Programmation Orientée Objet (POO) - JAVA

Prof. Asmaa EL HANNANI
ENSA El Jadida
2024/2025

Informations pratiques

Informations pratiques

- **Cours / TP**: Salles (B1/B2), site ENSAJ
  - Prof. Asmaa EL HANNANI
  - elhannani.a@ucd.ac.ma

- Matériel du cours et exercices sous:
  TEAMS et/ou Moodle (http://moodle.ensaj.ucd.ac.ma)

Règles d'or

- **Ponctualité: Arriver à l'heure !**
- **Respecter le droit des autres à écouter : pas de bavardage en classe !**
- **Respecter vos collègues !**

- **Prendre des notes pendant la séance du cours**
- **Poser des questions**
- **Faire les devoirs et les rendre dans les délais**

Organisation

- **~1H50 de cours**
  - Cours / TD
- **~1H50 de TP par groupe (1/2)**
  - Réponse aux questions, explication du TP courant, travail sur PC

Evaluation

- **Un contrôle** (**40%** de la note finale du module)
- **Un examen final** (**60%** de la note finale du module)

Objectifs du cours

- Apprendre à penser, concevoir et programmer objet !
  - Comprendre et exploiter l'utilité de la visibilité des attributs et des méthodes (encapsulation).
  - Comprendre les relations d'héritage, d'agrégation et de composition.
  - Comprendre et pouvoir exploiter le polymorphisme.
- Utiliser le langage Java pour mettre en œuvre le paradigme Orienté Objet.

Pages Java officielles

- Java Home Page : http://www.oracle.com/technetwork/java
- API: http://docs.oracle.com/javase/8/docs/api
- JDKs:

  http://www.oracle.com/technetwork/java/javase/downloads
- Forums:

  https://community.oracle.com/community/java

Part 1

INTRODUCTION

Notion d'objet

Notion d'objet

- C'est un concept servant à représenter, modéliser toute **entité** selon ses propriétés et son comportement.
- Ce comportement s'observe via les interactions que le monde extérieur peut avoir avec cette entité à travers les opérations de l'objet.

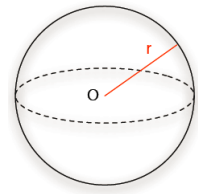
Notion de classe

- On regroupe un ensemble de caractéristiques communes aux objets sous la forme de structures formelles, appelées classes.
- C'est un concept orienté objet destiné à modéliser formellement des objets d'un certain type d'entité.
- La classe décrit les propriétés et les opérations des objets du type d'entité qu'elle représente.

Exemples d'objets







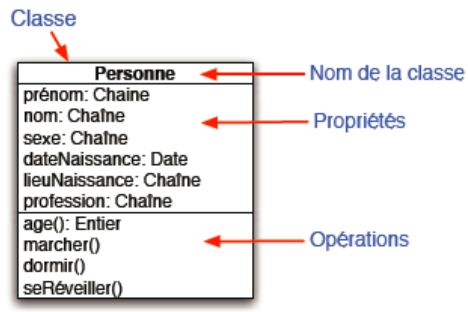
Exemple 1: Personne

- Une personne est caractérisée par son prénom, son nom de famille, sa date de naissance, son lieu de naissance, sa profession: ce sont ses **propriétés**.
- Une personne peut exécuter les **opérations** suivantes: se réveiller, se lever, marcher, courir, sauter, s'asseoir, dormir, manger, lire, écrire, parler, se taire,. . .

Modèle textuel des objets de type « Personne »

- **Classe** : Déclaration de type
  - nom = Personne
- **Propriétés** : Déclaration de propriétés
  - nom = prénom ; type de donnée = Chaîne
  - nom = nom de famille ; type de donnée = Chaîne
  - nom = sexe ; type de donnée = Chaîne
  - nom = date de naissance ; type de donnée = Date
  - nom = lieu de naissance ; type de donnée = Chaîne
  - nom = profession ; type de donnée = Chaîne
- **Opérations** : Comportement des objets de type Personne
  - nom = âge ; type de retour = Entier
  - nom = marcher ; type de retour = Rien
  - nom = dormir ; type de retour = Rien
  - nom = se réveiller ; type de retour = Rien

Modèle graphique des objets de type « Personne»



- Notation graphique utilisée : **Diagramme de classe UML**
- Trois compartiments : nom de la classe, propriétés et opérations.

