

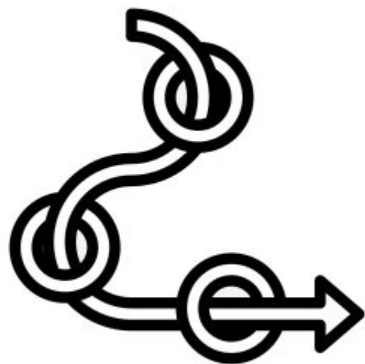


Le génie pour l'industrie

# **INF147/155 - Programmation Procédurale**

## **Cours #1**

**École de technologie supérieure - Été 2024**



## **Plan de Cours**

**Cours #1 - Intro. programmation OO**

**Cours #2 - Programmation Java**

**Cours #3 - Programmation Java (Suite)**

**Cours #4 - Algorithmie + révision**

**Intra #1 (sem 5)**

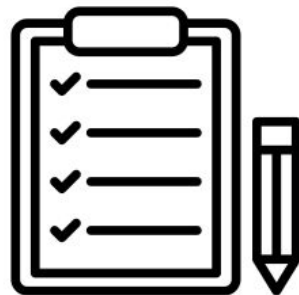
**Cours #5-6 - Algorithmie (suite) et Collections Java**

**Cours #7 - Programmation OO**

**Cours #8 - Programmation OO + révision**

**Intra #2 (sem 10)**

**Cours #9-11 - Java Swing et mise en pratique de l'OO + révision**



## **Plan de la séance**

- 1. Premier programme**
  - **Eclipse/IntelliJ - Hello world**
- 2. Interface Utilisateur Console**
  - **println**
  - **Scanner**
- 3. Programmation orienté-objet**
  - **Introduction**
  - **Pourquoi l'OO**
  - **Principes**
- 4. Exercices/démo**
  - **Classes de base**
  - **Héritage**
  - **Polymorphisme**
  - **Interface**



# **1. Premier programme**

## Installation Java 8

### 1. Premier programme

- Télécharger et installer: <https://www.java.com/fr/download/>
- Garder les options par défaut pour éviter les soucis.

Vous avez ensuite le choix entre deux environnements de développements (IDE):

- **Eclipse (slides 9 à 16)**
- **IntelliJ (slides 17 à 20)**

IntelliJ est généralement considéré comme le meilleur de deux.

Personnellement, j'utilise Eclipse, parce qu'il va mieux, selon moi, pour la gestion multi-projets. En classe, je vais utiliser Eclipse, mais n'hésitez pas à choisir IntelliJ

PS: Je vous conseille également d'installer Notepad++:

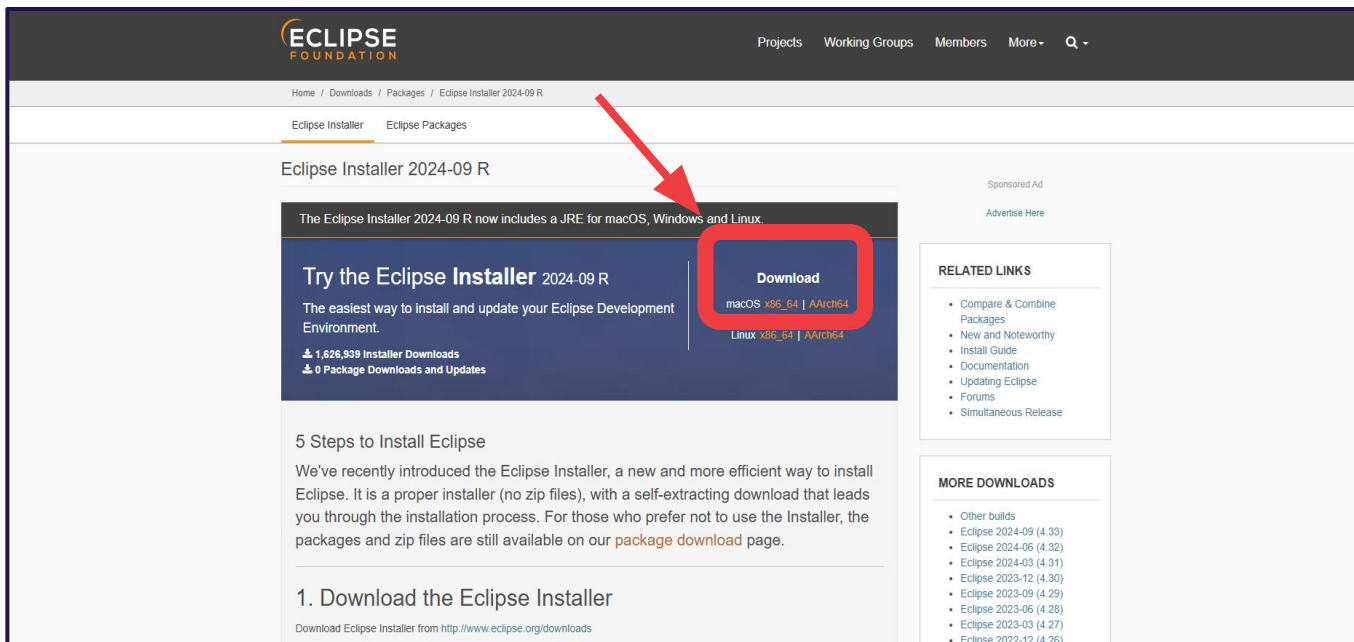
<https://notepad-plus-plus.org/downloads/>



# Eclipse - Installation

## 1. Premier programme

- Télécharger, puis lancer l'installateur:  
<https://www.eclipse.org/downloads/packages/installer>



The screenshot shows the Eclipse Foundation website's download page for the Eclipse Installer 2024-09 R. The page features a dark header with the Eclipse Foundation logo and navigation links. Below the header, the breadcrumb trail reads 'Home / Downloads / Packages / Eclipse Installer 2024-09 R'. The main content area has a dark blue background with the text 'Try the Eclipse Installer 2024-09 R' and 'The easiest way to install and update your Eclipse Development Environment.' A red arrow points to a 'Download' button, which is highlighted with a red box. The button lists download links for macOS x86\_64, AArch64, and Linux x86\_64, AArch64. To the right of the main content, there is a 'RELATED LINKS' section with links to 'Compare & Combine Packages', 'New and Noteworthy', 'Install Guide', 'Documentation', 'Updating Eclipse', 'Forums', and 'Simultaneous Release'. Below that is a 'MORE DOWNLOADS' section with links to various Eclipse builds.

ECLIPSE FOUNDATION

Projects Working Groups Members More- Q -

Home / Downloads / Packages / Eclipse Installer 2024-09 R

Eclipse Installer Eclipse Packages

Eclipse Installer 2024-09 R

The Eclipse Installer 2024-09 R now includes a JRE for macOS, Windows and Linux.

Try the Eclipse **Installer** 2024-09 R

The easiest way to install and update your Eclipse Development Environment.

1,826,939 Installer Downloads  
0 Package Downloads and Updates

**Download**

macOS x86\_64 | AArch64  
Linux x86\_64 | AArch64

Sponsored Ad  
Advertise Here

RELATED LINKS

- Compare & Combine Packages
- New and Noteworthy
- Install Guide
- Documentation
- Updating Eclipse
- Forums
- Simultaneous Release

MORE DOWNLOADS

- Other builds
- Eclipse 2024-09 (4.33)
- Eclipse 2024-06 (4.32)
- Eclipse 2024-03 (4.31)
- Eclipse 2023-12 (4.30)
- Eclipse 2023-09 (4.29)
- Eclipse 2023-06 (4.28)
- Eclipse 2023-03 (4.27)
- Eclipse 2022-12 (4.26)

5 Steps to Install Eclipse

We've recently introduced the Eclipse Installer, a new and more efficient way to install Eclipse. It is a proper installer (no zip files), with a self-extracting download that leads you through the installation process. For those who prefer not to use the Installer, the packages and zip files are still available on our [package download page](#).

