**Spring Batch:**

Batch processing—typified by bulk-oriented, non-interactive, and frequently long running, background execution—is widely used across virtually every industry and is applied to a diverse array of tasks. Batch processing may be data or computationally intensive, execute sequentially or in parallel, and may be initiated through various invocation models, including ad hoc, scheduled, and on-demand.

This Spring Batch tutorial explains the programming model and the domain language of batch applications in general and, in particular, shows some useful approaches to the design and development of batch applications using the current [**Spring Batch 3.0.7**](http://projects.spring.io/spring-batch/) version.

What is Spring Batch?

Spring Batch is a lightweight, comprehensive framework designed to facilitate development of robust batch applications. It also provides more advanced technical services and features that support extremely high volume and high performance batch jobs through its optimization and partitioning techniques. Spring Batch builds upon the [POJO-based](https://spring.io/understanding/POJO) development approach of the [Spring Framework](https://www.toptal.com/spring/beginners-guide-to-mvc-with-spring-framework), familiar to all [experienced Spring developers](https://www.toptal.com/spring).

By way of example, this article considers source code from a sample project that loads an XML-formatted customer file, filters customers by various attributes, and outputs the filtered entries to a text file. The source code for our Spring Batch example (which makes use of [Lombok](https://projectlombok.org/) annotations) is available [here](https://github.com/asaen/spring-batch-article) on GitHub and requires Java SE 8 and Maven.

Spring Batch is an amazing tool for efficiently processing large amounts of data.

A Spring Batch project uses a Job, JobLauncher, Step, and JobRepository. A Job is a container for Steps.Each Step may contain a Tasklet, which is nothing more than an object with custom logic for a Job. Each Job is started by a JobLauncher and resides within a JobRepository. Each JobRepository requires a data source. If a JobLauncher is not specified, a default object is instantiated.

**JSTLR==🡺 Job🡪Steps->Tasklet->JobLauncher->JobRepository**.

**What is Batch Processing? Key Concepts and Terminology**

It is important for any batch developer to be familiar and comfortable with the main concepts of batch processing. The diagram below is a simplified version of the batch reference architecture that has been proven through decades of implementations on many different platforms. It introduces the key concepts and terms relevant to batch processing, as used by Spring Batch.

As shown in our batch processing example, a batch process is typically encapsulated by a Job consisting of multiple Steps. Each Step typically has a single ItemReader, ItemProcessor, and ItemWriter. A Job is executed by a JobLauncher, and metadata about configured and executed jobs is stored in a JobRepository.

Each Job may be associated with multiple JobInstances, each of which is defined uniquely by its particular JobParameters that are used to start a batch job. Each run of a JobInstance is referred to as a JobExecution. Each JobExecution typically tracks what happened during a run, such as current and exit statuses, start and end times, etc.

A Step is an independent, specific phase of a batch Job, such that every Job is composed of one or more Steps. Similar to a Job, a Step has an individual StepExecution that represents a single attempt to execute a Step. StepExecution stores the information about current and exit statuses, start and end times, and so on, as well as references to its corresponding Step and JobExecution instances.

An ExecutionContext is a set of key-value pairs containing information that is scoped to either StepExecution or JobExecution. Spring Batch persists the ExecutionContext, which helps in cases where you want to restart a batch run (e.g., when a fatal error has occurred, etc.). All that is needed is to put any object to be shared between steps into the context and the framework will take care of the rest. After restart, the values from the prior ExecutionContext are restored from the database and applied.

JobRepository is the mechanism in Spring Batch that makes all this persistence possible. It provides CRUD operations for JobLauncher, Job, and Step instantiations. Once a Job is launched, a JobExecution is obtained from the repository and, during the course of execution, StepExecution and JobExecution instances are persisted to the repository.

There are two main approaches to building a step.

One approach, as shown in the above example, is **tasklet-based**. A Tasklet supports a simple interface that has only one method, execute(), which is called repeatedly until it either returns RepeatStatus.FINISHED or throws an exception to signal a failure. Each call to the Tasklet is wrapped in a transaction.

Another approach, **chunk-oriented processing**, refers to reading the data sequentially and creating “chunks” that will be written out within a transaction boundary. Each individual item is read in from an ItemReader, handed to an ItemProcessor, and aggregated. Once the number of items read equals the commit interval, the entire chunk is written out via the ItemWriter, and then the transaction is committed. A chunk-oriented step can be configured as follows:

