📘 Chapter 2 – Important Notes

🔄 From Procedural to Object-Oriented

In this chapter, we begin transitioning from **procedural programming** to **object-oriented programming (OOP)**.

Key Difference:
In procedural programming, every change may require going back to the main working code, modifying it, then re-testing everything. This process becomes inefficient and risky as the system grows.

OOP solves this by letting you:

- Reuse code using **inheritance**.
- Avoid duplication.
- Customize behavior using **method overriding** without duplicating logic.

🧠 Thinking in Objects

Before writing a class, ask yourself:

- What does the object **know**? → These are called **attributes (instance variables)**.
- What does the object **do**? → These are called **methods (behaviors)**.

A class is **not** an object — it is a **blueprint** that tells the Java Virtual Machine (JVM) how to create objects of that type.

🧪 Testing Classes & the main() Method

To create and use an object in Java, you generally need two classes:

1. The **real class** – the one you're designing.
2. A **test class** – which contains the main() method, used to:
    - Create and test objects.
    - Start your Java application.

🔹 The Dot Operator (.)

Use the dot operator to access:

- An object’s **state** (instance variables).
- An object’s **behavior** (methods).

Example:

car.startEngine();

System.out.println(car.color);

🧠 Heap Memory & Garbage Collection

When an object is created in Java, it is stored in the **heap memory**.

- Java manages this memory automatically using **garbage collection**.
- When the JVM detects that an object is no longer in use, it frees that memory.
- This is one reason Java uses more memory than some languages, but it greatly improves safety and memory management.

🔐 Access Modifiers & Code Structure

- Java doesn't support global variables or functions outside classes.
- Instead, it uses **access modifiers** (like public) to control visibility.
- Everything in Java must be written inside a class.

📦 Packaging & Delivering Java Code

Worried about delivering a project with many classes?

- You can **archive** all your classes into one file (e.g., .jar).
- On the client side, you just need to specify which class has the main() method to run the application.

✅ **In Short**

- **OOP** allows you to build flexible, reusable systems.
- **Java code always lives inside classes.**
- A **class** is a blueprint. An **object** is a real entity created from that blueprint.
- **Instance variables** hold object state. **Methods** define behavior.
- Objects talk to each other at runtime. That’s the essence of Java programs.