



THE COMPLETE JAVA

 @MehdiBouchachi

SECTION

JAVA ADVANCE

LECTURE

JAVA FUNDAMENTALS (IN ARABIC)

METHODS AND FUNCTIONS

METHODS

ت تكون ال Method من :

1. نوع ال .Modifier (public , private...)

2. void . اي ليس لديها (return)

3. ال اسم .Method

4. مدخلات .(Parameters)

5. Logic



A screenshot of a Java code editor showing a method definition. The code is:

```
1
2 public void affiche(String fName, String lName, int age){
3
4     System.out.println("Welcome "+fName+" "+lName);
5
6 }
```

The code is displayed in a dark-themed code editor. The method is named 'affiche' and takes three parameters: 'fName', 'lName', and 'age'. It prints a welcome message to the console.

METHODS AND FUNCTIONS

FUNCTIONS

ت تكون ال Method من :

1. نوع ال .Modifier (public , private...)

2. اي لها (return) .DataType

3. اسم ال .Method

4. مدخلات .(Parameters)

5. Logic



```
1 public double moy(double exam , double td){  
2  
3     double moy = (exam*0.6)+(td*04);  
4     return (moy);  
5  
6 }
```

CREATING OBJECTS

OBJECT صناعة

☞ يتم صناعة object عندما نريد استدعاء دالة من كلاس مختلف، من اجل صناعة object نحتاج الى :

- `.class name`
- اسم جديد يرمز الى `.class`
- نقوم بعمل `.new class name`

CREATING OBJECTS

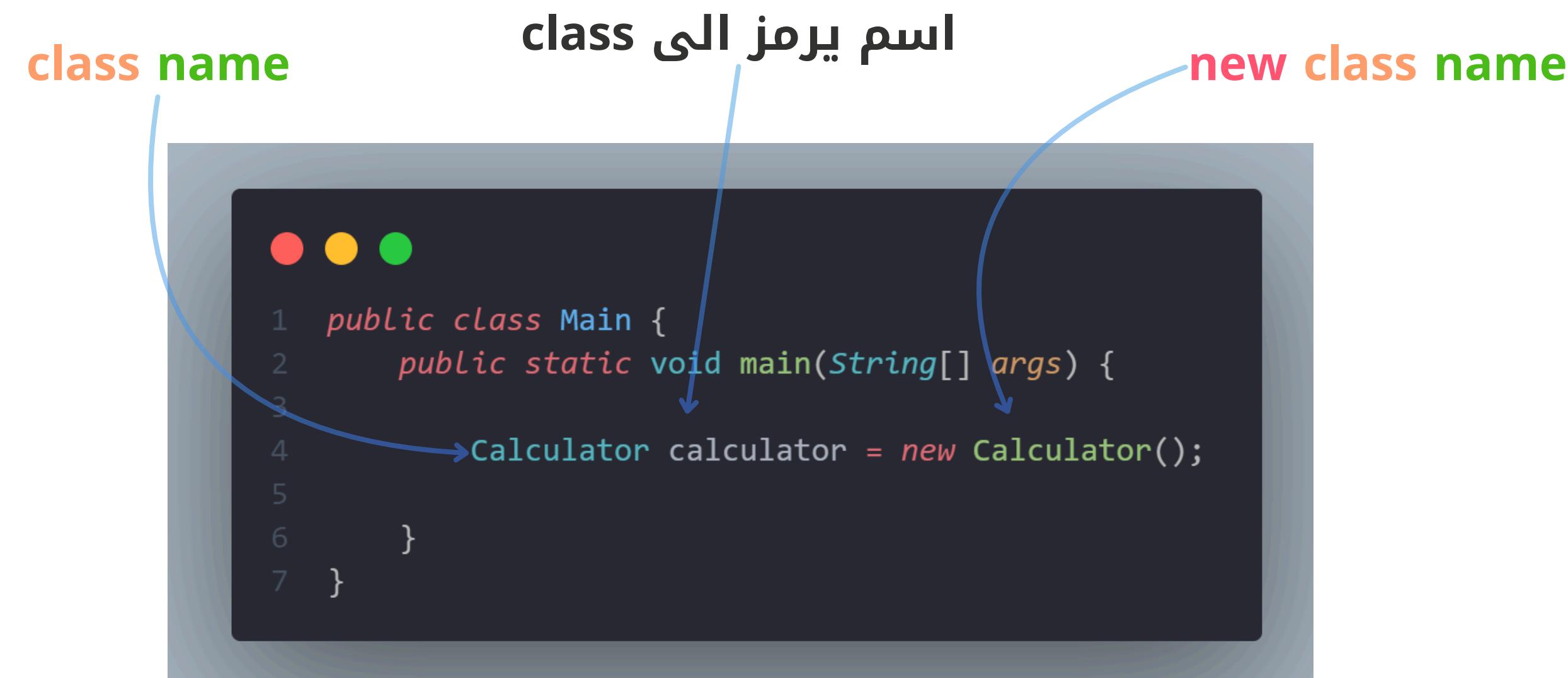
OBJECT صناعة

لدينا كلاس اسمه **object** يحتوي على دالة **calcul** سنقوم بصناعة **calculator** لهذا الكلاس من أجل استدعاء دالة **calcul** و استخدامها في الـ **Main**.

```
1 public class Calculator {  
2  
3     public void calcul(){  
4         //Code  
5     }  
6 }
```

CREATING OBJECTS

OBJECT صناعة

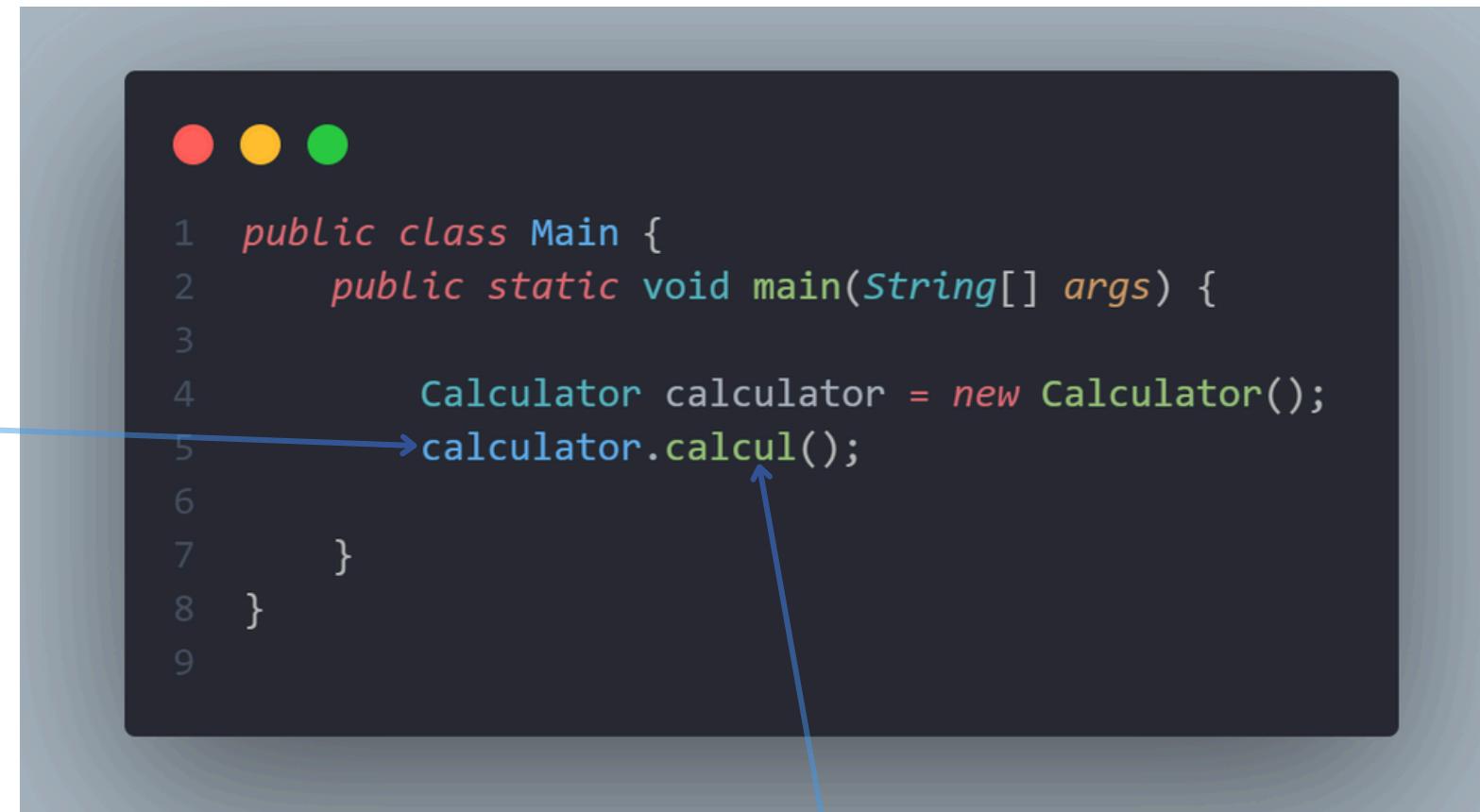


CREATING OBJECTS

CALLING METHOD

لاستدعاء ميثود او متغير ما يجب علينا كتابة : **. اسم يرمز لل class . variableName/methodName** :

اسم يرمز الى class



```
1 public class Main {  
2     public static void main(String[] args) {  
3  
4         Calculator calculator = new Calculator();  
5         calculator.calculate();  
6  
7     }  
8 }  
9
```

Method name

PUBLIC/PRIVATE METHODS

PUBLIC METHOD

يمكن استدعائهما و استخدامها في **جميع الكلاسات**. ➔

PRIVATE METHOD

يمكن استدعائهما و استخدامها **فقط** في الكلاس الموجودة فيه. ➔

EXAMPLE :



PUBLIC/PRIVATE METHODS

```
● ● ●  
1 public class PubPriExample {  
2     public void Moy(double s1, double s2, String name){  
3  
4         String result = "Grade of "+name+" : "+ (s1+s2)/2;  
5         System.out.println(result);  
6  
7     }  
8  
9     private void Moy1(double s3, double s4 ,String name){  
10  
11        String result = "Grade of "+name+" : "+ (s3+s4)/2;  
12        System.out.println(result);  
13  
14    }  
15 }
```

PUBLIC/PRIVATE METHODS

The screenshot shows a Java code editor with the following code:

```
> src > Main.java > Main > main(String[])
public class Main {
    Run | Debug
    public static void main(String[] args) {
        PubPriExample example = new PubPriExample();
        example. Syntax error, insert "VariableDeclarators"
    }
}
```

A code completion dropdown is open at the line `example.`. The first suggestion is `Moy(double s1, double s2, String name)`, which is highlighted with a blue rectangle. Other suggestions include `equals(Object obj)`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`, `toString()`, `wait()`, `wait(long timeoutMillis)`, and `wait(long timeoutMillis, int nanos)`. Below the suggestions are two additional options: `cast` (Casts the expression to a new type) and `nnull` (Creates an if statement and checks if the expression is null).

تظهر لنا دالة

واحدة متوفرة فقط

public void Moy

STATIC / NOSTATIC IN METHOD AND VARIABLES

STATIC METHOD

هي إنها ضمن الكلاس و متوفرة لجميع الكلاسات و يمكن استخدامها بدون **.object** ↗

```
● ● ●  
1 public class Calculator {  
2  
3     public static void calcul(){  
4         //Code  
5     }  
6 }
```

A diagram illustrating the use of a static method. On the left, a code snippet shows a class named 'Calculator' containing a static method 'calcul()'. A blue arrow points from this code to the right, where another code snippet shows the static method being called from the 'main()' method of a 'Main' class.

```
● ● ●  
1 public class Main {  
2     public static void main(String[] args) {  
3  
4         Calculator.calcul();  
5  
6     }  
7 }  
8
```

STATIC / NOSTATIC IN METHOD AND VARIABLES

NOSTATIC METHOD

هذا النوع من الـ **Methods** متوفرة لكل الكلاسات لكن تحتاج الى **object** حتى يمكن استخدامها.

```
● ● ●  
1 public class Calculator {  
2     public void calcul(){  
3         //Code  
4     }  
5 }  
6 }
```

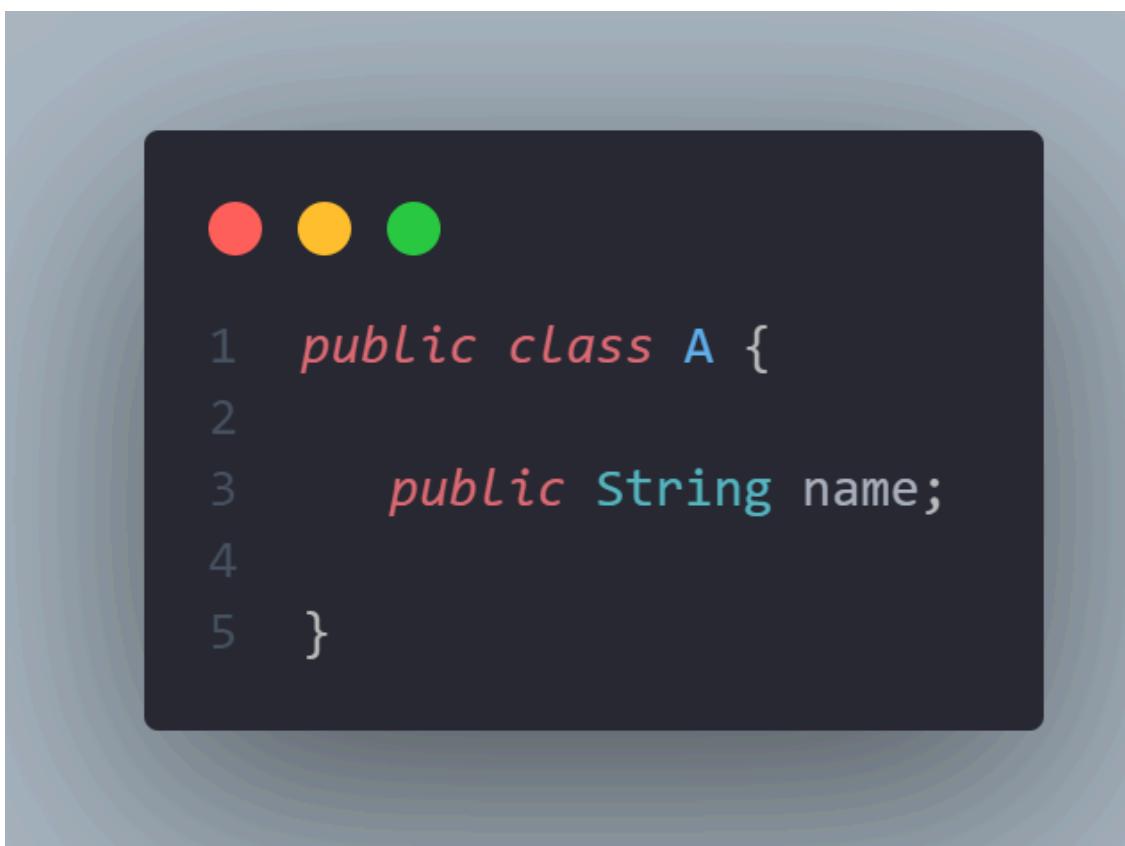


```
● ● ●  
1 public class Main {  
2     public static void main(String[] args) {  
3           
4             Calculator calculator = new Calculator();  
5             calculator.calcul();  
6         }  
7     }  
8 }  
9 }
```

STATIC / NOSTATIC IN METHOD AND VARIABLES

NOSTATIC VARIABLES

يمكن استخدام الـ **Variables** في **класс A** و **класс B** مختلف **class** و **method** في **نقطة مختلفة** و يمكن ان **تختلف** متغير **في** **класс A** و **يمكن** ان **تختلف** متغير **في** **класс B**.



```
1 public class A {  
2     public String name;  
3 }  
4  
5 }
```



```
1 public class B {  
2     public void show(){  
3         A myA = new A();  
4         myA.name = "Ta3alam Coding";  
5         System.out.println(myA.name);  
6     }  
7 }  
8  
9 }  
10
```

STATIC / NOSTATIC IN METHOD AND VARIABLES

NOSTATIC VARIABLES



```
1 public class Main {  
2     public static void main(String[] args) {  
3           
4         B myB = new B();  
5         myB.show();  
6     }  
8 }
```

OUTPUT

TA3ALAM CODING

STATIC / NOSTATIC IN METHOD AND VARIABLES

STATIC VARIABLES



```
1 public class A {  
2  
3     public static String name;  
4  
5 }
```



```
1 public class B {  
2     public void show(){  
3  
4         A.name = "Ta3alam Coding";  
5         System.out.println(A.name);  
6  
7     }  
8 }  
9
```

STATIC / NOSTATIC IN METHOD AND VARIABLES

STATIC VARIABLES



```
1 public class Main {  
2     public static void main(String[] args) {  
3           
4         B myB = new B();  
5         myB.show();  
6     }  
8 }
```

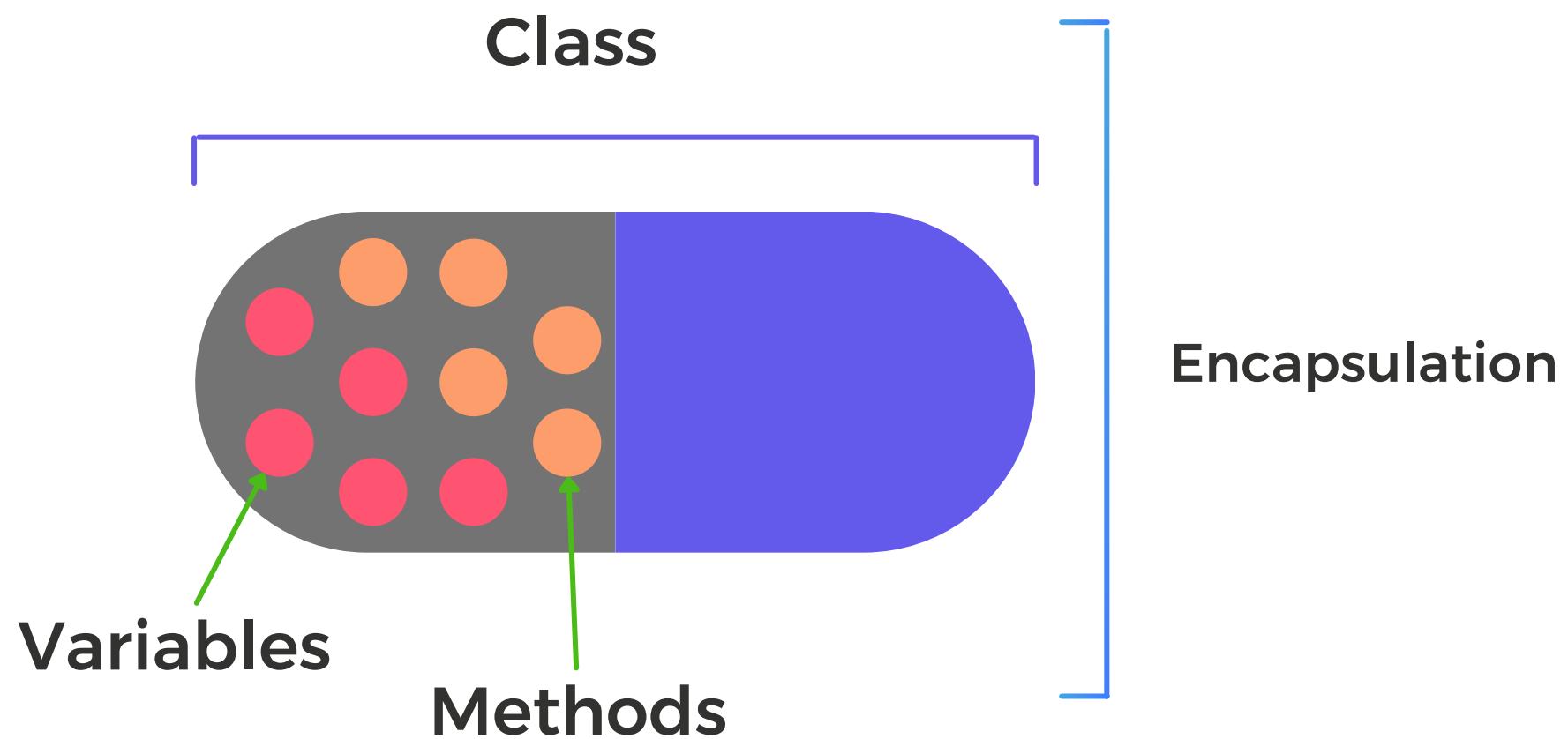
OUTPUT

TA3ALAM CODING

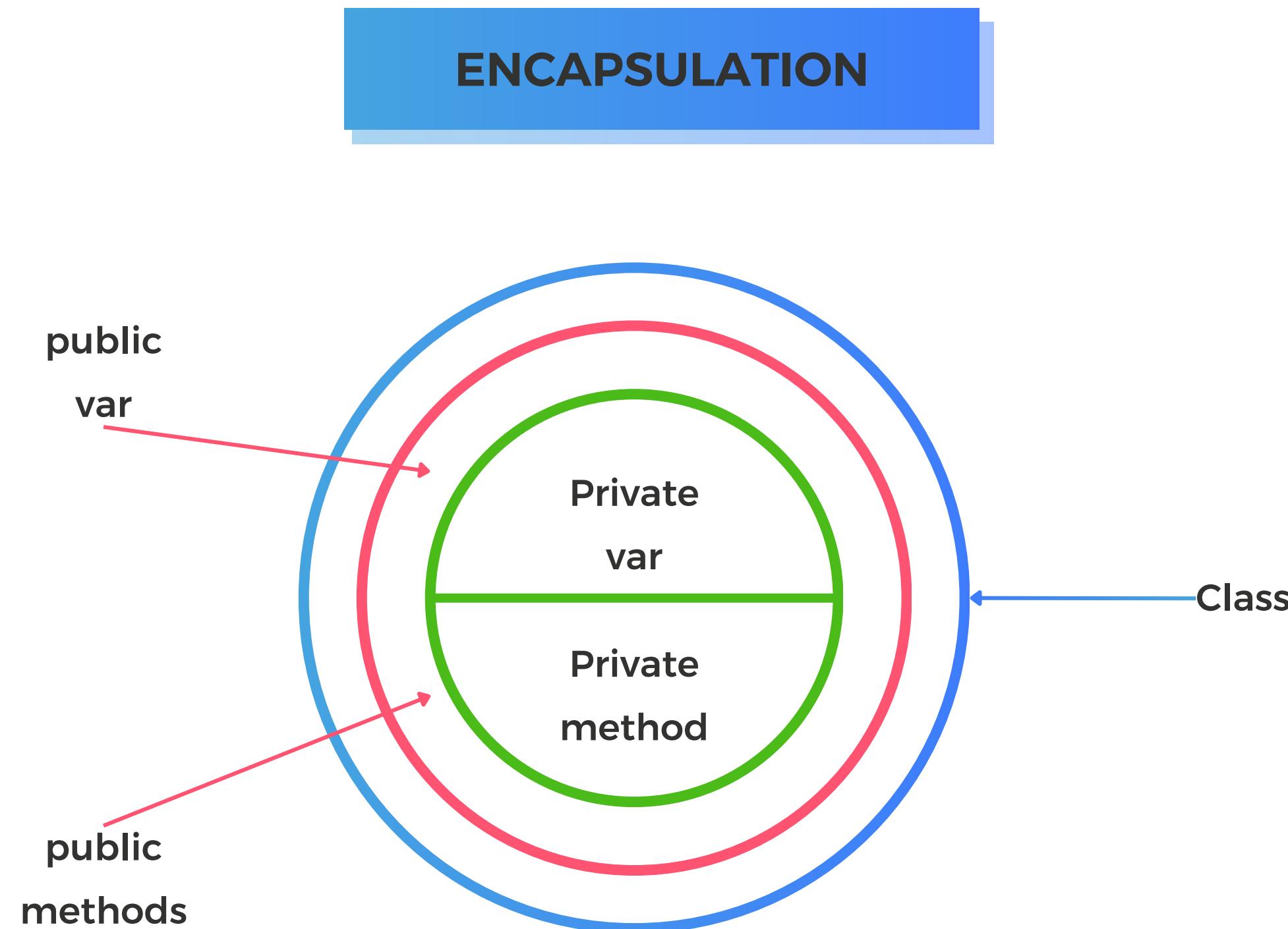
ENCAPSULATION

ENCAPSULATION

عبارة عن أسلوب يمكن اتباعه لـ**إخفاء البيانات و الخصائص الأساسية الموجودة في الكلاس**, و جعل الكلاسات الأخرى **قادرة على التعامل مع هذه البيانات او الخصائص فقط من خلال دوال يقوم بإنشائها المبرمج في الكلاس الأساسي.**



ENCAPSULATION



CONSTRUCTOR

CONSTRUCTOR

- هي دوال خاصة في جافا تنشئ **.Object** ↗
 - يجب ان يكون **اسمه** نفس اسم **الClass**.
 - لا يقوم بارجاع اي قيمة. (no return)

DEFAULT CONSTRUCTOR

- لا يحتوي على **.args** او **Parameters** ↗

CONSTRUCTOR

DEFAULT CONSTRUCTOR



```
1 public class Mehdi {  
2     public Mehdi(){  
3         System.out.println("Hello :));  
4     }  
5 }  
6  
7 }
```



```
1 public class Main {  
2     public static void main(String[] args) {  
3         Mehdi mehdi = new Mehdi();  
4     }  
5 }  
6  
7 }
```

يتم طباعة مباشرة الكلمة Hello . (يتم دائمًا تنفيذ الـ Constructor مع صناعة Object !) ➔

CONSTRUCTOR

CONSTRUCTOR

```
1 public class Mehdi {  
2  
3     int Id, age;  
4     String name;  
5  
6     public Mehdi(String n ,int i ){  
7         name = n;  
8         Id = i;  
9     }  
10  
11    public void show(){  
12        System.out.println("ID : "+Id+"\nName : "+name);  
13    }  
14  
15 }
```

```
1 public class Main {  
2     public static void main(String[] args) {  
3  
4         Mehdi mehdi = new Mehdi("Mehdi", 14);  
5         mehdi.show();  
6  
7     }  
8 }
```

ID : 14 , Name : Mehdi

يتم طباعة ➔

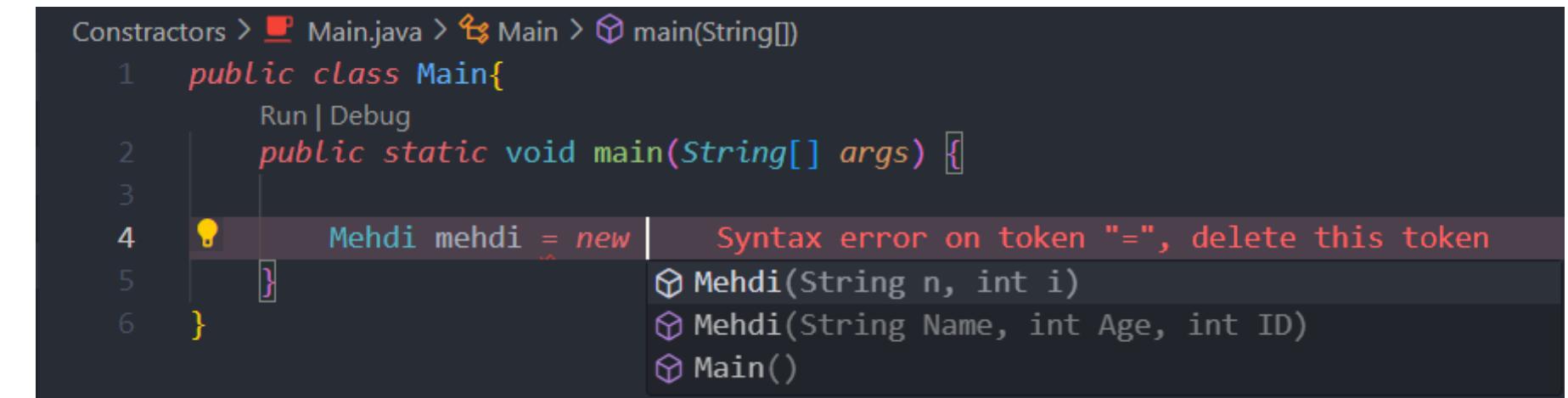
CONSTRUCTOR

CONSTRUCTOR OVERLOADING

هي صناعة 2 Constructors فما فوق مع تغيير نوع ال Parameters او عددهم او ترتيبهم.



```
1 public class Mehdı {
2
3     int Id, age;
4     String name;
5     public Mehdı(String n ,int i ){
6         name = n;
7         Id = i;
8     }
9     public Mehdı(String Name , int Age , int ID){
10        name = Name;
11        age = Age;
12        Id = ID;
13    }
14    public void show(){
15        System.out.println("ID : "+Id+"\nName : "+name);
16    }
17    public void show1(){
18        System.out.println("Name : "+name+"\nAge : "+age+"\nID : "+Id);
19    }
20 }
21 }
```



```
Constructors > Main.java > Main > main(String[])
1 public class Main{
2     Run | Debug
3     public static void main(String[] args) {
4         Mehdı mehdı = new | Syntax error on token "=", delete this token
5     }
6 }
```

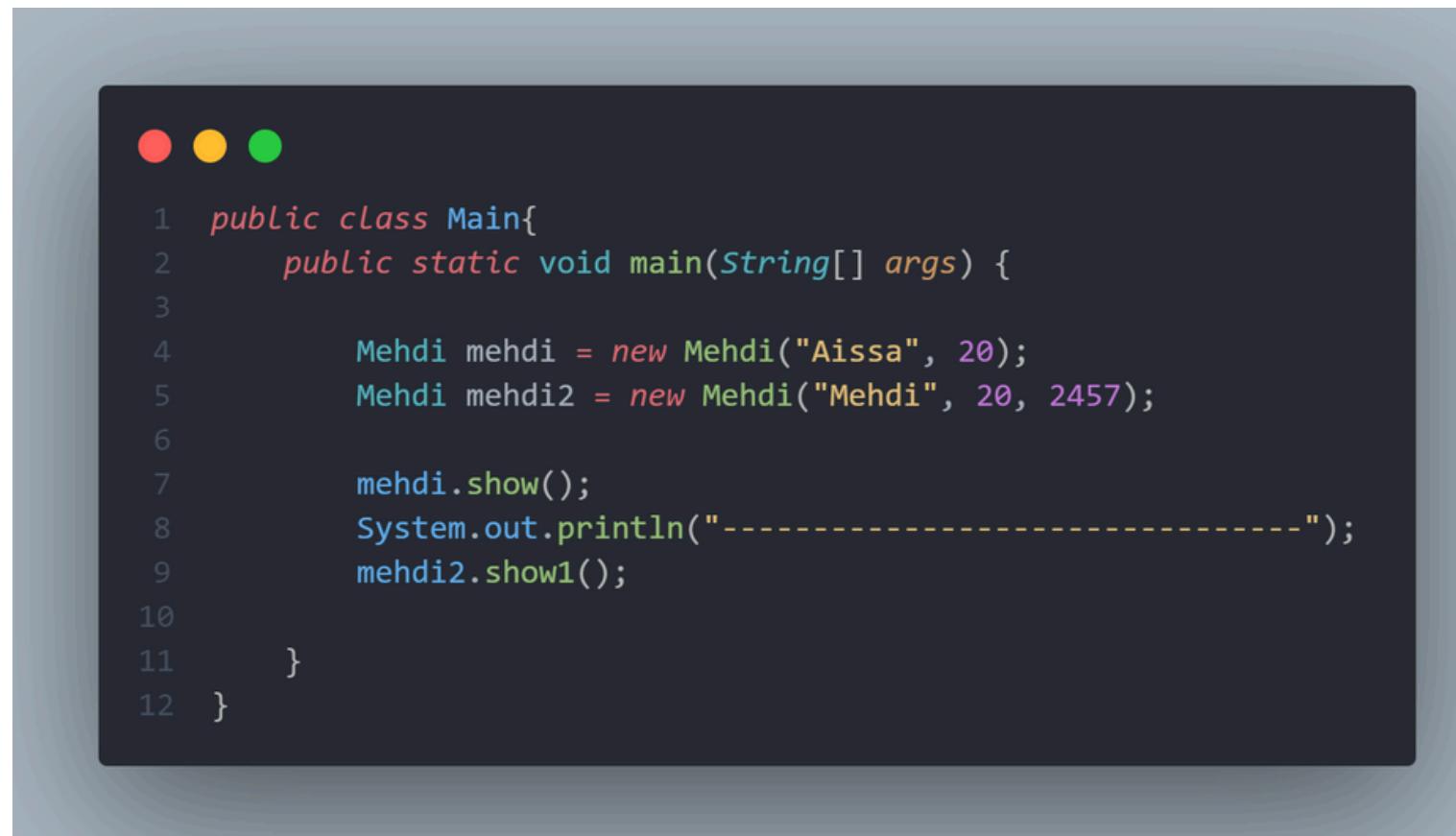
Mehdı(String n, int i)
Mehdı(String Name, int Age, int ID)
Main()

في مرحلة صناعة ال Constructors يظهر لـ Object ال الذين تم صناعته.

CONSTRUCTOR

Main

OUTPUT



```
1 public class Main{
2     public static void main(String[] args) {
3
4         Mehdi mehdi = new Mehdi("Aissa", 20);
5         Mehdi mehdi2 = new Mehdi("Mehdi", 20, 2457);
6
7         mehdi.show();
8         System.out.println("-----");
9         mehdi2.show1();
10
11    }
12 }
```



PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
in'

ID : 20
Name : Aissa
-----
Name : Mehdi
Age : 20
ID : 2457
PS C:\Users\JOHN\Desktop\Java course>
```

في نفس الوقت. Constructors 2 و استخدام Objects 2 معاً ↗

CONSTRUCTOR VS METHODS

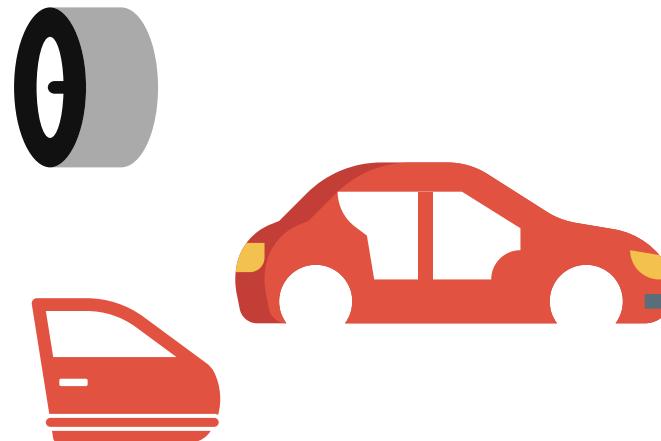
CONSTRUCTOR

.Object يريك عمل ال Method ↗

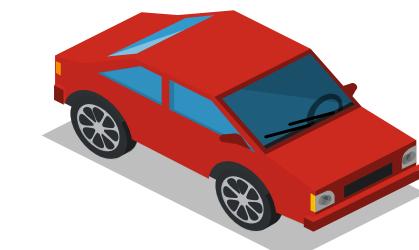
.Object يوفر لنا القيم التي نحتاجها لبناء Constructor ↗

.Object دائما يتم تنفيذه عند صناعة Constructor ↗

عند صناعة ال Object



Constructor



Mehtod



THIS / SETTER / GETTER

THIS KEYWORD

• هي متغير مرجعي يشير القيمة الحالية للObject (اتفكر هذى راح نطبقوا عليها فل PDF 03).
• يمكن من خلاله للميثود او الـ **Constructor** ان يستخدم **اسماء المتغيرات نفس** في الكلاس.

```
2
3 public class User {
4     int ID ;
5     String username;
6     String password;
7     public User(int ID, String username, String password){
8
9         ID = ID;      The assignment to variable ID has no effect
10        username = username;    The assignment to variable username has no effect
11        password = password;   The assignment to variable password has no effect
12    }
13
14
15    public void show(){
16        System.out.println("User ID : "+ID+
17                    "\nUser username : "+username+
18                    "\nUser password : "+password);
19    }
20 }
```

THIS / SETTER / GETTER

THIS KEYWORD

```
1  public class User {  
2      int ID ;  
3      String username;  
4      String password;  
5      public User(int ID, String username, String password){  
6  
7          this.ID = ID;  
8          this.username = username;  
9          this.password = password;  
10     }  
11  
12     public void show(){  
13         System.out.println("User ID : "+ID+  
14             "\nUser username : "+username+  
15             "\nUser password : "+password);  
16     }  
17 }  
18 }
```

THIS / SETTER / GETTER

get();

بما انه فيها كلمة **get** اي انك تطلب من الدالة ترجعلك قيمة **مهمما كانت** ، و بما انه يجعلك **قيمة اي فيها** **Void** ، وعليه يجب تحديد **Data type** في مكان ال **return** .

Syntax :

```
String name;  
  
public String getName( ) {  
    return name;  
}
```

THIS / SETTER / GETTER

get();

```
● ● ●  
1 public class Getter {  
2     String name;  
3     int ID;  
4  
5     public int getID() {  
6         return ID;  
7     }  
8  
9     public String getName() {  
10        return name;  
11    }  
12  
13}
```

THIS / SETTER / GETTER

set();

عند نسب قيمة الى **Constructor** عند صناعة **Object** لا يمكننا تغييرها بعدها و تتيح لنا خاصية **set()** تغيير القيمة في اي وقت. **بما انه فيها كلمة set** اي تقوم **بالتنفيذ** اي فيها **void** ولا ترجع اي قيمة.

Syntax :

```
String name;  
  
public void setName(String name) {  
    this.name = name;  
}
```

THIS / SETTER / GETTER

set();

```
1 public class Setter {  
2     int ID;  
3     String name;  
4  
5     public void setID(int ID) {  
6         this.ID = ID;  
7     }  
8  
9     public void setName(String name) {  
10        this.name = name;  
11    }  
12}
```

THIS / SETTER / GETTER

REAL EXAMPLE

```
● ● ●  
1 public class Employee {  
2  
3     private String name ;  
4     private int ID;  
5     private String address;  
6     private String department;  
7  
8     public Employee(int ID, String name, String department, String address){  
9  
10        this.ID = ID;  
11        this.name = name;  
12        this.department = department;  
13        this.address = address;  
14    }  
15}
```

```
● ● ●  
1 public String getName() {  
2     return name;  
3 }  
4 public void setName(String name) {  
5     this.name = name;  
6 }  
7 public int getID() {  
8     return ID;  
9 }  
10 public void setID(int ID) {  
11    this.ID = ID;  
12 }  
13 public String getAddress() {  
14    return address;  
15 }
```

```
● ● ●  
1 public String getAddress() {  
2     return address;  
3 }  
4 public void setAddress(String address) {  
5     this.address = address;  
6 }  
7 public String getDepartment() {  
8     return department;  
9 }  
10 public void setDepartment(String department) {  
11    this.department = department;  
12 }  
13 public void show(){  
14    System.out.println("Name : "+name+"\nID : "+ID  
15                           +"\\nDepartment : "+department+"\nAddress : "+address);  
16 }  
17 }
```

THIS / SETTER / GETTER

MAIN

```
● ● ●
1 public class Main {
2     public static void main(String[] args) {
3
4         Employee employee = new Employee(321086643,"Mehdi","MI","Cite N05");
5         employee.show();
6
7         String nameBefore = employee.getName(); //getting the name from constructor and adding it to the var
8         employee.setName("Mohamed"); //changing the name value in constructor from mehdi to mohamed
9         String nameAfter = employee.getName(); //getting the name from constructor after setName
10
11        System.out.println("\n-----\n");
12        System.out.println("Name before : "+nameBefore);
13        System.out.println("Name after : "+nameAfter);
14        System.out.println("\n-----");
15
16        employee.show();
17
18    }
19 }
```

OUTPUT

```
Name : Mehdi
ID : 321086643
Department : MI
Address : Cite N05
```

```
-----  
Name before : Mehdi
Name after : Mohamed
```

```
-----  
Name : Mohamed
ID : 321086643
Department : MI
Address : cite N05
PS C:\Users\JOHN\Desktop\Java course>
```

INHERITANCE

INHERITANCE

عبارة عن علاقه بين مجموعة كلاسات ، كمثال عندك كلاس **A**, **B** و **C** يمكن لـ**كلاس A** ان يعطي العناصر الموجودة داخله (ميثودس و متغيرات) لـ**كلاس B** و **C**.

في جافا لدينا **ParentClass** في معظم لغات البرمجة اخري يطلق عليهم **SubClass** و **SuperClass** .**ChildClass**

Syntax :

