# Spring batch

Processing finite amount of data/ batch of data without interaction or interruption

We can restart if that chunk fails

### **When to use spring batch**

If you are handling with huge volume of data then use spring batch – because it can process chunk by chunk

1. Suppose if bank wants to send BE-ALERT messages or current balance mails like our Citibank then use spring batch , because spring batch can fetch person records chunk by chunk and send the messages to the chunk of users-- else u cant send msgs to all people at once, if u do jvm will blast.
2. All the transactions happening under RTGS, NEFT , all the transactions will be stored and executed as a batch.
3. Some vulnerability scanner which will scan bulk files

Use cases:- generation of email statements, weather forecasting reports

Ex:- CSV to database

Item Reader

Item Processor

Item Writer

Step

Job

Job launcher

Job Repository

All the job status will be stored in job repository job ran or failed or the status ..& when job started & when job ended , job repo will store all that data into db

Features of spring batch

1. It will provides us the start job, stop the job options
2. Retry and skip mechanism

### Step object

1 job will have lot of steps, it can have either 1 or 100 steps, job is the main task , main task will have all

Here main task is money transfer, but it will have all these below steps

Create step using stepBuilderFactory (This factory will be available directly we can autowire directly)

@Autowired  
StepBuilderFactory **stepBuilderFactory**;

Debiting from 1 ac

Crediting to target account

Sending sms as transaction status

#### StepExecutionContext

It is the memory that is available across the entire step, for ex:- (components of step are – ItemReader, ItemProcessor, ItemWriter…)

U can set from ItemReader and use in ItemWriter

Here all the data which we set to Stepexecution\_context– will be stored in database once that step is completed immediately

For each step there will be separate step execution context (context is like memory).

And each step execution context data stored in table called stepexecution\_context

Because each step can have ItemReader, ItemProcessor, ItemWriter with this we can set values in ItemReader and we can fetch in ItemWriter

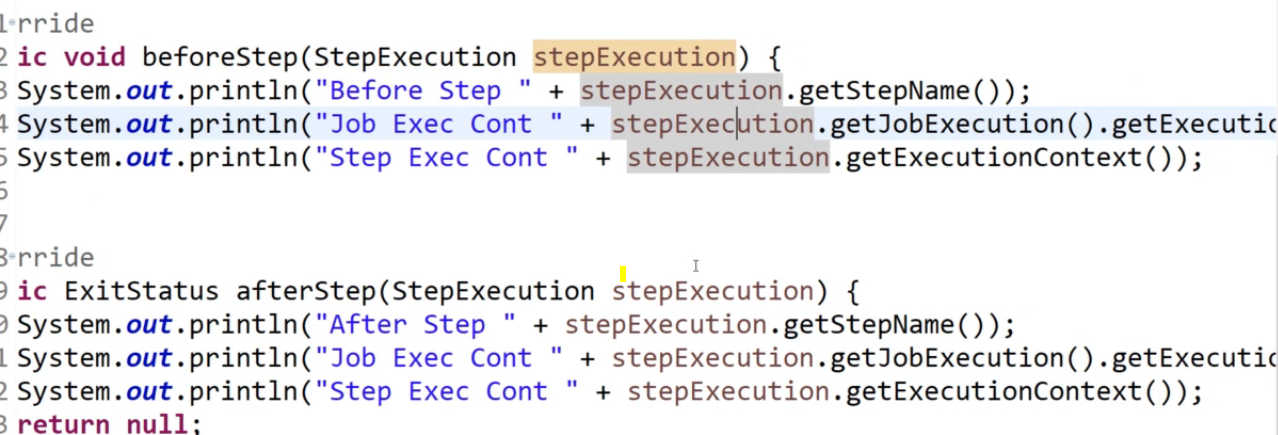
This is the memory available for that entire step.

