OBJECTIVES

✓ Check if **2 objects are equal**

✓ Understand what is **immutability**

✓ **Clone** objects

✓ Be able to choose between :

  ✓ An object with ID *(an **entity**)*

  ✓ An object without ID *(a **value object**)*

Equality of objects

What this code will print ?

```
let number1 = 45
let number2 = 45

console.log(number1 == number2);
```

1 true **2** false **3** error

What this code will print ?

```
let number1 = 45
let number2 = 45

console.log(number1 == number2);
```

The best team ??

[✓ 1] true

2 false

3 error

What this code will print ?

```
let array1 = [1, 2, 3];
let array2 = [1, 2, 3];

console.log(array1 == array2);
```

1 true

2 false

3 error

What this code will print ?

```
let array1 = [1, 2, 3];
let array2 = [1, 2, 3];

console.log(array1 == array2);
```

Can you explain why ?

1 true [✓ 2 false] 3 error

What this code will print ?

```
class Student {
  constructor(private name) {}
}

let student1 = new Student("hak");
let student2 = new Student("hak");

console.log(student1 == student2);
```

1 true

2 false

3 error

What this code will print ?

```
class Student {
  constructor(private name) {}
}

let student1 = new Student("hak");
let student2 = new Student("hak");

console.log(student1 == student2);
```

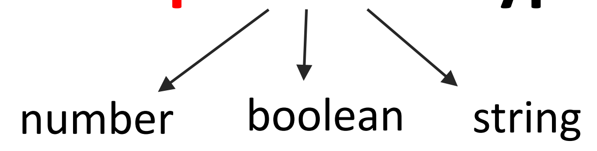
Can you explain why ?