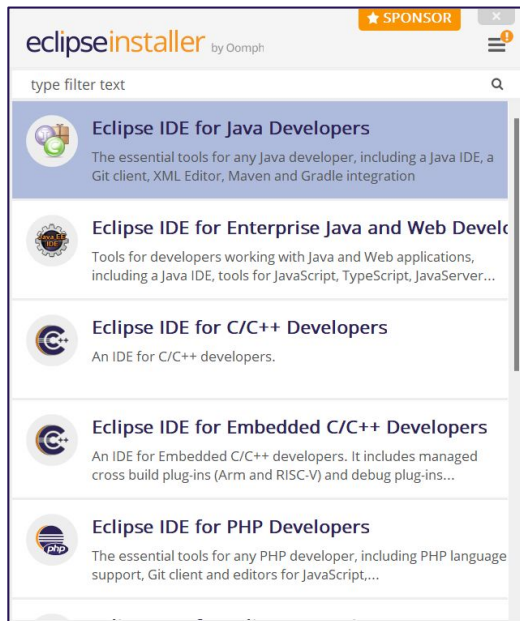
1. Download the Eclipse Installer

Download Eclipse Installer from <http://www.eclipse.org/downloads>

# Eclipse - Installation

## 1. Premier programme

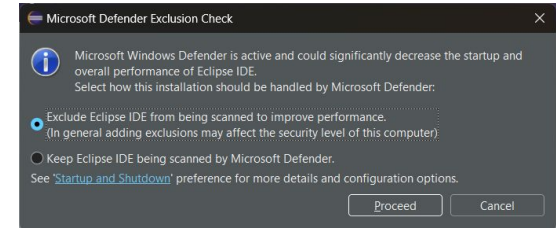
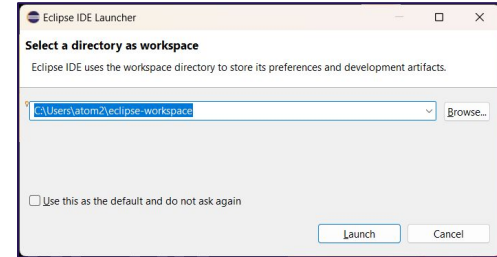
- Choisir cette option:



# Eclipse - Lancement et *workspace*

## 1. Premier programme

- Une fois complété, vous pouvez maintenant lancer Eclipse.
- Au lancement, on vous demande de sélectionner un répertoire comme *workspace*.
  - Le workspace est l'endroit où vous trouverez tous vos programmes. C'est possible de le changer plus tard, mais je vous suggère de garder le répertoire par défaut.
- Acceptez également, les éléments de sécurité:
  - **Exclude Eclipse IDE from being scanned...**
- Fermez ensuite l'écran d'accueil.

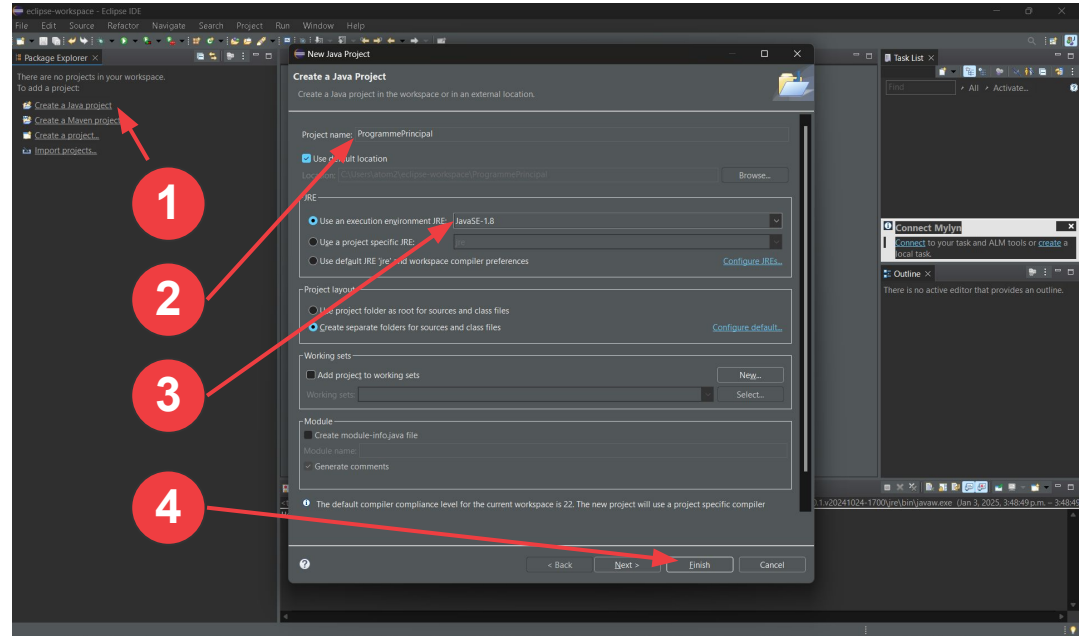


# Eclipse - Nouveau projet

## 1. Premier programme

Voici la séquence à suivre:

1. Create a new java project
2. Donner un nom au projet  
PremierProgramme
3. Choisir: JavaSE 1.8
4. Finish

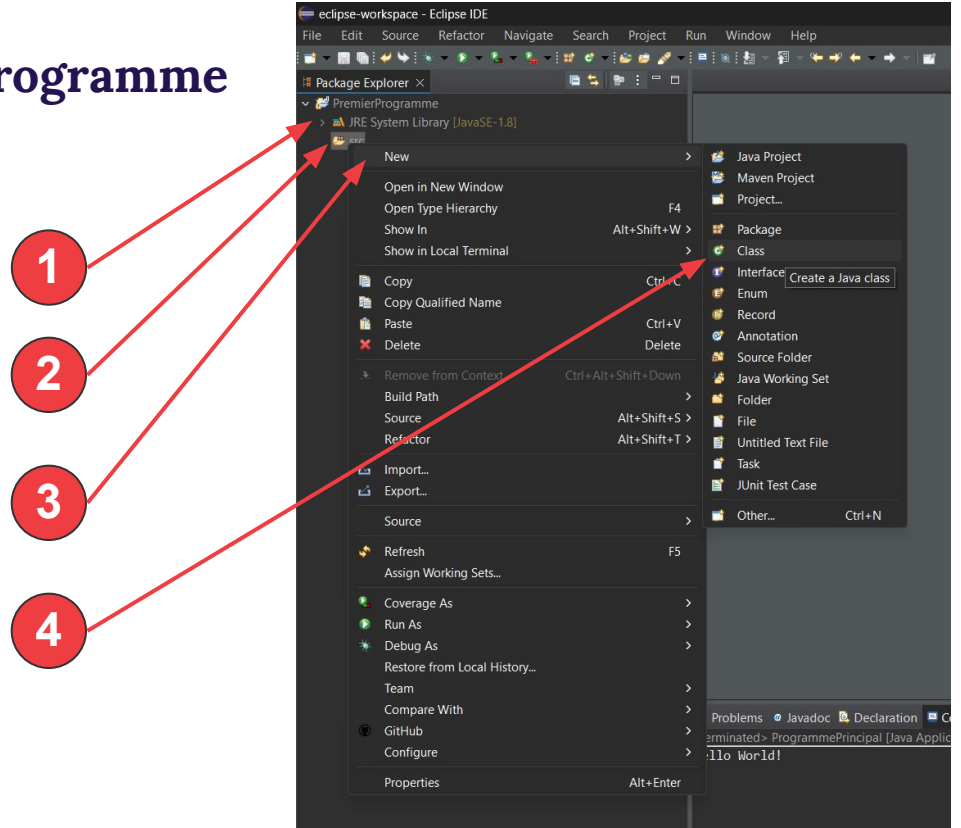


# Eclipse - Première classe

## 1. Premier programme

### Séquence à suivre:

1. Développer le dossier PremierProgramme
2. Bouton-droit sur src/
3. New->
4. Class

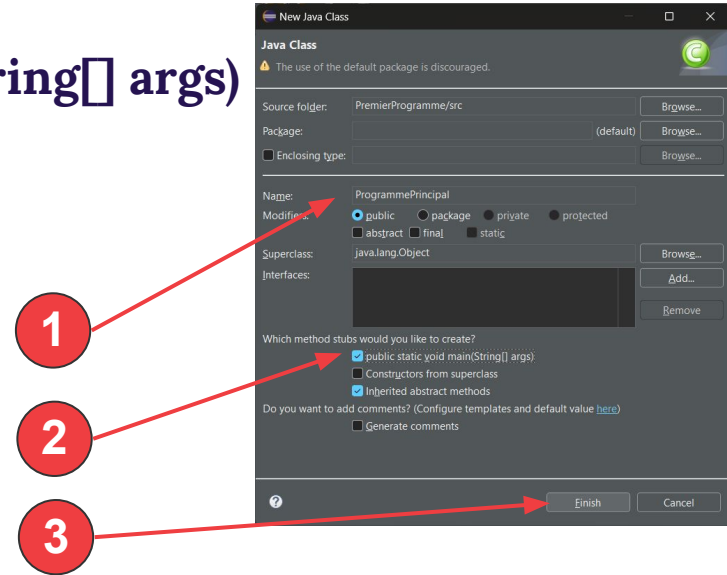


# Eclipse - Première classe

## 1. Premier programme

### Séquence à suivre:

1. Name: ProgrammePrincipal
2. Cocher ☒ public static void main(String[] args)
3. Finish



## Eclipse - Premier programme

### 1. Premier programme

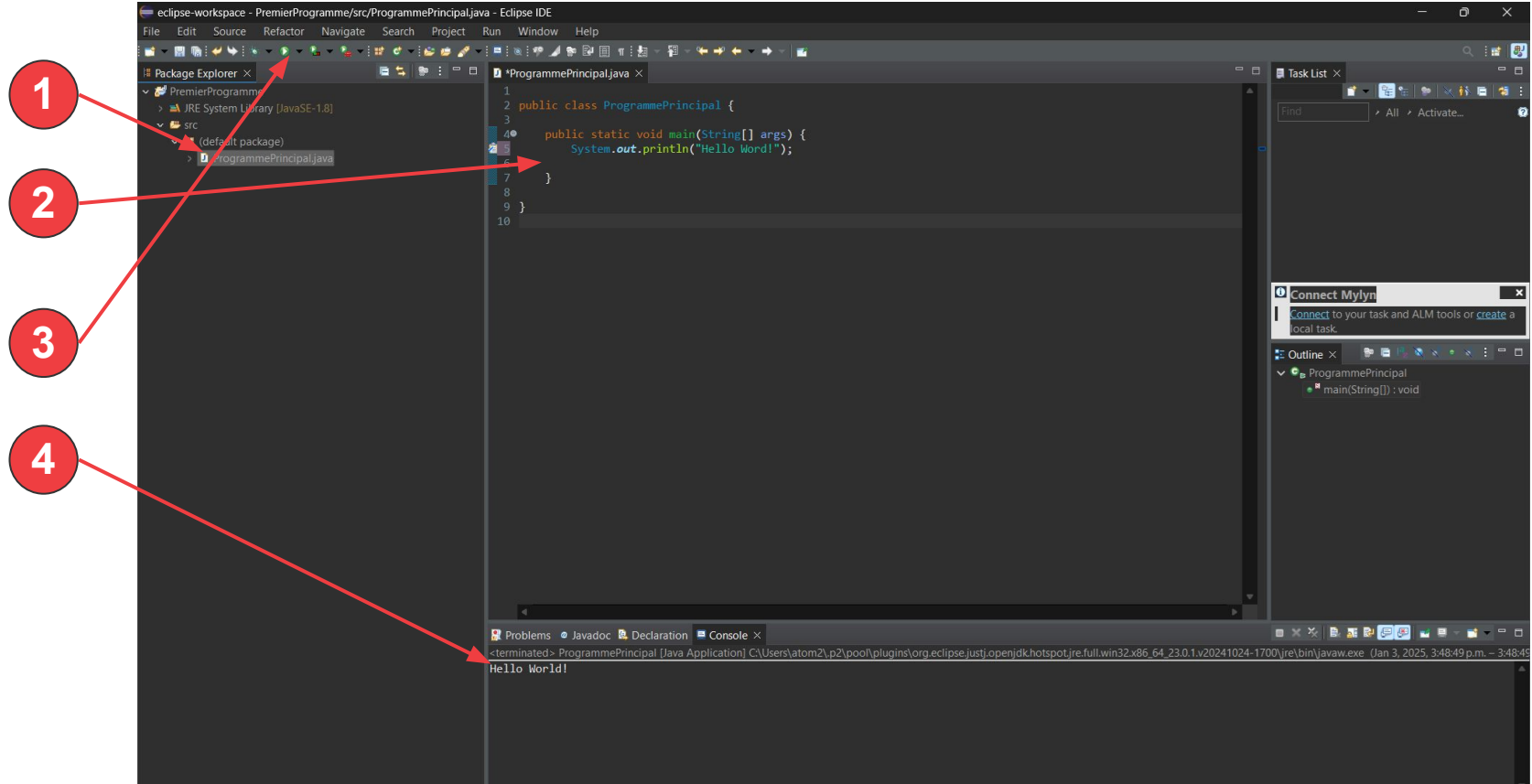
1. Ouvrir le fichier créé, si pas déjà fait.
2. Modifier le code pour avoir:

```
1
2 public class ProgrammePrincipal {
3
4     public static void main(String[] args) {
5         System.out.println("Hello Word!");
6     }
7 }
8
9 }
```

