Spring Batch 2.0 Overview

Guido Schmutz
Technology Manager
guido.schmutz@trivadis.com

Zurich, 18.3.2009

trivadis
makes IT easier.

Introduction

- Guido Schmutz
  - Working for Trivadis for more than 12 years
  - Co-Author of different books
  - Consultant, Trainer, Software Architect for Java, Oracle, SOA and EDA
  - Member of Trivadis Architecture Board
  - Trivadis Technology Manager
  - More than 20 years of software development experience
  - Contact: guido.schmutz@trivadis.com

Agenda

- Spring Batch Overview
- Domain Language of Batch
- Configuring and Running a Job
- Miscellaneous
- Summary

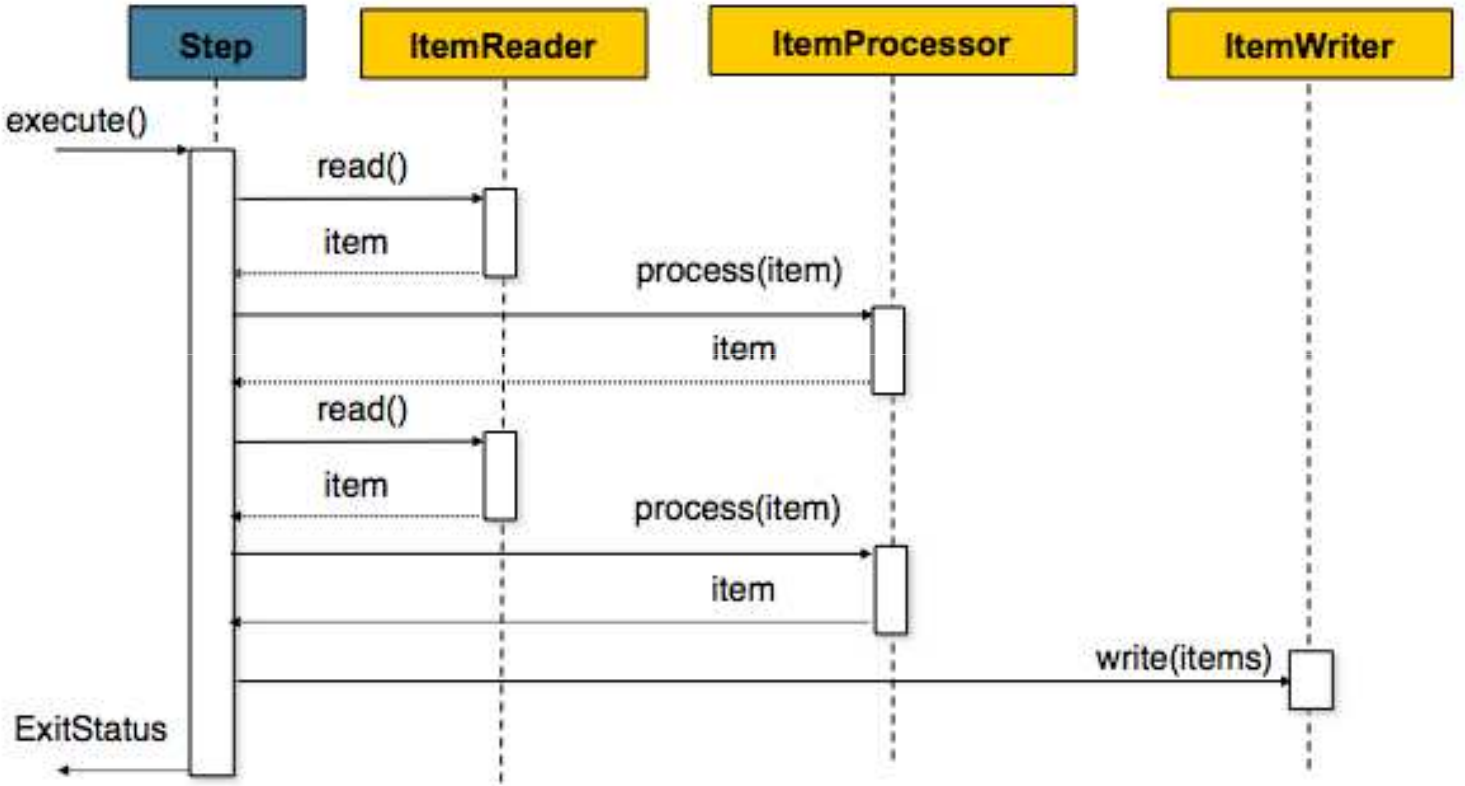
Spring Batch Introduction

- Spring Batch is the first java based framework for batch processing
  - a lightweight, comprehensive batch framework
  - builds upon the productivity, POJO-based development approach, known from the Spring Framework
  - current GA release is 1.1.4.RELEASE
  - Spring Batch 2.0 will be released in the next couple of months
  - Presentation is based on 2.0.0-RC1

Batch Processing

- What is a Batch Application?
  - Batch applications need to process high volume business critical transactional data
  - A typical batch program generally
    1. reads a large number of records from a database, file, or queue
    2. processes the data in some fashion, and
    3. then writes back data in a modified form

