B0 - HUB

B-INN-000

Discover Java

Oriented Object Programming

1.10

Discover Java

language: java
compilation: javac, automated compilation with IDE

- The totality of your source files, except all useless files (binary, temp files, obj files,...), must be included in your delivery.
- All the bonus files (including a potential specific Makefile) should be in a directory named *bonus*.
- Error messages have to be written on the error output, and the program should then exit with the 84 error code (0 if there is no error).

Découverte du Java

Introduction

- Qu'est-ce que le java ?
  - Java est un langage de programmation orienté objet, typé, "compilé" qui fonctionne grâce à une **J**ava **V**irtual **M**achine (**JVM**)
- Qu'est-ce que la **JVM** ?
  - La JVM est une machine virtuel qui permet l'interprétation du code java compilé (**.class**), et aussi des archives java (**.jar**)
  - C'est l'un des avantages du java ! Grâce à la **JVM** le code est très portable, il peut être exécuté partout tant qu'une **JVM** est présente !
  - D'autres langage peuvent aussi fonctionner sur la **JVM** c'est le cas du **Scala** et du **Kotlin**

INSTALLATION

ETAPE 1

Si vous avez le dump sur votre portable vous devriez avoir le **J**ava **D**eveloppement **K**it (**JDK**) d'installé par défaut en version 11.
Pour savoir si vous avez java :

```
Terminal
~/B-INN-000> java -version
java version "1.8.0_181"
Java(TM) SE Runtime Environment (build 1.8.0_181-b13)
Java HotSpot(TM) 64-Bit Server VM (build 25.181-b13, mixed mode)
```

Pour savoir si vous avez le **JDK**

```
Terminal
~/B-INN-000> javac -version
javac 1.8.0_181
```

ETAPE 2

Si vous n'avez pas java, il vous suffit d'installer **openjdk** la version 8 de préférence
Installez le tout simplement via dnf

```
Terminal
~/B-INN-000> sudo dnf install -y java-1.8.0-openjdk-devel.x86_64
java-1.8.0-openjdk.x86_64
```

ETAPE 3

Installation d'un **I**ntegrated **D**evelopment **E**nvironment

- Intellij Idea Community
- Eclipse

Installez Intellij Idea de préférence :)

L'auto complétion d'Intellij Idea sera très utile ! (CTRL + ESPACE) pour forcer l'autocomplétion

ETAPE 4

Création d'un projet avec Intellij Idea

- File => New Project => Java => Next => Next => "Workshop Java" => Finish

Le Java ! La théorie

Les modifieurs de visibilité et d'accès

En java on peut changer la visibilité d'une variable, d'une classe, d'une fonction, pour cela il faut utiliser des **keyword** spécifique

- Les modifieurs de visibilité
  - `private` personne ne peut modifier/accéder à cette variable à part la classe actuelle
  - `protected` uniquement les classes enfants, ou appartenant au même package peuvent modifier/accéder à cette variable
  - `public` tout le monde peut modifier/accéder à cette variable

Par défaut une variable, classe ou fonction déclaré sans modifieur de visibilité est en privé

- Les modifieurs d'accéssibilité
  - `static` une variable ou une fonction statique peut être accédée sans instancier la classe, elle est aussi unique à cette classe
  - `final` pour faire simple, c'est l'équivalent du `const` en C

Les variables statiques sont toutes les mêmes peu importe l'instance de la classe

Les types

Vu que le java est basé sur le C++ et donc le C, les types restent à peu près les mêmes, pour certains ils sont améliorés ou ajoutés

`boolean` permet de stocker les états `true` et `false` (comme bool en C)
`byte` permet de manipuler des entiers codés sur 1 byte (comme char)
`enum` les enums en java sont beaucoup plus complet que ceux en C, vu que ce sont des objets on peut y assigner plus qu'un simple Integer comme valeur
`double`, `int`, `long`, `float`, `short` restent les mêmes

LE POLYMORPHISME - L'HéRITAGE

Un des avantages de la Programmation Orienté Objet est l'héritage prennons un exemple :

```
public class Fruit {

    private String name;

    public Fruit(String name) { //Constructor of Fruit class
        this.name = name; //this.variable use variable "name" of class instead of
            variable "name" in the constructor
    }

    public String getName() { //a getter, return the private value name | The only
        way to get a private variable
        return name;
    }

    public void setName(String name) {//a setter, set the private value name to the
        value in param | The only way to set a private variable
        this.name = name;
    }

}

public class Apple extends Fruit { //only one extends is authorized

    public Apple() {
        super("apple"); //call Fruit class constructor with "apple" as name
    }

}

public class Main {

    public static void main(String[] args) {
        Fruit fruit = new Fruit("orange");
        Apple apple = new Apple();
        Fruit apple2 = new Apple(); //Fruit is parent class of Apple so you can init
            Fruit with new Apple()
        System.out.println(fruit.getName()); //method in Fruit class
        //display "orange"
        System.out.println(apple.getName()); //method declared in Fruit class, all
            method in Fruit class who is not private exists in Apple
        //display "apple"
    }

}
```

