

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 - 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

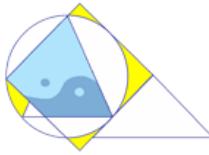
<http://jmdoudoux.developpez.com/cours/developpons/java/chap-presentation.php>

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 économique	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	Événements
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

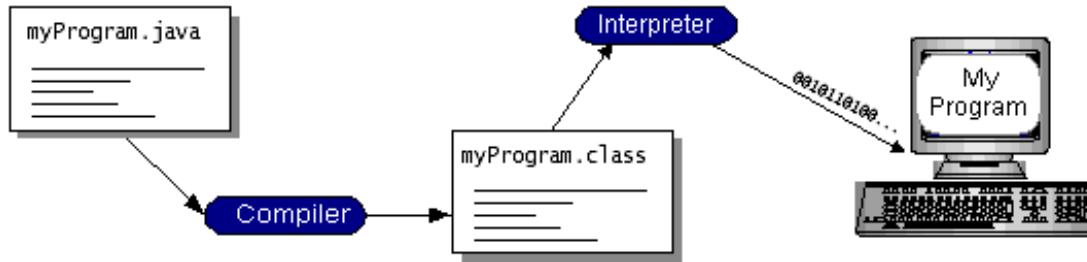
1 - Généralités -> Historique



Année	Événements
03/2014	Java 8
09/2017	Java 9
03/2018	Java 10
09/2018	java 11
03/2019	Java 12
09/2019	Java13
03/2020	Java 14
09/2020	Java 15
03/2021	Java 16
09/2021	Java 17

De nouvelles versions de Java tous **les six mois** depuis la sortie de Java 9 en septembre 2017
=> Java 9 date de moins de 3 ans, la dernière version est maintenant Java 17. (java 18 prévue le 03/2021)

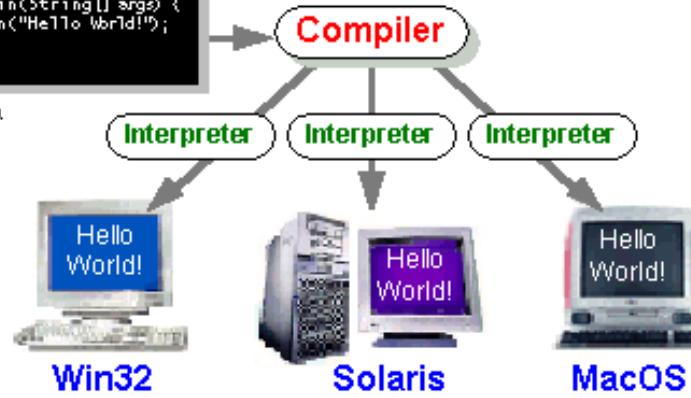
1 - Généralités -> Indépendance de la plate-forme

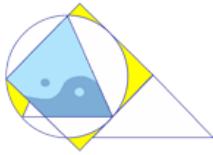


Java Program

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

`HelloWorldApp.java`





1 - Généralités -> API java



Yantra Technologies

		Java Language							http://docs.oracle.com/javase/8/docs/									
		java	javac	javadoc	jar	javap	jdeps	Scripting										
JDK	Tools & Tool APIs	Security	Monitoring	JConsole	VisualVM	JMC	JFR											
		JPDA	JVM TI	IDL	RMI	Java DB	Deployment											
		Internationalization		Web Services		Troubleshooting												
JRE	Deployment	Java Web Start				Applet / Java Plug-in												
		JavaFX																
	User Interface Toolkits	Swing		Java 2D		AWT	Accessibility											
		Drag and Drop		Input Methods		Image I/O	Print Service	Sound										
	Integration Libraries	IDL	JDBC	JNDI	RMI	RMI-IIOP		Scripting										
Java SE API	Other Base Libraries	Beans	Security		Serialization		Extension Mechanism											
		JMX	XML JAXP		Networking		Override Mechanism											
		JNI	Date and Time		Input/Output		Internationalization											
	lang and util Base Libraries	lang and util																
		Math		Collections		Ref Objects		Regular Expressions										
		Logging		Management		Instrumentation		Concurrency Utilities										
	Java Virtual Machine	Reflection	Versioning	Preferences API		JAR	Zip	Compact Profiles										
Java HotSpot Client and Server VM																		



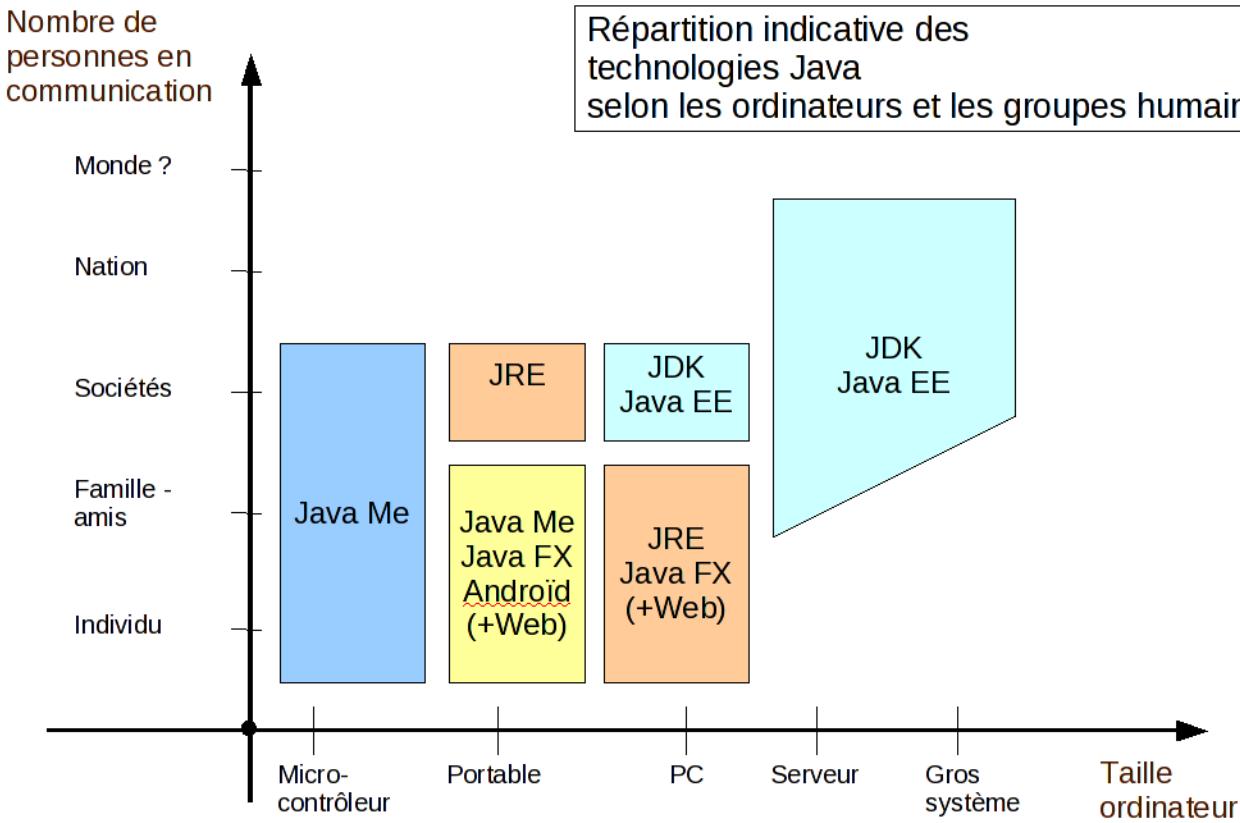


	Java 1.0	Java 1.1	Java 1.2	J2SE 1.3	J2SE 1.4	J2SE 5.0	Java SE 6	Java SE 7	Java SE 8
Nombre de packages	8	23	59	76	135	166	202	209	217
Nombre de classes	201	503	1520	1840	2990	3280	3780	4024	4240



- **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 -> De java 8 à java 14

Version	Nouveauté
Java 8 03/2014	<p>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.</p>
Java 9 09/2017	<p>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.</p>
Java 10 03/2018	<p>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</p>

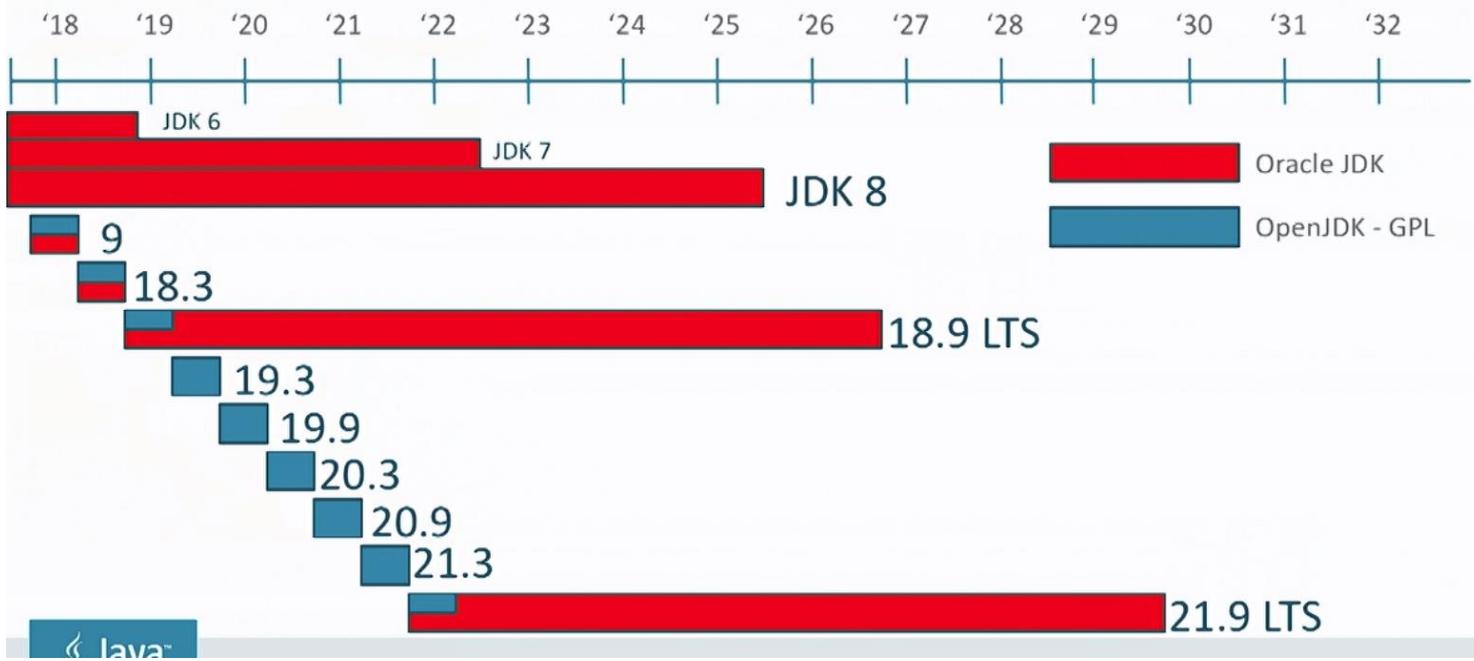
1 - Généralités -> De java 8 à java 14

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 métadonnées 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	Deux grosses nouveautés pour les développeurs sont apparues sur Java 13, les blocs de texte ainsi que l'apparition du mot clé « yield » sur les expressions switch
Java 14 03/2020	Amélioration des Text Blocks et Des NullPointerException

<https://zenika.developpez.com/tutoriels/java/comprendre-nouveautes-java12-a-java14/>

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

Oracle JDK & OpenJDK



<https://webdevdesigner.com/q/differences-between-oracle-jdk-and-openjdk-48296/>

<http://openjdk.java.net/install/>

<https://tekkollab.imdeo.com/java-devient-payant/>

https://www.youtube.com/watch?v=gV5T4AMFS_A

1 - Généralités -> Quelle version de java utilisez ?

Quelle version de Java utilisez-vous ?

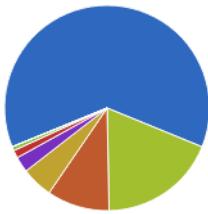
Qu'est-ce qui vous empêche de migrer vers une version plus récente ?

Le 13 mars 2019 à 14:44, par Michael Guilloux | 22 commentaires



1.2K PARTAGES

Quelle version de Java utilisez-vous ?



- Java 8 - 61,59%
- Java 11 - 18,29%
- Java 6 - 9,76%
- Java 7 - 4,88%
- Java 10 - 2,44%
- Java 9 - 1,22%
- Autre (à préciser dans les commentaires) - 0,61%
- Pas d'avis - 0,00%

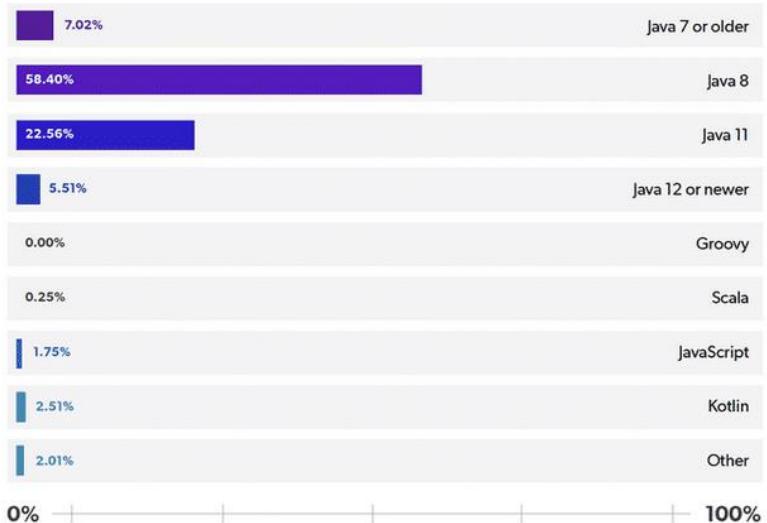
164 votants

Voter

Java : les développeurs priviléient Java 8 et Tomcat

Ariane Bely, 25 février 2020, 19:08

What Java programming language are you using in your main application?



<https://www.developpez.com/actu/250988/Quelle-version-de-Java-utilisez-vous-Qu'est-ce-qui-vous-empeche-de-migrer-vers-une-version-plus-recente/>

<https://www.silicon.fr/java-developpeurs-java8-tomcat-334720.html>

1 - Généralités -> Quelle version de java utilisez ?

Silicon CLOUD BLOCKCHAIN CYBERSÉCURITÉ MOBILITÉ / RESEAUX WORKPLACE L...

charge (mises à jour et maintenance) devait arriver à son terme fin 2025.

Pour quelles raisons Oracle a prolongé le support de cette version de Java ?

Java 8, sorti en mars 2014, toujours apprécié

Malgré ses six années de service et de nombreux successeurs (**Java 14** vient de sortir), Java 8 est la version la plus utilisée de la technologie, à la fois langage de programmation orienté objet et plateforme informatique (Java Development Kit, JDK).

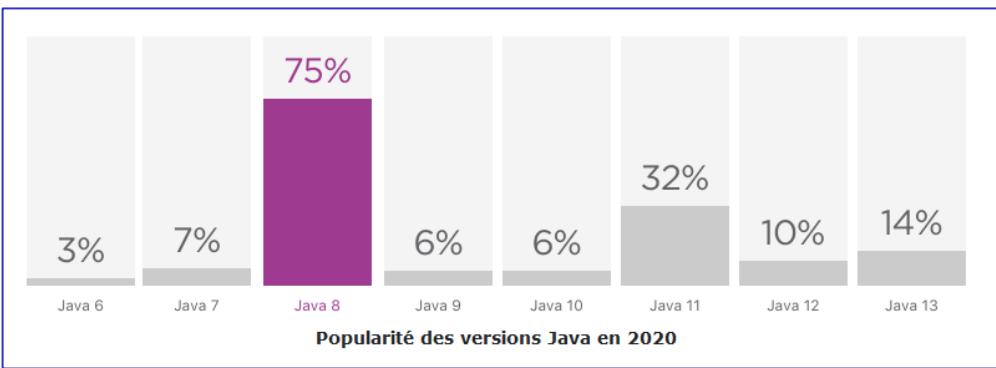
Selon un rapport de Perforce Software, 58% des **développeurs Java** interrogés l'automne dernier s'appuyaient sur Java 8, tandis que 22,5% s'étaient tournés vers Java 11. Une minorité privilégiant d'autres versions, antérieures ou plus récentes. Oracle, de son côté, estime que 30 à 40% utilisent Java 11 ou une version plus récente en production, actuellement. Java 8 restant encore la version la plus populaire, et de loin.

Java 8 est une version couplée à un support à long terme (LTS), comme la version 11 (jusqu'en 2026, à ce jour). Les éditions plus récentes du programme ne bénéficient pas de cette prise en charge étendue, mais d'un support initial de six mois.

<https://www.silicon.fr/oracle-rallonge-support-java-8-336554.html>

Oracle rallonge le support du très populaire Java 8

Ariane Beky, 19 mars 2020, 17:55 | Mis à jour le 3 janvier 2022, 16:24



<https://jetbrains.developpez.com/actu/309040/Les-chiffres-clés-de-la-communauté-Java-types-de-logiciels-développés-secteurs-d-activité-versions-EDI-framework-JetBrains-dresse-le-portrait-du-langage-en-2020/>

<https://www.silicon.fr/oracle-rallonge-support-java-8-336554.html>

2 - Les outils et techniques de base



- plate-forme de développement
- la gestion des dépendances

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



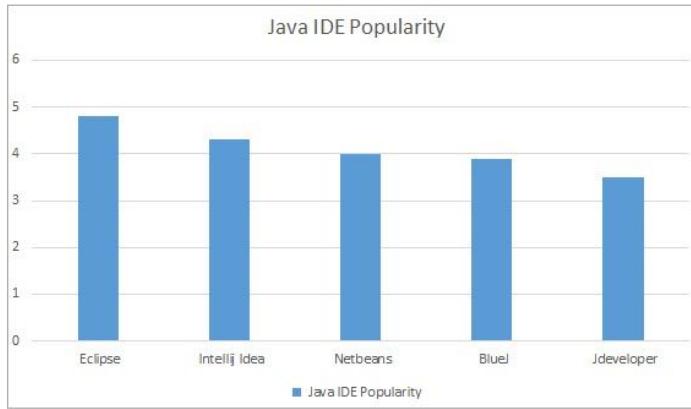
- **IntelliJ IDEA** est un IDE Java commercial développé par [Jetterais](https://www.jetbrains.com/idea/).
<https://www.jetbrains.com/idea/>
 - **Android Studio** est un environnement de développement pour développer des applications Android. Il est basé sur IntelliJ IDEA.
- **Netbeans** est un environnement 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/
- **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.

<https://waytolearnx.com/2020/04/top-10-des-meilleurs-ide-pour-les-developpeurs-java.html>

2 - Les outils et techniques de base -> plate-forme de développement

Graphique des 5 meilleurs logiciels Java IDE

Le graphique ci-dessous montre la popularité des 5 meilleurs IDE Java.



