COURS 1

1. INTRODUCTION AU JAVA

1.1 Points fort de Java ?

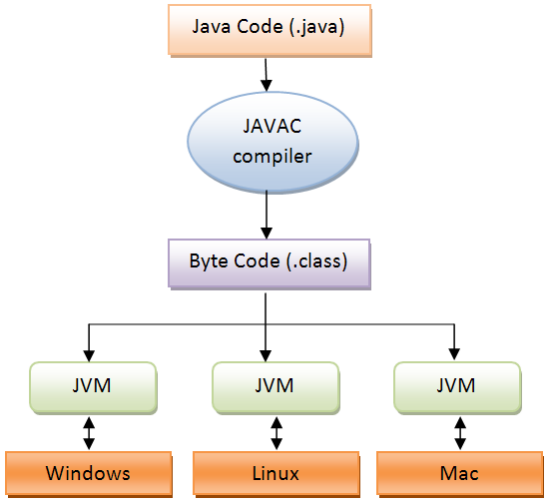
➢ **Un langage de programmation orienté-objet**

– Un technique qui se concentre sur le développement d'objets et la composition d'un système par objets

➢ **Portable / indépendant de la plateforme**

– Le compilateur Java génère un programme dont l'architecture est neutre – ce qu'on appelle les bytecodes.

– Les bytecodes peuvent être interprétés sur n'importe quelle machine sur laquelle le Java Virtual Machine (JVM) à été portée.



➢ **Interprété** : l'exécution d'un programme Java se fait par la JVM interprète les bytecodes générés par le compilateur Java.

➢ **Fiable** : il intègre en particulier un modèle de gestion de pointeurs qui écarte toute possibilité d'écrasement de données.

1.2 Quelles sont les 4 principales qualités de Java ?

- **Distribué**

  – Le langage a été conçu pour fonctionner en réseau et possède des bibliothèques (API) pour la gestion des protocoles TCP/IP, HTTP, FTP, etc.

- **Sécurisé**

  – Livré avec une architecture de sécurité moderne (cryptographie, authentification, autorisation, etc).

- **Multi-thread :**

  – Fournit un modèle multi-thread léger, efficace et facile à utiliser. Permet de créer des programmes interactifs et temps-réel.

- **API très développé**

1.3 Couche logicielle Java pour les plateformes matérielles

➢ **Une couche logicielle disponible pour multiples plateformes matériel.**

➢ **JRE : Java Runtime Environment**

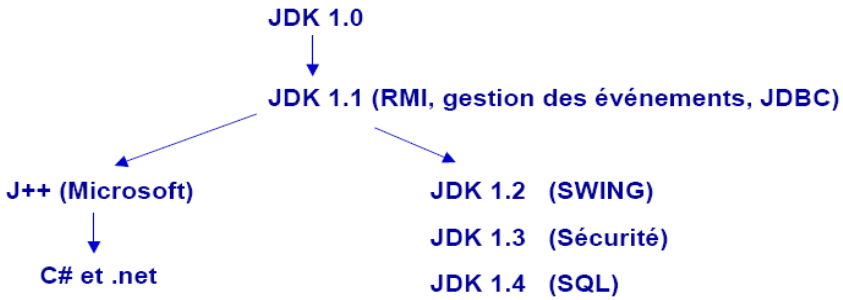
– L'interpréteur et les API standard, nécessaires à l'exécution d'une application Java déjà compilé.

➢ **JDK : Java Development Kit**

- Windows : https://www.oracle.com/java/technologies/downloads/#jdk19-windows
- MacOS : https://www.oracle.com/java/technologies/downloads/#jdk19-mac

– Les outils nécessaires pour l'écriture, compilation, teste, documentation, déploiement des applications et applets Java.

Le JDK : Société JavaSoft (branche Java de Sun)



Le JDK 1.2.1 a été officiellement renommé Java 2

JDK 2… est la dernière version

1.4 Trois plateformes Java

➢ Standard Edition (J2SE)

– Fournit un environnement idéal pour programmer la plupart des applications

– Contient les outils, services et API nécessaires pour l'écriture, teste, déploiement et exécution des applications et applets Java.

➢ Enterprise Edition (J2EE)

