

Java Initiation



Facile



Normal



Difficile



Professionnel



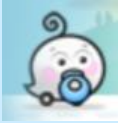
Expert

https://wiki.waze.com/wiki/Your_Rank_and_Points

- 1 - Généralités**
- 2 - Les outils et techniques de base**
- 3 - Les bases du langage Java**
- 4 - L'aspect objet du langage Java**
- 5 - Les packages de base**
- 6 - Sockets**
- 7 - Les interfaces graphiques**
- 8 - Thread**
- 9 - JUnit**
- 10 - Bibliographies**



- Les caractéristiques
- Historique
- Indépendance de la plate-forme
- API java
- Package de base
- Catégories des technologies Java
- De java 8 à java 12



Java est un langage de programmation à usage général, évolué et orienté objet dont la syntaxe est proche du C/C++.

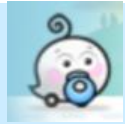
Java est notamment largement utilisé pour le développement d'applications d'entreprises et mobiles.

Quelques chiffres et faits à propos de Java en 2011 :

- 97% des machines d'entreprises ont une JVM installée
- Java est téléchargé plus d'un milliards de fois chaque année
- Il y a plus de 9 millions de développeurs Java dans le monde
- Java est un des langages les plus utilisés dans le monde
- Tous les lecteurs de Blu-Ray utilisent Java
- Plus de 3 milliards d'appareils mobiles peuvent mettre en œuvre Java
- Plus de 1,4 milliards de cartes à puce utilisant Java sont produites chaque année

<http://jmdoudoux.developpez.com/cours/developpons/java/chap-presentation.php>

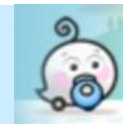
1 - Généralités -> Les caractéristiques



Java est interprété	le source est compilé en pseudo code ou bytecode puis exécuté par un interpréteur Java : la Java Virtual Machine (JVM).
Java est portable : il est indépendant de toute plate-forme	il n'y a pas de compilation spécifique pour chaque plate forme. Il est possible d'exécuter des programmes Java sur tous les environnements qui possèdent une Java Virtual Machine.
Java est orienté objet.	Chaque fichier source contient la définition d'une ou plusieurs classes. Java n'est pas complètement objet car il définit des types primitifs (entier, caractère, flottant, booléen ...).
Java est simple	le choix de ses auteurs a été d'abandonner des éléments mal compris ou mal exploités des autres langages tels que la notion de pointeurs, l'héritage multiple et la surcharge des opérateurs ...
Java est fortement typé	toutes les variables sont typées et il n'existe pas de conversion automatique qui risquerait une perte de données.
Java assure la gestion de la mémoire	l'allocation de la mémoire pour un objet est automatique à sa création et Java récupère automatiquement la mémoire inutilisée grâce au garbage collector
Java est économe	le pseudo code a une taille relativement petite car les bibliothèques de classes requises ne sont liées qu'à l'exécution.
Java est multitâche	il permet l'utilisation de threads qui sont des unités d'exécutions isolées. La JVM, elle même, utilise plusieurs threads.

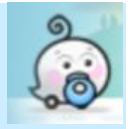
<http://jmdoudoux.developpez.com/cours/developpons/java/chap-presentation.php>

1 - Généralités -> Historique



Année	Evénements
1995	mai : premier lancement commercial du JDK 1.0
1996	janvier : JDK 1.0.1 septembre : lancement du JDC
1997	Java Card 2.0 février : JDK 1.1
1998	décembre : lancement de J2SE 1.2 et du JCP Personal Java 1.0
1999	décembre : lancement J2EE 1.2
2000	mai : J2SE 1.3
2001	J2EE 1.3
2002	février : J2SE 1.4
2003	J2EE 1.4
2004	septembre : J2SE 5.0
2005	Lancement du programme Java Champion
2006	mai : Java EE 5 décembre : Java SE 6.0
2007	Duke, la mascotte de Java est sous la licence Free BSD
2008	décembre : Java FX 1.0
2009	février : JavaFX 1.1 juin : JavaFX 1.2 décembre : Java EE 6
2010	janvier : rachat de Sun par Oracle avril : JavaFX 1.3
2011	juillet : Java SE 7 octobre : JavaFX 2.0
2012	août : JavaFX 2.2
2013	juin : Java EE 7

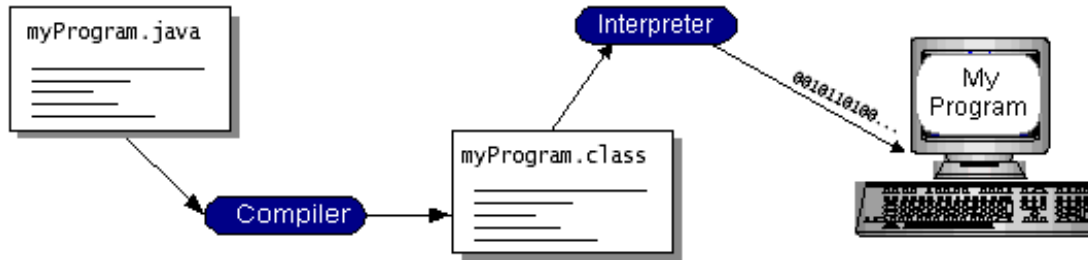
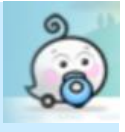
1 - Généralités -> Historique



Année	Evénements
03/2014	Java 8
09/2017	Java 9
03/2018	Java 10
09/2018	java 11
03/2019	Java 12
09/2019	Java13

De nouvelles versions de Java tous **les six mois** depuis la sortie de Java 9 en septembre 2017
=> Java 9 date de moins de deux ans, la dernière version est maintenant Java 12. (java 13 prévue le 17/09/2019)

1 - Généralités -> Indépendance de la plate-forme

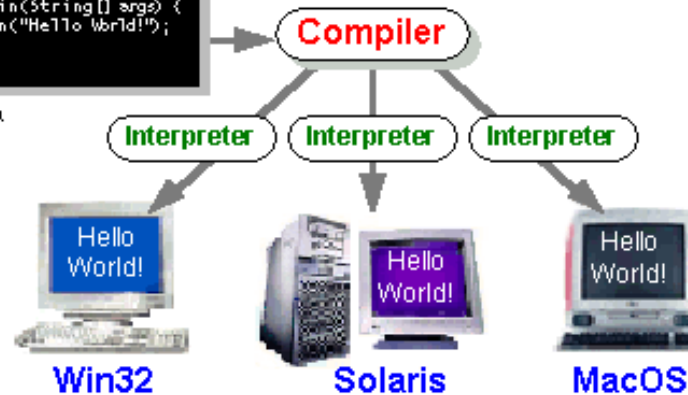


Java Program

```

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

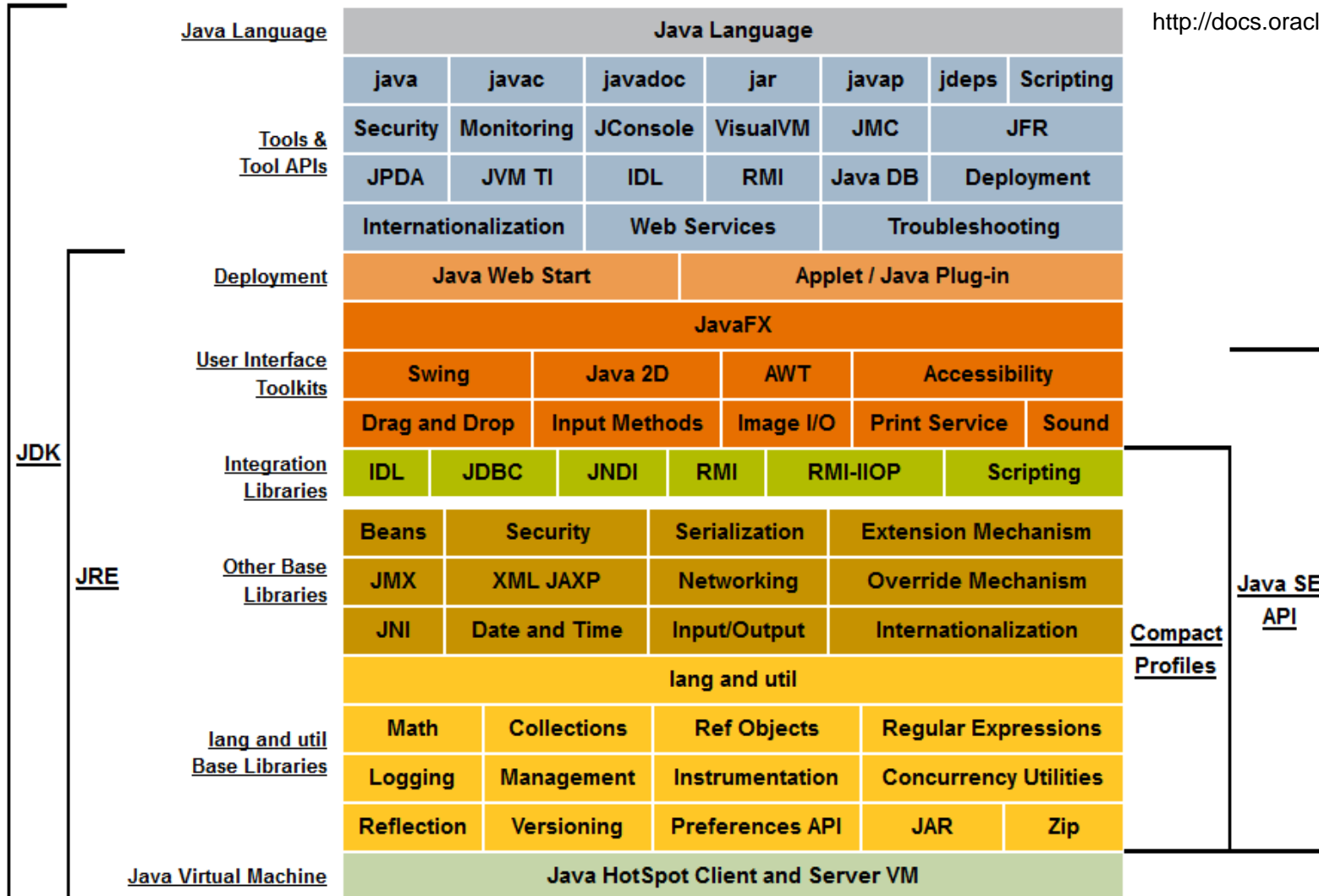
HelloWorldApp.java



1 - Généralités -> API java



<http://docs.oracle.com/javase/8/docs/>



1 - Généralités -> Package de base

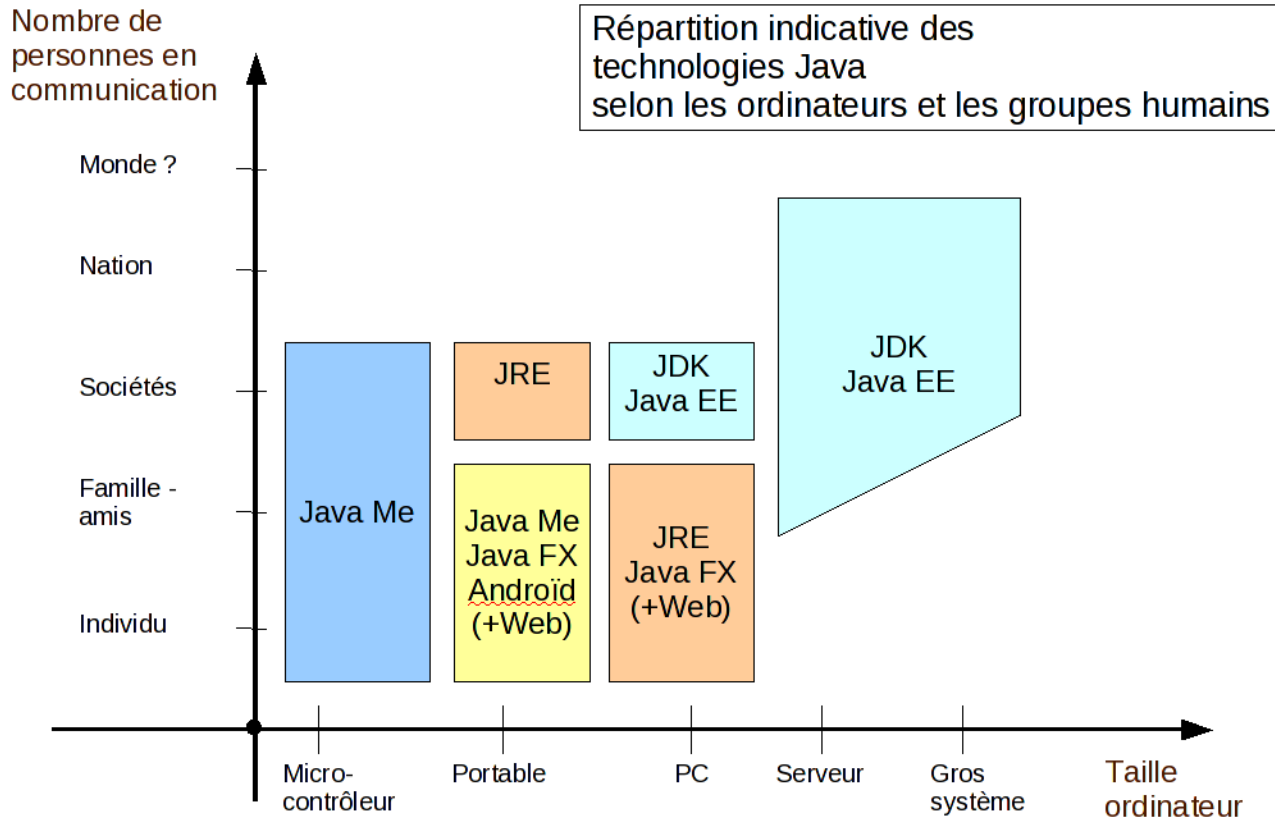


	Java 1.0	Java 1.1	Java 1.2	J2SE 1.3	J2SE 1.4	J2SE 5.0	Java SE 6	Java SE 7	Java SE 8
Nombre de packages	8	23	59	76	135	166	202	209	217
Nombre de classes	201	503	1520	1840	2990	3280	3780	4024	4240



- **Java Platform, Standard Edition (Java SE)**
 - **Java Platform, Enterprise Edition (Java EE)**
 - **JAVA PLATFORM, MICRO EDITION (JAVA ME)**
 - **JavaFX** contient des outils très divers, notamment pour les médias audios et vidéos, le graphisme 2D et 3D, la programmation Web, la programmation multi-fils etc.
 - **Le JRE** qui se compose d'une machine virtuelle, de bibliothèques logicielles utilisées par les programmes Java et d'un plugin pour permettre l'exécution de ces programmes depuis les navigateurs web.
-
- **ORACLE JAVA EMBEDDED**
 - Java DB is Oracle's supported distribution of the Apache Derby open source database.
 - **JAVA CARD TECHNOLOGY**
 - **JAVA TV**

1 - Généralités -> Catégories des technologies Java



1 - Généralités -> De java 8 à java 12

Version	Nouveauté
Java 8 03/2014	Ajout des <i>lambdas</i> , entraînant une refonte d'une partie de l'API, notamment les collections et la notion de <i>stream</i> . Les autres ajouts notables incluent les Optional , les implémentations par défaut au sein d'une interface, une refonte de l'API date, etc.
Java 9 09/2017	Le projet Jigsaw permettant de modulariser les modules chargés au sein du JDK ; Le projet Kulla visant la création d'un shell pour Java sur le format read-eval-print loop Le projet Valhalla visant une amélioration des types Java ; Un support natif du format JSON et de HTTP/253.
Java 10 03/2018	Inférence des types des variables locales (https://www.baeldung.com/java-10-local-variable-type-inference) Partage de binaire pour permettre un lancement plus rapide Activation de Graal un compilateur JIT en Java

1 - Généralités -> De java 8 à java 12

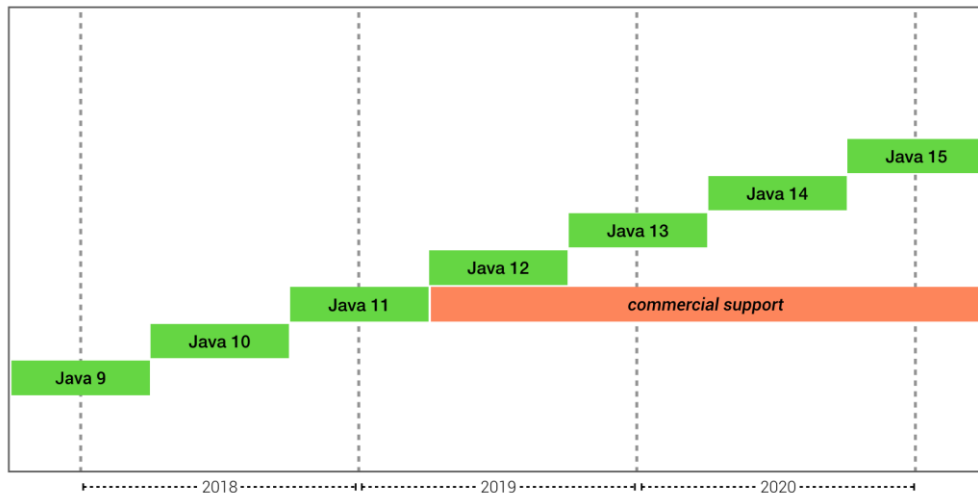
Version	Nouveauté
Java 11 09/2018	Amélioration sur les paramètres des lambda Un client HTTP plus évolué Suppression des modules CORBA et EE par défaut
Java 12 03/2019	Shenandoah: Un ramasse miette avec de courtes pauses (Expérimentale) Suite d'outils de Microbenchmark pour le code source du JDK Expressions Switch API Constants (permettre d'ajouter des informations dans les méta données dans les fichiers .class, utile pour les langages sur la JVM) Un seul portage pour l'architecture ARM 64bits Default CDS Archives (chargement des informations des classes de la JVM plus rapide) Améliorations du ramasse miette G1
Java 13 09/2019	<i>Attendu pour la fin 2019, Java 13 devrait comporter des littéraux de chaînes brutes et un outil pour emballer des applications Java autonomes.</i>

1 - Généralités -> OpenJDK

OpenJDK est l'implémentation libre (sous licence GPL) de la plateforme Java SE d'Oracle.

Elle se compose de plusieurs contributeurs dont Oracle, Red Hat, Azul Systems, SAP SE, IBM, Apple pour ne citer que ceux-là.

OpenJDK™



<http://openjdk.java.net/install/>
<https://tekcollab.imdeo.com/java-devient-payant/>



- plate-forme de développement
- la gestion des dépendances

<https://java.developpez.com/telecharger/index/categorie/336/RAD-et-EDI-Java>

2 - Les outils et techniques de base

-> plate-forme de développement



- **Netbeans** est un environnement de développement open source écrit en Java. Le produit est composé d'une partie centrale à laquelle il est possible d'ajouter des modules. netbeans.org/
- **IntelliJ IDEA** est un IDE Java commercial développé par [Jettersais](http://jetbrains.com).
<https://www.jetbrains.com/idea/>
 - **Android Studio** est un environnement de développement pour développer des applications Android. Il est basé sur IntelliJ IDEA.
- **Eclipse** est un projet open source à l'origine développé par IBM pour ses futurs outils de développement et offert à la communauté.
<http://www.eclipse.org>
- **Oracle JDeveloper** est un EDI complet et gratuit permettant de modéliser et développer des applications Java pour les plateformes J2SE, J2EE ou J2ME.



- **Junit** et **FestAssert** permettent d'écrire les tests unitaires.
- **Mockito** et **EasyMock** outils pour tester les classes en isolation
- Le trio **PMD/Findbugs/Checkstyle** qui permet de vérifier la qualité du code et publier les rapports obtenus
- **Apache CXF** est un framework open-source en langage Java, facilitant le développement de services web
- **Spring est un framework libre** qui permet de construire et de définir l'infrastructure d'une application java, dont il facilite le développement et les tests

3 - Les bases du langage Java



- La compilation à l'exécution
- Mots-clés du java
- Types prédéfinis
- Structure Lexicale
- Les commentaires
- Définition des types de base
- Les opérateurs
- Les structures de contrôle et boucle
- Les chaînes de caractères
- Les tableaux
- Les conversions de types
- La manipulation des chaînes de caractères
- Références
- Passage par valeur ou par référence

3 - Les bases du langage Java

-> La compilation à l'exécution



HelloWorld.java

javac HelloWorld.java

HelloWorld.class

java HelloWorld

```
Start Page x HelloWorld.java x
Source History
1  /**
2   * To change this license header, choose License Headers in Project Properties.
3   * To change this template file, choose Tools | Templates
4   * and open the template in the editor.
5   */
6   package helloworld;
7
8   /**
9   *
10   * @author Palermo David
11   */
12   public class HelloWorld {
13
14       /**
15       * @param args the command line arguments
16       */
17       public static void main(String[] args) {
18           System.out.println("Hello World ( Bonjour le monde )");
19       }
20
21   }
22
```

