Julio Pochet Edmead

Module 3 Discussion Board

**Java Methods: Pass-by-Value vs. Pass-by-Reference & Parameter Order**

One of the trickier parts of learning Java is understanding how it handles method arguments. Java uses **pass-by-value** for everything—but when it comes to objects, that value is actually a **copy of the reference**. So, if you pass an object to a method and change something inside that object, those changes stick. But if you try to point the object to something else inside the method, the original won’t change.

Here’s an example with a primitive:

public static void updateValue(int x) {

x = 99;

}

public static void main(String[] args) {

int num = 50;

updateValue(num);

System.out.println(num); // Prints 50

}

The value of num stays 50 because the method only got a copy.

Now, here’s a quick object example:

public static void updateArray(int[] arr) {

arr[0] = 42;

}

public static void main(String[] args) {

int[] numbers = {1, 2, 3};

updateArray(numbers);

System.out.println(numbers[0]); // Prints 42

}

That works because we're changing the data *inside* the object.

The second topic is **parameter order**. In Java, the order and type of method parameters matter a lot. You can’t mix them up.

Example:

public static void greet(String name, int age) {

System.out.println(name + " is " + age);

}

Calling greet("Tom", 30); is fine, but greet(30, "Tom"); will throw an error. Always match the order and data types.