Spring Batch

What is Spring Batch

Spring Batch is a lightweight, comprehensive batch framework designed to enable the development of robust batch applications vital for the daily operations of enterprise systems

A Typical Batch Job

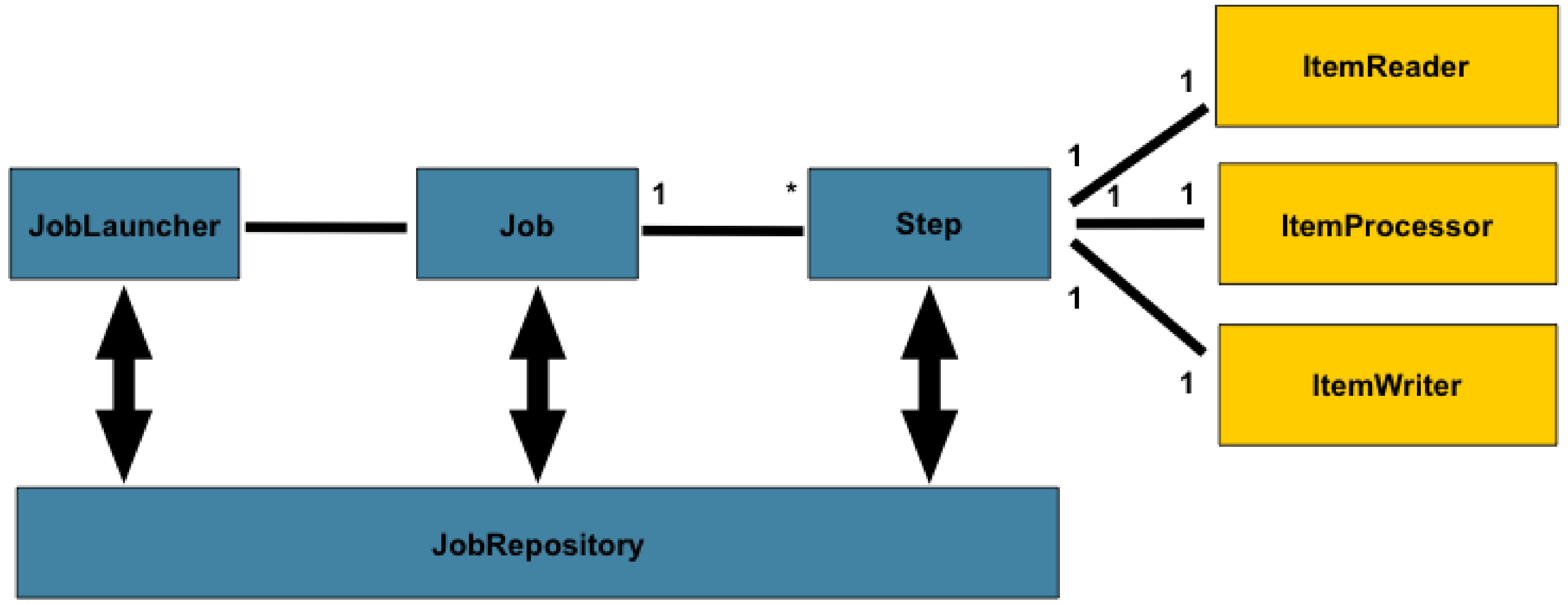
- A typical batch program generally:
  - Reads a large number of records from a database, file, or queue.
  - Processes the data in some fashion.
  - Writes back data in a modified form.

- Spring Batch automates this basic batch iteration, providing the capability to process similar transactions as a set, typically in an offline environment without any user interaction.

- Batch jobs are part of most IT projects, and Spring Batch is the only open source framework that provides a robust, enterprise-scale solution

