSSIBLE !!!!
        Circle c4 = Circle.bigger(c1, c2);       // ok, c4 = c2

        n=c4.count;                                    // possible également!
        Circle.count =10;
    }
}
```

Constantes: static final

- Une constante est une variable static final
  - Exemple: static final float pi = 3.14f ;
- « final » signifie que la valeur n ’est plus modifiable
  - (on peut utiliser final sans static!)
- Dans l’exemple précédent ca donnerait:

```
class Circle {
   static int count = 0;
   static final double PI = 3.14;
   double x, y, r;

   ...
```

final pour éviter par exemple Circle.PI = 4 dans le main.

Les packages

- Un package regroupe un ensemble de classes sous un même espace de nommage.
  - L'intérêt est de regrouper les classes par thème, lien logique, dépendance…
  - on peut se représenter le package comme un répertoire contenant des fichiers classes
  - cela permet de limiter la portée du nom des classes
- un package peut contenir des sous-packages
  - Les noms des packages suivent le schéma : `name.subname.subsubname ...`
- Les API de java sont organisées en packages (ex: java.lang, java.io, java.net …)

Les packages : import

- on peut se représenter le package comme un répertoire
- les noms complets des classes sont :
  `nom_du_package.nom_de_la_classe`
  ou encore
  `package.souspackage.classe`
- pour utiliser des classes sans les préfixer du nom de leur package il faut :
  - `import package.souspackage.classe;`
  - `import package.*; (TOUTE les classes)`
- Exemple:
  - `import java.io.File;` `// 1 seul classe`
  - `import java.io.*;` `// toute les classes`
- **le * n 'est pas récursif !**
- Implicite dans un prog.java : `import java.lang.*;`

Partie 3

Les *core* API java

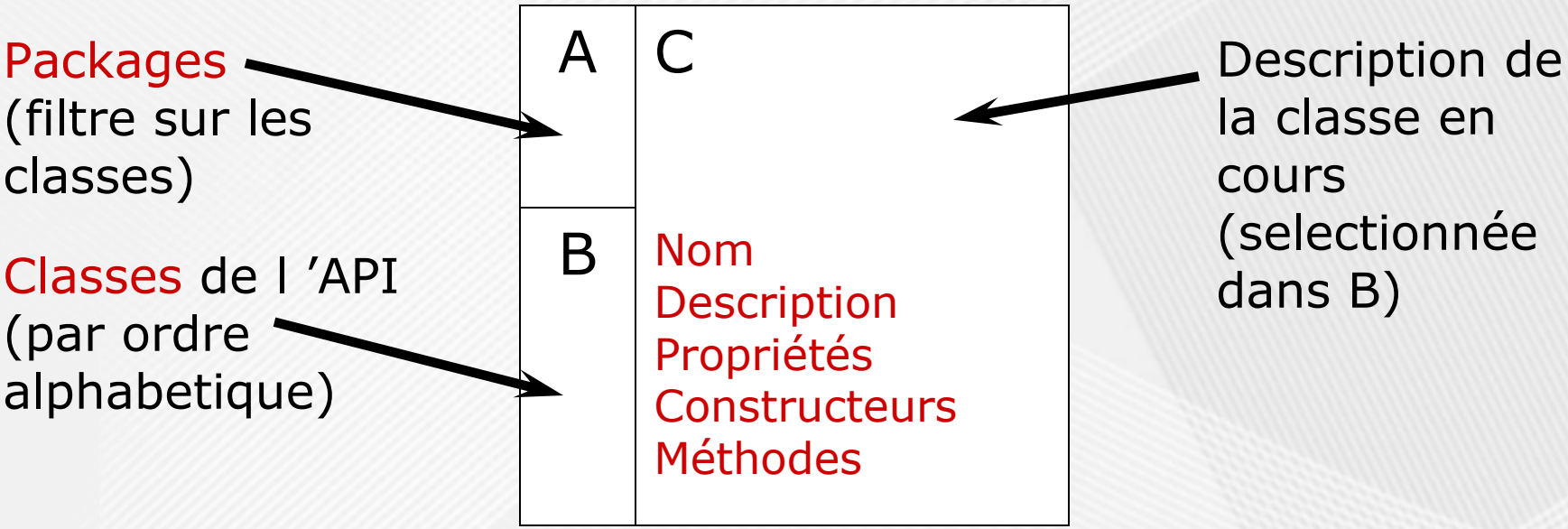
Les *core* API

- Equivalent des librairies standards de C/C++:
  - **java.lang** : Types de bases, Threads, Exception, Math, ...
  - **java.util** : Hashtable, Vector, Stack, Date, ...
  - **java.awt** : Interface graphique portable
  - **java.io** : accès aux i/o par flux (fichiers, stdin, stdout,..)
  - **java.net** : Socket (UDP, TCP, multicast), URL, ...
  - **java.lang.reflect** : introspection sur les classes et les objets
