



**FORMATION JAVA**

# QUI SUIS-JE ?

- Anthony Monteiro
- Spécialisations Android
  - Architecture (Graphique, technique, projet).
  - User Experience
- Application sur le store :
  - UrbanPulse
  - Acadomia (iOS & Android)
- Contact : [contacts@amonteiro.fr](mailto:contacts@amonteiro.fr)
- DropBox :  
[https://www.dropbox.com/sh/wa3zx61c3xoölfw/AAComA2AGp44BVlxCk6\\_cLuua?dl=0](https://www.dropbox.com/sh/wa3zx61c3xoölfw/AAComA2AGp44BVlxCk6_cLuua?dl=0)

# QUI SUIS-JE ?

- Site d'entraînement
  - <https://www.codecademy.com/>
  - <https://codesignal.com/>
  - <https://www.codingame.com>  
<https://www.linkedin.com/learning/java-pour-les-developpeurs-android?originalSubdomain=fr>
  - <https://java.developpez.com/livres-collaboratifs/javaenfants/?page=introduction>

Présentation et tour de table.

# PLAN

- Fonctionnement de Java et outils
- Les méthodes
- Conditions et boucles
- Tableau et algorithme
- Langage Objet
  - Classe, Pointeur
  - Encapsulation
  - Constructeur
  - Polymorphisme
  - Héritage
  - Classe Abstraite
  - Interface
  - Gestion des exceptions

# PLAN

- Collections et algorithmes
- Entrées / Sorties
- Interface graphique
- JDBC
- Webservice
  - Serveur
  - Client

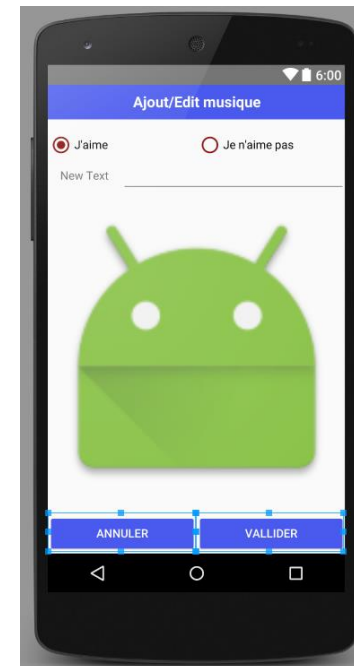


**JAVA**



# JAVA

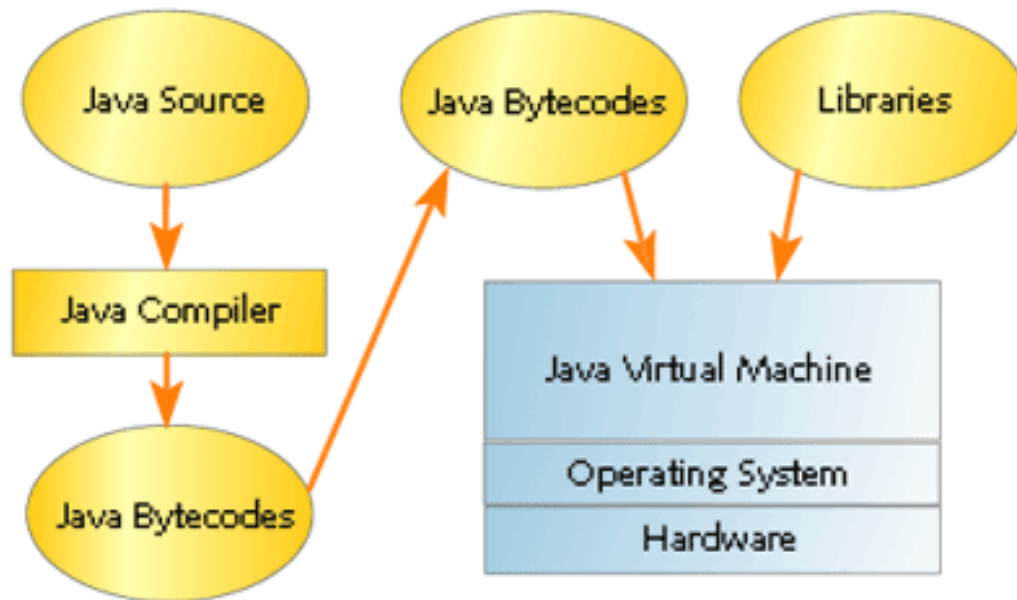
- Développé par Sun Microsystems(Racheté par Oracle)
- Ne pas confondre avec JavaScript
- Multiplateforme.
- Utilisations





# JAVA

- La JVM



# JAVA EST COMPILE ET INTERPRÉTÉ

1. Une classe est définie dans un fichier « .java » qui porte son nom

1

Code source Java  
fichiers \*.java

2

Compilateur  
Java

- **javac.exe** (Sun)

Si c'est une application :  
- **java.exe** (Sun)

Java bytecode  
fichiers \*.class

2. Elle est compilée en un fichier de bytecode « .class »

3. Elle est exécutée (ou interprétée) par une JVM spécifique à chaque plateforme

3

Machine virtuelle  
Java (Interpréteur)  
**Win32**

Machine virtuelle  
Java (Interpréteur)  
**Solaris**

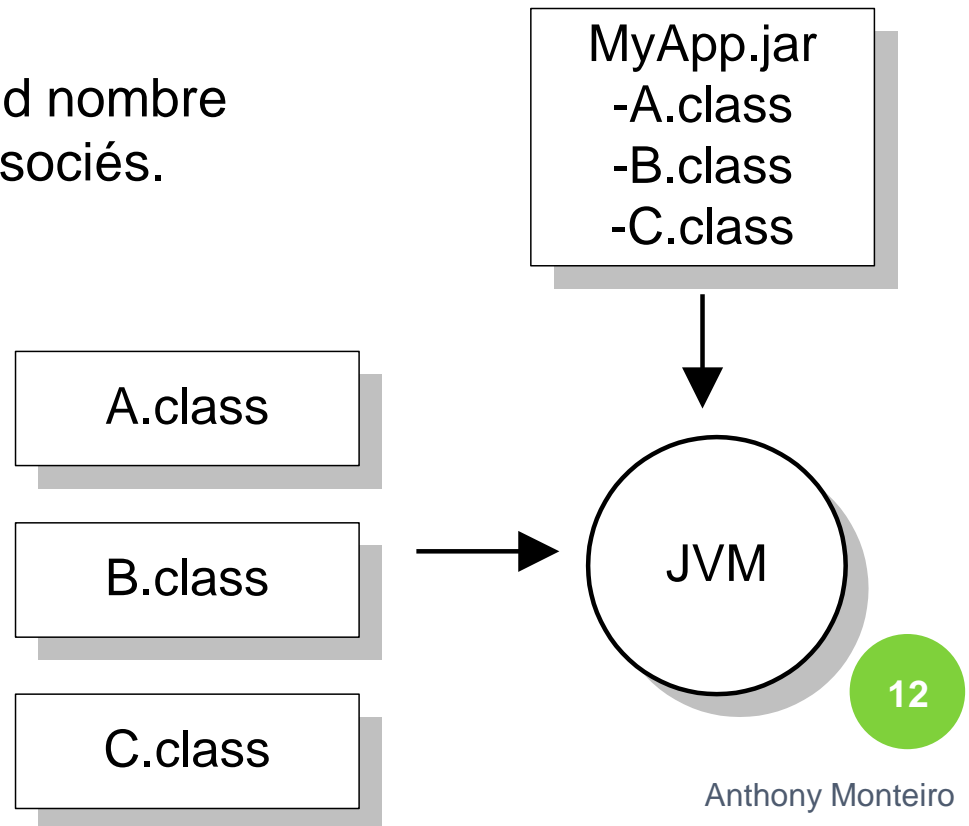
Machine virtuelle Java  
du navigateur  
Web

# LIAISON DYNAMIQUE

- Les applications Java se composent de nombreux fichiers .class — et non d'un seul .exe
- Les définitions de classes sont lues à partir des fichiers et liées à l'*exécution*
  - On parle de « chargement des classes »

# FICHIERS .CLASS ET .JAR

- Les fichiers de bytecode peuvent se trouver dans des fichiers \*.CLASS indépendants ou réunis dans un fichier \*.JAR (sorte de fichier ZIP)
- Les fichiers JAR (Java ARchive) utilisent le format de compactage zip
  - Utile pour déployer un grand nombre de classes et de fichiers associés.



# EXÉCUTION D'UNE APPLICATION

- Invoquer la JVM pour un fichier \*.CLASS (qui peut se trouver à l'intérieur d'un fichier \*.JAR)

> `java NomDeClasse`

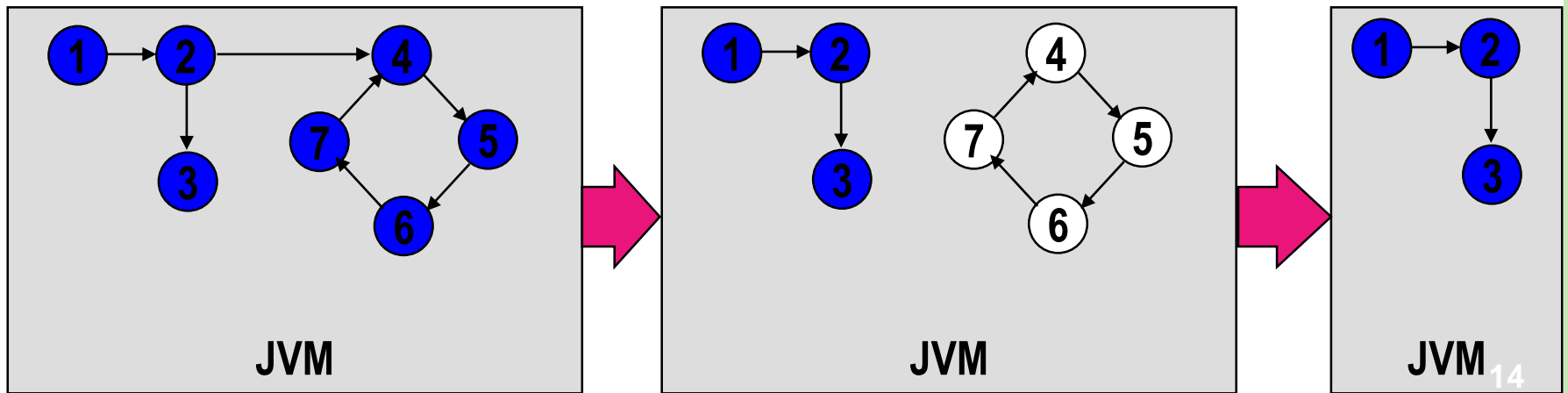
- Ne pas inclure l'extension `.class`

`c:\> java reservations.TestDe`

- Dans cet exemple, la JVM tente d'invoquer la méthode `main()` dans la classe `TestDe` du package `reservations`
- La JVM recherche les répertoires `CLASSPATH` et `jre\ext` pour la classe `TestDe`

# RAMASSE-MIETTES (GARBAGE COLLECTOR)

- Les programmeurs Java ne gèrent pas directement la mémoire
- Le ramasse-miettes (compris dans la JVM) identifie les objets qui ne sont plus accessibles
  - Supprime l'objet en question
  - Remet la mémoire à disposition



# OUTILS

- Pour exécuter du Java
  - JRE : Java Runtime Machine
- Pour faire compiler du Java
  - JDK : Java Development Kit
  - J2SE (Java Standard Edition) : Client lourd
  - J2EE (Java Entreprise Edition) : Application Web
  - JDK Android : Application pour Android
- Pour écrire du Java
  - Eclipse IDE

# MA PREMIÈRE CLASSE

- On définit une classe par fichier
- Le nom du fichier doit correspondre au nom de la classe

Corps de la  
définition de classe  
entre accolades

```
public class Main
{
    // ...
}
```

Main.java



# LA MÉTHODE *MAIN()*

- Première méthode appelée lors du démarrage d'une application

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

- Ne vous préoccupez pas de *static* ni de *String[ ]*
  - Ils seront expliqués plus loin



# LES MÉTHODES

18

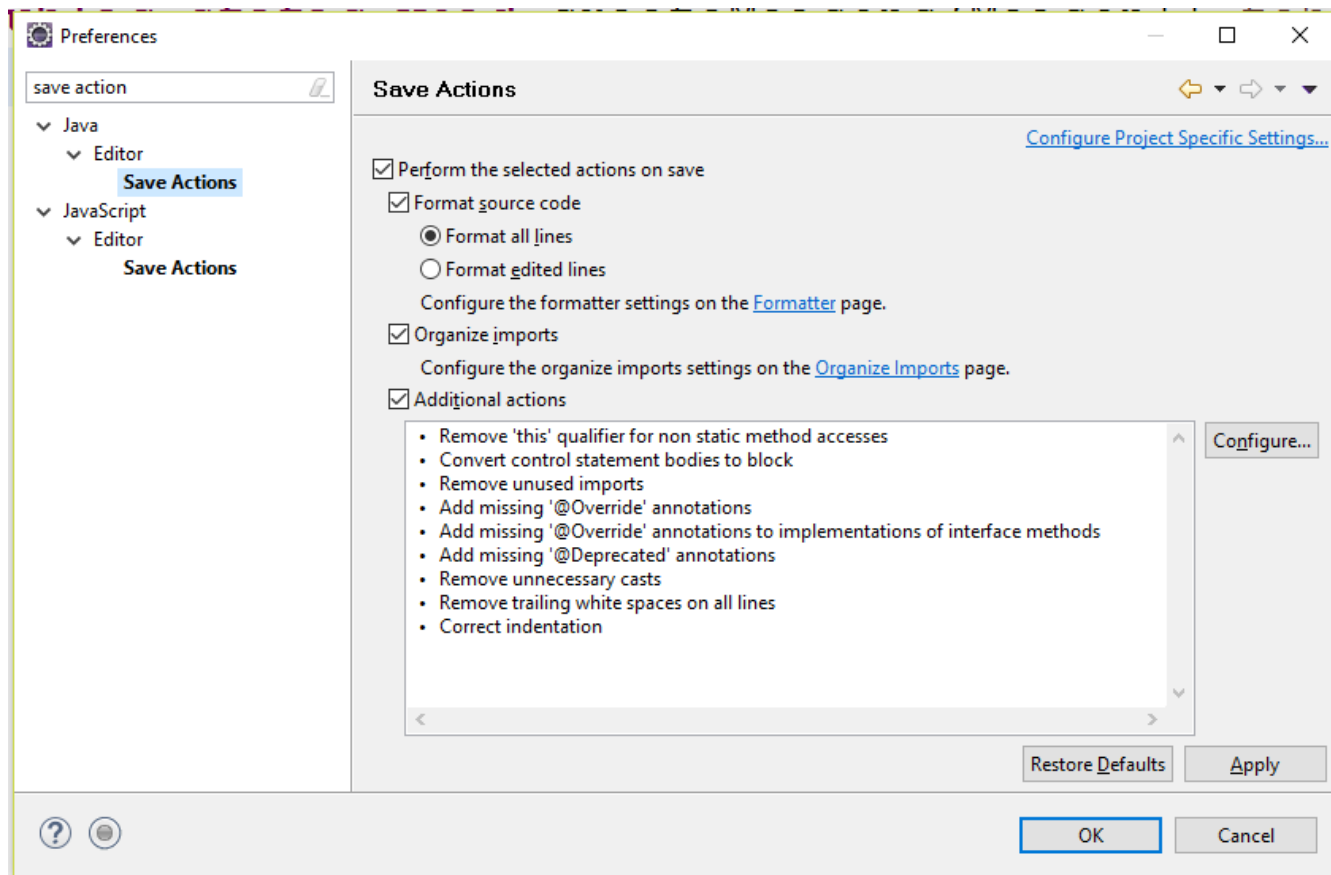
# LA CLASSE AYANT UNE MÉTHODE *MAIN()*

- Toute classe peut contenir une méthode *main()*
- L'une des pratiques de Java consiste à créer une classe appelée *Main* contenant la méthode *main()*

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

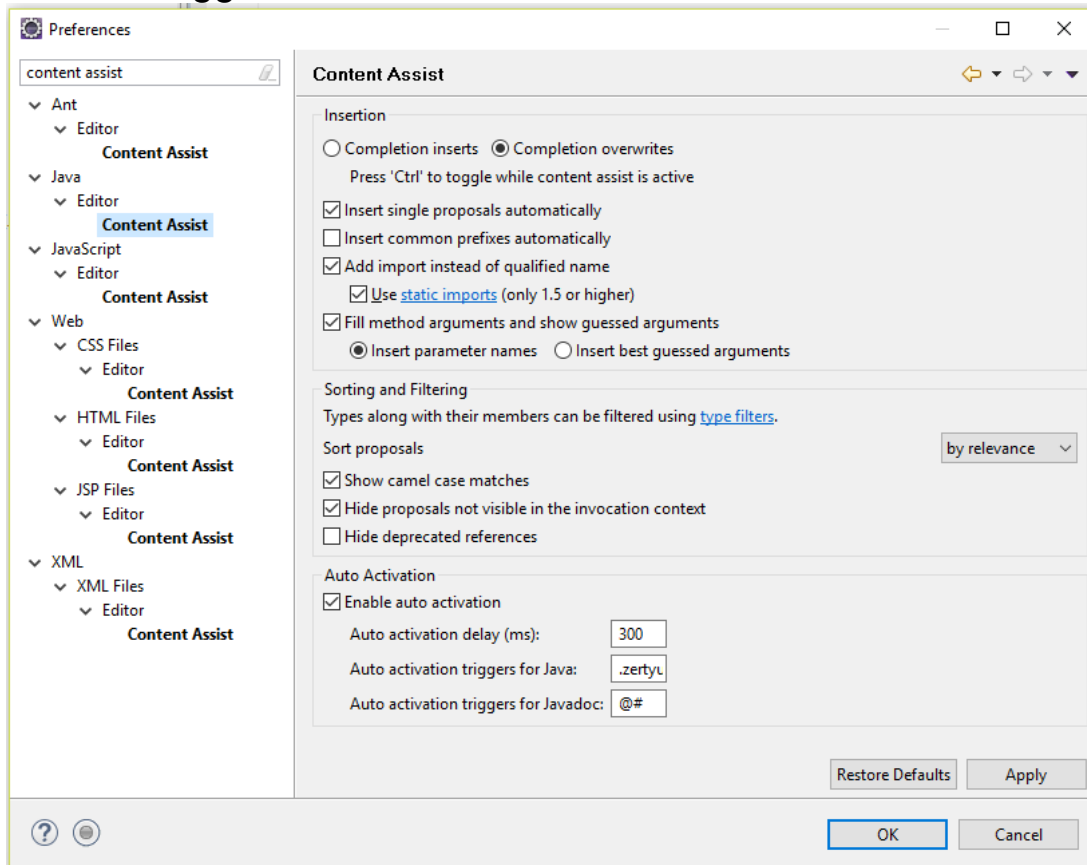
# RÉGLAGES ECLIPSE

- Save Action : Permet des actions au moment du Contrôle+S comme l'indentation, les imports automatiques...
  - Window-> Préférences



# RÉGLAGES ECLIPSE

- Content Assist: Permet d'afficher des propositions lors de l'écriture de code
  - Window-> Préférences
  - Mettre toutes les lettres en minuscule et majuscule ainsi que le '.' dans Auto Activation triggers for Java



# EXERCICE

- Créez un nouveau projet Java du nom de votre choix
- Créez une classe Main.java
- Ajoutez la méthode main du slide précédent.
- Exécutez et voir le message dans la console.

# CRÉATION D'UNE VARIABLE

- Syntaxe pour la création d'une variable
  - <typage> <nom de la variable> = <donnée> ;

- Exemple :

```
int i = 3;  
int j;  
int k = i;  
int l = 5 + 2;  
long m = k + j;  
double pi = 3.14;  
boolean jaiRaison = true;  
char firstLetter = 'A';
```

# TYPES DE BASE (PRIMITIVES)

Type	Taille	Valeurs
boolean	1 bit	« true » ou « false »
char	16 bits	Unicode \u0000 à \uFFFF
byte	8 bits	-128 à 127
short	16 bits	-32768 à 32767
int	32 bits	-2147483648 à 2147483647
long	64 bits	-9223372036854775808 à 9223372036854775807
float	32 bits	+/- 3.402E+38 à +/-1.402E-45
double	64 bits	+/- 1.798E+308 à +/- 4.941E-324



# CONVENTIONS DE NOMMAGE

- Les noms de classe commencent par une lettre majuscule
- Les noms d'attribut et de méthode commencent par une minuscule
- Si un nom comporte plus d'un terme, chaque terme suivant le premier commencera par une majuscule

Type de variable	Source Java
Classe	<code>CompteBancaire</code>
Attribut	<code>tauxDInteret</code>
Méthode	<code>lireValeurFace( )</code>
Constante	<code>MAX_LANCER</code>
Package	<code>com.jeudedes</code>

# CONVERSION DE PRIMITIVES JAVA

```
int i = 10;  
double d = 3.42;
```

```
d = i;
```

OK. Fait passer un entier en double.

```
i = d;
```

Erreur ! Pourquoi ?

```
i = ( int ) d;
```

OK. Les conversions qui tronquent les primitives doivent utiliser des « casts » explicites.

```
byte b = 34L;
```

```
byte b = (byte) 34L;
```

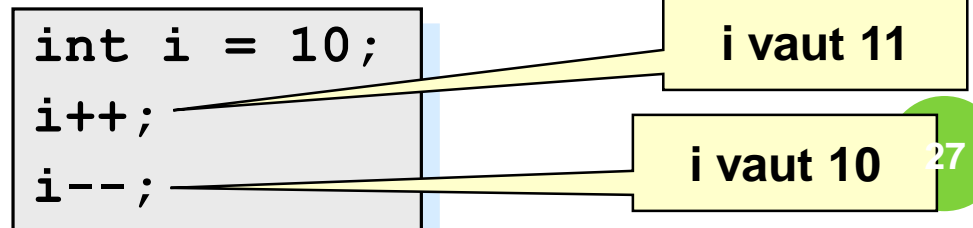
Erreur ! Pourquoi ?

# OPÉRATEURS MATHÉMATIQUES

<i>opérateur</i>	<i>but</i>	<i>exemple</i>	<i>x devient</i>
=	Affectation	<b>x = 10</b>	<b>10</b>
*	Multiplication	<b>x = 10 * 3</b>	<b>30</b>
/	Division	<b>x = 10 / 3</b>	<b>3</b>
%	Modulo	<b>x = 10 % 3</b>	<b>1</b>
+	Addition	<b>x = 10 + 3</b>	<b>13</b>
-	Soustraction	<b>x = 10 - 3</b>	<b>7</b>

## ○ Incrémentation/décrémentation

- « ++ » incrémente
- « -- » décrémente



# CRÉATION D'UNE VARIABLE

- Il existe 2 types de variable :

- Primitif : commence par une minuscule

```
int i = 3;
```

- Objet : commence par une majuscule

```
String nom = "Bob";
```

- Différence : Une valeur primitive possède une valeur par défaut.

```
String prenom = null;
```



# CHAÎNES (« STRINGS »)

- Les chaînes sont des instances de la classe *String*
- Concaténation de chaînes : utiliser l'opérateur +
- Toutes les primitives peuvent être concaténées avec des chaînes

```
String prenom = "Bob";  
String nom = "Henri";  
int age = 34;  
String login = prenom + "_" + nom + "_" + age;
```

```
int c = 1;  
String addition = "JCVD à dit 1+1=" + c + c;
```

# CHAÎNES (« STRINGS »)



# SORTIES SUR CONSOLE

```
//Sans saut de ligne  
System.out.print("Login : " + login);  
//Avec saut de ligne à la fin  
System.out.println("Login : " + login);
```

# VARIABLES « FINAL » ET COMMENTAIRES

- Les variables « final » sont constantes et en lecture seule

```
final int MAX = 10;
```

```
int i = MAX;
```

OK. Lecture à partir d'une variable « final ».

```
MAX = 20;
```

Erreur ! MAX est en lecture seule.

- Commentaires

```
// commentaire sur une ligne
```

```
/*  commentaire  
    sur plusieurs  
    lignes */
```



# EXERCICE

- `int i = 7;`  
`int j = 3;`
- Dans la méthode main affichez dans la console avec une phrase le résultat de:
  - La division entière de i par j
  - Le reste de la division entière de i par j;
  - La division classique de i par j ;
  - La somme des 3 premiers

# CRÉATION DE MÉTHODES

- Une classe peut contenir plusieurs méthodes, mais celle de départ sera la méthode main **avec cette syntaxe précise.**

```
1  public class Depart{
2
3      public static void main(String[] args) {
4          System.out.println("first");
5          exo1();
6          exo2();
7      }
8
9      public static void exo1() {
10         System.out.println("Appel de la méthode exo1");
11     }
12
13     public static void exo2() {
14         System.out.println("Appel de la méthode exo2");
15     }
16 }
```

- Ordre d'exécution des lignes :
  - 3 - 4 - 5 - 9 - 10 - 11 - 6 - 13 - 14 - 15 - 7

# EXERCICE

- Déplacez les exercices de calculs dans une méthode « `exo1` » et appelez la depuis la méthode `main`.

# CRÉATION DE MÉTHODES AVEC PARAMÈTRE

- Il est possible de passer un paramètre à une méthode.

```
1  public class Depart{
2
3      public static void main(String[] args) {
4          System.out.println("Coucou");
5          exo3("Toto");
6          exo3("Tata");
7      }
8
9      public static void exo3(String monParam) {
10         System.out.println("Paramètre de exo3 : " + monParam);
11     }
12
13 }
```

- Ordre d'exécution des lignes :
  - 3 - 4 - 5 - 9 - 10 - 11 - 6 - 9 - 10 - 11 - 7

# EXERCICE

- Créez une méthode qui prend en paramètre un entier (int) lui ajoute 5 et l'affiche dans la console.
- L'appeler avec différents paramètres depuis la méthode main.

# EXERCICE

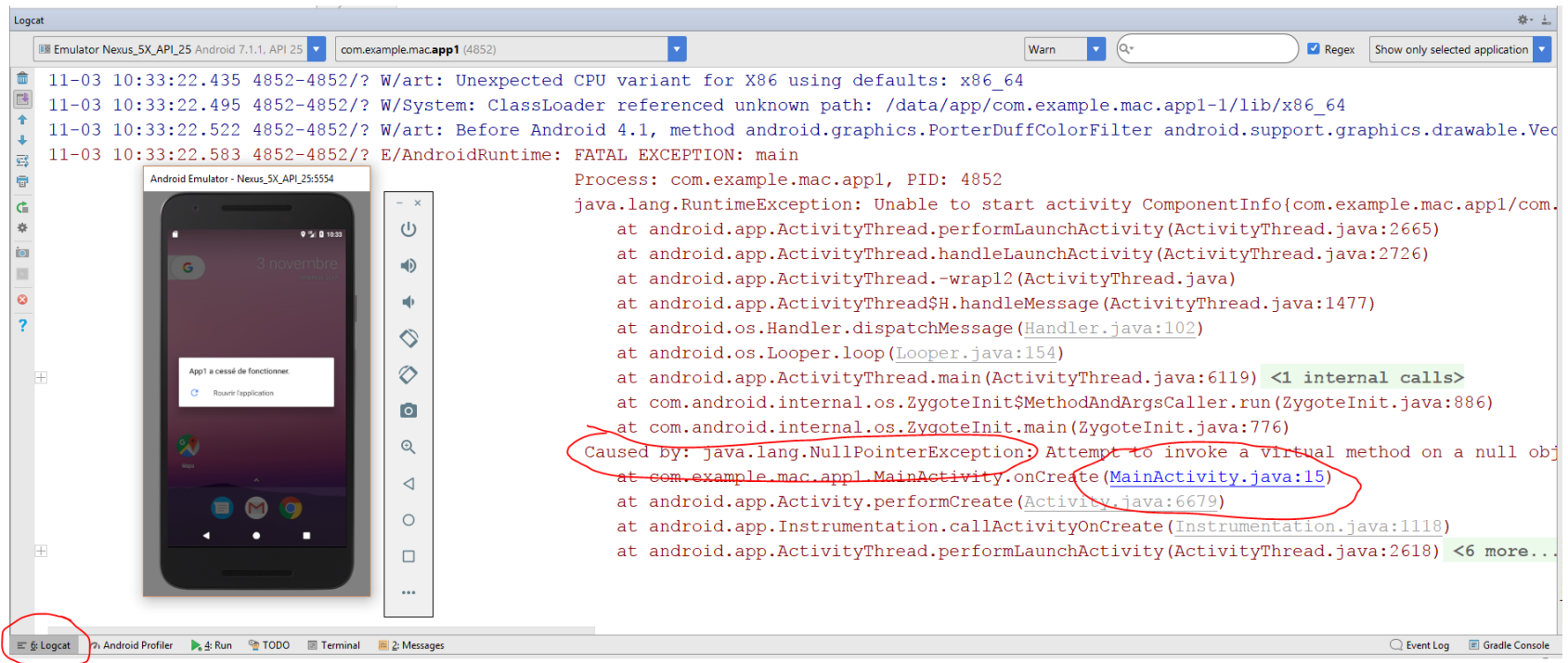
- Communiquer avec la console.

```
public static void main(String[] args) {  
    Scanner scanner = new Scanner(System.in);  
    System.out.print("Veuillez entrer un nombre : ");  
    int nombre = scanner.nextInt();  
    scanner.nextLine(); // Pour consommer le retour à la ligne  
    System.out.println("Le nombre choisi est : " + nombre);  
  
    System.out.print("Veuillez entrer un texte : ");  
    String texte = scanner.nextLine();  
    System.out.println("Le texte choisi est : " + texte);  
}
```

# EXERCICE

- Créer un programme qui demande à l'utilisateur un chiffre, lui ajoute 5 et affiche le résultat dans la console.