IDE Java	Note de l'utilisateur	Satisfaction des utilisateurs	Échelle de courbe d'apprentissage	Mise en évidence de la syntaxe	Performance
Éclipse	4,8 / 5	92%	Facile	Oui	Bien
Idée IntelliJ	4,3 / 5	89%	Moyen	Oui	Moyen
NetBeans	4.1 / 5	85%	Moyen	Non	Moyen
JDeveloper	4/5	80%	Facile	Oui	Moyen
Studio Android	4,3 / 5	90%	Tremper	Non	Bien
BLUEJ	4.1	82%	Moyen	Oui	Moyen

<https://fr.myservername.com/top-10-best-java-ides-online-java-compilers>

Démarrer avec intelliJ-idea

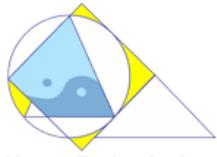
- <https://riptutorial.com/fr/intelliJ-idea>
- <https://www.jetbrains.com/help/idea/creating-and-running-your-first-java-application.html>

Documentation

- <https://www.jetbrains.com/fr-fr/idea/documentation/>



- **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



Yantra Technologies

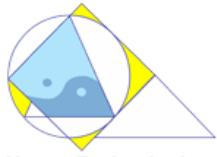


D.Palermo

Java Initiation
Version 4.0 – 09/2021

23

Copyright : Yantra Technologies 2004-2021



Yantra Technologies



D.Palermo

Java Initiation
Version 4.0 – 09/2021

24

Copyright : Yantra Technologies 2004-2021

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

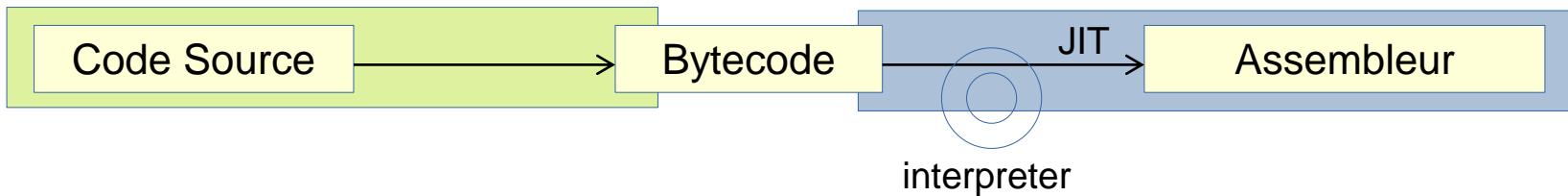
-> La compilation à l'exécution : Le bytecode portable

<http://www-igm.univ-mlv.fr/~forax/>

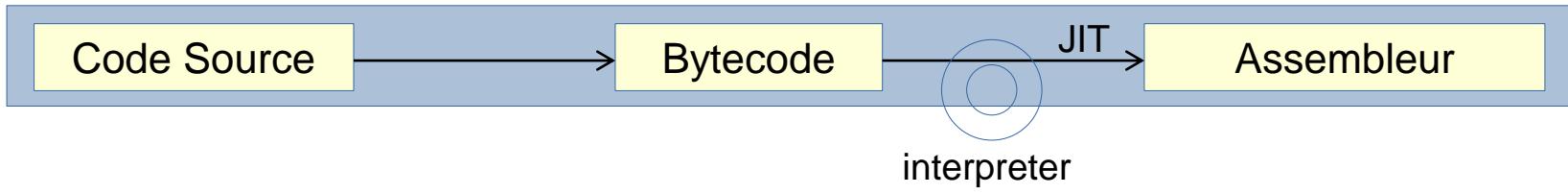
Modèle du C



Modèle de Java



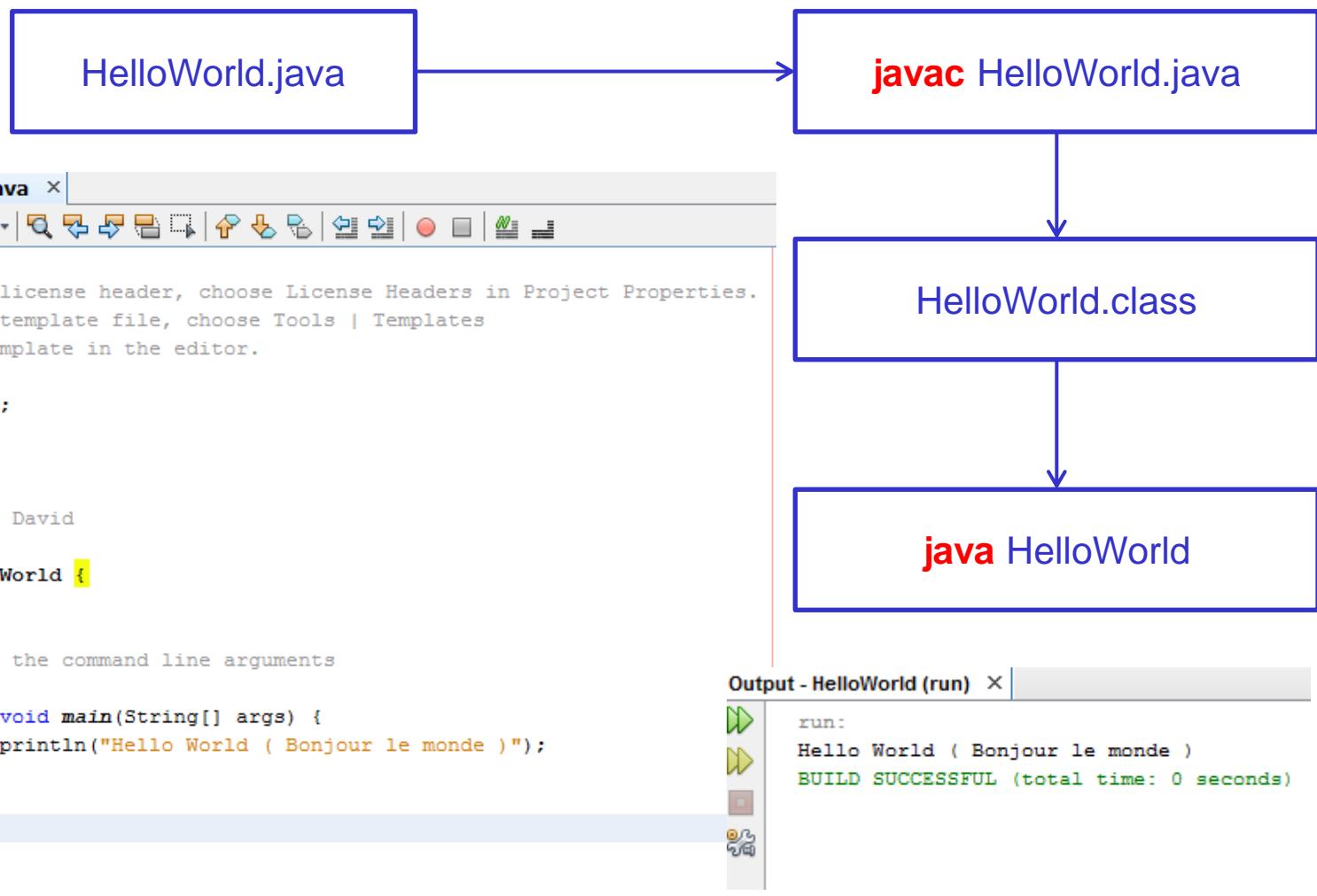
Modèle de JavaScript



A la compilation

A l'execution

3 - Les bases du langage Java -> La compilation à l'exécution





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 x 10^38	-1,40239846 x 10^-45
			1,40239846 x 10^-45	3,40282347 x 10^38
double	Nombre à virgule flottante double précision	8	-1,79769313486231570 x 10^308	-4,94065645841246544 x 10^-324
			4,94065645841246544 x 10^-324	1,79769313486231570 x 10^308

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



Type	Description
boolean	Les variables de type boolean 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



- 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



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 renournée 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 ⇒ il faut les initialiser ou les affecter.
- Syntaxe : <type> <identificateur>

```
int i = 0;  
boolean estVrai = false;  
double rayon ;  
rayon = 1.5;
```



- Listes des opérateurs
- 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				



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

3 - Les bases du langage Java

-> Les opérateurs : Listes des opérateurs



Symbol	Note	Priorité	Associativité
<code>^</code>	Ou exclusif logique bit à bit	6	Gauche à droite
<code> </code>	Ou inclusif logique bit à bit	5	Gauche à droite
<code>&&</code>	Et conditionnel	4	Gauche à droite
<code> </code>	Ou conditionnel	3	Gauche à droite
<code>? :</code>	Opérateur conditionnel	2	Droite à gauche
<code>= *= /= %=</code> <code>+= -= <=<=</code> <code>>>= >>>=</code> <code>&= ^= </code>	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 -)



3 - Les bases du langage Java -> Les opérateurs : binaires (1/3)



- &, 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

- ^, xor (ou exclusif)

Table de vérité XOR

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



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



- `~, compl` : complément à 1

Table de vérité NOT

A	NOT A
0	1
1	0

- `<<`, décalage à gauche
- `>>`, décalage à droite

Exemple : `p = n << 3;`

`p` égal `n` décalé de 3 bits sur la gauche



`expr1 ? expr2 : expr3`

si (expr1) alors expr2 sinon expr3

Exemple : fonction min

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

Equivalent à

```
x = ( y<z ) ? y : z ;
```



- $i=i+1$ 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 $a&=b;$



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



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



- 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



- 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*



- 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;
```

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



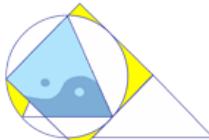
```
1 package helloworld;
2 import java.util.Scanner;
3
4 public class HelloWorld {
5
6
7     public static void main(String[] args) {
8         Scanner sc = new Scanner(System.in);
9         System.out.println("Hello World ( Bonjour le monde )");
10        System.out.println("Appuyer sur la touche q 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 }
```



Au cas où 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 :      ...;  
        ...;  
}
```



```
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
```

```
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.println("Hello World ( Bonjour le monde )");
    System.out.println("Appuyer sur la touche q pour quitter ");
    String quit = sc.nextLine();
    switch (quit.charAt(0)) {
        case 'q':
            return;
        case 'Q':
            System.out.println("Appuyer sur la touche q et pas Q pour quitter ");
            quit = sc.nextLine();
            break;
        case 'r':case 'R':case 'S':case 'T':
            System.out.println("Appuyer sur la touche q et pas " + quit + " pour quitter ");
            quit = sc.nextLine();
            break;
        default:
            System.out.println("Appuyer sur la touche q et pas " + quit + " pour quitter ");
            quit = sc.nextLine();
            break;
    }

    if ("q".equals(quit)) {
        return;
    } else {
        System.out.println("Dernier essai, Appuyer sur la touche q pour quitter ");
    }
    quit = sc.nextLine();
    System.out.println("Good bye");
}
```

Le switch “->”, *switch arrow* (depuis Java 14)

- Une “->” s'il y a une seul instruction
- Une flèche + un block s'il y a plusieurs instructions
- Il est possible de définir plusieurs valeurs dans une clause case en les séparant par une virgule.
- Le *switch* du C (le *switch colon*) existe toujours par compatibilité

```
1 public class Sample {
2
3     public static void main( String [] args ) {
4
5         int value = (int) (Math.random() * 11);
6
7         switch( value ) {
8             case 0, 1, 2, 3, 4 -> System.out.println( "Petit chiffre" );
9             case 5, 6, 7, 8, 9 -> System.out.println( "Grand chiffre" );
10            default -> System.out.println( "Ce n'est plus un chiffre, mais un nombre" );
11        }
12    }
13 }
14
15 }
```

Les types possibles pour le *switch*

- byte, short, char, int (Java 1.0) -> *switch* sur l'entier
- enum (Java 5) -> *switch* sur ordinal()
- string (Java 7) -> *switch* sur hashCode() + equals() si collision
- types (Java 17) -> une cascade de *if* / *instanceof* avec un *else* à la fin

un switch , case qui couvre toutes les valeurs possible du switch

```
public class SwitchEnum {
    enum Days {
        SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY
    }

    public static void main(String[] args) {

        Days day = Days.SATURDAY;

        switch (day) {
            case SUNDAY -> System.out.println("Sundays are wonderful");
            case MONDAY -> System.out.println("Mondays are boring");
            case TUESDAY -> System.out.println("Tuesdays are OK");
            case WEDNESDAY -> System.out.println("Wednesdays are tiring");
            case THURSDAY -> System.out.println("Thursdays are even more boring");
            case FRIDAY -> System.out.println("Fridays means work work and work");
            case SATURDAY -> System.out.println("Saturdays makes everybody happy");
        }
    }
}
```

3 - Les bases du langage Java -> Les structures de contrôle et boucle

: Au cas ... ou faire ...=> switch () { case }
switch sous forme d'expression

```
1 public class Sample {
2
3     public static void main( String [] args ) {
4
5         int value = (int) (Math.random() * 11);
6
7         String result = switch( value ) {
8             case 0, 1, 2, 3, 4 -> "Petit chiffre";
9             case 5, 6, 7, 8, 9 -> "Grand chiffre";
10            default -> "Ce n'est plus un chiffre, mais un nombre"
11        };
12
13        System.out.println( result );
14
15    }
16
17 }
```

Depuis java 15, Une classe ou une interface peut être déclarée **sealed**, ce qui signifie que seul un ensemble spécifique de classes ou d'interfaces peut directement l'étendre :

```
sealed interface Shape
permits Circle, Rectangle {

    record Circle(Point center, int radius) implements Shape {}

    record Rectangle(Point lowerLeft, Point upperRight) implements Shape -
}
```

```
float area = switch (shape) {
    case Circle c -> Math.PI * c.radius() * c.radius();
    case Rectangle r -> Math.abs((r.upperRight().y() - r.lowerLeft().y())
                                * (r.upperRight().x() - r.lowerLeft().x()));
    // no default needed!
}
```

<https://www.infoq.com/fr/articles/java-sealed-classes/>

le Pattern Matching est un moyen de se débarrasser du casting une fois qu'une condition instanceof est remplie.

```
static double getDoubleUsingIf(Object o) {
    double result;
    if (o instanceof Integer) {
        result = ((Integer) o).doubleValue();
    } else if (o instanceof Float) {
        result = ((Float) o).doubleValue();
    } else if (o instanceof String) {
        result = Double.parseDouble((String) o));
    } else {
        result = 0d;
    }
    return result;
}
```

```
static double getDoubleUsingSwitch(Object o) {
    return switch (o) {
        case Integer i -> i.doubleValue();
        case Float f -> f.doubleValue();
        case String s -> Double.parseDouble(s);
        default -> 0d;
    };
}
```

<https://www.baeldung.com/java-switch-pattern-matching>

Les cases peuvent étre utiliser en combinant des règles de typage et de condition boolean

```
static double getDoubleValueUsingIf(Object o) {  
    return switch (o) {  
        case String s -> {  
            if (s.length() > 0) {  
                yield Double.parseDouble(s);  
            } else {  
                yield 0d;  
            }  
        }  
        default -> 0d;  
    };  
}
```

```
static double getDoubleValueUsingGuardedPatterns(Object o) {  
    return switch (o) {  
        case String s && s.length() > 0 -> Double.parseDouble(s);  
        default -> 0d;  
    };  
}
```

<https://www.baeldung.com/java-switch-pattern-matching>

Les cases doivent être du plus précis au moins précis

Par ex, le *switch* ci-dessous ne compile pas

```
Object o = ...
switch(o) {
    case CharSequence seq -> ...
    case String s -> ...
    default -> ...
}
```

car String est plus précis que CharSequence

yield sera utile si vous utilisez le switch sous forme d'expression.

Car, il peut arriver que vous ayez besoin de plusieurs lignes de code pour calculer la valeur finale du switch.

```
1 public class Sample {
2
3     public static void main( String [] args ) {
4
5         int value = (int) (Math.random() * 11);
6
7         String result = switch( value ) {
8             case 0, 1, 2, 3, 4 -> {
9                 double sqrt = Math.sqrt( value );
10                yield "Petit chiffre dont la racine carré vaut " + sqrt;
11            }
12            case 5, 6, 7, 8, 9 -> {
13                double square = value * value;
14                yield "Grand chiffre dont le carré vaut " + square;
15            }
16            default -> "Ce n'est plus un chiffre, mais un nombre";
17        };
18
19         System.out.println( result );
20
21     }
22
23 }
```



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

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



```
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");
    }
}
```



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

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



```
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    }
```



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

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



```
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}
```



```
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.



```
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");
    }
}
```



```
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;`



`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
```



```
//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}}};
```



```
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.



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



- **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 chaîne de caractère

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



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)



- 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>

Java™ Platform Standard Ed. 8

OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES

SUMMARY NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

compact1, compact2, compact3
java.lang

Class String

java.lang.Object
java.lang.String

All Implemented Interfaces:

Serializable, CharSequence, Comparable<String>

public final class String
extends Object
implements Serializable, Comparable<String>, CharSequence

The String class represents character strings. All string literals in Java programs, such as "abc", are implemented as instances of this class.

Strings are constant; their values cannot be changed after they are created. String buffers support mutable strings. Because String objects are immutable they can be shared. For example:

```
String str = "abc";
```

is equivalent to:

```
char data[] = {'a', 'b', 'c'};  
String str = new String(data);
```

Here are some more examples of how strings can be used:

```
System.out.println("abc");  
String cde = "cde";  
System.out.println("abc" + cde);  
String c = "abc".substring(2,3);  
String d = cde.substring(1, 2);
```

The class String includes methods for examining individual characters of the sequence, for comparing strings, for searching substrings, and for creating a copy of a string with all characters translated to uppercase or to lowercase. Case mapping is based on the Unicode Standard version specified by the Character class.

The Java language provides special support for the string concatenation operator (+), and for conversion of other objects to strings. String concatenation is implemented through the StringBuilder or StringBuffer class and its append method. String conversions are implemented through the method `toString`, defined by Object and inherited by all classes in Java. For additional information on string concatenation and conversion, see Gosling, Joy, and Steele, *The Java Language Specification*.

Unless otherwise noted, passing a null argument to a constructor or method in this class will cause a `NullPointerException` to be thrown.

A String represents a string in the UTF-16 format in which supplementary characters are represented by surrogate pairs (see the section `Unicode Character Representations` in the `Character` class for more information). Index values refer to char code units, so a supplementary character uses two positions in a String.

The String class provides methods for dealing with Unicode code points (i.e., characters), in addition to those for dealing with Unicode code units (i.e., char values).

Since:
JDK1.0

See Also:
Object.`toString()`, `StringBuffer`, `StringBuilder`, `Charset`, `Serialized Form`

All Classes All Profiles Packages

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
java.awt.image.renderable

SynthHorizontalScrollBarUI
SynthScrollBarUI
SynthScrollPaneUI
SynthSeparatorUI
SynthSliderUI
SynthSpinnerUI
SynthStyle
SynthStyleFactory
SynthTablebedPaneUI
SynthTableHeaderUI
SynthTableLeafUI
SynthTextUI
SynthTextFieldUI
SynthTextPanelUI
SynthToggleButtonUI
SynthToolBarUI
SynthToolTipUI
SynthTreeUI
SynthViewportUI
SynthMessage
System
SYSTEM_EXCEPTION
SystemColor
SystemException
SystemFlavorMap
SystemLayout
TabbedPaneUI
TabbedPane
TabExpander
TableCellEditor
TableCellRenderer
TableColumn
TableColumnModel
TableColumnModelEvent
TableColumnModelListener
TableHeaderUI
TableModel
TableModelEvent
TableModelListener
TableModelSorter
TableStringConverter
TableUI
TableView
TabSet
TabStop

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][];`

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

`array=new int[3];`

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



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



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/>



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/>



- **Modification des Switch Expressions**
- **Optimisation de la mémoire et du Garbage collector**
- **Shenandoah: A Low-Pause-Time Garbage Collector**
- **Allocation of old generation of Java heap on alternate memory devices**
- **Paramétrage de la JVM : débogage et performances**

<https://blog.invivoo.com/a-la-decouverte-des-nouveautes-de-java-12/>



- **Les blocs de texte (Text Blocks) dans Java 13**
- **Nouvelle implémentation de l'API Socket**
- **Les améliorations apportées à Z Garbage Collector (ZCG)**
- **Archivage dynamique des classes**

<https://blog.invivoo.com/nouveautes-java-13/>



Switch expressions standardisées NullPointerException

<https://blog.invivoo.com/les-nouveautes-de-java-14/>

3 - Les bases du langage Java -> Les annotations

Les annotations de Java 5 apportent une standardisation des métadonnées dans un but généraliste.

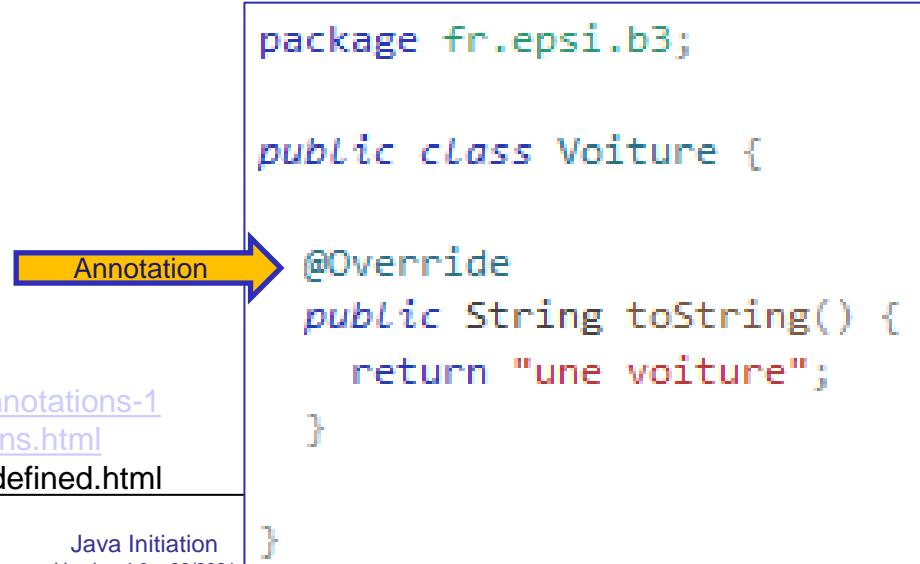
Ces métadonnées associés aux entités Java peuvent être exploitées à la compilation ou à l'exécution.