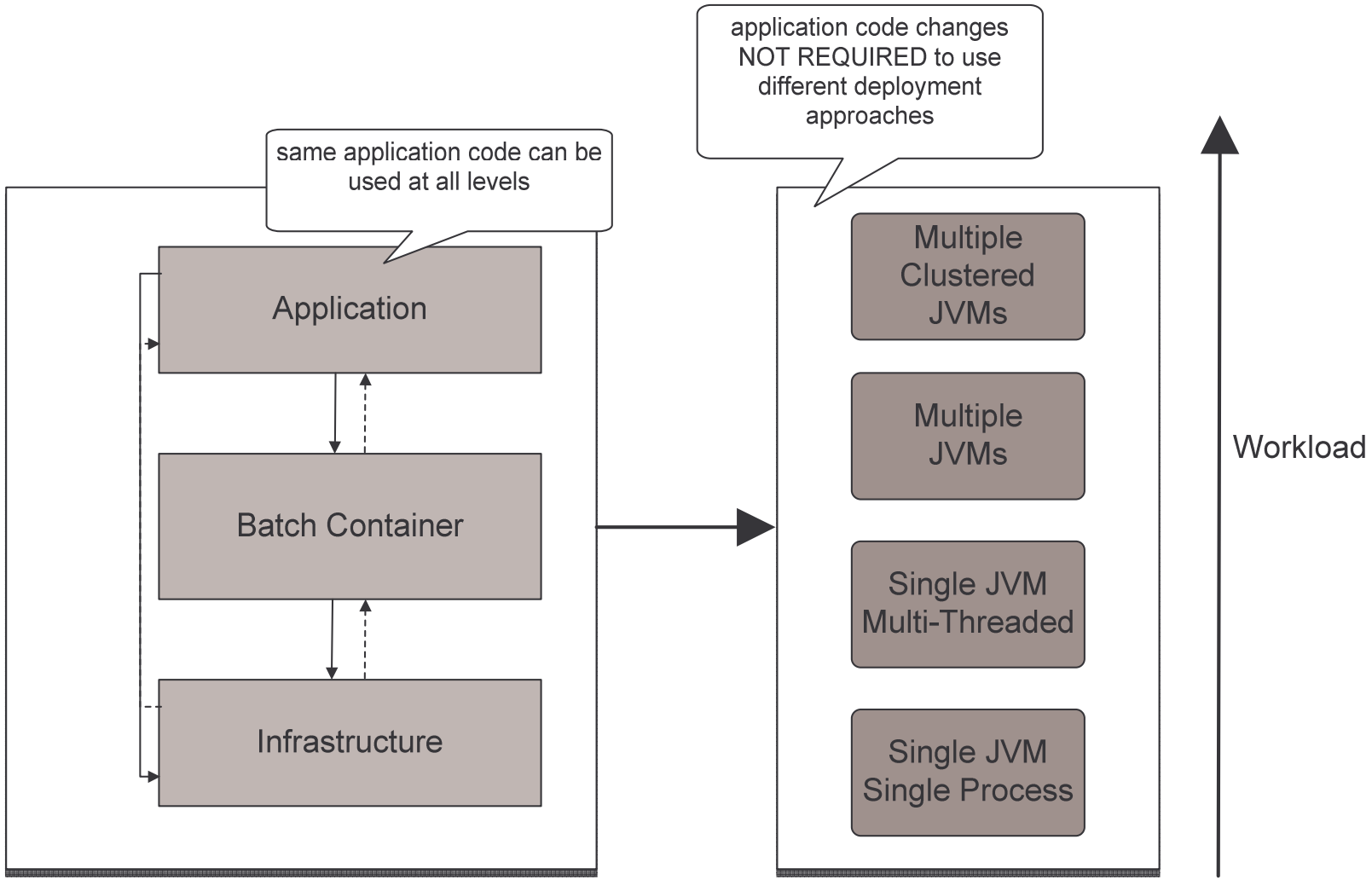
Item Oriented Processing



Usage Scenarios

- Commit batch process periodically
- Concurrent batch processing: parallel processing of a job
- Staged, enterprise message-driven processing
- Massively parallel batch processing
- Manual or scheduled restart after failure
- Sequential processing of dependent steps (with extensions to workflow-driven batches)
- Partial processing: skip records (e.g. on rollback)
- Whole-batch transaction: for cases with a small batch size or existing stored procedures/scripts

Spring Batch: Layered Architecture

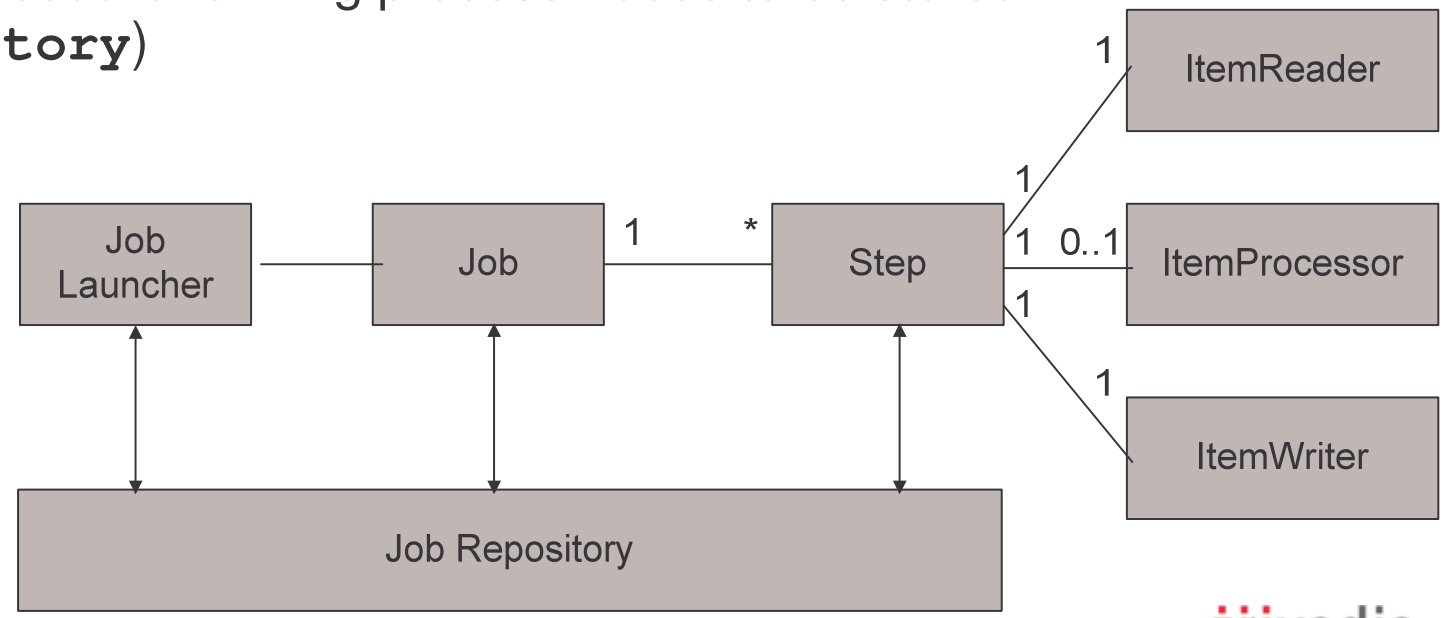


Agenda

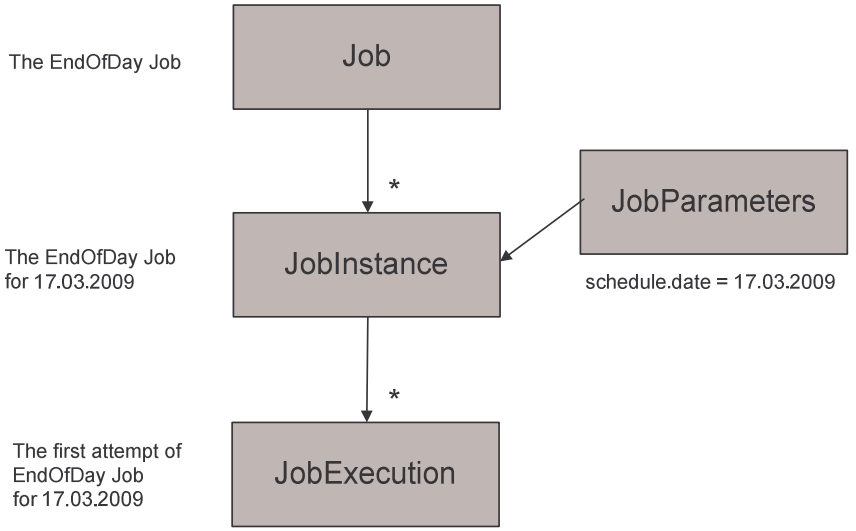
- Spring Batch Overview
- Domain Language of Batch
- Configuring and Running a Job
- Miscellaneous
- Summary

Domain Language of Batch

- A **job** has one to many steps
- A **step** has exactly one `ItemReader`, `ItemWriter` and optionally an `ItemProcessor`
- A job needs to be launched (`JobLauncher`)
- Meta data about the running process needs to be stored (`JobRepository`)

