Programmation Orientée Objet : Introduction à Java

E. Hyon, V. Bouquet[1]

[1]valentin.bouquet@parisnanterre.fr

Licence MIASHS Miage - 2023/2024

Grandes lignes du cours

Java c'est quoi ?

Comparatif : Java, C++
- Comparatif Concepts
- Production du binaire
- Comparatif Java C Sharp (C#)

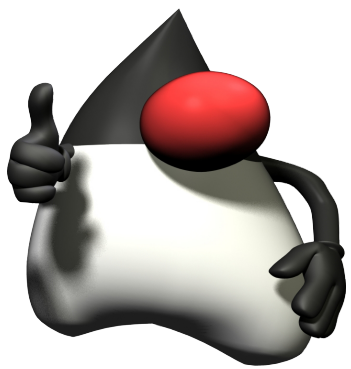
Le langage Java.

Les classes Java.
- Les attributs Java
- Les constructeurs
- Les méthodes en Java
- Classes utiles : String, tableau et classes enveloppes

Java c'est quoi ?

Java c'est quoi ?

- Deux déclinaisons Java SE (Standard Edition) et Java EE(Enterprise Edition)
- Un langage : Orienté objet fortement typé avec classes.
- Un environnement d'exécution (JRE) : Une machine virtuelle et un ensemble de bibliothèques.
- Un environnement de développement (JDK) : Une machine virtuelle et un ensemble d'outils.
- Une mascotte : Duke



Java c'est qui ?

La plate-forme et le langage Java sont issus d'un projet de *SUN Microsystems* datant de 1990 (présentation officielle en 1995). Généralement, on attribue sa paternité a trois de ses ingénieurs :

- James Gosling
- Patrick Naughton
- Mike Sheridan

Figure – 1990 Barbecue chez James Gosling

En 2010 *SUN* est racheté par *Oracle*.

Java c'est pourquoi ?

Java est devenu aujourd'hui l'un des langages de programmation les plus utilisés. Il est incontournable dans plusieurs domaines :

- **Systèmes dynamiques :** Chargement dynamique de classes.
- **Internet :** Les *Applets* java (dépréciés), servlets
- **Systèmes communicants :** *RMI*, *Corba*, *EJB*...

Java c'est pour qui ?

Pour tous : Le 13 novembre 2006, *SUN* annonce le passage de Java, c'est-à-dire le JDK (JRE et outils de développement) sous licence GPL.

Pour vous : Cette UE sur Java servira de base à l'ensemble des UE techniques du Master.

Comparatif : Java, C++

Java *versus* C++ : Historique

Deux langages orientés objets dominent : `C++` et `Java`.

- Le langage `C++` :
  - extension du langage `C`, intégrant les concepts de la programmation orientée objet,
  - né dans les laboratoires AT&T Bell,
  - première version opérationnelle date de 1983.
- Le langage `Java` :
  - mis au point en 1991 par la firme SUN Microsystems,
  - le but de Java était de constituer un langage de programmation pouvant être intégré dans des décodeurs de télévision par cable, afin de pouvoir les contrôler, de les rendre interactifs, et surtout de permettre une communication entre les appareils *write once, run everywhere*

Outline

Comparatif Concepts

Java *versus* C++ : concepts (1)

- Java est très proche du langage C++ (et donc du langage C) étant donné qu'il a quasiment la même syntaxe.
- Toutefois Java est plus simple que le langage C++, car les caractéristiques critiques du langage C++ (celles qui sont à l'origine des principales erreurs) ont été supprimées. Cela comprend :
  - Les pointeurs.
  - La surcharge d'opérateurs.
  - L'héritage multiple.

Java *versus* C++ : concepts (2)

De plus,

- ✍ Tout est dynamique : les instances d'une classe sont instanciées dynamiquement.
- ✍ La libération de mémoire est transparente pour l'utilisateur. Il n'est pas nécessaire de spécifier de mécanisme de destruction. La libération de l'espace mémoire est prise en charge un gestionnaire appelé garbage collector $\Rightarrow$ chargé de détecter les objets à détruire.

Notes

- gain de fiabilité (pas de désallocation erronée).
- a un coût (perte en rapidité par rapport au C++).

Outline

Production du binaire

Java *versus* C++ : production du binaire

Une fois achevée la production du logiciel, un choix doit être fait entre fournir le source ou le binaire pour la machine du client.

Généralement, une entreprise souhaite protéger le code source et distribuer le code binaire.

Le code binaire doit donc être portable sur des architectures différentes (processeur, système d'exploitation, etc.).

La portabilité est un point qui distingue JAVA de C++. Java est conçu pour être portable.

Il existe deux grandes manières d'exécuter des programmes :

