Mis-a-niveau-JAVA

Hossein Khani

Dauphine
UNIVERSITÉ PARIS

JAVA

1. Java est un langage multiplateforme, orienté objet. Il fait partie des langages de programmation les plus utilisés.

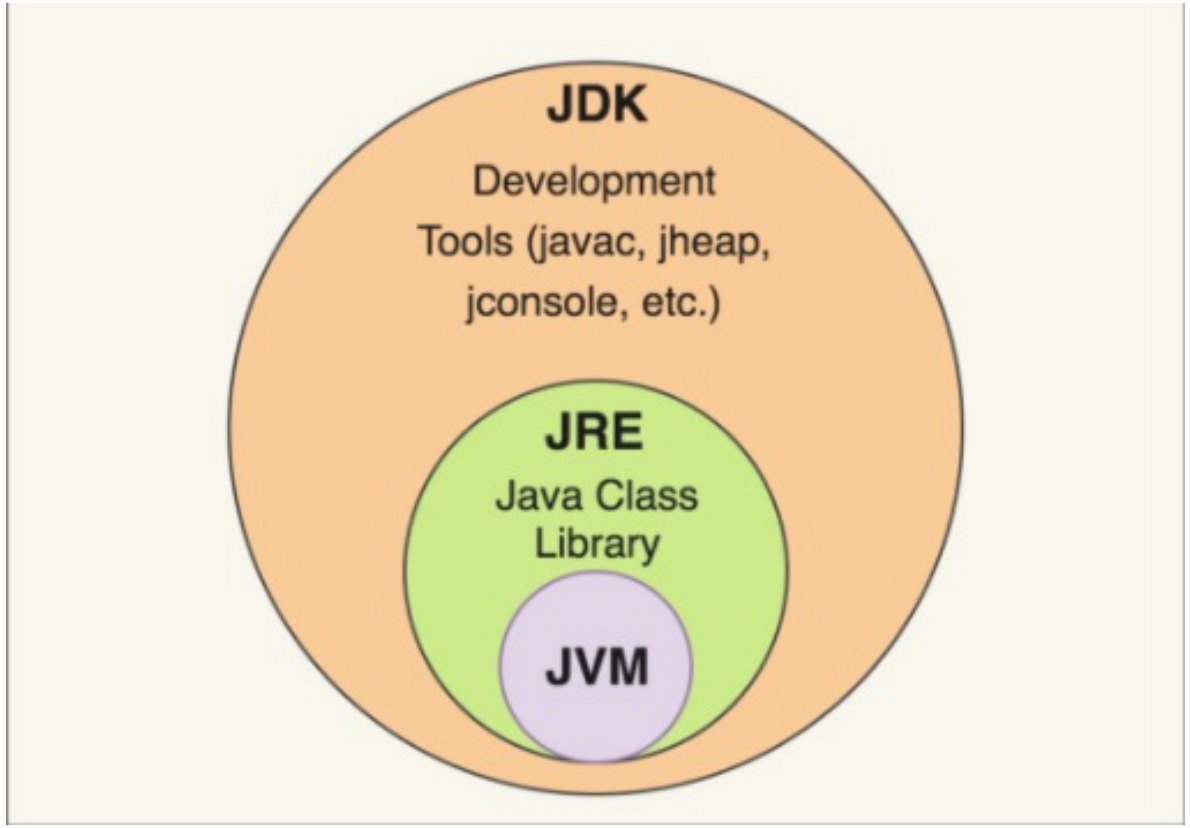
2. JAVA est un langage indépendant (write ones, execute many times)