3. Puis lancer en cliquant sur la flèche verte
4. Vous devriez voir, dans la console, Hello World!

# Eclipse - Premier programme

## 1. Premier programme

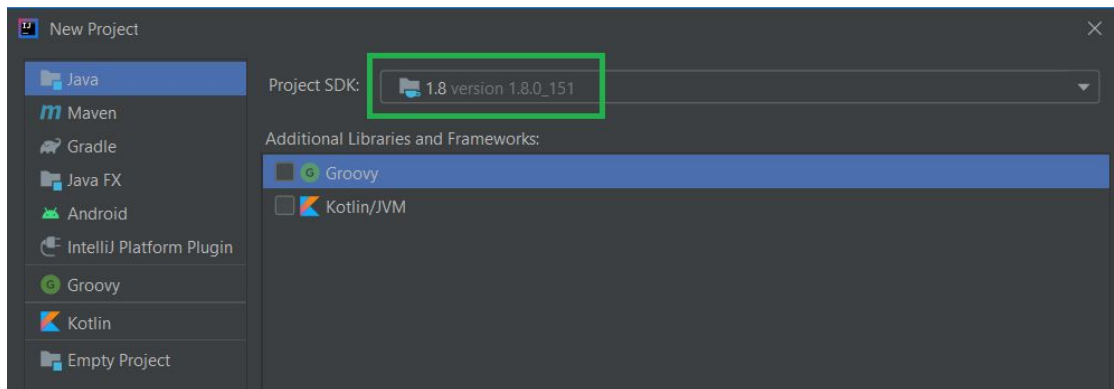




# IntelliJ - Nouveau projet

## 1. Premier programme

- Lancer IntelliJ
- File->New Project->Java
  - Important: choisir Java 1.8

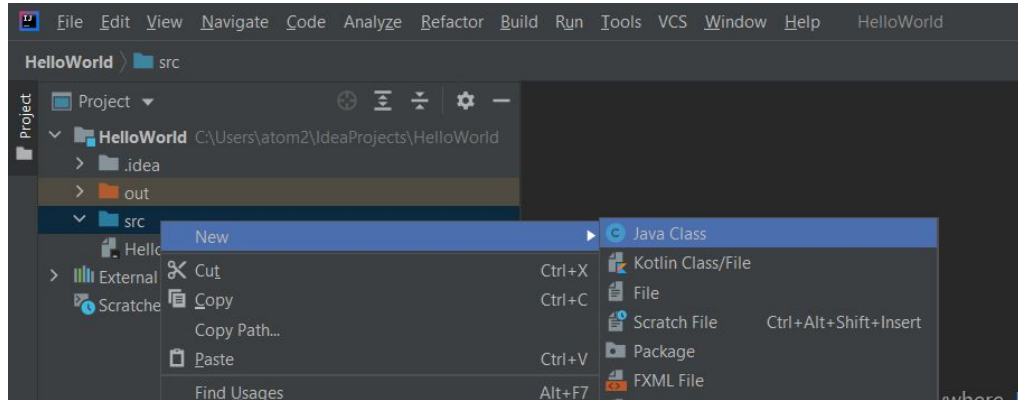


- Next, Next
- Project name: HelloWorld

# IntelliJ - Nouvelle Classe

## 1. Premier programme

- Bouton-droit sur src -> New -> Java Class
- Name: ProgrammePrincipal



## IntelliJ - Premier programme

## 1. Premier programme

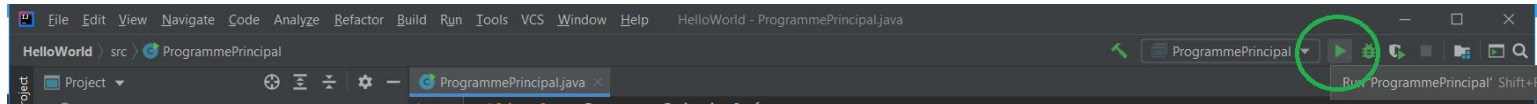
Écrire ce code:

```
public class ProgrammePrincipal {  
  
    public static void main(String[] args) {  
        System.out.println("Bonjour le monde");  
    }  
  
}
```

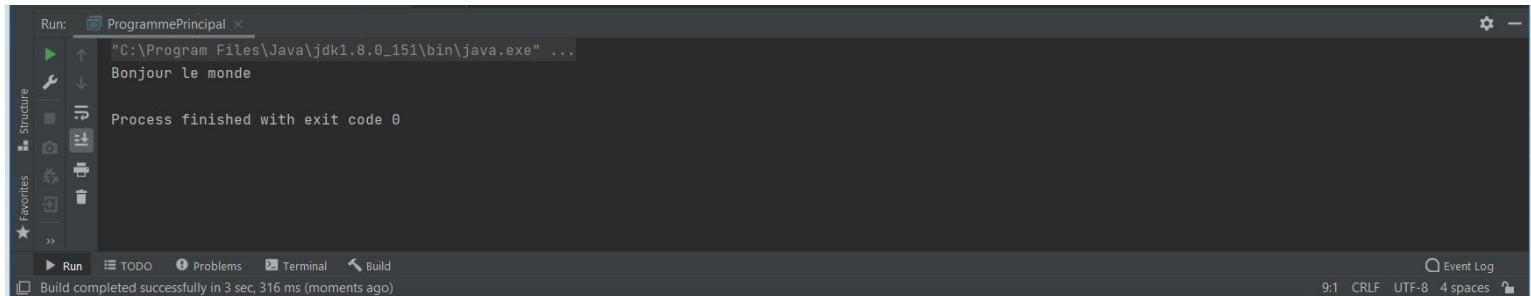
# IntelliJ - Exécution

## 1. Premier programme

Lancer le programme en cliquant sur la flèche verte ou par le menu Run->Run...



Regardez dans la fenêtre du bas (peut prendre quelques secondes). Le message “Bonjour le monde” s’affiche.



L'environnement de développement intégré (**IDE**) est un logiciel qui aide au développement de programmes en connectant des outils et en automatisant les opérations nécessaires à la compilation et exécution du code.

En plus, on retrouve généralement:

- Outils d'analyse et de complétion du code
- Outils d'IA (récent)
- Débugueur
- Intégration avec git

Sachez qu'il est possible d'écrire et exécuter des programmes en dehors d'un IDE, mais c'est moins pratique, dans un contexte de développement.

## Questions de Révision

### 1. Premier programme

- ❑ À quoi sert le *workspace* ?
- ❑ Avez-vous remarqué le nom de la fonction que nous avons défini ?
- ❑ Avez-vous remarqué le nom de la fonction utilisée par notre programme ? Que fait-elle ?
- ❑ Qu'est-ce qu'un IDE intègre ?

## Réponses

### 1. Premier programme

- ❑ Rassembler tous les projets
- ❑ `main()`
- ❑ `System.out.println()` elle sert à afficher.
- ❑ En plus, on retrouve généralement:

Outils d'analyse et de complétion du code

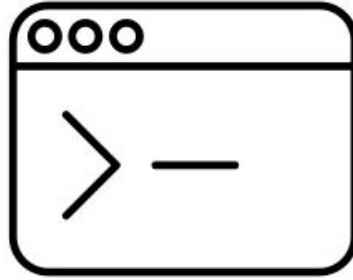
Intégration de l'IA (plus récent)

Débugueur

Intégration avec git

- ❑ Intelligence artificielle, no-/low-code
- ❑ Configuration et maintien d'IDE à l'échelle d'une organisation
- ❑ Publication d'un programme





## **2. Interface utilisateur console**

Les interfaces utilisateur permettent au programme d'échanger de l'information avec l'utilisateur. Deux exemples d'interfaces:

- Interface console

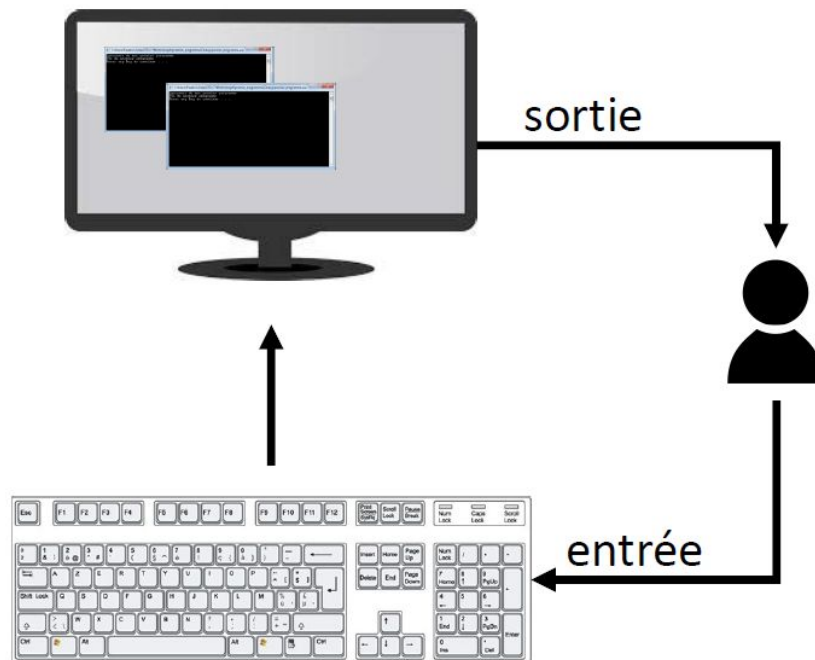
Échange par la **fenêtre de commande**, communication textuelle. Les systèmes simple où pour lesquelles l'interaction humaine est occasionnelle utilise cette interface.