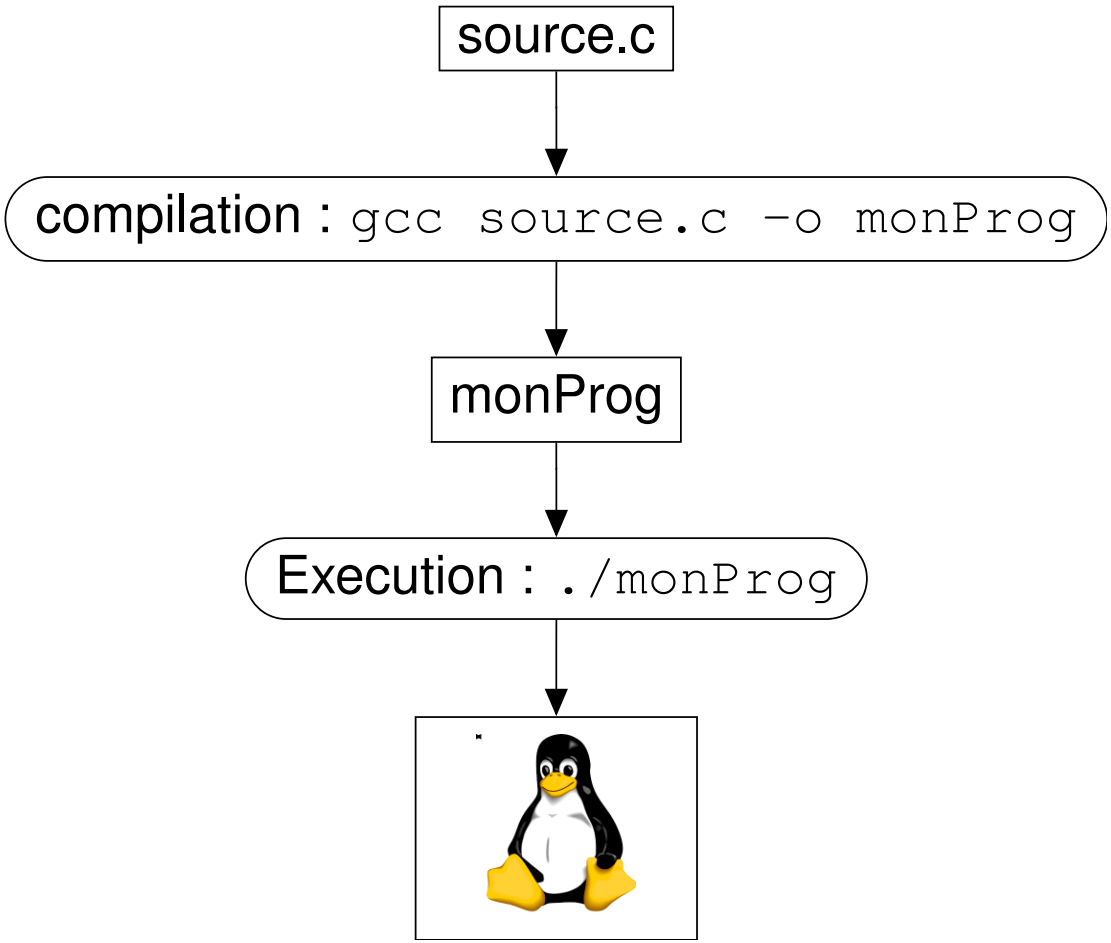
- L'interprétation.
  Chacune des instructions du code source est lue et exécutée par un interpréteur.
  Exemples de langages interprétés : python, matlab, javascript.
- La compilation.
  L'ensemble du programme est traduit en une seule fois en langage machine.
  Exemples de langages compilés : C, C++, Ada.

À l'instar du compilateur `C`, le compilateur `C++` produit du code natif, *i.e.*, qu'il produit un exécutable propre à l'environnement de travail ou le code source est compilé.

On doit donc créer les exécutables pour chaque type d'architecture potentielle des clients.

Java *versus* C++ : Production du binaire

Exemple : chaîne de production du C



Comme pour tout langage interprété (tel python ou matlab), l'interpréteur va lire le fichier source, interpréter et executer chacune des commandes les unes après les autres.

Il faut donc un logiciel interprète pour chacune des architectures. Le comportement du programme va être le même pour toutes les architectures puisqu'il ne dépend que de l'interprète.

En Java, la production du “binaire” est un mix entre interprétation et compilation.

le code source n’est pas traduit directement dans le langage de l’ordinateur.

Il est d’abord traduit dans un langage appelé "bytecode", langage d’une machine virtuelle (JVM – Java Virtual Machine) définie par *Oracle*.

Portabilité

Le *bytecode* généré par le compilateur ne dépend pas de l’architecture de la machine où a été compilé le code source :
les bytecodes produits sur une machine pourront s’exécuter (au travers d’une machine virtuelle) sur des architectures différentes.

