Mehdi Bouchachi

THE COMPLETE JAVA

SECTION

JAVA ADVANCE

LECTURE

JAVA FUNDAMENTALS (IN ARABIC)

@MehdiBouchachi

METHODS AND FUNCTIONS

METHODS

👈 تتكون الMethod من :

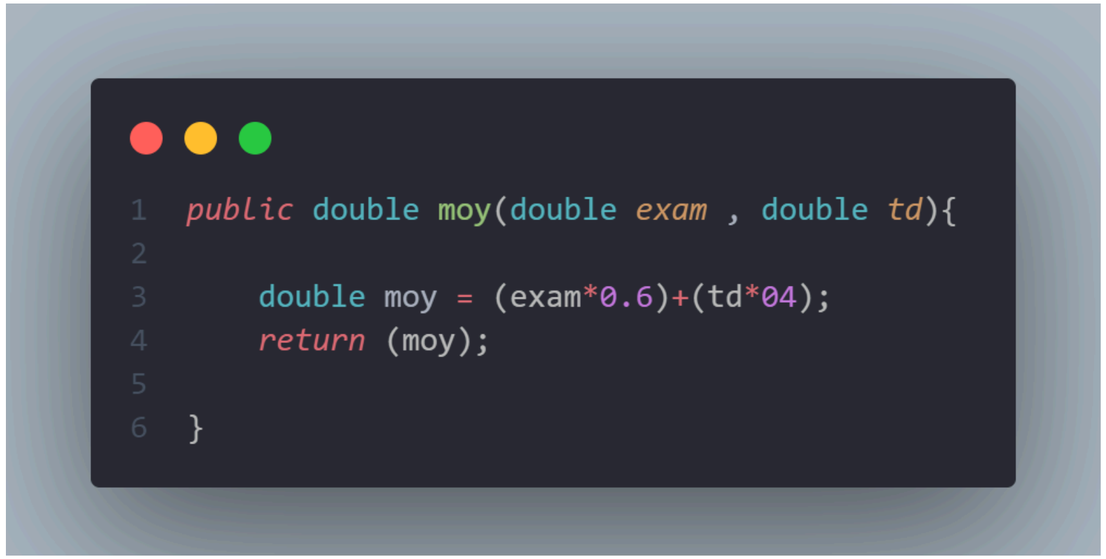
1. نوع ال Modifier (public , private...).
2. void. (اي ليس لديها return)
3. اسم ال Method.
4. مدخلات (Parameters).
5. Logic.

METHODS AND FUNCTIONS

FUNCTIONS

👈 تتكون الMethod من :

1. نوع ال **Modifier (public , private...)**.
2. **DataType**. (اي لديها return)
3. اسم ال **Method**.
4. مدخلات (**Parameters**).
5. **Logic**.



```
public double moy(double exam , double td){

    double moy = (exam*0.6)+(td*04);
    return (moy);

}
```

CREATING OBJECTS

صناعة OBJECT

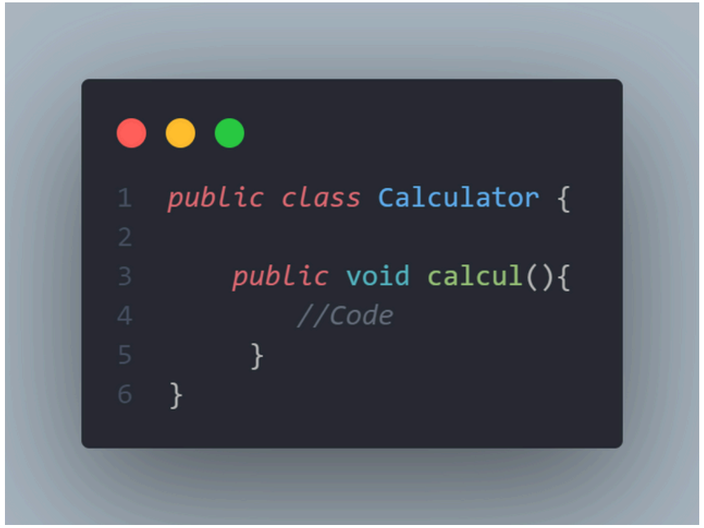
👈 يتم صناعة object عندما نريد استدعاء دالة من كلاس مختلف, من اجل صناعة object نحتاج الى :

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

CREATING OBJECTS

صناعة OBJECT

👈 لدينا **كلاس اسمه** calculator يحتوي على **دالة calcul**, سنقوم **بصناعة object** لهذا **الكلاس** من اجل **استدعاء دالة calcul** و **استخدامها** في **الMain**.



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

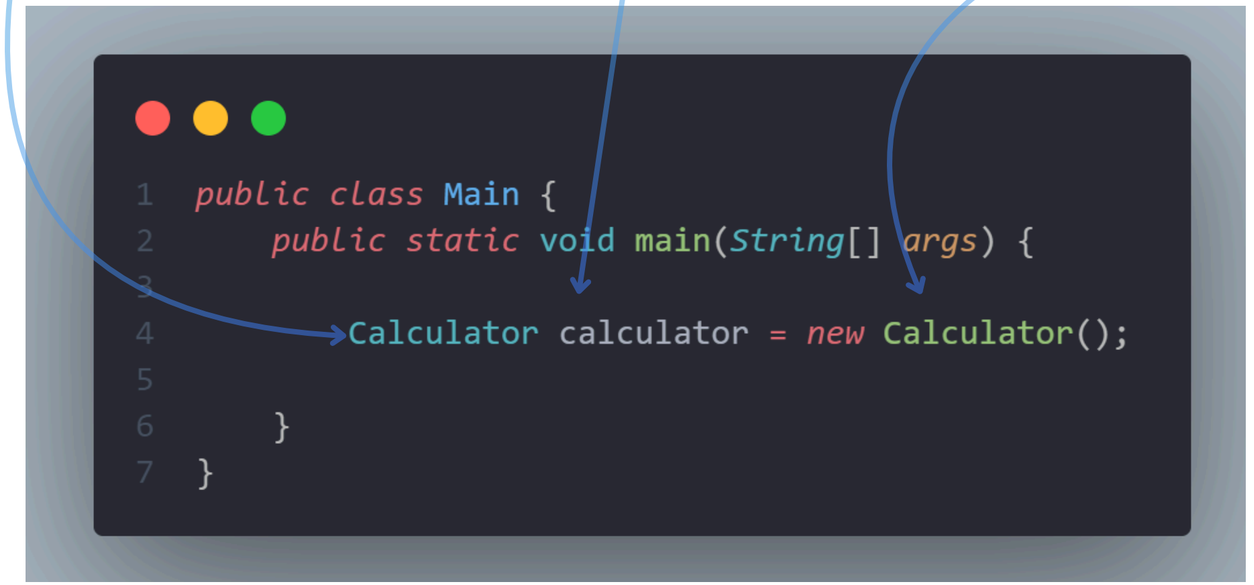
CREATING OBJECTS

صناعة OBJECT

class name

اسم يرمز الى class

new class name

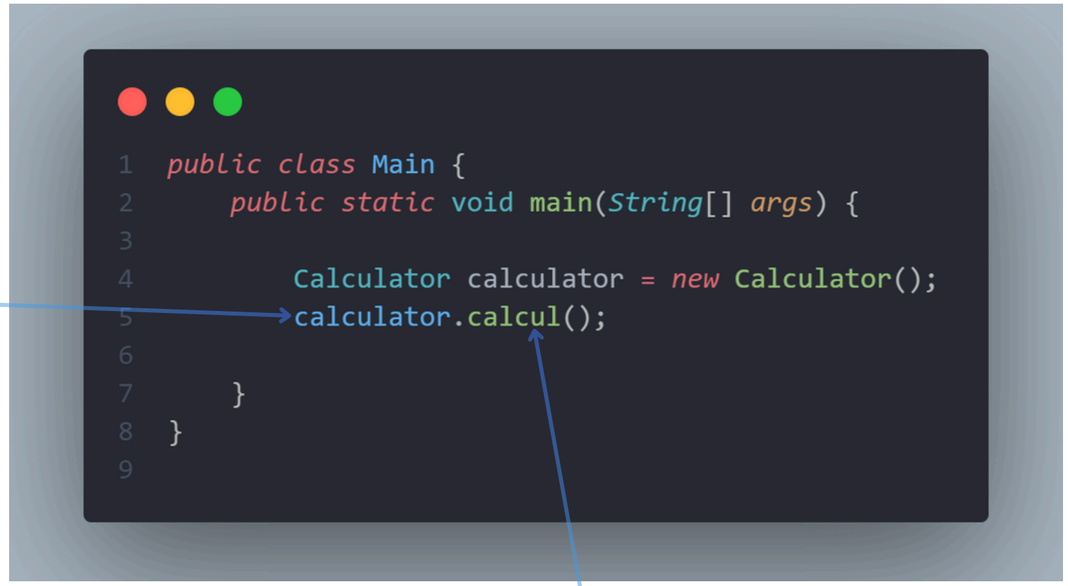


CREATING OBJECTS

CALLING METHOD

👈 لاستدعاء ميثود او متغير ما يجب علينا كتابة : variableName/methodName . اسم يرمز للclass.

اسم يرمز الى class



Method name

PUBLIC/PRIVATE METHODS

PUBLIC METHOD

👈 يمكن **استدعائها و استخدامها** في **جميع الكلاسات.**

PRIVATE METHOD

👈 يمكن **استدعائها و استخدامها فقط** في **الكلاس الموجودة فيه.**

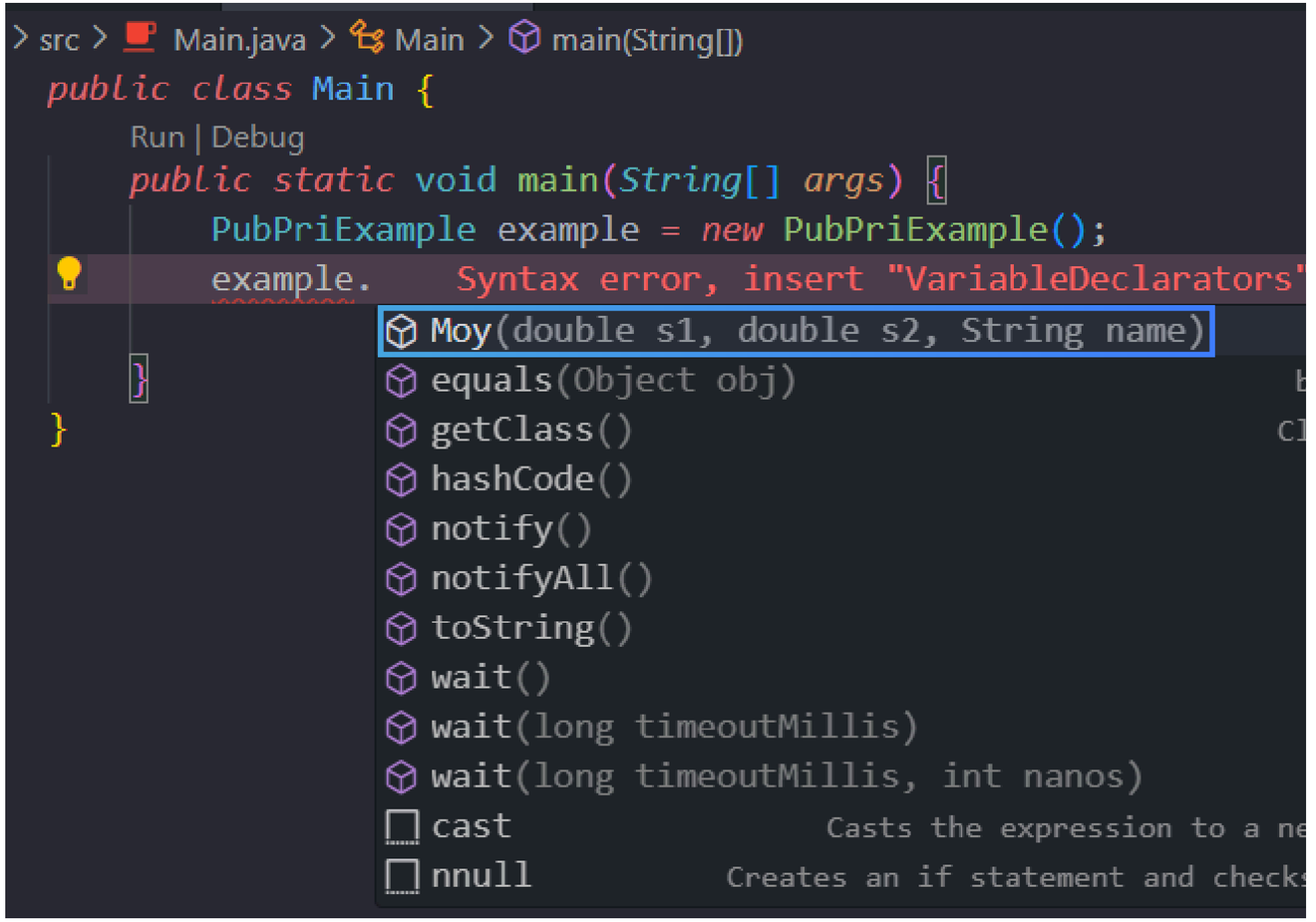
EXAMPLE :

👇

PUBLIC/PRIVATE METHODS

```java
public class PubPriExample {
    public void Moy(double s1, double s2, String name){

        String result = "Grade of "+name+" : "+ (s1+s2)/2;
        System.out.println(result);

    }

    private void Moy1(double s3, double s4 ,String name){

        String result = "Grade of "+name+" : "+ (s3+s4)/2;
        System.out.println(result);

    }
}
```

PUBLIC/PRIVATE METHODS



تظهر لنا دالة

واحدة متوفرة فقط

public void Moy

STATIC / NOSTATIC IN METHOD AND VARIABLES

STATIC METHOD

👈 **static method** هي انها **ضمن الكلاس و متوفرة لجميع الكلاسات** و يمكن **استخدامها** **بدون صناعة object.**

```java
public class Calculator {

    public static void calcul(){
        //Code
    }
}
```

```java
public class Main {
    public static void main(String[] args) {

        Calculator.calcul();

    }
}
```

STATIC / NOSTATIC IN METHOD AND VARIABLES

NOSTATIC METHOD

👈 هذا النوع من الMethods متوفرة **لكل الكلاسات لكن تحتاج الى object** حتى **يمكن استخدامها.**

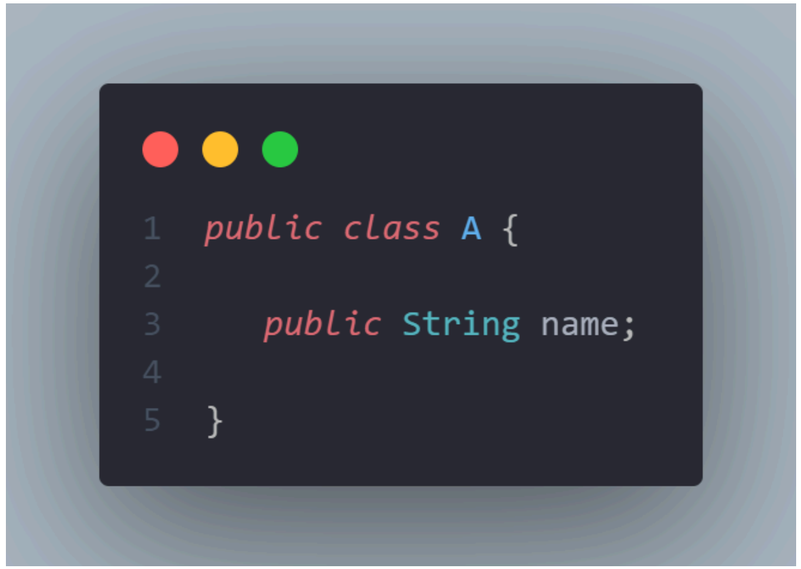
```java
public class Calculator {

    public void calcul(){
        //Code
    }
}
```

```java
public class Main {
    public static void main(String[] args) {

        Calculator calculator = new Calculator();
        calculator.calcul();

    }
}
```

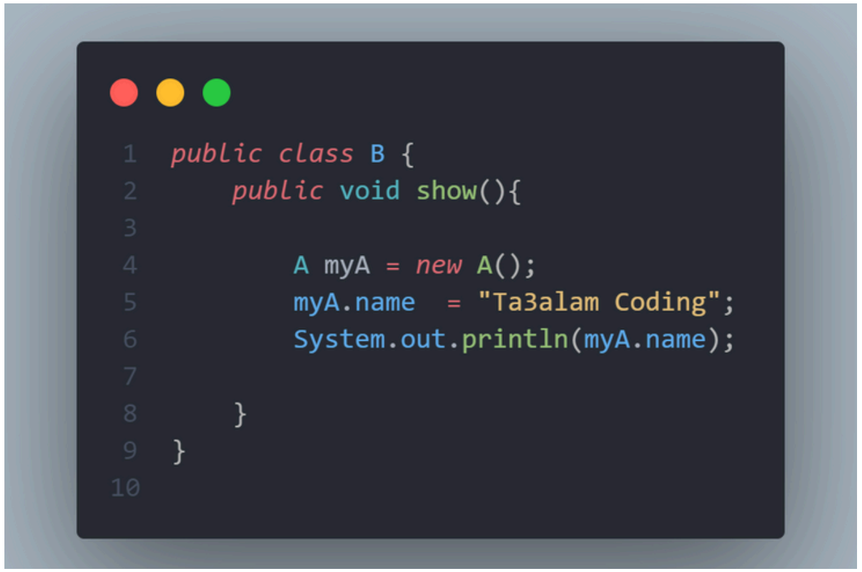
STATIC / NOSTATIC IN METHOD AND VARIABLES