- Interface graphique

**Interaction par l'entremise d'élément visuel.** L'informatique moderne utilise presque toujours l'interface graphique. Nous verrons les interfaces graphiques à la fin de la session (Java Swing).

## Interface Utilisateur (Console)

## 2. Interface utilisateur console



**Les interfaces utilisateurs forment une boucle avec l'utilisateur**

## Sortie: println/print/printf

## 2. Interface utilisateur console

Reprenons cette ligne de notre premier programme:

```
1
2 public class ProgrammePrincipal {
3
4     public static void main(String[] args) {
5         System.out.println("Hello Word!");
6
7     }
8
9 }
```

## Sortie: println/print/printf

## 2. Interface utilisateur console

```
System.out.println(String param)
```

```
System.out.print(String param)
```

- ❑ Le *println* permet d'afficher une String, suivi d'un saut de ligne
- ❑ Le *print* permet d'afficher une String, **sans** saut de ligne
- ❑ Une chaîne littérale, se présente comme ceci: "Hello Word!", mais on peut ajouter des chiffres et autres valeurs provenant de variables, par concatenation (+).

Fichier d'exemples: ExPrint.java

## Sortie: println/print/printf

## 2. Interface utilisateur console

`System.out.printf(String format, ...)`

- ❑ En java, println/print, sont très pratique, mais manque de flexibilité quand vient le temps d'afficher des doubles.
- ❑ `printf` offre plus d'options de formatage et s'utilise comme en C, mais est un peu plus complexe, car il faut mémoriser les codes.

Fichier d'exemples: ExPrint.java

Référence pour les options de formatage:

<https://alvinalexander.com/programming/printf-format-cheat-sheet/>

L'entrée clavier se fait par l'entremise d'un “objet” (concept présenté à la section suivante).

Le **Scanner** a la responsabilité d'interpréter les données entrées (System.in) par le clavier pour en extraire l'information demandée.

L'utilisation du Scanner:

1. import de la librairie
2. Instanciation de l'objet, on donne le nom **clavier** à l'instance
3. Utilisation

Voir: ExScanner.java

Référence: <https://docs.oracle.com/javase/8/docs/api/java/util/Scanner.html>

### Utilisation du Scanner:

- `clavier.nextInt()`, capture un entier
- `clavier.nextDouble()`, capture un double

Et pour la capture d'une chaîne de caractère:

- `clavier.nextLine()`, capture la chaîne terminée par un saut de ligne '\n'

**Prenez note que les appels à `nextX()` sont des appels bloquants.** L'exécution du programme est suspendu en attendant une entrée valide.

Remarquez également que `clavier` est `static`, ce qui veut dire que l'instance existe pendant toute la durée du programme.

Référence: <https://docs.oracle.com/javase/8/docs/api/java/util/Scanner.html>



## Questions de Révision

## 2. Interface utilisateur console

- ❑ Avez-vous remarqué le `System.out` et le `System.in`, est-ce que ça fait du sens dans le contexte ?
- ❑ Quels sont les étapes d'utilisation d'un Scanner ?
- ❑ Qu'est-ce que ça veut dire que le clavier soit static ?
- ❑ Qu'est-ce qui se passe avec l'exécution du programme, lors d'un appel à `clavier.nextInt()`

- ❑ **System.out** est le flux de données sortant vers le système et correspond à la console.

**System.in** est le flux de données entrant du système et correspond également à la console.

- ❑ **Les étapes:**
  - **importation de la librairie**
  - **instanciation static**
  - **utilisation**

## Réponses

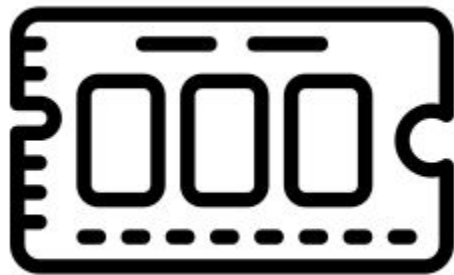
## 2. Interface utilisateur console

- ❑ L'instance du Scanner est valide pendant toute la durée du programme.
- ❑ Il s'agit d'un appel bloquant, l'exécution du programme est suspendue.

- ❑ Écrire un exemple qui demande son nom, prénom et l'âge à un utilisateur, puis affiche:
  - Bonjour <prénom> <nom de famille>!
  - Vous avez <age>
- ❑ Écrire un exemple qui affiche la valeur de pi, avec 3, 6 et 10

Prenez note de l'utilisation de la librairie math

- ❑ Est-ce que `printf` peut-être utilisé comme un outil de débogage ?
- ❑ Comment pourrait-on contourner l'aspect bloquant du `Scanner` ?
- ❑ Est-ce que l'on peut faire confiance à l'utilisateur pour entrer des valeurs valides à tous les coups ?
- ❑ Performance vs. `print`
- ❑ Pouvez-vous nommer d'autres types d'interfaces utilisateurs ?



### **3. Programmation orienté-objet**

## 2 types de programmation

### 3. Programmation OO

- ❑ En C, on parle de programmation procédurale.
- ❑ En Java, on parle de programmation orienté-objet (OO)

La programmation OO est le style de programmation le plus répandue et dicte l'architecture d'un logiciel, mais on retrouve quand même des éléments de programmation procédurale, même en Java.

On aborde cette distinction directement tout à l'heure.

**La classe** est le point de départ de la programmation OO.

Une classe:

- Possède de la mémoire (**attributs**/variables membres)
- Offre des services (**méthodes** publiques)

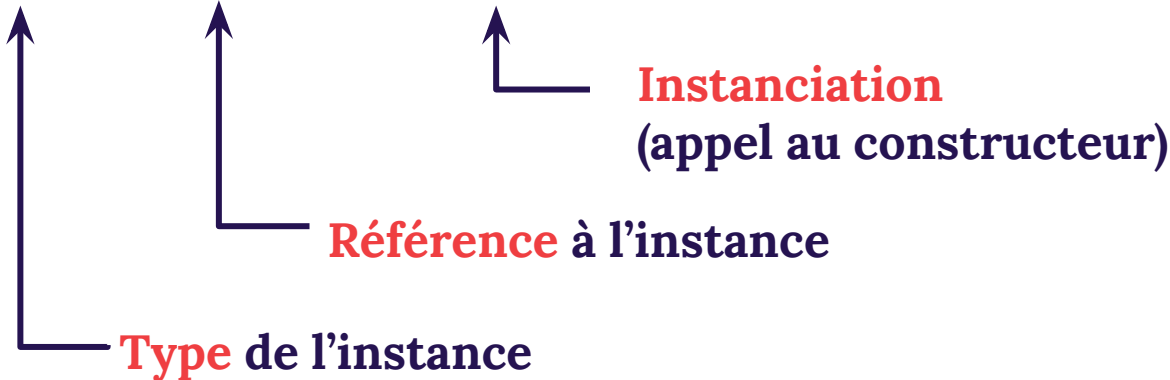
Voir: Voiture.java



Une définition de classe, c'est du code en l'air, on ne peut pas l'utiliser directement, **il faut créer une instance de la classe, pour l'utiliser.**

Une *instance*, c'est l'objet dans l'orienté-objet.

```
Object objet = new Object()
```



### Survol de l'exemples ExInstance.java

- Utilisation de la voiture (instance **locale**)
- Utilisation du Scanner (instance **static**)
- Utilisation du Random (instance **static**)

**Note:** Nous n'expliquons pas tout de suite la distinction entre instance locale et static, mais sachez que cette distinction est importante. Dans le contexte du cours, Scanner et Random sont toujours static.

