

A decorative graphic on the left side of the slide, consisting of a network of white lines and small circles on a dark blue background, resembling a circuit board or a stylized tree structure.

# COURS JAVA

CHARLES 'ARYS' YAICHE

# OBJECTIF DU COURS

- Comprendre & maîtriser la syntaxe JAVA
- Maîtrise de la POO avec JAVA
- Maîtrise des key feature de JAVA
- Maîtrise des outils & framework JAVA
- Maîtrise de l'utilisation des principaux design pattern

# QU'EST CE QUE JAVA



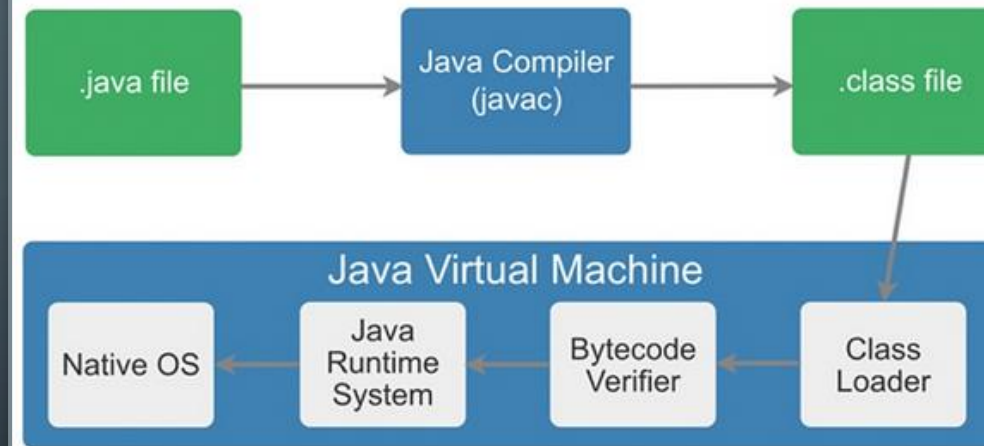
- Orienté objet
- Portabilité
- Gestion des IO simple
- Performance
- Simple d'utilisation

# JVM & BYTECODE



- Une fois écrit, le code tourne partout
  - Programme compilé tourne sur une machine virtuel portable
  - Code mobile

## What is Java bytecode?

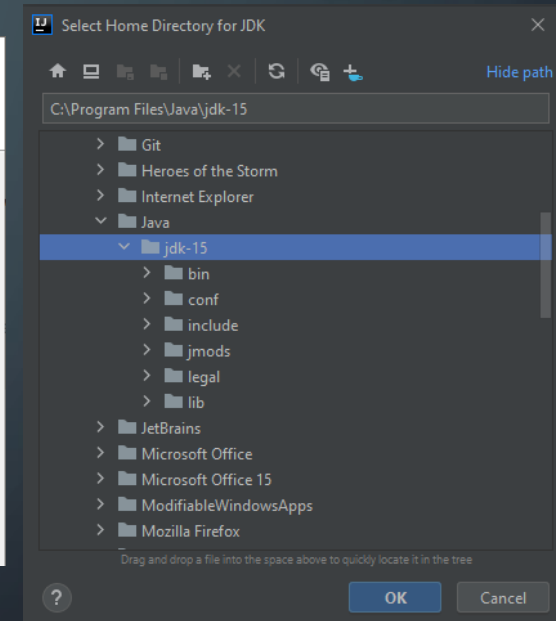
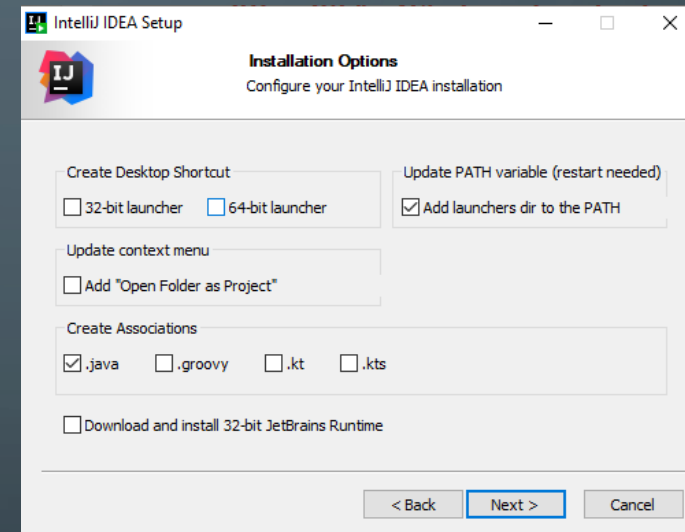


