



INDEX

Spring Boot Batch -----

1. Introduction [04](#)
2. Spring Batch Launch Environment [10](#)
 - a. POC (Proof of Concepts on Spring Batch processing) [18](#)
3. Story board of CSV file to DB table conversion using Spring Batch [20](#)
4. Spring Batch application converting DB table records into CSV file [42](#)
5. Spring Batch application to convert CSV file data to MongoDB documents [55](#)
6. How to enable Cron expression scheduling on Batch processing [66](#)
7. Converting CSV file data to Oracle DB table records using Spring Data JPA (Hibernate) [68](#)

Spring Boot Batch

Introduction

- ✚ Spring Batch (or) Spring Batch Processing (or) Spring Boot Batch.
- ✚ Spring Batch is given extension module of Spring framework by pivotal team (Spring team) + Accenture

Q. What Batch Processing?

Ans.

- ✚ It is the process handling huge amount of data (chunks of data) batch by batch by reading the from one source and processing data with changes by executing logics and writing data to another Destination.
- ✚ Spring Batch + Spring Scheduling is great combination to perform batching activities in automated environment either on "PERIOD OF TIME" or "POINT OF TIME".

Use cases:

- Collecting Senses Data (population count) and storing into DB s/w.
teacher collects family info on paper --> feeds to excel sheets --> uploads excel sheets Govt App --> Govt App validates the data, filters data, categorize data --> Writes to DB s/w.
- Collecting Log Messages from ATM Machines --> categorizing them --> writing to DB s/w.
- Collecting Insurance policy holders of LIC from DB s/w --> identifying the Policy matured people --> Sending that info to branches accordingly.
- HDFC collecting Loan Holders info from DB s/w every month --> processing and sending EMI Payment Remainder messages/ mails.
- Collecting Bank Loan Customers Info --> finding Defaulters and sending notices them.
and etc.

While doing batch processing

- Collect data from different sources in different formats.
- Process Data for filtering data or modifying data or categorizing data or ordering data and etc.
- Write data to different destinations in different formats.
- Converting DB data to XML data after processing.
- Converting one DB s/w data to another DB s/w data after processing.
- Converting JSON Data to CSV (comma separate data)/ Excel Data after processing.
- Converting DB s/w data to JSON data after processing and etc.

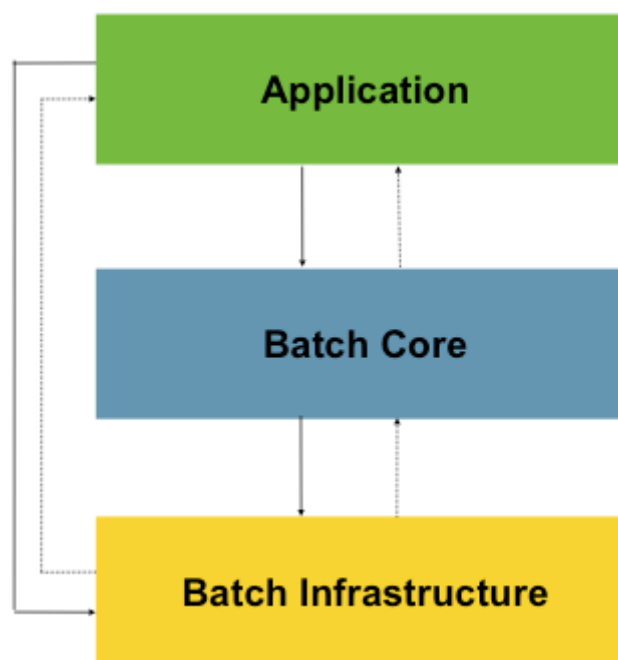
(All conversions are possible in all angels)

- ✚ If Data of the Project is MBs and GBs, then store data in DB s/w and manipulate it by using JDBC or Spring JDBC or Spring Data JPA or Spring ORM or Hibernate.
E.g., College or university Students Info
- ✚ If Data of the Project is GBs and TBs, then store data in DB s/w and manipulate it through Batch processing (Batch by batch processing) with the support of plain JDBC or Spring JDBC or Spring Batch (best).
E.g., Bank Data or Financial organization data
- ✚ If the data of the Project is TBs, PBs, XBs, YBs and etc. (very huge beyond storing and processing capacity of DB s/w - which is Big Data) then store and process that data using Bigdata frameworks (like Hadoop or spark).
E.g., Facebook, Google, Twitter, YouTube video's location data, Google maps and etc.

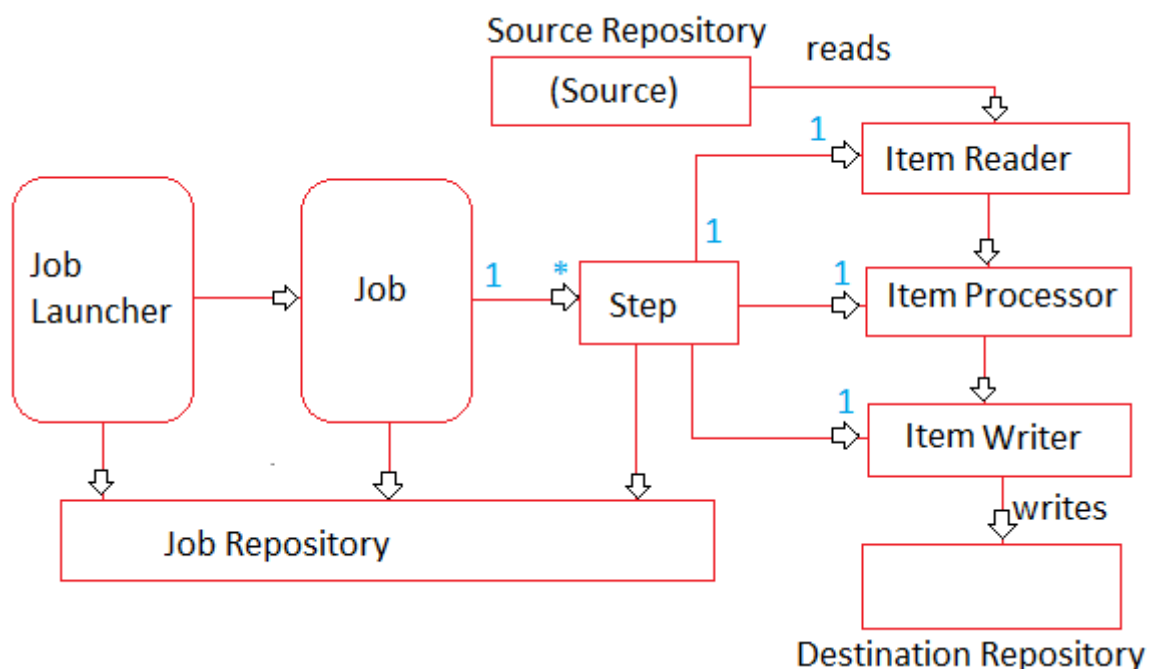
Note:

- ✓ In All these places we can use NoSQL DB s/w if schema/format of data is unstructured and growing dynamically. (MongoDB, Cassandra, GraphQL and etc.).
- ✓ FB kind of apps uses multiple storage technologies at a time in a single project.

Spring Batch high level architecture



The Basic look of Spring Batch application



- Job Launcher launches the Job.
- Every Job can contain 1 or more steps
- Every Step contains 1 ItemReader, 1 ItemWriter and 1 ItemProcessor
- ItemReader reads each Item (primitive/ object) from Source repository and gives that item to ItemProcessor for processing (filtering, calculating and sorting, etc.) and at the end processed items will be written to destination repository using ItemWriter.
- All activities Job launcher, Job, Step will be kept tracked by Job repository which is InMemory repository by default.
- org.springframework.batch.item package
 - All Item Readers are implementation classes of `ItemReader<T> (I)`
 - All Item Processors are implementation classes of `ItemProcessor<I, O> (I)`
 - All Item Writers are implementation classes of `ItemWriter<T> (I)`
- Generally, we work with Pre-defined ItemWriter classes and pre-defined ItemReader because they are designed to deal with different types of source/ destination repositories and with different types of data formats. But we always develop ItemProcessor manually.

Pre-defined ItemReader

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.
And etc..	

Pre-defined ItemWriter

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.

org.sf.batch.item.ItemReader<T> (I)

|--> <T> read() throws java.lang.Exception, UnexpectedInputException, ParseException, NonTransientResourceException

org.sf.batch.item.ItemWriter<T> (I)

|--> void write(java.util.List<? extends T> items)

org.sf.batch.item.ItemProcessor<I, O> (I)

|--> O process(@NonNull I item) throws java.lang.Exception

Note:

- ✓ The <T> of ItemReader must match with <I> of ItemProcessor and similarly the <O> of ItemProcessor must match with <T> ItemWriter.
- ✓ The <I> and <O> ItemProcessor can be same or can be different.

✚ After Processing the received item from ItemReader the ItemWriter can give either same object with modified data/ unmodified data or different object with same data or modified data to Destination.

Use case 1:

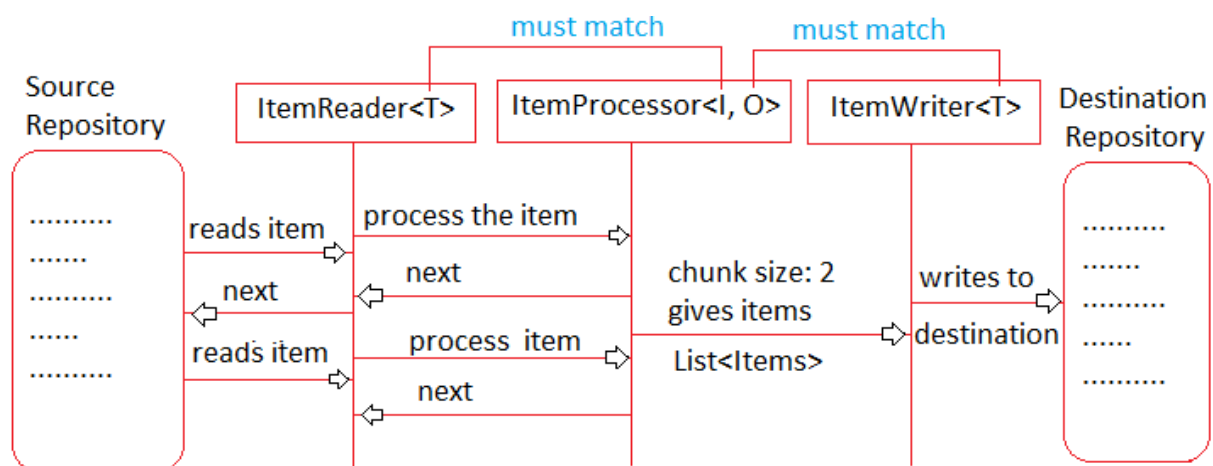
- ItemReaders gets Student object from source (college DB).

- ItemProcessor conducts interview and converts eligible Student object Employee object.
- ItemWriter gets List <Employee> objects to write to destination (Company DB).

Use case 2:

- ItemReader gets Policy Holder object from source (LIC DB).
- ItemProcessor checks payment schedule of policy holder is there in the current month or not if not there give nothing to ItemWriter otherwise gives same Policyholder object to ItemWriter.
- ItemWriter writes of List<PolicyHolder> objects to write to destination (Excel sheet/ another DB table)

Sequence Flow of diagram of Step



Step flow:

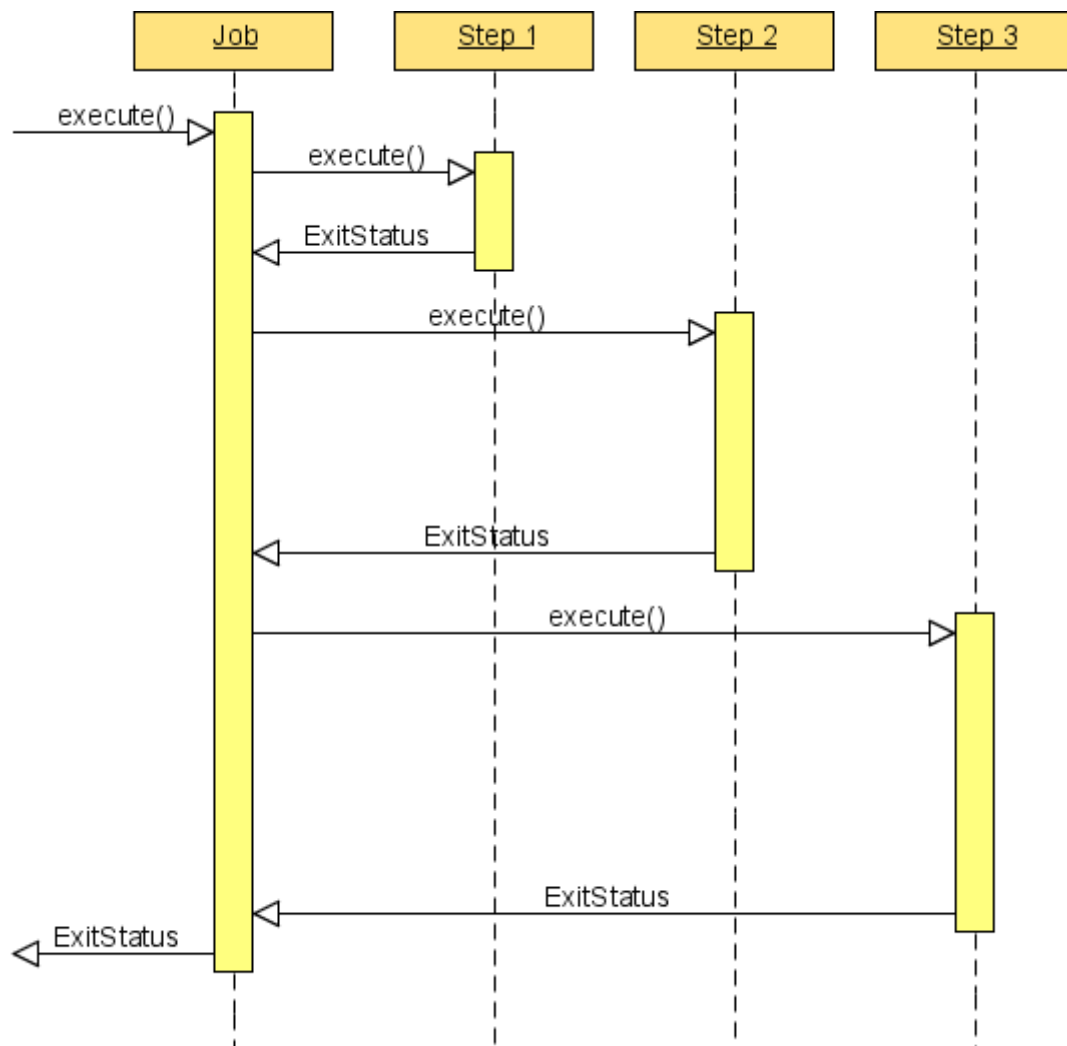
- ItemReader reads 1st item from source and gives that Item to ItemProcessor for processing.
- ItemReader reads 2nd item from source and gives that Item to ItemProcessor for processing
- Once chunk size items are in processing the ItemProcessor gives chunk size processed items to ItemWriter and they written to destination by ItemWriter.