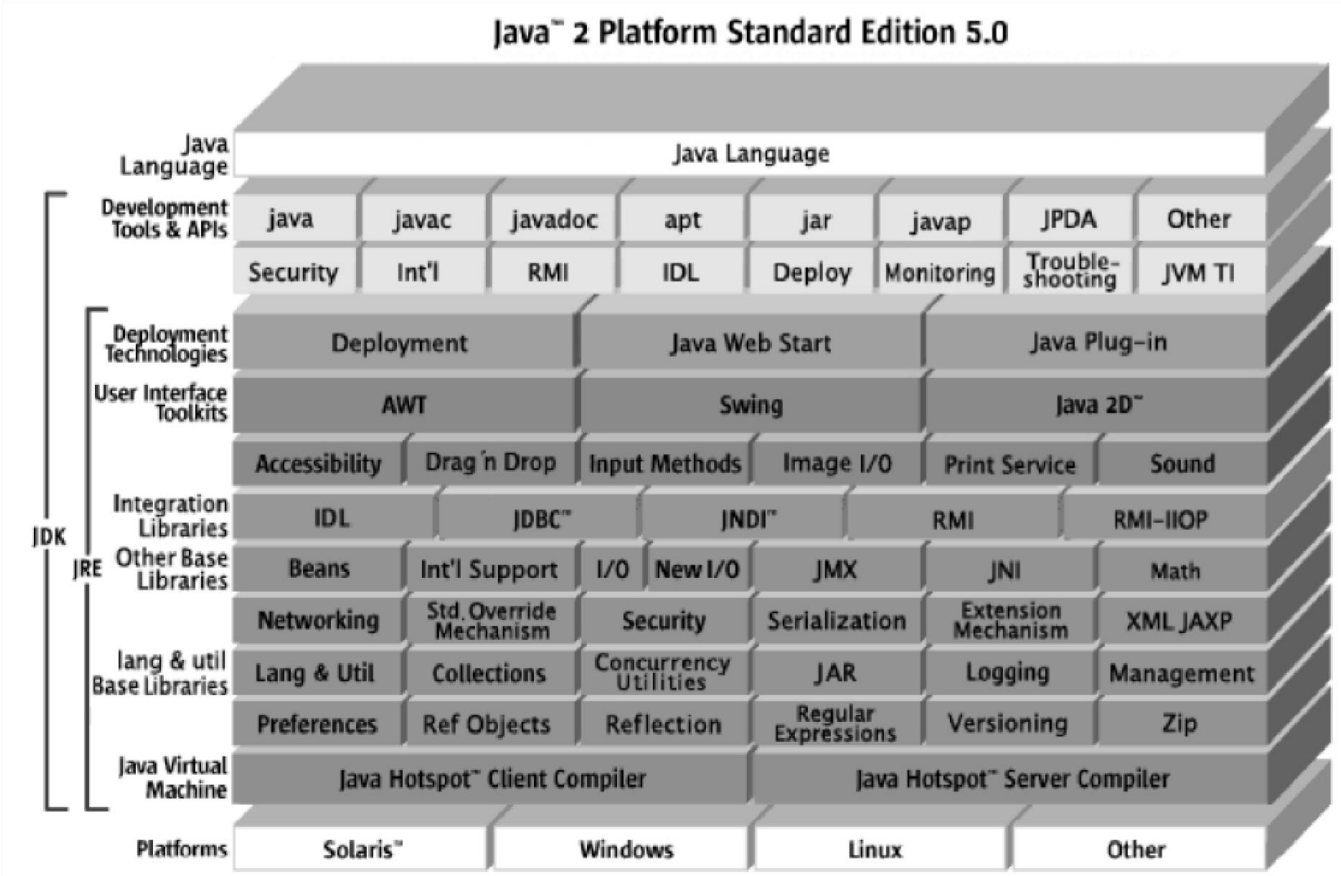
– Plateforme très complète pour le développement d'applications multi-tiers à base de composants (« Java Beans »)

– Contient J2SE + des outils, services et API pour faciliter le développement en entreprise

➢ Micro Edition (J2ME)

– plateforme restreinte spécialement adaptée au développement des systèmes embarqués

1.5 La plateforme J2SE



1.6 Couches de la plateforme J2SE

➢ Quelques outils de développement :

- **javac** – compilateur
- **java** - interpréteur
- **javadoc** – générateur de documentation
- **jdb** – débogueur
- **jar** – archive exécutable
- gestionnaire d'archives

➢ Quelques packages utiles:

- **java.lang** Object, System, String, Number, Integer, Math, …
- **java.applet** Application légères à inséré dans une page web
- **java.awt** et **javax.swing** Interfaces graphiques évolués
- **java.awt.event** Programmation évènementielle avec les listeners (souris, clavier) sur les composants graphiques
- **java.io** Gestion des Entrées - sorties
- **java.util** Classes utilitaires (Scanner, Random, ArrayList, HashMap, …)
- **java.sql** Accès aux bases de données

1.7 Les Environnements de Développement (IDE) pour Java

➢ **Un des meilleurs environnements de développement (IDE) pour Java**

IntelliJ IDEA – the Leading Java and Kotlin IDE (jetbrains.com)

➢ **Tutorial d'installation de l'un des meilleurs IDE de Java : IntelliJ de JetBrains**

IntelliJ : Tutoriel installation de IntelliJ

➢ **Tutorial complet sur la configuration et l'utilisation de IntelliJ**

https://www.jetbrains.com/help/idea/getting-started.html

1.8 Utilisation de la documentation des API Java

➢ **Téléchargement de la documentation des API : https://docs.oracle.com/javase/8/docs/api/**

- La documentation des API est sous forme HTML, en anglais.
- Elle documente toute classe fournie dans les API standards.
- Pour programmer efficacement en Java, il est essentiel d'en maîtriser l'utilisation.

1.9 Ressources en ligne

Site du cours POO Java

- Page BoostCamp : Planning du programme et intervenants, Supports de cours et TP, tests y compris avec **Safe Exam Browser**, ressources, annales DS et corrigés
- Documentation des API Java (javadoc) version 8
- Exemples avec la classe Scanner
- Exemples avec la classe ArrayList
- Etc.

Sites Java (aide, sources et forums) en Français

- http://java.developpez.com/cours/ - Club d'entraide français
- Openclassrooms - Apprenez à programmer en Java

2.1 Objectifs de l'approche objet

➢ **Ne montrer que la face visible de l'objet :**

– Grâce aux **méthodes** que l'on peut y effectuer.

➢ **Cacher son autre face :**

– Sa structure parfois complexe par le biais de types abstraits appelés **classes**.

Exemple : empiler et dépiler des objets sans se préoccuper sur quels objets (assiettes, …) s'effectuent ces 2 opérations.

➢ **Envoyer ou/et recevoir un flux d'informations à destination ou/et en provenance d'un acteur :**

– Un acteur ou **objet** exerce en général des fonctions ou **méthodes** plus ou moins complexe.

– Un flux d'informations représente un échange de données entrée/sortie entre deux acteurs.

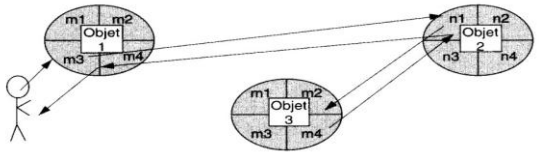


Figure 3 — Exécution d'un programme objet.

