Spring Batch - Quick Guide

Previous Page Next Page

Spring Batch - Overview

Batch processing is a processing mode which involves execution of series of automated complex jobs without user interaction. A batch process handles bulk data and runs for a long time.

Several Enterprise applications require to process huge data to perform operations involving –

- Time-based events such as periodic calculations.
- Periodic applications that are processed repetitively over large datasets.
- Applications that deals with processing and validation of the data available in a transactional manner.

Therefore, batch processing is used in enterprise applications to perform such transactions.

What is Spring Batch

Spring batch is a **lightweight framework** which is used to develop **Batch Applications** that are used in Enterprise Applications.

In addition to bulk processing, this framework provides functions for –

- Including logging and tracing
- Transaction management
- Job processing statistics
- Job restart
- Skip and Resource management

You can also scale spring batch applications using its portioning techniques.

Features of Spring Batch

Following are the notable features of Spring Batch –

Flexibility – Spring Batch applications are flexible. You simply need to change an XML file to alter the order of processing in an application.

Maintainability – Spring Batch applications are easy to maintain. A Spring Batch job includes steps and each step can be decoupled, tested, and updated, without effecting the other steps.

Scalability – Using the portioning techniques, you can scale the Spring Batch applications. These techniques allow you to –

Execute the steps of a job in parallel.

Execute a single thread in parallel.

Reliability – In case of any failure, you can restart the job from exactly where it was stopped, by decoupling the steps.

Support for multiple file formats – Spring Batch provides support for a large set of readers and writers such as XML, Flat file, CSV, MYSQL, Hibernate, JDBC, Mongo, Neo4j, etc.

Multiple ways to launch a job – You can launch a Spring Batch job using web applications, Java programs, Command Line, etc.

In addition to these, Spring Batch applications support –

Automatic retry after failure.

Tracking status and statistics during the batch execution and after completing the batch processing.

To run concurrent jobs.

Services such as logging, resource management, skip, and restarting the processing.

Spring Batch - Environment

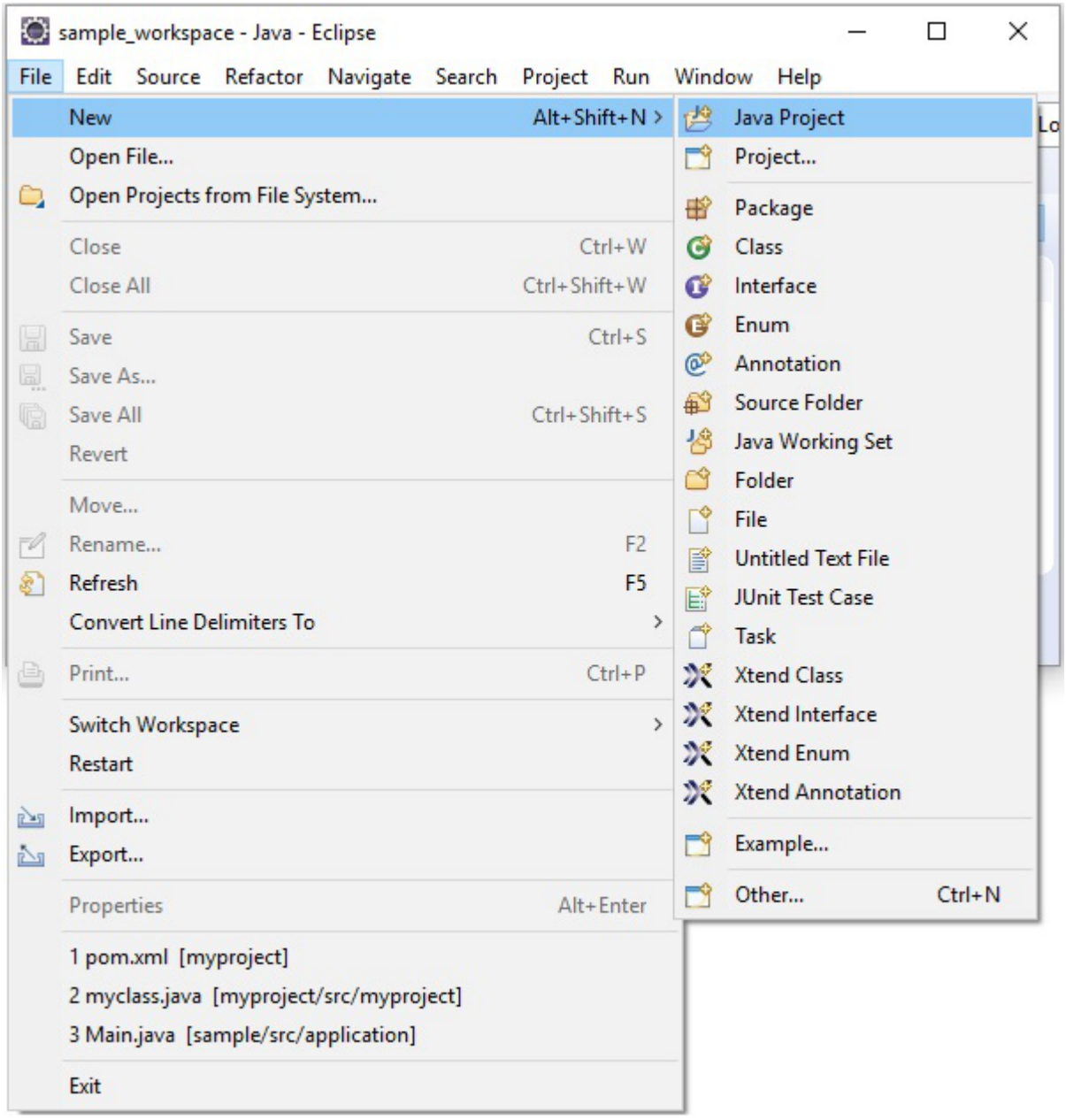
In this chapter, we will explain how to set Spring Batch environment in Eclipse IDE. Before proceeding with the installation, ensure that you have installed Eclipse in your system. If not, download and install Eclipse in your system.

For more information on Eclipse, please refer our Eclipse Tutorial.

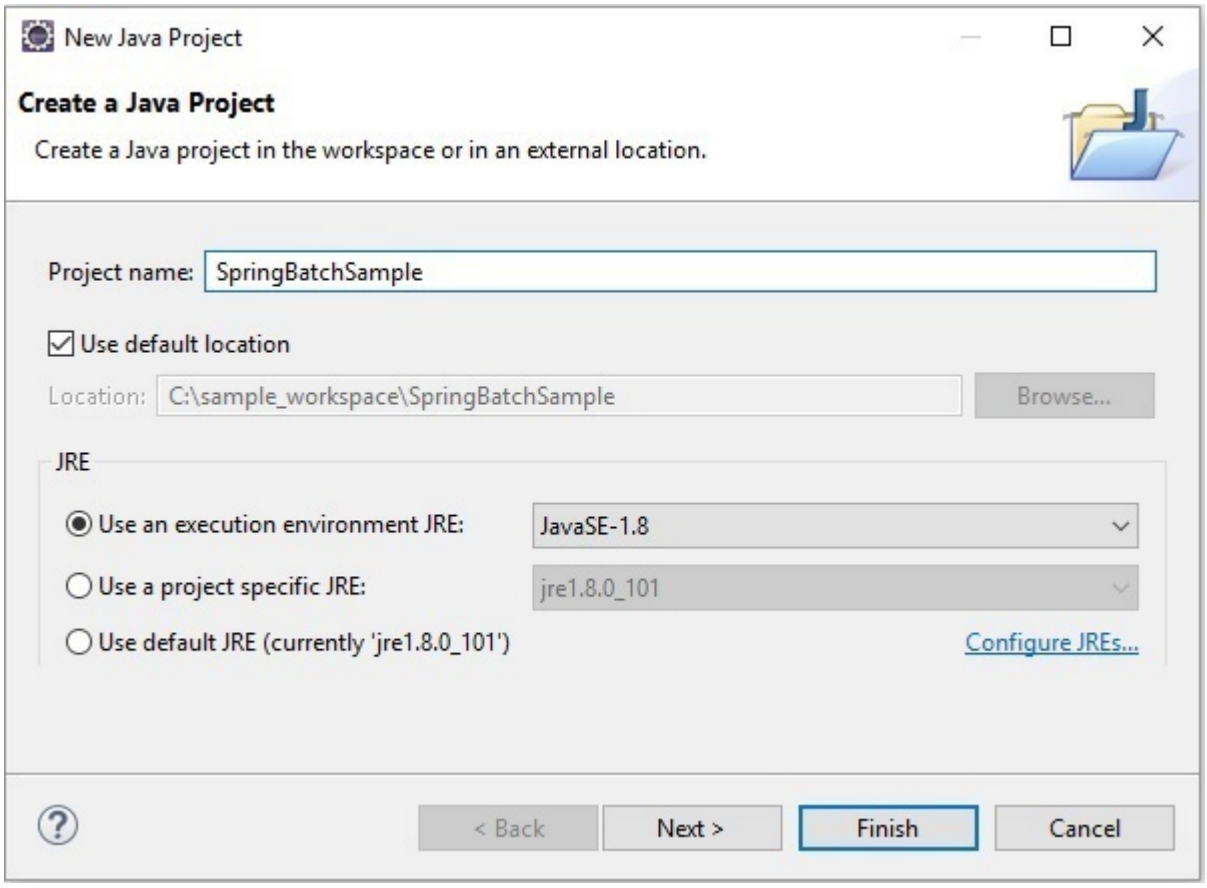
Setting Spring Batch on Eclipse

Follow the steps given below to set Spring Batch environment on Eclipse.

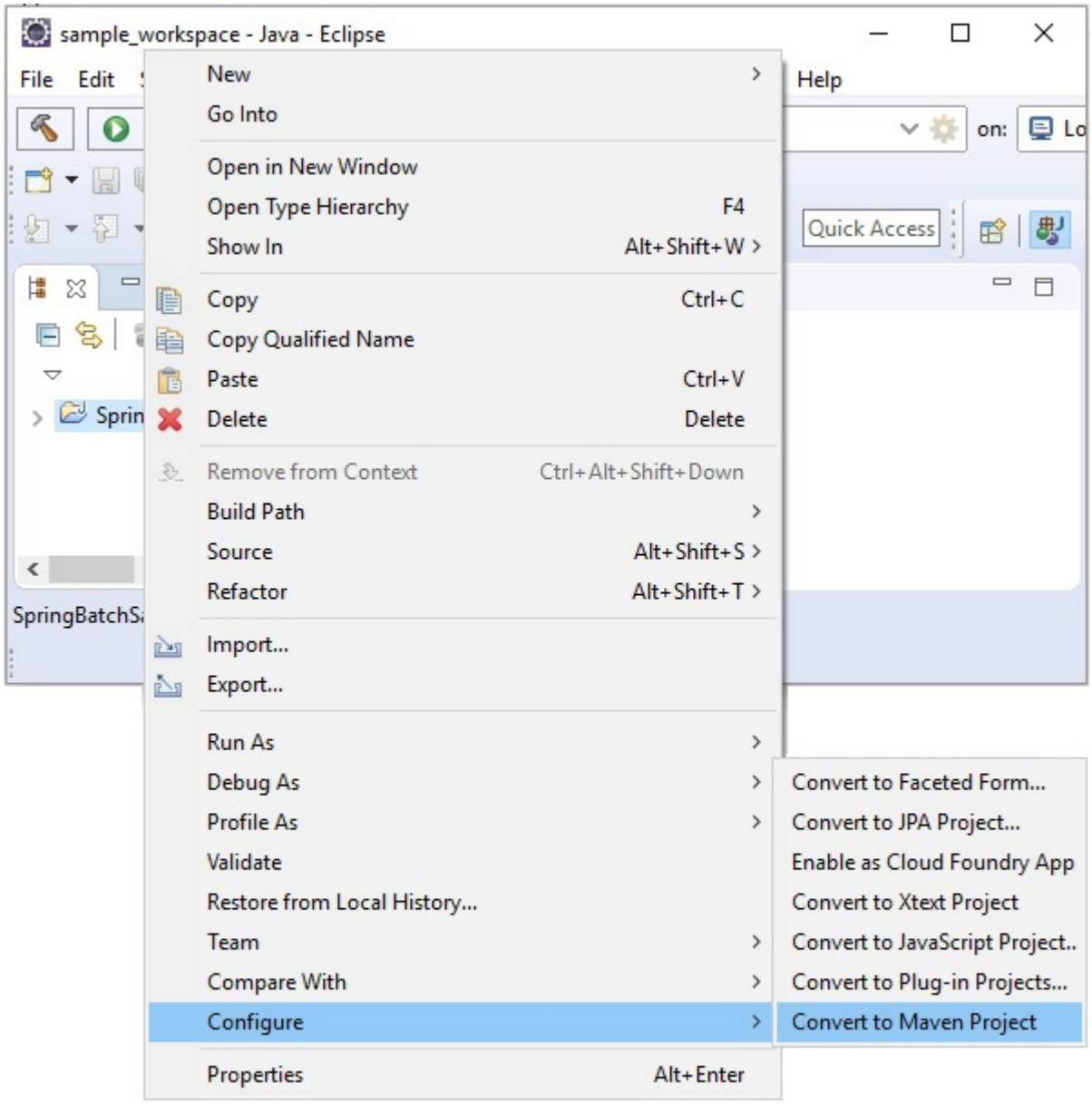
Step 1 – Install Eclipse and open a New Project as shown in the following screenshot.