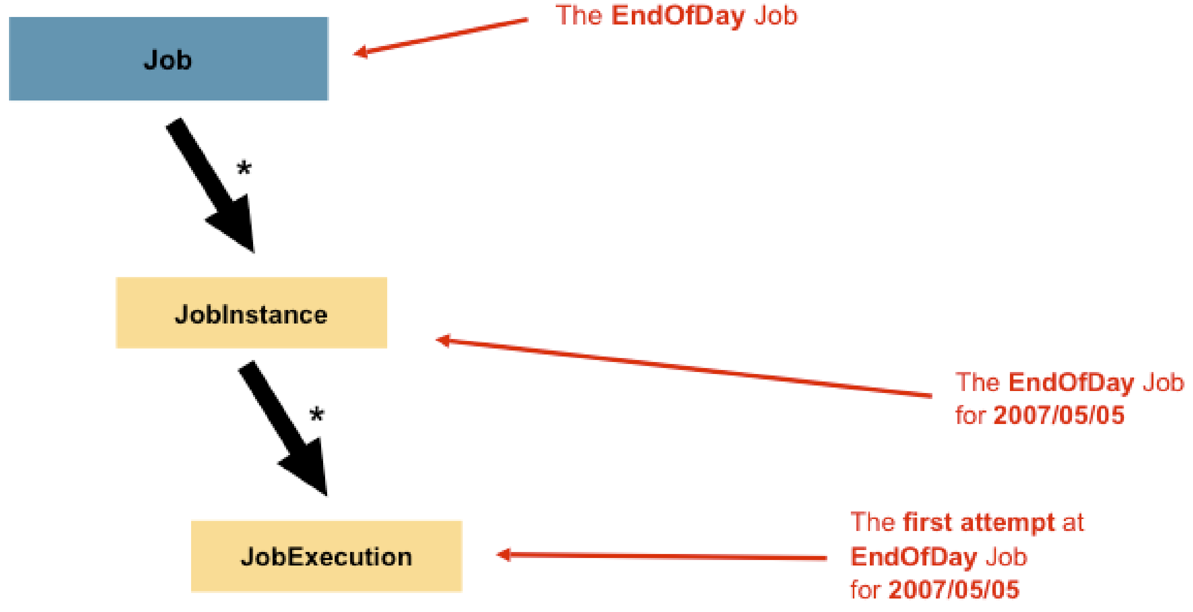
Spring Batch DSL

A Typical Batch Job Architecture



Job

- A Job is an entity that encapsulates an entire batch process.
- A Job is wired together with either an XML configuration file or Java-based configuration.
- This configuration may be referred to as the "job configuration".

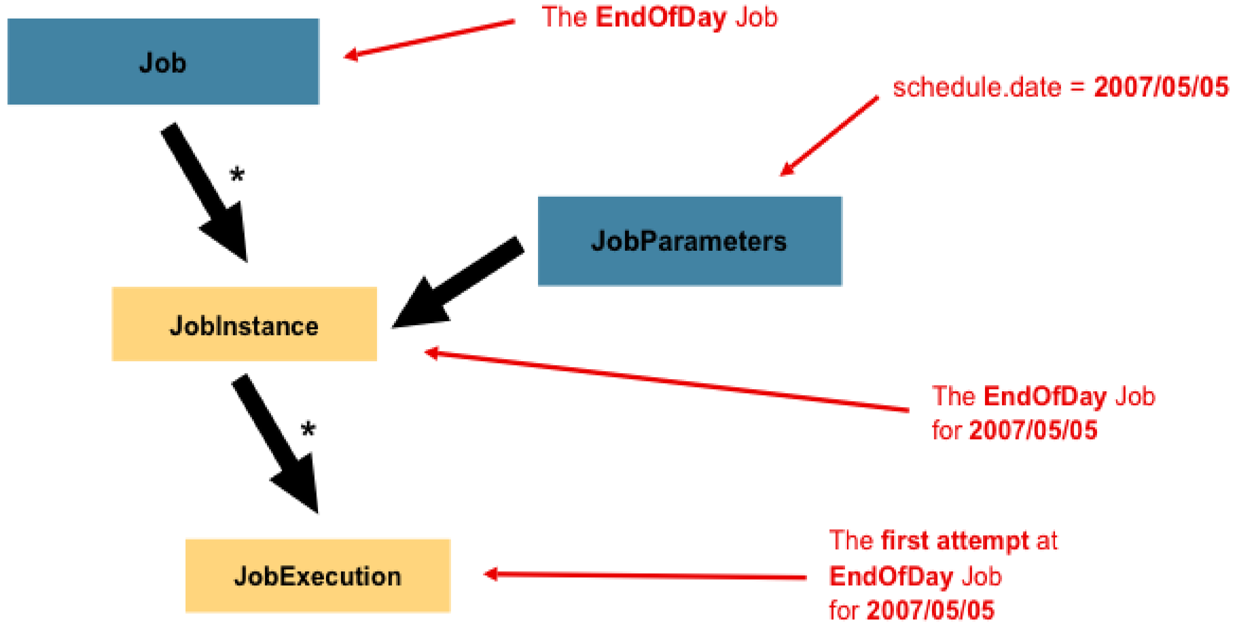

Job
The EndOfDay Job
*
JobInstance
The EndOfDay Job for 2007/05/05
*
JobExecution
The first attempt at EndOfDay Job for 2007/05/05