```
Output - HelloWorld (run) x
run:
Hello World ( Bonjour le monde )
BUILD SUCCESSFUL (total time: 0 seconds)
```



Compiler :

`javac -d build/classes/helloworld src/helloworld/HelloWorld.java`
crée le fichier HelloWorld.class dans build/classes/helloworld

Exécuter :

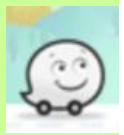
`java -classpath build\classes helloworld.HelloWorld`

On indique où se trouve les fichiers binaires (avec -classpath) ainsi que le nom de la classe qui contient le main.

<http://mescal.imag.fr/membres/vania.marangozova-martin/TEACHING/SRM1/CompilationJava.pdf>

3 - Les bases du langage Java

-> Mots-clés du java



Les mots-clés pour les **objets** :

abstract	class	enum	extends	implements	import	interface	native	package	super	this
-----------------	--------------	-------------	----------------	-------------------	---------------	------------------	---------------	----------------	--------------	-------------

Les mots-clés pour les **types** :

boolean	byte	char	double	float	int	long	short	void
----------------	-------------	-------------	---------------	--------------	------------	-------------	--------------	-------------

Les mots-clés pour les **états** :

const	false	final	new	null	static	strictfp	transient	true	volatile
--------------	--------------	--------------	------------	-------------	---------------	-----------------	------------------	-------------	-----------------

Les mots-clés pour les **modificateurs** :

private	protected	public
----------------	------------------	---------------

Les mots-clés pour les **boucles** :

continue	do	for	goto	while
-----------------	-----------	------------	-------------	--------------

Les mots-clés pour les **branchements** :

assert	break	case	default	else	if	instanceof	return	switch	synchronized
---------------	--------------	-------------	----------------	-------------	-----------	-------------------	---------------	---------------	---------------------

Les mots-clés pour les **exceptions** :

catch	finally	throw	throws	try
--------------	----------------	--------------	---------------	------------

3 - Les bases du langage Java

-> Types prédéfinis : numériques



Type	Description	Taille	Valeur minimale	Valeur maximale
byte	Tout petit entier signé	1	-2^7 ou -128	2^7-1 ou 127
short	Petit entier signé	2	-2^{15} ou -32768	$2^{15}-1$ ou 32767
int	Entier signé	4	-2147483648	2147483647
			-2^{31} ou -2147483648	ou $2^{31}-1$
long	Grand entier signé	8	-2^{63} ou -9223372036854770000	$2^{63}-1$ ou 9223372036854770000
float	Nombre à virgule flottante	4	$-3,40282347 \times 10^{38}$	$-1,40239846 \times 10^{-45}$
			$1,40239846 \times 10^{-45}$	$3,40282347 \times 10^{38}$
double	Nombre à virgule flottante double précision	8	$-1,79769313486231570 \times 10^{308}$	$-4,94065645841246544 \times 10^{-324}$
			$4,94065645841246544 \times 10^{-324}$	$1,79769313486231570 \times 10^{308}$

☞ La taille est fixe, indépendante de la plate-forme

3 - Les bases du langage Java

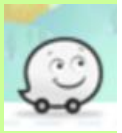
-> Types prédéfinis : non-numériques



Type	Description
boolean	Les variables de type <i>boolean</i> sont des variables qui ne peuvent contenir que les valeurs : <i>true</i> ou <i>false</i> . il n'est pas possible en Java de substituer un nombre entier à une expression conditionnelle comme en C ou C++
char	Les variables de type char contiennent des valeurs correspondant à des caractères du standard UNICODE. Les éléments de type char occupent 2 octets de mémoire chacun.
void	type vide

3 - Les bases du langage Java

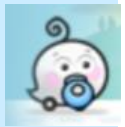
-> Structure Lexicale



- Format libre : blanc, espace, tabulation ignorés
- Jeu de caractère Unicode : ASCII sur 16 bits
- Commentaires
 - *// commentaire sur une ligne*
 - */* commentaire sur plusieurs lignes */*
 - */** ... */ pour javadoc*
- Identificateurs
 - Noms de variables ou de classes
 - Commençant par une lettre, suivi de lettres et chiffres

3 - Les bases du langage Java

-> Les commentaires : javadoc



Javadoc est un outil développé permettant de créer une documentation d'API en format HTML depuis les commentaires présents dans un code source en Java (voir aussi doxygen).

Attribut et syntaxe	Dans un commentaire de ...	Description
@author <i>auteur</i>	classe	Nom de l'auteur de la classe.
@version <i>version</i>	classe	Version de la classe.
@deprecated <i>description</i>	classe, constructeur, méthode, champ	Marquer l'entité comme obsolète (ancienne version), décrire pourquoi et par quoi la remplacer.
@see <i>référence</i>	classe, constructeur, méthode, champ	Ajouter un lien dans la section "Voir aussi".
@param <i>description de l'id</i>	constructeur et méthode	Décrire un paramètre de méthode.
@return <i>description</i>	méthode	Décrire la valeur retournée par une méthode.
@exception <i>description du type</i>	constructeur et méthode	Décrire les raisons de lancement d'une exception du type spécifié (clause throws).

3 - Les bases du langage Java

-> Les commentaires : javadoc



```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package helloworld;

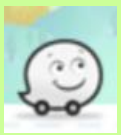
/**
 * Premier classe java
 * @author Palermo David
 */
public class HelloWorld {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        System.out.println("Hello World ( Bonjour le monde )");
    }

    /**
     * Obtenir la somme de deux entiers.
     * @param a Le premier nombre entier.
     * @param b Le deuxième nombre entier.
     * @return La valeur de la somme des deux entiers spécifiés.
     */
    public int somme(int a, int b) {
        return a + b;
    }
}
```



- Comme champ dans une classe :
 - initialisé par défaut : false ou 0
- Comme variable locale :
 - pas de valeur par défaut \Rightarrow il faut les initialiser ou les affecter.
- Syntaxe : `<type> <identificateur>`
 - `int i = 0;`
 - `boolean estVrai = false;`
 - `double rayon ;`
 - `rayon = 1.5;`



- Listes des operateurs
- Priorité des opérateurs
- Les opérateurs : arithmétiques
- Les opérateurs : binaires
- Les opérateurs : conditionnels
- Les opérateurs : simplifiés
- Les opérateurs : logiques

3 - Les bases du langage Java

-> Les opérateurs : Listes des opérateurs



<u>affectation</u>	<u>incrémentation décrémentation</u>	<u>arithmétique</u>	<u>logique</u>	<u>comparaison</u>	<u>accès aux membre</u>	<u>autre</u>
a = b	++a	+a	!a	a == b	a[b]	a(...)
	--a	-a	a && b	a != b	a.b	a, b
a += b	a++	a + b	a b	a < b		(type) a
a -= b	a--	a - b		a > b		? :
a *= b		a * b		a <= b		
a /= b		a / b		a >= b		
a %= b		a % b				
a &= b		~a				
a = b		a & b				
a ^= b		a b				
a <<= b		a ^ b				
a >>= b		a << b				
>>>=		a >> b				
		a>>> b				

3 - Les bases du langage Java

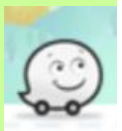
-> Les opérateurs : Listes des opérateurs



Symbole	Note	Priorité	Associativité
++a --a	Préincrémentation, prédécrémentation	16	Droite à gauche
a++ a--	Postincrémentation, postdécrémentation	15	Gauche à droite
~	Inversion des bits d'un entier	14	Droite à gauche
!	Non logique pour un booléen		Droite à gauche
- +	Moins et plus unaire		Droite à gauche
(type)	Conversion de type (cast)	13	Droite à gauche
* / %	Opérations multiplicatives	12	Gauche à droite
- +	Opérations additives	11	Gauche à droite
<< >> >>>	Décalage de bits, à gauche et à droite	10	Gauche à droite
instanceof < <= > >=	Opérateurs relationnels	9	Gauche à droite
== !=	Opérateurs d'égalité	8	Gauche à droite
&	Et logique bit à bit	7	Gauche à droite

3 - Les bases du langage Java

-> Les opérateurs : Listes des opérateurs



Symbole	Note	Priorité	Associativité
^	Ou exclusif logique bit à bit	6	Gauche à droite
	Ou inclusif logique bit à bit	5	Gauche à droite
&&	Et conditionnel	4	Gauche à droite
	Ou conditionnel	3	Gauche à droite
? :	Opérateur conditionnel	2	Droite à gauche
= *= /= %= += -= <<= >>= >>>= &= ^=	Opérateurs d'affectation	1	Droite à gauche



- Sur les réels : **+** **-** **/** *****
- Sur les entiers : **+** **-** **/** (quotient de la division), **%** (reste de la division)

l'affectation se fait grâce au signe '='

Hiérarchie habituelle (* et / prioritaires par rapport aux + et -)



- $\&$, bitand (et)

Table de vérité AND

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

- $|$, bitor (ou)

Table de vérité OR

A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

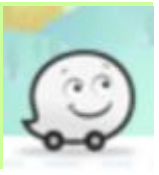
- \wedge , xor (ou exclusif)

Table de vérité XOR

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

3 - Les bases du langage Java

-> Les opérateurs : binaires (2/3)



- `&=`, `and_eq`, // $a \&= b \Leftrightarrow a = a \& b$
- `|=`, `or_eq`, // $a | b \Leftrightarrow a = a | b$
- `^=`, `xor_eq`, // $a \wedge= b \Leftrightarrow a = a \wedge b$

3 - Les bases du langage Java

-> Les opérateurs : binaires (3/3)



- \sim , compl : complément à 1

Table de vérité NOT

A	NOT A
0	1
1	0

- \ll , décalage à gauche
- \gg , décalage à droite

Exemple : $p = n \ll 3;$

p égal n décalé de 3 bits sur la gauche



$\text{expr1} \text{ ? } \text{expr2} \text{ : } \text{expr3}$

si (expr1) alors expr2 sinon expr3

Exemple : fonction min

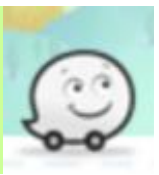
if (y<z) x=y; else x=z;

Equivalent à

$x = (y < z) \text{ ? } y \text{ : } z ;$

3 - Les bases du langage Java

-> Les opérateurs : simplifiés



- $i=i+1$ peut s'écrire $i++$; $(++i;)$
- $i=i-1$ peut s'écrire $i--$; $(--i;)$
- $a=a+b$ peut s'écrire $a+=b$;
- $a=a-b$ peut s'écrire $a-=b$;
- $a=a\&b$ peut s'écrire $a\&=b$;

3 - Les bases du langage Java

-> Les opérateurs : logiques



- Quelle utilisation ?
- Opérateurs classiques
- Les expressions
- Et - Ou - !

3 - Les bases du langage Java

-> Les opérateurs: logiques - Quelle utilisation ?



- Tous les tests :
 - si (expression vraie) ...
 - tant que (expression vraie) ...
- Valeurs booléennes
 - Vrai (1)
 - Faux (0)

3 - Les bases du langage Java

-> Les opérateurs : logiques - Opérateurs classiques



- Le 'ET'

A	B	ET
0	0	0
0	1	0
1	0	0
1	1	1

- Le 'OU'

A	B	OU
0	0	0
0	1	1
1	0	1
1	1	1



- Egalité

$a == b$

- Inégalité

$a != b$

- Relations d'ordre

$a < b$ $a <= b$ $a > b$ $a >= b$

- Expression

a est vraie si a est différent de 0

3 - Les bases du langage Java

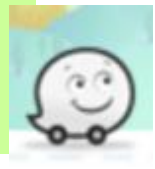
-> Les opérateurs : logiques - Et, Ou, !



- Et Logique
 - expression1 **&&** expression2
 - *si expression1 et expression2 sont vraies*
- Ou Logique
 - expression1 **||** expression2
 - *si expression1 ou expression2 est vraie*
- ! (Non)
 - **!**expression
 - *si expression est fausse*

3 - Les bases du langage Java

-> Les structures de contrôle et boucle



- Si ... alors ... sinon ... => `if () {} else {}`
- Au cas ... ou faire ... => `switch () { case }`
- Tant que ... faire ... => `while () {}`
- Répéter ... tant que ... => `do {} while ()`
- Boucle Pour ... faire ... => `for () {}`
- Boucle Pour chaque ... faire ... => `for () {}`
- Rupture de séquence => `break, continue, return`

3 - Les bases du langage Java

-> Les structures de contrôle et boucle

: si ... alors ... sinon ... => if () {} else {}



Si (expression conditionnelle vraie)
alors { instructions 1 }
sinon { instructions 2 }

```
if (expression) {  
    instructions 1;  
}  
else {  
    instructions 2;  
}
```

```
if (expression) {  
    instructions 1; /* bloc else  
    optionnel */  
}
```

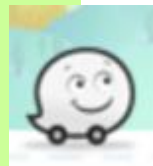
```
if (expression)  
    instruction; /* bloc d'1  
    instruction */
```

```
if (expression) instruction1;  
else instruction2;
```

3 - Les bases du langage Java

-> Les structures de contrôle et boucle

: si ... alors ... sinon ... => if () {} else {} : Exemple



```
1 package helloworld;
2 import java.util.Scanner;
3
4 public class HelloWorld {
5
6
7     public static void main(String[] args) {
8         Scanner sc = new Scanner(System.in);
9         System.out.println("Hello World ( Bonjour le monde )");
10        System.out.println("Appuyer sur la touche q pour quitter ");
11        String quit = sc.nextLine();
12        if ( "q".equals(quit) ) return ;
13        if ( "Q".equals(quit) ) {
14            System.out.println("Appuyer sur la touche q et pas Q pour quitter ");
15            quit = sc.nextLine();
16        } else {
17            System.out.println("Appuyer sur la touche q pour quitter ");
18            quit = sc.nextLine();
19        }
20        if ( "q".equals(quit) ) return ;
21        else System.out.println("Dernier essai, Appuyer sur la touche q pour quitter ");
22        quit = sc.nextLine();
23        System.out.println("Good bye");
24    }
25 }
26
27
```


3 - Les bases du langage Java

-> Les structures de contrôle et boucle

: Au cas ... ou faire ...=> switch () { case }



Au cas ou la variable vaut :

valeur 1 : faire ... ;

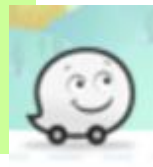
valeur 2 : faire ... ; etc.

```
switch (variable de type char ou int) {  
    case valeur 1 : ...;  
        ...;  
        break;  
    case valeur 2 : ...;  
        ...;  
        break;  
    default :      ...;  
        ...;  
}
```

3 - Les bases du langage Java

-> Les structures de contrôle et boucle

: Au cas ... ou faire ...=> switch () { case }Exemple



```

7 public static void main(String[] args) {
8     Scanner sc = new Scanner(System.in);
9     System.out.println("Hello World ( Bonjour le monde )");
10    System.out.println("Appuyer sur la touche q pour quitter ");
11    String quit = sc.nextLine();
12    switch (quit.charAt(0)) {
13        case 'q':
14            return;
15        case 'Q':
16            System.out.println("Appuyer sur la touche q et pas Q pour quitter ");
17            quit = sc.nextLine();
18            break;
19        case 'r':case 'R':case 'S':case 'T':
20            System.out.println("Appuyer sur la touche q et pas " + quit + " pour quitter ");
21            quit = sc.nextLine();
22            break;
23        default:
24            System.out.println("Appuyer sur la touche q et pas " + quit + " pour quitter ");
25            quit = sc.nextLine();
26            break;
27    }
28
29    if ("q".equals(quit)) {
30        return;
31    } else {
32        System.out.println("Dernier essai, Appuyer sur la touche q pour quitter ");
33    }
34    quit = sc.nextLine();
35    System.out.println("Good bye");
36 }

```

3 - Les bases du langage Java

-> Les structures de contrôle et boucle

: Tant que ... faire => while () {}



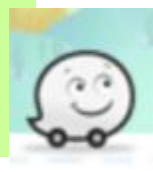
Tant que (expression vraie) faire
{bloc d 'instructions}

```
while (expression vraie) {  
    bloc d 'instructions }  
}
```

3 - Les bases du langage Java

-> Les structures de contrôle et boucle

: Tant que ... faire => while () {} : Exemple



```
package helloworld;
```

```
import java.util.Scanner;
```

```
public class HelloWorld {
```

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

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

```
        String quit = new String();
```

```
        while (! "q".equals(quit)) {
```

```
            System.out.println("Hello World ( Bonjour le monde )");
```

```
            System.out.println("Appuyer sur la touche q pour quitter ");
```

```
            quit = sc.nextLine();
```

```
        }
```

```
        System.out.println("Good bye");
```

```
    }
```

```
}
```

3 - Les bases du langage Java

-> Les structures de contrôle et boucle
: Répéter ... tant que ...=> `do {} while ()`



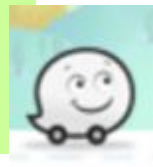
Répéter {bloc d 'instructions}
tant que (expression vraie)

```
do {bloc d 'instructions} while  
(expression vraie);
```

3 - Les bases du langage Java

-> Les structures de contrôle et boucle

: Répéter ... tant que ...=> do {} while () : Exemple

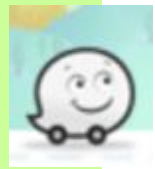


```
1  package helloworld;
2
3  import java.util.Scanner;
4
5  public class HelloWorld {
6
7      public static void main(String[] args) {
8          Scanner sc = new Scanner(System.in);
9          String quit;
10         do {
11             System.out.println("Hello World ( Bonjour le monde )");
12             System.out.println("Appuyer sur la touche q pour quitter ");
13             quit = sc.nextLine();
14         }while (! "q".equals(quit));
15         System.out.println("Good bye");
16     }
17 }
```

3 - Les bases du langage Java

-> Les structures de contrôle et boucle

: Boucle Pour ... faire ... => for () {}



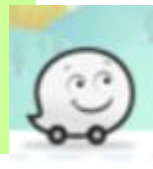
Pour (initialisation;condition;modification) faire {
 bloc d 'instructions}

```
for (initialisation;condition;modification) {  
    bloc d 'instructions}
```

3 - Les bases du langage Java

-> Les structures de contrôle et boucle

: Boucle Pour ... faire ... => for () {} : Exemple



```
1  package helloworld;
2
3  import java.util.Scanner;
4
5  public class HelloWorld {
6
7      public static void main(String[] args) {
8          Scanner sc = new Scanner(System.in);
9          String quit="";
10
11          for (; !"q".equals(quit);) {
12              for (int i = 0; i < 5; i++) {
13                  System.out.println("Hello World ( Bonjour le monde )");
14              }
15              System.out.println("Appuyer sur la touche q pour quitter ");
16              quit = sc.nextLine();
17          }
18
19          System.out.println("Good bye");
20      }
21  }
22
```




Pour (déclaration: expression) faire {
 bloc d 'instructions }





for (déclaration: expression) {
 bloc d 'instructions}

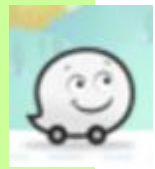
3 - Les bases du langage Java

-> Les structures de contrôle et boucle

: Boucle Pour chaque ... faire ... => for () {} : Exemple



```
1  package helloworld;
2
3    import java.util.Scanner;
4
5  public class HelloWorld {
6
7        public static void main(String[] args) {
8          System.out.println("Good bye");
9          int v[] = {0, 1, 2, 3, 4, 5};
10         for (int i : v) {
11             System.out.println("valeur " + i);
12         }
13         System.out.println();
14         String[] data = {"Nice", "Marseille"};
15         for (String s : data) {
16             System.out.println(s);
17         }
18     }
19 }
```



- ***Return[valeur]*** : permet de quitter immédiatement la fonction en cours.
- ***break*** : permet de passer à l'instruction suivant l'instruction *while*, *do*, *for* ou *switch* la plus imbriquée.
- ***continue*** : saute directement à la dernière ligne de l'instruction *while*, *do* ou *for* la plus imbriquée.

3 - Les bases du langage Java

-> Les structures de contrôle et boucle

: Rupture de séquence - break, continue, return : Exemple



```
package helloworld;

import java.util.Scanner;

public class HelloWorld {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String quit = "";
        while (!"q".equals(quit)) {
            for (int i = 0; i < 5; i++) {
                System.out.println("Hello World ( Bonjour le monde ) : num => " + i);
                System.out.println("Appuyer sur la touche 'o' pour quitter la boucle ");
                quit = sc.nextLine();
                if ("o".equals(quit)) {
                    break;
                }
            }
            System.out.println(" Appuyer sur la touche 'r' pour recommencer la boucle");
            System.out.println(" Appuyer sur la touche 'q' pour quitter ");
            quit = sc.nextLine();

            if ("r".equals(quit)) {
                continue;
            }
            System.out.println("Suite => ");
            if (!"q".equals(quit)) {
                return;
            }
        }
        System.out.println("Good bye");
    }
}
```

3 - Les bases du langage Java

-> Les structures de contrôle et boucle : Exercices => Nombre mystère



```
il vous reste 10 coups a jouer pour trouver un nombre compris entre [-1000:1000]
Entrez un nombre entre [-1000:1000]0
0 est plus petit que ??? .
il vous reste 9 coups a jouer
Entrez un nombre entre [-1000:1000]500
500 est plus grand que ??? .
il vous reste 8 coups a jouer
Entrez un nombre entre [-1000:1000]250
250 est plus petit que ??? .
il vous reste 7 coups a jouer
Entrez un nombre entre [-1000:1000]725
725 est plus grand que ??? .
il vous reste 6 coups a jouer
Entrez un nombre entre [-1000:1000]300
300 est plus petit que ??? .
il vous reste 5 coups a jouer
Entrez un nombre entre [-1000:1000]400
400 est plus grand que ??? .
il vous reste 4 coups a jouer
Entrez un nombre entre [-1000:1000]350
350 est plus grand que ??? .
il vous reste 3 coups a jouer
Entrez un nombre entre [-1000:1000]325
325 est plus grand que ??? . |
il vous reste 2 coups a jouer
Entrez un nombre entre [-1000:1000]310
310 est plus petit que ??? .
il vous reste 1 coups a jouer
Entrez un nombre entre [-1000:1000]3020
Erreur, nombre hors de l'intervalle [-1000:1000]
Entrez un nombre: 320
Perdu le nombre a trouver etait 313
```

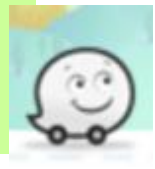
Remarque :

utilisation des imports :

```
import java.util.Scanner;
import java.util.Random;
```

3 - Les bases du langage Java

-> Les chaînes de caractères : la classe String



`String mes =" Ceci est " + " une chaine ";`

- Les chaînes peuvent être concaténées à des nombres :

```
int ans = 10;
```

```
String mes=" Je suis age de " + ans + " ans ";
```

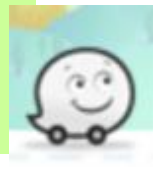
- Les chaînes peuvent être concaténées à tout type d'objet :

```
Personne durand = new Personne();
```

```
String mes=" Je suis une personne " + durand;
```

sera compilé comme

```
String mes=" Je suis une personne " + durand.toString();
```



String salut = "Hello";

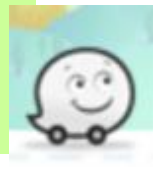
- égalité : méthode equals

```
if (salut.equals("Hello") ) // vrai
```

```
If ("Hello".equals(salut) )
```

- sous-chaînes : méthode substring

```
if (salut.substring(0,4).equals("Hell")) // vrai
```



```
int [] a ; // tableaux d 'entiers  
double [] [] m ; // matrice de double  
String [] jours= {"Lundi","mardi", ... "Dimanche"};
```

- **Construction du tableau par new**

```
a=new int[10] ;  
m=new double [3][5] ; // 3 lignes et 5 colonnes
```

- **Utilisation**

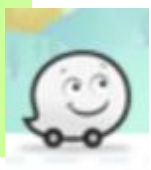
```
m[i][j]=x;
```

- **Taille**

```
a.length // = 10
```


3 - Les bases du langage Java

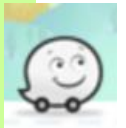
-> les tableaux : initialisation explicite d'un tableau