Exemple 2: Point du plan cartésien

- Un point est caractérisé par son abscisse (x) et son ordonnée (y) : ce sont ses **propriétés**.

- On peut exécuter les **opérations** suivantes: sur un point : est-il à l'origine des axes? translation vers un autre point selon un delta (dx, dy), distance d'un autre point….

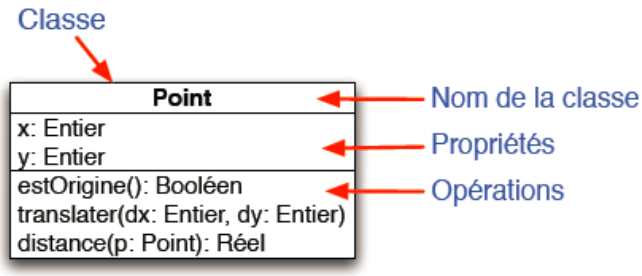
Modèle textuel des objets de type « Point»

- **Classe** : Déclaration de type
  - nom = Point

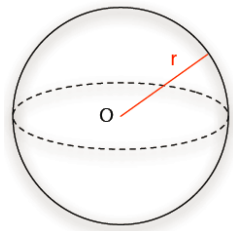
- **Propriétés** : Déclaration de propriétés
  - nom = x ; type de donnée = Entier
  - nom = y ; type de donnée = Entier

- **Opérations** : Comportement des objets de type Point
  - nom = estOrigine ; type de retour = Booléen
  - nom = translater(dx : Entier, dy : Entier) ; type de retour = Rien
  - nom = distance(p : Point) ; type de retour = Réel

Modèle graphique des objets de type « Point»



Exemple 3: Une sphère



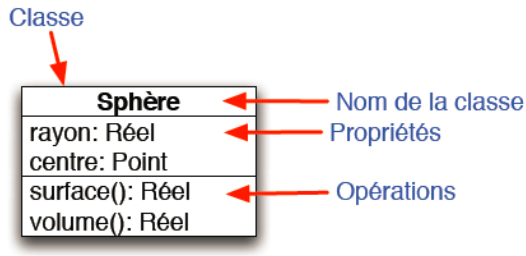
- Possède deux **propriétés** : son rayon r et son centre o, qui est un objet de **type Point**.
- Connaissant r, on peut calculer sa surface ($S = 4*Pi*R^2$)
- Connaissant r, on peut calculer son volume ($V = 4/3*Pi*R^3$)

Modèle textuel des objets de type « Sphère »

- **Classe** : Déclaration de type
  - nom = Sphère
- **Propriétés** : Déclaration de propriétés
  - nom = rayon ; type de donnée = Réel; valeur par défaut = 0
  - nom = centre; type de donnée = Point; valeur par défaut = Point(0,0)
- **Opérations** : Comportement des objets de type Sphère
  - nom = surface ; type de retour = Réel
  - nom = volume; type de retour = Réel

surface et volume ne possèdent pas de paramètres formels (sous l'hypothèse que r est connu au sein des objets de type « Sphère »)

Modèle graphique des objets de type « Sphère»



Instance

- On appelle **instance** d'une classe, un objet avec un état.
- Il s'agit donc d'un objet constituant un exemplaire de la classe.

- On dit aussi: un objet est une **instance** d'une classe
- Objet = Instance

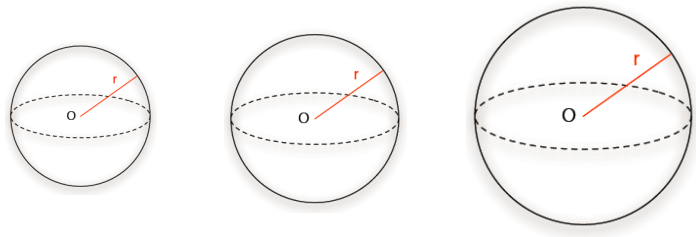
- **L'instanciation** est l'action d'instancier, de créer un objet à partir d'un modèle (classe).