Les usages des annotations sont nombreux : génération de documentations, de code, de fichiers, ORM (Object Relational Mapping), ...

Une annotation n'est pas instanciée, elle est simplement accolée à l'élément qu'elle vient enrichir

Les annotations fournissent des informations sur des entités : elles n'ont pas d'effets directs sur les entités qu'elles concernent.

Une annotation s'utilise avec le caractère @ suivi du nom de l'annotation



```
package fr.epsi.b3;

public class Voiture {

    @Override
    public String toString() {
        return "une voiture";
    }
}
```

<https://www.jmdoudoux.fr/java/dej/chap-annotations.htm#annotations-1>

https://gayerie.dev/epsi-b3-java/langage_java/les_annotations.html

<https://docs.oracle.com/javase/tutorial/java/annotations/predefined.html>

3 - Les bases du langage Java -> Les annotations : annotations standard

@Deprecated : précise que l'entité concernée est obsolète et qu'il ne faudrait plus l'utiliser

```
@Deprecated  
class MaSousClasse {  
  
    /**  
     * Afficher un message de test  
     * @deprecated methode non compatible  
     */  
    @Deprecated  
    public void maMethode() {  
        System.out.println("test");  
    }  
}
```

@overide : utilisé par le compilateur pour vérifier la réécriture de méthodes héritées.

```
public class MaClasseMere {  
}  
  
class MaClasse extends MaClasseMere {  
  
    @Override  
    public void maMethode() {  
    }  
}
```

ERREUR →

<https://www.jmdoudoux.fr/java/dej/chap-annotations.htm#annotations-1>

https://gayerie.dev/epsi-b3-java/langage_java/les_annotations.html

3 - Les bases du langage Java -> Les annotations : annotations standard

@SuppressWarnings: Permet de forcer le compilateur à ne plus émettre d'avertissement à la compilation dans certains cas.

```
public class MaClasse {  
  
    @SuppressWarnings("unchecked")  
    public class MaClasse{  
        // ...  
    }  
}
```

```
public class MaClasse {  
  
    @SuppressWarnings("unchecked")  
    public void maMethode() {  
        // ...  
    }  
}
```

```
public class MaClasse {  
  
    @SuppressWarnings({ "rawtypes", "unchecked" })  
    private List liste = (List<String>) new ArrayList();  
}
```

<https://www.jmdoudoux.fr/java/dej/chap-annotations.htm#annotations-1>

https://gayerie.dev/epsi-b3-java/langage_java/les_annotations.html

3 - Les bases du langage Java -> Les annotations : annotations communes

Les annotations communes sont intégrées dans Java 6, leur but est de définir des annotations couramment utilisées et ainsi d'éviter leur redéfinition pour chaque outil qui en aurait besoin.

@Generated : Utiliser pour indiquer qu'un code a été produit par un générateur.

```
@Generated(  
    value = "Nom du générateur de code",           // Obligatoire  
    comments = "Explications sur l'intérêt de ce code", // Facultatif (String)  
    date = "2019/04/02"                            // Facultatif (String)  
)  
public class GeneratedClass() {  
}
```

@Resource & @Resources : permet d'associer à un attribut ou à un setter une JNDI. JNDI (Java Naming and Directory Interface) est une API permettant d'accéder à des ressources stockées dans une base de données hiérarchique.

```
public class ABean {  
  
    @Resource(name="connectionName")  
    private java.sql.Connection connection;  
  
    // Suite de la classe.  
}
```

```
@Resources({  
    @Resource(name = "maQueue" type = javax.jms.Queue),  
    @Resource(name = "monTopic" type = javax.jms.Topic),  
})
```

3 - Les bases du langage Java -> Les annotations : annotations communes

**@PostConstruct et
@PreDestroy**
Permet
respectivement de
désigner des
méthodes qui seront
exécutées après
l'instanciation d'un
objet et avant la
destruction d'une
instance.

```
package fr.koor.annotations;

import javax.annotation.PostConstruct;
import javax.annotation.PreDestroy;

public class ABean {

    private int attribute1 = 0;
    private int attribute2 = 0;

    public ABean() {
    }

    public int getAttribute1() {
        return attribute1;
    }

    public int getAttribute2() {
        return attribute2;
    }

    public void setAttribute1( int attribute1 ) {
        this.attribute1 = attribute1;
    }

    public void setAttribute2( int attribute2 ) {
        this.attribute2 = attribute2;
    }

    @PostConstruct
    public void postInitialisation() {
        System.out.println( "Initialisation finalized" );
    }

    @PreDestroy
    public void preFinalize() {
        System.out.println( "Now this instance is eligible for garbage collecting" );
    }
}
```

<https://www.jmdoudoux.fr/java/dej/chap-annotations.html#annotations>

https://gaverie.dev/epsi-b3-java/langage_java/les_annotations.html

https://koor.fr/Java/Tutorial/java_annotations_introduction.wp

3 - Les bases du langage Java -> Les annotations : annotations personnalisées

Java propose la possibilité de définir ses propres annotations,
une annotation est une interface lors de sa déclaration et est une instance d'une classe qui
implémente cette interface lors de son utilisation.

```
public @interface MonAnnotation {  
    public enum Niveau {DEBUTANT, CONFIRME, EXPERT} ;  
    String arg1() default "";  
    String[] arg2();  
    String arg3();  
    Niveau niveau() default Niveau.DEBUTANT;  
}
```

```
@MonAnnotation  
public class MaClasse {  
}
```

<https://www.jmdoudoux.fr/java/dej/chap-annotations.htm#annotations-1>

https://gayerie.dev/epsi-b3-java/langage_java/les_annotations.html

<https://devstory.net/10197/java-annotation#a18827>

3 - Les bases du langage Java -> Les annotations : annotations personnalisées

https://koor.fr/Java/Tutorial/java_annotations_definition.wp

Coder un nouveau type d'annotations



Introduction à l'utilisation d'annotations



Coder un mini framework de test avec les annotations

Accès rapide :

- Définition d'une nouvelle annotation
- Syntaxe générale
- Les différents niveaux de rétention
- Ajout d'une cible à une annotation
- Ajout d'attributs à une annotation
- Manipulation de notre annotation
- Travaux pratiques
- Le sujet
- La correction

Définition d'une nouvelle annotation

Syntaxe générale

Il est possible de coder ses propres annotations. On procède à ce type de définitions quand on cherche à développer son propre framework de test, de persistance ou autre.

Une annotation se définit via la construction `@interface`. On doit donner un nom à l'annotation : les conventions de nommage à respecter sont les mêmes que pour les classes ou les interfaces. Voici un premier exemple de définition d'annotation : cette annotation pourrait être utilisée par un framework de test similaire à JUnit (ce que nous étudierons dans le chapitre suivant).

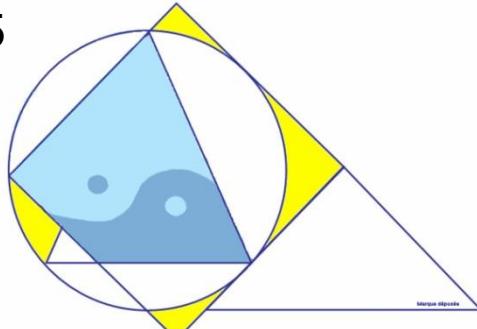
```
1 package fr.koor.sample;
2
3
4 public @interface TestMethod {
5
6 }
```

Notre première annotation

Les différents niveaux de rétention

Il existe différents niveaux de « rétention » : la rétention d'une annotation correspond, dans une certaine mesure, à sa durée de vie. On spécifie la rétention d'une annotation en lui ajoutant l'annotation `@Retention` (oui, une annotation peut être appliquée à une autre annotation). Voici un exemple de

Version 4.0 – 09/2021



Yantra Technologies

www.yantra-technologies.com

Programmation Orientée Objet

Les Concepts



Facile



Normal



Difficile



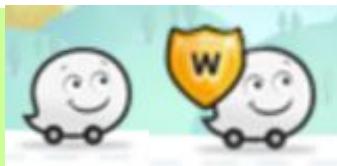
Professionnel



Expert

carole.grondein@yantra-technologies.com
david.palermo@yantra-technologies.com

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



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



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.



Une classe est composée de :

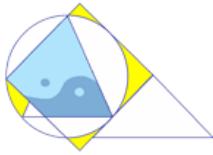
- déclarations de variables ou attributs,
- définitions de méthodes,
- les membres sont :
 - » des membres de classe (static),
 - » des membres d'objet (ou d'instance).

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 {

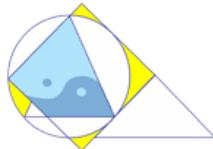
    public static void main(String[] args) {
        System.out.println("Exemple Cours");
        Voiture voiture = new Voiture("AX368BZ");
        voiture.afficher();
        System.out.println(voiture);
        voiture.setVitesse(100);
        System.out.println(voiture);
        try {
            voiture.finalize();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

class exemplecours

Cloneable	
Voiture	
-	a_vitesse: double
-	a_immatriculation: String
+	<u>GetCreate(): Voiture</u>
+	<u>GetCreate(v: String): Voiture</u>
#	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
<pre> 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) </pre>		





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

Cloneable	
Voiture	
-	a_vitesse: double
-	a_immatriculation: String
+	<u>GetCreate(): Voiture</u>
+	<u>GetCreate(v: String): Voiture</u>
#	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
```



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



```
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 ;

}
```



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.



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



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). deprecated a partir de java 9 remplacer par `java.lang.ref.Cleaner`*

<https://codippa.com/java-9-cleaner-example-for-garbage-collection/>

- L'appel au ramasse-miettes peut être forcé par l'appel :

`System.gc();`

<https://www.theserverside.com/video/5-ways-to-force-Java-garbage-collection>

<https://dev-mind.fr/blog/2021/java-memoire.html>



- 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 (surcharge)
- la dérivation se fait par `extends`
- toute classe dérive de la classe `Object`
- `super` : pour désigner la méthode de la classe de base



- 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



```
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 {

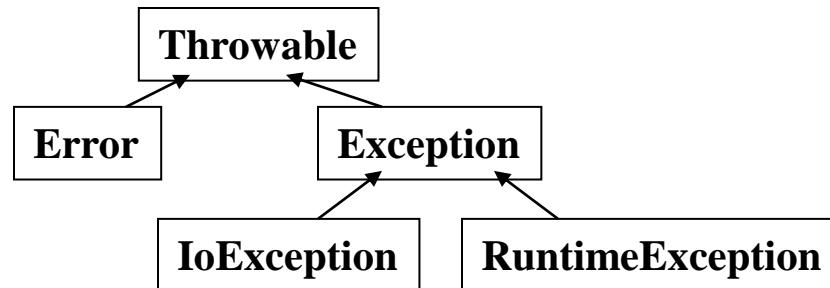
    public static void main(java.lang.String[] args) {
        int i = 100;
        int j = 0;
        System.out.println("résultat = " + (i / j));
    }

}
```

```
Exception in thread "main" java.lang.ArithmetricException: / 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)
```

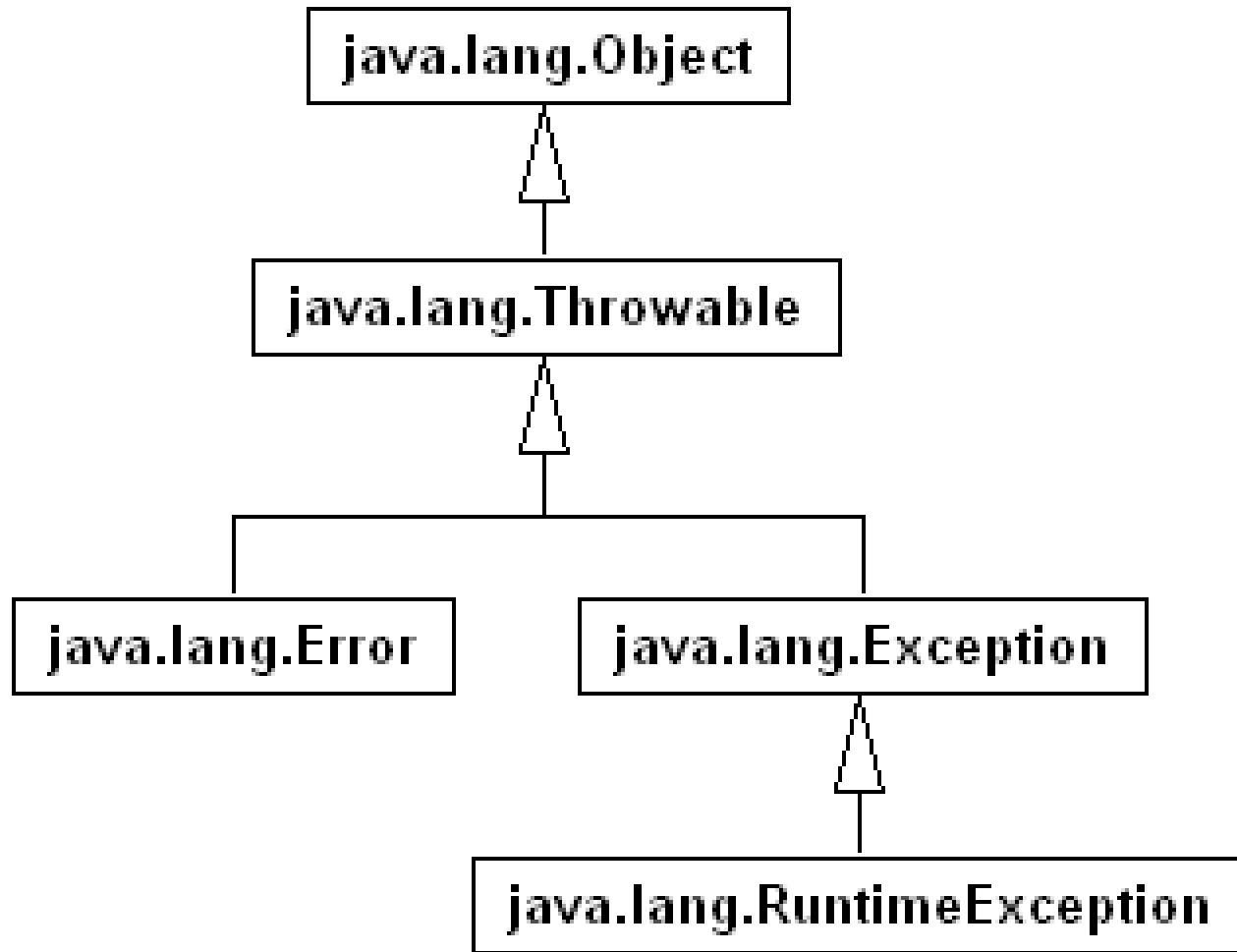


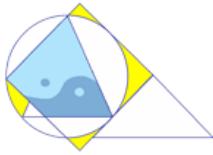
Traitement des Erreurs



- 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 : try {} catch {} finally {}



```
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 (ArithmeticsException 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.ArithmeticsException: / 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 ()



```

public class TestException {

package exemplecours;

public class MonException extends Exception {
    public MonException(String message) {
        super(message);
    }
}

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)
}
}

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 (Arithm

```

The code shows a Java program with two classes: `TestException` and `MonException`. The `TestException` class contains a `Test1()` method with十七行 (17 lines) of code and a `Test2()` method. The `Test2()` method attempts to divide 100 by 0, which triggers a `MonException` with the message "Erreur division par zero". The exception is caught in the `Test2()` method, and the result is printed. The `main()` method calls `Test2()`.

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 -> les classes scellées

Les classes scellées sont un nouveau moyen d'appliquer les règles d'héritage. Lorsque vous ajoutez le mot clé **sealed** à la définition d'une classe ou d'une interface, vous ajoutez également une liste de classes autorisées à l'étendre ou à l'implémenter

```
public abstract sealed class Color permits Red, Blue, Yellow
```

Vous pouvez également renoncer entièrement au mot-clé `permits` et conserver toutes les définitions des classes scellées dans le même fichier que la classe elle-même:

```
public abstract sealed class Color {...}
... class Red     extends Color {...}
... class Blue   extends Color {...}
... class Yellow extends Color {...}
```

<https://www.infoq.com/fr/articles/six-features-jdk12-to-jdk17/>

4 - L'aspect objet du langage Java -> Les classes record

<https://www.infoq.com/fr/articles/six-features-jdk12-to-jdk17/>

Les records (officiellement ajoutés dans Java 16) , sont des classes de données uniquement qui gèrent tout le code passe-partout associé aux **POJO**

- **Les méthodes equals() et hashCode() sont implémentées,**
- **toString() renverra une chaîne propre de toutes les valeurs de Record**
- **x() et y() seront des méthodes qui retournent, respectivement, les valeurs de x et y**
- **les records sont à la fois finaux et immuables**
- **Vous êtes autorisé à déclarer des méthodes dans un record, à la fois non statiques et statiques**
- **les Records peuvent avoir plusieurs constructeurs**

```
public record Coord(int x, int y) {  
}
```

```
public record Coord(int x, int y) {  
    public boolean isCenter() {  
        return x() == 0 && y() == 0;  
    }  
  
    public static boolean isCenter(Coord coord) {  
        return coord.x() == 0 && coord.y() == 0;  
    }  
}
```

POJO est un acronyme qui signifie **plain old Java object**

https://fr.wikipedia.org/wiki/Plain_old_Java_object

```
public record Coord(int x, int y) {  
    public Coord() {  
        this(0,0); // The default constructor is still implemented  
    }  
}
```

```
public record Coord(int x, int y) {  
    public Coord(int x, int y) {  
        this.x = x;  
        this.y = y;  
    } // Will replace the default constructor.  
}
```

https://koor.fr/Java/Tutorial/java_poo_record.wp

Les expressions lambda permettent d'écrire du code plus concis, donc plus rapide à écrire, à relire et à maintenir

Une expression lambda est une fonction anonyme

Une expression lambda permet d'encapsuler un traitement pour être passé à d'autres traitements.

```
for (int i = 0; i < list.size(); i++) {  
    System.out.println(list.get(i));  
}  
  
monBouton.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent event) {  
        System.out.println("clic");  
    }  
});  
  
monBouton.addActionListener(event -> System.out.println("clic"));
```



4 - L'aspect objet du langage Java

-> Les expressions lambda : La syntaxe

La syntaxe d'une expression lambda est composée de trois parties :

(liste d'arguments) -> {corps}

- Une liste de paramètres, d'aucun à plusieurs
- l'opérateur ->
- le corps de la fonction

```
Runnable monTraitement = () -> System.out.println("traitement");
```

Liste d'arguments peut être:

Zéro paramètre:

```
() -> System.out.println("Zéro paramètre");
```

Un seul paramètre:

```
(a) -> System.out.println("Un seul paramètre: " + a);
```

Plusieurs paramètres:

```
(a, b) -> System.out.println("Plusieurs paramètres: " + a + ", " + b);
```

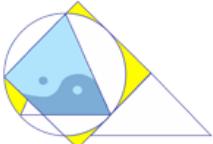
4 - L'aspect objet du langage Java

-> Les expressions lambda : La syntaxe

Caractéristiques importante des lambada :

- Type facultative : Pas besoin de déclarer le type d'un paramètre.
- Parenthèse facultative autour du paramètre : Pas besoin de déclarer un seul paramètre entre parenthèses.
- Accolades facultative : Pas besoin d'utiliser des accolades dans le corps d'une expression si le corps contient une seule instruction.
- Le mot clé « return » est facultatif : Le compilateur renvoie automatiquement la valeur si le corps a une seule expression. Des accolades sont nécessaires pour indiquer que l'expression renvoie une valeur