```
//int tableau5[5] = {10,20,30,40,50};  
//int tableau6[3][2] = {{5,1},{6,2},{7,3}};  
  
int [] tableauInt3 = {10,20,30,40,50};  
int tableauInt4[][] = {{5,1},{6,2},{7,3}};  
double tableauDouble1[][] = {{5.,1.,3.},  
                                {6.0,2.},  
                                {1.,2.,3.,4.,7,3}};
```

3 - Les bases du langage Java

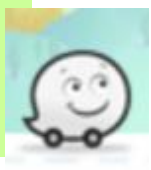
-> les tableaux : La déclaration des tableaux



```
int tableauInt1[] = new int[10]; /* <=>  int[] tableauInt1 = new int[20]; */  
|  
int tableauInt2[]; // déclaration  
tableauInt2 = new int[30]; //allocation  
  
float tableauFloatMulti[][] = new float[10][10];
```

3 - Les bases du langage Java

-> les tableaux : Le parcours d'un tableau



```
int [] tableauInt = {10,20,30,40,50};

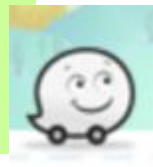
for (int i = 0; i < tableauInt.length ; i++) {
    System.out.println( "T["+i+"]=" + tableauInt[i] );
}

System.out.println("-----");
int k=0;
for (int val: tableauInt) {
    System.out.println( "T["+k+"]=" + val );
    ++k;
}
```

Remarque : un tableau qui dépasse sa capacité, lève une exception du type `java.lang.arrayIndexOutOfBoundsException`.

3 - Les bases du langage Java

-> Les conversions de types



Classe	Rôle
String	pour les chaînes de caractères Unicode
Integer	pour les valeurs entières (integer)
Long	pour les entiers longs signés (long)
Float	pour les nombres à virgule flottante (float)
Double	pour les nombres à virgule flottante en double précision (double)

Remarque : La conversion peut entraîner une perte d'informations

3 - Les bases du langage Java

-> Les conversions de types

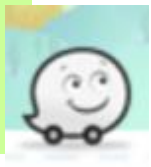


- La conversion d'un entier int en float

```
int entier = 5;  
float flottant = (float) entier;
```

- La conversion d'un entier int en entier long

```
int entier = 5;  
Integer nombre= new Integer(entier);  
long f = nombre.intValue();
```

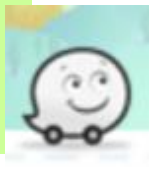


- La conversion d'un entier int en chaîne de caractères String

```
int i = 10;  
String texte = new String();  
texte = texte.valueOf(i);
```

- La conversion d'une chaîne de caractères String en entier int

```
String texte = new String(" 10 ");  
Integer monnombre = new Integer(texte);  
int i = monnombre.intValue();
```



- Un caractère

```
char uncaractere = 'a';
```

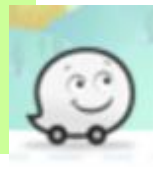
- Une chaine de caractère

```
String texte = "Bonjour ";
```

3 - Les bases du langage Java

-> La manipulation des chaînes de caractères

: Caractères spéciaux

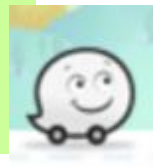


Caractères spéciaux	Affichage
\'	Apostrophe
\"	Guillemet
\\	Antislash
\t	Tabulation
\b	Retour arrière
\r	Retour chariot
\f	Saut de page
\n	Saut de ligne
\0ddd	Caractère ASCII ddd (octal)
\xdd	Caractère ASCII dd (hexadécimal)
\udddd	Caractère Unicode dddd (hexadécimal)

3 - Les bases du langage Java

-> La manipulation des chaînes de caractères

: opération divers



- Addition de chaine

```
String texte1 = " texte";  
texte1 += " texte" ;  
texte1 += " 2 | ";
```

- Comparaison de chaine

```
String texte2 = " texte 2 " ;  
if (texte1.equals(texte2)) {  
    System.out.println("OK");  
} else {  
    System.out.println("KO");  
}
```

- Longueur d'une chaine

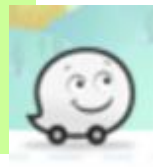
```
System.out.println(texte1.length());
```

- Modification de la casse d'une chaine

```
String textemin = " texte " ;  
String textemaj = textemin.toUpperCase();
```

3 - Les bases du langage Java

-> La manipulation des chaînes de caractères



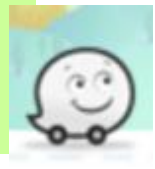
<https://docs.oracle.com/javase/8/docs/api/index.html?java/lang/String.html>



The screenshot shows the Java Platform Standard Ed. 8 API documentation for the `String` class. The left sidebar lists various packages and classes, including `java.applet`, `java.awt`, `java.awt.color`, `java.awt.datatransfer`, `java.awt.dnd`, `java.awt.event`, `java.awt.font`, `java.awt.geom`, `java.awt.im`, `java.awt.im.spi`, `java.awt.image`, and `java.awt.image.renderable`. The main content area displays the `String` class details, including its inheritance hierarchy (extending `Object` and implementing `Serializable`, `Comparable<String>`, and `CharSequence`), a description of the class, and examples of its usage. The examples show how to create a `String` object, how to concatenate strings, and how to extract substrings. The documentation also mentions that the `String` class includes methods for examining individual characters, comparing strings, searching strings, and creating a copy of a string with all characters translated to uppercase or lowercase.

Liens Divers:

- http://www.tutorialspoint.com/java/java_strings.htm
- <http://www.java-examples.com/java-string-examples>



Références

- **L'accès aux objets et tableaux se fait par référence**
 <nom-de-classe> <nom-objet>;
- **Le constructeur " new " alloue la mémoire et initialise les données de l'objet**

Personne Durand = new Personne(" Durand ", 20);

☞ String nom=" Durand " ou nom=new String(" Durand ");

int array[][] ;

array=new int[4][];

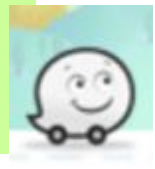
ou array =new int[4][3];

for (int i=0; i<4; i++)

 array=new int[3];

3 - Les bases du langage Java

-> Passage par valeur ou par référence

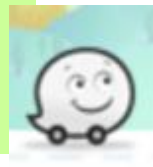


Les arguments aux méthodes Java sont passés :

- **par valeur** pour les types de base
c'est une copie qui est passée à la méthode
- **par référence** pour les objets et tableaux

3 - Les bases du langage Java

-> Nouveautés Java 9



jigsaw, le système modulaire

Jigsaw a pour objectif de rendre modulaire votre application.

- réduire la taille physique de la JRE sur les devices embarqués notamment,
- rendre les classes internes de la JVM réellement privées,
- augmenter la sécurité des applications : moins de classes chargées par la JVM = une surface d'attaque plus faible pour les hackers.

<https://blog.soat.fr/2017/05/java-9-la-revolution-des-modules>

Méthodes privées dans les interfaces

Pour alléger les méthodes par défaut des interfaces depuis Java 8 et éviter la duplication de code, il désormais possible d'implémenter des méthodes private dans une interface

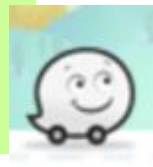
Try-with-resources

try-with-resources permet d'instancier des objets implémentant `java.lang.AutoCloseable` sans avoir à explicitement appeler la méthode `close()`. On peut en java 9 désormais utiliser des variables Closeable instanciées en dehors d'un bloc try-with-resources :

<https://blog.xebia.fr/2018/09/25/de-java-8-a-11-nouveautes-et-conseils-pour-migrer/>

3 - Les bases du langage Java

-> Nouveautés Java 9



Instanciation de collections immuables

L'instanciation de collections immuables ont été grandement facilitée

l'annotation **@Deprecated** enrichie

Deux nouveaux attributs pour cette annotation

API Flow

Java 9 intègre l'API Flow, vous permettant d'implémenter vos propres flux réactifs

<https://blog.xebia.fr/2018/03/06/introduction-aux-flux-reactifs-en-java>

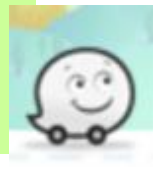
JShell

Il s'agit du read-eval-print loop (REPL) de Java. Il permet ainsi de tester des instructions Java sans avoir à démarrer tout un programme.

Améliorations de l'API Stream

Quatre nouvelles méthodes ont vu le jour dans l'API Stream :

<https://blog.xebia.fr/2018/09/25/de-java-8-a-11-nouveautes-et-conseils-pour-migrer/>



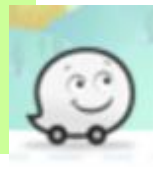
Inférence de type

À ne pas confondre avec des variables non typées, Java 10 intègre l'inférence de type afin de gagner en lisibilité et éviter ainsi la redondance

Instanciation de collections immuables, encore...

Tout d'abord, petit point sur la copie de List avec `Collections#unmodifiableList`

<https://blog.xebia.fr/2018/09/25/de-java-8-a-11-nouveautes-et-conseils-pour-migrer/>



Inférence de type pour les paramètres de lambdas

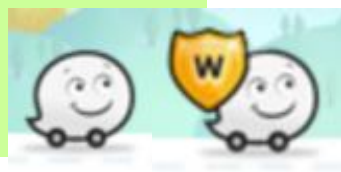
Java 10 a apporté les var, mais on ne pouvait pas les utiliser dans les paramètres des expressions lambda. C'est maintenant corrigé avec Java 11

Nouveau client HTTP

Initialement prévu avec Java 9, ce nouveau client HTTP est finalement sorti de son incubateur avec Java 11

<https://blog.xebia.fr/2018/09/25/de-java-8-a-11-nouveautes-et-conseils-pour-migrer/>

4 - L'aspect objet du langage Java



- Classes
- Membres de classe
- Méthodes de classe
- Héritages
- Polymorphisme
- Le traitement des exceptions
- Les espaces de nom
- Les patrons

4 - L'aspect objet du langage Java

-> Structure d'un programme



Un programme Java est constitué d'un ensemble de classes :

- groupées en paquetages (packages),
- réparties en fichiers,
- chaque classe est dans un fichier NomClasse.java,
- chaque classe compilée est dans un fichier NomClasse.class.

Un fichier source NomClasse.java comporte :

- des directives d'importation
`import java.io.*;`
- des déclarations de classes.

4 - L'aspect objet du langage Java

-> Structure d'un programme



Une classe est composée de :

- déclarations de variables ou attributs,
- définitions de méthodes,
- les membres sont :
 - » des membres de classe (static),
 - » des membres d'objet (ou d'instance).

Une classe à 3 rôles :

- de typage, en déclarant de nouveaux types,
- d'implémentation en définissant la structure et le comportement d'objets,
- de moule pour la création des instances.



- Déclaration
- Définition
- Encapsulation
- Constructeur/Destructeur

4 - L'aspect objet du langage Java

-> Classes



```
package exemplecours;

import static java.lang.System.runFinalization;
import java.util.logging.Level;
import java.util.logging.Logger;

public class ExempleCours {

    public static void main(String[] args) {
        System.out.println("Exemple Cours");
        Voiture voiture = new Voiture("AX368BZ");
        voiture.afficher();
        System.out.println(voiture);
        voiture.setVitesse(100);
        System.out.println(voiture);
        try {
            voiture.finalize();
        } catch (Throwable ex) {
            System.out.println("erreur : " + ExempleCours.class.getName() + "[" + ex + "]");
        }
    }
}
```

class exemplecours

Voiture		Cloneable
-	a_vitesse: double	
-	a_immatriculation: String	
+	GetCreate(): Voiture	
+	GetCreate(v: String): Voiture	
#	Voiture()	
#	Voiture(v: Voiture)	
#	Voiture(v: String)	
+	finalize(): void	
+	getVitesse(): double	
+	setVitesse(vitesse: double): void	
+	getImmatriculation(): String	
+	setImmatriculation(immatriculation: String): void	
+	afficher(): void	
+	toString(): String	
+	clone(): Voiture	
+	equals(obj: Object): boolean	

Output - ExempleCours (run) ×	Test Results	Inspector
run:		
Exemple Cours		
Voiture [AX368BZ]		
Vitesse : 0.0		
Voiture [AX368BZ]		
Vitesse : 0.0		
Voiture [AX368BZ]		
Vitesse : 100.0		
Voiture supprimé de la mémoire		
BUILD SUCCESSFUL (total time: 0 seconds)		

4 - L'aspect objet du langage Java

-> Classes



```
package exemplecours;

public class Voiture implements Cloneable {

    static public Voiture GetCreate() {
    static public Voiture GetCreate(String v) {
    protected Voiture() {
    protected Voiture(Voiture v) {
    protected Voiture(String v) {

        @Override
        public void finalize() throws Throwable {
        public double getVitesse() {
        public void setVitesse(double vitesse) {
        public String getImmatriculation() {
        public void setImmatriculation(String immatriculation) {
        public void afficher() {

        @Override
        public String toString() {

        @Override
        public Voiture clone() throws CloneNotSupportedException {

        @Override
        public boolean equals(Object obj) {

        private double a_vitesse;
        private String a_immatriculation;
    }
}
```

class exemplecours

Voiture		Cloneable
-	a_vitesse: double	
-	a_immatriculation: String	
+	GetCreate(): Voiture	
+	GetCreate(v: String): Voiture	
#	Voiture()	
#	Voiture(v: Voiture)	
#	Voiture(v: String)	
+	finalize(): void	
+	getVitesse(): double	
+	setVitesse(vitesse: double): void	
+	getImmatriculation(): String	
+	setImmatriculation(immatriculation: String): void	
+	afficher(): void	
+	toString(): String	
+	clone(): Voiture	
+	equals(obj: Object): boolean	

4 - L'aspect objet du langage Java

-> Classes



```
package exemplecours;

public class Voiture implements Cloneable {

    static public Voiture GetCreate() {
        return new Voiture();
    }
    static public Voiture GetCreate(String v) {
        return new Voiture(v);
    }
    protected Voiture() {
        super();
    }
    protected Voiture(Voiture v) {
        super();
        this.a_immatriculation = v.a_immatriculation;
        this.a_vitesse = v.a_vitesse;
    }
    protected Voiture(String v) {
        super();
        this.a_immatriculation = v;
    }
    //..... etc
```

4 - L'aspect objet du langage Java

-> Les modificateurs de classe



Modificateur	Rôle
abstract	la classe contient une ou des méthodes abstraites, qui n'ont pas de définition explicite. Une classe déclarée abstract ne peut pas être instanciée : il faut définir une classe qui hérite de cette classe et qui implémente les méthodes nécessaires pour ne plus être abstraite.
final	la classe ne peut pas être modifiée, sa redéfinition grâce à l'héritage est interdite. Les classes déclarées final ne peuvent donc pas avoir de classes filles.
private	la classe n'est accessible qu'à partir du fichier où elle est définie
public	la classe est accessible partout
(none)	accessible par la classe courante et les classes du même paquetage

4 - L'aspect objet du langage Java

-> Membres de classe : attribut



```
package exemplecours;
```

```
public class Rectangle {
```

```
    public double a_largeur = 0, a_hauteur = 0;
```

```
    protected String a_name = "";
```

```
    private final String a_type = "Rectangle";
```

```
    static int Compteur = 0 ;
```

```
    final double a_pi=3.14 ;
```

```
}
```

4 - L'aspect objet du langage Java

-> Les modificateurs d'attribut



Modificateur	Rôle
public	Une variable, méthode ou classe déclarée public est visible par tous les autres objets. Depuis la version 1.0, une seule classe public est permise par fichier et son nom doit correspondre à celui du fichier. Dans la philosophie orientée objet aucune donnée d'une classe ne devrait être déclarée publique : il est préférable d'écrire des méthodes pour la consulter et la modifier
par défaut : package friendly	Il n'existe pas de mot clé pour définir ce niveau, qui est le niveau par défaut lorsqu'aucun modificateur n'est précisé. Cette déclaration permet à une entité (classe, méthode ou variable) d'être visible par toutes les classes se trouvant dans le même package.
protected	Si une méthode ou une variable est déclarée protected, seules les méthodes présentes dans le même package que cette classe ou ses sous-classes pourront y accéder. On ne peut pas qualifier une classe avec protected.
private	C'est le niveau de protection le plus fort. Les composants ne sont visibles qu'à l'intérieur de la classe : ils ne peuvent être modifiés que par des méthodes définies dans la classe et prévues à cet effet. Les méthodes déclarées private ne peuvent pas être en même temps déclarées abstract car elles ne peuvent pas être redéfinies dans les classes filles.

4 - L'aspect objet du langage Java

-> Les modificateurs d'attribut



Modificateur	Rôle
final	la méthode ne peut être modifiée (redéfinition lors de l'héritage interdite)
static	la méthode appartient simultanément à tous les objets de la classe (comme une constante déclarée à l'intérieur de la classe). Il est inutile d'instancier la classe pour appeler la méthode mais la méthode ne peut pas manipuler de variable d'instance. Elle ne peut utiliser que des variables de classes.
synchronized	la méthode fait partie d'un thread. Lorsqu'elle est appelée, elle barre l'accès à son instance. L'instance est à nouveau libérée à la fin de son exécution.
native	le code source de la méthode est écrit dans un autre langage

4 - L'aspect objet du langage Java

-> Constructeur



Les objets sont instanciés au moyen de constructeurs : `new` qui réserve la place pour l'objet et initialise les attributs à leur valeur par défaut

- Toute classe a un constructeur par défaut sans argument
- Le constructeur port le même nom que la classe
- Le constructeur exécute le corps de la méthode
- Le constructeur retourne la référence à l'objet créé.

```
-----  
Class Pixel {  
int x, y ; }  
-----
```

```
Pixel p ; // p indéfini  
p = new Pixel() ; // p! null p.x=p.y=0  
p.x = 4 ; p.y = 5;
```

```
-----  
Class Pixel {  
int x, y ; Pixel(int x, int y) {  
this.x = x; this.y = y; } }  
-----
```

```
Pixel p, q ; // p et q indéfini  
p = new Pixel(4,5); // p.x=4 p.y=5  
q = new Pixel() // erreur !
```

4 - L'aspect objet du langage Java

-> Méthode de classe



```
package exemplecours;

public class Rectangle {

    public double a_largeur = 0, a_hauteur = 0;
    protected String a_name = "";
    private final String a_type = "Rectangle";

    static int Compteur = 0;
    final double a_pi = 3.14;

    public Rectangle(String name, double largeur, double hauteur) {...8 lines }

    @Override
    protected void finalize() throws Throwable {...8 lines }

    public static void main(String[] args) throws Throwable {...5 lines }

    @Override
    public String toString() {...7 lines }
    public double surface() {...4 lines }

}
```



La destruction des objets est effectuée :

- automatiquement par le ramasse-miettes (garbage collector)
- la méthode `dispose()` permet de libérer des champs, à utiliser à l'inverse du `new` dans l'héritage
- la méthode `finalize()` permet de spécifier des actions à la destruction de l'objet (nettoyage, fermeture de fichiers).
- L'appel au ramasse-miettes peut être forcé par l'appel :

```
System.gc();
```

```
System.runFinalization();
```

4 - L'aspect objet du langage Java

-> Destructeur



```
public Rectangle(String name, double largeur, double hauteur) {
    super();
    System.out.println("Rectangle crée");
    Rectangle.Compteur++;
    this.a_largeur = largeur;
    this.a_hauteur = hauteur;
    this.a_name = name;
}

@Override
protected void finalize() throws Throwable {
    super.finalize();
    try {
        System.out.println("Rectangle supprimé de la mémoire");
    } finally {
        super.finalize();
    }
}

public static void main(String[] args) throws Throwable {
    System.out.println("Exemple Rectangle => ");
    Rectangle rec = new Rectangle("Mon Rectangle", 5.2, 4.3);
    rec.finalize();
}
```

```
Exemple Rectangle =>
Rectangle crée
Rectangle supprimé de la mémoire
BUILD SUCCESSFUL (total time: 0 seconds)
```

4 - L'aspect objet du langage Java

-> Destructeur



```
public static void main(String[] args) throws Throwable {  
    System.out.println("Exemple Rectangle => ");  
    Rectangle rec = new Rectangle("Mon Rectangle",5.2,4.3);  
    System.out.println("Surface" + rec.surface());  
    System.out.println(rec);  
    rec.finalize();  
}
```

@Override

```
public String toString() {  
    String res="";  
    res+=this.a_type + " " + this.a_name + "\n";  
    res+="\t- largeur " + this.a_largeur + "\n";  
    res+="\t- hauteur " + this.a_hauteur + "\n";  
    return res;  
}  
  
public double surface() {  
    double surf = this.a_largeur * this.a_hauteur;  
    return surf;  
}
```

```
Exemple Rectangle =>  
Rectangle crée  
Surface22.36  
Rectangle Mon Rectangle|  
    - largeur 5.2  
    - hauteur 4.3  
  
Rectangle supprimé de la mémoire
```


4 - L'aspect objet du langage Java

-> Héritage



- la classe dérivée possède les attributs de la classe de base (sauf les privés)
- la classe dérivée possède les méthodes de la classe de base (sauf les privées)
- on peut définir des nouveaux attributs et méthodes
- on peut redéfinir des méthodes (surcharger)
- la dérivation se fait par `extends`
- toute classe dérive de la classe `Object`
- `super` : pour désigner la méthode de la classe de base

4 - L'aspect objet du langage Java

-> Conversion de type cast



- Utiliser un type de données comme s'il en était un autre : ne peut se faire que sur des objets liés (sous-classes et super-classes)
- Deux types de cast :
 - Upcasting : traiter un objet de la sous-classe comme un objet de la super-classe
 - tout objet peut être converti en `java.lang.Object`
 - on ne peut plus accéder qu'aux champs de la super-classe
 - les méthodes surchargées sont appelables.
 - Downcasting : traiter un objet d'une super-classe comme un objet de la sous-classe
 - un objet peut être converti dans la sous-classe que s'il est de ce type
 - opérateur `instanceof`

```
Article un_article = new Chemise(); ;  
if ( un_article instanceof Vetement )  
    Vetement pull= (Vetement) un_article;
```



- **Classes Abstraites :**

- représentent des idées
- on ne peut instancier une classe abstraite, il faut dériver et surcharger toutes les méthodes abstraites.

- **Interfaces :**

- Une interface sert à spécifier des méthodes qu'une classe doit avoir sans indiquer comment les réaliser.
- les méthodes d'interface sont abstraites et tacitement « public »,
- les champs sont « static » et « final »
- mot clé implements

4 - L'aspect objet du langage Java

-> Héritage



