# **Junior's Roadmap**

# **Labels**

🔴 **(Advanced Junior)** – Requires deeper knowledge - Advanced Juniors/ Mid-level developers

🟢 **(Junior)** – Fundamentals, should be known by any junior developer

## **Core Java Fundamentals** 🟢

### **Object-Oriented Programming (OOP)**

* **Encapsulation** – Private fields, getters, setters
* **Inheritance** – Superclass, subclass, extends
* **Polymorphism** – Static (method **overloading**) vs Dynamic (method **overriding**)
* **Abstraction** – Interfaces, abstract classes, partial vs complete abstraction
* The Diamond Problem

### **Java Basics**

* Primitive types vs Wrapper classes
* Strings & String Pool (what is and how does it work)
* Immutable vs Mutable objects
* Static keyword (fields, methods, blocks)
* Access Modifiers
* ***final*** keyword (class, method, variable)
* Enums & Usage
* ***equals****()* and ***hashCode****()* contract
* Important Java 8 / 11 Features

## **Data Structures, Collections & Algorithms** 🟢

* What is a data structure?
* Array vs. ArrayList vs. LinkedList (how do they actually work, how they are implemented)
* Stack, Queue, PriorityQueue
* Set (HashSet, LinkedHashSet, TreeSet)
* Map (HashMap, LinkedHashMap, TreeMap)
* HashTable vs HashMap
* Hash function and Hashing techniques
* Collisions in HashMap & how Java resolves them
* ***Comparable*** vs ***Comparator***
* ***Iterable*** vs ***Iterator***
* Big O Notation – Alg. Complexity
* Sorting Algorithms vs/and Searching Algorithms

## **Java Concurrency (Multithreading)** 🔴

* Thread Lifecycle
* ***Thread*** vs***Runnable*** (which one is preferred to be used)
* ***synchronized*** keyword & Locks
* Executors & Thread Pools
* Deadlock, Livelock, Starvation
* Atomic Variables (AtomicInteger, AtomicLong)

## **Functional Programming in Java** 🟢

* Lambda Expressions
* Stream API (methods, usage, operations)
* Intermediate vs Terminal Operations in Stream API
* Functional Interfaces (***Function***, ***Predicate***, ***Consumer***, ***Supplier***)
* Optional Datatype & Avoiding ***NullPointerException***

## **Design Patterns** 🟢

Here I will list most used patterns, but you may know more than just these.

* Categories in design patterns (Creational, Structural, Behavioral)
* Singleton (Eager vs Lazy initialization)
* Factory Method Pattern
* Builder Pattern
* Strategy Pattern
* Observer Pattern

Useful link: <https://refactoring.guru/design-patterns>

## **Java Memory Model** 🔴

* Java Memory Model
* Heap vs Stack
* Garbage Collection (GC)
* Strong, Weak, Soft, and Phantom References
* OutOfMemoryError
* Heap Space vs Metaspace

## **Spring & Spring Boot** 🟢

### **Core Spring (IoC, DI, Beans)**

* Inversion of Control (IoC) & Dependency Injection (types of DI)
* IoC container, IoC Container Implementation in Spring
* ApplicationContext vs. BeanFactory
* Stereotype annotations (***@Component***, ***@Service***, ***@Repository***, ***@Controller***)
* Bean (declaration, usage)
* Bean Scopes (***singleton***, ***prototype***, ***request***, ***session***)
* ***@Autowired*** on Field vs. Constructor Injection

### **Spring Boot Basics**

* Spring Boot vs Spring Framework
* Spring Boot Starter Packages (***spring-boot-starter-web***, ***spring-boot-starter-data-jpa***)
* Embedded Web Servers (Tomcat, Jetty, Undertow)
* Auto-configuration & Properties (***application.properties*** vs ***application.yml***)
* Schedulers and Caching
* Layered Architecture (Presentation Layer, Service Layer, Data Access Layer)

### **Spring Boot Configuration**

* ***@Configuration*** and ***@Bean***
* ***@Value*** usage
* Externalized Configuration with ***application.properties***
* Spring Profiles (***@ActiveProfiles***, ***spring.profiles.active***)

