CHAPITRE 01
BASES DE LA PROGRAMMATION JAVA

https://bit.ly/2S3h8nO

CONTENU DU COURS

CHAPITRE 01 : Bases de la programmation Java

CHAPITRE 02 : Programmation orientée objet en Java

CHAPITRE 03 : GUI & Programmation événementielle

CHAPITRE 04 : Android

EVALUATION

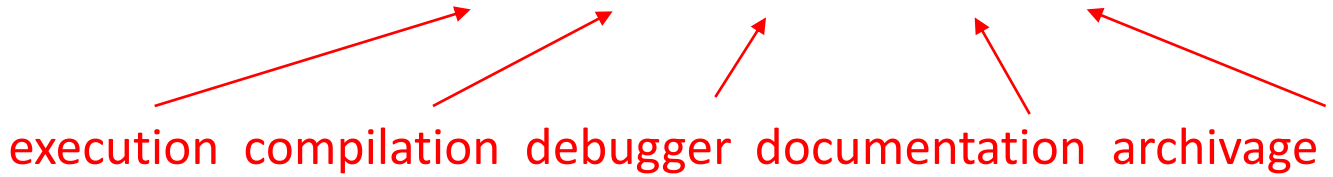
Contrôle Continu : exercices de programmation réalisés en classe

Contrôle terminal : sur table

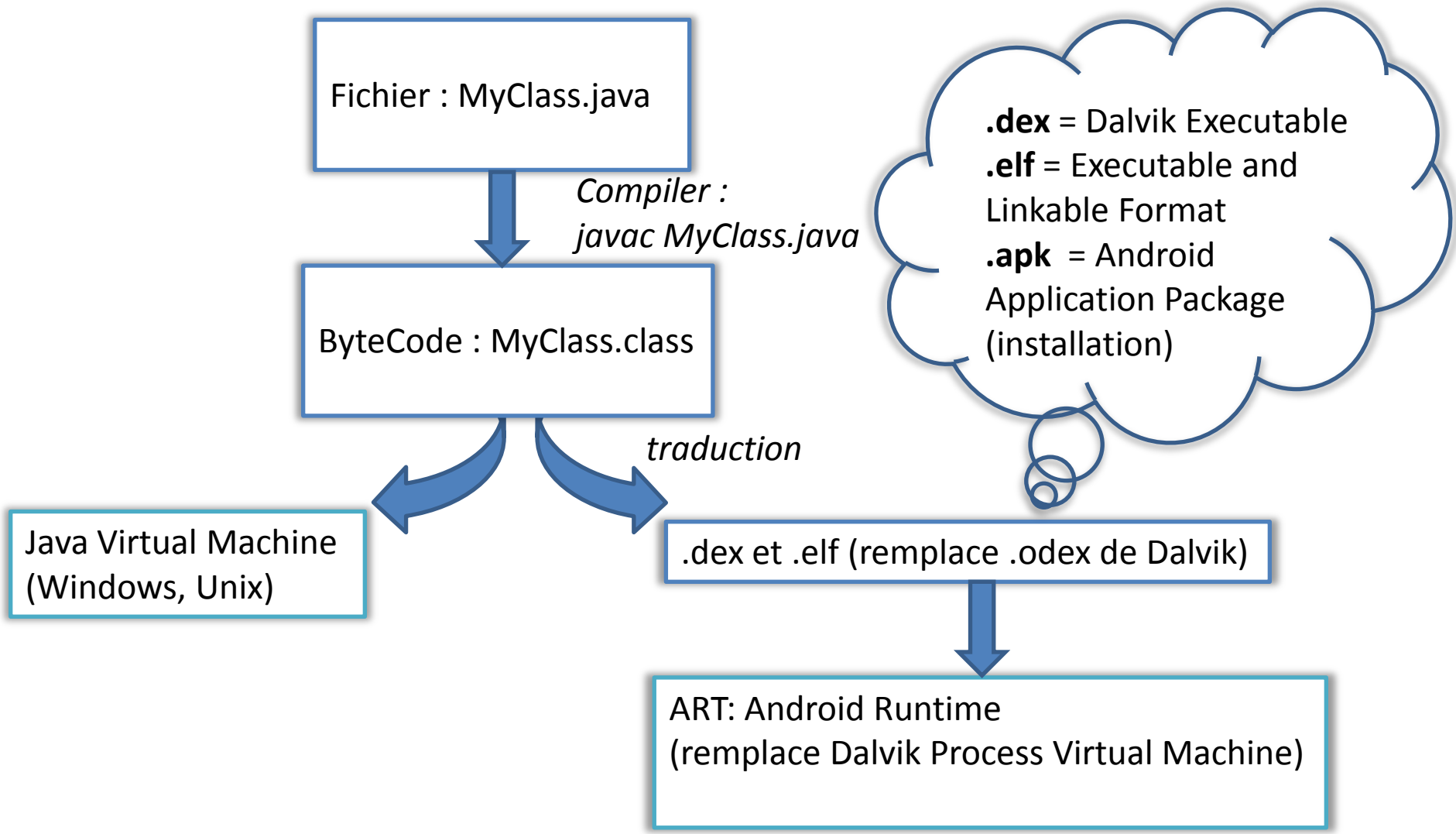
1.1 – LE LANGAGE JAVA

Langage

- Créé par Sun Microsystems en 1995.
- Orienté Objet (classes, objets, héritage, abstraction, polymorphisme…etc.).
- Langage simple (pas d'héritage multiple, pas de pointeurs).
- Exécutable portable sur différentes plateformes, Archi+OS (Windows, UNIX, Linux, Android…etc.).
- Des bibliothèques de classes ou des *packages : API Application Programming Interface (+4000 classes)*
- Ensemble d'outils : *java, javac, jdb, javadoc, jar, …*



Principe


Fichier : MyClass.java
Compiler :
javac MyClass.java
ByteCode : MyClass.class
traduction
Java Virtual Machine
(Windows, Unix)
.dex et .elf (remplace .odex de Dalvik)
ART: Android Runtime
(remplace Dalvik Process Virtual Machine)
.dex = Dalvik Executable
.elf = Executable and Linkable Format
.apk = Android Application Package (installation)

