AMBIENT-IT

+33 (0)1 48 10 09 46
formation@ambient-it.net
140 Avenue Jean Lolive 93500 Pantin - France

Updated on 07/30/2026

Sign Up

Spring Batch Training

2 days (14 hours)

Overview

Spring Batch is the official framework of the Spring ecosystem, designed to execute and automate large-scale, high-performance, and reliable batch processing within enterprise applications.

One of the cornerstones of Spring Batch is its native support for robustness and state management: it provides a structured, chunk-oriented model (read, process, write) combined with a persistent metadata system. Thanks to this architecture, the framework ensures automatic recovery from failures, strict transaction management, and complete traceability of executions without re-running steps that have already been validated.

Our Spring Batch training covers both the fundamental concepts of batch processing (Jobs, Steps, Tasklets, JobParameters) and the framework's advanced features, such as fault tolerance (skip and retry), dynamic parameter passing (@StepScope), flow routing, and optimization strategies through parallelism (multi-threading and partitioning).

By the end of this training, you will be able to design, secure, and test complex, production-ready batch jobs. You will know how to isolate your test components, handle corrupted files or network errors without interrupting your processing, and orchestrate high-volume workflows optimized for stringent performance constraints.

Like all our training courses, this one will introduce you to **the latest stable version** of the technology and its new features.

Objectives

- Understand Spring Batch's role within an information system and master its terminology (Job, Step, JobInstance, JobExecution, StepExecution, JobParameters, ExecutionContext).

- Set up a Spring Boot project with Spring Batch and launch a job via JobOperator.
- Write a TaskletStep for a single action, and a chunk-oriented Step (ItemReader / ItemProcessor / ItemWriter) to handle high volumes.
- Read and write common formats: flat files (CSV), databases (JPA), and JSON.
- Use the metadata persisted by Spring Batch (BATCH_* tables) to monitor executions and understand restart mechanisms.
- Configure a job and pass these settings down to the chunk components (late binding, @StepScope), and validate the parameters before startup.
- Make a process fault-tolerant: skip, retry, rejection thresholds, traceability of rejected rows via a SkipListener.
- Control the execution flow of a job: distinguish between BatchStatus and ExitStatus, build a (.on/.to/.from, terminations, wildcards, JobExecutionDecider).
- Speed up processing by choosing the right form of parallelism: split() of flows, multi-threaded steps (taskExecutor), partitioning—and identify the pitfalls (shared state, thread safety, recovery).
- Reliably testing a batch: components in isolation, isolated steps, complete job.
- Design a complete production batch from start to finish (final capstone project).

Target Audience

- Java/Kotlin developers, back-end developers, software architects
- Technical leads responsible for writing or maintaining bulk processing tasks (file imports/exports, database populations, data recovery, nightly processing)

Prerequisites

- Practical knowledge of the Spring/Spring Boot framework (dependency injection, starters, configuration via @Bean, application.properties).
- Proficiency in Java or Kotlin.
- Basic understanding of SQL and data persistence (JPA/Hibernate) for hands-on database loading tasks.
- No prior knowledge of Spring Batch is required.

Software Prerequisites

- IntelliJ IDEA installed on your workstation

Spring Batch Training Course Outline

[Day 1 - Morning]

What is a batch application?

- Use cases for batch processing: large-scale imports/exports, populating databases, data recovery, overnight processing, and customer follow-ups.
- What a batch framework offers compared to a simple script: recovery after an incident, transactions, monitoring, traceability.
- Spring Batch's position within the Spring ecosystem.

Architecture and terminology

- Job and Step: the reusable processing plan and its stages.
- The two types of Steps: TaskletStep (single-action) and chunk-oriented Step (batch read/process/write).
- The execution model: JobInstance (logical execution identified by the job name and its identification parameters), JobExecution (an execution attempt), StepExecution.
- JobParameters: their role in an instance's identity.
- ExecutionContext: the persistent map that enables resumption.
- Infrastructure components: JobRepository, JobOperator.

Setting up the project

- Required dependencies and starters.
- Configuration: metadata database, schema initialization, disabling automatic startup.

The TaskletStep

- The Tasklet interface and its execute() method.
- The RepeatStatus contract: FINISHED / CONTINUABLE, and the transaction per call.
- Use cases: file deletion, web service call, stored procedure.