3. JAVA- plateforme: Engine (Virtual Machine), compilateur, ensemble de bibliothèques pour gérer les dépendances.

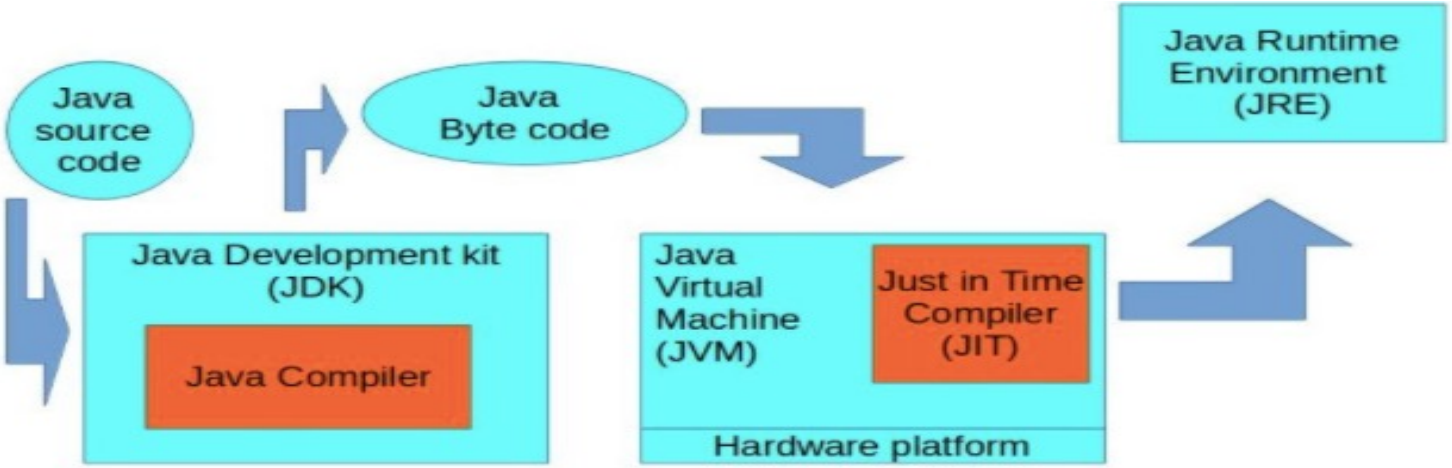
JVM: Java Virtual Machine

JRE: Java Runtime Environment

JDK: Java Development Kit



Comment cela fonctionne?



Editions de JAVA

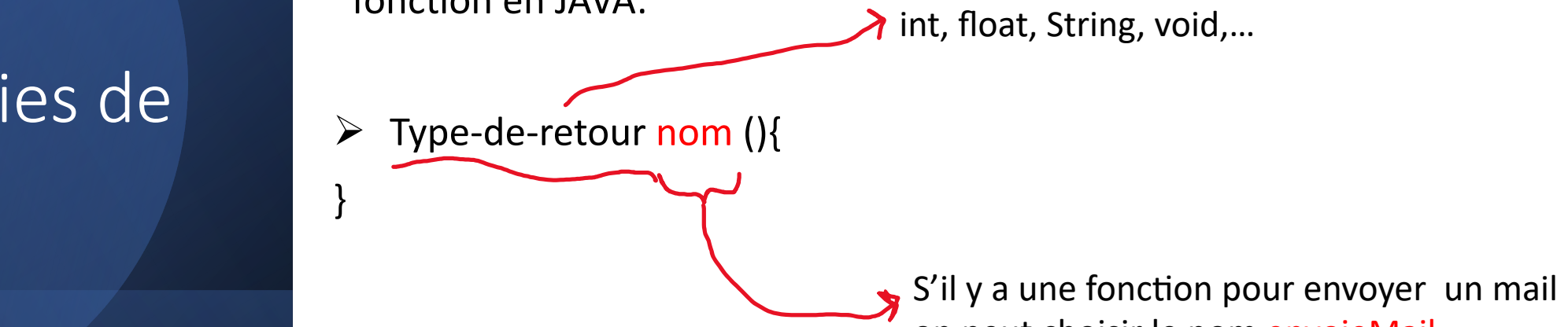
- JAVA SE: Java Standard Edition ; JDK = Java SE Development Kit

- JAVA EE: Enterprise Edition qui ajoute les API pour écrire des applications installées sur les serveurs dans des applications distribuées : servlet, JSP, JSF, EJB,...

- JAVA ME: Micro Edition, version pour écrire des programmes embarqués (carte à puce/Java carde, téléphone portable,…)

Anatomies de JAVA

- Les plus petits bloc de la programmation JAVA sont les fonctions.
- Comment Ecrire une fonction en JAVA:



Anatomie de JAVA

- Chaque programme de JAVA doit contenir une fonction qui s'appelle main.

- main c'est un point d'entrée pour exécution de programme (Quand on execute un programme, c'est la function main qui va être appelée).

Anatomie de JAVA

- Les functions ne sont pas disponibles toutes seules.
- Il faut qu'ils faitent partie un conteneur qui s'appelle classe.
- Les classes organisent notre programme (Comme les produits qui sont organisés dans un supermarché).
- Chaque class en JAVA contient des function connexes.
- Les classes en JAVA ont catégorisé par les fichiers et et les fichiers relatifs sont placés dans un package.

Anatomies de JAVA

- On désigne une fonction appartenant à une classe comme une méthode.