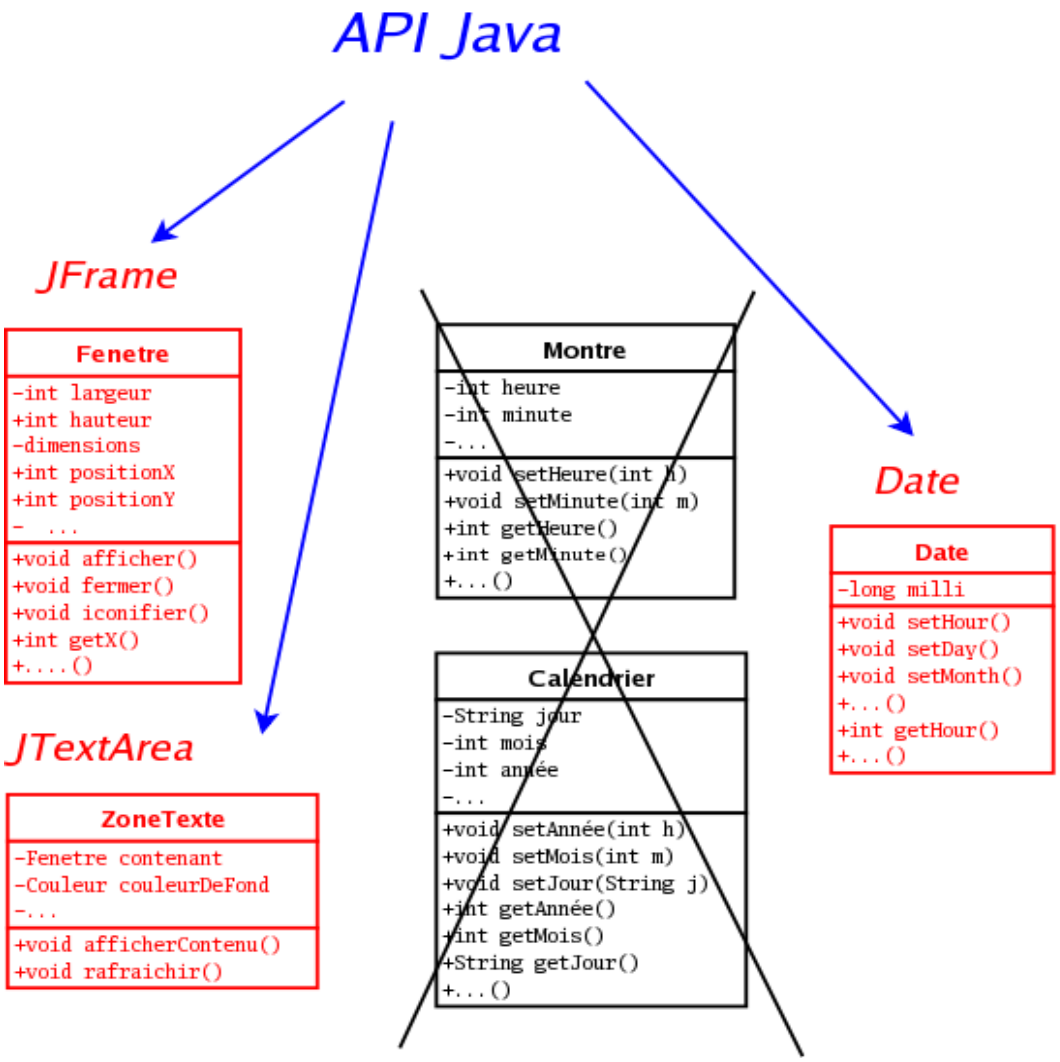
Avantages de JAVA

- Ecrire une fois, exécuter partout !
- Sécurité.
- Exécution dans un navigateur Web (Applet).
- Gestion automatique de la mémoire (Automatic Garbage Collector)
- Programmation modulaire.
- Lisibilité du code.
- Code compact (beaucoup est dans la JVM).
- L'API (Application Programming Interface).
- Réseaux, interfaces, son, pont avec les bases de données en natif.

Inconvénients de JAVA

- Compilé en bytecode puis (parfois) Interprété : lenteur.
- Nécessite une JVM pour fonctionner.
- La gestion de la mémoire est inefficace dans la plupart des cas.
- Difficulté face aux applications gourmandes en mémoire.
- Moins de mécanismes objet que C++ pourtant plus ancien.

Programmer en JAVA



Création d'un programme Java :

- Identifier les objets nécessaires et leurs relations
- Spécifier leurs types / leurs classes (données, comportement)
- Chercher dans l'API Java si des classes peuvent répondrent aux besoins (tout ou en partie)
- Création des classes nécessaires (en s'appuyant sur l'API)
- **Programmer en Java, c'est aussi savoir chercher dans l'API Java.**

Conventions

```
public class Sample {
   int ivar1;
   int ivar2;
   Sample(int i, int j) {
      ivar1 = i;
      ivar2 = j;
   }

   int emptyMethod() {}
   ...
}
```

- Nom de classe en majuscule
- Nom de méthodes et variables en minuscule
- Indentation

Types de programmes

Application

- Programme autonome (stand-alone)
- L'application doit posséder une classe principale ayant une méthode signature :

public static void main**(String[] args)**

- C'est la méthode exécutée/interprétée lors de l'exécution, le point d'entrée.
- main est static : ne nécessite pas de création d'objet pour être invoquable (méthodes statiques sont également appelées méthodes de classe).

Applet

- Programme exécutées dans l'environnement d'un navigateur.
- Le bytecode est sur un serveur http, il est chargé via une page html.
- Pas de méthode main() , mais les méthodes obligatoires :

init(), start(), stop(), paint(), destroy()

Mobile Apps

Activity, Service…etc.

Structure d'une application

- Lorsqu'on utilise une classe prédéfinie, il faut indiquer au compilateur où est le code correspondant : **import**.
- On peut importer une classe (Date) ou toutes les classes d'une collection.
- Lors de la création d'une application, une des classes **public** doit porter le même nom que le fichier et comporter une méthode *main.*
- Les commentaires commençant par /** sont destinés à *javadoc*. Les commandes *javadoc* commence par « @ ».
- Tout programme, toutes les classes, les attributs et méthodes sont à commenter afin de générer une documentation via *javadoc*.

```
import java.util.Date;
import javax.swing.*;

/** Le premier programme Java
 * @author X Y
 * @version 0
 */
public class MyApplication {

  /* Ici sont définis attributs et
   * méthodes.
   */

  public static void main(String[] args) {
    /* corps de la méthode main */
  }
}
```

Hello World!

```
public class HelloWorld {

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

}
```

- La fonction **main** est une méthode **static** → indépendante de l'instance → commune à toute les instances,
- **main** peut être appelée sans qu'aucune instance ne soit créée :

HelloWorld.main();

- **System** est une classe avec des attributs static (toujours pas besoin d'instance)
- **out** est une instance **static** de quelque chose (**PrintStream**) qui se trouve être une donnée de l'objet **System** (voir doc. de **System**).
- **System.out** réfère donc à un objet auquel on peut envoyer des messages.
- **println** est une méthode **public** de l'objet **out**.