Exemple : L'acteur Service des Ventes envoie un ensemble de documents administratifs (Bon de commandes, facture…) à l'acteur Client et vice-versa (ordre de virement …)

➢ **Structuration des classes avec héritage :**

– Les objets peuvent être reliés par des relations du type « est un ».

Exemples : système de multifenêtrage (hiérarchie), relation entre une personne et des profs, étudiants …

2.2 Pourquoi les objets ?

- **Se rapprocher le plus possible du monde réel**
- **Réutiliser et étendre des logiciels existants**
  - à partir de bibliothèques spécialisées et facilement modifiables
- **Travailler avec des environnements de développement riche**
  - interface, débogage et trace des exécutions
- **Disposer d'outils interactifs permettant la création rapide d'interfaces homme/machine graphiques**
  - Grande qualité, capables de réagir à tout événement extérieur (Exemple : un clic souris)
- **Faciliter le prototypage rapide des applications :**
  - Interfaces homme-machine et logique générale des traitements, sans besoin de tout coder
- **Faciliter l'exploitation du parallélisme :**
  - Implémentation sur des machines multiprocesseurs ou distribuées avec des threads

En résumé, 3 objectifs :

a) Structurel : l'objet est une **instance** d'un type de données caractérisé par une structure cachée par des opérations
b) Conceptuel : l'**objet** correspond à un concept du monde réel qui peut être spécialisé (héritage, hiérarchie)
c) Acteur : l'objet est une entité autonome et active qui répond à des messages ou **méthodes**

2.3 Première approche de la Programmation objet

➢ Aspect statique de l'objet :

– Son état à l'aide de variables d'instances ou **attributs**

Exemple (Figure 1) : état = dessin de la voiture ;

attributs = couleur, nombre de portes, vitesse, niveau d'essence …

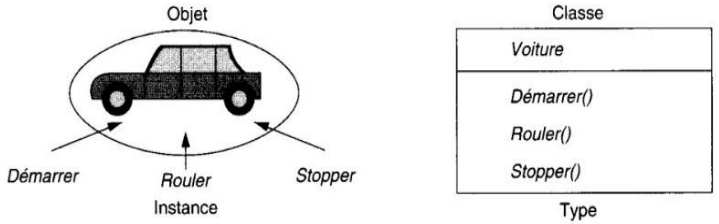


Figure 1 — Représentation d'objet et de type d'objet.

➢ Aspect dynamique de l'objet :

– Son comportement grâce aux opérations ou **méthodes** que l'on peut y effectuer peut changer l'état de l'objet

Exemple (Figure 1) : démarrer, faire rouler, stopper une voiture peut changer son état

➢ Echange de messages :

– Interactions entre des objets = méthodes + paramètres

Exemple : un conducteur démarre, fait rouler ou stoppe une voiture

➢ Type abstrait :

– Regroupement d'objets similaires dans lequel on définit les propriétés (état et comportement) de ces objets qui sont donc des **instances** du type appelé généralement **Classe**.

➢ Hiérarchie d'héritage des classes :

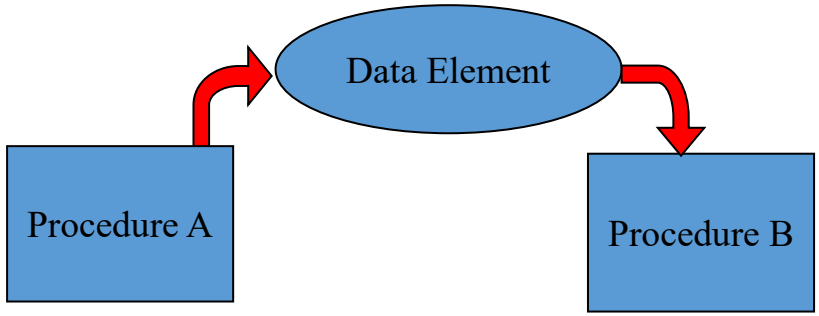
– elle permet facilement la spécialisation et la réutilisation du code

Exemple : la classe Voiture hérite de la classe Véhicule

2.4 Programmation fonctionnelle (langage C) contre programmation objet (Java)

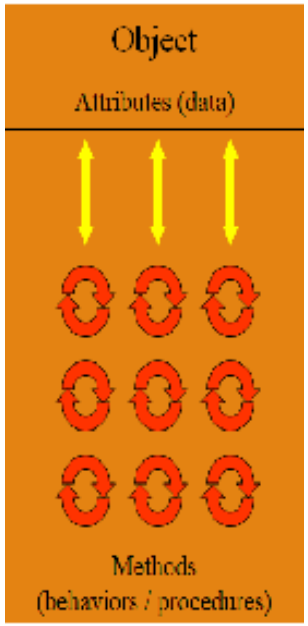
La programmation procédurale

Les langages de programmation plus anciens étaient procéduraux.
Une procédure est un ensemble d'instructions de langage de programmation qui, ensemble, exécutent une tâche spécifique.
Les procédures fonctionnent généralement sur des éléments de données qui sont séparés des procédures.
Dans un programme procédural, les éléments de données sont généralement passés d'une procédure à une autre.



La programmation objet

La programmation orientée objet est centrée sur la création d'objets plutôt que sur des procédures.
Les objets sont un mélange de données et de procédures qui manipulent ces données.
Les données d'un objet sont appelées attributs.
Les procédures dans un objet sont appelées méthodes.