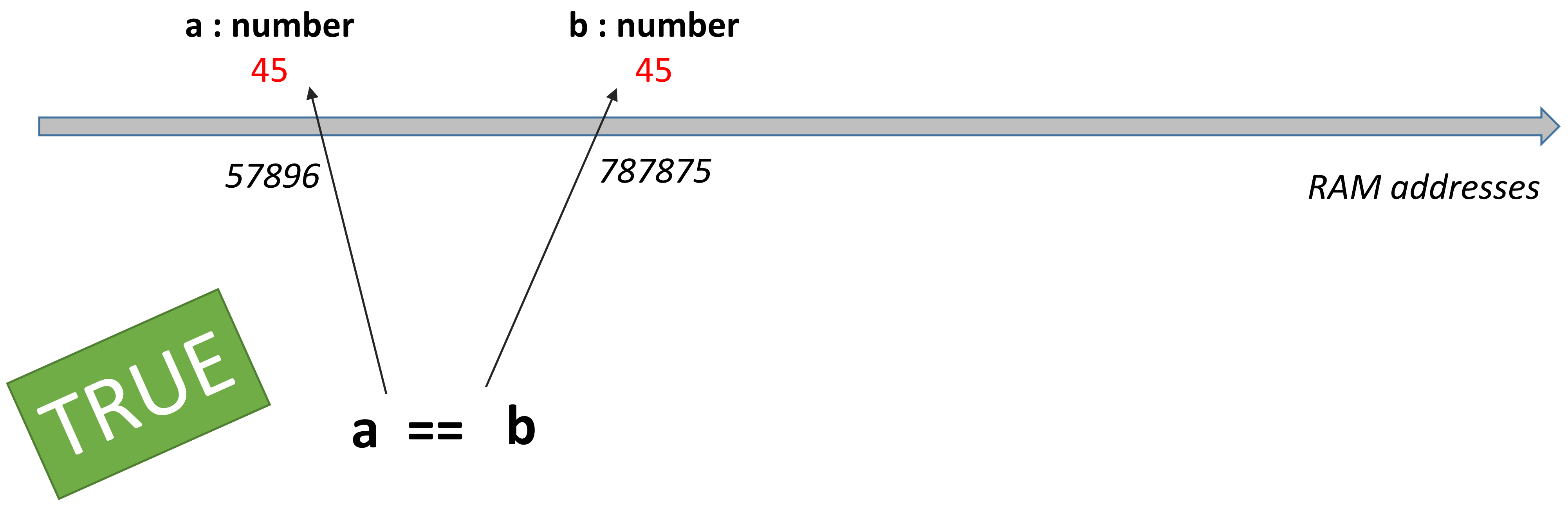
1 true [✓ 2 false] 3 error

EXPLAIN

10MIN

For primitive types, == is performed on values

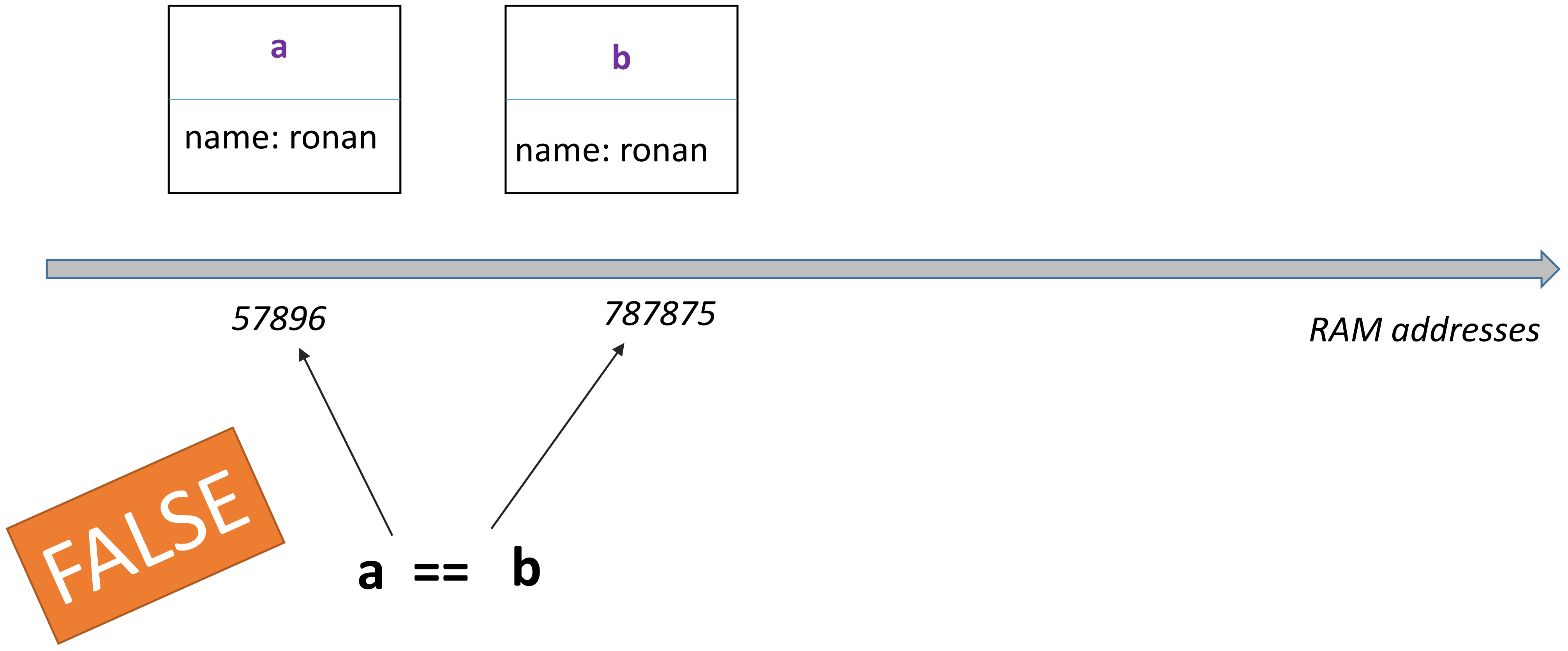




EXPLAIN

10MIN

For object or arrays, == is performed on RAM address



EXPLAIN

10MIN

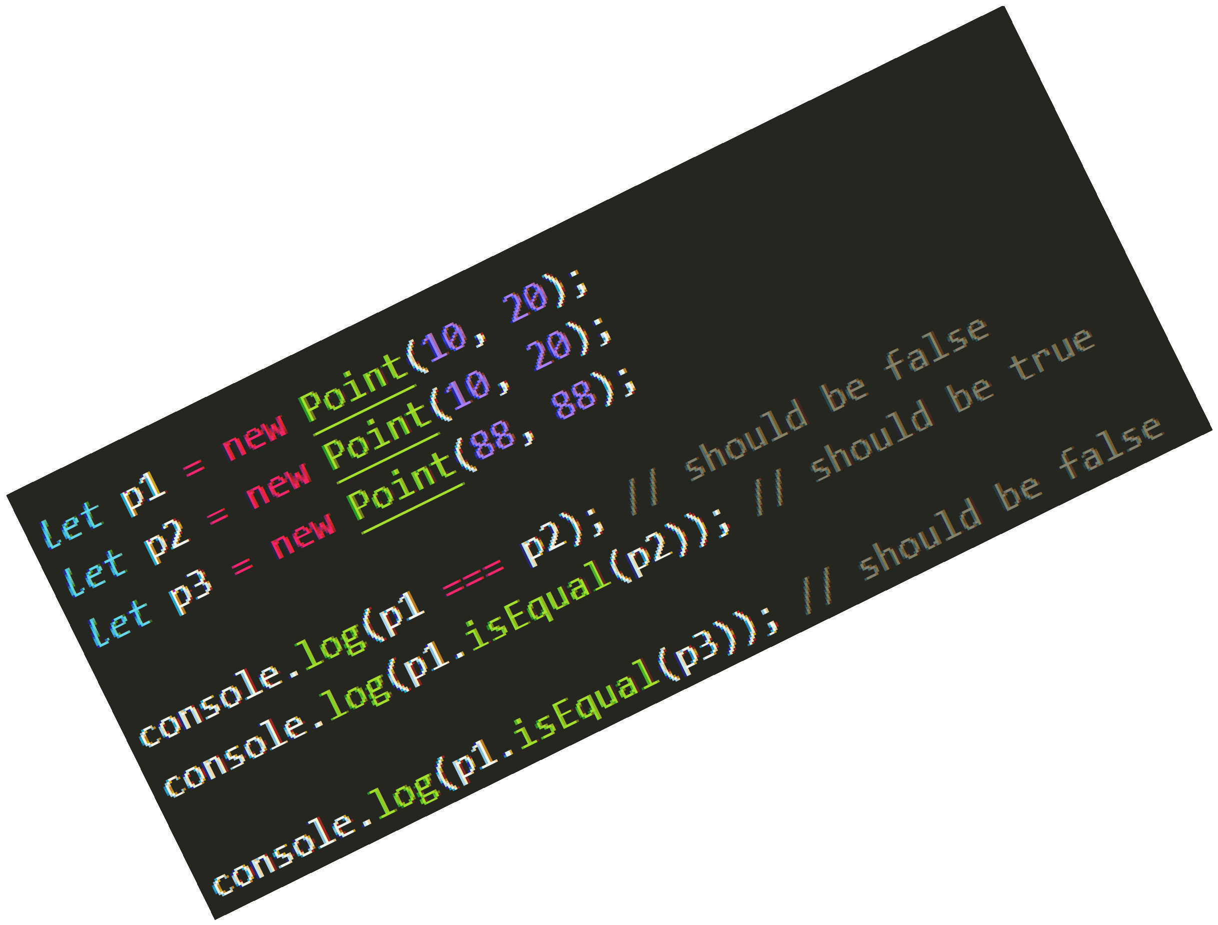
To compare 2 objects by values, we need to create a method equal

```
class Student {
  constructor(private name) {}

  isEqual(other: Student) {
    return this.name == other.name;
  }
}

let student1 = new Student("hak");
let student2 = new Student("hak");

console.log(student1.isEqual(student2));
```

What will
This code display now ?