Hello World! - Args

```
public class HelloWorldArgs {
   public static void main (String[] arguments) {
      System.out.println("Hello World! Your arguments are:");
                                    for (int i=0; i<arguments.length;i++)
                  System.out.println(i + ": " + arguments[i]);
   }
}
```

- Compilation :

```
> javac HelloWorldArgs.java
```

- Exécution :

```
> java HelloWorldArgs 1  deux  3.14
Hello World! Your arguments are:
0: 1
1: deux
2: 3.14
```

- Pas besoin d'une variable (argc) pour connaître la taille d'un tableau
- Boucle for = comme en C
- Concaténation et construction de chaînes de caractères avec le signe +

Packages

- Un package = regroupement cohérent de classes.
  - `import java.util.*;// invoque toutes les classes //du package`
  - `import java.util.ArrayList; //uniquement 1 classe`
- Pour constituer un package :

```
package mypackage;
public class MyClass{...}
```

- L'utilisation se fait soit via *import soit comme ceci :*

  mypackage**.MyClass m = new mypackage.MyClass();**
- *Mettre toutes les classes d'un même package dans un même répertoire*
- *Attention aux noms utilisés.*

  `package dir1.dir2.mypackage;`
- *Indique le chemin pour trouver les fichiers .class à partir du CLASSPATH*

API JAVA : packages standards

- java.lang : classes essentielles objet, types de base, processus
- java.util : structures de données (collections) listes, ensembles, hashtables, arbres, itérateurs
- java.awt : interface graphique (Abstract Window Toolkit) fenêtres, boutons, événements...
- java.io : entrées / sorties flot de données provenant de fichier, buffer, ou pipe
- java.net : réseau URL, sockets
- java.rmi : objets distribués (Remote Method Invocation)
- java.sql : JDBC (Java Data Base Connectivity) connexion à une base de données relationnelle envoi de requêtes SQL, récupération des résultats
- java.beans : composants logiciels, pièces logicielles autonomes pouvant être contrôlées dynamiquement et assemblées pour former des applications
- javax.swing : interface graphique composants d'interface de plus haut niveau que ceux de awt look and feel indépendant de l'OS exploitation du modèle MVC (Model View Controler)

Syntaxe de JAVA

- Reprend celle du C/C++
- Le ; marque la fin d'instruction
- Un bloc d'instructions est délimité par {}
- Les identificateurs sont comme en C (if, while, switch, etc.) et utilisés tel quel
- Les caractères sont codés en UNICODE
  - *16 bits au lieu de 8 : accents dans le code!*
- Les commentaires pertinents sont obligatoires.

Types de données

2 grands types de données :

- types primitifs
  - *les variables contiennent des valeurs*
  - *indépendant de l'architecture*
- objets (instances de classes)
  - *les variables contiennent des références*

Les types primitifs existent en version objet (wrapper classes) avec des méthodes associées : « même nom » de type avec la première lettre en majuscule.

Integer, Long, Float, Double, Boolean, Byte, Character, Void

Types primitifs

Type	Valeur/taille
boolean	true ou false
byte	Entier signé – 1 octet
short	entier signé – 2 octets
int	entier signé – 4 octets
long	entier signé – 8 octets
char	entier non signé – 2 octets – codage UTF-16
float	valuer flottante – 4 octets
double	valeur flottante – 8 octets

Déclaration :
int i;
byte b;
double d;

Les variables de type primitif sont toujours passées (à des fonctions) par valeur.

Pas de type pointeur.

Pas de référence.

Opérateurs

- arithmétiques : +, -, *, /
- logiques : ||, &&, !
- de comparaison : >, >=, <, <=, ==
- d'affectation : =, +=, -=, *=, /=, %=
- Opérateurs binaires : &, |, ^, ~

- Applicables aux types primitifs uniquement (sauf exception concernant String)
- Pas de surcharge d'opérateurs en JAVA

Constantes

- Déclarées avec le mot-clé `final`.

```
final int HEURES_DANS_JOUR = 24;
int jours;
int heuresDansJours;
jours = 7;
heuresDansJours = jours * HEURES_DANS_JOUR;
```

Attention à la syntaxe JAVA

- Les constantes sont en majuscules avec des _ entre les mots
- Les Variables commencent par une minuscule et utilisent une majuscule à chaque nouveau début de mot

Méthodes dans java.lang.Math

```
public static final double E;
public static final double PI;
public static double abs(double);
public static float abs(float);
public static int abs(int);
public static long abs(long);
public static native double acos(double);
public static native double asin(double);
public static native double atan(double);
public static native double atan2(double, double);
public static native double ceil(double);
public static native double cos(double);
public static native double exp(double);
public static native double floor(double);
public static native double log(double);
public static double max(double, double);
public static float max(float, float);
```

Méthodes dans java.lang.Math

```
public static int                max(int, int);
public static long               max(long, long);
public static double             min(double, double);
public static float              min(float, float);
public static int                min(int, int);
public static long               min(long, long);
public static native double      pow(double, double);
public static synchronized double random();
public static native double      rint(double);
public static long               round(double);
public static int                round(float);
public static native double      sin(double);
public static native double      sqrt(double);
public static native double      tan(double);
public static double             toDegrees(double);
public static double             toRadians(double);
```

Tableaux

- Le nom d'un tableau est une adresse.
- Principes et déclarations
  - *Les tableaux sont des collections de taille fixe d'objets de même type.*

```
double[] x = new double[10];
```

- Allocation et initialisation simultanées
  - *L'appel de new est alors inutile.*

```
double[] y = { 0.0, 0.25, 0.5, 0.75, 1.0};
```

Tableaux

- Affectation de tableaux
  - Copie de valeur

```
y[ 3 ] = 0.3;
for (int i=0; i<y.length; i++) {y[i]=0.5;}
```

  - Copie de référence

```
double[] z = y;
```

- Tableaux multidimensionnels
  - Il s'agit d'un tableau de tableaux pour lequel chaque rangée est accessible individuellement.

```
double[][] matrice_carree;
matrice_carree = new double[3][3];
matrice_carree[2] = y;
```

Flot de contrôle : if

Instruction conditionnelle :

```
if (condition)
{bloc 1}

if (condition) {bloc 1}
else {bloc 2}

if (condition 1) {bloc 1}
else if (condition 2){bloc
  2}
else {bloc N}
```

```
if (x > y) {
        int tmp = x;
        x = y;
        y = tmp;
} else x = 0;
        ...
if ((x > s1)&& (x < s2))
        y=3*x+1;
else y=0;
```

Flot de contrôle : while

Boucle tant que faire / répéter tant que

```
while (condition)
{bloc}

do {bloc}
while (condition)
```

```
// Chercher un élément nul dans un tableau
int i = 0;
while ((tab[i] != 0)&& (i<tab.length))
        i++;

System.out.println("Le premier élément nul est en "+ i);
...
int somme=1;
int i=borneSuperieure;

do{
        somme+=i
        i--
} while (i>0)
```

Flot de contrôle : for

Boucle pour – for :

```
for (expr1; expr2; expr3)
{bloc}
```

fonctionnement :

expr1
if (expr2==true){
bloc
expr3
}

```
float moyenne= 0;
// Initialisation d'un tableau d'entier
int[] tab = { 2, 5, -1, 4, 3 };

for (int i =0; i < tab.length; i++)
                        // conversion!
            moyenne+=tab[i];

moyenne /= tab.length;
System.out.println("La moyenne est "+moyenne);
// Si moyenne etait un int, la division
// serait entière
```

Flot de contrôle : switch

```
switch (nomVariable)
{
   case valeur1 : {...
      break;
   }
   ...
   case valeurn : {...
      break;
   }
   default : {...
      break;
   }
} ;
```

Attention en JAVA :

