Java SE

Programme

- Présentations
- Introduction à Java
- Découverte des IDEs
- Les bases de Java
- Java Orienté Objet
- JDBC

Simon Mielcarek

Développeur chez Synnaxium Studio

Formateur

Synnaxium

Présentez vous !

1. Introduction à Java

Développé par Oracle (Sun a été racheté en 2010)

Langage multi-plateforme

Basé sur une **machine virtuelle (JVM)**

Langage compilé

Langage orienté objet

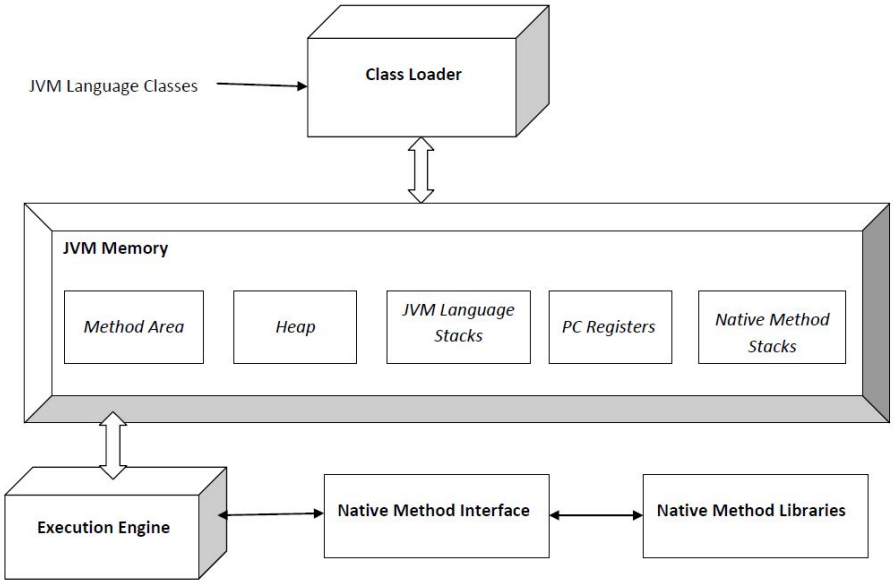
JAVA IS NOT JAVASCRIPT

JVM : Java Virtual Machine

“Appareil fictif” qui exécute le bytecode Java

Une machine virtuelle par Système d’exploitation

Schéma JVM

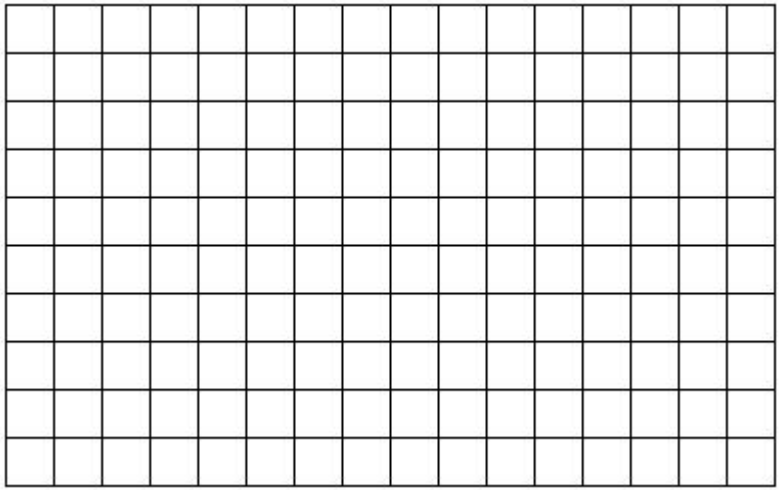

JVM Language Classes
Class Loader
JVM Memory
Method Area
Heap
JVM Language Stacks
PC Registers
Native Method Stacks
Execution Engine
Native Method Interface
Native Method Libraries

Gestion automatique de la mémoire

Principe simple :

- déterminer quels objets ne peuvent plus être utilisés par le programme
- récupérer l'espace utilisé par ces objets.

