You're not alone — many learners feel exactly like you when so many Java backend technologies are taught back-to-back without a clear **evolution map** or purpose-wise comparison. Let me clear all your confusion step by step, like a **timeline + purpose guide**, so you understand what each technology does, why it exists, how it works internally, and how it connects with others.

**✅ 1. Servlets**

**📌 Purpose:**

* To handle HTTP requests and generate dynamic responses using Java.

**🔧 Internals:**

* Java class that extends HttpServlet.
* You override doGet(), doPost() etc.
* Lives inside a servlet container like **Tomcat**.

**📉 Limitations:**

* Writing HTML in Java (with out.println()) is messy.
* Not separation of concerns — logic and presentation mixed.

**✅ 2. JSP (JavaServer Pages)**

**📌 Purpose:**

* To write HTML pages with Java embedded for **dynamic content**.

**🔧 Internals:**

* JSP is compiled into a servlet by the server (Tomcat).
* Has built-in objects: request, response, session, application, etc.

**📉 Limitations:**

* Logic still leaks into UI.
* Hard to maintain large apps.

**✅ What Replaced It:**

* **Thymeleaf, React, Angular** for cleaner frontends.
* **Spring MVC** to separate logic and view.

**✅ 3. Hibernate (ORM Framework)**

**📌 Purpose:**

* To map Java objects to database tables (ORM).
* Avoids writing SQL manually using @Entity, @Id, etc.

**🔧 Concepts:**

* **Entity Class** → Maps to DB table.
* **SessionFactory** → Creates sessions.
* **Session** → Used to save/fetch/update objects.
* **Transaction** → To commit changes.
* **Query HQL** → Hibernate Query Language (object-based SQL).

**📉 Problems Hibernate Solves:**

* JDBC has a lot of boilerplate code.
* Managing connections and mapping is manual in JDBC.

**✅ 4. Maven**

**📌 Purpose:**

* Build and dependency management tool.
* Manages external libraries (JARs) using pom.xml.

**🔧 Concepts:**

* **POM.xml** → Declares dependencies, plugins, goals.
* **Maven Repository** → Where dependencies are downloaded from.

**📉 Problems it Solves:**

* No need to manually add JARs to classpath.
* Automates builds, tests, and packaging.

**✅ 5. Spring Framework (Core)**

**📌 Purpose:**

* General-purpose enterprise framework for **dependency injection** and application config.

**🔧 Core Concepts:**

* **IoC (Inversion of Control)** → Framework manages object lifecycle.
* **DI (Dependency Injection)** → Inject dependencies via @Autowired.
* **Bean** → A managed component.
* **ApplicationContext** → Loads and manages beans.

**📉 Problems It Solves:**

* Manual object creation and tight coupling in Java.

**✅ 6. Spring MVC**

**📌 Purpose:**

* Web framework that follows MVC (Model-View-Controller).
* Cleanly separates logic (Controller), data (Model), and UI (View).

**🔧 Key Parts:**

* @Controller → Handles requests.
* @RequestMapping or @GetMapping, @PostMapping.
* ModelAndView or Model → Pass data to view.
* ViewResolver → Maps logical view name to JSP or Thymeleaf.

**📉 Problems It Solves:**

* Manages servlet logic cleanly.
* Decouples layers.
* More readable and scalable than plain JSP/Servlet.

**✅ 7. Spring Boot**

**📌 Purpose:**

* Rapid application development with **auto-configuration**.
* Reduces boilerplate config of Spring.

**🔧 Key Features:**

* **@SpringBootApplication** → Enables component scanning + config.
* **Embedded Tomcat** → No need to deploy WARs to external server.
* **application.properties/yml** → Central config file.
* **Starter Dependencies** → Simplified POM (like spring-boot-starter-web).
* **Spring Initializr** → Project generator.

**📉 Problems It Solves:**

* No need for XML configuration.
* No need to manage servers manually.
* Reduces setup time for Spring projects.

**✅ Evolution Map (Visual):**

1. JDBC → lots of code → ➡ Hibernate (ORM)

2. Servlet → logic only → ➡ Spring MVC (better controller layer)

3. JSP → messy UI logic → ➡ Thymeleaf / React / Angular

4. Spring → needs config → ➡ Spring Boot (auto-config, fast dev)

5. Manual JARs → painful → ➡ Maven (dependency mgmt)

**✅ Summary Table**

| **Technology** | **Category** | **Solves What?** | **Replaced What?** |
| --- | --- | --- | --- |
| Servlet | Web - Controller | Handle HTTP logic | Replaced by Spring MVC |
| JSP | Web - View | Dynamic HTML | Replaced by Thymeleaf |
| Hibernate | ORM | DB mapping, reduce JDBC code | Improves JDBC |
| Maven | Build Tool | JARs, build mgmt | Manual jar handling |
| Spring Core | Framework | DI, Bean Management | Manual object creation |
| Spring MVC | Web Framework | MVC pattern, RESTful APIs | Servlets, JSP logic |
| Spring Boot | Dev Framework | Auto-config, embedded server | Manual Spring setup |

**✅ Real-World Stack (Modern Web App):**

* **Spring Boot** → backbone of the app
* **Spring MVC** → REST APIs or controller logic
* **Hibernate (JPA)** → database interaction
* **Thymeleaf** or **React** → frontend
* **Maven** → project and dependency management

Would you like me to send you a **simple full-stack diagram** showing how Spring Boot + MVC + Hibernate work together? I can also break each module into a mini project plan if you want to practice.