- nomVariable : de type “intégral” : boolean , char, int, long et short (et certaines classes)

Flot de Contrôle : break

- Interruption de boucle

```
while( i <= 100 ) {
   i += 10;
   if( i >= 2 * n + 1 ) {
       break;
   }
}
```

- Interruption étiquetée : sortie d'une boucle imbriquée.

```
boucle_a_interrompre:
while( i <= 100 ) {
   i += 10;
   while( i < j ) {
       i++;
       if( i >= 2 * n + 1 ) {
           break boucle_a_interrompre;
         }
   }
}
```

Flot de Contrôle : continue

- Continuation de boucle = court-circuit de la fin d'une itération

```
int x = 0;
while (x >= 10)
{
  x++;
  if (x == 5)  continue;
  document.write(x + );
}
```

Produira

```
> 1  2  3  4  6  7  8  9  10
```

Chaînes de caractères

- Initialisation, affectation et concaténation :

```
public static void main(String[] args){
    String intro = "Ce programme peut écrire ";
    String affiche;

    affiche = intro + "une chaîne de " + (50 + 2) + "caractères.";
    System.out.println( affiche );
}
```

- Sous-chaînes

```
public static void main(String[] args){
    String motConstruit = "Sous-chaîne";
    String racine = motConstruit.substring( 5 ,  motConstruit.length() - 5 );
    char tiret = motConstruit.charAt( 4 );
}
```

1.2 LES CHAÎNES DE CARACTERES

Chaînes de caractères

- Égalité entre chaînes :
  - *Égalité de valeurs : méthode equals()*

```
public static void main(String[] args){
    if( racine.equals( "chaîne" ) ) {
        System.out.println( "Égalité de valeurs" );
    }
}
```

  - ***Attention :***

```
public static void main(String[] args){
        String s1 = new String("bonjour");
        String s2 = new String("bonjour");
        if (s1 == s2)
                System.out.println("C'est égal");
        else
                System.out.println("Différent");
}
```

→ Différent !

Chaînes : saisie (clavier)

- Lecture d'une chaîne au clavier

```
import java.io.*;
public static void main(String[] args){
    InputStreamReader is = new InputStreamReader( System.in );
    BufferedReader bf    = new BufferedReader( is );

    String chaineAAnalyser = "";
    System.out.print( "Entrez une chaîne : " );
    try{
        chaineAAnalyser = bf.readLine();
    }
    catch(IOException IOException_Arg) {
        System.out.println(IOException_Arg.getMessage()) ;
    }
}
```

Chaînes : formatage

- **Sortie non formatée**
  - *Affichage sans limite sur le nombre de décimales*

```
double x = 1.0 / 3.0;
System.out.println( x );
```

Formatage des entiers

- *Création d'un format d'affichage numérique*

```
DecimalFormat df = new DecimalFormat( "0.###" );
System.out.println( df.Format( x ) );
```

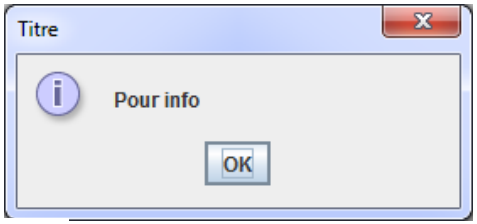
Formats :

0	*chiffre*
#	*chiffre non visible si 0 de complément*
.	*symbole décimal*
,	*séparateur de groupements*
-	*préfixe des nombres négatifs*
%	*affichage d'un pourcentage*

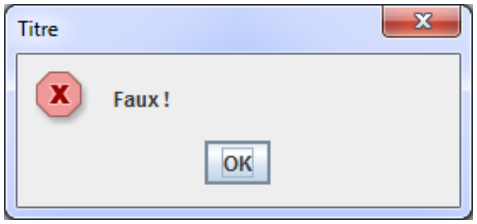
1.3 SWING

JOptionPane.showMessageDialog

```
import javax.swing.JOptionPane;

...



JOptionPane.showMessageDialog(null,"Pour info", "Titre", JOptionPane.INFORMATION_MESSAGE);

...
```




```
JOptionPane.showMessageDialog(null,"Faux !", "Titre", JOptionPane.ERROR_MESSAGE);

...
```



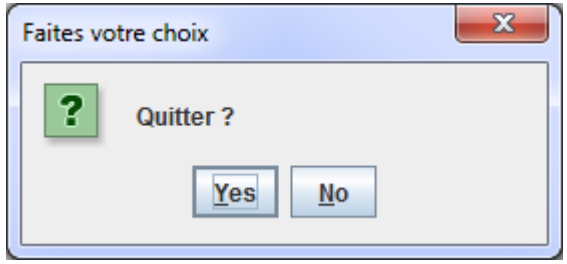

```
JOptionPane.showMessageDialog(null,"Attention !", "Titre", JOptionPane.WARNING_MESSAGE);
```

JOptionPane.showMessageDialog

```
import javax.swing.ImageIcon;
import javax.swing.JOptionPane;

public class DialogBoxApp
{
  public static void main(String[] args)
  {
    // show a joptionpane dialog using showMessageDialog
    JOptionPane.showMessageDialog(null,
        "Compte unitra désactivé",
        "UNISTRA",
        JOptionPane.INFORMATION_MESSAGE,
        new ImageIcon("logo_unistra_interieur.png")
    );

  }
}
```

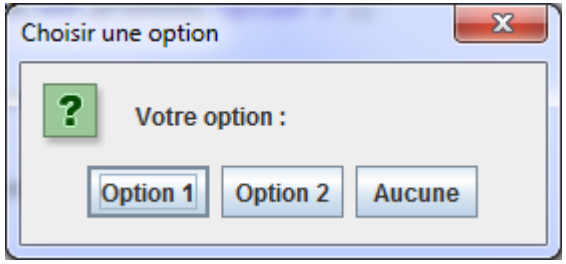
JOptionPane.showConfirmDialog

```
import javax.swing.ImageIcon;
import javax.swing.JOptionPane;

public class DialogBoxApp
{
  public static void main(String[] args)
  {
    String title = "Faites votre choix";
    String message = "Quitter ?";

    int reply = JOptionPane.showConfirmDialog(null, message, title, JOptionPane.YES_NO_OPTION);

     if (reply == JOptionPane.YES_OPTION) System.exit(0);


  }
}
```

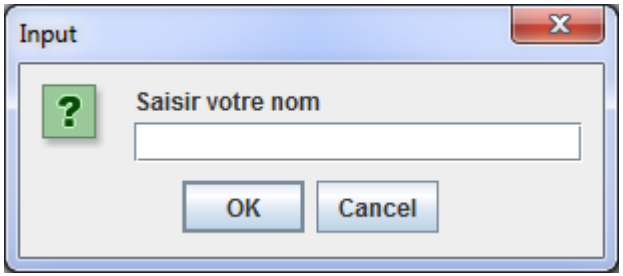


JOptionPane.showOptionDialog

```
import javax.swing.JOptionPane;

public class DialogBoxApp {
        public static void main(String[] args) {
                Object[] options = { "Option 1", "Option 2", "Aucune" };
                int choice = JOptionPane.showOptionDialog(null,
                "Votre option :",
                "Choisir une option",
                JOptionPane.YES_NO_OPTION,
                JOptionPane.QUESTION_MESSAGE,
                null,
                options,
                options[0]);

                switch (choice) {
                case 0 : {System.out.println("Option 1"); break;}
                case 2 : {System.out.println("Aucune"); break;}
                case 1 : {System.out.println("Option 2"); break;}
                }
        }
}
```