Comment on réfléchit la classe ?

En tant que programmeur, on voit la classe comme un objet logiciel. **Elle possède la mission de maintenir de la mémoire et de gérer les opérations sur celle-ci.**

L'organisation du code en classe et la conceptualisation de l'orienté-objet a permis de grandement **simplifier la gestion de la sécurité** (accès à la mémoire) et **favoriser la réutilisation du code, la rapidité de développement et l'intégration modulaire.**

Allons-y concept par concept.

L'encapsulation est le terme employé pour indiquer que **la mémoire est protégé des accès externes et sa gestion est encadré**. La mémoire est donc, mise en capsule.

NOTE: En C, le même concept est mis en pratique avec la représentation cachée.

Retournons à notre voiture. Sommes-nous capable de changer la valeur des variables membres, à partir de l'instance ?

```
voiture.modele = "RAV4"
```

Ce n'est pas possible, car les variables membres sont **private**. Leur accès direct, de l'extérieur de la définition est donc interdit.

Comment travailles-t-on avec une Classe, dans ce cas ?

Tout d'abord, le **constructeur** est utilisé pour initialiser la classe. Nous verrons 3 types de constructeurs, mais pour l'instant, ce constructeur par paramètres assigne les valeurs aux membres.

Ensuite, on va définir des méthodes d'accès:

- Accesseur (get)
- Mutateur (set)

Permettant d'accéder aux champs en lecture et en écriture. Dans notre exemple simpliste, ce n'est pas évident, mais **l'important est qu'on contrôle comment les membres sont accédés**, c'est ça l'aspect sécurité.

Prenons un petit moment pour amener 2 précisions:

- On peut également définir des membres **public**. À ce moment, on contourne les méthodes d'accès et c'est généralement considéré comme une mauvaise pratique (incorrect pour les variables, mais accepté pour les constantes).
- Le mot clé **this**, utilisé dans le constructeur **est une référence à l'instance courante**. C'est un élément souvent facultatif. On l'utilise pour désambigüiser les variables:
  - “marque” est le paramètre d'entrée
  - “this.marque” est la variable membre

En résumé, une classe:

- **Utilise l'encapsulation pour contrôler les accès à la mémoire**, une manière de réduire les bugs en empêchant les programmeurs de faire des opérations mémoires invalides. (sécurité)
- **Une classe doit être instancié**, à l'aide du constructeur, pour pouvoir être utilisé.
- **L'instance est accédé par la référence.**
- **Pour permettre l'accès à la mémoire, on définit des méthodes d'accès (getters/setters)**, dans lesquels on peut valider les accès et s'assurer que la classe est utilisé correctement.



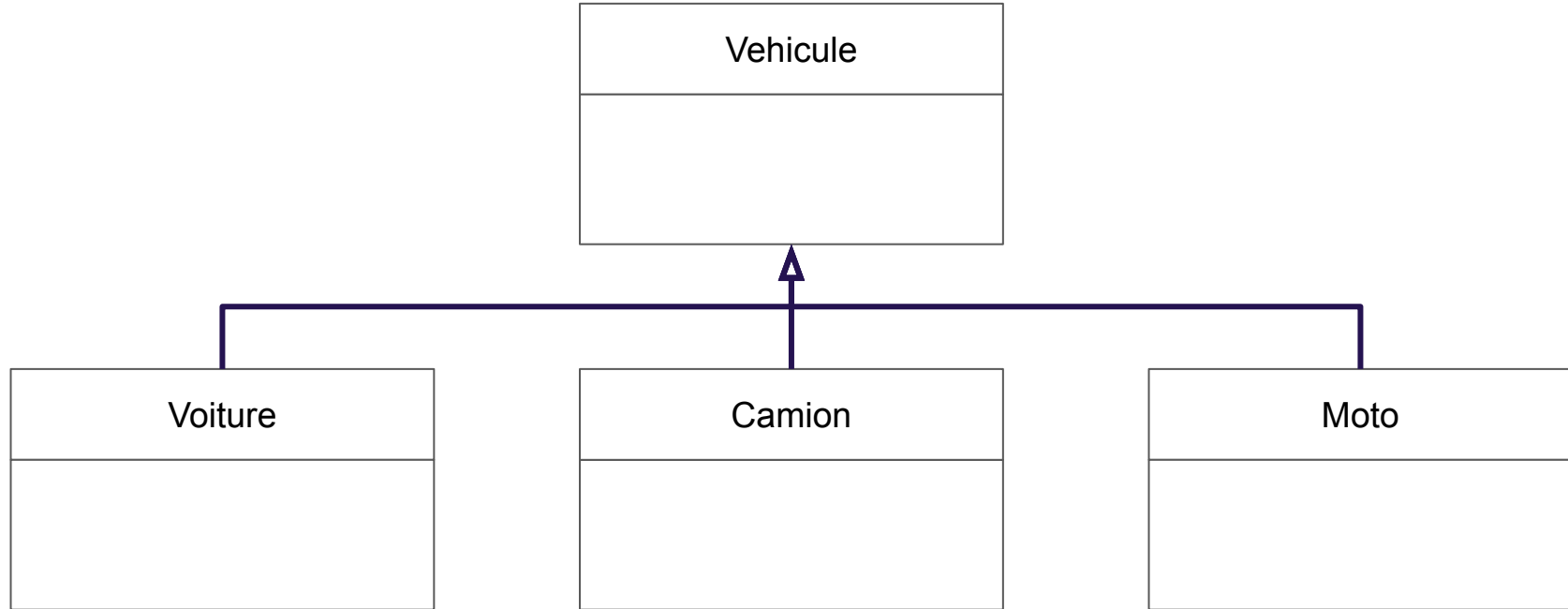
**L'héritage** est un concept un peu plus avancé et on est vraiment dans la conception orienté-objet.

Le concept est quand même assez intuitif. Deux questions pour vous:

- Pouvez-vous identifier la catégorie d'objet à laquelle appartient une voiture ?
- Pouvez-vous identifier d'autres objets appartenant à la même catégorie ?