ACTIVITY 1

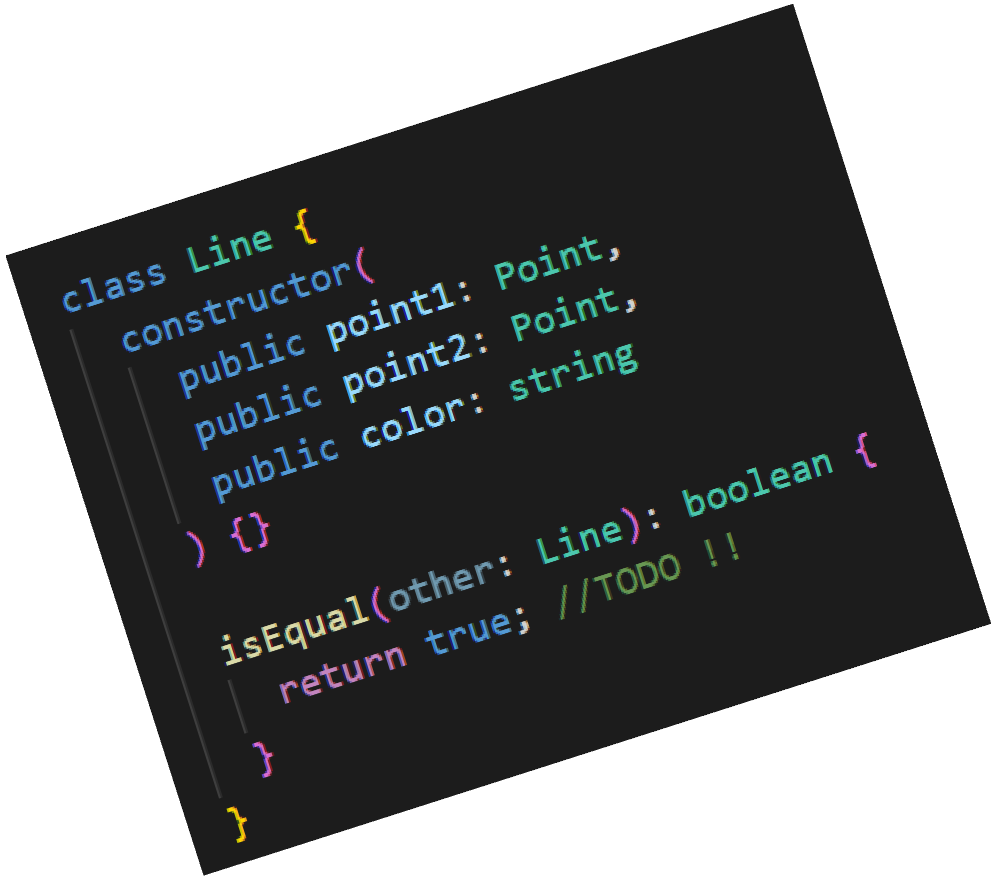
```
//  1 – Check p1 === p2 return false

//  2 – Implement the equal method on Point class

//  3 – Check that p1.equal(p2) return true

//  4 – Check that p1.equal(p3) return false
```



```
let p1 = new Point(10, 20);
let p2 = new Point(10, 20);
let p3 = new Point(88, 88);

console.log(p1 === p2); // should be false
console.log(p1.isEqual(p2)); // should be true

console.log(p1.isEqual(p3)); // should be false
```

ACTIVITY 2

```
//  1 – Implement the equal method on Line class

//  2 – Check  your code with 2 different cases
     (equal and not equal)
```



```
class Line {
  constructor(
    public point1: Point,
    public point2: Point,
    public color: string
  ) {}

  isEqual(other: Line): boolean {
    return true; //TODO !!
  }
}
```

Let’s sum up !

```
let a = 45;
let b = 45;

      a == b
```

True because a and b are

primitive types

For primitives   == is done is on the value

```
let a = new Person(‘x’);
let b = new Person(‘x’);

      a == b
```