Job

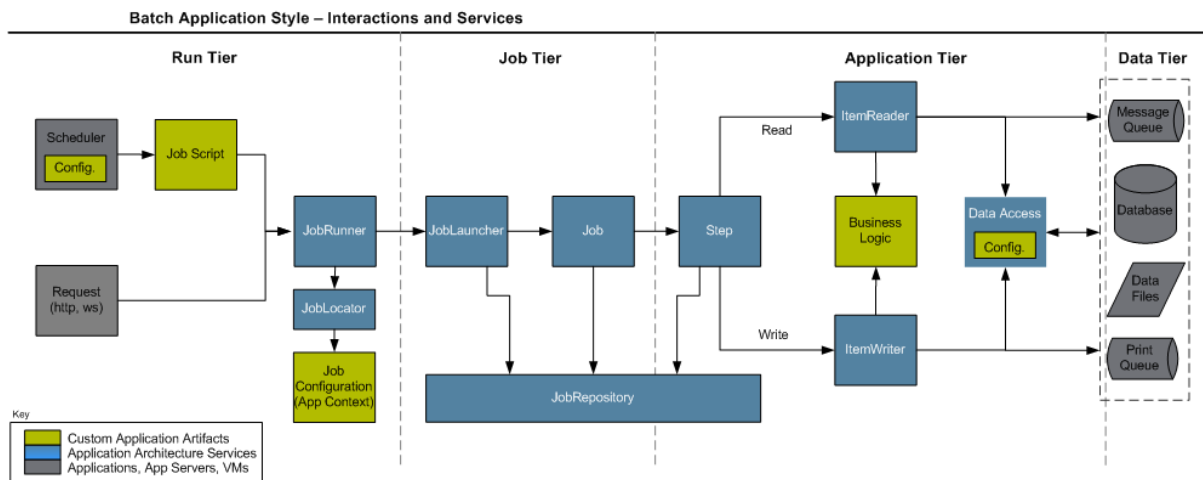
- Job represents work to be completed. Each Job contains 1 or more steps (Minimum 1 step will be there)
- Step object means it is the object of java class that implements org.springframework.batch.core.Step (I).

- Job object means it is the object of java class that implements pkg.Job (I) (org.springframework.batch.core.Job).
- Job represents total work to be completed whereas step represents task/ sub task of the Job in month.
- E.g., Job (Finding the Customers who are having due date this month to renew policies and triggering SMS messages).
 - Step 1 (sub task 1): Get all customer having due data in this method from DB s/w to Excel sheet
 - Step 2: Convert Excel sheet data to MySQL DB table (because SMS trigger app expecting details in MySQL DB table).
 - Step 3: Trigger SMS messages (Read from MySQL DB and trigger SMS messages)

Job with Steps sequence Flow diagram



Spring Batch Lunch Environment



- The application style is organized into four logical tiers, which include Run, Job, Application, and Data tiers. The primary goal for organizing an application according to the tiers is to embed what is known as "separation of concerns" within the system. Effective separation of concerns results in reducing the impact of change to the system.
- Run Tier:** The Run Tier is 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 as well as providing parallel processing capabilities.
- Job Tier:** The Job Tier is responsible for the overall execution of a batch job. It sequentially executes batch steps, ensuring that all steps are in the correct state and all appropriate policies are enforced.
- Application Tier:** The Application Tier contains components required to execute the program. It 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:** The Data Tier provides the integration with the physical data sources that might include databases, files, or queues.

Note: In some cases, the Job tier can be completely missing and in other cases one job script can start several batch jobs instances.

- Once we add "spring-boot-starter-batch" to the spring boot projects we multiple objects related spring batch processing through Autoconfiguration like
 - StepBuilderFactory (required to create Step object)
 - JobBuilderFactory (required to create Job object)

- c. JobLauncher (required to run the Job) and etc.

ItemReader

- All Item Reader classes in spring batch are the implementation classes of `org.springframework.batch.item.ItemReader<T>`.
- We generally do not implement ItemReader (I) i.e., we do not need to develop custom ItemReaders because Spring batch has provided multiple pre-defined ItemReaders.
- The read () of Each ItemReader reads info from source repository (like file, DB and etc.) and gives either String object or Model class object representing each record of the info.
- The readymade ItemReaders are `AmqpItemReader`, `AvroItemReader`, `FlatFileItemReader`, `HibernateCursorItemReader`, `HibernatePagingItemReader`, `ItemReaderAdapter`, `ItemReaderAdapter`, `IteratorItemReader`, `JdbcCursorItemReader`, `JdbcPagingItemReader`, `JmsItemReader`, `JpaCursorItemReader`, `JpaPagingItemReader`, `JsonItemReader`, `KafkaItemReader`, `LdifReader`, `ListItemReader`, `MappingLdifReader`, `MongoItemReader`, `MultiResourceItemReader`, `Neo4jItemReader`, `RepositoryItemReader`, `ResourceItemReader`, `SingleItemPeekableItemReader`, `StaxEventItemReader`, `StoredProcedureItemReader`, `SynchronizedItemStreamReader` and etc.
- Since all ItemReaders are pre-defined classes, we configure them as Spring bean using @Bean methods of @Configuration class/ Main class (which internally @Configuration class).

AppConfig.java (Configuration class)

@Bean

```
public ItemReader createReader() {  
    FlatFileItemReader reader = new FlatFileItemReader<....>();  
    .....  
    .....  
    return reader;  
}
```

ItemWriter

- It is use to write given chunk/ batch of information to destination.
- Generally, we do not develop ItemWrites because there are multiple readymade ItemWrites to use.
- ItemWriters are impl classes of

org.springframework.batch.item.ItemWriter<T>.

- The ready-made ItemWriters are AsyncItemWriter, AvroItemWriter, ChunkMessageChannelItemWriter, ClassifierCompositeItemWriter, CompositeItemWriter, [FlatFileItemWriter](#), GemfireItemWriter, [HibernateItemWriter](#), ItemWriterAdapter, ItemWriterAdapter, [JdbcBatchItemWriter](#), [JmsItemWriter](#), [JpaItemWriter](#), [JsonFileItemWriter](#), [KafkaItemWriter](#), KeyValueItemWriter, ListItemWriter, MimeMessageItemWriter, [MongoItemWriter](#), [MultiResourceItemWriter](#), Neo4jItemWriter, PropertyExtractingDelegatingItemWriter, RepositoryItemWriter, SimpleMailMessageItemWriter, SpELMappingGemfireItemWriter, [StaxEventItemWriter](#), SynchronizedItemStreamWriter and etc.
- Since all the ItemWriters are pre-defined classes, we generally configure them using @Bean methods of @Configuration class or main class.

[AppConfig.java \(Configuration class\)](#)

@Bean

```
public ItemWriter createWriter() {  
    JdbcBatchItemReader reader = new JdbcBatchItemReader();  
    .....  
    .....  
    return reader;  
}
```

Convert CSV file (Flat) data to DB table records after filter the records based on bill amount.

ItemProcessor

- It is implementation class of org.springframework.batch.item.ItemProcessor<I, O>
- Very Limited ready-made ItemProcessor are available. So, we generally develop custom ItemProcessors.
- The ready-made ItemProcessors are AsyncItemProcessor, BeanValidatingItemProcessor, ClassifierCompositeItemProcessor, CompositeItemProcessor, FunctionItemProcessor, ItemProcessorAdapter, ItemProcessorAdapter, PassThroughItemProcessor, ScriptItemProcessor, ValidatingItemProcessor

Note: The <I> of ItemProcessor must match <T> of ItemReader and similarly the <O> of ItemProcessor must match with <T> of ItemWriter.

CustomerFilterItemProcessor.java

@Component ("processor")

public class CustomerFilterItemProcessor implements
ItemProcessor<String, Customer>{

```
    public Customer process (String info) {  
        ..... //logic for convention and filtering  
        .....  
        return Customer object;  
    }
```

}

Step object

- It is implementation class object of org.springframework.batch.core.Step (I).
- Each Step object represents one task/ sub task of a Job.
- Every Step object must be linked with 1 reader object, 1 writer object and 1 processor object.
- We generally create Step object by Using the StepBuilderFactory object that comes Spring Boot application through AutoConfiguration.
- Every Step object must contain the following 5 details
 - a. Step name
 - b. <Input type, output type> + chunk size
 - c. Reader object
 - d. Writer object
 - e. Processor object
- We generally use @Bean method in @Configuration class to create Step object with StepBuilderFactory object and providing the above 5 details.

AppConfig.java (Configuration class)

@Configuration

@ComponentScan(basePackages="....")

public class AppConfig {

@Autowired

private StepBuilderFactory sbFactory;

@Autowired

private JobBuilderFactory jbFactory;

```

@Autowired
private CustomerFilterItemProcessor custProcessor;

//read configuration
.....
//writer configuration
.....

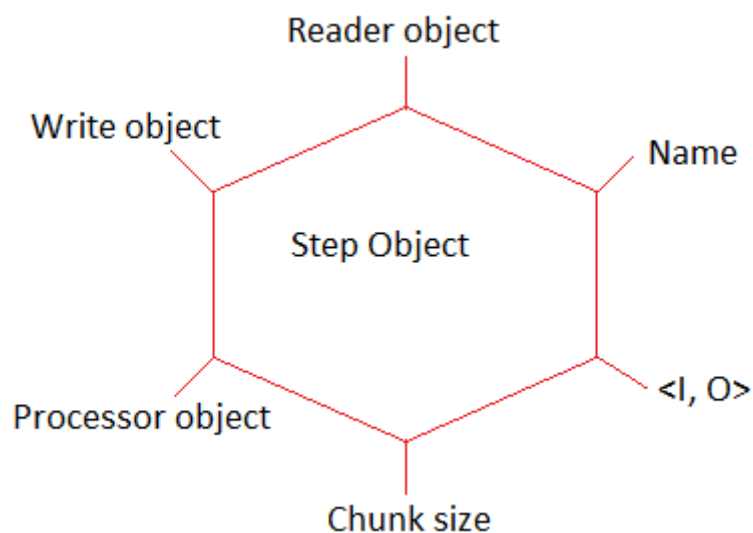
@Bean(name="step1")
public Step createStep() {
    return sbFactory.get("step1") //step name
        .<String, Customer>chunk(10)
        //chunk size + Input, output types
        .reader(createReader()) //reader object
        .processor(custProcessor) //processor object
        .writer(createWriter()) //writer object
        .build(); // build() method builds the step
}

//job configuration
.....
}

```

Note: method chaining, calling method().method().method().... is called Method Chaining i.e., The returned object of 1 method will be used as base object to call another method.

Spring Batch Configuration



JobExecutionListener (I)

- By implementing this interface, we can perform event handling on job activities like when Job is started, when job is completed, what is status of job completion (Success or failure) and etc.
- In creation Job object we need JobListener object.
`org.springframework.batch.core.JobExecutionListener(I)`

JobMonitoringListener.java

`@Component("jmListener")`

`public JobMonitoringListener implements JobExecutionListener{`

`public void beforeJob(JobExecution execution) {`

`.....`

`.....`

`}`

`public void afterJob(JobExecution execution){`

`.....`

`.....`

`}`

`}`

Job object

- It is the object of class that implements `org.springframework.batch.core.Job (I)`.
- Job defines the work to be completed
- Generally, one application contains one Job object with 1 or more Step object (tasks/ sub tasks).
- To create Job object, we use `JobBuilderFactory` that comes Spring batch application through `AutoConfiguration` process.
- Job object creation needs multiple details like name, listener, incrementor (specifies the order of executing steps), starting step, next step, next & next step

BatchConfig.java

`@Configuration`

`@EnableBatchProcessing`

`public class BatchConfig {`

`.....`

`.....`

```

@Autowired
private JobBuilderFactory jobFactory;

@Autowired
private JobExecutionListener listener;

//reader configuration
.....

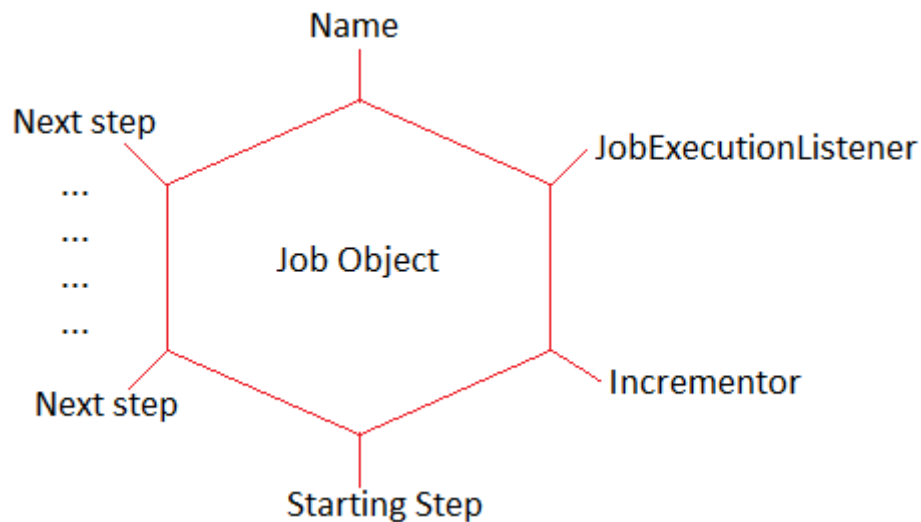
//writer configuration
.....

//process configuration
.....

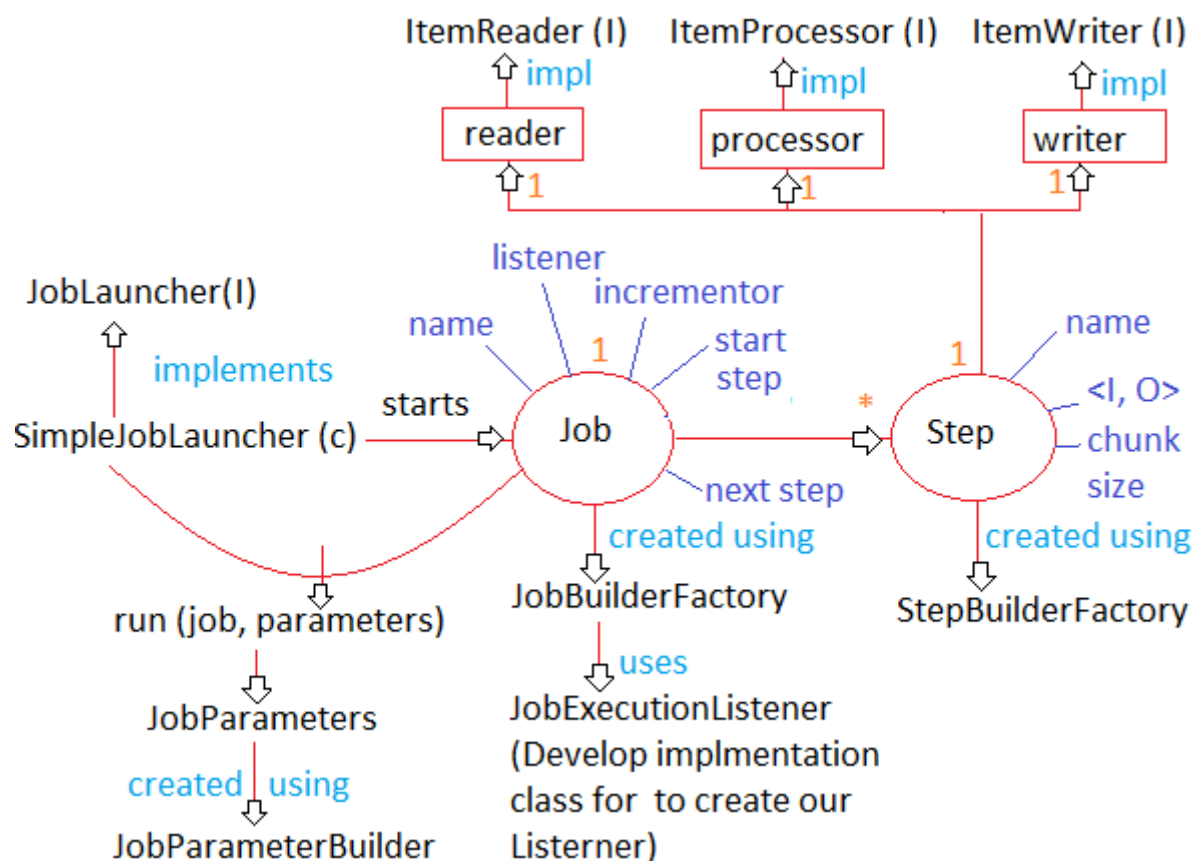
//Step configuration
@Bean(name="step1")
public Step createStep() {
    return sbFactory.get("step1") //step name
        .<String, Customer>chunk(10)
        //chunk size + Input, output types
        .reader(createReader()) //reader object
        .processor(custProcessor) //processor object
        .writer(createWriter()) //writer object
        .build(); // build() method builds the step
}

//Job configuration
@Bean(name="job1")
public Job createJob1(){
    return jobFactory.get("job1") // job name
        .incrementor(new RunIdIncrementor())
        // specifies the order executing given steps
        .listener(listener) //specifics the listener object
        .start(createStep1()) //starting step
        //.next(createStep2()) //next step
        //.next(createStep3()) //next step .....
        .build(); //buils the Job object
}
}

```

End to end technical view/ flow of Spring Batch application



- Here we inject JobLauncher object given by AutoConfiguration and also Job object configured in the configuration file using @Autowired annotation.
- We build JobParameters (optional) using JobParametersBuilder object.
- We call run (job, parameters) on JobLauncher object to run the job.

BatchProcessingTest.java

```
public class BatchProcessingTest {

    @Autowired
    private JobLauncher launcher;

    @Autowired
    private Job job;

    public static void main (String args[]) {
        JobParameters params= new JobParameterBuilder()
            .addLong("jobId",new
                Random().nextInt(10000)).toJobParameters();
        launcher.run(job,params);
    }
}
```

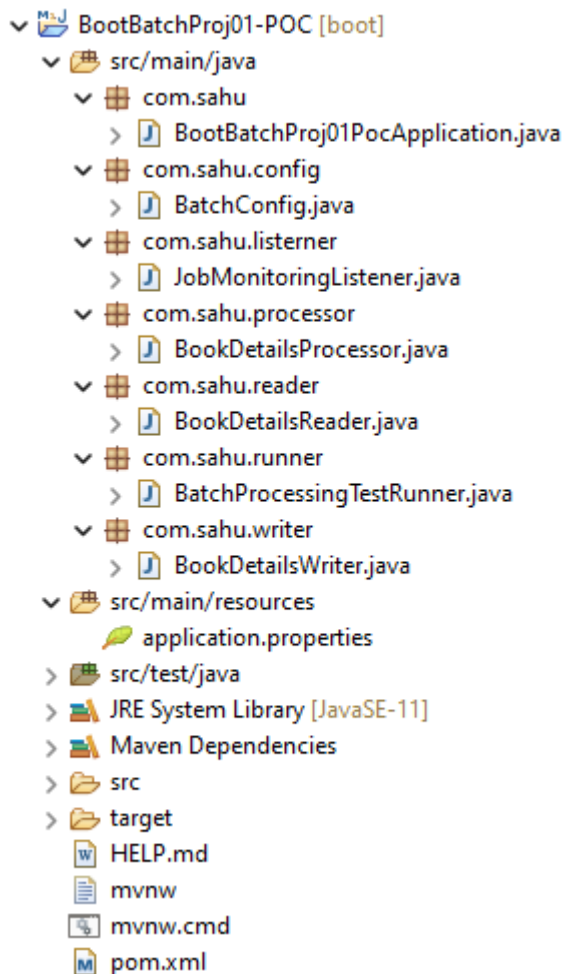
POC (Proof of Concepts on Spring Batch processing)

- Custom ItemReader
- Custom ItemWriter
- Custom ItemProcessor
- JobExecution Listeners (enable Batch Processing)
- BatchConfig class
 - JobBuilderFactory
 - StepBuilderFactory
 - Listener
 - Reader
 - Writer
 - Processor
 - @Bean method for Step object
 - @Bean method for Job object
- Client application
 - JobLauncher AutoConfiguration based @Autowired
 - Job @Autowired
 - Prepare JobParameters (optional)Run the job

Following dependencies are required

```
X Spring Batch
X H2 Database
```

Directory Structure of BootBatchProj01-POC:



- Develop the above directory structure using Spring Starter Project option (Packaging: Jar) and create the packages and classes also.
- Add the following starter during project creation.

```
X Spring Batch
X H2 Database
```

- Then place the following code with in their respective files.

application.properties

Indicate wheter batch code should execute on the application start up or # on demand [true: on the application start up (default) & false: on demand when launcher.run(-) is called]

spring.batch.job.enabled=false

It is underlying DB s/w to create lots of DB tables to keep track of job execution related operations. possible values - always, never, embedded

spring.batch.jdbc.initialize-schema=always

JobMonitoringListener.java

```
package com.sahu.listener;

import java.util.Date;

import org.springframework.batch.core.JobExecution;
import org.springframework.batch.core.JobExecutionListener;
import org.springframework.stereotype.Component;

@Component
public class JobMonitoringListener implements JobExecutionListener {

    private long startTime, endTime;

    public JobMonitoringListener() {

        System.out.println("JobMonitoringListener.JobMonitoringListener()");
    }

    @Override
    public void beforeJob(JobExecution jobExecution) {
        System.out.println("Job is about to begin at : "+new Date());
        startTime = System.currentTimeMillis();
        System.out.println("Job status : "+jobExecution.getStatus());
    }

    @Override
    public void afterJob(JobExecution jobExecution) {
        System.out.println("Job completed at : "+new Date());
        endTime = System.currentTimeMillis();
        System.out.println("Job Status : "+jobExecution.getStatus());
        System.out.println("Job Execution time : "+(endTime-
startTime));
        System.out.println("Job Exit status :
"+jobExecution.getExitStatus());
    }
}
```

BookDetailsReader.java

```
package com.sahu.reader;

import org.springframework.batch.item.ItemReader;
import org.springframework.batch.item.NonTransientResourceException;
import org.springframework.batch.item.ParseException;
import org.springframework.batch.item.UnexpectedInputException;
import org.springframework.stereotype.Component;

@Component("bdReader")
public class BookDetailsReader implements ItemReader<String> {

    String books[] = new String[] {"CRJ", "TIJ", "HFJ", "EJ", "BBJ"};

    int count=0;

    public BookDetailsReader() {
        System.out.println("BookDetailsReader.BookDetailsReader()");
    }

    @Override
    public String read() throws Exception, UnexpectedInputException,
    ParseException, NonTransientResourceException {
        System.out.println("BookDetailsReader.read()");
        if (count<books.length)
            return books[count++];
        else
            return null;
    }
}
```

BookDetailsProcessor.java

```
package com.sahu.processor;

import org.springframework.batch.item.ItemProcessor;
import org.springframework.stereotype.Component;

@Component("bdProcessor")
public class BookDetailsProcessor implements ItemProcessor<String,
```

```
String> {

    public BookDetailsProcessor() {

        System.out.println("BookDetailsProcessor.BookDetailsProcessor()");
    }

    @Override
    public String process(String item) throws Exception {
        System.out.println("BookDetailsProcessor.process()");
        String bookWithTitle = null;
        if (item.equalsIgnoreCase("CRJ"))
            bookWithTitle = item+" by HS and PN";
        else if (item.equalsIgnoreCase("TIJ"))
            bookWithTitle = item+" by BE";
        else if (item.equalsIgnoreCase("HFJ"))
            bookWithTitle = item+" by KS";
        else if (item.equalsIgnoreCase("EJ"))
            bookWithTitle = item+" by JB";
        else if (item.equalsIgnoreCase("BBJ"))
            bookWithTitle = item+" by RNR";

        return bookWithTitle;
    }
}
```

BookDetailsWriter.java

```
package com.sahu.writer;

import java.util.List;

import org.springframework.batch.item.ItemWriter;
import org.springframework.stereotype.Component;

@Component("bdWriter")
public class BookDetailsWriter implements ItemWriter<String> {

    public BookDetailsWriter() {
        System.out.println("BookDetailsWriter.BookDetailsWriter()");
    }
}
```

```

    }

    @Override
    public void write(List<? extends String> items) throws Exception {
        System.out.println("BookDetailsWriter.write()");
        items.forEach(System.out::println);
    }
}

```

BatchConfig.java

```

package com.sahu.config;

import org.springframework.batch.core.Job;
import org.springframework.batch.core.Step;
import org.springframework.batch.core.configuration.annotation.EnableBatchProcessing;
import org.springframework.batch.core.configuration.annotation.JobBuilderFactory;
import org.springframework.batch.core.configuration.annotation.StepBuilderFactory;
import org.springframework.batch.core.launch.support.RunIdIncrementer;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import com.sahu.listener.JobMonitoringListener;
import com.sahu.processor.BookDetailsProcessor;
import com.sahu.reader.BookDetailsReader;
import com.sahu.writer.BookDetailsWriter;

@Configuration
@EnableBatchProcessing //Gives Spring batch features through
                        autoconfiguration
//like giving InMemoryJobRepository, JobBuilderFactory, StepBuilderFactory
and etc.
public class BatchConfig {

```

```

@Autowired
private JobBuilderFactory jobBuilderFactory;

@Autowired
private StepBuilderFactory stepBuilderFactory;

@Autowired
private BookDetailsWriter detailsWriter;

@Autowired
private BookDetailsReader detailsReader;

@Autowired
private BookDetailsProcessor detailsProcessor;

@Autowired
private JobMonitoringListener monitoringListener;

//Create Step object using StepBuilderFactory
@Bean(name = "step1")
public Step createStep1() {
    System.out.println("BatchConfig.createStep1()");
    return stepBuilderFactory.get("step1")
        .<String, String>chunk(2)
        .reader(detailsReader)
        .writer(detailsWriter)
        .processor(detailsProcessor)
        .build();
}

//Create Job using JobBuilderFactory
@Bean(name = "job1")
public Job createJob() {
    System.out.println("BatchConfig.createJob()");
    return jobBuilderFactory.get("job1")
        .incrementer(new RunIdIncrementer())
        .listener(monitoringListener)
        .start(createStep1())
        .build();
}
}

```


BatchProcessingTestRunner.java

```
package com.sahu.runner;

import org.springframework.batch.core.Job;
import org.springframework.batch.core.JobParameters;
import org.springframework.batch.core.JobParametersBuilder;
import org.springframework.batch.core.launch.JobLauncher;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;

@Component
public class BatchProcessingTestRunner implements CommandLineRunner {

    @Autowired
    private JobLauncher launcher;

    @Autowired
    private Job job;

    @Override
    public void run(String... args) throws Exception {
        //Prepare job parameters
        JobParameters parameters = new
JobParametersBuilder().addLong("time",
System.currentTimeMillis()).toJobParameters();
        //Run the job
        launcher.run(job, parameters);
    }
}
```

Core Java Recap (About instance block)

Q. How many special blocks can be placed in a java class?

Ans.

- Instance block, static block
- We generally use static block to initialize static member variables of a class and instance block to initialize non-static member variables of class.
- Regular blocks in java are method blocks, try blocks, catch blocks, finally

blocks, constructor blocks, if blocks, loop blocks, and etc.

Q. When constructor is there to initialize non-static member variables then why should we take instance block?

Ans.

- Instead of placing same initialization logic in multiple overloaded constructors think about placing it in instance block.
- While creating anonymous classes or anonymous inner classes we can not place constructor in them because they do not have name, so we can use instance block support to place initialization logics.

```
class Test {  
    public void m1 () {  
        .....  
    }  
}
```

Case 1: Test t = new Test ();
t.m1(); *//valid*

Case 2: Test t1 = new Test () { };
t1.m1();
System.out.println(t.getClass());

Here anonymous sub class is created for Test class and that object is referred by Super class (Test) reference variable t1.

Case 3: Test t = new Test () {
 //Constructor cannot define here
 {
 m1();
 }
};

instance block in anonymous sub class of Test and calling m1 () method in it.

This anonymous sub class definition for Test were constructor defined cannot be defined.

Test.java

```
package com.sahu.basics;
```

```

public class Test {

    public Test() {
        System.out.println("Test.Test()");
    }

    public void m1() {
        System.out.println("Test.m1()");
    }

    public static void main(String[] args) {
        Test t = new Test();
        t.m1();
        System.out.println("Test object (t) class name : "+t.getClass()+"",
        Super class name : "+t.getClass().getSuperclass());

        Test t1 = new Test() {};
        t1.m1();
        System.out.println("Test object (t) class name :
        "+t1.getClass()+"", Super class name : "+t1.getClass().getSuperclass());

        Test t2 = new Test() {
            //Constructor cannot be defined here, so go for instance
            block
            {
                m1();
            }
        };
        System.out.println("Test object (t) class name :
        "+t2.getClass()+"", Super class name : "+t2.getClass().getSuperclass());
    }
}

```

- Spring batch/ Spring batch processing can collect data from various sources/ source repositories in different formats and can process them for modification or filtering or sorting, lastly writes that processed data to various destination/ destination repositories.
- Spring batch can deal with different types data to read from sources and process and also to write destinations.

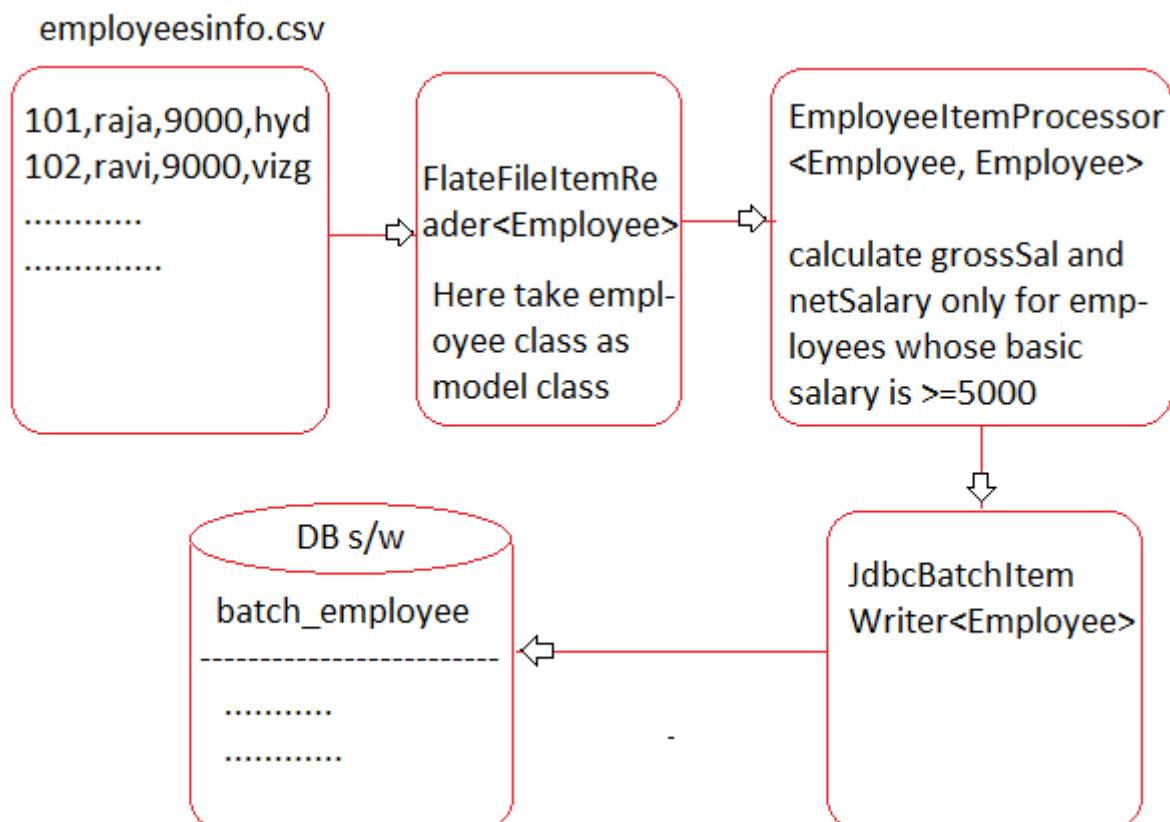
Reading CSV File Data (Excel Data), Processing data and writing oracle DB table using Spring batch

Use case 1: Census info comes Central ministry form every village/ town/ city in the form of csv files (excel files). So, the Batch app of central ministry should process data by categorizing and validating before writing to Oracle DB table.

