JAVA BACKEND ROADMAP

120 concepts to become a Java backend developer

8 areas, 15 concepts each. Every concept explained in one line, the order to study each area, and two practice questions to test yourself.

How to use it: one area at a time, tick each concept only when you can explain it out loud in under a minute, then answer the two practice questions without notes.

@chalkesahil · chalkesahil.ai

AREA 01 / 08

Core Java

Study order: Write small programs every day until the syntax is automatic.

01	**JDK, JRE and JVM**	What each one does.
02	**Data types**	Primitives vs wrapper classes.
03	**Strings**	Immutability and the string pool.
04	**StringBuilder**	Build strings in loops efficiently.
05	**Control flow**	if, switch and loops.
06	**Arrays**	Fixed-size lists and common operations.
07	**Methods**	Parameters, return types and overloading.
08	**static keyword**	Belongs to the class, not the object.
09	**final keyword**	Constants, and classes that can't be extended.
10	**Exceptions**	try, catch, finally and custom exceptions.
11	**Checked vs unchecked**	Which ones the compiler forces you to handle.
12	**Java memory**	Stack vs heap.
13	**Garbage collection**	How unused objects get cleaned up.
14	**Java 8 features**	Lambdas, streams and Optional.
15	**Records and var**	Newer Java features you'll see in code.

Practice

1. Why are Strings immutable in Java?
2. Reverse the words in a sentence using streams.

AREA 02 / 08

OOP

Study order: Interviews ask OOP in every round; explain each idea with your own example.

01	**Classes and objects**	The blueprint and the thing built from it.
02	**Constructors**	Set up an object when it's created.
03	**Encapsulation**	Private fields with getters and setters.
04	**Inheritance**	Reuse a parent class's behaviour.
05	**Polymorphism**	One method name, different behaviour.
06	**Overloading vs overriding**	Compile time vs run time.
07	**Abstraction**	Show what it does, hide how.
08	**Abstract classes**	Partly built classes.
09	**Interfaces**	Contracts a class promises to follow.
10	**Composition**	Has-a instead of is-a.
11	**equals and hashCode**	Why you override both together.
12	**SOLID principles**	Five rules for clean class design.
13	**Singleton pattern**	Exactly one instance.
14	**Factory pattern**	Create objects without new everywhere.
15	**Builder pattern**	Readable objects with many fields.

Practice

1. Abstract class vs interface: when do you use which?
2. Design classes for a parking lot.

AREA 03 / 08

Collections

Study order: Know the time complexity of every collection you use.

01	**Collection hierarchy**	List, Set, Queue and Map.
02	**ArrayList vs LinkedList**	Fast reads vs fast inserts.
03	**HashMap**	Key-value lookups in constant time.
04	**How HashMap works**	Hashing, buckets and collisions.
05	**HashSet**	Unique items backed by a HashMap.
06	**TreeMap and TreeSet**	Sorted keys.
07	**LinkedHashMap**	Remembers insertion order.
08	**PriorityQueue**	Always get the smallest item first.
09	**Deque**	Stack and queue in one.
10	**Comparable vs Comparator**	Natural order vs custom order.
11	**Iterators**	Loop and remove safely.
12	**Fail-fast iterators**	Why you get ConcurrentModificationException.
13	**Immutable collections**	List.of and friends.
14	**Generics**	Type-safe collections.
15	**Big O of collections**	Pick the right one for the job.

Practice

1. How does HashMap handle two keys with the same hash?
2. Find the first non-repeating character in a string.

AREA 04 / 08

Multithreading

Study order: Learn the basics well; most backend bugs at scale live here.

01	**Process vs thread**	Separate memory vs shared memory.
02	**Creating threads**	Thread class vs Runnable.
03	**Thread lifecycle**	New, runnable, waiting and terminated.
04	**Race conditions**	Two threads changing the same data.
05	**synchronized**	Let one thread in at a time.
06	**volatile**	Make changes visible across threads.
07	**Deadlock**	Two threads waiting on each other forever.
08	**wait and notify**	Threads signalling each other.
09	**ExecutorService**	Thread pools instead of raw threads.
10	**Callable and Future**	Threads that return a result.
11	**CompletableFuture**	Chain async tasks.
12	**Concurrent collections**	ConcurrentHashMap and friends.
13	**Locks**	ReentrantLock for finer control.
14	**Atomic variables**	Thread-safe counters without locks.
15	**Virtual threads**	Lightweight threads in Java 21.