- Chaque classe et chaque méthode en JAVA devraient avoir le modificateur d'accès:

Est-ce que les autres classes et méthodes peuvent accéder au contenu de la classe ou de la méthode?

Notre Environnement de Travail

```
(base) hosseins-Air:~ hosseinkhani$ java -version
java version "12.0.2" 2019-07-16
Java(TM) SE Runtime Environment (build 12.0.2+10)
Java HotSpot(TM) 64-Bit Server VM (build 12.0.2+10, mixed mode, sharing)
(base) hosseins-Air:~ hosseinkhani$
```

- JRE 12 SE -- ORACLE website.
- JDK 12 SE – ORACLE website.
- IDE : Eclipse Eclipse JAVA IDE.

À vous:

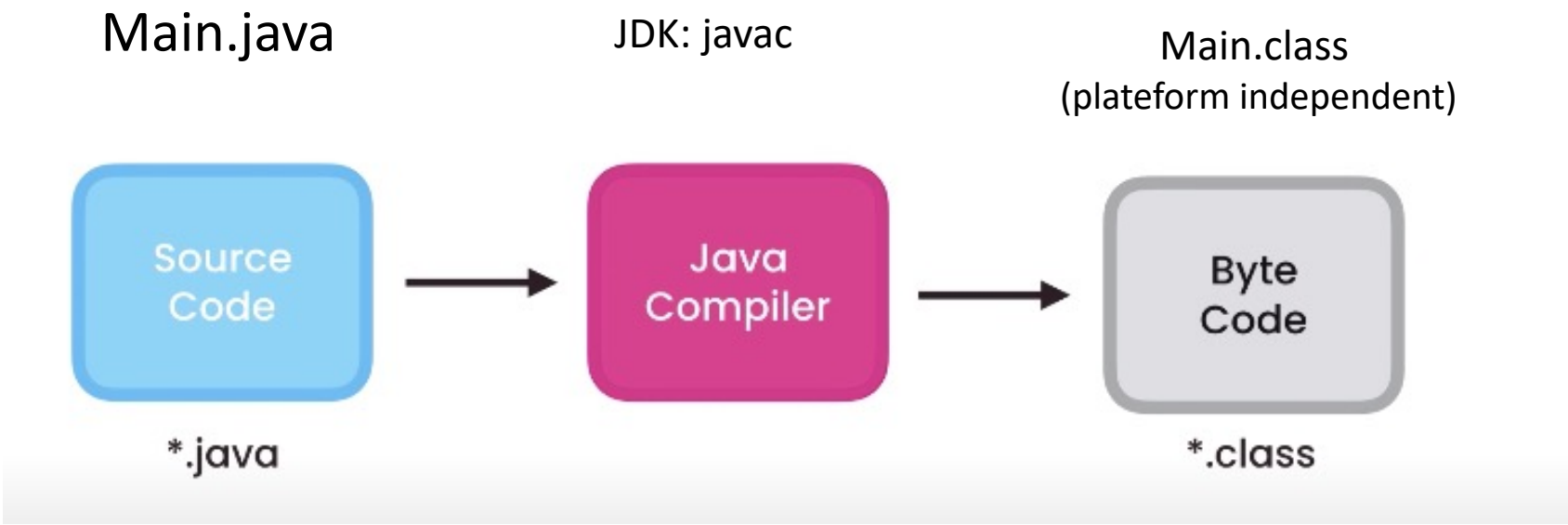
Bonjour à toutes et tous Project

- Créez un projet JAVA,
- Créez une classe qui contient un méthode main.

Output:

Bonjour à toutes et tous.

Exécution


Main.java
JDK: javac
Main.class
(plateform independent)
Source Code
Java Compiler
Byte Code
*.java
*.class

Execution en Terminal

- Javac : Pour appeler compilateur de JDK. Le code source sera converti en un “byte code”.

```
hosseins-Air:Hello hosseinkhani$ cd src
hosseins-Air:src hosseinkhani$ javac Hello.java
```

- Java (full path): Pour appeler JRE.

```
(base) hosseins-Air:src hosseinkhani$ java Hello
Bonjour a toutes et tous.
```

Nommage en JAVA

Pourquoi le nom de la classe Main commence par majuscule?