JobInstance

- A **JobInstance** refers to the concept of a logical job run
- Each **JobInstance** can have multiple executions
- Only one **JobInstance** corresponding to a particular **Job** and identifying **JobParameters** can run at a given time

JobParamters

- A **JobParameters** object holds a set of parameters used to start a batch job.
- They can be used for identification or even as reference data during the run
- JobInstanes are distinguished from each other based on JobParameters



JobExecution

- A JobExecution refers to the technical concept of a single attempt to run a Job.
- An execution may end in failure or success, but the JobInstance corresponding to a given execution is not considered to be complete unless the execution completes successfully.

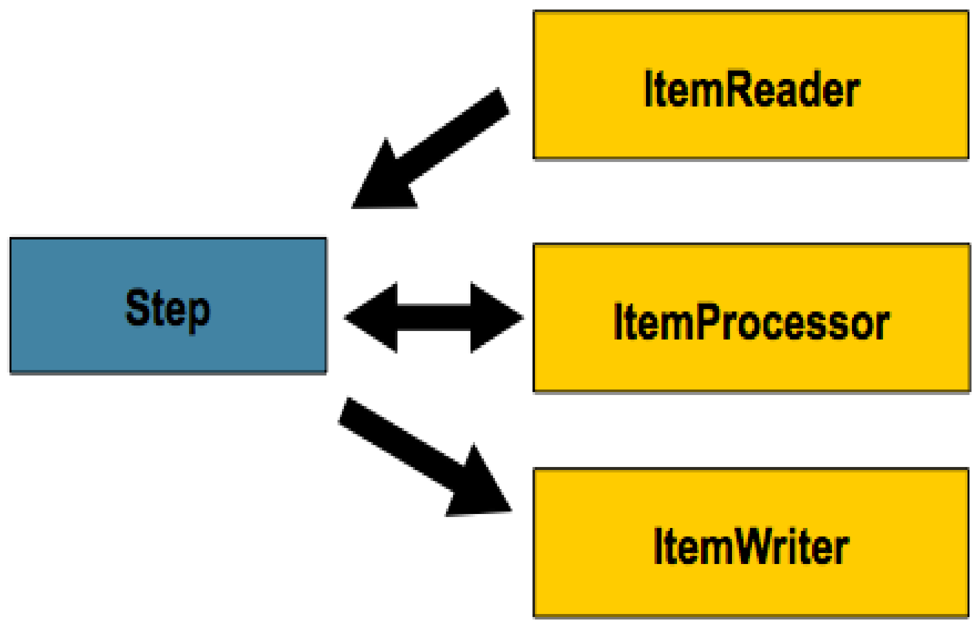
JobExecution

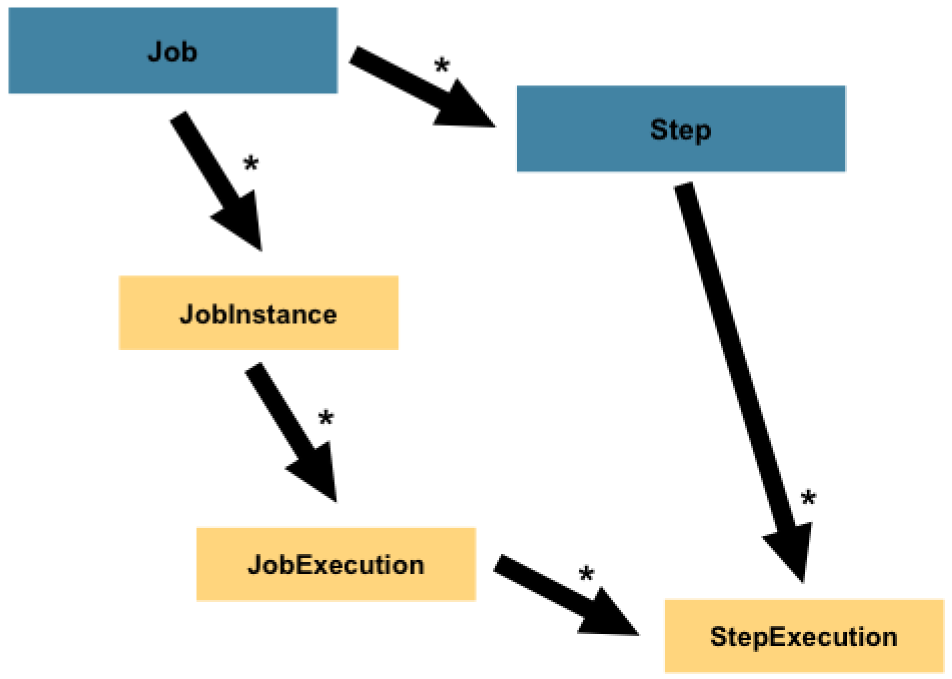
- A **Job** defines what a job is and how it is to be executed, and a **JobInstance** is a purely organizational object to group executions together, primarily to enable correct restart semantics.
- A **JobExecution**, however, is the primary storage mechanism for what actually happened during a run and contains many more properties that must be controlled and persisted.
- Some of the JobExecution properties are:
  - Status ,startTime, endTime, createTime etc.