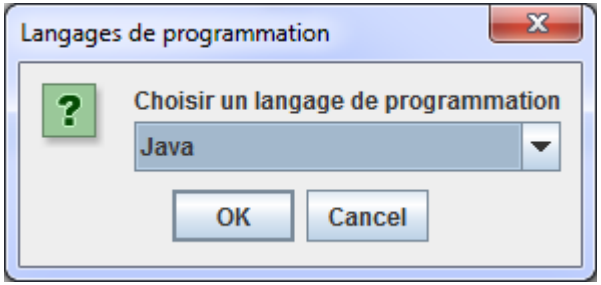
JOptionPane.showInputDialog

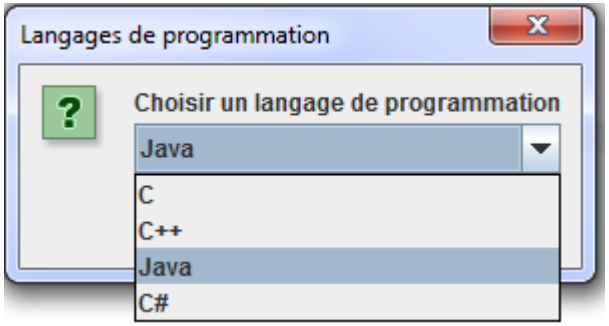



```
import javax.swing.*;

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

          String name = JOptionPane.showInputDialog(
                                                    null,
                                                    "Saisir votre nom",
                                                    "UNISTRA",
                                                    JOptionPane.WARNING_MESSAGE);
          System.out.println("Merci "+name);
          System.exit(0);
        }
}
```

JOptionPane.showInputDialog

```
import javax.swing.*;
public class DialogBoxApp {
        public static final String[] langages = { "C", "C++", "Java", "C#" };

        public static void main(String[] args) {
          String lang = (String) JOptionPane.showInputDialog(null,
             "Choisir un langage de programmation",
             "Langages de programmation",
             JOptionPane.QUESTION_MESSAGE,
             null,
             langages,
             langages[2]);

          System.out.println("Votre choix : "+ lang);
          System.exit(0);
        }
}
```





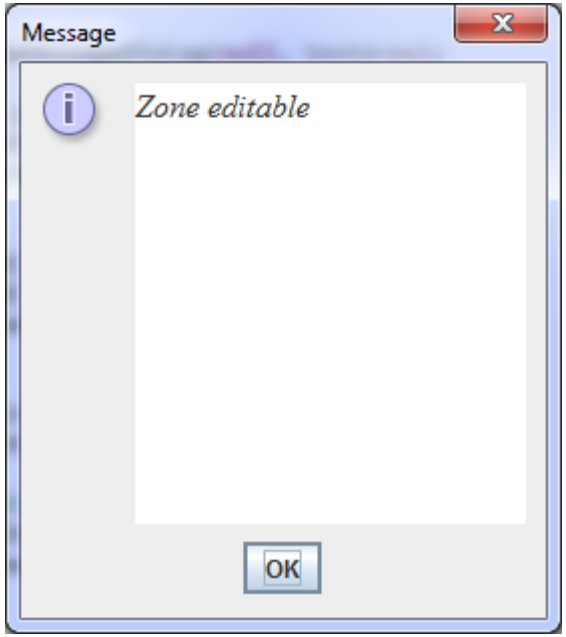
Conversion

```
String lastYearStr = "2015";

//conversion String vers int
int lastYear = Integer.parseInt(lastYearStr);
int newYear = lastYear + 1;

System.out.println("New Year = "+ newYear);

//conversion int vers String
String newYearStr = ""+newYear;

System.out.println("New Year = "+ newYearStr);
//ou
newYearStr = (new Integer(newYear)).toString();

System.out.println("New Year = "+ newYearStr);
```

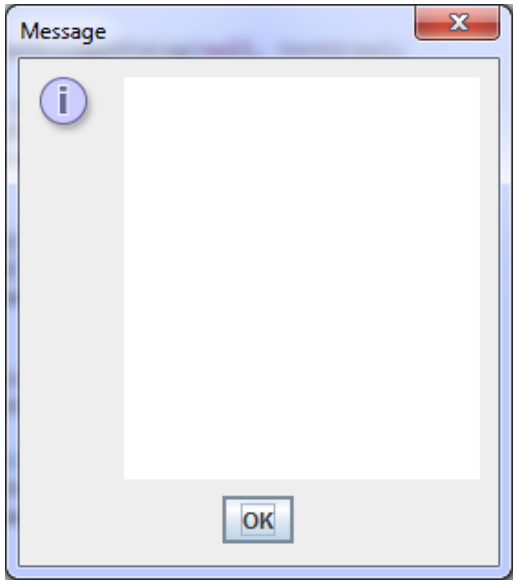
voir :
Double.parseDouble
Float.parseFloat()
…etc.

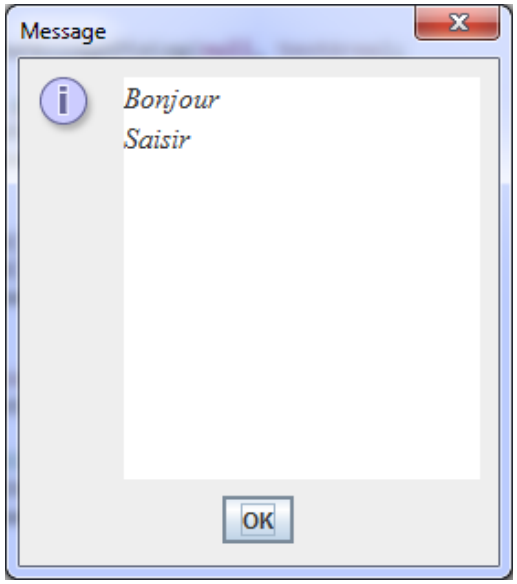
JTextArea

```
import java.awt.Font;
import javax.swing.*;
public class DialogBoxApp {

    public static void main(String[] args) {

        JTextArea textArea = new JTextArea(
        "Zone editable",10,10
        );
        textArea.setFont(new Font("Serif", Font.ITALIC, 16));
        textArea.setLineWrap(true);
        textArea.setWrapStyleWord(true);
```