```
package exemplecours;
```

```
public interface FormeGeometrique {  
    public abstract double surface() ;  
    public abstract double perimetre() ;  
    public abstract String getNom() ;  
}
```

4 - L'aspect objet du langage Java

-> Héritage



```
public class Rectangle implements FormeGeometrique {

    public double a_largeur = 0, a_hauteur = 0;
    protected String a_name = "";
    private final String a_type = "Rectangle";

    static int Compteur = 0;
    final double a_pi = 3.14;

    public Rectangle(String name, double largeur, double hauteur) {...8 lines }
    @Override
    protected void finalize() throws Throwable {...8 lines }
    public static void main(String[] args) throws Throwable {
        System.out.println("Exemple Rectangle => ");
        FormeGeometrique rec = new Rectangle("Mon Rectangle", 5.2, 4.3);
        System.out.println("Surface" + rec.surface());
        System.out.println(rec);
        //rec.finalize();
    }
    @Override
    public String toString() {...7 lines }
    @Override
    public double surface() {...4 lines }
    @Override
    public double perimetre() {...3 lines }
    @Override
    public String getNom() {...3 lines }
    @Override
    public String getType() {...3 lines }
}
```

4 - L'aspect objet du langage Java

-> Héritage



```
package exemplecours;
```

```
public class Carre extends Rectangle {
```

```
    public Carre(String name, double largeur) {  
        super(name, largeur, largeur);  
        this.a_type = "Carre";  
    }
```

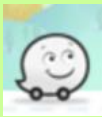
```
    public static void main(String[] args) throws Throwable {  
        System.out.println("Exemple Carre => ");  
        FormeGeometrique rec = new Carre("Mon Carre ", 5);  
        System.out.println("Surface" + rec.surface());  
        System.out.println(rec);  
        //rec.finalize();  
    }
```

```
}
```

```
Exemple Carre => *  
Rectangle crée  
Surface25.0  
Carre Mon Carre |  
    - largeur 5.0  
    - hauteur 5.0
```

4 - L'aspect objet du langage Java

-> Le traitement des exceptions



```
package exemplecours;

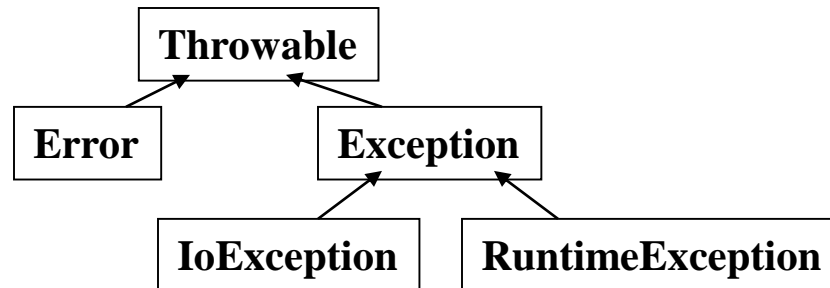
public class TestException {

    public static void main(java.lang.String[] args) {
        int i = 100;
        int j = 0;
        System.out.println("résultat = " + (i / j));
    }
}
```

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at exemplecours.TestException.main(TestException.java:8)
C:\Users\David\AppData\Local\NetBeans\Cache\8.1\executor-snippets\run.xml:53: Java returned: 1
BUILD FAILED (total time: 0 seconds)
```



Traitement des Erreurs



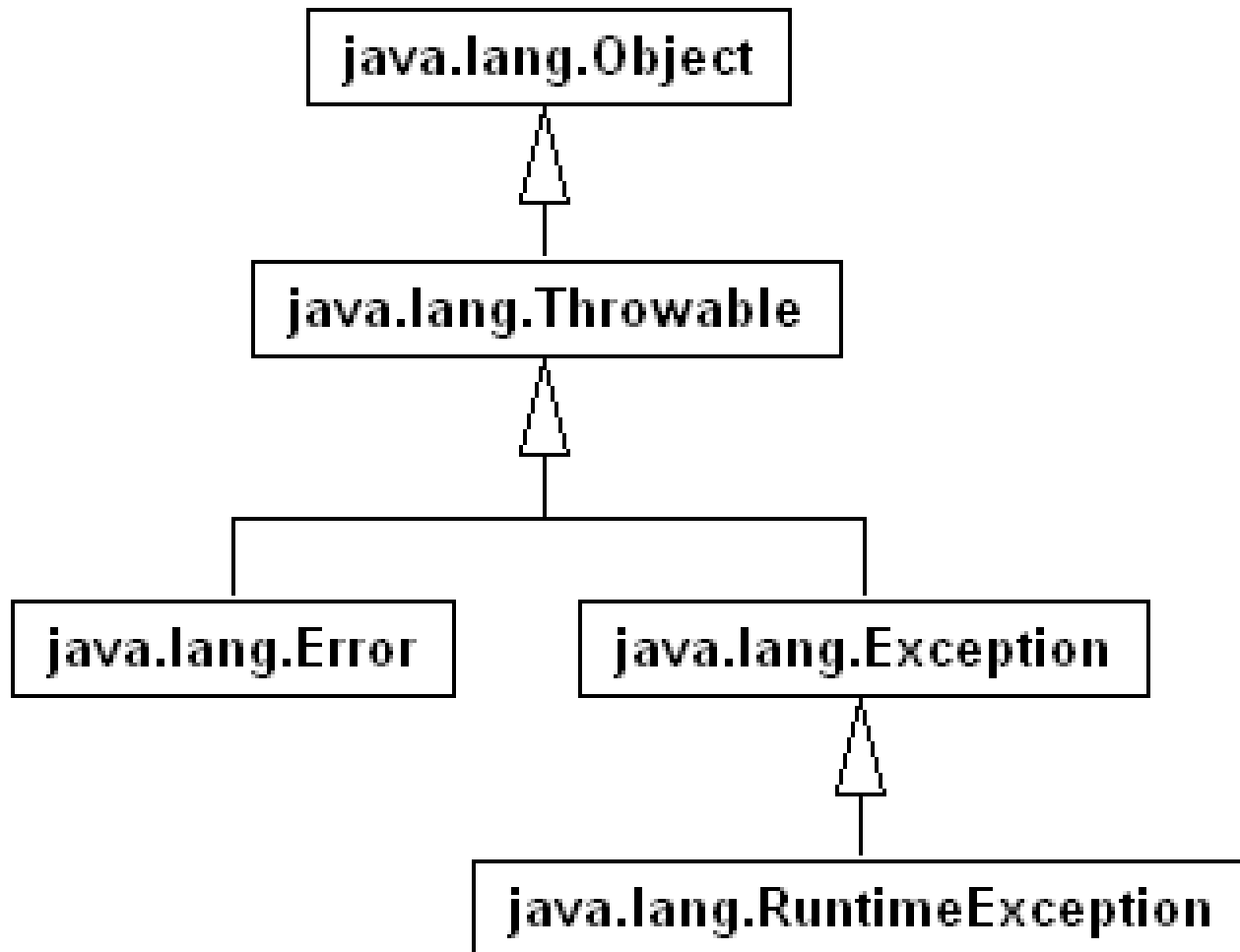
- lancer une exception et arrêt : **throw**

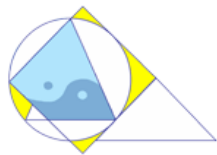
String readData(BufferedReader in) throws EOFException

```
{ while ( ... )  
{  
    if (ch == - 1) // EOF  
        { if (n < len ) throw new EOFException( ) ; }  
}  
}
```


4 - L'aspect objet du langage Java

-> Le traitement des exceptions





4 - L'aspect objet du langage Java

-> Le traitement des exceptions : try {} catch {} finally {}



Yantra Technologies

```
package exemplecours;
```

```
public class TestException {

    static public void Test1() {
        int i = 100;
        int j = 0;
        System.out.println("Debut test 1 ");
        try {
            System.out.println("résultat = " + (i / j));
        } catch (ArithmeticException e1) {
            System.out.println("Exception 1");
            e1.printStackTrace();
        } catch (Exception e2) {
            System.out.println("Exception 2 ");
            e2.printStackTrace();
        } finally {
            System.out.println("Fin");
        }
        System.out.println("Fin test 1 ");
    }

    public static void main(java.lang.String[] args) {
        TestException.Test1();
    }
}
```

```
Debut test 1
Exception 1
java.lang.ArithmeticException: / by zero
Fin
Fin test 1
    at exemplecours.TestException.Test1(TestException.java:10)
    at exemplecours.TestException.main(TestException.java:24)
```

4 - L'aspect objet du langage Java

-> Le traitement des exceptions : throw ()



```

package exemplecours;

public class MonException extends Exception {

    public MonException(String message) {
        super(message);
    }
}

public class TestException {

    static public void Test1() { ...17 lines }

    static public void Test2() {
        int i = 100;
        int j = 0;
        System.out.println("Debut test 2");
        try {
            if (j == 0) {
                throw new MonException("Erreur division par zero");
            }
            System.out.println("résultat = " + (i / j));
        } catch (ArithmeticException e1) {
            System.out.println("Exception 1");
            e1.printStackTrace();
        } catch (Exception e2) {
            System.out.println("Exception 2 ");
            e2.printStackTrace();
        } finally {
            System.out.println("Fin");
        }
        System.out.println("Fin test 2");
    }

    public static void main(java.lang.String[] args) {
        //TestException.Test1();
        TestException.Test2();
    }
}

```

```

Debut test 2
Exception 2
exemplecours.MonException: Erreur division par zero
Fin
Fin test 2
    at exemplecours.TestException.Test2(TestException.java:29)
    at exemplecours.TestException.main(TestException.java:46)

```

4 - L'aspect objet du langage Java

-> Le traitement des exceptions : throw ()



```
public class TestException {  
  
    static public void Test1() { ...17 lines }  
  
    static public void Test2() { ...20 lines }  
  
    static public void Test3() throws MonException {  
        int i = 100;  
        int j = 0;  
        System.out.println("Debut test 3");  
        if (j == 0) {  
            throw new MonException("Erreur division par zero");  
        }  
        System.out.println("résultat = " + (i / j));  
  
        System.out.println("Fin test 3");  
    }  
    public static void main(java.lang.String[] args) throws MonException {  
        //TestException.Test1();  
        //TestException.Test2();  
        TestException.Test3();  
    }  
}
```

```
Debut test 3  
Exception in thread "main" exemplecours.MonException: Erreur division par zero  
    at exemplecours.TestException.Test3(TestException.java:49)  
    at exemplecours.TestException.main(TestException.java:58)  
C:\Users\David\AppData\Local\NetBeans\Cache\8.1\executor-snippets\run.xml:53: Java returned: 1
```

4 - L'aspect objet du langage Java

-> La classe `java.lang.Object`

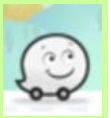


La classe `java.lang.Object`

- `protected Object clone ()` : crée une copie de l'objet
- `boolean equals (Object o)` : égalité de l'objet
- `protected finalize ()` : appelé par le garbage collector
- `Class getClass ()` : rend le nom de la classe de l'objet
- `int hashCode ()` : hashcode de l'objet
- `void notify ()` : réveille le thread qui attend l'objet
- `void notifyAll ()` : réveille tous les threads
- `String toString()` : représentation en String de l'objet
- `void wait ()` : attente d'un notify

4 - L'aspect objet du langage Java

-> Les espaces de nom : définition



package nom-du-package;

avec la création du répertoire associé

4 - L'aspect objet du langage Java

-> Les espaces de nom : Utilisation



Exemple	Rôle
import nomPackage.*;	toutes les classes du package sont importées
import nomPackage.nomClasse;	appel à une seule classe : l'avantage de cette notation est de réduire le temps de compilation

4 - L'aspect objet du langage Java

-> Les classes internes



```
package exemplecours;

public class ClassePrincipale {

    class ClasseInterne {

        void afficher() {
            System.out.println("Classe Interne ");
        }

        void afficher() {
            ClasseInterne val = this. new ClasseInterne();
            System.out.println("Classe ClassePrincipale ");
            val.afficher();
        }

        public static void main(String[] args) {
            new ClassePrincipale(). afficher();
        }
    }
}
```

Classe ClassePrincipale
Classe Interne

4 - L'aspect objet du langage Java

-> les patrons : avant Java 5



```
package exemplecours;
```

```
import java.util.LinkedList;
```

```
public class MaListeAvantJava5
```

```
{  
    private LinkedList liste;  
    public MaListeAvantJava5() {  
        this.liste= new LinkedList();  
    }
```

```
    public void setMembre(String s)  
    {  
        liste.add(s);  
    }
```

```
    public int getMembre(int i)  
    {  
        //probleme a l'execution  
        return (int)liste.get(i);  
    }
```

```
    public static void main(java.lang.String[] args) throws MonException {  
        MaListeAvantJava5 malist = new MaListeAvantJava5();  
        malist.setMembre("chiffre");  
        int i = malist.getMembre(0);  
        System.out.println(i);  
    }
```

```
Exception in thread "main" java.lang.ClassCastException: java.lang.String cannot be cast to java.lang.Integer  
    at exemplecours.MaListeAvantJava5.getMembre (MaListeAvantJava5.java:19)  
    at exemplecours.MaListeAvantJava5.main (MaListeAvantJava5.java:25)
```

4 - L'aspect objet du langage Java

-> les patrons : après Java 5



```
import java.util.LinkedList;
```

```
public class MaListeApresJava5< MaClasse >
```

```
{
    private final LinkedList< MaClasse > liste;
    public MaListeApresJava5() {
        this.liste= new LinkedList<>();
    }
    public void setMembre(MaClasse s)
    {
        liste.add(s);
    }
    public MaClasse getMembre(int i)
    {
        return liste.get(i);
    }
}
```

run:

chiffre
100

BUILD SUCCESSFUL (total time: 0 seconds)

```
public static void main(java.lang.String[] args) throws MonException {
```

```
    MaListeApresJava5<String> liststring = new MaListeApresJava5<>();
    liststring.setMembre("chiffre");
    String s = liststring.getMembre(0);
    System.out.println(s);
```

```
    MaListeApresJava5<Integer> listint = new MaListeApresJava5<>();
    listint.setMembre(100);
    int i = (int)listint.getMembre(0);
    System.out.println(i);
}
```

run:

chiffre

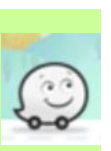
100

BUILD SUCCESSFUL (total time: 0 seconds)

- Les types génériques java n'acceptent que des type Objet pas de type simple (int, float, double)
- il n'est pas possible d'implémenter plusieurs fois la même interface avec des paramètres différents,
- il n'est pas possible de créer deux versions surchargées d'une méthode (deux méthodes portant le même nom) l'une utilisant la classe Object et l'autre utilisant un type générique.

4 - L'aspect objet du langage Java

-> Autoboxing et Unboxing



L'autoboxing permet de transformer automatiquement une variable de type primitif en un objet du type du wrapper correspondant. L'unboxing est l'opération inverse.

L'un des apports de Java 5 est la notion d'auto-boxing, et d'auto-unboxing.

En Java 4, on ne peut construire un objet wrapper qu'explicitement, par appel à son constructeur ou à une de ses méthodes.

```
List list = new ArrayList();  
Integer myInt = 100; // boxing  
int i = myInt;        // unboxing  
list.add(i);
```

```
// avant :  
// List list = new ArrayList();  
// Integer myInt = new Integer( 100 );  
// int i = myInt.intValue();  
// list.add(myInt);
```

```
Integer i = new Integer(1) ; // instantiation classique d'un Integer  
Integer j = 1 ;              // conversion automatique -> boxing  
int k = i ;                  // conversion automatique -> unboxing
```

4 - L'aspect objet du langage Java

-> Énumération<>



- Une énumération est une classe contenant une liste de sous-objets.
- Une énumération se construit grâce au mot clé enum.
- Les enum héritent de la classe java.lang.Enum.
- Chaque élément d'une énumération est un objet à part entière.
- Vous pouvez compléter les comportements des objets d'une énumération en ajoutant des méthodes.

```
public enum LangageEnumeration {  
  
    //Objets directement construits  
    JAVA("Langage JAVA"),  
    C("Langage C"),  
    CPlus("Langage C++"),  
    PHP("Langage PHP");  
  
    private String name = "";  
  
    //Constructeur  
    LangageEnumeration(String name) {  
        this.name = name;  
    }  
  
    @Override  
    public String toString() {  
        return name;  
    }  
  
    public static void main(String args[]) {  
  
        LangageEnumeration l1 = LangageEnumeration.JAVA;  
        LangageEnumeration l2 = LangageEnumeration.PHP;  
        System.out.println(l1);  
        System.out.println(l2);  
  
    }  
}
```

<https://openclassrooms.com/courses/apprenez-a-programmer-en-java/les-enumerations-1>



```
public enum Animal
{
    // Il faut appeler l'un des constructeurs déclarés :
    KANGOUROU("kangourou", false),
    TIGRE("tigre", false),
    CHIEN("chien", true),
    SERPENT("serpent", false, "tropical"),
    CHAT("chat", true); // <- NB: le point-virgule pour mettre fin à la liste de

    // Membres :
    private final String environnement;
    private final String nom;
    private final boolean domestique;

    Animal(String nom, boolean domestique)
    { this(nom, domestique, null); }

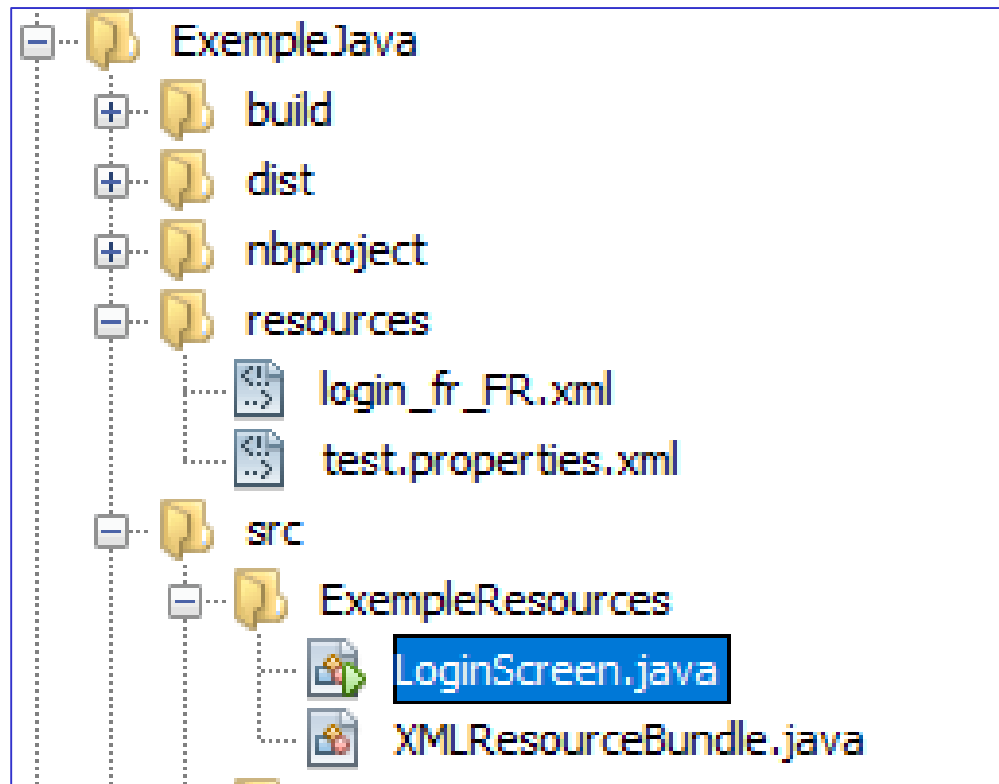
    Animal(String nom, boolean domestique, String environnement)
    {
        this.nom = nom;
        this.domestique = domestique;
        this.environnement = environnement;
    }

    public String getNom(){ return this.nom; }
    public String getEnvironnement(){ return this.environnement; }
    public boolean isDomestique(){ return this.domestique; }

    public static void main(String[] args)
    {
        Animal animal = Animal.TIGRE;
        System.out.print("L'animal est un "+animal.getNom());
        System.out.print(animal.isDomestique()?" (domestique)":" (sauvage)");
        String env = animal.getEnvironnement();
        if (env!=null)
            System.out.print(" vivant dans un milieu "+env);
        System.out.println();
    }
};
```

4 - L'aspect objet du langage Java

-> Fichier propriétés et Internationalisation : Répertoire



4 - L'aspect objet du langage Java

-> Fichier propriétés et Internationalisation : Fichier XML



```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd>
<properties>
    <entry key="français">Français</entry>
    <entry key="login">Identification</entry>
    <entry key="username">Identifiant</entry>
    <entry key="password">Mot de passe</entry>
</properties>
```

4 - L'aspect objet du langage Java

-> Fichier propriétés et Internationalisation : lire les ressources



```
public class XMLResourceBundle extends ResourceBundle {

    private final Properties properties = new Properties();

    protected XMLResourceBundle(String FileNameXML) throws IOException {

        InputStream stream = getClass().getClassLoader().getResourceAsStream(FileNameXML);

        try {
            this.properties.loadFromXML(stream);
        } finally {
            stream.close();
        }
    }

    @Override
    public Object handleGetObject(String key) {
        return this.properties.get(key);
    }

    @Override
    public Enumeration getKeys() {
        return this.properties.keys();
    }
}
```


4 - L'aspect objet du langage Java

-> Fichier propriétés et Internationalisation : lire les ressources



```
public class LoginScreen {  
  
    public ResourceBundle rb;  
    public LoginScreen(String propertiesClassName) {  
        try {  
            this.rb = new XMLResourceBundle(propertiesClassName);  
        } catch (IOException ex) {  
            Logger.getLogger(LoginScreen.class.getName()).log(Level.SEVERE, null, ex);  
        }  
    }  
  
    public static void main(String[] args) throws IOException { ...27 lines }  
  
}
```

4 - L'aspect objet du langage Java

-> Fichier propriétés et Internationalisation : lire les ressources



```
public class LoginScreen {

    public ResourceBundle rb;
    public LoginScreen(String propertiesClassName) { ...7 lines }

    public static void main(String[] args) throws IOException {

        Properties p = new Properties();
        InputStream instr = null;

        try {
            instr = new FileInputStream("resources/login_fr_FR.xml");
            p.loadFromXML(instr);
        } catch (FileNotFoundException ex) {
            Logger.getLogger(HelloWorld.class.getName()).log(Level.SEVERE, null, ex);
        } catch (IOException ex) {
            Logger.getLogger(HelloWorld.class.getName()).log(Level.SEVERE, null, ex);
        } finally {
            instr.close();
        }
        p.list(System.out);
        System.out.println("-----");
        for (Enumeration e = p.keys(); e.hasMoreElements();) {
            System.out.println(e.nextElement());
        }
        System.out.println("-----");
        LoginScreen l = new LoginScreen("login_fr_FR.xml");
        System.out.println(l.rb.getString("password"));
        System.out.println("-----");
    }
}
```

4 - L'aspect objet du langage Java

-> Fichier propriétés et Internationalisation



```
-- listing properties --
```

```
login=Identification
```

```
password=Mot de passe
```

```
francais=Français
```

```
username=Identifiant
```

```
-----
```

```
login
```

```
password
```

```
francais
```

```
username
```

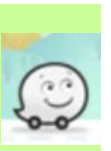
```
-----
```

```
Mot de passe
```

```
-----
```

4 - L'aspect objet du langage Java

-> l'API Logging de Java



L'usage de fonctionnalités de logging dans les applications est tellement répandu que SUN a décidé de développer sa propre API et de l'intégrer au JDK à partir de la version 1.4.