<https://waytolearnx.com/2020/04/les-expressions-lambda-en-java.html>



4 - L'aspect objet du langage Java

-> Les expressions lambda : La syntaxe

```
1. public class Main {  
2.  
3.     public static void main(String args[]) {  
4.  
5.         //avec la déclaration de type  
6.         Operation addition = (int x, int y) -> x + y;  
7.  
8.         //sans déclaration de type  
9.         Operation soustraction = (x, y) -> x - y;  
10.  
11.        //avec 'return' et les accolades  
12.        Operation multiplication = (int x, int y) -> { return x * y; };  
13.  
14.        //sans 'return' et sans les accolades  
15.        Operation division = (int x, int y) -> x / y;  
16.  
17.        System.out.println("8 + 2 = " + calculer(8, 2, addition));  
18.        System.out.println("8 - 2 = " + calculer(8, 2, soustraction));  
19.        System.out.println("8 x 2 = " + calculer(8, 2, multiplication));  
20.        System.out.println("8 / 2 = " + calculer(8, 2, division));  
21.    }  
22.  
23.    interface Operation {  
24.        int calc(int x, int y);  
25.    }  
26.  
27.    private static int calculer(int x, int y, Operation op) {  
28.        return op.calc(x, y);  
29.    }  
30. }
```

Dans le corps d'une expression lambda, il est possible d'utiliser :

- les variables passées en paramètre de l'expression
- les variables définies dans le corps de l'expression
- les variables final définies dans le contexte englobant

```
public class EventConsumerImpl {  
    private static String someStaticVar = "Some text";  
  
    public void attach(MyEventProducer eventProducer){  
        eventProducer.listen(e -> {  
            System.out.println(someStaticVar);  
        });  
    }  
}
```

```
public class EventConsumerImpl {  
  
    private String name = "MyConsumer";  
  
    public void attach(MyEventProducer eventProducer){  
        eventProducer.listen(e -> {  
            System.out.println(this.name);  
        });  
    }  
}
```

<https://www.jmdoudoux.fr/java/dej/chap-lambdas.htm>
<https://jenkov.com/tutorials/java/lambda-expressions.html>

4 types de référence de méthodes

Type	Syntaxe	Exemple
Référence à une méthode statique	nomClasse::nomMethodeStatique	String::valueOf
Référence à une méthode sur une instance	objet::nomMethode	personne::toString
Référence à une méthode d'un objet arbitraire d'un type donné	nomClasse::nomMethode	Object::toString
Référence à un constructeur	nomClasse::new	Personne::new

4 - L'aspect objet du langage Java

-> Les expressions lambda : Les références de méthodes

exemples de références de méthodes et leurs expressions lambda équivalentes :

Type	Référence de méthode	Expression lambda
Référence à une méthode statique	System.out::println	x -> System.out.println(x)
	Math::pow	(x, y) -> Math.pow(x, y)
Référence à une méthode sur une instance	monObject::maMethode	x -> monObject.maMethode(x)
Référence à une méthode d'un objet arbitraire d'un type donné	String::compareToIgnoreCase	(x, y) -> x.compareToIgnoreCase(y)
Référence à un constructeur	MaClasse::new	() -> new MaClasse(); (int valeur) -> new Integer(valeur) (int size) -> new String[size]

4 - L'aspect objet du langage Java

-> Les expressions lambda : Les interfaces fonctionnelles

Une interface fonctionnelle (functional interface) est une interface dans laquelle une seule méthode abstraite est définie.

Elle doit respecter certaines contraintes :

- elle ne doit avoir qu'une seule méthode déclarée abstraite
- les méthodes définies dans la classe Object ne sont pas prises en compte comme étant des méthodes abstraites
- toutes les méthodes doivent être public
- elle peut avoir des méthodes par défaut et static

Exemple :

- Comparator<T> qui définit la méthode int compare(T o1, T o2)
- Callable<V> qui définit la méthode V call() throws exception
- Runnable qui définit la méthode void run()
- ActionListener qui définit la méthode void actionPerformed(ActionEvent)
- ...

```
Consumer<String> afficher = (message) -> { System.out.println(message) };  
BiConsumer<Integer, Integer> additionner = (x, y) -> x + y;  
BiFunction<Integer, Integer, Long> additionner = (x, y) -> (long) x + y;
```

Les interfaces fonctionnelles peuvent être annotées avec `@FunctionalInterface` : cette annotation permet de préciser l'intention que l'interface soit fonctionnelle.

```
@FunctionalInterface
public interface MonInterface {
    public void traiter();
}
```

Les méthodes **publiques** de la classe Object peuvent être redéfinies dans une interface fonctionnelle. (car normalement une interfaces fonctionnelles ne doit avoir qu'une seule méthode déclarée abstraite)

```
@FunctionalInterface
public interface MonInterfaceFonctionnelle {
    String executer();
    boolean equals(Object obj);
}
```

```
@FunctionalInterface  
public interface Greeting {  
  
    String greeting(String name);  
  
    default String hello(String name) {  
        return "Hello " + name;  
    }  
}
```

Une interface fonctionnelle définit une méthode qui pourra être utilisée pour passer en paramètre :

- une référence sur une méthode d'une instance
- une référence sur une méthode statique
- une référence sur un constructeur
- une expression lambda
- une classe anonyme interne

Le package `java.util` utilise énormément les expression lambda



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



package nom-du-package;

avec la création du répertoire associé



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

4 - L'aspect objet du langage Java -> Les classes internes



```
package exemplecours;

public class ClassePrincipal {

    class ClasseInterne {

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

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

    }

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

Classe ClassePrincipal
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);
    }
}

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);
}
  
```

- 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.

```

run:
chiffre
100
BUILD SUCCESSFUL (total time: 0 seconds)
  
```

Avant avant Java 5 :

```
ArrayList list = new ArrayList();
list.add("hello");
list.add(new URI(...));
String s = (String) list.get(0); // ERREUR à l'exécution
```

A partir de Java 5

```
ArrayList<String> list = new ArrayList<String>();
list.add("hello");
list.add(new URI(...)); // ERREUR à la compilation
String s = (String) list.get(0);
```

Les Patrons introduisent la possibilité de déclarer **une variable de type**

```
public class ArrayList<E> {  
    public void add(E element) { ... }  
    public E get(int index) { ... }  
}
```

A la compilation, si on utilise une instance de type `ArrayList<String>`, le compilateur substitue **E** par **String** donc on obtient les méthodes

- Void add(**String**)
- **String** get(int)

Un type paramétrisé peut avoir plusieurs variables de types

```
public class HashMap<K, V> {  
    public void put(K key, V value) { ... }  
    public V get(Object key) { ... }  
}
```

```
Map<Integer, String> customers = new HashMap<>();  
customers.put(1, "David");
```

Les patrons appliqués avec les méthodes

```
class Stack<T> {  
  
    void push(T t) { ... }  
    void T pop() { ... }  
}
```

```
class Utils {  
  
    public static void copy(List dst, List src) { ... } // avant java 5  
  
    public static <T> void copy(List<T> dst, List<T> src) {  
        for(T element: src) { dst.add(element); }  
    }  
  
    public <T> void transfer(Stack<T> s1, Stack<T> s2) { ... }  
}
```

```
class Element<E> {  
  
    E e;  
    List<E> list;  
    E Element1(E e) { return e; }  
    void Element2() { E e; }  
  
  
  
  
  
  
  
  
  
  
  
    }  
}
```

static E element; // Erreur
static E val1(E e) { return e; } // Erreur
static void val2() { E e; } // Erreur

```
public static <R extends Runnable> R foo(R r) { ...}
```

```
static <E extends Closeable & CharSequence> E bar(E e) { .. }
```

Remarque : Une des bornes peut être une classe mais elle doit être déclarée en premier

```
interface Orderable<T extends Orderable<T>> {
    public boolean lessThan(T element);
}
```

```
class MyInteger implements Orderable<MyInteger> {
    private final int value;
    public boolean lessThan(MyInteger element) { return value < element.value; }
}
```

4 - L'aspect objet du langage Java -> les patrons : Type <?>

<http://blog.paumard.org/cours/java-api/chap02-generiques-type-anonyme.html>

Problème :

Une règle est qu'une classe générique ne peut étendre aucune version d'elle-même.

Ainsi, `List<T>` n'étend jamais `List<U>`, quelle que soit la relation qui puisse exister entre `T` et `U`

```
// création d'une liste d'Integer
List<Integer> integerList = Arrays.asList(1, 2, 3) ;

// ce cast n'est pas autorisé et ne compile pas, mais supposons qu'il le soit
List<Number> numberList = (List<Number>)listOfInteger ;

// il deviendrait possible d'ajouter à integerList des éléments qui ne
// seraient pas des Integer !
numberList.add(1.0F) ; // ajout d'un Float, illégal
```

4 - L'aspect objet du langage Java -> les patrons : Type <?>

<http://blog.paumard.org/cours/java-api/chap02-generiques-type-anonyme.html>

Résolution : L'introduction du Type <?>

```
List<Integer> listOfInteger = Arrays.asList(1, 2, 3) ;  
  
// ce cast est correct : List<? extends Number> est bien étendu  
// par List<Integer>  
List<? extends Number> listOfNumberExtensions = listOfInteger ;  
  
// en revanche cet appel ne compile pas, car la méthode add de  
// List<? extends Number> prend le type ? extends Number en paramètre,  
// qui n'est pas connu  
listOfNumberExtensions.add((Integer)2) ;  
  
// cet appel est correct  
int i = listOfNumberExtensions.get(0) ;
```

4 - L'aspect objet du langage Java-> les patrons : Type <?>

<http://blog.paumard.org/cours/java-api/chap02-generiques-type-anonyme.html>

Type <? super T> : désigne tous les types dont le type T est un type dérivé.

```
List<Integer> listOfInteger = Arrays.asList(1, 2, 3);

// cette déclaration n'a pas besoin de cast, car List<? super Integer> étend
// List<Integer>
List<? super Integer> listOfNumberExtensions = listOfInteger;

// cet ajout est valide, on peut passer un Integer au add()
listOfNumberExtensions.add(2);

// ce cast compile bien, mais n'est pas sûr, car le type de retour peut être Object
int i = (Integer)listOfNumberExtensions.get(0);
```

L'inférence de type est une technique utilisée par les langages à typage statique, où les types de variables peuvent être déduits du contexte par le compilateur.

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

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

`List<String> list = Collection.<String>emptyList(); =>`

`List<String> list = Collection.emptyList();`

`list.forEach((String s) -> System.out.println(s)) =>`

`list.forEach(s -> System.out.println(s))`

<https://www.infoq.com/fr/articles/java-local-variable-type-inference/>

4 - L'aspect objet du langage Java -> les patrons : Erasure (effacement)

Les types génériques en Java ont été **ajoutés** au langage pour permettre une vérification de type au moment de la compilation,

Mais pour s'assurer que le bytecode généré soit toujours **compatible** avec celui généré par les versions précédentes de Java, le compilateur applique un processus appelé **effacement de type (type erasure)** sur les génériques lors de la compilation.

```
public <T> List<T> methodeGenerique(List<T> liste) { .... }
```

le bytecode généré

```
public List methodeGenerique(List liste) { ... }
```

```
public <T extends MaClasse> void methodeGenerique(T donnees) { ... }
```

le bytecode généré

```
public void methodeGenerique(MaClasse donnees) { ... }
```

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

4 - L'aspect objet du langage Java -> les patrons : Erasure (effacement) -> les effets de bord

L'effacement de type empêche de conserver le type générique dans le byte-code
donc

il n'est pas possible de distinguer le type de deux instances d'une même classe

```
import java.util.List;
import java.util.ArrayList;

public class MaClasse {

    public static void main(String[] args) {
        List<Integer> entiers = new ArrayList<Integer>();
        List<Double> doubles = new ArrayList<Double>();
        System.out.println(entiers.getClass() == doubles.getClass());
    }
}
```

Résultat => true

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

4 - L'aspect objet du langage Java -> les patrons : Erasure (effacement) -> les effets de bord

L'effacement de type remplace le type générique par le type Object
donc

dans le bytecode, les types génériques sont remplacés par le type Object

```
public class MaClasse<T> {  
  
    public static void main(String[] args) {  
        MaClasse<String> maClasse = new MaClasse<>();  
        maClasse.traiter("test");  
    }  
  
    String obtenirDonnees(Object s) {  
        return "object";  
    }  
  
    String obtenirDonnees(String s) {  
        return "string";  
    }  
  
    public void traiter(T t) {  
        System.out.println(obtenirDonnees(t));  
    }  
}
```

Résultat => object

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

4 - L'aspect objet du langage Java -> les patrons : Erasure (effacement) -> les effets de bord

L'effacement de type remplace le type générique ce qui ne permet pas au runtime de connaître le type paramétré.

donc

Il n'est pas possible d'utiliser un type générique comme type dans un opérateur instanceof

```
public class MaClasse<T> {  
  
    public boolean tester(Object o) {  
        return o instanceof T;  
    }  
}
```

Résultat => Erreur compilation

```
C:\java>javac MaClasse.java  
MaClasse.java:4: error: illegal generic type for instanceof  
        return o instanceof T;  
                           ^  
1 error
```

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

4 - L'aspect objet du langage Java -> les patrons : Erasure (effacement) -> les effets de bord

L'effacement de type remplace le type générique par le type Object donc

Il n'est pas possible qu'une méthode possède deux surcharges, l'un avec un paramètre de type et l'autre qui attend un paramètre de type Object et non plus si une surcharge est déjà héritée.

Résultat => Erreur compilation

```
public class MaClasseGenerique<T> {
    public void traiter(Object o) {
    }
    public void traiter(T t) {
    }
}
```

```
C:\java>javac MaClasseGenerique.java
MaClasseGenerique.java:6: error: name clash: traiter(T) and traiter(Object) have the same
erasure
    public void traiter(T t) {
                           ^
      where T is a type-variable:
        T extends Object declared in class MaClasse
1 error
```

```
public class MaClasse<T> {
    public boolean equals(T t) {
    }
}
```

```
C:\java>javac MaClasse.java
MaClasse.java:3:
error: name clash: equals(T) in MaClasse and equals(Object) in Object have the same erasure,
yet neither overrides the other
    public boolean equals(T t) {
                           ^
      where T is a type-variable:
        T extends Object declared in class MaClasse
1 error
```

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

4 - L'aspect objet du langage Java -> les patrons : Erasure (effacement) -> les effets de bord

L'effacement de type remplace le type générique ce qui ne permet pas au runtime de connaître le type paramétré.
donc

Il n'est pas possible de créer une instance d'un paramètre de type avec l'opérateur new

```
public class MaClasseGenerique<T> {  
  
    private T instance;  
  
    public MaClasseGenerique() {  
        this.instance = new T();  
    }  
}
```

Résultat => Erreur compilation

```
C:\java>javac MaClasseGenerique.java  
MaClasseGenerique.java:6: error: unexpected type  
    this.instance = new T();  
                           ^  
       required: class  
       found:   type parameter T  
       where T is a type-variable:  
           T extends Object declared in class MaClasse  
1 error
```

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

4 - L'aspect objet du langage Java -> les patrons : Erasure (effacement) -> les effets de bord

Pour créer une instance, il faut utiliser l'API Reflection qui impose d'avoir une instance de type Class du type générique utilisé passé en paramètre.

```
import java.util.Date;

public class MaClasseGenerique<T> {

    private T instance;

    public MaClasseGenerique(Class<T> classe) throws InstantiationException,
        IllegalAccessException {
        this.instance = classe.newInstance();
        System.out.println(instance);
    }

    public static void main(String[] args) throws Exception {
        MaClasseGenerique<Date> maClasse = new MaClasseGenerique<Date>(Date.class);
    }
}
```

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

4 - L'aspect objet du langage Java -> les patrons : Erasure (effacement) -> les effets de bord

L'effacement de type remplace le type générique ce qui ne permet pas au runtime de connaître le type paramétré.
donc

Il n'est pas possible de créer une instance d'un tableau d'un paramètre de type avec l'opérateur new

```
import java.lang.reflect.Array;

public class MaClasse<T> {

    private T[] tableau;

    public MaClasse() {
        this.tableau = new T[10];
    }
}
```

Résultat => Erreur compilation

```
C:\java>javac MaClasse.java
MaClasse.java:9: error: generic array creation
    this.tableau = new T[10];
                           ^
1 error
```

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

4 - L'aspect objet du langage Java -> les patrons : Erasure (effacement) -> les effets de bord

Pour créer une instance, il faut utiliser l'API Reflection qui impose d'avoir une instance de type Class du type générique utilisé passé en paramètre.

```
import java.lang.reflect.Array;
import java.util.Date;

public class MaClasseGenerique<T> {

    private T[] tableau;

    public MaClasseGenerique (Class<T> classe) throws InstantiationException,
        IllegalAccessException {
        this.tableau = (T[]) Array.newInstance(classe, 10);
        System.out.println(tableau);
    }

    public static void main(String[] args) throws Exception {
        MaClasseGenerique<Date> maClasse = new MaClasseGenerique<Date>(Date.class);
    }
}
```

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

4 - L'aspect objet du langage Java -> Varargs

Varargs permet de passer l'argument final à une méthode sous forme de array ou de séquence d'arguments, en plaçant trois points après le type du paramètre final.

Exemple :

```
public static void foo(String... string_array) { ... }
```

remplace

```
public static void bar(String[] string_array) { ... }
```

Remarque :

- Les méthodes Vararg peuvent également être surchargées, mais la surcharge peut entraîner une ambiguïté.
- Avant Java 5, les arguments de longueur variable pouvaient être traités de deux manières : l'une utilisait la surcharge, l'autre utilisait l'argument tableau
- Il ne peut y avoir qu'un seul argument de variable dans une méthode.
- L'argument variable (varargs) doit être le dernier argument.