L'influence du langage C sur le Java

- Syntaxe restreinte proche du C, du C++
- Les types primitifs de données : **int char float**… mais pas de pointeur explicite

Types de donnée primitifs

Type	Valeur	Valeur par défaut	Taille (bits)	Valeur min	Valeur max
boolean	false ou true	false			
char	caractère Unicode	'\u0000'	16	'\u0000'	'\uffff'
byte	entier signé	0	8	-128	+127
short	entier signé	0	16	-32768	+32767
int	entier signé	0	32	-2147483648	+2147483647
long	entier signé	0	64	-9223372036854775808	+9223372036854775807
float	réel en virgule flottante	0.0	32	-3.40282347**E38**	+3.40282347**E38**
double	réel en virgule flottante	0.0	64	-1.79769313486231570**E308**	+1.79769313486231570**E308**

Réf : https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html http://imss-www.upmf-grenoble.fr/prevert/Prog/Java/CoursJava/TypeDeDonneePrimitifs.html https://unicode-table.com/en/#latin-extended-a

12

- Le cast
  Exemple : int x = 5 ; byte y = (byte) x ;

- pas de **struct** mais des **class**

- **pas de pointeur** et pas de problème de gestion mémoire
- Les commentaires : **//** ou **/* ... */** ou **/**... */** pour les commentaires **javadoc** (fourni avec le JDK) qui documente votre code source Java.
- Affectation simple : **=**
- Affectation composée : **+= -= *= /=**
- Egalité : **==**
- Incrémentation et décrémentation pré ou post : **++ --**
- opérateurs mathématiques : **+ - * / %**
- opérateurs relationnels : **< > <= >= == !=**
- opérateurs logique : **&& || !** mais n'acceptent que les opérandes booléens de type **boolean** dont le résultat est **true** ou **false**
- Instructions de test : **if...else switch**
- Boucles : **while do...while for break** (sortie de boucle) **continue** (passer directement au tour suivant)

```
for ( int i = 0; i < 10; i++ ) {
        System.out.println(" * * * * * "); }
```

```
Random rand = new Random();
int n = rand.nextInt(),   nbch = 1;
while ( n > 9 ) {
    n = n / 10;
    nbch++;}
```

```
Random rand = new Random();
int n;
do {
      n = rand.nextInt(101);
} while ( (n % 2 == 0)  ||  (n % 3 == 0) ) ;
```

2.5 Les évolutions du langage Java par rapport au langage C

- Programmation fonctionnelle structurée (C, …) :
  - Associer des traitements spécifiques à des structures de données formelles qu'ils manipulent

- Tout programme d'application procédurale se partage en 3 parties :
  - La déclaration des constantes, types et variables (données)
  - La définition des procédures et fonctions (opérations)
  - Le programme principal

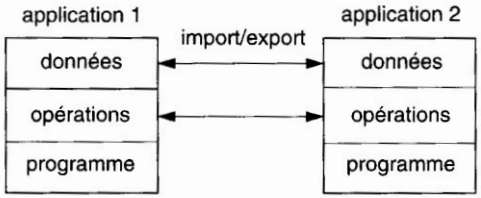


Figure 1 - Approche structurée

Limites de la programmation fonctionnelle :

Approche peu évolutive

Si les structures de données ou les procédures doivent être partagées par différents programmes à l'aide d'ordres import/export et que les données évoluent.

Modifier une structure de données ⇒ modifier tous les programmes la manipulant

Exemple : présentoir composé d'objets hétéroclites à dessiner sur un écran. Chaque objet a un type qui le caractérise (une voiture, un avion, un vélo, une pomme, …)

```
Approche classique :
   pour chaque x du présentoir dessiner(x) ;
   dessiner(x)
      selon type(x)
      A : dessiner-A (x) ;
      B : dessiner-B (x) ;
      .....

Approche objet :
pour chaque x du présentoir, message : [x dessiner]
```

Figure 2 — Exemples de programmes classique et objet.

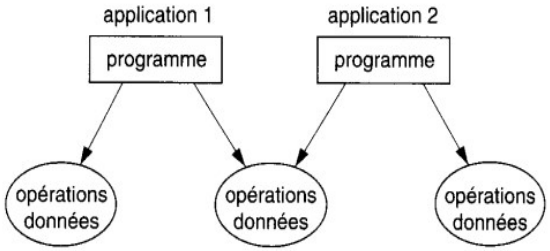


Figure 2 - Approche objet

L'**encapsulation** permet d'étendre l'approche modulaire des langages structurés avec des **types abstraits de données**

Réutilisation et extensibilité de l'application

2 concepts propres à l'approche objet : l'**héritage** et le **polymorphisme**

3.1 La Programmation Orientée Objet

➢ La POO (Programmation Orientée Objet)

- Fondée sur le concept d'**objet**
- Par analogie avec l'équation de Wirth (concepteur du langage Pascal) :

Méthodes + Données = Objet où méthodes = procédures associées à un objet.

➢ Un programme contient :

- Des types de données basiques : nombres et caractères
- La possibilité de définir les entités d'intérêt, c'est-à-dire de nouveaux types d'objets appelés **classes**.

Exemples :
✓ Si vous êtes concerné par le service des ressources humaines, vous créerez une classe d'objet *Employé*.
✓ Si vous travaillez aux eaux et forêts, vous créerez une classe d'objets *Arbre*.

3.2 Conventions

- **Pour connaitre toutes les conventions en Java : https://www.jmdoudoux.fr/java/dej/chap-normes-dev.htm**
- **Un programme Java consiste en un ensemble de classes, avec au minimum 1 classe**
- **Le nom d'une classe commence par une majuscule et le nom d'un objet par une minuscule**
- **On met le code de chaque classe dans un fichier séparé**
  - ✓ C'est mieux au démarrage
  - ✓ On DOIT donner le même nom au fichier et à la classe qui y est déclarée
- **Un fichier source Java doit avoir le suffixe .java**
- **Les commentaires en Java /** à la mode javadoc */**
  - ✓ Les commentaires javadoc sont des balises HTML permettant de générer automatiquement de la documentation
  - ✓ « javadoc » ignore tous les espaces au début d'une ligne de même que le premier astérisque
  - ✓ Il ne peut y avoir d'espace entre le commentaire de classe et la classe
  - ✓ L'arobas (@) permet l'introduction d'informations spécifiques :

`@see class`	Permet de spécifier les classes liées.
`@author Nom`	Permet de citer un ou plusieurs auteurs.
`@param Nom Descript`	Permet de donner de l'information sur un ou plusieurs paramètres.
`@throws Except Desc.`	Permet de donner de l'information sur les exceptions.