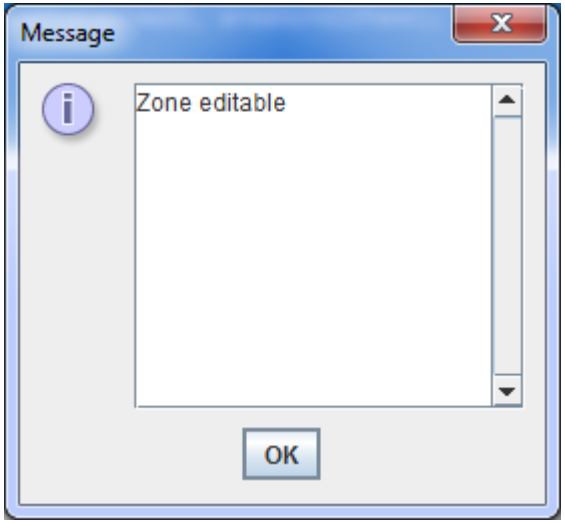
```
        //Afficher
        JOptionPane.showMessageDialog(null, textArea);

        //Remplacer ou affacer contenu
        textArea.setText("");
        JOptionPane.showMessageDialog(null, textArea);

        //Ajouter text
        textArea.append("Bonjour\n");
        textArea.append("Saisir\n");
        JOptionPane.showMessageDialog(null, textArea);

        //get
        String s = textArea.getText();
        System.out.println(s);

        //désactiver edition
        textArea.setEditable(false);
        JOptionPane.showMessageDialog(null, textArea);
    }// fin main
}//fin classe
```





AreaScrollPane

```
import java.awt.Font;
import javax.swing.*;
public class DialogBoxApp {

        public static void main(String[] args) {

                JTextArea textArea = new JTextArea(
                "Zone editable",10,10
                );

                JScrollPane areaScrollPane = new JScrollPane(textArea,
                        JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
                        JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);

                //Afficher
                JOptionPane.showMessageDialog(null, areaScrollPane);
        }

}
```