NOSTATIC VARIABLES

👈 يمكن استخدام الVariables في method و class مختلف و يمكن ان تحدد متغير في كلاس A و تقوم بوضع قيمة بداخله في كلاس B.



```java
public class A {

    public String name;

}
```



```java
public class B {
    public void show(){

        A myA = new A();
        myA.name  = "Ta3alam Coding";
        System.out.println(myA.name);

    }
}
```

STATIC / NOSTATIC IN METHOD AND VARIABLES

NOSTATIC VARIABLES

```java
public class Main {
    public static void main(String[] args) {

        B myB = new B();
        myB.show();

    }
}
```

OUTPUT

TA3ALAM CODING

STATIC / NOSTATIC IN METHOD AND VARIABLES

STATIC VARIABLES

```java
public class A {

    public static String name;

}
```

```java
public class B {
    public void show(){

        A.name  = "Ta3alam Coding";
        System.out.println(A.name);

    }
}
```

STATIC / NOSTATIC IN METHOD AND VARIABLES

STATIC VARIABLES

```java
public class Main {
    public static void main(String[] args) {

        B myB = new B();
        myB.show();

    }
}
```

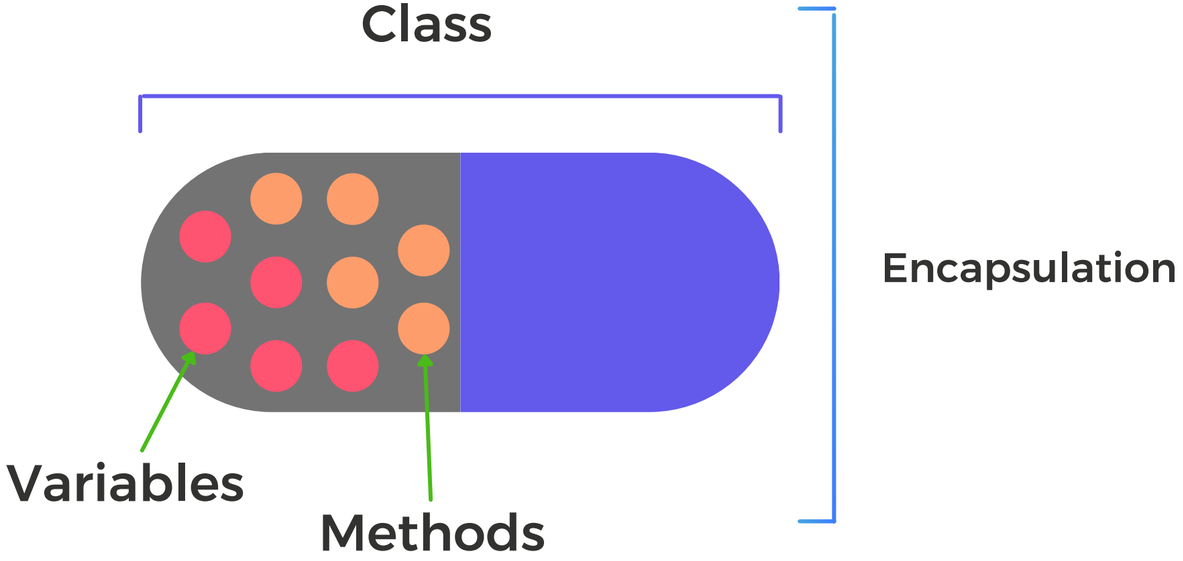
OUTPUT

TA3ALAM CODING

ENCAPSULATION

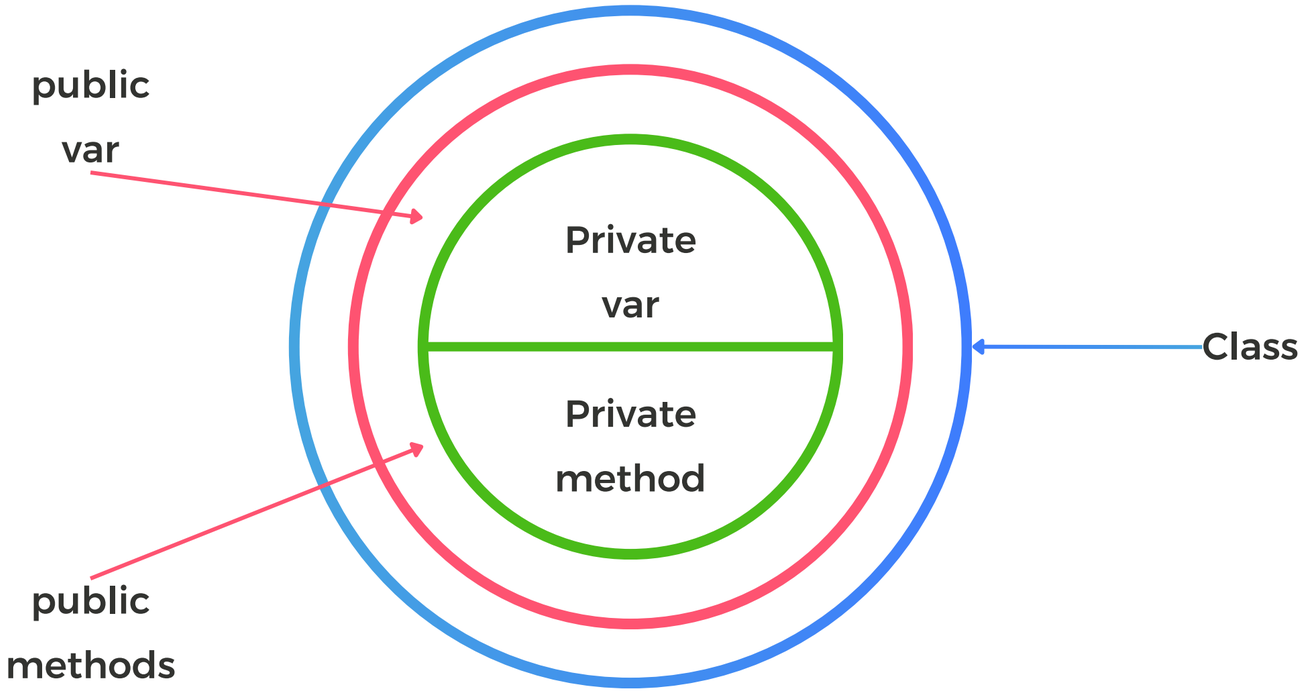
ENCAPSULATION

👈 عبارة عن **أسلوب** يمكن اتباعه **لإخفاء 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 الاساسي.



ENCAPSULATION

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

👈لايحتوي على Parameters **او** args.

CONSTRUCTOR

DEFAULT CONSTRUCTOR

```java
public class Mehdi {

    public Mehdi(){

        System.out.println("Hello :)");

    }
}
```

```java
public class Main {
    public static void main(String[] args) {

            Mehdi mehdi = new Mehdi();

    }
}
```

👈يتم طباعة **مباشرة** كلمة Hello :) . (**يتم دائما تنفيذ الConstructor عند صناعة Object اتفكرها!**)

CONSTRUCTOR

CONSTRUCTOR

```java
public class Mehdi {

    int Id, age;
    String name;

    public Mehdi(String n ,int i ){
        name = n;
        Id = i;
    }

    public void  show(){
        System.out.println("ID : "+Id+"\nName : "+name);
    }

}
```

```java
public class Main {
    public static void main(String[] args) {

            Mehdi mehdi = new Mehdi("Mehdi", 14);
            mehdi.show();

    }
}
```

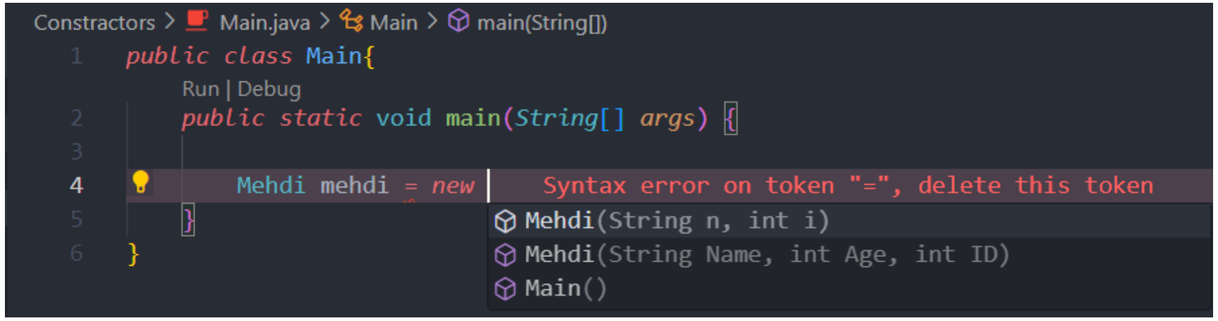
يتم طباعة 👈 ID : 14 , Name : Mehdi

CONSTRUCTOR

CONSTRUCTOR OVERLOADING

👈 هي صناعة 2 Constructors فما فوق مع تغيير نوع الParameters او عددهم او ترتيبهم.

```java
public class Mehdi {

    int Id, age;
    String name;
    public Mehdi(String n ,int i ){
        name = n;
        Id = i;
    }
    public Mehdi(String Name , int Age , int ID){
        name = Name;
        age = Age;
        Id = ID;
    }
    public void  show(){
        System.out.println("ID : "+Id+"\nName : "+name);
    }
    public void show1(){
        System.out.println("Name : "+name+"\nAge : "+age+"\nID : "+Id);
    }

}
```



```
Constractors > Main.java > Main > main(String[])
1  public class Main{
       Run | Debug
2      public static void main(String[] args) {
3
4          Mehdi mehdi = new     Syntax error on token "=", delete this token
5      }                         Mehdi(String n, int i)
6  }                             Mehdi(String Name, int Age, int ID)
                                 Main()
```

👈 في مرحلة صناعة الObject يظهر لنا الConstructors

الذين تم صناعتهم.

CONSTRUCTOR

Main

```java
public class Main{
    public static void main(String[] args) {

        Mehdi mehdi = new Mehdi("Aissa", 20);
        Mehdi mehdi2 = new Mehdi("Mehdi", 20, 2457);

        mehdi.show();
        System.out.println("------------------------------");
        mehdi2.show1();

    }
}
```

OUTPUT

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

in'

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

قمنا بصناعة **Objects 2** و استخدام **Constructors 2** في نفس الوقت.👈

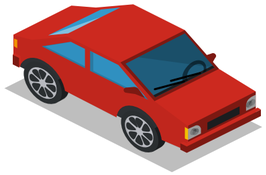
CONSTRUCTOR VS METHODS

CONSTRUCTOR

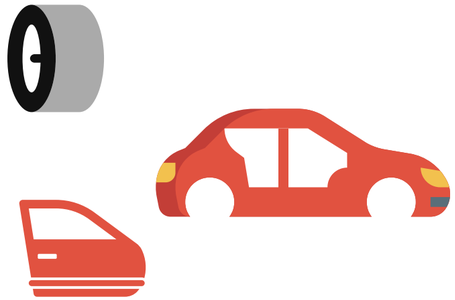
- ال**Method** يريك عمل ال**Object**.
- ال**Constructor** يوفر لنا القيم التي نحتاجها لبناء **Object**.
- ال**Constructor** **دائما** **يتم تنفيذه عند** **صناعة Object**.

Mehtod

Constructor



عند صناعة ال Object



THIS / SETTER / GETTER

THIS KEYWORD

👈 هي متغير مرجعي يشير القيمة الحالية للObject (اتفكر هذي راح نطبقوا عليها فل PDF 03).

👈 **يمكن من خلاله للميثود او الConstructor** ان يستخدم **نفس اسماء المتغيرات في الكلاس**.

```
public class User {
    int ID ;
    String username;
    String password;
    public User(int ID, String username, String password){

        ID = ID;    The assignment to variable ID has no effect
        username = username;    The assignment to variable username has no effect
        password = password;    The assignment to variable password has no effect

    }

    public void show(){
        System.out.println("User ID : "+ID+
                "\nUser username : "+username+
                "\nUser password : "+password);
    }
}
```

THIS / SETTER / GETTER

THIS KEYWORD

```java
public class User {
    int ID ;
    String username;
    String password;
    public User(int ID, String username, String password){

        this.ID = ID;
        this.username = username;
        this.password = password;

    }

    public void show(){
        System.out.println("User ID : "+ID+
                "\nUser username : "+username+
                "\nUser password : "+password);
    }
}
```

THIS / SETTER / GETTER

get( );

👈 بما انه فيها كلمة **get** اي انك تطلب من الدالة **ترجعلك قيمة** مهما كانت , و بما انه **يرجعلك قيمة** اي فيها **return** , وعليه يجب تحديد **Data type** في مكان ال **Void** .

Syntax :

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

THIS / SETTER / GETTER

get( );

```java
public class Getter {
    String name;
    int ID;

    public int getID() {
        return ID;
    }

    public String getName() {
        return name;
    }

}
```

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( );

```java
public class Setter {
    int ID;
    String name;

    public void setID(int ID) {
        this.ID = ID;
    }

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

THIS / SETTER / GETTER

REAL EXAMPLE

```java
public class Employee {

    private String name ;
    private int ID;
    private String address;
    private String department;

    public Employee(int ID,String name, String department, String address){

        this.ID = ID;
        this.name = name;
        this.department = department;
        this.address = address;

    }
```

```java
public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getID() {
        return ID;
    }
    public void setID(int ID) {
        this.ID = ID;
    }
    public String getAddress() {
        return address;
    }
```

```java
    public String getAddress() {
        return address;
    }
    public void setAddress(String address) {
        this.address = address;
    }
    public String getDepartment() {
        return department;
    }
    public void setDepartment(String department) {
        this.department = department;
    }
    public void show(){
        System.out.println("Name : "+name+"\nID : "+ID
                           +"\nDepartment : "+department+"\nAddress : "+address);
    }
}
```

THIS / SETTER / GETTER

MAIN

```java
public class Main {
    public static void main(String[] args) {

        Employee employee = new Employee(321086643,"Mehdi","MI","Cite N05");
        employee.show();

        String nameBefore = employee.getName();//getting the name from constructor and adding it to the var
        employee.setName("Mohamed");//changing thr name value in constructor from mehdi to mohamed
        String nameAfter = employee.getName();//getting the name from constructor after setName

        System.out.println("\n-------------------------\n");
        System.out.println("Name before : "+nameBefore);
        System.out.println("Name after : "+nameAfter);
        System.out.println("\n-------------------------");

        employee.show();

    }
}
```

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 **علاقة بين مجموعة كلاسات** , كمثال عندك **كلاس** B, A **و** C يمكن **للكلاس** A ان يعطي **العناصر الموجودة** داخله (ميثودس و متغيرات) **للكلاس** B و C.

👈 في جافا لدينا SuperClass و SubClass في معظم لغات البرمجة اخرى يطلق عليهم ParentClass و ChildClass.

Syntax :

```
class SubClass extends SuperClass {
 Methods / Vars
}
```

INHERITANCE TYPES

SIMPLE INHERITANCE

SuperClass → Class A

SubClass → Class B

```java
public class ClassA {

    public void show(){

        System.out.println("Ta3alam Coding From Class A");

    }
}
```

```java
public class ClassB extends ClassA{

}
```

INHERITANCE TYPES

SIMPLE INHERITANCE

MAIN

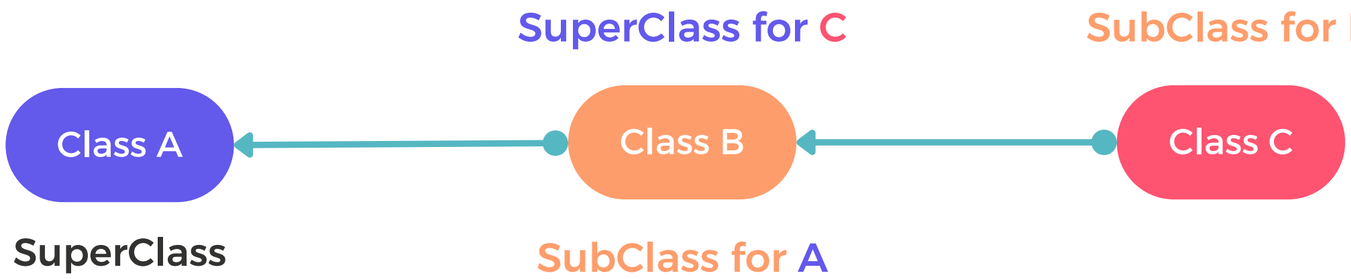
```java
public class Main {
    public static void main(String[] args) {