# LIRE UNE STACKTRACE





# LE DEBUGAGE

The screenshot shows an IDE with a Java project named 'Dev18-1'. The source code is as follows:

```
1 package toto;  
2  
3 public class Depart {  
4  
5     public static void main(String[] args) {  
6         int resultat = Exo.max(2, 4, 4);  
7         System.out.println("resultat = " + resultat);  
8     }  
9  
10 }  
11
```

Handwritten annotations in red include:

- Numbers 6, 5, 5, and 3 above the first four toolbar icons.
- A red circle around the 'Run' button in the toolbar.
- A red circle around the line number '7' in the code editor.
- A red arrow pointing from the line number '7' to the `System.out.println` statement.
- A red circle around the 'Variables' tab in the debug console.

The debug console shows the following state:

- Thread [main] (Suspended (breakpoint at line 7 in Depart))
- Method: Depart.main(String[]) line: 7
- Variables table:

Name	Value
args	String[0] (id=16)
resultat	4

# Le debugger

1. Mettre un point en cliquant
2. Lancer l'application en mode "Debug"  
(clic droit-> debug as... Java Application)
3. L'application s'arrêtera **si elle passe** sur un point d'arrêt
4. Dans le menu **Variables**, on peut voir l'état des variables. (Windows -> ShowView->Variables)
5. Piloter le débogage
  - a) Passer à l'instruction suivante
  - b) Rentrer dans la méthode
  - c) Sortir de la méthode.
6. Avancer jusqu'au prochain point d'arrêt.
7. Arrêter l'exécution.

# MÉTHODES AVEC PLUSIEURS PARAMÈTRES

- Il est possible de passer plusieurs paramètre à une méthode.

```
1 public class Depart{
2
3     public static void main(String[] args) {
4         System.out.println("first");
5         exo4("Toto : ", 5, 6);
6         exo4("Tata : ", 1, 2);
7     }
8
9     public static void exo4(String monParam1, int monParam2,
10        int monParam3) {
11         //Attention à l'ordre des priorités.
12         System.out.println(monParam1 + monParam2 + monParam3);
13         System.out.println(monParam1 + (monParam2 + monParam3));
14     }
15 }
```

- Ordre d'exécution des lignes :

- 3 - 4 - 5 - 9 - 10 - 11- 12 - 6 - 9 - 10 -11-12 - 7

# EXERCICE

- Créez la méthode `sub(int a, int b)` qui affiche dans la console le résultat de  $a - b$
- L'appeler depuis la méthode `main`.

# MÉTHODES RETOUR DE VALEUR

## ○ Une méthode peut retourner une valeur

```
1  public class Depart{
2
3      public static void main(String[] args) {
4          System.out.println("first");
5          int result = multiply(1, 2);
6          System.out.println("Result : " + result);
7          result = multiply(result, multiply(3, 4));
8          System.out.println("Result : " + result);
9      }
10
11     public static int multiply(int monParam2, int monParam3) {
12         return monParam2 * monParam3;
13     }
14 }
```

## ○ Ordre d'exécution des lignes :

- 3 - 4 - 5 - 11- 12 -13 - 5 - 6 -7 -11-12 -13 - 7 - 11-12 -13 - 7 +8

# EXERCICE

- Créez la méthode `readIntFromConsole()` qui **retourne (n'affiche pas)** 1 et un seul entier lu sur la console.
- Créer une **autre** méthode `multiplyFromConsole()`, qui demande 2 entiers sur la console (appelle 2 fois la méthode `readIntFromConsole`), et retourne leur multiplication

# PORTÉ D'UNE VARIABLE

- Une variable n'est utilisable que dans la méthode qui la crée.
- Le passage d'une variable de type primitif (commence par une minuscule) est faite par copie

```
public static void main(String[] args) {  
    int i = 0;  
    int j = 0;  
    //La valeur de i sera copiée au "i" et "k" de afficherPlus1  
    afficherPlus1(i, i);  
    //Comme elle a été copiée, ici i vaut toujours 0 et k n'existe pas.  
    System.out.println("i=" + i + " j=" + j);  
}  
  
public static void afficherPlus1(int i, int k) {  
    //Ici i et k valent 0, par contre j n'existe pas.  
    //Le i d'ici est différent du i de main, ils ont juste le même nom  
    i = 1; //i=1 et k=0  
    System.out.println("i=" + i + " k=" + k);  
}
```

# PORTÉ D'UNE VARIABLE

- Une variable n'est utilisable que dans la méthode qui la crée.
- Le passage d'une variable de type primitif (commence par une minuscule) est faite par copie

```
public static void main(String[] args) {  
    int i = 2;  
    int j = 3;  
    afficherPlus1(i, i);  
    System.out.println("i=" + i + " j=" + j);  
}  
  
public static void afficherPlus1(int i, int k) {  
    i = 1;  
    System.out.println("i=" + i + " k=" + k);  
}
```



# PORTÉ D'UNE VARIABLE

- Une variable n'est utilisable que dans la méthode qui la crée.
- Le passage d'une variable de type primitif (commence par une minuscule) est faite par copie

```
public static void main(String[] args) {  
    int i = 5;  
    int j = i+1;  
    afficherPlus1(i+1, j-1);  
    System.out.println("i=" + i + " j=" + j);  
}  
  
public static void afficherPlus1(int i, int k) {  
    i = 1;  
    System.out.println("i=" + i + " k=" + k);  
}
```

# PORTÉ D'UNE VARIABLE

- Une variable n'est utilisable que dans la méthode qui la crée.
- Le passage d'une variable de type primitif (commence par une minuscule) est faite par copie

```
public static void main(String[] args) {  
    int i = 5;  
    int j = afficherPlus1(i, 5);  
    j = afficherPlus1(j, j-1);  
    System.out.println("i=" + i + " j=" + j);  
}  
  
public static int afficherPlus1(int i, int k) {  
    i = 1;  
    System.out.println("i=" + i + " k=" + k);  
    return i+k;  
}
```

# PORTÉ D'UNE VARIABLE

- Une variable n'est utilisable que dans la méthode qui la crée.
- Le passage d'une variable de type primitif (commence par une minuscule) est faite par copie

```
public static void main(String[] args) {  
    int i = 5;  
    int j = afficherPlus1(i, 3) + afficherPlus1(i, i+1);  
    j += afficherPlus1(i, j);  
    System.out.println("i=" + i + " j=" + j);  
}
```

```
public static int afficherPlus1(int i, int k) {  
    i = k%i;  
    System.out.println("i=" + i + " k=" + k);  
    return i*k;  
}
```

# PORTÉ D'UNE VARIABLE

- Une variable n'est utilisable que dans la méthode qui la crée.
- Le passage d'une variable de type primitif (commence par une minuscule) est faite par copie

```
public static void main(String[] args) {  
    int i = 5;  
    int j = afficherPlus1(i, 3) + afficherPlus1(i, i+1);  
    j += afficherPlus1(i, j);  
    System.out.println("i=" + i + " j=" + j);  
}
```

```
public static int afficherPlus1(int i, int k) {  
    if(i>k) {  
        i = i%k;  
    }  
    else {  
        i += k;  
    }  
    System.out.println("i=" + i + " k=" + k);  
    return i;  
}
```

# MÉTHODES DANS UNE AUTRE CLASSE

- Pour appeler une méthode qui se trouve dans une autre classe dans un autre fichier : <nomClass>.<nomMéthode>
- Fonctionne si la méthode est «public » et «static»

*//Exo.java*

```
1 public class Exo{  
2     public static void exo1() {  
3         System.out.println("exo1");  
4     }  
5 }
```

*//Main.java*

```
1 public class Main {  
2     public static void main(String[] args) {  
3         Exo.exo1();  
4     }  
5 }
```

- Ordre d'exécution des lignes :
  - 2 - 3 - 2- 3 -4 - 4

The left side of the slide features a series of vertical stripes in various shades of green and grey. Overlaid on these stripes are several green circles of different sizes. One large circle is positioned near the top left, and several smaller circles are scattered below it, some overlapping the stripes.

# CONDITIONS ET BOUCLES

54

# CONDITION ET BOUCLE

- Les diverses structures de contrôle de Java
  - Instructions « if »
  - Utilisation de « == » pour la comparaison des identités
  - Instructions « switch »
  - Boucles « while »
  - Boucles « for »

# IF – ELSE IF - ELSE

## ○ Condition sur une variable

```
1  public class Main {
2
3      public static void main(String[] args) {
4          exo5(0);
5          exo5(-1);
6          exo5(1);
7      }
8
9      public static void exo5(int monParam) {
10         if (monParam == 0) {
11             System.out.println("monParam est égal à 0");
12         }
13         //optionnelle et autant que l'on souhaite
14         else if (monParam < 0) {
15             System.out.println("monParam est négatif");
16         }
17         else {
18             System.out.println("monParam est positif");
19         }
20     }
21 }
```

## ○ Ordre d'exécution des lignes :

- 3 - 4 - 10 - 11 - 12 - 20 - 5 - 10 - 14 - 15 - 16 - 20 - 6 - 10 - 14 - 18 - 20 - 7



# OPÉRATEURS CONDITIONNELS

Opérateur	Description
<, <=, >, >=	Comparaison numérique
==, !=	Relationnel (égalité, inégalité)
!	« Non » logique
&&,	Opérateurs et/ou

## ■ Exemple

```
public void doIt( int i )  
{  
    if ( ( i < 0 ) || ! ( i < 10 ) ) x++;  
}
```

# EXERCICE

- Ecrire la méthode `int max(int a, int b, int c)` qui retourne le plus grand des 3 dans une 2eme classe(fichier) `Exo.java`
- La méthode `max` n'affiche rien dans la console.

# L'INSTRUCTION « SWITCH »

- Applicable aux types `int`, `short`, `byte`, `char`
- Autre façon de faire un `if`

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

# LA BOUCLE WHILE

- Permet d'exécuter un ensemble de lignes plusieurs fois

```
1 public class Main {  
2  
3     public static void main(String[] args) {  
4         exoWhile(5);  
5     }  
6  
7     public static void exoWhile(int max) {  
8         int i = 0;  
9         while (i < max) {    //condition pour continuer  
10             System.out.print(i + " ");  
11             i++;    //incrémente i de 1  
12         }  
13     }  
14 }
```

- Ordre d'exécution des lignes :

- 3 - 4 - 7 - 8 -(9 - 10 - 11 - 12) x5 - 9 -13 - 5

# LA BOUCLE FOR

- Permet d'exécuter un ensemble de lignes plusieurs fois
- Comme un While mais avec le i à l'intérieur

```
1 public class Main {  
2  
3     public static void main(String[] args) {  
4         exoFor(5);  
5     }  
6  
7     public static void exoFor(int max) {  
8         for (int i=0; i < max; i++) {  
9             System.out.print(i + " ");  
10        }  
11    }  
12}
```

- Ordre d'exécution des lignes :

- 3 - 4 - 7 - 8 (8 - 9 - 8) x5 - 9 - 13 - 5

7 - i=0 (i<max - 9 - i++) x 5

# EXERCICE

```
String monString = "abcd";
```

```
// Obtenir la taille d'une chaine de caractère
```

```
int taille = "abcd".length();
```

```
// Obtenir le caractère numéro 3 d'une chaine
```

```
char unCaractere = monString.charAt(3); // retourne 'd'
```

```
//Parcourir une chaine de caractère
```

```
for (int i = 0; i < monString.length(); i++) {
```

```
    //c prendra à chaque itération la valeur d'un caractère de la  
    chaine : a puis b puis c ...
```

```
    char c = monString.charAt(i);
```

```
    //TODO Faire un traitement
```

```
}
```

# EXERCICE

- Retourne la phrase en entrée sans les 'e'
- Retourne le nombre de 'a'
- Retourne la chaîne à l'envers « toto » -> « otot »
- Nombre de majuscules
- Sans les voyelles (chaîne d'entrée en minuscule uniquement)
- Retourne la chaîne sans les majuscules
- Plus grand caractère de la chaîne (chaîne d'entrée en minuscule)
- Retirer les espaces mais uniquement au début de la chaîne  
" toto en vacance " -> "toto en vacance "
- Remplacer les espaces par des \_ " toto en vacance " -> "\_toto\_en\_vacance\_"

## Pour les plus rapides :

- Retirer les espaces au début et à la fin (trim)  
" toto en vacance " -> "toto en vacance"
- Indique si une chaîne est un palindrome
- Code ascii moyen (Somme des code Ascii / nombre de caractère de la chaîne)

# EXERCICE

- A l'aide d'une boucle FOR afficher les nombres multiples de 47 entre 1 et 10000 dans la console.
- A l'aide d'une boucle While trouver le plus petit nombre respectant toutes ces conditions :
  - Divisible par 7, par 11, par 5
  - (Dont la somme avec son prédécesseur) divisé par 36 donne un reste de 1





# TABLEAU ET ALGORITHME SANS OBJET

65

# TABLEAUX (ARRAYS)

- Ils détiennent une collection de taille fixe
- Ils détiennent des objets ou des primitives
- Les tableaux en Java sont des objets, des instances d'une classe spéciale de tableaux
  - Les tableaux doivent être créés à l'aide de **new**.

# TABLEAUX : UTILISATION

- Tableau

```
//Déclaration d'un tableau de primitive ou d'objet
float[] monTab;

//Instanciatioin d'un tableau de taille 10. Taille fixe obligatoire
//Le « new » correspondant à la création d'objet qu'on verra plus tard.
//Un tableau est un objet.
monTab = new float[10];

//initialise et remplit le tableau qui sera du coup de taille 6
monTab = new float[]{3,5,6,4,2,8};

//Ecrire dans un tableau
monTab[0] = 3;

//Lire dans un tableau. Attention les indices commencent à 0
float i = monTab[0];

//Taille du tableau
int size = monTab.length;

public static int[] exoTableau(int max) {
    int[] monTab = new int[max]; //Création d'un tableau d'entier de taille 'max'
    for (int i = 0; i < monTab.length; i++) {
        monTab[i] = new Random().nextInt(100); //Chiffre Aléatoire de 0 à 99
    }
    return monTab;
}
```

# EXERCICE

- Créer une classe ArrayUtils qui contiendra les exo sur les tableaux.
- Implémentez les méthodes suivantes
  - Le tableau étant un objet, il ne sera pas copié.
  - Méthode à appeler depuis la méthode main de la classe Main

```
/**Remplis le tableau de valeurs aléatoires comprises entre 0 et99 */  
public static void fillTab(int[] tab) { }
```

```
/** Affiche le tableau dans la console sur 1 seule ligne */  
public static void printTab(int[] tab) { }
```

```
/** Retourne la valeur maximum du tableau */  
public static int getMax(int[] tab) { }
```

```
/** Permute l'emplacement i et j dans le tableau */  
public static void permute(int[] tab, int i, int j) { }
```

- Retourne la somme
- Retourne la moyenne
- Affiche les valeurs supérieurs à la moyenne (ne retourne rien)
- Retourne le nombre d'occurrence de la valeur maximum (Version  $o(2n)$  et  $O(n)$ )

```
/** Crée et retourne un nouveau tableau qui est la concaténation des  
2 tableaux [1,5,7] [3,4,8,9] -> [1,5,7,3,4,8,9] */  
public static int[] fusion(int[] tab1, int[] tab2) { }
```



# EXERCICE $O(N^2)$

- Créer une classe ArrayUtils qui contiendra les exo sur les tableaux.
- Implémentez les méthodes suivantes
  - Le tableau étant un objet, il ne sera pas copié.
  - Methode à appeler depuis la méthode main de la classe Main

```
/** Trie le tableau */
```

```
public static void sortTab(int[] tab) {}
```

```
/** Crée et retourne un tableau trié qui est la fusion des 2 tableaux triés  $O(n)$  */
```

```
public static int[] fusionTrier(int[] sortTab1, int[] sortTab2) {}
```

# EXERCICE PERFORMANCE

- Calculer le temps que cela prend pour trier 100 fois un tableau de taille 10 000 avec la méthode `ArraysUtils.sortTab()`.
  - Ne pas recréer le tableau entre chaque trie, simplement y remettre des valeurs aléatoires.
  - `long debut = System.currentTimeMillis();`//Pour calculer le temps
- Faire de même avec la méthode de Java : `Arrays.sort()`
  - Si c'est plus long que votre méthode vous pouvez postuler chez Google !!

# ENUMERATION

- Permet de créer un ensemble fini de valeur

```
/** Déclaration de l'ensemble fini*/
```

```
public enum Couleur {BLEU, BLANC, ROUGE}
```

```
public static void exo6() {
```

```
/** Création d'un tableau avec cet ensemble*/
```

```
Couleur[] monTab = new Couleur[10];
```

```
/** Utilisation*/
```

```
monTab[0] = Couleur.BLANC;
```

```
switch(monTab[0]) {
```

```
    case BLEU:
```

```
        System.out.println("La couleur est bleu");
```

```
        break;
```

```
    case BLANC:
```

```
        System.out.println("La couleur est blanche");
```

```
        break;
```

```
    case ROUGE:
```

```
        System.out.println("La couleur est rouge");
```

```
        break;
```

```
}
```

## EXERCICE : LE DRAPEAU FRANÇAIS

- Créez une énumération contenant 3 valeurs : Bleu, Blanc, Rouge
- Créer un tableau de taille 15 contenant aléatoirement ces 3 valeurs.
- Triez le tableau de manière à mettre les bleus en premier puis les blancs puis les rouges.
- Améliorez votre algorithme pour ne le faire qu'en 1 passage du tableau (Une seule boucle for).





## PASSER À UN LANGAGE OBJET

73

# LA CLASSE STRING

- La classe String (commence par une majuscule) permet de gérer une chaîne de caractères

```
String prenom = "bob";
```

- On peut y appliquer un ensemble de méthodes définies dans la classe String
  - <variable> '.' <nomMethode>

```
int size = prenom.length(); //Retour la taille : '3'  
char firstLetter = prenom.charAt(0); //retourne la 1er lettre : 'b'  
//compare en ignorant les majuscule : true  
prenom.equalsIgnoreCase("BoB");
```

- Ces méthodes ne sont pas « static » car elles s'appliquent sur un objet en particulier, ici « prenom »

# DÉFINIR SA PROPRE CLASSE

- Il est possible en Java de définir ses propres classes ou chaque instance sera différente.
  - Main, Exo, ArraysUtils...
- On peut y définir des attributs qui seront différents pour chaque instance

```
//Chaque maison sera définie par une largeur, une longueur et une couleur  
public class MaisonBean {  
    public int largeur = 20; //20 sera la valeur par défaut  
    public int longueur = 10;  
    public String couleur;  
}
```

# UTILISER SA CLASSE

- Pour créer une instance (ou une maison)

```
MaisonBean maison1 = new MaisonBean();
```

- Pour utiliser les attribut <nomVariable>.<nomAttribut>

```
maison1.largeur = 3;
```

- Exemple avec 2 maisons :

```
/** (Classe Main) */  
public static void main(String[] args) {  
    MaisonBean maison1 = new MaisonBean();  
    maison1.largeur = 3;  
    maison1.couleur = "Bleu";  
    System.out.println("La maison1 est " + maison1.couleur + " de largeur " + maison1.largeur);  
  
    MaisonBean maison2 = new MaisonBean();  
    maison2.largeur = 5;  
    maison2.couleur = "Rouge";  
    System.out.println("La maison2 est " + maison2.couleur + " de largeur " + maison2.largeur);  
}
```

# POINTEUR

- Un pointeur mais qu'est ce que c'est ?



`MaisonBean maison1` = `new MaisonBean();`

- Un pointeur ce n'est rien d'autre que l'adresse où se situe la maison.

# COPIE DE POINTEUR

- Copier un pointeur, ne revient qu'à copier l'adresse, on ne duplique pas la maison
- On ne crée pas de nouvelle maison.

```
MaisonBean maison1 = new MaisonBean();  
MaisonBean maison2 = maison1;
```



maison1

maison2

@2a43c1

- Un pointeur ce n'est rien d'autre que l'adresse où se situe la maison.

```
maison1.largueur == maison2.largueur  
maison1 == maison2
```

# MULTIPLE MAISON

- Chaque « new » correspond à une nouvelle maison

```
MaisonBean maison1 = new MaisonBean();
```

```
MaisonBean maison2 = new MaisonBean();
```

```
MaisonBean maison3 = maison1;
```



maison1



maison2



maison3



@2a43c1



@45cd3

# POINTEUR NULL

- Un pointeur null est un pointeur qui ne contient plus ou pas d'adresse

```
MaisonBean maison1 = null;
```



maison1

- Attention au `NullPointerException`



# EXERCICE

- Créer la classe MaisonBean avec les 2 attributs longueur et largeur.
- Créer une classe ExoMaisonUtils et implémenter les méthodes suivantes :

```
/** Affiche la largeur, longueur de la maison :
```

```
12x25 */
```

```
public static void printMaison(MaisonBean m) {}
```

```
/** Double la taille de la maison*/
```

```
public static void doubleMaison(MaisonBean m) { }
```

```
/** Retourne la maison la plus grande, et null si de la même  
taille*/
```

```
public static MaisonBean bigMaison(MaisonBean m1, MaisonBean  
m2) {}
```

# EXERCICE

- Créer la classe EleveBean avec les 2 attributs note et nom.
- Créer une classe ExoEleve et implémenter les méthodes suivantes :

```
/** Affiche le nom et la note de l'élève */
```

```
public static void printEleve(EleveBean e) {}
```

```
/** Double la note de l'élève */
```

```
public static void doubleNote(EleveBean e) { }
```

```
/** Retourne l'élève qui a la meilleur note ou nulle en cas  
d'égalité */
```

```
public static EleveBean meilleurEleve(EleveBean e1, EleveBean  
e2) {}
```

# TABLEAU D'OBJET

- Comme on peut faire un tableau de String, on peut faire un tableau de maison

```
MaisonBean [] monTab = new MaisonBean [5];
```

- Mais au final, qu'est ce que j'ai crée ? Des maisons? Des pointeurs?
- Réponse :



- Pour le remplir

```
monTab[0] = new MaisonBean ();
```

# EXERCICE

- Créer la classe MaisonBean avec les 2 attributs longueur et largeur.
- Créer une classe ExoMaisonUtils et implémenter les méthodes suivantes :

```
/** Remplit le tableau de maison avec des largeurs et longueurs aléatoires */
```

```
public static void createMaisons(MaisonBean[] tab) {}
```

```
/** Affiche les maisons avec leur taille. 1 maison par ligne */
```

```
public static void printMaisons(MaisonBean[] tab) { }
```

```
/** Retourne la maison la plus grande (Longueur * largeur) */
```

```
public static MaisonBean bigMaison(MaisonBean[] tab){}
```

# EXERCICE

- Créer la classe EleveBean avec les 2 attributs note et prenom.
- Créer une classe ExoEleve et implémenter les méthodes suivantes :

```
/** Remplit le tableau d'élève avec des noms et une note aléatoire */
```

```
public static void createEleves(EleveBean[] tab) {}
```

```
/** Affiche les élèves avec leur note*/
```

```
public static void printEleves(EleveBean[] tab) { }
```

```
/** Retourne l'élève s'appelant bob ayant la meilleurs note*/
```

```
public static EleveBean bestBob(EleveBean[] tab){}
```

```
/** Retourne un prénom aléatoire */
```

```
public static String getRandomName(){
```

```
    String[] name = new String[] {"Toto", "Tata", "Titi", "Bob", "Alfred"};
```

```
    return name[new Random().nextInt(name.length)];
```

```
}
```

# DES MÉTHODES PROPRE À UN OBJET

- Si on souhaite la surface d'une maison

```
int surface = maison.largeur * maison.longueur;
```

- On va pouvoir ajouter une méthode à la classe maison qui effectue le travail, pour éviter la duplication de code.

```
public int getSurface() {  
    return largeur * longueur;  
}
```

- Comme la méthode doit s'appliquer à une maison, elle n'est pas « static »

```
MaisonBean maison = new MaisonBean();  
maison.largeur = 12;  
maison.longueur = 15;  
int surface = maison.getSurface();
```

# VOCABULAIRE

- MaisonBean c'est la classe
- Un pointeur ou une variable maison :  
`MaisonBean maison;`
- Je crée une « **instance** » de la classe MaisonBean  
`MaisonBean maison = new MaisonBean();`
- J'appelle un attribut de la classe MaisonBean sur l'instance maison  
`int size = maison.longueur;`
- J'appelle une méthode de la classe MaisonBean sur l'instance Maison  
`int surface = maison.getSurface();`

## ENCAPSULATION : VISIBILITÉ D'UN ATTRIBUT OU D'UNE MÉTHODE

- **public** : Visible partout
  - **private** : Visible uniquement dans le fichier
  - Sans rien : Visible uniquement dans le même package
  - **protected**: Visible uniquement pour les classes héritantes (On verra plus loin ce que c'est)
- 
- Certaines méthodes ou attributs n'ont d'intérêt qu'à être utilisé dans la classe, on va donc les déclarer private.



# CONVENTION JAVA

- Par convention en Java les attributs sont déclarés **private**. Pour les lire ou les modifier on utilisera des méthodes appelées Getter et Setter

```
public class MaisonBean {  
    private int largeur;  
  
    public int getLargeur() {  
        return largeur;  
    }  
  
    public void setLargeur(int largeur) {  
        this.largeur = largeur;  
    }  
}
```

- Le mot clé **this** représente l'instance courante.
  - maison.largeur équivalent à **this**.largeur
  - Permet de différencier la variable **largeur** de la Classe Maison et le paramètre **largeur** de la méthode **setLargeur**.

# EXERCICE

- Passer en private les attributs de la classe MaisonBean et faire les corrections qui vont bien

# EXERCICE

- Créez une classe `DeBean.java` avec un attribut `private` de type `int` nommé « valeur » ainsi que son Getter et son Setter
- Ajoutez une méthode `lancer` à la classe `DeBean` changeant la valeur du dé avec une valeur aléatoire comprise entre 1 et 6.
  - `Lancer ne retourne pas de valeur`
  - « Random value java » sur Google pour créer une valeur aléatoire.
- Dans la méthode `main` de la classe `Main`, créez un `De` et affichez sa valeur dans la console avant et après un lancé.