[Day 1 - Afternoon]

Assembling a Step and a Job, then running it

- Building a Step with StepBuilder (name, JobRepository, transaction manager).
- Building a Job with JobBuilder and chaining steps.
- Launching with JobOperator; retrieving jobs via dependency injection.
- Practical Exercise 1 — HelloWorld
  - Setting up the project, writing your first Tasklet, assembling the Step and Job, running the job, and reading the result.

Job Parameters

- Creating and typing JobParameters at launch.

- Identifying and non-identifying parameters: direct impact on the identity of the JobInstance and, therefore, on restarting the job.
- First way to read a parameter: from the execution context of a Tasklet.
- Practical Exercise 2 — Parameters
  - Pass a parameter to the job, process it in the Tasklet, and intentionally cause a failure to observe a FAILED status.

The metadata tables (BATCH_*)

- The Persistent Model
- How a job's identity is calculated (name + parameters, summarized in JOB_KEY).
- Why two runs with the same parameters target the same instance, and the role of a technical parameter (timestamp) in making each call unique.
- Reading the processing counters carried by each StepExecution.

Chaining multiple steps and handling retries

- Sequential chaining and failure propagation: subsequent steps are not launched.
- Behavior on restart: a COMPLETED step is skipped.
- allowStartIfComplete: Force a step to be rerun (cleanup, preparation).
- startLimit: Limit the number of attempts.
- Practical Exercise 3 — Multiple Steps
  - Create a job with multiple steps, trigger a failure, restart, and observe the resumption at the failed step; force the rerun of a step that has already completed.

Chunk-based processing

- The principle: N items are read and processed one by one, then written in a single transaction.
- The chunk as a commit interval, and its impact on performance and memory.
- What Spring Batch supports natively: short transactions, streaming, saving the position for resumption, read/write/commit counters.
- ItemReader: flat file (CSV), JDBC database, JPA database, JSON/XML, custom reader, composite reader.
- ItemProcessor: transformation, filtering (returns null), validation, composite processor.
- ItemWriter: flat file, JDBC database, JPA database, JSON/XML, composite writer.
- Assembling a chunk-oriented Step with typing of input and output items.
- TP 4 — Chunk
  - Import a CSV sales file into a database: write the read DTO, the mapping to the entity, and the business logic; observe the effect of chunk size on commits.
- Practical Exercise 5 — Reverse Process
  - Export the table to a sorted CSV file, with a header row and a total row (paged read from the database, aggregation, file write).

[Day 2 - Morning]

Passing Parameters Through All Layers

- Late binding: read a JobParameter using a SpEL expression in a Reader, Processor, or Writer.
- @StepScope and @JobScope: why they're essential, and what breaks without them.
- The return type trap: declare an interface rather than a concrete class so that the step recognizes the component as a stream to be opened and closed.
- Validating parameters before startup: JobParametersValidator and DefaultJobParametersValidator (required/optional parameters), and their limitations.
- Listeners: JobExecutionListener (beforeJob / afterJob) and StepExecutionListener (beforeStep / afterStep).
- Practical Exercise 6 — Parameterized Chunk
  - Control the same job using four parameters, each read from a different point in the pipeline: the source file (reader), a filtering threshold and a calculation rule (processor), and the CSV or JSON output format (writer).
  - Implementation of a parameter validator.

Fault tolerance: skip and retry

- Default behavior: an exception causes the step to fail, and the consequences for bulk processing.
- Enable tolerant mode, with the intervention order set to retry followed by skip.
  - retry: retry a temporary error (unavailability, timeout) and limit the number of attempts.
  - Skip: Ignore a faulty row and continue; the rejection threshold (skipLimit) acts as a quality safeguard—if exceeded, the file is invalid and the job must fail.
- Carefully define the relevant exceptions (skip / noSkip).
- The special case of the writer, which receives a batch and processes it line by line.
- SkipListener: Log rejected lines to a rejection file that can be used by production.
- Reading counters after an incident (read, write, commit, rollback, skip).
- Practical Exercise 7 — Corrupted File
  - Make an import tolerant of three types of errors (unreadable line, violated business rule, temporary failure), set a rejection threshold, generate a rejection file, then compare an acceptable file with a file to be rejected.

BatchStatus and ExitStatus

- BatchStatus: the internal status that controls recovery and transitions (COMPLETED, FAILED, COMPLETED WITH SKIPS...).
- ExitStatus: the customizable exit code, intended for reporting and routing.
- Generate a business-specific ExitStatus from afterStep (interface or annotation).
- Propagation of the ExitStatus from the last step to the job.
- Practical Exercise 8 — ExitStatus
  - Generate three different business exit codes from the same step based on a flag, and verify that the step remains successful while carrying a nuanced status.