        ClassB classB = new ClassB();
        classB.show();

    }
}
```

OUTPUT

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

Process finished with exit code 0
```

INHERITANCE TYPES

MULTIPLE INHERITANCE




```java
public class ClassA {

    public void show(){

        System.out.println("Ta3alam Coding From Class A");

    }
}
```

```java
public class ClassB extends ClassA{

}
```

```java
public class ClassC extends ClassB{

}
```

INHERITANCE TYPES

MULTIPLE INHERITANCE

MAIN

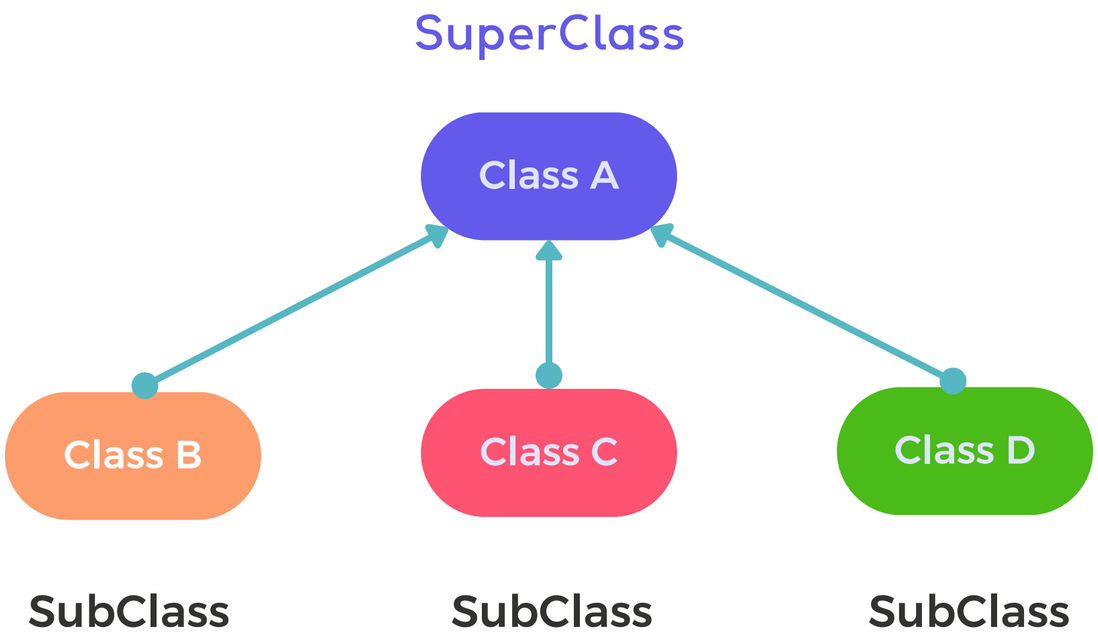
```java
public class Main {
    public static void main(String[] args) {

        ClassC classC = new ClassC();
        classC.show();

    }
}
```

OUTPUT

```
"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

```java
public class ClassA {

    public void show(){

        System.out.println("Ta3alam Coding From Class A");

    }
}
```

```java
public class ClassB extends ClassA{
  public ClassB(){

      System.out.println("This msg from class B");

  }
}
```

```java
public class ClassC extends ClassA{
    public ClassC(){

        System.out.println("This msg from class C");

    }
}
```

INHERITANCE TYPES

INHERITANCE WITH CONSTRUCTOR

MAIN

```java
public class Main {
    public static void main(String[] args) {

        ClassC classC = new ClassC();
        classC.show();

    }
}
```

OUTPUT

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

This msg from class C
Ta3alam Coding From Class A

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

METHOD OVERRIDING

METHOD OVERRIDING

👈 اذا كان اسم الMethod الموجود في SubClass هو نفسه الموجود في SuperClass, ولكن **الكود داخلها مختلف** في هذه الحالة تسمى ب Method overriding.

EXAMPLE :

👇

METHOD OVERRIDING

METHOD OVERRIDING

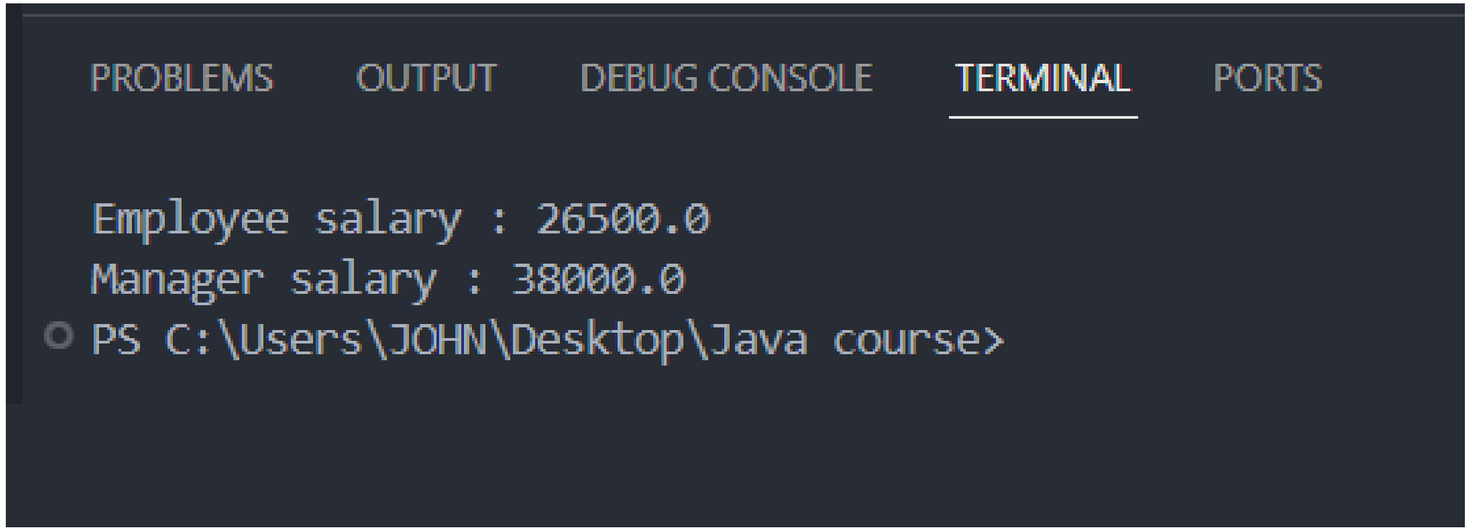
```java

public class Main {
    public static void main(String[] args) {

        Employee employee = new Employee();
        employee.Salary(1);
        Manager boss = new Manager();
        boss.Salary(1);

    }
}

```

```java
public class Manager {
    double prime;
    double Salary;
    double fullSalary;
    public void Salary(int exp){

        if (exp <= 2){
            Salary = 35000;
            prime = 3000;
        }else if( exp <5){
            Salary = 40000;
            prime = 5000;
        }else {
            Salary = 45000;
            prime = 10000;
        }
        fullSalary = Salary + prime;
        System.out.println("Salary boss : "+fullSalary);

    }
}
```

```java
public class Employee extends Manager{
    public void Salary(int exp){

        if (exp <= 2){
            Salary = 25000;
            prime = 1500;
        }else if( exp <5){
            Salary = 35000;
            prime = 3000;
        }else {
            Salary = 40000;
            prime = 5000;
        }
        fullSalary = Salary + prime;
        System.out.println("Salary employee : "+fullSalary);

    }
}

```

METHOD OVERRIDING

OUTPUT



SUPER KEYWORD

SUPER KEYWORD

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

👈 يمكننا استخدام **Super keyword** للوصول إلى **متغير او الميثود** الموجود في **SuperClass.**

👈 يتم استخدامه إذا كان **SuperClass** و **SubClass** **يحتويان على نفس اسم المتغيرات و الميثودس و مختلفين في القيم.**

SUPER KEYWORD

WITH VARIABLES

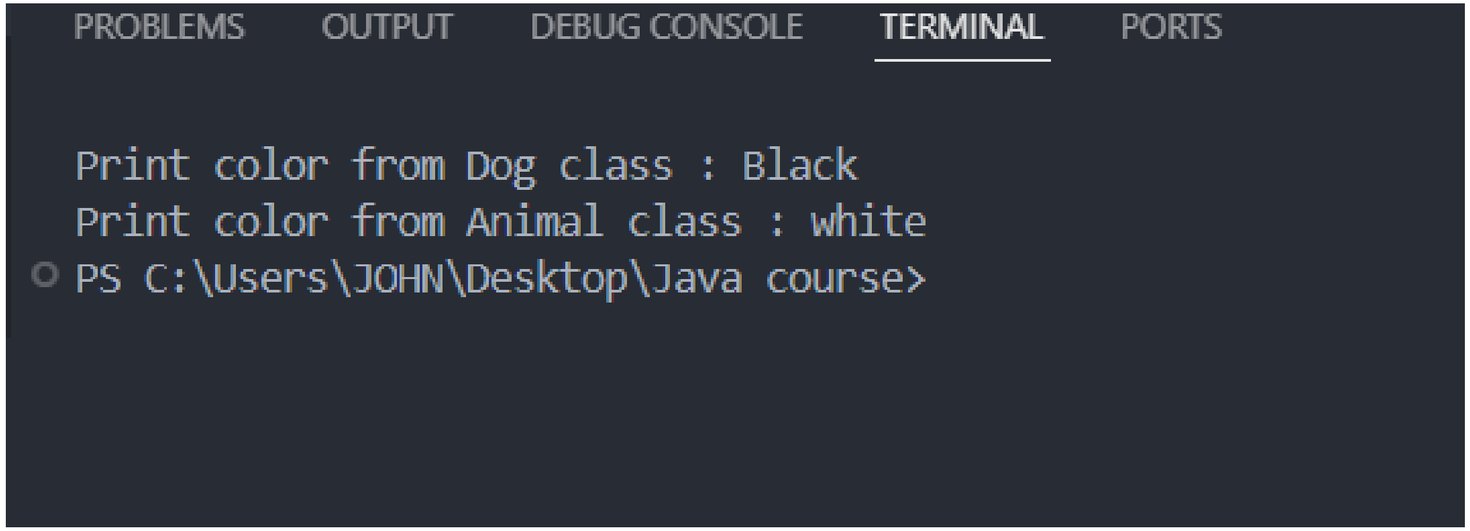
```java
public class Animal {

    String color= "white";

}
```

```java
public class Dog extends Animal{

    String color = "Black";

    public void showColor(){

        System.out.println("Print color from Dog class : "+color);

        System.out.println("Print color from Animal class : "+super.color);
    }
}
```

```java
public class Main {
    public static void main(String[] args) {

        Dog dog = new Dog();
        dog.showColor();

    }
}

```

METHOD OVERRIDING

OUTPUT



SUPER KEYWORD

SUPER KEYWORD

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

عندما نقوم **بطبع** color في **الكلاس الحالي** يقوم بطبع Dog color , وحتى نقوم بطبع Animal color نقوم باستخدام Super **اشارة منا باننا نريد المتغير** color **الموجود في** SuperClass.

👈 في **بعض المرات** نقوم بعمل override **للميثود** الموجودة في superClass داخل الsubClass و نريد ان نستخدم **الميثود** الموجود في superClass داخل الsubClass عندها نقوم **باستدعائها بواسطة الكلمة** super.

SUPER KEYWORD

WITH METHODS

```java
public class Cat {

    public void eat(){

        System.out.println("The cat is eating...");
    }
    public void sleep(){

        System.out.println("Zzz Zzz");

    }

}
```

```java
public class Dog extends Cat{

   public void eat(){

        System.out.println("The dog is eating");

    }
    public void sleep(){

        System.out.println("The dog did not sleep");

    }
    public void show(){

        System.out.println("This msg from Dog class\n");
        eat();
        sleep();
        System.out.println("\nThis msg from cat class\n");
        super.eat();
        super.sleep();

    }
}
```

SUPER KEYWORD

WITH METHODS

MAIN

```java
public class Main {
    public static void main(String[] args) {

        Dog dog = new Dog();
        dog.show();

    }
}

```

OUTPUT

```
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 ايضا استخدام Super مع الـ SuperClass Constructors.

```java
public class Cat {

    public Cat(){
           System.out.println("Meow Meow");
    }

    public Cat(String name){
        System.out.println("if u pass an name in super keyword i'll execute");
    }

}
```

```java
public class Dog extends Cat{

  public Dog(){

    super();
    System.out.println("Wof Wof");

  }
}
```

SUPER KEYWORD

WITH CONSTRUCTORS

MAIN

```java
public class Main {
    public static void main(String[] args) {

        Dog dog = new Dog();


    }
}
```

OUTPUT

```
PROBLEMS 1    OUTPUT    DEBUG CONSOLE

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

SUPER KEYWORD

WITH CONSTRUCTORS

```java
public class Cat {

    public Cat(){
           System.out.println("Meow Meow");
    }

    public Cat(String name){
        System.out.println("if u pass an name in super keyword i'll execute");
    }

}
```

```java
public class Dog extends Cat{

  public Dog(){

    super("Mehdi");
    System.out.println("Wof Wof");

  }
}
```

SUPER KEYWORD

WITH CONSTRUCTORS

MAIN

```java
public class Main {
    public static void main(String[] args) {

        Dog dog = new Dog();


    }
}
```

OUTPUT

```
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 الMethod **عدة مرات** مع **تغير عدد** الparameters **او** **انواعهم** **او** **ترتيبهم**.

```java
public  void show(int ID ,String name ){
        System.out.println("Your ID : "+ID+"\nYour name : "+name);
    }
    public void  show(String address,int Salary){
        System.out.println("Your address : "+address+"\nYour Salary : "+Salary);
    }

```

METHOD OVERLOADING

REAL EXAMPLE

```java
public class User {
    int ID;
    int Salary;
    String name;
    String address;
    public User(int ID , String name ,String address,int Salary ){
        this.ID = ID;
        this.name = name;
        this.address = address;
        this.Salary = Salary;
        show(ID,name);
        show(address,Salary);
    }

    public  void show(int ID ,String name ){
        System.out.println("Your ID : "+ID+"\nYour name : "+name);
    }
    public void  show(String address,int Salary){
        System.out.println("Your address : "+address+"\nYour Salary : "+Salary);
    }


```

```java

public int getID() {
    return ID;
}

public void setID(int ID) {
    this.ID = ID;
}

public int getSalary() {
    return Salary;
}

public void setSalary(int salary) {
    Salary = salary;
}

```

```java
    public void setSalary(int salary) {
        Salary = salary;
    }

    public String getName() {
        return name;
    }

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

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }
}
```

METHOD OVERLOADING

MAIN

```java
public class Main {
    public static void main(String[] args) {

      User user = new User(32018664,"Mehdi","Some Where",250000);

        System.out.println("------------------------------------------------------");
        user.show(user.getID(), user.getName());
        user.show(user.getAddress(), user.getSalary());
        System.out.println("------------------------------------------------------");

        user.show(15054661,"Mohamed");
        user.show("some where",35000);
    }
}
```

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

👈 **يسلط الضوء** على ال**Data** و ال**Methods** و ال**Constractor** و ال**Class**.