Exemple de garbage collection



Java propose plusieurs algorithmes

Avantages : le développeur est déchargé de la gestion mémoire

Inconvénients : le garbage collector consomme des ressources

Plusieurs versions de Java

Dependent d'un JDK

- SE : Standard Edition
- EE : Enterprise Edition
- ME : Mobile Edition

https://www.oracle.com/java/technologies/java-se-glance.html

2. Découverte des IDEs

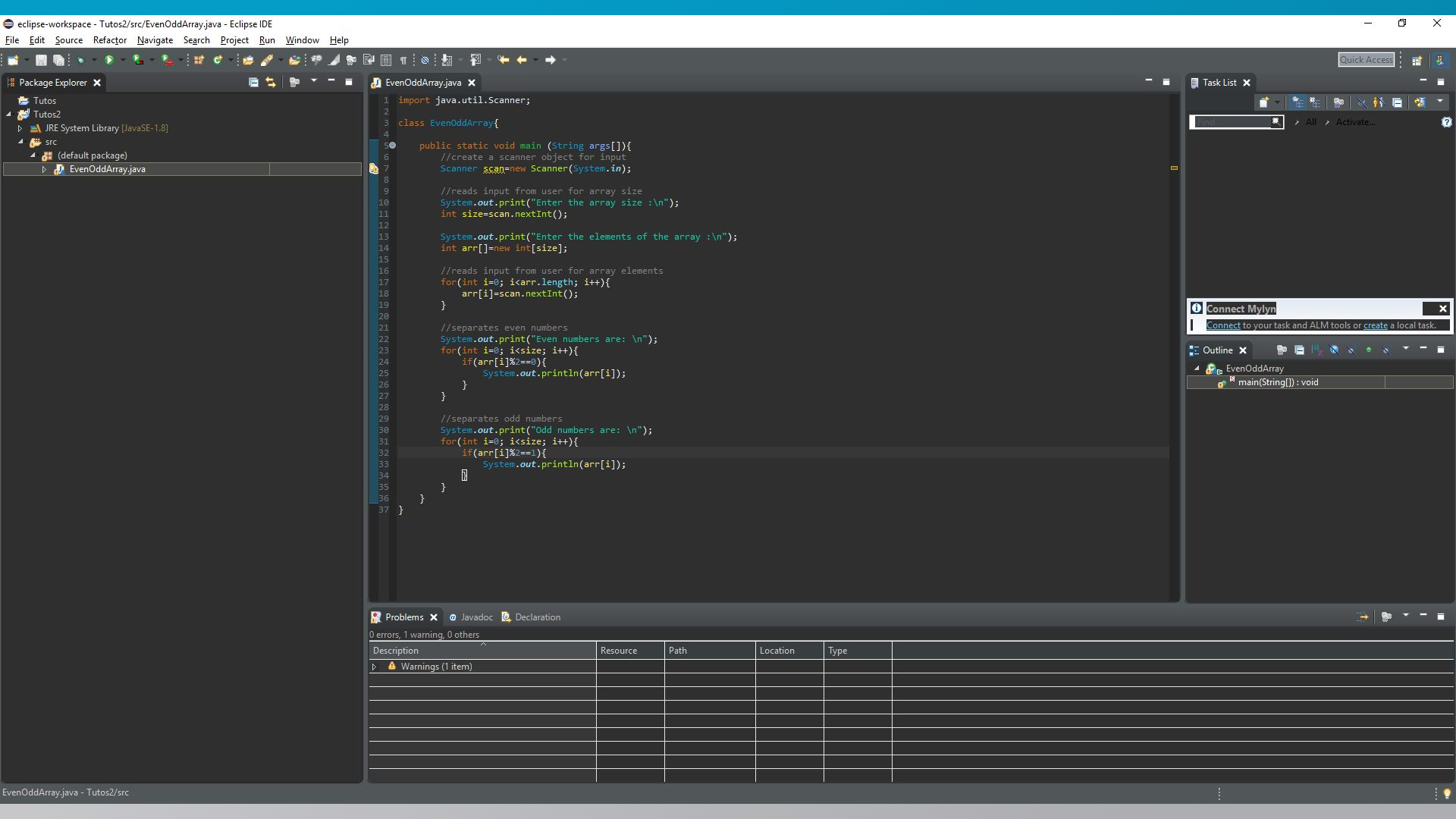
Eclipse & IntelliJ IDEA

Fondation Eclipse

Open Source

https://www.eclipse.org/

Essentiellement Orienté Java


eclipse-workspace - Tutos2/src/EvenOddArray.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Package Explorer
Tutos
Tutos2
JRE System Library [JavaSE-1.8]
src
(default package)
EvenOddArray.java
EvenOddArray.java
1 import java.util.Scanner;
2
3 class EvenOddArray{
4
5 public static void main (String args[]){
6 //create a scanner object for input
7 Scanner scan=new Scanner(System.in);
8
9 //reads input from user for array size
10 System.out.print("Enter the array size :\n");
11 int size=scan.nextInt();
12
13 System.out.print("Enter the elements of the array :\n");
14 int arr[]=new int[size];
15
16 //reads input from user for array elements
17 for(int i=0; i<arr.length; i++){
18 arr[i]=scan.nextInt();
19 }
20
21 //separates even numbers
22 System.out.print("Even numbers are: \n");
23 for(int i=0; i<size; i++){
24 if(arr[i]%2==0){
25 System.out.println(arr[i]);
26 }
27 }
28
29 //separates odd numbers
30 System.out.print("Odd numbers are: \n");
31 for(int i=0; i<size; i++){
32 if(arr[i]%2==1){
33 System.out.println(arr[i]);
34 }
35 }
36 }
37 }
Task List
Find
All
Activate...
Connect Mylyn
Connect to your task and ALM tools or create a local task.
Outline
EvenOddArray
main(String[]) : void
Problems
Javadoc
Declaration
0 errors, 1 warning, 0 others
Description
Resource
Path
Location
Type
Warnings (1 item)
EvenOddArray.java - Tutos2/src