False because a and b are

object types

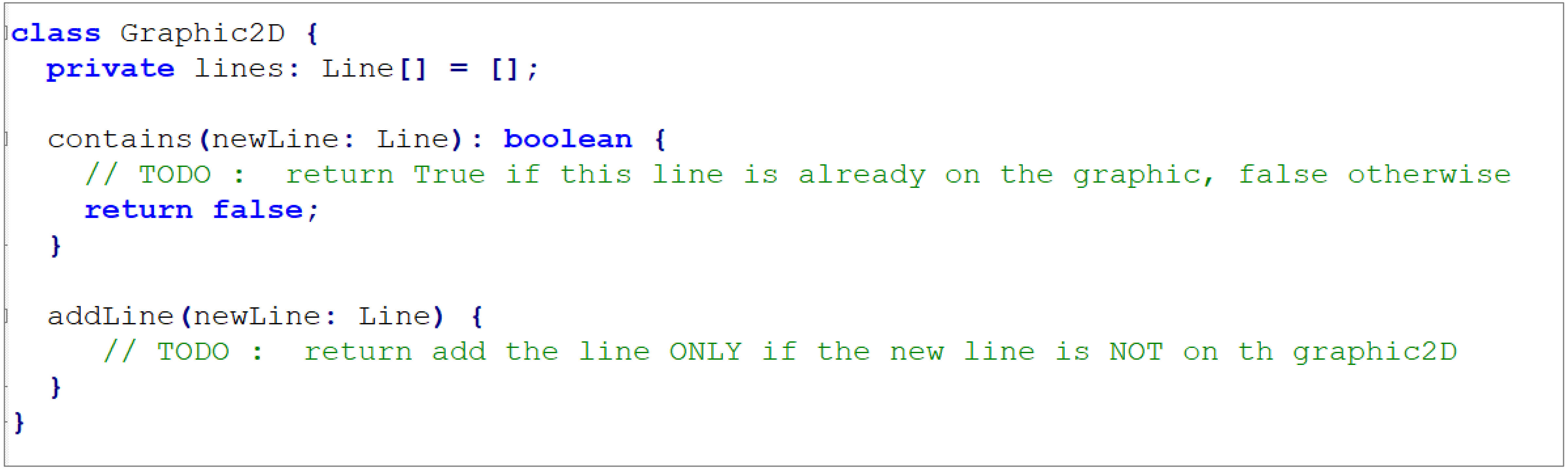
For object == is done is on the ***@ in RAM***

```
let a = new Person(‘x’);
let b = new Person(‘x’);

      a.equals(b)
```