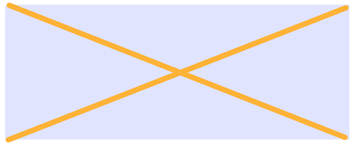
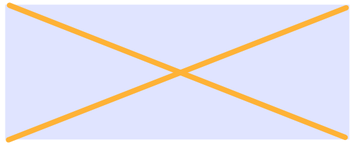
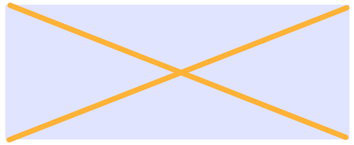
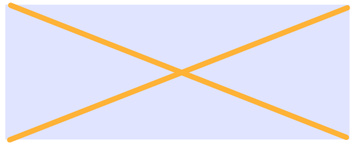
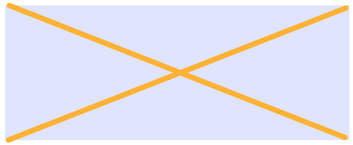
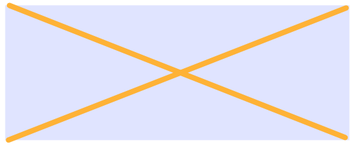
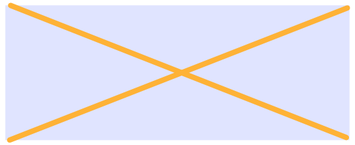
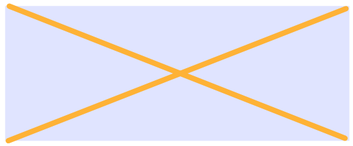
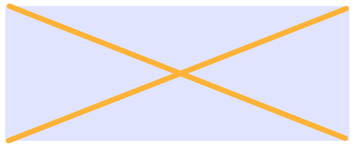
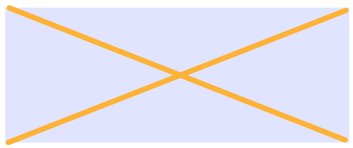
👈 انواعه :

- **private.**
- **public.**
- **protected.**
- **Default.**

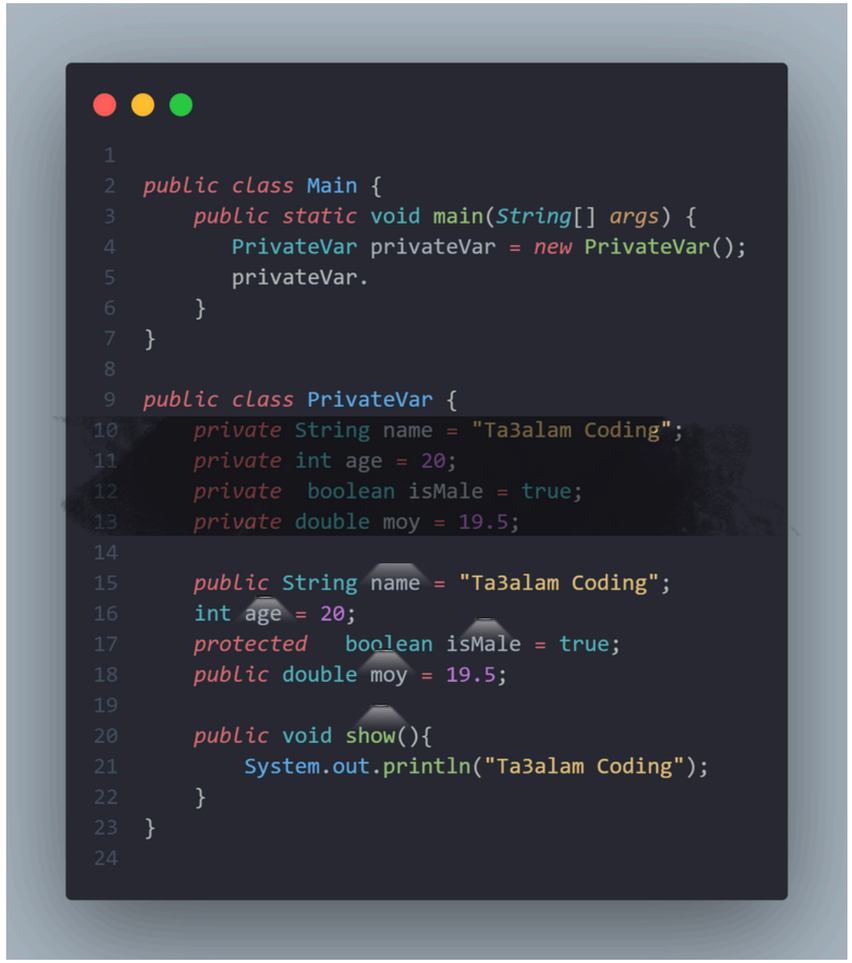
👈 **يلخص الجدول امكانية منادة ال**Methods** او ال**Variables** بالنسبة لل**Encpsol**.**

ACCESS MODIFIERS

ACCESS MODIFIERS

	Class	Package	Outside	SubClass
Public				
Protected				
Default				
Private				

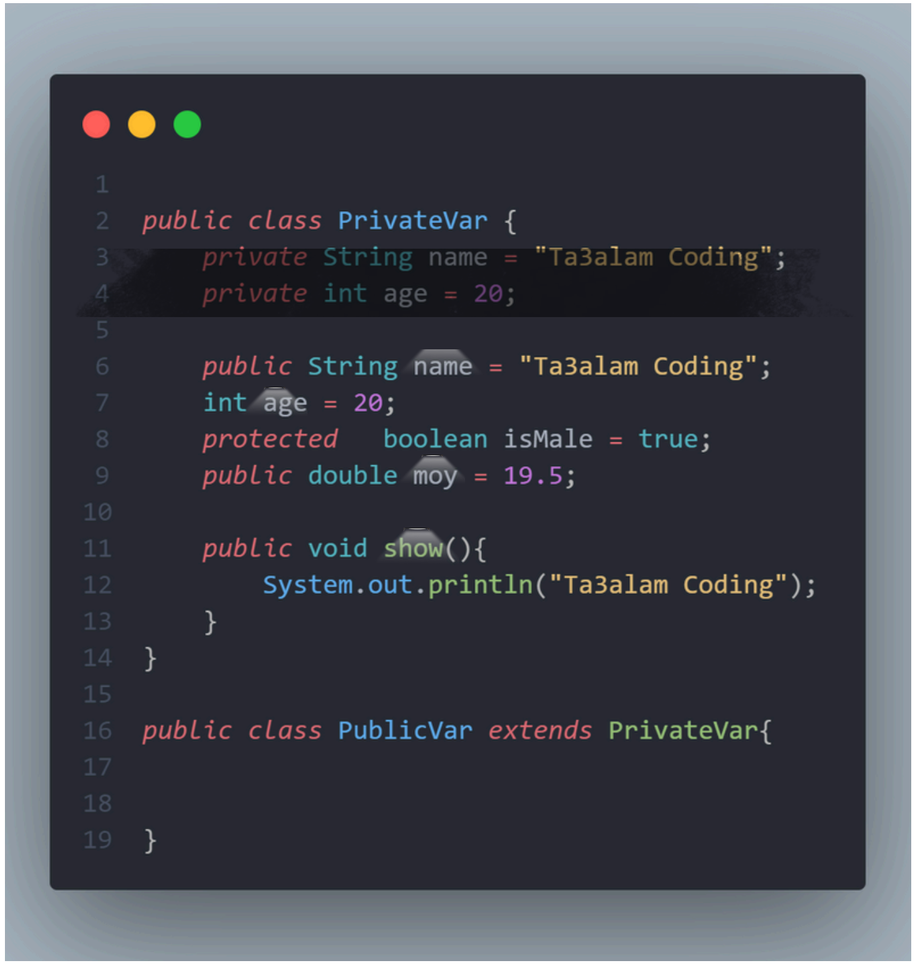
ACCESS MODIFIERS



```java

public class Main {
    public static void main(String[] args) {
       PrivateVar privateVar = new PrivateVar();
       privateVar.
    }
}

public class PrivateVar {
    private String name = "Ta3alam Coding";
    private int age = 20;
    private  boolean isMale = true;
    private double moy = 19.5;

    public String name = "Ta3alam Coding";
    int age = 20;
    protected   boolean isMale = true;
    public double moy = 19.5;

    public void show(){
        System.out.println("Ta3alam Coding");
    }
}

```

👈 المنظور بالنسبة لل **Main**.

ACCESS MODIFIERS

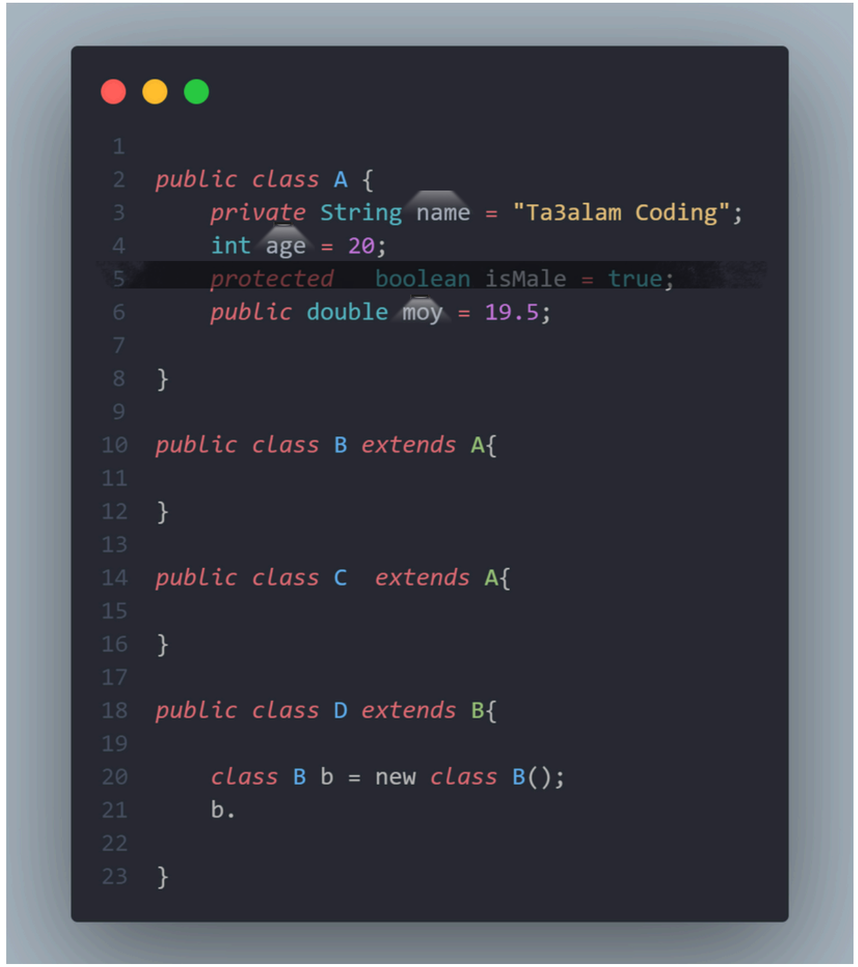


```java

public class PrivateVar {
    private String name = "Ta3alam Coding";
    private int age = 20;

    public String name = "Ta3alam Coding";
    int age = 20;
    protected   boolean isMale = true;
    public double moy = 19.5;

    public void show(){
        System.out.println("Ta3alam Coding");
    }
}

public class PublicVar extends PrivateVar{


}
```

الكلاس PublicVar راح يعمل extend فقط

للمتغيرات و الميثودس 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
20     class B b = new class B();
21     b.
22
23 }
```

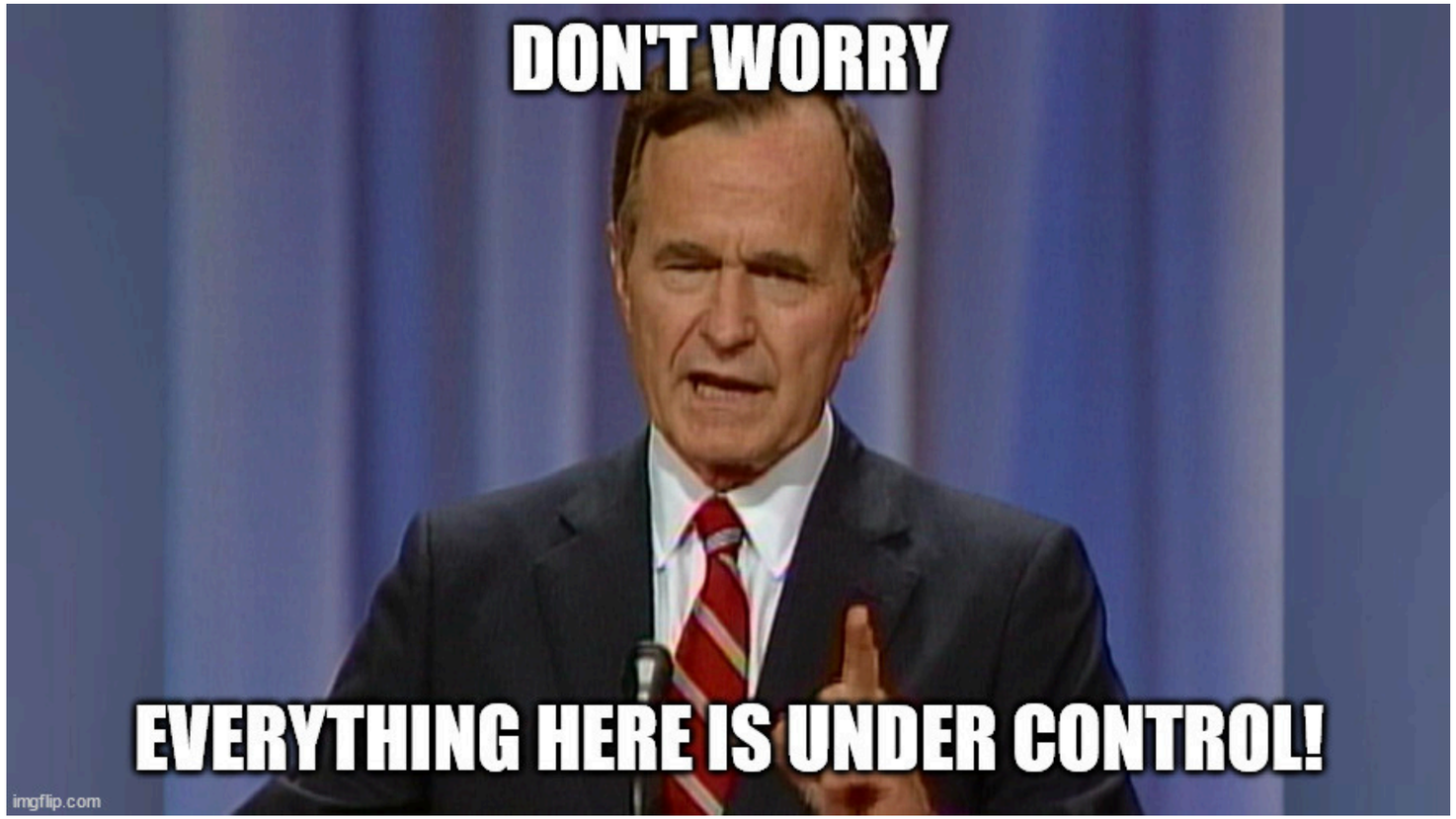
👈المنظور بالنسبة **لل** D لان D ليس Subclass **ل** A

ولكن **ل** B

POLYMORPHISM

POLYMORPHISM😉

POLYMORPHISM



POLYMORPHISM😉

POLYMORPHISM

👈 هو **مجرد أسلوب في كتابة الكود** و يمكنك تحقيقه بعدة أشكال مختلفة.

👈 المقصود من **البوليمورفيزم** هو **بناء دالة تنفذ أوامر مختلفة** على حسب **الObject** الذي يمرر لها عند استدعائها.

👈 تبنى في **العادة بداخل الدوال,** حيث أننا نقوم **ببناء دالة تأخذ Parameter** عبارة عن **كلاس** أو إنترفيس. (ماقريتوش انترفيس).

👈 **خذ نفسا عميقا و شاهد الكود ادناه.**

POLYMORPHISM😉

POLYMORPHISM

```java
public class FirstClass {

    public void show(){

        System.out.println("Ta3alam Coding");

    }

}
```

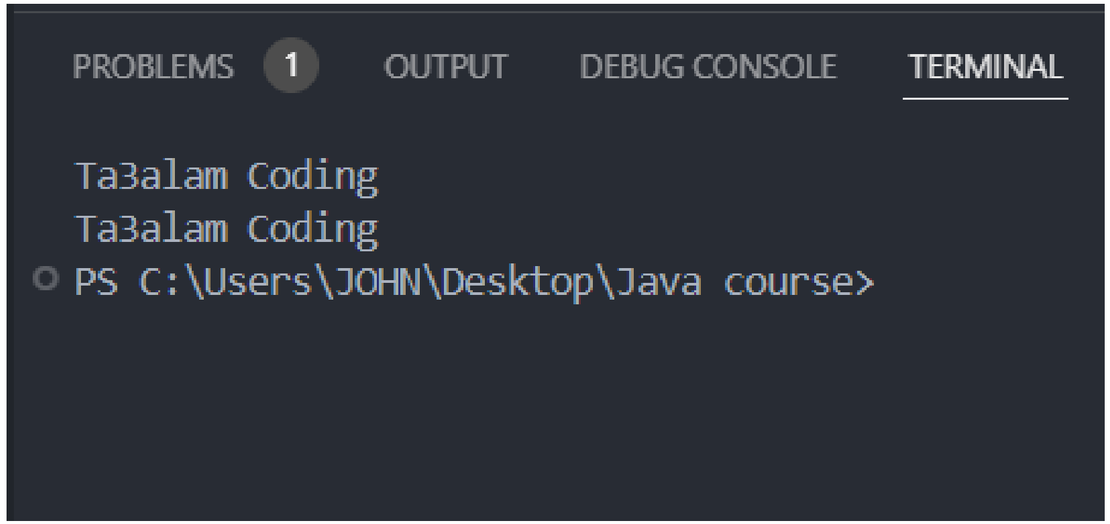
```java
public class SecondClass {

    public void print(FirstClass [] arrays){

        for (FirstClass first : arrays) {
            first.firstClass();
        }

    }
}

```

POLYMORPHISM😉

MAIN

```java
public class Main {
    public static void main(String[] args) {

        FirstClass [] name = {new FirstClass(), new FirstClass()};