# LES CONSTRUCTEURS

- A l'instanciation d'un Objet on appelle une méthode qui porte le même nom que l'objet

```
MaisonBean maison = new MaisonBean();
```

- Cette méthode s'appelle un **Constructeur**. Elle ne peut être appelée qu'à l'instanciation de l'objet (new).
- Dans la classe MaisonBean on peut redéfinir le constructeur, c'est une méthode sans valeur de retour.

```
public class MaisonBean {  
    private int largeur;  
  
    public MaisonBean() {  
        largeur = 0;  
    }  
  
    public int getLargeur() {  
        return largeur;  
    }  
}
```

# LES CONSTRUCTEURS

- A l'instanciation d'un Objet on appelle une méthode qui porte le même nom que l'objet

```
1 public class MaisonBean {  
2     private int largeur;  
3  
4     public MaisonBean() {  
5         largeur = 0;  
6     }  
7  
8     public int getLargeur() {  
9         return largeur;  
10    }  
11}
```

```
1 public static void main(String[] args) {  
2     MaisonBean maison = new MaisonBean();  
3     System.out.print("Largeur : " + maison.getLargeur());  
4 }
```

- Ordre d'exécution des lignes :

- 1 - 2 - 4 - 5 - 6 - 2 - 3 - 8 - 9 - 10 - 3

# LES CONSTRUCTEURS

- Il est possible d'ajouter des paramètres à un constructeur

```
public class MaisonBean {  
    private int largeur;  
  
    public MaisonBean(int largeur) {  
        this.largeur = largeur;  
    }  
  
    public int getLargeur() {  
        return largeur;  
    }  
}
```

- Dans ce cas, l'instanciation demande de remplir ce paramètre.

```
public static void main(String[] args) {  
    MaisonBean maison = new MaisonBean(10);  
    System.out.print("Largeur : " + maison.getLargeur());  
}
```

- Et le constructeur sans paramètre ?

- Si aucun constructeur n'existe il prend par défaut un constructeur sans paramètre qui ne fait rien, sinon il prend le votre.

# SIGNATURE DE MÉTHODE

- La signature d'une méthode est définie par
  - Sa visibilité (public, private..)
  - Static ou non
  - Sa valeur de retour (void, int, String...)
  - Son nom
  - Le nombre et le type de paramètre
  - Si elle retourne une exception ou non

# POLYMORPHISME

- Définition : Le polymorphisme veut dire que la même méthode, peut avoir un comportement différent selon les situations.
- Reprenons le cas de notre méthode `lancer()` de la classe `De`
- Il est possible d'en ajouter une autre de même nom mais **avec des paramètres différents : Surcharge**

```
1 public class DeBean {
2
3     public void lancer() {
4         value = new Random().nextInt(6) + 1;
5     }
6
7     public void lancer(int maxValue) {
8         value = new Random().nextInt(maxValue) + 1;
9     }
10    ...
11 }
12
13 public static void main(String[] args) {
14     DeBean de = new DeBean ();
15     de.lancer();
16     de.lancer(20);
17 }
```

- Ordre d'exécution des lignes :

• 1 - 2 - 3 - 3 - 4 - 5 - 4 - 7 - 8 - 9 - 5



# POLYMORPHISME CONSTRUCTEUR

- Un constructeur peut en appeler un autre en utilisant

`this(...);`

```
1 public static class MaisonBean {
2     private int largeur, longueur;
3
4     public MaisonBean() {
5         this(10, 20);
6     }
7
8     public MaisonBean(int largeur, int longueur) {
9         this.largeur = largeur;
10        this.longueur = longueur;
11    }
12}
```

```
1 public static void main(String[] args) {
2     MaisonBean maison1 = new MaisonBean(50, 60);
3     MaisonBean maison2 = new MaisonBean();
4 }
```

- Ordre d'exécution des lignes :

• 1 - 2 - 8 - 9 - 10-11 - 2 - 3 - 4 - 5 - 8 - 9 - 10-11 - 6 - 3 - 4

# RÉSUMÉ SYNTAXES JAVA

```
//Classe
public class EleveBean {
    //Attribut
    private String nom;
    //constructeur
    public EleveBean(String nom) {
        this.nom = nom;
    }
    //methode
    private void doSomething(){
        ...
    }

    //getter
    public String getNom() {
        return nom;
    }
    //setter
    public void setNom(String nom) {
        this.nom = nom;
    }
}
```

# EXERCICE

- Ajoutez 2 constructeurs à la classe DeBean.
  - Un prenant la taille du dé par défaut en paramètre.
  - Un ne prenant pas de paramètre et donnant une taille de 6. (Pour un dé à 6 faces)
- Faire en sorte qu'un constructeur appelle l'autre constructeur.
- Testez ces méthodes dans la classe Main.

# POINTEUR JAVA

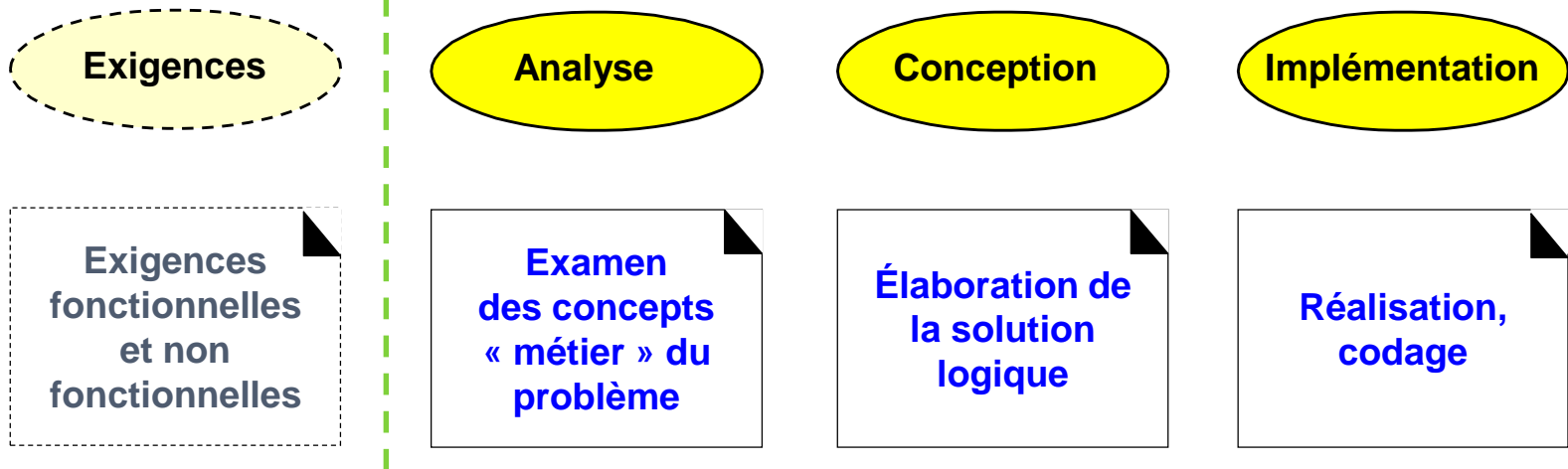
```
private void genocide() {  
  
    EleveBean eleve = null;  
    //NullPointerException  
    eleve.getNom();  
  
    eleve = new EleveBean("Bob"); //Allocation Mémoire pour créer Bob  
    eleve.setNom("John"); //Bob préfère qu'on l'appelle John  
  
    //On réassigne le pointeur, plus personne ne pointe sur l'espace mémoire de John  
    //il va être « garbage collecté ». Adieu John  
    eleve = new EleveBean("Candy");  
  
    //Nous avons 2 pointeurs sur Candy  
    EleveBean eleve2 = eleve;  
    eleve.setNom("John");  
    System.out.println("e=" + eleve.getNom() + " e2=" + eleve2.getNom());  
    //Nous n'avons plus qu'un pointeur sur Candy  
    eleve = null;  
  
    //Plus personne ne pointe sur Candy. Prépare toi à être recycler  
    eleve2 = null;  
}
```



# PENSER OBJET

101

# ORIENTATION OBJET



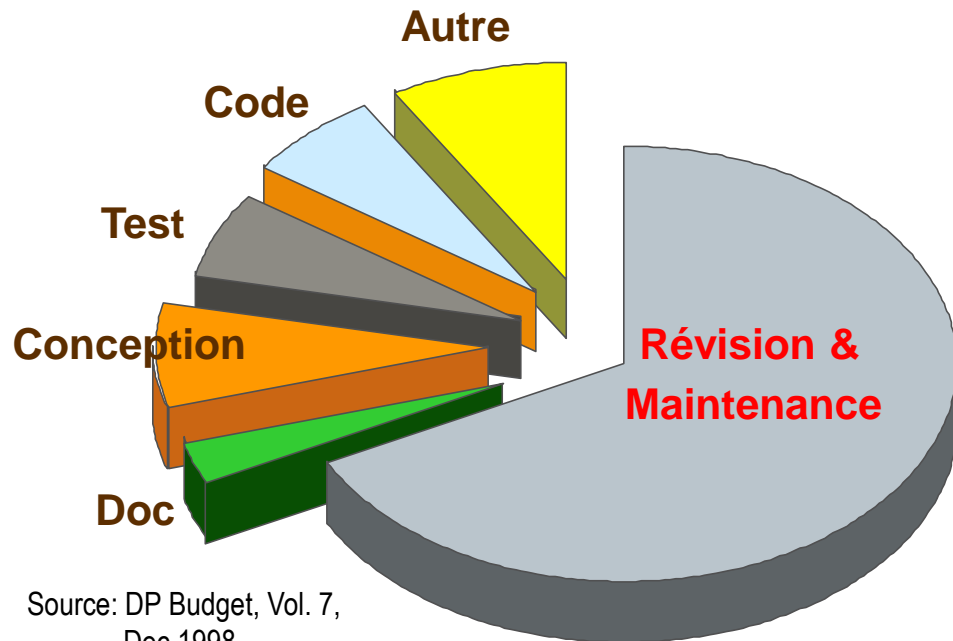
- Analyse orientée objet
  - Quels sont les objets du monde réel ?
- Conception orientée objet
  - Les objets logiciels ? Leurs responsabilités ?
- Programmation orientée objet
  - Codage dans un langage de programmation orienté objet

# POURQUOI LA TECHNOLOGIE OBJET ?

## ■ Parce que les systèmes objet sont :

- *Faciles à maintenir*
- *Extensibles*
- *Flexibles*

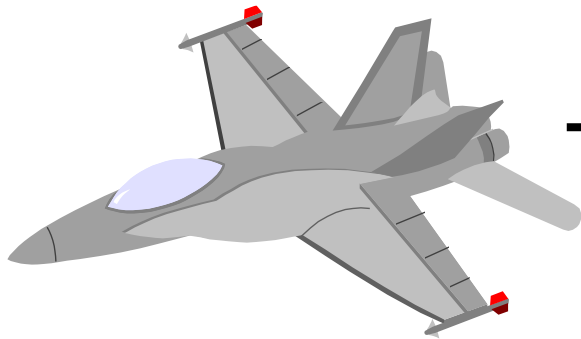
## ■ Coût des projets logiciels



Source: DP Budget, Vol. 7,  
Dec 1998.

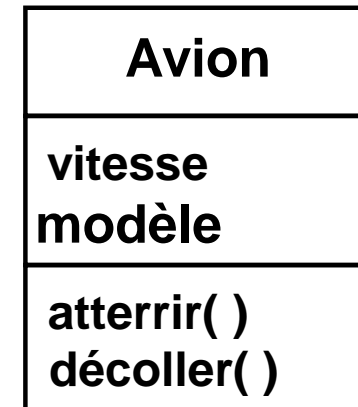
# LE MONDE RÉEL ET LE PAYS DES OBJETS

## Monde réel



abstraction

## Monde logiciel



Un objet associe des données (attributs) et du code (comportement) au sein d'une même unité

```
class Avion
```

```
{
```

```
    private Velocite vitesse;  
    private String modele;
```

```
    public void atterrir( ){...}
```

```
    public void decoller( ){...}
```

```
    //...
```

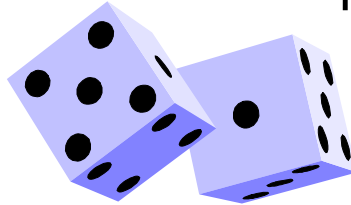
```
}
```

implémentation



# NOTRE PROJET

- Nous allons mettre au point la représentation orientée objet d'un jeu de dés simple
- Règles :
  - Chaque joueur lance dix fois deux dés
  - Le joueur tirant le plus souvent sept ou plus a gagné



- Exercice : identifiez les objets du monde réel liés au jeu de dés

# CLASSE ET INSTANCE DANS LE LOGICIEL

## ○ Une classe :

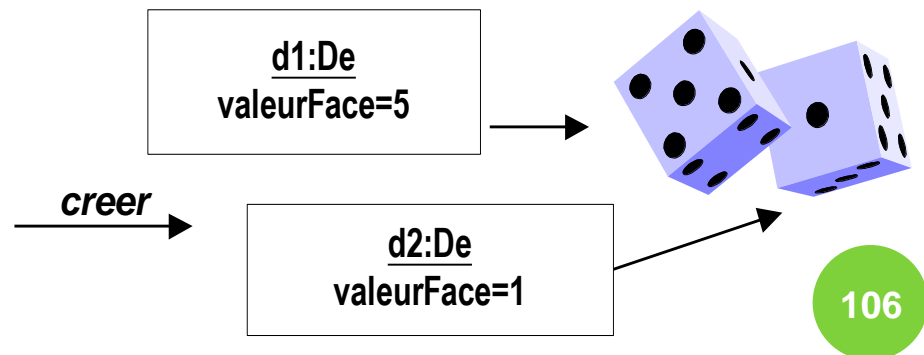
- Définit – la classe définit les caractéristiques et le comportement d'une famille d'objets (ses instances)
- Crée – c'est une usine à créer des objets logiciels d'un certain type



De
valeurFace
lancer () getValeurFace()

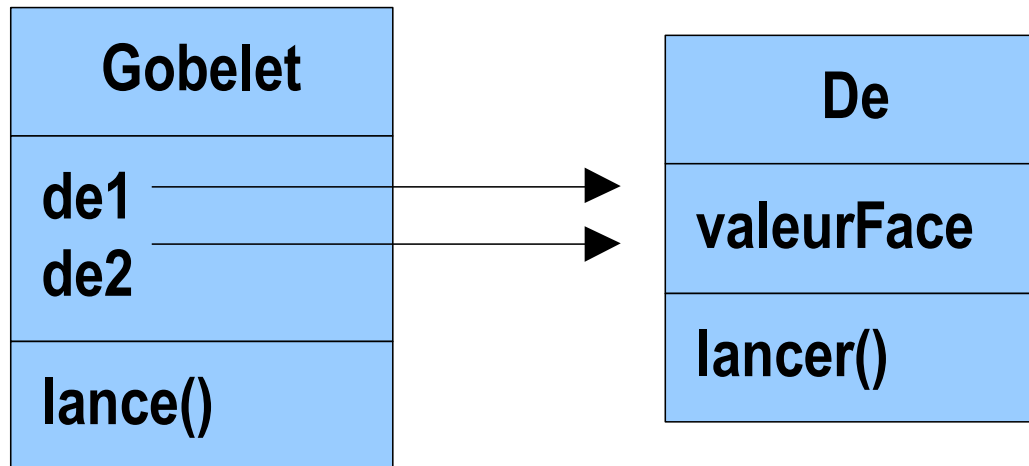
## ○ Une instance :

- Est instanciée (créée) par une classe
- Occupe de l'espace dans la mémoire d'un ordinateur
- Conserve les valeurs des attributs
- A un comportement
- Connaît sa classe



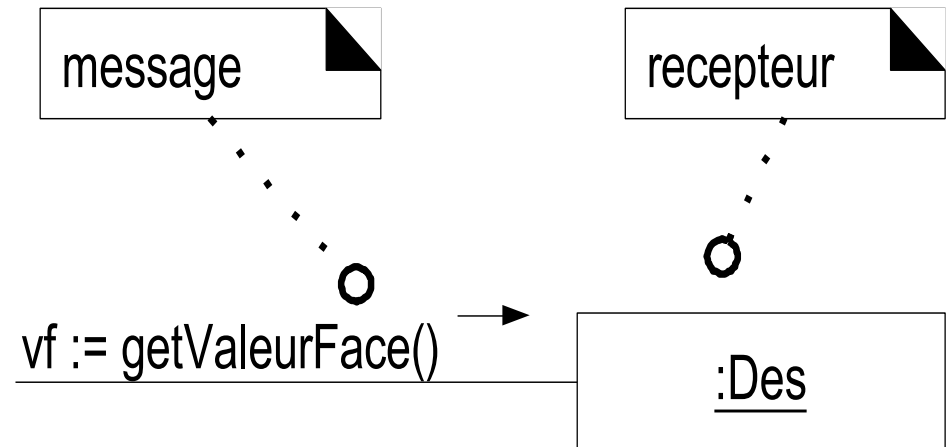
# CONTENANCE

- Des objets peuvent en contenir d'autres
- Les objets sont liés les uns aux autres



# COLLABORATIONS ET MESSAGES

- La collaboration s'effectue par l'envoi de messages



- Message

- Signal envoyé à un objet (récepteur) pour l'invocation d'une méthode
- Peut retourner une valeur
- Semblable à un appel de fonction, mais dirigé vers un objet

# ATTRIBUTS

- Les attributs décrivent les informations dont doit se souvenir un objet

- Exemples :

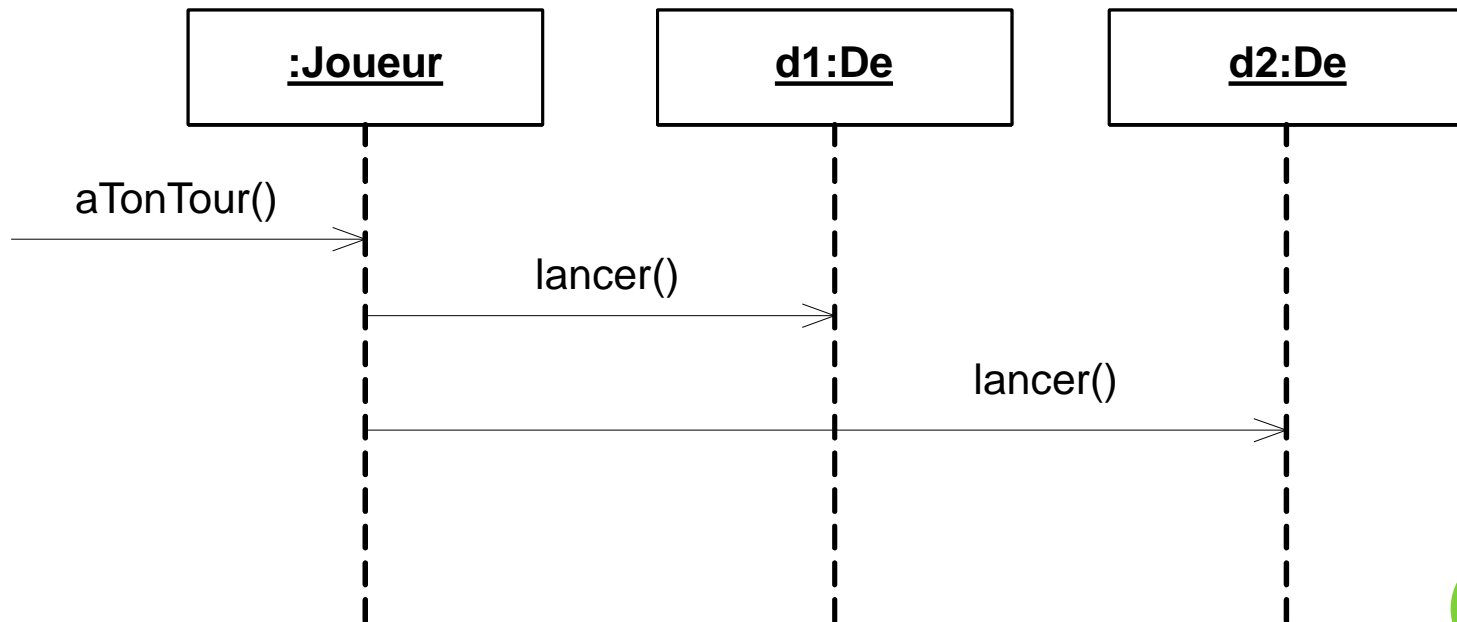
- Le *Pilote* a un nom
- Un *Avion* a une vitesse et un numéro de modèle

Avion
vitesse modèle

- Exercice : identifiez les attributs de chaque objet du jeu de dés

# DIAGRAMME DE SÉQUENCE

- Le diagramme de séquence est exactement équivalent au diagramme de communication
  - Le temps s'écoule de haut en bas
  - Met l'accent sur l'ordre chronologique des messages



# EXERCICE

- Concevoir et créer le jeu.
- Règles :
  - Chaque joueur lance dix fois deux dés
  - Le joueur tirant le plus souvent sept ou plus a gagné
- Pour les plus rapides, sans changer le model:
  - Chaque joueur lance 2 dés jusqu'à arriver à faire un score de 7, 3 fois de suite.
  - Le joueur ayant fait le moins de tentative gagne.
  - Si l'architecture est bien conçue, DeBean, GobeletBean, JoueurBean et PartieBean n'ont pas besoin d'être modifiés.



# INTERFACE GRAPHIQUE SWING

112



# PRÉSENTATION

- L'API Java2 comprend les packages Swing indépendants de toute plate-forme pour la création d'interfaces utilisateur graphiques (IHM)
- Objectifs :
  - Explorer la hiérarchie des composants
  - Créer une fenêtre simple à l'aide d'un package Swing
  - Examiner les divers composants
  - Modifier les attributs des conteneurs et des composants

# LES CLASSES SWING

- AWT = lourde (ou native). Dessin effectué par des classes homologues natives
  - Panel
  - Frame
- Swing = légère. Dessin effectué par des classes « pur Java »
  - JPanel
  - JFrame
- Se trouvent dans les packages javax.swing.\*
  - javax.swing.
  - javax.swing.event.
  - ...

# SWING ET AWT

- L'AWT figure toujours dans Java2
- L'architecture Swing est fondée sur le framework interne de l'AWT
- Certaines classes Swing référencent des classes AWT
- De nombreuses classes « utilitaires », telles que Font et Color, ont été laissées dans l'AWT
- Utilisez, autant que possible, les classes Swing plutôt que les classes de l'AWT
- Évitez de mélanger les classes Swing et celles de l'AWT

# CONTENEURS ET COMPOSANTS

## ○ Les conteneurs de niveau supérieur

- sont autonomes et peuvent contenir n'importe quel autre composant
  - Exemples : JFrame, JApplet, JDialog et JWindow

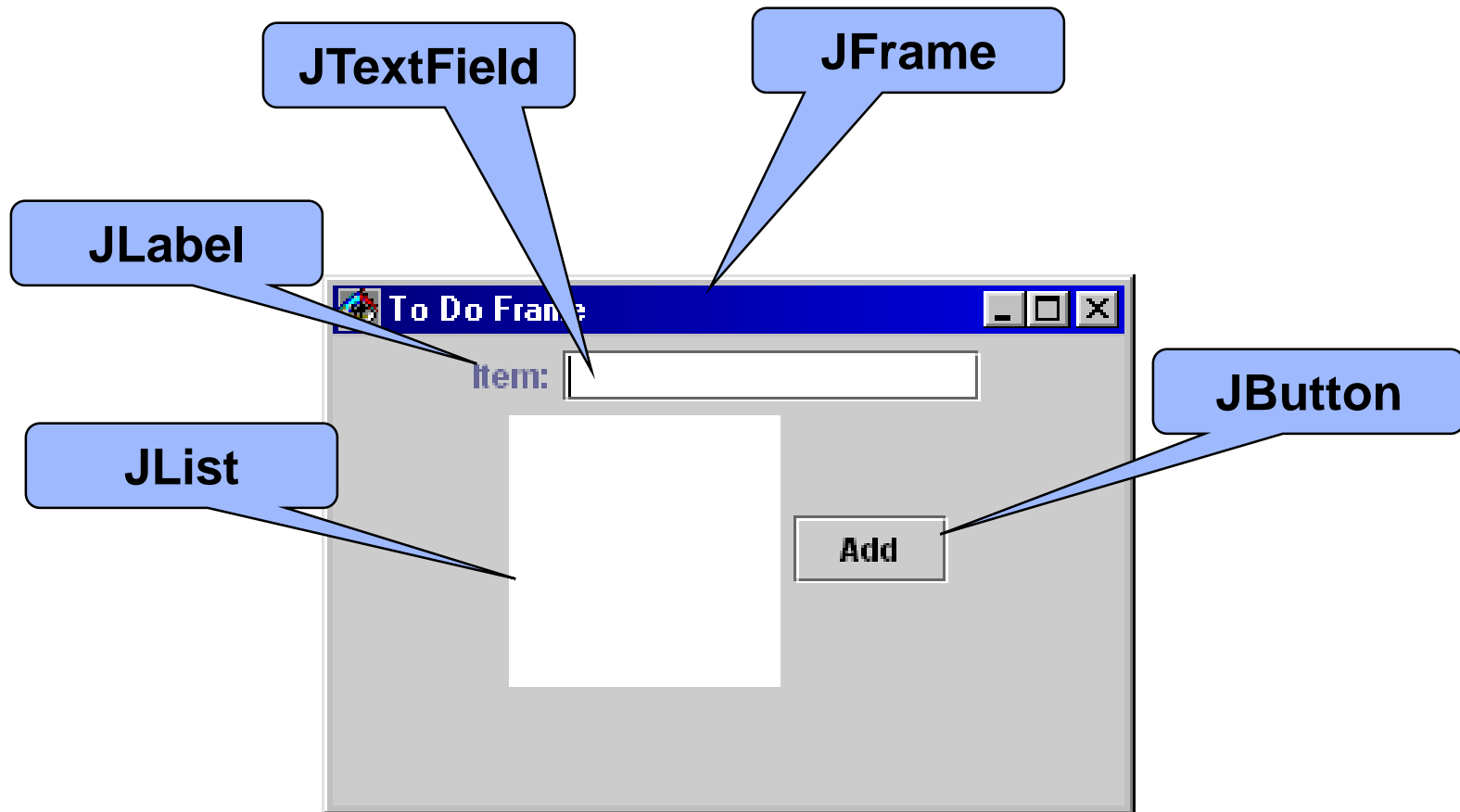
## ○ Les conteneurs intermédiaires

- peuvent contenir d'autres conteneurs intermédiaires ou des composants atomiques
  - Exemples : JPanel, JScrollPane,...

## ○ Les composants atomiques

- sont des composants graphiques (ou « widgets ») qui ne contiennent pas d'autre composant
  - Exemples : JButton, JTextField,...

## EXEMPLE D'IHM SWING : « TO-DO LIST »



# TODOFRAME - DÉFINITION DE CLASSE

```
public class ToDoFrame extends JFrame {  
    private JLabel itemLabel;  
    private JTextField itemField;  
    private JList itemList;  
    private JButton addButton;  
    private JPanel contentPane;  
}
```

- Déclarez tous les composants graphiques (« widgets ») en tant que champs d'instances du conteneur
- Ils sont initialisés dans le constructeur...

