📝 Java Foundations for Spring Boot Backend Development

Module 1: Collections Framework

> Spring Boot apps rely heavily on collections to manage data before persisting, transforming, or sending responses.

Topics:

1. Introduction to Collections
    - Collection hierarchy
    - Interfaces: `Collection`, `List`, `Set`, `Map`
2. ArrayList
    - Creating and using `ArrayList`
    - CRUD operations (`add`, `get`, `set`, `remove`)
    - Iteration (`for`, `for-each`, `Iterator`)
    - Common methods (`contains`, `indexOf`, `size`)
3. HashMap / LinkedHashMap / TreeMap
    - Key-Value storage
    - CRUD operations (`put`, `get`, `remove`, `containsKey`)
    - Iterating over keys, values, and entries
    - Differences (ordering, sorting, hashing)
4. HashSet / LinkedHashSet / TreeSet
    - Uniqueness in data
    - Use cases (e.g., deduplicating)

Exercises:

- Store and retrieve user objects in an `ArrayList`.
- Build a phone directory with `HashMap`.

Module 2: Java 8+ Functional Features

> Modern Spring Boot relies heavily on functional style programming.

Topics:

1. Lambda Expressions
    - Syntax: `(args) -> expression`

- Replacing anonymous inner classes
- Examples: `Comparator`, event listeners

2. Functional Interfaces
    - `Runnable`, `Callable`
    - Java's built-in: `Predicate`, `Function`, `Consumer`, `Supplier`, `BiFunction`
    - Custom functional interfaces with `@FunctionalInterface`
3. Method References
    - Static methods (`Class::method`)
    - Instance methods (`obj::method`)
    - Constructors (`Class::new`)

Exercises:

- Sort a list of users using lambdas.
- Filter even numbers with `Predicate`.
- Convert strings to uppercase using `Function`.

Module 3: Stream API

> Streams simplify bulk operations (filtering, mapping, reducing) — very common in backend data processing.

Topics:

1. Introduction
    - Stream vs Collection
    - Stream pipeline (source → intermediate ops → terminal ops)
2. Creating Streams
    - From collections (`list.stream()`)
    - From arrays (`Arrays.stream()`)
    - From values (`Stream.of()`)
3. Intermediate Operations
    - `filter`, `map`, `flatMap`
    - `sorted`, `distinct`, `limit`, `skip`, `peek`
4. Terminal Operations
    - `collect` (`toList`, `toSet`, `toMap`)
    - `forEach`

- `reduce`
- `count`, `min`, `max`, `anyMatch`, `allMatch`, `noneMatch`

5. Collectors
    - Grouping (`Collectors.groupingBy`)
    - Partitioning (`Collectors.partitioningBy`)
    - Summarizing (`Collectors.summarizingInt`)

Exercises:

- Filter users above age 18 and collect names.
- Count orders per customer using `groupingBy`.
- Convert list of strings to uppercase list.

Module 4: Utility Classes for Backend Work

> These are common helpers you'll encounter while building Spring Boot applications.

Topics:

1. Optional
    - Avoiding `NullPointerException`
    - Methods: `of`, `ofNullable`, `empty`, `isPresent`, `ifPresent`, `orElse`, `orElseGet`, `orElseThrow`
2. LocalDate / LocalDateTime / ZonedDateTime
    - Date/time handling
    - Formatting and parsing (`DateTimeFormatter`)
3. String & StringBuilder
    - Common methods (`substring`, `split`, `replace`, `join`)
    - String immutability vs `StringBuilder`
4. Wrapper Classes & Autoboxing
    - `Integer`, `Double`, `Boolean` etc.
    - Parsing (`Integer.parseInt`, `Double.valueOf`)
5. Records (Java 14+)
    - Immutable data carriers (often useful for DTOs)

Exercises:

- Use `Optional` to safely fetch data from a `Map`.
- Format today's date in `yyyy-MM-dd` format.
- Parse a CSV string into a list.

Module 5: Concurrency & Parallelism (Advanced, Optional for Backend)

Useful for async tasks in Spring Boot.

Topics:

1. Threads and Executors
   - `Thread`, `Runnable`, `Callable`
   - `ExecutorService`
2. CompletableFuture
   - `supplyAsync`, `thenApply`, `thenAccept`, `exceptionally`
3. Parallel Streams
   - `parallelStream()` usage & pitfalls

Exercises:

- Run two independent tasks using `CompletableFuture`.
- Sum numbers with `parallelStream`.

Module 6: Integration with Spring Boot

Apply all the above in Spring Boot context.

Topics:

1. Using Collections in Controllers & Services
   - Return `List<User>` in REST APIs
2. Streams for Data Transformation
   - Convert Entity → DTO
   - Aggregate results before sending response
3. Optional in Repositories

- Spring Data JPA returns `Optional<T>`
- Handling missing data

4. Lambdas in Configurations
    - EventListeners, Filters, Custom comparators
5. Date/Time in REST APIs
    - Formatting JSON with `@JsonFormat`

Exercises:

- Create a REST API that returns a filtered list of users (`age > 18`).
- Use `Optional` when fetching a user by ID from repository.
- Group orders by customer and return JSON.

✅ By the end, the learner will be confident in Java's core utility classes & functional programming, making Spring Boot programming much smoother.