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
```



- 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;
    }

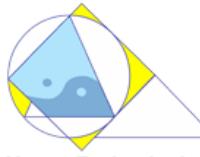
    public static void main(String args[]) {

        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>
<http://blog.paumard.org/cours/java/chap04-structure-classe-enumeration.html>



```
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;

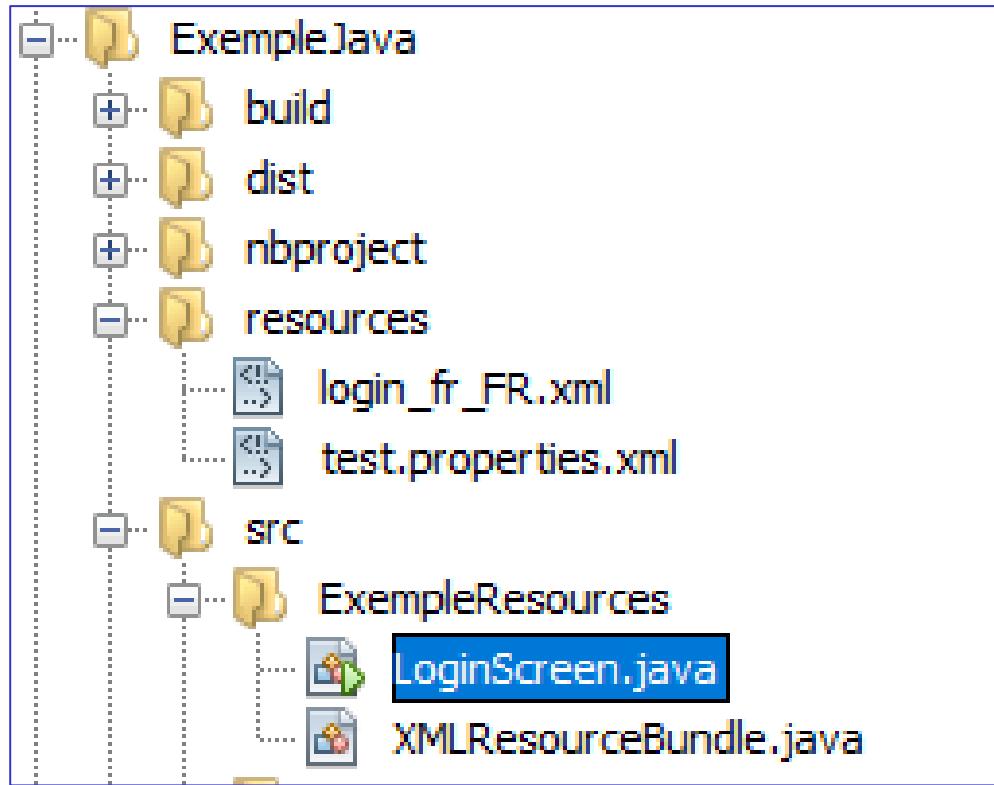
    Animal(String nom, boolean domestique)
    { this(nom, domestique, null); }

    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





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

<https://docs.oracle.com/javase/tutorial/i18n/intro/quick.html>

<https://jmdoudoux.developpez.com/cours/developpons/java/chap-i18n.php>



```
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();  
    }  
}
```



```
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("-----");  
    }  
}
```



```
-- listing properties --
login=Identification
password=Mot de passe
francais=Français
username=Identifiant

-----
login
password
francais
username

-----
Mot de passe
```



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>

<https://javaetmoi.com/2021/01/bonnes-pratiques-de-logging/>

<https://sematext.com/blog/java-logging/>

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



```
<?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



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`

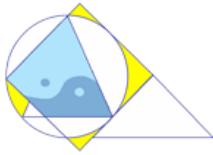


L'extension ou *package* `java.lang` est l'extension de base de Java (il n'est pas nécessaire de faire un import de cette extension, il est implicite).

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.lang.Class

<https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/Class.html>

La classe Class<T>

- permet d'accéder aux informations sur les classes.
- il existe un objet Class pour toute les classe utilisées.
- Les méthodes :
 - getName() retourne le nom (String) de la classe
 - getSuperclass() renvoie des informations (de type Class) sur la super-classe de la classe
 - toString() = getName() + "class" ou "interface" selon le cas
 - newInstance() produit une nouvelle instance du type de l'objet.
 - Autres :
<https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/Class.html>
- cette class est visible comme un objet
- On peut demander à chaque objet quel est l'objet correspondant à sa classe Class<?> Object.getClass()

5 - Les packages de base -> java.lang.Class

Object.getClass() renvoie à l'exécution la classe d'un objet correspond à la sa creation

```
Object objet = new MaClasse();
```

object.getClass() ou MaClasse.class rends la classe de l'objet

```
Object object = "hello";
```

object.getClass() corresponds a String.class;

La methode Class.newInstance() permet de créer une nouvelle instance de la classe

```
Date d = new Date();
```

```
Class cls = d.getClass();
```

```
Object obj = cls.newInstance();// obj est du type Date
```

5 - Les packages de base -> java.lang.Class : Introspection

https://koor.fr/Java/Tutorial/java_reflexion_introduction.wp

Java connaît la classe des objets à l'exécution.

java.lang.Class permet dynamiquement

- Trouver les méthodes
- Appeler les méthodes avec les arguments

Method[] Class.getMethods() : Renvoyer un tableau des méthodes publiques de la classe incluant celles héritées

Method Class.getMethod(String methodName, Class[] parameterType) : Permet d'utilisée pour obtenir la méthode spécifiée de cette classe avec le type de paramètre spécifié elle renvoie la méthode spécifiée de cette classe sous la forme d'objets Method.

La classe Method a des méthodes tel que;

- getModifier()
- getName(),
- getParameterTypes(),
- getReturnType()
- Object invoke(Object instance, Object... args)
- Autres : <https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/reflect/Method.html>

5 - Les packages de base -> java.lang.Class : Introspection Java

https://koor.fr/Java/Tutorial/java_reflexion_introduction.wp

```
1 import java.lang.reflect.Method;
2
3
4 public class TestReflection {
5
6     public void publicMethod() {}
7     private void privateMethod() {}
8
9     public static void publicStaticMethod() {}
10    private static void privateStaticMethod() {}
11
12
13    public static void main( String[] args ) throws Exception {
14
15        // Récupération des métadonnées à partir de la classe.
16        Class<?> metadata = TestReflection.class;
17
18        // On récupère les méthodes déclarées dans le type courant
19        Method [] methods = metadata.getDeclaredMethods();
20
21        // On affiche des informations sur les méthodes de la classe
22        for( Method method : methods ) {
23            System.out.println( method.getName() );
24        }
25
26    }
27
28 }
```

```
main
publicStaticMethod
publicMethod
privateMethod
privateStaticMethod
```

La classe Class contient les méthodes

- **getDeclaredConstructor ()**
- **getDeclaredConstructors ()**

Elle permettent de récupérer les constructeurs d'une classe, et les classes Constructor et Array déclarent elles aussi les méthodes **newInstance ()**.

La méthode **getConstructor ()** requiert en paramètre un tableau des classes que doit recevoir le constructeur recherché.

La méthode **newInstance ()** requiert en paramètre un tableau d'objets qui correspondent à chacun des paramètres du constructeur invoqué.

[https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/Class.html#getConstructor\(java.lang.Class...\)](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/Class.html#getConstructor(java.lang.Class...))

<http://www.eteks.com/tips/tip1.html>

5 - Les packages de base -> java.lang.Class : Introspection Java

```
import java.util.*;  
  
public class Test {  
  
    public Test() {}  
  
    public static void main(String[] args)  
        throws ClassNotFoundException, NoSuchMethodException  
    {  
  
        // returns the Class object for this class  
        Class myClass = Class.forName("Test");  
  
        System.out.println("Class represented by myClass: "  
                           + myClass.toString());  
  
        Class[] parameterType = null;  
  
        // Get the constructor of myClass  
        // using getConstructor() method  
        System.out.println(  
            "Constructor of myClass: "  
            + myClass.getConstructor(parameterType));  
    }  
}
```

```
Class represented by myClass: class Test  
Constructor of myClass: public Test()
```

<https://www.geeksforgeeks.org/class-getconstructor-method-in-java-with-examples/>

5 - Les packages de base -> java.lang.Class : Introspection Java

La méthode `getFields()`/ `getField(string)` de la classe `java.lang.Class` sont utilisées pour obtenir un tableau des champs ou un champ public de la classe,

Syntaxe :

- `public Field getField(String fieldName) throws NoSuchFieldException, SecurityException`
- `public Field[] getFields()`

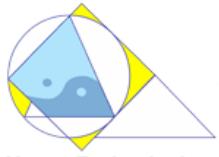
<https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/reflect/Field.html>

5 - Les packages de base -> java.lang.Class : Introspection Java

```
import java.util.*;  
  
public class Test {  
    public Object obj;  
  
    public static void main(String[] args)  
        throws ClassNotFoundException  
    {  
  
        // returns the Class object for this class  
        Class myClass = Class.forName("Test");  
  
        System.out.println("Class represented by myClass: "  
                           + myClass.toString());  
  
        // Get the fields of myClass  
        // using getFields() method  
        System.out.println("Fields of myClass: "  
                           + Arrays.toString(  
                               myClass.getFields()));  
    }  
}
```

```
Class represented by myClass: class Test  
Fields of myClass: [public java.lang.Object Test.obj]
```

<https://www.geeksforgeeks.org/class-getfields-method-in-java-with-examples/>





5 - Les packages de base -> java.io



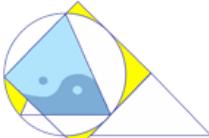
java.io (io pour in out) est un package qui va nous fournir des classes pour manipuler des **flux de données**. On les appelle aussi les classes d'entrées/sorties.

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)



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 & java.nio : Fichiers et répertoires –**
java.nio.file.Path & java.io.File

les classes Java **Path** et **Files** permettent de travailler avec des fichiers.

En Java, Path et File appartiennent à des packages différents mais exécutent le même mécanisme. On peut dire que la classe Java Path est la version avancée de la classe File .

- La classe **java.io.File** est utilisée pour effectuer des opérations sur les fichiers et les répertoires
- La classe **java.nio.file.Path** permet de créer des objets contenant des informations sur les fichiers et les répertoires,

Path est un chemin dans l'arborescence soit

- relatif -> Path chemin = fs.getPath("./Java\\cours.txt");
- absolue
 - Path chemin = fs.getPath("c:\\Program Files\\Java\\cours.txt");
 - Path chemin = Paths.get("/Volumes/Sauvegarde/intro.txt");

Un Path est :

- une liste chaînée des éléments du plus éloigné vers la racine : user <- david <- java <- cours.txt
- un Iterable<Path>

Constante **java.io.separatorChar** : / (Unix), \ (Windows) et : ou / (Mac)

<https://developpement-informatique.com/article/263/comprendre-les-fichiers-informatiques>

<https://www.javatpoint.com/java-path-vs-file>



5 - Les packages de base

-> java.nio : Fichiers et répertoires – file.Path

```
import java.nio.file.*;  
  
public class Test {  
    public static void main(String args[]) {  
        Path chemin = Paths.get("/Volumes/Sauvegarde/intro.txt");  
        int count = chemin.getNameCount();  
        System.out.println("le chemin est " + chemin.toString());  
        System.out.println("le nom du fichier est " + chemin.getFileName());  
        System.out.println("Il y a " + count + " éléments dans cet chemin");  
        for (int x = 0; x < count; x++)  
            System.out.println("Elément " + x + " : " + chemin.getName(x));  
    }  
}
```

```
import java.nio.file.*;  
  
public class Test {  
    public static void main(String args[]) {  
        Path chemin = Paths.get("intro.txt");  
        Path absolu = chemin.toAbsolutePath();  
        System.out.println("chemin absolu : " + absolu.toString());  
    }  
}
```

5 - Les packages de base

-> java.nio : Fichiers et répertoires – file.Path

```
import java.io.IOException;
import java.nio.file.*;
import static java.nio.file.AccessMode.*;

public class Test {
    public static void main(String args[]) {
        Path chemin = Paths.get("intro.txt");
        try {
            chemin.getFileSystem().provider().checkAccess(chemin, READ, EXECUTE)
            System.out.println("Le fichier peut être lu et exécuté");
        } catch (IOException e) {
            System.out.println("Le fichier ne peut pas être utilisé");
        }
    }
}
```

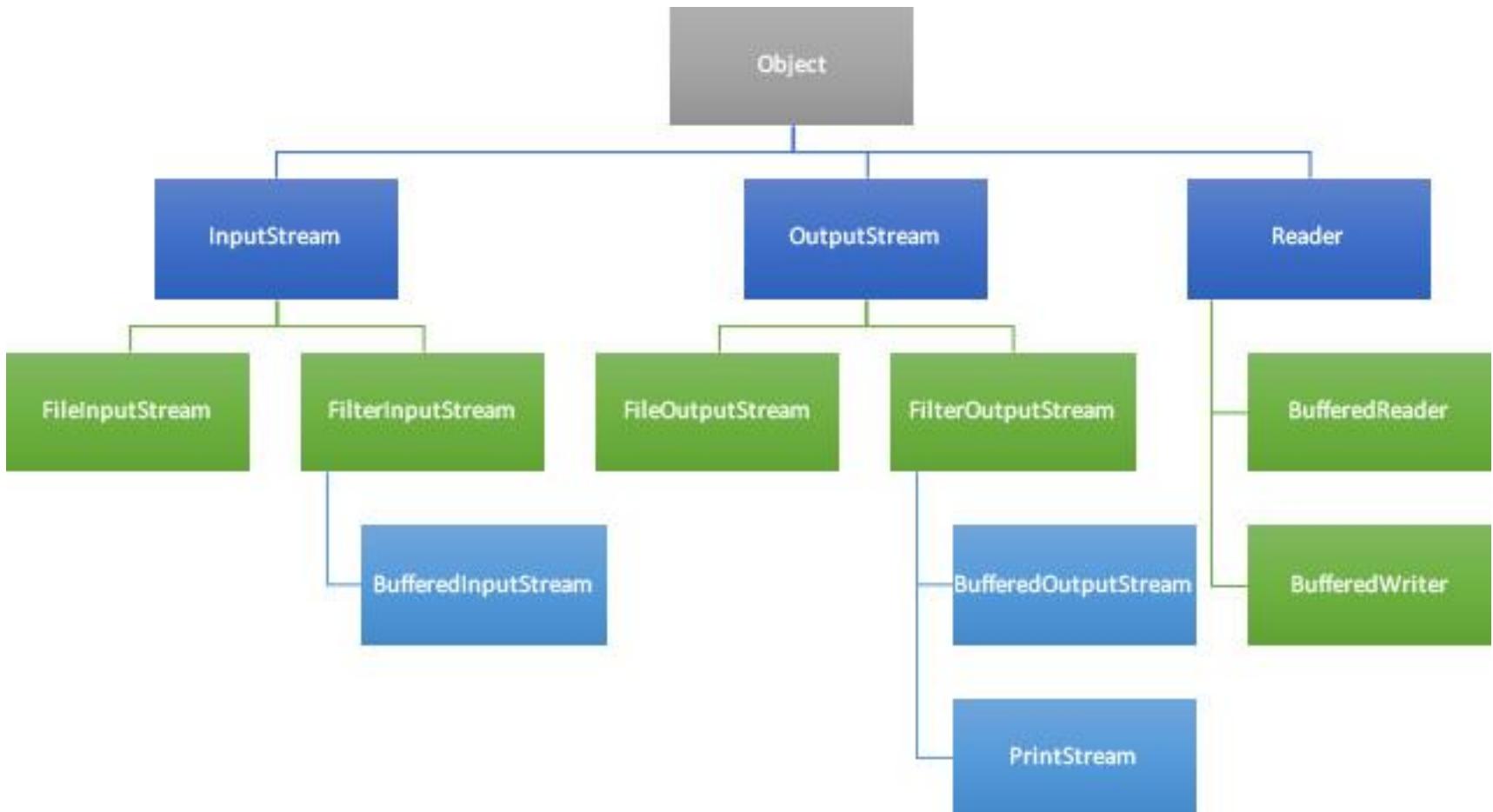
5 - Les packages de base

-> **java.nio : Fichiers et répertoires – java.nio.file.Path**

Méthode	Description
String <code>toString()</code>	Renvoyer la représentation de la chaîne du chemin, en éliminant les doubles barres obliques inverses.
Path <code>getFileName()</code>	Renvoyer le fichier ou le répertoire désigné par ce chemin; c'est le dernier élément de la séquence d'éléments nommés
int <code>getNameCount()</code>	Renvoyer le nombre d'éléments nommés dans le chemin.
Path <code>getName(int)</code>	Renvoyer le nom dans la position du chemin spécifié par le paramètre entier

5 - Les packages de base

-> **java.io & java.nio : Lecture et écriture de fichiers**



<https://developpement-informatique.com/article/265/lecture-et-ecriture-dans-un-fichier-en-java>

5 - Les packages de base

-> java.io & java.nio : Lecture et écriture de fichiers

```
import java.io.*;
import java.nio.file.*;
// charger les options
import static java.nio.file.StandardOpenOption.*;

public class Test {
    public static void main(String args[]) {
        Path chemin = Paths.get("intro.txt");
        String s = "Hello, world";

        // convertit String en un tableau d'octets
        byte[] data = s.getBytes();

        OutputStream output = null;
        try {
            // Un objet BufferedOutputStream est affecté à la référence OutputStream.
            output = new BufferedOutputStream(Files.newOutputStream(chemin, CREATE));
            // Ecrire dans le fichier
            output.write(data);

            // vider le tampon
            output.flush();

            // fermer le fichier
            output.close();

        // contenu de intro.txt
        Hello, world
    }
}
```

<https://developpement-informatique.com/article/265/lecture-et-ecriture-dans-un-fichier-en-java>

5 - Les packages de base

-> **java.io & java.nio : Lecture et écriture de fichiers**

```
import java.io.*;
import java.nio.file.*;

public class Test {
    public static void main(String args[]) {
        Path chemin = Paths.get("intro.txt");
        InputStream input = null;
        try {
            input = Files.newInputStream(chemin);

            BufferedReader reader = new BufferedReader(new InputStreamReader(input));
            String s = null;
            s = reader.readLine();
            System.out.println(s);
            input.close();

        } catch (IOException e) {
            System.out.println("Message " + e);
        }
    }
}
```

<https://developpement-informatique.com/article/265/lecture-et-ecriture-dans-un-fichier-en-java>



```
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);
}
```

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

BufferedReader pour lire un fichier ligne par ligne en Java

```
import java.io.*;

public class Main {
    public static void main(String[] args) {
        String file = "my-file.txt";
        try(BufferedReader br = new BufferedReader(new FileReader(file)))
        {
            String line;
            while ((line = br.readLine()) != null) {
                System.out.println(line);
            }
        }
        catch (IOException e) {
            System.out.println("An error occurred.");
            e.printStackTrace();
        }
    }
}
```

<https://www.delftstack.com/fr/howto/java/how-to-read-a-large-text-file-line-by-line-in-java/>

Stream pour lire un fichier ligne par ligne en Java

```
import java.io.*;
import java.nio.file.*;
import java.util.stream.*;

public class Main {
    public static void main(String[] args) {
        String file = "my-file.txt";
        try (Stream<String> stream = Files.lines(Paths.get(fileName))) {
            stream.forEach(System.out::println);
        }
        catch (IOException e) {
            System.out.println("An error occurred.");
            e.printStackTrace();
        }
    }
}
```

<https://www.delftstack.com/fr/howto/java/how-to-read-a-large-text-file-line-by-line-in-java/>

5 - Les packages de base-> java.io : Le Scanner

Le Scanner pour lire un fichier ligne par ligne en Java

```
import java.io.*;
import java.util.*;

public class Main {
    public static void main(String [] args) throws IOException {
        String fileName = "my-file.txt";
        Scanner scan = new Scanner(new File(fileName));
        while(scan.hasNextLine()){
            String line = scan.nextLine();
            System.out.println(line);
        }
    }
}
```

<https://www.delftstack.com/fr/howto/java/how-to-read-a-large-text-file-line-by-line-in-java/>



Sérialiser un objet consiste à le convertir en un tableau d'octets, que l'on peut ensuite écrire dans un fichier, envoyer sur un réseau au travers d'une socket etc..

Les objets écrits dans les flots doivent implémenter l'interface
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.

Remarque :

- ObjectInputStream et ObjectOutputStream permettent de stocker et restaurer des objets en binaire
- Le système inclus un système de version de classe (serialVersionUID)
- EOFException est lancée à la fin du flot
- java.lang.Object n'est pas sérialisable.

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

```
public class Marin implements Serializable {

    private static final long serialVersionUID = 1350092881346723535L;

    private String nom, prenom ;

    private int salaire ;

    public Marin(String nom, String prenom) {
        this.nom = nom ;
        this.prenom = prenom ;
    }

    public String toString() {
        StringBuffer sb = new StringBuffer() ;
        return sb.append(nom).append(" ").append(prenom).toString() ;
    }
}
```

<http://blog.paumard.org/cours/java/chap10-entrees-sorties-serialization.html>



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

```
// dans une méthode main
// on simplifie le code en retirant la gestion des exceptions
File fichier = new File("tmp/marin.ser") ;

// ouverture d'un flux sur un fichier
ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(fichier)) ;

// création d'un objet à sérialiser
Marin m = new Marin("Surcouf", "Robert") ;

// sérialization de l'objet
oos.writeObject(m) ;

// fermeture du flux dans le bloc finally

// dans une méthode main
// on simplifie le code en retirant la gestion des exceptions
File fichier = new File("tmp/marin.ser") ;

// ouverture d'un flux sur un fichier
ObjectInputStream ois = new ObjectInputStream(new FileInputStream(fichier)) ;

// désérialization de l'objet
Marin m = (Marin)ois.readObject() ;
System.out.println(m) ;

// fermeture du flux dans le bloc finally
```

<http://blog.paumard.org/cours/java/chap10-entrees-sorties-serialization.html>



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

```
// dans une méthode main
// on simplifie le code en retirant la gestion des exceptions
File fichier = new File("tmp/marin.ser") ;

// ouverture d'un flux sur un fichier
ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(fichier)) ;

// création d'un objet à sérialiser
Marin m = new Marin("Surcouf", "Robert") ;

// sérialization de l'objet
oos.writeObject(m) ;

// fermeture du flux dans le bloc finally

// dans une méthode main
// on simplifie le code en retirant la gestion des exceptions
File fichier = new File("tmp/marin.ser") ;

// ouverture d'un flux sur un fichier
ObjectInputStream ois = new ObjectInputStream(new FileInputStream(fichier)) ;

// désérialization de l'objet
Marin m = (Marin)ois.readObject() ;
System.out.println(m) ;