# CONSTRUCTEUR DE TODOFRAME

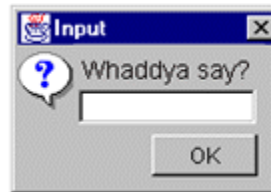
```
public ToDoFrame() {  
    super("To Do Frame");  
    //Instanciation des composants  
    itemLabel = new JLabel("Item:");  
    itemField = new JTextField(14);  
    itemList = new JList(model);  
    itemList.setPreferredSize(new java.awt.Dimension(100, 100));  
    addButton = new JButton("Add");  
    contentPane = new JPanel();  
    //Ajout des composants  
    contentPane.add(itemLabel);  
    contentPane.add(itemField);  
    contentPane.add(itemList);  
    contentPane.add(addButton);  
    setContentPane(contentPane);  
  
    this.setSize(300, 200);  
}
```

# AUTRES COMPOSANTS SWING :

## CONTENEURS DE HAUT NIVEAU

- JDialog

- Boîte de dialogue modale ou non modale



- JApplet

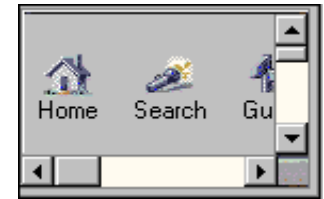
- Conteneur de widgets affiché et exécuté dans un navigateur



# AUTRES COMPOSANTS SWING : CONTENEURS INTERMÉDIAIRES

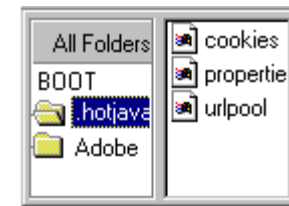
## ○ JScrollPane

- Fournit une « vue » pouvant défiler sur les données d'un JViewport



## ○ JSplitPane

- Affiche deux composants côte à côte



## ○ JTabbedPane

- Affiche la vue par onglets de panneaux multiples



## ○ JToolBar

- Conteneur de boutons ancrable



# AUTRES COMPOSANTS SWING : COMMANDES DE BASE

## ○ Boutons

- JButton, JToggleButton, JCheckbox et JRadioButton



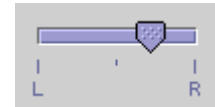
## ○ Menus

- JMenu, JMenuBar, JPopupMenu, JMenuItem et JSeparator



## ○ JSlider

- Sélectionne une valeur à l'aide d'un curseur glissant



## AUTRES COMPOSANTS SWING : AFFICHAGES NON ÉDITABLES D'INFORMATIONS

### ○ JToolTip

- Permet d'afficher un « message d'information » pour n'importe quel JComponent



### ○ JProgressBar

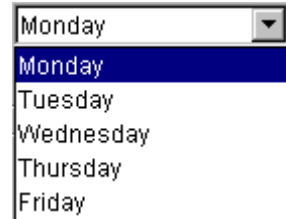
- Affiche le degré de réalisation d'un événement sous forme de barre graphique



## AUTRES COMPOSANTS SWING : AFFICHAGES NON ÉDITABLES D'INFORMATIONS

### ○ JComboBox

- Association de TextField et de liste déroulante
  - Peut également servir de liste déroulante normale



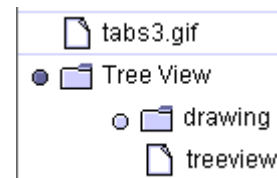
### ○ JTable

- Tableau bi-dimensionnel de données modifiables

First Na...	Last Name
Mark	Andrews
Tom	Ball
Alan	Chung
Jeff	Dinkins

### ○ JTree

- Vue arborescente de données hiérarchiques
  - Permet de refermer/déployer des nœuds individuels

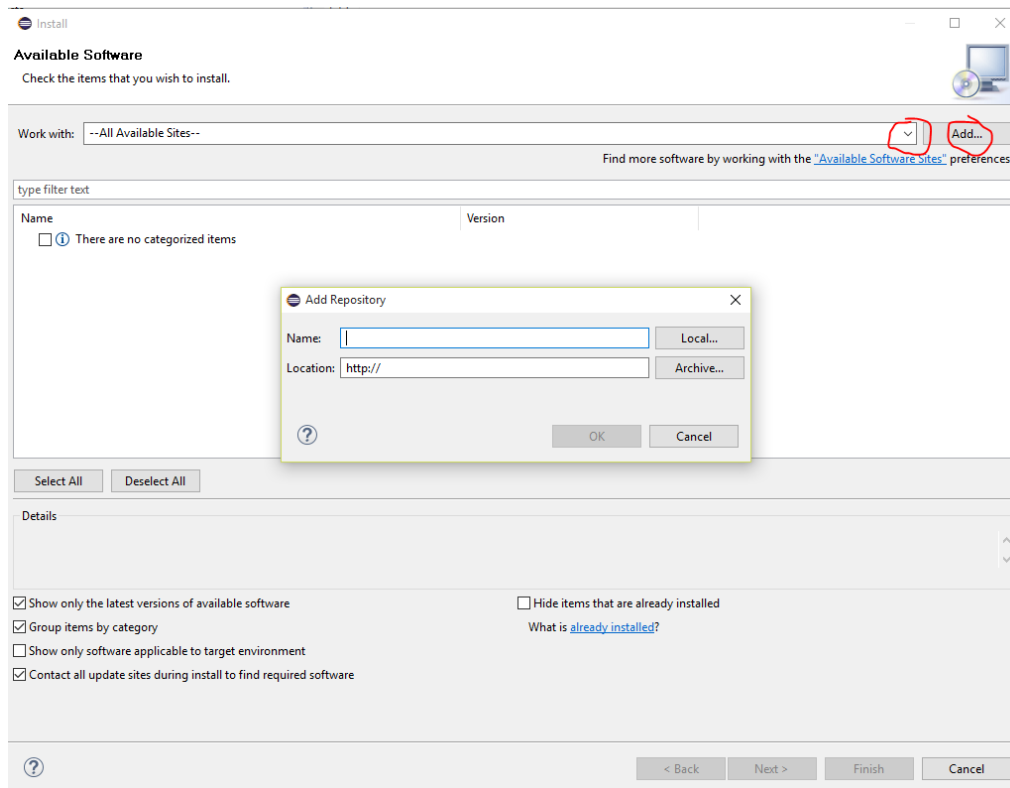


## PLUGIN ECLIPSE : WINDOWBUILDER ENGINE

- Windowbuilder engine sur Google 1<sup>er</sup> lien
  - Menu Download
    - Clic droit sur link et copier le lien

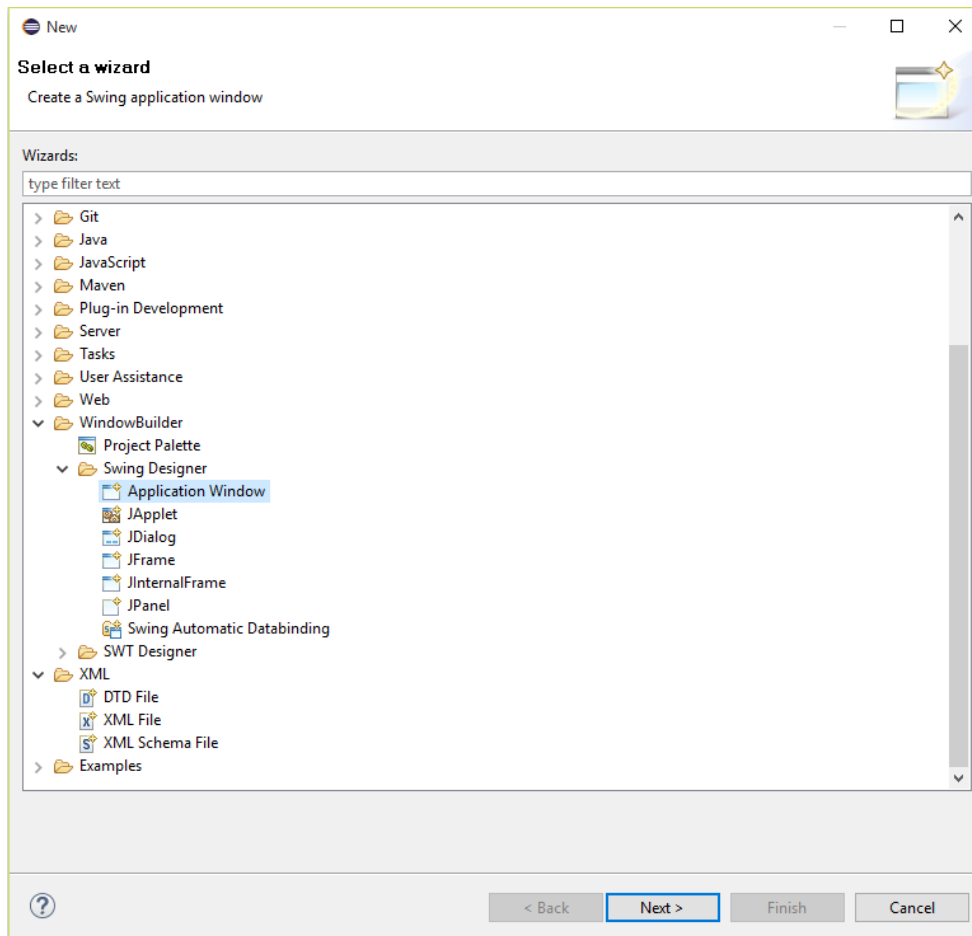
# PLUGIN ECLIPSE : WINDOWBUILDER ENGINE

- Ajouter un plugin à Eclipse
  - Help > Install new Software...




# PLUGIN ECLIPSE : WINDOWBUILDER ENGINE

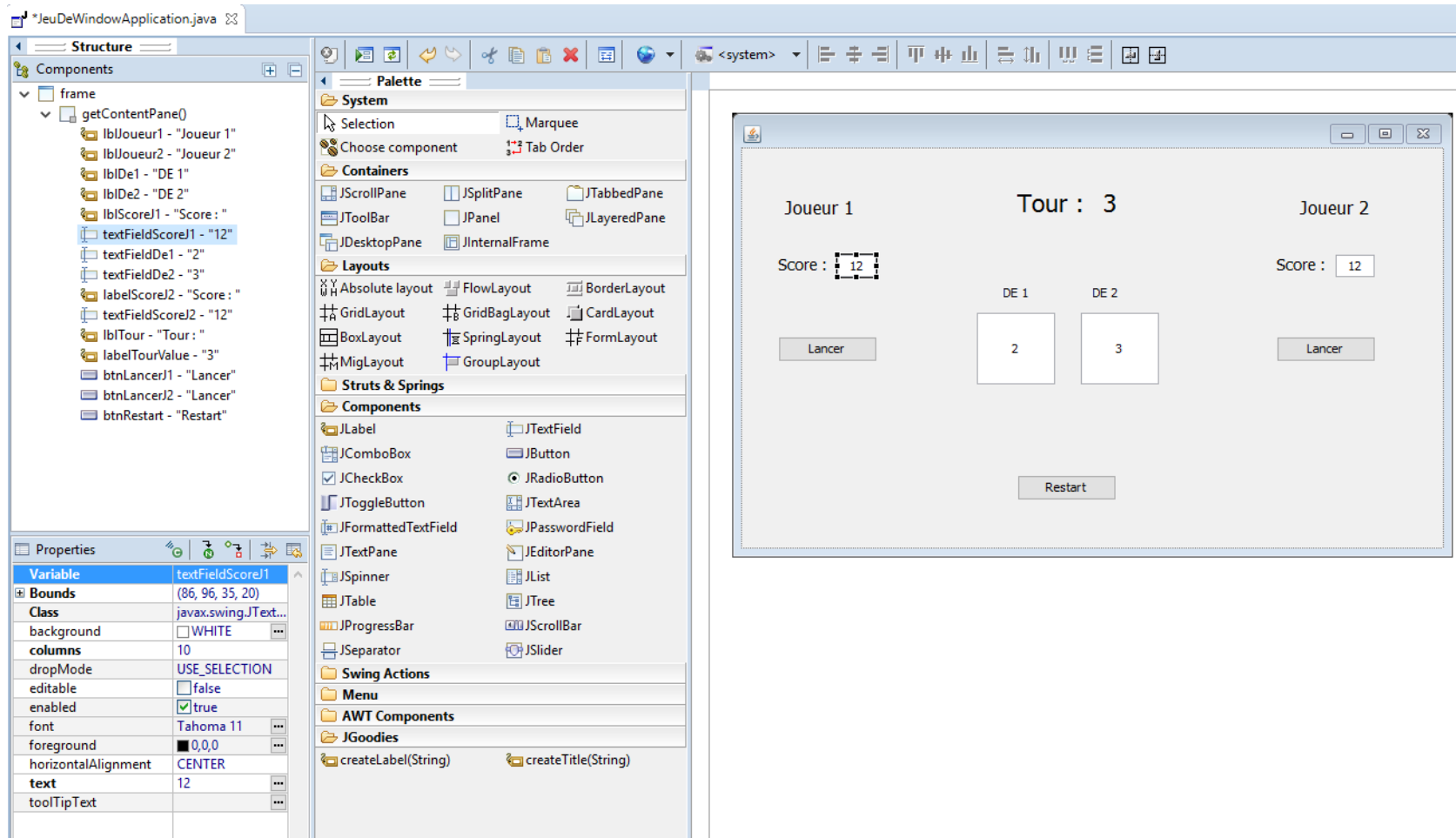
## ○ Créer une nouvelle application Window



```
29      ^ Create the application.
30      */
31     public JeuDeWindowApplication() {
32         initialize();
33     }
34
35     /**
36      * Initialize the contents of the fram
37      */
38     private void initialize() {
39         frame = new JFrame();
40         frame.setBounds(100, 100, 450, 300
41         frame.setDefaultCloseOperation(JFr
42     }
43
44 }
45
```



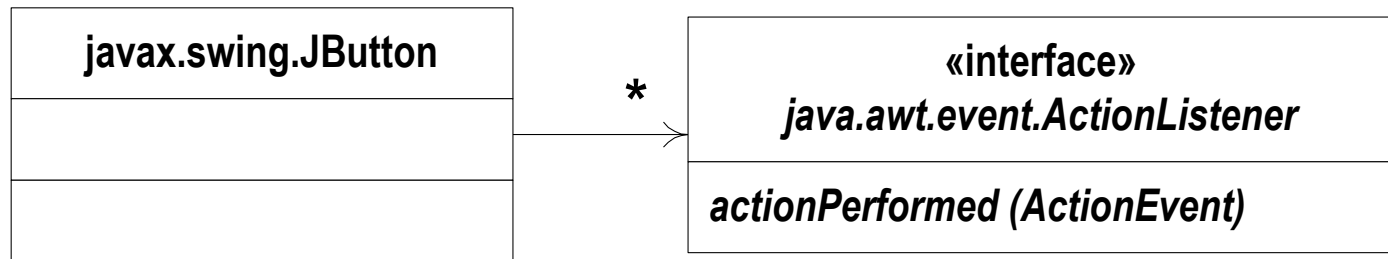
# EXERCICE : INTERFACE GRAPHIQUE JEU





# LES ÉVÉNEMENTS

- Intercepter le clic sur un bouton
- Ils entretiennent une collection d'objets listeners pour des événements spécifiques



- Ils génèrent des événements
  - Exemple : lorsqu'on appuie dessus, un *JButton* génère un *ActionEvent*
- Ils notifient tous les listeners lorsque se produit un événement



User

# S'ABONNER A UN BOUTON

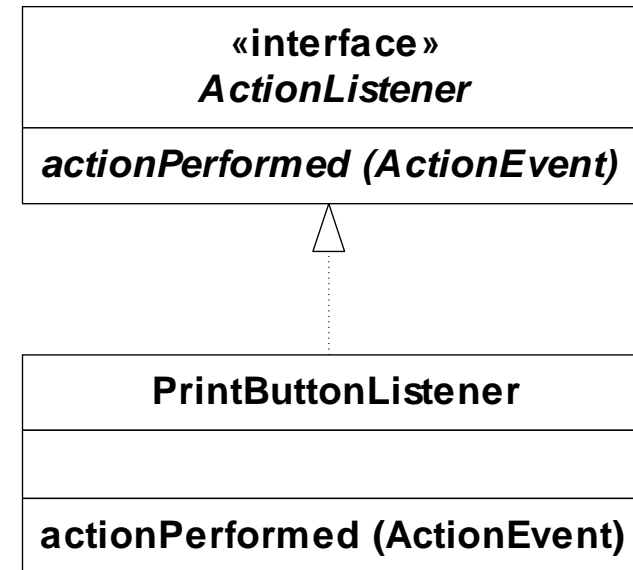
- Les objets sources définissent les méthodes permettant d'ajouter (et de supprimer) des listeners

```
aButton.addActionListener (ActionListener al) ;  
aButton.removeActionListener (ActionListener al) ;
```

javax.swing.Button
<b>addActionListener (ActionListener)</b> <b>removeActionListener (ActionListener)</b>

# IMPLÉMENTATION D'UN LISTENER

- Un objet listener est n'importe quel objet dont la classe implémente une interface de réception



```
public class MyClass implements ActionListener {
    @Override
    public void actionPerformed(ActionEvent arg0) {
        System.out.println ("Button pressed...");
    }
}
```

```
//JeuDeWindowApplication
```

```
public void setLancerJ1ActionListener(ActionListener actionListener) {
    btnLancerJ1.addActionListener(actionListener);
}
```

# EXERCICE

- Relier l'interface graphique à votre jeu!!
- Analysons ça tous ensemble!!!

# HÉRITAGE

- Imaginons une classe `Personne` avec un nom et un prénom
- Si on veut créer un `Elève`, comme une `Personne` mais avec des choses en plus. (Sa section, son emploi du temps...)
- Pour éviter la duplication de code on va pouvoir créer une extension de la classe `PersonneBean` avec le mot clé **`extends`**

```
public class EleveBean extends PersonneBean {  
  
}
```

- Un élève est donc un Objet de type `EleveBean` et un Objet de type `PersonneBean`.
  - On peut lui ajouter des nouveaux attributs
  - Des nouvelles méthodes privées et publiques.
  - Des nouveaux constructeurs
    - Attention les Constructeurs de `EleveBean` doivent appeler un constructeur de `PersonneBean`

# HÉRITAGE : UTILISATION DE LA CLASSE MÈRE

- Pour accéder aux attributs de la classe mère ceux-ci doivent avoir été déclarés avec la visibilité **protected** ou **public**
- L'équivalent de **this** pour la classe mère est **super**

```
1 public class PersonneBean {
2     protected String nom;
3     public PersonneBean(String nom) {
4         this.nom = nom;
5     }
6 }
1 public class EleveBean extends PersonneBean {
2     protected String section;
3     public EleveBean(String nom, String section) {
4         super(nom); //Appel du constructeur de Personne
5         this.section = section;
6     }
7 }
1 public static void main(String[] args) {
2     EleveBean e = new EleveBean("Bob", "Math"); //ok
3     PersonneBean p = new PersonneBean("Bob2"); //ok
4     PersonneBean p2 = new EleveBean("Bob3", "Math"); //Ok Eleve est aussi de type
personne
5     EleveBean im1 = new PersonneBean("Bob4"); // KO, la classe personne n'est pas de
type Eleve.
}
```

# HÉRITAGE : REDÉFINITION

- Une extension peut redéfinir une méthode de sa classe mère
  - Elle doit conserver la même signature

```
1 public class PersonneBean {
2     protected String nom;
3     public void print() {
4         System.out.println("Je m'appelle " + nom);
5     }
6 }
```

```
1 public class EleveBean extends PersonneBean {
2     protected String section;
3
4     @Override
5     public void print() {
6         //Si je souhaite appeler la classe de la méthode mère
7         super.print();
8         System.out.println(" et je suis en classe de " + section);
9     }
10 }
```

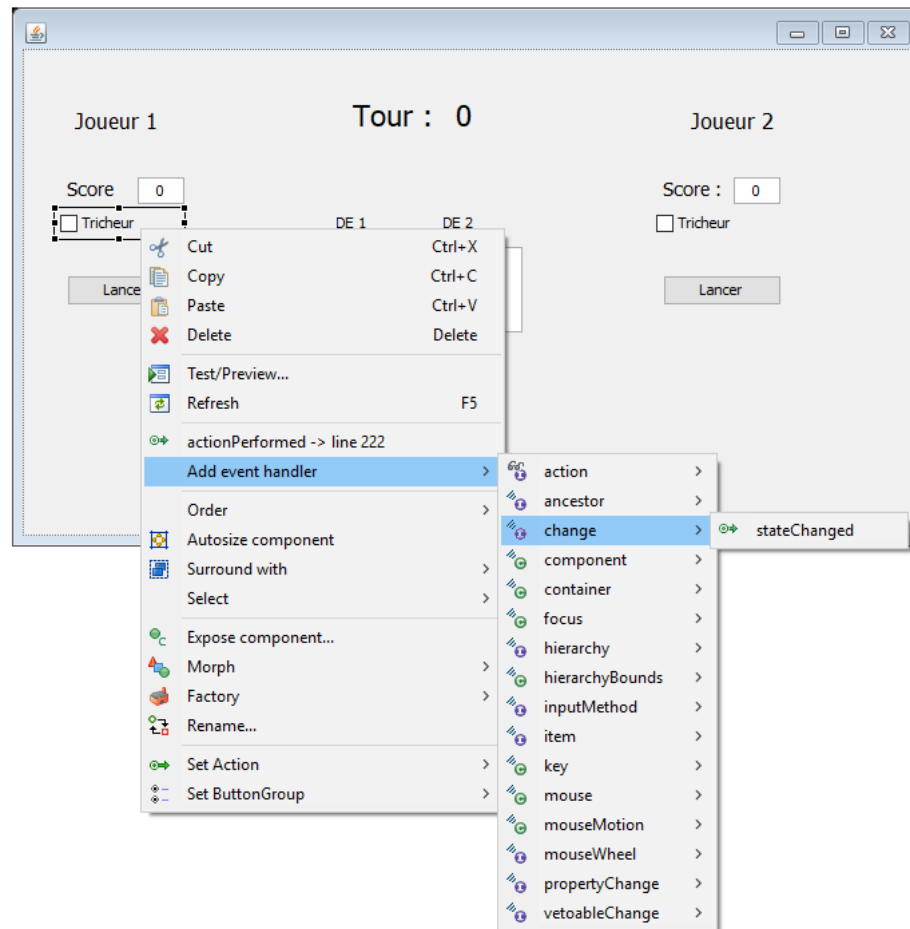
```
1 public static void main(String[] args) {
2     PersonneBean m1 = new EleveBean("Toto", "Math");
3     m1.print();
4     PersonneBean m2 = new PersonneBean("Bob");
5     m2.print();
6 }
```

- Ordre d'exécution des lignes :

- 1 - 2 - 3 - 5 - 7 - 3 - 4 - 5 - 7 - 9 - 4 - 5 - 3 - 4 - 5 - 6

# EXERCICE

- Intercepter le changement de checkbox





# EXERCICE

- Ajoutez la notion de dé pipé au jeu, un dé pipé aura ayant 58% de chance de faire un 6 lors de son appel de la méthode « lancer ».
- Ajoutez un constructeur à GobeletBean permettant d'indiquer que c'est un gobelet truqué contenant 1 dé pipé.
- Ajoutez un attribut tricheur et son setter à JoueurBean indiquant que c'est un tricheur et qu'il utilise un gobelet contenant un dé pipé.

# TERMINOLOGIE (1/3)

- **Attribut** — information ou état associé à un objet
- **Classe** — définition des caractéristiques et du comportement d'une famille d'objets. « Usine d'instanciation d'objets ».
- **Classe abstraite** — super-classe incomplète définissant les parties communes. Non instanciée.
- **Classe concrète** — classe complète. Classe décrivant un concept de façon complète. Classe destinée à être instanciée.
- **Agrégation (contenance)** — construction d'un composant à partir d'autres composants. Agrégat: objet contenant d'autres objets.

## TERMINOLOGIE (2/3)

- **Héritage** — dans les sous-classes, acquisition automatique des définitions d'attributs et de comportement de la super-classe
- **Instance** — objet créé par une classe
- **Instanciation** — opération de création d'une instance
- **Message** — requête envoyée à un objet pour invoquer une méthode. Un message s'apparente à un appel de fonction, mais il s'en distingue en ceci qu'un objet ne répond qu'à un ensemble prédéterminé de messages définis par sa classe
- **Méthode** — implémentation d'un message. L'envoi d'un message invoque une méthode

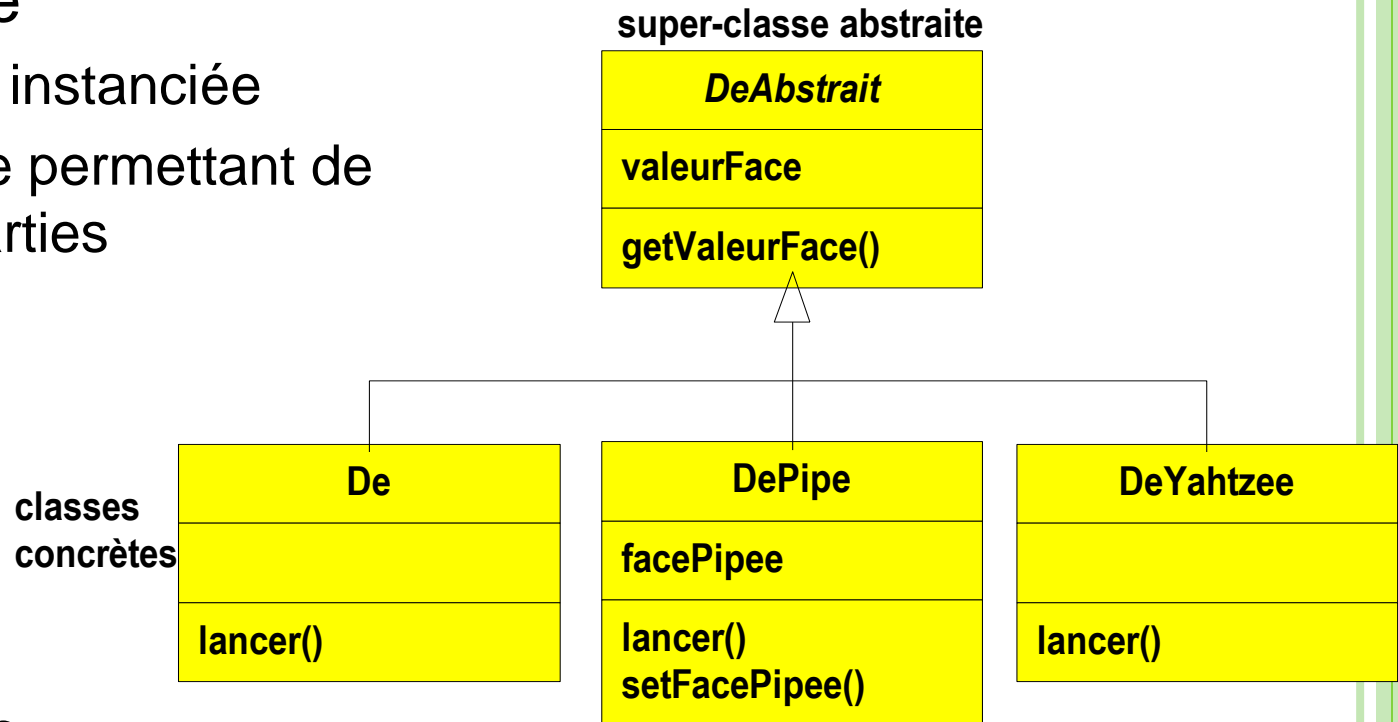
# TERMINOLOGIE (3/3)

- **Objet** — composant logiciel. Instance d'une classe. Structure regroupant les attributs et le comportement
- **Polymorphisme** — objets de différentes classes répondant au même message. La réponse de chaque objet risque de varier en fonction de sa classe
- **Sous-classe** — classe définie en termes de super-classe à l'aide de l'héritage
- **Spécialisation** — acte de définition d'une classe comme raffinement d'une autre classe
- **Super-classe** — classe servant de base à l'héritage dans une hiérarchie

# CLASSES ABSTRAITES ET CONCRÈTES

## Classe abstraite

- Ne peut être instanciée
- Super-classe permettant de définir les parties communes