```
class SubClass extends SuperClass {
```

Methods / Vars

```
}
```

INHERITANCE TYPES

SIMPLE INHERITANCE

SuperClass

Class A

SubClass

Class B

```
● ● ●  
1 public class ClassA {  
2  
3     public void show(){  
4  
5         System.out.println("TaBalam Coding From Class A");  
6  
7     }  
8 }
```

```
● ● ●  
1 public class ClassB extends ClassA{  
2  
3 }  
4
```

INHERITANCE TYPES

SIMPLE INHERITANCE

MAIN

OUTPUT

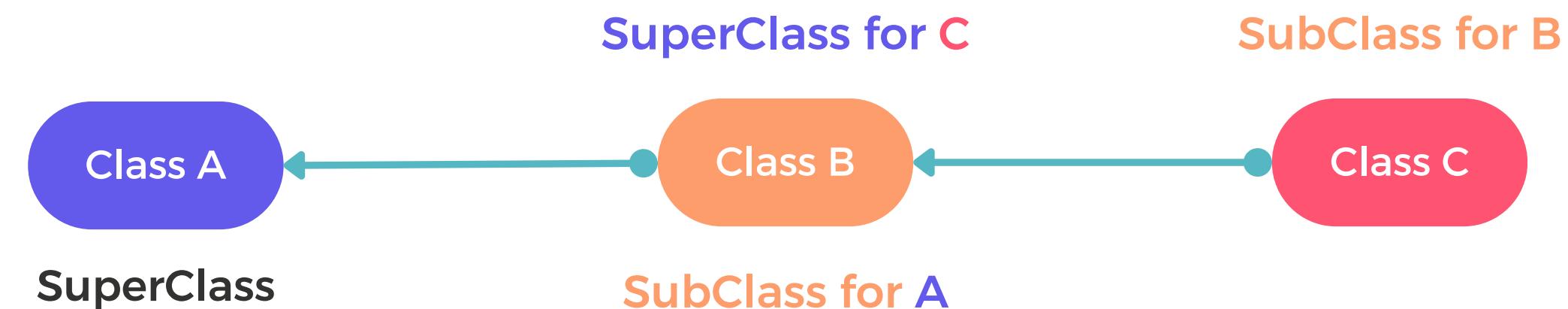
```
● ● ●
1 public class Main {
2     public static void main(String[] args) {
3
4         ClassB classB = new ClassB();
5         classB.show();
6
7     }
8 }
```

```
"C:\Program Files\Java\jdk-19\bin\java.exe"
Ta3alam Coding From Class A

Process finished with exit code 0
```

INHERITANCE TYPES

MULTIPLE INHERITANCE



```
● ● ●  
1 public class ClassA {  
2     public void show(){  
3         System.out.println("TaBalam Coding From Class A");  
4     }  
5 }
```

```
● ● ●  
1 public class ClassB extends ClassA{  
2  
3 }  
4
```

```
● ● ●  
1 public class ClassC extends ClassB{  
2  
3 }
```

INHERITANCE TYPES

MULTIPLE INHERITANCE

MAIN

OUTPUT

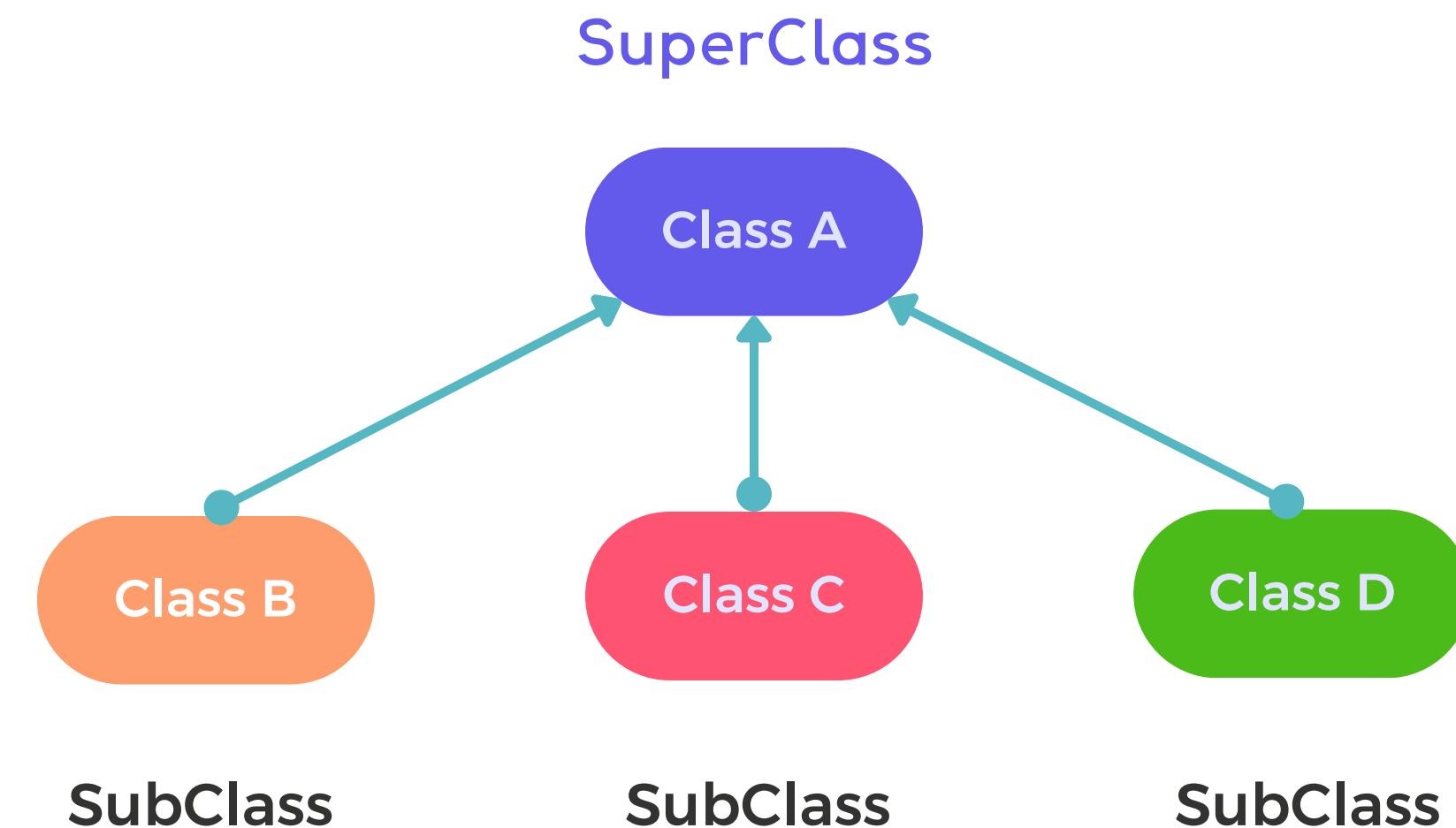
```
● ● ●
1 public class Main {
2     public static void main(String[] args) {
3
4         ClassC classC = new ClassC();
5         classC.show();
6
7     }
8 }
```

```
"C:\Program Files\Java\jdk-19\bin\java.exe"
Ta3alam Coding From Class A

Process finished with exit code 0
```

INHERITANCE TYPES

HIERARCHICAL INHERITANCE



INHERITANCE TYPES

INHERITANCE WITH CONSTRUCTOR

```
1 public class ClassA {  
2     public void show(){  
3         System.out.println("TaBalam Coding From Class A");  
4     }  
5 }  
6  
7 }
```

```
1 public class ClassB extends ClassA{  
2     public ClassB(){  
3         System.out.println("This msg from class B");  
4     }  
5 }  
6  
7 }
```

```
1 public class ClassC extends ClassA{  
2     public ClassC(){  
3         System.out.println("This msg from class C");  
4     }  
5 }  
6  
7 }
```

INHERITANCE TYPES

INHERITANCE WITH CONSTRUCTOR

MAIN

```
● ● ●  
1 public class Main {  
2     public static void main(String[] args) {  
3         ClassC classC = new ClassC();  
4         classC.show();  
5     }  
6 }  
7 }  
8 }
```

OUTPUT

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

This msg from class C
TaBalam Coding From Class A

PS C:\Users\JOHN\Desktop\Java course>

METHOD OVERRIDING

METHOD OVERRIDING

إذا كان اسم **Method** الموجود في **SuperClass** هو نفسه الموجود في **SubClass**، ولكن الكود داخلها مختلف في هذه الحالة تسمى بـ **Method overriding**.

EXAMPLE :



METHOD OVERRIDING

METHOD OVERRIDING



```
1 public class Main {  
2     public static void main(String[] args) {  
3         Employee employee = new Employee();  
4         employee.Salary(1);  
5         Manager boss = new Manager();  
6         boss.Salary(1);  
7     }  
8 }
```



```
1 public class Manager {  
2     double prime;  
3     double Salary;  
4     double fullSalary;  
5     public void Salary(int exp){  
6         if (exp <= 2){  
7             Salary = 35000;  
8             prime = 3000;  
9         }else if(exp < 5){  
10            Salary = 40000;  
11            prime = 5000;  
12        }else {  
13            Salary = 45000;  
14            prime = 10000;  
15        }  
16        fullSalary = Salary + prime;  
17        System.out.println("Salary boss : "+fullSalary);  
18    }  
19 }
```



```
1 public class Employee extends Manager{  
2     public void Salary(int exp){  
3         if (exp <= 2){  
4             Salary = 25000;  
5             prime = 1500;  
6         }else if(exp < 5){  
7             Salary = 35000;  
8             prime = 3000;  
9         }else {  
10            Salary = 40000;  
11            prime = 5000;  
12        }  
13        fullSalary = Salary + prime;  
14        System.out.println("Salary employee : "+fullSalary);  
15    }  
16 }  
17 }
```

METHOD OVERRIDING

OUTPUT

```
PROBLEMS      OUTPUT      DEBUG CONSOLE      TERMINAL      PORTS
Employee salary : 26500.0
Manager salary : 38000.0
PS C:\Users\JOHN\Desktop\Java course>
```

SUPER KEYWORD

SUPER KEYWORD

الكلمة الأساسية **Super** في Java مقتبسة من الكلمة **SuperClass** وهي متغير مرجعي يستخدم للإشارة إلى **SuperClass variable/Methods** او **SuperClass object**.

يمكننا استخدام **Super keyword** للوصول إلى متغير او الميثود الموجود في **SuperClass** يمكننا استخدامه إذا كان **SubClass** يحتويان على نفس اسم **المتغيرات و الميثودس** و مختلفين في القيم.

SUPER KEYWORD

WITH VARIABLES



```
1 public class Animal {  
2  
3     String color= "white";  
4  
5 }
```



```
1 public class Dog extends Animal{  
2  
3     String color = "Black";  
4  
5     public void showColor(){  
6  
7         System.out.println("Print color from Dog class : "+color);  
8  
9         System.out.println("Print color from Animal class : "+super.color);  
10    }  
11 }
```



```
1 public class Main {  
2     public static void main(String[] args) {  
3  
4         Dog dog = new Dog();  
5         dog.showColor();  
6  
7     }  
8 }  
9
```

METHOD OVERRIDING

OUTPUT

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

PORTS

```
Print color from Dog class : Black
```

```
Print color from Animal class : white
```

```
○ PS C:\Users\JOHN\Desktop\Java course>
```

SUPER KEYWORD

SUPER KEYWORD

في المثال السابق لكل من الكلاس (SubClass) Dog و الكلاس (SuperClass) Animal لهم نفس اسم المتغير color.

عندما نقوم بطبع color في الكلاس الحالي Animal يقوم بطبع Dog color ، وحتى نقوم بطبع color الموجود في SuperClass اشارة هنا باننا نريد المتغير color الموجود في Super.

في بعض المرات نقوم بعمل override للميثود superClass الموجودة في superClass داخل ال subClass و نريد ان نستخدم الميثود superClass الموجود في superClass داخل subClass عندما نقوم باستدعائهما بواسطة الكلمة super.

SUPER KEYWORD

WITH METHODS

```
1 public class Cat {  
2     public void eat(){  
3         System.out.println("The cat is eating...");  
4     }  
5     public void sleep(){  
6         System.out.println("Zzz Zzz");  
7     }  
8 }
```

```
1 public class Dog extends Cat{  
2     public void eat(){  
3         System.out.println("The dog is eating");  
4     }  
5     public void sleep(){  
6         System.out.println("The dog did not sleep");  
7     }  
8     public void show(){  
9         System.out.println("This msg from Dog class\n");  
10        eat();  
11        sleep();  
12        System.out.println("\nThis msg from cat class\n");  
13        super.eat();  
14        super.sleep();  
15    }  
16 }
```

SUPER KEYWORD

WITH METHODS

MAIN

OUTPUT

```
public class Main {  
    public static void main(String[] args) {  
        Dog dog = new Dog();  
        dog.show();  
    }  
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

This msg from Dog class
The dog is eating
The dog did not sleep
This msg from cat class
The cat is eating...
Zzz Zzz
PS C:\Users\JOHN\Desktop\Java course>

SUPER KEYWORD

WITH CONSTRUCTORS

يمكننا أيضًا استخدام **Super** مع **SuperClass Constructors** ↗

```
● ● ●  
1 public class Cat {  
2  
3     public Cat(){  
4         System.out.println("Meow Meow");  
5     }  
6  
7     public Cat(String name){  
8         System.out.println("if u pass an name in super keyword i'll execute");  
9     }  
10  
11 }
```

```
● ● ●  
1 public class Dog extends Cat{  
2  
3     public Dog(){  
4  
5         super();  
6         System.out.println("Wof Wof");  
7     }  
8  
9 }
```

SUPER KEYWORD

WITH CONSTRUCTORS

MAIN

OUTPUT

```
● ● ●  
1 public class Main {  
2     public static void main(String[] args) {  
3         Dog dog = new Dog();  
4     }  
5 }  
6  
7 }
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE

Meow Meow
Wof Wof
PS C:\Users\JOHN\Desktop\Java course>

SUPER KEYWORD

WITH CONSTRUCTORS



```
1 public class Cat {  
2  
3     public Cat(){  
4         System.out.println("Meow Meow");  
5     }  
6  
7     public Cat(String name){  
8         System.out.println("if u pass an name in super keyword i'll execute");  
9     }  
10 }  
11 }
```



```
1 public class Dog extends Cat{  
2  
3     public Dog(){  
4  
5         super("Mehdi");  
6         System.out.println("Wof Wof");  
7     }  
8 }  
9 }
```

SUPER KEYWORD

WITH CONSTRUCTORS

MAIN

OUTPUT

```
● ● ●  
1 public class Main {  
2     public static void main(String[] args) {  
3         Dog dog = new Dog();  
4     }  
5     }  
6     }  
7     }  
8 }
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORT

if u pass an name in super keyword i'll execute
Wof Wof
PS C:\Users\JOHN\Desktop\Java course>

METHOD OVERLOADING

METHOD OVERLOADING

☞ تساعد في قابلية قراءة البرنامج و هي عبارة عن كتابة **Method** **الـ** **عده مرات مع تغير عدد انواعهم او ترتيبهم.** او **parameters** **الـ**



```
● ● ●
1 public void show(int ID ,String name ){
2     System.out.println("Your ID : "+ID+"\nYour name : "+name);
3 }
4 public void show(String address,int Salary){
5     System.out.println("Your address : "+address+"\nYour Salary : "+Salary);
6 }
7
```

METHOD OVERLOADING

REAL EXAMPLE

```
1  public class User {  
2      int ID;  
3      int Salary;  
4      String name;  
5      String address;  
6      public User(int ID , String name ,String address,int Salary ){  
7          this.ID = ID;  
8          this.name = name;  
9          this.address = address;  
10         this.Salary = Salary;  
11         show(ID,name);  
12         show(address,Salary);  
13     }  
14  
15     public void show(int ID ,String name ){  
16         System.out.println("Your ID : "+ID+"\nYour name : "+name);  
17     }  
18     public void show(String address,int Salary){  
19         System.out.println("Your address : "+address+"\nYour Salary : "+Salary);  
20     }  
21  
22 }
```

```
1  
2  public int getID() {  
3      return ID;  
4  }  
5  
6  public void setID(int ID) {  
7      this.ID = ID;  
8  }  
9  
10 public int getSalary() {  
11    return Salary;  
12 }  
13  
14 public void setSalary(int salary) {  
15    Salary = salary;  
16 }  
17
```

```
1  public void setSalary(int salary) {  
2      Salary = salary;  
3  }  
4  
5  public String getName() {  
6      return name;  
7  }  
8  
9  public void setName(String name) {  
10    this.name = name;  
11 }  
12  
13  public String getAddress() {  
14      return address;  
15 }  
16  
17  public void setAddress(String address) {  
18      this.address = address;  
19 }  
20 }
```

METHOD OVERLOADING

MAIN

```
1 public class Main {  
2     public static void main(String[] args) {  
3         User user = new User(32018664,"Mehdi","Some Where",250000);  
4         System.out.println("-----");  
5         user.show(user.getID(), user.getName());  
6         user.show(user.getAddress(), user.getSalary());  
7         System.out.println("-----");  
8         user.show(15054661,"Mohamed");  
9         user.show("some where",35000);  
10    }  
11 }
```

OUTPUT

```
Your ID : 32018664  
Your name : Mehdi  
Your address : Some Where  
Your Salary : 250000  
-----  
Your ID : 32018664  
Your name : Mehdi  
Your address : Some Where  
Your Salary : 250000  
-----  
Your ID : 15054661  
Your name : Mohamed  
Your address : some where  
Your Salary : 35000  
○ PS C:\Users\JOHN\Desktop\Java course>
```

ACCESS MODIFIERS

ACCESS MODIFIERS

.Class الـ **Constructor** الـ **Methods** الـ **Data** على الـ **يسلط الضوء**

: انواع

.private •

.public •

.protected •

.Default •

.Encapsulation منادة الـ **Variables** او الـ **Methods** بالنسبة للـ **Objects**

ACCESS MODIFIERS

ACCESS MODIFIERS				
	Class	Package	Outside	SubClass
Public				
Protected				
Default				
Private				

ACCESS MODIFIERS

```
1
2 public class Main {
3     public static void main(String[] args) {
4         PrivateVar privateVar = new PrivateVar();
5         privateVar.
6     }
7 }
8
9 public class PrivateVar {
10    private String name = "Ta3alam Coding";
11    private int age = 20;
12    private boolean isMale = true;
13    private double moy = 19.5;
14
15    public String name = "Ta3alam Coding";
16    int age = 20;
17    protected boolean isMale = true;
18    public double moy = 19.5;
19
20    public void show(){
21        System.out.println("Ta3alam Coding");
22    }
23 }
24
```

المنظور بالنسبة لـ **Main** ↗

ACCESS MODIFIERS

```
● ● ●  
1  public class PrivateVar {  
2      private String name = "Ta3alam Coding";  
3      private int age = 20;  
4  
5      public String name = "Ta3alam Coding";  
6      int age = 20;  
7      protected boolean isMale = true;  
8      public double moy = 19.5;  
9  
10     public void show(){  
11         System.out.println("Ta3alam Coding");  
12     }  
13 }  
14  
15  
16    public class PublicVar extends PrivateVar{  
17  
18  
19 }
```

الكلasse **extend** راح يعمل **هاد** **PublicVar**  **للمتغيرات و الميتوودس المضاءة.**

ACCESS MODIFIERS

```
● ● ●  
1  
2 public class A {  
3     private String name = "Ta3alam Coding";  
4     int age = 20;  
5     protected boolean isMale = true;  
6     public double moy = 19.5;  
7  
8 }  
9  
10 public class B extends A{  
11 }  
12  
13  
14 public class C extends A{  
15 }  
16  
17  
18 public class D extends B{  
19     class B b = new class B();  
20     b.  
21  
22 }  
23 }
```