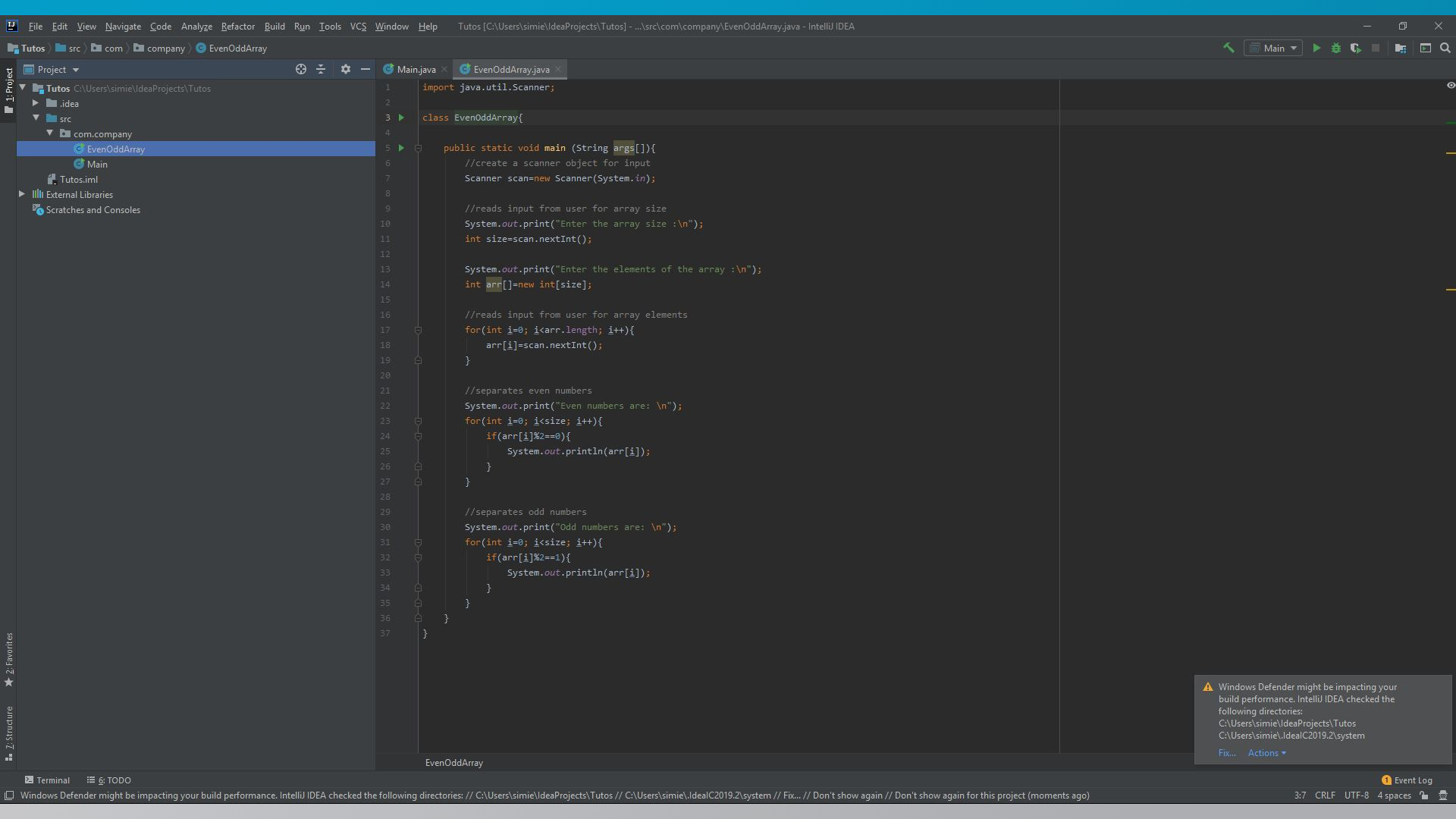
JetBrains

Open source pour la version community

Propriétaire pour la version Ultimate

http://www.jetbrains.com/idea/

Java, android, etc


File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
Tutos [C:\Users\simie\IdeaProjects\Tutos] - ...\src\com\company\EvenOddArray.java - IntelliJ IDEA
Tutos src com company EvenOddArray
Main
Project
Tutos C:\Users\simie\IdeaProjects\Tutos
.idea
src
com.company
EvenOddArray
Main
Tutos.iml
External Libraries
Scratches and Consoles
Main.java
EvenOddArray.java
import java.util.Scanner;
class EvenOddArray{
public static void main (String args[]){
//create a scanner object for input
Scanner scan=new Scanner(System.in);
//reads input from user for array size
System.out.print("Enter the array size :\n");
int size=scan.nextInt();
System.out.print("Enter the elements of the array :\n");
int arr[]=new int[size];
//reads input from user for array elements
for(int i=0; i<arr.length; i++){
arr[i]=scan.nextInt();
}
//separates even numbers
System.out.print("Even numbers are: \n");
for(int i=0; i<size; i++){
if(arr[i]%2==0){
System.out.println(arr[i]);
}
}
//separates odd numbers
System.out.print("Odd numbers are: \n");
for(int i=0; i<size; i++){
if(arr[i]%2==1){
System.out.println(arr[i]);
}
}
}
}
Windows Defender might be impacting your build performance. IntelliJ IDEA checked the following directories:
C:\Users\simie\IdeaProjects\Tutos
C:\Users\simie\.IdeaIC2019.2\system
Fix... Actions
EvenOddArray
Terminal
6: TODO
Event Log
Windows Defender might be impacting your build performance. IntelliJ IDEA checked the following directories: // C:\Users\simie\IdeaProjects\Tutos // C:\Users\simie\.IdeaIC2019.2\system // Fix... // Don't show again // Don't show again for this project (moments ago)
3:7 CRLF UTF-8 4 spaces