Step 2 – Create a Sample Spring Batch project as shown below.



Step 3 – Right-click on the project and convert it into a Maven project as shown below. Once you convert it into Maven project, it will give you a **Pom.xml** where you need to mention the required dependencies. Thereafter, the **jar** files of those will be automatically downloaded into your project.



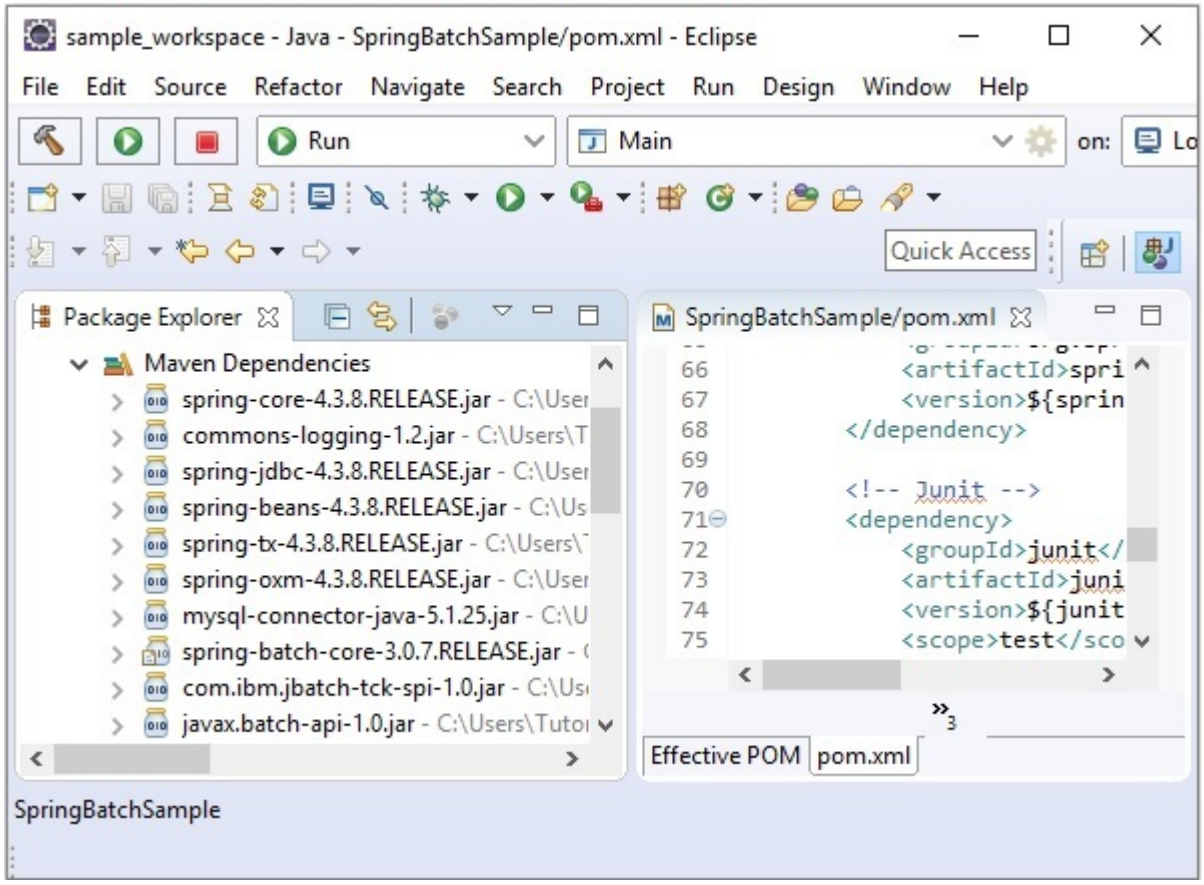
Step 4 – Now, in the **pom.xml** of the project, copy and paste the following content (dependencies for spring batch application) and refresh the project.

```
<project xmlns = "http://maven.apache.org/POM/4.0.0"
   xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation = "http://maven.apache.org/POM/4.0.0
   http://maven.apache.org/maven-v4_0_0.xsd">
   <modelVersion>4.0.0</modelVersion>
   <groupId>com.tutorialspoint</groupId>
   <artifactId>SpringBatchSample</artifactId>
   <packaging>jar</packaging>
   <version>1.0-SNAPSHOT</version>
   <name>SpringBatchExample</name>
   <url>http://maven.apache.org</url>

   <properties>
      <jdk.version>1.8</jdk.version>
      <spring.version>4.3.8.RELEASE</spring.version>
      <spring.batch.version>3.0.7.RELEASE</spring.batch.version>
      <mysql.driver.version>5.1.25</mysql.driver.version>
```

```
    <junit.version>4.11</junit.version>
  </properties>

  <dependencies>
    <!-- Spring Core -->
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-core</artifactId>
      <version>${spring.version}</version>
    </dependency>

    <!-- Spring jdbc, for database -->
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-jdbc</artifactId>
      <version>${spring.version}</version>
    </dependency>

    <!-- Spring XML to/back object -->
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-oxm</artifactId>
      <version>${spring.version}</version>
    </dependency>

    <!-- MySQL database driver -->
    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>${mysql.driver.version}</version>
    </dependency>

    <!-- Spring Batch dependencies -->
    <dependency>
      <groupId>org.springframework.batch</groupId>
      <artifactId>spring-batch-core</artifactId>
      <version>${spring.batch.version}</version>
    </dependency>

    <dependency>
      <groupId>org.springframework.batch</groupId>
      <artifactId>spring-batch-infrastructure</artifactId>
      <version>${spring.batch.version}</version>
    </dependency>

    <!-- Spring Batch unit test -->
    <dependency>
      <groupId>org.springframework.batch</groupId>
      <artifactId>spring-batch-test</artifactId>
      <version>${spring.batch.version}</version>
    </dependency>

    <!-- Junit -->
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>${junit.version}</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
```

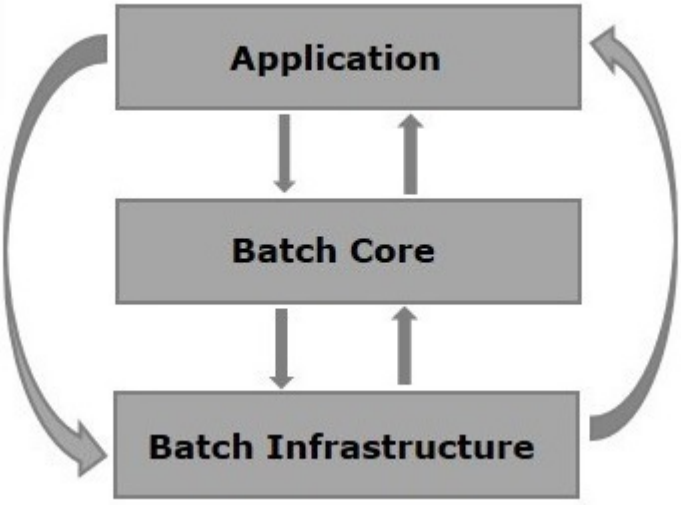
```
   <build>
      <finalName>spring-batch</finalName>
      <plugins>
         <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-eclipse-plugin</artifactId>
            <version>2.9</version>
            <configuration>
               <downloadSources>true</downloadSources>
               <downloadJavadocs>false</downloadJavadocs>
            </configuration>
         </plugin>

         <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>2.3.2</version>
            <configuration>
               <source>${jdk.version}</source>
               <target>${jdk.version}</target>
            </configuration>
         </plugin>
      </plugins>
   </build>
</project>
```

Finally, if you observe the Maven dependencies, you can observe that all the required **jar** files have been downloaded.



Spring Batch - Architecture

Following is the diagrammatic representation of the architecture of Spring Batch. As depicted in the figure, the architecture contains three main components namely, **Application, Batch Core**, and **Batch Infrastructure**.



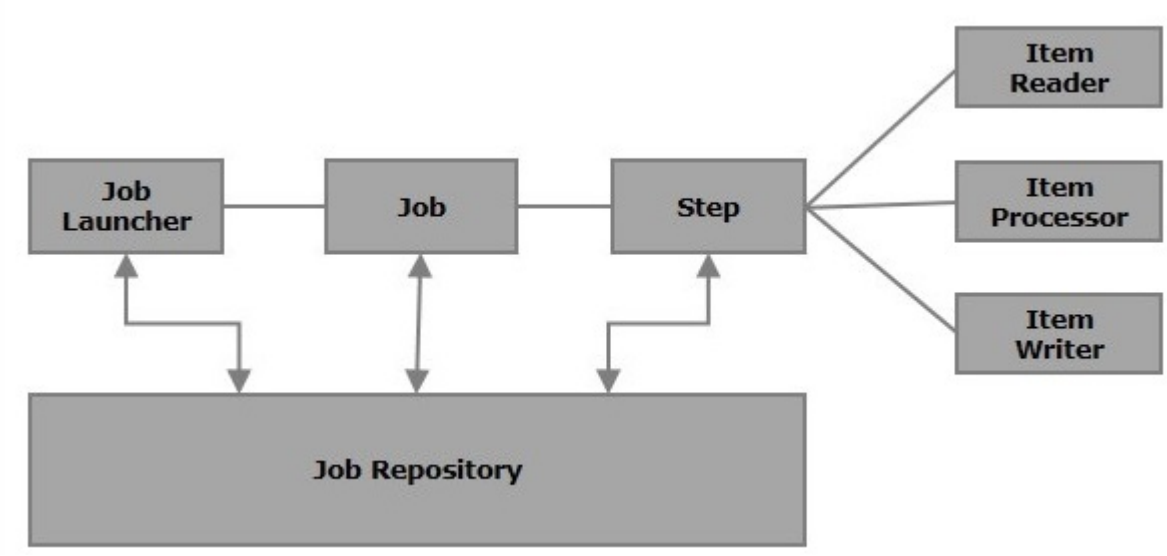
Application – This component contains all the jobs and the code we write using the Spring Batch framework.

Batch Core – This component contains all the API classes that are needed to control and launch a Batch Job.

Batch Infrastructure – This component contains the readers, writers, and services used by both application and Batch core components.

Components of Spring Batch

The following illustration shows the different components of Spring Batch and how they are connected with each other.



Job

In a Spring Batch application, a job is the batch process that is to be executed. It runs from start to finish without interruption. This job is further divided into steps (or a job contains steps).

We will configure a job in Spring Batch using an XML file or a Java class. Following is the XML configuration of a Job in Spring Batch.

```
<job id = "jobid">
   <step id = "step1" next = "step2"/>
   <step id = "step2" next = "step3"/>
   <step id = "step3"/>
</job>
```

A Batch job is configured within the tags <job></job>. It has an attribute named **id**. Within these tags, we define the definition and ordering of the steps.