Source: <http://www.techlila.com/write-programs-linux/>

# LE BON IDE & INSTALLATION DU SDK



- IntelliJ
- Téléchargement de java sur ORACLE



Windows x64 Installer

159.68 MB

 [jdk-15\\_windows-x64\\_bin.exe](#)

# BASES



- Syntaxe similaire au C/C++
- Type primitif : int, float, boolean
- Objet type : Integer, Float, Boolean
- Exception mécanisme
- Scope { }

# HELLO WORLD



```
public class Hello {  
    public static void main(final String... args) {  
        System.out.println("Hello, World");  
    }  
}
```

# MÉTHODE



- Il y a uniquement des méthode en java
- Les arguments sont passé :
  - Par valeur (pour les type primitif)
  - Par référence (pour les objets)
- Il n'y a pas de pointeurs en java (seulement des références)
- Attention au comparaison == (pas toujours valide pour les objets)



# CLASSES



- Le java est un langage orienté objets, tout est classe et objets.
- Une classe en java est un package de :
  - Propriété (attribut)
  - Méthode
  - Code statique (à éviter)



# LES PROPRIÉTÉ (ATTRIBUTS)

- Possèdent (dans l'ordre):
  - Une annotation\*\*
  - Une visibilité (public, private, protected)\*\*
  - Une ancre (anchor) : static, « default »\*\*
  - Un type
  - Un nom
  - Une valeur\*\*

\*\* : peut être optionnel

```
helloWorld.java x
5 public class helloWorld {
4     int d;
3     int a = 1;
2     public int b = 2;
1     public static int c = 3;
6 }
```

# LES MÉTHODE



- Possèdent (dans l'ordre):
  - Une annotation\*\*
  - Une visibilité (public, private, protected)\*\*
  - Une ancre (anchor) : static, « défaut »\*\*
  - Un type de retour
  - Un nom
  - Une déclaration d'argument\*\*
  - Un corps (body)

\*\* : peut être optionnel

```
@Transactional
protected String commit(final String value) {
    // Body: do stuff
    return value + "--";
}
```

# POINT D'ENTRÉE



- il faut au moins une méthode main dans au moins une classe
- On ne peut avoir qu'une méthode main par classe
- On entre dans un programme java via une classe

# LES COMMENTAIRES



```
// Les commentaires sur une seule ligne commencent par //
```

```
/*
```

```
Les commentaires sur plusieurs lignes ressemblent à ceci.
```

```
*/
```

```
/**
```

```
 * Les commentaires de la JavaDoc ressemblent à ceci. Ils sont utilisés pour
```

```
 * décrire la classe et ses différents attributs.
```

```
 * Attributs principaux:
```

```
 *
```

```
 * @author      Nom (et information de contact comme l'email) de(s) auteur(s).
```

```
 * @version     Version actuelle du programme.
```

```
 * @since       Date à laquelle cette partie du programme a été ajouté.
```

```
 * @param       Décrit les différents paramètres pour d'une méthode.
```

```
 * @return      Décrit le retour de la méthode.
```

```
 * @deprecated  Indique si le code est déprécié ou ne doit plus être utilisé.
```

```
 * @see         Lien vers une autre partie de la documentation.
```

```
*/
```

```
// Termine la classe PointInt qui se trouve dans le package java.util
```



# LES TYPES DE BASE

- Il existe 8 type de base en java

Type de base	Type	Nombre de bits	Valeurs possible
boolean	Booléan	32	true et false
byte	entier	8	signées
short	entier	16	signées
int	entier	32	signées
long	entier	64	signées
float	virgule flottante	32	non signées
double	virgule flottante	64	non signées
char	caractère	16	Unicode



# LES VARIABLES

- Une variable se déclare avec un type, un identifiant et une valeur d'initialisation optionnel, un point virgule

`<Type> <identifiant> = <valeur>;`

Ou

`<Type> <identifiant>;` (attention avant d'utiliser une variable non initialisé)

*Tips : on peut déclarer plusieurs variable à la suite*

```
// <names>  
int fooInt1, fooInt2, fooInt3;
```