Le but est de proposer un système qui puisse être exploité facilement par toutes les applications. L'API repose sur plusieurs classes principales et une interface:

- **Logger** : cette classe permet d'envoyer des messages dans le système de log
- **LogRecord** : cette classe encapsule le message
- **Handler** : cette classe représente la destination des messages
- **Formatter** : cette classe permet de formater le message avant son envoi vers la destination
- **Filter** : cette interface, dont le but est de déterminer si le message sera enregistré, doit être implémentée par les classes désireuses de filtrer les messages
- **Level** : cette classe représente le niveau de gravité du message
- **LogManager** : cette classe est un singleton qui permet de gérer l'état des Loggers

Un logger possède un ou plusieurs Handler qui sont des entités recevant les messages. Chaque Handler peut avoir un filtre associé en plus du filtre associé au Logger.

Chaque message possède un niveau de sévérité représenté par la classe Level.

<https://docs.oracle.com/javase/8/docs/api/java/util/logging/package-summary.html>

4 - L'aspect objet du langage Java

-> l'API Logging de Java



```
Logger LOGGER = Logger.getLogger(TestLogging.class.getName());

{
    LOGGER.setLevel(Level.ALL);
    LOGGER.info("1er exemple");
}

{
    Handler fh;
    try {
        fh = new FileHandler("TestLogging.log");
        LOGGER.addHandler(fh);
    } catch (SecurityException e) {
        LOGGER.severe("Impossible d'associer le FileHandler");
    } catch (IOException e) {
        LOGGER.severe("Impossible d'associer le FileHandler");
    }
    LOGGER.log(Level.INFO, "2eme exemple");
}
```

```
févr. 27, 2018 2:19:24 PM test.logging.TestLogging main
INFOS: 1er exemple
févr. 27, 2018 2:19:24 PM test.logging.TestLogging main
INFOS: 2eme exemple
```

4 - L'aspect objet du langage Java

-> l'API Logging de Java



```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE log SYSTEM "logger.dtd">
<log>
<record>
  <date>2018-02-27T14:20:54</date>
  <millis>1519737654548</millis>
  <sequence>1</sequence>
  <logger>test.logging.TestLogging</logger>
  <level>INFO</level>
  <class>test.logging.TestLogging</class>
  <method>main</method>
  <thread>1</thread>
  <message>2eme exemple</message>
</record>
</log>
```

TestLogging.log

4 - L'aspect objet du langage Java

-> l'API Logging de Java



Le type Level définit 7 + 2 niveaux pour les messages OFF

OFF	Aucun niveau
SEVERE	Pour indiquer un problème sérieux
WARNING	Pour signaler un problème potentiel
INFO	Message d'information
CONFIG	Configuration
FINE	Trace d'exécution
FINER	Trace d'exécution plus précise
FINEST	Trace d'exécution encore plus précise
ALL	Tous les niveaux

4 - L'aspect objet du langage Java

-> l'API Logging de Java



```
Handler fh;
try {
    fh = new FileHandler("TestLogging.log");
    LOGGER.addHandler(fh);
} catch (SecurityException e) {
    LOGGER.severe("Impossible d'associer le FileHandler");
} catch (IOException e) {
    LOGGER.severe("Impossible d'associer le FileHandler");
}
LOGGER.log(Level.INFO, "2eme exemple");

LOGGER.setLevel(Level.ALL); //pour envoyer les messages de tous les niveaux
LOGGER.setUseParentHandlers(false); // pour supprimer la console par défaut
ConsoleHandler ch = new ConsoleHandler();
ch.setLevel(Level.INFO); // pour n'accepter que les message de niveau &Ge; INFO
LOGGER.addHandler(ch);
LOGGER.log(Level.WARNING, " 3eme exemple ");
LOGGER.log(Level.SEVERE, " 4eme exemple ", new Exception("4eme exemple")); // les messages + la pile d'exécution
```

```
févr. 27, 2018 2:22:51 PM test.logging.TestLogging main
INFOS: 1er exemple
févr. 27, 2018 2:22:51 PM test.logging.TestLogging main
INFOS: 2eme exemple
févr. 27, 2018 2:22:51 PM test.logging.TestLogging main
AVERTISSEMENT:  3eme exemple |
févr. 27, 2018 2:22:51 PM test.logging.TestLogging main
GRAVE:  4eme exemple
java.lang.Exception: 4eme exemple
    at test.logging.TestLogging.main(TestLogging.java:42)
```


5 - Les packages de base

- java.lang
- java.io
- java.nio
- java.util

5 - Les packages de base -> java.lang



L'extension ou *package* java.lang est l'extension de base de Java (il n'est pas nécessaire de faire un import de cette extension, il est implicite).

Interface Hierarchy

- [java.lang.Appendable](#)
- [java.lang.AutoCloseable](#)
- [java.lang.CharSequence](#)
- [java.lang.Cloneable](#)
- [java.lang.Comparable<T>](#)
- [java.lang.Iterable<T>](#)
- [java.lang.Readable](#)
- [java.lang.Runnable](#)
- [java.lang.Thread.UncaughtExceptionHandler](#)

<https://docs.oracle.com/javase/8/docs/api/java/lang/package-summary.html>

5 - Les packages de base -> java.lang



java.lang.[Object](#)

- [java.lang.Boolean](#) (implements [java.lang.Comparable<T>](#), [java.io.Serializable](#))
- [java.lang.Character](#) (implements [java.lang.Comparable<T>](#), [java.io.Serializable](#))
- [java.lang.Character.Subset](#)
- [java.lang.Class<T>](#) (implements [java.lang.reflect.AnnotatedElement](#), [java.lang.reflect.GenericDeclaration](#), [java.io.Serializable](#), [java.lang.reflect.Type](#))
- [java.lang.ClassLoader](#)
- [java.lang.ClassValue<T>](#)
- [java.lang.Compiler](#)
- [java.lang.Enum<E>](#) (implements [java.lang.Comparable<T>](#), [java.io.Serializable](#))
- [java.lang.Math](#)
- [java.lang.Number](#) (implements [java.io.Serializable](#))
 - [java.lang.Byte](#) (implements [java.lang.Comparable<T>](#))
 - [java.lang.Double](#) (implements [java.lang.Comparable<T>](#))
 - [java.lang.Float](#) (implements [java.lang.Comparable<T>](#))
 - [java.lang.Integer](#) (implements [java.lang.Comparable<T>](#))
 - [java.lang.Long](#) (implements [java.lang.Comparable<T>](#))
 - [java.lang.Short](#) (implements [java.lang.Comparable<T>](#))
- [java.lang.Package](#) (implements [java.lang.reflect.AnnotatedElement](#))
- [java.lang.Process](#)
- [java.lang.ProcessBuilder](#)
- [java.lang.ProcessBuilder.Redirect](#)
- [java.lang.Runtime](#)
- [java.lang.SecurityManager](#)
- [java.lang.StackTraceElement](#) (implements [java.io.Serializable](#))
- [java.lang.StrictMath](#)
- [java.lang.String](#) (implements [java.lang.CharSequence](#), [java.lang.Comparable<T>](#), [java.io.Serializable](#))
- [java.lang.StringBuffer](#) (implements [java.lang.CharSequence](#), [java.io.Serializable](#))
- [java.lang.StringBuilder](#) (implements [java.lang.CharSequence](#), [java.io.Serializable](#))
- [java.lang.System](#)
- [java.lang.Thread](#) (implements [java.lang.Runnable](#))
- [java.lang.ThreadGroup](#) (implements [java.lang.Thread.UncaughtExceptionHandler](#))
- [java.lang.ThreadLocal<T>](#)
- [java.lang.Throwable](#) (implements [java.io.Serializable](#))
 - [java.lang.Error](#)
 - [java.lang.Exception](#)

<https://docs.oracle.com/javase/8/docs/api/java/lang/package-summary.html>

5 - Les packages de base -> java.io



java.io (io pour in out) est un package qui va nous fournir des classes pour manipuler des **flux de données**. On les appelle aussi les classes d'entrées/sorties.

Principe de fonctionnement des entrées /sorties :

1. Ouverture d'un moyen de communication
2. Lecture ou écriture de données
3. Fermeture du moyen de communication

Interface Hierarchy

- [java.lang.AutoCloseable](#)
 - [java.io.Closeable](#)
 - [java.io.ObjectInput](#) (also extends [java.io.DataInput](#))
 - [java.io.ObjectOutput](#) (also extends [java.io.DataOutput](#))
- [java.io.DataInput](#)
 - [java.io.ObjectInput](#) (also extends [java.lang.AutoCloseable](#))
- [java.io.DataOutput](#)
 - [java.io.ObjectOutput](#) (also extends [java.lang.AutoCloseable](#))
- [java.io.FileFilter](#)
- [java.io.FilenameFilter](#)
- [java.io.Flushable](#)
- [java.io.ObjectInputValidation](#)
- [java.io.ObjectStreamConstants](#)
- [java.io.Serializable](#)
 - [java.io.Externalizable](#)

5 - Les packages de base -> java.io



java.lang.Object

- [java.io.Console](#) (implements [java.io.Flushable](#))
- [java.io.File](#) (implements [java.lang.Comparable<T>](#), [java.io.Serializable](#))
- [java.io.FileDescriptor](#)
- [java.io.InputStream](#) (implements [java.io.Closeable](#))
- [java.io.ObjectInputStream.GetField](#)
- [java.io.ObjectOutputStream.PutField](#)
- [java.io.ObjectStreamClass](#) (implements [java.io.Serializable](#))
- [java.io.ObjectStreamField](#) (implements [java.lang.Comparable<T>](#))
- [java.io.OutputStream](#) (implements [java.io.Closeable](#), [java.io.Flushable](#))
- [java.security.Permission](#) (implements [java.security.Guard](#), [java.io.Serializable](#))
 - [java.security.BasicPermission](#) (implements [java.io.Serializable](#))
 - [java.io.SerializablePermission](#)
 - [java.io.FilePermission](#) (implements [java.io.Serializable](#))
- [java.io.RandomAccessFile](#) (implements [java.io.Closeable](#), [java.io.DataInput](#), [java.io.DataOutput](#))
- [java.io.Reader](#) (implements [java.io.Closeable](#), [java.lang.Readable](#))
 - [java.io.BufferedReader](#)
 - [java.io.LineNumberReader](#)
 - [java.io.CharArrayReader](#)
 - [java.io.FilterReader](#)
 - [java.io.PushbackReader](#)
 - [java.io.InputStreamReader](#)
 - [java.io.FileReader](#)
 - [java.io.PipedReader](#)
 - [java.io.StringReader](#)
- [java.io.StreamTokenizer](#)
- [java.lang.Throwable](#) (implements [java.io.Serializable](#))
- [java.io.Writer](#) (implements [java.lang.Appendable](#), [java.io.Closeable](#), [java.io.Flushable](#))

5 - Les packages de base

-> java.io: Fichiers et répertoires : Java.io.File



Cette classe fournit une définition indépendante de la plate-forme des fichiers et des répertoires.

```
public class TestIO {  
    public static void main(String[] args) {  
        File f = new File("c:\\windows\\csup.txt");  
        System.out.println(f.exists());  
        System.out.println(f.canRead());  
        System.out.println(f.canWrite());  
        System.out.println(f.length());  
  
        File f2 = new File("c:/windows/csup.txt");  
        System.out.println(f2.exists());  
        System.out.println(f2.canRead());  
        System.out.println(f2.canWrite());  
        System.out.println(f2.length());  
  
        File d = new File("c:\\windows");  
        System.out.println(d.isDirectory());  
  
        String[] files = d.list();  
        for (String file : files) {  
            System.out.println(file);  
        }  
    }  
}
```

5 - Les packages de base

-> java.io : Lecture et écriture de fichiers binaires



```
public static void main(String[] args) {  
  
    FileInputStream fin = null;  
    try {  
        fin = new FileInputStream("source.txt ");  
        byte[] data = new byte[fin.available()];  
        fin.read(data);  
        fin.close();  
        try (FileOutputStream fos = new FileOutputStream("cible.txt ")) {  
            fos.write(data);  
        }  
    } catch (FileNotFoundException ex) {  
        System.out.println(ex);  
    } catch (IOException ex) {  
        System.out.println(ex);  
    } finally {  
        try {  
            if ( fin != null)  
                fin.close();  
        } catch (IOException ex) {  
            System.out.println(ex);  
        }  
    }  
}
```

5 - Les packages de base

-> java.io : Lire et écrire des types primitifs : java.io.Data



```
public static void main(String[] args) {
    FileInputStream fin = null;
    try {
        fin = new FileInputStream("source.txt");
        DataInputStream din = new DataInputStream(fin);

        int i = din.readInt();
        double d = din.readDouble();
        // BufferedReader din = new BufferedReader(new InputStreamReader(in));
        // String s = din.readLine();
        FileOutputStream fos = new FileOutputStream("cible.txt ");
        DataOutputStream dos = new DataOutputStream(fos);
        dos.writeInt(123);
        dos.writeDouble(123.456);
        dos.writeChars("Une chaîne");
    } catch (FileNotFoundException ex) {
        System.out.println(ex);
    } catch (IOException ex) {
        System.out.println(ex);
    } finally {
        try {
            if (fin != null) {
                fin.close();
            }
        } catch (IOException ex) {
            System.out.println(ex);
        }
    }
}
```




manipuler un `OutputStream` au travers des méthodes

`print()` et `println()`

```
PrintStream ps;  
try {  
    ps = new PrintStream(new FileOutputStream("cible.txt"));  
    ps.println(" une ligne ");  
    ps.println(123);  
    ps.print(" une autre ");  
    ps.print(" ligne");  
    ps.flush();  
    ps.close();  
} catch (FileNotFoundException ex) {  
    System.out.println(ex);  
}
```



Ecrire des objets sur des flux implements java.io.serializable

```
// Ecriture
FileOutputStream fos;
try {
    fos = new FileOutputStream("tmp");

    ObjectOutputStream oos = new ObjectOutputStream(fos);
    oos.writeObject("Today");
    oos.writeObject(new Date());
    oos.flush();
} catch (FileNotFoundException ex) {
    System.out.println(ex);
} catch (IOException ex) {
    System.out.println(ex);
}
```



- Par défaut, tous les champs sont sérialisés (y compris private)
- Cela peut poser des problèmes de sécurité
- 3 solutions :
 - Ne pas implémenter Serializable
 - Réécrire les méthodes writeObjet() et readObject()
 - Le mot clé **transcient** permet d'indiquer qu'un champ ne doit pas être sérialisé.



Découpe une chaîne de caractères en fonction des séparateurs

```
class lecture {  
  
    public static void main(String[] args) {  
        StringTokenizer st  
            = new StringTokenizer("chaine1,chaine2,chaine3,chaine4", ",");  
        while (st.hasMoreTokens()) {  
            System.out.println(st.nextToken());  
        }  
    }  
}
```

5 - Les packages de base -> java.io : Exemple



```
package com.tutorialspoint;

import java.io.File;

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

        File f = null;
        boolean bool = false;

        try{
            // create new files
            f = new File("test.txt");

            // create new file in the system
            f.createNewFile();

            // tests if file exists
            bool = f.exists();

            // prints
            System.out.println("File exists: "+bool);

            if(bool == true)
            {
                // delete() invoked
                f.delete();
                System.out.println("delete() invoked");
            }

            // tests if file exists
            bool = f.exists();

            // prints
            System.out.print("File exists: "+bool);

        }catch(Exception e){
            // if any error occurs
            e.printStackTrace();
        }
    }
}
```

http://www.tutorialspoint.com/java/io/file_exists.htm



L'API NIO 2 a été développée sous la [JSR 203](#) et a été ajoutée au JDK dans la version 7 de Java SE.

NIO 2 est une API plus moderne et plus complète pour l'accès au système de fichiers. Son but est en partie de remplacer la classe File de la très ancienne API IO.

NIO 2 propose d'étendre les fonctionnalités relatives aux entrées/sorties : l'utilisation du système de fichiers de manière facile et les lectures/écritures asynchrones.



Class Hierarchy

- [java.lang.Object](#)
 - [java.nio.Buffer](#)
 - [java.nio.ByteBuffer](#) (implements [java.lang.Comparable<T>](#))
 - [java.nio.MappedByteBuffer](#)
 - [java.nio.CharBuffer](#) (implements [java.lang.Appendable](#), [java.lang.CharSequence](#), [java.lang.Comparable<T>](#), [java.lang.Readable](#))
 - [java.nio.DoubleBuffer](#) (implements [java.lang.Comparable<T>](#))
 - [java.nio.FloatBuffer](#) (implements [java.lang.Comparable<T>](#))
 - [java.nio.IntBuffer](#) (implements [java.lang.Comparable<T>](#))
 - [java.nio.LongBuffer](#) (implements [java.lang.Comparable<T>](#))
 - [java.nio.ShortBuffer](#) (implements [java.lang.Comparable<T>](#))
 - [java.nio.ByteOrder](#)
 - [java.lang.Throwable](#) (implements [java.io.Serializable](#))
 - [java.lang.Exception](#)
 - [java.lang.RuntimeException](#)
 - » [java.nio.BufferOverflowException](#)
 - » [java.nio.BufferUnderflowException](#)
 - » [java.lang.IllegalStateException](#)
 - » [java.nio.InvalidMarkException](#)
 - » [java.lang.UnsupportedOperationException](#)
 - » [java.nio.ReadOnlyBufferException](#)

5 - Les packages de base -> java.nio : Exemple



```
package com.tutorialspoint;

import java.io.*;
import java.nio.CharBuffer;

public class ReaderDemo {

    public static void main(String[] args) {

        String s = "Hello world";

        // create a new Char Buffer with capacity of 12
        CharBuffer cb = CharBuffer.allocate(12);

        // create a StringReader
        Reader reader = new StringReader(s);

        try {
            // read characters into a char buffer
            reader.read(cb);

            // flip the char buffer
            cb.flip();

            // print the char buffer
            System.out.println(cb.toString());

            // Close the stream
            reader.close();

        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
}
```

http://www.tutorialspoint.com/java/io/reader_read_charbuffer.htm

5 - Les packages de base -> java.util



Le package `java.util` contient beaucoup de structures de données que l'on peut aussi appeler collections.

Il a d'autres classes dans `java.util`, mais leur usage est un peu plus ponctuel.

La classe `Calendar` permet de gérer tout ce qui peut toucher aux dates et aux calendriers. C'est une classe abstraite dont la principale sous classe est `GregorianCalendar`, notre calendrier gregorien. On trouve aussi des utilitaires comme `StringTokenizer` qui sert à découper un texte en unité lexicale.