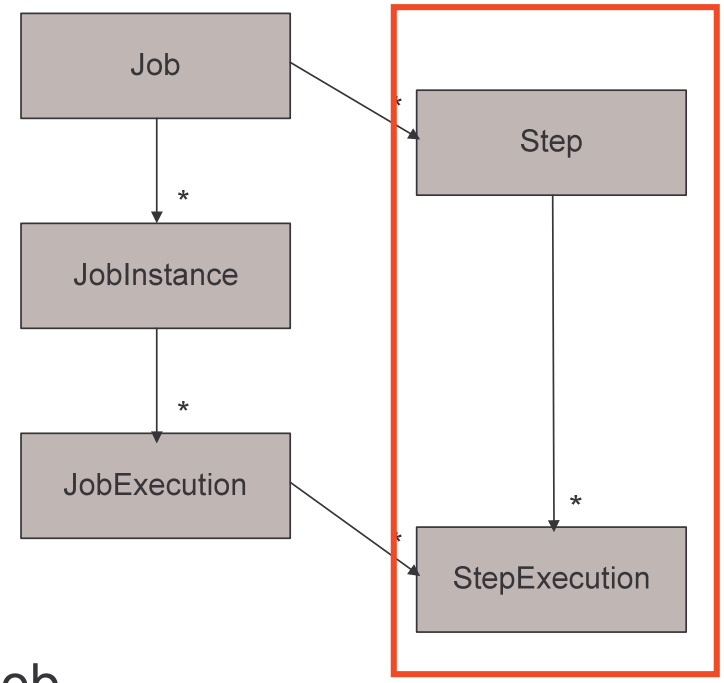


Domain Language of Batch



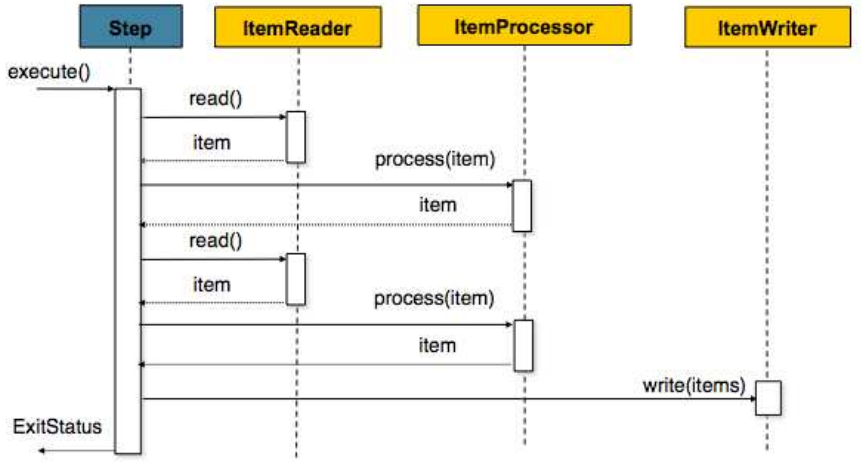
- Job
  - encapsulates an entire batch process
- Job Instance
  - refers to the concept of a logical job run
  - job running once at end of day, will have one logical `JobInstance` per day
  - each `JobInstance` can have multiple executions
- Job Execution
  - refers to the technical concept of a single attempt to run a Job
  - An execution may end in failure or success, but the `JobInstance`
    - will not be considered complete unless the execution completes successfully
- Job Parameters
  - is a set of parameters used to start a batch job
  - `JobInstance` = `Job` + `JobParameters`

Domain Language of Batch



- Step
  - a domain object that encapsulates an independent, sequential phase of a batch job
  - can be as simple or complex as the developer desires
- Step Execution
  - represents a single attempt to execute a Step
  - A new `StepExecution` will be created each time a `Step` is run, similar to `JobExecution`
  - A `StepExecution` will only be created when its `Step` is actually started

Domain Language of Batch



- Item Reader
  - an abstraction that represents the retrieval of input for a Step
    - one item at a time
  - When it has exhausted the items it can provide, it will indicate this by returning null
  - Various implementation available out-of-the-box
- Item Writer
  - an abstraction that represents the output of a Step
    - Chunk-oriented processing
  - Generally, an item writer has no knowledge of the input it will receive next
  - Various implementation available out-of-the-box
- Item Processor
  - an abstraction that represents the business processing of an item
  - provides access to transform or apply other business processing
  - returning null indicates that the item should not be written out

Domain Language of Batch

- ItemReader

```
public interface ItemReader<T> {
   T read() throws Exception, UnexpectedInputException,
                   ParseException;
}
```

- ItemWriter

```
public interface ItemWriter<T> {
   void write(List<? extends T> items) throws Exception;
}
```

- ItemProcessor

```
public interface ItemProcessor<I, O> {
   O process(I item) throws Exception;
}
```

Domain Language of Batch

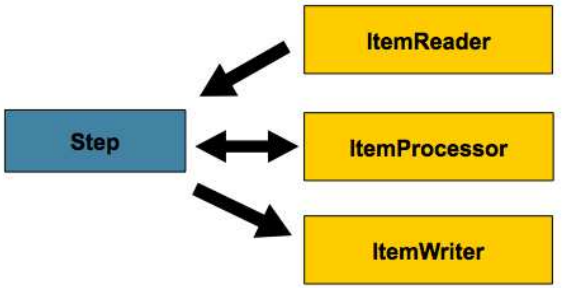
- Job Repository
  - the persistence mechanism for all of the Stereotypes
  - provides CRUD operations for `JobLauncher`, `Job`, and `Step` implementations
- Job Launcher
  - represents a simple interface for launching a Job with a given set of `JobParameters`

```
public interface JobLauncher {
  public JobExecution run(Job job,
                          JobParameters jobParameters)
            throws JobExecutionAlreadyRunningException,
                   JobRestartException;
}
```

Agenda

- Spring Batch Overview
- Domain Language of Batch
- Configuring and Running a Job
- Miscellaneous
- Summary

Configuring and Running a Job



- Configuring a Job and its steps
  - There are multiple implementations of the `Job` interface, however, the namespace abstracts away the differences in configuration
  - It has only three required dependencies: a name, `JobRepository`, and a list of Steps

```
<job id="sampleJob">
  <step id="step1" job-repository="jobRepository"
        transaction-manager="transactionManager">
    <tasklet reader="itemReader" writer="itemWriter"
             commit-interval="10"/>
  </step>
</job>

<bean id="itemReader" ...>
<bean id="itemWriter" ...>
```