Step

- A Step is a domain object that encapsulates an independent, sequential phase of a batch job.
- Every Job is composed entirely of one or more steps.
- A Step contains all of the information necessary to define and control the actual batch processing.
- A Step can be as simple or complex as the developer desires.
  - A simple Step might load data from a file into the database, requiring little or no code.
  - A more complex Step may have complicated business rules that are applied as part of the processing.
- As with a Job, a Step has an individual **StepExecution** that correlates with a unique **JobExecution**,

Visualisation of Step





StepExecution

- A **StepExecution** represents a single attempt to execute a Step.
- A new **StepExecution** is created each time a Step is run, similar to **JobExecution**.
- However, if a step fails to execute because the step before it fails, no execution is persisted for it.
- A **StepExecution** is created only when its Step is actually started

ExecutionContext

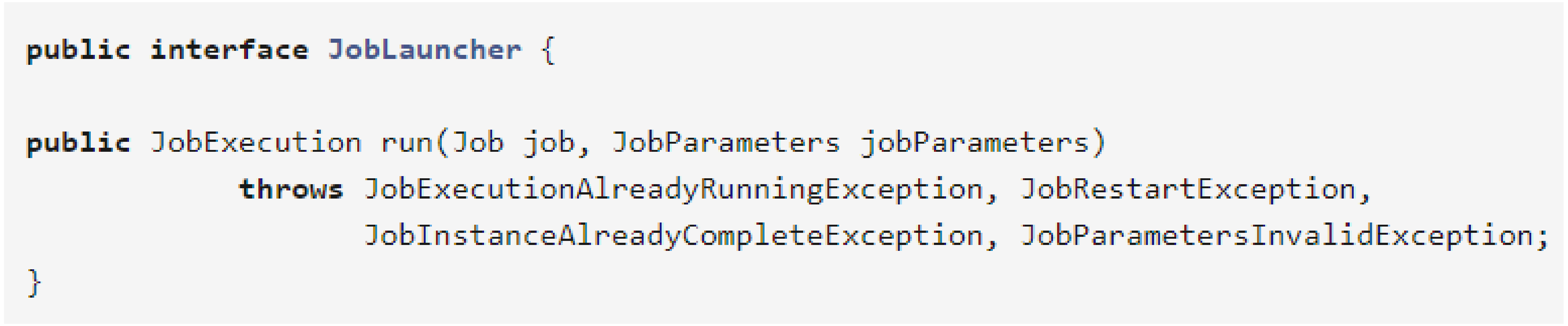
- An **ExecutionContext** represents a collection of key/value pairs.
- Persisted and controlled by the framework in order to allow developers a place to store persistent state that is scoped to a **StepExecution** object or a **JobExecution** object.

JobRepository

- JobRepository is the persistence mechanism for all of the Stereotypes mentioned above.
- It provides CRUD operations for JobLauncher, Job, and Step implementations.
- When a Job is first launched, a JobExecution is obtained from the repository, and, during the course of execution, StepExecution and JobExecution implementations are persisted by passing them to the repository.
- When using java configuration, @EnableBatchProcessing annotation provides a **JobRepository** as one of the components automatically configured out of the box.