This is also a map, which will be stored in database table

BATCH\_STEP\_EXECUTION\_CONTEXT refer schema-oracle10.sql in spring-batch-core-\*.jar all these sql files are present in that jar, dropping the

StepExecutionListener

This will be executed before and after executing the step



Dividing into chunks is mandatory, else memory out of bounds exceptions only, it says how many records u want to process at a time



Tasklet step

Mostly we wont use tasklet step, we will use chunk oriented step

Like if u have some job just to call a micro service, so to print 1-100 element, for that as we don’t need itemreader, ItemWriter then

In that cases instead of Chunk oriented step we can use Tasklet

*//Creating a tasklet step and use that step in the job* @Bean  
 public *Step* taskletOrientedStep(){  
 return **stepBuilderFactory**.get("Tasklet oriented step")  
 .tasklet(new *Tasklet*() {  
 @Override  
 public *RepeatStatus* execute(StepContribution *contribution*, ChunkContext *chunkContext*) throws Exception {  
 System.***out***.println("My Tasklet oriented step");  
 return *RepeatStatus*.***FINISHED***;  
 }  
 }).build();  
 }  
   
*// use the above step for our job* @Bean  
 public *Job* taskletJob(*Step taskletOrientedStep*){  
 System.***out***.println("Building a tasklet oriented stepped job");  
 return **jobBuilderFactory**.get("Tasklet oriented job")  
 .start(*taskletOrientedStep*)  
 .next(*taskletOrientedStep*) //wantedly used same task in next step  
 .build();  
 }

#### Chunk oriented step

ItemReader will read the chunk of records at a time from the excel file- means it will read 5/6/some chunk of records at a time from file- here read method will be invoked until chunk size is reached.so that’s why configure chunk size for each and every step, If u don’t configure itemreader will keep on reading and it will finally throw exception

ItemProcessor will be called for each and every item present in the chunk

### Job

It is the main task ex:- sending data from one to another destination, 1 job will have many steps

Think climbing steps Is a job- so that job will have multiple steps, not a single step.

Job is nothing but a process which run from start to finishes without any interuption

Job is also created by job BuilderFactory.

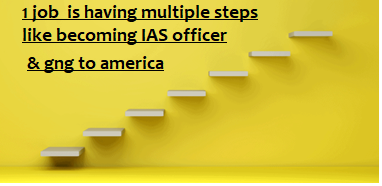
Job is broken up into steps- each step is independent,

Job means lot of steps – like if u want to become IAS officer u have lot of steps

* We can have a job with 🡪 Tasklet and chunk oriented step combo

Using job builder factory we will create job (jobBuilderFactory bean is readily created by spr batch auto configuration, we just need to autowire it)

@Autowired  
JobBuilderFactory **jobBuilderFactory**;

   
*// use the above step for our job, Here to create 1 job it involves lot of steps* @Bean  
 public *Job* taskletJob(*Step taskletOrientedStep*){  
 System.***out***.println("Building a tasklet oriented stepped job");  
 return **jobBuilderFactory**.get("Tasklet oriented job")  
 .start(*taskletOrientedStep*) //create Step-1  
 .next(*taskletOrientedStep*) //create Step -2  
 .build();  
 }

Ex:- to get software job

1. we have to complete B.Tech
2. learn all trainings in ameerpet
3. prepare resume
4. attend interviews