Découvrir un code et le décortiquer

```java
import java.util.Scanner;

class EvenOddArray{

    public static void main (String args[]){
        //create a scanner object for input
        Scanner scan=new Scanner(System.in);

        //reads input from user for array size
        System.out.print("Enter the array size :\n");
        int size=scan.nextInt();

        System.out.print("Enter the elements of the array :\n");
        int arr[]=new int[size];

        //reads input from user for array elements
        for(int i=0; i<arr.length; i++){
            arr[i]=scan.nextInt();
        }

        //separates even numbers
        System.out.print("Even numbers are: \n");
        for(int i=0; i<size; i++){
            if(arr[i]%2==0){
                System.out.println(arr[i]);
            }
        }

        //separates odd numbers
        System.out.print("Odd numbers are: \n");
        for(int i=0; i<size; i++){
            if(arr[i]%2==1){
                System.out.println(arr[i]);
            }
        }
    }
}
```

3. Les bases de Java

Syntaxe, programmes simples

Java est un langage typé

Entiers : *byte, short, int, long*

Décimaux : *float, double*

Booléens

Caractère : *char*

Cas spécial : String (objet)

Valeurs des types

Type	Taille	Syntaxe	Description	Intervalle
`char`	2 octets 16 bits	`'caractère'`	Une unité de code, suffisant à représenter un grand nombre de point de code, et même un caractère Unicode (UTF-16) `'b'` `'\u250C'`	`'\u0000'` à `'\uFFFF'`
`byte`	1 octet 8 bits		Un nombre entier de 8 bits (soit un octet) signé	`-128` à `127`
`short`	2 octets 16 bits		Un nombre entier de 16 bits signé entre -32 768 et +32 767	`-32768` à `32767`
`int`	4 octets 32 bits	*`[+|-]chiffres...`*	Un nombre entier de 32 bits signé entre -2 147 483 648 et +2 147 483 647	`-2147483648` à `2147483647`
`long`	8 octets 64 bits	*`[+|-]chiffres...`*`L`	Un nombre entier de 64 bits signé entre -9 223 372 036 854 775 808 et +9 223 372 036 854 775 807	`-9223372036854775808L` à `9223372036854775807L`
`float`	4 octets 32 bits	*`[+|-][chiffres].[chiffres]`* *`[E[+|-]chiffres]`*`F`	Un nombre à virgule flottante de 32 bits signé (simple précision)	• de 2^{-149} (`Float.MIN_VALUE`) à $2^{128} - 2^{104}$ (`Float.MAX_VALUE`), • `0.0F` , • $-\infty$ (`Float.NEGATIVE_INFINITY`), • $+\infty$ (`Float.POSITIVE_INFINITY`), • pas un nombre (`Float.NaN`).
`double`	8 octets 64 bits	*`[+|-][chiffres].[chiffres]`* *`[E[+|-]chiffres]`*`[D]`	Un nombre à virgule flottante de 64 bits signé (double précision)	• de 2^{-1074} (`Double.MIN_VALUE`) à $2^{1024} - 2^{971}$ (`Double.MAX_VALUE`), • `0.0D` , • $-\infty$ (`Double.NEGATIVE_INFINITY`), • $+\infty$ (`Double.POSITIVE_INFINITY`), • pas un nombre (`Double.NaN`).
`boolean`	1 octet	`false|true`	Une valeur logique	`false` (faux) ou `true` (vrai)

/!\ l'opérateur ! n'est utilisable que sur un type *bool* ou un résultat *booléen*

Opérateur	Action
-	Valeur négative
~	Complément à un
++	Incrémentation
--	Décrémentation
!	Négation

Opérateurs arithmétiques

Opérateur	Opération réalisée	Exemple	Résultat
+	Addition	6+4	10
-	Soustraction	12-6	6
*	Multiplication	3*4	12
/	Division	25/3	8.3333333333
%	Modulo (reste de la division entière)	25 mod 3	1