Use case 2: While conducting election survey the field workers get people's pulse in the form of excel sheets, so we should process and store in DB table to generate certain report or graphs

- To read data from csv file or excel file or json file or formatted text file we can take support of `FlatFileItemReader<T>` as reader class.
- To Write processed data to DB table as records batch by batch by executing SQL queries in batch we can use `JdbcBatchItemWriter<T>` class.
- To process data always prefer custom Item Processor.

High level architecture:



FlatFileItemReader<T>

1. Create object for `FlatFileItemReader` specifying Model class name.

2. Specify name and location of the csv file from where it has to read data.
3. Specify LineMapper to read each line from csv file.

101,raja,9000,hyd

4. Specify LineTokenizer to tokenize/ split content of the line in to values based on given delimiter (",").

101

raja

9000

hyd

5. Specify FiledSetMapper to map the values of each line to the properties/ variables of Model class and set each Line content to one Model class object.

Employee class object

eno: 101, ename:
raja, eadd: hyd
basicSalary: 9000
grossSalary: null,
netSalary: null

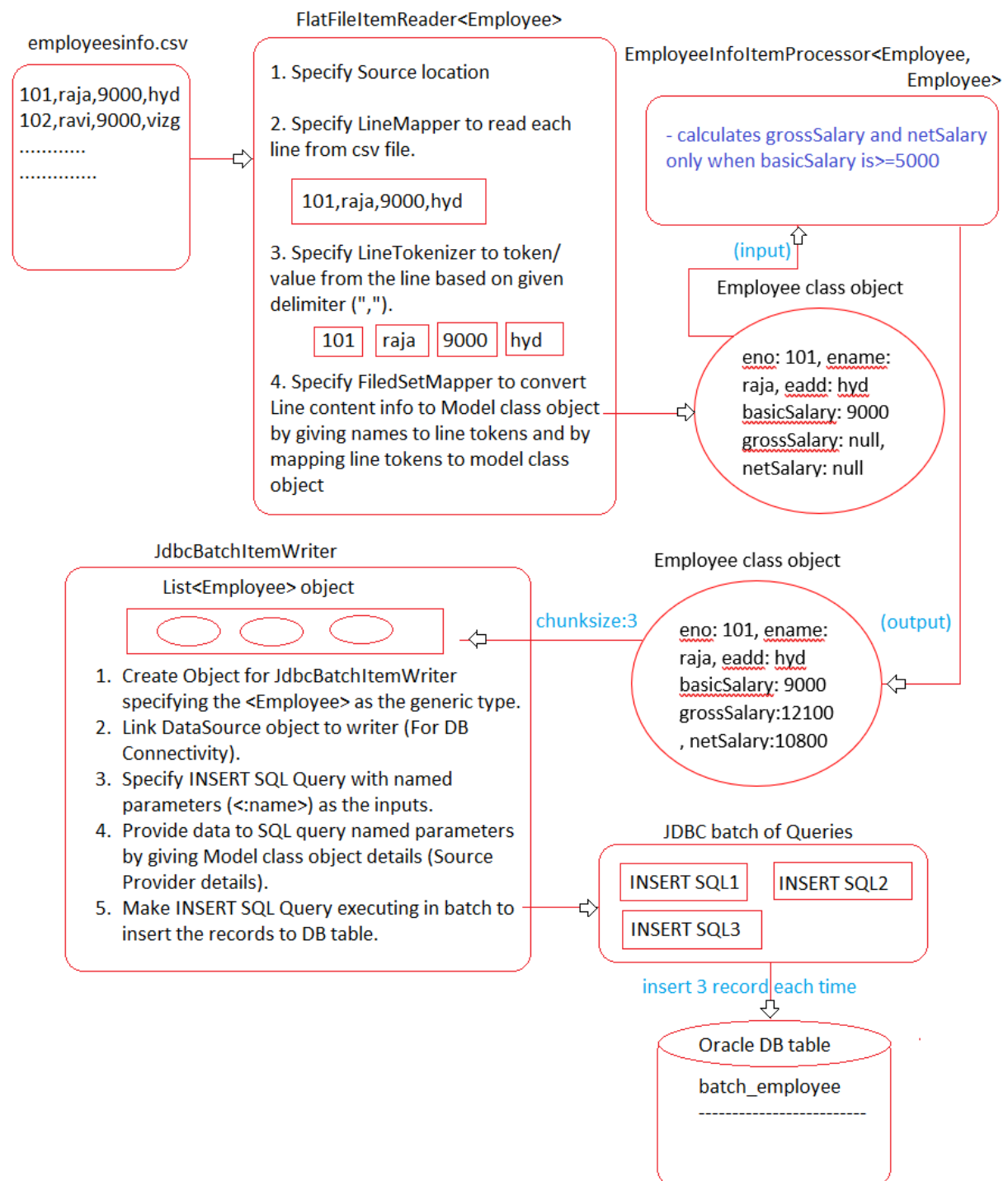
JdbcBatchItemWriter

- ✚ It gets inputs from ItemProcessor as List<T> (In our case it List<Employee>).
1. Create Object for JdbcBatchItemWriter specifying the <Employee> as the generic type.
 2. Link DataSource object to writer (For DB Connectivity).
 3. Specify INSERT SQL Query with named parameters (<:name>) as the inputs.
 4. Provide data to SQL query named parameters by giving Model class object details (Source Provider details).
 5. Make INSERT SQL Query executing in batch to insert the records to DB table.

Note: CSV stands for Comma-separated values.

For the below application we have to create a csv file having 500 records with employee Id, employee name, employee salary and employee address.

Story board of CSV file to DB table conversion using Spring Batch



Note: We generally try to match the names of named param with model class property names to make binding process easy.

While working with FlatFileItemWriter<T> we use following classes as the supporting classes,

- DefaultLineReader: To read each line from csv file having enter key as delimiter.
- DelimitedLineTokenizer: To split each Line content to values having names based on the given delimiter (generally ",").
- BeanWrapperFieldSetMapper: To map each line tokens to given Model class object (here tokens and model class property names must match).

✚ While working with JdbcBatchItemWriter<T> we use following classes as the supporting classes

- DataSource (HikariDataSource)
- BeanPropertySqlParameterSourceProvider: To provide given java bean class object/ model class object property values as the SQL query named param values (Here the named param names and model class property names must match)













Example App

Step 1: Create Spring Boot starter project adding following starters.

☒ Lombok
☒ Spring Batch
☒ JDBC API
☒ Oracle Driver







Step 2: Place csv file (source file) in main/src/resources folder.

We can generate csv file by using online tool [\[click here\]](#)

	Field name	Data type	Setting
  	<input type="text" value="empno"/>	<input type="text" value="Index"/>	From <input type="text" value="7900"/>
  	<input type="text" value="ename"/>	<input type="text" value="First name"/>	
  	<input type="text" value="salary"/>	<input type="text" value="Random (integer)"/>	From <input type="text" value="3000"/> To <input type="text" value="10000"/>
  	<input type="text" value="eadress"/>	<input type="text" value="City"/>	

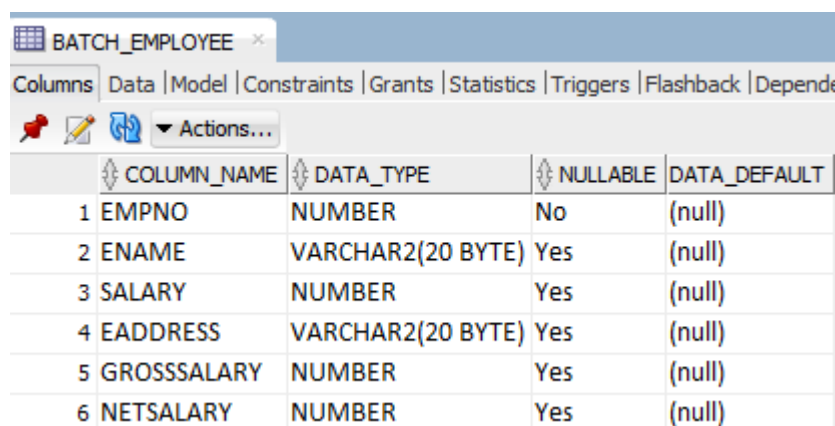
Note: Go to the website spend some time you will get the idea how to do it, and you can check the tutorial in their website (scroll little bit down you will get all the information about how to generate and about the Data type).

Step 3: Create the following packages in src/main/java folder

- >  com.sahu.config
- >  com.sahu.listener
- >  com.sahu.processor
- >  com.sahu.reader
- >  com.sahu.runner
- >  com.sahu.writer

Step 4: Add DataSource, Spring batch configuration in application.properties file.

Step 5: Make sure that DB table is created in Oracle DB s/w.



The screenshot shows the 'BATCH_EMPLOYEE' table structure in Oracle SQL Developer. The table has six columns: EMPNO (NUMBER), ENAME (VARCHAR2(20 BYTE)), SALARY (NUMBER), EADDRESS (VARCHAR2(20 BYTE)), GROSSSALARY (NUMBER), and NETSALARY (NUMBER). The NULLABLE column indicates that EMPNO is not nullable, while the others are. The DATA_DEFAULT column shows that all columns have a default value of (null).

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT
1	EMPNO	NUMBER	No	(null)
2	ENAME	VARCHAR2(20 BYTE)	Yes	(null)
3	SALARY	NUMBER	Yes	(null)
4	EADDRESS	VARCHAR2(20 BYTE)	Yes	(null)
5	GROSSSALARY	NUMBER	Yes	(null)
6	NETSALARY	NUMBER	Yes	(null)

Step 6: Develop the model class Employee.java.

Step 7: Develop JobExecutionListener.

Step 8: Develop ItemProcessor.

Note: The null object given by ItemProcessor will not got to ItemWriter.

Step 9: Configure ItemProcessor in BatchConfig.java using @Bean method.

Step 10: Inject JobBuilderFactory, SetpBuilderFactory to BatchConfig.java class by using @Autowired.

Step 11: Configure JobExecutionListener as Spring bean in BatchConfig.java class using @Bean method.

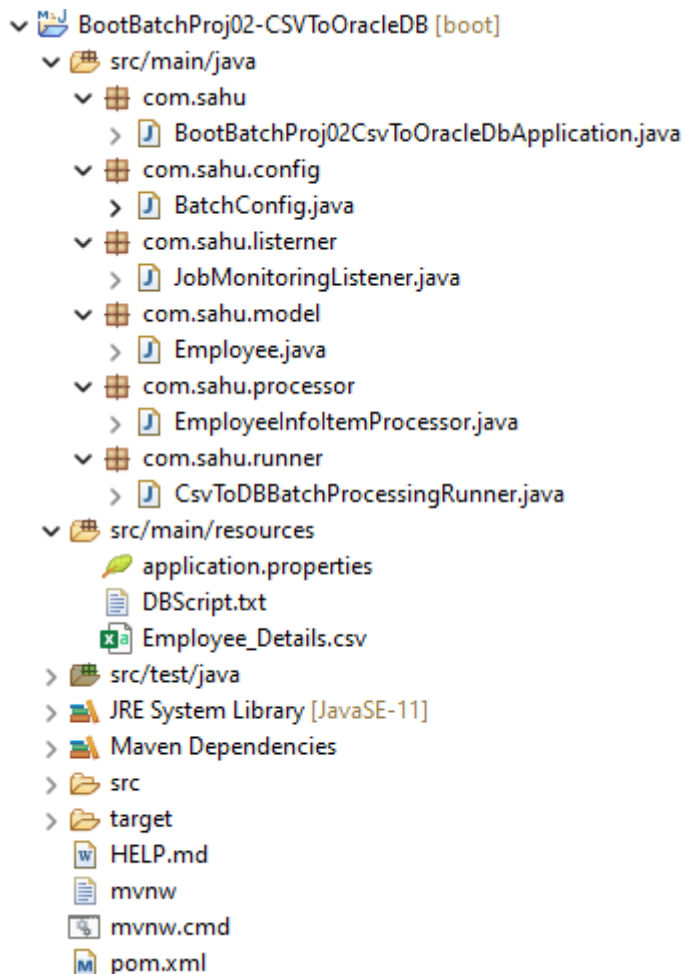
Step 12: Configure JdbcBatchWriter in BatchConfig.java class using @Bean method.

Step 13: Create Step object, Job object as Spring bean in BatchConfig.java class using @Bean method.

Step 14: Develop the Runner class having code to run the job using JobLauncher object.

Step 15: Run the application.

Directory Structure of BootBatchProj02-CSVToOracleDB:



- Develop the above directory structure using Spring Starter Project option (Packaging: Jar) and create the packages and classes also.
- Add the following starter during project creation.
 - X Lombok
 - X Spring Batch
 - X JDBC API
 - X Oracle Driver
- Collect the csv file using online tool [\[click here\]](#) and place in resources.
- Create the table using DBScript.txt details.
- Then place the following code with in their respective files.

DBScript.txt

```
CREATE TABLE "SYSTEM"."BATCH_EMPLOYEE"  
(  
  "EMPNO" NUMBER NOT NULL ENABLE,  
  "ENAME" VARCHAR2(20 BYTE),  
  "SALARY" NUMBER,  
  "EADDRESS" VARCHAR2(50 BYTE),  
  "GROSSSALARY" NUMBER,  
  "NETSALARY" NUMBER,  
  CONSTRAINT "BATCH_EMPLOYEE_PK" PRIMARY KEY ("EMPNO"))
```

application.properties

```
#Spring batch configuration  
spring.batch.job.enabled=false  
spring.batch.jdbc.initialize-schema=always  
  
#Datasource configuration  
spring.datasource.driver-class-name=oracle.jdbc.OracleDriver  
spring.datasource.url=jdbc:oracle:thin:@localhost:1521:xe  
spring.datasource.username=system  
spring.datasource.password=manager
```

JobMonitoringListener.java

```
package com.sahu.listerner;  
  
import java.util.Date;  
  
import org.springframework.batch.core.JobExecution;  
import org.springframework.batch.core.JobExecutionListener;  
import org.springframework.stereotype.Component;  
  
@Component  
public class JobMonitoringListener implements JobExecutionListener {  
  
    private long startTime, endTime;  
  
    @Override  
    public void beforeJob(JobExecution jobExecution) {  
        startTime = System.currentTimeMillis();  
        System.out.println("Job is about to start at : "+new Date());  
    }  
}
```

```

    }

    @Override
    public void afterJob(JobExecution jobExecution) {
        endTime = System.currentTimeMillis();
        System.out.println("Job has completed at : "+new Date());
        System.out.println("Job Execution time : "+(endTime-
startTime)+"ms");
        System.out.println("Job Execution status : 
"+jobExecution.getStatus());
    }
}

```

Employee.java

```

package com.sahu.model;

import lombok.Data;

@Data
public class Employee {
    private Integer empno;
    private String ename;
    private Integer salary;
    private String eaddress;
    private Integer grossSalary;
    private Integer netSalary;
}

```

EmployeeInfoItemProcessor.java

```

package com.sahu.processor;

import org.springframework.batch.item.ItemProcessor;
import org.springframework.stereotype.Component;

import com.sahu.model.Employee;

@Component
public class EmployeeInfoItemProcessor implements

```

```

ItemProcessor<Employee, Employee> {

    @Override
    public Employee process(Employee emp) throws Exception {
        if (emp.getSalary() >= 5000) {

            emp.setGrossSalary(Math.round(emp.getSalary() + emp.getSalary() * 0.4f));

            emp.setNetSalary(Math.round(emp.getSalary() + emp.getSalary() * 0.2f));
        }
        else {
            emp = null;
        }
        return emp;
    }
}

