**Julio Pochet – Module 6: Java Methods and Object-Oriented Concepts**

One of the biggest “aha” moments I had when learning Java was understanding the difference between **creating a class and an object**. A class is like a blueprint—it describes an object's data and behavior, but nothing happens until we actually create (instantiate) an object from that class.

Here’s a beginner-friendly example:

// This is the class blueprint

public class Car {

String color;

int year;

// Constructor: sets default values when a new Car is created

public Car(String color, int year) {

this.color = color;

this.year = year;

}

// Method: behavior of the Car

void drive() {

System.out.println("The " + color + " car from " + year + " is driving.");

}

}

// Somewhere else in your code (like in your main method), you’d create an object:

Car myCar = new Car("Red", 2022);

myCar.drive(); // Output: The Red car from 2022 is driving.

Now, let’s talk about **working with multiple classes**. This is super helpful for organizing programs. Each class should have a specific purpose; you can use them together like puzzle pieces. For example, you can have a Driver class that interacts with the Car class.

public class Driver {

String name;

// Constructor

public Driver(String name) {

this.name = name;

}

// Method that uses another class (Car)

void startCar(Car c) {

System.out.println(name + " is starting the " + c.color + " car.");

}

}

// Example use:

Driver driver = new Driver("Julio");

driver.startCar(myCar); // Output: Julio is starting the Red car.

Working with multiple classes like this is the foundation of Object-Oriented Programming. It makes your code more readable and maintainable, especially as your programs become complex. I'm starting to see how powerful OOP can be when used correctly.