➢ Il y a des conventions:

- PascalNamingConvention (des classes , Constructeurs)
- camelNamingConvention (des méthodes et des variables)

Mot clé

- abstract, boolean, break, byte, case, catch, char, class, const*, continue, default, do, double, enum**, else, extends, final, finally, float, for, goto*, if, implements, import, instanceof, int, interface, long, native, new, null, package, private, protected, public, return, short, static, strictfp, super, switch, synchronized, this, throw, throws, transient, try, void, volatile, while .

Variables

- Les **variables** sont utilisées pour réserver un espace dans la mémoire pour les **valeurs.**

Déclaration d'une variable.

```
package com.Hello;

public class Main {

    public static void main(String[] args) {

    int age = 30;
    System.out.println(age);
    }

}
```

Nommage des Variables

- Faites attention aux noms des variables: camelNamingConvention

```
package com.Hello;

public class Main {

    public static void main(String[] args) {

    int myAge = 30;
    float yourAge;
    yourAge = myAge;
    System.out.println(yourAge);
    }

}
```

Types de variables en JAVA

Primitif

Pour enregistrer des **valeurs simples**

Réference

Pour enregistrer des **objets Complexes**

Types Primitifs

Primitive Types

Type	Bytes	Range
byte	1	[-128, 127]
short	2	[-32K, 32K]
int	4	[-2B, 2B]
long	8	
float	4	
double	8	
char	2	A, B, C, ...
boolean	1	true / false

Exemple:

```java
package com.Hello;

public class Main {

    public static void main(String[] args) {

    int age = 30;
    System.out.println(age);
    }

}
```

int reserve un space de taille 4 bytes.

byte reserve un space de taille 1 byte.

```java
package com.Hello;

public class Main {

    public static void main(String[] args) {

    byte age = 30;
    System.out.println(age);
    }

}
```

Exemple:

```java
package com.Hello;

public class Main {

    public static void main(String[] args) {

    byte age = 30;
    int countViews = 3123456789;
    System.out.println(age);
    }

}
```

The literal 3123456789 is out of range

```java
package com.Hello;

public class Main {

    public static void main(String[] args) {

    byte age = 30;
    long countViews = 3123456789;
    System.out.println(age);
    }

}
```

Erreur de compilation

```java
package com.Hello;

public class Main {

    public static void main(String[] args) {

    byte age = 30;
    long countViews = 3123456789l;
    System.out.println(age);
    }

}
```

On ajoute L (minuscule ou majuscule) a la fin.

Exemple:

```java
package com.Hello;

public class Main {

    public static void main(String[] args) {

    double temprature = 27.4;
    System.out.println(temprature);
    }

}
```

```java
package com.Hello;

public class Main {

    public static void main(String[] args) {

    float temprature = 27.4;
    System.out.println(temprature);
    }

}
```

```java
package com.Hello;

public class Main {

    public static void main(String[] args) {

    float temprature = 27.4f;
    System.out.println(temprature);
    }

}
```

On ajoute F (minuscule ou majuscule) à la fin.

Type Réference

- Différence entre type primitive et type reference:
  - Quand on définie le type primitive JRE alloue la mémoire automatiquement,
  - Quand on définie le type reference on doit faire allocation avec le mot clé new .

```
package com.Hello;

import java.util.Date;

public class Main {

    public static void main(String[] args) {
    byte age = 30;
    Date date = new Date();

    }

}
```

Allocation de mémoire

Difference entre les deux types

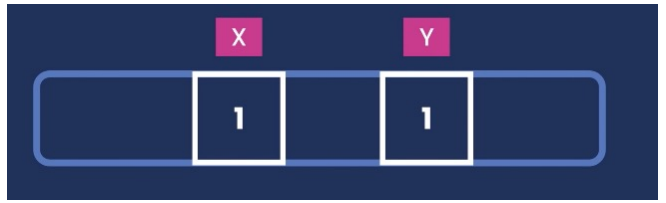
- A vous de faire:

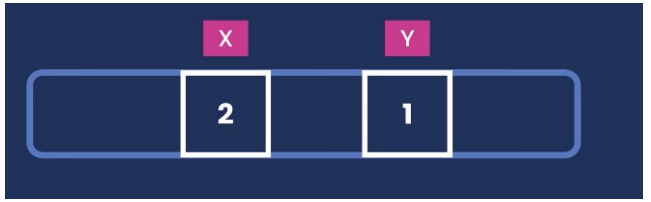
```java
package com.Hello;

public class Main {

    public static void main(String[] args) {
    byte x = 1;
    byte y = x;
    x = 2;
    System.out.println(y);

    }

}
```

```java
package com.Hello;

import java.awt.Point;

public class Main {

    public static void main(String[] args) {
    byte x,y;
    Point point1 = new Point(x=1 , y=1);
    Point point2 = point1;
    point1.x = 2;
    System.out.println(point2);

    }

}
```