        SecondClass secondClass = new SecondClass();

        secondClass.print(name);

    }
}
```

OUTPUT



POLYMORPHISM😉

CODE EXPLANATION

👈 قمنا بعمل كلاس FirstClass و صناعة بداخله Method show التي تقوم بطباعة كلمة Ta3alam Coding.

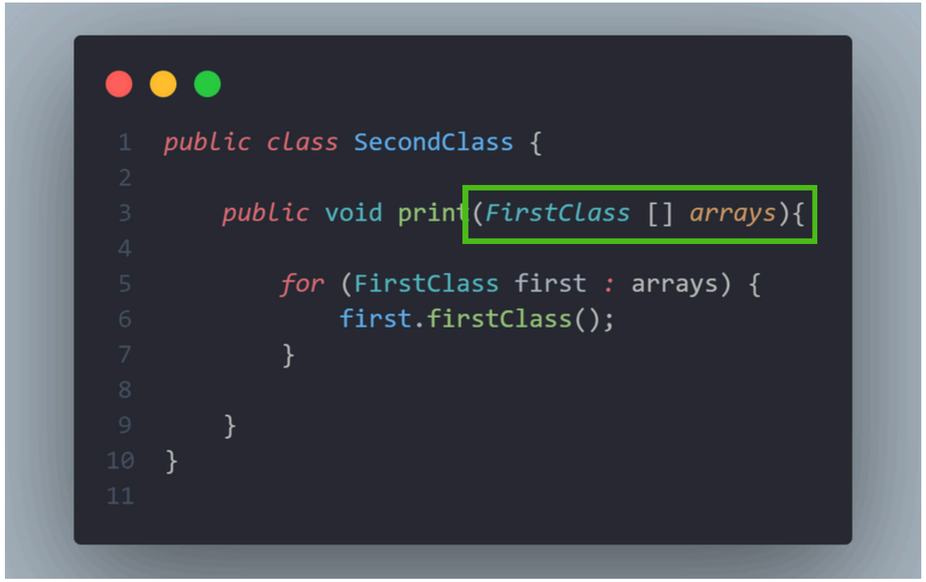
```java
public class FirstClass {

    public void show(){

        System.out.println("Ta3alam Coding");

    }

}
```

POLYMORPHISM😉

CODE EXPLANATION

👈 قمنا بعمل كلاس SecondClass و صناعة بداخله Method Print.

👈 الMethod Print تاخذ Array بداخله Objects من نوع الFirstClass.

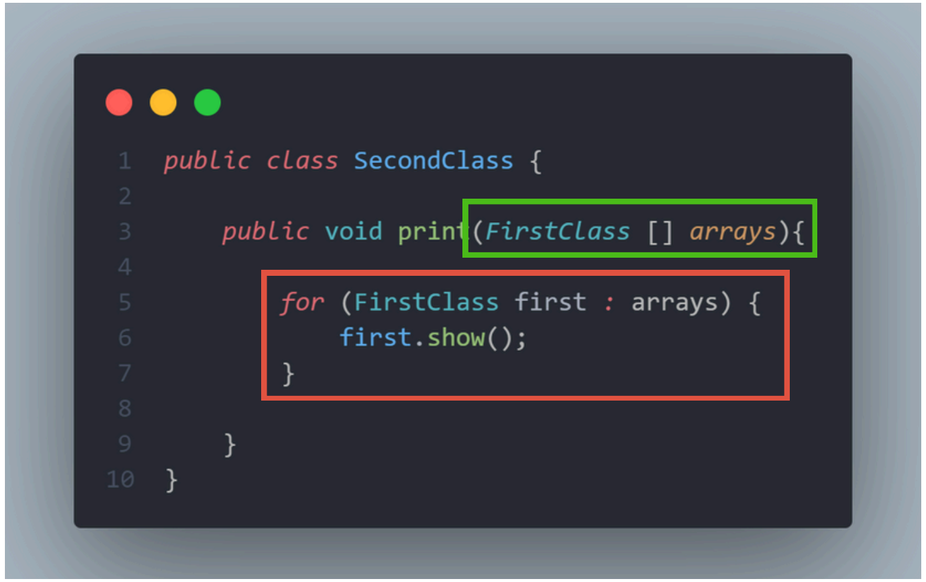


```java
public class SecondClass {

    public void print(FirstClass [] arrays){

        for (FirstClass first : arrays) {
            first.firstClass();
        }

    }
}
```

POLYMORPHISM😉

CODE EXPLANATION

👈 و بما انه **Array** فهو يحتاج **Foreach** **للوصل الى كل القيم التي بداخله.**

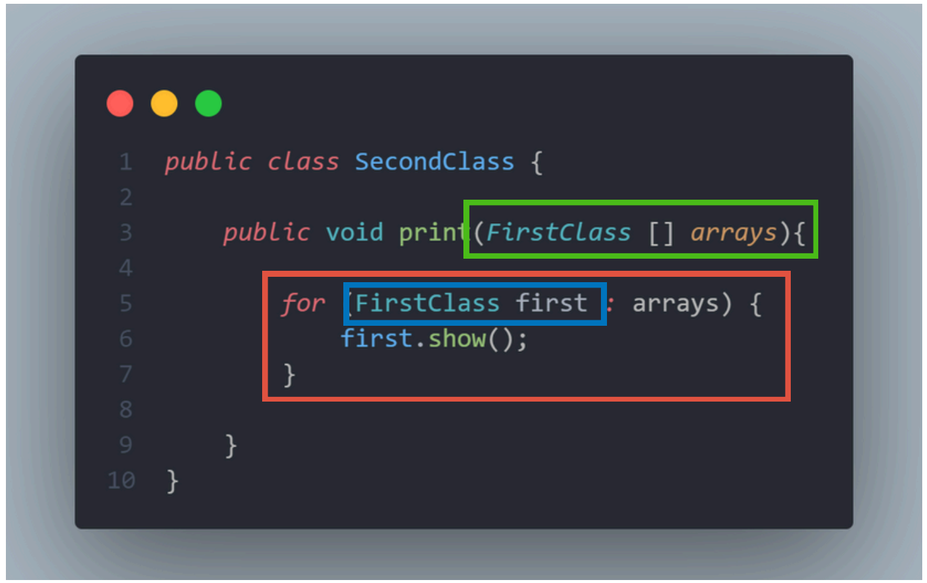


```java
public class SecondClass {

    public void print(FirstClass [] arrays){

        for (FirstClass first : arrays) {
            first.show();
        }

    }
}
```

POLYMORPHISM😉

CODE EXPLANATION

👈 وبما اننا نستعمل **Foreach** **فاعلينا تحديد DataType لل .name الname في هذه الحالة هو first.**



```
public class SecondClass {

    public void print(FirstClass [] arrays){

        for (FirstClass first : arrays) {
            first.show();
        }

    }
}
```

```
for ( DataType name : arrayName ) {

    //Code

}
```

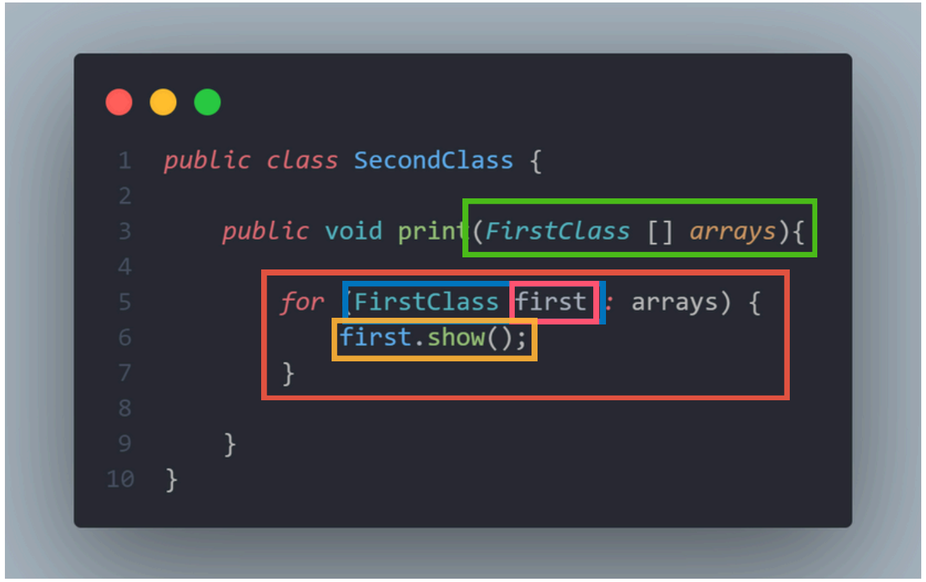
👈 في هذه الحالة **الArray** يحتوي على **Objects من نوع FirstClass,** و بالتالي ال**DataType** ال**name** سيكون من نوع **FirstClass.** (اذا مافهمتش ارجع لل 1 pdf اخر 5 صفحات)

POLYMORPHISM😉

CODE EXPLANATION

👈 وبما ان **المتغير first من نوع FirstClass** فانه **اصبح Object تابع له ,** وبالتالي **الObject first يستطيع الوصول او استداعاء اي متغير او Method موجود في ال FirstClass.**

و في هذه الحالة نريد ان نستدعي ال**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 بانشاء المتغير Array name من نوع FirstClass اي انه سوف يحتوي بداخله على Objects من نوع FirstClass.

```java
public class Main {
    public static void main(String[] args) {

        FirstClass [] name = {new FirstClass(), new FirstClass()};

        SecondClass secondClass = new SecondClass();

        secondClass.print(name);

    }
}
```

POLYMORPHISM😉

CODE EXPLANATION

👈 قمنا باعطاء الArray name قيمة تحتوي على 2 من الObjects FirstClass.

```java
public class Main {
    public static void main(String[] args) {

        FirstClass [] name = {new FirstClass(), new FirstClass()};

        SecondClass secondClass = new SecondClass();

        secondClass.print(name);

    }
}
```

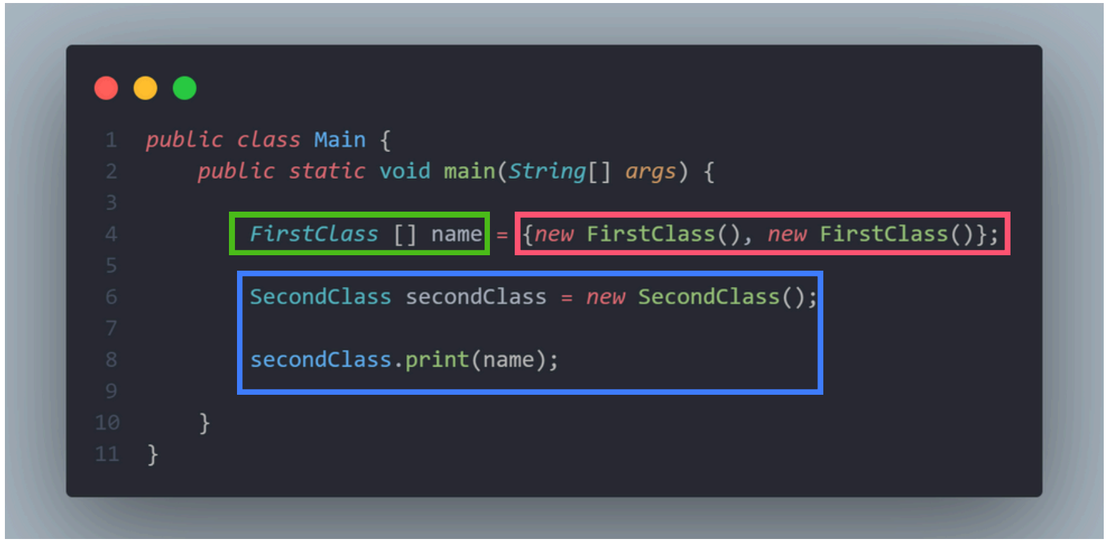
FirstClass [ ] name = {Object , Object};

index	0	1

POLYMORPHISM😉

CODE EXPLANATION

👈 قمنا بصناعة Object لل SecondClass **كالمعتاد** من اجل استخدام ال Method print.



```java
public class Main {
    public static void main(String[] args) {

        FirstClass [] name = {new FirstClass(), new FirstClass()};

        SecondClass secondClass = new SecondClass();

        secondClass.print(name);

    }
}
```

POLYMORPHISM😉

CODE EXPLANATION

👈 بعد تمرير الObject الFirstClass arrays الى داخل ال Method print اصبح لدينا كود هكذا :

```
print(arrays) ->
    for ( FirstClass first : arrays ) {
        first.show();
    }
```

```
FirstClass arrays [ ] = {Object , Object};
index                    0          1
```

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 منادة** في جافا تكون **by copy.**

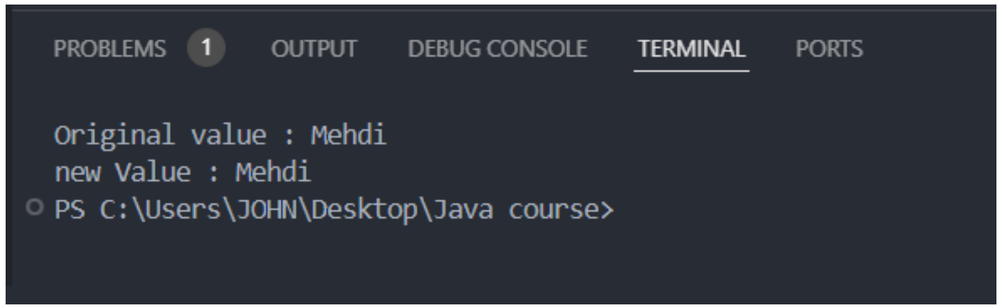
👈 **شاهد الكود ادناه :**

```java
public class byValue {

    String name = "Mehdi";

    public void addName(String Lastname){
        name = name + " " + Lastname;
    }
    //method that tries to change the name value to Mehdi bch
}
```

```java
public class Main {
    public static void main(String[] args) {

        byValue byValue = new byValue();

        String myName = byValue.name;

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

        byValue.addName(" Bch");

        System.out.println("new Value : "+myName); // we expect Mehdi Bch
    }
}
```

BY REFRENCE / BY VALUE

OUTPUT



BY REFRENCE / BY VALUE

BY VALUE

👈 لماذا ! لم تتغير القيمة !

👈 by value : لا تتغير القيمة حتى ولو حدث عليها تغيير داخل الMethod, لان الMethod تقوم باخذ ما تم ادخاله لها و استنساخه , و تقوم بالتعديل و استخدام تلك النسخة وليس القيمة التي تم ادخالها لها.

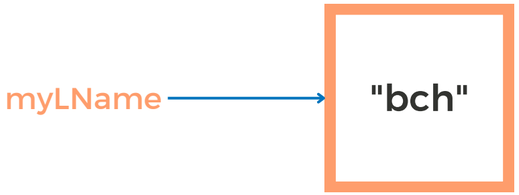
ملاحظة : تعيش النسخة فقط داخل ال Method.

BY REFRENCE / BY VALUE

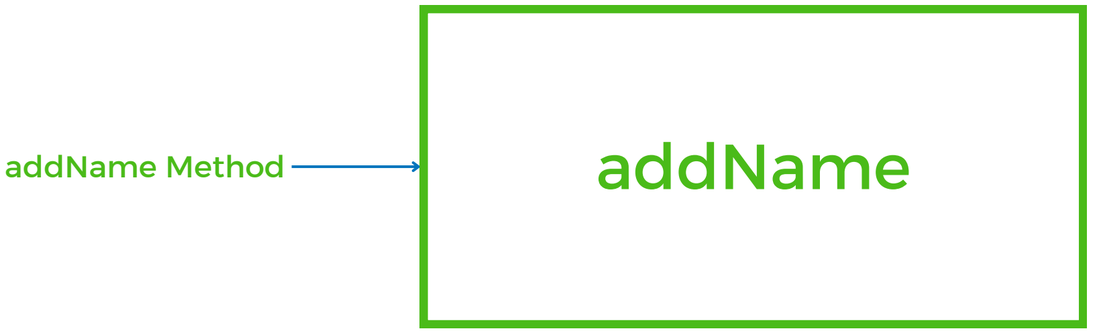
BEHIND THE SCENES

👈 قمنا بانشاء متغير myLName و ادخاله كا parameter للميثود addName.