Instance de la classe Sphère

- Objet : Déclaration d'une instance
  - type = Sphere
- Propriétés : Initialisation des propriétés
  - rayon = 3,5 ; (Donnée à la création de cet objet)
  - centre = Point(2,6) ; (Positionné à la création de cet objet)
- A l'invocation des opérations : (on suppose que r n'a pas changé)
  - surface = 153,938
  - volume = 179,594

Cet objet est une instance de la classe Sphère. On peut en avoir d'autres.

Apparence de « différents objets » de type Sphère



Ce sont des instances de la classe Sphère.

Apparence de « différents objets » de type Personne

Ce sont des instances de la classe Personne.

Exercice

- Décrivez un type d'objet de votre choix en donnant ses propriétés et quelques unes de ses opérations.
- Donnez deux instances de cette classe.

Programmation orientée objet (POO)

L'idée de base de la programmation orientée objet est de rassembler dans une même entité appelée **objet** les données (propriétés) et les traitements (opérations) qui s'y appliquent.

Les données d'un objet sont appelées ses **attributs** et ses traitements sont ses **méthodes**.

Caractéristiques essentielles de la POO

- **Tout est objet** ! : chaque objet **encapsule** des attributs et des méthodes agissant sur ces données.
- Le concept de classe généralise la notion de type. Entre classe et objet il y a, en quelque sorte, le même rapport qu'entre **type** et **variable**.
- Instancier une classe consiste à créer un objet sur son modèle avec ses données propres (cela se fait via le constructeur).
- L'encapsulation réalise une **abstraction** des données : vu de l'extérieur de l'objet, les détails d'implémentation sont cachés.
- Des classes peuvent **hériter** d'autres classes (classe mère -classes filles). La notion d'héritage permet d'établir une **hiérarchie** entre les classes.
- Avec l'héritage, il devient possible de **redéfinir des méthodes** au sein des classes filles. On parle de polymorphisme.

POO vs programmation procédurale

- **Programmation procédurale** : s'intéresse en priorité aux **traitements** que son programme devra effectuer: détermination d'abord des procédures pour manipuler des données prises comme paramètres d'entrée.
- **POO** propose une méthodologie centrée sur les **données**: détermination d'abord des données de façon cohérente dans des entités structurantes (objets), puis réalisation des opérations sur ces données.

POO avec Java

Qu'est-ce que Java ?

- Java est un langage de programmation
  - Orienté objet
  - Compilé et interprété
  - **« write once, run everywhere »** = écrivez une seule fois votre programme, exécutez-le sur toutes plates-formes
- Java est une plateforme
  - La plateforme Java, uniquement software, est exécutée sur le système d'exploitation

Origine du langage JAVA

- Java a été développé à partir de décembre 1990 par une équipe de Sun Microsystems dirigée par James Gosling (projet Oak).
- Au départ, il s'agissait de développer un langage de programmation pour permettre le dialogue entre de futurs appareils électroménager.
- Or, les langages existants tels que C++ utilisé chez Sun, ses interfaces de programmation en langage C, ainsi que les outils associés n'étaient pas satisfaisants.
- Java a été présenté officiellement le 23 mai 1995 au SunWorld.

Historique de java

- Java suit un cycle de publication prévisible :
  - Les versions avec de nouvelles fonctionnalités (**Feature releases**) sont publiées tous les six mois et apportent les dernières avancées en matière de fonctionnalités du langage, de bibliothèques et d'optimisations.
  - Les versions à support long terme (**Long-Term Support-LTS**) sont publiées tous les trois ans et bénéficient d'un support prolongé par Oracle (au moins 8 ans). Ces versions constituent la base des applications d'entreprise.