Difference entre les deux types

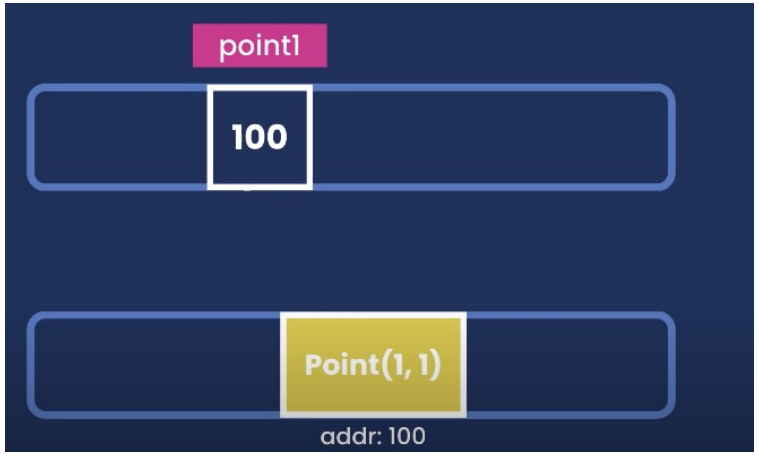
```java
package com.Hello;

public class Main {

    public static void main(String[] args) {
    byte x = 1;
    byte y = x;
    x = 2;
    System.out.println(y);

    }

}
```

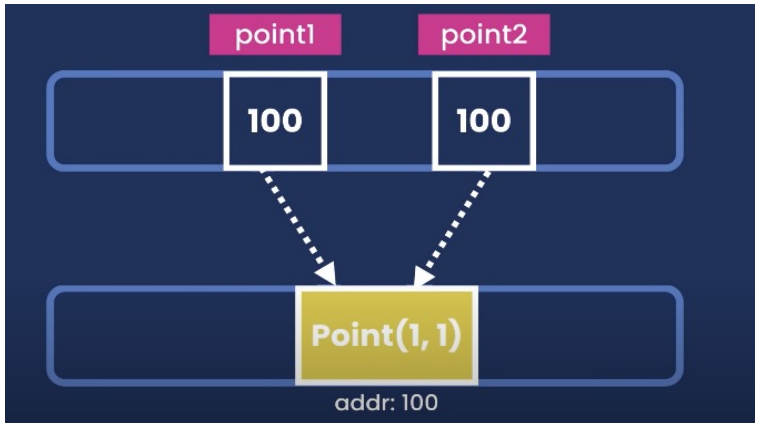




Difference entre les deux types

```java
package com.Hello;

import java.awt.Point;

public class Main {

    public static void main(String[] args) {
    byte x,y;
    Point point1 = new Point(x=1 , y=1);
    Point point2 = point1;
    point1.x = 2;
    System.out.println(point2);

    }

}
```