## Concepts OO - Héritage par extension

### 3. Programmation OO

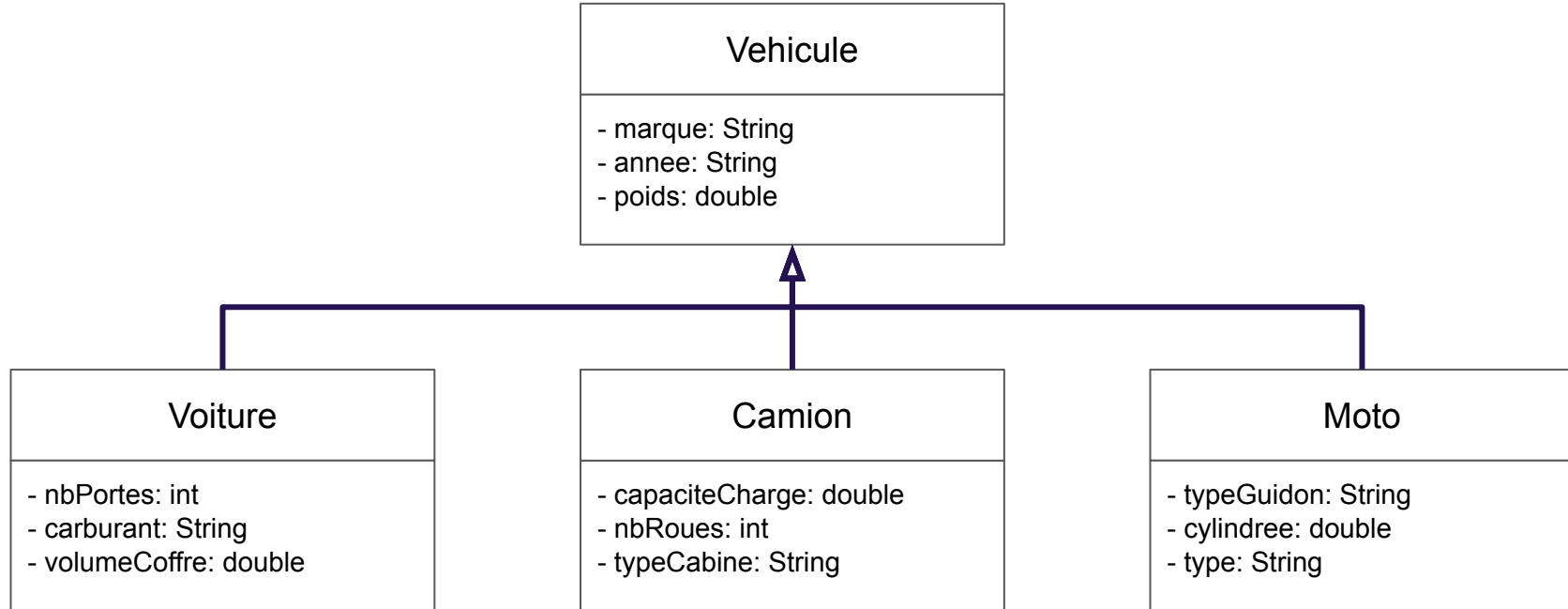


Le Vehicule est la **classe de base**. Il s'agit d'une **généralisation**.

Voiture, Camion et Moto sont des **classes dérivés**, il s'agit d'une **spécialisation**.

**L'héritage permet de créer un lien logique entre ces classes.** Voyons voir comment ça marche.

- 1) Identifions des attributs **généraux**, qui s'appliquent à tous les véhicules.
- 2) Identifions des attributs **spécifiques** à chaque classe dérivée.



**Note:** Cette représentation s'appelle un **diagramme UML**.

Voir dans l'exemple: ExVehicule

- **extends** est le mot clef qui permet d'indiquer qu'une classe dérive d'une autre.

```
public class Voiture extends Vehicule
```

- L'exemple démontre ensuite le véritable objectif de l'héritage. **Les attributs de la classe de bases, tout comme les méthodes publiques, sont également membre de la classe dérivé.**
- On remarque un nouveau mot clé: **protected**, voir prochaine slide
- Et l'appel à **super()**, qui permet d'exécuter le constructeur de la classe de base.

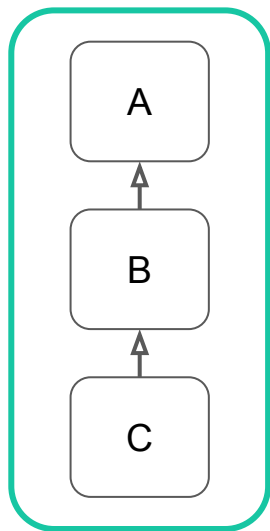
## Concepts OO - Modificateur de visibilité

### 3. Programmation OO

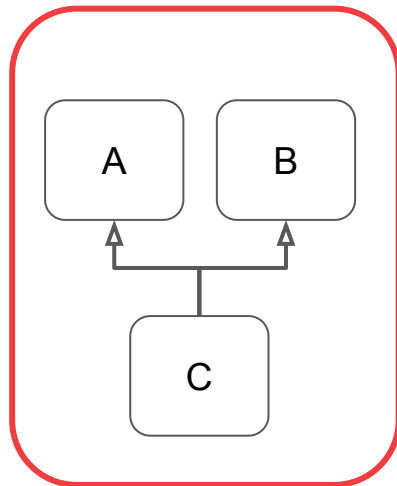
Modifier	Class	Package	Subclasses	World
public	✓	✓	✓	✓
protected	✓	✓	✓	✗
no modifier	✓	✓	✗	✗
private	✓	✗	✗	✗

L'héritage peut fonctionner à plusieurs niveaux. Une classe peut dériver d'une autre, qui en dérive d'une autre...

Par contre, le langage Java impose une restriction importante: une classe ne peut hériter, par extension, de plus d'une classe.