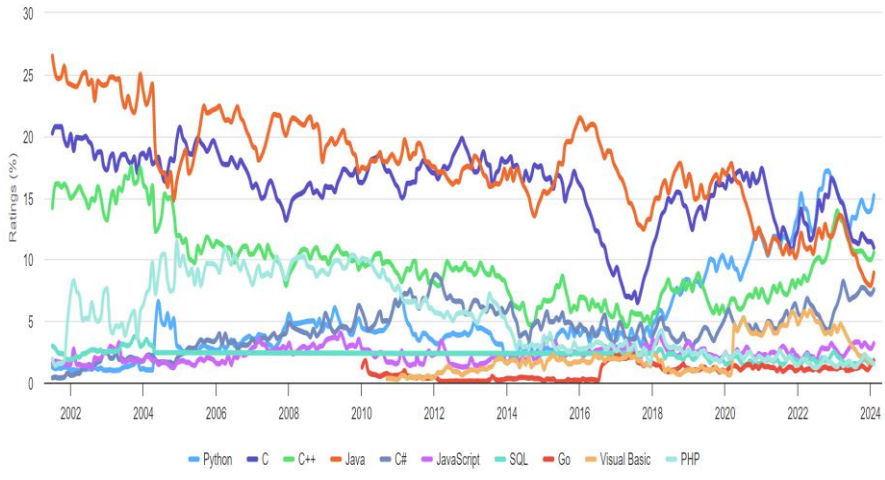
Historique de java

Source: https://en.wikipedia.org/wiki/Java_version_history

Version	Type	Class file format version[1]	Release date	End of public updates (free)	End of extended support (paid)
Java SE 19		63	20th September 2022	March 2023	—
Java SE 20		64	21st March 2023	September 2023	—
Java SE 21	LTS	65	19th September 2023	September 2028 for Oracle[12] September 2028 for Microsoft Build of OpenJDK[13] December 2029 for Red Hat[4] December 2029 for Eclipse Temurin[9] October 2030 for Amazon Corretto[10] September 2031 for Azul[3] December 2029 for IBM Semeru Runtimes[11]	September 2031 for Oracle[12] March 2032 for BellSoft Liberica[6]
Java SE 22		66	19th March 2024	September 2024	—
Java SE 23		67	17th September 2024	March 2025 for Oracle September 2032 for Azul[3] March 2025 for IBM Semeru Runtimes[11]	—
Java SE 24		68	March 2025	September 2025	—
Java SE 25	LTS	69	September 2025	September 2030 for Oracle[12]	September 2033 for Oracle[12] March 2034 for BellSoft Liberica[6]

Legend: Old version, not maintained | Old version, still maintained | **Latest version** | Future release

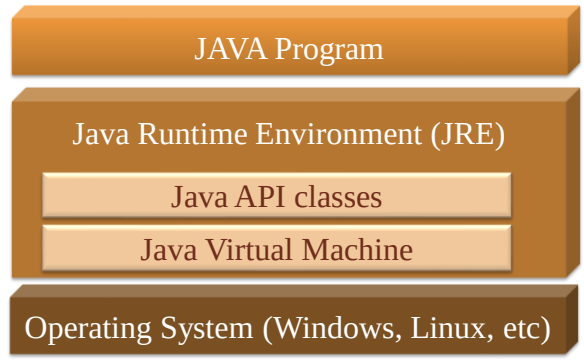
JAVA aujourd'hui



Source: http://www.tiobe.com

Java comme Plateforme

- La Java « Platform » se compose de:
  - la Java Virtual Machine (Java VM)
  - la Java Application Programming Interface (Java API)



Java comme langage de programmation

Java possède un certain nombre de caractéristiques qui ont largement contribué à son énorme succès :

- Java est simple
- Java est orienté objet
- Java est robuste
  - Java est fortement typé
  - Java assure la gestion de la mémoire
- Java est portable
- Java est sûr
- Java est économe
- Java est multitâche