```

BatchConfig.java

```

package com.sahu.config;

import javax.sql.DataSource;

import org.springframework.batch.core.Job;
import org.springframework.batch.core.JobExecutionListener;
import org.springframework.batch.core.Step;
import org.springframework.batch.core.configuration.annotation.EnableBatchProcessing;
import org.springframework.batch.core.configuration.annotation.JobBuilderFactory;
import org.springframework.batch.core.configuration.annotation.StepBuilderFactory;
import org.springframework.batch.core.launch.support.RunIdIncrementer;
import org.springframework.batch.item.ItemProcessor;
import org.springframework.batch.item.ItemReader;

```

```

import org.springframework.batch.item.ItemWriter;
import
org.springframework.batch.item.database.BeanPropertyItemSqlParameterS
ourceProvider;
import org.springframework.batch.item.database.JdbcBatchItemWriter;
import
org.springframework.batch.item.database.builder.JdbcBatchItemWriterBuil
der;
import org.springframework.batch.item.file.FlatFileItemReader;
import
org.springframework.batch.item.file.builder.FlatFileItemReaderBuilder;
import
org.springframework.batch.item.file.mapping.BeanWrapperFieldSetMapper
;
import org.springframework.batch.item.file.mapping.DefaultLineMapper;
import
org.springframework.batch.item.file.transform.DelimitedLineTokenizer;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.core.io.ClassPathResource;

import com.sahu.listener.JobMonitoringListener;
import com.sahu.model.Employee;
import com.sahu.processor.EmployeeInfoItemProcessor;

@Configuration
@EnableBatchProcessing
public class BatchConfig {

    @Autowired
    private DataSource dataSource;

    @Autowired
    private JobBuilderFactory jobBuilderFactory;

    @Autowired
    private StepBuilderFactory stepBuilderFactory;

    //Listener

```

```

@Bean
public JobExecutionListener createListener() {
    return new JobMonitoringListener();
}

//ItemProcessor
@Bean
public ItemProcessor<Employee, Employee> createProcessor() {
    return new EmployeeInfoltemProcessor();
}

//Way #1 of ItemReader creation
@Bean
public ItemReader<Employee> createReader() {
    //Create object for FlatFileItemReader
    FlatFileItemReader<Employee> reader = new
FlatFileItemReader<Employee>();
    //Set source CSV file location
    reader.setResource(new
ClassPathResource("Employee_Details.csv"));
    //Set LineMapper
    reader.setLineMapper(new DefaultLineMapper<Employee>(){
        {
            //Set LineTokenizer
            setLineTokenizer(new DelimitedLineTokenizer() {
                {
                    setDelimiter(",");
                    setNames("empno", "ename",
"salary", "eaddress");
                }
            });
            //Set FieldSetMapper to write each line content to
model class object
            setFieldSetMapper(new
BeanWrapperFieldSetMapper<Employee>() {
                {
                    setTargetType(Employee.class);
                }
            });
        }
    });
}

```

```

    });
    return reader;
}

//Way #2 of ItemReader creation
@Bean
public ItemReader<Employee> createReader1() {
    //Create object for FlatFileItemReader
    FlatFileItemReader<Employee> reader = new
FlatFileItemReader<Employee>();
    //Set source CSV file location
    reader.setResource(new
ClassPathResource("Employee_Details.csv"));
    //Line Mapper
    DefaultLineMapper<Employee> lineMapper = new
DefaultLineMapper<Employee>();
    //Line Tokenizer
    DelimitedLineTokenizer lineTokenizer = new
DelimitedLineTokenizer();
    lineTokenizer.setDelimiter(",");
    lineTokenizer.setNames("empno", "ename", "salary",
"eaddress");
    //Line Tokenizer
    BeanWrapperFieldSetMapper<Employee> setMapper = new
BeanWrapperFieldSetMapper<Employee>();
    setMapper.setTargetType(Employee.class);
    //Add Line Tokenize, Line Tokenizer to Line Mapper
    lineMapper.setLineTokenizer(lineTokenizer);
    lineMapper.setFieldSetMapper(setMapper);
    //add line mapper to reader
    reader.setLineMapper(lineMapper);
    return reader;
}

//Way #3 of ItemReader creation
@Bean
public ItemReader<Employee> createReader2() {
    return new FlatFileItemReaderBuilder<Employee>()
        .name("file-reader")
        .resource(new

```

```

ClassPathResource("Employee_Details.csv"))
        .delimited().delimiter(",") //makes us to use
DefaultLineMapper, DelimitedLineTokenizer
        .names("empno", "ename", "salary", "eaddress")
        .targetType(Employee.class)
        .build();
    }

    //Way #1 of ItemWriter creation
    @Bean
    public ItemWriter<Employee> createWriter(){
        JdbcBatchItemWriter<Employee> writer = new
JdbcBatchItemWriter<Employee>();
        //Set Datasource
        writer.setDataSource(dataSource);
        //set SQL query with named params
        writer.setSql("INSERT INTO BATCH_EMPLOYEE
VALUES(:empno,:ename,:salary,:eaddress,:grossSalary,:netSalary)");
        //Set Model class object as SqlParameterSourceProvider (here
named parameter names and model class object property name must same)
        writer.setItemSqlParameterSourceProvider(new
BeanPropertySqlParameterSourceProvider<Employee>());
        return writer;
    }

    //Way #2 of ItemWriter creation
    @Bean
    public ItemWriter<Employee> createWriter1() {
        return new JdbcBatchItemWriterBuilder<Employee>()
            .dataSource(dataSource)
            .sql("INSERT INTO BATCH_EMPLOYEE
VALUES(:empno,:ename,:salary,:eaddress,:grossSalary,:netSalary)")
            .beanMapped() //makes to use
BeanPropertySqlParameterSourceProvider
            .build();
    }

    @Bean("step1")
    public Step createStep1() {

```



```

        return stepBuilderFactory.get("step1")
            .<Employee, Employee>chunk(3)
            .reader(createReader())
            .writer(createWriter())
            .processor(createProcessor())
            .build();
    }

    @Bean(name = "job1")
    public Job createJob1() {
        return jobBuilderFactory.get("job1")
            .incrementer(new RunIdIncrementer())
            .listener(createListener())
            .start(createStep1())
            .build();
    }
}

```

CsvToDBBatchProcessingRunner.java

```

package com.sahu.runner;

import org.springframework.batch.core.Job;
import org.springframework.batch.core.JobExecution;
import org.springframework.batch.core.JobParameters;
import org.springframework.batch.core.JobParametersBuilder;
import org.springframework.batch.core.launch.JobLauncher;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;

@Component
public class CsvToDBBatchProcessingRunner implements
CommandLineRunner {

    @Autowired
    private JobLauncher jobLauncher;

    @Autowired
    private Job job;
}

```

```

@Override
public void run(String... args) throws Exception {
    JobParameters params = new
JobParametersBuilder().addLong("sysTime",
System.currentTimeMillis()).toJobParameters();
    JobExecution execution = jobLauncher.run(job, params);
    System.out.println("Job Complete Status :
"+execution.getStatus());
}
}

```

Spring Batch application converting DB table records into CSV file (Excel file)

Use case:

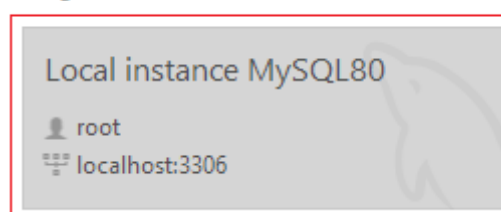
- Sending Bank account statement to customer from DB table in the form of excel sheet/ csv file.
- Giving Voters List of PS (polling station) to the Leader as csv file by collection information from DB table.
- University publishing selected students for different companies.
- Publishing list of students who have not fees.
- Publishing list of customers in a society showing their power bill details. and etc.

✚ For reading from Db table, we can use JdbcCursorItemReader<T> and for writing processed data to csv file/ json file/ text file we can use FlatFileItemWriter<T>.

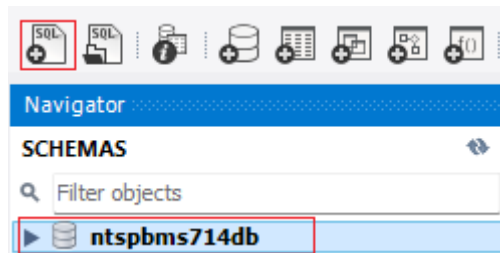
DB script creating huge number of random records in MySQL DB table

Step 1: You can use MySQL workbench or MySQL Command Line Client. Open MySQL Workbench and connect to you Connection by double click on that.

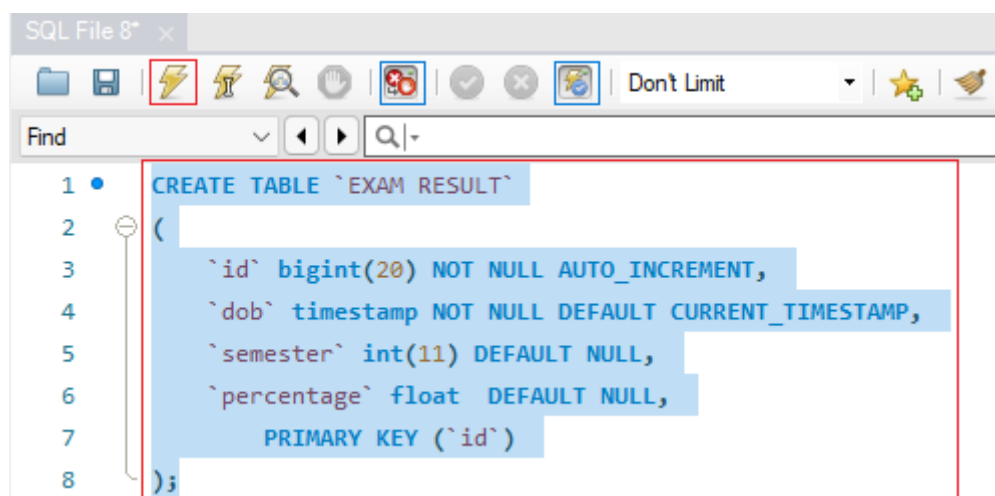
MySQL Connections + -



Step 2: Choose your Logical DB and then click on Create a new SQL tab



Step 3: To create the table copy & paste the following query select the query and click on Execute button or (CTRL + enter).



```
CREATE TABLE `exam_result`  
(  
    `id` bigint(20) NOT NULL AUTO_INCREMENT,  
    `dob` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,  
    `semester` int(11) DEFAULT NULL,  
    `percentage` float DEFAULT NULL,  
    PRIMARY KEY (`id`)  
);
```

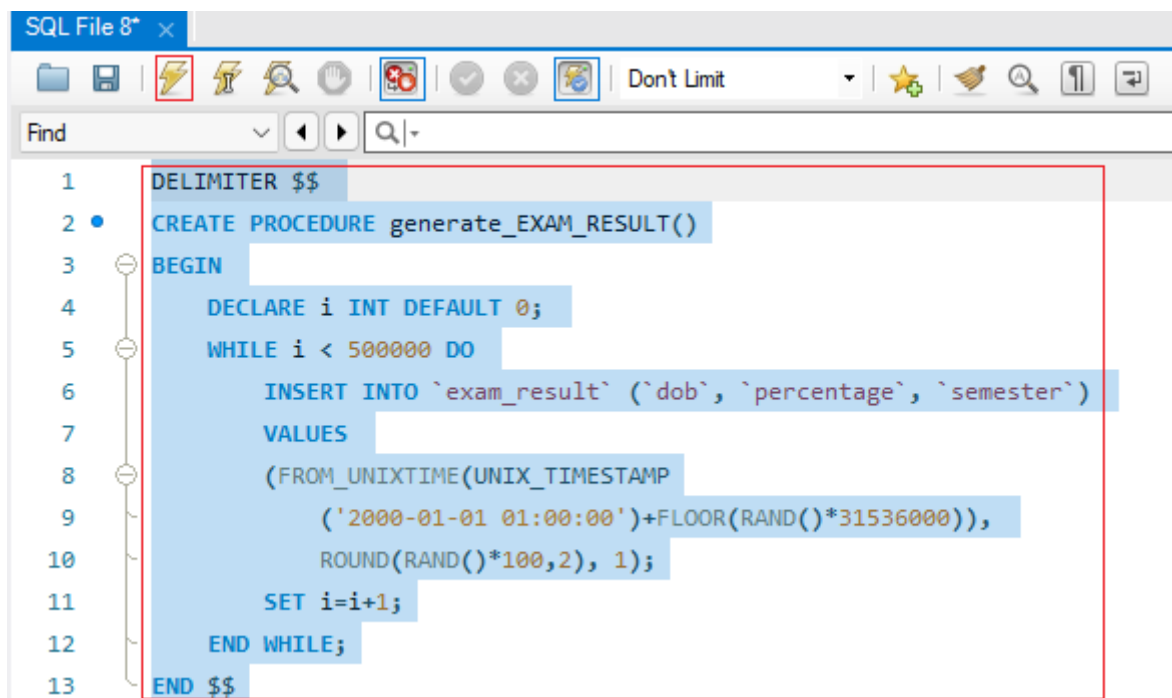
Step 4: Create a procedure to insert the record for that copy & paste the below query and click on Execute button.

```
DELIMITER $$  
CREATE PROCEDURE generate_EXAM_RESULT()  
BEGIN  
    DECLARE i INT DEFAULT 0;
```

```

WHILE i < 500000 DO
    INSERT INTO `exam_result` (`dob`, `percentage`, `semester`)
    VALUES
    (FROM_UNIXTIME(UNIX_TIMESTAMP('2000-01-01
01:00:00')+FLOOR(RAND()*31536000)), ROUND(RAND()*100,2), 1);
    SET i=i+1;
END WHILE;
END $$

```



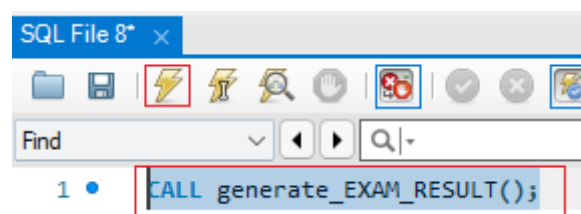
```

1  DELIMITER $$
2  CREATE PROCEDURE generate_EXAM_RESULT()
3  BEGIN
4      DECLARE i INT DEFAULT 0;
5      WHILE i < 500000 DO
6          INSERT INTO `exam_result` (`dob`, `percentage`, `semester`)
7          VALUES
8          (FROM_UNIXTIME(UNIX_TIMESTAMP
9          ('2000-01-01 01:00:00')+FLOOR(RAND()*31536000)),
10          ROUND(RAND()*100,2), 1);
11          SET i=i+1;
12      END WHILE;
13  END $$