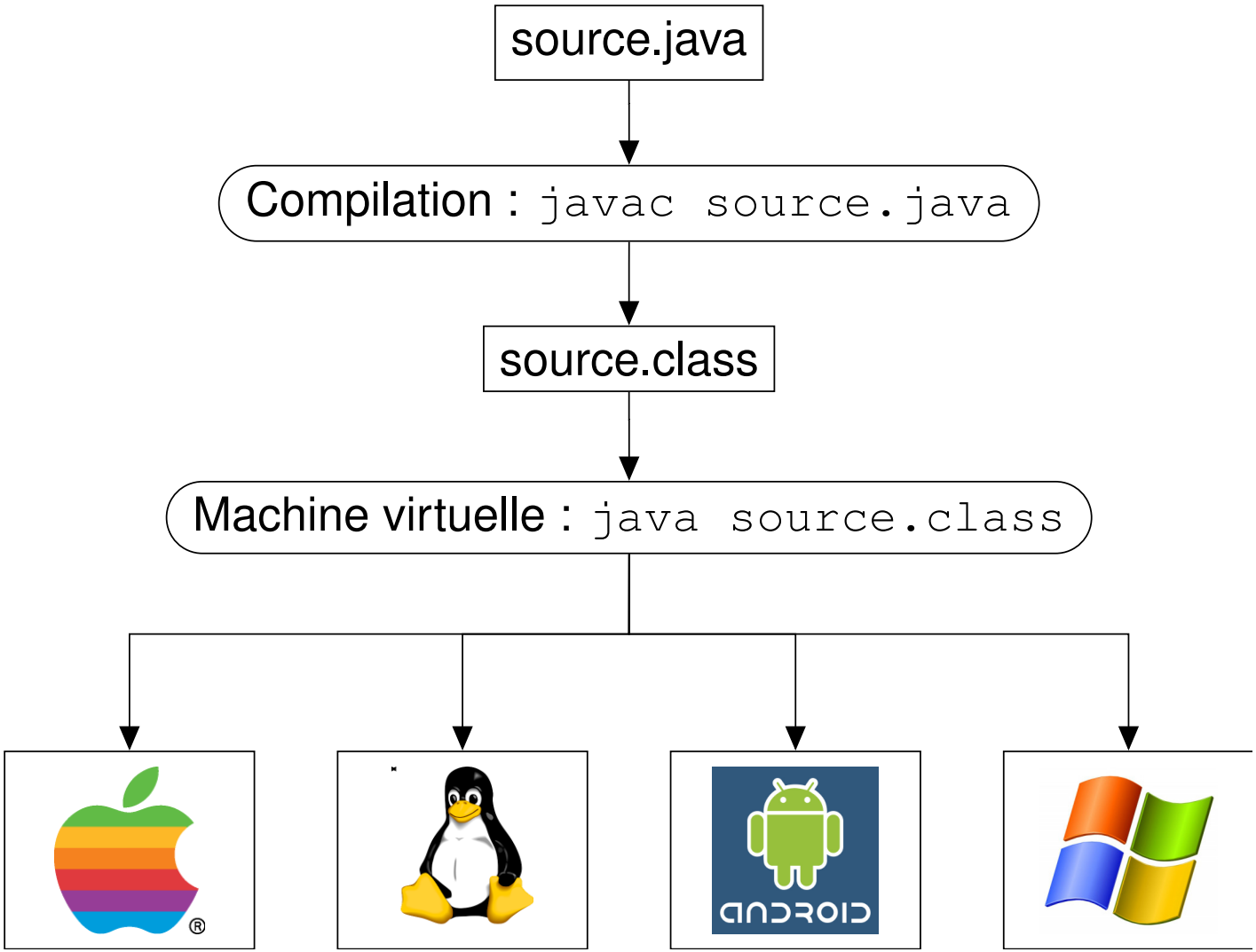
Exécution du bytecode

Le bytecode doit être exécuté par une Machine Virtuelle Java.

Cette JVM n'existe pas. Elle est simulée par un programme qui :

1. lit les instructions (en bytecode) du programme `.class`
2. fait une passe de vérification (type opérande, taille de pile, flot données, variable bien initialisé,...) pour s'assurer qu'il n'y a aucune action dangereuse.
3. fait plusieurs passes d'optimisation du code
4. les traduit dans le langage natif du processeur de l'ordinateur
5. lance leur exécution

Java *versus* C++ : chaîne de production du Java



Coût de la JVM sur les performances.

Les vérifications effectuées sur le bytecode et la compilation du bytecode vers le langage natif du processeur, ralentissent l'exécution des classes Java.

Mais les techniques de compilation à la volée "Just In Time (JIT)" ou "Hotspot" réduisent ce problème : elles permettent de ne traduire qu'une seule fois en code natif les instructions qui sont (souvent pour Hotspot) exécutées.

Java *versus* C++ : en résumé...

Le langage Java est :

- « C-like » : Syntaxe familière aux programmeurs de C.
- Orienté objet : Tout est objet, sauf les types primitifs (entiers, flottants, booléens, ...) !
- Robuste : Typage fort, pas de pointeurs, etc ...
- Code intermédiaire : Le compilateur ne produit que du bytecode indépendant de l'architecture de la machine ou a été compilé le code source.

Note

Java perd en efficacité par rapport à C++ mais gagne en portabilité.

Outline

Comparatif Java C Sharp (C#)

Java *versus* C#

- Conflit entre *Sun* et *Microsoft* en 1996 : certaines spécifications Java sont mal prises en compte par l'environnement Windows.
- Création en 1999 d'un Langage issu du `C++` pour faire tourner le framework `.net`.

Java *versus* C#

Similitudes

- Langage Objet fortement typé.
- Syntaxe proche du C
- Un *garbage collector*
- d'après Gosling C# est une imitation de Java.

Différences

- C# pas portable (ne tourne que sous windows et utilise *Common Language Runtime*)
- C# est normalisé
- Surcharge d'opérateur et arithmétique des pointeurs autorisé en C#

Les environnements JAVA

Plusieurs environnements Java :

- Java SE (appelé avant J2SE) : Java Standard Edition. langage de base.
- Java EE (appelé avant J2EE) : Java Enterprise Edition contient API et des composants pour un environnement d'entreprise (JDBC,EJB).
- Java ME (appelé avant J2ME) : Java Platform Mobile Edition applications pour systèmes embarqués et mobiles.

L'outil de base : le JDK (Java Development Kit) de *Oracle* :

- `http://www.java.com/fr/`.
- gratuit.
- Dernière version : 17 (septembre 2021) LTS.
- comprend de nombreux outils :
  - le compilateur.
  - le compilateur à la volée "JIT".
  - le débogueur.
  - le générateur de documentation.

Documentation des API fournie (ici pour Java 8) :
`http://docs.oracle.com/javase/8/docs/api/`

Environnement de développement

Des environnements de développements (IDE) gratuits

- NetBeans : `http://www.netbeans.org/`
- Eclipse : `http://www.eclipse.org/`
- IntelliJ : `https://www.jetbrains.com/idea/`

Java évolue tout le temps

Java n'est pas un langage normalisé et il continue d'évoluer. Cette évolution se fait en ajoutant de **nouvelles API**, mais aussi en **modifiant la machine virtuelle**.