Configuring and Running a Job

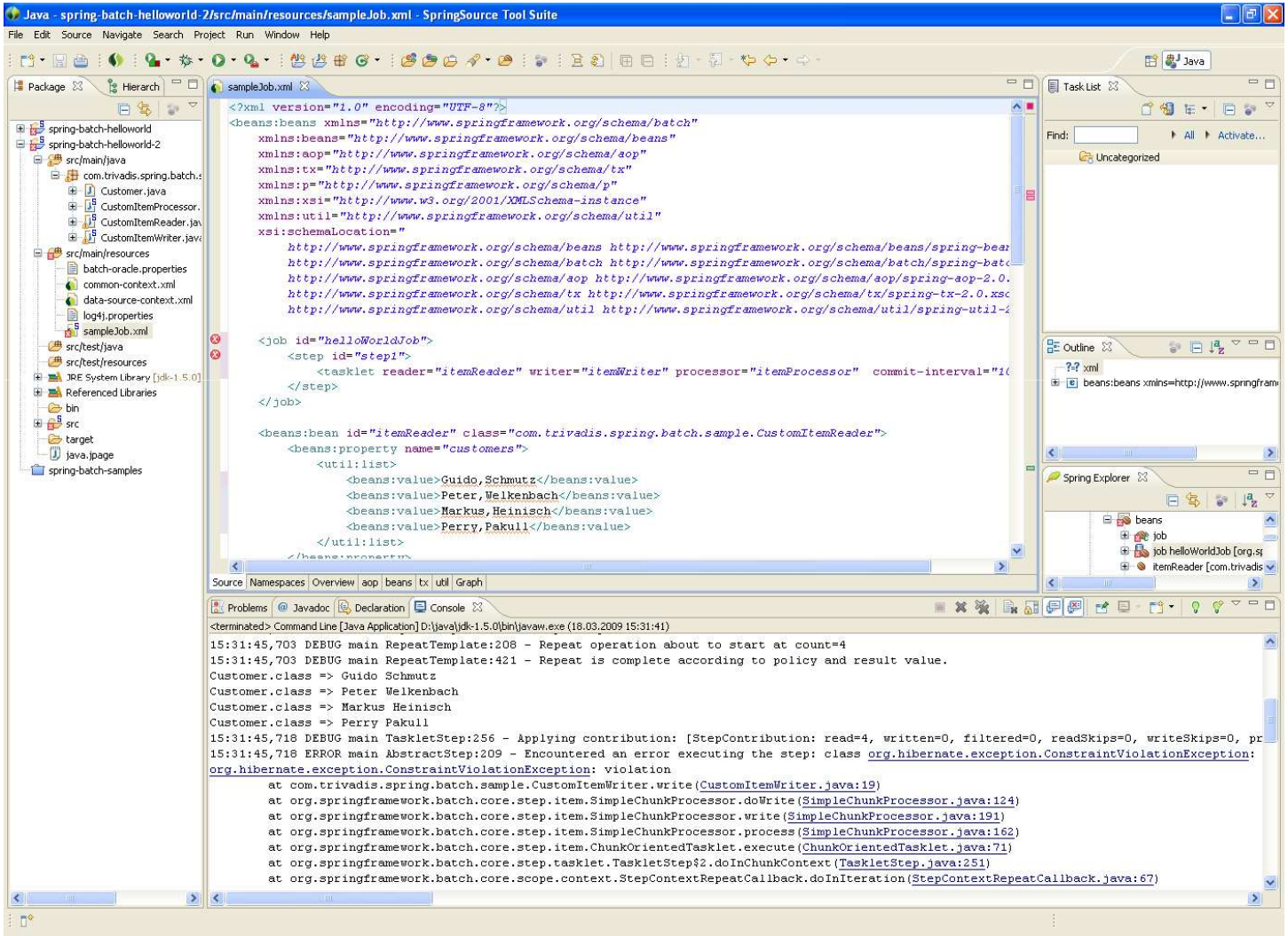
- Configuring a Job Repository
  - used for basic CRUD operations of the various persisted domain objects such as `JobExecution` and `StepExecution`
  - batch namespace abstracts away many of the implementation details

```
<job-repository id="jobRepository"
                dataSource="dataSource"
                transactionManager="transactionManager"
                isolation-level-for-create="serializable"
                table-prefix="BATCH_" />
```

- Configuring a Job Launcher

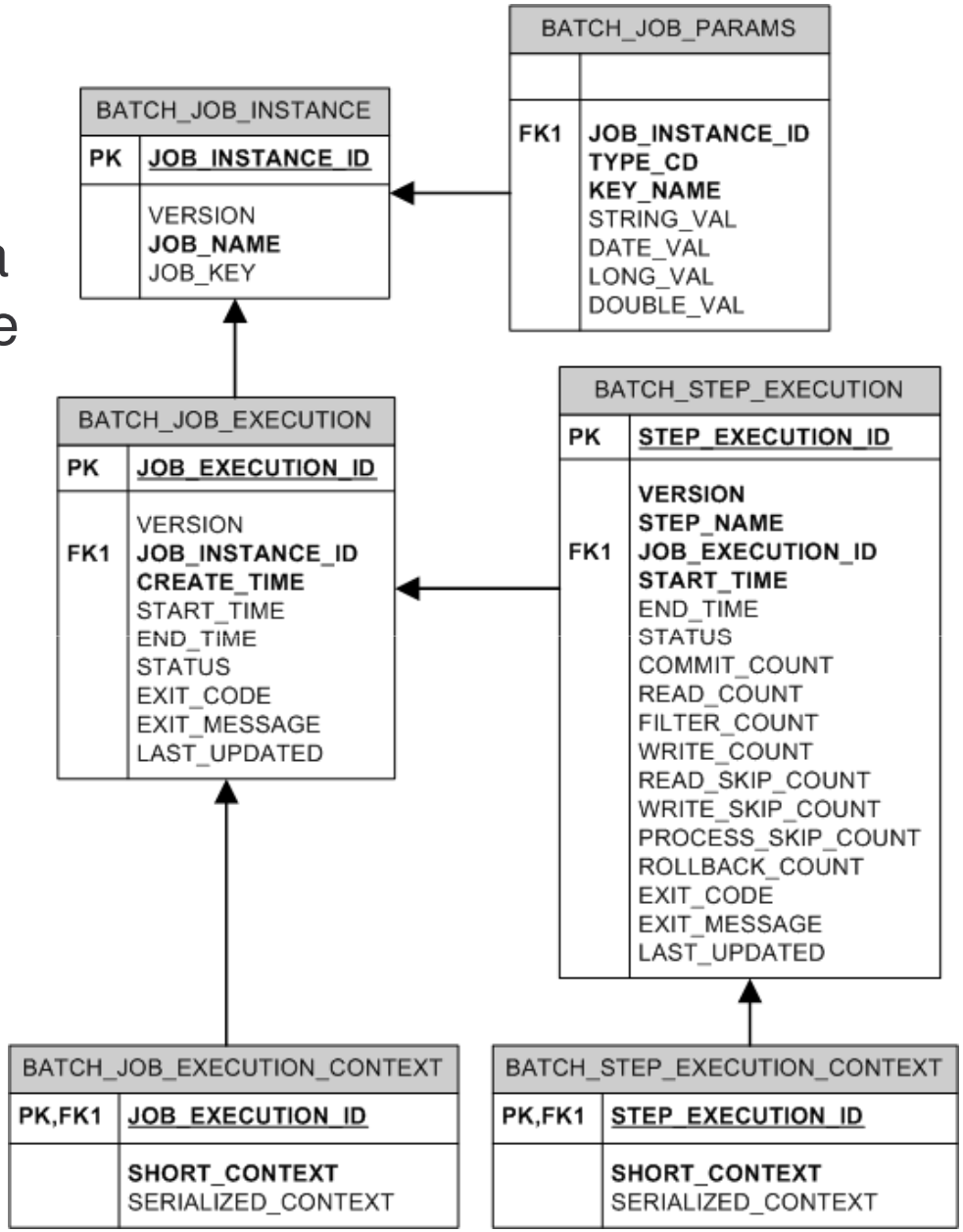
```
<bean id="jobLauncher"
    class="...batch.execution.launch.SimpleJobLauncher">
  <property name="jobRepository" ref="jobRepository" />
</bean>
```