Interface Hierarchy

- `java.util.Comparator<T>`
- `java.util.Enumeration<E>`
- `java.util.EventListener`
- `java.util.Formattable`
- `java.lang.Iterable<T>`
 - `java.util.Collection<E>`
 - `java.util.List<E>`
 - `java.util.Queue<E>`
 - `java.util.Deque<E>`
 - `java.util.Set<E>`
 - `java.util.SortedSet<E>`
 - » `java.util.NavigableSet<E>`
- `java.util.Iterator<E>`
 - `java.util.ListIterator<E>`
- `java.util.Map<K,V>`
 - `java.util.SortedMap<K,V>`
 - `java.util.NavigableMap<K,V>`
- `java.util.Map.Entry<K,V>`
- `java.util.Observer`
- `java.util.RandomAccess`

5 - Les packages de base -> java.util



java.lang.[Object](#)

- java.util.[AbstractCollection](#)<E> (implements java.util.[Collection](#)<E>)
 - java.util.[AbstractList](#)<E> (implements java.util.[List](#)<E>)
 - java.util.[AbstractSequentialList](#)<E>
 - java.util.[LinkedList](#)<E> (implements java.lang.[Cloneable](#), java.util.[Deque](#)<E>, java.util.[List](#)<E>, java.io.[Serializable](#))
 - java.util.[ArrayList](#)<E> (implements java.lang.[Cloneable](#), java.util.[List](#)<E>, java.util.[RandomAccess](#), java.io.[Serializable](#))
 - java.util.[Vector](#)<E> (implements java.lang.[Cloneable](#), java.util.[List](#)<E>, java.util.[RandomAccess](#), java.io.[Serializable](#))
 - java.util.[Stack](#)<E>
 - java.util.[AbstractQueue](#)<E> (implements java.util.[Queue](#)<E>)
 - java.util.[PriorityQueue](#)<E> (implements java.io.[Serializable](#))
 - java.util.[AbstractSet](#)<E> (implements java.util.[Set](#)<E>)
 - java.util.[EnumSet](#)<E> (implements java.lang.[Cloneable](#), java.io.[Serializable](#))
 - java.util.[HashSet](#)<E> (implements java.lang.[Cloneable](#), java.io.[Serializable](#), java.util.[Set](#)<E>)
 - java.util.[TreeSet](#)<E> (implements java.lang.[Cloneable](#), java.util.[NavigableSet](#)<E>, java.io.[Serializable](#))
 - java.util.[ArrayDeque](#)<E> (implements java.lang.[Cloneable](#), java.util.[Deque](#)<E>, java.io.[Serializable](#))
- java.util.[AbstractMap](#)<K,V> (implements java.util.[Map](#)<K,V>)
 - java.util.[EnumMap](#)<K,V> (implements java.lang.[Cloneable](#), java.io.[Serializable](#))
 - java.util.[HashMap](#)<K,V> (implements java.lang.[Cloneable](#), java.util.[Map](#)<K,V>, java.io.[Serializable](#))
 - java.util.[IdentityHashMap](#)<K,V> (implements java.lang.[Cloneable](#), java.util.[Map](#)<K,V>, java.io.[Serializable](#))
 - java.util.[TreeMap](#)<K,V> (implements java.lang.[Cloneable](#), java.util.[NavigableMap](#)<K,V>, java.io.[Serializable](#))
 - java.util.[WeakHashMap](#)<K,V> (implements java.util.[Map](#)<K,V>)
- java.util.[AbstractMap.SimpleEntry](#)<K,V> (implements java.util.[Map.Entry](#)<K,V>, java.io.[Serializable](#))
- java.util.[AbstractMap.SimpleImmutableEntry](#)<K,V> (implements java.util.[Map.Entry](#)<K,V>, java.io.[Serializable](#))
- java.util.[Arrays](#)
- java.util.[BitSet](#) (implements java.lang.[Cloneable](#), java.io.[Serializable](#))
- java.util.[Calendar](#) (implements java.lang.[Cloneable](#), java.lang.[Comparable](#)<T>, java.io.[Serializable](#))
- java.util.[Collections](#)
- java.util.[Currency](#) (implements java.io.[Serializable](#))
- java.util.[Date](#) (implements java.lang.[Cloneable](#), java.lang.[Comparable](#)<T>, java.io.[Serializable](#))
- java.util.[Dictionary](#)<K,V>
 - java.util.[Hashtable](#)<K,V> (implements java.lang.[Cloneable](#), java.util.[Map](#)<K,V>, java.io.[Serializable](#))

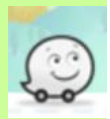
5 - Les packages de base -> java.util



java.lang.Object

- java.util.EventListenerProxy<T> (implements java.util.EventListener)
- java.util.EventObject (implements java.io.Serializable)
- java.util.FormattableFlags
- java.util.Formatter (implements java.io.Closeable, java.io.Flushable)
- java.util.Locale (implements java.lang.Cloneable, java.io.Serializable)
- java.util.Locale.Builder
- java.util.Objects
- java.util.Observable
- java.security.Permission (implements java.security.Guard, java.io.Serializable)
- java.util.Random (implements java.io.Serializable)
- java.util.ResourceBundle
- java.util.ResourceBundle.Control
- java.util.Scanner (implements java.io.Closeable, java.util.Iterator<E>)
- java.util.ServiceLoader<S> (implements java.lang.Iterable<T>)
- java.util.StringTokenizer (implements java.util.Enumeration<E>)
- java.lang.Throwable (implements java.io.Serializable)
- java.util.Timer
- java.util.TimerTask (implements java.lang.Runnable)
- java.util.TimeZone (implements java.lang.Cloneable, java.io.Serializable)
- java.util.UUID (implements java.lang.Comparable<T>, java.io.Serializable)

5 - Les packages de base -> java.util : Exemple



```
package com.tutorialspoint;

import java.util.*;

public class CollectionsDemo {
    public static void main(String args[]) {
        // create an array of string objs
        String init[] = { "One", "Two", "Three", "One", "Two", "Three" };

        // create one list
        List list = new ArrayList(Arrays.asList(init));

        System.out.println("List value before: "+list);

        // sort the list
        Collections.sort(list);

        System.out.println("List value after sort: "+list);
    }
}
```

http://www.tutorialspoint.com/java/util/collections_sort_comparable.htm

6 - Socket



- Présentation
- Le modèle client-serveur Java
- URL



Qu'est ce qu'une socket?

Point d'entrée entre 2 appli. du réseau

Permet l'échange de données entre elles à l'aide des mécanismes d'E/S (java.io)

Différents types de sockets :

Stream Sockets (TCP)

établir une communication en mode connecté

si connexion interrompue : applications informées

Datagram Sockets (UDP)

établir une communication en mode non connecté

données envoyées sous forme de paquets

indépendants de toute connexion. Plus rapide, moins fiable que TCP.



Applications TCP les plus connues :
FTP, SMTP, TELNET ...

Applications UDP les plus connues :
Simple Network Management Protocol (SNMP)
Trivial File Transfer Protocol (TFTP): version
datagramme de FTPD (pour le boot par le réseau)

6 – Socket -> Le modèle client-serveur Java



Serveur

- Enregistrer le service:
serverSocket(port,#nb,_
cnx)
- Attendre une connexion
client accept()
- retourne un objet
Socket
- Utiliser le socket
- close

```
public class Serveur {  
  
    public static void main(String[] zero) {  
  
        ServerSocket socketserver;  
  
        Socket socketduserveur;  
  
        try {  
  
            socketserver = new ServerSocket(2009);  
            socketduserveur = socketserver.accept();  
            System.out.println("Un zéro s'est connecté !");  
            socketserver.close();  
            socketduserveur.close();  
  
        } catch (IOException e) {  
  
            e.printStackTrace();  
  
        }  
  
    }  
  
}
```


6 – Socket -> Le modèle client-serveur Java



Client

- Etablir la connexion:
Socket(host,port) =>
création d'un objet
Socket : Socket
- Utiliser le socket
- close

```
public class Client {  
  
    public static void main(String[] zero) {  
  
        Socket socket;  
        BufferedReader in;  
        PrintWriter out;  
  
        try {  
  
            socket = new Socket(InetAddress.getLocalHost(), 2009);  
            System.out.println("Demande de connexion");  
  
            in = new BufferedReader(new InputStreamReader(socket.getInputStream()));  
            String message_distant = in.readLine();  
            System.out.println(message_distant);  
  
            socket.close();  
  
        } catch (UnknownHostException e) {  
  
            e.printStackTrace();  
        } catch (IOException e) {  
  
            e.printStackTrace();  
        }  
    }  
}
```

6 – Socket -> Le modèle client-serveur Java



Serveur

- Enregistrer le service:
serverSocket(port,#nb,_
cnx)
- Attendre une connexion
client accept()
- retourne un objet
Socket
- Utiliser le socket
- close

```
public class ConnexionServeur extends Connexion {

    private ServerSocket serverSocket;
    private int fileAttente = 100;

    public ConnexionServeur() {
        port = 11111;
        IP = "192.168.1.44";
        try {
            serverSocket = new ServerSocket(port, fileAttente, InetAddress.getByName(IP));
        } catch (IOException e) {
            System.out.println("Probleme " + e);
        }
    }

    public void accept(TextArea Communication, String nom) {
        try {
            Socket socket = serverSocket.accept();
            entree = new BufferedReader(new InputStreamReader(socket.getInputStream()));
            sortie = new PrintStream(socket.getOutputStream());
            this.reception(Communication, nom);
        } catch (IOException e) {
            System.out.println("Probleme " + e);
        }
    }

    public void close() throws IOException {
        serverSocket.close();
    }
}
```

6 – Socket -> Le modèle client-serveur Java



Client

- Etablir la connexion:
Socket(host,port) =>
création d'un objet
Socket : Socket
- Utiliser le socket
- close

```
public class ConnexionClient extends Connexion {

    private Socket socket;

    public ConnexionClient(TextArea Communication, String serveur, String nom) {

        port = 11111;
        IP = "192.168.1.44";
        try {
            socket = new Socket(serveur, port);
            entree = new BufferedReader(new InputStreamReader(socket.getInputStream()));
            sortie = new PrintStream(socket.getOutputStream());
            this.reception(Communication, nom);
        } catch (IOException e) {
            System.out.println("Probleme : " + e);
        }
    }

    public void close() throws IOException{

        socket.close();
    };
}
```

6 – Socket -> URL



```
URL url = new URL("https://www.yantra-technologies.com/index.html");
DataInputStream dis = new DataInputStream(url.openStream());
String line;
while ((line = dis.readUTF()) != null) {
    System.out.println(line);
}
```

7 - Les interfaces graphiques



- Présentation
- Architecture Modèle-Vue-Contrôleur
- LayoutManager
- Composant
- Gestion des évènements
- Menu
- Boites de dialogue
- Netbeans



Les interfaces graphiques

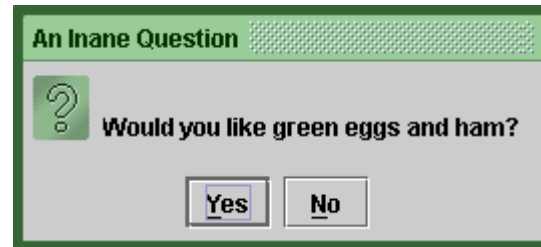
- Abstract Window Toolkit : AWT
 - fenêtrage natif
 - Look and feel des applications natives
- Java Foundation Classes (JFC) : Swing
 - construit sur AWT
 - plus de fonctionnalités
 - look and feel configurable
 - architecture Modèle-Vue-Contrôleur

7 - Les interfaces graphiques -> Présentation

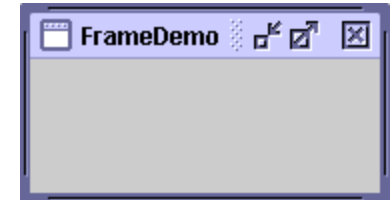


<http://java.sun.com/docs/books/tutorial/uiswing/components/components.html>

Containers de haut niveau

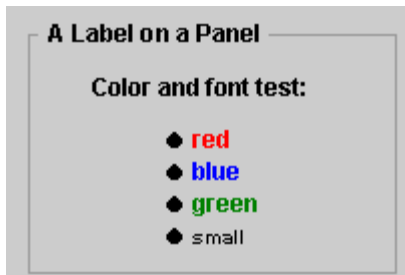


Dialog

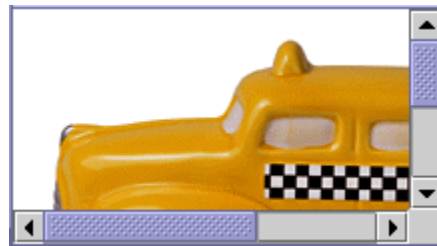


Frame

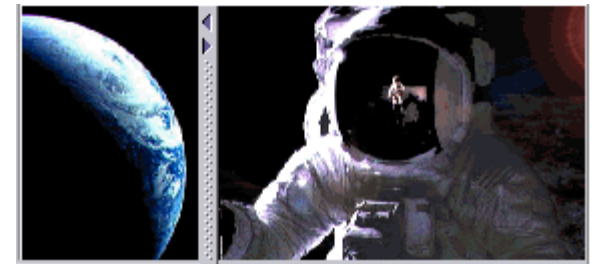
Containers généraux



Panel



Scrollpane



Splitpane



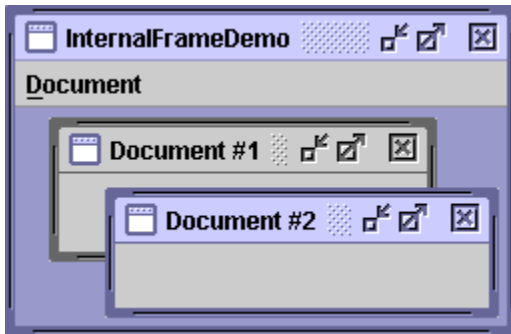
TabbedPane



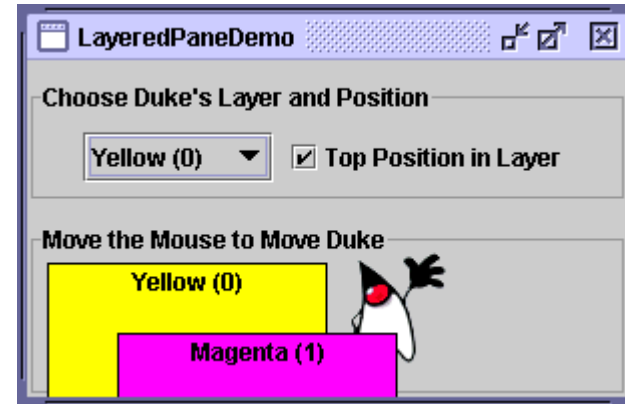
ToolBar



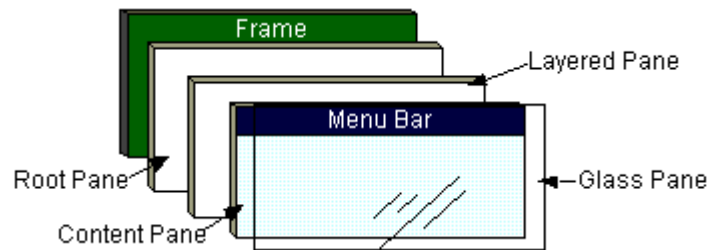
Containers spéciaux



Internal frame



Layred pane



root pane

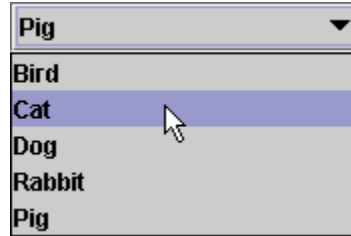
7 - Les interfaces graphiques -> Présentation



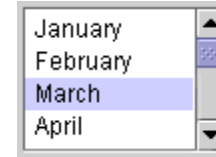
Basics



Buttons



Combo box



List



Menu



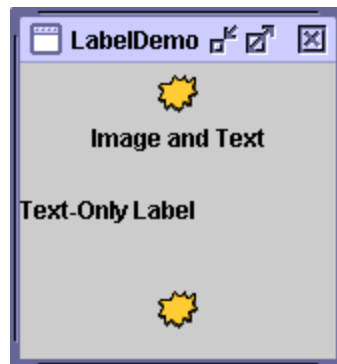
Slider



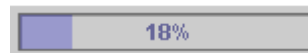
textfield



Spinner



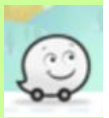
Label



Progress bar



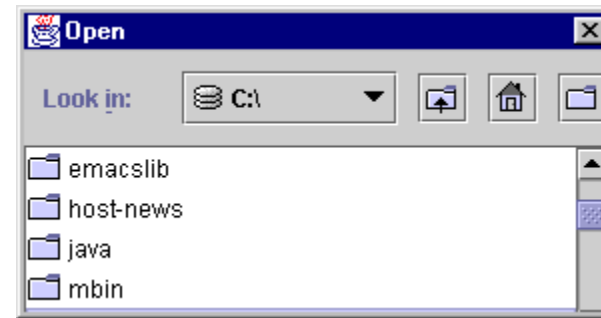
toolTip



Ecrans Interactifs



Color chooser



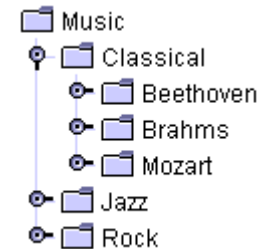
File chooser

First Name	Last Name	Favorite Food
Jeff	Dinkins	
Ewan	Dinkins	
Amy	Fowler	
Hania	Gajewska	
David	Geary	

Table



Text



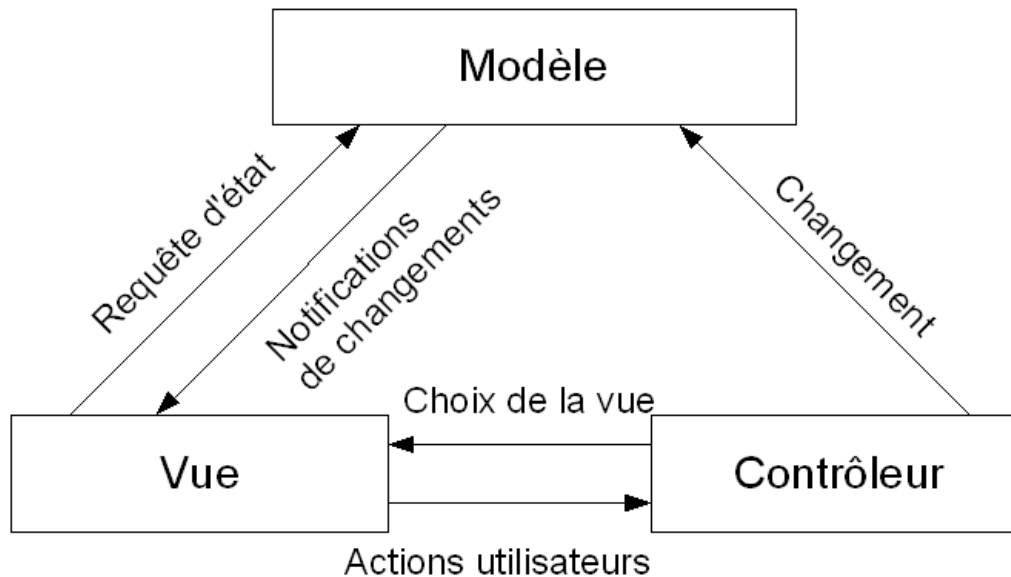
Tree

7 - Les interfaces graphiques

-> Architecture Modèle-Vue-Contrôleur



- **Le modèle** : représente les données de l'application..
- **La vue** : représente l'interface utilisateur, Il peut tout à fait y avoir plusieurs vues qui présentent les données d'un même modèle.
- **Le contrôleur** : gère l'interface entre le modèle et le client. Il effectue la synchronisation entre le modèle et les vues.



<http://baptiste-wicht.developpez.com/tutoriels/conception/mvc/#LI>

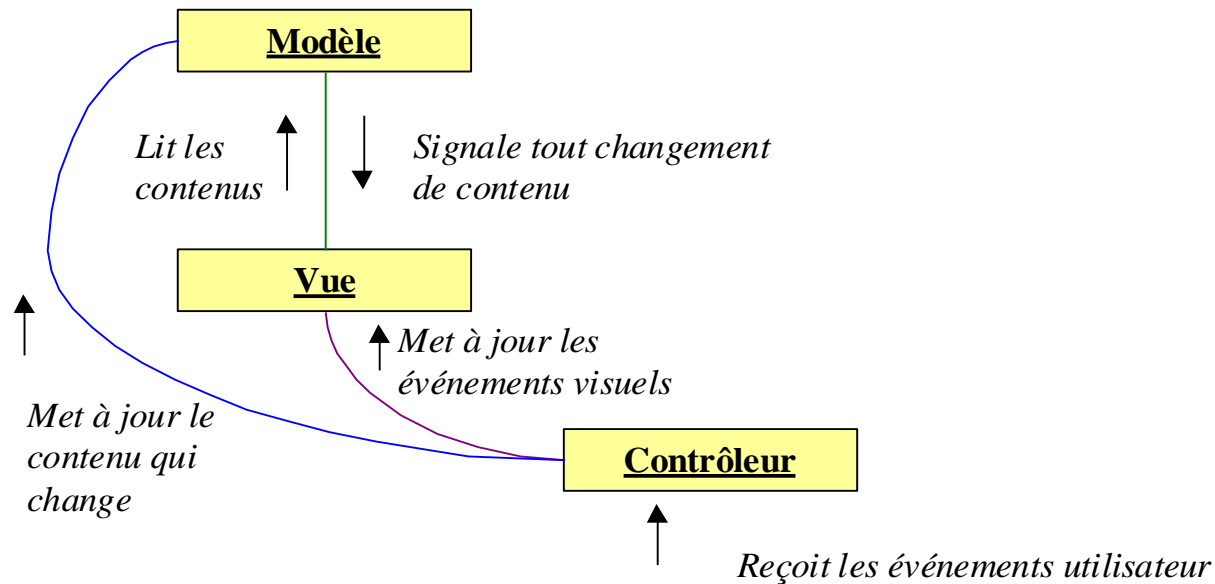
7 - Les interfaces graphiques

-> Architecture Modèle-Vue-Contrôleur

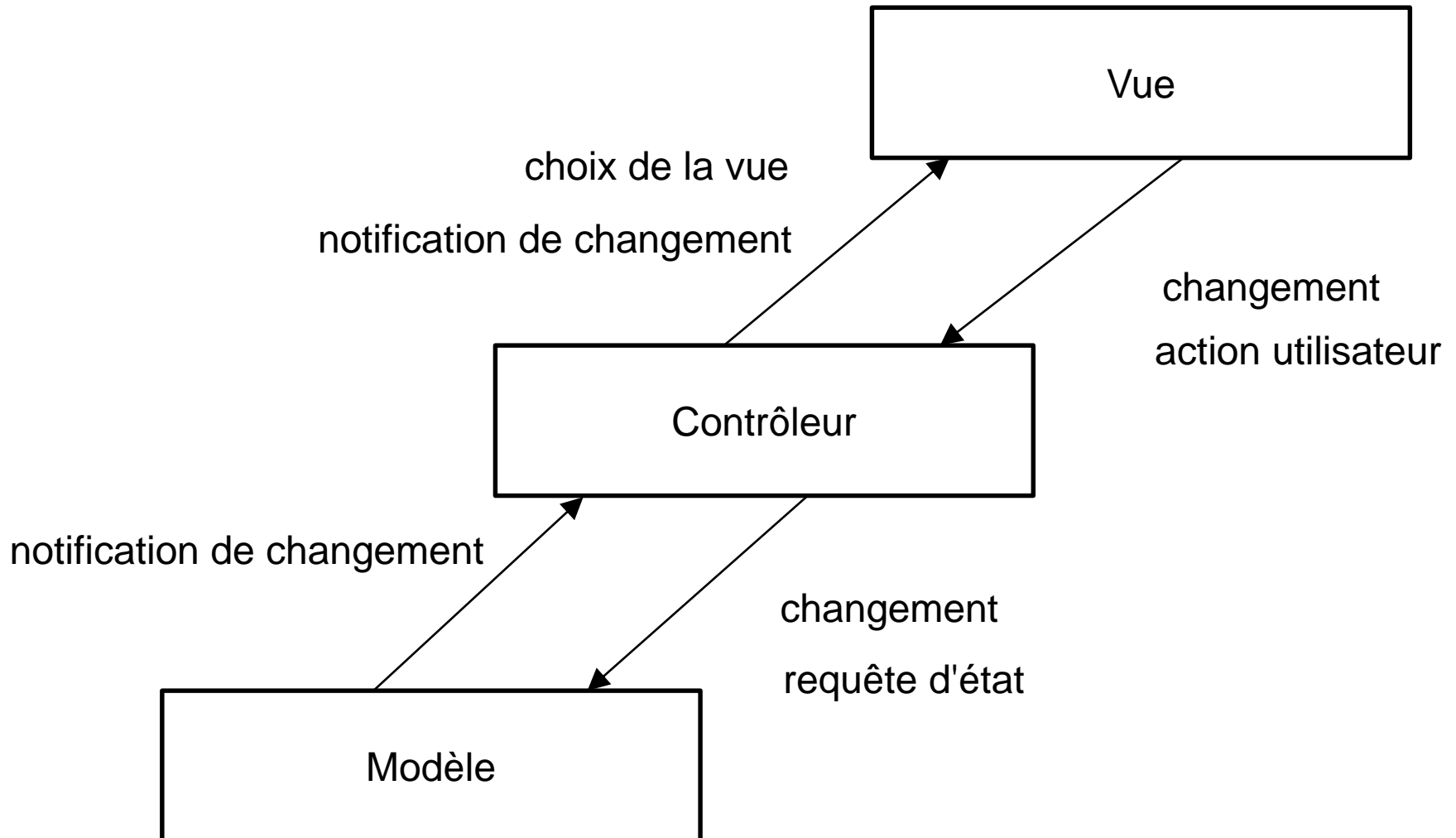


On implémente 3 classes séparées :