Java est simple

- Simplicité relative par rapport au C++
- Certains concepts du C++ à l'origine de nombreux bugs ont été supprimés
  - Pointeurs
  - Surcharge des opérateurs
  - …
- Des études comparatives de développements montrent une diminution du coût de développement de 30 à 35 %

Java est un langage objet

- Java reprend des caractéristiques de différents langages à objets:
  - La syntaxe du C++
  - La gestion dynamique de la mémoire SmallTalk
  - Certains concepts de ADA et Objective C
- Chaque fichier source contient la définition d'une ou plusieurs classes qui sont utilisées les unes avec les autres pour former une application.
- Toute ligne de code JAVA se trouve obligatoirement dans une méthode à l'intérieur d'une classe.

Java est robuste

- Contrôle de typage « fort »
  - Toutes les variables sont typées et il n'existe pas de conversion automatique qui risquerait une perte de données.
- Le compilateur vérifie que les erreurs sont traitées par le développeur
  - Si une erreur retournée par une méthode n'est pas traitée, le programme ne compilera pas
- La gestion de la mémoire n'est pas à la charge du développeur
  - Java récupère automatiquement la mémoire inutilisée grâce au garbage collector (ramasse-miettes) qui restitue les zones de mémoire laissées libres suite à la destruction des objets.
- Un débordement d'index dans un tableau provoque une erreur
  - La mémoire n'est pas écrasée

Java est portable

- Il n'y a pas de compilation spécifique pour chaque plate forme.
- Le code reste indépendant de la machine sur laquelle il s'exécute.
- Il est possible d'exécuter des programmes Java sur tous les environnements qui possèdent une Java Virtual Machine.
  - Ce concept est à la base du slogan de Sun pour Java : WORA (**Write Once, Run Anywhere** : écrire une fois, exécuter partout)

Java est sûr

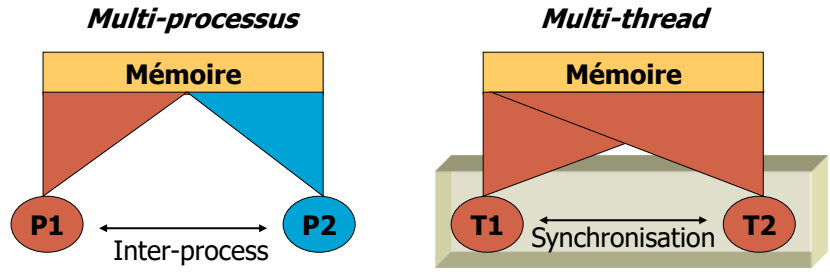
- Le compilateur interdit toute manipulation en mémoire
- L'accès au disque dur est réglementé dans une applet:
  - Aucun programme ne peut ouvrir, lire, écrire ou effacer un fichier sur le système de l'utilisateur·
  - Aucun programme ne peut lancer un autre programme sur le système de l'utilisateur
  - Toute fenêtre créée par le programme est clairement identifiée comme étant une fenêtre Java, ce qui interdit par exemple la création d'une fausse fenêtre demandant un mot de passe
  - Les programmes ne peuvent pas se connecter à d'autres sites Web que celui dont ils proviennent.

Java est économe

- Le code source a une taille relativement petite car les bibliothèques de classes requises ne sont liées qu'à l'exécution.