Demo



Meta-Data Schema

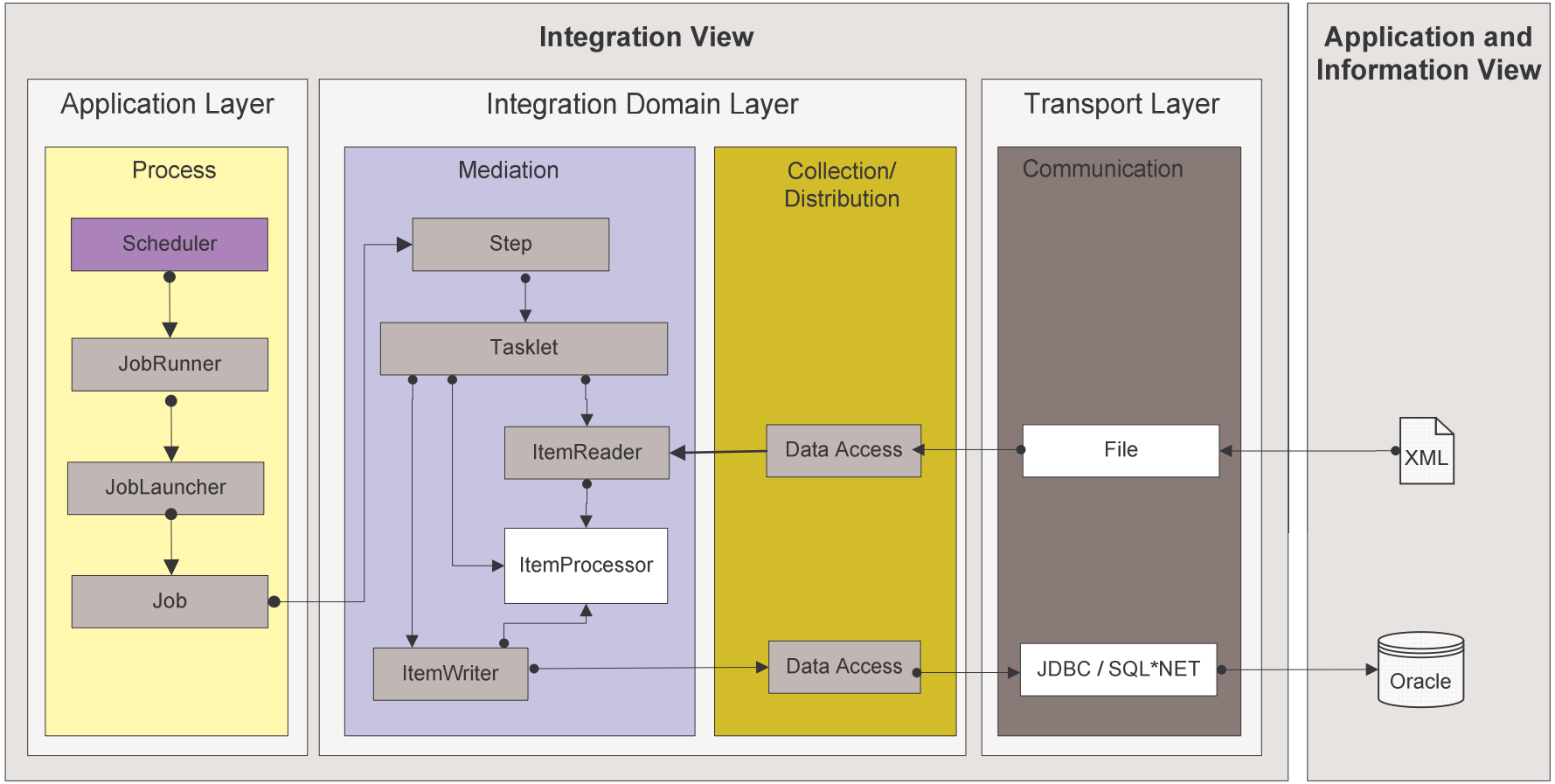
- The Spring Batch Meta-Data tables very closely match the Domain objects that represent them in Java



Agenda

- Spring Batch Overview
- Domain Language of Batch
- Configuring and Running a Job
- Miscellaneous
- Summary

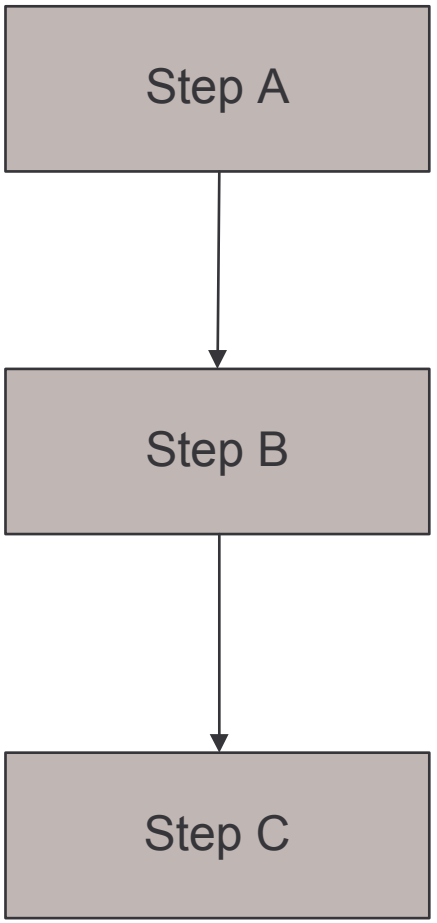
Spring Batch in Trivadis Integration Architecture Blueprint