المنظور بالنسبة لـ **D** لأن **A** ليس **Subclass** **B** ولكن ↗

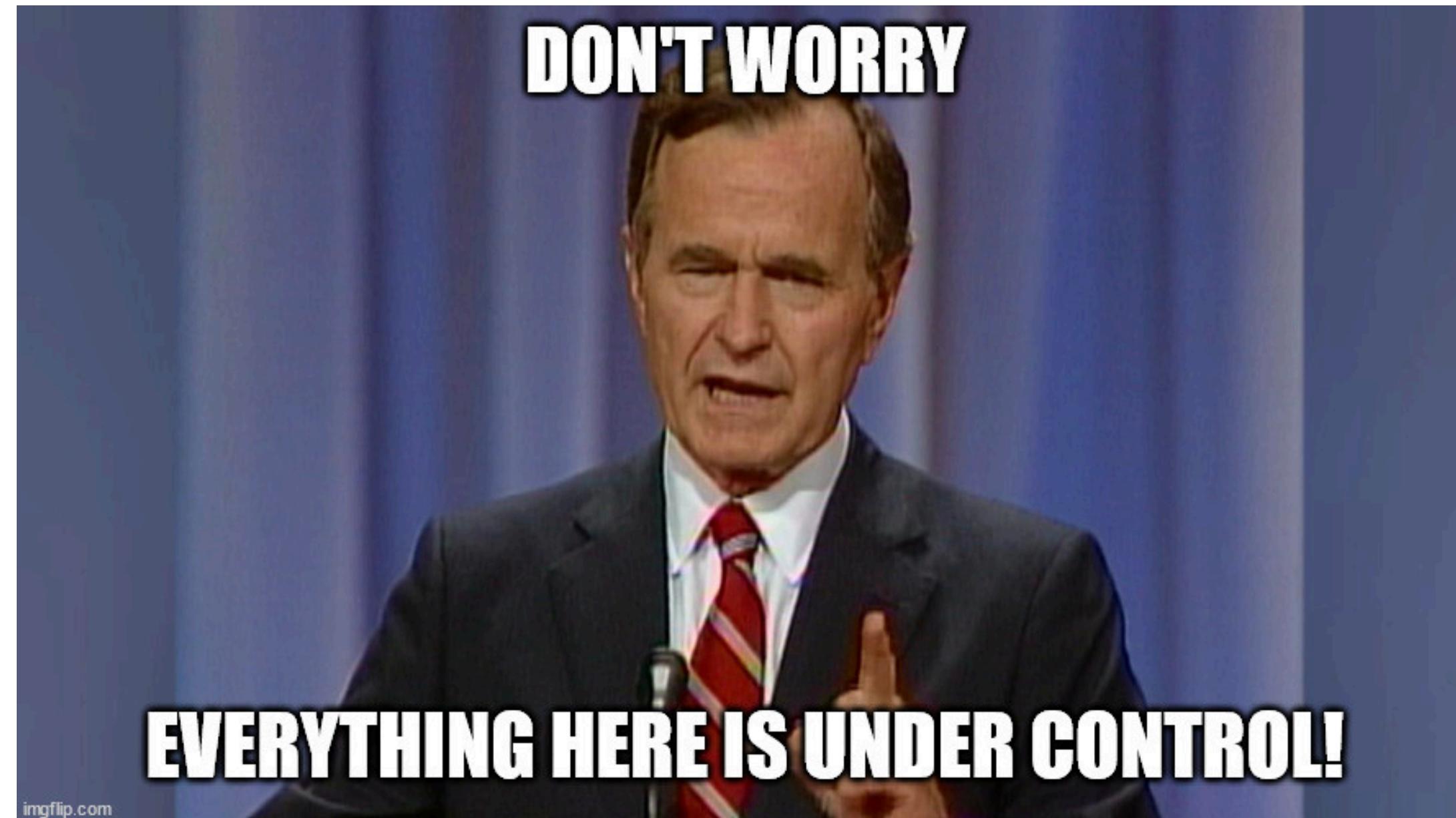
POLYMORPHISM

POLYMORPHISM



POLYMORPHISM 😊

POLYMORPHISM



POLYMORPHISM



POLYMORPHISM

- هو مجرد أسلوب في كتابة الكود و يمكنك تجسيده بعده أشكال مختلفة.
- المقصود من البوليمورفيزم هو بناء دالة تنفذ أوامر مختلفة على حسب الـ Object الذي يمرر لها عند استدعائهما.
- تبني في العادة بداخل الدوال، حيث أننا نقوم ببناء دالة تأخذ Parameter عبارة عن كلاس أو إنترفيس. (ماوريتوش إنترفيس).
- خذ نفسا عميقا و شاهد الكود أدناه.

POLYMORPHISM 😊

POLYMORPHISM

```
● ● ●  
1 public class FirstClass {  
2  
3     public void show(){  
4  
5         System.out.println("Ta3alam Coding");  
6  
7     }  
8  
9 }
```

```
● ● ●  
1 public class SecondClass {  
2  
3     public void print(FirstClass [] arrays){  
4  
5         for (FirstClass first : arrays) {  
6             first.firstClass();  
7         }  
8     }  
9  
10 }  
11
```

POLYMORPHISM 😊

MAIN

OUTPUT



```
1 public class Main {  
2     public static void main(String[] args) {  
3         FirstClass [] name = {new FirstClass(), new FirstClass()};  
4         SecondClass secondClass = new SecondClass();  
5         secondClass.print(name);  
6     }  
7 }
```

PROBLEMS

1

OUTPUT

DEBUG CONSOLE

TERMINAL

Ta3alam Coding

Ta3alam Coding

PS C:\Users\JOHN\Desktop\Java course>

POLYMORPHISM 😊

CODE EXPLANATION

قمنا بعمل **класс** **FirstClass** 📌 و صناعة بداخله **Method show** التي تقوم بطباعة كلمة **Ta3alam Coding**.

```
● ● ●  
1 public class FirstClass {  
2  
3     public void show(){  
4  
5         System.out.println("Ta3alam Coding");  
6     }  
7  
8 }  
9 }
```

POLYMORPHISM 😊

CODE EXPLANATION

قمنا بعمل **класс SecondClass** 🔍

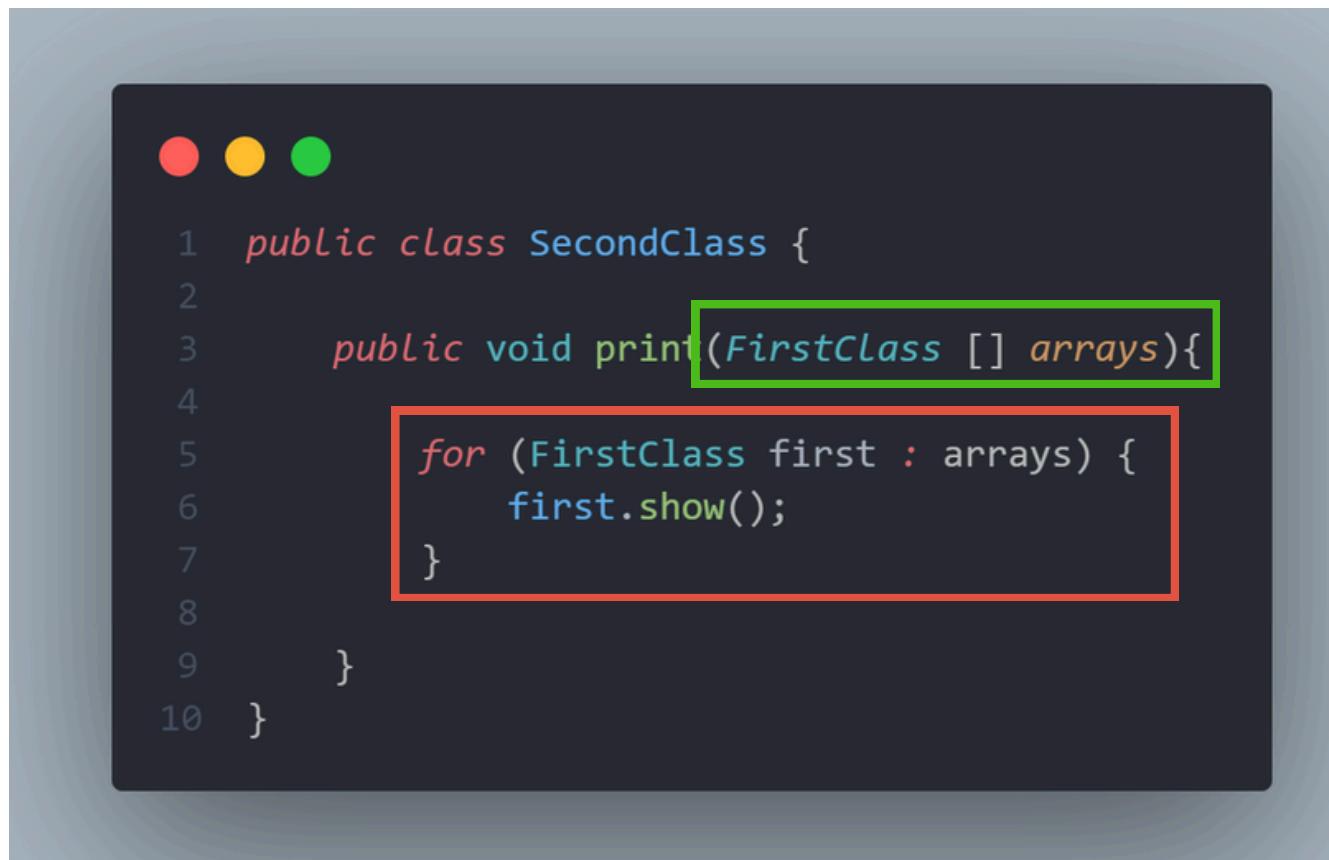
Method Print و صناعة بداخله **SecondClass** 🔍

```
1 public class SecondClass {  
2  
3     public void print(FirstClass [] arrays){  
4  
5         for (FirstClass first : arrays) {  
6             first.firstClass();  
7         }  
8     }  
9 }  
10 }  
11 }
```

POLYMORPHISM 😊

CODE EXPLANATION

و بما انه فهو يحتاج **Array** 🔍 لوصول الى كل القيم التي بداخله.



```
1 public class SecondClass {  
2       
3     public void print(FirstClass [] arrays){  
4           
5             for (FirstClass first : arrays) {  
6                 first.show();  
7             }  
8         }  
9     }  
10 }
```

POLYMORPHISM 😊

CODE EXPLANATION

وبما اتنا نستعمل **Foreach** فاعلينا تحديد **DataType** لـ **.name** في هذه الحالة هو ↗

```
for ( DataType name : arrayName ) {
```

//Code

}

في هذه الحالة الـ **Array** يحتوي على **Objects** من نوع **FirstClass** و بالتالي الـ **name** سيكون من نوع **DataType** (اذا ماقهمنتش ارجع لـ 1 اخر 5 صفحات) .**FirstClass**

```
public class SecondClass {
    public void print(FirstClass [] arrays){
        for (FirstClass first : arrays) {
            first.show();
        }
    }
}
```

POLYMORPHISM 😊

CODE EXPLANATION

وبما ان المتغير **first** من نوع **Object** فانه اصبح **Object** تابع له ، وبالتالي الـ **.FirstClass** او استدعاء اي متغير او **Method** موجود في الـ **.FirstClass** او **.Method show** و في هذه الحالة نريد ان نستدعي الـ **.Method show** ↗

```
● ● ●

1 public class SecondClass {
2
3     public void print(FirstClass [] arrays){
4
5         for (FirstClass first : arrays) {
6             first.show();
7         }
8     }
9 }
10 }
```

POLYMORPHISM 😊

CODE EXPLANATION

قمنا بإنشاء المتغير  من نوع **Array name** من نوع **FirstClass** اي انه سوف يحتوي بداخله على **Objects**

```
● ● ●
1 public class Main {
2     public static void main(String[] args) {
3
4         FirstClass [] name = {new FirstClass(), new FirstClass()};
5
6         SecondClass secondClass = new SecondClass();
7
8         secondClass.print(name);
9
10    }
11 }
```

POLYMORPHISM 😊

CODE EXPLANATION

.Objects FirstClass قيمة تحتوي على 2 من ال Array name ↗

```
● ● ●  
1 public class Main {  
2     public static void main(String[] args) {  
3  
4         FirstClass [] name = {new FirstClass(), new FirstClass()};  
5  
6         SecondClass secondClass = new SecondClass();  
7  
8         secondClass.print(name);  
9  
10    }  
11 }
```

FirstClass [] name = {Object , Object};

index

0 1

POLYMORPHISM 😊

CODE EXPLANATION

.Method print من اجل استخدام الـ **SecondClass** لـ **Object** **بصورة** 

```
● ● ●  
1 public class Main {  
2     public static void main(String[] args) {  
3  
4         FirstClass [] name = {new FirstClass(), new FirstClass()};  
5  
6         SecondClass secondClass = new SecondClass();  
7  
8         secondClass.print(name);  
9  
10    }  
11 }
```

POLYMORPHISM

CODE EXPLANATION

بعد تمرير الـ **Object** إلى داخل الـ **FirstClass arrays** الـ **Method print** أصبح لدينا كود هكذا :

print(arrays) ->

FirstClass arrays [] = {Object , Object};

```
for ( FirstClass first : arrays ) {
```

index 0 1

first.show();

}

POLYMORPHISM 😊

CODE EXPLANATION

EXECUTION OF FOREACH 01: **Object , Object**

first = Object;

Object.show();

show() -> Sout("Ta3alam Coding");

Output :

Ta3alam Coding

EXECUTION OF FOREACH 02: **Object , Object**

first = Object;

Object.show();

show() -> Sout("Ta3alam Coding");

Output :

Ta3alam Coding

Ta3alam Coding

***Note : Sout() shortHand of System.out.println();**

BY REFRENCE / BY VALUE

BY VALUE

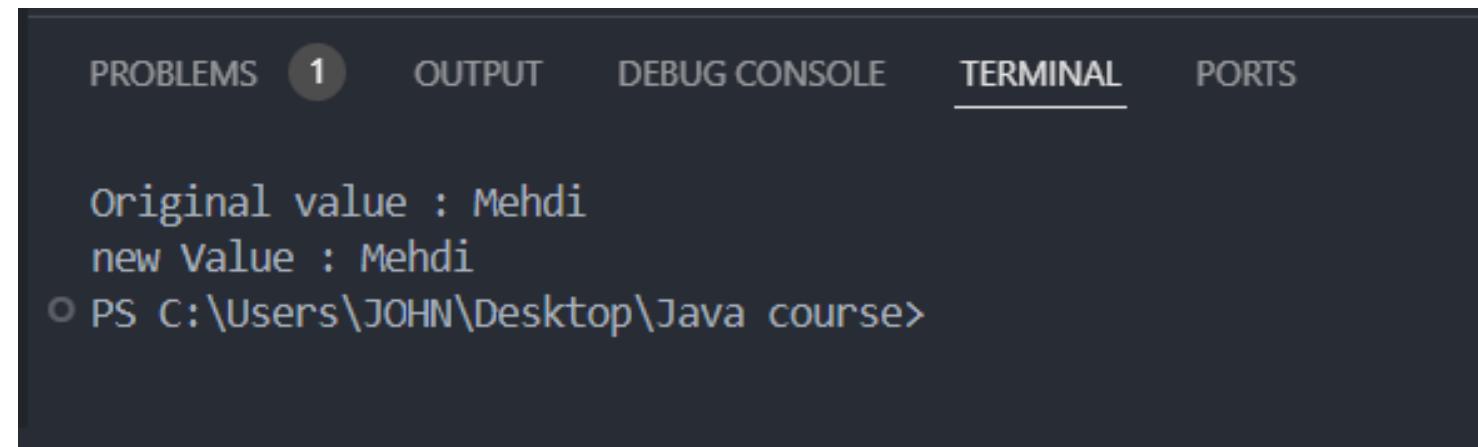
كل مناداة في جافا تكون **.by copy** ↗
شاهد الكود أدناه : ↗

```
● ● ●
1 public class byValue {
2
3     String name = "Mehdi";
4
5     public void addName(String Lastname){
6         name = name + " " + Lastname;
7     }
8     //method that tries to change the name value to Mehdi bch
9 }
```

```
● ● ●
1 public class Main {
2     public static void main(String[] args) {
3
4         byValue byValue = new byValue();
5
6         String myName = byValue.name;
7
8         System.out.println("Original value : "+ myName); // we expect Mehdi
9
10        byValue.addName(" Bch");
11
12        System.out.println("new Value : "+myName); // we expect Mehdi Bch
13    }
14 }
```

BY REFRENCE / BY VALUE

OUTPUT



A screenshot of a terminal window from a code editor. The window has tabs at the top: PROBLEMS (1), OUTPUT, DEBUG CONSOLE, TERMINAL (underlined), and PORTS. The terminal output shows the following text:

```
Original value : Mehdi
new Value : Mehdi
○ PS C:\Users\JOHN\Desktop\Java course>
```

BY REFRENCE / BY VALUE

BY VALUE

لماذا ! لم تتغير القيمة ! ➔

: لا تتغير القيمة حتى ولو حدث عليها تغيير داخل الـ **Method**, لأن الـ **Method** تقوم باخذ ما تم ادخاله لها و استنساخه , و تقوم بالتعديل و استخدام تلك النسخة وليس القيمة التي تم ادخالها لها.

ملاحظة : تعيش النسخة فقط داخل الـ **Method**.

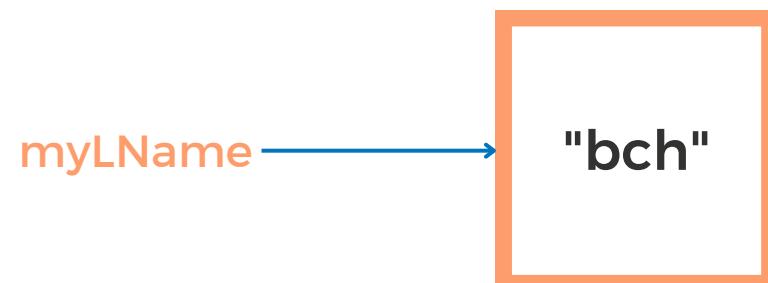
BY REFRENCE / BY VALUE

BEHIND THE SCENES

```
String myLName = " bch";
```

```
byValue.addName();
```

قمنا بإنشاء متغير myLName و أدخلناه كـ parameter للـ addName Method



```
addName Method → addName
```

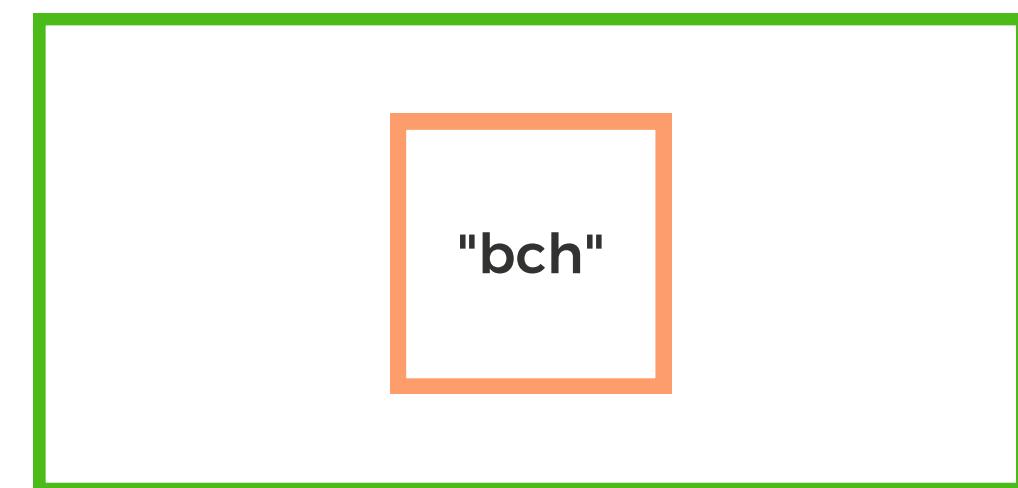
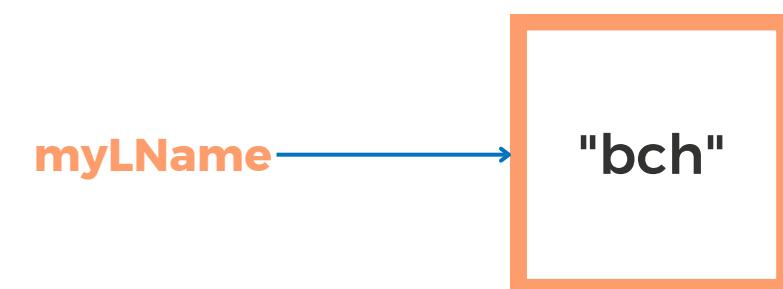


