OBJECT EQUALITY IN JAVA

A SIMPLE RECTANGLE CLASS

```java
public class MyRectangle {
    private int length;
    private int breadth;

    public MyRectangle(int length, int breadth) {
        this.setLength(length);
        this.setBreadth(breadth);
    }

    public int getLength() {
        return length;
    }

    public void setLength(int length) {
        this.length = length;
    }

    public int getBreadth() {
        return breadth;
    }

    public void setBreadth(int breadth) {
        this.breadth = breadth;
    }
}
```

TWO MEMBER VARIABLES FOR THE LENGTH AND BREADTH

A CONSTRUCTOR..

..AND GETTERS AND SETTERS

NOTHING TOO COMPLICATED.

NOW LET'S SAY WE INSTANTIATE TWO RECTANGLE OBJECTS WITH THE SAME LENGTH AND BREADTH

```
// two rectangle objects, with the same length and breadth
MyRectangle rectangle1 = new MyRectangle(5,10);
MyRectangle rectangle2 = new MyRectangle(5,10);

// are these two objects 'equal'
if(rectangle1 == rectangle2) {
    System.out.println("These two rectangles are 'equal'");
} else {
    System.out.println("These two rectanges are 'not equal'");
}
```

ARE THE TWO RECTANGLES EQUAL?

TO ANSWER THIS, WE NEED TO UNDERSTAND
THE IDEA OF OBJECT IDENTITY IN JAVA

REMEMBER THAT ALL CLASSES IMPLICITLY
DERIVE FROM THE OBJECT BASE CLASS

Class Object

java.lang.Object

public class **Object**

Class `Object` is the root of the class hierarchy. Every class has `Object` as a superclass. All objects, including arrays, implement the methods of this class.

Since:

JDK1.0

See Also:

`Class`

Constructor Summary

Constructors

Constructor and Description
`Object()`

Method Summary

Methods

Modifier and Type	Method and Description
`protected Object`	`clone()` Creates and returns a copy of this object.
`boolean`	`equals(Object obj)`

THE OBJECT BASE CLASS HAS A METHOD
CALLED ".EQUALS()" WHICH IS USED TO
TEST IF TWO OBJECTS ARE EQUAL

Constructor Summary

Constructors

Constructor and Description
`Object()`

Method Summary

Methods

Modifier and Type	Method and Description
protected `Object`	`clone()` Creates and returns a copy of this object.
boolean	`equals(Object obj)` Indicates whether some other object is "equal to" this one.
protected void	`finalize()` Called by the garbage collector on an object when garbage collection determi
`Class<?>`	`getClass()` Returns the runtime class of this `Object`.
int	`hashCode()` Returns a hash code value for the object.
void	`notify()` Wakes up a single thread that is waiting on this object's monitor.
void	`notifyAll()` Wakes up all threads that are waiting on this object's monitor.
`String`	`toString()` Returns a string representation of the object.
void	`wait()` Causes the current thread to wait until another thread invokes the `notify()` f

BY DEFAULT, THIS ".EQUALS" METHOD WILL ACTUALLY ONLY RETURN TRUE IF THE TWO OBJECTS ARE LITERALLY THE SAME OBJECT

("LITERALLY THE SAME" : OCCUPY THE SAME MEMORY IN THE COMPUTER)

ARE THE TWO RECTANGLES EQUAL?

TO ANSWER THIS, WE NEED TO UNDERSTAND THE IDEA OF OBJECT IDENTITY IN JAVA

THE ANSWER IS NO!

IF WE WOULD LIKE TWO RECTANGLES TO BE 'EQUAL' IF THEY HAVE THE SAME LENGTH AND BREADTH, WE WOULD NEED TO OVERRIDE THE EQUALS METHOD OF THE OBJECT BASE CLASS

```java
@Override
public boolean equals(Object object) {
    // is the other object even the same class as 'this'?
    if (object instanceof MyRectangle) {
        // if yes, cast it to be a rectangle
        MyRectangle otherRectangle = (MyRectangle) object;
        // if the length and breadth are the same as ours,
        // then return true (the two objects are equal)
        if(otherRectangle.breadth == this.breadth &&
                otherRectangle.length == this.length) {
            return true;
        }
    }
    // if not, return false (the two objects are not equal)
    return false;
}
```

NOW, WILL THE TWO OBJECTS BE CONSIDERED EQUAL?

STILL NO!

```java
// two rectangle objects, with the same length and breadth
MyRectangle rectangle1 = new MyRectangle(5,10);
MyRectangle rectangle2 = new MyRectangle(5,10);

// are these two objects 'equal'
if(rectangle1 == rectangle2) {
    System.out.println("These two rectangles are 'equal'");
} else {
    System.out.println("These two rectanges are 'not equal'");
}
```

THAT'S BECAUSE THE == OPERATOR ALWAYS GOES WITH REFERENCE EQUALITY!

TO GET SEMANTIC EQUALITY, EXPLICITLY USE THE .EQUALS() METHOD

```java
// are these two objects 'equal'
if(rectangle1.equals(rectangle2)) {
    System.out.println("These two rectangles are 'equal'");
} else {
    System.out.println("These two rectanges are 'not equal'");
}
```

NOW ARE THE TWO OBJECTS EQUAL?

FINALLY - YES!