3.3 4 principales caractéristiques d'un objet

L'identité d'un objet :

- Représenté par un identifiant unique et invariable permettant de référencer l'objet indépendamment des autres objets

Les attributs de l'objet :

- **variables d'instance** de la classe instanciée

L'état d'un objet :

- la valeur de ses attributs

Exemples : Un livre comporte un nombre de pages, un poids, un titre... Cela traduit l'état de l'objet à tout moment. Une personne a une taille, un poids, une couleur de peau... Ces attributs sont susceptibles de changer de valeur, le poids d'une personne évolue sans cesse. Certains attributs par contre ne verront pas leurs valeurs évoluer, la couleur de la peau par exemple.

Le comportement de l'objet :

- Ensemble des **méthodes** (opérations, actions) applicables à l'objet et définies dans sa classe d'appartenance
- L'objet réagit en réponse à des messages d'un autre objet : la réception d'un message va engendrer la réaction de l'objet
- L'objet doit posséder dans son comportement une action capable de traiter le message

Un **objet** = identité + état + comportement

3.4 La classe et ses membres

Si la notion d'objet est le point de départ, il est nécessaire d'imaginer un niveau conceptuel qui permette de regrouper les objets selon le comportement et l'état. Ainsi une classe va rassembler les objets de même nature.

➢ Classe

Type abstrait de données caractérisé par des propriétés communes à ses objets, et mécanisme permettant de créer des objets ayant ses propriétés. Par convention, son nom commence par une majuscule.

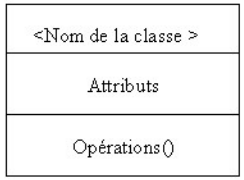
– Syntaxe partielle : visibilité **class** nom **{** corps de la classe **}**

Exemple : **public class** Train {…}

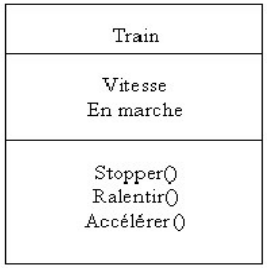
➢ Les attributs d'une classe

Ils ont un niveau de visibilité (**public**, **private** ou **protected**), un nom et un type (soit de base, soit une classe). Par convention, son nom commence par une minuscule.

3.5 Représentation d'une classe



Exemple :



Classe Train

Exercice 1 : D'après la classe *Train* modélisée ci-dessus, implémenter en Java cette classe avec les propriétés suivantes :

- Ses 2 attributs **public** *: vitesse* (entier) initalisé à 0 et *enmarche* (**boolean**) initialisé à false.
- Un constructeur **public** par défaut (sans paramètre) initialise la *vitesse* à 150.
- Sa méthode *stopper()* : modifie les attributs *vitesse* et *enmarche* pour stopper le train.
- Sa méthode *ralentir()* : diminue la *vitesse* en vérifiant qu'elle n'est pas négative. Si elle arrive à 0, l'attribut *enmarche* devient **false**.
- Sa méthode *accereler()* : augmenter la *vitesse* dans la limite de 350.
- Ecrire le **main** (voir le chapitre **Le programme dans une classe**) qui effectue les instructions suivantes :

- Instancier un objet (voir le chapitre **Instanciation d'un objet**) de la classe *Train*
- Stopper cet objet en appelant sa méthode *stopper()*
- Ralentir cet objet en appelant sa méthode *ralentir()*
- Accélérer cet objet en appelant sa méthode *accereler()*
- Afficher ses attributs *vitesse* et *enmarche* avant et après chaque appel des 3 méthodes précédentes.

Observez les résultats affichés. Qu'en concluez-vous ?

➢ Les méthodes d'une classe

Une méthode est une fonction ou procédure définie dans le corps d'une classe. Une méthode peut accéder directement à tous les autres membres (attributs et membres) de sa classe. Elle peut définir des variables locales qui seront définit pour la durée de l'exécution de la méthode. Par convention, son nom commence par une minuscule.

- Syntaxe partielle : *visibilité typeDeRetour nom (listeParamètres) { corps de la méthode }*
- Liste de paramètres :
  - ✓ Une liste de déclarations séparées par des virgules.
  - ✓ Les arguments (paramètres) sont passés par valeur.
  - ✓ La méthode crée et manipule une copie locale; Toute modification est limité au contexte de la méthode.

- Type de retour :
  - ✓ Un type primitif (**int float char** …), une référence, ou **void**.
  - ✓ On utilise le mot-clef **return** pour quitter la méthode et renvoyer une valeur.
  - ✓ La valeur retournée doit obligatoirement avoir le même type que le type de retour de la méthode

Exercice 2 : écrire une classe avec les 2 méthodes suivantes :

- Une méthode qui retourne la concaténation de 2 chaînes (**String**) en paramètres
- Une méthode avec 2 paramètres (un objet de la classe *Train* de l'**exercice 1** et un **String**) qui affiche un message suivi de l'appel de la méthode précédente avec en paramètres la *vitesse* de l'objet du *Train* et du **String**.