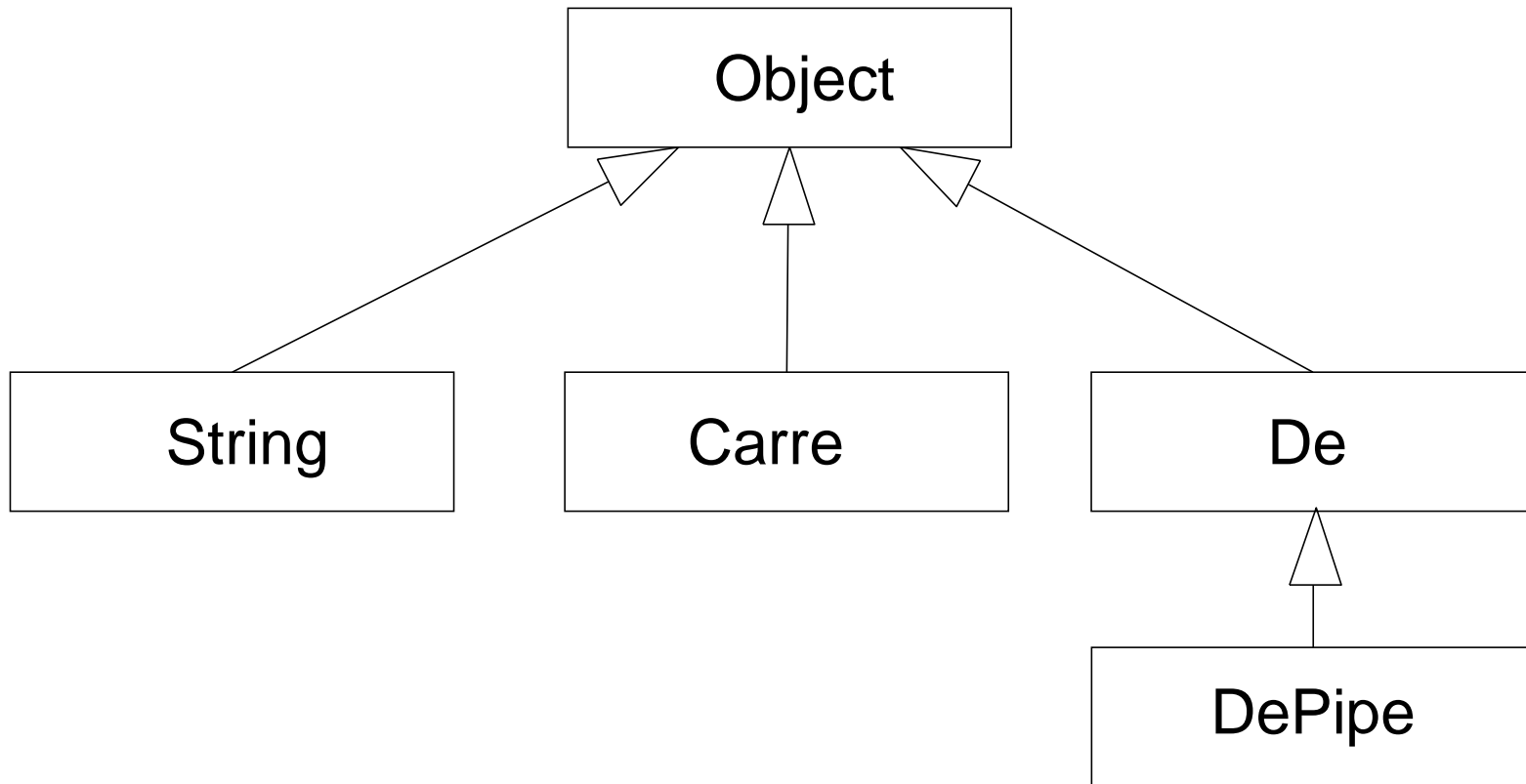


## Classe concrète

- Peut être instanciée

# PRÉSENTATION

- *Object* est une classe
- Chaque classe en hérite directement ou indirectement



# MÉTHODES DE LA CLASSE OBJECT

- La classe *Object* définit des méthodes

```
public class Object
{
    public boolean equals( Object obj ) {...}
    public String  toString( ) {...}
    // clip...
}
```

- Elle leur fournit une implémentation par défaut qui peut être redéfinie par les sous-classes

# ToString( )

- *toString( )* permet de fournir une représentation sous forme de chaîne de l'objet
  - Appelée implicitement par *System.out.println()* et « + »

```
System.out.println( obj );  
String s = "The object is " + obj;
```

- Par défaut, elle retourne le nom de la classe et la référence de l'instance
- Redéfinissez-la avec une implémentation adaptée !

```
public class De  
{  
    public String toString( )  
    {  
        return "De avec valeurFace = " + getValeurFace( );  
    }  
}
```

144



## EQUALS( )

- Retourne vrai si les deux objets ont des valeurs égales
  - par défaut **equals** compare les références d'instances à l'aide de **==**
- À redéfinir par chaque classe pour un comportement approprié

```
public class De
{
    public boolean equals( Object obj )
    {
        if ( obj instanceof De )
        {
            De other = (De) obj;
            return this.getValeurFace( ) == other.getValeurFace( );
        }
        return false;
    }
}
```

# UTILISATION EQUALS()

- Utilisez *equals()* pour effectuer une comparaison de valeurs au lieu d'une comparaison d'identités

```
De de1 = new De(3);  
De de2 = new De(3);  
  
if ( de1 == de2 ) {...}           // toujours faux  
  
if ( de1.equals(de2) ) {...}      // renvoie "true"
```

# RÉSUMÉ

- La classe `Object` est la super-classe de toutes les autres classes
- Plusieurs méthodes sont héritées de la classe `Object`
  - `toString()` : permet d'obtenir une représentation sous forme de chaîne d'un objet
  - `equals()` : à redéfinir pour fournir un test d'égalité de valeur
  - et bien d'autres...

# EXERCICE

- Ajouter un bouton « Afficher les données » qui affiche la console l'ensemble des données.

Juste en faisant « `System.out.println(partieBean);` »

# INTERFACE

- Le problème d'avoir plusieurs parents
  - Maison, Pavillon, Immeuble, Cabane en bois, Bunker, Tente, Camping Car, Caravane
  - Jardin, Piscine, Mobile
- Solution les interfaces

```
public interface Mobile {  
    void deplacer(Location location);  
}
```

```
public interface Jardin {  
    void tondre();  
}
```

# INTERFACE

```
public class Caravane extends Maison
    implements Mobile, Jardin
{

    @Override
    public void tondre() {
    }

    @Override
    public void deplacer(Location Location) {

    }

}
```

# INTERFACE COMPARABLE

- Permet de comparer les Objets entre eux afin de pouvoir les classer (Existe déjà).

```
public interface Comparable<T> {  
    public int compareTo(T arg0);  
}
```

```
public class De implements Comparable<De> {  
  
    @Override  
    public int compareTo(De arg0) {  
        return this.value - arg0.value;  
    }  
}
```

# EXERCICE

- Rendre le GobeletBean comparable, afin de pouvoir le comparer avec un autre gobelet.

```
if(j1.getGobelet().compareTo(j2.getGobelet()) > 0) {  
  
}
```





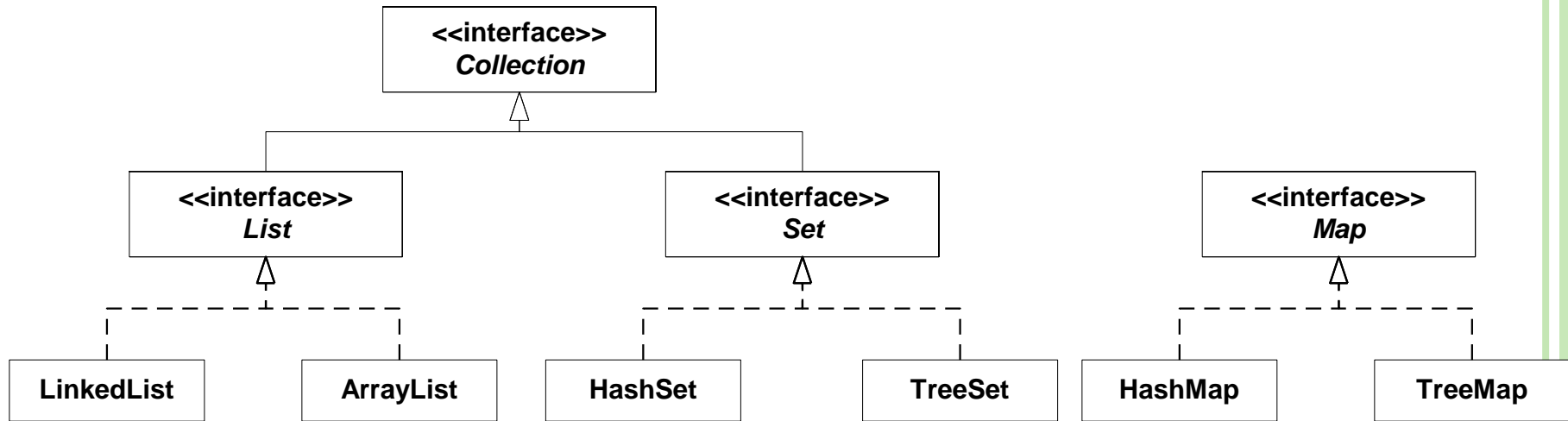
# COLLECTIONS ET ALGORITHMES

153

# COLLECTIONS

- Bien qu'il soit possible de créer des tableaux d'objets en Java, ce n'est pas une pratique courante
- On utilise plutôt des collections (semblables à des tableaux) pour stocker des groupes d'objets
- Java 2 propose un cadre de collections très sophistiqué
- Trois grands types de collections existent :
  - Les listes
  - Les maps
  - Les ensembles (sets)

# TYPES DE COLLECTION JAVA 2



- Situés dans le package java.util
- Quelle différence y a-t-il entre les types List, Set et Map ?
  - Les types « list » sont ordonnés et permettent l'indexation des éléments
  - Les types « set » ne permettent pas la duplication des éléments
  - Les types « map » associent les éléments avec des valeurs clés

# OPÉRATIONS DES COLLECTIONS

## List

- Insertion avec add()
- Récupération avec get()
- Clé par entier

## et aussi

- remove(Object)
- size()
- clear()
- contains(Object)
- isEmpty()
- toArray()
- ...

## Map

- Insertion avec put()
- Récupération avec get()
- Clé d'objet arbitraire

## et aussi

- remove(Object)
- size()
- clear()
- containsKey(Object)
- containsValue(Object)
- isEmpty()
- keySet()
- values()...

# COLLECTIONS GÉNÉRIQUES

```
ArrayList<Type> , HashMap<TypeKey, TypeValue>, ...
```

Où Type est à remplacer  
par le type opportun

- Ce sont exactement les mêmes que précédemment, mais on spécifie un type à la création de la collection

```
List<String> myList = new ArrayList<String>();
```

- La collection ne peut contenir QUE le type spécifié

# PARCOURIR UNE LISTE

```
//ArrayList
ArrayList<EleveBean> eleveArrayList = new ArrayList<>();
EleveBean eleve = new EleveBean();
//Ajout
eleveArrayList.add(eleve);

//Parcours de liste (ForEach)
for (EleveBean e : eleveArrayList) {
    e.setNote(e.getNote() + 1);
}

//Parcours de liste (For classique)
for(int i=0; i<eleveArrayList.size(); i++) {
    EleveBean e = eleveArrayList.get(i);
    e.setNom(e.getNote() + 1);
}
```

# EXERCICE

- Créer une classe CollectionUtils qui contiendra les exo sur les listes.
- Implémentez les méthodes suivantes
  - La liste étant un objet, il ne sera pas copié.
  - Méthode à appeler depuis la méthode main de la classe Main

```
/**Remplis la liste d'eleve leur donnant des noms et note aléatoire*/  
public static void fillList(ArrayList<EleveBean> eleveArrayList, int nbEleve)  
{ }
```

```
/** Affiche la liste dans la console */  
public static void printList(ArrayList<EleveBean> eleveArrayList) { }
```

```
/** Affiche et retourne le 1er élève avec la meilleur note*/  
public static EleveBean getMax(ArrayList<EleveBean> eleveArrayList) { }
```

```
/** Retire les élèves n'ayant pas la moyenne de la classe */  
public static void supInfMoyenne (ArrayList<EleveBean> eleveArrayList) { }
```

```
/** Retourne un prénom aléatoire */  
public static String getRandomName() {  
    String[] name = new String[] { "Toto", "Tata", "Titi", "Bob", "Alfred" };  
    return name[new Random().nextInt(name.length)];  
}
```

# HASHMAP

- HashMap

```
//Déclaration
HashMap<Integer, EleveBean> mesEleves;

//Instanciación de la HashMap
mesEleves = new HashMap<Integer, EleveBean>();

//Ajout / remplace
mesEleves.put(3, new EleveBean ());

//Récupération
EleveBean eleve = mesEleves.get(3);

//Taille de la HashMap
int size = mesEleves.size();

//Parcours
for (Map.Entry<Integer, EleveBean> unEleve : mesEleves.entrySet()) {
    Integer cle = unEleve.getKey();
    EleveBean temp = unEleve.getValue();
}
```



# EXERCICE

- Créez 50 élèves avec un prénom commençant par un chiffre(0-9) aléatoire.
- Les positionner dans une `HashMap<Integer, ArrayList<EleveBean>>` en fonction de leur chiffre de départ.
- Afficher dans la console la `HashMap` sous cette forme :  
    0 : 0Eleve32 0Eleve35 0Eleve42  
    1 : 1Eleve3 1Eleve33 1Eleve43  
    ...

# GESTION DES EXCEPTIONS

## ○ Définition

- Une exception est un signal qui indique que quelque chose d'exceptionnel (comme une erreur) s'est produit. Elle interrompt le flot d'exécution normal du programme.

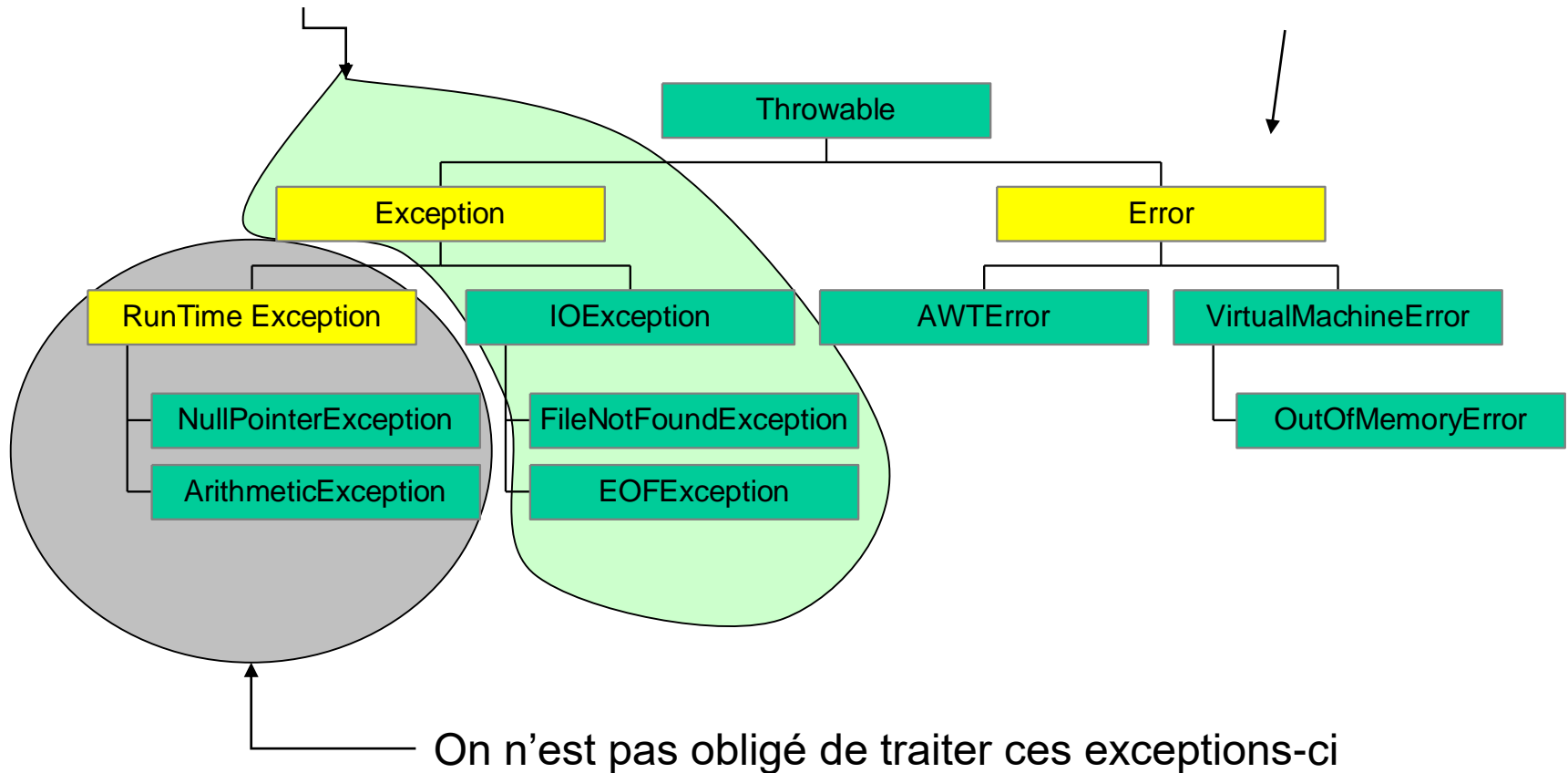
## ○ A quoi ça sert

- Gérer les erreurs est indispensable : Ariane 5
- Mécanisme simple et lisible
  - Regroupement du code de gestion de l'erreur
  - Possibilité de transmettre l'erreur.

# HIÉRARCHIE DES EXCEPTIONS

On doit traiter ces exceptions-ci

On ne peut pas traiter  
les « Error »



# GESTION DES EXCEPTIONS

- Créer sa propre classe d'exception

```
public class MyException extends Exception {  
    public MyException(final String errorMessage) {  
        super(errorMessage);  
    }  
}
```

- Déclencher une exception

```
if (number > 10) {  
    throw new Exception("Bad number : " + number);  
}
```

# GESTION DES EXCEPTIONS

- Propager une exception

```
public void test(final int number) throws Exception {  
    if (number > 10) {  
        throw new Exception("Bad number : " + number);  
    }  
}
```

- Traiter une exception

```
public void test2() {  
    try {  
        test(8);  
        test(12);  
        test(7);  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
}
```

# GESTION DES EXCEPTIONS

```
public void test2() throws Exception {  
    try {  
        test(5);  
    }  
    catch (MyException e) {  
        e.printStackTrace();  
    }  
    catch (Exception e) {  
        throw e;  
    }  
    finally {  
        // Toujours faire ceci, quelle que soit l'exception  
    }  
}
```

# GESTION DES EXCEPTIONS

- Propager la stacktrace

```
public void test(int number) throws Exception {  
    try {  
        if (number > 10) {  
            throw new Exception("Bad number : " + number);  
        }  
    }  
    catch (final Exception e) {  
        //On perd le message contenu dans e  
        throw new Exception("Invalid test");  
    }  
}
```

# GESTION DES EXCEPTIONS

- Créer sa propre classe d'exception

```
public class MyException extends Exception {  
    public MyException(final String errorMessage) {  
        super(errorMessage);  
    }  
  
    //un 2eme constructeur pour transmettre la StackTrace  
    public MyException(String message, Throwable e){  
        super(message, e);  
    }  
}
```



# GESTION DES EXCEPTIONS

- Propager la stacktrace

```
public void test(final int number) throws Exception {  
    try {  
        if (number > 10) {  
            throw new MyException("Bad number : " + number);  
        }  
    }  
    catch (final Exception e) {  
        //On conserve le message contenu dans e  
        throw new MyException("Invalid test", e);  
    }  
}
```

# EXERCICE

- Créez votre classe InvalidFaceNumberException
- Faire en sorte que le constructeur de De retourne cette exception si la valeur en paramètre du constructeur est incorrecte.
- Faire en sorte que la classe qui traite cette exception, appelle le constructeur vide dans le catch.

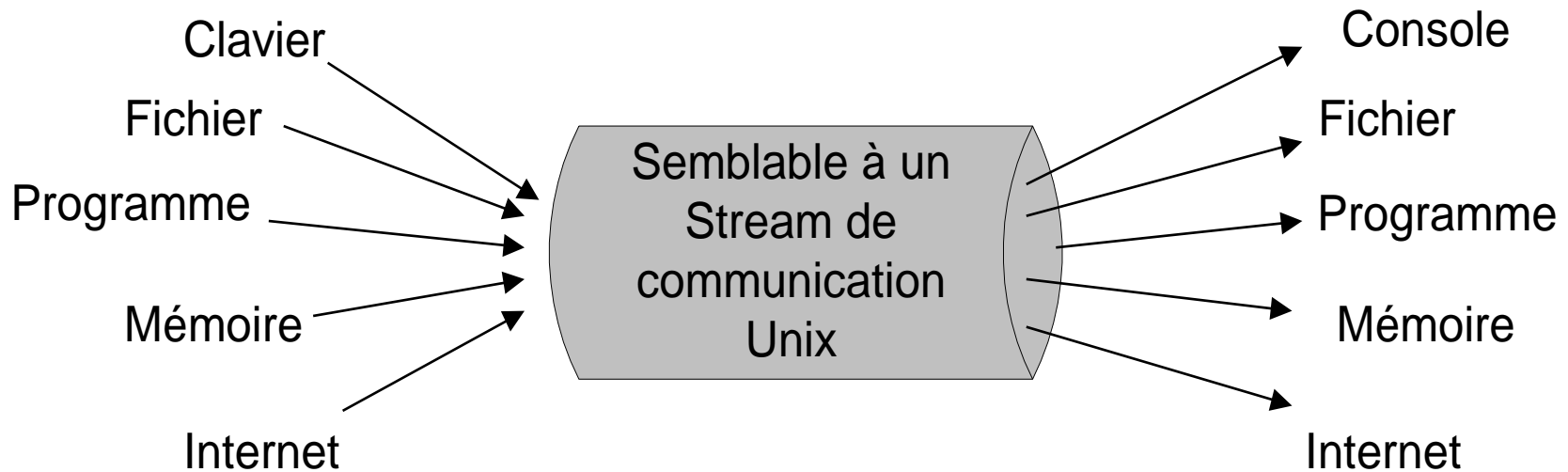


# LECTURE ET ÉCRITURE À L'AIDE DES STREAMS

171

# QU'EST-CE QU'UN STREAM ?

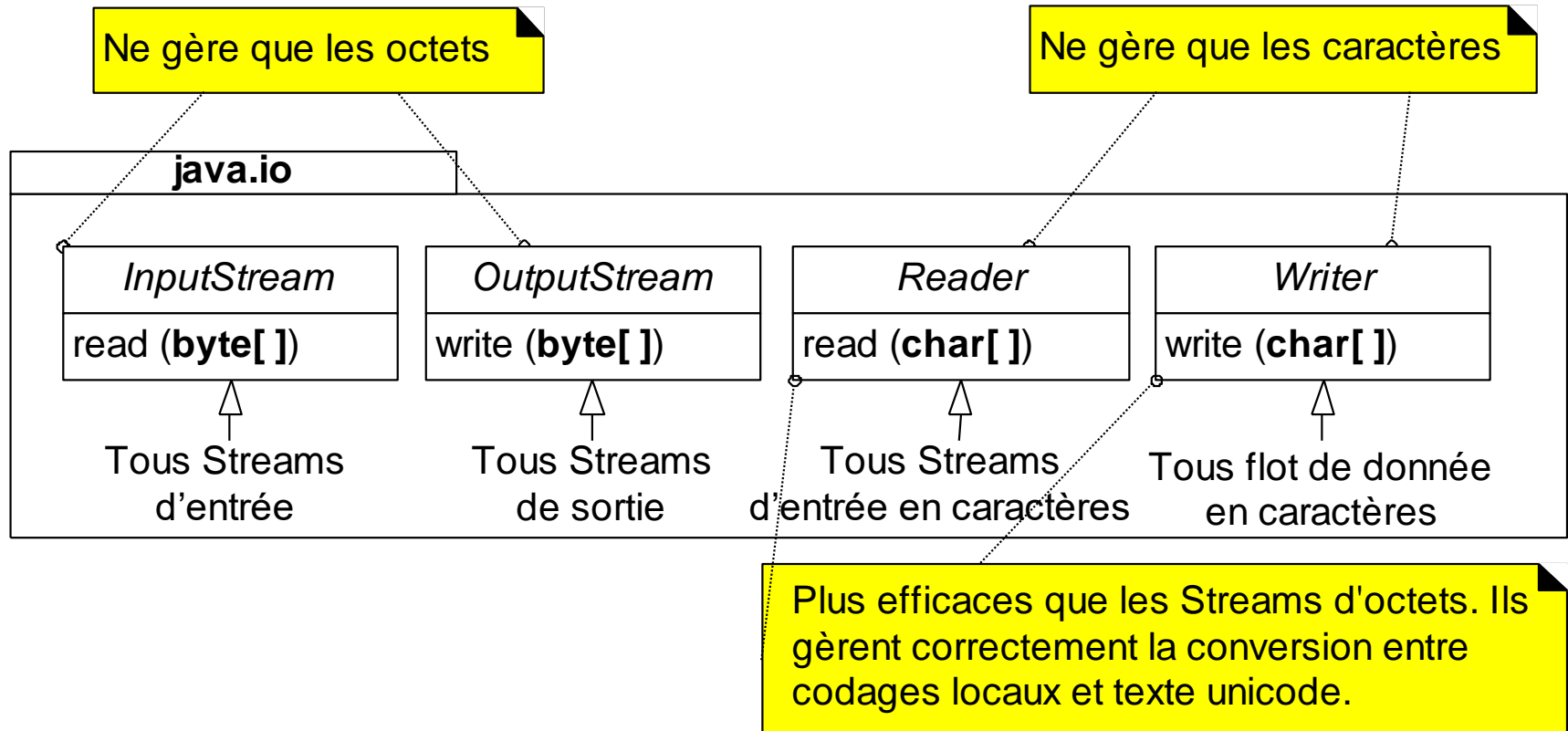
- Un chemin de communication entre la source d'une information et sa destination



## TROIS STREAMS STANDARD SONT DISPONIBLES

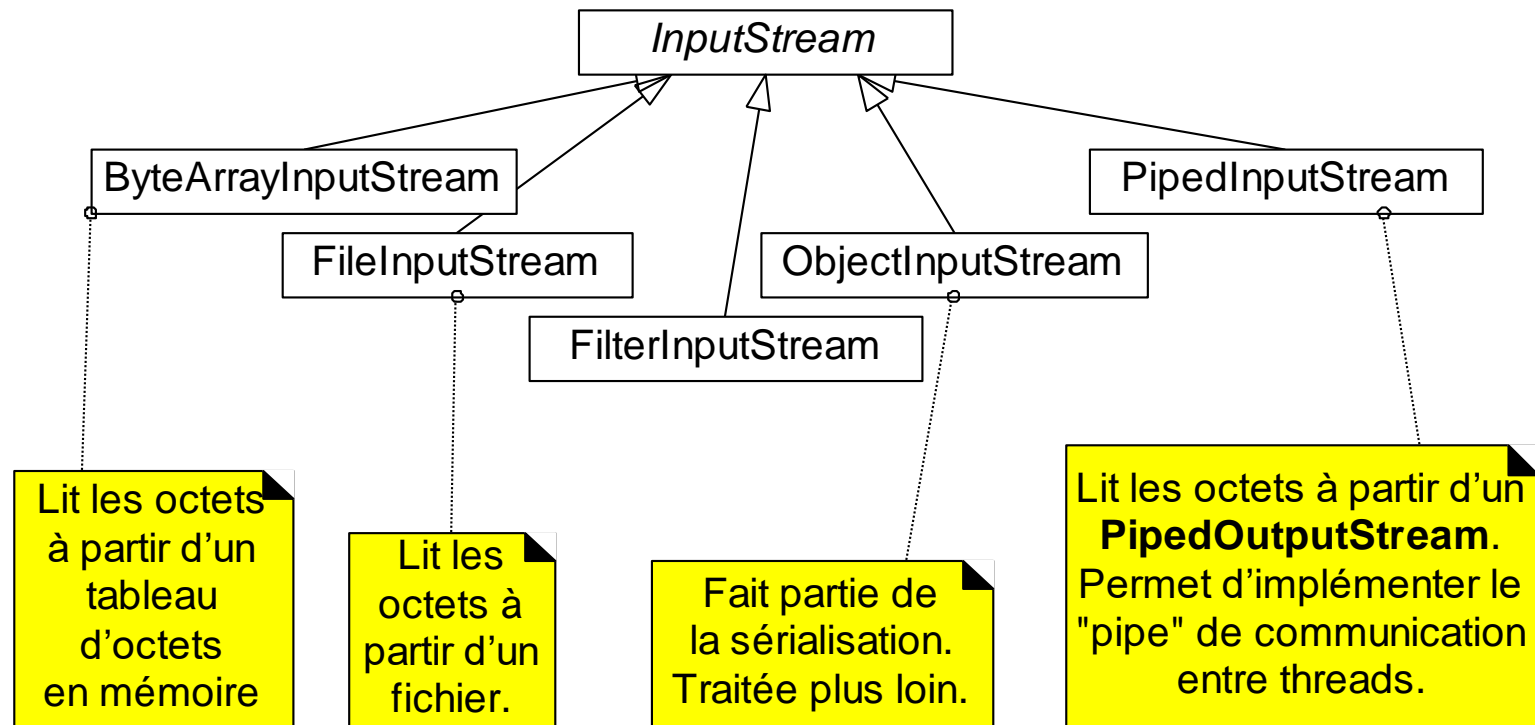
- Tous sont des champs statiques publics de la classe `java.lang.System`
  - *in* - *InputStream* pour la lecture depuis le clavier
  - *out* - *PrintStream* pour l'écriture sur la console
    - **`System.out.println`** (« Ah, oui, on a déjà fait ça »)
  - *err* - *PrintStream* pour l'écriture sur la console
    - **`System.err.println`** (« Certains systèmes d'exploitation permettent la redirection des erreurs »)

# LA HIÉRARCHIE JAVA.IO (SIMPLIFIÉE)



- *InputStream* et *Reader* sont spécialisés pour lire à partir de sources
- *OutputStream* et *Writer* sont spécialisés pour écrire sur des sources

# HIÉRARCHIE INPUTSTREAM PARTIELLE



- *InputStream* est spécialisé pour lire à partir de sources diverses
- Une hiérarchie semblable existe pour *OutputStream*, *Reader*, ...
- Jusqu'à présent, rien ne nous permet de lire un float, int, long, double, ...