C'est ce qu'on appelle l'héritage, la classe enfant hérite de toutes les fonctions, variables qui ne sont pas en privé de la classe parent
Comme dans l'exemple la classe Apple hérite des méthodes getName() et setName() de la classe parent Fruit

L'INSTANCIATION D'OBJET

En java pour créer un nouvel objet il suffit d'ajouter le **keyword** `new` devant la classe

```
public class Main {

    public static void main(String[] args) {
        Object object = new Object(); //initialisation d'une variable "object" qui a
            pour type "Object"
        String str = new String("Hey"); //initialisation d'une variable "str" qui a
            pour type "String"
        String coucou = "coucou"; //initialisation d'une string, c'est ce qu'on
            utilise plutot que "new String()"
    }

}
```

LE POLYMORPHISME - LA SURCHARGE

Une des fonctionnalités bien pratique en Java est la surcharge, elle permet de définir plusieurs méthodes avec le même nom, mais pas les mêmes paramètres
L'exemple le plus simple pour comprendre ce principe est avec la méthode de print

```
public class Main {

    public static void main(String[] args) {
        System.out.println("coucou"); //type of parameter is String
        System.out.println(1); //type of parameter is Integer
        System.out.println(0.0); //type of parameter is Double
        System.out.println(0.0f); //type of parameter is Float
        System.out.println(5 < 0); //type of parameter is Boolean
        System.out.println(new Object()); //type of parameter is Object, print a
            Object call the method toString() of the object
    }

}
```

LE POLYMORPHISME - LA RéDéFINITION

En java on peut redéfinir des méthodes de la classe parente, par défaut la classe parente est java.lang.Object même si **extends Object** n'est pas présent, les méthodes `toString()`, `hashCode()`, `equals()` existent par défaut

```
public class OverridableClass {

    public static void main(String[] args) {
        System.out.println(this.toString()); //print "Hi I'm an Overridable method"
    }

    @Override //This annotation is not necessary but it's preferable to use it to
        clarify the code
    public String toString() {
        return "Hi I'm an Overridable method";
    }

}
```

Le Java ! La pratique :3

Création d'un package

Cliquez droit sur le dossier `src` => New => Package, généralement un package se construit de la sorte : `re.alwyn974.workshop.sub`

`re.alwyn974` est le nom de domaine inversé, ça peut être par exemple `com.github.username`
`workshop` est le nom du projet
`sub` est un exemple des sous package qu'on peut créer

Créez votre package de la sorte `com.gitub.username.workshop`

La création de package n'est pas obligatoire pour faire du Java, mais elle est nécessaire pour structurer son code

Création d'une classe

Vu que le java est orientée objet nous devons créer des **Objets** aka les **classes**
Exemple : création d'un simple "Hello World"
Cliquez droit sur le package crée => New => Java Class => Main (*bien chosir class*)

La convention pour nommer les classes en java est de toujours commencer par une Majuscule

Ajouter le main du programme :

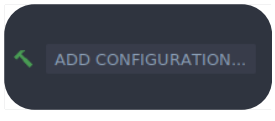
```java
package re.alwyn974.workshop;

public class Main {

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

}
```

Lançons le programme maintenant !
Vous devez ajouter une configuration de lancement dans Intellij Idea, pour cela il suffit de cliquer ici



Cliquez ensuite sur le + => Application
Mettez en nom Workshop
Dans la partie "Build and run"
Mettez dans "Main class" `votre.package.Main`
Cliquez ensuite sur ok
Ensuite il vous suffit de cliquer sur le bouton de lancement pour lancer :x

EXERCICE 1

Vous allez devoir créer une classe “Entity”, elle devra posséder :

- Un constructeur qui initialise name et x,y,z
- `String name` accessible uniquement via getter/setter
- `double x,y,z` accessible uniquement via la classe enfant (ou package) et par getter/setter

EXERCICE 2

Vous allez devoir créer une classe “EntityLiving” qui à pour parent la classe “Entity”, elle devra posséder :

- `double life` accessible uniquement via getter/setter
- `boolean isAlive()` une fonction publique pour dire si l’entitée est en vie (life > 0)
- `String sentence` une variable publique (créer les getter/setter quand même)
- `void displaySentence()` une fonction qui affichera la sentence

Pour approfondir votre apprentissage

Beaucoup de site propose des cours sur le java, en voici quelques un :
- Jmdoudoux.fr
- Koor.fr