Restartable – In general, when a job is running and we try to start it again that is considered as **restart** and it will be started again. To avoid this, you need to set the **restartable** value to **false** as shown below.

```
<job id = "jobid" restartable = "false" >

</job>
```

Step

A **step** is an independent part of a job which contains the necessary information to define and execute the job (its part).

As specified in the diagram, each step is composed of an ItemReader, ItemProcessor (optional) and an ItemWriter. **A job may contain one or more steps**.

Readers, Writers, and Processors

An **item reader** reads data into a Spring Batch application from a particular source, whereas an **item writer** writes data from the Spring Batch application to a particular destination.

An **Item processor** is a class which contains the processing code which processes the data read into the spring batch. If the application reads **"n"** records, then the code in the processor will be executed on each record.

When no reader and writer are given, a **tasklet** acts as a processor for SpringBatch. It processes only a single task. For example, if we are writing a job with a simple step in it where we read data from MySQL database and process it and write it to a file (flat), then our step uses –

A **reader** which reads from MySQL database.

A **writer** which writes to a flat file.

A **custom processor** which processes the data as per our wish.

```
<job id = "helloWorldJob">
   <step id = "step1">
      <tasklet>
         <chunk reader = "mysqlReader" writer = "fileWriter"
            processor = "CustomitemProcessor" ></chunk>
      </tasklet>
   </step>
</ job>
```

Spring Batch provides a long list of **readers** and **writers**. Using these predefined classes, we can define beans for them. We will discuss **readers** and **writers** in greater detail in the coming chapters.

JobRepository

A Job repository in Spring Batch provides Create, Retrieve, Update, and Delete (CRUD) operations for the JobLauncher, Job, and Step implementations. We will define a job repository in an XML file as shown below.

```
<job-repository id = "jobRepository"/>
```

In addition to **id**, there are some more options (optional) available. Following is the configuration of job repository with all the options and their default values.

```
<job-repository id = "jobRepository"
   data-source = "dataSource"
   transaction-manager = "transactionManager"
   isolation-level-for-create = "SERIALIZABLE"
   table-prefix = "BATCH_"
   max-varchar-length = "1000"/>
```

In-Memory Repository – In case you don't want to persist the domain objects of the Spring Batch in the database, you can configure the in-memory version of the jobRepository as shown below.

```
<bean id = "jobRepository"
   class = "org.springframework.batch.core.repository.support.MapJobRepositoryFactoryBean ">
   <property name = "transactionManager" ref = "transactionManager"/>
</bean>
```

JobLauncher

JobLauncher is an interface which launces the Spring Batch job with the **given set of parameters**. **SampleJoblauncher** is the class which implements the **JobLauncher** interface. Following is the configuration of the JobLauncher.

```
<bean id = "jobLauncher"
   class = "org.springframework.batch.core.launch.support.SimpleJobLauncher">
   <property name = "jobRepository" ref = "jobRepository" />
</bean>
```

JobInstance

A **JobIinstance** represents the logical run of a job; it is created when we run a job. Each job instance is differentiated by the name of the job and the parameters passed to it while running.

If a JobInstance execution fails, the same JobInstance can be executed again. Hence, each JobInstance can have multiple job executions.

JobExecution and StepExecution

JobExecution and StepExecution are the representation of the execution of a job/step. They contain the run information of the job/step such as start time (of job/step), end time (of job/step).

Spring Batch - Application

Almost all the examples in this tutorial contain the following files –

- Configuration file (XML file)
- Tasklet/processor (Java class)
- Java class with setters and getters (Java class (bean))
- Mapper class (Java class)
- Launcher class (Java class)

Configuration File

The configuration file (XML) contains the following –

- The **job** and **step** definitions.
- Beans defining **readers** and **writers**.
- Definition of components like JobLauncher, JobRepository, Transaction Manager, and Data Source.

In our examples, for better understanding, we have divided this in to two files the **job.xml** file (defines job, step, reader and writer) and **context.xml** file (job launcher, job repository, transaction manager and data source).

Mapper Class

The Mapper class, depending upon the reader, implements interfaces such as **row mapper**, **field set mapper**, etc. It contains the code to get the data from the reader and to set it to a Java class with **setter** and **getter** methods (Java Bean).

Java Bean Class

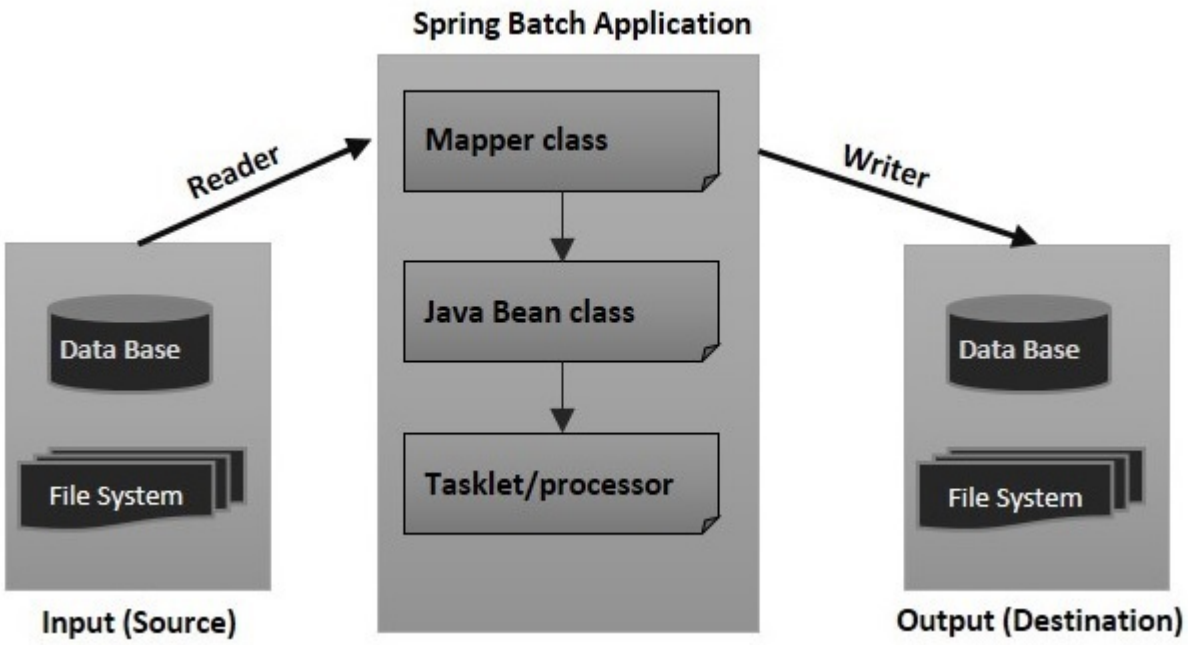
A Java class with **setters** and **getters** (Java bean) represents data with multiple values. It acts as a helper class. We will pass the data from one component (reader, writer, processer) to other in the form of object of this class.

Tasklet/processor

The Tasklet/processor class contains the processing code of the Spring Batch application. A processor is a class which accepts an object that contains the data read, processes it, and returns the processed data (in the form object).

Launcher class

This class (App.java) contains the code to launch the Spring Batch application.



Spring Batch - Configuration

While writing a Spring Batch application, we will configure the job, step, JobLauncher, JobRepository, Transaction Manager, readers, and writers using the XML tags provided in the Spring Batch namespace. Therefore, you need to include this namespace in your XML file as shown below.

```
<beans xmlns = "http://www.springframework.org/schema/beans"
   xmlns:batch = "http://www.springframework.org/schema/batch"
   xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation = "http://www.springframework.org/schema/batch

   http://www.springframework.org/schema/batch/spring-batch-2.2.xsd
```

```
   http://www.springframework.org/schema/bean
   http://www.springframework.org/schema/beans/spring-beans-3.2.xsd">
```

In the following sections, we will discuss the various tags, their attributes and examples, available in the Spring Batch namespace.

Job

This tag is used to define/configure the job of the SpringBatch. It contains a set of steps and it can be launched using the JobLauncher.

This tag has 2 attributes as listed below –

S.No	Attribute & Description
1	**Id** It is the Id of the job, it is mandatory to specify value to this attribute.
2	**restartable** This is the attribute which is used to specify whether the job is restartable or not. This attribute is optional.

Following is the XML configuration of the job of a SpringBatch.

```
<job id = "jobid" restartable = "false" >
   . . . . . .
   . . . . . .
   . . . . . . . . // Step definitions
</job>
```

Step

This tag is used to define/configure the steps of a SpringBatch job. It has the following three attributes –

S.No	Attribute & Description
1	**Id** It is the Id of the job, it is mandatory to specify value to this attribute.
2	**next** It is the shortcut to specify the next step.

3	**parent** It is used to specify the name of the parent bean from which the configuration should inherit.

Following is the XML configuration of the step of a SpringBatch.

```
<job id = "jobid">
   <step id = "step1" next = "step2"/>
   <step id = "step2" next = "step3"/>
   <step id = "step3"/>
</job>
```

Chunk

This tag is used to define/configure a chunk of a **tasklet**. It has the following four attributes –

S.No	Attribute & Description
1	**reader** It represents the name of the item reader bean. It accepts the value of the type **org.springframework.batch.item.ItemReader**.
2	**writer** It represents the name of the item reader bean. It accepts the value of the type **org.springframework.batch.item.ItemWriter**.
3	**processor** It represents the name of the item reader bean. It accepts the value of the type **org.springframework.batch.item.ItemProcessor**.
4	**commit-interval** It is used to specify the number of items to be processed before committing the transaction.

Following is the XML configuration of the chunk of a SpringBatch.

```
<batch:step id = "step1">
   <batch:tasklet>
      <batch:chunk reader = "xmlItemReader"
         writer = "mysqlItemWriter" processor = "itemProcessor" commit-interval = "10">
```

```
      </batch:chunk>
   </batch:tasklet>
</batch:step>
```

JobRepository

The JobRepository Bean is used to configure the JobRepository using a relational database. This bean is associated with the class of type **org.springframework.batch.core.repository.JobRepository**.

S.No	Attribute & Description
1	**dataSource** It is used to specify the bean name which defines the datasource.
2	**transactionManager** It is used specify the name of the bean which defines the transactionmanager.
3	**databaseType** It specifies the type of the relational database used in the job repository.

Following is the example configuration of the JobRepository.

```
<bean id = "jobRepository"
   class = "org.springframework.batch.core.repository.support.JobRepositoryFactoryBean">
   <property name = "dataSource" ref = "dataSource" />
   <property name = "transactionManager" ref="transactionManager" />
   <property name = "databaseType" value = "mysql" />
</bean>
```

JobLauncher

The JobLauncher bean is used to configure the JobLauncher. It is associated with the class **org.springframework.batch.core.launch.support.SimpleJobLauncher** (in our programs). This bean has one property named **jobrepository**, and it is used to specify the name of the bean which defines the **jobrepository**.

Following is the example configuration of the jobLauncher.

```
<bean id = "jobLauncher"
   class = "org.springframework.batch.core.launch.support.SimpleJobLauncher">
   <property name = "jobRepository" ref = "jobRepository" />
</bean>
```

TransactionManager

The TransactionManager bean is used to configure the TransactionManager using a relational database. This bean is associated with the class of type **org.springframework.transaction.platform.TransactionManager**.

```
<bean id = "transactionManager"
   class = "org.springframework.batch.support.transaction.ResourcelessTransactionManager" />
```

DataSource

The datasource bean is used to configure the **Datasource**. This bean is associated with the class of type **org.springframework.jdbc.datasource.DriverManagerDataSource**.

S.No	Attribute & Description
1	**driverClassName** This specifies the class name of the driver used to connect with the database.
2	**url** This specifies the URL of the database.
3	**username** This specifies the username to connect with the database.
4	**password** This specifies the password to connect with the database.

Following is the example configuration of the **datasource**.

```
<bean id = "dataSource"
   class = "org.springframework.jdbc.datasource.DriverManagerDataSource">
   <property name = "driverClassName" value = "com.mysql.jdbc.Driver" />
   <property name = "url" value = "jdbc:mysql://localhost:3306/details" />
   <property name = "username" value = "myuser" />
   <property name = "password" value = "password" />
</bean>
```

Spring Batch - Readers, Writers & Processors

An **Item Reader** reads data into the spring batch application from a particular source, whereas an **Item Writer** writes data from Spring Batch application to a particular

destination.