3.6 Le programme dans une classe

Le point d'entrée pour tout programme Java est la méthode :

public static void main (String args**[]) {}**

- ✓ Cette méthode doit être définie dans une classe **public**,
- ✓ Cette classe doit être définie dans un fichier **.java** éponyme (de même nom que la classe),
- ✓ Une fois ces conditions remplies, on exécute la méthode en envoyant le nom de la classe à l'interpréteur Java (java),
- ✓ Quand java l'exécute, il fournit à la méthode les arguments **args** reçus sur la ligne de commande (cmd sur Windows, terminal sur macOS/Linux) sous forme d'un tableau de **String** : voir lien **https://openclassrooms.com/forum/sujet/comment-passer-les-argument-de-la-methode-main-30238**
- ✓ Pour configurer les arguments du **main()** sur **IntelliJ** : https://www.jetbrains.com/help/idea/program-arguments-and-environment-variables.html#program_arguments
- ✓ **main()** devrait servir uniquement comme point d'entrée et contenir aussi peu d'instructions que possible,
- ✓ Idéalement, elle devrait instancier un objet et passer le contrôle à leurs méthodes,
- ✓ La méthode est **static** : elle peut donc être exécutée par l'interpréteur sans avoir à créer un objet.

Exercice 3 : Ecrire une classe contenant le **main** dont les arguments sont les suivants :

- Le premier argument est un nom (**String**)
- Le second argument est un âge entier

Le **main** affiche le nom et l'âge en arguments. Servez-vous des liens ci-dessus pour y arriver.

3.7 Les membres de classe

- Une constante **final** est un membre de classe typé dont la valeur est initialisée à la déclaration et ne peut pas changer
  - ✓ Syntaxe partielle : **final** type NomConstante **=** Valeur **;**

- Par défaut, les attributs et méthodes d'une classe sont des membres d'instance
  - ✓ Chaque instance aura sa propre copie du membre
  - ✓ Le membre est créé et initialisé avec l'instance

- Mais il existe également des membres de classe **:**
  - ✓ les instances de la classe partagent une seule copie du membre,
  - ✓ le membre est créé et initialisé avec la première utilisation de la classe.

- Les membres de classe peuvent être utiles:
  - ✓ pour partager une donnée entre toutes les instances (eg compteur),
  - ✓ quand on veut exécuter du code sans avoir à créer un objet (eg **main**),
  - ✓ quand il n'est pas logique de créer un objet (eg System).

- Les membres de classe sont **static** :
  - ✓ Pour créer un membre de classe on utilise le mot-clef **static** placé devant son type : **static** visibilté type nomMembre
  - ✓ Pour accéder à un membre de classe, on passe par le nom de sa classe : NomDeLaClasse.nomDuMembre
  - ✓ On peut également y accéder par une référence (un objet), mais cette pratique peut créer de la confusion et est à éviter.
  - ✓ Il faut utiliser les membres de classes aussi peu que possible
  - ✓ ils ne sont pas orienté objet !

Exercice 4 : Ecrire une classe dont les propriétés sont les suivantes :

- Un attribut public entier **static** initialisé avec une valeur.

- Un attribut public entier pas **static** initialisé avec une valeur.
- Un attribut constante **String** initialisé avec une valeur.
- Méthode **static** qui incrémente l'attribut **static**.

Le **main** de cette classe effectue les instructions suivantes :

- Afficher la valeur de l'attribut entier **static** de la classe.
- Afficher la valeur de l'attribut entier pas **static** de la classe.

Pourquoi cette instruction est illégale ? Que faut-il modifier dans le code pour qu'elle soit légale ?

- Incrémenter et afficher la valeur de l'attribut entier **static** de la classe.
- Afficher la valeur de l'attribut constante **String**.
- Modifier la valeur de l'attribut constante **String**.

Pourquoi cette instruction est illégale ?

3.8 Instanciation d'un objet

- La création d'objets nécessite l'allocation d'une zone mémoire et l'initialisation de l'état (variables d'instances ou attributs).
- La création d'objets s'effectue par activation d'un constructeur d'objets.
- Le constructeur est une méthode particulière disponible au niveau de la classe qui sert à initialiser un objet : son rôle est d'initialiser les valeurs des attributs de l'objet.
- Le constructeur par défaut ne comporte pas de paramètre : son rôle est d'initialiser les valeurs par défaut des attributs de l'objet instancié. S'il n'est pas défini dans la classe, la valeur de chaque attribut est initialisée par défaut : 0 pour les nombres, **null** pour les objets.
- Le constructeur porte le nom de la classe.

- Ce n'est pas une méthode donc sans type, sans retour.
- Un objet s'instancie à partir de sa classe avec **new** qui alloue un espace mémoire pour l'objet instancié.
- Après son instanciation, l'objet bénéficie des attributs et des méthodes de sa classe : **objet.nomAttribut** ou **objet.nomMethode(…)**
- L'appel d'un constructeur n'est pas considéré comme un envoi de message car l'objet récepteur n'existe pas.
- Le constructeur peut être surchargé (redéfini, en Anglais « overload ») en définissant une ou plusieurs fonctions portant le même nom de la classe à l'intérieur de celle-ci.
- Le choix du constructeur s'effectue alors en fonction du type des arguments d'appel, en utilisant la capacité de polymorphisme du langage.

3.8 Typage d'un objet

- Chaque objet aura sa propre copie des attributs de la classe : cette copie existera pour la vie de l'objet.
- Tout objet peut être référencé et partagé
  - ✓ Une référence est un pointeur vers un objet avec lequel nous pouvons manipuler les membres de l'objet.
  - ✓ La valeur **null** indique qu'il ne pointe actuellement pas vers un objet

  Exemple : Train trainsdi = **null** ; // trainsdi dont le type est la classe Train ne pointe sur aucun objet

- Un type primitif (**int**, **float**, …) définit une plage de valeurs sans méthodes associées.

- Une variable automatique de type primitif n'est pas un objet :
  - ✓ Il définit un emplacement mémoire contenant une valeur tombant dans la plage d'un type primitive.
  - ✓ Les variables automatiques ont quelques avantages : elles sont allouées automatiquement au moment de la déclaration et leur exécution est plus rapide
  - ✓ Son identifiant (nom) ne référence donc pas un pointeur mais contient directement sa valeur

- Un attribut est une variable automatique ou une référence (objet) définie dans le corps d'une classe.