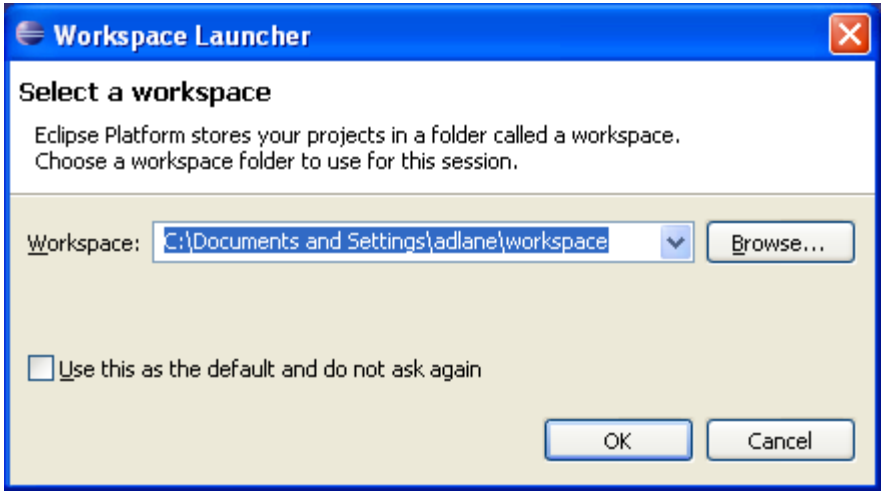


1.4 ECLIPSE IDE

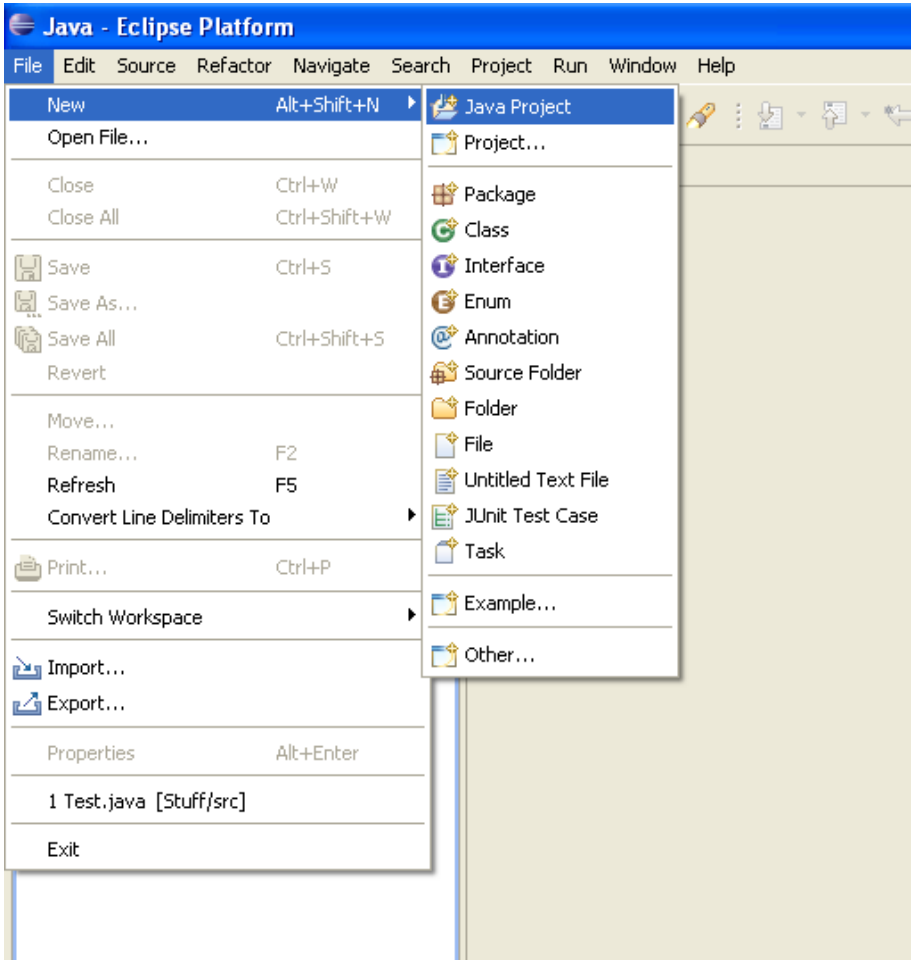
https://www.eclipse.org/downloads/

Utilisation de l'environnement Eclipse

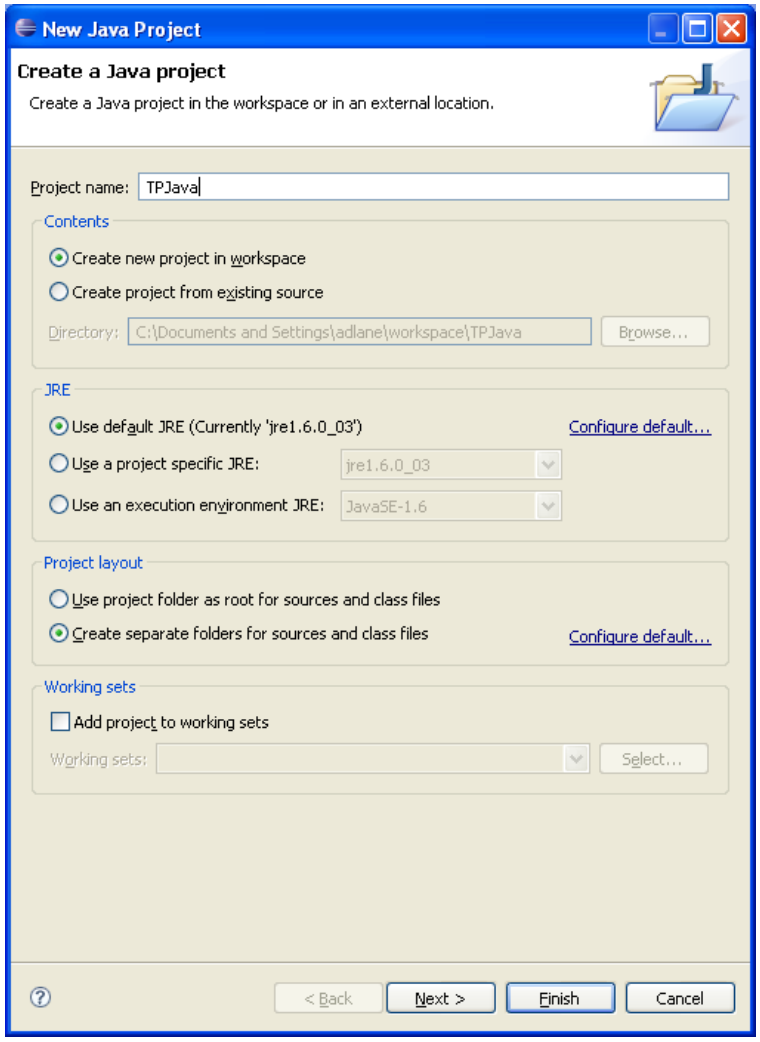
1. Connectez-vous à votre compte
2. Lancez Eclipse à partir de la liste des programmes. Puis, sélectionnez ok lorsque la boite de dialogue ci-dessous apparaît.



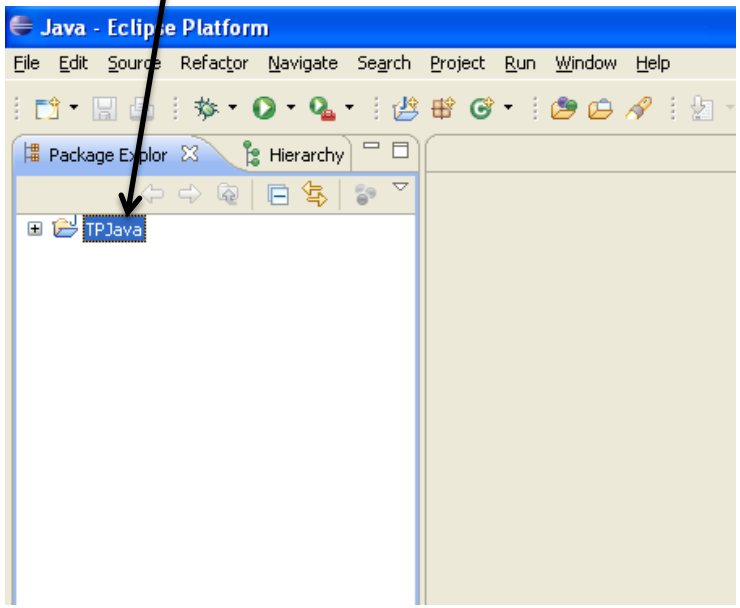
3. Ouvrez un nouveau projet Java comme indiqué dans la figure suivante :



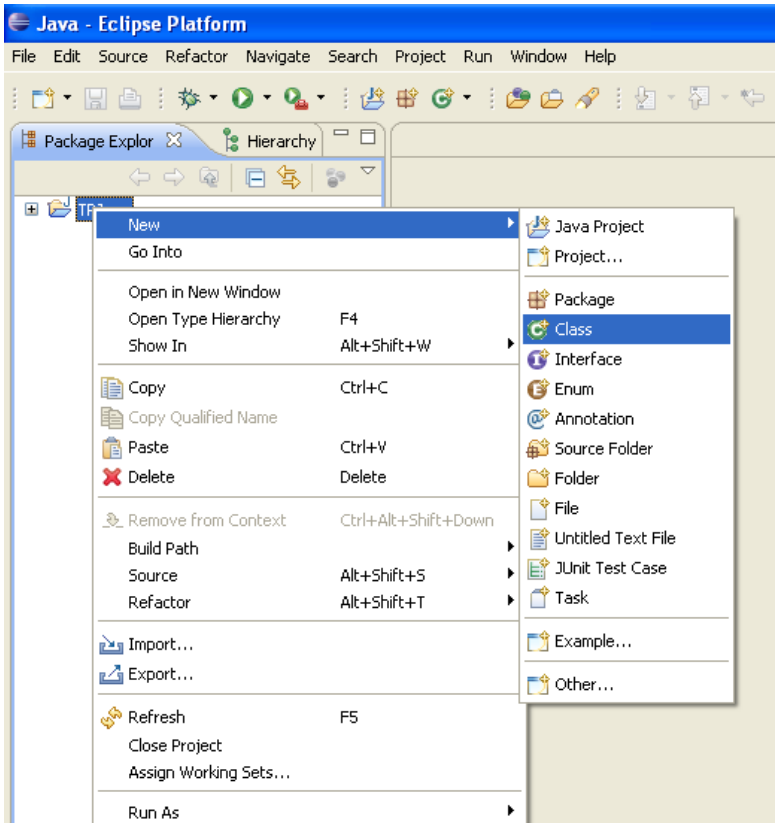
4. Nommez votre nouveau projet (par exemple TPJava), puis sélectionnez finish.



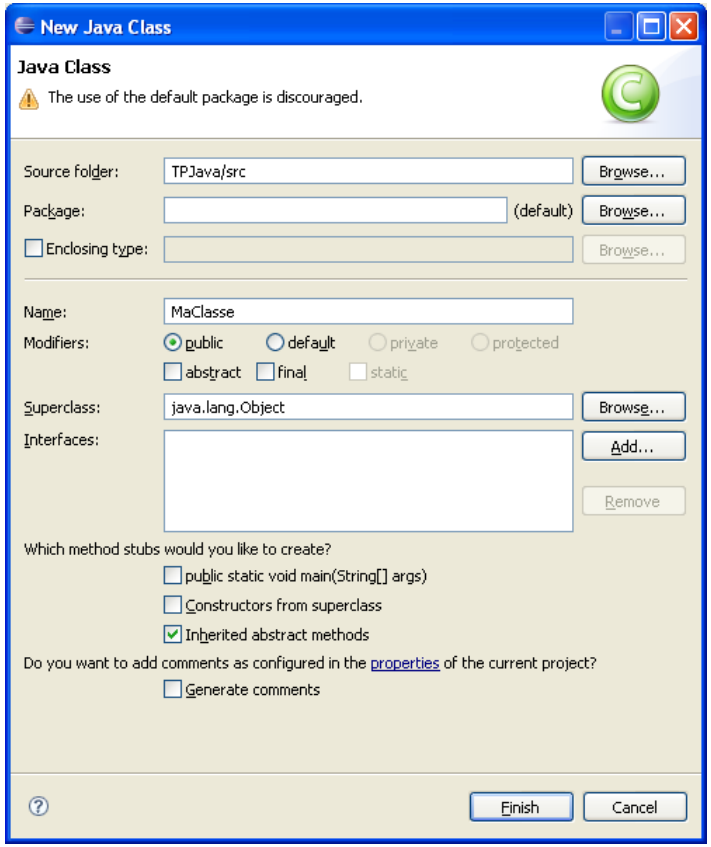
Le nouveau projet apparaît alors dans la sous-fenêtre de gauche.