Java est multitâche

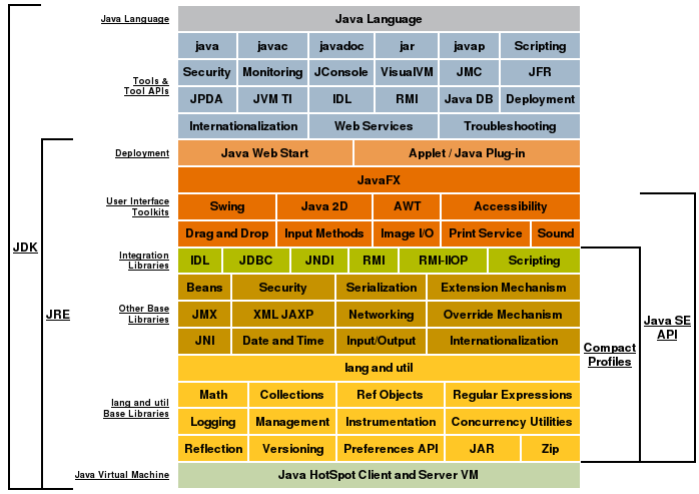
- Il permet l'utilisation de threads qui sont des unités d'exécution isolées.
- La gestion des multi-threads est intégrée dans le langage et dans la Machine Virtuelle.
- Il permet de synchroniser les threads.



Java et son environnement d'exécution

- **Environnement d'exécution**: **Java Runtime Environment (JRE)**
  - contient la Machine Virtuelle qui exécute votre programme compilé (.class) et grâce à qui la portabilité est possible,
  - le traducteur du bytecode en code machine (à la volée),
  - les binaires des bibliothèques natives de Java dont dépend votre programme.
- **Environnement de programmation**: **Java Development Kit (JDK)**
  - contient la JRE,
  - le langage Java pour écrire le code source de votre programme,
  - Le compilateur de code source en bytecode (javac),
  - d'autres outils de développement,
  - la documentation (javadoc).

Java et son environnement d'exécution



Source : https://www.oracle.com/technetwork/java/javase/tech

Plateformes d'exécution

- Sun définit trois plateformes d'exécution (ou éditions) pour Java selon les besoins des applications à développer :
  - **Java Standard Edition (Java SE) :**
    - environnement d'exécution et ensemble complet d'API
    - sert de base en tout ou partie aux autres plateformes
  - **Java Enterprise Edition (Java EE):**
    - environnement d'exécution reposant intégralement sur Java SE
    - pour le développement d'applications d'entreprises
  - **Java Micro Edition (Java ME):**
    - environnement d'exécution et API
    - pour le développement d'applications sur appareils mobiles et embarqués

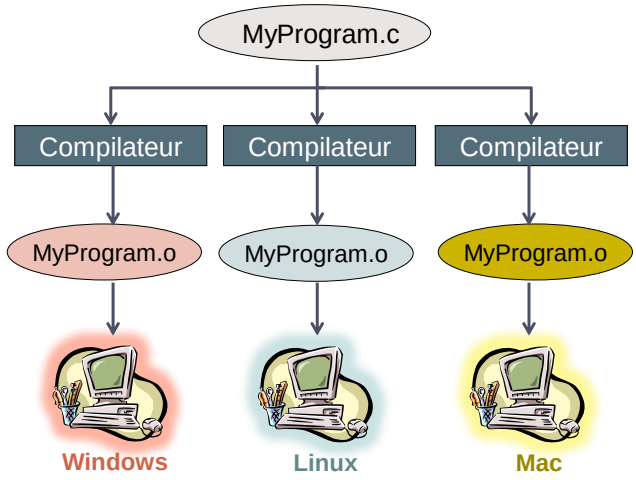
Applications Java

- Avec différentes éditions, les types d'applications qui peuvent être développées en Java sont nombreux et variés :
  - Applications desktop
  - Applications web
  - Applications pour appareil mobile
  - Applications pour carte à puce
  - Applications temps réel
  - La platforme Android (Google phone OS) se base sur Java pour les user apps.