Conditional job flow

- The limitations of linear sequencing.
- The .on(...).to(...) and .from(...) transitions: branching based on the ExitStatus.
- Terminations: .end() (early successful termination), .fail() (explicit failure), .stopAndRestart(step) (pause and resume point).

- Handling a managed failure: route to .on("FAILED") and end the job successfully.
- Wildcards ? and * and the “most specific pattern” rule.
- Nested flows to build complex sequences.
- JobExecutionDecider: route based on a calculated decision rather than an ExitStatus.

[Day 2 - Afternoon] Parallelism

(1/3)

- Execute multiple independent flows in parallel and merge them before proceeding.
- The role of the TaskExecutor, which provides the threads.
- Chain multiple successive split() operations.
- Practical Exercise 9 — Flow
  - Build about ten routing jobs of increasing complexity: sequential, two branches, early termination, explicit failure, failure handling, wildcards, pause and resume, nested flows, reproducing a sequence based on a diagram and then based on written specifications, decision-making, and parallelization using split().

Parallelism (2/3): The Multi-Threaded Step

- Parallelize the processing of items within the same step using a TaskExecutor.
- What remains single-threaded and what becomes concurrent: practical implications for the reader, the processor, and the writer.
- The real point of concern: the shared state of a processor, and its correction through thread-safe structures.
- Sizing the execution pool: size, queue, and saturation policy.
- Lab 10 — Parallelized Chunk
  - Parallelize a slow import operation, measure the performance gain, identify the concurrency bug introduced by a shared counter, and then fix it.

Parallelism (3/3): Partitioning

- The master/slave principle: divide the data into partitions processed in parallel, each with its own reader.
- Write a Partitioner and populate the execution context for each partition.
- The slave step and the late binding of its data slice.
- Configure the master step: partitioner, grid size, executor.
- The major advantage: partition-by-partition recovery after a failure.
- Choose between split(), multi-threaded steps, and partitioning.
- Practical Exercise 11 — Partitioning
  - Import ten files in parallel (one partition per file), verify partition isolation and time savings, cause a single partition to fail, and observe that only that partition is replayed upon restart.

Test a batch

- Test a component in isolation, without the Spring context (reader, processor).
- Test a component that requires the Spring context (writer).
- Test a step-scoped component that reads a job parameter: create a StepExecution context to validate late binding resolution.
- Isolate the execution of a single step, using a real StepExecution and real transactional behavior.
- Test a complete job and assert on statuses, counters, and the actual path taken rather than on the console.
- Neutralize a batch's external dependencies in tests.
- TP 12 — Testing a Batch (Reverse Exercise)
  - The job is provided and functional: you need to write the test class. Validate a reader in isolation (correct header, header in the wrong order, invalid number of columns), then the complete file merge job without duplicates.

Capstone Project

- Analysis of a complete business requirement and breakdown into steps.
- Reasoned choices: Tasklet or chunk, tolerance threshold, form of parallelism, isolation of external calls.
- FINAL PROJECT PRACTICAL — Truck Scheduling Batch
  - Implementation of a complete production batch: checking the presence and validity of the order file, error-tolerant import with a rejection threshold and a rejection file, enrichment by an external service (adjusting quantities based on the weather in each city), generation of loading files for the warehouse and for each driver, followed by archiving and purging. This project applies all the concepts covered in the course.

Further Reading

Target Companies

This training is intended for both individuals and companies—large or small—that wish to train their teams in new, advanced IT technology or to acquire specific industry knowledge or modern methods.

Entry-Level Assessment

The pre-training assessment complies with Qualiopi quality standards. Upon final registration, the learner receives a self-assessment questionnaire that allows us to evaluate their estimated proficiency with various types of technologies, as well as their expectations and personal goals for the upcoming training, within the limits imposed by the selected format. This questionnaire also allows us to anticipate certain connection or internal security issues within the company (intra-company or virtual classroom) that could pose challenges for monitoring and ensuring the smooth running of the training session.

Teaching Methods

Hands-On Training: 60% Practical, 40% Theoretical. Training materials distributed in digital format to all participants.

Organization

The course alternates between theoretical input from the instructor—supported by examples and reflection sessions—and group work.

Assessment

At the end of the session, a multiple-choice quiz is administered to verify that participants have successfully acquired the skills.

Certification

A certificate will be issued to each trainee who has completed the entire training program.