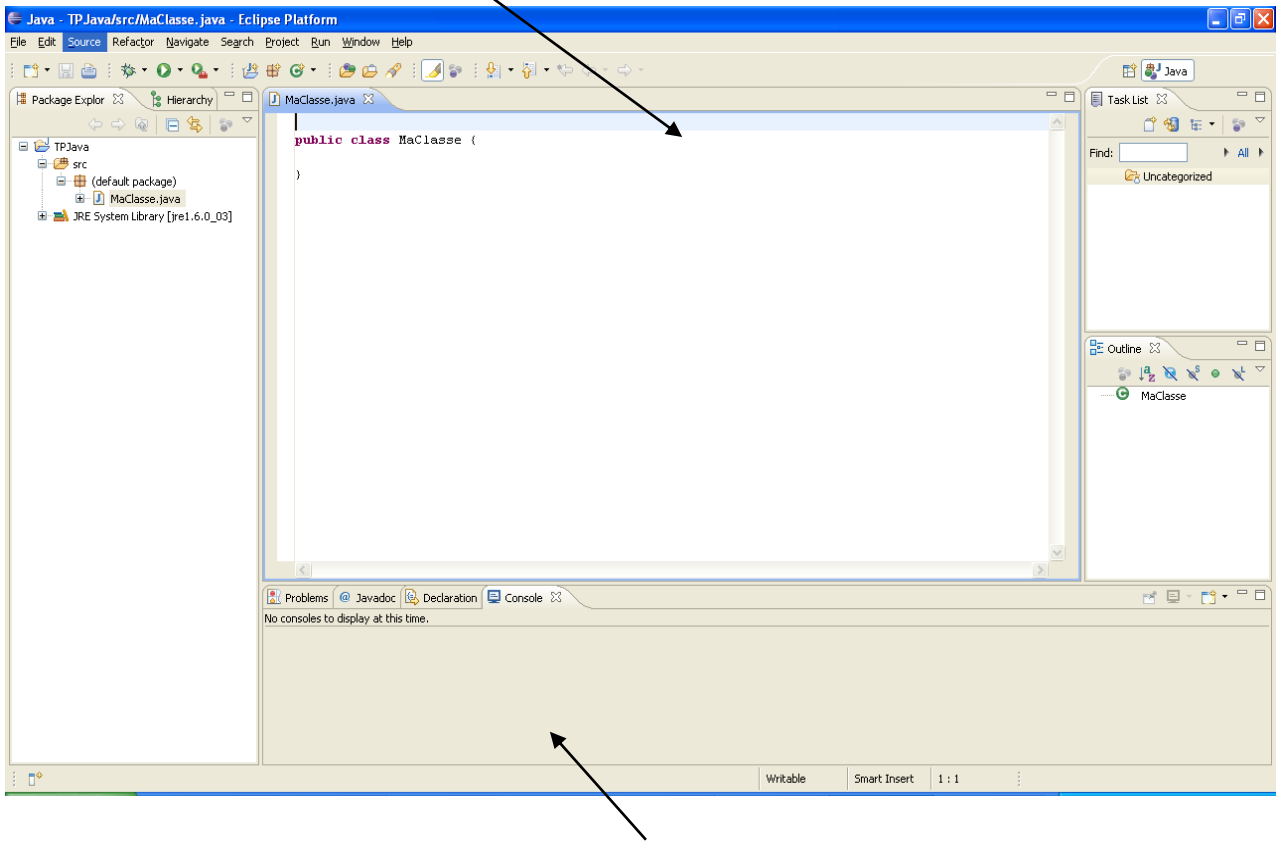
5. Avec le bouton droit de la souris, cliquez sur le nom du projet et créez une nouvelle classe :



6. Saisissez le nom de la classe que vous souhaitez créer (par exemple MaClasse) puis sélectionnez finish :

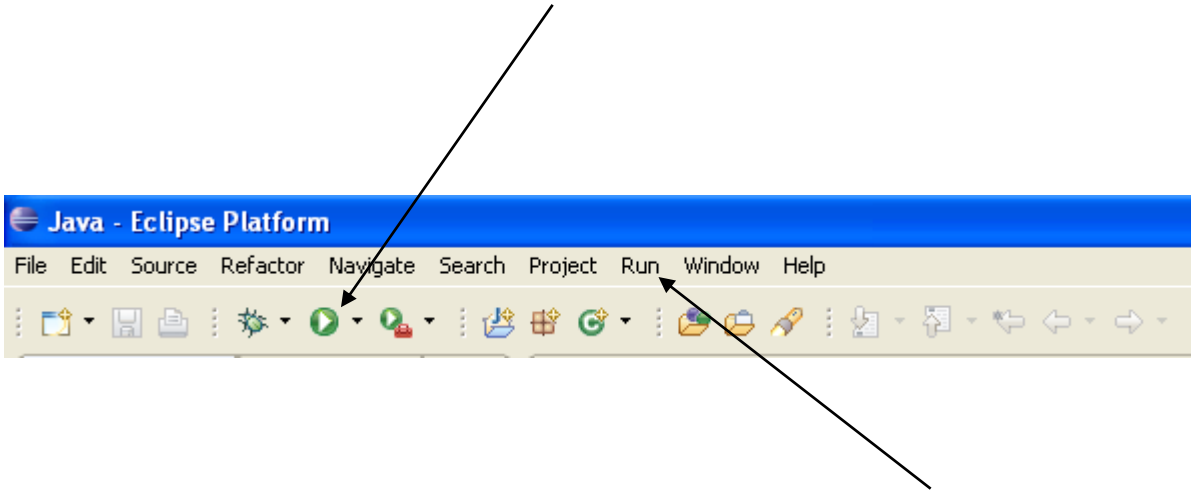


La fenêtre au milieu est l'éditeur qui vous permet de compléter la programmation de cette nouvelle classe.



Les résultats de vos programmes seront affichés dans la sous-fenêtre du bas.

Une fois une application écrite, ce bouton permet d'exécuter celle-ci.



Vous pouvez également exécuter une application à partir de **Run** dans le menu.

Remarques :
Les erreurs apparaissent dans l'éditeur au moment où vous tapez votre programme.
Votre programme est compilé automatiquement.

JAVA API

http://docs.oracle.com/javase/7/docs/api/