// fermeture du flux dans le bloc finally
```

<http://blog.paumard.org/cours/java/chap10-entrees-sorties-serialization.html>

5 - Les packages de base -> java.io : Sérialisation – Objets contenant des objets

Si l'on sérialise des objets en relation, alors ces objets sont eux aussi sérialisés, et placés dans le flux, sauf s'ils sont déclarés transient

La première manière est de créer deux méthodes dans la classe que l'on souhaite sérialiser :

- `writeObject()`
- `readObject()`

```
public class Marin implements Serializable {  
  
    private String nom, prenom ;  
    private int salaire ;  
  
    // suivent les getters / setters  
  
    // méthode readObject, utilisée pour reconstituer un objet sérialisé  
    private void readObject(ObjectInputStream ois)  
    throws IOException, ClassNotFoundException {  
  
        // l'ordre de lecture doit être le même que l'ordre d'écriture d'un objet  
        this.nom = ois.readUTF() ;  
        this.prenom = ois.readUTF() ;  
        // le salaire n'est pas relu, vu qu'il n'a pas été écrit  
    }  
  
    // méthode writeObject, utilisée lors de la sérialisation  
    private void writeObject(ObjectOutputStream oos)  
    throws IOException {  
  
        // écriture de toute ou partie des champs d'un objet  
        oos.writeUTF(nom) ;  
        oos.writeUTF(prenom) ;  
        // on choisit de ne pas écrire le salaire, qui ne fait  
        // pas partie de l'état d'une instance de marin  
    }  
}
```

<http://blog.paumard.org/cours/java/chap10-entrees-sorties-serialization.html>

Deux autres méthodes :

- **utilisation d'un *externalizer***
- **utilisation d'un objet proxy**

<http://blog.paumard.org/cours/java/chap10-entrees-sorties-serialization.html>



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





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)



[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.utilEnumeration<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](#))

La classe **java.util.Arrays** est une structure de données contenant un groupe d'éléments tous du même type, avec des adresses consécutives sur la mémoire (memory).

Le tableau a le nombre fixé d'éléments et vous ne pouvez pas changer sa taille.

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

Méthode	Description
static int binarySearch(type[] a, type key)	Rechercher dans le tableau spécifié la valeur de key spécifiée à l'aide de l'algorithme de recherche binaire.
static boolean equals(type[] a, type[] a2)	Renvoyer true si les deux tableaux du même type spécifiés sont égaux.
static void fill(type[] a, type val)	Affecter la valeur spécifiée à chaque élément du tableau spécifié
static void sort(type[] a)	Trier le tableau spécifié en ordre croissant
static void sort(type[] a, int fromIndex, int toIndex)	Trier la plage spécifiée du tableau en ordre croissant
static void parallelSort(type[] a)	Trier le tableau spécifié en ordre croissant à l'aide d'un tri en parallèle.
static void parallelSort(type[] a, int fromIndex, int toIndex)	Trier la plage spécifiée du tableau en ordre croissant en utilisant le tri en parallèle
static List<type> asList(type ... a)	Renvoyer une liste de taille fixe contenant les éléments des tableaux spécifiés.
static type[] copyOf(type[] originalArray, int newLength)	Copier le tableau spécifié, en tronquant ou en remplissant avec la valeur par défaut (si nécessaire) afin que la copie ait la longueur spécifiée.
static type[] copyOfRange(type[] originalArray, int fromIndex, int endIndex)	Copier la plage spécifiée du tableau spécifié dans un nouveau tableau.
static String toString(type[] originalArray)	renvoyer une représentation sous forme de chaîne du contenu du tableau spécifié.



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

```
import java.util.Arrays;

public class Test {
    public static void main(String args[]) {
        int[] T1 = { 2, 4, 5, 9, 5, 9 };
        int[] T2 = { 25, 30, 45, 50, 52, 60 };

        System.out.println(Arrays.binarySearch(T2, 45));
    }
}
```

```
1 import java.util.Arrays;
2
3 public class Test {
4     public static void main(String args[]) {
5         int[] T1 = { 2, 4, 5, 9, 5, 9 };
6         int[] T2 = { 25, 30, 45, 50, 52, 60 };
7
8         System.out.println(Arrays.equals(T1, T2));
9     }
10 }
```

```
import java.util.Arrays;

public class Test {

    public static void main(String args[]) {
        int[] T1 = { 2, 4, 5, 9, 5, 9 };
        Arrays.sort(T1);
        System.out.println(Arrays.toString(T1));
    }
}
```

```
import java.util.*;

public class Test {

    public static void main(String args[]) {
        int[] T1 = { 2, 4, 5, 9, 5, 9 };

        int[] res = Arrays.copyOf(T1, 3);
        System.out.println(Arrays.toString(res));

        int[] res2 = Arrays.copyOf(T1, 10);
        System.out.println(Arrays.toString(res2));
    }
}
```



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

```
import java.util.*;
import java.util.stream.*;

public class Main
{
    public static void main(String[] args)
    {
        List<Integer> slist = Arrays.asList(4,5,1,2,8,9,6);
        Collections.sort(slist);
        System.out.println("After Sorting: " + slist);
    }
}
```

<https://www.delftstack.com/fr/howto/java/sort-a-list-in-java/>

```
import java.util.*;
import java.util.stream.*;

public class SortListExample1
{
    public static void main(String[] args)
    {
        //returns a list view
        List<String> slist = Arrays.asList("Tanu", "Kamal", "Suman", "Lucky", "Bunty", "Amit");
        List<String> sortedList = slist.stream().sorted().collect(Collectors.toList());
        sortedList.forEach(System.out::println);
    }
}
```

<https://www.javatpoint.com/how-to-sort-a-list-in-java>

```
List<Movie> movies = Arrays.asList(
    new Movie("Lord of the rings"),
    new Movie("Back to the future"),
    new Movie("Carlito's way"),
    new Movie("Pulp fiction"));

movies.sort(Comparator.comparing(Movie::getTitle));

movies.forEach(System.out::println);
```

<https://dzone.com/articles/java-8-comparator-how-to-sort-a-list>

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

Les collections sont des objets qui permettent de gérer des ensembles d'objets

L'API Collections propose un ensemble d'interfaces et de classes dont le but est de stocker de multiples objets.

Il existe quatre grandes familles de collections qui sont définie par une interface de base :

- List : collection d'éléments ordonnés qui accepte les doublons
- Set : collection d'éléments non ordonnés par défaut qui n'accepte pas les doublons
- Map : collection sous la forme d'une association de paires clé/valeur
- Queue et Deque : collections qui stockent des éléments dans un certain ordre avant qu'ils ne soient extraits pour traitement

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

List : collection d'éléments ordonnés qui accepte les doublons

- `ls.add (élément E);` // Ajoute un élément
- `ls.remove (élément E);` // Supprime un élément
- `for (élément E: ls) {}` // Itère sur chaque élément
- `ls.toArray (new String [ls.length]);` // Convertit une liste de chaînes en un tableau de chaînes
- `ls.get (int index);` // Renvoie l'élément à l'index spécifié.
- `ls.set (int index, E element);` // Remplace l'élément à une position spécifiée.
- `ls.isEmpty ();` // Renvoie true si le tableau ne contient aucun élément, sinon il retourne false.
- `ls.indexOf (objet o);` // Renvoie l'index du premier emplacement de l'élément spécifié o ou, s'il n'est pas présent, renvoie -1.
- `ls.lastIndexOf (objet o);` // Renvoie l'index du dernier emplacement de l'élément spécifié o ou, s'il n'est pas présent, renvoie -1.
- `ls.size ();` // Renvoie le nombre d'éléments de la liste.

<https://learntutorials.net/fr/java/topic/2989/des-listes>

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

```
List<String> list = new ArrayList<>();
list.add("world");
System.out.println(list.indexOf("world"));      // Prints "0"
// Inserting a new value at index 0 moves "world" to index 1
list.add(0, "Hello");
System.out.println(list.indexOf("world"));      // Prints "1"
System.out.println(list.indexOf("Hello"));      // Prints "0"
```

```
public class User implements Comparable<User> {
    public final Long id;
    public final String username;

    public User(Long id, String username) {
        this.id = id;
        this.username = username;
    }

    @Override
    public String toString() {
        return String.format("%s:%d", username, id);
    }

    @Override
    /** The natural ordering for 'User' objects is by the 'id' field. */
    public int compareTo(User o) {
        return id.compareTo(o.id);
    }
}
```

<https://learntutorials.net/fr/java/topic/2989/des-listes>

```
List<User> users = Lists.newArrayList(
    new User(33L, "A"),
    new User(25L, "B"),
    new User(28L, ""));
Collections.sort(users);

System.out.print(users);
```

5 - Les packages de base -> java.util : Set<>

Set : collection d'éléments non ordonnés par défaut qui n'accepte pas les doublons, Le tri est déterminé par le type de set.

Exemple :

```
Set<String> h = new HashSet<String>(Arrays.asList("a", "b"));
```

Type de Set :

- HashSet - Tri aléatoire
- LinkedHashSet - Ordre d'insertion
- TreeSet - Par compareTo() ou Comparator

<https://learntutorials.net/fr/java/topic/3102/ensembles>

5 - Les packages de base -> java.util : Set<>

- HashSet - Tri aléatoire

```
Set<String> set = new HashSet<> ();
set.add("Banana");
set.add("Banana");
set.add("Apple");
set.add("Strawberry");

// Set Elements: ["Strawberry", "Banana", "Apple"]
```

- LinkedHashSet - Ordre d'insertion

```
Set<String> set = new LinkedHashSet<> ();
set.add("Banana");
set.add("Banana");
set.add("Apple");
set.add("Strawberry");

// Set Elements: ["Banana", "Apple", "Strawberry"]
```

- TreeSet - Par compareTo() ou Comparator

```
Set<String> set = new TreeSet<> ();
set.add("Banana");
set.add("Banana");
set.add("Apple");
set.add("Strawberry");

// Set Elements: ["Apple", "Banana", "Strawberry"]
```

```
Set<String> set = new TreeSet<> ((string1, string2) -> string2.compareTo(string1));
set.add("Banana");
set.add("Banana");
set.add("Apple");
set.add("Strawberry");

// Set Elements: ["Strawberry", "Banana", "Apple"]
```

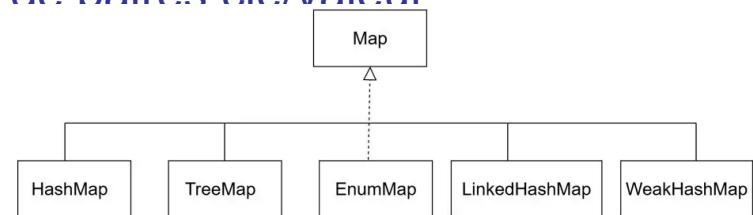
<https://learntutorials.net/fr/java/topic/3102/ensembles>

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

Map : collection sous la forme d'une association de paires clé/valeur

```
Map<Integer, String> listes = new HashMap<>();
```

```
listes.put(1, "David");
listes.put(2, "Laure");
listes.put(3, "Sam");
```



```
listes.forEach((id, name) -> {
    System.out.println("Key : " + id + " value : " + name);
});
```

```
// remove by value
customers.values().removeIf(value -> value.contains("Sally"));
```

```
// remove by key
customers.keySet().removeIf(key -> key != 1);
```

```
// remove by entry / combination of key + value
customers.entrySet().removeIf(entry -> entry.getKey() == 3);
```



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

```
package org.o7planning.map.ex;

import java.util.HashMap;
import java.util.Map;
import java.util.Set;

public class MapEx1 {

    public static void main(String[] args) {
        Map<String, String> phonebook = new HashMap<String, String>();

        phonebook.put("01000005", "Tom");
        phonebook.put("01000002", "Jerry");
        phonebook.put("01000003", "Tom");
        phonebook.put("01000004", "Donald");

        Set<String> phoneNumbers = phonebook.keySet();

        for (String phoneNumber : phoneNumbers) {
            String name = phonebook.get(phoneNumber);

            System.out.println("Phone Number: " + phoneNumber + " ==> Name: " + name);
        }
    }
}
```

```
package org.o7planning.map.ex;

import java.util.HashMap;
import java.util.Map;

public class MapEx3 {

    public static void main(String[] args) {
        Map<String, Integer> map = new HashMap<String, Integer>();

        map.put("AA", null);

        Integer value = map.get("AA");
        System.out.println("AA ==> " + value);

        boolean test = map.containsKey("AA");
        System.out.println("Contains AA? " + test); // true

        test = map.containsKey("BB");
        System.out.println("Contains BB? " + test); // false
    }
}
```

```
package org.o7planning.map.ex;

import java.util.Map;

public class Map_of_ex1 {

    public static void main(String[] args) {

        // Create an unmodifiable Map:
        Map<String, Integer> empMap = Map.of("E01", 1000, "E02", 2000, "E03", 1200);

        empMap.forEach((empNumber, salary) -> {
            System.out.println("Emp Number: " + empNumber + ", Salary: " + salary);
        });
    }
}
```

<https://devstory.net/13445/java-map#a64430893>



Queue et Deque : collections qui stockent des éléments dans un certain ordre avant qu'ils ne soient extraits pour traitement

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

Une Queue est une collection pour contenir des éléments avant le traitement.

Les files d'attente ordonnent généralement, mais pas nécessairement, les éléments d'une manière FIFO (premier entré, premier sorti).

```
public interface Queue<E> extends Collection<E> {
    boolean add(E e);

    boolean offer(E e);

    E remove();

    E poll();

    E element();

    E peek();
}
```

boolean add(E)	Insérer un élément dans Queue . S'il n'y a plus d'espace à insérer, cette méthode lancera une exception. La méthode renvoie true si l'insertion réussit.
boolean offer(E)	Insérer un élément dans Queue . Si Queue n'a plus d'espace ou échoue, la méthode renvoie false .
E poll()	Renvoyer le premier élément de Queue et le supprimer de Queue . Cette méthode renvoie à null si Queue ne dispose d'aucun élément.
E element()	Renvoyer le premier élément de Queue mais ne pas le supprimer de Queue . Cette méthode lance une exception si Queue ne dispose d'aucun élément.
E peek()	Renvoyer le premier élément de Queue mais ne pas le supprimer de Queue . Cette méthode renvoie à null si Queue ne dispose d'aucun élément.

<https://devstory.net/13451/java-queue#a64434995>



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

LinkedList représente une file d'attente traditionnelle

```
package org.o7planning.queue.ex;
```

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

```
import java.util.Queue;
```

```
public class LinkedListEx1 {
```

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

```
        // Create Queue
```

```
        Queue<String> queue = new LinkedList<String>();
```

```
        queue.offer("One");
```

One

```
        queue.offer("Two");
```

Two

```
        queue.offer("Three");
```

Three

```
        queue.offer("Four");
```

Four

```
        queue.offer("Five");
```

Five

```
        String current;
```

```
        while((current = queue.poll())!= null) {
```

```
            System.out.println(current);
```

```
}
```

```
}
```

<https://devstory.net/13451/java-queue#a64434995>



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

PriorityQueue<String> trie les éléments par ordre alphabétique.

```
package org.o7planning.queue.ex;

import java.util.PriorityQueue;
import java.util.Queue;

public class PriorityQueueEx1 {

    public static void main(String[] args) {
        // Create Queue
        Queue<String> queue = new PriorityQueue<String>();

        queue.offer("One");
        queue.offer("Two");
        queue.offer("Three");
        queue.offer("Four");
        queue.offer("Five");

        String current;

        while((current = queue.poll())!= null) {
            System.out.println(current);
        }
    }
}
```

Five
Four
One
Three
Two

<https://devstory.net/13451/java-queue#a64434995>

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

<https://devstory.net/13433/java-deque#a64435049>

Un Deque est une "file d'attente double", ce qui signifie que des éléments peuvent être ajoutés au début ou à la fin de la file.

La file d'attente seulement peut ajouter des éléments à la queue d'une file d'attente.

Deque est une sous-interface de Queue

public interface **Deque**<E> extends **Queue**<E>

boolean addFirst(E)	Insérer un élément devant Deque , cette méthode lancera une exception si la capacité de Deque est pleine. Cette méthode renvoie à true si l'insertion réussit.
boolean offerFirst(E)	Insérer un élément devant Deque . Si la capacité de Deque est déjà pleine ou l'insertion échoue, la méthode renverra à false , sinon true .
E removeFirst()	Renvoyer le premier élément de Deque et le supprimer de Deque . Cette méthode lancera une exception si Deque ne dispose d'aucun élément.
E pollFirst()	Renvoyer le premier élément de Deque et le supprimer de Deque . Cette méthode renverra null si Deque ne dispose d'aucun élément.
E removeLast()	Renvoyer le dernier élément de Deque et le supprimer de Deque . Cette méthode lancera une exception si Deque ne dispose d'élément.
E pollLast()	Renvoyer le dernier élément de Deque et le supprimer de Deque . Cette méthode renverra null si Deque ne dispose d'élément
E getFirst()	Renvoyer le premier élément de Deque mais ne le supprimer pas de Deque . Cette méthode lance une exception si Deque ne dispose d'élément.
E peekFirst()	Renvoyer le premier élément de Deque mais ne le supprimer pas de Deque . Cette méthode renvoie null si Deque ne dispose d'aucun élément.
E getLast()	Renvoyer le dernier élément de Deque mais ne le supprimer pas de Deque . Cette méthode lancera une exception si Deque ne dispose d'aucun élément.
E peekLast()	Renvoyer le dernier élément de Deque mais ne le supprimer pas de Deque . Cette méthode renvoie null si Deque n'a aucun élément.

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

<https://devstory.net/13433/java-deque#a64435049>

Deque est une interface, donc pour créer un objet **Deque**, il faut utiliser l'une de ses sous-classes, telles que **ArrayDeque**, **ConcurrentLinkedDeque**, **LinkedList**, **LinkedBlockingDeque**.