```

Step 5: Call Procedure to Insert 50K data/ records in that table, it will take at least 30-45 Min to Insert the records, for that Execute the, below query.

```
CALL generate_EXAM_RESULT();
```



```

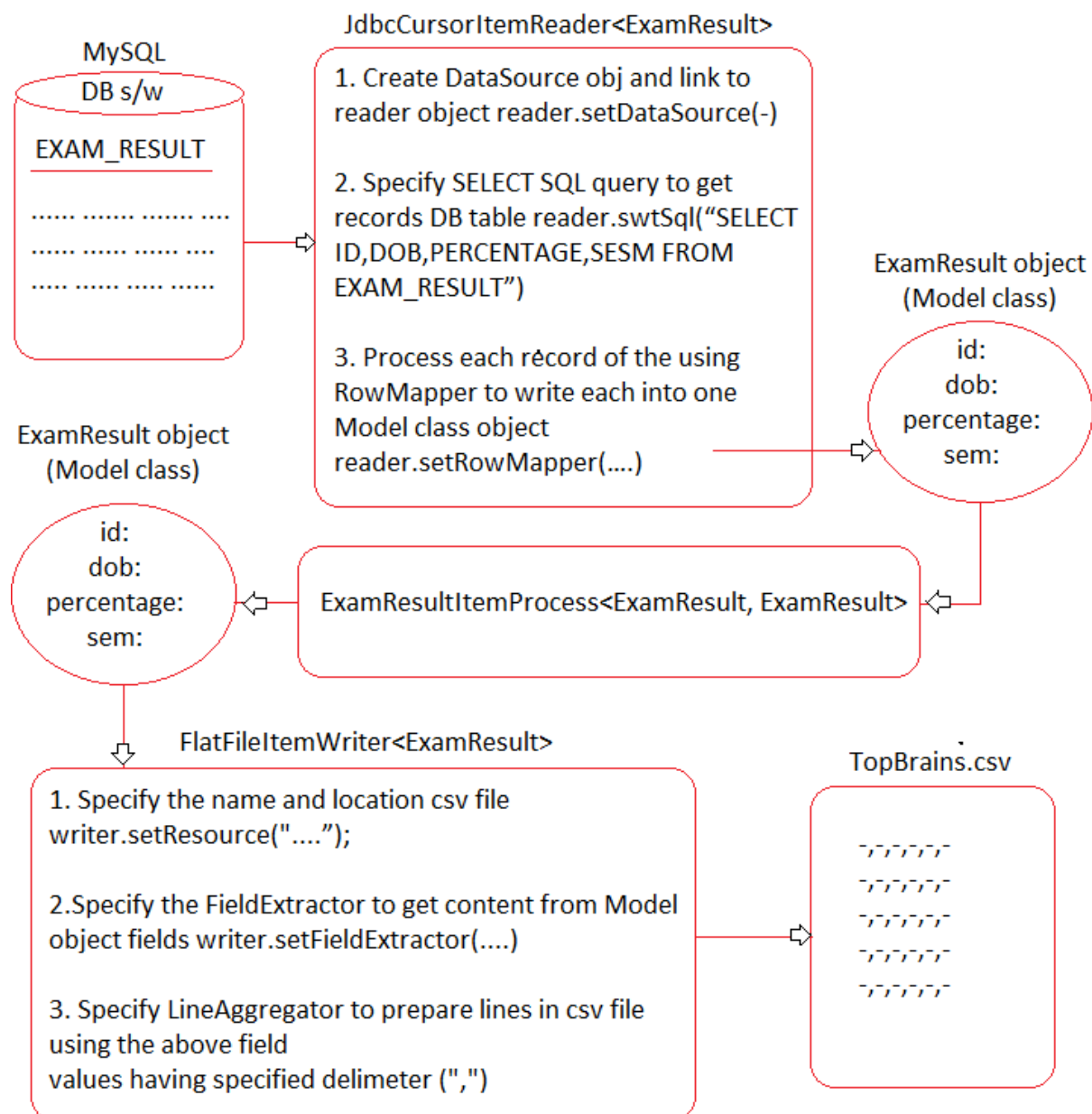
1  CALL generate_EXAM_RESULT();

```

Step 6: Check the Records, execute the below query.

```
select * from exam_result;
```

Story board of batching (DB table records to csv file)



Note: By RowMapper either separate class or inner class or anonymous inner class or LAMDA expression inner class we can convert each record coming to RS object into given Model class object.

@Override

```

public ExamResult mapRow(ResultSet rs, int rowNum) throws
                        SQLException {
    return new ExamResult(rs.getInt(1), rs.getDate(2),
                        rs.getDouble(3), rs.getInt(4));
}
  
```

RowMapper<T>

|--> public <T> mapRow(ResultSet rs, int rowNum) throws SQLException

(Functional interface because it is having only one method declaration).

Syntax for Lambda based interface implementation class object creation:

<Interface Name> ref = (<params>) -> {Body of the method};

Lambda based Anonymous inner class

```
RowMapper<ExamResult> mapper = (rs, rowNum) -> {  
    return new ExamResult(rs.getInt(1), rs.getDate(2),  
                           rs.getDouble(3), rs.getInt(4));  
};
```

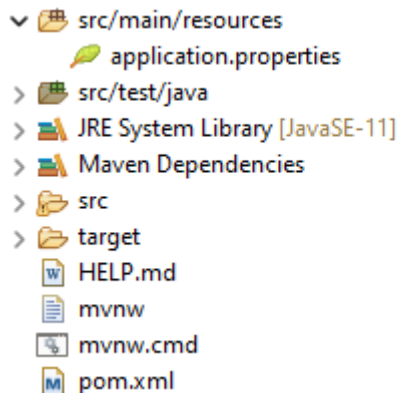
```
reader.setRowMapper((rs, rowNum)->{  
    return new ExamResult(rs.getInt(1), rs.getDate(2), rs.getDouble(3),  
                           rs.getInt(4));  
});
```

In this code following operations are performed,

- Anonymous inner class is created implementing RowMapper<T>.
- In that inner class mapRow(rs, rowNum) method is implemented having logic to copy ResultSet(rs) record to model class object.
- Anonymous inner class object is created and passed to reader.setRowMapper(-) as argument value.

Directory Structure of BootBatchProj03-DBTableToCSV:

```
▼ BootBatchProj03-DBTableToCSV [boot]  
  > Spring Elements  
  ▼ src/main/java  
    ▼ com.sahu  
      > BootBatchProj03DbTableToCsvApplication.java  
    ▼ com.sahu.config  
      > BatchConfig.java  
    ▼ com.sahu.listener  
      > JobMonitoringListener.java  
    ▼ com.sahu.mapper  
      > ExamResultRowMapper.java  
    ▼ com.sahu.model  
      > ExamResult.java  
    ▼ com.sahu.processor  
      > ExamResultItemProcessor.java  
    ▼ com.sahu.runner  
      > SpringBatchRunner.java
```



- Develop the above directory structure using Spring Starter Project option (Packaging: Jar) and create the packages and classes also.
- Add the following starter during project creation.
 - X Lombok
 - X Spring Batch
 - X JDBC API
 - X MySQL Driver
- By, following the above steps create the tables with data.
- Then place the following code with in their respective files.

application.properties

```
#Spring Batch configuration
spring.batch.job.enabled=false
spring.batch.jdbc.initialize-schema=always

#Database configuration
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql:///EXAM_DATA
spring.datasource.username=root
spring.datasource.password=root
```

ExamResult.java

```
package com.sahu.model;

import java.sql.Date;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
```

```

@NoArgsConstructor
@AllArgsConstructor
public class ExamResult {
    private Integer id;
    private Date dob;
    private Double percentage;
    private Integer semester;
}

```

JobMonitoringListener.java

```

package com.sahu.listener;

import java.util.Date;

import org.springframework.batch.core.JobExecution;
import org.springframework.batch.core.JobExecutionListener;
import org.springframework.stereotype.Component;

@Component
public class JobMonitoringListener implements JobExecutionListener {

    private long startTime, endTime;

    @Override
    public void beforeJob(JobExecution jobExecution) {
        startTime = System.currentTimeMillis();
        System.out.println("Job is about to start at : " + new Date());
    }

    @Override
    public void afterJob(JobExecution jobExecution) {
        endTime = System.currentTimeMillis();
        System.out.println("Job has completed at : " + new Date());
        System.out.println("Job Execution time : " + (endTime -
startTime) + "ms");
        System.out.println("Job Execution status : " +
jobExecution.getStatus());
    }
}

```


ExamResultItemProcessor.java

```
package com.sahu.processor;

import org.springframework.batch.item.ItemProcessor;

import com.sahu.model.ExamResult;

public class ExamResultItemProcessor implements
ItemProcessor<ExamResult, ExamResult> {

    @Override
    public ExamResult process(ExamResult item) throws Exception {
        if (item.getPercentage()>=75)
            return item;
        return null;
    }
}
```

ExamResultRowMapper.java

```
package com.sahu.mapper;

import java.sql.ResultSet;
import java.sql.SQLException;

import org.springframework.jdbc.core.RowMapper;

import com.sahu.model.ExamResult;

public class ExamResultRowMapper implements RowMapper<ExamResult>
{

    @Override
    public ExamResult mapRow(ResultSet rs, int rowNum) throws
SQLException {
        return new ExamResult(rs.getInt(1), rs.getDate(2),
rs.getDouble(3), rs.getInt(4));
    }
}
```

BatchConfig.java

```
package com.sahu.config;

import javax.sql.DataSource;

import org.springframework.batch.core.Job;
import org.springframework.batch.core.JobExecutionListener;
import org.springframework.batch.core.Step;
import org.springframework.batch.core.configuration.annotation.EnableBatchProcessing;
import org.springframework.batch.core.configuration.annotation.JobBuilderFactory;
import org.springframework.batch.core.configuration.annotation.StepBuilderFactory;
import org.springframework.batch.core.launch.support.RunIdIncrementer;
import org.springframework.batch.item.ItemProcessor;
import org.springframework.batch.item.database.JdbcCursorItemReader;
import org.springframework.batch.item.database.builder.JdbcCursorItemReaderBuilder;
import org.springframework.batch.item.file.FlatFileItemWriter;
import org.springframework.batch.item.file.transform.BeanWrapperFieldExtractor;
import org.springframework.batch.item.file.transform.DelimitedLineAggregator;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.core.io.FileSystemResource;

import com.sahu.listener.JobMonitoringListener;
import com.sahu.model.ExamResult;
import com.sahu.processor.ExamResultItemProcessor;

@EnableBatchProcessing
@Configuration
public class BatchConfig {
```

```

@Autowired
private DataSource dataSource;

@Autowired
private JobBuilderFactory jobBuilderFactory;

@Autowired
private StepBuilderFactory stepBuilderFactory;

//Listener
@Bean
public JobExecutionListener createListener() {
    return new JobMonitoringListener();
}

//Processor
@Bean
public ItemProcessor<ExamResult, ExamResult> createProcessor() {
    return new ExamResultItemProcessor();
}

/*@Bean
public JdbcCursorItemReader<ExamResult> createReader() {
    //Create object
    JdbcCursorItemReader<ExamResult> reader = new
JdbcCursorItemReader<>();
    //Specify DataSource
    reader.setDataSource(dataSource);
    //Specify SQL query
    reader.setSql("SELECT ID,DOB,PERCENTAGE,SEMESTER FROM
EXAM_RESULT");
    //Specify RowMapper
    //reader.setRowMapper(new ExamResultRowMapper());
    reader.setRowMapper((rs, rowNum)->{
        return new ExamResult(rs.getInt(1), rs.getDate(2),
rs.getDouble(3), rs.getInt(4));
    });
    return reader;
}*/

```

```

@Bean
public JdbcCursorItemReader<ExamResult> createReader() {
    return new JdbcCursorItemReaderBuilder<ExamResult>()
        .name("jdbc-reader")
        .dataSource(dataSource)
        .sql("SELECT ID,DOB,PERCENTAGE,SEMESTER
FROM EXAM_RESULT")
        .beanRowMapper(ExamResult.class)//Internally
use BeanPropertyRowMapper to convert the record of
//RS to given Model class object but DB table
column name and Model class properties must match
        .build();
}

//Writer
@Bean
public FlatFileItemWriter<ExamResult> createWriter(){
    FlatFileItemWriter<ExamResult> writer = new
FlatFileItemWriter<ExamResult>();
    //Set Logical name
    //writer.setName("writer-csv");
    //Specify the destination CSV file location
    //writer.setResource(new ClassPathResource("topbrain.csv"));
    writer.setResource(new
FileSystemResource("d:\\csv\\topbrain.csv"));
    //Specify LineAggregator
    writer.setLineAggregator(new
DelimitedLineAggregator<ExamResult>() {
        {
            //Delimiter
            setDelimiter(",");
            //Filed extractor
            setFieldExtractor(new
BeanWrapperFieldExtractor<ExamResult>() {
                {
                    setNames(new String[] {"id", "dob",
"percentage", "semester"});
                }
            });
        }
    }
}

```

```

    });

    return writer;
}

/*
//Writer
@Bean
public FlatFileItemWriter<ExamResult> createWriter(){
    return new FlatFileItemWriterBuilder<ExamResult>()
        .name("Line123")
        .resource(new ClassPathResource("topbrain.csv"))
        .lineSeparator("\r\n")
        .delimited().delimiter(",")
        .names("id", "dob", "percentage", "semester")
        .build();
}
*/

//Step
@Bean(name = "Step1")
public Step createStep1() {
    return stepBuilderFactory.get("Step1")
        .<ExamResult, ExamResult>chunk(3)
        .reader(createReader())
        .writer(createWriter())
        .processor(createProcessor())
        .build();
}

//Job
@Bean(name = "Job1")
public Job createJob1() {
    return jobBuilderFactory.get("Job1")
        .incrementer(new RunIdIncrementer())
        .listener(createListener())
        .start(createStep1())
        .build();
}
}