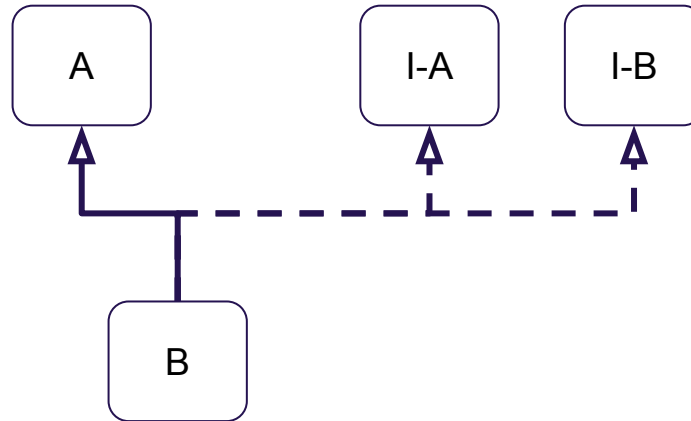
OK



Pas ok

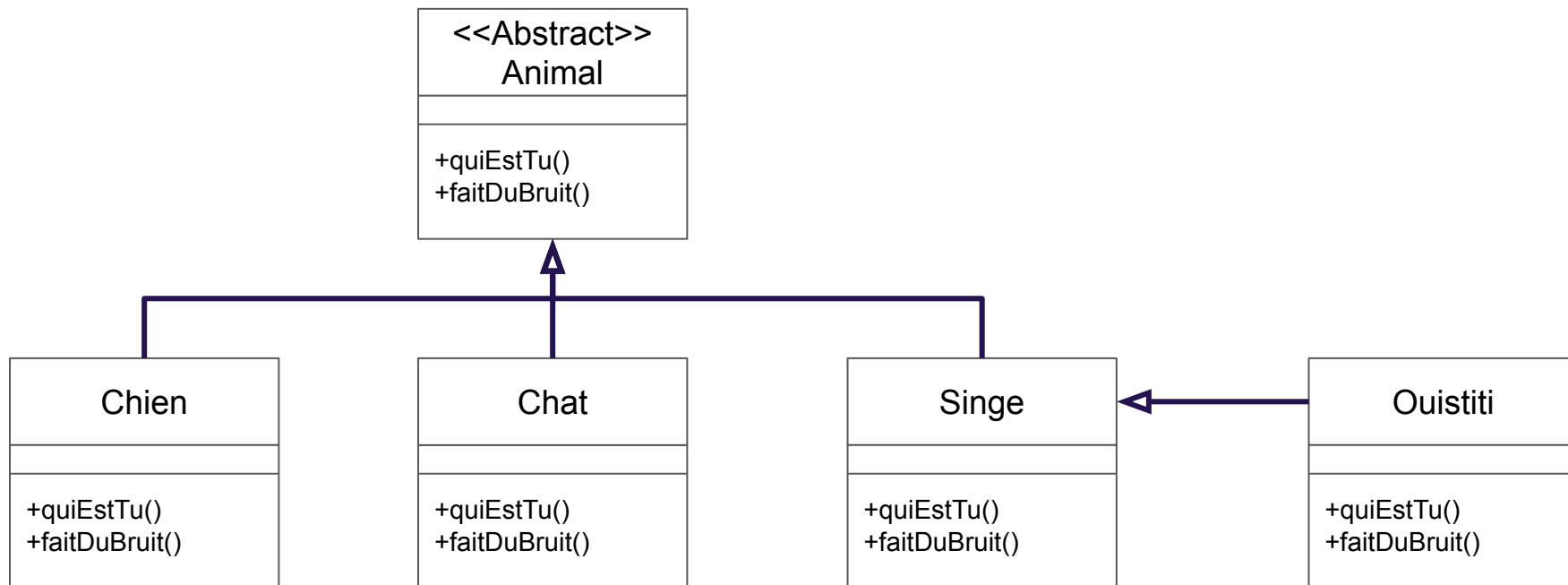


Pour contourner cela, par contre, il existe l'**héritage par implémentation**, qui utilise des interfaces. Et là, une classe peut implémenter un nombre arbitraire d'Interfaces. **Voir moto dans l'exemple.**



Les Interfaces ne sont pas des classes, mais je dois limiter la discussion pour aujourd'hui.

Nouvel exemple, les animaux!



Dans notre exemple, tous les animaux font du bruit, mais est-ce que tous les animaux font le même bruit?

Dans notre exemple, tous les animaux font du bruit, mais est-ce que tous les animaux font le même bruit?

On peut remplacer la définition d'une fonction hérité par une autre définition. On parle alors d'**overriding** ou sur-définition. (voir classe Singe)

Très pratique, quand la classe de base ne fait pas exactement ce dont on n'a besoin.

Prenez note:

- Il est possible d'exécuter la fonction hérité en écrivant **super.nomDeLaFonction()**. (voir classe Ouistiti)
- L'exemple inclut une classe **abstraite**, une classe qui ne peut être instantiée. (voir classe Animal)
- Pas démontré, mais une classe peut-être **final**, pour interdire la définition d'une classe dérivée.

**Polymorphisme veut dire plusieurs(poly)-formes(morph).**

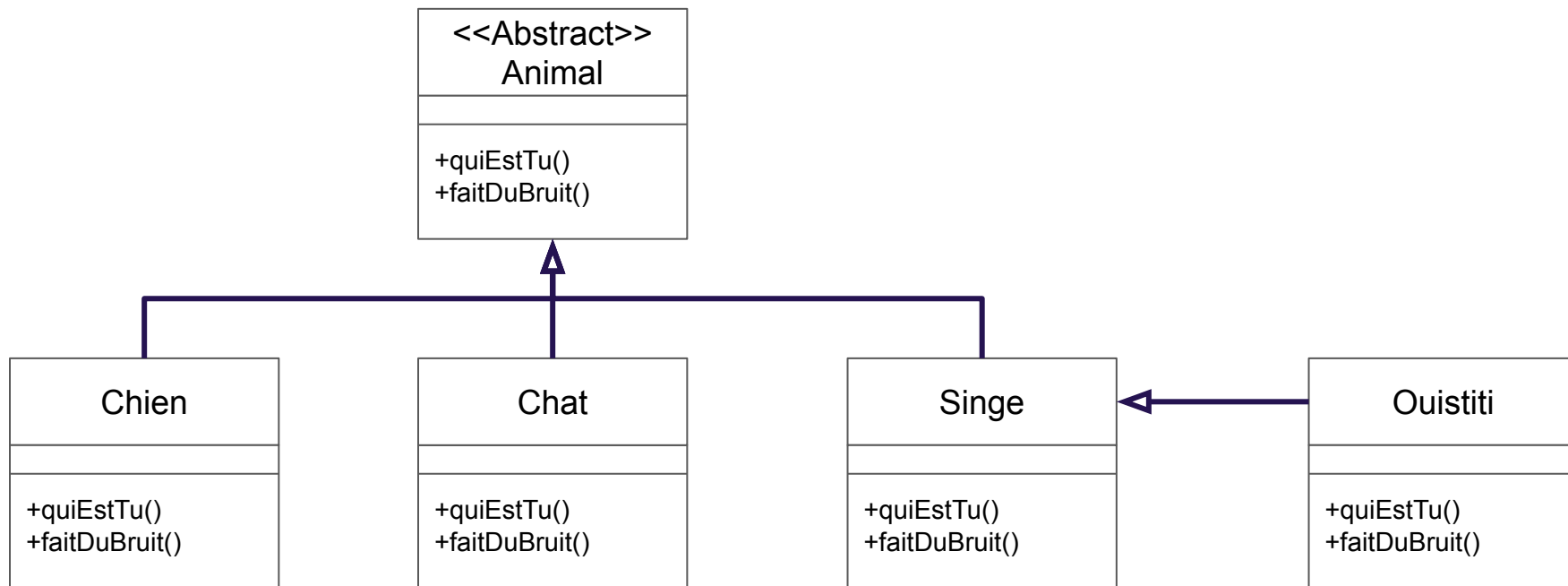
**C'est un concept qui découle de celui de l'héritage et une des forces d'un langage orienté-objet.**

**De ce qu'on connaît, le type d'une référence doit correspondre à la classe qui est instancié:**

```
Singe monSinge = new Singe() ;
```

Mais, selon notre diagramme, est-ce qu'un Singe **est un** Animal?

Nouvel exemple, les animaux!





Le polymorphisme permet de faire un truc comme celui-ci:

```
Animal monAnimal = new Singe();
```

Très pratique, car ça permet de rendre notre code très flexible, par exemple, faire un tableau d'Animal qui contient à la fois des Singe, des Chat, des Chiens, etc...

On peut toujours redétecter un type, par la suite en utilisant:

```
monAnimal instanceof Singe
```

qui retourne true/false.

La version courte, **les méthodes déclarés comme static, sont des “fonctions” comme celle que l’on retrouvait en C, donc une logique procédurale.**

Les classes qui ne comportent que des fonctions static (java.lang.math) sont des classes qui n’ont pas à être instancié.

Le main est également static.

Les fonctions statiques ont leur place dans le langage Java, et peuvent se mêler avec l’OO, sans problème. **Attention, les méthodes static ne peuvent utiliser les variables membres, on reviendra sur cela plus tard.**

#### ❑ Expliquer simplement les concepts et mots-clés suivants:

- méthode
- encapsulation
- private/public/protected
- attributs
- override ou sur-définition
- Héritage par extension
- Constructeur
- Interface
- Polymorphisme
- méthodes d'accès get/set
- objet
- classe de base
- classe dérivée
- instanceof
- static

## Réponses

- ❑ Valider vos réponses avec les notes de cours ou avec ChatGPT.

- ❑ Classe Object
- ❑ OO dans les autres langages

## Exemple Intégrateur Point 2D/3D

### 3. Programmation OO

- Classe de base Point2D
- Classe dérivée Point3D
- Factory