```
Deque<String> deque = new ArrayDeque<>();
```

```
Deque<String> deque = new LinkedList<>();
```

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

<https://devstory.net/13433/java-deque#a64435049>

```
package com.o7planning.deque.ex;
```

```
public class Customer {
```

```
    private Integer cusId;
```

```
    private String cusName;
```

```
    public Customer(Integer cusId, String cusName) {
```

```
        super();
```

```
        this.cusId = cusId;
```

```
        this.cusName = cusName;
```

```
}
```

```
    public Integer getcusId() {
```

```
        return cusId;
```

```
}
```

```
    public String getcusName() {
```

```
        return cusName;
```

```
}
```

```
}
```

```
package com.o7planning.deque.ex;
```

```
import java.util.ArrayDeque;
```

```
import java.util.Deque;
```

```
public class DequeEx1 {
```

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

```
        // Create a Deque with maximum capacity of 10 elements.
```

```
        Deque<Customer> deque = new ArrayDeque<Customer>(10);
```

```
        deque.addFirst(new Customer(1, "Tom"));
```

```
        deque.addFirst(new Customer(2, "Jerry"));
```

```
        deque.addLast(new Customer(3, "Donald"));
```

```
        Customer current = null;
```

```
        // Retrieve first element and remove it from Deque.
```

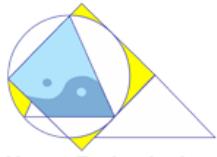
```
        while((current = deque.pollFirst())!= null) {
```

```
            System.out.println("Serving customer: " + current.getcusName());
```

```
        }
```

```
}
```

```
}
```



Yantra Technologies

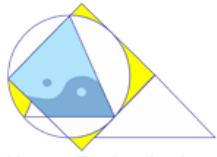


D.Palermo

Java Initiation
Version 4.0 – 09/2021

247

Copyright : Yantra Technologies 2004-2021



Yantra Technologies



D.Palermo

Java Initiation
Version 4.0 – 09/2021

248

Copyright : Yantra Technologies 2004-2021



- 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

<https://devstory.net/10393/java-socket#a766994>

Le Tutoriel de Java Socket

View more Tutorials:

 Java Basic

1.  Qu'est-ce qu'un socket?
2.  Un exemple simple avec Socket
3.  Exemple Socket + Thread

<http://sdz.tdct.org/sdz/introduction-aux-sockets-1.html#TPunmini-chatentreleclientetleserveur>

Tutoriel : Introduction aux sockets

Table des matières

- Introduction aux sockets
- Récupérer une adresse IP avec InetAddress
- Qu'est-ce qu'un socket ?
- Échange de message
- TP : un mini-chat entre le client et le serveur !

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);
}
```



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



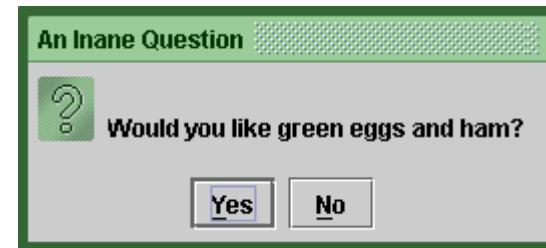
- Abstract Window Toolkit : AWT permettent d'écrire des interfaces graphiques indépendantes du système d'exploitation sur lesquel elles vont fonctionner
 - fenêtrage natif
 - Look and feel des applications natives
- Java Foundation Classes (JFC) : Swing
 - construit sur AWT
 - plus de fonctionnalités
 - look and feel configurable
 - architecture Modèle-Vue-Contrôleur
- JavaFX est un framework et une bibliothèque d'interface utilisateur issue du projet OpenJFX, **JavaFX** est le successeur de Swing

<https://perso.telecom-paristech.fr/hudry/coursJava/interSwing/index.html>
<https://fr.wikipedia.org/wiki/JavaFX>



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

Containers de haut niveau

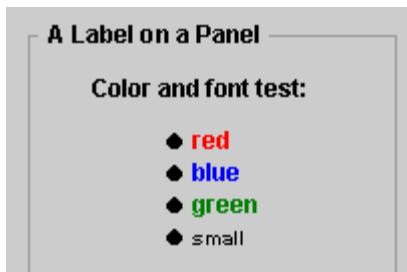


Dialog

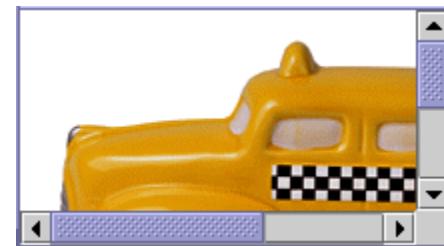


Frame

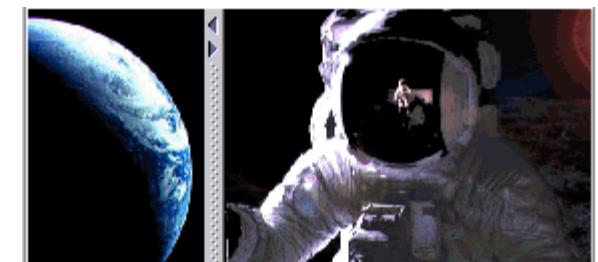
Containers généraux



Panel



ScrollPane



Splitpane



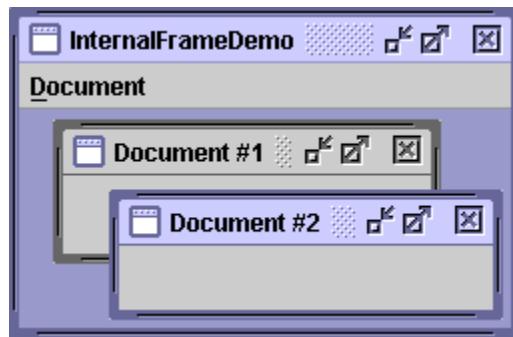
Tabbedpane



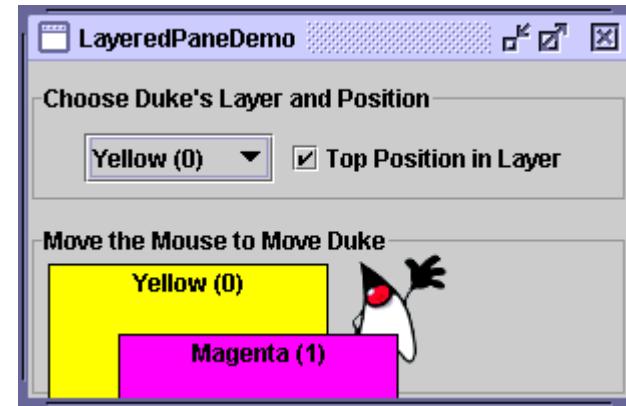
ToolBar



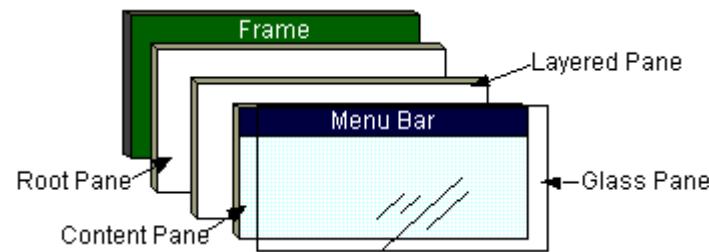
Containers spéciaux



Internal frame



Layred pane



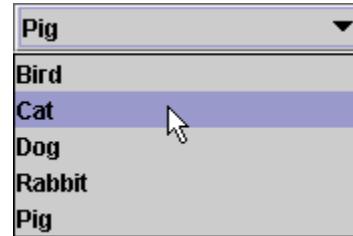
root pane



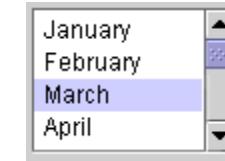
Basics



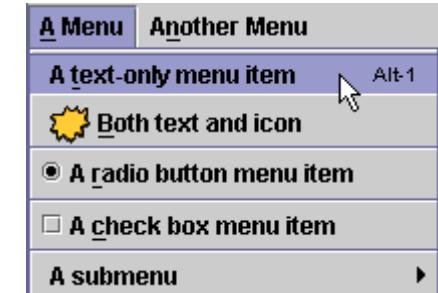
Buttons



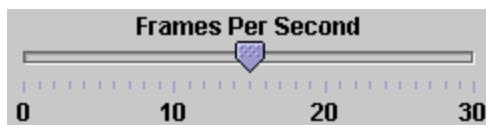
Combo box



List



Menu



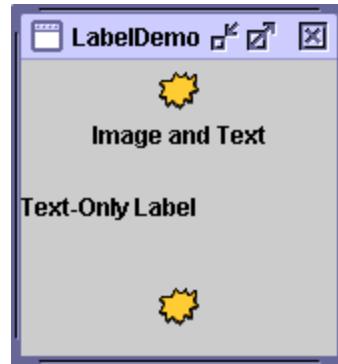
Slider



textfield



Spinner



Label



Progress bar



toolTip



Ecrans Interactifs



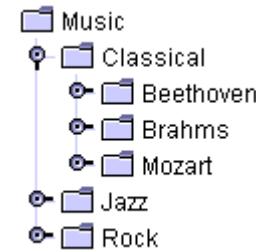
Color chooser

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

Table



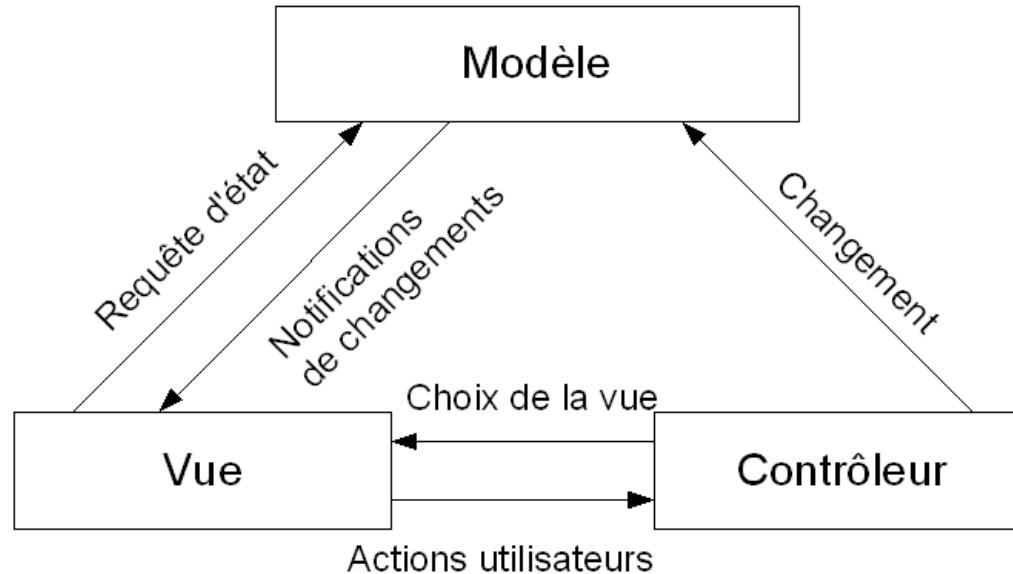
Text



Tree



- **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.

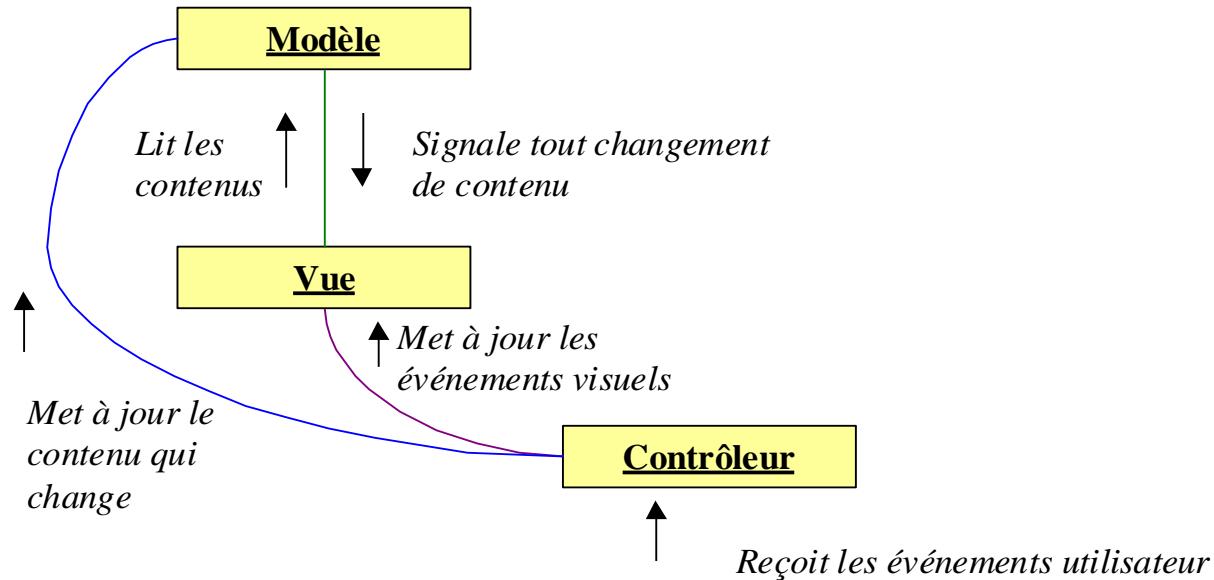


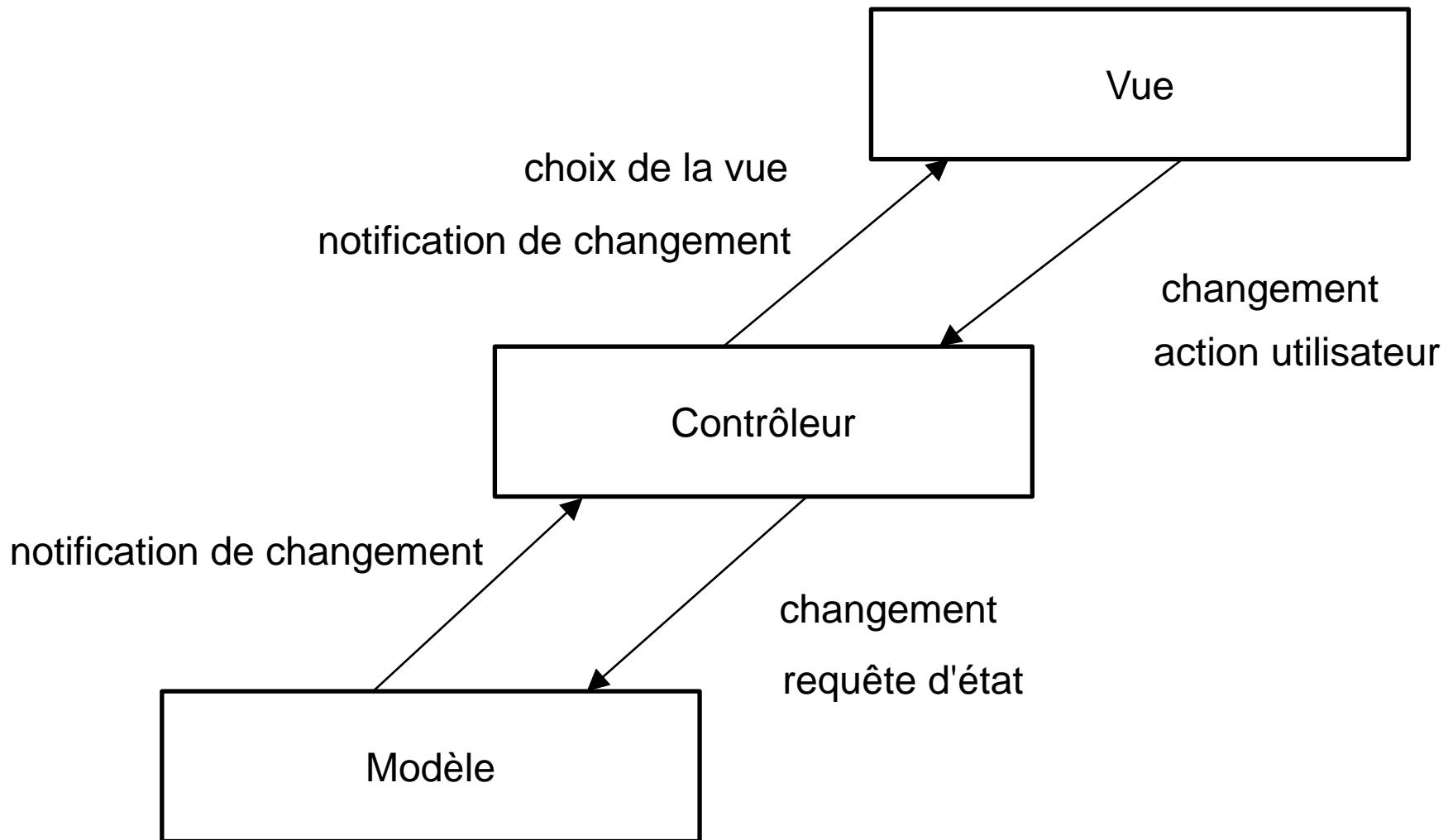
<http://baptiste-wicht.developpez.com/tutoriels/conception/mvc/#LI>

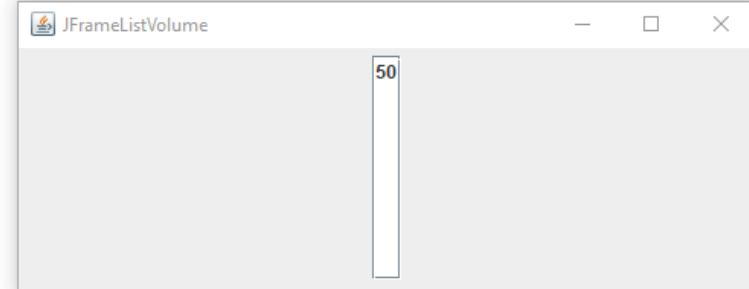
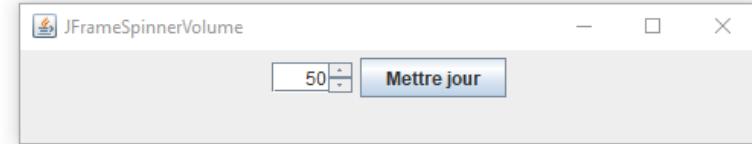
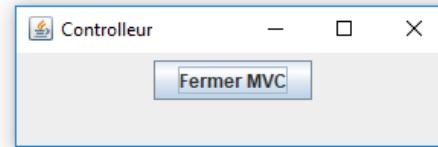
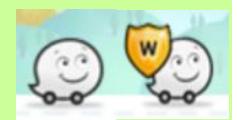


On implémente 3 classes séparées :

- 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

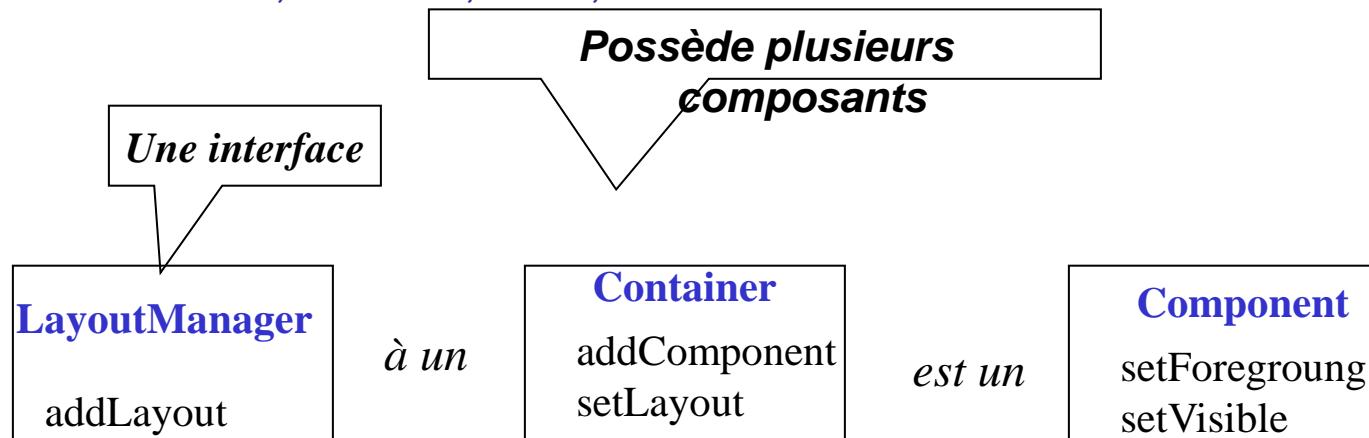






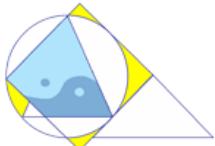


- Le layoutManager est une interface qui gère le placement des Components dans un Container
 - FlowLayout, BorderLayout, GridLayout, GridBagLayout, CardLayout
- chaque Container a un Layoutmanager
 - Jframe, JPanel, JDialog ...
- le Container contient une collection de Components
 - JLabel, JButton,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





- 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 -> Swing : LayoutManager : BorderManager

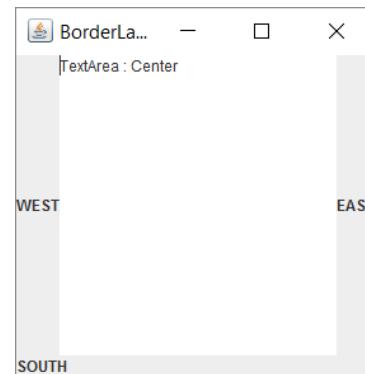
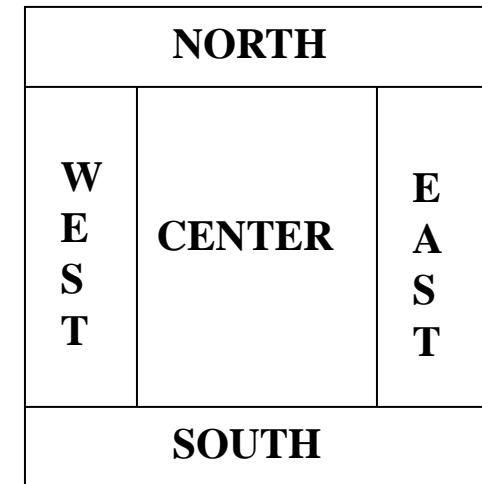


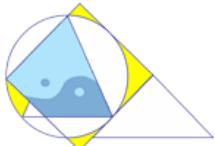
```
package exemplecours.IHM;

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

-> Swing : LayoutManager : GridLayout



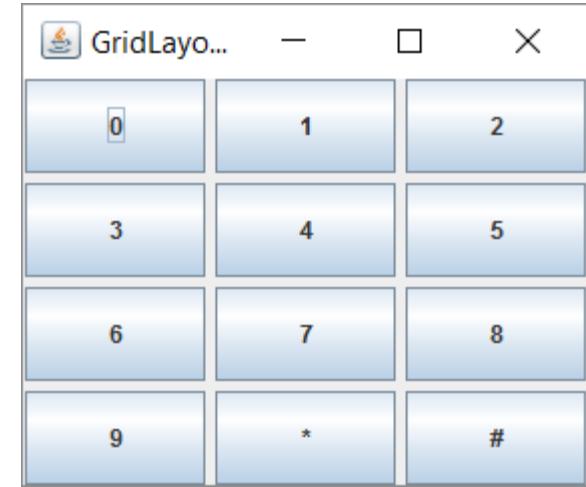
```
import java.awt.*;
import javax.swing.*;