  - **java.sql** (JDBC) : accès homogène aux bases de données
  - **java.security** : signature, cryptographie, authentification
  - **java.rmi** : *Remote Method Invocation*
  - **java.idl** : interopérabilité avec CORBA

- les API sont installées en version binaire (pas de ".h"!)
  - utilisables via la javadoc associée (au format HTML)
  - javadoc à télécharger en même temps que le JDK!

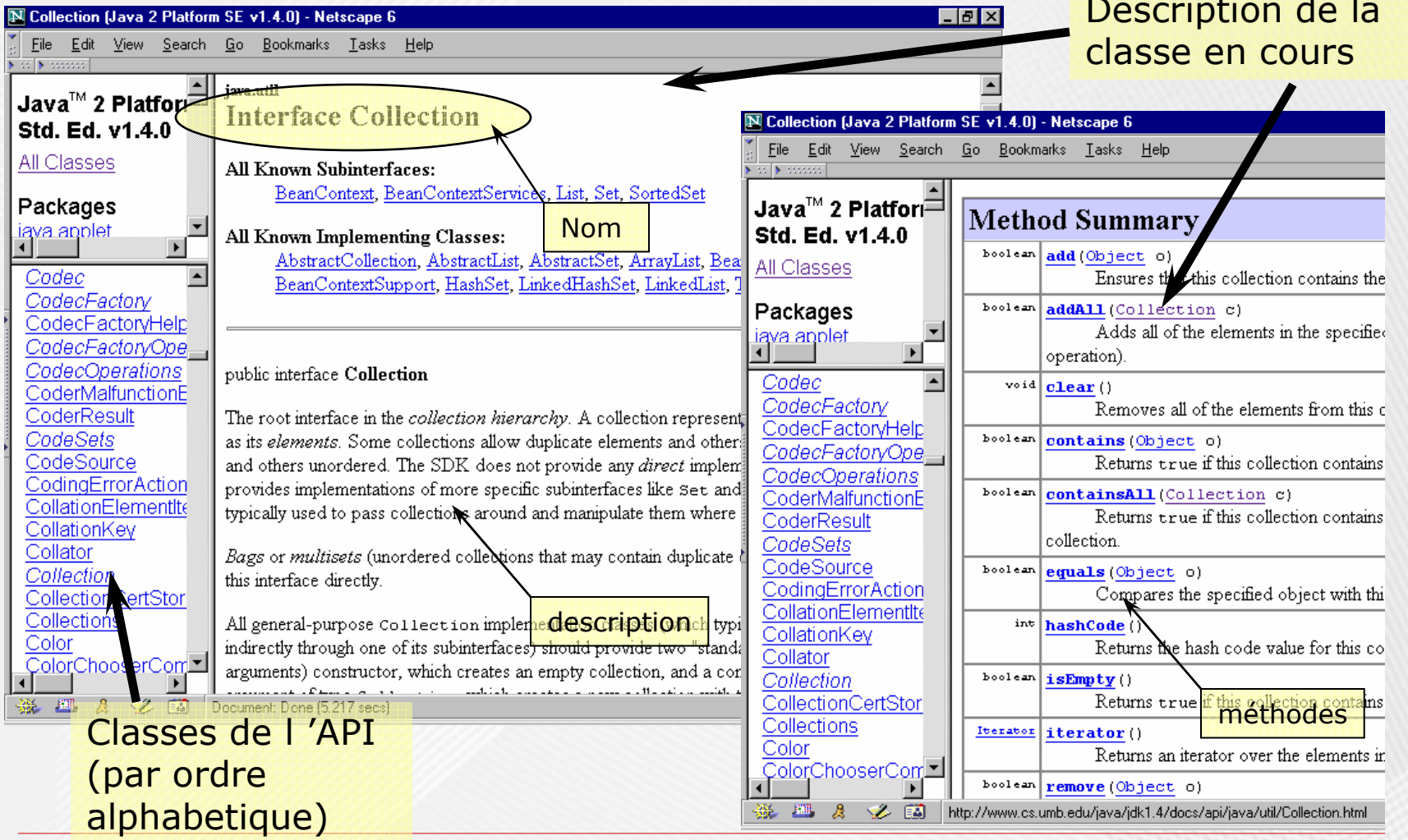
Les *core* API et la Javadoc

- La documentation des API de Java est en ligne à :
  - http://java.sun.com ou http://www.javasoft.com
  - Accès direct à l 'API v1.3:
    - http://java.sun.com/j2se/1.3/docs/api/
- Format de la javadoc (dans le navigateur):



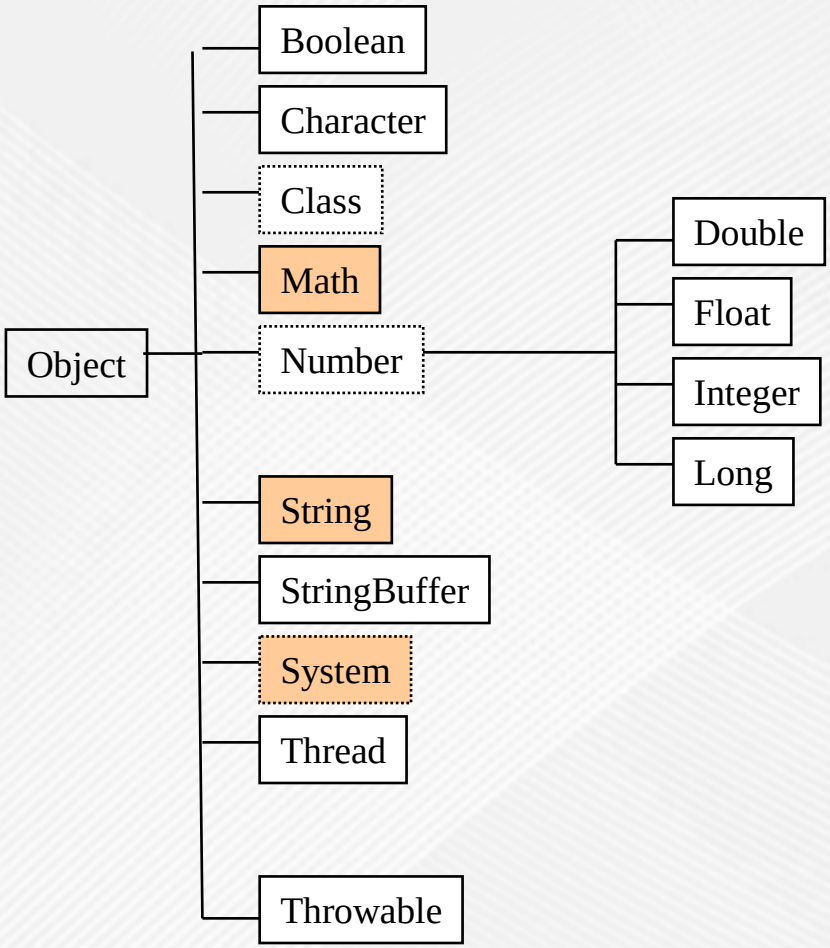
Les *core* API et la Javadoc

Exemple de la javadoc des API:



Les API java: java.lang.*

Classes à approfondir en débutant (sur TP)



Classes Incluses par défaut dans tous les programmes java **(inutile de les importer)**

Les API java: java.lang.Math

- Fonctions mathématiques:
  - `random, abs, sin, co, tan, sqrt, min, max, log, exp…`
- Constantes mathématiques:
  - `Math.PI,…`
- Exemple :