Opérateurs bit à bit

Opérateur	Opération réalisée	Exemple	Résultat
&	Et Binaire	45 & 255	45
|	Ou Binaire	99 | 46	111
^	Ou exclusif	99 ^ 46	77
>>	Décalage vers la droite (division par 2)	26>>1	13
<<	Décalage vers la gauche (multiplication par 2)	26<<1	52

Opérateur de comparaison

Opérateur	Opération réalisée	Exemple	Résultat
==	Egalité	2 == 5	false
!=	Inégalité	2 != 5	true
<	Inférieur	2 < 5	true
>	Supérieur	2 > 5	false
<=	Inférieur ou égal	2 <= 5	true
>=	Supérieur ou égal	2 >= 5	false
instanceof	Comparaison du type de la variable avec le type indiqué	O1 instanceof Client	True si la variable O1 référence un objet créé à partir de la classe client ou d'une sous-classe

Opérateurs logiques

Opérateur	Opération	Exemple	Résultat
&	Et logique	If ((test1) & (test2))	vrai si test1 et test2 est vrai
|	Ou logique	If ((test1) | (test2))	vrai si test1 ou test2 est vrai
^	Ou exclusif	If ((test1) ^ (test2))	vrai si test1 ou test2 est vrai mais pas si les deux sont vrais simultanément
!	Négation	If (! Test)	Inverse le résultat du test
&&	Et logique	If((test1) && (test2))	Idem et logique mais test2 ne sera évalué que si test1 est vrai
||	Ou logique	If ((test1) || (test2))	Idem ou logique mais test2 ne sera évalué que si test1 est faux

Concaténation : +

```
tortue = tortue + " " + i;
```

Opérateur ternaire : *? (ou condition ternaire)*

```
1 int x = 10, y = 20;
2 int max = (x < y) ? y : x ; //Maintenant, max vaut 20
```

Structures de contrôles : conditions


```
if (condition)
{
      Instruction 1;
      ...
      Instruction n;
}
```

```
if (condition)
{
      Instruction 1;
      ...
      Instruction n;
}
else
{
      Instruction 1;
      ...
      Instruction n;
}
```

```
if (condition1)
{
      Instruction 1
      ...
      Instruction n
}
else if (Condition 2)
{
      Instruction 1
      ...
      Instruction n
}
else if (Condition 3)
{
      Instruction 1
      ...
      Instruction n
}
else
{
      Instruction 1
      ...
      Instruction n
}
```

```
If (condition)
expression1 ;
else
expression2 ;
```

Structure de copntrôle : switch


```
Switch (expression)
{
     Case valeur1 :
               Instruction 1
               ...
               Instruction n
               Break ;
     Case valeur2 :
               Instruction 1
               ...
               Instruction n
               Break ;
     Default :
               Instruction 1
               ...
               Instruction n
}
```

Structure de contrôle : boucles

TANT QUE

```
while (condition)
{
     Instruction 1
     ...
     Instruction n
}
```

FAIRE… TANT QUE

```
do
{
     System.out.println(i);
     i++;
}
while(i<10);
```

POUR…

```
for(initialisation ;condition ;instruction d'itération)
{
     Instruction 1
     ...
     Instruction n

}
```

Interruption :

Break, continue, return

System.out

=> Sortie standard

System.in

=> Entrée standard

System.err

=> Sortie d’erreurs

Exercices simple sur les structures et les types

Découverte des tableaux

- Regroupement de classes
  - Organiser nos classes de manière logique
  - Ex : package lang contenant System
- Possibilité de créer des packages et sous packages
- Voyons le fonctionnement en direct

4. Java orienté objet

Classe, encapsulation, polymorphisme

Toutes les classes dans Java héritent de java.lang.Object

https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html

Cette classe possède déjà des méthodes et attributs

```
[liste de modificateurs] class NomDeLaClasse
    [extends NomDeLaClasseDeBase]
        [implements NomDeInterface1,NomDeInterface2,...]
{
    Code de la classe
}
```

Modificateurs :

public : indique que la classe peut être utilisée par toutes les autres classes. Sinon elle n'est utilisable que par les classes du même *package*

abstract : indique que la classe est abstraite et doit être implémentée pour pouvoir être instanciée.

final : la classe ne peut pas être utilisée comme classe parente pour de l'héritage

extends : la classe hérite de la classe qui suit

extends ClasseParent

implements : la classe utilise l'interface qui suit

implements ClasseInterface

Visibilité :

public : visible par toutes les autres classes

private : visible uniquement au sein de la classe

protected : visible au sein de la classe et par les enfants

Rien : visible au sein d'un même package

static : attribut ou méthode de classe.