L'ensemble de ces modifications est géré par le **JCP (Java Community Process ; http ://www.jcp.org/)** dans lequel Oracle continue de tenir une place prépondérante.

Il peut alors être nécessaire d'identifier une version précise du compilateur et/ou de la machine virtuelle : Ça n'est pas simple.

La numérotation des versions :

$$1.0 \longrightarrow 1.1 \longrightarrow \underbrace{1.2 \longrightarrow 1.3 \longrightarrow 1.5}_{\text{Toutes ces versions : Java 2}} \longrightarrow 1.8(\mathit{JavaSE}8)$$

Les versions de Java

JDK 1.0 (1996 - 211 classes et interfaces)
Version initiale.

JDK 1.1 (1997 - 477 classes et interfaces)
Ajoute : classes internes, JavaBeans, JDBC, Java Remote Invocation (RMI).

J2SE 1.2 (1998 - 1 524 classes et interfaces) – Playground
Ajoute : réflection, SWING, compilateur JIT (Just in Time).

J2SE 1.3 (2000 - 1 840 classes et interfaces) – Kestrel
Ajoute : HotSpot JVM, service de nomage (JNDI) et JavaSound.

J2SE 1.4 (2002 - 2 723 classes et interfaces) – Merlin
Ajoute : expressions rationnelles, chaînage d'exception, parser XML (JAXP), sécurité JCE (Java Cryptography Extension) et Java Web Start.

Les versions de Java (suite)

J2SE 5.0 (2004 - 3 270 classes et interfaces) – Tiger
Ajoute : syntaxe à la foreach, enumerations (*enum*), classe Integer, autoboxing/unboxing

Java SE 6 (2006-2013 - 3 777 classes et interfaces) – Mustang
Ajoute : covariance (redéfinition avec modification du type de retour), @overhiding.

Java SE 7 2011 – Nom de code Dolphin
Ajoute : closures.
Première Version 100% open source.

Java SE 8 2014 – Nom de code Kenai
LTS avec Fin version en 2019 pour commercial ou 2030

Java SE 11 2018 –
LTS avec Fin version en 2026

Java SE 17 2021 –
LTS avec Fin version en 2029

Les classes Java.

Classe : déclaration.

Pour déclarer une classe, on utilise le mot-clé **class**.

Exemple

```
public class Logement {
// declaration des membres
}
```

Règle

La première lettre du nom d'une classe doit être une lettre majuscule.

Remarque(s)

Dans le cadre de ce cours, nous ne parlerons que des classes dites publiques (*i.e.*, déclarées **public class**).

Classe : définition.

Définir une classe en `Java`, c'est définir ses membres à savoir :
(1) ses attributs, (2) le(s) constructeur(s), (3) et ses méthodes.

- Il n'est pas nécessaire de spécifier de mécanisme de destruction,
- transparente pour l'utilisateur,
- prise en charge par un gestionnaire appelé **garbage collector** chargé de détecter les instances à détruire.

Conséquences

- gain de fiabilité (pas de désallocation erronée).
- a un coût (perte en rapidité).

Classe : visibilité d'un membre

Pour chaque membre, on doit préciser sa visibilité, c'est-à-dire s'il est un membre privé (déclaré **private**), s'il est un public (déclaré **public**), etc.

public : membre visible par l'ensemble des instances de l'ensemble des classes.

private : membre visible seulement par les instances de la classe.

Remarque(s)

Il existe d'autres types de visibilité pour un membre que nous n'évoquerons pas dans l'immédiat.

Classe : membres d'instance et de classes

Définition

*Un **membre de classe** est un membre **commun à toutes** les instances de la classe. Il existe dès que la classe est définie. Ce, en dehors et indépendamment de toute instanciation.*
*Les membres de classe sont déclarés à l'aide du mot-clé **static**.*

*Un membre qui n'est pas de classe est dit **d'instance**. C'est un membre dont la valeur est propre à l'instance.*

Important

Chaque instance d'une classe possède **son propre exemplaire** d'un attribut d'instance.

Classe : l'accès aux membres de classe

Les membres de classes d'une classe donnée sont communs à toutes les instances de la classe.
Donc l'accès à un membre de classe doit se faire à travers le nom de la classe.
Au moyen de l'opérateur « `.` »,

i.e., pour accéder à un membre de classe, nommé `x`, d'une classe `A`, on écrit `A.x`.

Classe : l'accès aux membres d'instance

Pour accéder à un membre d'instance, d'une instance donnée, on utilise une variable contenant une **référence** vers l'instance (« son adresse »).

Si la variable *a* contient une **référence** vers une instance d'une classe `A`, on accède à un membre d'instance à travers le nom de la référence de l'instance au moyen de l'opérateur « `.` »,

i.e. pour accéder à `x` : un membre d'instance de l' instance `a`, on écrit `a.x`.

Classe : l'accès aux membres d'instance II

Exemple

Avec la classe Logement définie par

```
public Class Logement {
    final public double surface;
    public double prix;
    public String proprietaire;
    private boolean vendu;

   public void changerPrix(double prix){ plus tard}

   public void vendre(String acquereur){ plus tard}
```

L'appel se fait par

```
asnieres92.surface
asnieres92.changerPrix(210000.0d)
paris14.vendre("Djikstra")
```

Classe : accès au membres exemple

Exemple

Quelles sont les instructions qui renvoyent une erreur ci-dessous ?

```
public class A{
   static String nomClasse="La classe est A";
   String nomInstance;

   public A(String noml){nomInstance=noml;}

   public static void main(String[] args){
      A a= new A("son nom est personne");
      System.out.print(A.nomClasse + a.nomClasse);
      System.out.print(A.nomInstance + a.nomInstance);
   }
}
```

Classe : références **Null** et **this**

- ✍ Il existe une référence particulière : la référence **null** qui ne réfère aucune instance.
- ✍ Une instance peut accéder à sa propre référence grâce à la variable **this** (variable en lecture seule).