```
String myLName = “ bch”;
```




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

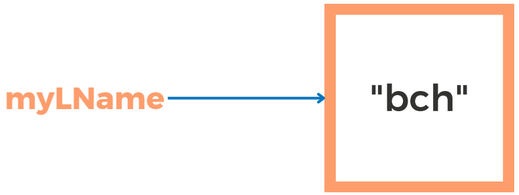


BY REFRENCE / BY VALUE

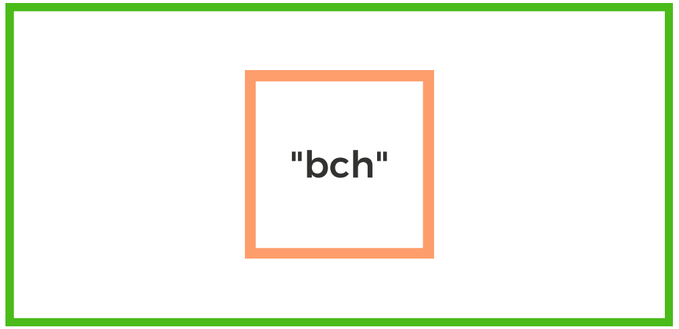
BEHIND THE SCENES

👈 قمنا بانشاء متغير myLName و ادخاله كا parameter للميثود addName.

```
String myLName = “ bch”;
```




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

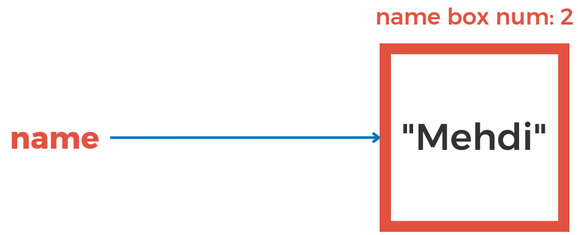


BY REFRENCE / BY VALUE

BEHIND THE SCENES

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

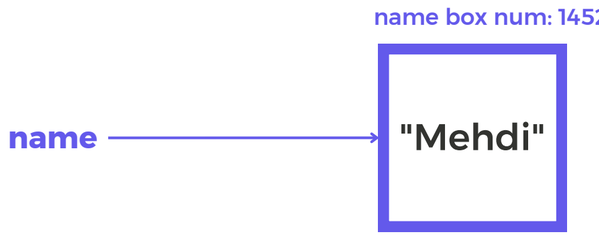
```
class byValue {
      String name = “Mehdi”;
```



👈قبل عمل **الميثود** **addName** على المتغير **name** تقوم

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

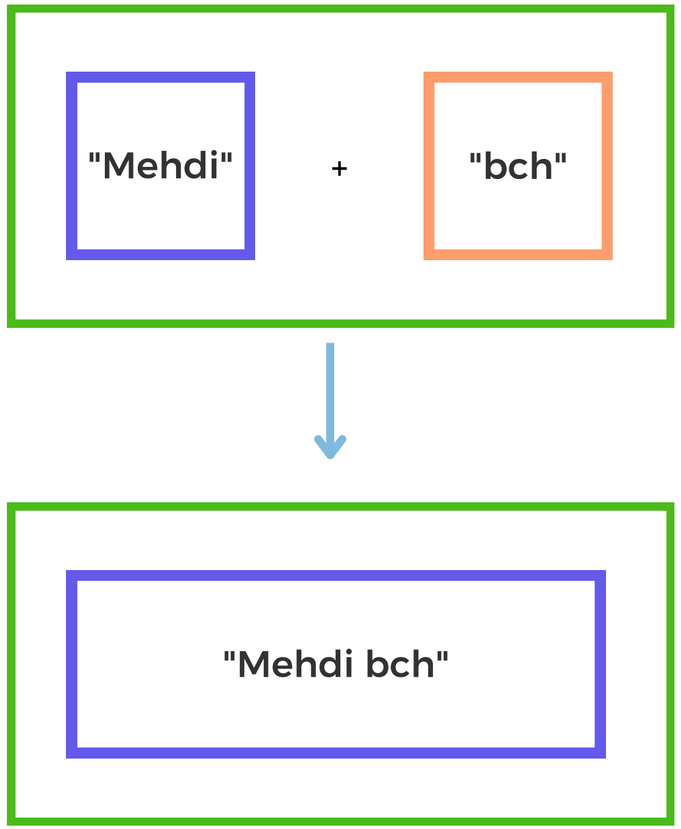
```
      addName(myLName ) {
           name = name + myLName ;
       }
}
```

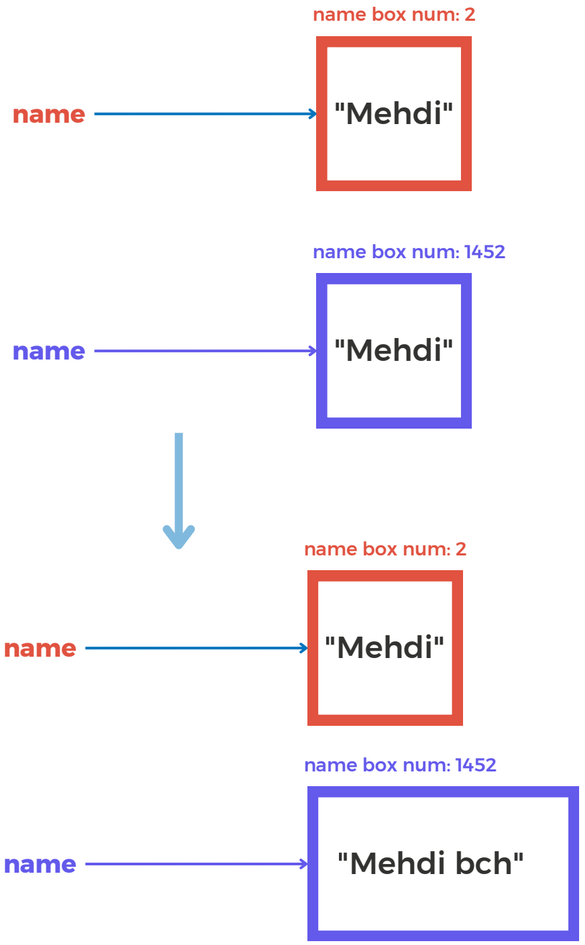


BY REFRENCE / BY VALUE

```
class byValue {
      String name = “Mehdi”;


       addName(myLName ) {
            name = name + myLName ;
        }
}
```

BEHIND THE SCENES

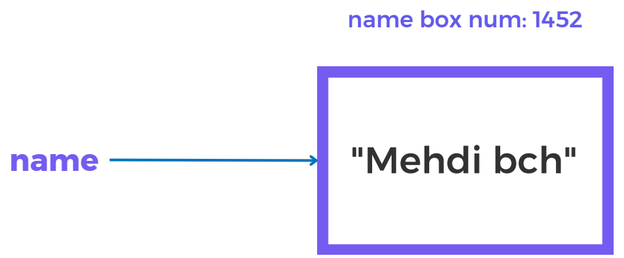




BY REFRENCE / BY VALUE

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
```



BY REFRENCE / BY VALUE

BY REFRENCE

👈 تتغير القيمة اذا حدث عليها تغيير داخل الMethod.

👈 في جافا الObjects فقط التي تكون دائما by refrence, اي بمعنى اخر اي Object يتم ادخاله كParameter داخل الMethod لا يتم استنساخه ولكن استعماله كما هو, اي سيحدث عليه تغيير.

👈 لناخذ المثال السابق و نعدل عليه.

BY REFRENCE / BY VALUE

BY REFRENCE

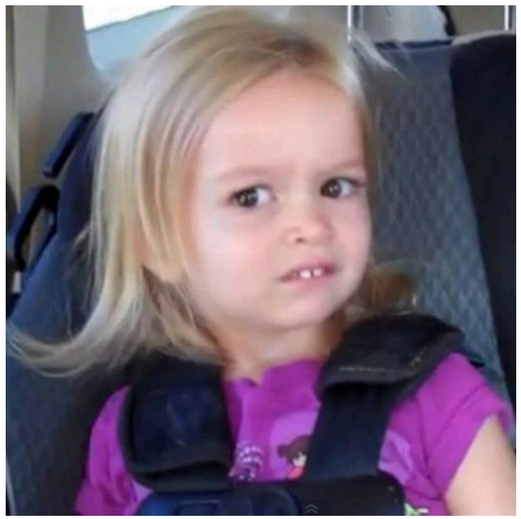
```java
public class byReference {

    String name = "Mehdi";

    public void addName(byReference myName , String lastName){

        myName.name = myName.name  + lastName;

    }
}
```

```java
public class Main {
    public static void main(String[] args) {

        byReference byRf = new byReference();

        byReference myName = new byReference();

        String myLName = " Bch";

        System.out.println("Original value : "+ myName.name); // we expect Mehdi

        byRf.addName(myName,myLName);

        System.out.println("new Value : "+myName.name); // we expect Mehdi Bch
    }
}
```

BY REFRENCE / BY VALUE

BY REFRENCE

my reaction ida 9oteli mafehamtch :))



nefahmek bel derahme chriki :))

OUTPUT

```
PROBLEMS 1   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

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

BY REFRENCE / BY VALUE

CODE EXPLANATION

👈 عندنا **كلاس اسمه** byReference يحتوي على Global variable (name) و Method addName و نريد عمل **تغيير على القيمة الحقيقية** لل**name** **الموجود خارج ال**Method **باستخدام هذه الاخيرة.**

```
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 الMethod addName الObject myName ذو النوع byReference (اي انه instance) على شكل Parameters.

👈 اذا myName عبارة عن Object.

```
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 الObject myName ذو النوع byReference (myName هو instance لل byReference)

فانه يستطيع الوصول الى كل الMethods و كل المتغيرات الموجودة بداخل الكلاس byReference.

```
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

👈 اذا يستطيع الوصول الى **المتغير** name الموجود بداخل **الكلاس** و استخدامه.

👈 قلنا : اجلب الname الموجود بداخل الكلاس **byReference** و اعمل عليه هذه التغييرات .

BY REFRENCE / BY VALUE

CODE EXPLANATION

👈 اذا يستطيع الوصول الى **المتغير** name الموجود بداخل **الكلاس** و استخدامه.

👈 قلنا : اجلب الname الموجود بداخل الكلاس **byReference** و اعمل عليه هذه التغييرات .

BY REFRENCE / BY VALUE

CODE EXPLANATION

- قمنا بصناعة **Object byRf** كالمعتاد من اجل استخدام مابداخله من **متغيرات و ميثودس.**
- قمنا بصناعة **Object myName** كالمعتاد من اجل استخدام مابداخله من **متغيرات و ميثودس.**
- **قمنا بعمل متغير myLName يحمل قيمة Bch.**

```java
public class Main {
    public static void main(String[] args) {

        byReference byRf = new byReference();

        byReference myName = new byReference();

        String myLName = " Bch";

        System.out.println("Original value : "+ myName.name); // we expect Mehdi

        byRf.addName(myName,myLName);

        System.out.println("new Value : "+myName.name); // we expect Mehdi Bch
    }
}
```

BY REFRENCE / BY VALUE

CODE EXPLANATION

👈 قمنا بجلب **المتغير name** الموجود بداخل **الكلاس byReference** بواسطة ال**Object myName** وطباعته.

👈 بعد ذلك قمنا **باستدعاء ال**Method addName بواسطة **ال**Object byRf و ادخال **ال**Object myName

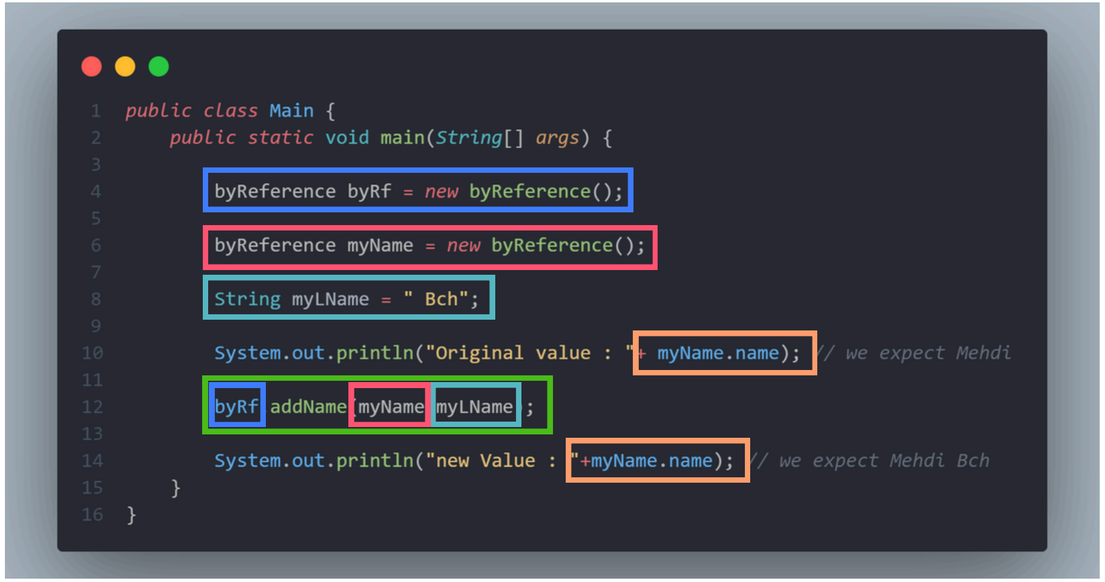
و المتغير myLName كا Parameters لتقوم **الميثود بتغيير قيمة** .name

```java
public class Main {
    public static void main(String[] args) {

        byReference byRf = new byReference();

        byReference myName = new byReference();

        String myLName = " Bch";

        System.out.println("Original value : "+ myName.name); // we expect Mehdi

        byRf.addName(myName, myLName);

        System.out.println("new Value : "+myName.name); // we expect Mehdi Bch
    }
}
```

BY REFRENCE / BY VALUE

CODE EXPLANATION

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



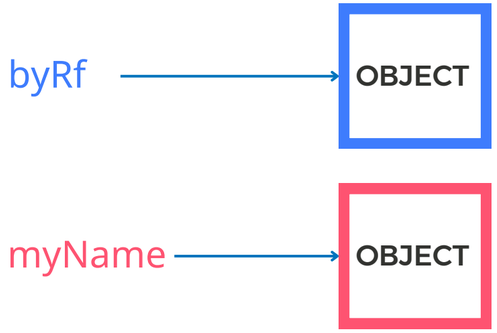
```java
public class Main {
    public static void main(String[] args) {

        byReference byRf = new byReference();

        byReference myName = new byReference();

        String myLName = " Bch";

        System.out.println("Original value : "+ myName.name); // we expect Mehdi

        byRf.addName(myName, myLName);

        System.out.println("new Value : "+myName.name); // we expect Mehdi Bch
    }
}
```

BY REFRENCE / BY VALUE

BEHIND THE SCENES

👈 عند صناعة Object لكلاس معين, ذلك الObject سيحمل جميع المتغيرات و جميع الMethods الموجودة بداخل هذا الكلاس ,اي يستطيع الوصول الى اي متغير او اي Method التابعة للكلاس و تنفيذها.

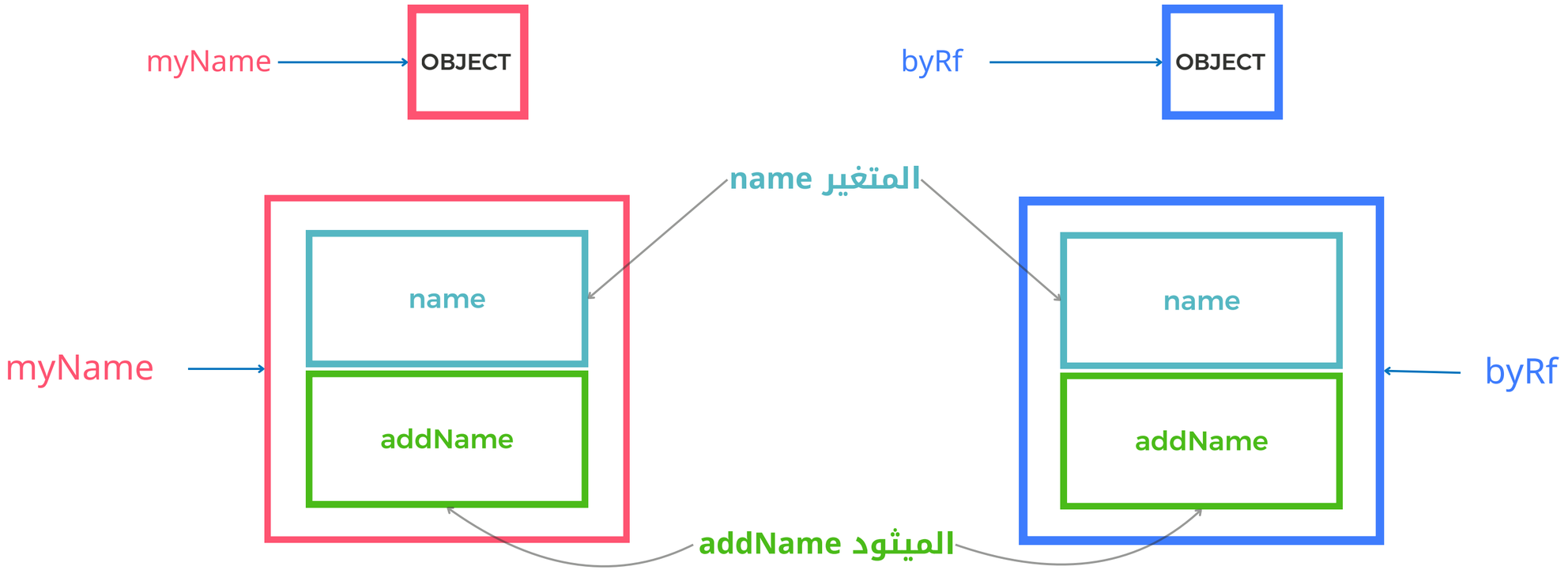
```
byReference byRf = new byReference();
byReference myName = new byReference();
String myLName = " Bch";
```



BY REFRENCE / BY VALUE

BEHIND THE SCENES

👈 في هذه الحالة **الكلاس** byReference يحتوي على **متغير واحد** و **ميثود واحدة** فقط.