# LES STRUCTURE DE CONTRÔLE : CONDITIONNELLE



```
// Les instructions conditionnelle sont identiques aux langage C
int j = 10;
if (j == 10) {
    System.out.println("I get printed");
} else if (j > 10) {
    System.out.println("I don't");
} else {
    System.out.println("I also don't");
}
```





# LES STRUCTURE DE CONTRÔLE : WHILE

```
// Boucle while
int fooWhile = 0;
while(fooWhile < 100) {
    System.out.println(fooWhile);
    // Incrémente le compteur
    // Itéré 100 fois, fooWhile 0,1,2...99
    fooWhile++;
}
System.out.println("fooWhile Value: " + fooWhile);
```



# LES STRUCTURE DE CONTRÔLE : DO... WHILE

```
// Boucle do-while
int fooDoWhile = 0;
do {
    System.out.println(fooDoWhile);
    // Incrémente le compteur
    // Itéré 99 fois, fooDoWhile 0->99
    fooDoWhile++;
} while(fooDoWhile < 100);
System.out.println("fooDoWhile Value: " + fooDoWhile);
```



# LES STRUCTURE DE CONTRÔLE : FOR

```
// Boucle for
// De la forme for(<start_statement>; <conditional>; <step>)
for (int fooFor = 0; fooFor < 10; fooFor++) {
    System.out.println(fooFor);
    // Itéré 10 fois, fooFor 0->9
}
System.out.println("fooFor Value: " + fooFor);

// Fin d'une boucle for avec un label
outer:
for (int i = 0; i < 10; i++) {
    for (int j = 0; j < 10; j++) {
        if (i == 5 && j == 5) {
            break outer;
            // termine l'itération de la boucle englobante avec le label outer
        }
    }
}
```



# LES STRUCTURE DE CONTRÔLE : FOREACH

```
// Boucle for-each
// La boucle for est également capable d'itérer aussi bien sur un
// tableau que sur des objets qui implémentent l'interface Iterable.
int[] fooList = {1, 2, 3, 4, 5, 6, 7, 8, 9};
// De la forme: for (<object> : <iterable>)
// Lu comme: "Pour chaque élément du tableau"
// note: le type doit correspondre à celui de l'objet itérable
for (int bar : fooList) {
    System.out.println(bar);
    //Itère 9 fois et affiche les chiffres de 1 à 9
}
```



# LES STRUCTURE DE CONTRÔLE : SWITCH-CASE

```
// Le switch-case
// Un switch fonctionne avec les données de type byte, short, char et
// int.
// On peut également utiliser le type Enum, la classe String et les
// classes spéciales qui englobent les types primitifs (Character, Byte,
// Short et Integer).
// Depuis Java 7, on peut utiliser le type String.
int month = 3;
String monthString;
switch (month) {
    case 1: monthString = "January";
        break;
    case 2: monthString = "February";
        break;
    case 3: monthString = "March";
        break;
    default: monthString = "Some other month";
        break;
}
System.out.println("Switch Case Result: " + monthString);
```



# BOXING

- Tout type primitif à une version Objet
- L'{un}Boxing (transformation primitive  $\Leftrightarrow$  version objet) est en généralement fait automatiquement

```
public Integer convert(int i) {  
    i = new Integer(5); // unboxing  
    Integer j = 5;      // boxing  
  
    return j;  
}
```

# EXEMPLE DE CLASSE



```
// A public class
public class MyClass {

    // Some properties
    private int someInt;
    private String aString;

    // Default Constructor
    public MyClass() {
        someInt = 42;
        aString = "42";
    }

    // Overloaded constructor
    public MyClass(final int x, final String str) {
        someInt = x;
        aString = str;
    }

    //Some methods
    @Override
    public String toString() {
        return "A MyClass instance";
    }
}
```



# EXEMPLE DE CLASSE



```
// A public class
public class MyClass {

    // Some properties
    private int someInt;
    private String aString;

    // Default Constructor
    public MyClass() {
        this(42, "42"); // Smarter use of constructors.
    }

    // Overloaded constructor
    public MyClass(final int x, final String str) {
        someInt = x;
        aString = str;
    }

    //Some methods
    @Override
    public String toString() {
        return "A MyClass instance";
    }
}
```



# CYCLE DE VIE D'UNE VARIABLE

- La pile (type primitif)
- Le tas (objets)



# SOURCE

- <https://learnxinyminutes.com/docs/java/>
- <http://gaetan.dussaux.free.fr/cours/java/7.htm>
- <http://blog.paumard.org/cours/java/chap02-programmer-types-objets.html>