## LECTURE/ÉCRITURE DE TYPES PRIMITIFS

- Comment lire un float à partir d'un fichier binaire ?

```
FileInputStream fs = new FileInputStream ("file.txt");  
DataInputStream in = new DataInputStream (fs);  
float n = in.readFloat();
```



# LECTURE DE FICHIERS TEXTE

- Comment lire une ligne à partir d'un fichier texte ?
- Emballez FileReader dans un BufferedReader
  - *BufferedReader* a une méthode
    - `String readLine();`

```
FileReader fr = new FileReader ("file.txt");  
BufferedReader br = new BufferedReader( fr );  
String s = br.readLine();
```

# CONVERSION DE CHAÎNES EN NOMBRES

- Les classes d'encapsulation des types primitifs contiennent la méthode statique *parse<Type>*
- Exemple : la classe Integer
  - encapsule une valeur primitive de type *int* dans un objet
  - dispose d'une méthode *parseInt*

```
public static int parseInt(String s) throws NumberFormatException
```

## ANALYSE (PARSING) D'UNE CHAÎNE

- Il peut être nécessaire d'analyser la chaîne
- Par exemple, un ensemble de nombres délimités par virgules
  - `String s = "3.1, 5.66, -87, 14.1";`
- Utilisez `java.util.StringTokenizer` pour procéder à l'analyse
  - Le constructeur prend un ensemble de délimiteurs
    - `StringTokenizer tokenizer = new StringTokenizer(s, ",");`
    - `String t = tokenizer.nextToken();`
- Méthode `split` de la classe `String`

```
String[] results = "Petit,Grand,Geant".split(",");
```

# ANALYSE ET CONVERSION DE CHAÎNES

- Comment lire plusieurs nombres à partir d'une chaîne de caractères ?
  - Utilisez `java.util.StringTokenizer` pour analyser les chaînes
  - Utilisez des wrappers de primitives pour convertir en types de primitives
  - Dans notre exemple, deux floats délimités par virgules

```
StringTokenizer tokenizer = new StringTokenizer(s, ",");  
String token = tokenizer.nextToken();  
float f1 = Float.parseFloat(token);  
token = tokenizer.nextToken();  
float f2 = Float.parseFloat(token);
```

Découpe la chaîne en mots délimités par le caractère spécifié

Méthode statique de la classe `Float` convertissant une chaîne en float. Il existe également `Double.parseDouble()`, `Integer.parseInt()`,...

# ÉTAPES POUR LIRE LES ENTRÉES DU CLAVIER

- Emballez *System.in* dans le type approprié de stream d'entrée

```
InputStreamReader rdr = new InputStreamReader(System.in);  
BufferedReader in = new BufferedReader(rdr);
```

- Lisez ligne par ligne

```
try  
{  
    String name = in.readLine();  
}  
catch (IOException e)  
{ ... }
```

- La chaîne obtenue peut alors être analysée, convertie,...

# RÉCAPITULATIF

```
BufferedReader in = new BufferedReader
    (new InputStreamReader(System.in));

try {
    System.out.println("Entrer nom :");
    String name = in.readLine();

    System.out.println("Bonjour " + name);

    System.out.println("Veuillez entrer votre age:");
    int age = Integer.parseInt(in.readLine());
}
catch (IOException e) { System.err.println(e); }
```

# EXERCICE

- Créez une interface de contrôle à l'aide des entrées/sorties pour le jeu de dés.
  - Score : Afficher le score
  - Lancer : lance les dés pour le joueur en cours



**JDBC**



184





# JDBC & MySQL

- Une Base de donnée... pourquoi?

# VOCABULAIRE

- La **base** désigne le volume englobant l'ensemble, la boîte qui contient tous les tableaux.
- Une **table** désigne un tableau de données, elle contient des lignes et des colonnes
- Une **entrée** désigne une ligne
- Un **champ** désigne une colonne

# EXAMPLE

idID	pseudo	email	age
1	Coyote	coyote@bipbip.com	25
2	Thunderseb	jadorejquery@unefois.be	24
3	Kokotchy	decapsuleur@biere.org	27
4	Marcel	marcel@laposte.net	47
...	...	...	...

# VOCABULAIRE

- **SGBD** : System de gestion des bases de données
  - La solution standard se nomme **JDBC** : c'est une API qui fait partie intégrante de la plate-forme Java
  - L'avantage de JDBC, c'est qu'il est nativement prévu pour pouvoir s'adapter à n'importe quel SGBD
- **MySQL** : solution libre et gratuite, c'est le SGBD le plus répandu. C'est d'ailleurs celui que nous allons utiliser dans ce cours !
- **PostgreSQL** : solution libre et gratuite, moins connue du grand public mais proposant des fonctionnalités inexistantes dans MySQL ;
- **Oracle** : solution propriétaire et payante, massivement utilisée par les grandes entreprises. C'est un des SGBD les plus complets, mais un des plus chers également ;
- **SQL Server** : la solution propriétaire de Microsoft ;
- **DB2** : la solution propriétaire d'IBM, utilisée principalement dans les très grandes entreprises sur des *Mainframes*.

# SQL

Le langage SQL n'est absolument pas lié au langage Java : c'est un langage à part entière, uniquement destiné aux bases de données.

Pour vous donner une première idée de la syntaxe employée, voici un exemple de requête SQL :

```
SELECT id, nom, prenom, email FROM utilisateurs ORDER BY id;
```

# SQLITE

- **SQLite Studio**
  - <https://sqlitestudio.pl/index.rvt?act=download>
- **Driver Java Sqlite (Mettre Jar dans lib du projet)**
  - <https://bitbucket.org/xerial/sqlite-jdbc/downloads/sqlite-jdbc-3.21.0.jar>
- **Aller sur Sqlite Studio**
  - **Créer une Base et une table**
  - **Mettre des données**
- **Lien vers la base**
  - `.../SQLiteStudio/nomDeMaBase`

# MySQL

- **Wamp**
  - <http://www.wampserver.com/>
- **Driver Java MySQL (Mettre Jar dans lib du projet)**
  - <https://dev.mysql.com/downloads/connector/j/>
- **Aller sur PhpMyadmin**
  - **Créer une Base et une table**
  - **Mettre des données**

# MySQL

```
public class ConnexionJDBC {  
  
    //Version Wamp  
    public static final String URL = "jdbc:mysql://localhost:3306/nomMaBase";  
    public static final String LOGIN = "root";  
    public static final String PASSWORD = "";  
  
    //Version SQLite  
    public static String URL = "jdbc:sqlite:C:/.../sqlite/maBase";  
}
```



# REQUÊTE INSERT / UPDATE (VERSION MYSQL)

```
private final static String QUERY_SAVE_ELEVES = "INSERT INTO Eleve (prenom, note) VALUES (?, ?);";
public static void insertEleve(EleveBean eleve) throws Exception {
    Connection con = null; //java.sql pour le choix
    PreparedStatement stmt = null; //java.sql pour le choix
    try {
        //Pour travailler avec Tomcat
        DriverManager.registerDriver(new com.mysql.jdbc.Driver());
        con = DriverManager.getConnection(URL, LOGIN, PASSWORD); // La connexion
        stmt = con.prepareStatement(QUERY_SAVE_ELEVES);
        // Remplir la requête
        stmt.setString(1, eleve.getPrenom());
        stmt.setInt(2, eleve.getNote());
        // Lancer la requête
        stmt.executeUpdate();
    } finally {
        // On ferme la connexion
        if (con != null) {
            try {
                con.close();
            } catch (final SQLException e) {
                e.printStackTrace();
            }
        }
    }
}
```

# REQUÊTE INSERT / UPDATE (VERSION SQLITE)

```
private final static String QUERY_SAVE_ELEVES = "INSERT INTO Eleve (prenom, note) VALUES (?, ?);";
public static void insertEleve(EleveBean eleve) throws Exception {
    Connection con = null;
    PreparedStatement stmt = null;
    try {
        //Pour travailler avec Tomcat + SQLITE Rajouter :
        Class.forName("org.sqlite.JDBC");
        con = DriverManager.getConnection(URL); // La connexion
        stmt = con.prepareStatement(QUERY_SAVE_ELEVES);
        // Remplir la requête
        stmt.setString(1, eleve.getPrenom());
        stmt.setInt(2, eleve.getNote());
        // Lancer la requête
        stmt.executeUpdate();
    } finally {
        // On ferme la connexion
        if (con != null) {
            try {
                con.close();
            } catch (final SQLException e) {
                e.printStackTrace();
            }
        }
    }
}
```

# REQUÊTE SELECT

```
private final static String QUERY_FIND_ELEVES = "SELECT * FROM Eleve ;";

public static void requetSelect() throws Exception {
    Connection con = null;
    Statement stmt = null;
    try {
        //Pour travailler avec Tomcat et wamp Rajouter :
        DriverManager.registerDriver(new com.mysql.jdbc.Driver());
        con = DriverManager.getConnection(URL, LOGIN, PASSWORD); //La connexion
        stmt = con.createStatement();
        ResultSet rset = stmt.executeQuery(QUERY_FIND_ELEVES);
        while (rset.next()) {
            EleveBean eleve = rsetToEleve(rset);
        }
    }
    finally {
        if (con != null) { // On ferme la connexion
            try {
                con.close();
            } catch (final SQLException e) {
                e.printStackTrace();
            }
        }
    }
}
```

# REQUÊTE

```
private static EleveBean rsetToEleve(ResultSet rset) throws SQLException {  
    final EleveBean eleve = new EleveBean();  
  
    eleve.setId(rset.getInt("id"));  
    eleve.setNom(rset.getString("nom"));  
    eleve.setPrenom(rset.getString("prenom"));  
  
    return eleve;  
}
```

# EXERCICE

- Créer une table Eleve (id, nom, note) dans la base MySQL
- Créer la classe EleveBean possédant les attributs correspondants
- Créer la classe ConstanteJDBC avec les constantes de la base
- Créer un dossier lib dans le projet (au même niveau que src)
- Ajouter la librairie de Mysql dans le repertoire lib
- Clic droit sur la librairie -> Add to build path
- Créer la classe EleveDaoUtils contenant les differentes méthodes pour travailler avec la base
- Sauvegarder et Charger un élève en bdd dans le main



198



# PROJET J2EE

# COMPARAISON PHP J2EE

- o <https://blog.axopen.com/2014/06/java-vs-php-creation-dune-application-web-site-web-en-2014/>

Critère	JAVA	PHP
Paradigme	Orientée objet, structurée, impérative	Impératif, objet, fonctionnel, procédural, réflexif. Langage interprété
Typage	Statique, fort, sûr, nominatif	Dynamic, Faible
Compatibilité ascendante	OUI	Pas toujours
Multiplateforme	OUI	OUI
Type d'application cible	Toutes	Web
Gestion UTF-8	OUI	Partielle
Popularité (selon l'index TIOBE)	16,9%	3,38%
Pré-requis	Une JVM	Un interpréteur PHP et un serveur web (Souvent Apache)
Lourdeur de mise en place	Moyen	Faible
Environnement de développement	Eclipse, IntelliJ, NetBeans, JDeveloper...	Simple éditeur de texte, ou IDE intégré

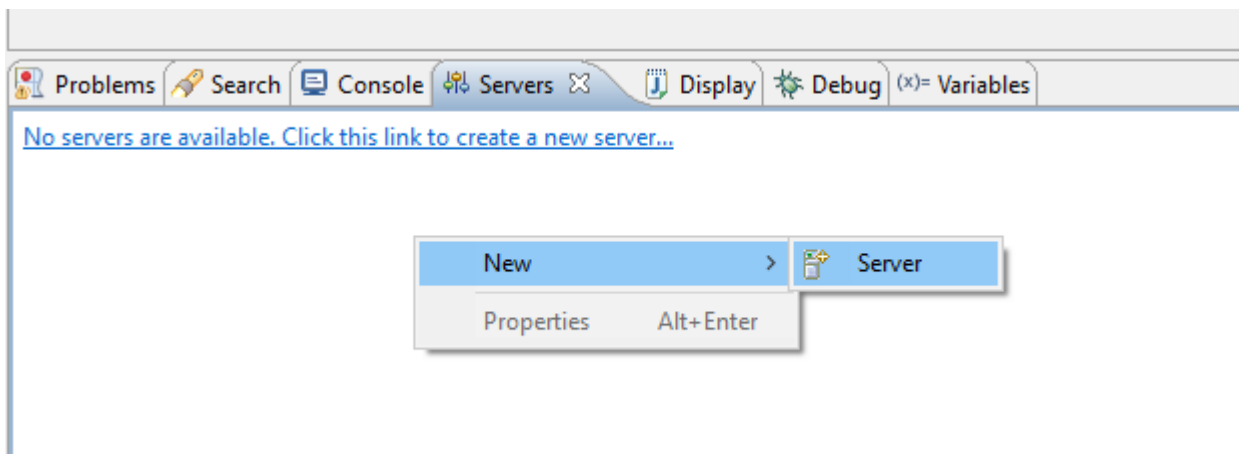
# TOMCAT LE SERVEUR D'APPLICATION

- <http://tomcat.apache.org/download-80.cgi>
- Le relier à Eclipse

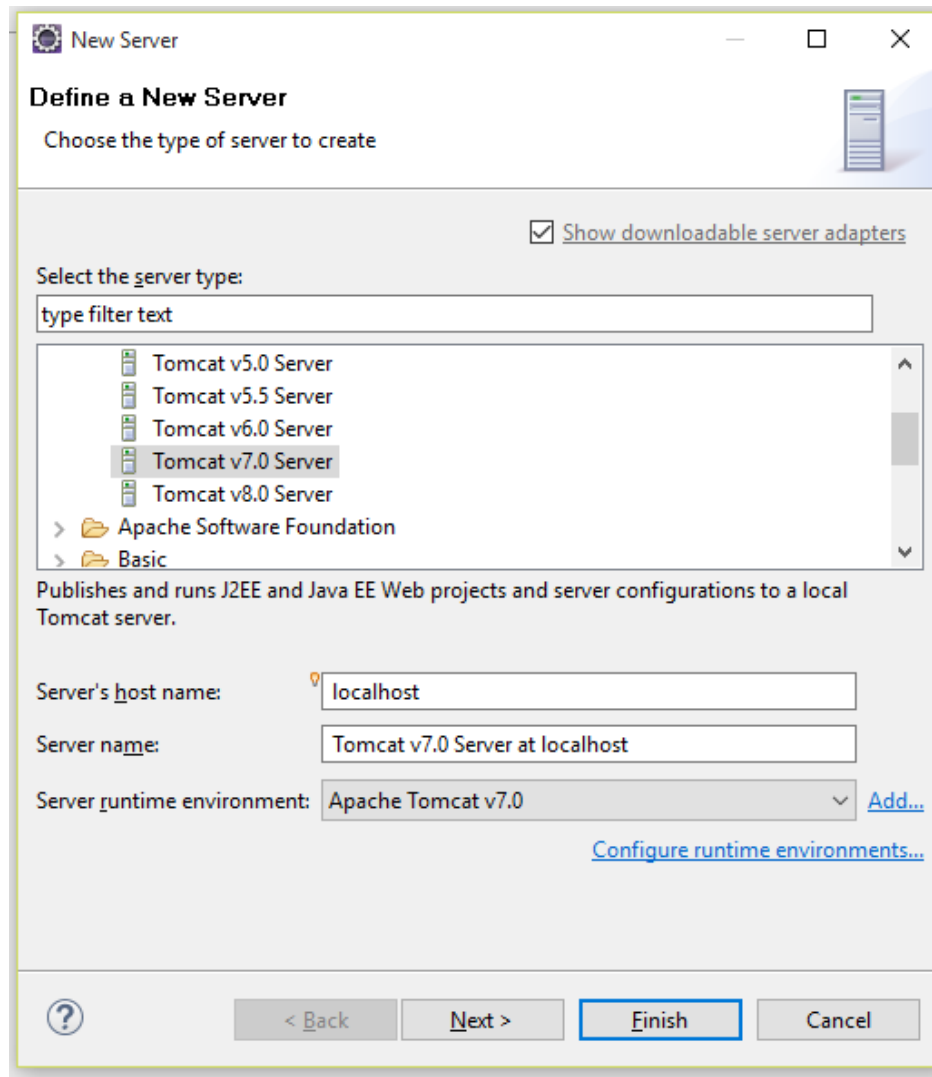


# CRÉER UN SERVEUR TOMCAT

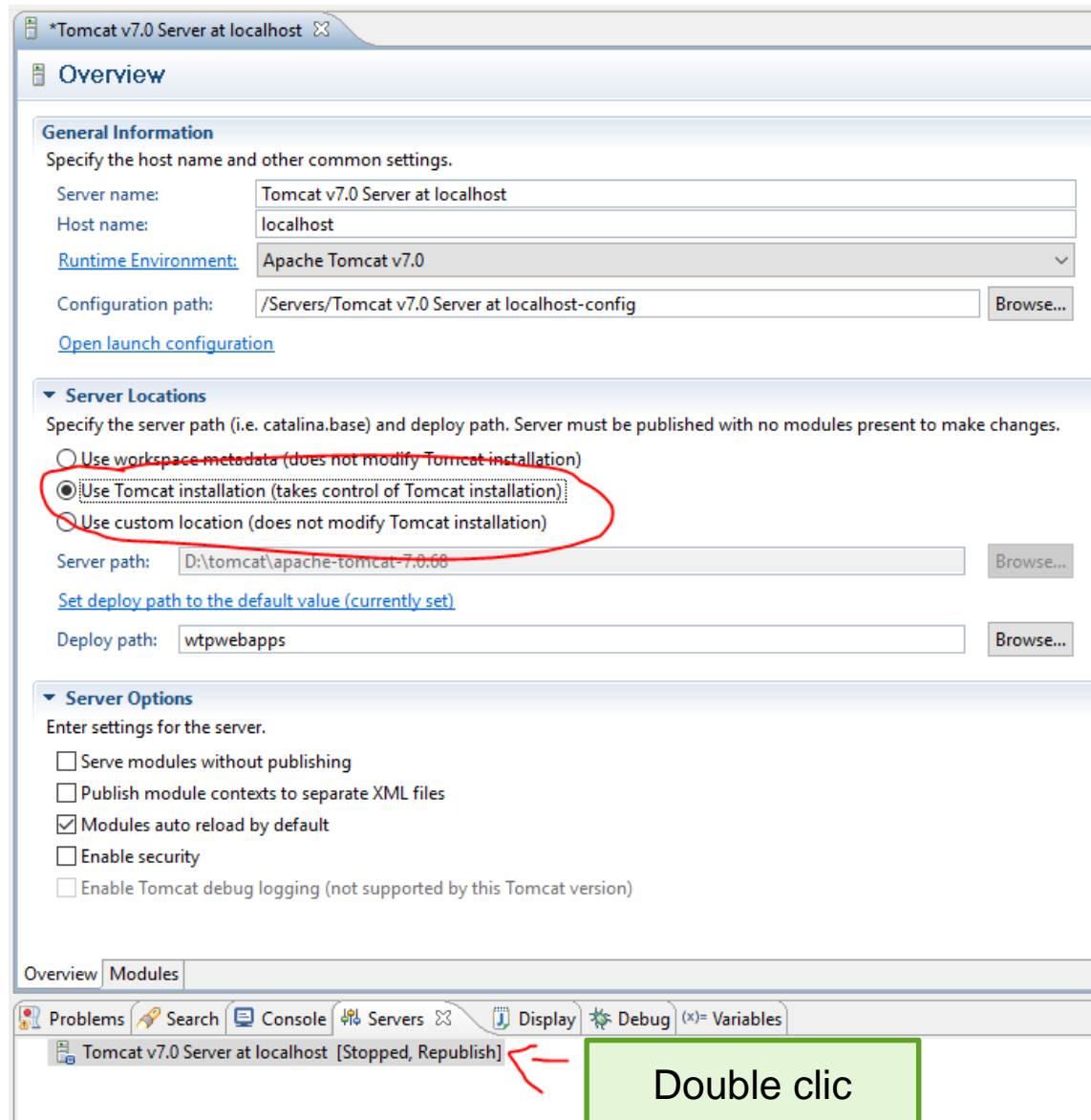
- (Dans Eclipse) Window -> showView -> Server
- Dans l'onglet Server -> clic droit



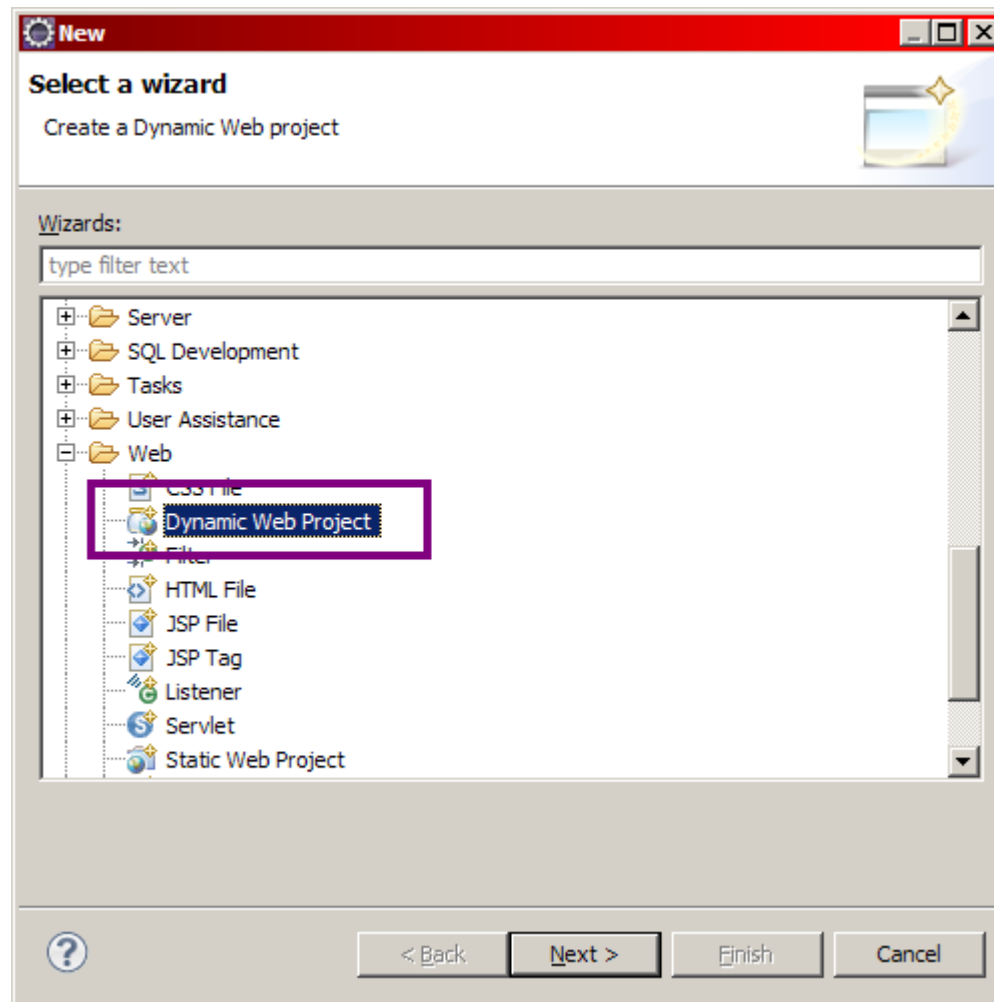
# CRÉER UN SERVEUR TOMCAT



# CRÉER UN SERVEUR TOMCAT

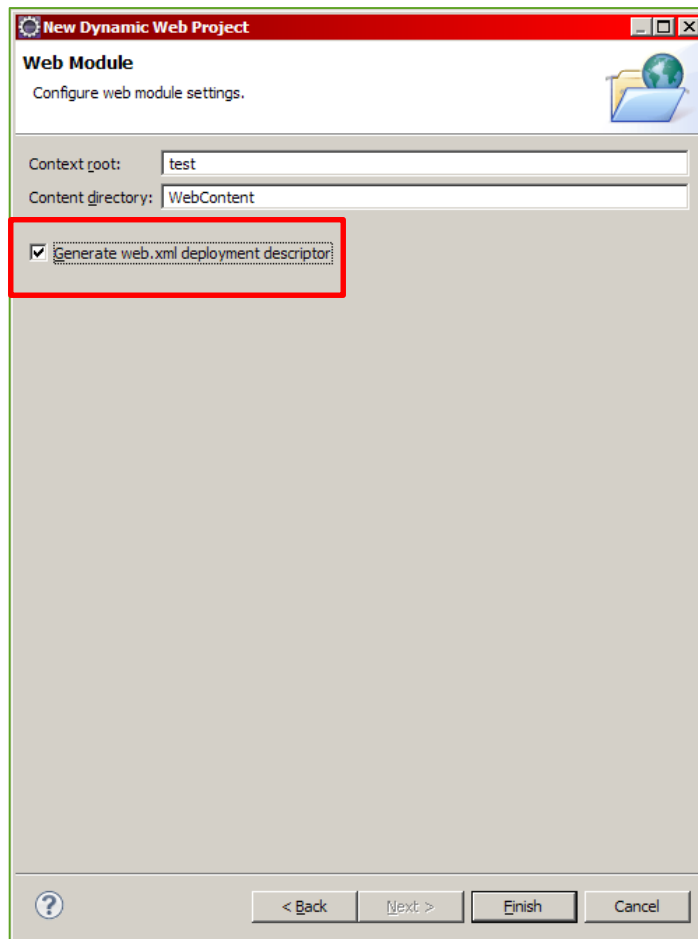


# CRÉER UN PROJET

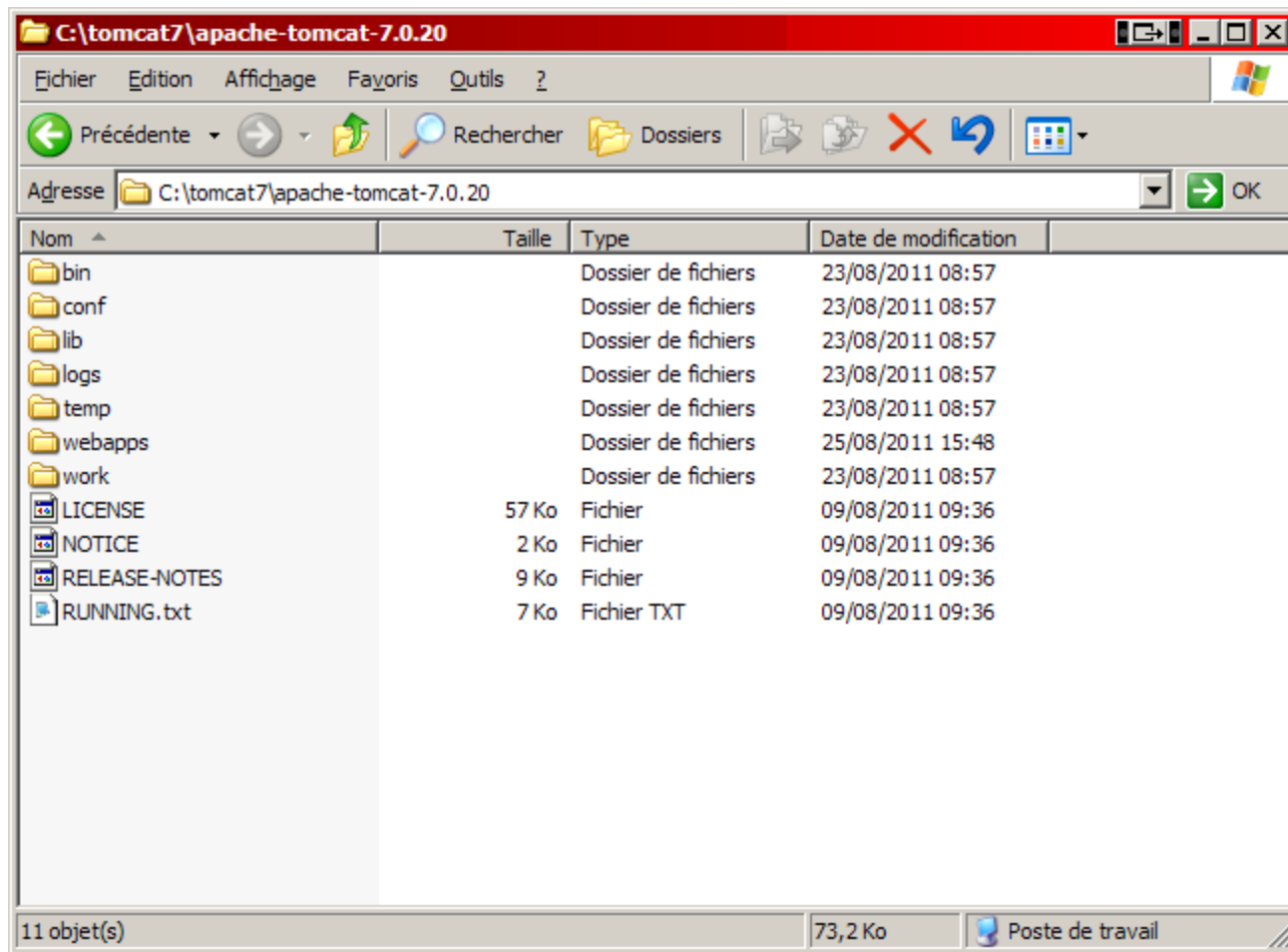


# CRÉER UN PROJET

- Next -> Next ->... jusqu'à :