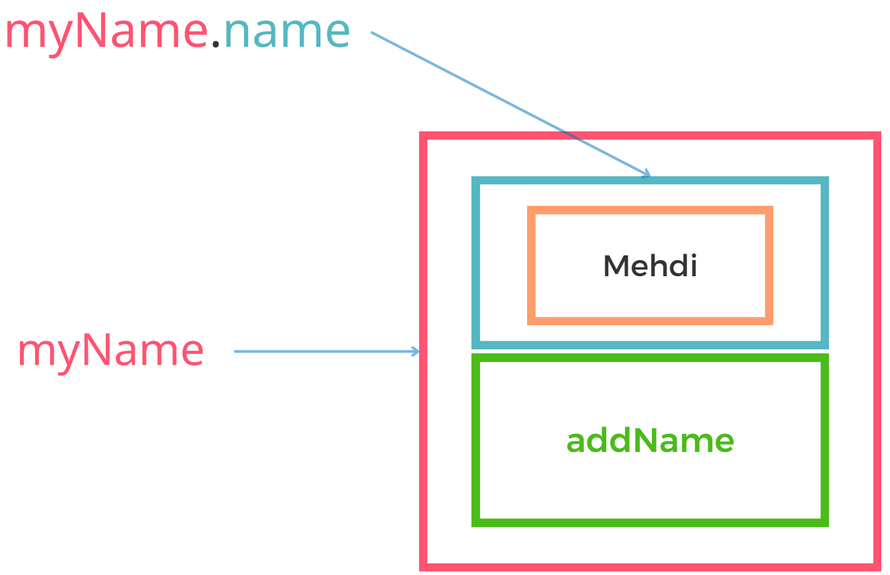


BY REFRENCE / BY VALUE

BEHIND THE SCENES

لجلب اي متغير او ميثود ما من Object نقوم ب Object.variableName/MethodName 👈

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



Original value : Mehdi

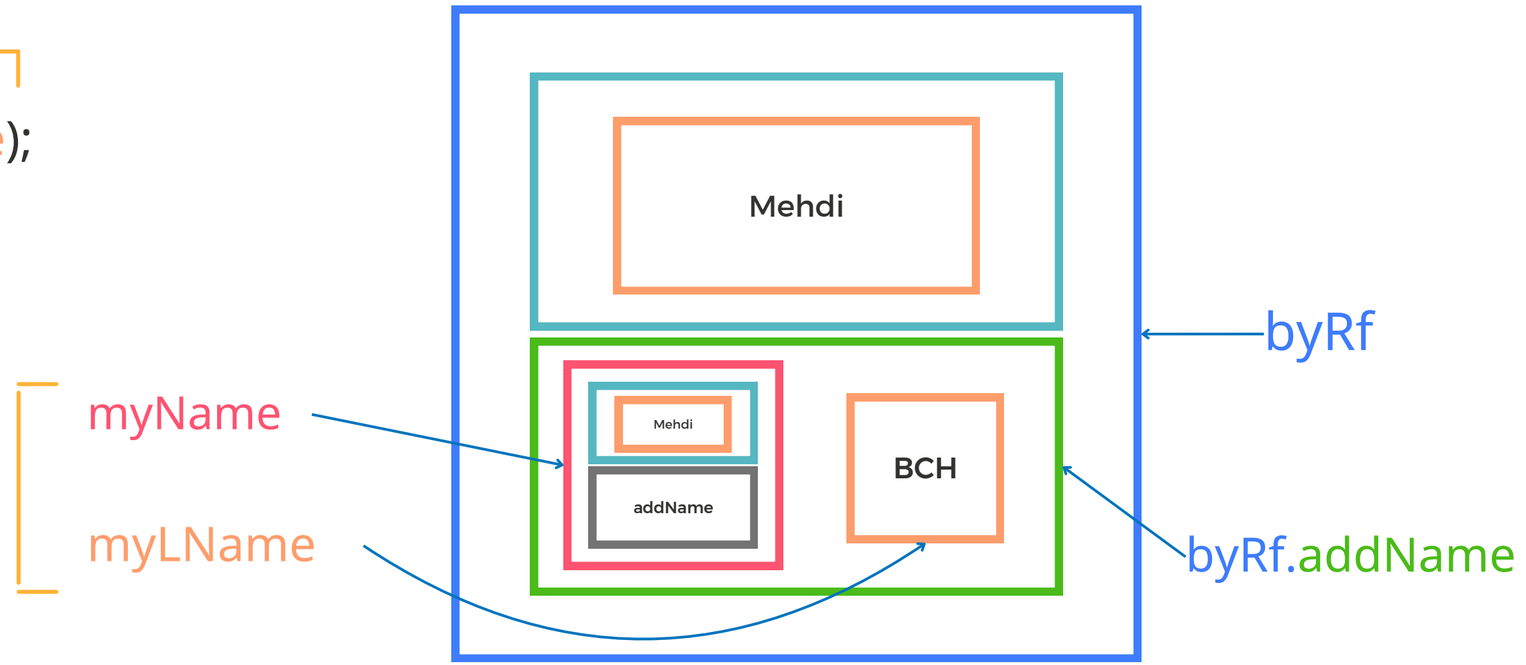
BY REFRENCE / BY VALUE

BEHIND THE SCENES

👈 قمنا باستدعاء **الميثود addName** **الموجودة في الbyRf Object** و **ادخال myName Object** و

myLName كا**Parameters** الى داخل ال **Method addName**

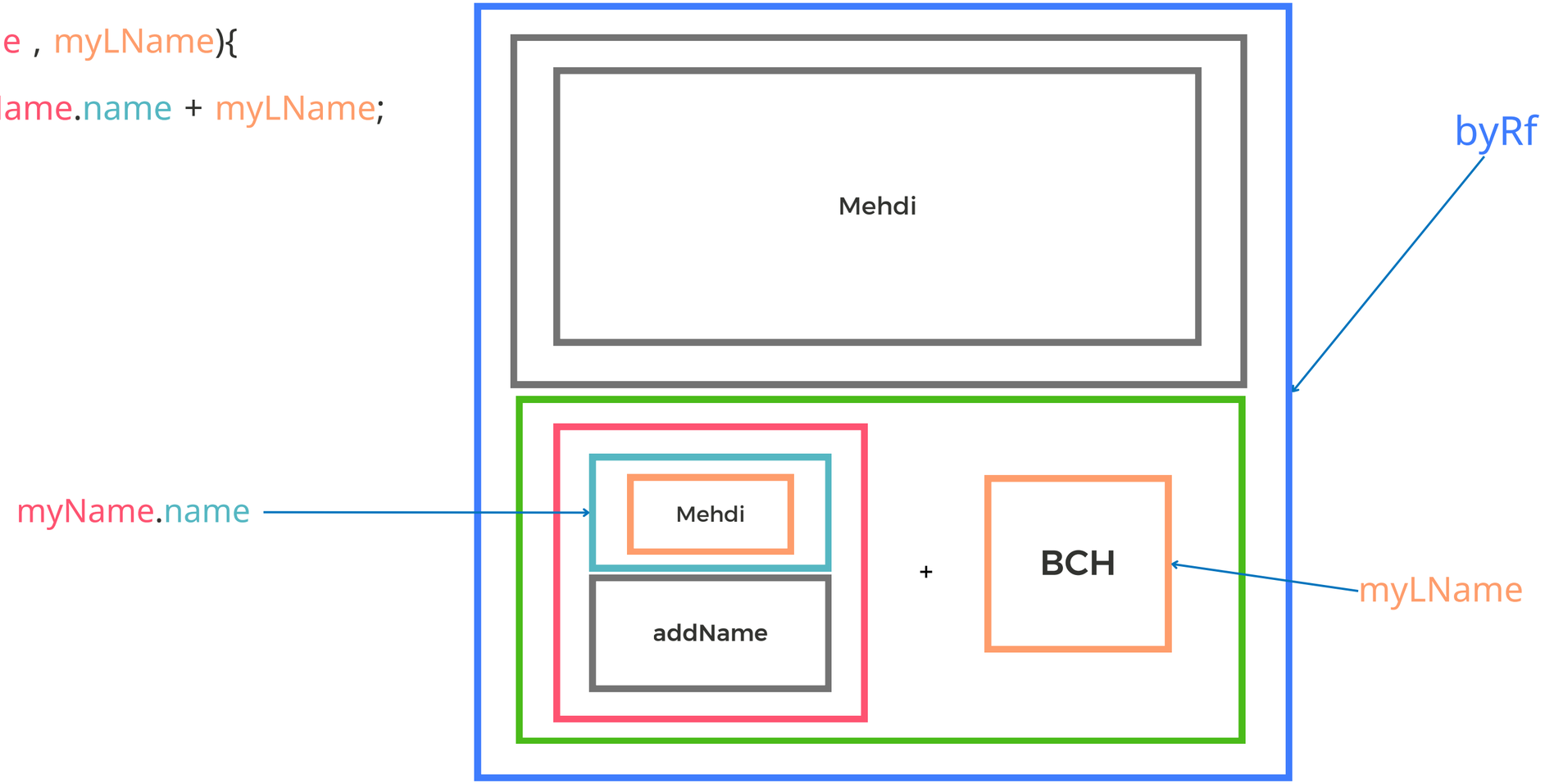
```
byRf.addName(myName,myLName);
```



BY REFRENCE / BY VALUE

BEHIND THE SCENES

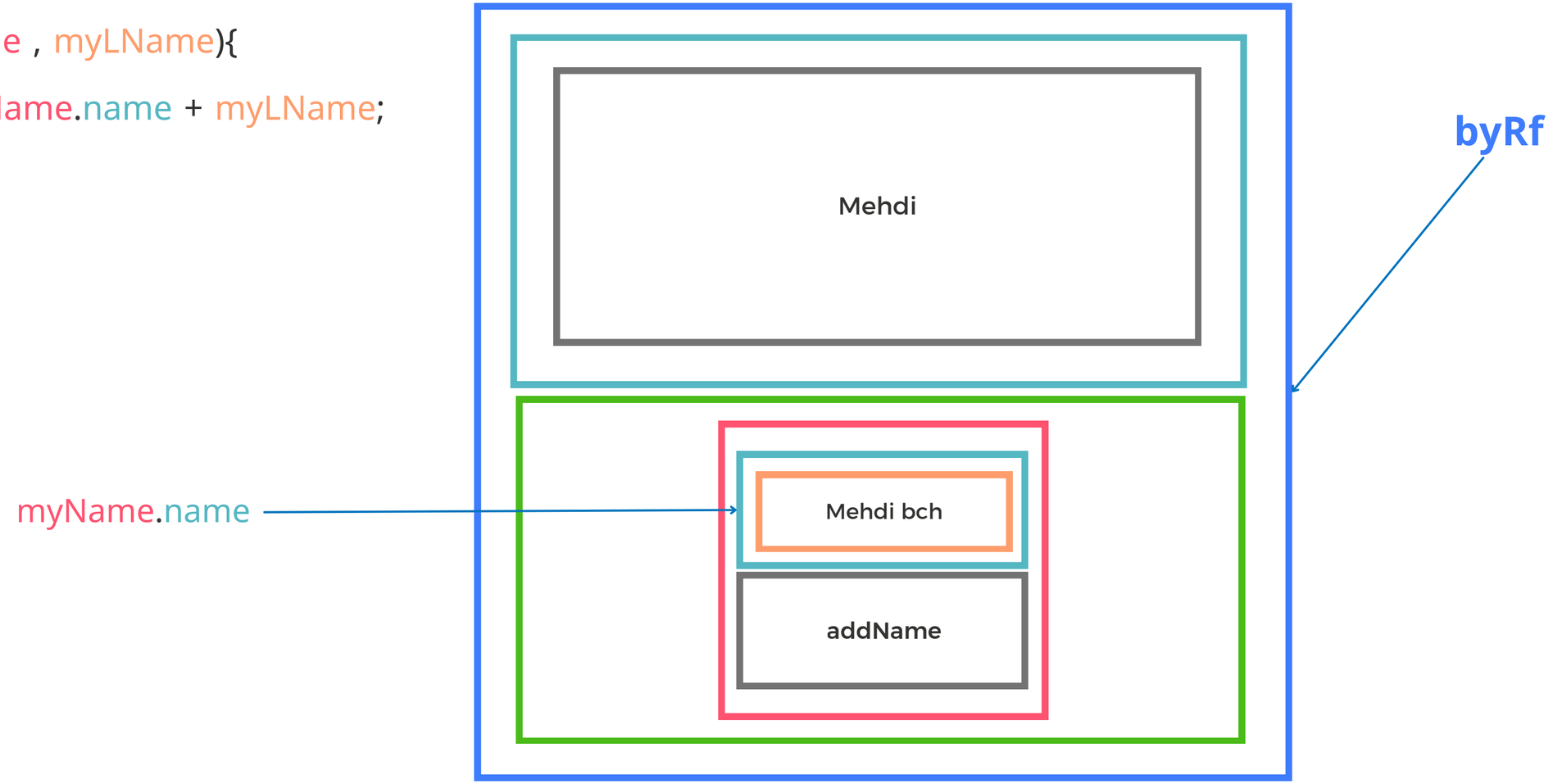
```
byRef.addName(myName , myLName){
    myName.name = myName.name + myLName;
}
```



BY REFRENCE / BY VALUE

BEHIND THE SCENES

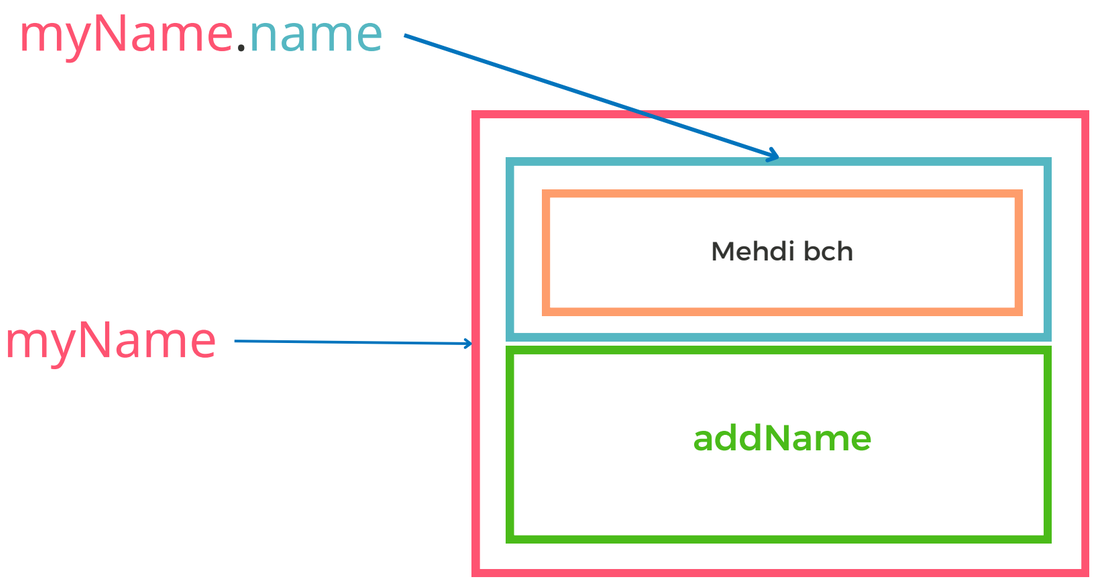
```
byRef.addName(myName , myLName){
    myName.name = myName.name + myLName;
}
```



BY REFRENCE / BY VALUE

BEHIND THE SCENES

👈 بعد عمل ال**Method addName** يصبح ال**myName Object** هكذا :

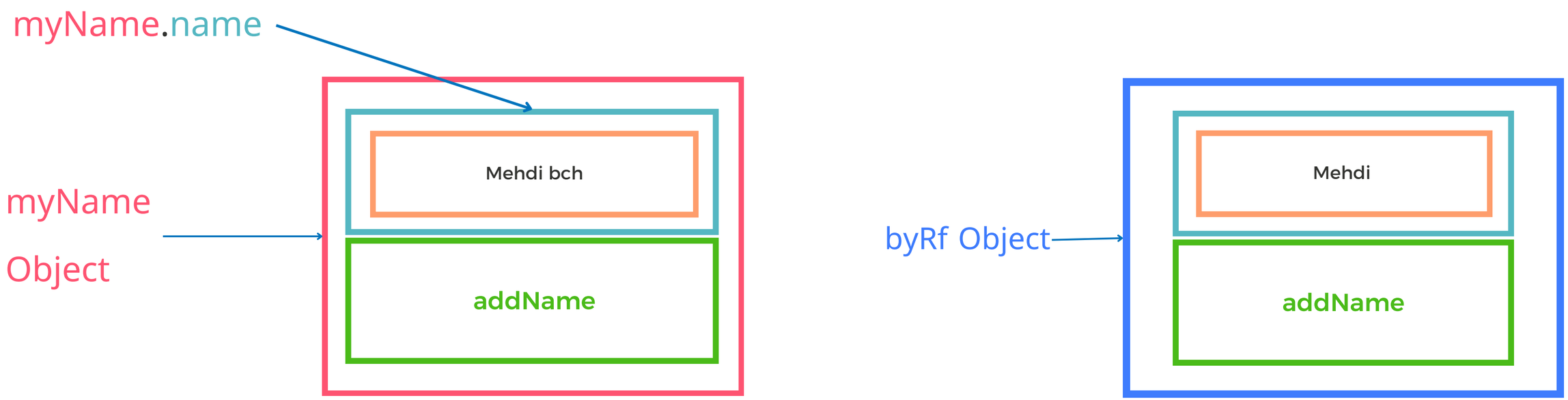


BY REFRENCE / BY VALUE

BEHIND THE SCENES

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

new Value : Mehdi bch



BY REFRENCE / BY VALUE

REMINDER!