An **Item processor** is a class which contains the processing code which processes the data read in to the spring batch. If the application reads n records the code in the processor will be executed on each record.

A **chunk** is a child element of the **tasklet**. It is used to perform read, write, and processing operations. We can configure reader, writer, and processors using this element, within a step as shown below.

```
<batch:job id = "helloWorldJob">
   <batch:step id = "step1">
      <batch:tasklet>
         <batch:chunk reader = "cvsFileItemReader" writer = "xmlItemWriter"
            processor = "itemProcessor" commit-interval = "10">
         </batch:chunk>
      </batch:tasklet>
   </batch:step>
</batch:job>
```

Spring Batch provides readers and writers to read and write data form various file systems/databases such as MongoDB, Neo4j, MySQL, XML, flatfile, CSV, etc.

To include a reader in your application, you need to define a bean for that reader, provide values to all the required properties within the bean, and pass the **id** of such bean as a value to the attribute of the chunk element **reader** (same for **writer**).

ItemReader

It is the entity of a step (of a batch process) which reads data. An ItemReader reads one item a time. Spring Batch provides an Interface **ItemReader**. All the **readers** implement this interface.

Following are some of the predefined ItemReader classes provided by Spring Batch to read from various sources.

Reader	Purpose
FlatFIleItemReader	To read data from flat files.
StaxEventItemReader	To read data from XML files.
StoredProcedureItemReader	To read data from the stored procedures of a database.
JDBCPagingItemReader	To read data from relational databases database.
MongoItemReader	To read data from MongoDB.
Neo4jItemReader	To read data from Neo4jItemReader.

We need to configure the **ItemReaders** by creating the beans. Following is an example of **StaxEventItemReader** which reads data from an XML file.

```
<bean id = "mysqlItemWriter"
   class = "org.springframework.batch.item.xml.StaxEventItemWriter">
   <property name = "resource" value = "file:xml/outputs/userss.xml" />
   <property name = "marshaller" ref = "reportMarshaller" />
   <property name = "rootTagName" value = "Tutorial" />
</bean>

<bean id = "reportMarshaller"
   class = "org.springframework.oxm.jaxb.Jaxb2Marshaller">
   <property name = "classesToBeBound">
      <list>
         <value>Tutorial</value>
      </list>
   </property>
</bean>
```

As observed, while configuring, we need to specify the respective class name of the required reader and we need to provide values to all the required properties.

ItemWriter

It is the element of the **step** of a batch process which writes data. An ItemWriter writes one item a time. Spring Batch provides an Interface **ItemWriter**. All the writers implement this interface.

Following are some of the predefined ItemWriter classes provided by Spring Batch to read from various sources.

Writer	Purpose
FlatFIleItemWriter	To write data into flat files.
StaxEventItemWriter	To write data into XML files.
StoredProcedureItemWriter	To write data into the stored procedures of a database.
JDBCPagingItemWriter	To write data into relational databases database.
MongoItemWriter	To write data into MongoDB.
Neo4jItemWriter	To write data into Neo4j.

In same way, we need to configure the ItemWriters by creating the beans. Following is an example of **JdbcCursorItemReader** which writes data to an MySQL database.

```
<bean id = "dbItemReader"
   class = "org.springframework.batch.item.database.JdbcCursorItemReader" scope = "step">
   <property name = "dataSource" ref = "dataSource" />
```

```
   <property name = "sql" value = "select * from tutorialsdata" />
   <property name = "rowMapper">
      <bean class = "TutorialRowMapper" />
   </property>
</bean>
```

Item Processor

ItemProcessor: An ItemProcessor is used to process the data. When the given item is not valid it returns **null**, else it processes the given item and returns the processed result. The interface **ItemProcessor<I,O>** represents the processor.

Tasklet class – When no **reader** and **writer** are given, a Tasklet acts as a processor for SpringBatch. It processes only single task.

We can define a custom item processor by implementing the interface **ItemProcessor** of the package **org.springframework.batch.item.ItemProcessor**. This ItemProcessor class accepts an object and processes the data and returns the processed data as another object.

In a batch process, if **"n"** records or data elements are read, then for each record, it will read the data, process it, and writes the data in the writer. To process the data, it relays on the processor passed.

For example, let's suppose you have written code to load a particular PDF document, create a new page, write the data item on to the PDF in a tabular format. If you execute this application, it reads all the data items from the XML document, stores them in the MySQL database, and prints them in the given PDF document in individual pages.

Example

Following is a sample ItemProcessor class.

```
import org.springframework.batch.item.ItemProcessor;

public class CustomItemProcessor implements ItemProcessor<Tutorial, Tutorial> {

   @Override
   public Tutorial process(Tutorial item) throws Exception {
      System.out.println("Processing..." + item);
      return item;
   }
}
```

Spring Batch - Basic Application

This chapter shows you the basic Spring Batch application. It will simply execute a **tasklet** to displays a message.

Our Spring Batch application contains the following files –

Configuration file – This is an XML file where we define the Job and the steps of the job. (If the application involves readers and writers too, then the configuration of **readers** and **writers** is also included in this file.)

Context.xml – In this file, we will define the beans like job repository, job launcher and transaction manager.

Tasklet class – In this class, we will write the processing code job (In this case, it displays a simple message)

Launcher class – in this class, we will launch the Batch Application by running the Job launcher.

jobConfig.xml

Following is the configuration file of our sample Spring Batch application.

```
<beans xmlns = "http://www.springframework.org/schema/beans"
   xmlns:batch = "http://www.springframework.org/schema/batch"
   xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation = "http://www.springframework.org/schema/batch
      http://www.springframework.org/schema/batch/spring-batch-2.2.xsd
      http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans-3.2.xsd ">
   <import resource="context.xml" />
   <!-- Defining a bean -->
   <bean id = "tasklet" class = "a_sample.MyTasklet" />
   <!-- Defining a job-->
   <batch:job id = "helloWorldJob">
      <!-- Defining a Step -->
      <batch:step id = "step1">
         <tasklet ref = "tasklet"/>
      </batch:step>
   </batch:job>
</beans>
```

Context.xml

Following is the **context.xml** of our Spring Batch application.

```
<beans xmlns = "http://www.springframework.org/schema/beans"
   xmlns:xsi = http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation = "http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans-3.2.xsd">

   <bean id = "jobRepository"
      class="org.springframework.batch.core.repository.support.MapJobRepositoryFactoryBean">
      <property name = "transactionManager" ref = "transactionManager" />
   </bean>

   <bean id = "transactionManager"
      class = "org.springframework.batch.support.transaction.ResourcelessTransactionManager" />
   <bean id = "jobLauncher"
      class = "org.springframework.batch.core.launch.support.SimpleJobLauncher">
```

```
      <property name = "jobRepository" ref = "jobRepository" />
   </bean>
</beans>
```

Tasklet.java

Following is the Tasklet class which displays a simple message.

```
import org.springframework.batch.core.StepContribution;
import org.springframework.batch.core.scope.context.ChunkContext;
import org.springframework.batch.core.step.tasklet.Tasklet;
import org.springframework.batch.repeat.RepeatStatus;

public class MyTasklet implements Tasklet {

   @Override
   public RepeatStatus execute(StepContribution arg0, ChunkContext arg1) throws Exception {
      System.out.println("Hello This is a sample example of spring batch");
      return RepeatStatus.FINISHED;
   }
}
```

App.java

Following is the code which launces the batch process.

```
import org.springframework.batch.core.Job;
import org.springframework.batch.core.JobExecution;
import org.springframework.batch.core.JobParameters;
import org.springframework.batch.core.launch.JobLauncher;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class App {
   public static void main(String[] args)throws Exception {

      // System.out.println("hello");
      String[] springConfig  =  {"a_sample/job_hello_world.xml"};

      // Creating the application context object
      ApplicationContext context = new ClassPathXmlApplicationContext(springConfig);

      // Creating the job launcher
      JobLauncher jobLauncher = (JobLauncher) context.getBean("jobLauncher");

      // Creating the job
      Job job = (Job) context.getBean("helloWorldJob");

      // Executing the JOB
      JobExecution execution = jobLauncher.run(job, new JobParameters());
      System.out.println("Exit Status : " + execution.getStatus());
   }
}
```

On executing, the above SpringBatch program will produce the following output –

```
Apr 24, 2017 4:40:54 PM org.springframework.context.support.AbstractApplicationContext prepareRefresh
INFO:Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@2ef1e4fa: startup date
Apr 24, 2017 4:40:54 PM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions
Apr 24, 2017 4:40:54 PM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions
INFO: Loading XML bean definitions
Apr 24, 2017 4:40:54 PM org.springframework.beans.factory.support.DefaultListableBeanFactory preInstantia
Apr 24, 2017 4:40:55 PM org.springframework.batch.core.launch.support.SimpleJobLauncher afterPropertiesSe
INFO: No TaskExecutor has been set, defaulting to synchronous executor.
Apr 24, 2017 4:40:55 PM org.springframework.batch.core.launch.support.SimpleJobLauncher$1 run
INFO: Job: [FlowJob: [name=helloWorldJob]] launched with the following parameters: [{}]
Apr 24, 2017 4:40:55 PM org.springframework.batch.core.job.SimpleStepHandler handleStep INFO: Executing s
Hello This is a sample example of spring batch
Apr 24, 2017 4:40:55 PM org.springframework.batch.core.launch.support.SimpleJobLauncher$1 run
INFO: Job: [FlowJob: [name=helloWorldJob]] completed with the following parameters: [{}] and the followin
Exit Status : COMPLETED
```

Spring Batch - XML to MySQL

In this chapter, we will create a Spring Batch application which uses an XML Reader and a MySQL Writer.

Reader – The reader we are using in the application is **StaxEventItemReader** to read data from XML documents.

Following is the input XML document we are using in this application. This document holds data records which specify details like tutorial id, tutorial author, tutorial title, submission date, tutorial icon, and tutorial description.

```
<?xml version="1.0" encoding="UTF-8"?>
<tutorials>
   <tutorial>
      <tutorial_id>1001</tutorial_id>
      <tutorial_author>Sanjay</tutorial_author>
      <tutorial_title>Learn Java</tutorial_title>
      <submission_date>06-05-2007</submission_date>
      <tutorial_icon>https://www.tutorialspoint.com/java/images/java-minilogo.jpg</tutorial_icon>
      <tutorial_description>Java is a high-level programming language originally
         developed by Sun Microsystems and released in 1995.
         Java runs on a variety of platforms.
         This tutorial gives a complete understanding of Java.');</tutorial_description>
   </tutorial>

   <tutorial>
      <tutorial_id>1002</tutorial_id>
      <tutorial_author>Abdul S</tutorial_author>
      <tutorial_title>Learn MySQL</tutorial_title>
      <submission_date>19-04-2007</submission_date>
      <tutorial_icon>https://www.tutorialspoint.com/mysql/images/mysql-minilogo.jpg</tutorial_ico
      <tutorial_description>MySQL is the most popular
         Open Source Relational SQL database management system.
```

```
         MySQL is one of the best RDBMS being used for developing web-based software applications
         This tutorial will give you quick start with MySQL
         and make you comfortable with MySQL programming.</tutorial_description>
   </tutorial>

   <tutorial>
      <tutorial_id>1003</tutorial_id>
      <tutorial_author>Krishna Kasyap</tutorial_author>
      <tutorial_title>Learn JavaFX</tutorial_title>
      <submission_date>06-07-2017</submission_date>
      <tutorial_icon>https://www.tutorialspoint.com/javafx/images/javafx-minilogo.jpg</tutorial_i
      <tutorial_description>JavaFX is a Java library used to build Rich Internet Applications.
         The applications developed using JavaFX can run on various devices
         such as Desktop Computers, Mobile Phones, TVs, Tablets, etc.
         This tutorial, discusses all the necessary elements of JavaFX that are required
         to develop effective Rich Internet Applications</tutorial_description>
   </tutorial>
</tutorials>
```

Writer – The **writer** we are using in the application is **JdbcBatchItemWriter** to write the data to MySQL database. Assume we have created a table in MySQL inside a database called **"details"**.

```
CREATE TABLE details.TUTORIALS(
   tutorial_id int(10) NOT NULL,
   tutorial_author VARCHAR(20),
   tutorial_title VARCHAR(50),
   submission_date VARCHAR(20),
   tutorial_icon VARCHAR(200),
   tutorial_description VARCHAR(1000)
);
```

Processor – The processor we are using in the application is a custom processor which writes the data of each record on the PDF document.