Job launcher

This is to start the actual job by taking some inputs from job parameters

#### JobExecutionContext

This is the memory which is available across all the steps of that job,

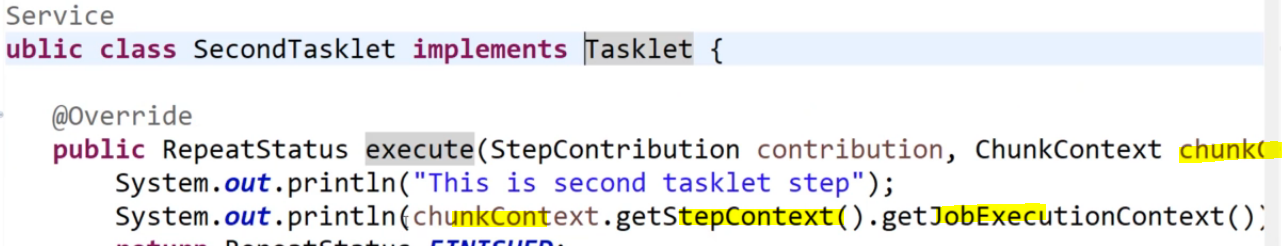
It is the memory that is available across all the steps, for ex:- if u want to use step-1 data in step-2 , then set the data to context and fetch in step2 from step execution context

Remember all that JobExecutionContext data which we set will be saved in db by spring batch framework

Using below, if u set something to jobContext



Then this is available as per above

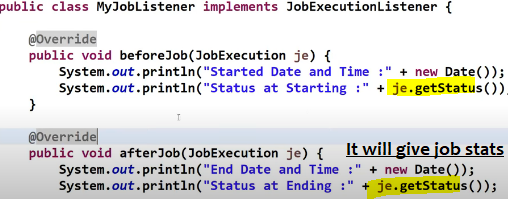


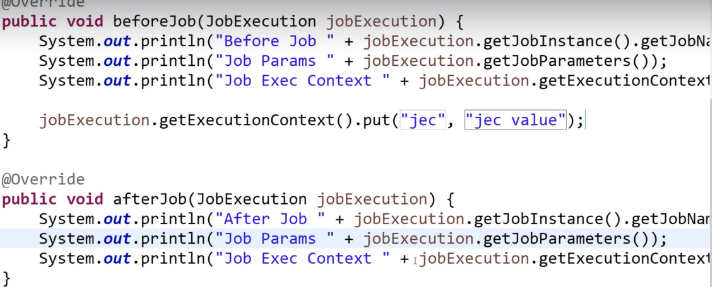
#### JobExecutionListener



This must be a spring bean, so better mark it as a spring bean, marking is not enough

We need to attach to the job





Job Instance & Job Execution

Job instance – means 1 day only 1 job instance – , for (1 instance, 1 execution) 1 job instance there will be only 1 successful execution

If a job is instantiated for that day

if that job execution failed, then it can re-start that job , then there will be another job execution

Job

Job Instance 2-May

Job Instance 1-May

Job Execution 1-May - restarted

Job Execution 1-May failed

Under 1 instance, there will be only 1 successful execution, 2nd time job wont execute as previous execution is success

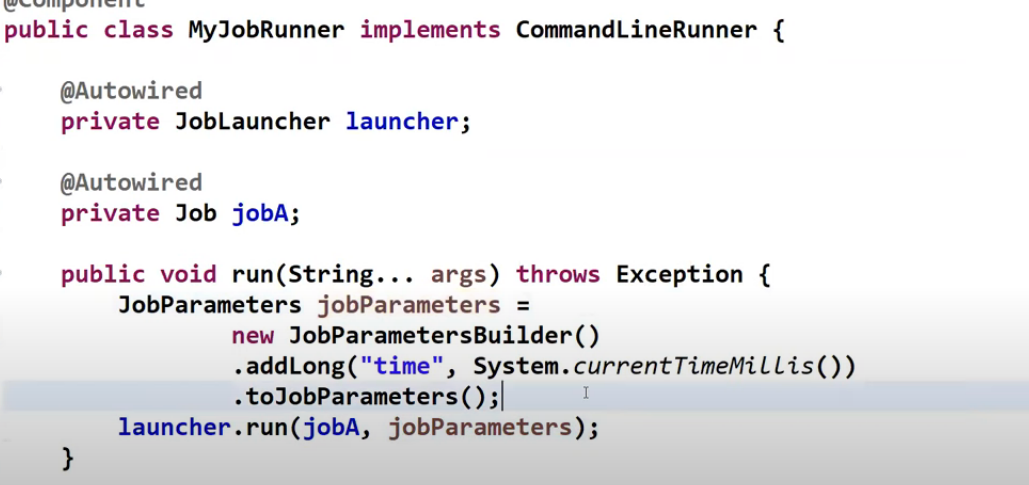
Job Repository

all the job statuses is maintained in the job repository, so that if any chunk fails , the job can restart from there