👈 الArrays عبارة عن Object (تفكرت كي قتلك في PDF 01 ابقى شافي عليها), اذا مدام Object فادائما عندما يتم ادخاله على شكل Parameter يكون by Reference. **بمعنى اخر اي تعديل عليه داخل الميثود سيكون عنده تاثير.**

EXAMPLE :

👇

BY REFRENCE / BY VALUE

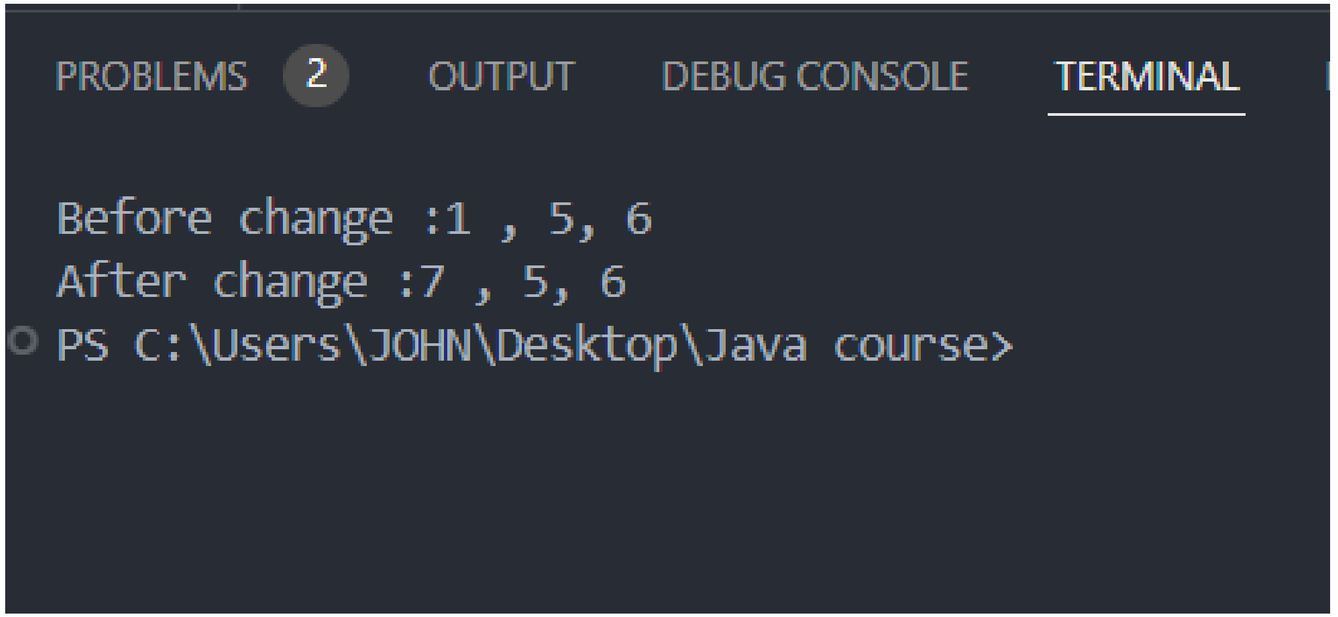
REMARQUE!

```java
public class Main {
    public static void main(String[] args){

    int[] T = new int[] {1,2,3,4};
    int x = 5;
    int y = 6;

    Mehdi mehdi = new Mehdi();

    System.out.println("Before change :"+ T[0]+" , "+x+", "+y);

    mehdi.changeValues(T, x, y);
    System.out.println("After change :"+ T[0]+" , "+x+", "+y);

    }
}
```

```java
public class Mehdi{
    public void changeValues( int[] T, int x , int y) {

        T[0] = 7;
        x = 8;
        y= 9;

    }
}
```

BY REFRENCE / BY VALUE

OUTPUT



CLASS INSTANCES

CLASS INSTANCES

👈 عندما نقوم **بصناعة الObject لكلاس** ما من اجل استخدام **ميثود او متغير الذي بداخله** فنحن في الحقيقة نقوم **بصناعة instance لهذا الكلاس** او بكلمة اقرب **نحن نستنسخه و نقوم باستعمال تلك النسخة**.

(مش نسخة ماتروحش تبهدلنا مع البراني راني نمدلك برك في الصورة باش تفهم واش معناه instance).

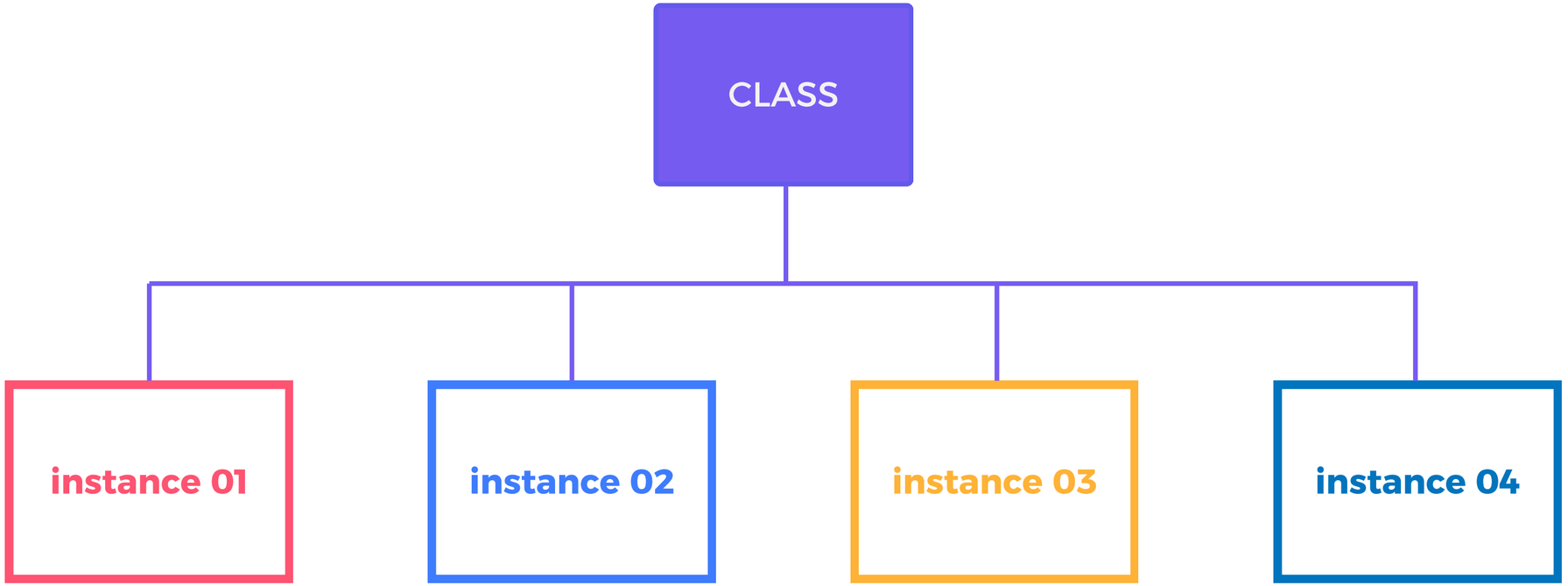
👈 كل **instance** او كل **نسخة لديها id** خاص بها (behind the scenes) لذلك عندما نقوم **بانشاء نسختين من نفس الكلاس و نقوم بمقارنتهم دائما الناتج يكون false** (راني نقولك كي **ننشاء** ماشي كي **نحيل** نسخة الى نسخة).

👈 يمكن لاي كلاس ان يكون لديه عدد غير محدد من **الinstances**.

CLASS INSTANCES

CLASS INSTANCES

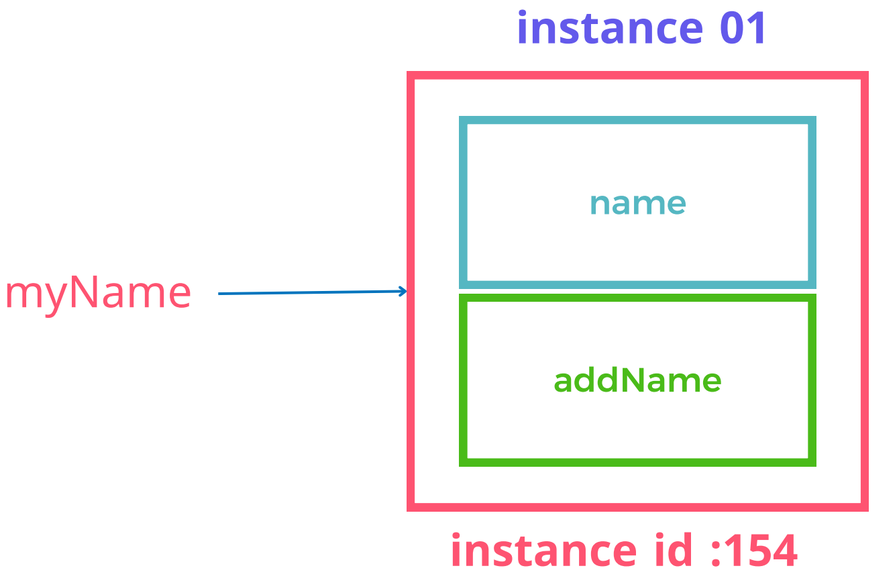
👈 يحتوي الinstance على كل الConstructors و الMethods و الVariables و الObjects الموجودين داخل الClass.

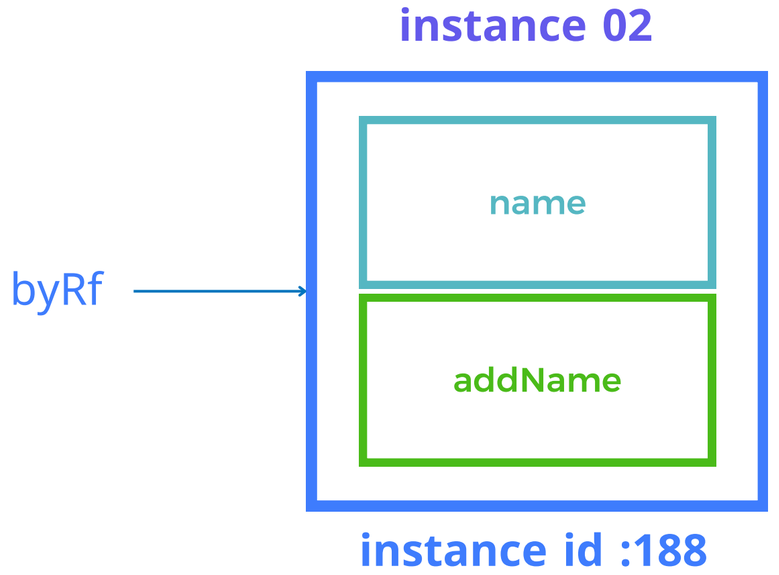


CLASS INSTANCES

CLASS INSTANCES

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

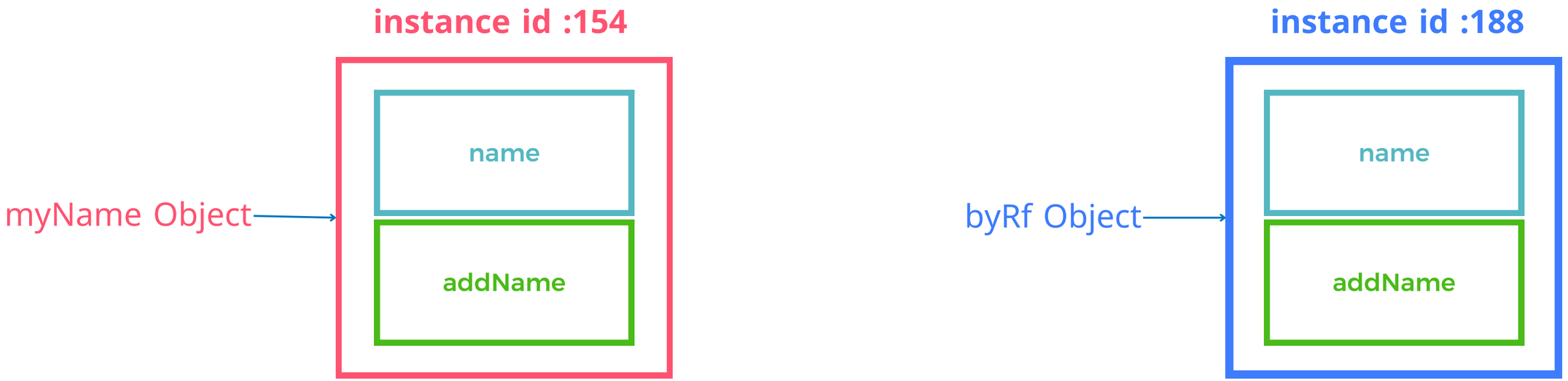




CLASS INSTANCES

CLASS INSTANCES

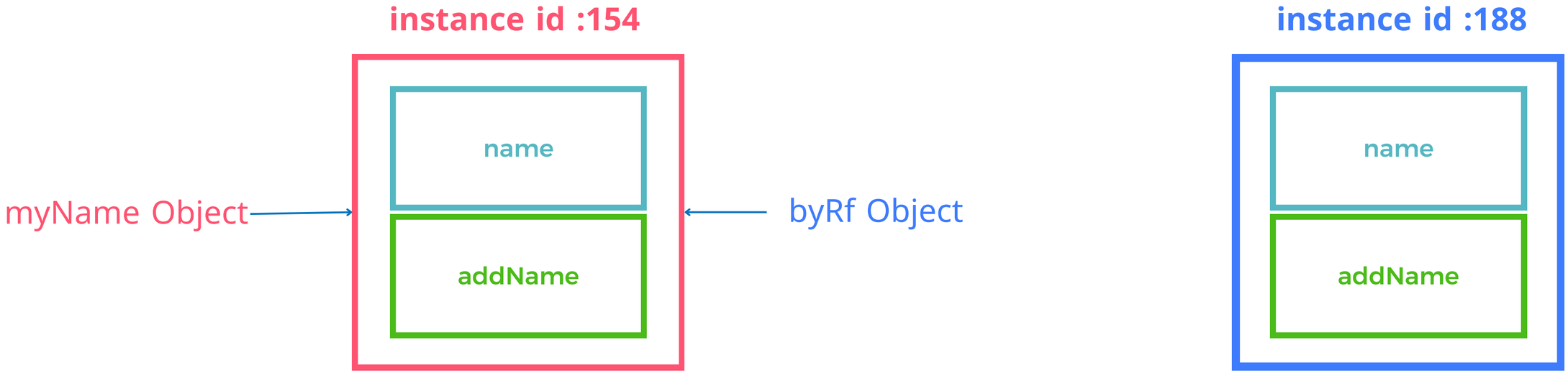
```
byReference byRf = new byReference();

byReference myName = new byReference();

byRf == myName ? => FALSE
```



CLASS INSTANCES

CLASS INSTANCES

```
byReference byRf = new byReference();
byReference myName = new byReference();
byRf = myName ; //نقارن مش ربي ياعبد قيمة الى قيمة **نحيل/نمد** راني
byRf == myName ? => TRUE
```



THE END

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

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

Mehdi Bouchachi

SEE YOU SOON...