Based on Gabriel Petersson's philosophy, **you should not start by studying multithreading basics, syntax nuances, or design patterns in isolation.** That is the "bottom-up" trap.
You stated your Java syntax is a **9/10**. This means you are already over-qualified to start building. Petersson argues you should not learn "prerequisites" for a potential solution; you should start with the problem and learn the prerequisites **only when they block you from solving it** 1.
Here is the **Top-Down Roadmap** to learn Advanced Java, DSA, and System Design starting from "nowhere," using Petersson's "Recursive Gap Filling" method.

Phase 1: Pick a "forcing function" (A Complex Project)

Do not watch a course. Pick a project that **forces** you to use multithreading and design patterns.

- **Recommendation:** Build a **Multi-threaded Web Server** or a **High-Frequency Stock Trading Simulation**.
- **Why?** You cannot build these without encountering concurrency, locking, and queue management (Advanced Java topics).

Phase 2: The Loop (How to study concepts Top-Down)

Here is how you tackle your specific weak points (rated 0-2/10) using this project:

1. Advanced Java (Multithreading & Collections)

Don't: Read a book on the Synchronized keyword.**Do:** Ask AI to write the core engine for your Stock Trading Simulation.

- **The Prompt:** "Write a Java class for a stock order book that handles thousands of buy/sell orders per second from different threads. Use modern concurrency utilities."
- **The Gap Fill:** The AI will likely use ConcurrentHashMap, ReentrantLock, or CompletableFuture. You won't know what these are.
- **The Study:** Stop. Ask the AI:
- "Why did you use ConcurrentHashMap here instead of a normal HashMap?" (You now learn about thread safety).
- "What would happen if I removed this ReentrantLock?" (You now learn about Race Conditions).
- "Explain the lifecycle of this thread like I'm 12."
- **Result:** You learn multithreading **contextually**, which helps it stick 2, 3.

2. Design Patterns & Principles

Don't: Memorize the Singleton or Factory pattern.**Do:** Write "bad" code first, then ask AI to refactor it.

- **The Action:** Implement a feature in your project (e.g., handling different types of stock orders: Market, Limit, Stop). Write it using a giant if-else block.
- **The Prompt:** "Here is my code. It looks messy. Refactor this using the best Design Patterns for this specific scenario and explain **why** you chose that pattern."
- **The Learning:** The AI might introduce the **Strategy Pattern** or **Factory Pattern**.
- **The Click:** Ask: "What is the benefit of the Factory pattern here compared to my if-else block?"

- *Petersson's Logic:* You are using the AI to simulate a senior engineer reviewing your code 4. You learn the pattern because you saw the problem it solves first.

3. System Design

Don't: Watch a generic "How to design Twitter" video.**Do:** Break your own project.

- **The Prompt:** "I have this Trading Engine running on my laptop. What happens if 1 million users try to submit orders at the exact same second? How does it crash?"
- **The Gap Fill:** The AI will introduce concepts like **Load Balancers**, **Message Queues (Kafka)**, and **Database Sharding**.
- **The Deep Dive:** Ask: "I don't know what Kafka is. Explain how it differs from a Java ArrayList or Queue."
- **Result:** You are learning System Design components (Kafka, Sharding) as solutions to a scalability problem you just simulated 5.

4. Algorithms (DSA)

Don't: Grind LeetCode blindly.**Do:** Ask AI to optimize your project's bottlenecks.

- **The Scenario:** Your order book search is slow.
- **The Prompt:** "My search for open orders is taking too long. How can I optimize this lookup?"
- **The Learning:** The AI might suggest a **Red-Black Tree** or a **Binary Search**.
- **The Study:** Ask: "Show me the step-by-step state of the Tree as we insert data. Explain the Big-O difference between this and my list."
- **Interview Prep:** When you *do* practice LeetCode, use the "Debug-to-Learn" loop. If you can't solve a problem, look at the solution and ask the AI to visualize the data structure step-by-step until the logic "clicks" 6.

Summary of the Petersson Approach

You are currently thinking: *Study $\rightarrow$ Practice $\rightarrow$ Build.*Gabriel Petersson is saying: *Build $\rightarrow$ Fail $\rightarrow$ Ask "Why?" $\rightarrow$ Study the specific answer $\rightarrow$ Build.*

Your Immediate Next Step: Open your IDE. Ask ChatGPT: *"I want to build a multi-threaded web scraper in Java from scratch without frameworks. Give me a high-level architecture and the first file I should write."*

When you see code you don't understand (e.g., ExecutorService), **that** is your curriculum for the day. Dig into it recursively until you understand it fully 7.