Les concepts de base

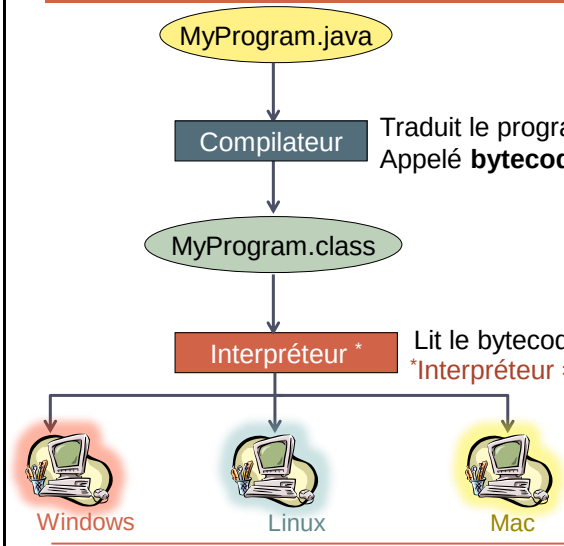
- La plate-forme Java utilise quelques notions base dans sa mise en œuvre notamment :
  - La compilation du code source dans un langage indépendant de la plate-forme d'exécution : le byte code
  - L'exécution (l'interprétation) du byte code par une machine virtuelle nommée JVM
  - La notion de package qui permet d'organiser les classes
  - Le classpath qui permet de préciser au compilateur et à la JVM où elle peut trouver les classes requises par l'application
  - Le packaging des classes compilés dans une archive de déploiement nommé jar (Java ARchive)

Déploiement d'un programme en C



Génère du code natif directement exécutable, mais **spécifique à chaque environnement.**

Déploiement d'un programme en JAVA



Traduit le programme en un code intermédiaire
Appelé **bytecode** – **indépendant de la machine**

Lit le bytecode et exécute le sur la machine
*Interpréteur = JVM + API

Première application JAVA

Comment développer une application?

Deux façons d'écrire des programmes Java:

- En écrivant le code dans un simple éditeur de texte (emacs, Xemacs, gedit, notepad, . . .).
  - Compilation et exécution du code en ligne de commande
- En utilisant un environnement de développement (Integrated Development Environment - IDE)
  - **Eclipse** (http://www.eclipse.org)
  - Netbeans (http://www.netbeans.com)
  - IntelliJ Idea (https://www.jetbrains.com/idea/download)
  - Jdeveloper (http://www.oracle.com/technetwork/developer-tools/jdev)
  - Android Studio (https://developer.android.com/studio/index.html)

Premier programme en Java

```
public class PremierProg {

    public static void main(String[] args) {
        // Affichage d'un message dans la console
        System.out.println("Premier programme en Java à l'ENSAJ");
    }
}// fin de la classe PremierProg
```

- Ce programme doit être écrit dans un fichier source qui porte exactement le même nom que la classe : PremierProg.java
  - Règle de bonne pratique : **1 seule classe par fichier source** !

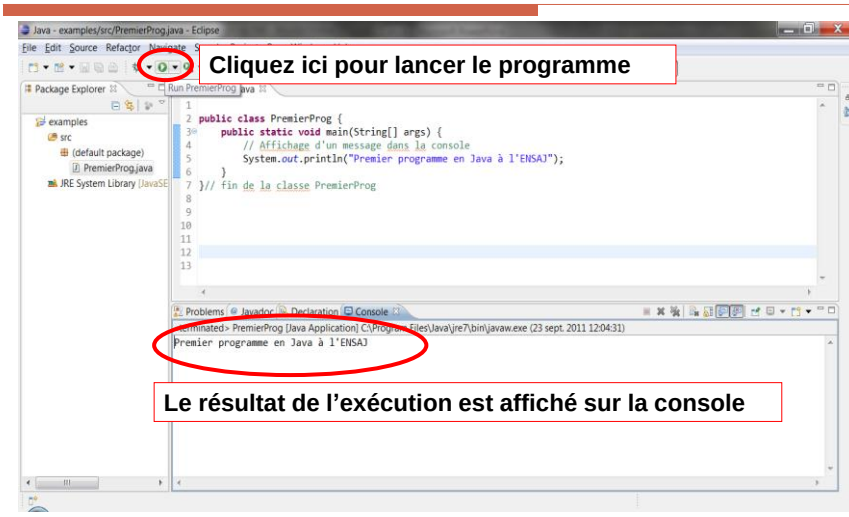
Compilation et interprétation (1/2)