True because now we really compare the 2 persons using their attributes

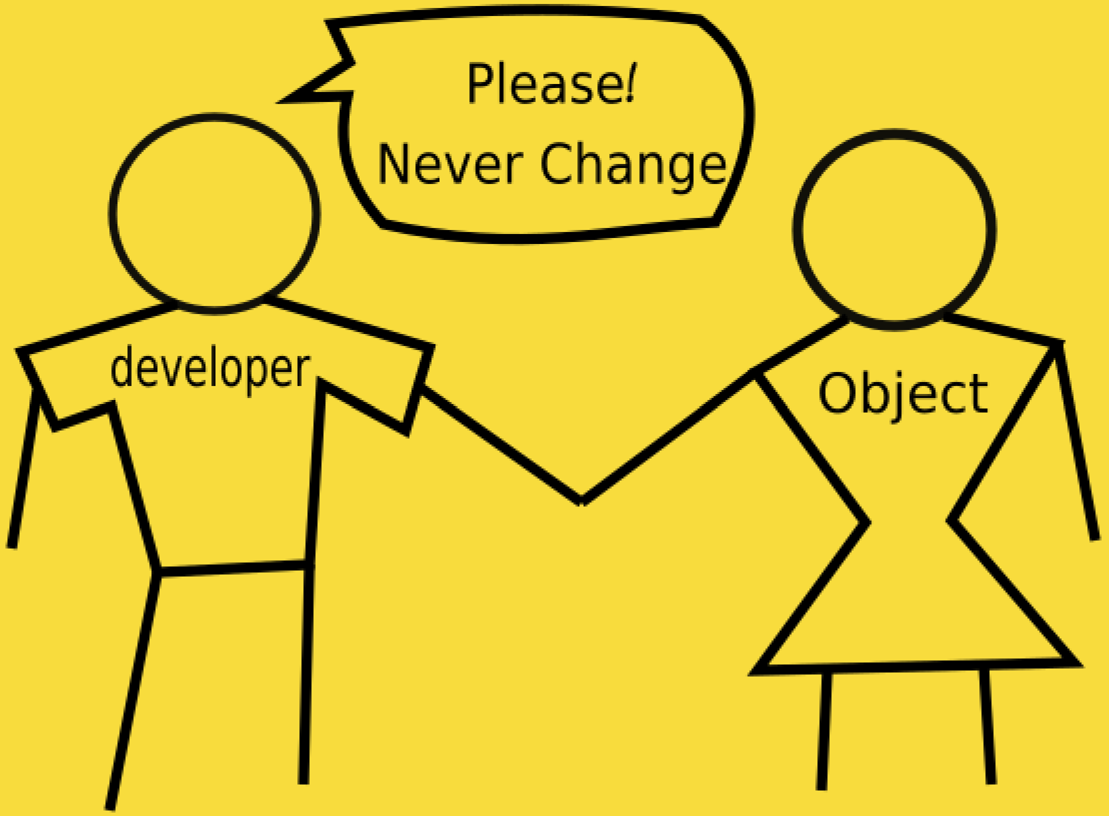
ACTIVITY 3

```
//  1 – Implement the contains method on Graphic class

//  2 – Complete the addLine to add a line ONLY
If the new line is not included in the Graphic
```



```
class Graphic2D {
  private lines: Line[] = [];

  contains(newLine: Line): boolean {
    // TODO :  return True if this line is already on the graphic, false otherwise
    return false;
  }

  addLine(newLine: Line) {
     // TODO :  return add the line ONLY if the new line is NOT on th graphic2D
  }
}
```

Immutable objects



Is it possible to change p1 values ?

```
class Point {
  private x: number;
  private y: number;

  constructor(x: number, y: number) {
    this.x = x;
    this.y = y;
  }

  public getX(): number {
    return this.x;
  }
  public getY(): number {
    return this.y;
  }
}

Let p1 = new Point(10,20);
```

NO

Is it possible to change p1 values ?

```
class Point {
  private x: number;
  private y: number;

  constructor(x: number, y: number) {
    this.x = x;
    this.y = y;
  }

  public getX(): number {
    return this.x;
  }
  public getY(): number {
    return this.y;
  }
}

Let p1 = new Point(10,20);
```

Class Point does not allow any change

↓

point are immutable

To change p1, we need to create new objects !!

Create a new point,
But translated

Clone the object
With the same values

```
class Point {

  constructor(private x: number, private y: number) {}

  public translateX(deltaX : number): number {
    return new Point(this.x + deltaX, this.y);
  }
  public clone(): number {
    return new Point(this.x, this.y);
  }
}

Let p1 = new Point(10,20);
Let p2 = p1.clone();
Let p3 = p1.transposeX(50);
```

ACTIVITY 4

```
//  2 – Create method to create new point translated on X ,Y
```

```
translate(x: number, y:number) : Point {
        // TODO create a new Point translated of (x, y)
}
```

```
//  2 – Create method to create new line translated on X ,Y
```

```
translate(x: number, y:number) : Line {
        // TODO create a new line translated of (x, y)
}
```

DEBATE

5 MIN CLASS

What is the benefit of Immutable objects ?

ID or not ID ?

Which class **should have an ID** attribute ?

A
```
class Address {
}
```

B
```
class Car {
}
```

C
```
class Date {
}
```

D
```
class Student {
}
```

E
```
class Point {
}
```

F
```
class Computer {
}
```

Which class **should have an ID** attribute ?