Sequential Flow

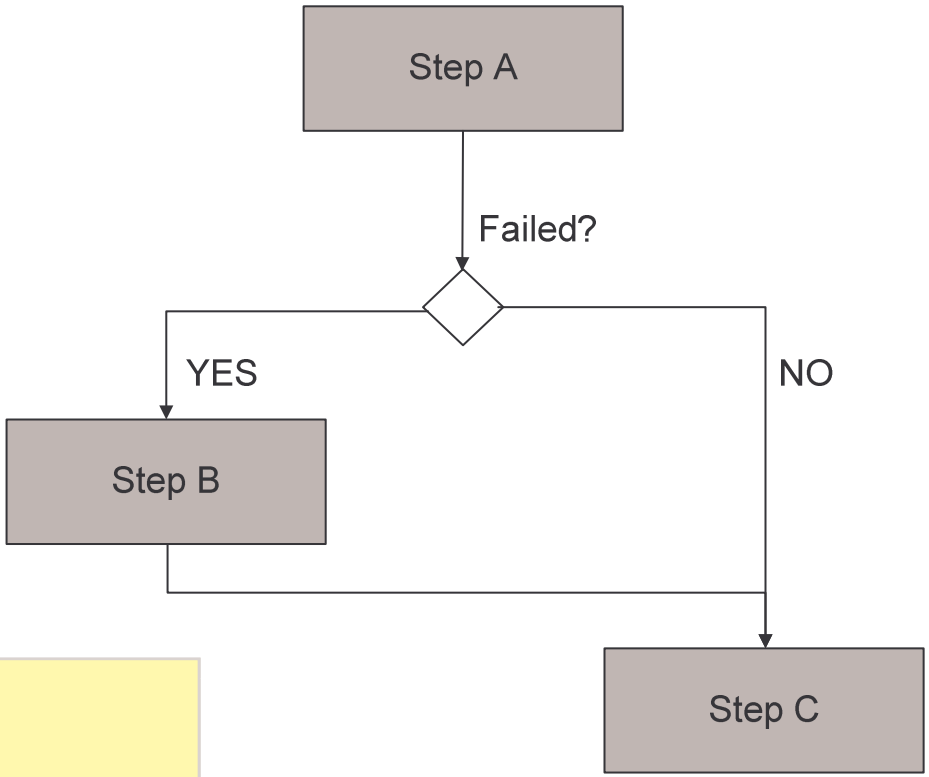
- The simplest flow scenario is a job where all of the steps execute sequentially
- This can be achieved using the 'next' attribute of the step element

```
<job id="job">
  <step id="stepA" next="stepB" />
  <step id="stepB" next="stepC"/>
  <step id="stepC"/>
</job>
```



Conditional Flow

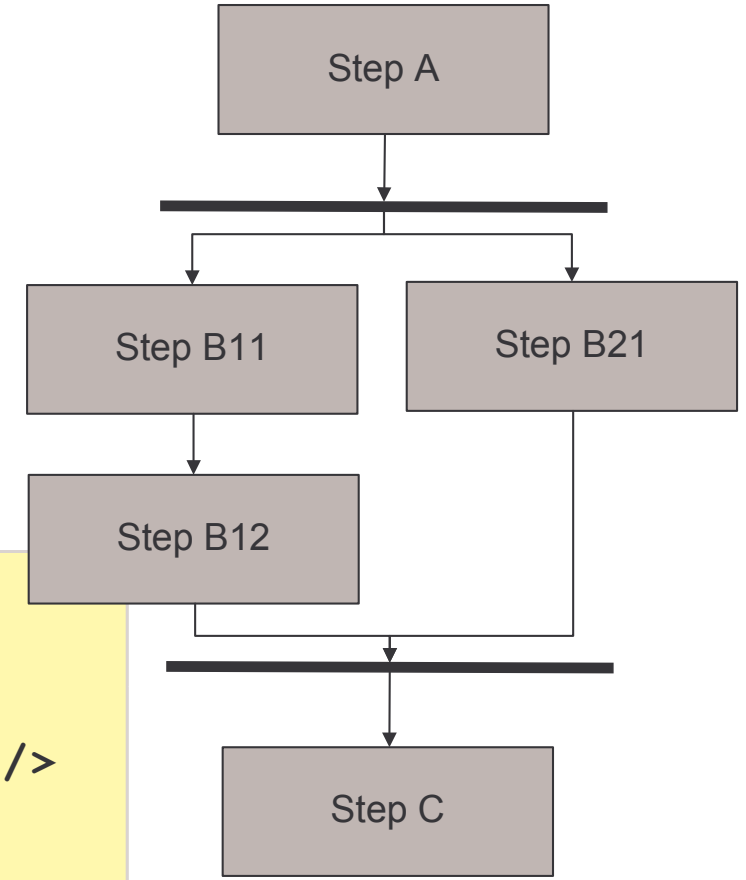
- In order to handle more complex scenarios, Spring Batch allows transition elements to be defined within the step element




```
<job id="job">
  <step id="stepA">
    <next on="FAILED" to="stepB" />
    <next on="*" to="stepC" />
  </step>
  <step id="stepB" next="stepC" />
  <step id="stepC" />
</job>
```

Split Flow

- Spring Batch also allows for a job to be configured with parallel flows using the 'split' element




```
<step id="stepA" next="stepB"/>
<split id="stepB" next="stepC">
  <flow>
    <step id="stepB11" next="stepB11"/>
    <step id="stepB12"/>
  </flow>
  <flow>
    <step id="stepB21"/>
  </flow>
</split>
<step id="stepC"/>
```

Restartablility

- The launching of a Job is considered to be a 'restart' if a `JobExecution` already exists for the particular `JobInstance`.
  - Ideally, all jobs should be able to start up where they left off
  - but there are scenarios where this is not possible
- If a Job should never be restarted, but should always be run as part of a new `JobInstance`, then the restartable property may be set to 'false‘

```
<job id="footballJob" restartable="false">
  <step id="playerload" next="gameLoad"/>
  <step id="gameLoad" next="playerSummarization"/>
  <step id="playerSummarization"/>
</job>
```

Configuring a Step for Restart

- Setting a StartLimit
  - control the number of times a Step may be started

```
<step id="step1">
 <tasklet reader="itemReader" writer="itemWriter"
          commit-interval="10" start-limit="1"/>
 </step>
```

- Restarting a completed step
  - In a restartable job, one or more steps should always be run, regardless of whether or not they were successful the first time

```
<step id="step1">
 <tasklet reader="itemReader" writer="itemWriter"
          commit-interval="10"
          allow-start-if-complete="true"/>
 </step>
```

Configuring Skip Logic

- there are scenarios where errors encountered should not result in Step failure, but should be skipped instead

```
<step id="step1">
 <tasklet reader="flatFileItemReader"
          writer="itemWriter" commit-interval="10"
          skip-limit="10">
   <skippable-exception-classes>
org.springframework.batch.item.file.FlatFileParseExcept
ion
   </skippable-exception-classes>
 </tasklet>
</step>
```

Configuring Fatal Exceptions

- it may be easier to identify which exceptions should cause failure and skip everything else

```
<step id="step1">
 <tasklet reader="flatFileItemReader"
          writer="itemWriter" commit-interval="10"
          skip-limit="10">
  <skippable-exception-classes>
     java.lang.Exception
  </skippable-exception-classes>
  <fatal-exception-classes>
     java.io.FileNotFoundException
  </fatal-exception-classes>
 </tasklet>
</step>
```

Intercepting Step Execution

- You might need to perform some functionality at certain events during the execution of a Step

```
<step id="step1">
  <tasklet reader="reader" writer="writer"
           commit-interval="10"/>
 <listeners>
    <listener ref="stepListener"/>
 </listeners>
</step>
```

  - can be accomplished with one of many Step scoped listeners, like
    - `StepExecutionListener`
    - `ChunkListener`
    - `ItemReadListener`
    - `ItemProcessListener`
    - `ItemWriteListener`
    - `SkipListener`