Classe java type

Donnons le format général d'une classe

```java
// Type + nom (de classe)
public class Point {
  // (1) Attributs
  private double x,y;
```

Classe java type

Donnons le format général d'une classe

```
// Type + nom (de classe)
public class Point {
  // (1) Attributs
  private double x,y;
  // (2) Constructeurs
  public Point(double x, double y) {...}
```

Classe java type

Donnons le format général d'une classe

```
// Type + nom (de classe)
public class Point {
  // (1) Attributs
  private double x,y;
  // (2) Constructeurs
  public Point(double x, double y) {...}
  // (3) Methodes de classe
  static distanceEntre(Point p1,p2) {...}
```

Classe java type

Donnons le format général d'une classe

```
// Type + nom (de classe)
public class Point {
  // (1) Attributs
  private double x,y;
  // (2) Constructeurs
  public Point(double x, double y) {...}
  // (3) Methodes de classe
  static distanceEntre(Point p1,p2) {...}
  // (4) Accesseurs
  public double getX() {...}
```

Classe java type

Donnons le format général d'une classe

```
// Type + nom (de classe)
public class Point {
  // (1) Attributs
  private double x,y;
  // (2) Constructeurs
  public Point(double x, double y) {...}
  // (3) Methodes de classe
  static distanceEntre(Point p1,p2) {...}
  // (4) Accesseurs
  public double getX() {...}
  // (5) Methodes Publiques
  public double distanceAvec(Point p2) {...}
```

Classe java type

Donnons le format général d'une classe

```
// Type + nom (de classe)
public class Point {
  // (1) Attributs
  private double x,y;
  // (2) Constructeurs
  public Point(double x, double y) {...}
  // (3) Methodes de classe
  static distanceEntre(Point p1,p2) {...}
  // (4) Accesseurs
  public double getX() {...}
  // (5) Methodes Publiques
  public double distanceAvec(Point p2) {...}
  // (6) Methodes privees
  private void deplace(double nx, double ny) {...}
```

Classe java type

Donnons le format général d'une classe

```
// Type + nom (de classe)
public class Point {
  // (1) Attributs
  private double x,y;
  // (2) Constructeurs
  public Point(double x, double y) {...}
  // (3) Methodes de classe
  static distanceEntre(Point p1,p2) {...}
  // (4) Accesseurs
  public double getX() {...}
  // (5) Methodes Publiques
  public double distanceAvec(Point p2) {...}
  // (6) Methodes privees
  private void deplace(double nx, double ny) {...}
  // (7) Methodes standards
  public String toString() {...} }
```

Outline

Les attributs Java

Déclaration d'un attribut

En Java, toutes les variables doivent être déclarées avant d'être utilisées (les attributs comme les variable locales des méthodes).

La déclaration des attributs se fait de préférence en debut de classe.

On indique au compilateur :

1. Un ensemble de modificateurs (*facultatif*) : visibilité, constante,. . ..
2. Le type de la variable.
3. Le nom de la variable.

Exemple

```
private double z;
public String str;
```

Les types en Java

Le type d'une variable en Java peut être :

- **Types dits primitifs : int**, **double**, **boolean**, etc.
- **nom d'une classe :** par exemple, les chaînes de caractères sont des instances de la classe **String**.

Remarque(s)

Nous laissons de côté pour l'instant le cas des tableaux sur lesquels nous reviendrons plus tard.

Les types primitifs

Type	Taille (en bits)	Exemple
byte	8	1
short	16	345
int	32	-2
long	64	2L
float	32	3.14f, 2.5e+5
double	64	0.2d, 1.567e-5
boolean	1	true ou false
char	16	'a'

Attention

Un attribut de type primitif n'est pas un objet !

Affecter une valeur à un attribut

L'affectation d'une valeur à une variable est effectuée par l'instruction

```
variable = expression;
```

L'expression est calculée et ensuite la valeur calculée est affectée à la variable

Exemple

```
x = 36.0d;
y = x + 1;
str = "Linus";
```

Initialisation d'un attribut

En Java, une variable doit être initialisée (recevoir une valeur) avant d'être utilisée dans une expression.

S'ils ne sont pas initialisés expressement par le programmeur, les attributs sont automatiquement initialisés par le compilateur. Ils reçoivent alors une valeur par défaut de leur type : 0 pour les **int**, $0.0d$ pour les **double**, **false** pour les **boolean**, **null** pour les `String`, etc.

Remarque(s)

On peut initialiser un attribut lors de sa déclaration.

Exemple

```
private int x = -1;
public String str = "Licence MIAGE";
```

Le modificateur **final**

On peut bloquer la modification de la valeur d'un attribut (en dehors de l'instanciation) à l'aide du mot-clé **final**.

```
final private int x;
```

Remarque(s)

On utilise souvent un attribut de classe déclaré **final** pour définir une constante :

```
final static public double PI = 3.14d;
```

Les types en Java : déclarer les attributs

Exemple

```
public class Logement {
  // les attributs
  final public double surface;
  public double prix;
  public String proprietaire;
  private boolean vendu;

}
```

Remarque(s)

Traditionnellement, le nom d'un attribut ou d'une méthode commence par une lettre minuscule.

Outline

Les constructeurs

Constructeur en Java : définition

Les instances d'une classe sont créées (construits) par une méthode particulière de la classe appelée **constructeur**.