A
```
class Address {
}
```

[✓ B]
```
class Car {
}
```

C
```
class Date {
}
```

[✓ D]
```
class Student {
}
```

E
```
class Point {
}
```

[✓ F]
```
class Computer {
}
```

VALUE *vs* ENTITY objects

Objects **without proper identity** **do not need ID**

Example: Address, Date, Time, Point, Vector

```
class Point {
   x: number;
   y: number
}
```

Objects **which refer to something unique** **needs an ID**

Example: Student, School, Travel, Bus

```
class Student {
   id: number;
   name: string;
}
```

VALUE *vs* ENTITY objects

Objects **without proper identity** **do not need ID**

VALUE objects

Objects **which refer to something unique** **needs an ID**

ENTITY objects

VALUE *vs* ENTITY objects

Objects **without proper identity** **do not need ID**

```
public isEqual(other: point):
boolean {
    return
          this.x === other.x
       && this.y === other.y;
}
```

Objects **which refer to something unique** **needs an ID**

```
public isEqual(other: student): boolean
{
    return  this.id === other.id;
}
```

isEqual() with id/no id

```
public isEqual(other: student): boolean
{
    return  this.id === other.id;
}
```

```
public isEqual(other: point): boolean {
    return
            this.x === other.x
         && this.y === other.y;
}
```