```
double pi = Math.PI; // On accède a l'attribut
   statique de la classe Math
double r = Math.sqrt(2); // On invoque la méthode
   statique de la classe Math
```

Les API java: java.lang.String (1)

- La classe String gère des chaînes de caractères (char).
- Une String n’est pas modifiable.
- Toute modification entraine la création d'une nouvelle String.
- Les valeur littérales ("abc") sont transformées en objets String.
- L'opérateur “+” permet la concaténation de 2 String ("abc"+"def" -->”abcdef")

Les API java: java.lang.String (2)

```
String s = "\u00catre ou ne pas \u00eaetre"; // s="Être ou ne pas être"
int lg = s.length();                         // lg = 19
String s = "Java" + "Soft";                  // s = "JavaSoft"

char[] data = {'J', 'a', 'v', 'a'};
String name = new String(data);

String s = String.valueOf(2 * 3.14159); // s = "6.28318"
String s = String.valueOf(new Date());  // s = "Sat Jan 18 1997 12:10:36"
int i = Integer.valueOf("123");         // i = 123

String s = "java";

if (s == "java") {...}                  // Erreur
if (s.equals("java") {...}              // Ok
```

java.lang.String (3)

- “==“ compare les références pas les contenus!
  - `boolean **equals**(Object chaine)`
  - `boolean **equalsIgnoreCase**(String chaine);`
  - `int **compareTo**(String chaine); **// +,-,0**`
- Exemples:
  - `if (s.equals(“goodbye”)) ... // et non ==`
  - `if (“goodbye”.equals(s)) ... // ok aussi`
- Autres facilités:

```
boolean startsWith(String prefix);
boolean endsWith(String suffix);
String substring(int debut, int fin);
```

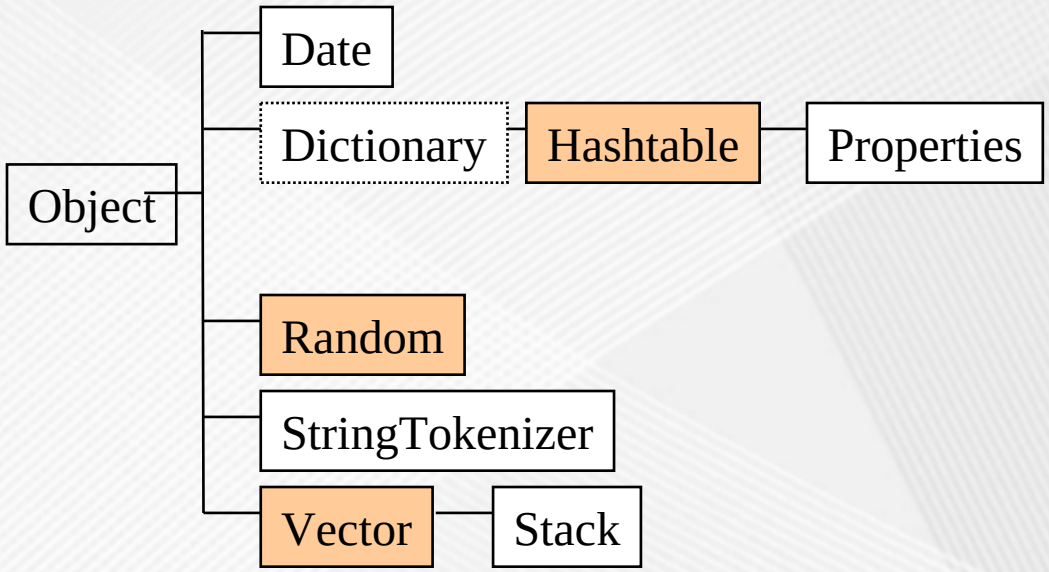
  - `if (s.startsWith(“EN”)) ...`
  - `if(s.substring(0,2).equals(“EN”) ...`

```
int indexOf(String str);
int lastIndexOf(...);
```

  - `int position= s.indexOf(‘C’);`

Les API java: java.util.*

Classes à approfondir en débutant (sur TP)



Les API java: java.util.Vector

- Cette classe gère une collection d'objets dans un tableau de taille dynamique
- Inconvénients:
  - plus gourmand en mémoire qu'un tableau []
  - plus lent également
  - ne gère que des objets (pas de types primitifs)
  - mais permet l'heterogeneité des elements
- Exemple:

```
Vector v = new Vector();

v.add("une chaine");                    // element 0
v.add(new date());                      // element 1: date du jour
v.add(1,new String("abcde"));           // nouvel element 1
v.add(new Vector());                    //un sous-vecteur!
System.out.println(v.size()+v.elementAt(2));  // --> 4abcde
```

Les API java: java.util.Vector

- Résumé de la classe java.util.Vector :

```
void        add(int index, Object element)
boolean     add(Object o)
void        clear()
boolean     contains(Object elem)
Object      elementAt(int index)
Object      get(int index)
int         indexOf(Object elem)
boolean     isEmpty()
Object      remove(int index)
boolean     remove(Object o)
int         size()
Object[]    toArray()
```

Les API java: java.util.Hashtable

- Cette classe gère une collection d'objets au travers d'une “table de hachage”
- la HashTable equivaut à un Vector sauf que les clés sont des **String** au lieu de **numériques**
- Pas de notion d’ordre comme dans un vecteur !!

```
Hashtable ht = new Hashtable();

ht.put("noel", new Date("25 Dec 1997"));
ht.put("une chaine”,”abcde”);
ht.put("un vecteur", new Vector());

Vector v = (Vector)ht.get("un vecteur");
System.out.println(ht.get(" une chaine ")); // --> abcde
```

- la HashTable permet de retrouver la liste des valeur des clés présentes