- Le constructeur est une méthode portant le même nom que la classe.
- Toute classe comporte au moins un constructeur. Il peut y avoir plusieurs constructeurs !
- Le constructeur permet d'initialiser les attributs de l'instance créée.

Constructeur en Java : définition

Remarque(s)

Si jamais une classe ne comporte pas de définition d'un constructeur, JAVA lors de la compilation rajoute le constructeur sans paramètres (constructeur par défaut).

Si jamais une classe comporte un constructeur avec paramètres, le constructeur sans paramètres n'est pas ajouté.
NB cela peut poser problème si jamais une classe héritée fait appel au constructeur par défaut.

Si jamais un constructeur est défini sans débuter par un appel de constructeur, alors Java insère un appel du constructeur sans paramètres de la super classe `super()`.

Constructeur en Java : exemple de constructeur

Exemple

```
public class Logement {
  // constructeur
  public Logement(double surface,
                  double prix)
  {
    this.surface = surface;
    this.vendu = false;
    this.prix = prix;
  }
}
```

Constructeur en Java : définition imbriquée

On peut utiliser un constructeur pour définir un autre constructeur.
L'utilisation d'un autre constructeur se fait au moyen du mot-clé **this**.
L'invocation d'un autre constructeur doit être la première instruction.

Exemple

```
public class Logement {
// un deuxieme constructeur
 public Logement(double surface, double prix,
                                    String proprietaire)
   {
     this(surface,prix);
     this.proprietaire = proprietaire;
   }
}
```

Constructeur en Java : invocation

Pour créer une instance d'une classe `A`, on utilise l'opérateur **new** avec un des constructeurs de la classe.
Cela alloue l'espace mémoire nécessaire pour stocker les propriétés de l'objet, crée une référence sur cet espace mémoire et retourne la référence ainsi créée.

À retenir

Tout est dynamique : les instances d'une classe sont instanciées dynamiquement.

Exemple

```
Logement asnieres92;
asnieres92 = new Logement(80.0d,250000.0d,"Linus");
```

Outline

Les méthodes en Java

Les méthodes en Java : syntaxe et définition

La syntaxe pour définir le corps d'une méthode est identique à celle utilisée en `C` pour définir une fonction. Notamment, les noms et la syntaxe

- des instructions (conditionnelles, itératives, etc.) et,
- des opérateurs (arithmétiques, de comparaison, logiques, etc.)

sont les mêmes qu'en `C`.

Les méthodes en Java : exemple de définition

Exemple

```
public class Logement {

  public void changerPrix(double prix)
  {
    if (prix >= 0) {this.prix = prix;}
  }

  public void vendre(String acquereur)
  {
    this.proprietaire = acquereur;
    this.vendu = true;
  }
}
```

Les méthodes en Java : exemple de définition (suite)

Exemple (suite)

```
public class Logement {
  public String annonce()
  {
    String str = "Logement␣(" + this.surface + ")";
    str = str + "␣:␣" + this.prix + "␣euros";
    if (this.vendu) {str = str + "␣–␣vendu";}
    return str;
  }
}
```

Remarque(s)

L'opérateur + permet de concaténer deux chaînes de caractères (instances de la classe `String`).

Conseil

On ne doit pouvoir modifier l'**état** interne d'une instance (*i.e.*, les valeurs de ses attributs) qu'à travers des méthodes d'instance de la classe : les **accesseurs** ou les **mutateurs**

Exemple

La définition de la classe `Logement` permet à n'importe quel utilisateur de modifier les valeurs des attributs `prix` et `proprietaire` sans utiliser les méthodes `changerPrix` et `vendre`.

Les méthodes en java : les accesseurs (suite)

Exemple (suite)

Une meilleure conception de la classe `Logement` est de déclarer ces attributs comme suit

```
private double prix;
private String proprietaire;
```

et de lui ajouter les méthodes

```
public double prix()
{
  return this.prix;
}
public String proprietaire()
{
  return this.proprietaire;
}
```

Les méthodes **static** en java (*i.e.* méthodes de classe)

- Comme les attributs de classe, une méthode de classe existe dès que la classe est définie. Ce, en dehors et indépendamment de toute instanciation.
- Une méthode **static** ne peut utiliser directement aucun attribut ni aucune méthode d'instance de la classe ; une erreur surviendrait à la compilation.
- Une méthode qui manipule, en lecture ou en écriture des attributs d'instance de la classe **doit être** une méthode d'instance.

Les méthodes **static** en java (suite)

Remarque(s)

Une classe qui ne comporterait que des membres **static** ne sert pas à créer des objets ! ! !

Exemple

La classe `System` est une classe ne possédant que des attributs et des méthodes de classes.

- Elle permet d'arrêter brutalement un programme avec méthode de classe `exit` :
  `System.exit(0);`
- Elle permet d'afficher des messages dans la console

```
System.out.println("Hello, World !");
System.out.println(5);
```

et `out` est un attribut de classe de `System` qui est de type `PrintStream`.

La méthode `main` de Java : concept

Il ne suffit pas de définir les attributs, constructeurs et méthodes des différentes classes.
Il faut pouvoir exécuter un programme, les concepteurs de `Java` ont choisi pour cela de particulariser une méthode : **la méthode** `main`.