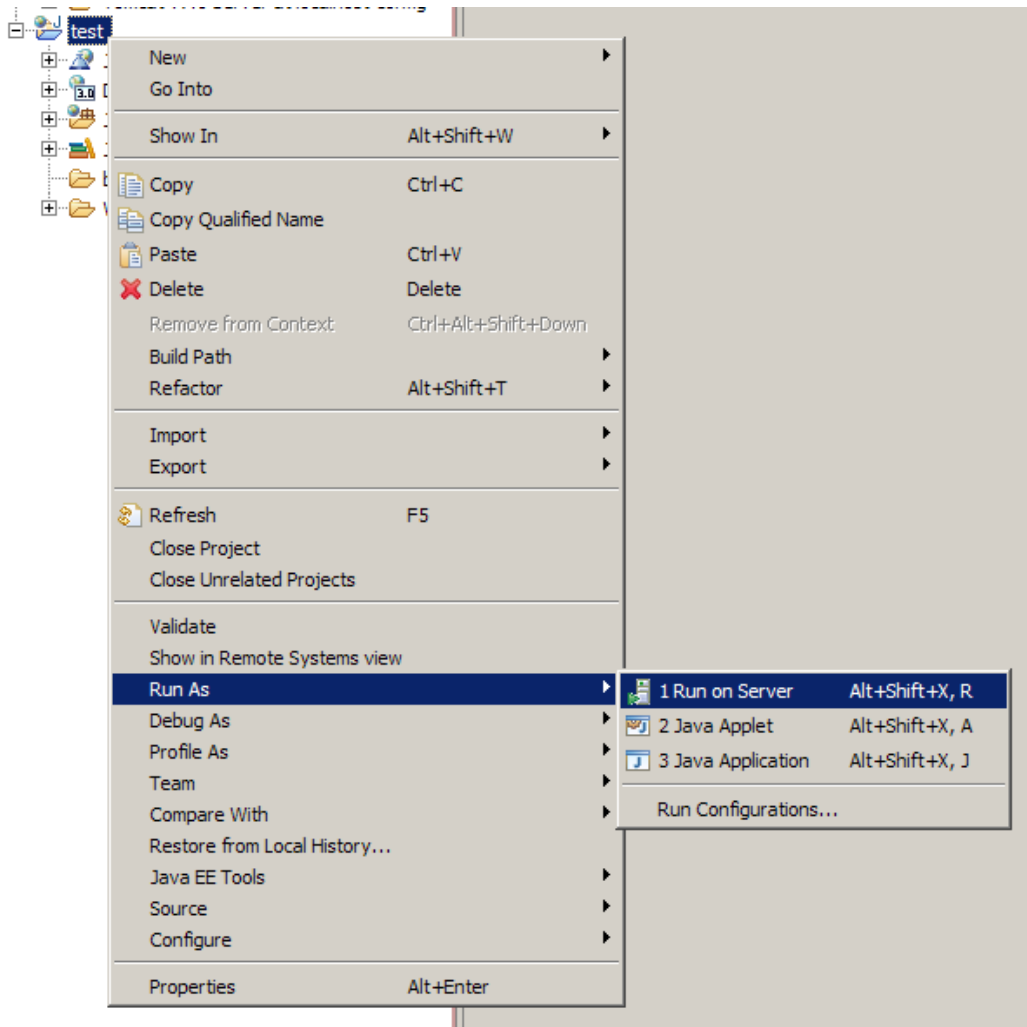
# TOMCAT (LE RÉPERTOIRE)



# TOMCAT (LE RÉPERTOIRE)

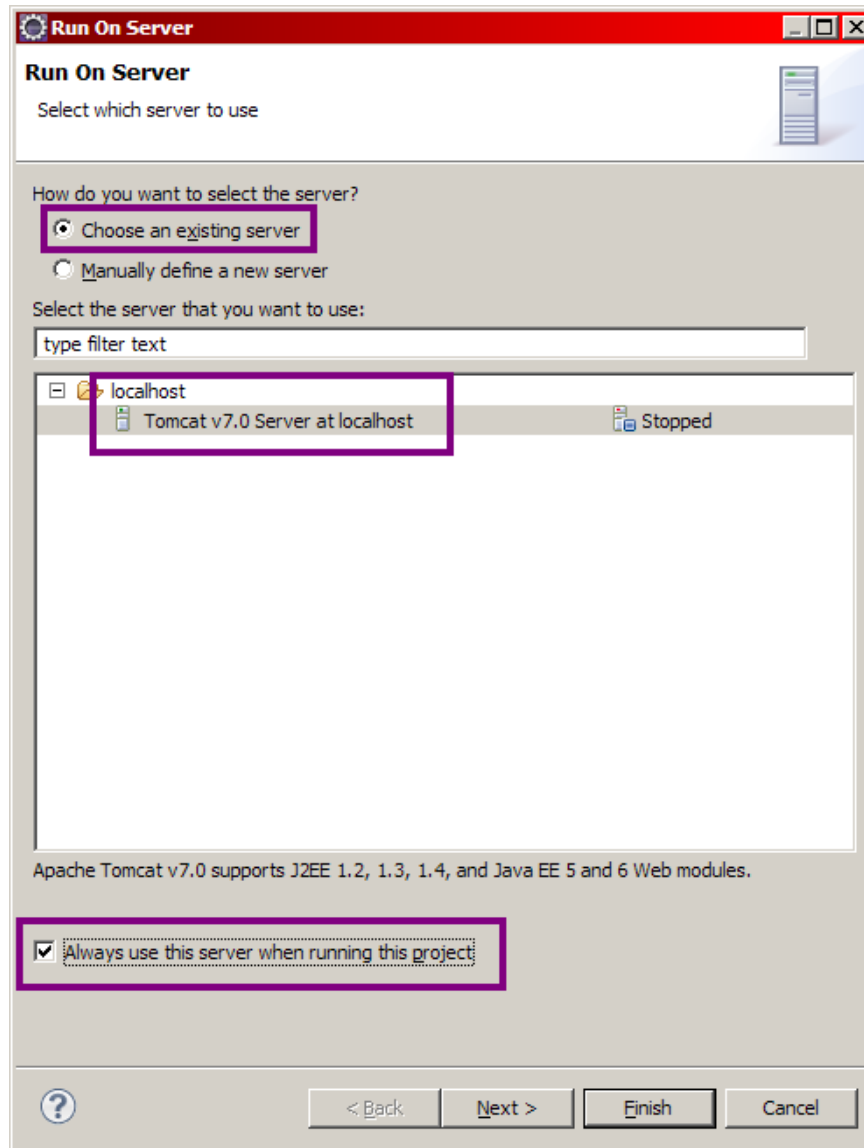
- Un dossier nommé **webapps** : c'est ici que seront stockées par défaut vos applications.
- Dans le dossier **conf** les fichiers suivants :
  - **server.xml** : contient les éléments de configuration du serveur ;
  - **context.xml** : contient les directives communes à toutes les applications web déployées sur le serveur ;
  - **tomcat-users.xml** : contient entre autres l'identifiant et le mot de passe permettant d'accéder à l'interface d'administration de votre serveur Tomcat ;
  - **web.xml** : contient les paramètres de configuration communs à toutes les applications web déployées sur le serveur.

# LANCER LE PROJET SUR TOMCAT



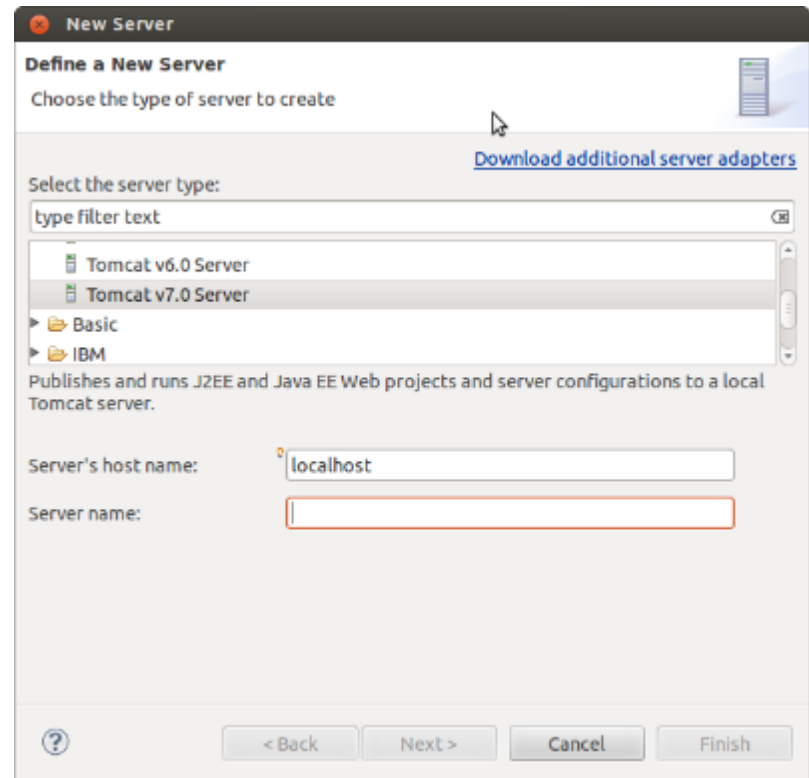


# LANCER LE PROJET SUR TOMCAT

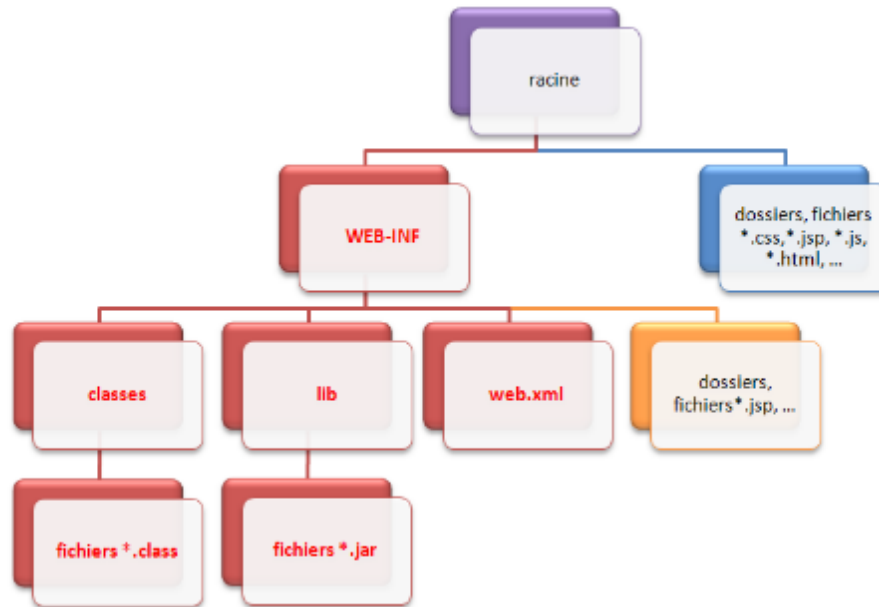


# LANCER LE PROJET SUR TOMCAT

- Erreur fréquente :
  - On ne peut pas sélectionner le serveur, ou en créer un
- Solution
  - <http://stackoverflow.com/questions/14791843/eclipse-add-tomcat-7-blank-server-name>

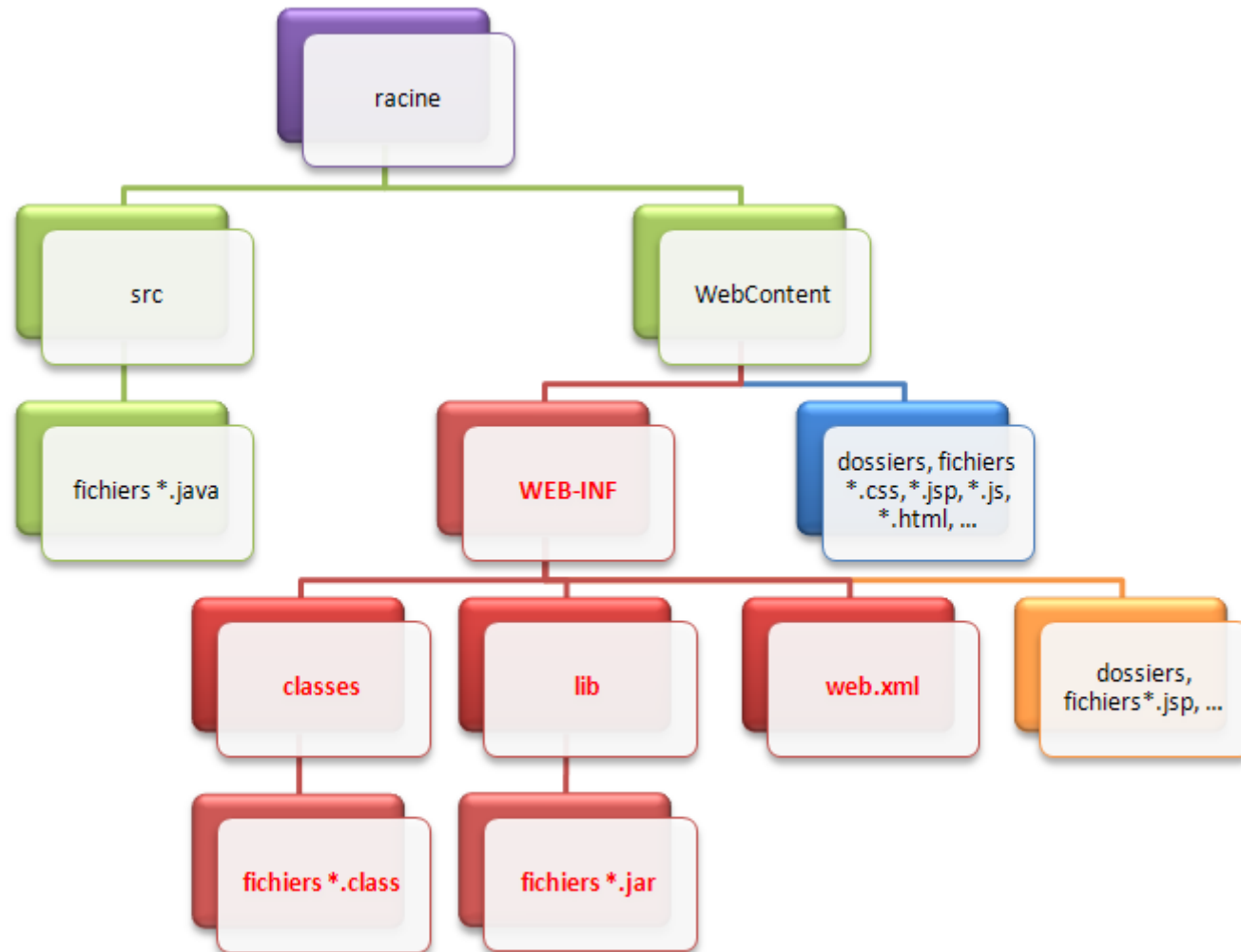


# STRUCTURE D'UNE APPLICATION JEE

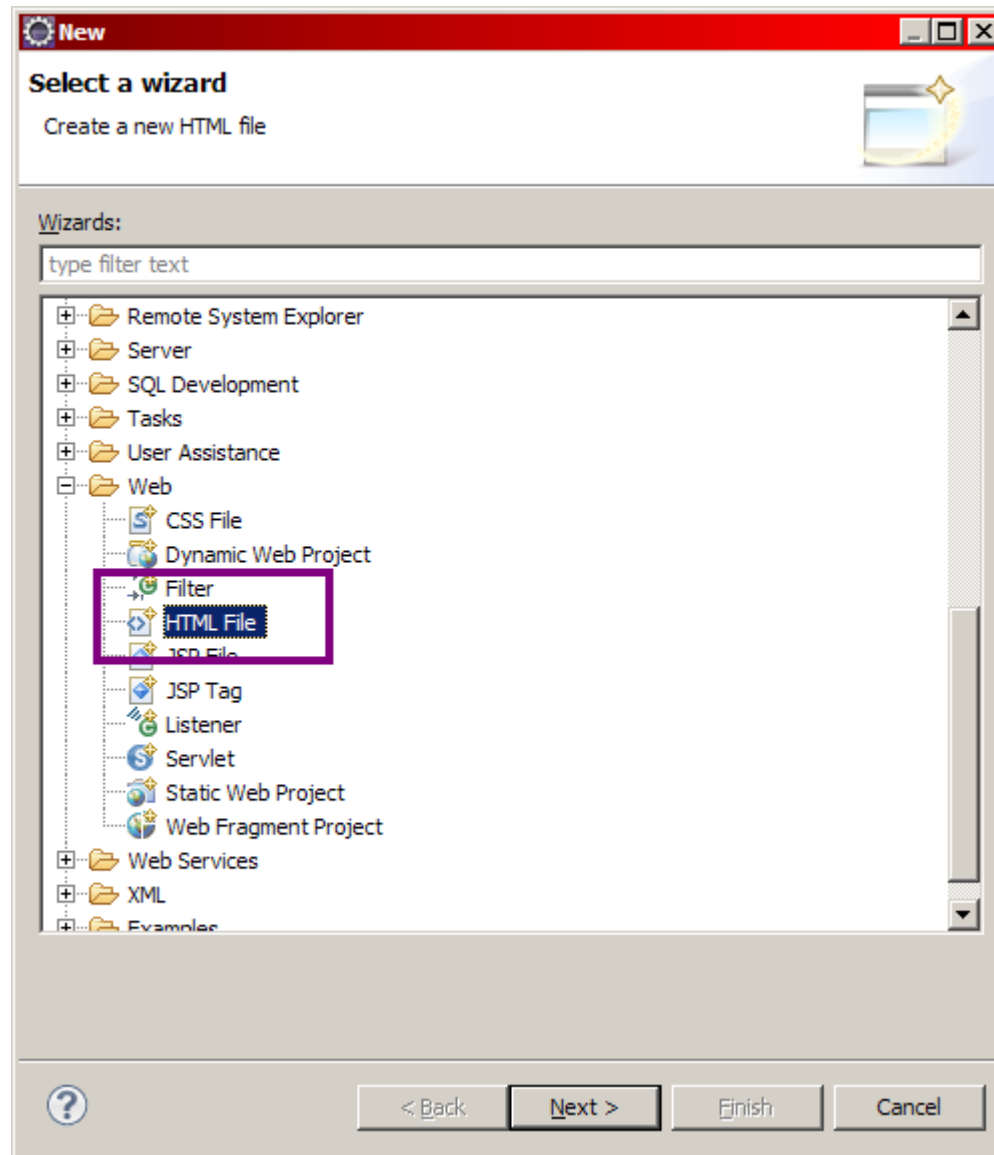


- En rouge : Obligatoire
- En bleu : Public donc accessible par le client à l'aide d'une URL
- En orange : Privée

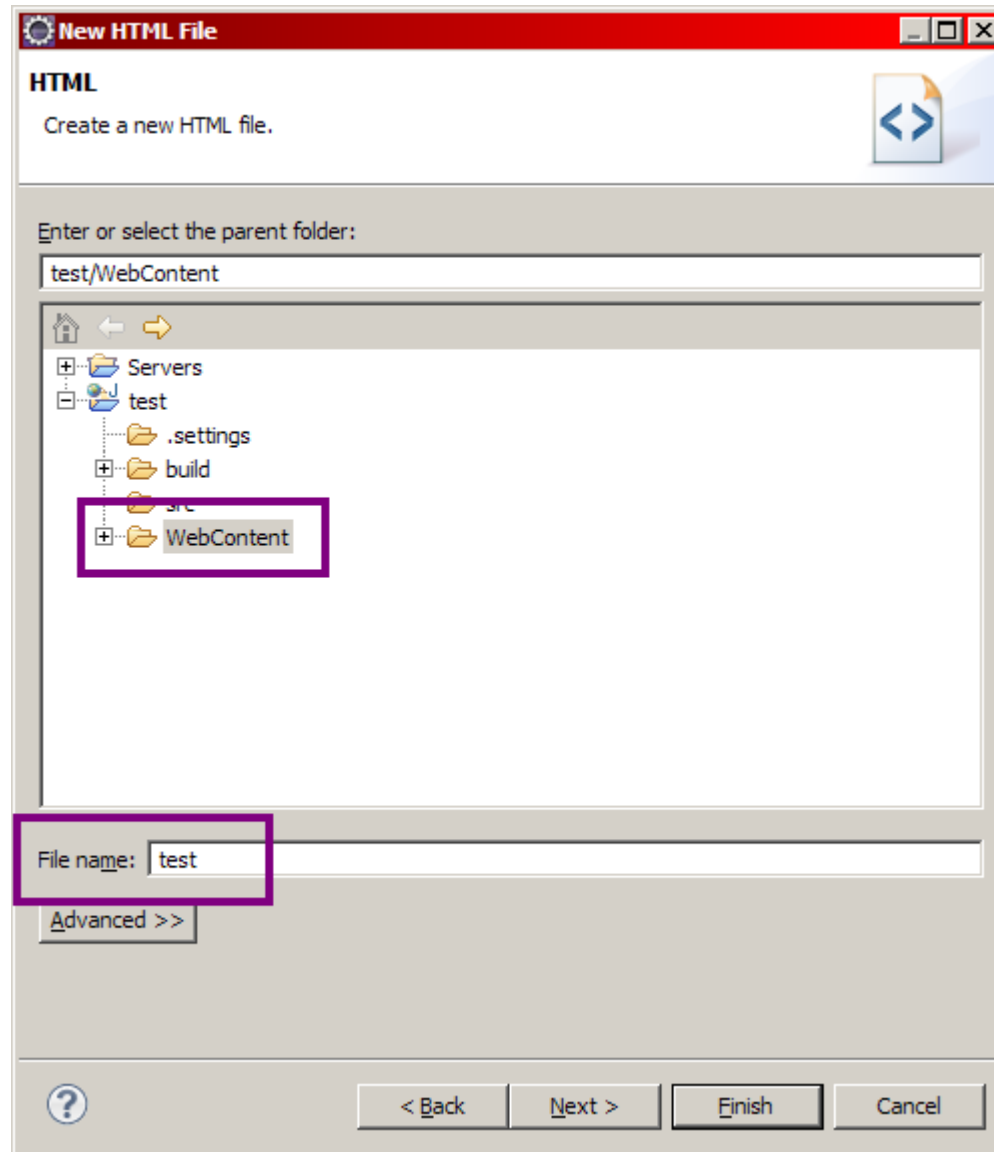
# APPLICATION JEE D'ECLIPSE



# PREMIER PAGE



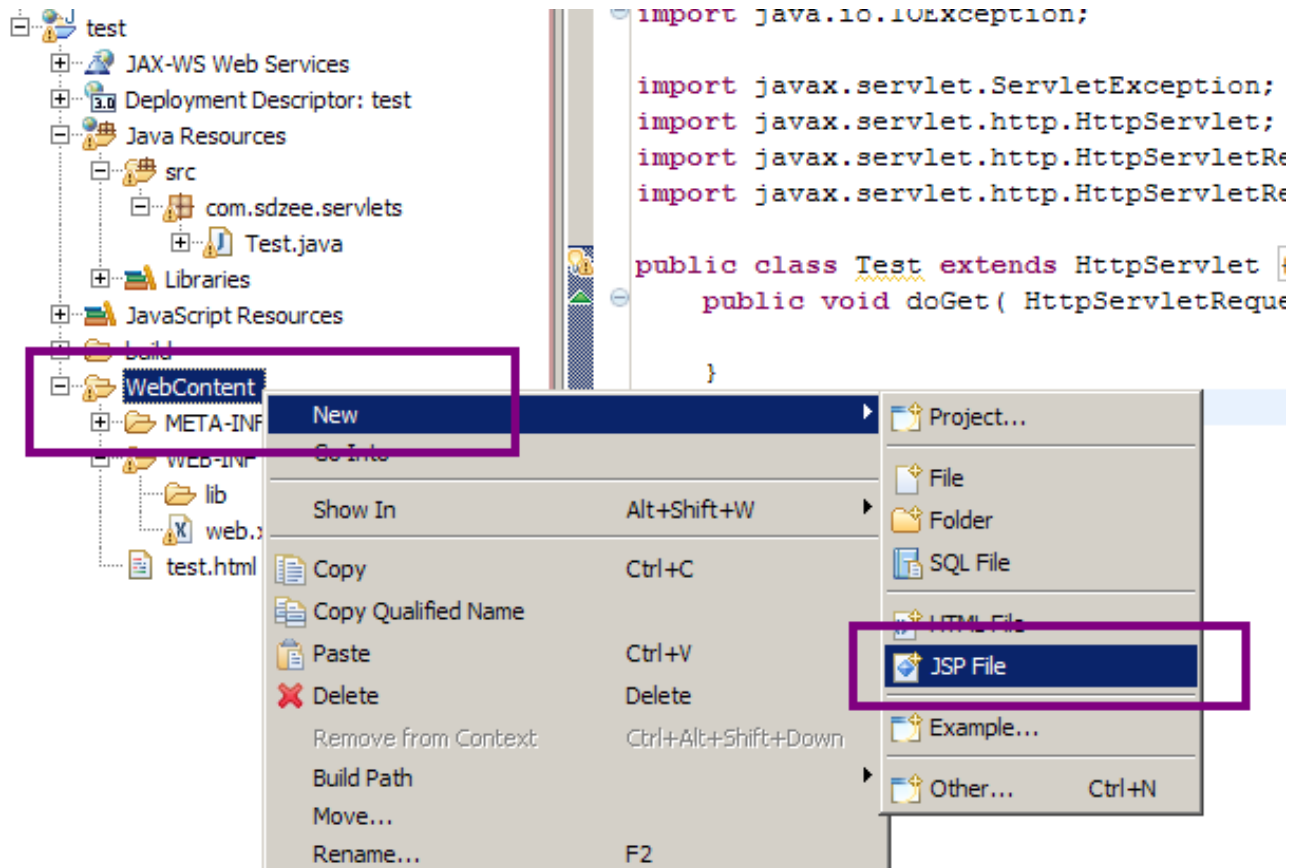
# PREMIER PAGE



# AFFICHER LA PAGE

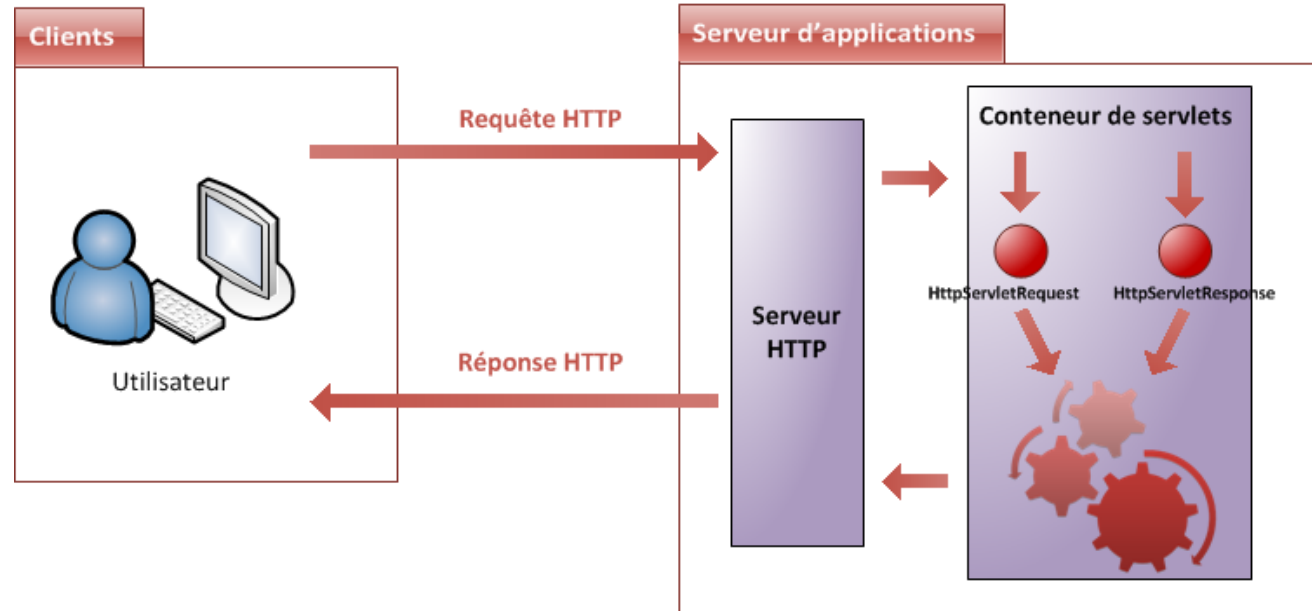
- Ajouter quelque chose dans le body
- Relancer le run du projet
- L'afficher sur le navigateur web
  - <http://localhost:8080/ExempleJSP/test.html>

# PREMIERE JSP





# SERVLET



- Une servlet est une classe Java, qui a la particularité de **permettre le traitement de requêtes et la personnalisation des réponses**.
- Dans la très grande majorité des cas une servlet n'est rien d'autre qu'une classe capable de recevoir une requête HTTP envoyée depuis le navigateur de l'utilisateur, et de lui renvoyer une réponse HTTP.