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);
    }
}
```





Component

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

...

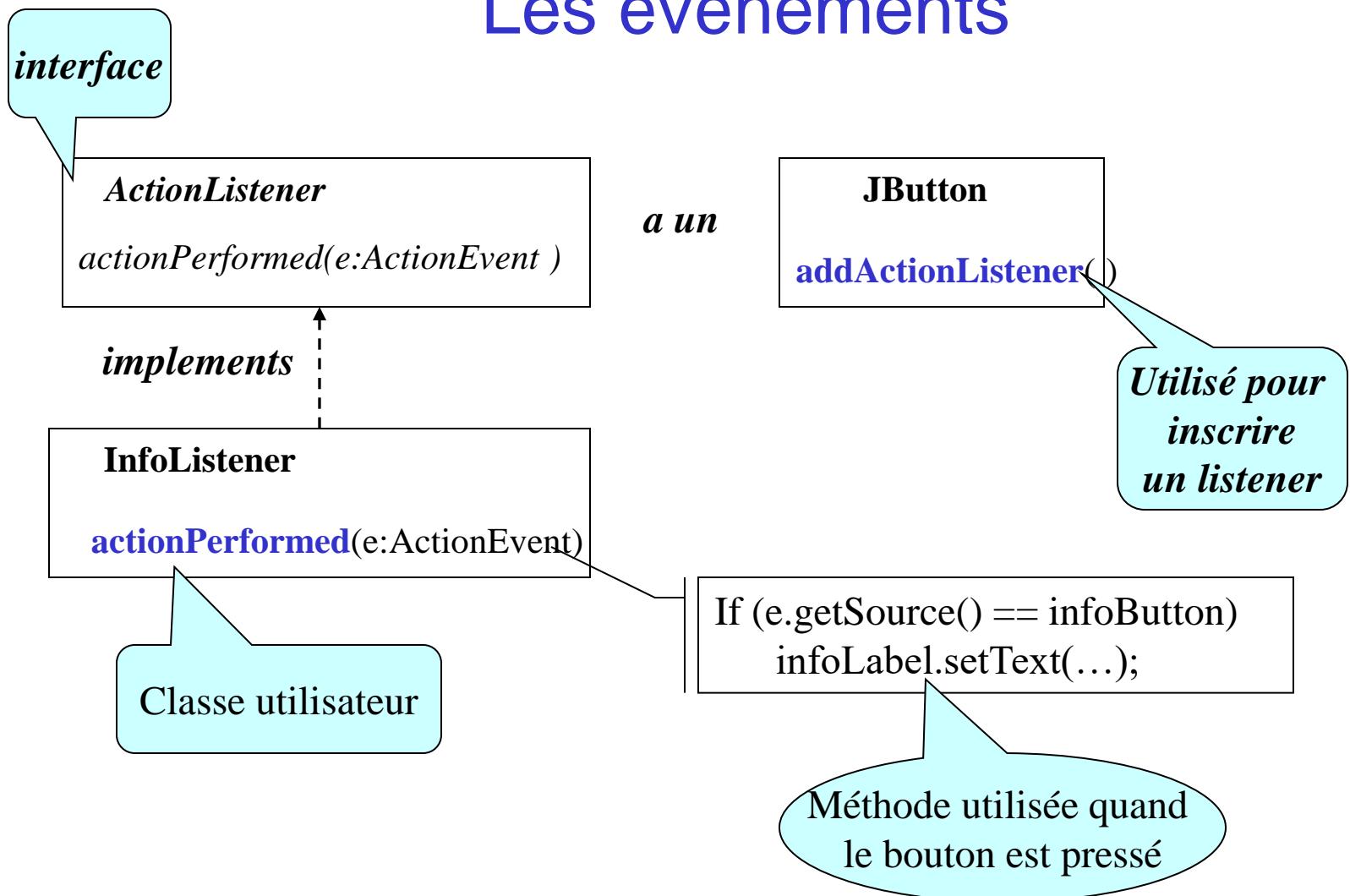
- Propriétés ajustables :
 - **Police** : setFont , getFont setFont (new Font(« Serif », Font.BOLD,14));
 - **Couleur de premier plan** : setForeground , getForeground
 setForeground (Color.blue);
 - **couleur du fond** : setBackground, getBackground
 setBackground(Color.white);
 - **curseur** : setCursor, 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)



- 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

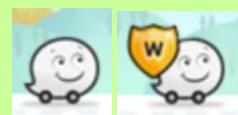


Les événements





7 - Les interfaces graphiques -> Swing : Gestion des Evénements



```
package exemplecours.IHM;

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

public class EventFrame extends JFrame {

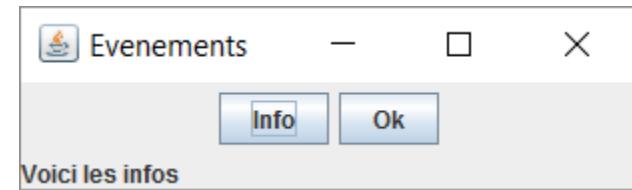
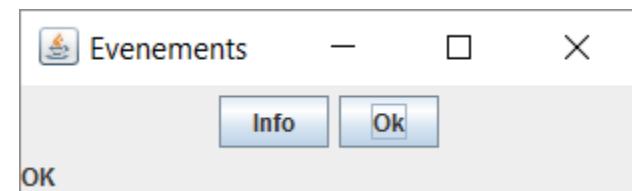
    private final JButton infoButton, okButton;
    private final JLabel infoLabel;

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

    class InfoListener implements ActionListener { ...11 lines ... }

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

}
```



7 - Les interfaces graphiques

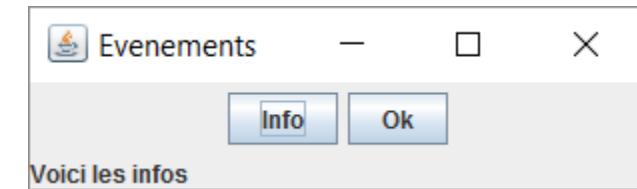
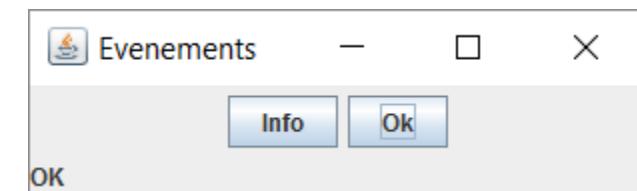
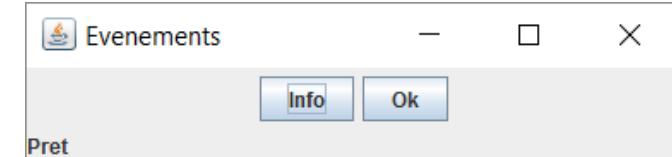
-> Swing : Gestion des Evénements



```

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 fenêtre
    setSize(400, 100);
    setVisible(true);
}

```



7 - Les interfaces graphiques

-> Swing : Gestion des Evénements

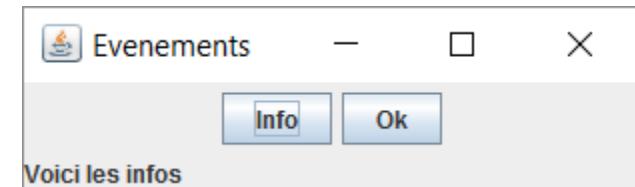
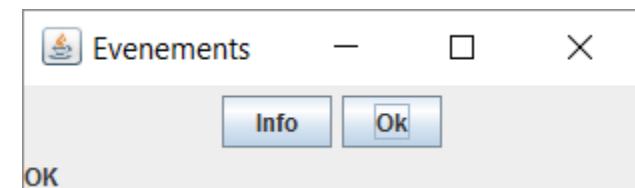


```

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);
}

```





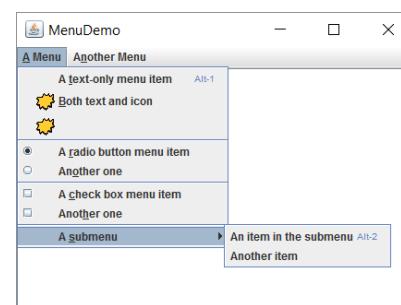
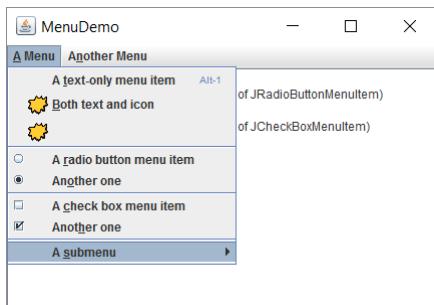
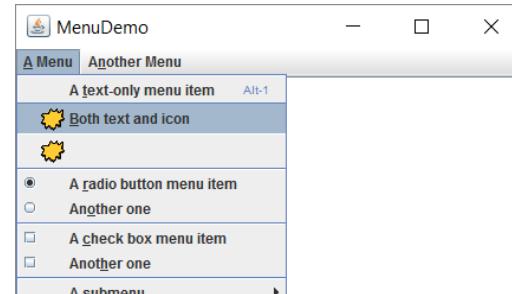
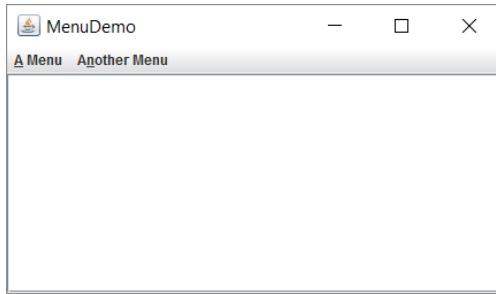
Principaux événements

Composant	Événement	Méthode
JButton	ActionEvent	actionPerformed
JCheckBox	ItemEvent	itemStateChanged
JList	ListSelectionEvent	valueChanged
JTextField	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 JMenuItem et JMenu





JOptionPane

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

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

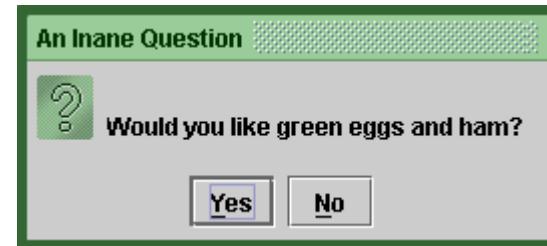
ERROR_MESSAGE

INFORMATION_MESSAGE

WARNING_MESSAGE

QUESTION_MESSAGE

PLAIN_MESSAGE



7 - Les interfaces graphiques -> Swing : Boites 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 -> Swing : Boîtes de dialogue

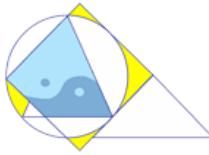


```
private static void createAndShowGUI() {  
    //Create and set up the window.  
    JFrame frame = new JFrame("MenuDemo");  
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
  
    //Create and set up the content pane.  
    MenuDemo demo = new MenuDemo();  
    frame.setJMenuBar(demo.createMenuBar());  
    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();}});  
     * remplacer par lambda fonctions  
     */  
    javax.swing.SwingUtilities.invokeLater(() -> {  
        createAndShowGUI();  
    });  
}
```

7 - Les interfaces graphiques -> Swing : Boites 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.out.println("Couldn't find file: " + path);  
        return null;  
    }  
}
```

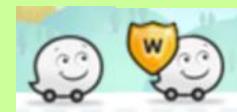


7 - Les interfaces graphiques -> Swing : Boîtes de dialogue



```
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 JMenuItems  
    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 -> Swing : 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 -> Swing : 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);

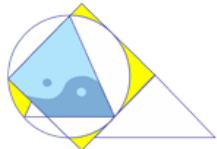
return menuBar;
}
```

7 - Les interfaces graphiques -> Swing : Boites 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 -> Swing : NetBeans



File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

<default config> Create Branch Switch To Branch

Projects X Files Services

BoutonTest.java X MenuDemo.java X IHM_HelloWorld.java X

To change layout manager of a container use Set Layout submenu from its context menu.

New File

Steps

1. Choose File Type

2. ...

Project: HelloWorld

Filter:

Categories:

- UML
- Java
- JavaFX
- Swing GUI Forms
- JavaBeans Objects
- AWT GUI Forms
- Unit Tests
- Persistence
- Python

File Types:

- JDialog Form
- JFrame Form
- JInternalFrame Form
- JPanel Form
- JApplet Form
- Bean Form
- Application Sample Form
- MDI Application Sample Form
- Master/Detail Sample Form
- OK / Cancel Dialog Sample Form

Description:

Creates a new JFC (Swing) Dialog. Dialogs are modal or modeless windows that are typically used to prompt users for input.

< Back Next > Finish Cancel Help

Output X Test Results Inspector Search Results

ExempleCours (run) #5 X HelloWorld (run) X

Palette X

Swing Containers

- Panel
- Tabbed Pane
- Split Pane
- Scroll Pane
- Tool Bar
- Desktop Pane
- Internal Frame
- Layered Pane

Swing Controls

- Label
- OK Button
- Toggle Button
- Check Box
- Radio Button
- Button Group
- Combo Box
- List

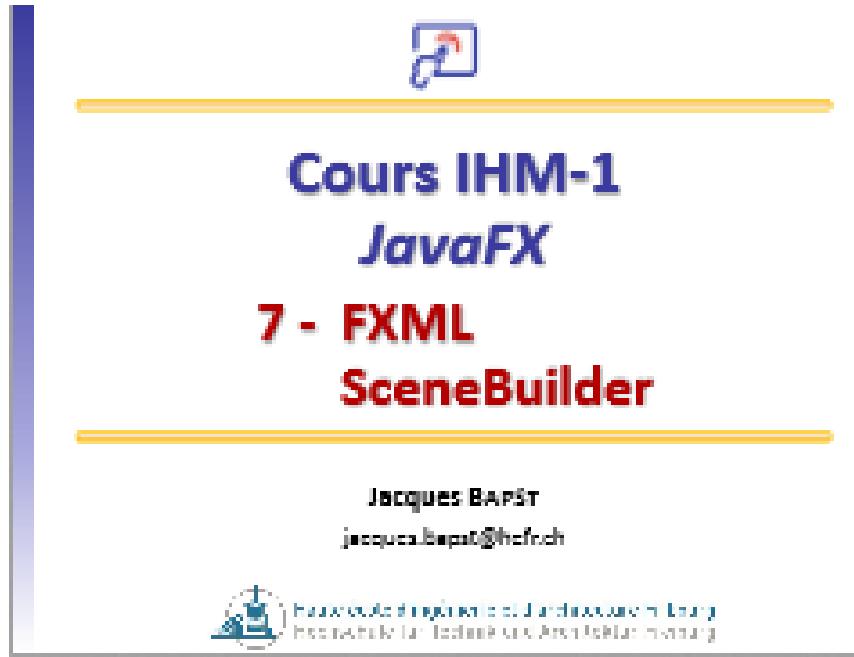
Other Components ...

<No Properties>

Other Components

7 - Les interfaces graphiques -> JavaFX

<https://docplayer.fr/10524039-Cours-ihm-1-javafx-7-fxml-scenebuilder-jacques-bapst.html>



<https://riptutorial.com/Download/javafx-fr.pdf>



Getting Started with JavaFX

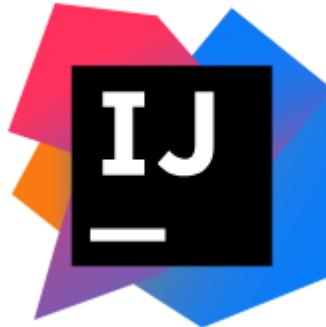
<https://openjfx.io/openjfx-docs/>

Tutoriel pour apprendre les concepts de base de JavaFX au travers d'un exemple

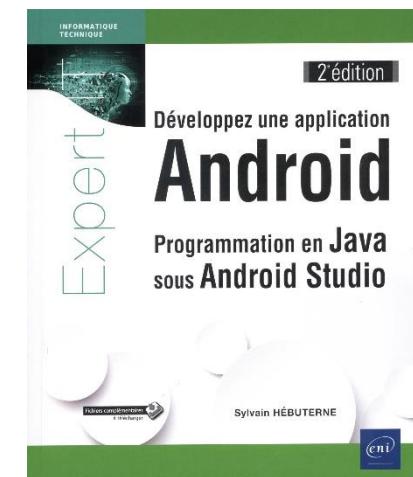
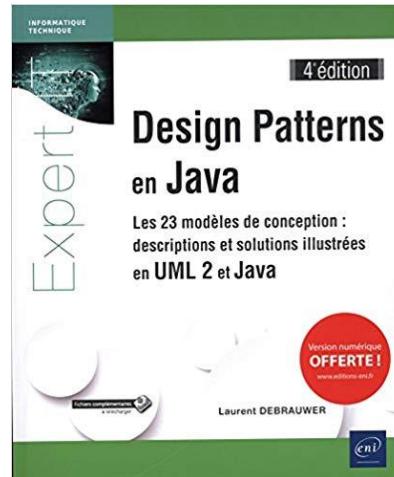
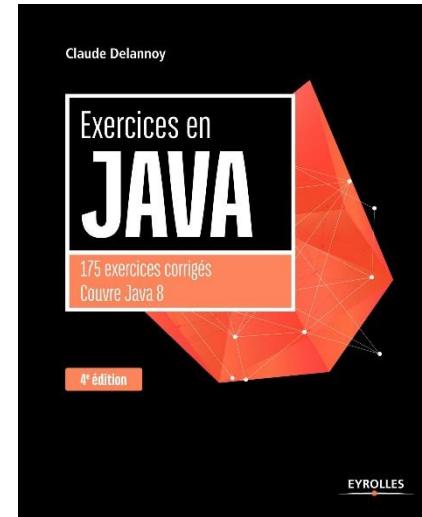
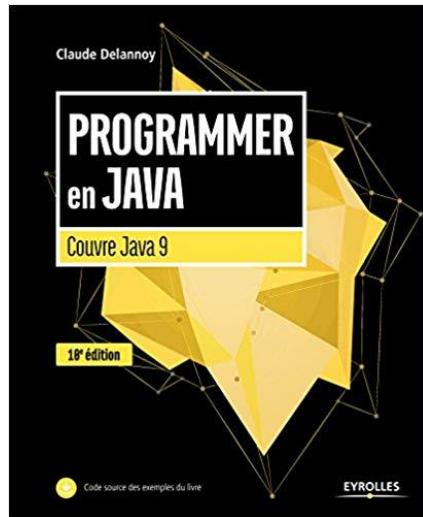
<https://xebia.developpez.com/tutoriels/java/concepts-javafx-illustration-exemple/>

Develop a basic JavaFX application

<https://www.jetbrains.com/help/idea/developing-a-javafx-application-examples.html>



8 - Bibliographies



8 - 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/>

IntelliJ Idea :

https://www.tutorialspoint.com/intellij_idea/index.htm

Netbeans :

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

Android :

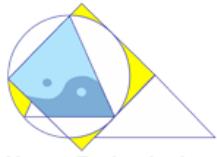
- <http://www.univ-orleans.fr/lifo/Members/Jean-Francois.Lalande/enseignement/android/cours-android.pdf>
- <https://olegoaer.developpez.com/cours/mobile/>
- <https://perso.univ-rennes1.fr/pierre.nerzic/Android/poly.pdf>



9 - Bibliographies

<https://objis.com/tutoriel-java-n12-acces-base-de-donnees-mysql/>

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



Yantra Technologies

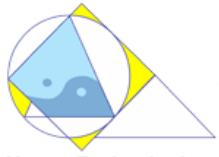


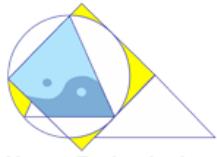
D.Palermo

Java Initiation
Version 4.0 – 09/2021

292

Copyright : Yantra Technologies 2004-2021





Yantra Technologies



D.Palermo

Java Initiation
Version 4.0 – 09/2021

294
Copyright : Yantra Technologies 2004-2021