In batch process, if **"n"** records or data elements were read, then for each record, it will read the data, process it, and write the data in the Writer. To process the data, it relays on the processor passed. In this case, in the custom processor class, we have written code to load a particular PDF document, create a new page, write the data item onto the PDF in a tabular format.

Finally, if you execute this application, it reads all the data items from the XML document, stores them in the MySQL database, and prints them in the given PDF document in individual pages.

jobConfig.xml

Following is the configuration file of our sample Spring Batch application. In this file, we will define the Job and the steps. In addition to these, we also define the beans for ItemReader, ItemProcessor, and ItemWriter. (Here, we associate them with their respective classes and pass the values for the required properties to configure them.)

```
<beans xmlns = "http://www.springframework.org/schema/beans"
   xmlns:batch = "http://www.springframework.org/schema/batch"
   xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
   xmlns:util = "http://www.springframework.org/schema/util"
   xsi:schemaLocation = "http://www.springframework.org/schema/batch

      http://www.springframework.org/schema/batch/spring-batch-2.2.xsd
      http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
      http://www.springframework.org/schema/util
      http://www.springframework.org/schema/util/spring-util-3.0.xsd ">

   <import resource = "../jobs/context.xml" />

   <bean id = "itemProcessor" class = "CustomItemProcessor" />
   <batch:job id = "helloWorldJob">
      <batch:step id = "step1">
         <batch:tasklet>
            <batch:chunk reader = "xmlItemReader" writer = "mysqlItemWriter" processor = "itemPro
            </batch:chunk>
         </batch:tasklet>
      </batch:step>
   </batch:job>

   <bean id = "xmlItemReader"
      class = "org.springframework.batch.item.xml.StaxEventItemReader">
      <property name = "fragmentRootElementName" value = "tutorial" />
      <property name = "resource" value = "classpath:resources/tutorial.xml" />
      <property name = "unmarshaller" ref = "customUnMarshaller" />
   </bean>

   <bean id = "customUnMarshaller" class = "org.springframework.oxm.xstream.XStreamMarshaller">
      <property name = "aliases">
         <util:map id = "aliases">
            <entry key = "tutorial" value = "Tutorial" />
         </util:map>
      </property>
   </bean>
   <bean id = "mysqlItemWriter" class = "org.springframework.batch.item.database.JdbcBatchItemWri
      <property name = "dataSource" ref = "dataSource" />
      <property name = "sql">
         <value>
            <![CDATA[insert into details.tutorials (tutorial_id, tutorial_author, tutorial_title,
               submission_date, tutorial_icon, tutorial_description)
               values (:tutorial_id, :tutorial_author, :tutorial_title, :submission_date,
               :tutorial_icon, :tutorial_description);]]>
         </value>
      </property>

      <property name = "itemSqlParameterSourceProvider">
         <bean class = "org.springframework.batch.item.database.BeanPropertyItemSqlParameterSourc
      </property>
   </bean>
</beans>
```

Context.xml

Following is the **context.xml** of our Spring Batch application. In this file, we will define the beans like job repository, job launcher, and transaction manager.

```
<beans xmlns = "http://www.springframework.org/schema/beans"
   xmlns:jdbc = "http://www.springframework.org/schema/jdbc"
   xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation = "http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
      http://www.springframework.org/schema/jdbc
      http://www.springframework.org/schema/jdbc/spring-jdbc-3.2.xsd">

   <!-- stored job-meta in database -->
   <bean id = "jobRepository"
      class = "org.springframework.batch.core.repository.support.JobRepositoryFactoryBean">
      <property name = "dataSource" ref = "dataSource" />
      <property name = "transactionManager" ref = "transactionManager" />
      <property name = "databaseType" value = "mysql" />
   </bean>

   <bean id = "transactionManager"
   class = "org.springframework.batch.support.transaction.ResourcelessTransactionMana ger" />
   <bean id = "jobLauncher"
      class = "org.springframework.batch.core.launch.support.SimpleJobLauncher">
      <property name = "jobRepository" ref = "jobRepository" />
   </bean>

   <!-- connect to MySQL database -->
   <bean id = "dataSource"
      class = "org.springframework.jdbc.datasource.DriverManagerDataSource">
      <property name = "driverClassName" value = "com.mysql.jdbc.Driver" />
      <property name = "url" value = "jdbc:mysql://localhost:3306/details" />
      <property name = "username" value = "myuser" />
      <property name = "password" value = "password" />
   </bean>

   <!-- create job-meta tables automatically -->
   <jdbc:initialize-database data-source = "dataSource">
      <jdbc:script location = "org/springframework/batch/core/schema-drop-mysql.sql"/>
      <jdbc:script location = "org/springframework/batch/core/schema-mysql.sql"/>
   </jdbc:initialize-database>
</beans>
```

CustomItemProcessor.java

Following is the **processor** class. In this class, we write the code of processing in the application. Here, we are loading a PDF document, creating a new page, creating a table, and inserting the following values for each record: tutorial id, tutorial name, author, date of submission in the table.

```
import java.io.File;
import java.io.IOException;

import org.apache.pdfbox.pdmodel.PDDocument;
import org.apache.pdfbox.pdmodel.PDPage;
import org.apache.pdfbox.pdmodel.PDPageContentStream;
import org.apache.pdfbox.pdmodel.font.PDType1Font;
```

```
import org.springframework.batch.item.ItemProcessor;

public class CustomItemProcessor implements ItemProcessor<Tutorial, Tutorial> {

   public static void drawTable(PDPage page, PDPageContentStream contentStream,
      float y, float margin, String[][] content) throws IOException {
      final int rows = content.length;
      final int cols = content[0].length;
      final float rowHeight = 50;
      final float tableWidth = page.getMediaBox().getWidth()-(2*margin);
      final float tableHeight = rowHeight * rows;
      final float colWidth = tableWidth/(float)cols;
      final float cellMargin=5f;

      // draw the rows
      float nexty = y ;
      for (int i = 0; i <= rows; i++) {
         contentStream.drawLine(margin,nexty,margin+tableWidth,nexty);
         nexty-= rowHeight;
      }

      //draw the columns
      float nextx = margin;
      for (int i = 0; i <= cols; i++) {
         contentStream.drawLine(nextx,y,nextx,y-tableHeight);
         nextx += colWidth;
      }

      // now add the text
      contentStream.setFont(PDType1Font.HELVETICA_BOLD,12);

      float textx = margin+cellMargin;
      float texty = y-15;
      for(int i = 0; i < content.length; i++){
         for(int j = 0 ; j < content[i].length; j++){
            String text = content[i][j];
            contentStream.beginText();
            contentStream.moveTextPositionByAmount(textx,texty);
            contentStream.drawString(text);
            contentStream.endText();
            textx += colWidth;
         }

         texty-=rowHeight;
         textx = margin+cellMargin;
      }
   }

   @Override
   public Tutorial process(Tutorial item) throws Exception {
      System.out.println("Processing..." + item);

      // Creating PDF document object
      PDDocument doc = PDDocument.load(new File("C:/Examples/test.pdf"));

      // Creating a blank page
      PDPage page = new PDPage();
      doc.addPage( page );
      PDPageContentStream contentStream =  new PDPageContentStream(doc, page);

      String[][] content = {{"Id",""+item.getTutorial_id()},
```

```
      {"Title", item.getTutorial_title()},
      {"Authour", item.getTutorial_author()},
      {"Submission Date", item.getSubmission_date()}} ;
      drawTable(page, contentStream, 700, 100, content);

      contentStream.close();
      doc.save("C:/Examples/test.pdf" );
      System.out.println("Hello");
      return item;
   }
}
```

TutorialFieldSetMapper.java

Following is the ReportFieldSetMapper class which sets the data to the Tutorial class.

```
import org.springframework.batch.item.file.mapping.FieldSetMapper;
import org.springframework.batch.item.file.transform.FieldSet;
import org.springframework.validation.BindException;

public class TutorialFieldSetMapper implements FieldSetMapper<Tutorial> {

   @Override
   public Tutorial mapFieldSet(FieldSet fieldSet) throws BindException {
      // instantiating the Tutorial class
      Tutorial tutorial = new Tutorial();

      // Setting the fields from XML
      tutorial.setTutorial_id(fieldSet.readInt(0));
      tutorial.setTutorial_title(fieldSet.readString(1));
      tutorial.setTutorial_author(fieldSet.readString(2));
      tutorial.setTutorial_icon(fieldSet.readString(3));
      tutorial.setTutorial_description(fieldSet.readString(4));
      return tutorial;
   }
}
```

Tutorial.java

Following is the **Tutorial** class. It is a simple class with **setter** and **getter** methods.

```
public class Tutorial {
   private int tutorial_id;
   private String tutorial_author;
   private String tutorial_title;
   private String submission_date;
   private String tutorial_icon;
   private String tutorial_description;

   @Override
   public String toString() {
      return " [id=" + tutorial_id + ", author=" + tutorial_author
         + ", title=" + tutorial_title + ", date=" + submission_date + ", icon ="
         +tutorial_icon +", description = "+tutorial_description+"]";
   }

   public int getTutorial_id() {
```

```
      return tutorial_id;
   }

   public void setTutorial_id(int tutorial_id) {
      this.tutorial_id = tutorial_id;
   }

   public String getTutorial_author() {
      return tutorial_author;
   }

   public void setTutorial_author(String tutorial_author) {
      this.tutorial_author = tutorial_author;
   }

   public String getTutorial_title() {
      return tutorial_title;
   }

   public void setTutorial_title(String tutorial_title) {
      this.tutorial_title = tutorial_title;
   }

   public String getSubmission_date() {
      return submission_date;
   }

   public void setSubmission_date(String submission_date) {
      this.submission_date = submission_date;
   }

   public String getTutorial_icon() {
      return tutorial_icon;
   }

   public void setTutorial_icon(String tutorial_icon) {
      this.tutorial_icon = tutorial_icon;
   }

   public String getTutorial_description() {
      return tutorial_description;
   }

   public void setTutorial_description(String tutorial_description) {
      this.tutorial_description = tutorial_description;
   }
}
```

App.java

Following is the code which launces the batch process. In this class, we will launch the Batch Application by running the JobLauncher.

```
public class App {
   public static void main(String[] args) throws Exception {
      String[] springConfig  = {    "jobs/job_hello_world.xml" };

      // Creating the application context object
```

```
      ApplicationContext context = new ClassPathXmlApplicationContext(springConfig);

      // Creating the job launcher
      JobLauncher jobLauncher = (JobLauncher) context.getBean("jobLauncher");

      // Creating the job
      Job job = (Job) context.getBean("helloWorldJob");

      // Executing the JOB
      JobExecution execution = jobLauncher.run(job, new JobParameters());
      System.out.println("Exit Status : " + execution.getStatus());
   }
}
```

On executing this application, it will produce the following output.