BY REFRENCE / BY VALUE

BEHIND THE SCENES

```
String myLName = " bch";
```

```
byValue.addValue(myLName);
```



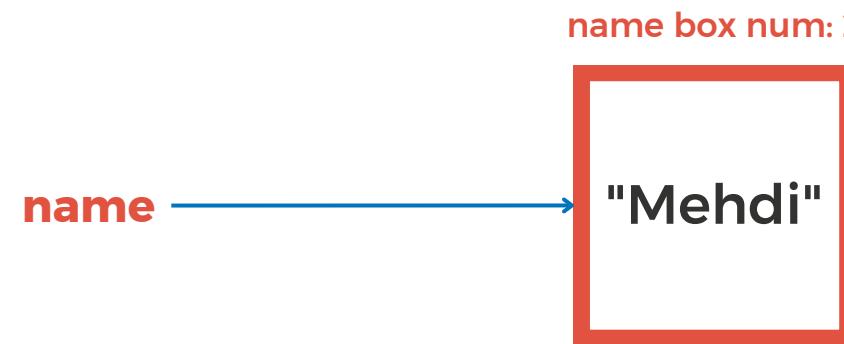
قمنا بإنشاء متغير `myLName` و ادخلناه كـ `parameter` للـ `method .addName` ↗

BY REFRENCE / BY VALUE

BEHIND THE SCENES

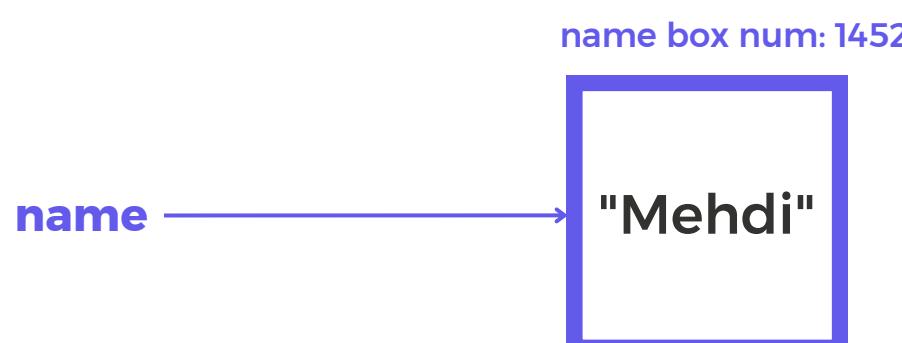
داخل الكلاس **byValue** يوجد متغير **name** و ميثود **.addName** ↗

```
class byValue {  
    String name = "Mehdi";  
  
    addName(myLName) {  
        name = name + myLName;  
    }  
}
```



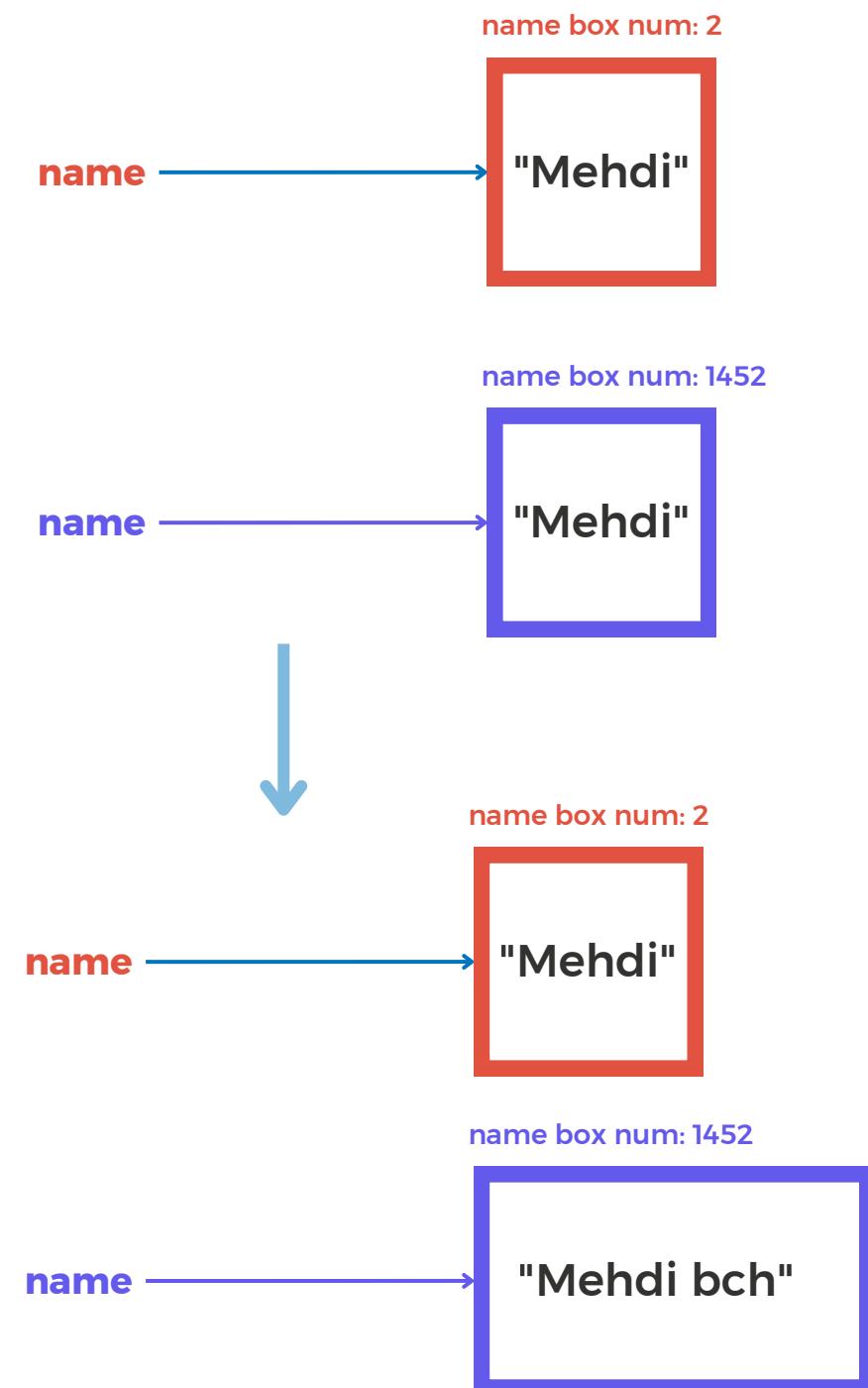
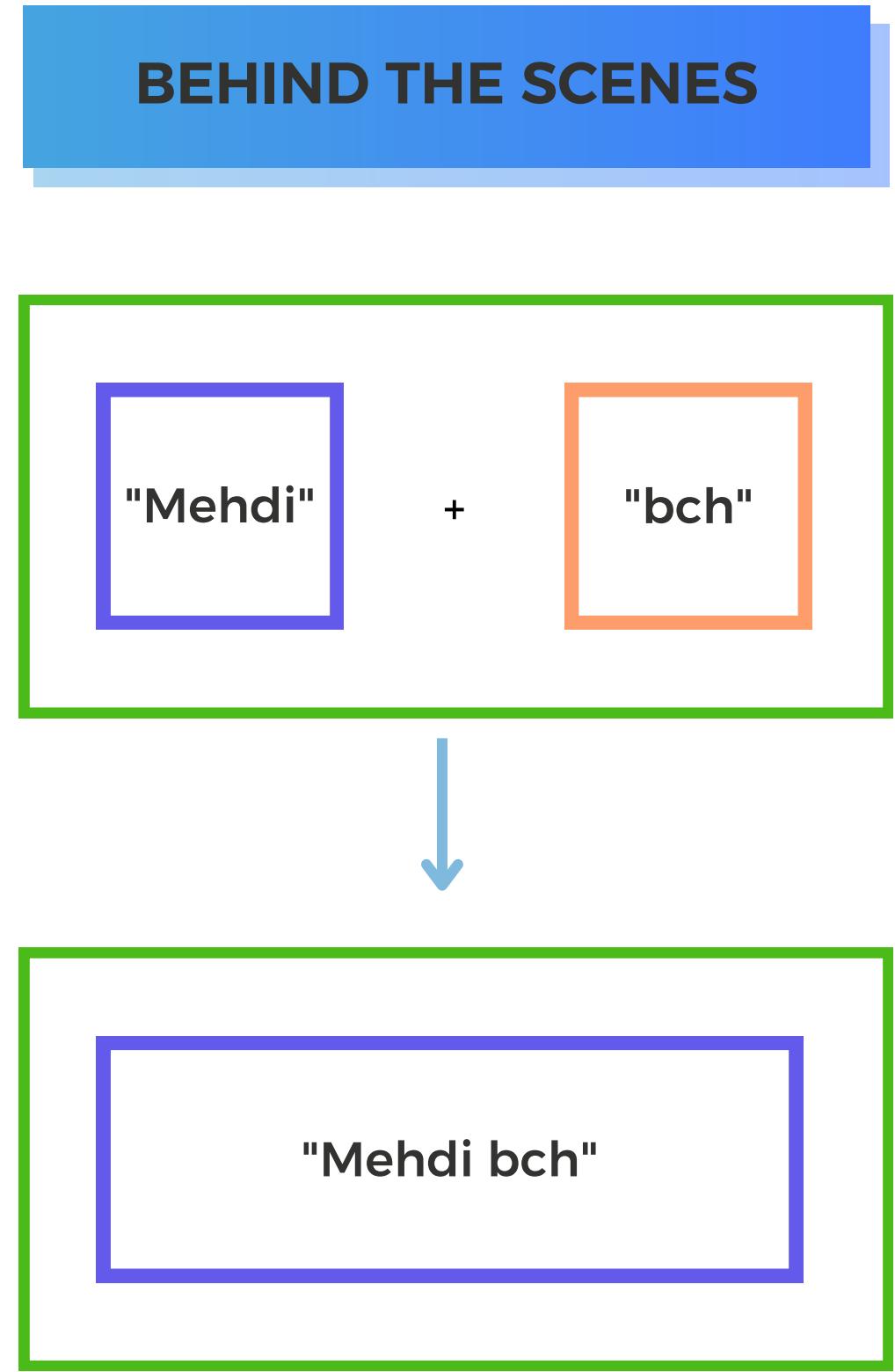
↗ قبل عمل الميثود **.addName** على المتغير **name** تقوم

باستنساخه و العمل بتلك النسخة من **name** ↗



BY REFRENCE / BY VALUE

```
class byValue {  
    String name = "Mehdi";  
  
    addName(myLName ) {  
        name = name + myLName ;  
    }  
}
```



BY REFRENCE / BY VALUE

BEHIND THE SCENES

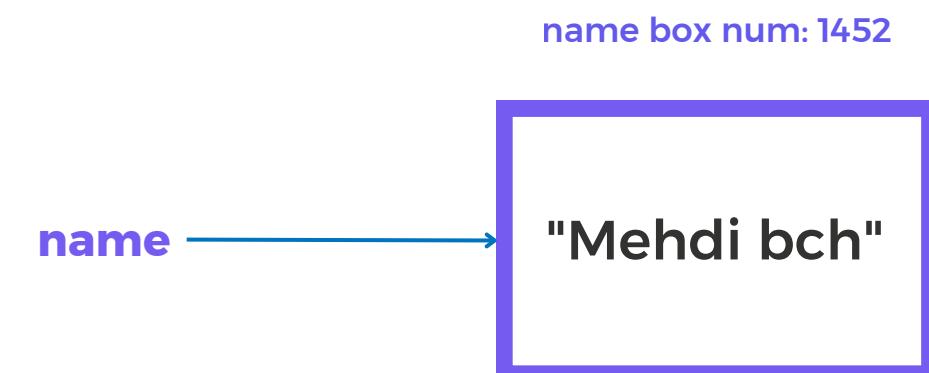
```
String myName = byValue.name
```

```
String myLName = " Bch";
```

```
System.out.println("Original value : "+ myName); //Mehdi
```

```
byValue.addName(myLName);
```

```
System.out.println("new Value : "+myName); //Mehdi bch
```



BY REFRENCE / BY VALUE

BY REFRENCE

• تغيير القيمة اذا حدث عليها تغيير داخل الـ Method 

في جافا الـ Objects فقط التي تكون دائمًا **Object by reference**, أي بمعنى آخر أي **Object** يتم إدخاله كـ **Parameter** داخل الـ **Method** لا يتم استنساخه ولكن استعماله كما هو، أي سيحدث عليه تغيير.

BY REFRENCE / BY VALUE

BY REFRENCE



```
1 public class byReference {  
2  
3     String name = "Mehdi";  
4  
5     public void addName(byReference myName , String LastName){  
6  
7         myName.name = myName.name + lastName;  
8  
9     }  
10 }
```

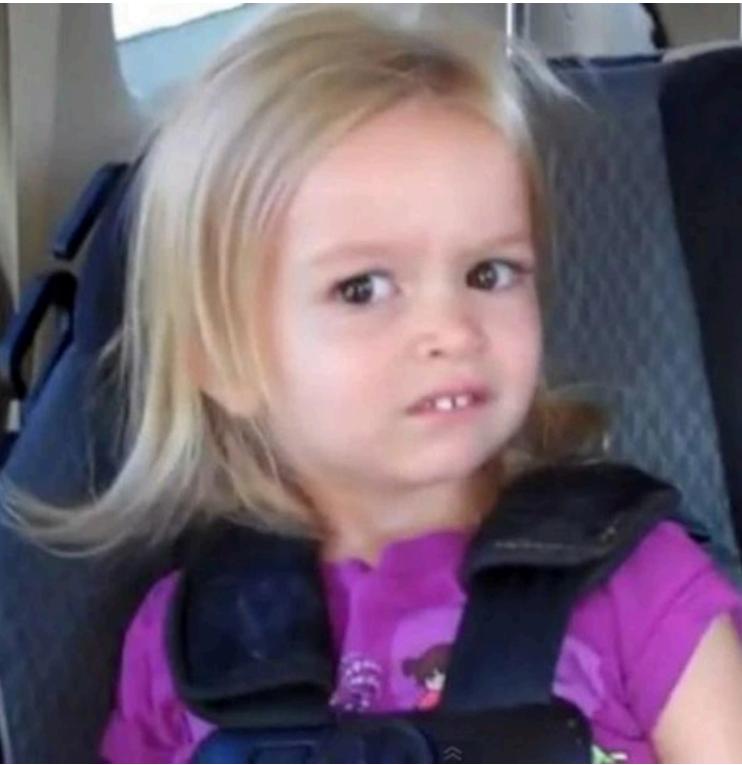


```
1 public class Main {  
2     public static void main(String[] args) {  
3  
4         byReference byRf = new byReference();  
5  
6         byReference myName = new byReference();  
7  
8         String myLName = " Bch";  
9  
10        System.out.println("Original value : "+ myName.name); // we expect Mehdi  
11  
12        byRf.addName(myName,myLName);  
13  
14        System.out.println("new Value : "+myName.name); // we expect Mehdi Bch  
15    }  
16 }
```

BY REFRENCE / BY VALUE

BY REFRENCE

my reaction ida 9oteli mafehamtch :))



OUTPUT

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS

Original value : Mehdi
new Value : Mehdi Bch
PS C:\Users\JOHN\Desktop\Java course>

nefahmek bel derahme chriki :))

BY REFRENCE / BY VALUE

CODE EXPLANATION

عندنا كلاس اسمه **byReference** و نريد Method **addName** و Global variable (**name**) يحتوي على **byReference** عمل **تغير على القيمة الحقيقة** لل**name** الموجود خارج الـ **Method**.
☞

```
1 public class byReference {  
2       
3     String name = "Mehdi";  
4       
5     public void addName(byReference myName , String lastName){  
6           
7             myName.name = myName.name + lastName;  
8           
9     }  
10 }
```

BY REFRENCE / BY VALUE

CODE EXPLANATION

تستقبل الـ **Method addName**  **Object myName** ذو النوع **Object**  **Parameters**  **على (instance اى اى) byReference** 

Object  **عن myName اذا** 

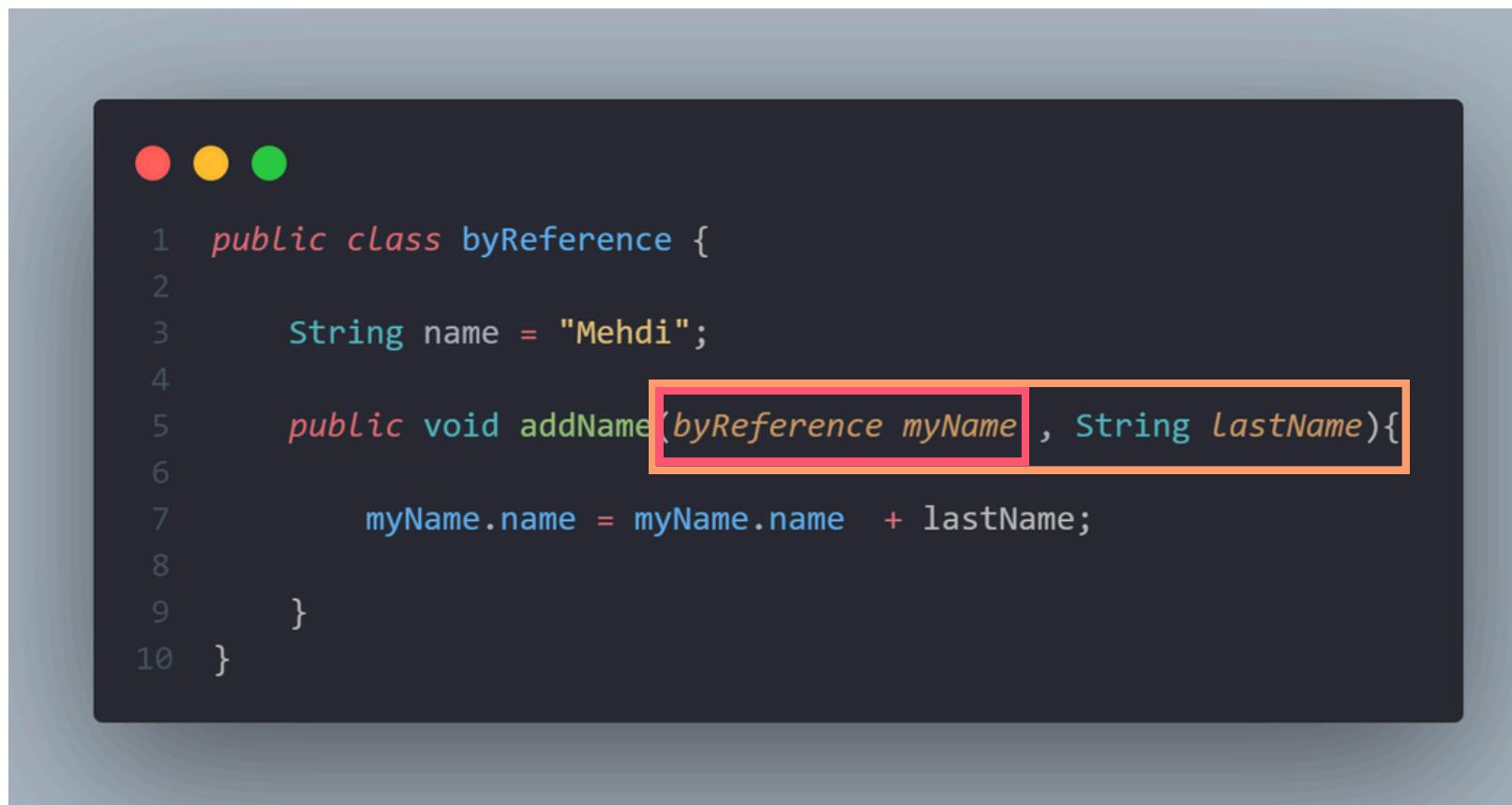


```
1 public class byReference {
2
3     String name = "Mehdi";
4
5     public void addName(byReference myName, String lastName){
6
7         myName.name = myName.name + lastName;
8
9     }
10 }
```

BY REFRENCE / BY VALUE

CODE EXPLANATION

وبما ان الـ **(byReference** لـ **instance** هو **myName**) **byReference** ذو النوع **Object myName**👉
فانه يستطيع الوصول الى كل **Methods** و كل **المتغيرات** الموجودة بداخل **الكلاس**