La méthode main est une méthode de classe publique, qui contient le « programme principal » à exécuter et qui a pour signature :

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

ATTENTION

La méthode *main* ne peut pas retourner d'entier comme en C.

```
public static int main(String[] args)
```

La méthode `main` de java : exemple

Exemple

```
public class Logement {
  // le programme principal
  public static void main(String[] args)
  {
    Logement asnieres92;
    asnieres92 = new Logement(80.0d,2.5e+5,"Linus");
    System.out.println(asnieres92.annonce());
    asnieres92.changerPrix(300000.0d);
    System.out.println(asnieres92.surface);
    System.out.println(asnieres92.prix());
    asnieres92.vendre("Djikstra");
    System.out.println(asnieres92.proprietaire());
  }
}
```

Le polymorphisme en Java : définition

On peut

1. déclarer plusieurs fois la même méthode avec la même signature et des implémentations différentes dans plusieurs classes : **redéfinition**.
2. déclarer plusieurs fois la même méthode avec des signatures différentes et des implémentations différentes dans la même classe : **surcharge**.

Précision

La signature d'une méthode est constituée du nom de la méthode, du nombre et du type de ses paramètres, et du type qu'elle renvoie.

Le polymorphisme en Java : exemple

Exemple

Soit 2 classes qui héritent de AppareilElectrique

```
public class FourElectrique {
  public void demarrer()
  {
    System.out.println("Montee en temperature");
  }
}

public class Ordinateur {
  public void demarrer()
  {
    System.out.println("Demarrage de Linux");
  }
}
```

Le polymorphisme en Java : exemple (suite)

```
public class Rectangle {
  public float largeur;
  public float longueur;

  Rectangle(float largeur,float longueur) {
    this.largeur=largeur;
    this.longueur=longueur;
  }
  public void redimensionner (int facteur) {
    largeur = largeur * facteur;
    longueur = longueur * facteur;
  }
  public void redimensionner (float a,float b) {
    largeur = largeur + a;
    longueur = longueur +b;
  }}
```

Le polymorphisme en Java : important

Important

On peut définir autant de signatures que l'on veut de la même méthode (faire varier le nombre d'arguments, changer le type des arguments, changer le type de retour). Il y a néanmoins une restriction : si l'on définit plusieurs fois la même méthode dans une classe, la différence entre deux signatures ne doit pas seulement être le type de retour, *i.e.*, par exemple, on ne peut pas écrire dans la même classe :

```
public int f(int a) { ... }

public float f(int a) { ... }
```

Compilation et exécution (1)

Etape 1 : on crée un fichier texte dont le nom est le nom de la classe auquel on rajoute l'extension `.java`.

Exemple

On écrit la définition de la classe `Logement` dans un fichier texte nommé `Logement.java`.

Important

Un fichier ne peut contenir qu'une seule classe publique.

Compilation et exécution (2)

Etape 2 : on compile le programme Java à l'aide de la commande `javac`.