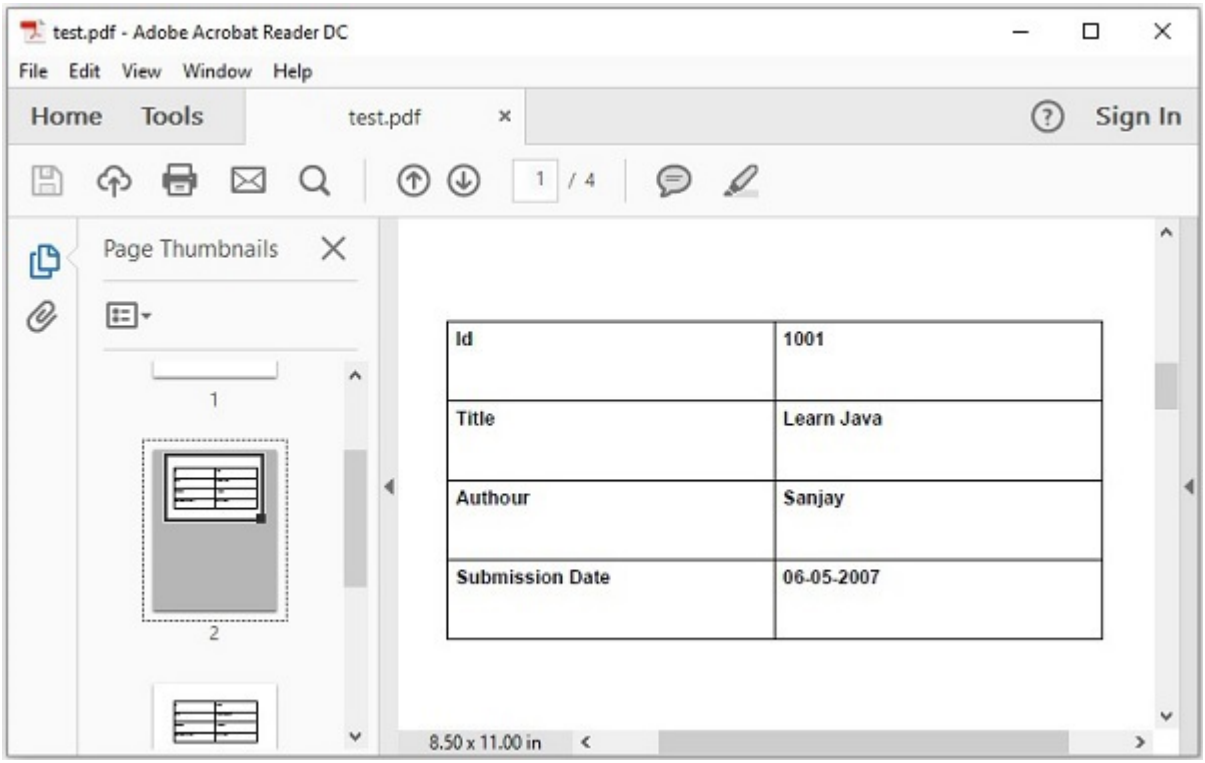
```
May 05, 2017 4:39:22 PM org.springframework.context.support.ClassPathXmlApplicationContext
prepareRefresh
INFO: Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@306a30c7:
startup date [Fri May 05 16:39:22 IST 2017]; root of context hierarchy
May 05, 2017 4:39:23 PM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions
May 05, 2017 4:39:32 PM org.springframework.batch.core.job.SimpleStepHandler handleStep
INFO: Executing step: [step1]
Processing... [id=1001, author=Sanjay, title=Learn Java, date=06-05-2007,
icon =https://www.tutorialspoint.com/java/images/java-mini-logo.jpg,
description = Java is a high-level programming language originally developed by Sun Microsystems
and released in 1995. Java runs on a variety of platforms.
This tutorial gives a complete understanding of Java.');]
Hello
Processing.. [id=1002, author=Abdul S, title=Learn MySQL, date=19-04-2007,
icon =https://www.tutorialspoint.com/mysql/images/mysql-mini-logo.jpg,
description = MySQL is the most popular Open Source Relational SQL database management system.
MySQL is one of the best RDBMS being used for developing web-based software applications.
This tutorial will give you quick start with MySQL and make you comfortable with MySQL programming.]
Hello
Processing... [id=1003, author=Krishna Kasyap, title=Learn JavaFX, date=06-072017,
icon =https://www.tutorialspoint.com/javafx/images/javafx-mini-logo.jpg,
description = JavaFX is a Java library used to build Rich Internet Applications.
The applications developed using JavaFX can run on various devices
such as Desktop Computers, Mobile Phones, TVs, Tablets, etc.
This tutorial, discusses all the necessary elements of JavaFX
that are required to develop effective Rich Internet Applications]
Hello
May 05, 2017 4:39:36 PM org.springframework.batch.core.launch.support.SimpleJobLauncher run
INFO: Job: [FlowJob: [name=helloWorldJob]] completed with the following parameters: [{}]
and the following status: [COMPLETED]
Exit Status : COMPLETED
```

If you verify the **details.tutorial** table in the database, it will show you the following output –

tutorial _id	tutorial _author	tutorial _title	submission _date	tutorial _icon	tutorial _description
1001	Sanjay	Learn Java	06-05-2007	**https://www.tutorialspoint.com /java/images/ java-mini-logo.jpg**	Java is a high-level programming language originally developed by Sun Microsystems and released in 1995. Java runs on a variety of platforms. This tutorial gives a complete understanding of Java.
1002	Abdul S	Learn MySQL	19-04-2007	**https://www. tutorialspoint.com /mysql/images /mysql-minilogo.jpg**	MySQL is the most popular Open Source Relational SQL database management system. MySQL is one of the best RDBMS being used for developing web-based software applications. This tutorial will give you quick start with MySQL and make you comfortable with MySQL programming.
1003	Learn JavaFX	Krishna Kasyap	06-07-2017	**https://www. tutorialspoint.com /javafx/images/ javafx-minilogo.jpg**	MySQL is the most popular Open Source Relational SQL database management system. MySQL is one of the best RDBMS being used for developing web-

					based software applications. This tutorial will give you quick start with MySQL and make you comfortable with MySQL programming.

This will generate a PDF with the records on each page as shown below.



Spring Batch - CSV to XML

In this chapter, we will create a simple Spring Batch application which uses a CSV Reader and an XML Writer.

Reader – The **reader** we are using in the application is **FlatFileItemReader** to read data from the CSV files.

Following is the input CSV file we are using in this application. This document holds data records which specify details like tutorial id, tutorial author, tutorial title, submission date, tutorial icon and tutorial description.

```
1001, "Sanjay", "Learn Java", 06/05/2007
1002, "Abdul S", "Learn MySQL", 19/04/2007
1003, "Krishna Kasyap", "Learn JavaFX", 06/07/2017
```

Writer – The Writer we are using in the application is **StaxEventItemWriter** to write the data to XML file.

Processor – The Processor we are using in the application is a custom processor which just prints the records read from the CSV file.

jobConfig.xml

Following is the configuration file of our sample Spring Batch application. In this file, we will define the Job and the steps. In addition to these, we also define the beans for ItemReader, ItemProcessor, and ItemWriter. (Here, we associate them with respective classes and pass the values for the required properties to configure them.)

```
<beans xmlns = " http://www.springframework.org/schema/beans"
   xmlns:batch = "http://www.springframework.org/schema/batch"
   xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation = "http://www.springframework.org/schema/batch
      http://www.springframework.org/schema/batch/spring-batch-2.2.xsd
      http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans-3.2.xsd">

   <import resource = "../jobs/context.xml" />

   <bean id = "report" class = "Report" scope = "prototype" />
   <bean id = "itemProcessor" class = "CustomItemProcessor" />

   <batch:job id = "helloWorldJob">

      <batch:step id = "step1">

         <batch:tasklet>
            <batch:chunk reader = "cvsFileItemReader" writer = "xmlItemWriter"
               processor = "itemProcessor" commit-interval = "10">
            </batch:chunk>
         </batch:tasklet>
      </batch:step>
   </batch:job>

   <bean id = "cvsFileItemReader"
      class = "org.springframework.batch.item.file.FlatFileItemReader">
      <property name = "resource" value = "classpath:resources/report.csv" />
      <property name = "lineMapper">
         <bean
            class = "org.springframework.batch.item.file.mapping.DefaultLineMapper">
            <property name = "lineTokenizer">
               <bean
                  class = "org.springframework.batch.item.file.transform.DelimitedLineTokenizer">
                  <property name = "names" value = "tutorial_id,
                     tutorial_author, Tutorial_title, submission_date" />
               </bean>
            </property>

            <property name = "fieldSetMapper">
               <bean class = "ReportFieldSetMapper" />
            </property>
         </bean>
```

```
      </property>
   </bean>

   <bean id = "xmlItemWriter"
      class = "org.springframework.batch.item.xml.StaxEventItemWriter">
      <property name = "resource" value = "file:xml/outputs/tutorials.xml" />
      <property name = "marshaller" ref = "reportMarshaller" />
      <property name = "rootTagName" value = "tutorials" />
   </bean>

   <bean id = "reportMarshaller"
      class = "org.springframework.oxm.jaxb.Jaxb2Marshaller">
      <property name = "classesToBeBound">
         <list>
            <value>Tutorial</value>
         </list>
      </property>
   </bean>
</beans>
```

Context.xml

Following is the **context.xml** of our Spring Batch application. In this file, we will define the beans like job repository, job launcher, and transaction manager.

```
<beans xmlns = "http://www.springframework.org/schema/beans"
   xmlns:jdbc = "http://www.springframework.org/schema/jdbc"
   xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation = "http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
      http://www.springframework.org/schema/jdbc
      http://www.springframework.org/schema/jdbc/spring-jdbc-3.2.xsd">
   <!-- stored job-meta in database -->
   <bean id = "jobRepository"
      class = "org.springframework.batch.core.repository.support.JobRepositoryFactoryBean">
      <property name = "dataSource" ref = "dataSource" />
      <property name = "transactionManager" ref = "transactionManager" />
      <property name = "databaseType" value = "mysql" />
   </bean>

   <bean id = "transactionManager"
      class = "org.springframework.batch.support.transaction.ResourcelessTransactionManager" />
   <bean id = "jobLauncher"
      class = "org.springframework.batch.core.launch.support.SimpleJobLauncher">
      <property name = "jobRepository" ref = "jobRepository" />
   </bean>

   <bean id = "dataSource" class = "org.springframework.jdbc.datasource.DriverManagerDataSource">
      <property name = "driverClassName" value = "com.mysql.jdbc.Driver" />
      <property name = "url" value = "jdbc:mysql://localhost:3306/details" />
      <property name = "username" value = "myuser" />
      <property name = "password" value = "password" />
   </bean>

   <!-- create job-meta tables automatically -->
   <jdbc:initialize-database data-source = "dataSource">
      <jdbc:script location = "org/springframework/batch/core/schema-drop-mysql.sql" />
```

```
      <jdbc:script location = "org/springframework/batch/core/schema-mysql.sql" />
   </jdbc:initialize-database>
</beans>
```

CustomItemProcessor.java

Following is the Processor class. In this class, we write the code of processing in the application. Here, we are printing the contents of each record.

```
import org.springframework.batch.item.ItemProcessor;

public class CustomItemProcessor implements ItemProcessor<Tutorial, Tutorial> {

   @Override
   public Tutorial process(Tutorial item) throws Exception {
      System.out.println("Processing..." + item);
      return item;
   }
}
```

TutorialFieldSetMapper.java

Following is the TutorialFieldSetMapper class which sets the data to the Tutorial class.

```
import org.springframework.batch.item.file.mapping.FieldSetMapper;
import org.springframework.batch.item.file.transform.FieldSet;
import org.springframework.validation.BindException;

public class TutorialFieldSetMapper implements FieldSetMapper<Tutorial> {

   @Override
   public Tutorial mapFieldSet(FieldSet fieldSet) throws BindException {

      //Instantiating the report object
      Tutorial tutorial = new Tutorial();

      //Setting the fields
      tutorial.setTutorial_id(fieldSet.readInt(0));
      tutorial.setTutorial_author(fieldSet.readString(1));
      tutorial.setTutorial_title(fieldSet.readString(2));
      tutorial.setSubmission_date(fieldSet.readString(3));

      return tutorial;
   }
}
```

Tutorial.java class

Following is the **Tutorial** class. It is a simple Java class with **setter** and **getter** methods. In this class, we are using annotations to associate the methods of this class with the tags of the XML file.

```java
import javax.xml.bind.annotation.XmlAttribute;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement(name = "tutorial")
public class Tutorial {
   private int tutorial_id;
   private String tutorial_author;
   private String tutorial_title;
   private String submission_date;

   @XmlAttribute(name = "tutorial_id")
   public int getTutorial_id() {
      return tutorial_id;
   }

   public void setTutorial_id(int tutorial_id) {
      this.tutorial_id = tutorial_id;
   }

   @XmlElement(name = "tutorial_author")
   public String getTutorial_author() {
      return tutorial_author;
   }
   public void setTutorial_author(String tutorial_author) {
      this.tutorial_author = tutorial_author;
   }

   @XmlElement(name = "tutorial_title")
   public String getTutorial_title() {
      return tutorial_title;
   }

   public void setTutorial_title(String tutorial_title) {
      this.tutorial_title = tutorial_title;
   }

   @XmlElement(name = "submission_date")
   public String getSubmission_date() {
      return submission_date;
   }

   public void setSubmission_date(String submission_date) {
      this.submission_date = submission_date;
   }

   @Override
   public String toString() {
      return "  [Tutorial id=" + tutorial_id + ",
         Tutorial Author=" + tutorial_author  + ",
         Tutorial Title=" + tutorial_title + ",
         Submission Date=" + submission_date + "]";
   }
}
```