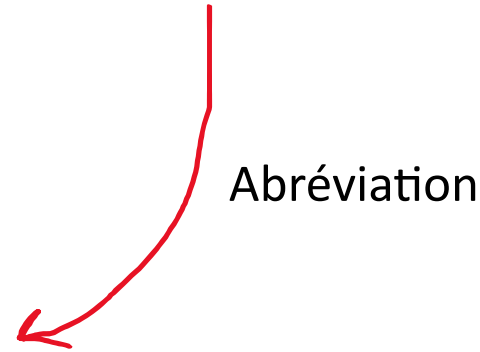




Strings

```java
package com.Hello;

public class Main {

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

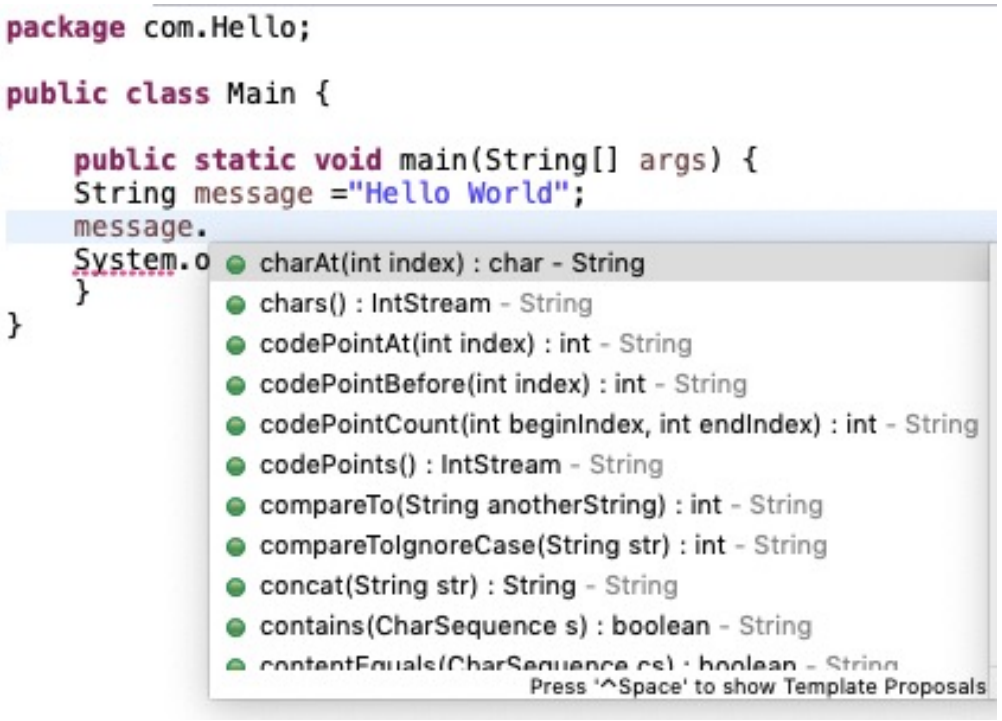
```java
package com.Hello;

public class Main {

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




```java
package com.Hello;

public class Main {

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

Operation en Strings

- Opérateur +



- **A vous:**

Length()

Concat()

Compare()

Equals()

Split()

Indexof()

Replace() *

toLowercase()

Trim()

….

Charactèr Spécial

- \”

```
String message ="My name is "Hossein" ";
String message ="My name is \"Hossein\" ";
```

- \n

```
String message ="My name is \nHossein ";
```

- \\

```
String path ="c:\\windows\\users\\... ";
```

- \t

```
String message ="My name is \tHossein ";
```

Arrays

```java
public class Main {

    public static void main(String[] args) {
    int number = 1;
    System.out.println(number);
    }
}
```

Pour enregister un list des variables

```java
public class Main {

    public static void main(String[] args) {
    int [] numbers = new int[5];
    numbers[0] = 1;
    numbers[1] = 3;
    System.out.println(numbers);
    }
}
```

Output: adresse de memoire

```java
package com.Hello;

import java.util.Arrays;

public class Main {

    public static void main(String[] args) {
    int [] numbers = new int[5];
    numbers[0] = 1;
    numbers[1] = 3;
    System.out.println(Arrays.toString(numbers));
    }
}
```

Method overloading

Arrays

- Si on connaît les valeurs en avance:

```
public class Main {

    public static void main(String[] args) {
    int [] numbers = {1,3,4,5,6};
    System.out.println(Arrays.toString(numbers));
    System.out.println(numbers.length);
    }
}
```

Arrays have a fixed length

Trier un Arrays

```java
public class Main {

    public static void main(String[] args) {
    int [] numbers = {1,4,3,2,5,6};
    Arrays.sort(numbers);
    System.out.println(Arrays.toString(numbers));
    System.out.println(numbers.length);
    }
}
```

Array multi-dimensionnel

```java
import java.util.Arrays;