Available Item Readers

Item Reader	Description
AbstractItemCountingItemStreamItemReader	Abstract base class that provides basic restart capabilities by counting the number of items returned from an `ItemReader`.
ListItemReader	Provides the items from a list, one at a time
ItemReaderAdapter	Adapts any class to the `ItemReader` interface.
AggregateItemReader	An ItemReader that delivers a list as its item, storing up objects from the injected ItemReader until they are ready to be packed out as a collection. This ItemReader should mark the beginning and end of records with the constant values in FieldSetMapper AggregateItemReader**#BEGIN_RECORD** and AggregateItemReader**#END_RECORD**
FlatFileItemReader	Reads from a flat file. Includes ItemStream and Skippable functionality. See section on Read from a File
StaxEventItemReader	Reads via StAX. See HOWTO - Read from a File
JdbcCursorItemReader	Reads from a database cursor via JDBC. See HOWTO - Read from a Database
HibernateCursorItemReader	Reads from a cursor based on an HQL query. See section on Reading from a Database
IbatisPagingItemReader	Reads via iBATIS based on a query. Pages through the rows so that large datasets can be read without running out of memory. See HOWTO - Read from a Database
JmsItemReader	Given a Spring JmsOperations object and a JMS Destination or destination name to send errors, provides items received through the injected JmsOperations receive() method
JpaPagingItemReader	Given a JPQL statement, pages through the rows, such that large datasets can be read without running out of memory
JdbcPagingItemReader	Given a SQL statement, pages through the rows, such that large datasets can be read without running out of memory

Available Item Writers

Item Writer	Description
AbstractItemStreamItemWriter	Abstract base class that combines the `ItemStream` and `ItemWriter` interfaces.
CompositeItemWriter	Passes an item to the process method of each in an injected **List** of **ItemWriter** objects
ItemWriterAdapter	Adapts any class to the `ItemWriter` interface.
PropertyExtractingDelegatingItemWriter	Extends AbstractMethodInvokingDelegator creating arguments on the fly. Arguments are created by retrieving the values from the fields in the item to be processed (via a SpringBeanWrapper) based on an injected array of field name
FlatFileItemWriter	Writes to a flat file. Includes ItemStream and Skippable functionality. See section on Writing to a File
HibernateItemWriter	This item writer is hibernate session aware and handles some transaction-related work that a non-"hibernate aware" item writer would not need to know about and then delegates to another item writer to do the actual writing.
JdbcBatchItemWriter	Uses batching freatures from a `PreparedStatement`, if available, and can take rudimentary steps to locate a failure during a `flush`.
JpaItemWriter	This item writer is JPA EntityManager aware and handles some transaction-related work that a non-"jpa aware" `ItemWriter` would not need to know about and then delegates to another writer to do the actual writing.
StaxEventItemWriter	Uses an **ObjectToXmlSerializer** implementation to convert each item to XML and then writes it to an XML file using StAX.