App.java

Following is the code which launches the batch process. In this class, we will launch the batch application by running the JobLauncher.

```
import org.springframework.batch.core.Job;
import org.springframework.batch.core.JobExecution;
import org.springframework.batch.core.JobParameters;
import org.springframework.batch.core.launch.JobLauncher;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class App {
   public static void main(String[] args) throws Exception {

      String[] springConfig  =  { "jobs/job_hello_world.xml" };

      // Creating the application context object
      ApplicationContext context = new ClassPathXmlApplicationContext(springConfig);

      // Creating the job launcher
      JobLauncher jobLauncher = (JobLauncher) context.getBean("jobLauncher");

      // Creating the job
      Job job = (Job) context.getBean("helloWorldJob");

      // Executing the JOB
      JobExecution execution = jobLauncher.run(job, new JobParameters());
      System.out.println("Exit Status : " + execution.getStatus());
   }
}
```

On executing this application, it will produce the following output.

```
May 08, 2017 10:10:12 AM org.springframework.context.support.ClassPathXmlApplicationContext prepareRefres
INFO: Refreshing
org.springframework.context.support.ClassPathXmlApplicationContext@3d646c37: startup date
[Mon May 08 10:10:12 IST 2017]; root of context hierarchy
May 08, 2017 10:10:12 AM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinition
May 08, 2017 10:10:15 AM org.springframework.jdbc.datasource.init.ScriptUtils executeSqlScript
INFO: Executing step: [step1]
Processing...  [Tutorial id=1001, Tutorial Author=Sanjay,
Tutorial Title=Learn Java, Submission Date=06/05/2007]
Processing...  [Tutorial id=1002, Tutorial Author=Abdul S,
Tutorial Title=Learn MySQL, Submission Date=19/04/2007]
Processing...  [Tutorial id=1003, Tutorial Author=Krishna Kasyap,
Tutorial Title=Learn JavaFX, Submission Date=06/07/2017]
May 08, 2017 10:10:21 AM org.springframework.batch.core.launch.support.SimpleJobLauncher run
INFO: Job: [FlowJob: [name=helloWorldJob]] completed with the following parameters:
[{}] and the following status: [COMPLETED]
Exit Status : COMPLETED
```

This will generate an XML file with the following contents.

```
<?xml version = "1.0" encoding = "UTF-8"?>
<tutorials>
   <tutorial tutorial_id = "1001">
      <submission_date>06/05/2007</submission_date>
      <tutorial_author>Sanjay</tutorial_author>
      <tutorial_title>Learn Java</tutorial_title>
   </tutorial>

   <tutorial tutorial_id = "1002">
      <submission_date>19/04/2007</submission_date>
      <tutorial_author>Abdul S</tutorial_author>
      <tutorial_title>Learn MySQL</tutorial_title>
   </tutorial>

   <tutorial tutorial_id = "1003">
      <submission_date>06/07/2017</submission_date>
      <tutorial_author>Krishna Kasyap</tutorial_author>
      <tutorial_title>Learn JavaFX</tutorial_title>
   </tutorial>
</tutorials>
```

Spring Batch - MySQL to XML

In this chapter, we will create a Spring Batch application which uses a MySQL reader and an XML Writer.

Reader – The reader we are using in the application is **JdbcCursorItemReader** to read data from MySQL database.

Assume we have created a table in the MySQL database as shown below –

```
CREATE TABLE details.xml_mysql(
   person_id int(10) NOT NULL,
   sales VARCHAR(20),
   qty int(3),
   staffName VARCHAR(20),
   date VARCHAR(20)
);
```

Assume we have inserted the following records in to it.

```
mysql> select * from tutorialsdata;
+-------------+-----------------+---------------+-----------------+
| tutorial_id | tutorial_author | tutorial_title | submission_date |
+-------------+-----------------+---------------+-----------------+
|         101 | Sanjay          | Learn Java     | 06-05-2007      |
|         102 | Abdul S         | Learn MySQL    | 19-04-2007      |
|         103 | Krishna Kasyap  | Learn JavaFX   | 06-07-2017      |
+-------------+-----------------+---------------+-----------------+
3 rows in set (0.00 sec)
```

Writer – The Writer we are using in the application is **StaxEventItemWriter** to write the data to the XML file.

Processor – The Processor we are using in the application is a custom processor which just prints the records read from the CSV file.

jobConfig.xml

Following is the configuration file of our sample Spring Batch application. In this file, we will define the Job and the Steps. In addition to these, we also define the beans for ItemReader, ItemProcessor, and ItemWriter. (Here, we associate them with their respective classes and pass the values for the required properties to configure them.)

```
<beans xmlns = "http://www.springframework.org/schema/beans"
   xmlns:batch = "http://www.springframework.org/schema/batch"
   xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
   xmlns:util = "http://www.springframework.org/schema/util"
   xsi:schemaLocation = " http://www.springframework.org/schema/batch
      http://www.springframework.org/schema/batch/spring-batch-2.2.xsd
      http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans-3.2.xsd">

   <import resource = "../jobs/context.xml" />

   <bean id = "report" class = "Report" scope = "prototype" />
   <bean id = "itemProcessor" class = "CustomItemProcessor" />

   <batch:job id = "helloWorldJob">
      <batch:step id = "step1">
         <batch:tasklet>
            <batch:chunk reader = "dbItemReader"
               writer = "mysqlItemWriter" processor = "itemProcessor" commit-interval = "10">
            </batch:chunk>
         </batch:tasklet>
      </batch:step>
   </batch:job>

   <bean id = "dbItemReader"
      class = "org.springframework.batch.item.database.JdbcCursorItemReader" scope = "step">
      <property name = "dataSource" ref = "dataSource" />
      <property name = "sql" value = "select * from tutorials_data" />
      <property name = "rowMapper">
         <bean class = "TutorialRowMapper" />
      </property>
   </bean>
   <bean id = "mysqlItemWriter"
      class = "org.springframework.batch.item.xml.StaxEventItemWriter">
      <property name = "resource" value = "file:xml/outputs/tutorials.xml" />
      <property name = "marshaller" ref = "reportMarshaller" />
      <property name = "rootTagName" value = "Tutorial" />
   </bean>

   <bean id = "reportMarshaller" class = "org.springframework.oxm.jaxb.Jaxb2Marshaller">
      <property name = "classesToBeBound">
         <list>
            <value>Tutorial</value>
         </list>
      </property>
   </bean>
</beans>
```

Context.xml

Following is the **context.xml** of our Spring Batch application. In this file, we will define the beans like job repository, job launcher, and transaction manager.

```
<beans xmlns = " http://www.springframework.org/schema/beans"
   xmlns:jdbc = "http://www.springframework.org/schema/jdbc"
   xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation = "http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
      http://www.springframework.org/schema/jdbc
      http://www.springframework.org/schema/jdbc/spring-jdbc-3.2.xsd ">

   <!-- stored job-meta in database -->
   <bean id = "jobRepository"
      class = "org.springframework.batch.core.repository.support.JobRepositoryFactoryBean">
      <property name = "dataSource" ref = "dataSource" />
      <property name = "transactionManager" ref = "transactionManager" />
      <property name = "databaseType" value = "mysql" />
   </bean>

   <bean id = "transactionManager"
      class = "org.springframework.batch.support.transaction.ResourcelessTransactionMana ger" />
   <bean id = "jobLauncher"
      class = "org.springframework.batch.core.launch.support.SimpleJobLauncher">
      <property name = "jobRepository" ref = "jobRepository" />
   </bean>

   <!-- connect to MySQL database -->
   <bean id = "dataSource"
      class = "org.springframework.jdbc.datasource.DriverManagerDataSource">
      <property name = "driverClassName" value = "com.mysql.jdbc.Driver" />
      <property name = "url" value = "jdbc:mysql://localhost:3306/details" />
      <property name = "username" value = "myuser" />
      <property name = "password" value = "password" />
   </bean>

   <!-- create job-meta tables automatically -->
   <jdbc:initialize-database data-source = "dataSource">
      <jdbc:script location = "org/springframework/batch/core/schema-drop-mysql.sql" />
      <jdbc:script location = "org/springframework/batch/core/schema-mysql.sql" />
   </jdbc:initialize-database>
</beans>
```

CustomItemProcessor.java

Following is the Processor class. In this class, we write the code of processing in the application. Here, we are printing the contents of each record.

```
import org.springframework.batch.item.ItemProcessor;

public class CustomItemProcessor implements ItemProcessor<Tutorial, Tutorial> {
```

```
   @Override
   public Tutorial process(Tutorial item) throws Exception {
      System.out.println("Processing..." + item);
      return item;
   }
}
```

TutorialRowMapper.java

Following is the **TutorialRowMapper** class which sets the data to the **Tutorial** class.

```
import java.sql.ResultSet;
import java.sql.SQLException;
import org.springframework.jdbc.core.RowMapper;

public class TutorialRowMapper implements RowMapper<Tutorial> {

   @Override
   public Tutorial mapRow(ResultSet rs, int rowNum) throws SQLException {

      Tutorial tutorial = new Tutorial();
      tutorial.setTutorial_id(rs.getInt("tutorial_id"));
      tutorial.setTutorial_author(rs.getString("tutorial_author"));
      tutorial.setTutorial_title(rs.getString("tutorial_title"));
      tutorial.setSubmission_date(rs.getString("submission_date"));
      return tutorial;
   }
}
```

Tutorial.java

Following is the **Tutorial** class. It is a simple Java class with **setter** and **getter** methods. In this class, we are using annotations to associate the methods of this class with the tags of the XML file.

```
import javax.xml.bind.annotation.XmlAttribute;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement(name = "details")
public class Tutorial {

   int tutorial_id;
   String tutorial_author;
   String submission_date;

   @XmlAttribute(name = "tutorial_id")
   public int getTutorial_id() {
      return tutorial_id;
   }

   public void setTutorial_id(int tutorial_id) {
      this.tutorial_id = tutorial_id;
   }

   @XmlElement(name = "tutorial_author")
```

```
   public String getTutorial_author() {
      return tutorial_author;
   }

   public void setTutorial_author(String tutorial_author) {
      this.tutorial_author = tutorial_author;
   }

   @XmlElement(name = "tutorial_title")
   public String getTutorial_title() {
      return tutorial_title;
   }

   public void setTutorial_title(String tutorial_title) {
      this.tutorial_title = tutorial_title;
   }

   @XmlElement(name = "submission_date")
   public String getSubmission_date() {
      return submission_date;
   }

   public void setSubmission_date(String submission_date) {
      this.submission_date = submission_date;
   }

   public String toString() {
      return " [Tutorial Id=" + tutorial_id + ",
      Tutorial Author =" + tutorial_author  + ",
      Tutorial Title =" + tutorial_title + ",
      Submission Date =" + submission_date + "]";
   }
}
```

App.java

Following is the code which launces the batch process. In this class, we will launch the Batch Application by running the JobLauncher.

```
import org.springframework.batch.core.Job;
import org.springframework.batch.core.JobExecution;
import org.springframework.batch.core.JobParameters;
import org.springframework.batch.core.launch.JobLauncher;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class App {
   public static void main(String[] args) throws Exception {

      String[] springConfig  =  { "jobs/job_hello_world.xml" };

      // Creating the application context object
      ApplicationContext context = new ClassPathXmlApplicationContext(springConfig);

      // Creating the job launcher
      JobLauncher jobLauncher = (JobLauncher) context.getBean("jobLauncher");

      // Creating the job
```

```
      Job job = (Job) context.getBean("helloWorldJob");

      // Executing the JOB
      JobExecution execution = jobLauncher.run(job, new JobParameters());
      System.out.println("Exit Status : " + execution.getStatus());
   }
}
```