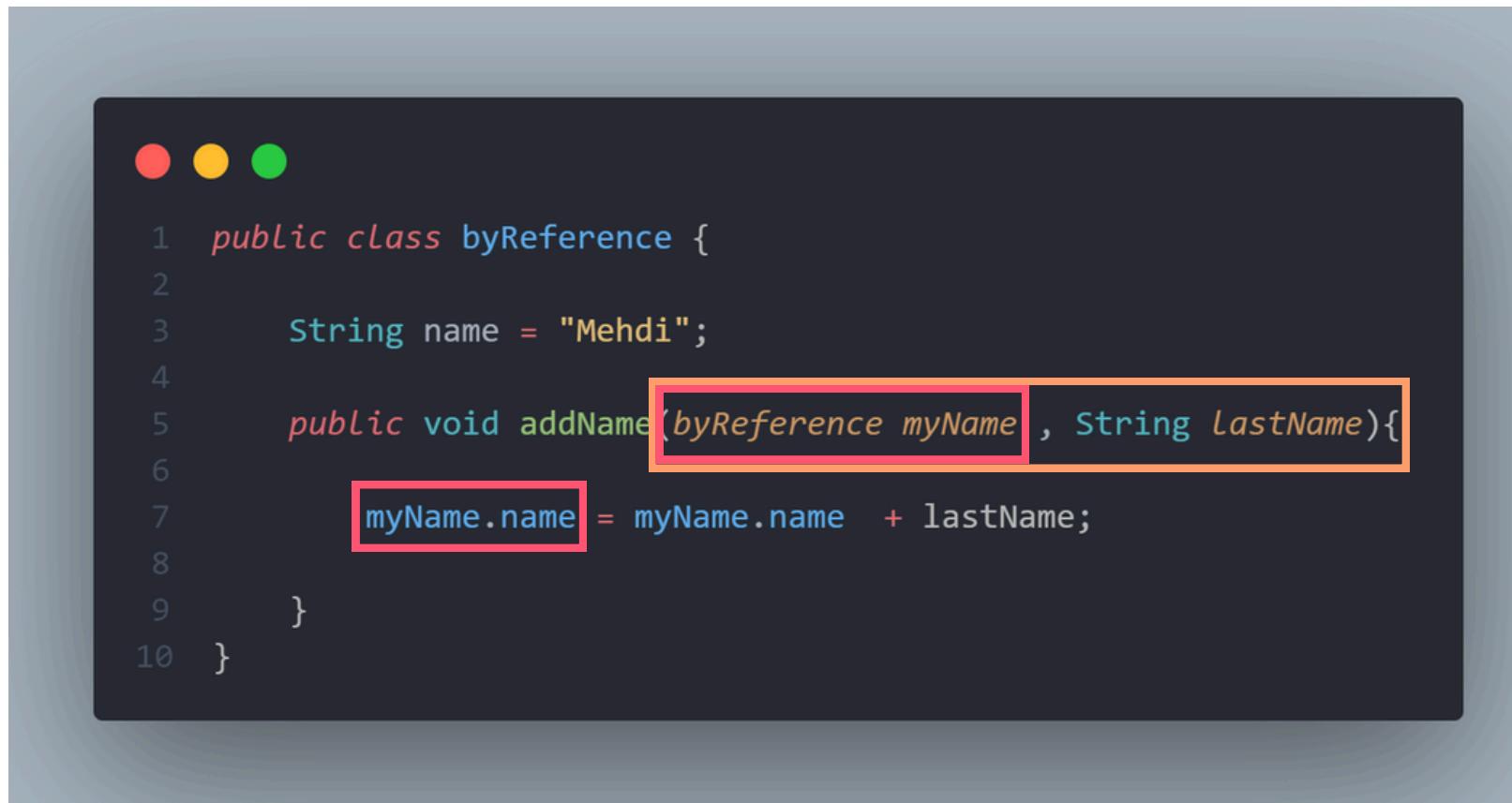


```
1 public class byReference {
2
3     String name = "Mehdi";
4
5     public void addName(byReference myName, String lastName){
6
7         myName.name = myName.name + lastName;
8
9     }
10 }
```

BY REFRENCE / BY VALUE

CODE EXPLANATION

إذا يستطيع الوصول الى المتغير **name** الموجود بداخل **الكلاس** و استخدامة .
قلنا : اجلب ال **name** الموجود بداخل **الكلاس** **byReference** و اعمل عليه هذه التغييرات .



```
1 public class byReference {
2
3     String name = "Mehdi";
4
5     public void addName(byReference myName, String lastName){
6
7         myName.name = myName.name + lastName;
8
9     }
10 }
```

The code editor interface is visible at the top left, showing three colored dots (red, yellow, green) as window control buttons. The Java code is displayed in the main pane. Lines 5 and 7 are highlighted with red boxes, indicating the specific parts of the code being discussed.

BY REFRENCE / BY VALUE

CODE EXPLANATION

إذا يستطيع الوصول الى المتغير **name** الموجود بداخل **الكلاس** و استخدامة .
قلنا : اجلب ال **name** الموجود بداخل **الكلاس** **byReference** و اعمل عليه هذه التغييرات .



```
1 public class byReference {
2
3     String name = "Mehdi";
4
5     public void addName(byReference myName, String lastName){
6
7         myName.name = myName.name + lastName;
8
9     }
10 }
```

The code editor interface is visible at the top left, showing three colored dots (red, yellow, green) as window control buttons. The Java code is displayed in the main pane. Lines 5 and 7 are highlighted with red boxes, indicating the specific parts of the code being discussed.

BY REFRENCE / BY VALUE

CODE EXPLANATION

قم بـ **Object byRef** كالمعتاد من أجل استخدام مابداخله من **متغيرات و ميثودس**.

قم بـ **Object myName** كالمعتاد من أجل استخدام مابداخله من **متغيرات و ميثودس**.

قم بـ **عمل متغير myLName** يحمل قيمة **.Bch**.

```
1 public class Main {  
2     public static void main(String[] args) {  
3         byReference byRef = new byReference();  
4         byReference myName = new byReference();  
5         String myLName = " Bch";  
6         System.out.println("Original value : "+ myName.name); // we expect Mehdi  
7         byRef.addName(myName,myLName);  
8         System.out.println("new Value : "+myName.name); // we expect Mehdi Bch  
9     }  
10 }
```

BY REFRENCE / BY VALUE

CODE EXPLANATION

👉 قمنا بجلب **Object myName** المتغير **name** الموجود بداخل **الكلasse** **byReference** بواسطة الـ **Object byRf** وطباعته.

👉 بعد ذلك قمنا باستدعاء الـ **Method addName** بواسطة الـ **Object byRf** و ادخال الـ **Object myName** و المتغير **myLName** كـ **Parameters** كـ **myLName** و تقوم **الميثود** بـ **تغيير قيمة .name**.

```
1 public class Main {  
2     public static void main(String[] args) {  
3         byReference byRf = new byReference();  
4         byReference myName = new byReference();  
5         String myLName = " Bch";  
6         System.out.println("Original value : "+ myName.name); // we expect Mehdi  
7         byRf.addName(myName, myLName);  
8         System.out.println("new Value : "+myName.name); // we expect Mehdi Bch  
9     }  
10 }
```

BY REFRENCE / BY VALUE

CODE EXPLANATION

قمنا بجلب المتغير **name** الموجود بداخل **الكلasse** **byReference** بواسطة **Object myName** بعد عمل **الميثود** وطباعته.

```
1 public class Main {
2     public static void main(String[] args) {
3
4         byReference byRf = new byReference();
5
6         byReference myName = new byReference();
7
8         String myLName = " Bch";
9
10        System.out.println("Original value : "+ myName.name); // we expect Mehdi
11
12        byRf.addName(myName,myLName);
13
14        System.out.println("new Value : "+myName.name); // we expect Mehdi Bch
15    }
16 }
```

BY REFRENCE / BY VALUE

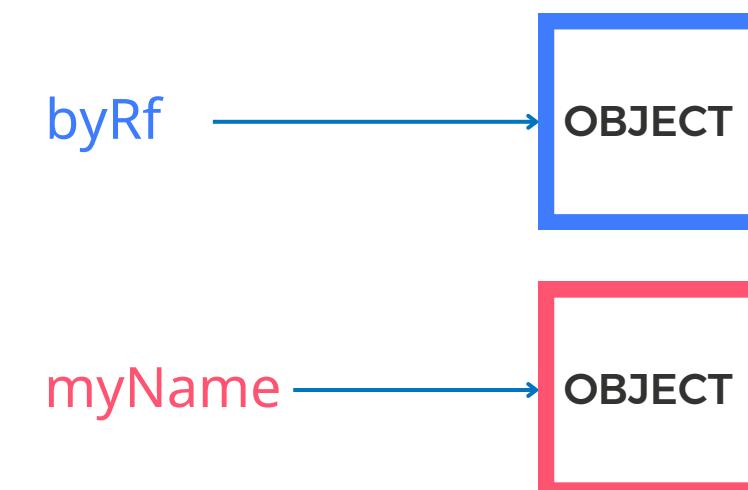
BEHIND THE SCENES

عند صناعة Object لـ **Methods** معين, ذلك الـ **Object** سيحمل جميع المتغيرات و جميع الـ **الموجودة** بداخل هذا **الكلasse**, اي يستطيع الوصول الى اي **Method** او اي **متغير** او اي **التابعة** لـ **الكلasse** و تنفيذها.

```
byReference byRf = new byReference();
```

```
byReference myName = new byReference();
```

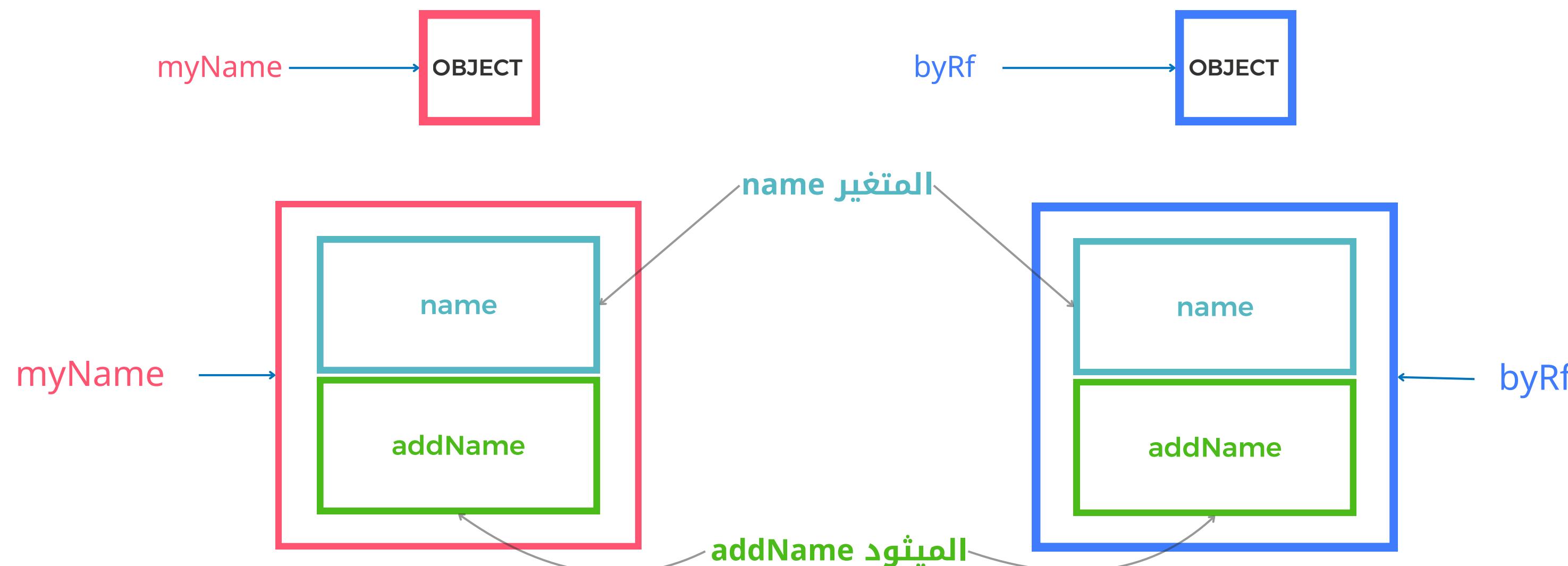
```
String myLName = " Bch";
```



BY REFRENCE / BY VALUE

BEHIND THE SCENES

في هذه الحالة **الكلasse** يحتوي على **متغير واحد** و **ميثود واحدة** فقط.



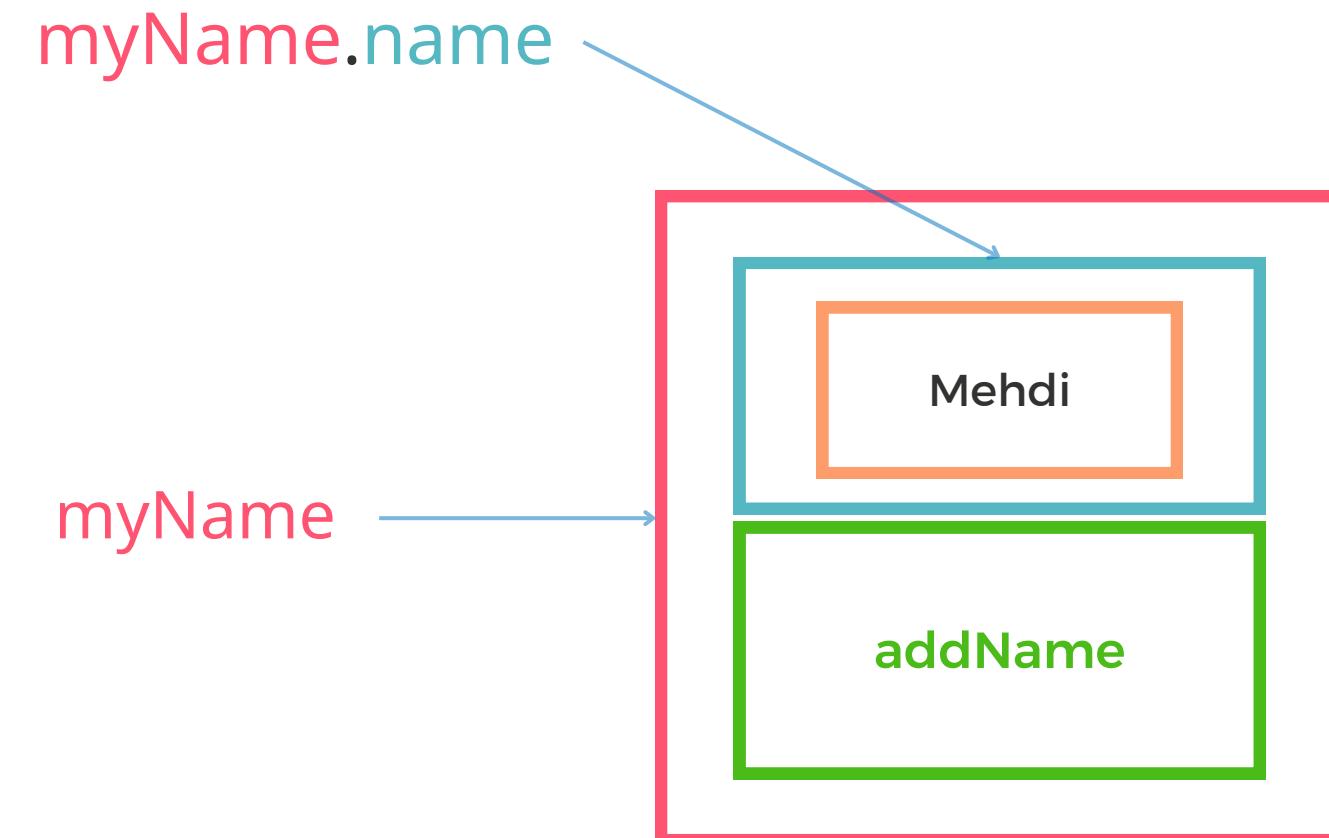
BY REFRENCE / BY VALUE

BEHIND THE SCENES

لجلب اي متغير او ميثود ما من Object نقوم بـ **Object.variableName/MethodName**

```
System.out.println("Original value : "+ myName.name);
```

Original value : Mehdi



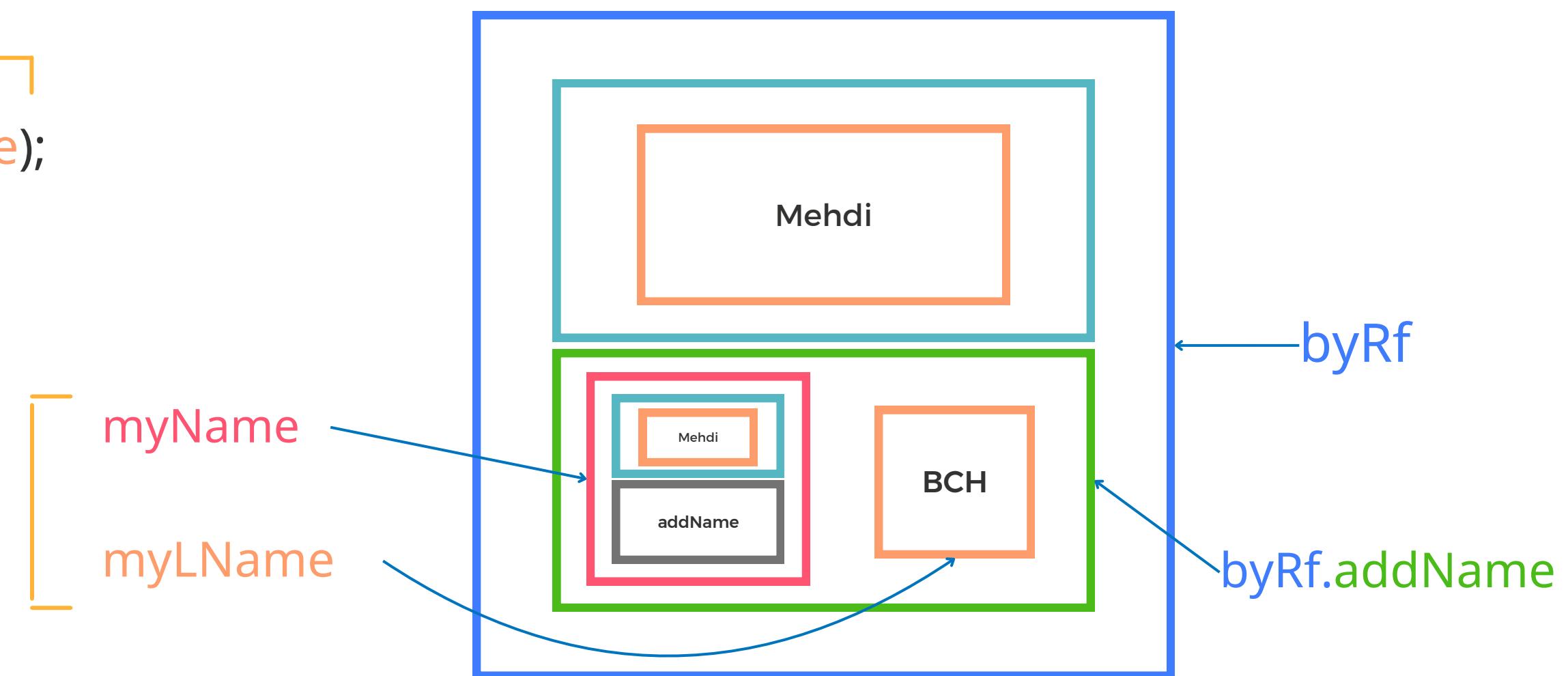
BY REFRENCE / BY VALUE

BEHIND THE SCENES

قمنا باستدعاء الميثود addName في الـ myName Object و ادخل him في myLName

Method addName الى داخل الـ Parameters كـ myLName

```
byRf.addName(myName,myLName);
```