La déclaration des attributs se fait assez facilement :

```
[private | protected | public] typeDeLaVariable nomDeLaVariable ;
```

Ils peuvent être de types primitifs ou des classes disponibles et visibles

S’ils ne sont pas visibles il faudra les importer

```
[modificateurs] typeDeRetour nomDeLaMethode ([listeDesParamètres])
      [throws listeException]
{
    CODE DE LA METHODE
}
```

Modificateurs

abstract : la méthode est abstraite, elle ne contient pas de code et devra être implémentée par les classes filles

final : La méthode ne pourra pas être redéfinie par une classe enfant

Même nom que la classe

Au moins un constructeur obligatoire

Peut prendre ou non des arguments

Méthode appelée lors de la destruction de l'objet

Signature forcée

```
protected void finalize() throws Throwable
{

}
```

Finalise

Il ne peut y avoir qu'un seul destructeur

Permet de libérer des ressources utilisées

this : fait référence à l'objet en cours

super : fait référence à la méthode parente en cas d'héritage

```java

public class Film {
    String titre;
    String synopsis;
    int annee;

    // Constructeur
    Film(String titre, String synopsis, int annee){
        // On remplit l'attribut de l'objet par le paramètre
        this.titre = titre;
        this.synopsis = synopsis;
        this.annee = annee;
    }

    @Override
    public String toString() {
        return this.titre+" ("+this.annee+") : "+this.synopsis;
    }
}

```

En java il est possible de rajouter des annotations pour faciliter la lecture et l'utilisation :

@Deprecated

@Override

Mot clé : *new*

Créer une instance d'un objet

On peut désormais le manipuler

```
public static void main(String[] args) {
    Film godzilla = new Film("Godzilla", "C'est un monstre qui fait graouh", 1953);
    Film jp = new Film("Jurassic Park", "C'est des dinos qui font graouh", 1993);

    // On s'était plantés sur l'année on la corrige
    godzilla.annee = 1954;

    // On affiche les films dans la console :
    System.out.println(jp.toString());
    System.out.println(godzilla.toString());
}
```

Que fait ce code ?

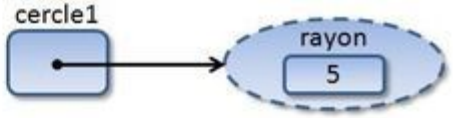

```java
public class Main {

    public static void main(String[] args) {
        int a,b;

        a = 5;
        b = 10;
        a = b;
        b = 12;
    }

}
```

```java

public class Main {

    public static void main(String[] args) {
        Film godzilla = new Film("Godzilla", "C'est un monstre qui fait graouh", 1953);
        Film jp = new Film("Jurassic Park", "C'est des dinos qui font graouh", 1993);

        // On s'était plantés sur l'année on la corrige
        godzilla.annee = 1954;

        godzilla = jp;

        jp.titre = "Jurassic Park 2";

        // On affiche les films dans la console :
        System.out.println(jp.toString());
        System.out.println(godzilla.toString());
    }

}
```

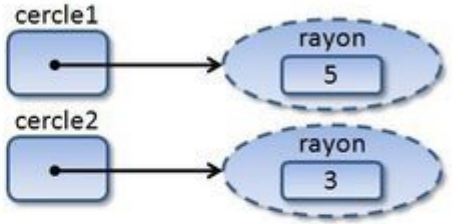
```
<terminated> Main [Java Application] C:\Program Files\Java\jre1.8.0_221\bin\java
Jurassic Park 2 (1993) : C'est des dinos qui font graouh
Jurassic Park 2 (1993) : C'est des dinos qui font graouh
```

Pourquoi ?