On executing this application, it will produce the following output.

```
May 08, 2017 11:32:06 AM org.springframework.context.support.ClassPathXmlApplicationContext prepareRefres
INFO: Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@3d646c37:
startup date [Mon May 08 11:32:06 IST 2017]; root of context hierarchy
May 08, 2017 11:32:06 AM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinition
INFO: Loading XML bean definitions from class path resource [jobs/job_hello_world.xml]
May 08, 2017 11:32:07 AM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinition
May 08, 2017 11:32:14 AM org.springframework.batch.core.job.SimpleStepHandler handleStep
INFO: Executing step: [step1]
Processing... [Tutorial Id=101, Tutorial Author=Sanjay,
Tutorial Title=Learn Java, Submission Date=06-05-2007]
Processing... [Tutorial Id=102, Tutorial Author=Abdul S,
Tutorial Title=Learn MySQL, Submission Date=19-04-2007]
Processing... [Tutorial Id=103, Tutorial Author=Krishna Kasyap,
Tutorial Title=Learn JavaFX, Submission Date=06-07-2017]
May 08, 2017 11:32:14 AM org.springframework.batch.core.launch.support.SimpleJobLauncher run
INFO: Job: [FlowJob: [name=helloWorldJob]] completed with the following parameters:
[{}] and the following status: [COMPLETED]
Exit Status : COMPLETED
```

This will generate an XML file with the following contents.

```
<?xml version = "1.0" encoding = "UTF-8"?>
<Tutorial>
   <details tutorial_id = "101">
      <submission_date>06-05-2007</submission_date>
      <tutorial_author>Sanjay</tutorial_author>
      <tutorial_title>Learn Java</tutorial_title>
   </details>

   <details tutorial_id = "102">
      <submission_date>19-04-2007</submission_date>
      <tutorial_author>Abdul S</tutorial_author>
      <tutorial_title>Learn MySQL</tutorial_title>
   </details>

   <details tutorial_id = "103">
      <submission_date>06-07-2017</submission_date>
      <tutorial_author>Krishna Kasyap</tutorial_author>
      <tutorial_title>Learn JavaFX</tutorial_title>
   </details>
</Tutorial>
```

Spring Batch - MySQL to Flat File

In this chapter, we will create a Spring Batch application which uses an MySQL Reader and a **Flatfile** Writer (.txt).

Reader – The Reader we are using in the application is **JdbcCursorItemReader** to read data from MySQL database.

Assume we have created a table in the MySQL database as shown below.

```
CREATE TABLE details.xml_mysql(
   person_id int(10) NOT NULL,
   sales VARCHAR(20),
   qty int(3),
   staffName VARCHAR(20),
   date VARCHAR(20)
);
```

Assume we have inserted the following records into it.

```
mysql> select * from tutorialsdata;
+-------------+-----------------+----------------+-----------------+
| tutorial_id | tutorial_author | tutorial_title | submission_date |
+-------------+-----------------+----------------+-----------------+
|         101 | Sanjay          | Learn Java     | 06-05-2007      |
|         102 | Abdul S         | Learn MySQL    | 19-04-2007      |
|         103 | Krishna Kasyap  | Learn JavaFX   | 06-07-2017      |
+-------------+-----------------+----------------+-----------------+
3 rows in set (0.00 sec)
```

Writer – The Writer we are using in the application is **FlatFileItemWriter** to write the data to **flatfile** (.txt).

Processor – The Processor we are using in the application is a custom processor which just prints the records read from the CSV file.

jobConfig.xml

Following is the configuration file of our sample Spring Batch application. In this file, we will define the Job and the Steps. In addition to these, we also define the beans for ItemReader, ItemProcessor, and ItemWriter. (Here, we associate them with respective classes and pass the values for the required properties to configure them.)

```
<beans xmlns = "http://www.springframework.org/schema/beans"
   xmlns:batch = "http://www.springframework.org/schema/batch"
   xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
   xmlns:util = "http://www.springframework.org/schema/util"
   xsi:schemaLocation = "http://www.springframework.org/schema/batch

      http://www.springframework.org/schema/batch/spring-batch-2.2.xsd
```

```
      http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans-3.2.xsd">

   <import resource = "../jobs/context.xml" />
   <bean id = "tutorial" class = "Tutorial" scope = "prototype" />
   <bean id = "itemProcessor" class = "CustomItemProcessor" />

   <batch:job id = "helloWorldJob">
      <batch:step id = "step1">
         <batch:tasklet>
            <batch:chunk reader = "mysqlItemReader"
               writer = "flatFileItemWriter" processor = "itemProcessor"
               commit-interval = "10">
            </batch:chunk>
         </batch:tasklet>
      </batch:step>
   </batch:job>

   <bean id = "mysqlItemReader"
      class = "org.springframework.batch.item.database.JdbcCursorItemReader" >
      <property name = "dataSource" ref = "dataSource" />
      <property name = "sql" value = "select * from details.tutorialsdata" />
      <property name = "rowMapper">
         <bean class = "TutorialRowMapper" />
      </property>
   </bean>

   <bean id = "flatFileItemWriter"
      class = " org.springframework.batch.item.file.FlatFileItemWriter">
      <property name = "resource" value = "file:target/outputfiles/employee_output.txt"/>
      <property name = "lineAggregator">
         <bean class = " org.springframework.batch.item.file.transform.PassThroughLineAggregator"
      </property>
   </bean>
</beans>
```

Context.xml

Following is the **context.xml** of our Spring Batch application. In this file, we will define the beans like job repository, job launcher, and transaction manager.

```
<beans xmlns = "http://www.springframework.org/schema/beans"
   xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
   xmlns:jdbc = "http://www.springframework.org/schema/jdbc"
   xsi:schemaLocation = "http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
      http://www.springframework.org/schema/jdbc
      http://www.springframework.org/schema/jdbc/spring-jdbc-3.2.xsd ">

   <!-- stored job-meta in database -->
   <bean id = "jobRepository"
      class = "org.springframework.batch.core.repository.support.JobRepositoryFactoryBean">
      <property name = "dataSource" ref = "dataSource" />
      <property name = "transactionManager" ref = "transactionManager" />
      <property name = "databaseType" value = "mysql" />
   </bean>
```

```
   <bean id = "transactionManager"
      class = "org.springframework.batch.support.transaction.ResourcelessTransactionManager" />

   <bean id = "dataSource"
      class = "org.springframework.jdbc.datasource.DriverManagerDataSource">
      <property name = "driverClassName" value = "com.mysql.jdbc.Driver" />
      <property name = "url" value = "jdbc:mysql://localhost:3306/details" />
      <property name = "username" value = "myuser" />
      <property name = "password" value = "password" />
   </bean>

   <bean id = "jobLauncher"
      class = "org.springframework.batch.core.launch.support.SimpleJobLauncher">
      <property name = "jobRepository" ref = "jobRepository" />
   </bean>

   <!-- create job-meta tables automatically -->
   <jdbc:initialize-database data-source = "dataSource">
      <jdbc:script location = "org/springframework/batch/core/schema-drop-mysql.sql" />
      <jdbc:script location = "org/springframework/batch/core/schema-mysql.sql" />
   </jdbc:initialize-database>
</beans>
```

CustomItemProcessor.java

Following is the Processor class. In this class, we write the code of processing in the application. Here, we are printing the contents of each record.

```
import org.springframework.batch.item.ItemProcessor;

// Implementing the ItemProcessor interface
public class CustomItemProcessor implements ItemProcessor<Tutorial, Tutorial> {

   @Override
   public Tutorial process(Tutorial item) throws Exception {
      System.out.println("Processing..." + item);
      return item;
   }
}
```

TutorialRowMapper.java

Following is the **TutorialRowMapper** class which sets the data to the **Tutorial** class.

```
public class TutorialRowMapper implements RowMapper<Tutorial> {

   @Override
   public Tutorial mapRow(ResultSet rs, int rowNum) throws SQLException {

      Tutorial tutorial = new Tutorial();

      tutorial.setTutorial_id(rs.getInt("tutorial_id"));
      tutorial.setTutorial_title(rs.getString("tutorial_title"));
      tutorial.setTutorial_author(rs.getString("tutorial_author"));
      tutorial.setSubmission_date(rs.getString("submission_date"));
```

```
      return tutorial;
   }
}
```

Tutorial.java

Following is the **Tutorial** class. It is a simple Java class with **setter** and **getter** methods. In this class, we are using annotations to associate the methods of this class with the tags of the XML file.

```
public class Tutorial {
   private int tutorial_id;
   private String tutorial_title;
   private String tutorial_author;
   private String submission_date;

   public int getTutorial_id() {
      return tutorial_id;
   }

   public void setTutorial_id(int tutorial_id) {
      this.tutorial_id = tutorial_id;
   }

   public String getTutorial_title() {
      return tutorial_title;
   }

   public void setTutorial_title(String tutorial_title) {
      this.tutorial_title = tutorial_title;
   }

   public String getTutorial_author() {
      return tutorial_author;
   }

   public void setTutorial_author(String tutorial_author) {
      this.tutorial_author = tutorial_author;
   }

   public String getSubmission_date() {
      return submission_date;
   }
   public void setSubmission_date(String submission_date) {
      this.submission_date = submission_date;
   }

   @Override
   public String toString() {
      return " [id=" + tutorial_id + ", title=" +
      tutorial_title                        + ",
      author=" + tutorial_author + ", date=" +
      submission_date + "]";
   }
}
```

App.java

Following is the code which launces the batch process. In this class, we will launch the Batch Application by running the JobLauncher.

```
import org.springframework.batch.core.Job;
import org.springframework.batch.core.JobExecution;
import org.springframework.batch.core.JobParameters;
import org.springframework.batch.core.launch.JobLauncher;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class App {

   public static void main(String[] args) throws Exception {

      String[] springConfig  =  { "jobs/job_hello_world.xml" };

      // Creating the application context object
      ApplicationContext context = new ClassPathXmlApplicationContext(springConfig);

      // Creating the job launcher
      JobLauncher jobLauncher = (JobLauncher) context.getBean("jobLauncher");

      // Creating the job
      Job job = (Job) context.getBean("helloWorldJob");

      // Executing the JOB
      JobExecution execution = jobLauncher.run(job, new JobParameters());
      System.out.println("Exit Status : " + execution.getStatus());
   }
}
```

On executing this application, it will produce the following output.

```
May 09, 2017 5:44:48 PM org.springframework.context.support.ClassPathXmlApplicationContext prepareRefresh
INFO: Refreshing org.springframework.context.support.ClassPathXml
ApplicationContext@3d646c37: startup date [Tue May
09 17:44:48 IST 2017]; root of context hierarchy
May 09, 2017 5:44:48 PM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions
May 09, 2017 5:44:56 PM org.springframework.batch.core.launch.support.SimpleJobLauncher run
INFO: Job: [FlowJob: [name=helloWorldJob]] launched
with the following parameters: [{}]
May 09, 2017 5:44:56 PM org.springframework.batch.core.job.SimpleStepHandler handleStep
INFO: Executing step: [step1]
Processing...Report [id=101, title=Learn Java, author=Sanjay, date=06-05-2007]
Processing...Report [id=102, title=Learn MySQL, author=Abdul S, date=19-04-2007]
Processing...Report [id=103, title=Learn JavaFX, author=Krishna Kasyap, date=0607-2017]
May 09, 2017 5:44:57 PM org.springframework.batch.core.launch.support.SimpleJobLauncher run
INFO: Job: [FlowJob: [name=helloWorldJob]] completed with the following parameters:
[{}] and the following status: [COMPLETED]
```

```
Hello
Exit Status : COMPLETED
```

This will generate a **.txt** file with the following contents.

```
Report [id=101, title=Learn Java, author=Sanjay, date=06-05-2007]
Report [id=102, title=Learn MySQL, author=Abdul S, date=19-04-2007]
Report [id=103, title=Learn JavaFX, author=Krishna Kasyap, date=06-07-2017]
```

Previous Page Next Page

Enter email for newsletter go