

Advanced Java: Design Patterns and Event-Driven Integration

Refine Java design and apply selected concepts in a Kafka lab

FA-0717 · 3 days

Software Development

Introduction

Deepen existing Java skills through design-pattern decisions, robust class and interface design, and a small event-driven integration exercise. The course assumes working Java and object-oriented experience.

Three days provide selected implementation practice and a survey of specialist runtime APIs, not mastery of every advanced Java feature or production Kafka operation. Use an explicitly chosen supported JDK and compatible dependencies.

Learning outcomes

- Choose and implement selected behavioural and creational patterns and explain their trade-offs.
- Represent a design with simple UML and relate it to event-driven, MVC or microservice architecture.
- Improve class, interface, generic and exception-handling designs in a Java example.
- Recognise concurrency, deserialisation and reflection risks and assess suitable alternatives.
- Use Java Kafka clients to produce and consume sample events and explain partition and offset behaviour.
- Review specialist APIs and identify which require deeper version-specific study.

Prerequisites

- Good understanding of OOP.
- Ability to program in Java.
- Ability to use IDE for Java.
- Compile and Deploy software in Java.
- Understand the entire development lifecycle in Java.

Comfort with file management and basic command-line use.

A supported JDK, suitable IDE/build tools and access to the supplied isolated Kafka lab. Only the required lab permissions and network endpoints are needed; unrestricted root access or open ports on a live system are not required.

Training outline

1. Day 1 — Patterns and design decisions

- Revisit object construction and use a small UML model to communicate responsibilities and dependencies.
- Behavioural patterns: chain of responsibility, command and iterator; select a suitable example rather than applying patterns mechanically.
- Creational patterns: abstract factory, builder and singleton, including testability and concurrency trade-offs.
- Compare event-driven, microservice and MVC architectural patterns and their effects on the example application.

2. Day 2 — Robust Java implementation

- Recursion, constructors, static initialisation and object methods; design classes and interfaces with clear contracts.
- Apply generics, enums and annotations; improve method design and follow consistent coding practices.
- Handle specific exceptions and preserve meaningful failure information.
- Review concurrency correctness and selected coordination techniques; distinguish thread safety from assumed speed gains.
- Built-in serialization and deserialization filtering; inspect risks and avoid treating arbitrary incoming object streams as safe.
- Survey reflection, dynamic-language integration, the compiler API and Java agents with version-specific constraints; selected demonstrations do not constitute deep coverage of every API.

3. Day 3 – Kafka integration and review

- Start or access the supplied Kafka environment and inspect a topic; use compatible Java clients and serializers.
- Produce sample records, handle failures and compare behaviour with multiple producers.
- Read records using a consumer, inspect partition assignment and offset commits; auto-commit is a consumer concern.
- Refactor the client example using relevant design concepts and test its expected behaviour.
- Distinguish Kafka producer/consumer APIs from the separate Kafka Streams processing library.
- Assess the small application, document unresolved trade-offs and identify next steps.