- L'identifiant de l'objet instancié avec **new** est un fait un pointeur caché sur le contenu de l'objet :
  - ✓ Contrairement à C ou C++, il n'y a pas le pointeur visible *

Exemple : Train trainsdi = **new** Train**()** ; // trainsdi pointe sur un objet contenant son état

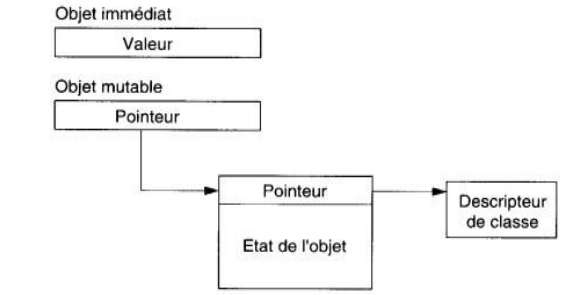
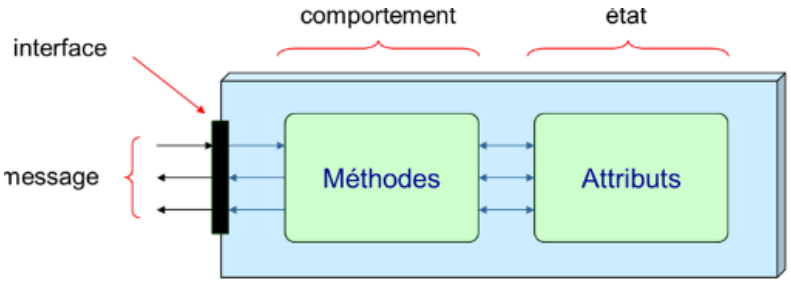
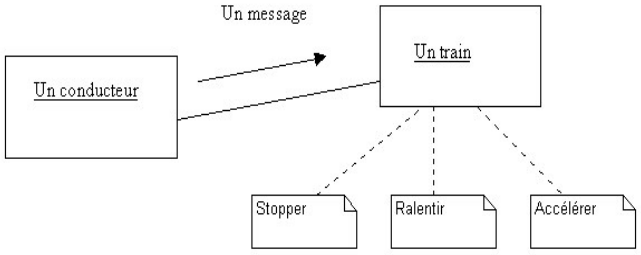


Figure 1 — Structure typique d'un identifiant d'objet et de l'objet identifié.

3.9 Echange de messages entre objets





Exemple en Java :

- ✓ Un conducteur envoie le message *stopper* à son train. C'est de la responsabilité du train de s'arrêter.
- ✓ le conducteur n'a pas besoin de savoir comment le train s'arrête.
- ✓ Les objets des deux classes communiquent à l'aide de messages, chaque objet de classe traitant les messages qu'il reçoit.

Exercice 5 : reprendre l'**exercice 1** avec la classe *Train* mais écrire le **main** dans une autre classe *Conducteur*.

Quel est l'intérêt d'appeler les méthodes de la classe *Train* dans le **main** de la classe *Conducteur* ?

3.10 Notion et intérêt des packages

- **Regroupement logique d'un ensemble de classes sous un identificateur commun.**
- **Facilite la cohabitation de logiciels.**
- **Permet de créer des classes qui ont le même nom sans créer d'interférences.**
- **L'attribution d'un nom de paquetage se fait au niveau du fichier source**

  package *sncf*; // déclare le package *sncf* en début du fichier source .java

- **Pour importer une classe d'un paquetage**

  sncf.Train p = **new** *sncf*.Train(); // appel le constructeur Train du package *sncf*
  import *sncf*.Train; // importe la classe Point du package *sncf*
  import *sncf*.* ; // importe toutes les classes du package *sncf*

- Mise à disposition « structurée » de classes à la communauté
- Trouver les classes à charger lors du lancement de la MV
  - Les répertoires qui correspondent au nom des packages
  - Un fichier d'archive (.zip ou .jar) contenant une arborescence de packages
- Éviter les conflits de noms entre classes développées par des programmeurs différents
- Délimiter un espace de visibilité des variables (public, private …)

➢ **Exemple de packages dans l'API Java :**

Nom du package	Contenu
java.lang	Bases du langages (type de données)
java.io	Entrées - Sorties
java.math	Fonctions mathématiques (sin, cos, tang …)
java.awt	Gestion des fenètres, GUI
javax.swing	Classes graphiques, IHM
java.util	Classes utilitaires (Vector, Stack …)
java.applet	Codage des applets
java.net	Applications réseau
java.rmi	Réseau avec technologie RMI
java.beans	Objets « métier »
java.security	Classes liées aux aspects sécurité
…	

Le package **java.lang** définit les classes "fondamentales" qui sont importées automatiquement : String, System, Integer, Character, …

3.11 Affichage écran / Lecture clavier

➢ **Affichage :**

System.out.print sans saut de ligne
System.out.println avec saut de ligne
System.out.print ("Texte" **[+** arg1 **+** arg2 **+ ...]);** // les [ ] signifie optionnel

➢ **Lecture (saisie) au clavier :**

- ✓ La saisie au clavier n'était pas standardisée avant JAVA5, il fallait écrire sa propre classe.
- ✓ JAVA 5 a introduit la classe **Scanner** :
  - voir le tutoriel **Exemples avec la classe Scanner** et la classe Scanner du package java.util

Exercice 6 **:** Ecrire une classe avec le **main** qui effectue les instructions suivantes :

- Déclarer et instancier un objet du clavier.
- Saisir un **String** pour un nom, sans oublier d'afficher un message pour le nom à saisir.
- Afficher un message de bienvenue suivi du nom saisi.
- Saisir un entier, sans oublier d'afficher un message pour l'entier à saisir.
- Afficher un message suivi de l'entier saisi.
- Fermer le clavier avec la méthode **close**() de la classe **Scanner**.