3 types of Equality

REFERENCE
EQUALITY

ID
EQUALITY

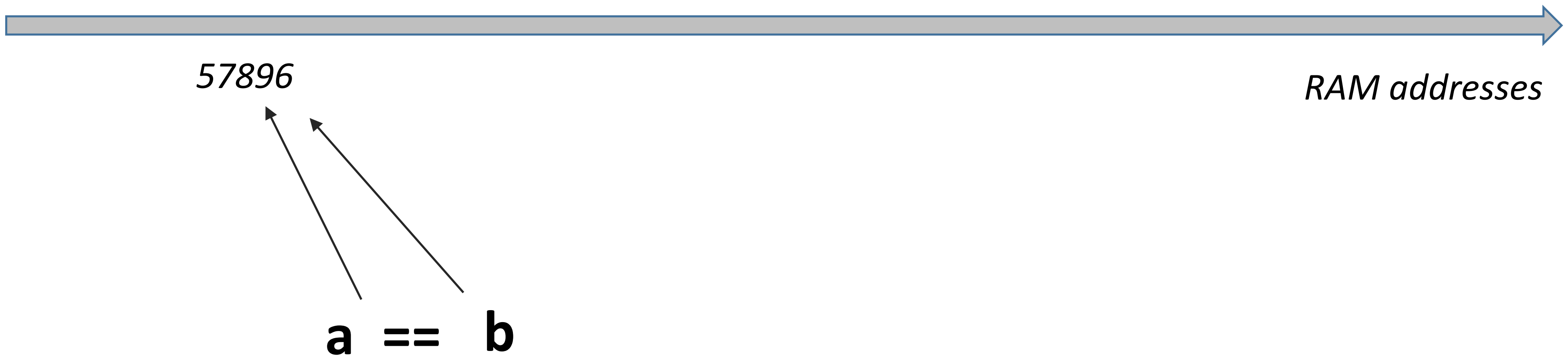
For ENTITY objects

VALUES
EQUALITY

For VALUE objects

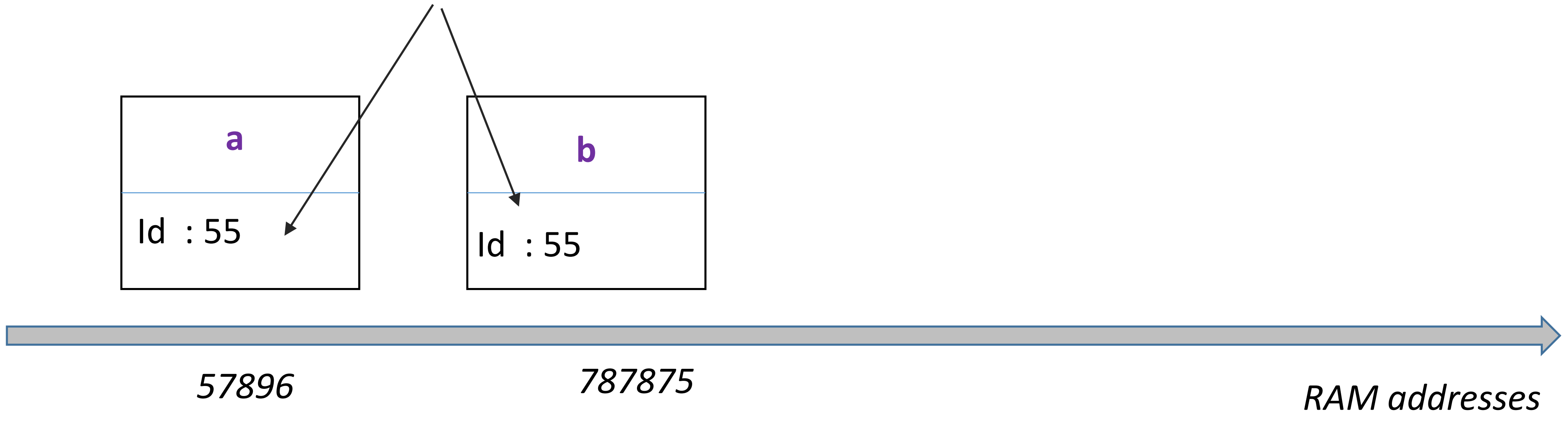
REFERENCE EQUALITY

Two objects are equal if they **reference** the **same address** in the memory



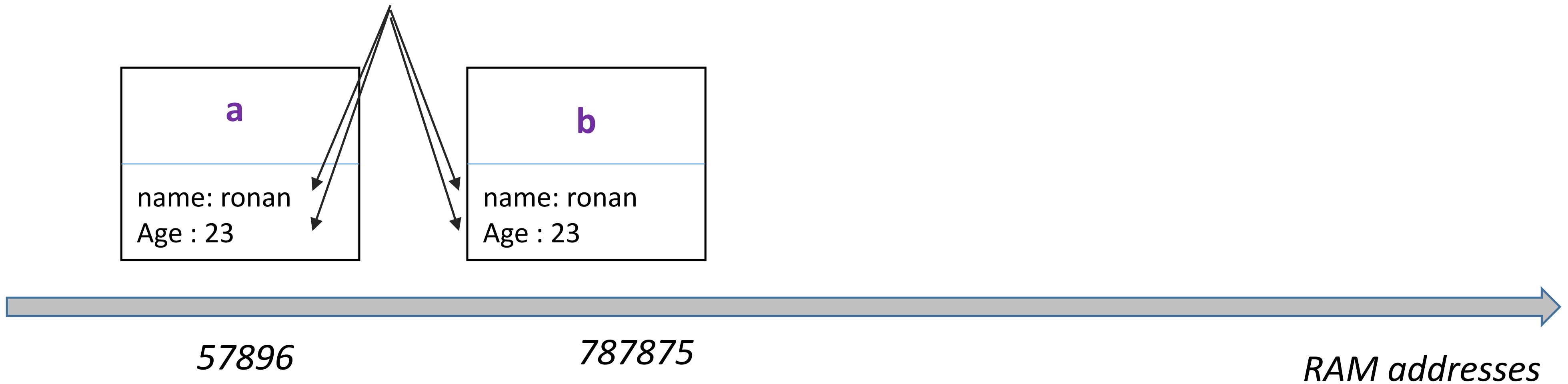
ID EQUALITY

Two objects are equal if they have the same ID



VALUES EQUALITY

Two objects are equal if all theirs **attributes are equal**


a
name: ronan
Age : 23
b
name: ronan
Age : 23
57896
787875
RAM addresses