```

SpringBatchRunner.java

```
package com.sahu.runner;

import org.springframework.batch.core.Job;
import org.springframework.batch.core.JobExecution;
import org.springframework.batch.core.JobParameters;
import org.springframework.batch.core.JobParametersBuilder;
import org.springframework.batch.core.launch.JobLauncher;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;

@Component
public class SpringBatchRunner implements CommandLineRunner {

    @Autowired
    private JobLauncher launcher;

    @Autowired
    private Job job;

    @Override
    public void run(String... args) throws Exception {
        //Prepare JobParameters
        JobParameters parameter = new JobParametersBuilder()
            .addLong("time",
                System.currentTimeMillis())
            .toJobParameters();

        //Run the job
        try {
            JobExecution execution = launcher.run(job, parameter);
            System.out.println("Job completion status : 
"+execution.getStatus());
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

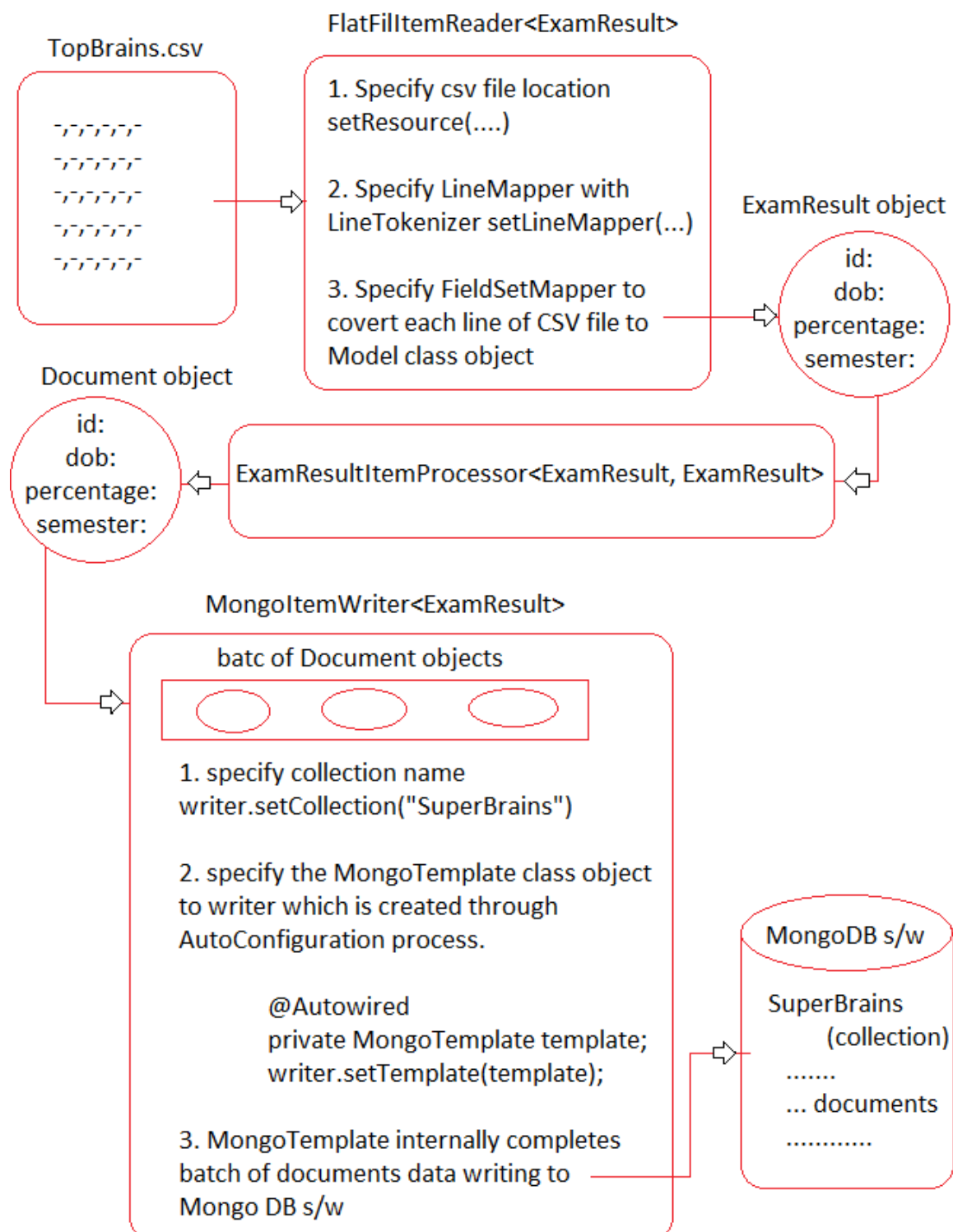
Spring Batch application to convert CSV file data to MongoDB documents

Writer: `MongoItemWriter<T>`

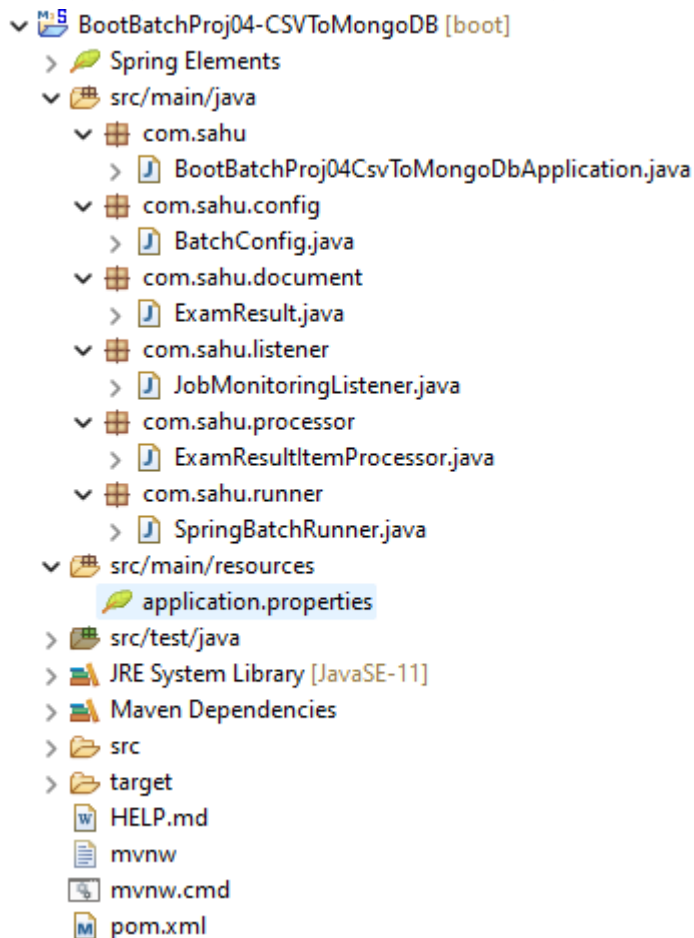
`<T>`: Document class (java bean with `@Document`)

Reader: `FlatFileItemReader<T>`

`<T>`: Model class (java bean)



Directory Structure of BootBatchProj04-CSVToMongoDB:



- Develop the above directory structure using Spring Starter Project option (Packaging: Jar) and create the packages and classes also.
- Add the following starter during project creation.
 - X Lombok
 - X Spring Batch
 - X Spring Data MongoDB
 - X H2 Database
- Collect JobMonitoringListener.java and SpringBatchRunner.java from previous project.
- Then place the following code with in their respective files.

application.properties

```
#Spring Batch configuration
spring.batch.job.enabled=false
spring.batch.jdbc.initialize-schema=always

#MongoDB setting
spring.data.mongodb.host=localhost
spring.data.mongodb.port=27017
```



```
spring.data.mongodb.database=NTSPBMS714DB
spring.data.mongodb.username=testuser
spring.data.mongodb.password=testuser
```

ExamResult.java

```
package com.sahu.document;

import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Document
@Data
@NoArgsConstructor
@AllArgsConstructor
public class ExamResult {
    @Id
    private Integer id;
    private String dob;
    private Float percentage;
    private Integer semester;
}
```

ExamResultItemProcessor.java

```
package com.sahu.processor;

import org.springframework.batch.item.ItemProcessor;

import com.sahu.document.ExamResult;

public class ExamResultItemProcessor implements
ItemProcessor<ExamResult, ExamResult> {

    @Override
    public ExamResult process(ExamResult item) throws Exception {
        if (item.getPercentage()>=90)
            return item;
    }
}
```

```
        return null;
    }
}
```

BatchConfig.java

```
package com.sahu.config;

import org.springframework.batch.core.Job;
import org.springframework.batch.core.JobExecutionListener;
import org.springframework.batch.core.Step;
import org.springframework.batch.core.configuration.annotation.EnableBatchProcessing;
import org.springframework.batch.core.configuration.annotation.JobBuilderFactory;
import org.springframework.batch.core.configuration.annotation.StepBuilderFactory;
import org.springframework.batch.core.launch.support.RunIdIncrementer;
import org.springframework.batch.item.ItemProcessor;
import org.springframework.batch.item.data.MongoItemWriter;
import org.springframework.batch.item.file.FlatFileItemReader;
import org.springframework.batch.item.file.mapping.BeanWrapperFieldSetMapper;
import org.springframework.batch.item.file.mapping.DefaultLineMapper;
import org.springframework.batch.item.file.transform.DelimitedLineTokenizer;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.core.io.FileSystemResource;
import org.springframework.data.mongodb.core.MongoTemplate;

import com.sahu.document.ExamResult;
import com.sahu.listener.JobMonitoringListener;
import com.sahu.processor.ExamResultItemProcessor;
```

```

@Configuration
@EnableBatchProcessing
public class BatchConfig {

    @Autowired
    private JobBuilderFactory jobBuilderFactory;

    @Autowired
    private StepBuilderFactory stepBuilderFactory;

    @Autowired
    private MongoTemplate template;

    //Listener
    @Bean
    public JobExecutionListener createListener() {
        return new JobMonitoringListener();
    }

    //Processor
    @Bean
    public ItemProcessor<ExamResult, ExamResult> createProcessor() {
        return new ExamResultItemProcessor();
    }

    //Reader
    @Bean
    public FlatFileItemReader<ExamResult> createReader() {
        FlatFileItemReader<ExamResult> reader = new
FlatFileItemReader<ExamResult>();
        reader.setResource(new
FileSystemResource("d:/csv/topbrain.csv"));
        reader.setLineMapper(new DefaultLineMapper<ExamResult>())
{{
            setLineTokenizer(new DelimitedLineTokenizer() {
                {
                    setDelimiter(",");
                    setNames("id","dob", "percentage",
"semester");
                }
            });
}}

```

```

        setFieldSetMapper(new
BeanWrapperFieldSetMapper<ExamResult>() {
            {
                setTargetType(ExamResult.class);
            }
        });
    });
    return reader;
}

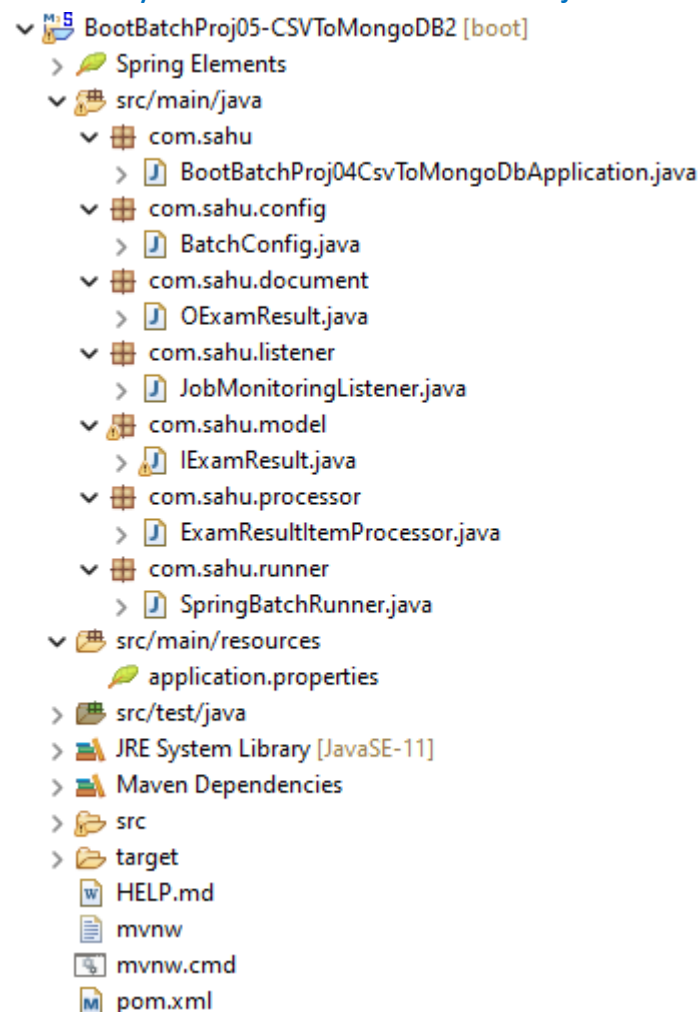
//Writer
@Bean
public MongolItemWriter<ExamResult> createWriter() {
    MongolItemWriter<ExamResult> writer = new
MongolItemWriter<ExamResult>();
    writer.setCollection("SuperBrains");
    writer.setTemplate(template);
    return writer;
}

//Step
@Bean(name = "Setp1")
public Step createStep1() {
    return stepBuilderFactory.get("Step1")
        .<ExamResult, ExamResult>chunk(3)
        .reader(createReader())
        .writer(createWriter())
        .processor(createProcessor())
        .build();
}

//Job
@Bean(name = "Job1")
public Job createJob1() {
    return jobBuilderFactory.get("Job1")
        .incrementer(new RunIdIncrementer())
        .listener(createListener())
        .start(createStep1())
        .build();
}
}

```

Directory Structure of BootBatchProj05-CSVToMongoDB2:



- Copy & pastes the previous project and change add the package and class according to new structure.
- Then place the following code with in their respective files.

OExamResult.java

```
package com.sahu.document;  
  
import java.time.LocalDate;  
  
import org.springframework.data.annotation.Id;  
import org.springframework.data.mongodb.core.mapping.Document;  
  
import lombok.AllArgsConstructor;  
import lombok.Data;  
import lombok.NoArgsConstructor;
```

```
@Document
@Data
@NoArgsConstructor
@AllArgsConstructor
public class OExamResult {
    @Id
    private Integer id;
    private LocalDate dob;
    private Float percentage;
    private Integer semester;
}
```

IExamResult.java

```
package com.sahu.model;

import org.springframework.data.annotation.Id;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
@AllArgsConstructor
public class IExamResult {

    private Integer id;
    private String dob;
    private Float percentage;
    private Integer semester;
}
```

IExamResult.java

```
package com.sahu.processor;

import java.time.LocalDate;

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

```

import com.sahu.document.OExamResult;
import com.sahu.model.IExamResult;

public class ExamResultItemProcessor implements
ItemProcessor<IExamResult, OExamResult> {

    @Override
    public OExamResult process(IExamResult item) throws Exception {
        if (item.getPercentage()>=90) {
            OExamResult result = new OExamResult();
            result.setId(item.getId());
            result.setDob(LocalDate.parse(item.getDob()));
            result.setPercentage(item.getPercentage());
            result.setSemester(item.getSemester());
            return result;
        }
        return null;
    }
}

```

IExamResult.java

```

package com.sahu.config;

import org.springframework.batch.core.Job;
import org.springframework.batch.core.JobExecutionListener;
import org.springframework.batch.core.Step;
import
org.springframework.batch.core.configuration.annotation.EnableBatchProc
essing;
import
org.springframework.batch.core.configuration.annotation.JobBuilderFactory;
import
org.springframework.batch.core.configuration.annotation.StepBuilderFactory;
import org.springframework.batch.core.launch.support.RunIdIncrementer;
import org.springframework.batch.item.ItemProcessor;
import org.springframework.batch.item.data.MongoItemWriter;
import org.springframework.batch.item.file.FlatFileItemReader;

```

```

import
org.springframework.batch.item.file.mapping.BeanWrapperFieldSetMapper
;
import org.springframework.batch.item.file.mapping.DefaultLineMapper;
import
org.springframework.batch.item.file.transform.DelimitedLineTokenizer;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.core.io.FileSystemResource;
import org.springframework.data.mongodb.core.MongoTemplate;

import com.sahu.document.OExamResult;
import com.sahu.listener.JobMonitoringListener;
import com.sahu.model.IExamResult;
import com.sahu.processor.ExamResultItemProcessor;

```

```
@Configuration
```

```
@EnableBatchProcessing
```

```
public class BatchConfig {
```

```
    @Autowired
```

```
    private JobBuilderFactory jobBuilderFactory;
```

```
    @Autowired
```

```
    private StepBuilderFactory stepBuilderFactory;
```

```
    @Autowired
```

```
    private MongoTemplate template;
```

```
    //Listener
```

```
    @Bean
```

```
    public JobExecutionListener createListener() {
```

```
        return new JobMonitoringListener();
```

```
    }
```

```
    //Processor
```

```
    @Bean
```

```
    public ItemProcessor<IExamResult, OExamResult> createProcessor() {
```

```
        return new ExamResultItemProcessor();
```

```
    }
```



```

    }

    //Reader
    @Bean
    public FlatFileItemReader<IExamResult> createReader() {
        FlatFileItemReader<IExamResult> reader = new
FlatFileItemReader<IExamResult>();
        reader.setResource(new
FileSystemResource("d:/csv/topbrain.csv"));
        reader.setLineMapper(new DefaultLineMapper<IExamResult>()
{{
            setLineTokenizer(new DelimitedLineTokenizer() {
                {
                    setDelimiter(",");
                    setNames("id", "dob", "percentage",
"semester");
                }
            });
            setFieldSetMapper(new
BeanWrapperFieldSetMapper<IExamResult>() {
                {
                    setTargetType(IExamResult.class);
                }
            });
        }});
        return reader;
    }

    //Writer
    @Bean
    public MongolItemWriter<OExamResult> createWriter() {
        MongolItemWriter<OExamResult> writer = new
MongolItemWriter<OExamResult>();
        writer.setCollection("SuperBrains");
        writer.setTemplate(template);
        return writer;
    }

    //Step
    @Bean(name = "Setp1")

```

```

public Step createStep1() {
    return stepBuilderFactory.get("Step1")
        .<IExamResult, OExamResult>chunk(3)
        .reader(createReader())
        .writer(createWriter())
        .processor(createProcessor())
        .build();
}

//Job
@Bean(name = "Job1")
public Job createJob1() {
    return jobBuilderFactory.get("Job1")
        .incrementer(new RunIdIncrementer())
        .listener(createListener())
        .start(createStep1())
        .build();
}
}

```

How to enable Cron expression scheduling on Batch processing

Step 1: Place Cron expression in the application.properties as key-value (custom key).