- le modèle qui maintient le contenu (état du bouton activé, valeur d'un champ)
- la vue qui affiche le contenu (couleur, taille)
- le contrôleur qui gère l'entrée utilisateur

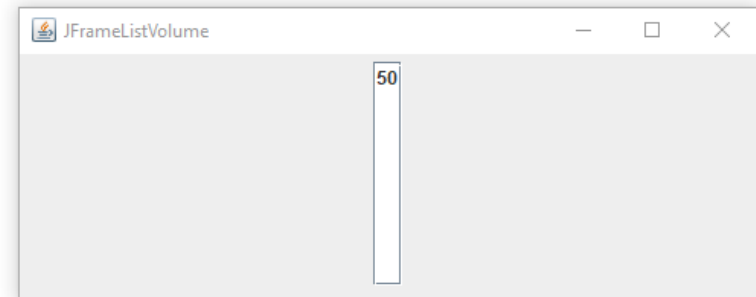
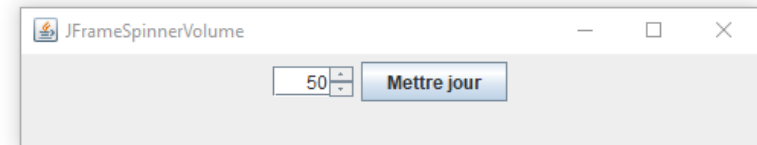
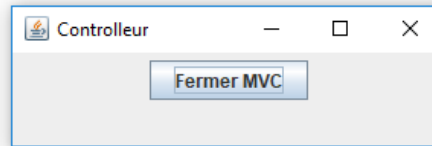


7 - Les interfaces graphiques -> Architecture Modèle-Vue-Contrôleur



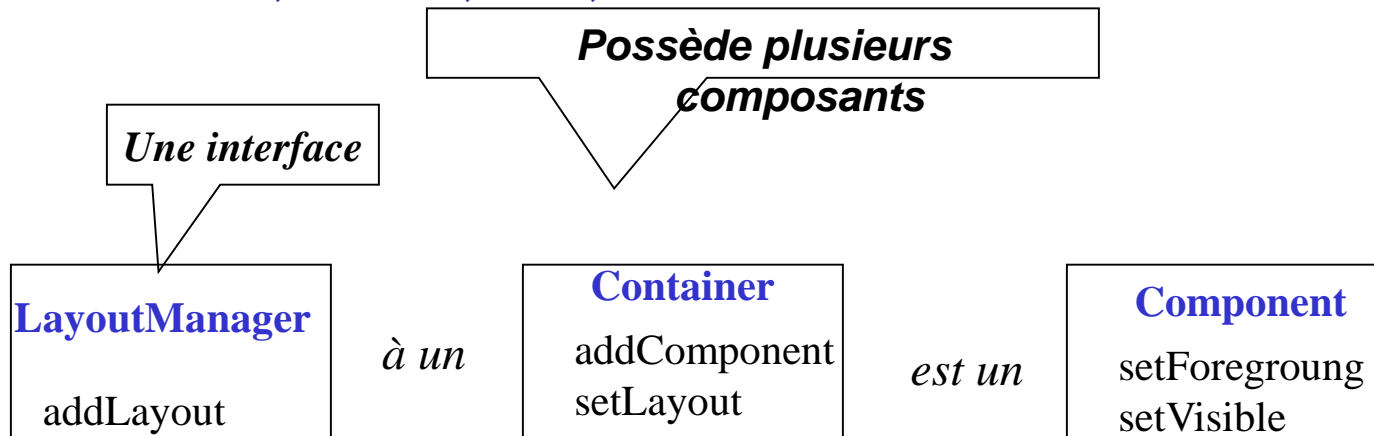
7 - Les interfaces graphiques

-> Architecture Modèle-Vue-Contrôleur





- Le layoutManager est une interface qui gère le placement des Components dans un Container
FlowLayout, BorderLayout, GridLayout, GridBagLayout, CardLayout
- chaque Container a un Layoutmanager
Jframe, Jpanel, Jdialog ...
- le Container contient une collection de Components
JLabel, JButton, JList, JTextField ...



7 - Les interfaces graphiques -> LayoutManager



- FlowLayout : le plus simple, place les composants de gauche à droite (défaut des panel)
- BorderLayout : Cinq zones NORTH, EAST, WEST, SOUTH et CENTER (défaut des Frame)
- GridLayout : lignes et colonnes régulièrement espacées
- GridBagLayout : agencement complexe
- CardLayout : une fiche visible à la fois

7 - Les interfaces graphiques -> LayoutManager : BorderManager

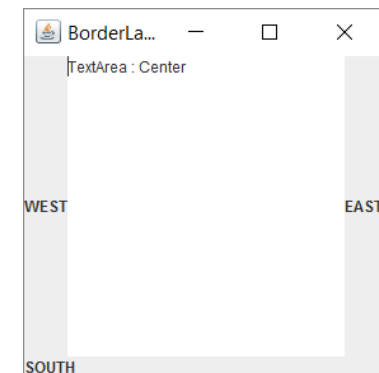
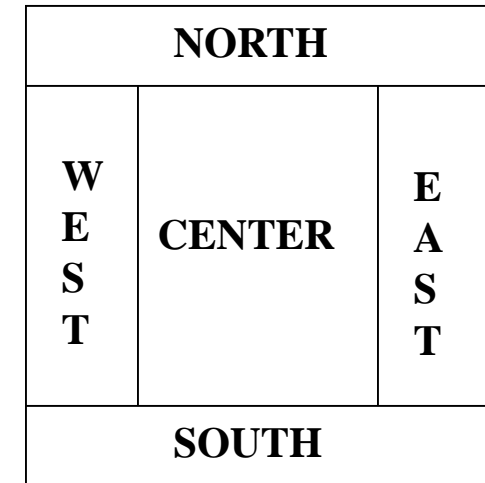


```
package exemplecours.IHM;

import java.awt.*;
import javax.swing.*;

public class BorderFrame extends JFrame {
    private final JTextArea texteArea;
    public BorderFrame(String titre) {
        setTitle(titre);
        // mise en place du BorderLayout
        Container contentPane = getContentPane();
        contentPane.setLayout(new BorderLayout());
        // ajout d'un TextArea au centre
        texteArea = new JTextArea("TextArea : Center");
        contentPane.add(texteArea, BorderLayout.CENTER);
        // ajout d'un label au Sud
        contentPane.add(new JLabel("SOUTH"), BorderLayout.SOUTH);
        // ajout d'un label au East
        contentPane.add(new JLabel("EAST"), BorderLayout.EAST);
        // ajout d'un label au West
        contentPane.add(new JLabel("WEST"), BorderLayout.WEST);

        setSize(300, 300);
        setVisible(true);
    }
    public static void main(String[] args) {
        BorderFrame frame = new BorderFrame("BorderLayout");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```



7 - Les interfaces -> LayoutManager : GridLayout



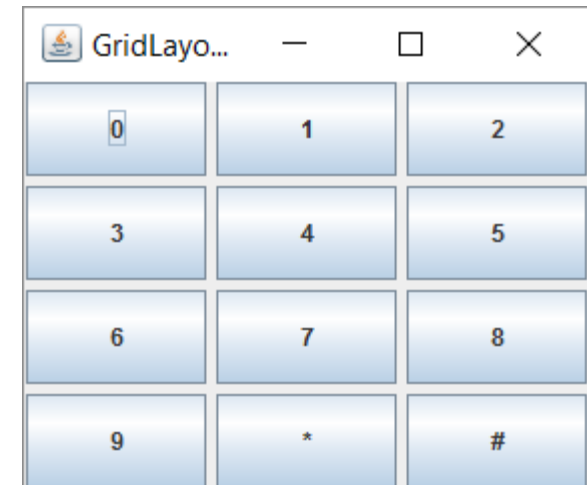
```
import java.awt.*;
import javax.swing.*;

public class GridFrame extends JFrame {

    private final JButton[] buttons;

    public GridFrame(String titre) {
        setTitle(titre);
        Container contentPane = getContentPane();
        // definition du Grid avec 4 lignes, 3 colonnes et 5 pixels d'espace
        // entre les composants
        contentPane.setLayout(new GridLayout(4, 3, 5, 5));
        // creation des douze boutons
        buttons = new JButton[12];
        for (int i = 0; i < 10; i++) {
            buttons[i] = new JButton(Integer.toString(i));
            contentPane.add(buttons[i]);
        }
        buttons[10] = new JButton("*");
        buttons[11] = new JButton("#");
        contentPane.add(buttons[10]);
        contentPane.add(buttons[11]);
        setSize(300, 250);
        setVisible(true);
    }

    public static void main(String[] args) {
        GridFrame frame = new GridFrame("GridLayout");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```





7 - Les interfaces graphiques : Composants

Component

JLabel, JButton, JList, JTextField, JTextArea, JToggleButton, JComboBox

□ □ □

- Propriétés ajustables :
 - **Police** : setFont , getFont setFont (new Font(« Serif », Font.BOLD,14));
 - **Couleur de premier plan** : setForeground , getForeground
setForeground (Color.blue);
 - **couleur du fond** : setBackground, getBackground
setBackground(Color.white);
 - **curseur** : setCursor, getCursor
setCursor(Cursor.getPredefinedCursor, Cursor.WAIT_CURSOR);
 - **focus** : setFocus, getFocus setFocus()
 - **visibilité** : setVisible, getVisible setVisible(true);
 - **activation** : setEnabled, getEnabled setEnabled(false);
 - **taille** : setSize, getSize setSize(100,200)



7 - Les interfaces graphiques

-> Gestion des Événements : Fonctionnement des événements



- Un composant enregistre des auditeurs d'événements (Listeners)
- l'événement est envoyé aux auditeurs enregistrés
- chaque auditeur définit les actions à faire.
 - Un Button -> ActionListener
 - clic -> ActionEvent -> ActionListener enregistré
 - provoque l'exécution de la méthode `ActionPerformed` de chaque ActionListener



Les événements

interface

ActionListener

actionPerformed(e:ActionEvent)

a un

JButton

addActionListener()

*Utilisé pour
inscrire
un listener*

implements

InfoListener

actionPerformed(e:ActionEvent)

Classe utilisateur

If (e.getSource() == infoButton)
infoLabel.setText(...);

Méthode utilisée quand
le bouton est pressé

7 - Les interfaces graphiques

-> Gestion des Événements



```
package exemplecours.IHM;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class EventFrame extends JFrame {

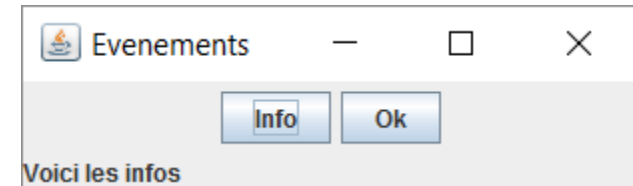
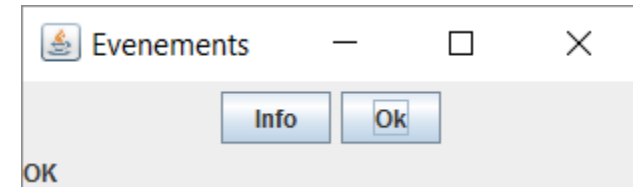
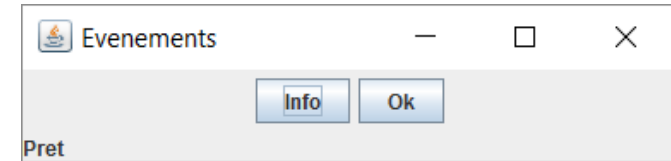
    private final JButton infoButon, okBouton;
    private final JLabel infoLabel;

    public EventFrame(String titre) {...24 lines }

    class InfoListener implements ActionListener {...11 lines }

    public static void main(String[] args) {...4 lines }

}
```

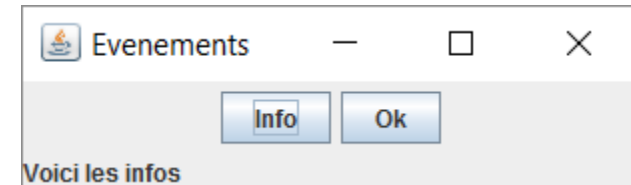
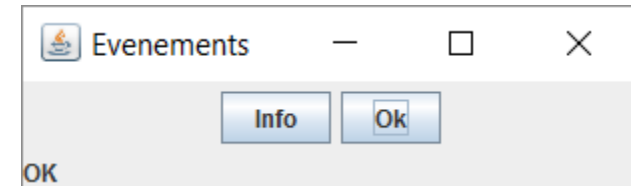
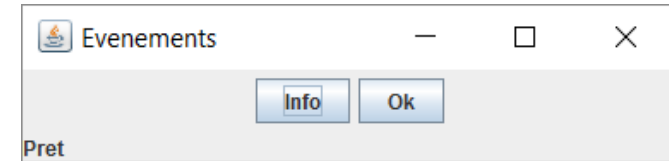


7 - Les interfaces graphiques

-> Gestion des Événements



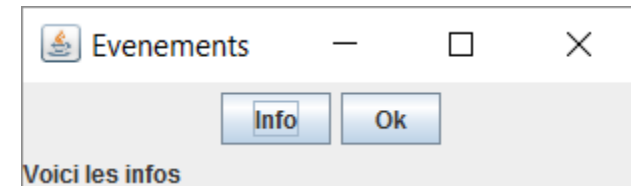
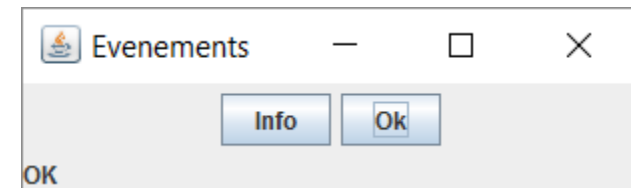
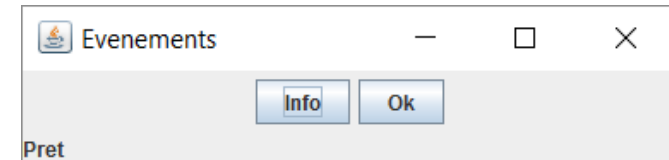
```
public EventFrame(String titre) {
    setTitle(titre);
    getContentPane().setLayout(new BorderLayout());
    // Definition du Panel et du gestionnaire
    JPanel boutonPanel = new JPanel();
    boutonPanel.setLayout(new FlowLayout());
    // creation d'un InfoListener
    InfoListener listener = new InfoListener();
    // creation du bouton Info
    infoBouton = new JButton("Info");
    infoBouton.addActionListener(listener);
    boutonPanel.add(infoBouton);
    // creation du bouton OK
    okBouton = new JButton("Ok");
    okBouton.addActionListener(listener);
    boutonPanel.add(okBouton);
    getContentPane().add(boutonPanel, BorderLayout.CENTER);
    // creation du Label
    infoLabel = new JLabel("Pret");
    getContentPane().add(infoLabel, BorderLayout.SOUTH);
    //mise en place de la fenetre
    setSize(400, 100);
    setVisible(true);
}
```



7 - Les interfaces graphiques -> Gestion des Événements



```
class InfoListener implements ActionListener {  
  
    public void actionPerformed(ActionEvent e) {  
        if (e.getSource() == infoBouton) {  
            infoLabel.setText("Voici les infos");  
        }  
        if (e.getSource() == okBouton) {  
            infoLabel.setText("OK");  
        }  
    }  
}  
  
public static void main(String[] args) {  
    EventFrame application = new EventFrame("Evenements");  
    application.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
}
```





Principaux événements

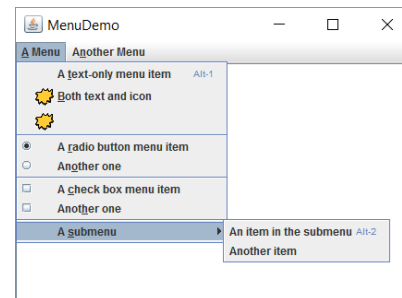
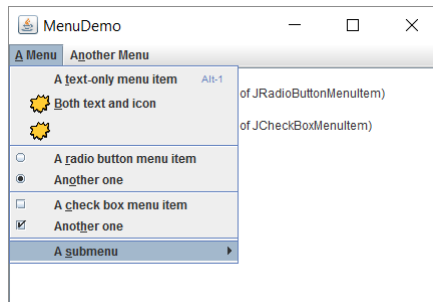
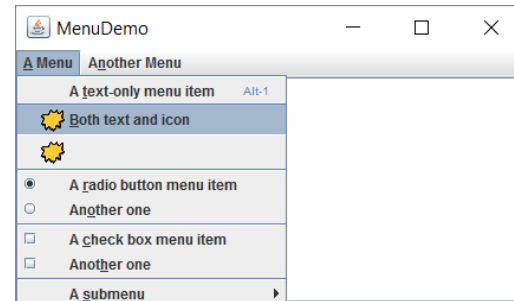
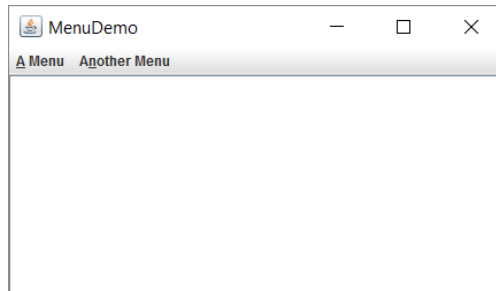
Composant	Événement	Méthode
JButton	ActionEvent	actionPerformed
JCheckBox	ItemEvent	itemStateChanged
JList	ListSelectionEvent	valueChanged
TextField	ActionEvent	actionPerformed
JWindow	WindowEvent	windowOpened/Closed...
JComponent	ComponentEvent	componentMoved/Resized...
JComponent	FocusEvent	focusLost, focusGained
JComponent	KeyEvent	KeyPressed/Released
JComponent	MouseEvent	MouseClicked/Pressed...



Les menus

- menus fixes en haut d'une JFrame : **JMenuBar**
- menus dynamiques avec la souris : **JPopupMenu**
- classes **JMenu** et **JMenuItem**

File	
Nouveau	
Ouvrir	
Exit	





JOptionPane

```
JOptionPane.showMessageDialog (null, « Message »,  
    « Ceci est mon message »,  
    JOptionPane.PLAIN_MESSAGE) ;
```

- 1 : null ou Frame
- 2 : titre
- 3 : message
- 4 : type de message

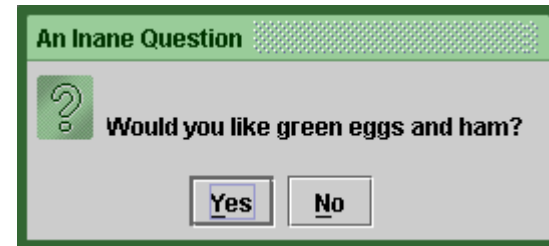
ERROR_MESSAGE

INFORMATION_MESSAGE

WARNING_MESSAGE

QUESTION_MESSAGE

PLAIN_MESSAGE



7 - Les interfaces graphiques : Boîtes de dialogue



```
package exemplecours.IHM;

import java.awt.*;
import java.awt.event.*;
import java.net.URL;
import javax.swing.*;

/*
 * This class is just like MenuLookDemo, except the menu items
 * actually do something, thanks to event listeners.
 */
public class MenuDemo implements ActionListener, ItemListener {

    private JTextArea output;
    private JScrollPane scrollPane;
    private final String newline = "\n";

    public JMenuBar createMenuBar() { ...87 lines }

    public Container createContentPane() { ...15 lines }

    @Override
    public void actionPerformed(ActionEvent e) { ...9 lines }

    @Override
    public void itemStateChanged(ItemEvent e) { ...12 lines }

    // Returns just the class name -- no package info.
    protected String getClassName(Object o) { ...5 lines }

    //Returns an ImageIcon, or null if the path was invalid.
    protected static ImageIcon createImageIcon(String path) { ...6 lines }

    private static void createAndShowGUI() { ...14 lines }

    public static void main(String[] args) { ...11 lines }
}
```

7 - Les interfaces graphiques : Boîtes de dialogue



```
private static void createAndShowGUI() {
    //Create and set up the window.
    JFrame frame = new JFrame("MenuDemo");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    //Create and set up the content pane.
    MenuDemo demo = new MenuDemo();
    frame.setJMenuBar(demo.createMenuBar());
    frame.setContentPane(demo.createContentPane());

    //Display the window.
    frame.setSize(450, 260);
    frame.setVisible(true);
}

public static void main(String[] args) {
    /*
    javax.swing.SwingUtilities.invokeLater(new Runnable()
        @Override
        public void run() {createAndShowGUI();});
    remplaceur par lambda fonctions
    */
    javax.swing.SwingUtilities.invokeLater(() -> {
        createAndShowGUI();
    });
}
```

7 - Les interfaces graphiques : Boîtes de dialogue



```
public Container createContentPane() {  
    JPanel contentPane = new JPanel(new BorderLayout());  
    contentPane.setOpaque(true);  
  
    this.output = new JTextArea(5, 30);  
    this.output.setEditable(false);  
    this.scrollPane = new JScrollPane(output);  
  
    //Add the text area to the content pane.  
    contentPane.add(this.scrollPane, BorderLayout.CENTER);  
  
    return contentPane;  
}
```

```
// Returns just the class name -- no package info.  
protected String getClassName(Object o) {  
    String classString = o.getClass().getName();  
    int dotIndex = classString.lastIndexOf(".");  
    return classString.substring(dotIndex + 1);  
}
```

```
//Returns an ImageIcon, or null if the path was invalid.  
protected static ImageIcon createImageIcon(String path) {  
    URL imgURL = MenuDemo.class.getResource(path);  
    if (imgURL != null) return new ImageIcon(imgURL);  
    System.err.println("Couldn't find file: " + path);  
    return null;  
}
```