public class Main {

    public static void main(String[] args) {
    int [][] numbers = new int [2][3];
    numbers[0][0] = 3;
    numbers[0][1] = 2;
    numbers[0][2] = 1;
    numbers[1][0] = 5;
    numbers[1][1] = 4;
    numbers[1][2] = 5;
    System.out.println(Arrays.deepToString(numbers));
    System.out.println(numbers.length);
    }
}
```

Exercice 1

Ecrivez un code JAVA, qui convertit les degrés Ferenheit en degrés Celcius.

Matériaux necessaire:

- C = (F-32) * 5/9,
- pour demander à l'utilisateur d'entrer une valeur:

1. Créez un objet de la classe Scanner: Scanner input = new Scanner(System.in);
2. Afficher votre demande sur l'ecran: System.out.print("Input a degree in Fahrenheit: ");
3. Enregistrez la valeur entrée: double fahrenheit = input.nextDouble();

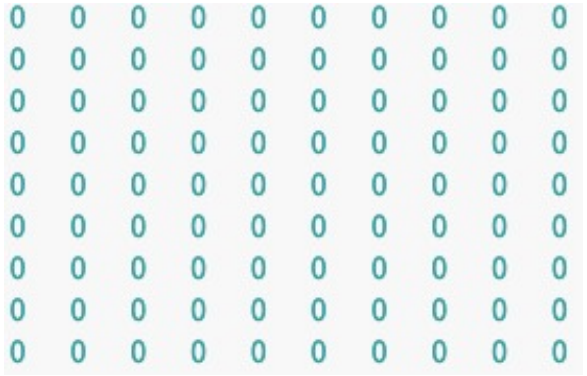
- Faite attention à priorité des operateurs.

Output anticipé: pour input 212

212.0 degrees Fahrenheit is equal to 100.0 in Celsius

Exercice 2

Un code JAVA pour imprimer un grid de Zero:



9 * 10

Matérieux necessaire:

- For loop: syntax

for (*statement 1*; *statement 2*; *statement 3*) {
// code block to be executed
}

- System.out.printf("formatting role", value)

%d ou %f,…

Paradigme Objet:

- La programmation objet est un paradime (une manière de modéliser le monde),
- Toutes les choses dans le monde sont objets (les humaines, les animaux, ….),
- des objets ont un état interne: (mon nom, nom adresse),
- des objets ont un comportement: (parler, marcher,…,),
- des objets sont capables d’échanger des message.

Classes et Objet:

- La **classe** est la description d'un objet.
- Un objet est une instance d'une **classe**.
- Pour chaque instance d'une **classe**, le code est le même, seules les données sont différentes à chaque objet.

Paradigme Objet

- **Un objet possède:**

  - Une adresse en mémoire (associer à nom d'objet)
  - Un comportement (des functions ou methods definie dans la classe)
  - Un état interne (des variables et leurs valeurs)

Exemple

Les variables instantanées

Méthodes

```
class Person {
    String name;
    int age;

    void speak() {
        for(int i=0; i<3; i++) {
            System.out.println("My name is: " + name + " and I am " + age + " years old ");
        }
    }

    void sayHello() {
        System.out.println("Hello there!");
    }
}

public class Main {

    public static void main(String[] args) {

        // Create a Person object using the Person class
        Person person1 = new Person();
        person1.name = "Joe Bloggs";
        person1.age = 37;
        person1.speak();
        person1.sayHello();

        // Create a second Person object
        Person person2 = new Person();
        person2.name = "Sarah Smith";
        person2.age = 20;
        person2.speak();
        person1.sayHello();

        System.out.println(person1.name);

    }

}
```

Constructeurs:

- Grace aux constructeurs d'une classe, on peut créer les objets d'une manière plus efficace.

- Une fois qu'elle est crée, l'instance.

  - a son propre état interne (les valeurs des variables d'instance).

  - partage le code qui détermine son comportement (les méthodes) avec les autres instances de la classe.

Exemple

Constructeur: Toujours **public**, à le même nom que la classe, n'a pas de type retour,

```
class Person {
    public Person() {
        System.out.println("Constructor running!");
    }

}

public class Main {

    public static void main(String[] args) {
            Person person1 = new Person();


    }

}
```

Output: Constructor running!

Exemple

- Le Constructeur peut utiliser pour initialiser les variables instantées d'un objet:

```
class Person {
    private String name;

    public Person(String name) {
        System.out.println("Constructor running!");
        this.name = name;
    }

}

public class Main {

    public static void main(String[] args) {
            Person person1 = new Person("Hossein");


    }

}
```

- On peut avoir plusieurs constracteurs dans une classe.

```
class Person {
    public String name;
    public int code;

    public Person() {
        System.out.println("Constructor running!");
    }

    public Person(String name) {
        System.out.println("Constructor running!");
        this.name = name;
    }

    public Person(String name , int code) {
        System.out.println("Constructor running!");
        this.name = name;
        this.code = code;
    }

}

public class Main {

    public static void main(String[] args) {

        Person person0 = new Person();
        Person person1 = new Person("Hossein");
        Person person2 = new Person("you", 1);