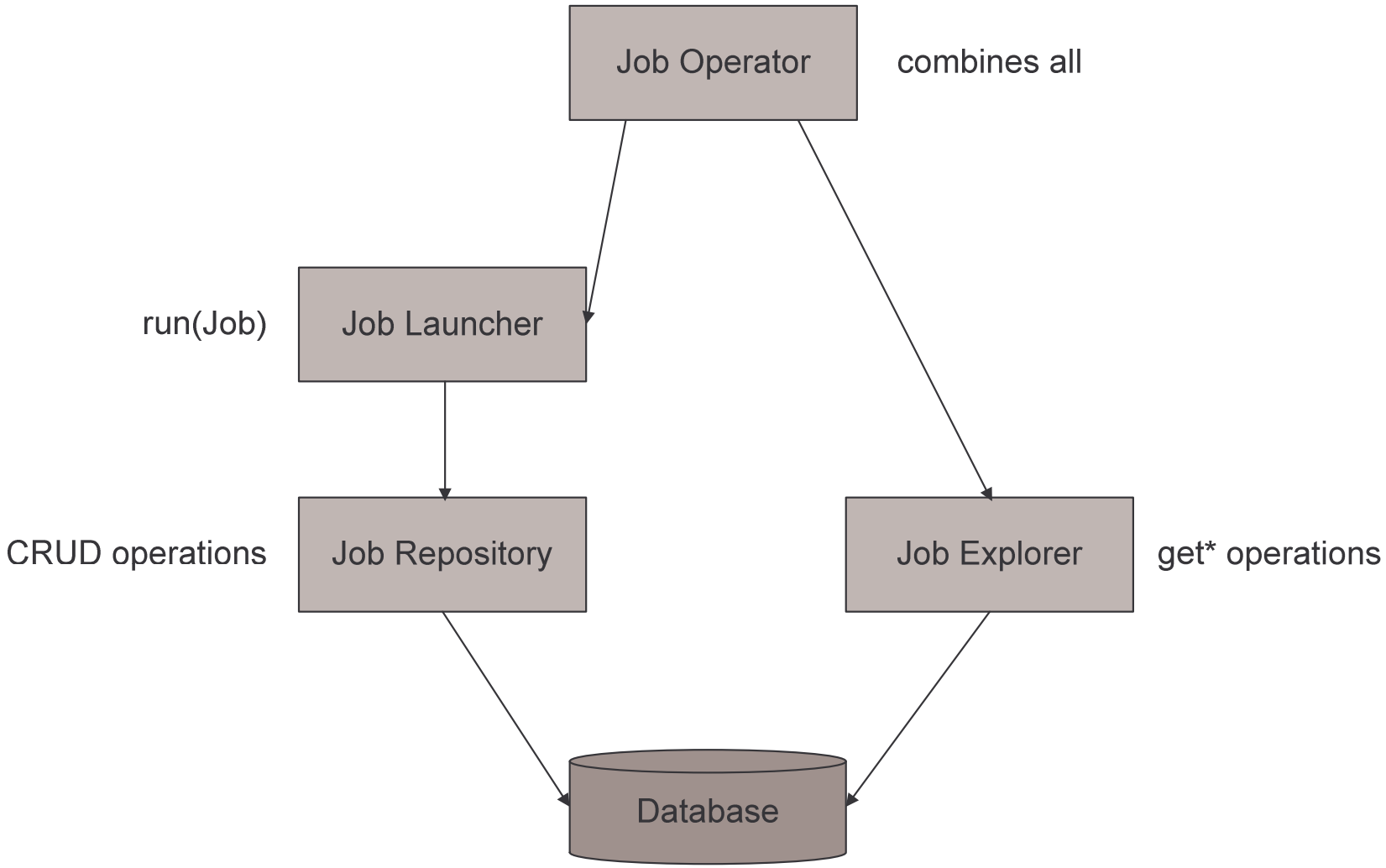
Agenda

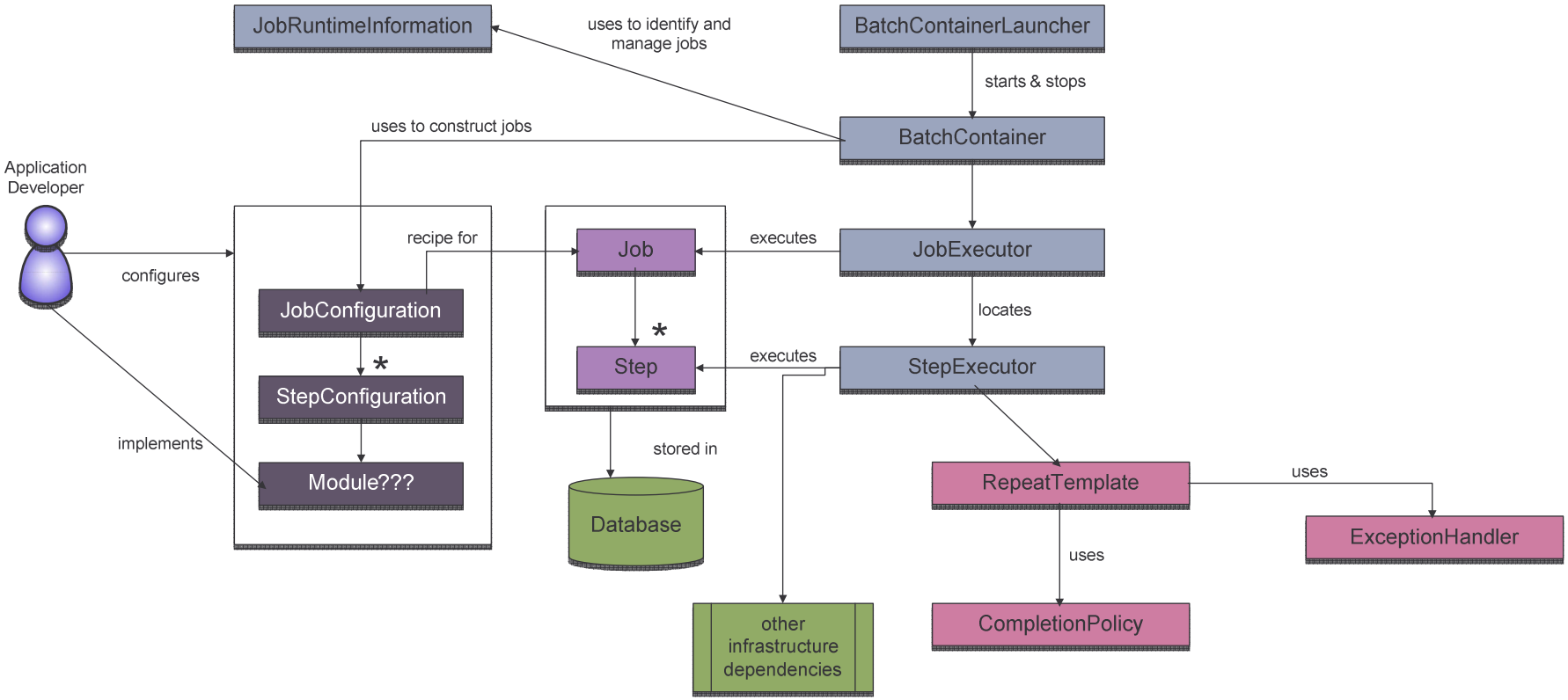
- Spring Batch Overview
- Domain Language of Batch
- Configuring and Running a Job
- Miscellaneous
- Summary

Summary

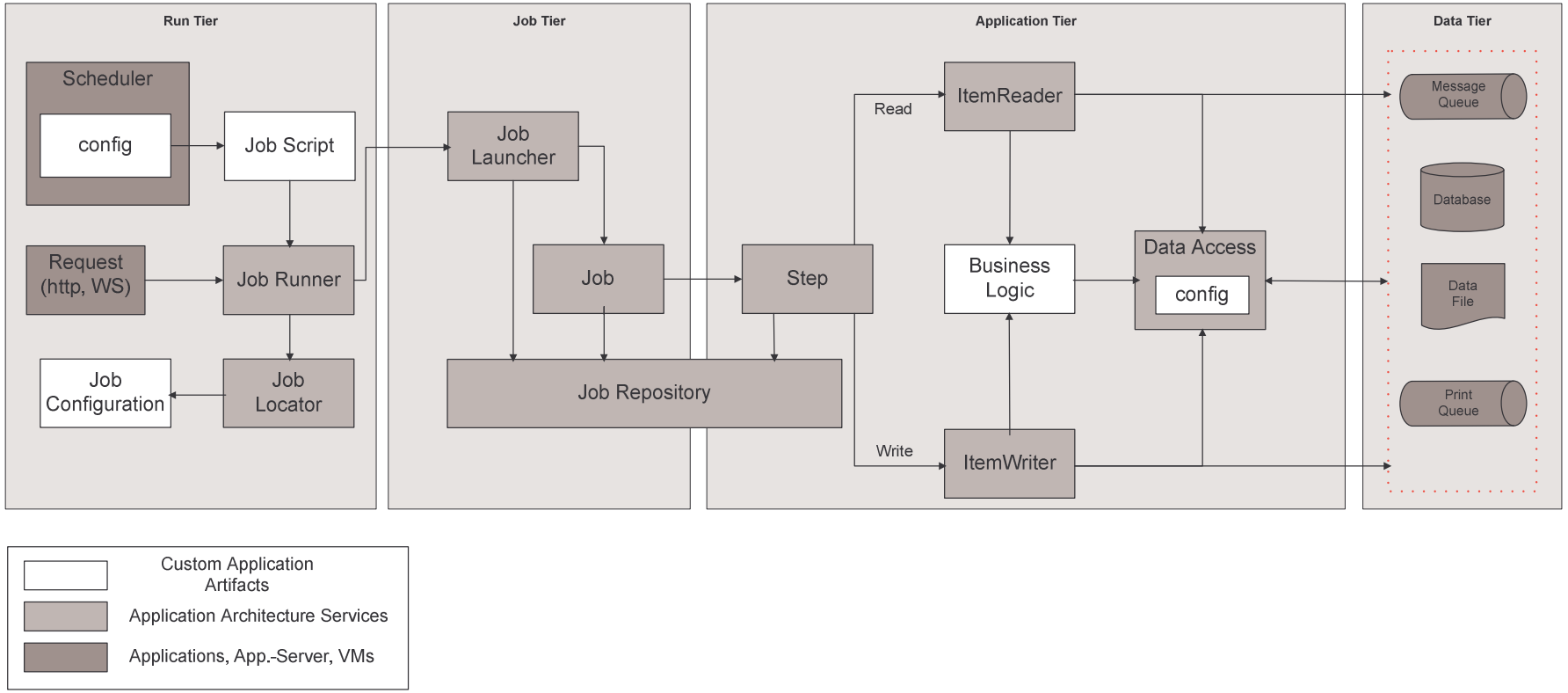
- Lack of a standard enterprise batch architecture is resulting in higher costs associated with the quality and delivery of solutions.
- Spring Batch provides a highly scalable, easy-to-use, customizable, industry-accepted batch framework collaboratively developed by Accenture and SpringSource
- Spring patterns and practices have been leveraged allowing developers to focus on business logic, while enterprise architects can customize and extend architecture concerns

Thank you!





Spring Batch Launch Environment



Spring Batch Launch Environment

- Run Tier
  - concerned with the scheduling and launching of the application
  - a vendor product is typically used in this tier to allow
    - time-based and interdependent scheduling of batch jobs
    - providing parallel processing capabilities
- Job Tier
  - responsible for the overall execution of a batch job
  - sequentially executes batch steps, ensuring that all steps are in the correct state and all appropriate policies are enforced
- Application Tier
  - contains components required to execute the program
  - contains specific modules that address the required batch functionality and enforces policies around a module execution (e.g., commit intervals, capture of statistics, etc.)
- Data Tier
  - provides the integration with the physical data sources that might include databases, files, or queues.