## **Spring MVC & RESTful APIs**

### **MVC** 🟢

* Model-View-Controller Components
* Spring MVC Flow (from sending the HTTP request to returning the response)
* DispatcherServlet
* Model-Attribute
* Form Handling & Validation (***BindingResult***, ***@Valid***, ***.hasErrors()***)
* Session & Cookies
* Interceptors
* View Technology
* View Resolver
* Thymeleaf Integration

### **REST** 🟢

* What is REST?
* What makes an API RESTful? (REST Constraints)
* HTTP Methods
* RESTful Endpoint Design Best Practices
* HATEOAS
* API Versioning (***/v1/users*** vs. ***Accept-Version*** header)
* Status Codes (***200 OK, 201 Created, 400 Bad Request, 404 Not Found, 500 Internal Server Error***)

### **REST APIs with Spring** 🟢

* ***@RestController***, ***@RequestMapping***, ***@GetMapping***, ***@PostMapping***
* Path Variables (***@PathVariable***) vs. Request Params (***@RequestParam***)
* Request Body (***@RequestBody***)
* Response Handling (***ResponseEntity<T>***)
* Exception Handling with ***@ExceptionHandler*** and ***@ControllerAdvice***
* Swagger Documentation

## **Security** 🟢

* How does Spring Security work?
* Stateless vs Stateful
* ***SecurityFilterChain***
* Authentication vs Authorization
* Principle (what & how to retrieve it in methods)
* SecurityContext and Authentication object 🔴
* Authentication provider vs Authentication manager 🔴
* UserDetails vs Principle
* UserDetailsService + PasswordEncoder
* JWT - what & why (Header, Payload, Signature) (Stateless authentication) 🔴
* What is JWT Secret? 🔴
* How to implement JWT Authentication in Spring 🔴
* Security Filters (***OncePerRequestFilter***)
* Role-Based Access Control (***@PreAuthorize, @Secured***)

## **Database & JPA/Hibernate** 🟢

* ORM (Object-Relational Mapping)
* ORM Implementation - Hibernate
* JPA vs Hibernate
* What is Entity?
* ***@Entity***, ***@Table***, ***@Column***, ***@Id***, ***@GeneratedValue***
* Entity Relationships ***(@OneToOne***, ***@OneToMany***, ***@ManyToMany***)
* Cascade Types & Fetch Types
* ACID
* Transaction Management (***@Transactional***)
* Isolation levels and propagation in transactions 🔴
* Connection Pooling (HikariCP) 🔴
* Database indexes (what is and why do we need them)
* How to create column indexes in Hibernate

## **Aspect-Oriented Programing** 🔴

* AOP (what is and why do we need this)
* Spring AOP
* AOP Core concepts (Aspect, Advice, Pointcut)
* Proxy Mechanisms (JDK Dynamic Proxies vs CGLIB Proxies)
* Performance Impact of AOP
* ***@Before, @After, @Around*** and other useful annotations

## **Microservices & Cloud** 🔴

* Monolith vs Microservices Architecture
* Benefits of using Microservices
* Data consistency problem across microservices: SAGA Pattern - **THIS IS REALLY ADVANCED TOPIC FOR JUNIORS**
* Monolith -> Microservice migration strategies (Parallel runs, Strangler Pattern)
* What is API Gateway, how to implement one in Spring?
* Load balancing (what is and why do we need this?)
* Inter-Service Communication (***FeignClient***)
* Docker & Kubernetes Basics - images, containers, basic commands
* Datadog/Grafana tools (what is observability and why do we need this)
* CI/CD tools - just the basics (what is and why do we need this)
* Event Driven Architecture - Message Brokers, Message Queues, Publisher, Listener, Events, Kafka
* DLQ (dead letter queue)

## **Testing** 🟢

* JUnit & Assertions - *Unit Tests*
* Mockito & Mocking Dependencies
* Test Lifecycle (***@BeforeEach***, ***@AfterEach)***
* ***@WebMvcTest*** – *API Testing*
* ***MockMvc***
* ***@SpringBootTest*** – Full Context Testing (*Integration Testing*)
* H2 Database
* WireMock – Mocking External APIs 🔴
* TestContainers 🔴