JobLauncher

- JobLauncher represents a simple interface for launching a Job with a given set of JobParameters, as shown in the following



```
public interface JobLauncher {

public JobExecution run(Job job, JobParameters jobParameters)
            throws JobExecutionAlreadyRunningException, JobRestartException,
                   JobInstanceAlreadyCompleteException, JobParametersInvalidException;
}
```

Item Reader

- **ItemReader** is an abstraction that represents the retrieval of input for a Step, one item at a time.
- When the **ItemReader** has exhausted the items it can provide, it indicates this by returning null

Item Writer

- ItemProcessor is an abstraction that represents the business processing of an item.
- While the ItemReader reads one item, and the ItemWriter writes them, the ItemProcessor provides an access point to transform or apply other business processing.
- If, while processing the item, it is determined that the item is not valid, returning null indicates that the item should not be written out.

Configuring And Running A Job

Java Configuration

- Use @EnableBatchProcessing to Enable Spring Batch Processing Features in your application.
- @EnableBatchProcessing provides a base configuration for building batch jobs.
- Within this base configuration, an instance of **StepScope** is created in addition to a number of beans made available to be autowired:

  JobRepository - bean name "jobRepository"

  JobLauncher - bean name "jobLauncher"

  JobRegistry - bean name "jobRegistry"

  PlatformTransactionManager - bean name "transactionManager"

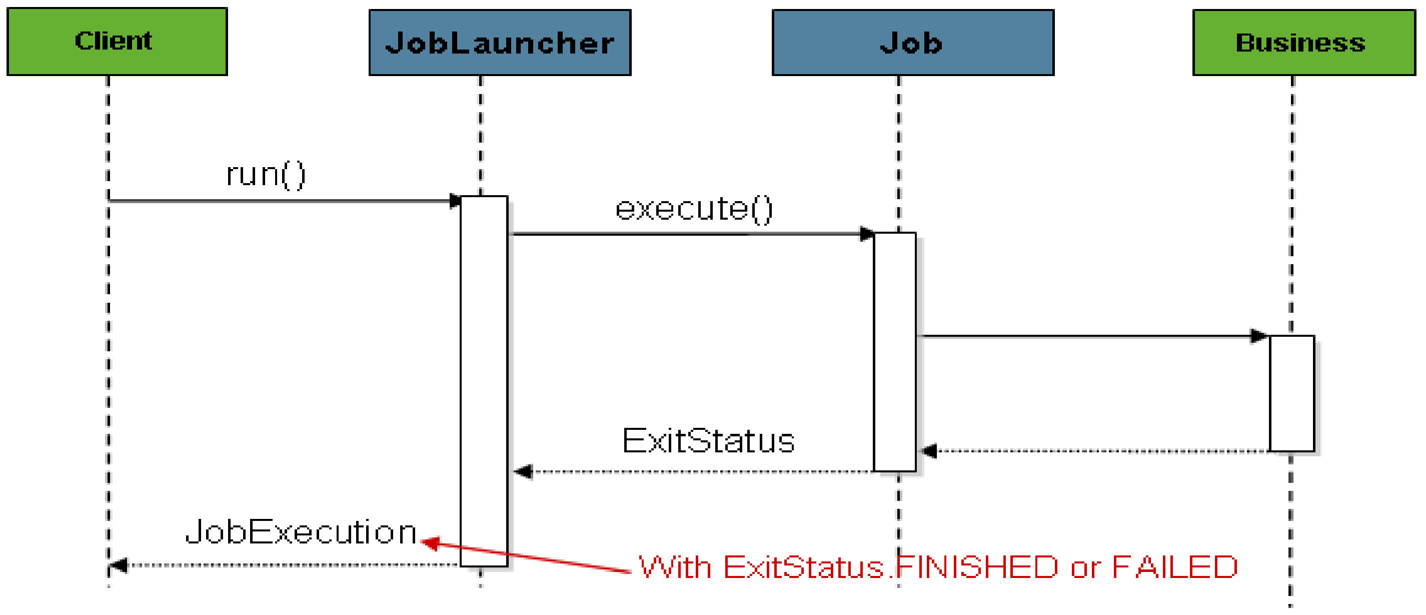
  JobBuilderFactory - bean name "jobBuilders"

  StepBuilderFactory - bean name "stepBuilders"

Configuring a JobRepository

- When using @EnableBatchProcessing, a JobRepository is provided out of the box for you.
- A JDBC based one is provided out of the box if a DataSource is provided, the Map based one if not.
- However you can customize the configuration of the JobRepository via an implementation of the **BatchConfigurer** interface.

The Job Execution


Client
JobLauncher
Job
Business
run()
execute()
ExitStatus
JobExecution
With ExitStatus.FINISHED or FAILED