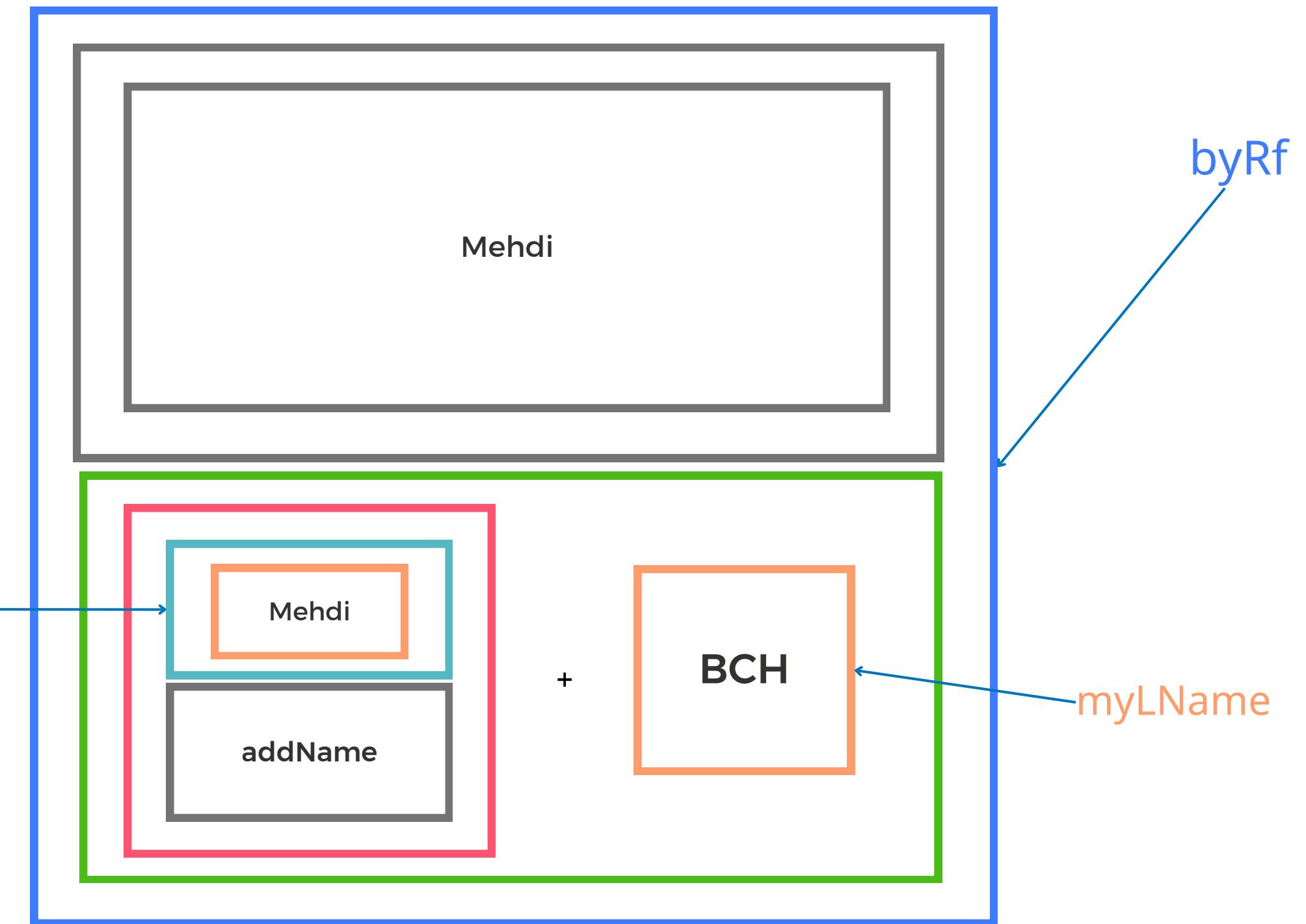


BY REFRENCE / BY VALUE

BEHIND THE SCENES

```
byRef.addName(myName , myLName){  
    myName.name = myName.name + myLName;  
}
```

myName.name

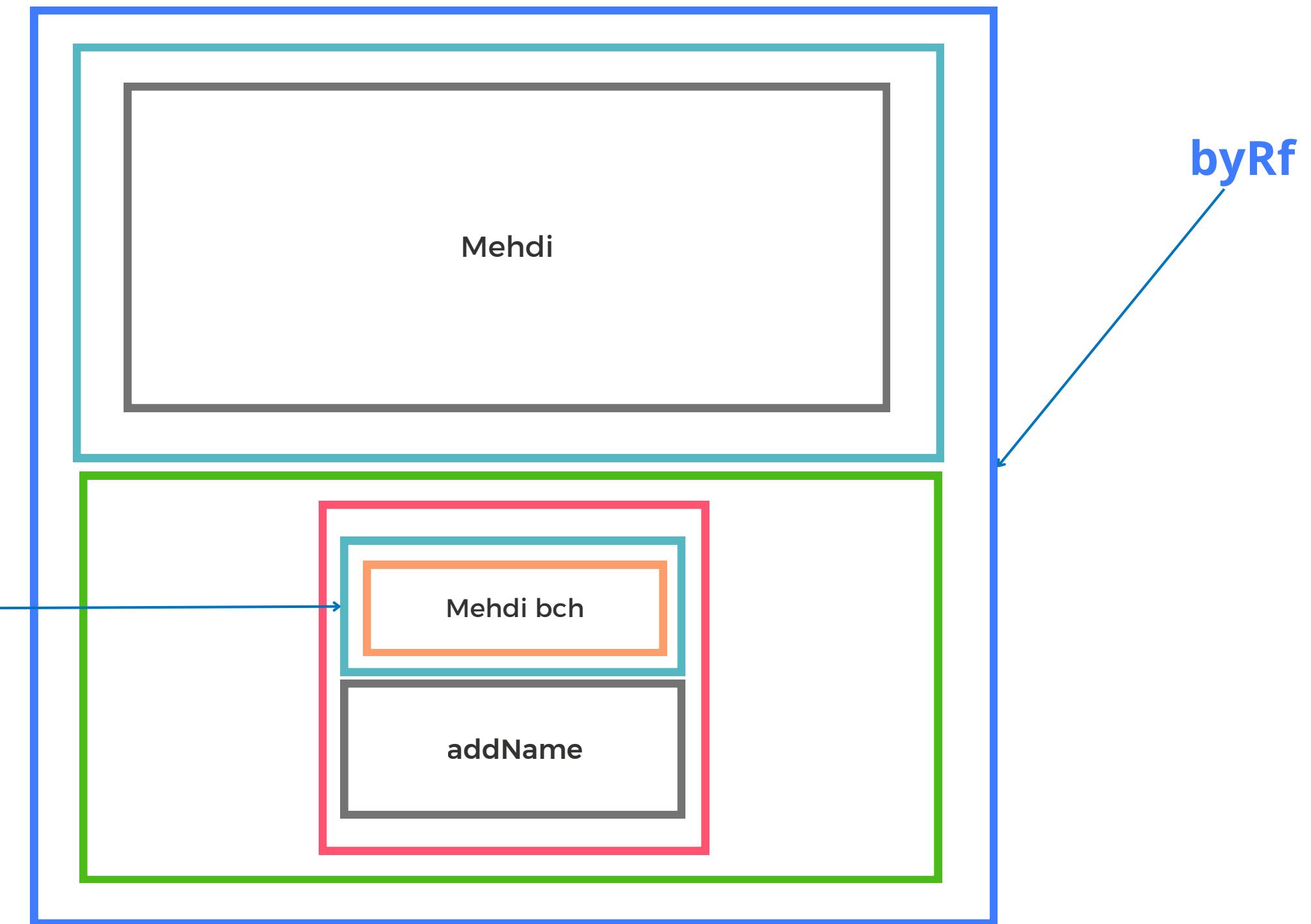


BY REFRENCE / BY VALUE

BEHIND THE SCENES

```
byRef.addName(myName , myLName){  
    myName.name = myName.name + myLName;  
}
```

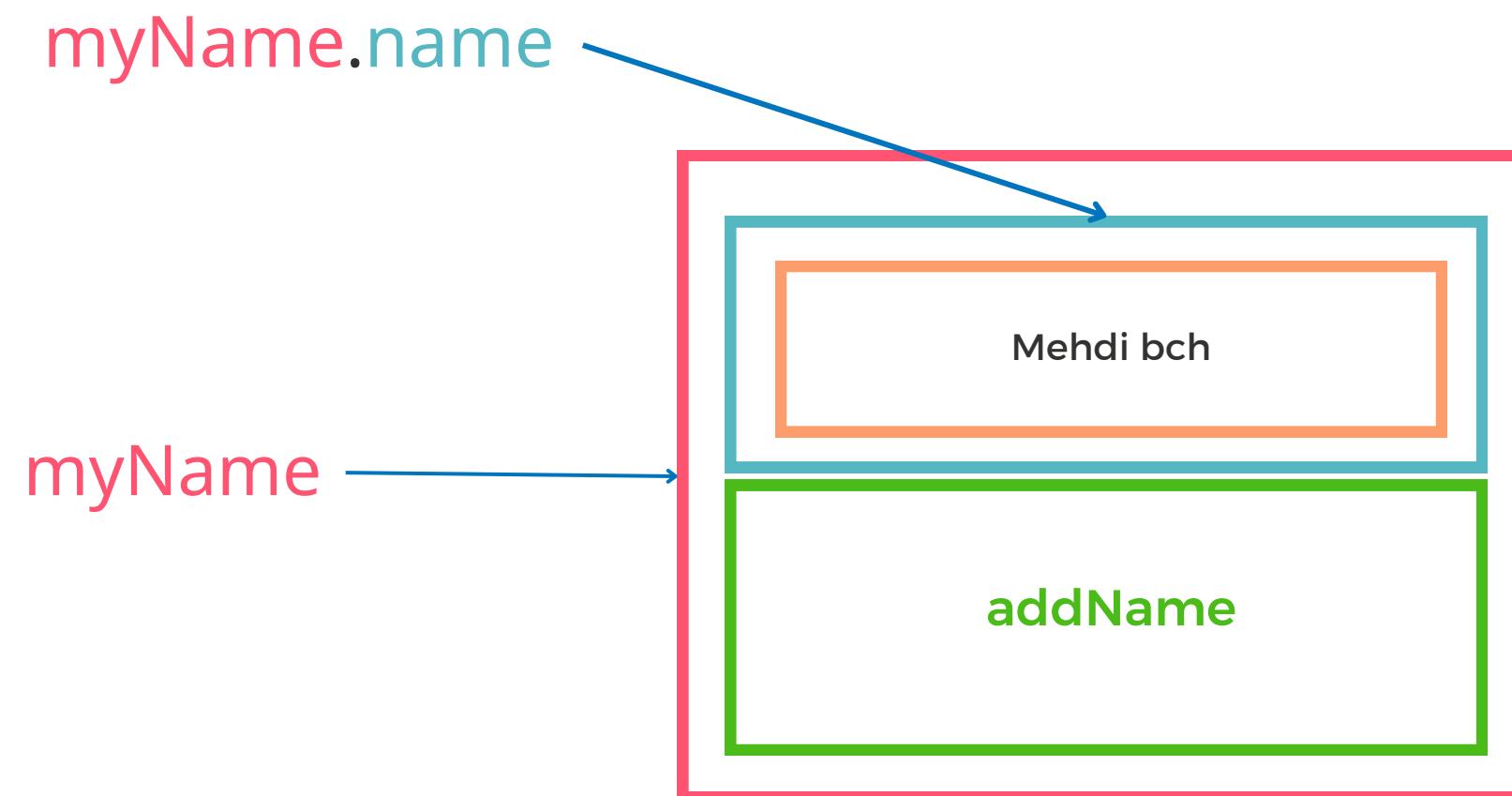
myName.name



BY REFRENCE / BY VALUE

BEHIND THE SCENES

بعد عمل الـ Method addName ↗
يصبح الـ myName Object ذا :

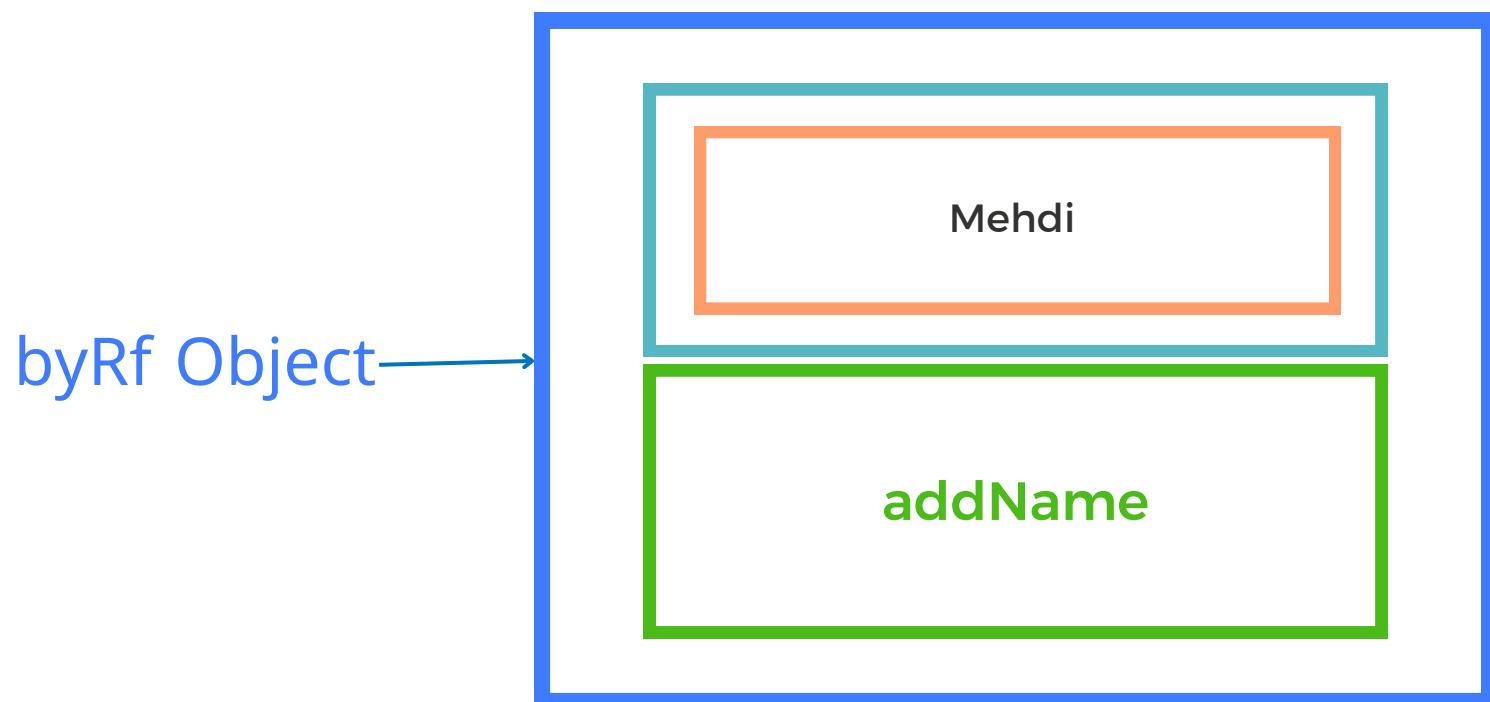
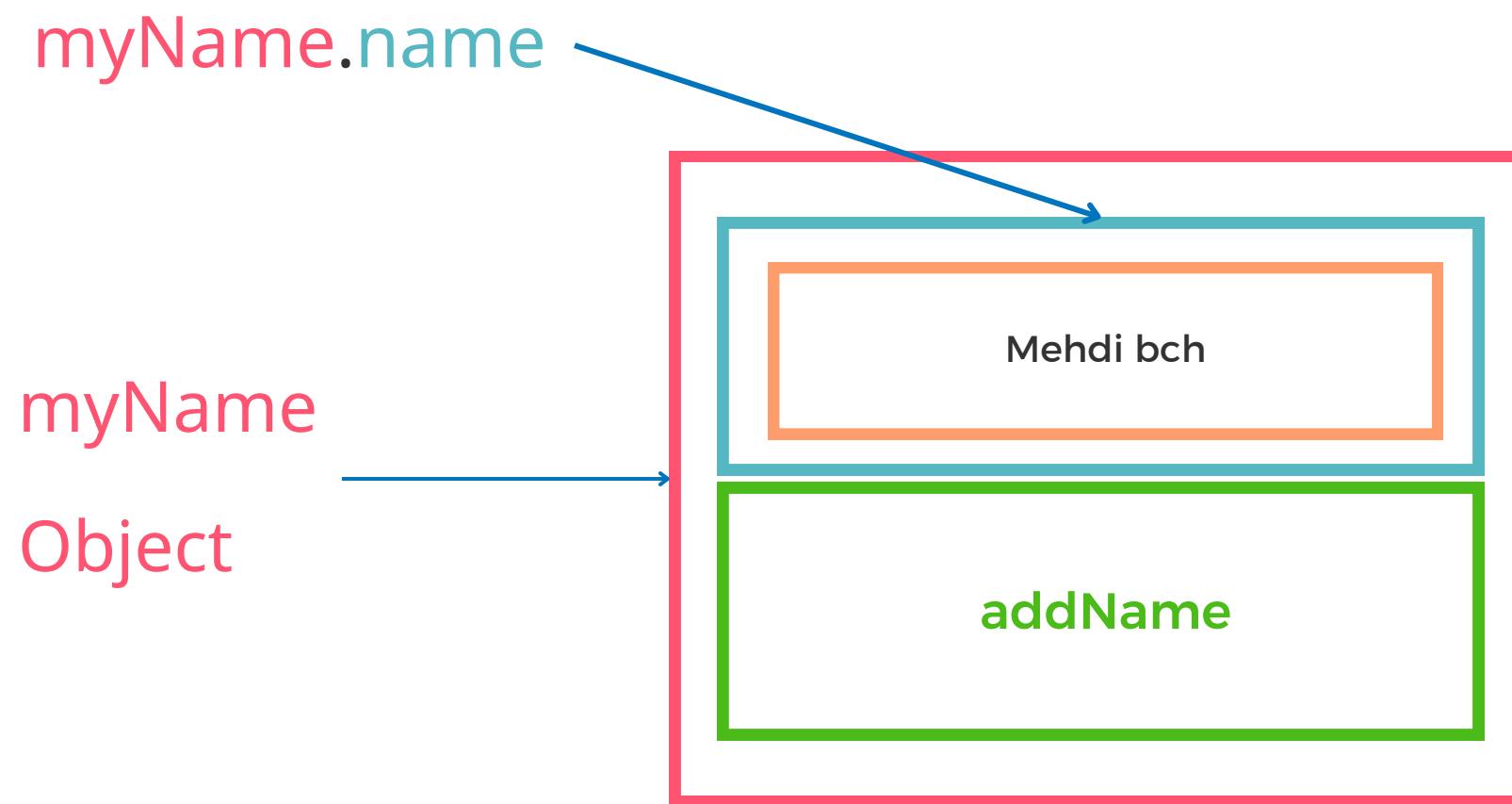


BY REFRENCE / BY VALUE

BEHIND THE SCENES

```
System.out.println("new Value : "+myName.name);
```

new Value : Mehdi bch



BY REFRENCE / BY VALUE

REMINDER!

العبارة عن **Object** (تفكرت كي قتلك في PDF 01 ابقي شافي عليها), اذا مدام **Object** فادئما
عندما يتم ادخاله على شكل **Parameter** يكون **by Reference**. بمعنى اخر اي تعديل عليه داخل **الميثود**
سيكون عنده تأثير.

EXAMPLE :



BY REFRENCE / BY VALUE

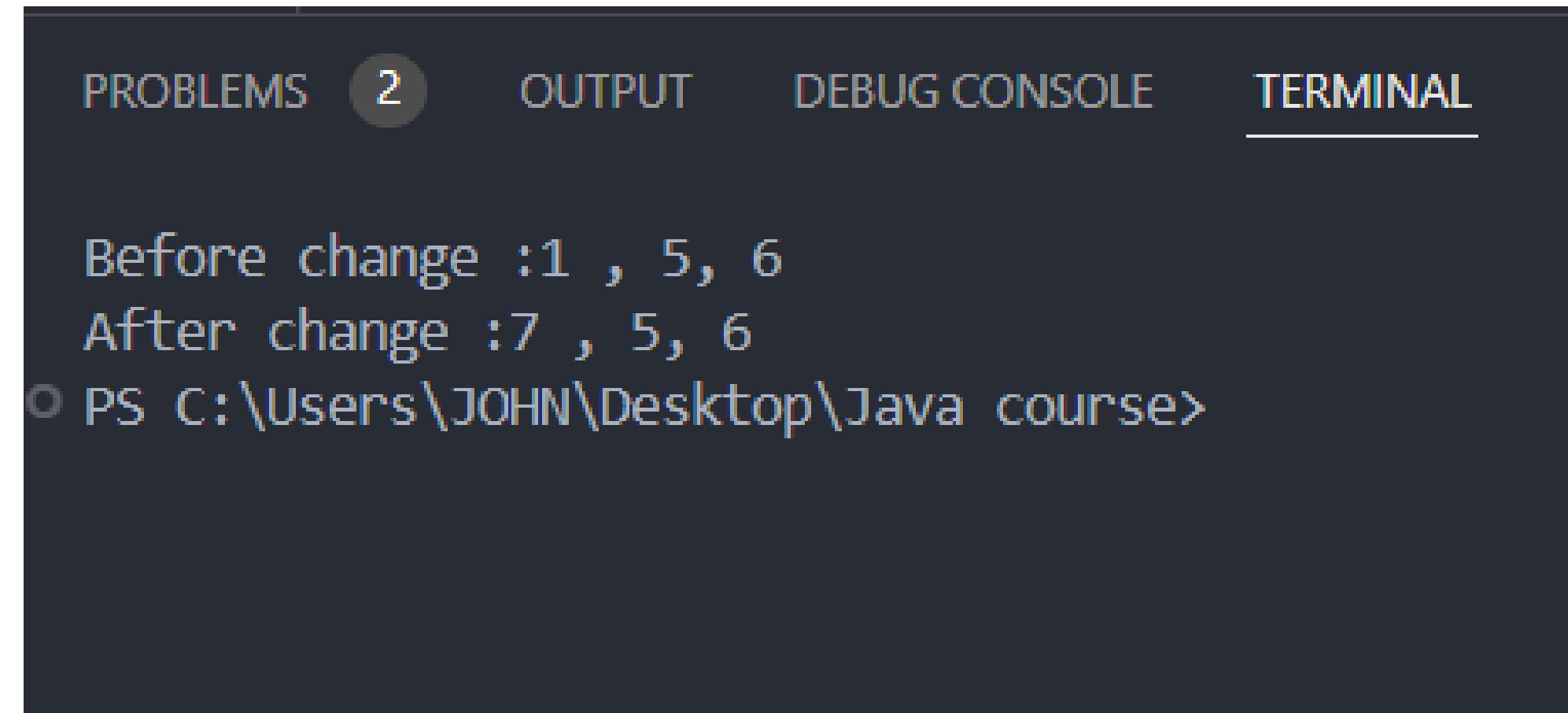
REMARQUE!

```
1 public class Main {  
2     public static void main(String[] args){  
3         int[] T = new int[] {1,2,3,4};  
4         int x = 5;  
5         int y = 6;  
6         Mehdi mehdi = new Mehdi();  
7         System.out.println("Before change :" + T[0] + " , " + x + " , " + y);  
8         mehdi.changeValues(T, x, y);  
9         System.out.println("After change :" + T[0] + " , " + x + " , " + y);  
10    }  
11 }
```

```
1 public class Mehdi{  
2     public void changeValues( int[] T, int x , int y) {  
3         T[0] = 7;  
4         x = 8;  
5         y= 9;  
6     }  
7 }  
8 }
```

BY REFRENCE / BY VALUE

OUTPUT



A screenshot of a terminal window from a code editor. The window has tabs at the top: PROBLEMS (2), OUTPUT, DEBUG CONSOLE, and TERMINAL. The TERMINAL tab is active. The output in the terminal is:

```
Before change :1 , 5, 6
After change :7 , 5, 6
PS C:\Users\JOHN\Desktop\Java course>
```

CLASS INSTANCES

CLASS INSTANCES

عندما نقوم بصناعة الـ **Object** لـ **كلاس** ما من أجل استخدام ميثود او متغير الذي بداخله فنحن في الحقيقة نقوم بصناعة **instance** لهذا الكلاس او بكلمة اقرب نحن نستنسخه و نقوم باستعمال تلك النسخة.

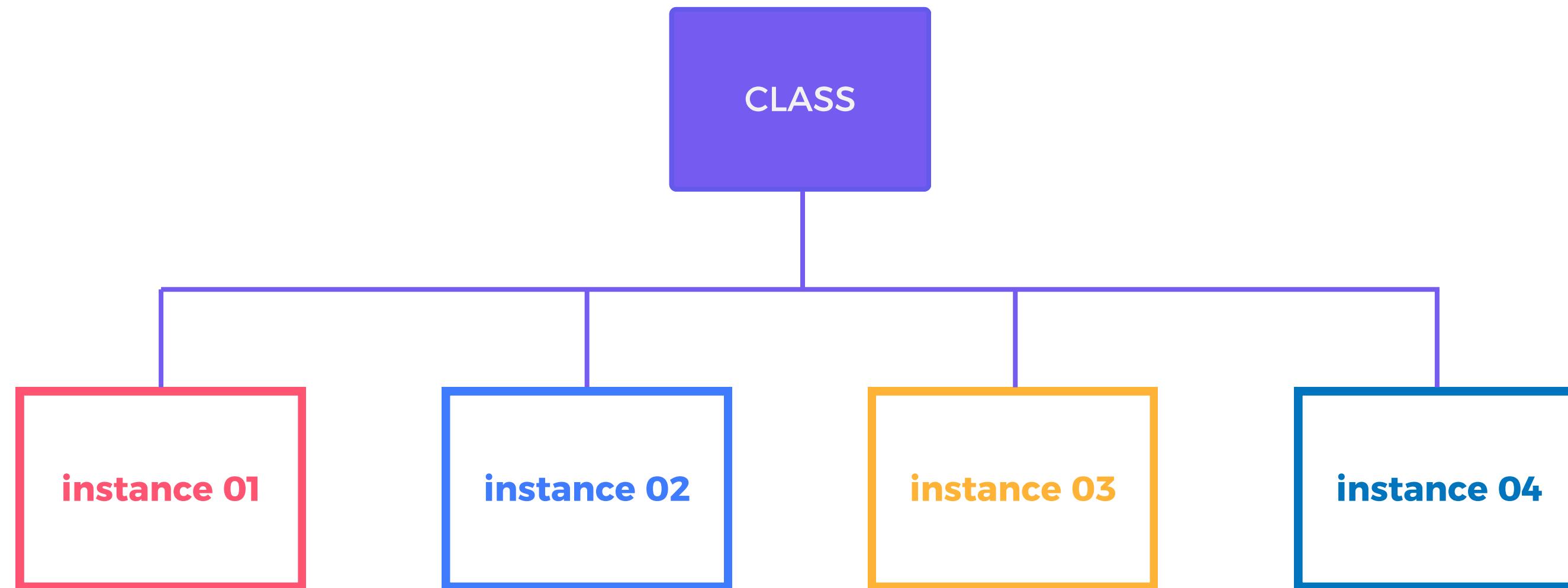
(مثـن نسخـة مـاتـرـوـحـش تـبـهـدـلـنـا مع البرـانـي رـانـي نـمـدـلـك بـرـك فـي الصـورـة بـاـش تـفـهـم وـاـش معـناـه **instance**).
كل **instance** او كل نسخة لديها **id** خاص بها (**behind the scenes**) لذلك عندما نقوم بـ **انـشـاء نـسـخـتـين** من **نفس الكلاس** و نـقـوم بـ **مـقـارـنـتـهـم** دائمـا النـاتـج يـكـون **false** (رانـي نـقـولـك كـي نـشـاء ماـشـي كـي نـحـيل نـسـخـة إـلـى نـسـخـة).

يمـكـن لـ اي كـلاـس ان يـكـون لـ دـيـه عـدـد غـير مـدـدـد مـن الـ **.instances**

CLASS INSTANCES

CLASS INSTANCES

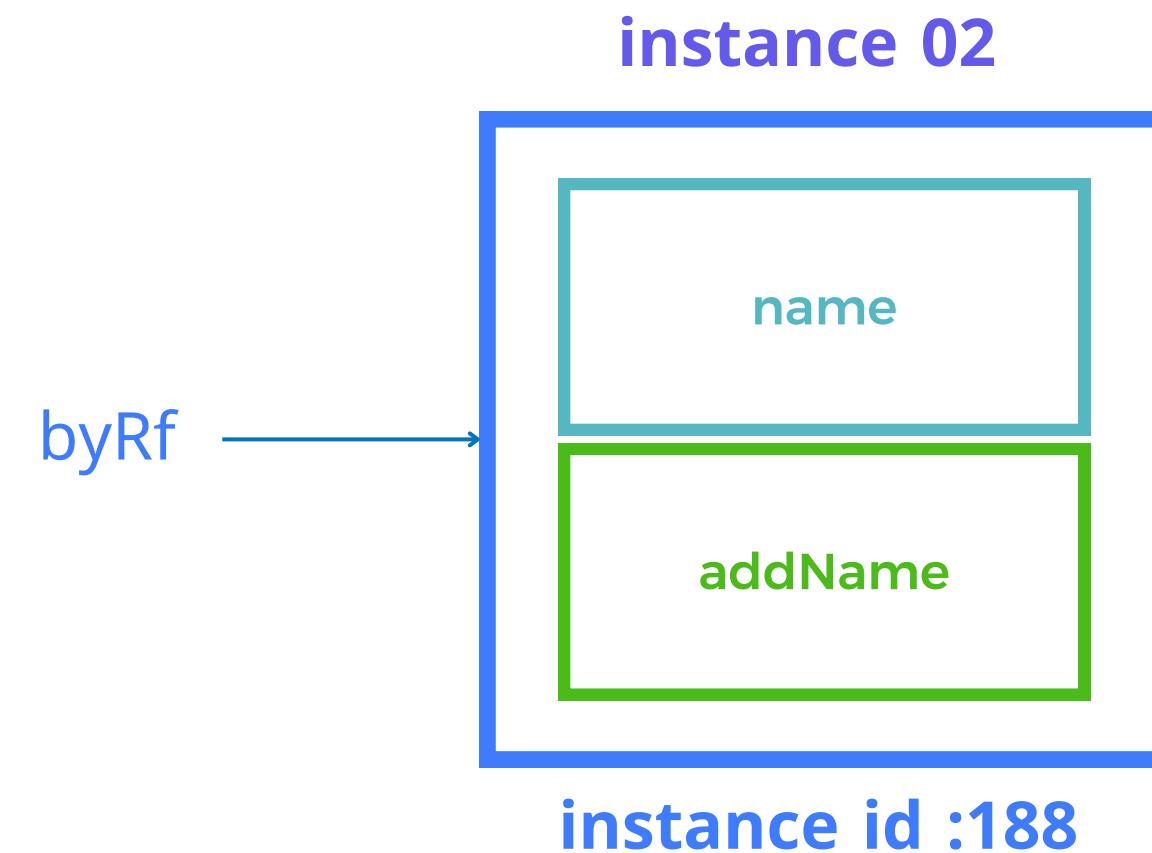
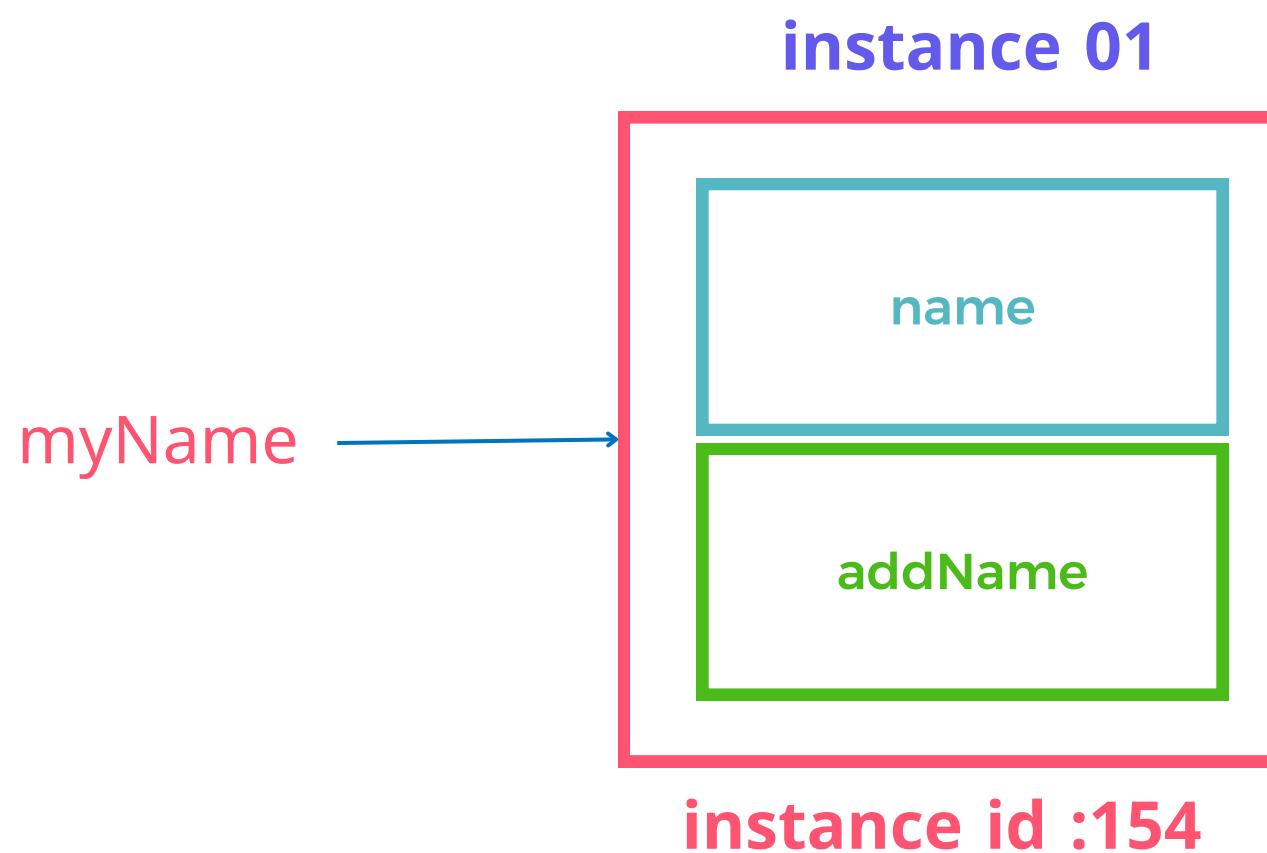
يحتوي الـ **Objects** و **Variables** و **Methods** و **Constructors** على كل الـ **instance** الموجودين
داخل الـ **.Class**.



CLASS INSTANCES

CLASS INSTANCES

```
byReference byRf = new byReference();  
byReference myName = new byReference();
```



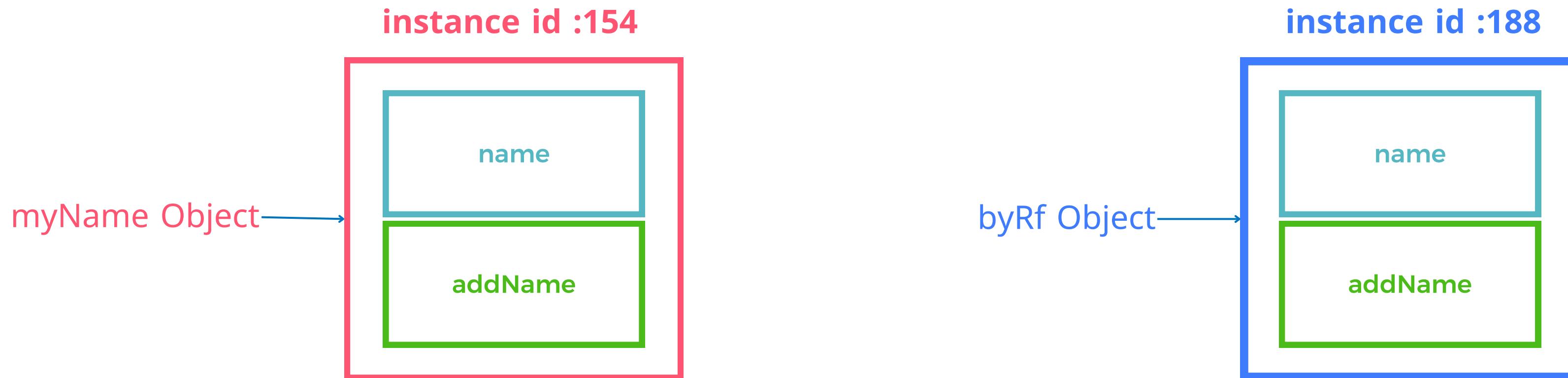
CLASS INSTANCES

CLASS INSTANCES

```
byReference byRf = new byReference();
```

```
byReference myName = new byReference();
```

```
byRf == myName ? => FALSE
```



CLASS INSTANCES

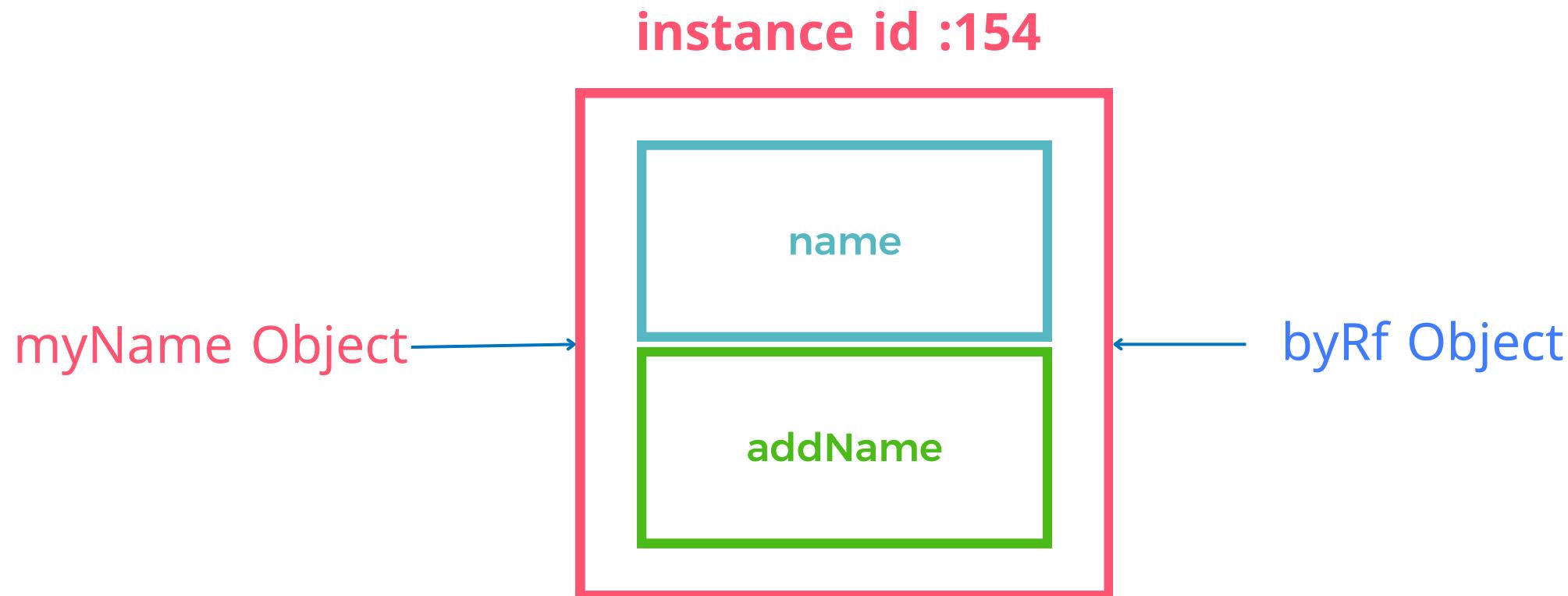
CLASS INSTANCES

```
byReference byRf = new byReference();
```

```
byReference myName = new byReference();
```

راني نحيل/نمد قيمة الى قيمه يااعد ربي متى نقارن//

```
byRf == myName ? => TRUE
```



THE END

راح نزيد نوجد 03 PDF لي راح نحلوا فيه exam exo و منو راح تزيدوا تفهموا كلش كيفاه يخدم مع بعض.

اذا كان كайн اي خطاء اولا نسيت كاش حاجة قولي خطرش عندي ثلاث سنين من يلي قريت جافا.



Mehdi Bouchachi

SEE YOU SOON...