    }

}
```

- A vous:

Imprimez les valeur de variable pour chaque objet. Comparez les valeurs:

Réponse

- Si les variables ne sont pas initialisées par le programmeur, elles reçoivent les valeurs par défaut de leur type (0 pour les types numériques, null pour les objets) .

Surcharger d'une méthode:

- En general:

En JAVA, on peut ajouter une méthode qui a le même nom mais pas la même signature qu'une autre méthode :

calculerSalaire(int)

calculerSalaire(int, double)

- Faite attention:

Il est interdit de surcharger une méthode en changeant seulement le type de retour

int calculerSalaire(int)

double calculerSalaire(int)

Exercice

1. Créez une classe Box avec quatre variables qui illustrent les coordonnées du coin en haut à gauche et du coin en bas à droite.

2. Créez un constructeur pour initialiser les variables.

3. Créez une méthode String toString() qui retourne une chaîne de caractères qui indique les caractéristique de la Box. Par exemple la méthode peut retourner une chaîne comme celle-ci : "(100, 200) – (300,100)".

➢ toString:

La méthode **toString**() renvoie une représentation textuelle d'un objet.

Main.JAVA

```
public class Main {

    public static void main(String[] args) {
    Person person1 = new Person();
    System.out.println(person1);
    }
}
```

Person.JAVA

```
public class Person {
    String name;

    public String toString() {
        return "String Represetaion of object";
    }
}
```

Output: String Representaion of object

Natures des variables

➢ Les variable d’instances:

- ➢ sont déclarées en dehors de toute méthode.
- ➢ sont créées lorsqu'un objet est instancié.
- ➢ sont accessibles à tous les constructeurs, méthodes ou blocs de la classe.
- ➢ Chaque objet aura sa propre copie des variables d'instance.

Natures des variables

➢ Les variables locales

- ➢ sont déclarées à l'intérieur d'une méthode.
- ➢ ne sont accessibles que dans le bloc dans lequel elles ont été déclarées.
- ➢ conservent une valeur utilisée pendant l'exécution de la méthode.

Exemple: Variable d'instance

```java
package com.Hello;

class Dog {
    int age;
    public Dog(int age) {
        this.age = age;
    }
    public void about(int a) {
        System.out.println("This dog is"+ a +".");
    }
}

public class Main {

    public static void main(String[] args) {

            Dog dog1 = new Dog(5);
            Dog dog2 = new Dog(3);

            System.out.println(dog1.age);
            System.out.println(dog2.age);


    }

}
```

Exemple: Variable Local

```java
package com.Hello;

class Dog {
    int age;
    public Dog(int age) {
        this.age = age;
    }
    public void about(int a) {
        int nextYear = a + 1;
        System.out.println("This dog is"+ a +".");
        System.out.println("Next year, this dog will be" + nextYear + "years old.");
    }
}

public class Main {

    public static void main(String[] args) {

            Dog dog1 = new Dog(5);
            Dog dog2 = new Dog(3);

            System.out.println(dog1.age);
            System.out.println(dog2.age);


    }

}
```

Static Variable ou variable de classe

```
class Dog {
    int age;
    static int numberOfLegs = 4;
    public Dog(int age) {
        this.age = age;
    }
    public void about(int a) {
        int nextYear = a + 1;
        System.out.println("This dog is"+ a +".");
        System.out.println("Next year, this dog will be" + nextYear + "years old.");
    }
}

public class Main {

    public static void main(String[] args) {

            Dog dog1 = new Dog(5);
            Dog dog2 = new Dog(3);

            System.out.println(dog1.age);
            System.out.println(dog2.age);
            System.out.println(dog1.numberOfLegs);
            System.out.println(dog2.numberOfLegs);
    }

}
```

Variable Static (Usage)

```
class Dog {
    int age;
    static int numberOfInstances;
    public Dog(int age) {
        this.age = age;
        this.numberOfInstances++;
    }
    public void about(int a) {
        int nextYear = a + 1;
        System.out.println("This dog is"+ a +".");
        System.out.println("Next year, this dog will be" + nextYear + "years old.");
    }
}

public class Main {

    public static void main(String[] args) {

            Dog dog1 = new Dog(5);
            Dog dog2 = new Dog(3);

            System.out.println(dog1.age);
            System.out.println(dog2.age);
            System.out.println(Dog.numberOfInstances);

    }

}
```

Exercice

- Créez une classe Pixel pour représenter un pixel de l'écran. Pour cette classe, vous pouvez également implémenter une méthode String toString().

- Modifiez l'implémentation de la classe Box pour utiliser maintenant votre classe Pixel à la place des coordonnées de vos points.

- On désire maintenant ajouter un identifiant sous la forme d'un **int**. Modifiez le constructeur pour donner de manière automatique un unique identifiant.

Indice : on pourrait compter le nombre d'instances de la classe Box et faire en sorte que la *i*^{eme} instance créée ait pour identifiant *i*.