7 - Les interfaces graphiques : Boîtes de dialogue



```
public JMenuBar createMenuBar() {
    JMenuBar menuBar;
    JMenu menu, submenu;
    JMenuItem menuItem;
    JRadioButtonMenuItem rbMenuItem;
    JCheckBoxMenuItem cbMenuItem;

    //Create the menu bar.
    menuBar = new JMenuBar();

    //Build the first menu.
    menu = new JMenu("A Menu");
    menu.setMnemonic(KeyEvent.VK_A);
    menu.getAccessibleContext().setAccessibleDescription("The only menu in this program that has menu items");
    menuBar.add(menu);

    //a group of JMenuItem
    menuItem = new JMenuItem("A text-only menu item", KeyEvent.VK_T);
    menuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_1, ActionEvent.ALT_MASK));
    menuItem.getAccessibleContext().setAccessibleDescription("This doesn't really do anything");
    menuItem.addActionListener(this);
    menu.add(menuItem);

    ImageIcon icon = createImageIcon("../Images/middle.gif");
    menuItem = new JMenuItem("Both text and icon", icon);
    menuItem.setMnemonic(KeyEvent.VK_B);
    menuItem.addActionListener(this);
    menu.add(menuItem);

    menuItem = new JMenuItem(icon);
    menuItem.setMnemonic(KeyEvent.VK_D);
    menuItem.addActionListener(this);
    menu.add(menuItem);
}
```

7 - Les interfaces graphiques : Boites de dialogue



```
//a group of radio button menu items
menu.addSeparator();
ButtonGroup group = new ButtonGroup();

rbMenuItem = new JRadioButtonMenuItem("A radio button menu item");
rbMenuItem.setSelected(true);
rbMenuItem.setMnemonic(KeyEvent.VK_R);
group.add(rbMenuItem);
rbMenuItem.addActionListener(this);
menu.add(rbMenuItem);

rbMenuItem = new JRadioButtonMenuItem("Another one");
rbMenuItem.setMnemonic(KeyEvent.VK_O);
group.add(rbMenuItem);
rbMenuItem.addActionListener(this);
menu.add(rbMenuItem);

//a group of check box menu items
menu.addSeparator();
cbMenuItem = new JCheckBoxMenuItem("A check box menu item");
cbMenuItem.setMnemonic(KeyEvent.VK_C);
cbMenuItem.addItemListener(this);
menu.add(cbMenuItem);

cbMenuItem = new JCheckBoxMenuItem("Another one");
cbMenuItem.setMnemonic(KeyEvent.VK_H);
cbMenuItem.addItemListener(this);
menu.add(cbMenuItem);
```


7 - Les interfaces graphiques : Boites de dialogue



```
//a submenu
menu.addSeparator();
submenu = new JMenu("A submenu");
submenu.setMnemonic(KeyEvent.VK_S);

menuItem = new JMenuItem("An item in the submenu");
menuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_2, ActionEvent.ALT_MASK));
menuItem.addActionListener(this);
submenu.add(menuItem);

menuItem = new JMenuItem("Another item");
menuItem.addActionListener(this);
submenu.add(menuItem);
menu.add(submenu);

//Build second menu in the menu bar.
menu = new JMenu("Another Menu");
menu.setMnemonic(KeyEvent.VK_N);
menu.getAccessibleContext().setAccessibleDescription("This menu does nothing");
menuBar.add(menu);

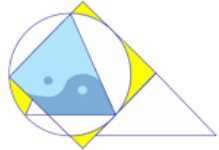
return menuBar;
}
```

7 - Les interfaces graphiques : Boîtes de dialogue

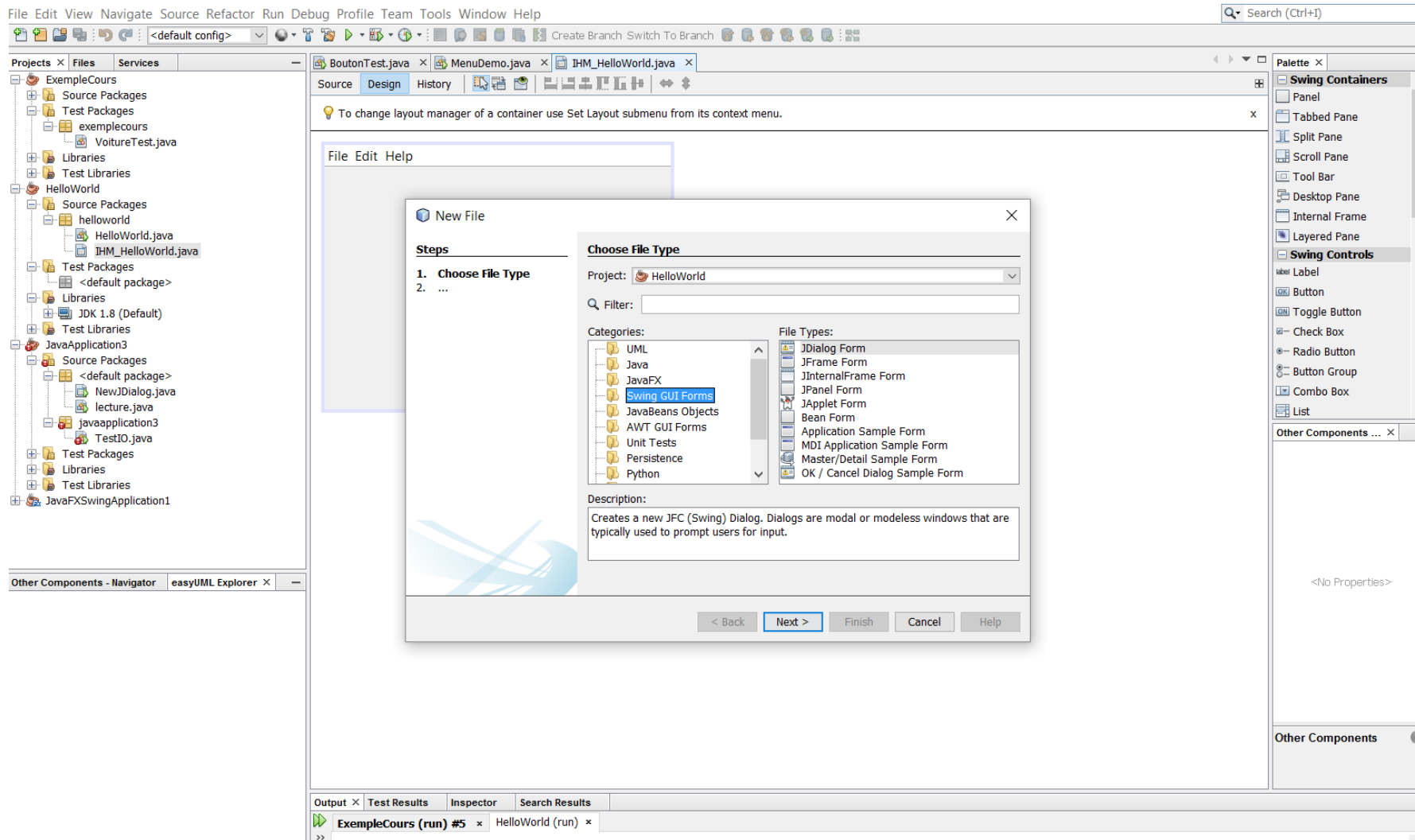


```
@Override
public void actionPerformed(ActionEvent e) {
    JMenuItem source = (JMenuItem) (e.getSource());
    String s = "Action event detected."
        + this.newline
        + "    Event source: " + source.getText()
        + " (an instance of " + getClassName(source) + ")";
    this.output.append(s + this.newline);
    this.output.setCaretPosition(this.output.getDocument().getLength());
}

@Override
public void itemStateChanged(ItemEvent e) {
    JMenuItem source = (JMenuItem) (e.getSource());
    String s = "Item event detected."
        + this.newline
        + "    Event source: " + source.getText()
        + " (an instance of " + getClassName(source) + ") "
        + this.newline
        + "    New state: "
        + ((e.getStateChange() == ItemEvent.SELECTED) ? "selected" : "unselected");
    this.output.append(s + this.newline);
    this.output.setCaretPosition(this.output.getDocument().getLength());
}
```



7 - Les interfaces graphiques : NetBeans



8 – Thread

8.1 – Définition : Thread

8.2 - Threads et multitâches

8.3 - Avantages et inconvénients

8.4 - Création et exécution des threads

8.5 - Annulation et fin des threads

8.6 - Définition : Mutex

8.7 – Fonctions : Mutex

8.1 – Définition : Thread

Un thread ou fil (d'exécution) ou tâche ou processus léger est similaire à un processus car tous deux représentent l'exécution d'un ensemble d'instructions du langage machine d'un processeur.

Du point de vue de l'utilisateur, ces exécutions semblent se dérouler en parallèle.

Toutefois, là où chaque processus possède sa propre mémoire virtuelle, les threads d'un même processus se partagent sa mémoire virtuelle.

Par contre, tous les threads possèdent leur propre pile d'appel.

[http://fr.wikipedia.org/wiki/Thread_\(informatique\)](http://fr.wikipedia.org/wiki/Thread_(informatique))

8.2 - Threads et multitâches

Les threads se distinguent du **multi-processus** plus classique par le fait que deux processus sont typiquement **indépendants** et peuvent interagir uniquement à travers une API fournie par le système telle que IPC.

D'un autre côté les **threads** partagent une information sur l'état du processus, des zones de mémoires ainsi que d'autres ressources. Puisqu'il n'y a pas de changement de mémoire virtuelle, la commutation de contexte entre 2 threads est **moins coûteuse** que la commutation de contexte entre 2 processus.

On peut y voir un avantage de la programmation utilisant des threads multiples.

8.3 - Avantages et inconvénients

Dans certains cas, les programmes utilisant des threads sont plus rapides que des programmes architecturés plus classiquement, en particulier sur les machines multi-processeurs.

Le principal surcoût dû à l'utilisation de processus multiples provient de la communication entre processus séparés.

Le partage de certaines ressources entre **threads** permet une communication plus efficace entre les différents threads d'un processus. Là où deux **processus** séparés doivent utiliser un **mécanisme** fourni par le **système** pour communiquer car **les threads partagent une partie de l'état du processus**.

8.3 - Avantages et inconvénients

La programmation utilisant des threads est plus difficile que la programmation multi-processus car l'accès à certaines ressources partagées doit être **restreint par le programme lui-même**, pour éviter que l'état d'un processus ne devienne temporairement incohérent, tandis qu'un autre thread va avoir besoin de consulter cette portion de l'état du processus.

Il est donc obligatoire de mettre en place des mécanismes de synchronisation (à l'aide de sémaphores par exemple), tout en conservant à l'esprit que l'utilisation de la synchronisation peut aboutir à des situations d'inter-blocage ce qui augmente la complexité du programme.

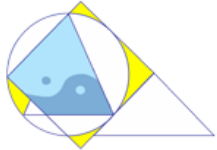
8.4 – Thread : exemple

```
public class Creation1Thread {  
  
    public static void main(String[] args) {  
        Thread t = new Thread() {  
            @Override  
            public void run() {  
                System.out.println("Traitement a faire ..... ");  
            }  
        };  
        t.start();  
    }  
}
```

8.4 – Thread : exemple

```
public class MonTraitement implements Runnable {  
  
    @Override  
    public void run() {  
        System.out.println("Mon Traitement a faire ..... ");  
        for (int i = 0; i < 10; i++) {  
            System.out.print(" " + i);  
        }  
        System.out.println();  
    }  
}
```

```
public class Creation2Thread {  
  
    public static void main(String[] args) {  
        Thread thread = new Thread(new MonTraitement());  
        thread.start();  
    }  
}
```



8.4 – Thread : exemple

```
public class ExempleArretThread extends Thread {

    @Override
    public void run() {
        System.out.println("Traitement a faire ..... ");
        int i = 0;
        while (true) {
            // Traitement
            System.out.print(" " + i);
            ++i;
            try {
                Thread.sleep(100); // Pause de 0.1 secondes
            } catch (InterruptedException ex) {
                Thread.currentThread().interrupt(); // Très important de réinterrompre
                break; // Sortie de la boucle infinie
            }
        }
    }

    public static void main(String[] args)
        throws InterruptedException {
        Thread t = new ExempleArretThread();
        t.start();
        Thread.sleep(1000); // Attente 1 seconde avant d'interrompre
        System.out.println();
        t.interrupt();
        t.join(); // Attente de la fermeture du thread.
        System.out.println("Fin ...");
    }
}
```

8.4 – Thread : exemple

```
public class Exemple2ArretThread extends Thread {  
  
    MonTraitement a_traitement;  
  
    public Exemple2ArretThread(MonTraitement monTraitement) {  
        super(monTraitement);  
    }  
    @Override  
    public void run() {  
        System.out.println("<" + this.getId() + "> Traitement a faire ..... ");  
        int i = 1;  
        while (true) {  
            System.out.print(" " + i + " ");  
            a_traitement.run();  
            ++i;  
            try {  
                Thread.sleep(100); // Pause de 0.1 secondes  
            } catch (InterruptedException ex) {  
                Thread.currentThread().interrupt(); // Très important de réinterrompre  
                break; // Sortie de la boucle infinie  
            }  
        }  
    }  
  
    public static void main(String[] args) throws InterruptedException {  
  
        Thread[] listthread = {  
            new Exemple2ArretThread(new MonTraitement()),  
            new Exemple2ArretThread(new MonTraitement()),  
            new Exemple2ArretThread(new MonTraitement())  
        };  
        for (Thread t : listthread) {  
            t.start();  
        }  
        Thread.sleep(1000); // Attente 1 seconde avant d'interrompre  
        for (Thread t : listthread) {  
            t.interrupt();  
        }  
        for (Thread t : listthread) {  
            t.join();  
        }  
        System.out.println("Fin ...");  
    }  
}
```

8.4 – Thread : exemple

```
public class Exemple3ArretThread extends Thread {

    MonTraitement a_traitement;

    public Exemple3ArretThread(ThreadGroup monThreadGroup, MonTraitement monTraitement) {
        super(monThreadGroup, monTraitement);
    }

    @Override
    public void run() {
        System.out.println("<" + this.getId() + "> Traitement a faire ..... ");
        int i = 1;
        while (true) {
            System.out.print(" " + i + " ");
            a_traitement.run();
            ++i;
            try {
                Thread.sleep(100); // Pause de 0.1 secondes
            } catch (InterruptedException ex) {
                Thread.currentThread().interrupt(); // Très important de réinterrompre

                break; // Sortie de la boucle infinie
            }
        }
    }

    public static void main(String[] args) throws InterruptedException {
        ThreadGroup monThreadGroup = new ThreadGroup("Mon groupe de threads");

        Thread[] listthread = {new Exemple2ArretThread(new MonTraitement()),
                                new Exemple2ArretThread(new MonTraitement()),
                                new Exemple2ArretThread(new MonTraitement())};

        for (Thread t : listthread) {
            t.start();
        }

        Thread.sleep(1000); // Attente 1 seconde avant d'interrompre
        System.out.println();
        monThreadGroup.interrupt();
        System.out.println("Fin ...");
    }
}
```

8.4 – Thread : exemple

```
public class ExempleSynchroniserThread extends Thread {

    static Semaphore mutex = new Semaphore(1); // 1 à la fois

    public ExempleSynchroniserThread(Runnable monTraitement) {
        super(monTraitement);
    }

    @Override
    public void run() {

        int i = 1;
        while (true) {

            try {
                mutex.acquire();
                traitement(i);
                mutex.release();
            } catch (InterruptedException ex) {
            }

            // synchronized (this) {
            //     traitement(i);
            // }
            ++i;
            try {
                Thread.sleep(100); // Pause de 0.1 secondes
            } catch (InterruptedException ex) {

                Thread.currentThread().interrupt(); // Très important de réinterrompre
                break; // Sortie de la boucle infinie
            }
        }
    }

    public synchronized void traitement(int i) {...7 lines }

    public static void main(String[] args) throws InterruptedException {...17 lines }
}
```

8.4 – Thread : exemple

```
public class ExempleSynchroniserThread extends Thread {

    static Semaphore mutex = new Semaphore(1); // 1 à la fois

    public ExempleSynchroniserThread(Runnable monTraitement) {...3 lines }

    @Override
    public void run() {...25 lines }

    public synchronized void traitement(int i) {
        System.out.println("<" + this.getId() + "> Traitement a faire ..... " + i + " ");
        for (int j = 0; j < 10; j++) {
            System.out.print(" " + j);
        }
        System.out.println();
    }

    public static void main(String[] args) throws InterruptedException {

        Thread[] listthread = {new ExempleSynchroniserThread(new MonTraitement()),
                                new ExempleSynchroniserThread(new MonTraitement()),
                                new ExempleSynchroniserThread(new MonTraitement())};

        for (Thread t : listthread) {
            t.start();
        }
        Thread.sleep(1000); // Attente 1 seconde avant d'interrompre
        for (Thread t : listthread) {
            t.interrupt();
        }

        System.out.println();
        System.out.println("Fin ...");
    }
}
```


9 - JUNIT

JUnit est un Framework de tests unitaires pour le langage de programmation Java.

JUnit définit deux types de fichiers de tests.

- Les **TestCase** (cas de test) sont des classes contenant un certain nombre de méthodes de tests. Un TestCase sert généralement à tester le bon fonctionnement d'une classe. Dans un TestCase il n'y a pas de *main* méthode, chaque test étant indépendant.
- Une **TestSuite** permet d'exécuter un certain nombre de TestCase déjà définis.

<https://www.jmdoudoux.fr/java/dej/chap-junit.htm>

<https://fr.wikipedia.org/wiki/JUnit>



Yantra

9 - JUNIT

```
import junit.framework.TestCase;  
import org.junit.*;
```

```
public class nomdemaiclass extends TestCase{
```

```
    @BeforeClass
```

```
    public static void setUpClass() throws Exception {  
        // Code exécuté avant l'exécution du premier test (et de la méthode @Before)  
    }
```

```
    @AfterClass
```

```
    public static void tearDownClass() throws Exception {  
        // Code exécuté après l'exécution de tous les tests  
    }
```

```
    @Before
```

```
    public void setUp() throws Exception {  
        // Code exécuté avant chaque test  
    }
```

```
    @After
```

```
    public void tearDown() throws Exception {  
        // Code exécuté après chaque test  
    }
```

```
    @Test
```

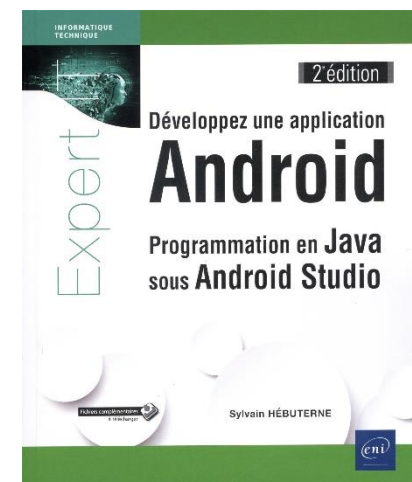
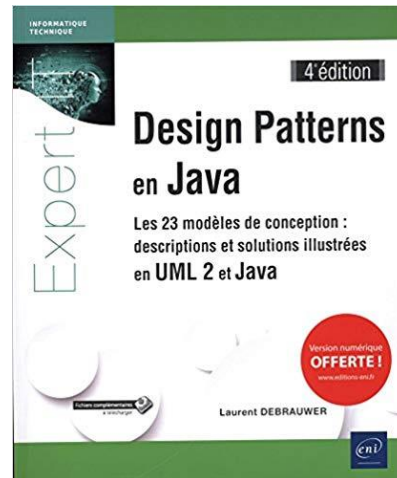
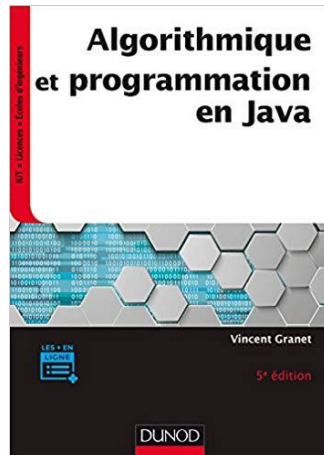
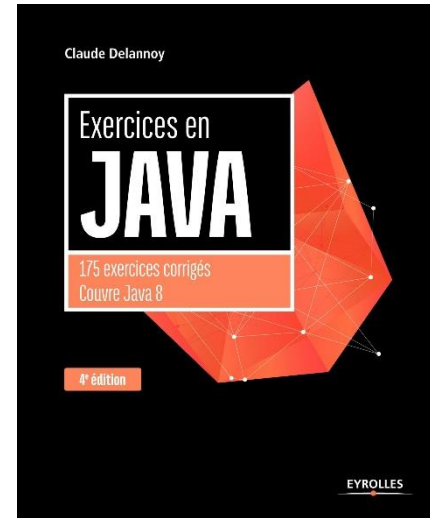
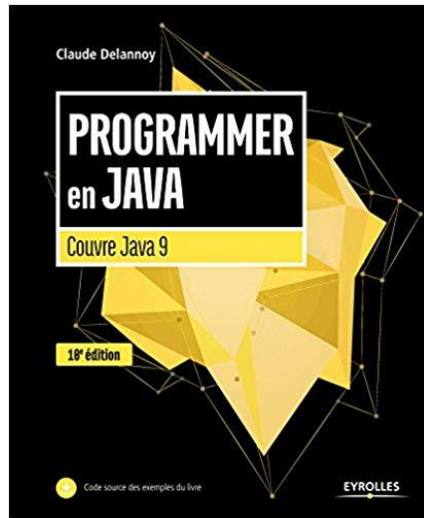
```
    public void nomdutest1() {  
        // code qui teste une chose, appelé "test1".  
        // Le code contient généralement une assertion pour vérifier si une condition est vraie ou fausse.  
    }
```

```
    @Test
```

```
    public void nomdutest2() {  
        // code qui teste autre chose, appelé "test2"  
        // Le code contient généralement une assertion pour vérifier si une condition est vraie ou fausse.  
    }
```

```
{  
}
```

10 - Bibliographies



Java :

- <http://gaetan.dussaux.free.fr/cours/java/12.htm#autres>
- <http://www.jmdoudoux.fr/java/dej/indexavecframes.htm>
- <http://www.tutorialspoint.com/java/>

Netbeans :

- <http://www.bestcours.com/cours-pdf-netbeans-java>

Android :

- <http://www.univ-orleans.fr/lifo/Members/Jean-Francois.Lalande/enseignement/android/cours-android.pdf>
- <https://olegoaer.developpez.com/cours/mobile/>
- <https://perso.univ-rennes1.fr/pierre.nerzic/Android/poly.pdf>