3.12 Tableaux

➢ En Java, un tableau est un objet.

➢ Déclaration d'un tableau avec des **[]** sans longueur crée une référence vers un tableau

- ✓ La définition de la longueur d'un tableau ne fait pas partie de sa déclaration. La longueur sera spécifiée à son instanciation.

- Le tableau lui-même doit être instancié avec **new**.
  - ✓ **new** prend comme argument la longueur du tableau

- Pour connaître la longueur d'un tableau, on lit son attribut **length**, qui est de type **int**
  - ✓ Les éléments d'un tableau sont numérotés de 0 à **length**-1.

- Différence importante : la longueur d'un tableau est fixée à l'exécution, pas à la compilation.

- Les tableaux en Java sont sécurisés
  - ✓ Il est impossible de dépasser les bornes d'un tableau : l'interpréteur lancera une exception **ArrayOutOfBoundsException**

- Les éléments d'un tableau sont toujours initialisés (à la valeur par défaut du type du tableau)
  - ✓ chaque élément est initialisé à la valeur par défaut de son type, ou **null** si c'est un tableau de références

Exercice 7 **:** Ecrire une classe avec le **main** qui effectue les instructions suivantes :

- Définir une référence vers un tableau d'entiers.
- Définir une référence vers un tableau de références **String** et pas un tableau d'objets
- Définir deux tableaux d'entiers.
- Définir un tableau d'entiers et un entier.
- Instancier un tableau de 5 entiers.
- Afficher les valeurs du tableau précédent et conclure.
- Instancier un tableau de 4 **String**.
- Afficher les valeurs du tableau précédent et conclure.
- Instancier un tableau de 5 caractères.
- Calculer la taille du tableau précédent.

➢ L'initialisation classique d'un tableau se fait avec une boucle **for** de 0 to **length**-1

Exercice 8 **:** Ecrire une classe avec le **main** qui effectue les instructions suivantes :
- Déclarer et instancier un tableau d'entiers et un tableau de String.
- Parcourir et afficher les valeurs des 2 tableaux précédents.

➢ Une autre notation permet de spécifier le contenu d'un tableau comme une valeur initiale dans la déclaration

Exercice 9 : Ecrire une classe avec le **main** qui effectue les instructions suivantes :
- Définir et initialiser les valeurs d'un tableau d'entiers sans l'instancier.

➢ Tableaux multidimensionnels

Exercice 10 : Ecrire une classe avec le **main** qui effectue les instructions suivantes :
- Définir et initialiser les valeurs d'un tableau d'entiers à 2 dimensions sans l'instancier.
- Définir un tableau d'entiers à 2 dimensions en instanciant seulement la première dimension.
- Initialiser les valeurs de la seconde dimension du tableau précédent entre 1 et 10.
- Initialiser et afficher les valeurs du tableau précédent.

➢ Copier un tableau

✓ Rappel : Un tableau est un objet! Copier la référence copie l'adresse du tableau, mais pas son contenu

Exercice 11 : Ecrire une classe avec le **main** qui effectue les instructions suivantes :
- Définir et initialiser les valeurs d'un tableau d'entiers.
- Copier la référence de ce tableau dans un second tableau.
- Afficher les valeurs des 2 tableaux.
- Instancier le second tableau avec la longueur du premier tableau.
- Copier les valeurs du premier tableau dans le second tableau.

- Modifier les valeurs du premier tableau et afficher les valeurs des deux tableaux.

3.13 La classe **String** : manipulation de chaînes de caractères

➢ **String** est une classe qui a des constructeurs et des méthodes, pas un tableau de caractères.

✓ Voir la classe **String** dans le package java.lang

➢ Quelques constructeurs de la classe **String**

✓ Extrait de la **documentation des API Java *(javadoc)***

Constructor and Description
String() Initializes a newly created String object so that it represents an empty character sequence.
String(byte[] bytes) Constructs a new String by decoding the specified array of bytes using the platform's default charset.

➢ Initialisation d'un objet de la classe **String**

Exercice 12 : Ecrire une classe avec le **main** qui effectue les instructions suivantes :
- Initialiser un **String**.
- Copier un **String** dans un autre.
- Initialiser un **String** avec un chaîne vide.

➢ L'opérateur **+** de concaténation

Exercice 13 : Ecrire une classe avec le **main** qui effectue les instructions suivantes :
- Concaténer plusieurs **String**.

➢ Quelques méthodes (signatures) de la classe **String**

Modifier and Type	Method and Description
char	charAt(int index) Returns the char value at the specified index.
int	compareTo(String anotherString) Compares two strings lexicographically.
String	concat(String str) Concatenates the specified string to the end of this string.
boolean	contains(CharSequence s) Returns true if and only if this string contains the specified sequence of char values.
int	length() Returns the length of this string.

✓ Pour convertir vers **String** :

```
int i = Integer.parseInt(chaine); // convertit la chaine en nombre de type int
float f = Float.parseFloat(chaine); // convertit la chaine en nombre de type float
double d = Double.parseDouble(chaine); // convertit la chaine en nombre de type float
String ch = String.valueOf(n);   // ch = "123"
```

Exercice 14 **:** Ecrire une classe avec le **main** qui effectue les instructions suivantes :

- Convertir un **String** en entier si possible, sinon erreur, en affichant le **String** et l'entier.
- Convertir un entier en **String**, en affichant l'entier et le **String**.
- Convertir un **String** en flottant (**float**) si possible, sinon erreur, en affichant le **String** et le **float**.
- Convertir un **String** en double si possible, sinon erreur, en affichant le **String** et le double.