- Pour la Compilation et exécution du code en ligne de commande:
  - **La phase de compilation** se fait à l'aide de la commande **javac**.
    - javac PremierProg.java
    - Le compilateur génère le bytecode dans le fichier : PremierProg.class
  - **La phase d'interprétation** (exécution) se fait à l'aide de la commande **java**.
    - java PremierProg
    - «Premier programme en Java à l'ENSAJ» s'affiche à l'écran

Compilation et interprétation (2/2)

- Avec un environnement de développement intégré (EDI), par exemple Eclipse, le programmeur n'appelle pas explicitement les commandes javac ou java, mais elles sont appelées par l'EDI.
  - **La phase de compilation** se fait à la volée par l'EDI (il y a donc bel et bien une compilation qui est faite implicitement!) lorsque des changements ont été effectués sur les fichiers sources.
  - **La phase d'interprétation** se lance graphiquement à l'aide du bouton Run. L'EDI lance l'interprétation (c'est-à-dire la commande java !) lors du clic sur le bouton Run. Le résultat apparaît dans une console intégrée à l'EDI.

Exemple d'Eclipse



Structure de la classe PremierProg

```
public class PremierProg {

    public static void main(String[] args) {
        // Affichage d'un message dans la console
        System.out.println("Premier programme en Java à l'ENSAJ");
    }
}// fin de la classe PremierProg
```

- **public :** définit le droit d'accès des autres classes à la classe PremierProg.
- **class :** indique que tout programme Java réside dans une classe. Le contenu de celle-ci est délimité par des accolades, **{}**.
- **PremierProg** : est le nom de la classe. Tout nom de classe doit commencer par une lettre et être en CamelCase.
  - Ex. : PlaneteSolaire, Point, Voiture …

Structure de la classe PremierProg

```
public class PremierProg {

    public static void main(String[] args) {
        // Affichage d'un message dans la console
        System.out.println("Premier programme en Java à l'ENSAJ");
    }
}// fin de la classe PremierProg
```

- **public static void main :** doit toujours être la signature de la méthode principale de votre programme pour que la machine virtuelle puisse l'exécuter.
- **static :** indique que la méthode main n'est pas liée à une instance (objet) particulière de la classe. Par analogie, c'est l'équivalent d'une procédure ou d'une fonction usuelles dans C.
- **String[] args** : tableau de chaînes de caractères, qui sont les arguments passés à votre programme. Le nom du paramètre **args** est arbitraire, mais *conventionnel*.

Structure de la classe PremierProg

```
public class PremierProg {

    public static void main(String[] args) {
        // Affichage d'un message dans la console
        System.out.println("Premier programme en Java à l'ENSAJ");
    }
}// fin de la classe PremierProg
```

- Littéralement, elle signifie "la méthode **println()** va écrire « Premier programme en Java à l'ENSAJ » en utilisant l'objet **out** de la classe **System**".
  - **System** : l'appel d'une classe qui s'appelle "System". qui permet surtout d'utiliser l'entrée et la sortie standard.
  - **out** : objet de la classe **System** qui gère la sortie standard.
  - **println** : méthode qui écrit dans la console la chaîne passée en paramètre.

Structure générale d'un programme

```
public class NomDeLaClasse {
    public static void main(String[] args) {
        Instructions du programme;
    }
}
```

- Tous les programmes Java sont composés d'au moins une classe
- Une méthode main() qui est le point de départ du programme.
- Une méthode est une suite d'instructions à exécuter. Une méthode contient :
  - **une entête :** celle-ci va être un peu comme la carte d'identité de la méthode.
  - **un corps :** le contenu de la méthode. Ce qui va être fait ! Délimité par des accolades **{}**.
  - **une valeur de retour :** le résultat que la méthode va retourner. **SAUF** pour les méthodes de type **void** qui ne renvoient rien.

N’oublions pas !

“Everyone knows that debugging is twice as hard as writing a program in the first place. So if you’re as clever as you can be when you write it, how will you ever debug it?”

Brian Kernighan, "The Elements of Programming Style", 2nd edition, chapter 2