# MASQUER NOS JSP

- C'est nos Servlet qui vont appeler nos Jsp
- Créer une servlet avec Eclipse.
- `Extends` `HttpServlet`
- L'url de redirection se fait grâce à l'annotation de **classe** (donc au dessus de `public class...`) :

```
@WebServlet("/toto")
```

# MASQUER NOS JSP

```
@WebServlet("/toto") //http://localhost:8080/ExempleJSP/toto
public class MaServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
        super.doGet(req, resp);
    }

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
        super.doPost(req, resp);
        doGet(req, resp); //Pour faire la même action que le get
    }
}
```

# REDIRIGER LA SERVLET SUR LA JSP CACHÉE

- On redirige vers notre jsp grâce à l'instruction :

- ```
getServletContext().getRequestDispatcher("/WEB-INF/majsp.jsp").forward(request, response);
```

# MASQUER NOS JSP

```
@WebServlet("/toto") //http://localhost:8080/ExempleJSP/toto
public class MaServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
        ServletException, IOException {

        getServletContext().getRequestDispatcher("/WEB-INF/majsp.jsp").forward(req, resp);
    }
}
```

- On redémarre le serveur et on lance à l'aide de l'URL :  
<http://localhost:8080/ExempleJSP/toto>

# COMMUNICATION SEVLET - JSP

## ○ Java

```
protected void doGet(HttpServletRequest req, HttpServletResponse resp)
    throws Exception{

    String message = "<b>Transmission de variables</b> : OK !";
    req.setAttribute("macle", message);

    getServletContext().getRequestDispatcher("/WEB-INF/majsp.jsp").forward(req, resp);
}
```

## ○ JSP (majsp.jsp)

```
<p>
<%
    String attribut = (String) request.getAttribute("macle");
    out.println( attribut );
%>
</p>
```

# TRANSMETTRE UN JAVA BEAN

- Transmettre un élève du Java à la JSP
- La classe doit être dans un package et posséder un constructeur sans paramètre

//Généré automatiquement

```
<%@page import="bean.EleveBean"%>
```

```
<p>
```

Eleve :

```
<%
```

```
    EleveBean eleve = (EleveBean) request.getAttribute("eleve");  
    out.println( eleve.getPrenom() );  
    out.println( eleve.getNom() );
```

```
%>
```

```
</p>
```

# JSP SANS CODE JAVA

<p>

Eleve sans code java :

```
<!-- bean de type Elevebean : cherche en l'attribut l'id -->
```

```
<jsp:useBean id="eleve" class="bean.EleveBean" scope="request" />
```

```
<jsp:getProperty name="eleve" property="prenom" />
```

```
<jsp:getProperty name="eleve" property="nom" />
```

</p>

<p>

Eleve avec EL :

```
${eleve.prenom}
```

```
${eleve.nom}
```

</p>



# POUR CONTINUER...

- <https://openclassrooms.com/courses/creez-votre-application-web-avec-java-ee>



# WEBSERVICE REST

226

# GÉNÉRALITÉS

- Développement des services Web de type REST
- Qu'est ce qu'un Webservice ?
- Qu'est que REST ?
  - JAX-RS : Java API for RESTful Web Service
  - Le développement des services web REST repose sur l'utilisation de classe Java et d'annotation.
  - JERSEY : implémentation de référence fournie par Oracle
    - <https://jersey.java.net/>
- Code Serveur
  - Code du traitement du web service
- Code Client
  - Code qui permet d'appeler le web service

# OUTILS

- Eclipse J2EE
- Lib Jersey
  - <https://jersey.java.net/>
- Tomcat 7
  - <http://tomcat.apache.org/download-70.cgi>
- Plugin Chrome : Advanced Rest Client
- GSON
  - <https://github.com/google/gson>

# COURS J2EE

- Création d'un projet avec Tomcat cours J2EE

# BUT D'UN WEBSERVICE

- Le serveur nous retourne l'élève de la base de donnée du nom passé en paramètre de l'url.



# ANNOTATION

- @Consumes
  - Le type MIME que la méthode attend(Text, XML, JSON...)
  - Les différents types sont définis dans la classe MediaType
- @Produce
  - Type MIME que la méthode retourne
- @Path
  - Chemin traitant la classe.

# ANNOTATION EXEMPLE

```
@Path("/MonService")
public class TestService {

    @Path("/helloWorld")
    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String hello() {
        System.out.println("HelloWorld !!");
        return "HelloWorld";
    }
}
```

- URL : <http://localhost:8080/TestWS/rest/MonService/helloWorld>



# RÉSULTAT

>

http://localhost:8080/TestWS/rest/MonService/helloWorld

⋮

☒ GET ☐ POST ☐ PUT ☐ PATCH ☐ DELETE ☐ HEAD ☐ OPTIONS ☐ Other

Raw

Form

Headers

Clear

Send

Status: 200: OK ? Loading time:22ms

Response headers (4)

Request headers (5)

Redirects (0)

Server: Apache-Coyote/1.1

Content-Type: text/plain

Content-Length: 10

Date: Thu, 14 Apr 2016 16:15:25 GMT

Raw

Parsed

Response

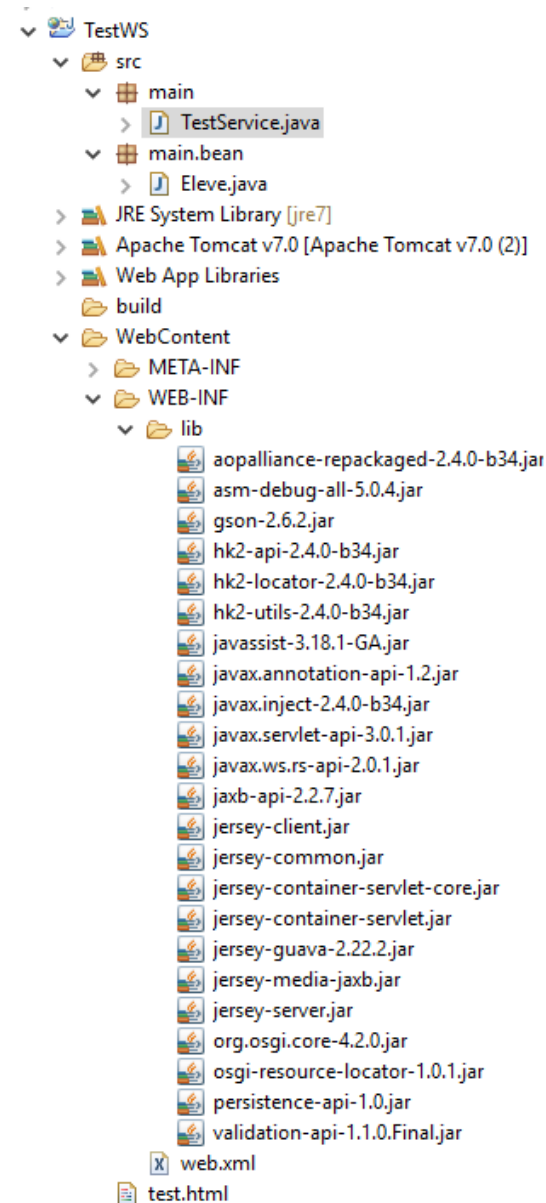
OPEN OUTPUT IN NEW WINDOW COPY TO CLIPBOARD SAVE AS FILE OPEN IN JSON TAB

HelloWorld

Code highlighting thanks to [CODE MIRROR](#)

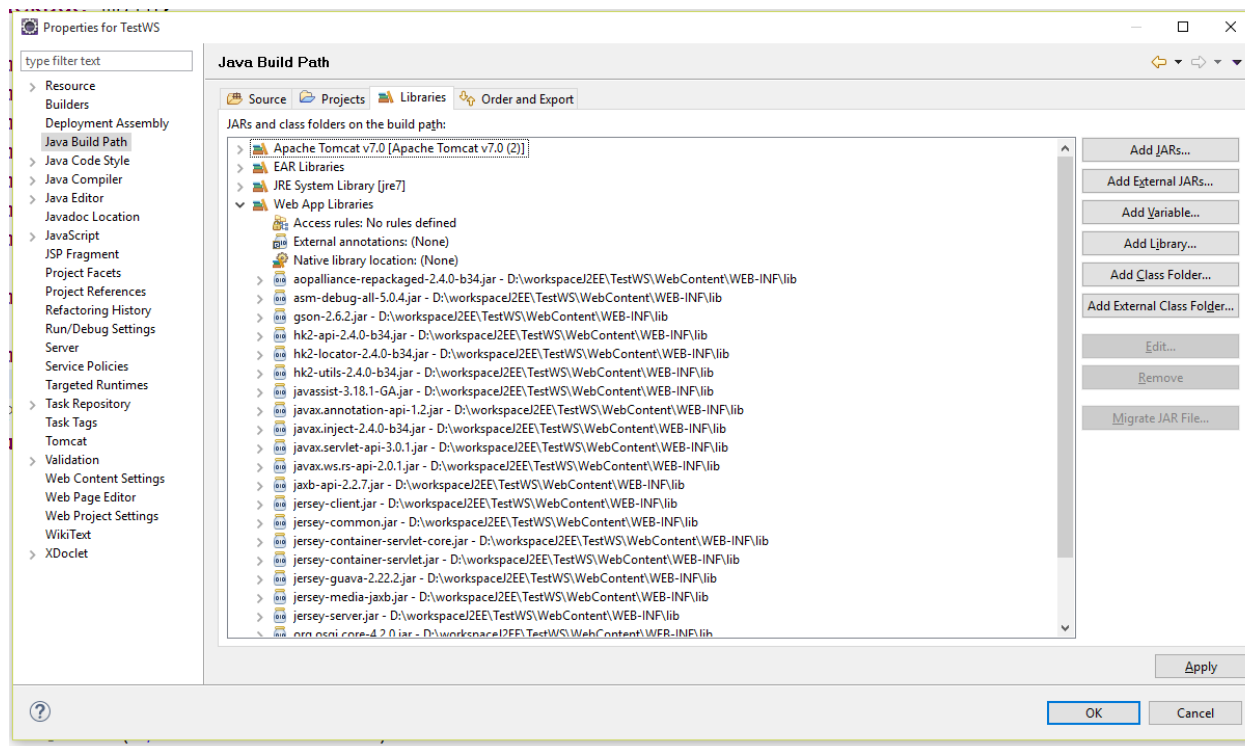
# 1<sup>ER</sup> SERVICE WEB

- Ajouter toute les libs de Jersey dans lib



# 1<sup>ER</sup> SERVICE WEB

- Vérifier qu'elles sont bien ajoutées au projet.
  - Clic droit sur le projet -> Properties



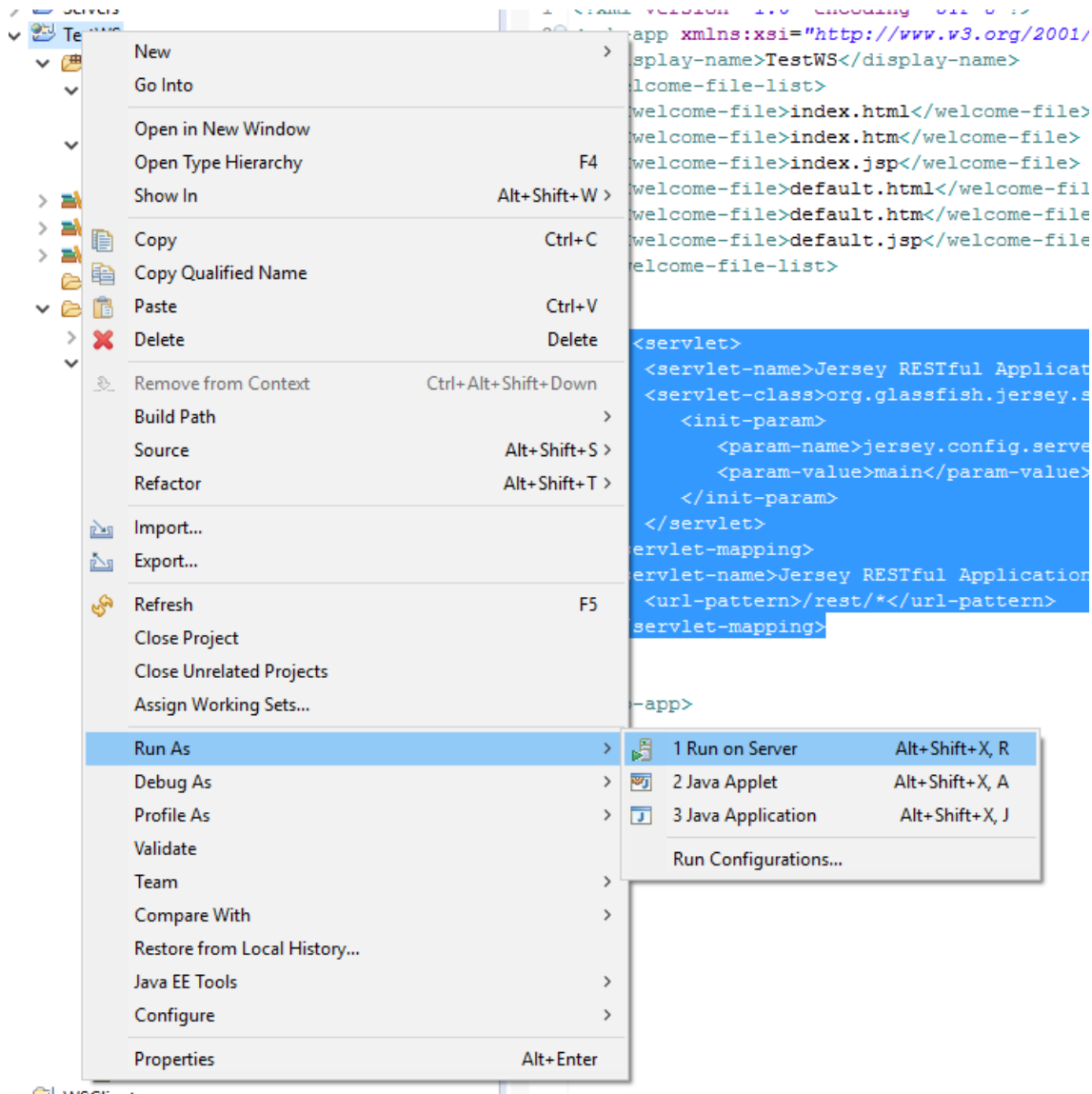
# 1<sup>ER</sup> SERVICE WEB

- Créer la classe TestService de l'exemple
- Dans Web.xml insérer la balise servlet dans web-app

```
<servlet>
  <servlet-name>toto</servlet-name>
  <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
  <init-param>
    <param-name>jersey.config.server.provider.packages</param-name>
    <param-value>main</param-value>
  </init-param>
</servlet>
<servlet-mapping>
  <servlet-name>toto</servlet-name>
  <url-pattern>/rest/*</url-pattern>
</servlet-mapping>
```

- Main : nom du package de base qui contient les WebService
- /rest/\* : redirection d'URL

# LANCER LE PROJET



# TEST

## ○ Installer Advanced Rest Client sur Chrome

Request

Socket

Projects

Saved



History

Settings

About

RATE THIS APPLICATION ▼

DONATE

[Unnamed]  Save  Open

>

☐ GET ☒ POST ☐ PUT ☐ PATCH ☐ DELETE ☐ HEAD ☐ OPTIONS ☐ Other

Raw Form

Headers

Content-Type

application/json

×

ADD NEW HEADER

Raw Form Files (0)

Payload

ADD NEW VALUE Values from here will be URL encoded!

test2

test3

×

Clear

Send

Status: 200: OK ? Loading time:689ms

Response headers (4)

Request headers (6)

Redirects (0)

Server: Apache-Coyote/1.1

Content-Type: application/json

Content-Length: 43

Date: Thu, 14 Apr 2016 17:03:59 GMT

Raw JSON

Response

COPY TO CLIPBOARD SAVE AS FILE

```
{
  "nom": "Param"
  "prenom": "test2=test3"
}
```

# URL

- URL : <http://localhost:8080/TestWS/rest/MonService/helloWorld>
  - TestWS : nom projet
  - Rest : définit dans le Web.xml
  - MonService : @Path de la classe
  - helloWorld : @Path de la méthode

```
@Path("/MonService")

public class TestService {

    @Path("/helloWorld")
    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String hello() {
        System.out.println("HelloWorld !!");
        return "HelloWorld";
    }
}
```

# TP

- Faire fonctionner son 1<sup>er</sup> WebService



# GET POST

- Requête de type GET
  - Pour extraire des informations
  - Intègre les paramètres à l'Url
  - [www.exemple.com/hello?param1=bob&param2=key2](http://www.exemple.com/hello?param1=bob&param2=key2)
- Requête de type POST
  - Pour poster des informations secrètes
  - Transmis dans le corps de la requête

# CODE RETOUR

- 100 – 199 : Information
  - 100 : Continuer d'envoyer la suite de la requête.
- 200-299 : Succès
- 300-399 : Re-direction
- 400-499 : Erreur client
  - 401 : Unauthorized
  - 404 : Not found
- 500-599 : Erreur serveur
  - 503 : Service Unavailable

# JSON

- JSON (*JavaScript Object Notation*)

- Dérivé de la notation des objets du langage JavaScript
- 2 types d'élément
  - Clé / valeur
  - Liste ordonnée

- Avantages

- Peu verbeux, ce qui le rend lisible aussi bien par un humain que par une machine
- Facile à apprendre, car sa syntaxe est réduite et non extensible
- Ses types de données sont connus et simples à décrire.

# JSON

## ○ A quoi cela ressemble ?

// Succès

```
{ "results":  
  [  
    {  
      "ville": "Saint-Ouen",  
      "cp": 93400  
    },  
    {  
      "ville": "La Plaine-Saint-Denis",  
      "cp": 93210  
    },  
    {  
      "ville": "Levallois-Perret",  
      "cp": 92300  
    }  
  ],  
  "nbr": 3  
}
```

// Echec

```
{  
  "errors": {  
    "message": "Aucun terme trouve",  
    "code": "2"  
  }  
}
```

# JSON OUTILS

- Notepad++
  - JSON Viewer
- Chrome
  - JSON Formatter
- Librairie GSON pour gagner du temps.
  - Permet de sérialiser/désérialiser du JSON
  - <https://sites.google.com/site/gson/gson-user-guide>

# GSON FONCTIONNEMENT

- Créer les beans de réception
- Parser le résultat

```
//Création de l'objet
```

```
private Gson gson = new Gson();
```

```
//Java -> Json
```

```
String jsonResult = gson.toJson(result);
```

```
//JSON -> Java (Parser 1 objet)
```

```
VilleBean result = gson.fromJson(monStringJson, VilleBean.class);
```

```
//JSON -> Java (Parser une ArrayList typée)
```

```
ArrayList<VilleBean> list = gson.fromJson((String | InputStreamReader),  
    new TypeToken<ArrayList<VilleBean>>() {}.getType());
```

# RETOURNER DU JSON

```
@Path("/helloWorldJson")
@GET
@Produces(MediaType.APPLICATION_JSON)
public Response getEleve() {
    // Pour savoir que la méthode a été appelé
    System.out.println("/helloWorldJson");
    Eleve eleve = new Eleve("bob", "john");
    Gson gson = new Gson();
    // On convertit l'objet en JSON
    String json = gson.toJson(eleve);

    return Response.status(200).entity(json).build();
}
```

# TEST

>

☒ GET ☐ POST ☐ PUT ☐ PATCH ☐ DELETE ☐ HEAD ☐ OPTIONS ☐ Other

Raw Form Headers

[ADD NEW HEADER](#)

Clear Send

Status: 200: OK ? Loading time:635ms

Response headers (4)	Request headers (5)	Redirects (0)
<p>Server: Apache-Coyote/1.1</p> <p>Content-Type: application/json</p> <p>Content-Length: 29</p> <p>Date: Thu, 14 Apr 2016 17:30:54 GMT</p>		

Raw JSON Response

[COPY TO CLIPBOARD](#) [SAVE AS FILE](#)

```
{  
  "nom": "bob"  
  "prenom": "john"  
}
```



# TP

- Créer une classe EleveBean avec nom et note.
- Ajouter une méthode au WebService retournant un élève sous forme JSON

# LIRE UN PARAMÈTRE GET

```
/*  
http://localhost:8080/TestWS/rest/MonService/helloWorldJsonWithParam?name=toto  
*/  
  
@Path("/helloWorldJsonWithParam")  
@GET  
@Produces(MediaType.APPLICATION_JSON)  
public Response askWrite(@QueryParam("name") String nom) {  
    System.out.println("/ helloWorldJsonWithParam ");  
    Eleve eleve = new Eleve(nom, 12);  
    Gson gson = new Gson();  
  
    return Response.status(200).entity(gson.toJson(eleve)).build();  
}
```

# RETOURNER ET RECEVOIR DU JSON

```
@POST
@Path("/helloWorldAllJson")
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.APPLICATION_JSON)
public Response getEleveWithParam(String recu) {
    System.out.println("/helloWorldAllJson \njson:" + recu);
    Gson gson = new Gson();
    EleveBean eleve = gson.fromJson(recu, EleveBean.class);
    //Traitement des données
    return Response.status(200).entity(gson.toJson(eleve)).build();
}
```

# RETOURNER ET RECEVOIR DU JSON

The screenshot displays the ARC REST client interface. The top bar is blue with the 'ARC' logo and a 'Request' tab. The left sidebar contains navigation options: 'HTTP request' (selected), 'Socket', 'History', 'Saved', and 'Projects'. The main area is divided into sections for configuring the request. The 'Method' is set to 'POST' and the 'Request URL' is 'http://localhost:8080/TestWS/rest/MonService/helloWorldAllJson'. The 'Body' tab is selected, showing a JSON payload: 

```
{  "nom": "bob",  "prenom": "john"}
```

. Below the request configuration, the response status is '200 OK' with a duration of '4.98 ms'. The response body is a JSON object: 

```
{  "nom": "bob_ok",  "prenom": "john_ok",  "id": 0}
```

. Red circles highlight the 'POST' method, the 'Request URL', the 'Body' tab, and the request JSON payload.

ARC Request

HTTP request

Method: POST

Request URL: http://localhost:8080/TestWS/rest/MonService/helloWorldAllJson

SEND

Parameters

Headers

Body

Variables

Body content type: application/json

Editor view: Text input

```
{  "nom": "bob",  "prenom": "john"}
```

200 OK 4.98 ms

DETAILS

```
{  "nom": "bob_ok",  "prenom": "john_ok",  "id": 0}
```

# TP

- Créer un webservice qui reçoit un élève, et le retourne en ayant incrémenté sa note

# GÉNÉRER LE CATALOGUE

- Jersey permet de générer la liste des méthodes disponibles de votre Web Service : le **WADL**
- `http://localhost:8080/TestWS/rest/application.wadl`

# CLIENT D'UN WEBSERVICE

- Chrome, firefox...
- Application Android
- Site Web
- Webservice
- Code Java

# FAIRE UNE REQUÊTE GET

## ○ OkHttp

- <http://square.github.io/okhttp/>
- Télécharger le jar sur le site pour utiliser la librairie
- Attention : *"You'll also need Okio, which OkHttp uses for fast I/O and resizable buffers."*

```
public static String sendGetOkHttpRequest(String url) throws Exception {  
    System.out.println("Url : " + url);  
    OkHttpClient client = new OkHttpClient();  
    //Création de la requete  
    Request request = new Request.Builder().url(url).build();  
    //Execution de la requête  
    Response response = client.newCall(request).execute();  
    //Analyse du code retour  
    if (response.code() < 200 || response.code() > 299) {  
        throw new Exception("Réponse du serveur incorrect : " + response.code());  
    }  
    else {  
        //Résultat de la requete.  
        return response.body().string();  
    }  
}
```



# TP GET

- Créer un nouveau projet « client » qui appelle la méthode helloWorldJson et parse son résultat en Elève

# FAIRE UNE REQUÊTE POST

```
public static String sendPostOkHttpRequest(String url, String paramJson )
    throws Exception {
    System.out.println("Url : " + url);
    OkHttpClient client = new OkHttpClient();
    MediaType JSON = MediaType.parse("application/json; charset=utf-8");
    //Corps de la requête
    RequestBody body = RequestBody.create(JSON, paramJson);

    //Création de la requete
    Request request = new Request.Builder().url(url).post(body).build();
    //Execution de la requête
    Response response = client.newCall(request).execute();
    //Analyse du code retour
    if (response.code() < 200 || response.code() > 299) {
        throw new Exception("Réponse du serveur incorrect : " + response.code());
    }
    else {
        //Résultat de la requete.
        return response.body().string();
    }
}
```

# TP POST

- Appeler la méthode `helloWorldAllJson` et lui transmettre l'élève reçu dans le POST

# TP PROJET

- Créer une boîte de dépôt de message.