Practice

1. Write code that causes a deadlock, then fix it.
2. Two threads print odd and even numbers in order.

AREA 05 / 08

SQL and JDBC

Study order: Every Java backend talks to a database; learn SQL properly.

01	**SELECT and WHERE**	Read the rows you need.
02	**Joins**	Combine tables.
03	**GROUP BY**	Totals per group.
04	**Subqueries**	Queries inside queries.
05	**Indexes**	Faster lookups.
06	**Normalisation**	Avoid duplicate data.
07	**Transactions and ACID**	Safe, all-or-nothing writes.
08	**Isolation levels**	What other transactions can see.
09	**JDBC basics**	Connect Java to a database.
10	**PreparedStatement**	Safe queries that block SQL injection.
11	**Connection pooling**	Reuse connections with HikariCP.
12	**JPA and Hibernate**	Map tables to Java classes.
13	**Entity relationships**	OneToMany, ManyToOne and ManyToMany.
14	**Lazy vs eager loading**	When related data gets fetched.
15	**N+1 problem**	The classic Hibernate slowdown.

Practice

1. Find the second highest salary in each department.
2. Your Hibernate page makes 200 queries. Why?

AREA 06 / 08

Spring Boot

Study order: Build one complete REST API; most Indian Java jobs are Spring Boot.

01	**Why Spring Boot**	Auto-configuration and fast setup.
02	**Dependency injection**	Spring creates and wires your objects.
03	**Beans and scopes**	Singleton vs prototype.
04	**Annotations**	@Component, @Service, @Repository.
05	**REST controllers**	@RestController and request mappings.
06	**Request handling**	Path variables, params and request bodies.
07	**Validation**	@Valid and constraint annotations.
08	**Exception handling**	@ControllerAdvice for clean errors.
09	**Spring Data JPA**	Repositories with almost no code.
10	**Profiles and config**	application.yml for dev and prod.
11	**Spring Security**	Login and protected routes.
12	**JWT auth**	Stateless login for APIs.
13	**Logging**	SLF4J and log levels.
14	**Actuator**	Health checks and metrics.
15	**Caching**	@Cacheable with Redis.

Practice

1. Build a REST API for a library with books and members.
2. What happens between a request arriving and your controller running?

AREA 07 / 08

Microservices

Study order: Understand why companies split apps before you split one yourself.

01	**Monolith vs microservices**	One app vs many small ones.
02	**Service boundaries**	Split by business area.
03	**REST between services**	Services calling each other.
04	**API gateway**	One front door for all services.
05	**Service discovery**	Find where a service is running.
06	**Config server**	Central config for all services.
07	**Circuit breaker**	Stop calling a service that's down.
08	**Retries and timeouts**	Fail fast instead of hanging.
09	**Kafka basics**	Services talking through events.
10	**Saga pattern**	Transactions across services.
11	**Docker**	Package each service.
12	**Kubernetes basics**	Run and scale containers.
13	**Distributed tracing**	Follow one request across services.
14	**Centralised logging**	All logs in one place.
15	**System design basics**	Load balancing, caching and scaling.

Practice

1. Order service calls payment service, and payment is down. What happens?
2. Design the backend for a food delivery app.

AREA 08 / 08

Testing and interviews

Study order: Service companies and product companies both test DSA and Spring basics.

01	**JUnit 5**	Write unit tests.
02	**Mockito**	Mock dependencies in tests.
03	**Integration tests**	@SpringBootTest with a real context.
04	**Testcontainers**	Real databases in tests.
05	**Maven or Gradle**	Build and manage dependencies.
06	**Git workflow**	Branches and pull requests.
07	**DSA: arrays and strings**	The most asked category.
08	**DSA: hashing**	HashMap tricks for fast lookups.
09	**DSA: linked lists and trees**	Classic interview staples.
10	**Java output questions**	Tricky snippets on strings and static.
11	**Low-level design**	Parking lot, elevator, BookMyShow.
12	**Project explanation**	Problem, design and your decisions.
13	**Resume**	Projects with numbers, not just tools.
14	**Mock interviews**	Practise answering out loud.
15	**AI tools**	Copilot writes boilerplate; you check every line.

Practice

1. Write a unit test for a service that calls a repository.
2. Explain your Spring Boot project in two minutes.