# Java Core (OOP) — Learning Path & Lesson Summary

Generated: 2025-10-28 02:26:46

## Overview

This document summarizes the hands-on lessons we have completed so far while learning Java Core (with a focus on OOP principles and how these map to Spring Boot). It is written for a developer with prior JavaScript/functional-programming knowledge who wants to understand Java’s strict typing, encapsulation, and service-style design. Each lesson includes the goal, the key code patterns used, and practical notes.

## Lesson 1 — Java Strict Typing & First Method

Goal: Create a simple method that accepts a String and prints a Todo message.

Key points:

- Java is strictly typed: parameter and return types must be declared.

- Public class filename must match file name when class is public.

- Every statement ends with a semicolon.

Why it matters for Spring: Spring treats classes as types (DTOs), so understanding types early is crucial.

## Lesson 2 — Classes as Types & Encapsulation

Goal: Replace raw strings with a custom Task class to show that classes are types.

Key points:

- Creating a Task class (fields + methods) makes a new type you can use in method signatures.

- Encapsulation: make fields private and expose getters/setters to control access.

- Semantics: 'markAsDone()' expresses behaviour; avoid exposing fields publicly.

Why it matters for Spring: DTOs and Entities are defined with private fields and getters/setters; validation and frameworks expect this structure.

## Static vs Instance Methods

Goal: Understand 'static' (class-level) methods vs instance (object-level) methods.

Key points:

- static methods belong to the class and can be called without creating an instance.

- instance methods require an object; they operate on that object's internal state.

Why it matters for Spring: Controllers/Services are instantiated objects (non-static) managed by the IoC container.

## Collections & Manager Class (TodoList)

Goal: Manage multiple Task objects with a TodoList class using List<T>.

Key points:

- Use List<Task> tasks = new ArrayList<>(); for typed collections.

- Use addTask(), showAll() methods; loop with enhanced for-loop.

- Use equals() for string comparison, not ==.

Why it matters for Spring: Services manage collections of entities; type safety prevents runtime surprises.

## Validation Flow (Spring Boot concept)

Goal: Explain how @Valid and Bean Validation (Jakarta Validation) work conceptually.

Key points:

- Spring populates DTOs (via setters) from JSON, then validates annotated constraints before controller runs.

- If validation fails, controller is not executed and a 400 Bad Request is returned.

Why it matters for Spring: Validation is pre-controller middleware that ensures clean inputs and safer business logic.

## Returning Objects from Methods

Goal: Upgrade manager to return objects (findBookByTitle), not just print.

Key points:

- Methods can return typed objects (e.g., Book) enabling the caller to decide next steps.

- Always null-check returned objects before accessing properties to avoid NullPointerException.

Why it matters for Spring: Services return domain objects which controllers transform into responses.

## Hands-on Exercises Completed

- Basic addTodo(String) function and debugging Java filename/semicolons.  
- Created Task class with private fields + getters/setters and methods (markAsDone, markAsNotDone).  
- Built TodoList manager (addTask, showAllTasks, findTaskAndMarkDone).  
- Built Library/Book example with add, borrow, find, and filter borrowed books.  
- Implemented method that returns an object and practiced null-checking.

## Common Java Pitfalls (and how to avoid them)

- Using '==' to compare strings: use .equals() instead.  
- Accessing properties on null objects → NullPointerException; always null-check before accessing.  
- Public fields: allow uncontrolled mutation; use private + getter/setter and validation.  
- Running the wrong file: Java runs the class with the main method. Ensure you execute Main.java.

## Cheat Sheet (Quick Reminders)

- Use private fields, public getters/setters.  
- Use List<T> with generics for collections.  
- Use enhanced for-loops to iterate.  
- Use break when you want to stop after the first match.  
- Validate inputs early (Spring @Valid in controllers).

## Next Suggested Lessons

1. Constructors & Immutability: learn how to create constructors, use final fields, and make objects immutable when appropriate.  
2. Method Overloading & Overriding: learn how to create multiple methods with same name but different signatures and how polymorphism works.  
3. Packages & Imports: organize classes into packages and learn import rules to prepare for Spring Boot structure.  
4. Translate to Spring Boot: annotate classes (@RestController, @Service, @Repository), wire via dependency injection, and build endpoints for your Library.

## Appendix — Useful Java Notes

- String equality: use Objects.equals(a, b) to be null-safe.  
- Prefer List interface on types, e.g., List<Task> tasks = new ArrayList<>();  
- Use System.out.println for quick debugging; switch to logging framework for production code.