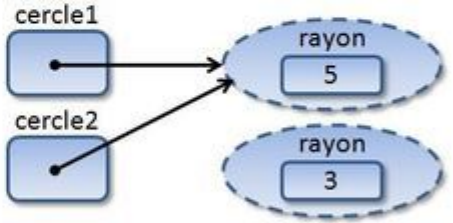

```
Cercle cercle1;
cercle1 = new Cercle(5);
```



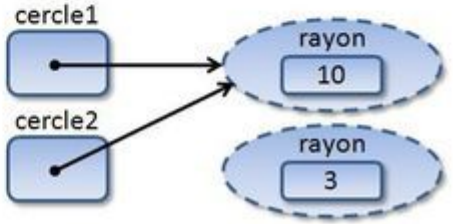

```
Cercle cercle2 = new Cercle(3);
```




```
cercle2 = cercle1;
```




```
cercle1.Rayon = 10;
```



En objet on ne stocke pas directement les informations

La variable contient une référence à l'emplacement mémoire de l'objet

Si on remplace cette référence par une autre… On ne fait plus appel au même objet

Etudions ensemble : java.lang.Object

Comparaison d’objets : *equals*

Exercices

Quelques exercices pour s’habituer à la création et la manipulation des classes et des objets

Getters, setters, portée des attributs

	Class	Package	Subclass (same pkg)	Subclass (diff pkg)	World
public	+	+	+	+	+
protected	+	+	+	+	
no modifier	+	+	+		
private	+				

+ : accessible blank : not accessible

Utilisation des getters et setters :

Permet de contrôler facilement qui peut accéder à quelles informations

Permet de contrôler la modification des informations

Une classe abstraite est une classe qui ne peut pas être instanciée.

Elle sert uniquement de modèle et devra être héritée

Comme les classes abstraites les interfaces ne peuvent pas être instanciées.

Elles servent à décrire un comportement particulier.

Elles permettent de faire de “l’héritage multiple”

Les Interfaces ne contiennent que des signatures de méthodes.

Ces méthodes devront être redéfinies pour pour être utilisées.

L’enumeration est une structure qui représente un ensemble de données.

C’est une liste de valeurs possibles

Exemple :
Elements : Feu, Eau, Terre, Vent

Enumerations


```
1 public enum Langage {
2   JAVA,
3   C,
4   CPlus,
5   PHP;
6 }
```

Interface permettant de gérer des Objets contenant plusieurs éléments.

Sorte de “Tableau” dynamique

https://docs.oracle.com/javase/8/docs/api/java/lang/Iterable.html

List est une implémentation de “Iterable”

C’est l’une des plus simple à utiliser.

https://docs.oracle.com/javase/8/docs/api/java/util/List.html

Une exception est une erreur qui amène souvent à l'arrêt du programme

En java ces erreurs peuvent être gérées.

Vous en avez déjà sûrement rencontré lors des TP

NullPointerException : l'emplacement mémoire ciblé est vide

ArrayOutOfBound : Vous avez essayé d'accéder à un emplacement de tableau qui n'existe pas

ArithmeticException, etc

Le bloc Try Catch permet de gérer les Exceptions

Le bloc Try est le bloc sensible aux exceptions

Le bloc Catch est le bloc qui permet de les résoudre ou de les interpréter

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

  int j = 20, i = 0;
  try {
    System.out.println(j/i);
  } catch (ArithmeticException e) {
    System.out.println("Division par zéro !");
  }
  System.out.println("coucou toi !");
}
```

Le bloc Finally est joué systématiquement

Il permet surtout de vérifier l'intégrité d'un système : fermeture de fichiers, clotûre de BDD, etc

Pour créer une Exception spéciale, étendre la classe Exception

Les méthodes peuvent envoyer des exceptions par le mot clé “*throws*”