HelloWorld.java

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

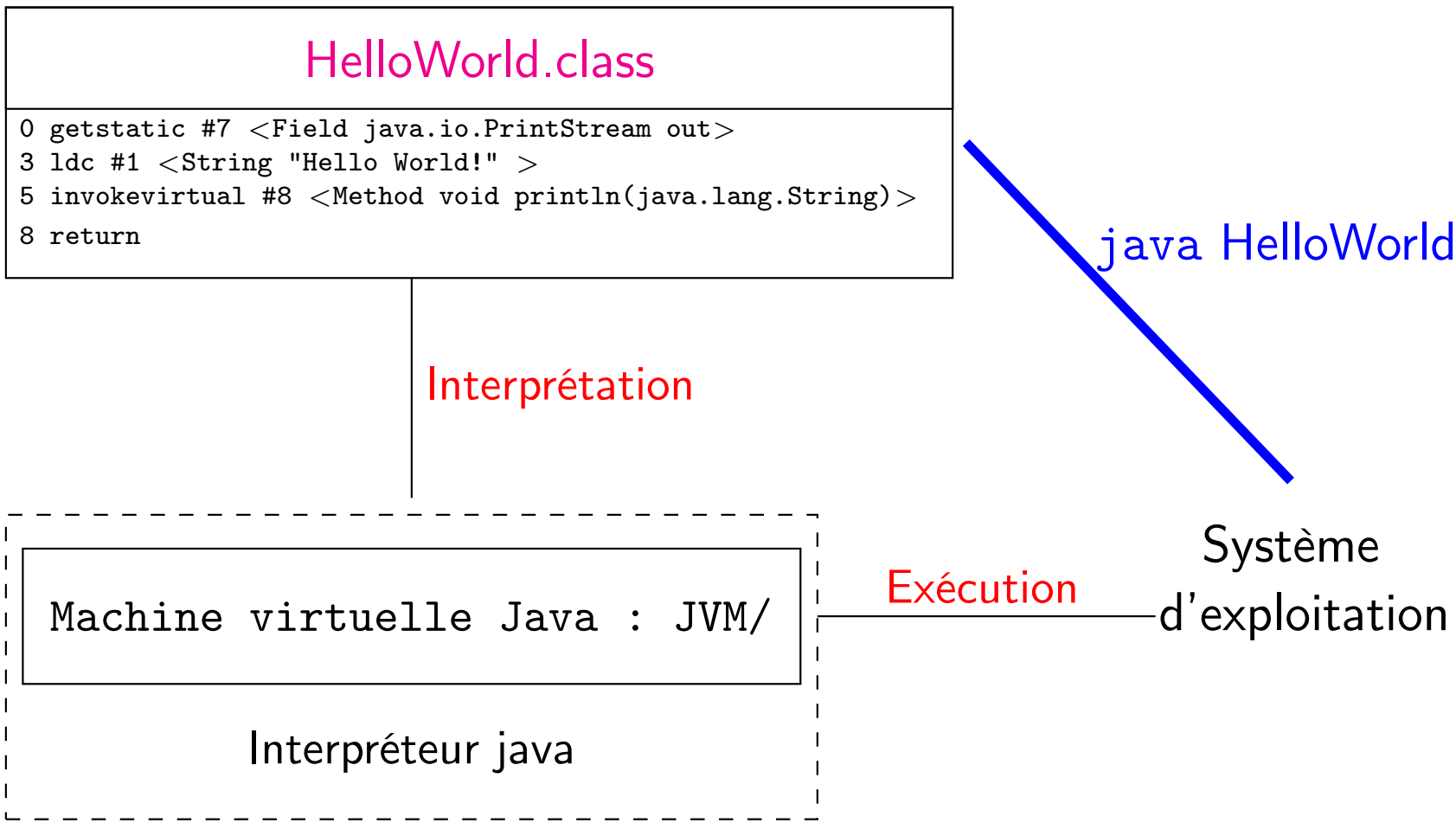
`javac HelloWorld.java`

HelloWorld.class

```
0 getstatic #7 <Field java.io.PrintStream out>
3 ldc #1 <String "Hello World!" >
5 invokevirtual #8 <Method void println(java.lang.String)>
8 return
```

Compilation et exécution (3)

Etape 3 : le JIT de la machine virtuelle compile à la volée le bytecode produit (c'est-à-dire on éxecute la méthode `main`) à l'aide la commande `java`.



Compilation et interprétation (4)

Important **Bien positionner les variables d'environnement**

`PATH` : doit contenir le répertoire du compilateur et de la machine virtuelle.

`CLASSPATH` : indique le chemin ou se trouve les classes (par défaut, le répertoire courant) sinon vous aurez le message d'erreur

```
Exception in thread "main"
java.lang.NoClassDefFoundError: HelloWorld
```

Outline

Classes utiles : String, tableau et classes enveloppes

Les chaînes de caractères : le type String (1)

Les chaînes de caractères sont des instances de la classe `String`. L'opérateur de concaténation des chaînes de caractères est l'opérateur +.

Attention

Pour comparer deux chaînes de caractères, on utilise la méthode `equals` (ou `equalsIgnoreCase`) de la classe `String`.

```
String str1 = ....;
String str2 = ....;

if (str1.equals(str2)) {...} else {...}
```

Les chaînes de caractères : le type String (2)

La classe `String` offre de nombreuses autres possibilités :

- `length()` renvoie la longueur de la chaîne de caractères.
- `toUpperCase` et `toLowerCase` permettent, respectivement, de mettre la chaîne de caractères en lettres majuscules et minuscules.
- `int indexOf (char ch)` renvoie l'indice de la première occurence du caractère `ch` dans la chaîne de caractères.
- `String substring(int beginIndex,int endIndex)` qui retourne la sous-chaîne constitué des caractères d'indice `beginIndex` à `endIndex - 1`.
- etc ...

Important

Un tableau est un objet !

Deux étapes :

1. Déclaration : déterminer le type de ses éléments.
2. Dimensionnement : déterminer la taille du tableau (c'est-à-dire le nombre d'éléments).

Les tableaux unidimensionnels

La déclaration d'un tableau précise simplement le type des éléments du tableau :

```
int [] tableau;
```

La dimension du tableau est précisé lors de son instanciation

```
// cree un tableau pouvant contenir 50 entiers
tableau = new int [50];
```

- La taille d'un tableau ne peut plus être modifiée par la suite.
- Dimension du tableau : `tableau.length`

Les tableaux unidimensionnels

- On accède à l'élément d'indice `i` du tableau en écrivant `tableau[i]`.
- Les indices commencent à 0.
- `Java` vérifie automatiquement l'indice lors de l'accès.

Remarque(s)

On peut aussi donner explicitement la liste des éléments :d'un tableau au moment de son instanciation :

```
int [] tableau = {1,2,3};
String [] mots = {"Licence","MIAGE","POO"};
```

Remarque(s)

Si l'on déclare un tableau dont les éléments sont des références de type `logement` :

```
Logement[] ville;
```

Alors, la ligne de commande

```
ville = new Logement[100];
```

instancie un tableau de 100 références initialisées à **null**. Si l'on veut qu'une case contienne une référence vers une instance de la classe `Logement`, on doit instancier une instance de la classe `Logement` et écrire la référence de l'instance dans la case :

```
ville[50] = new Logement(67.0d,5e+5);
```

Les tableaux unidimensionnels

L'argument `args` de la méthode `main` est un tableau dont les éléments sont des références vers des chaînes de caractères. Il permet de passer à la méthode main des paramètres en ligne de commande

Exemple

Si l'on écrit

```
java Programme a 3 5.7 true
```

alors `args` est un tableau de dimension 4 contenant les références vers les chaînes de caractères "a", "3", "5.7" et "true". Les références sont mises dans le même ordre que sur la ligne de commande. Par exemple, `args[1]` est une référence vers la chaîne de caractères "3".

Les classes enveloppes des types primitifs

A chaque type primitif est associé une classe qu'on appelle **classe enveloppe** de ce type primitif.

Exemple

La classe enveloppe du type **int** est la classe `Integer`.

```
Integer entier = new Integer(56);
```

Chaque classe enveloppe d'un type primitif possède une méthode pour convertir une instance de la classe en une valeur du type primitif associé.

Exemple

```
int i = entier.intValue();
```

Les classes enveloppes des types primitifs

Pour les autres types primitifs, les noms des classes enveloppes sont :

Type primitif	classe enveloppe
short	Short
long	Long
byte	Byte
float	Float
double	Double
boolean	Boolean
char	Character

Les classes enveloppes des types primitifs

Elles permettent notamment de convertir une chaîne de caractères en un entier, un flottant, un booléen, etc ...

Exemple

```
Integer.parseInt("6");
Double.parseDouble("6.89");
Boolean.parseBoolean("false");
```