Step 2: Place @EnableScheduling on the main class @SpringBootApplication.

Step 3: Develop service/ Spring bean class having business method to run the Job by enabling scheduling.

Directory Structure of BootBatchProj05-CSVToMongoDB-CronExpression:

- Copy & pastes the previous project and remove the runner class and create a simple Spring bean class as below.

```

└─ com.sahu.runner
    └─ BatchProcessTest.java

```

- Then place the following code with in their respective files.

application.properties

```

cron.expr=0/50 * * * * *

```

BootBatchProj04CsvToMongoDbApplication.java

```
@SpringBootApplication
@EnableScheduling
public class BootBatchProj04CsvToMongoDbApplication {

    public static void main(String[] args) {

        SpringApplication.run(BootBatchProj04CsvToMongoDbApplication.class, args);
    }

}
```

BatchProcessTest.java

```
package com.sahu.runner;

import org.springframework.batch.core.Job;
import org.springframework.batch.core.JobExecution;
import org.springframework.batch.core.JobParameters;
import org.springframework.batch.core.JobParametersBuilder;
import org.springframework.batch.core.launch.JobLauncher;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Component;

@Component
public class BatchProcessTest {

    @Autowired
    private JobLauncher launcher;

    @Autowired
    private Job job;

    @Scheduled(cron = "${cron.expr}")
    public void runJob() throws Exception {
        //Prepare JobParameters
        JobParameters parameter = new JobParametersBuilder()
            .addLong("time",
                System.currentTimeMillis())
    }
```

```

System.currentTimeMillis()).toJobParameters();
    //Run the job
    try {
        JobExecution execution = launcher.run(job, parameter);
        System.out.println("Job completion status :
"+execution.getStatus());
    }
    catch (Exception e) {
        e.printStackTrace();
    }
}
}

```

Converting CSV file data to Oracle DB table records using Spring Data JPA (Hibernate)

Reader: FlatFileItemReader <T> (needs LineMapper with LineTokenizer, FieldSetMapper)

Writer: JpaItemWriter<T> (needs EntityManager)

Model class1 (To read from CSV file)

@Data

```

public class IExamResult {
    public Integer id;
    public String dob;
    public Float percentage;
    public Integer semester;
}

```

Model class2 (To write into DB table records)

@Data

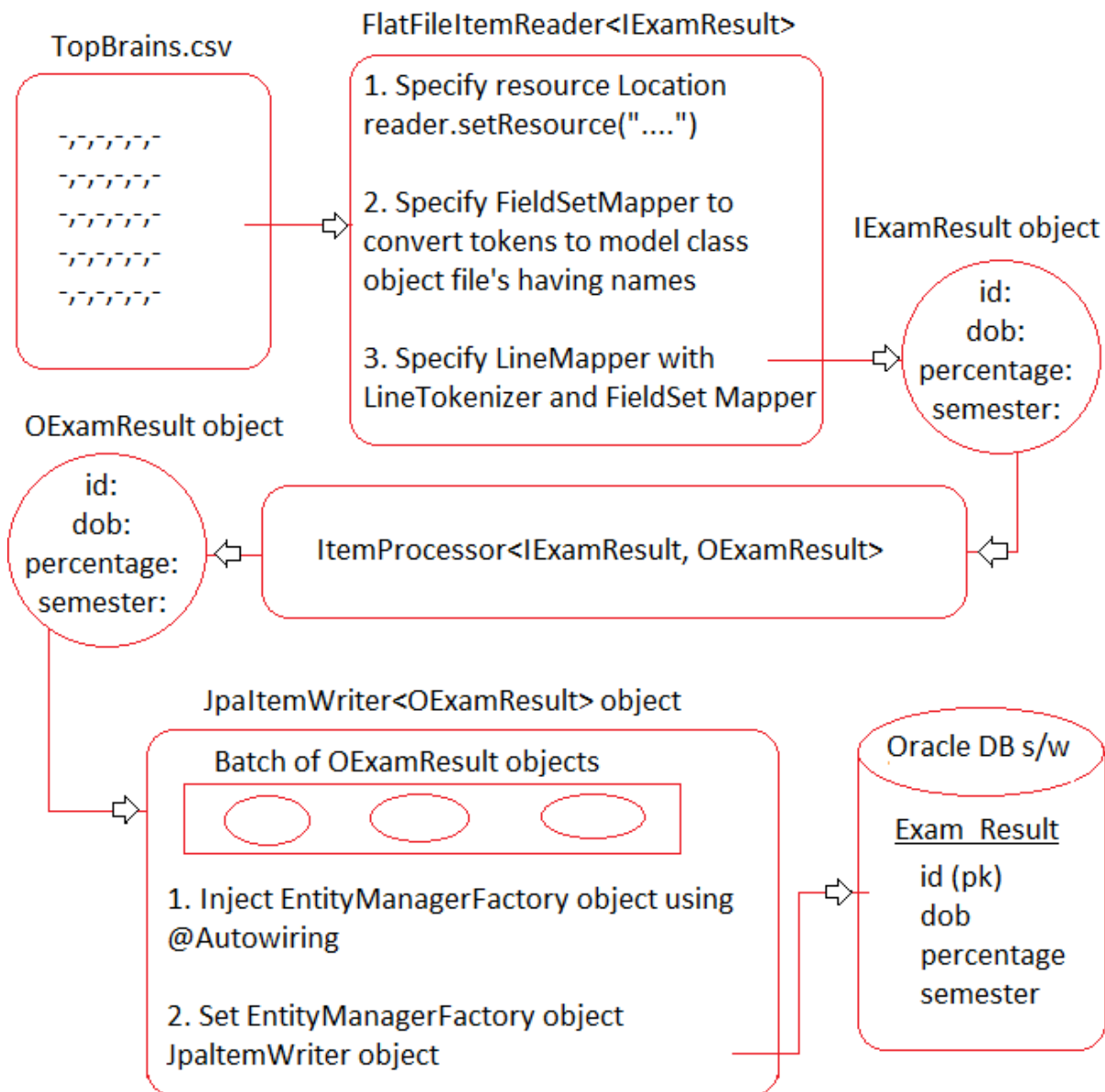
@Entity

@Table(name= "Exam_result")

```

public class OExamResult {
    @Id
    public Integer id;
    public LocalDate dob;
    public Float percentage;
    public Integer semester;
}

```



JobRepository contains multiple details about job executions

spring.batch.jdbc.initialize-schema=**always**

never

embedded

always:

- Batch Processing activities should be kept tracked by creating multiple DB tables in the underlying DB s/w.
- If no DataSource configuration then exception will be raised.
- If we configuration embedded DB s/w like H2, HSQL and etc. then the DB tables will be created in that Embedded DB s/w.

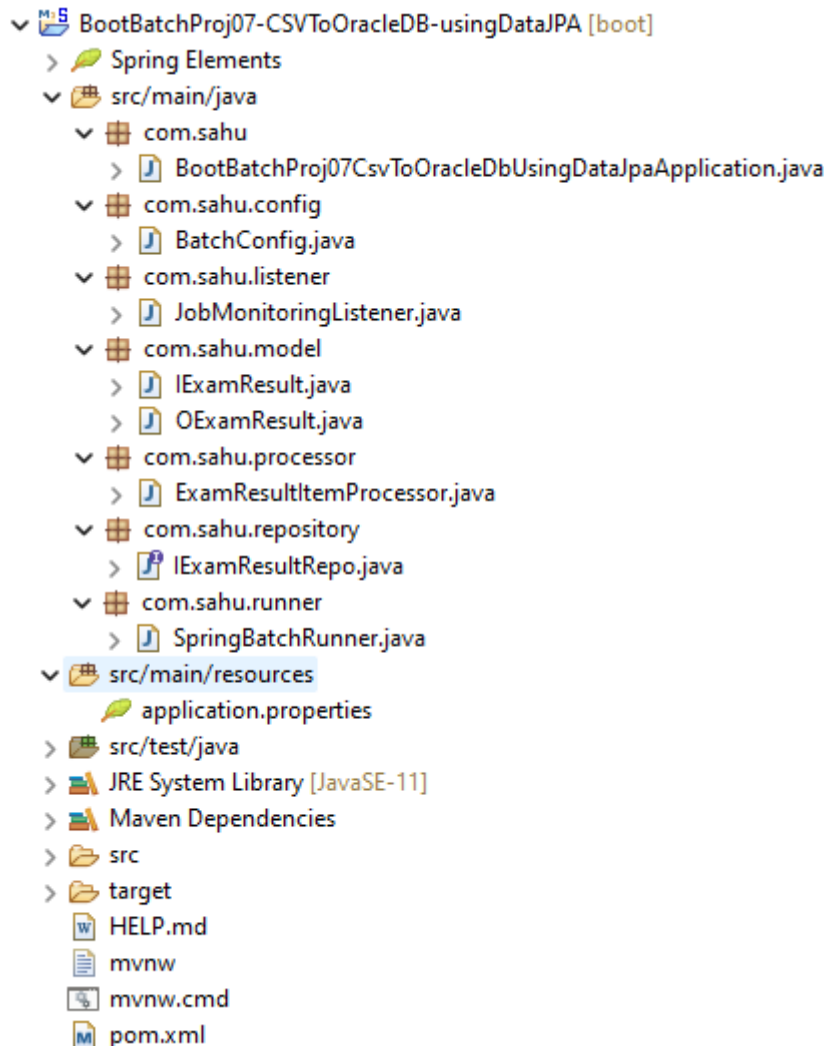
never:

- Batch processing activities should not be kept tracked.

embedded:

- Batch processing activities should keep tracked only in the Embedded DB s/w through external DB s/w are configured.

Directory Structure of BootBatchProj07-CSVToOracleDB-usingDataJPA



- Develop the above directory structure using Spring Starter Project option and create the package and classes also.
- Use the following starters during project creation.
 - X Lombok
 - X Spring Batch
 - X Spring Data JPA
 - X Oracle Driver
- Copy the JobMonitoringListener.java and SpringBatchRunner.java class from previous projects.
- Then place the following code with in their respective files.

application.properties

```
#Spring Batch configuration
spring.batch.job.enabled=false
spring.batch.jdbc.initialize-schema=always

#DataSource configuration
spring.datasource.driver-class-name=oracle.jdbc.driver.OracleDriver
spring.datasource.url=jdbc:oracle:thin:@localhost:1521:xe
spring.datasource.username=system
spring.datasource.password=manager

#Data JPA configurations
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.format_sql=true
```

OExamResult.java

```
package com.sahu.model;

import java.time.LocalDate;

import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name = "EXAM_RESULT")
public class OExamResult {
    @Id
    private Integer id;
    private LocalDate dob;
    private Float percentage;
    private Integer semester;
}
```

IExamResult.java

```
package com.sahu.model;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
@AllArgsConstructor
public class IExamResult {
    private Integer id;
    private String dob;
    private Float percentage;
    private Integer semester;
}
```

IExamResultRepo.java

```
package com.sahu.repository;

import org.springframework.data.jpa.repository.JpaRepository;

import com.sahu.model.OExamResult;

public interface IExamResultRepo extends JpaRepository<OExamResult, Integer> {

}
```

ExamResultItemProcessor.java

```
package com.sahu.processor;

import java.time.LocalDate;

import org.springframework.batch.item.ItemProcessor;

import com.sahu.model.IExamResult;
import com.sahu.model.OExamResult;
```



```

public class ExamResultItemProcessor implements
ItemProcessor<IExamResult, OExamResult> {

    @Override
    public OExamResult process(IExamResult item) throws Exception {
        if (item.getPercentage()>=90) {
            OExamResult result = new OExamResult();
            result.setId(item.getId());
            result.setDob(LocalDate.parse(item.getDob()));
            result.setPercentage(item.getPercentage());
            result.setSemester(item.getSemester());
            return result;
        }
        return null;
    }
}

```

BatchConfig.java

```

package com.sahu.config;

import javax.persistence.EntityManagerFactory;

import org.springframework.batch.core.Job;
import org.springframework.batch.core.JobExecutionListener;
import org.springframework.batch.core.Step;
import
org.springframework.batch.core.configuration.annotation.EnableBatchProc
essing;
import
org.springframework.batch.core.configuration.annotation.JobBuilderFactory;
import
org.springframework.batch.core.configuration.annotation.StepBuilderFactory;
import org.springframework.batch.core.launch.support.RunIdIncrementer;
import org.springframework.batch.item.ItemProcessor;
import org.springframework.batch.item.database.JpaItemWriter;
import
org.springframework.batch.item.database.builder.JpaItemWriterBuilder;

```

```

import org.springframework.batch.item.file.FlatFileItemReader;
import
org.springframework.batch.item.file.builder.FlatFileItemReaderBuilder;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.core.io.FileSystemResource;

import com.sahu.listener.JobMonitoringListener;
import com.sahu.model.IExamResult;
import com.sahu.model.OExamResult;
import com.sahu.processor.ExamResultItemProcessor;

@Configuration
@EnableBatchProcessing
public class BatchConfig {

    @Autowired
    private JobBuilderFactory jobBuilderFactory;

    @Autowired
    private StepBuilderFactory stepBuilderFactory;

    @Autowired
    private EntityManagerFactory entityManagerFactory;

    //Listener
    @Bean
    public JobExecutionListener createListener() {
        return new JobMonitoringListener();
    }

    //Processor
    @Bean
    public ItemProcessor<IExamResult, OExamResult> createProcessor() {
        return new ExamResultItemProcessor();
    }

    //Reader
    /*@Bean

```

```

        public FlatFileItemReader<IExamResult> createReader() {
            FlatFileItemReader<IExamResult> reader = new
FlatFileItemReader<IExamResult>();
            reader.setResource(new
FileSystemResource("d:/csv/topbrain.csv"));
            reader.setLineMapper(new DefaultLineMapper<IExamResult>())
{{
                setLineTokenizer(new DelimitedLineTokenizer() {
                    {
                        setDelimiter(",");
                        setNames("id", "dob", "percentage",
"semester");
                    }
                });
                setFieldSetMapper(new
BeanWrapperFieldSetMapper<IExamResult>() {
                    {
                        setTargetType(IExamResult.class);
                    }
                });
            }};
            return reader;
        }
    }
}

```

```

@Bean
public FlatFileItemReader<IExamResult> createReader() {
    return new FlatFileItemReaderBuilder<IExamResult>()
        .name("csv")
        .resource(new
FileSystemResource("d:/csv/topbrains.csv"))
        .delimited()
        .delimiter(",")
        .names("id", "dob", "percentage", "semester")
        .targetType(IExamResult.class)
        .build();
}

```

```

//Writer
/*@Bean
public JpitemWriter<OExamResult> createWriter() {

```

```

        JpaltemWriter<OExamResult> writer = new
JpaltemWriter<OExamResult>();
        //Set EntityManager factory
        writer.setEntityManagerFactory(entityManagerFactory);
        return writer;
    }*/

    @Bean
    public JpaltemWriter<OExamResult> createWriter() {
        return new JpaltemWriterBuilder<OExamResult>()
            .entityManagerFactory(entityManagerFactory)
            .build();
    }

    //Step
    @Bean(name = "Setp1")
    public Step createStep1() {
        return stepBuilderFactory.get("Step1")
            .<IExamResult, OExamResult>chunk(3)
            .reader(createReader())
            .writer(createWriter())
            .processor(createProcessor())
            .build();
    }

    //Job
    @Bean(name = "Job1")
    public Job createJob1() {
        return jobBuilderFactory.get("Job1")
            .incrementer(new RunIdIncrementer())
            .listener(createListener())
            .start(createStep1())
            .build();
    }
}

```

----- The END -----