when job started, when its ended, status –success or failed

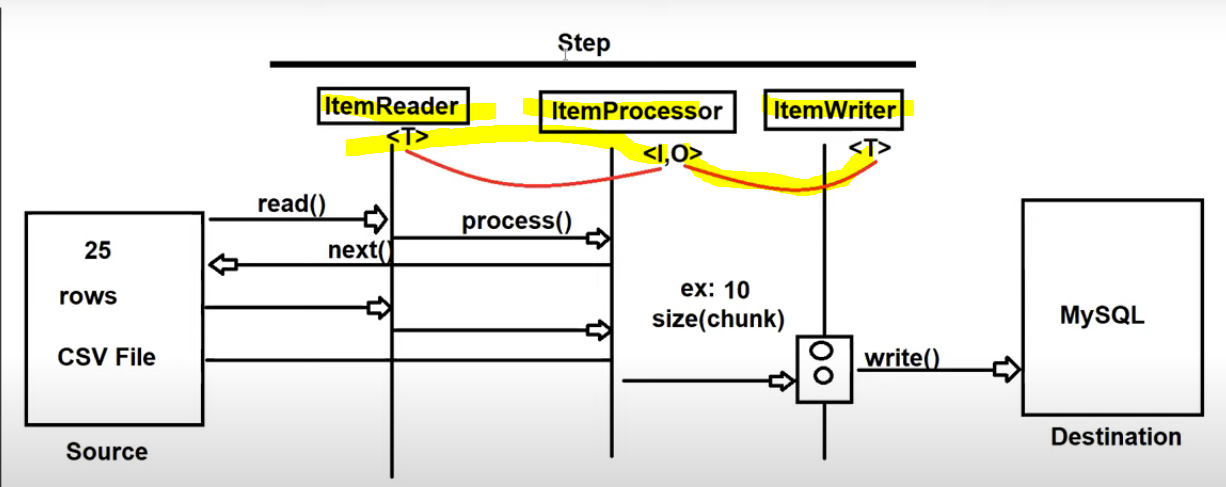
Run job using jobParametersBuilder

And job parameters is created by job parameterBuilder



### Item Reader or Item Processor vs Item Writer

Item is nothing but an individual record.



If 25 times reader executed, then 25 times Processor will be executed, whereas only 3 times the writer will be executed. Because 25 times the writer should not hit the database, we will mention the chunk size as 10, only when it has 10 records then it will hit the ItemWriter.

### ItemReader

In most of the scenarios we will use predefined readers only

ItemReader –the powerful feature is it will read chunk of data continuously

ex:- FlatFileItemReader (predefined ItemReader given by batch api used to read data from file and convert to objects)

Ex:- If chunk size is 20 , that item reader read() method will be executed 20 times continuously / will be reading until chunk size is met/ until chunk of records are read,

And it will keep on running/executing the reader until last element is null.

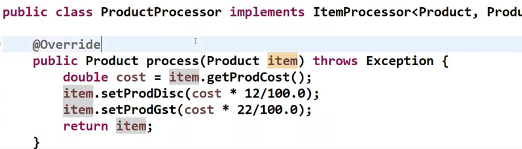
And of course, for each record it will hit ItemProcessor, so 20 times it will hit ItemProcessor

Ex:- reading data from csv and writing to MySQL

### ItemProcessor

The item processor is going to hold the output until the chunk size is met.

If there is no business logic , u can have a chunk oriented step without item processor.



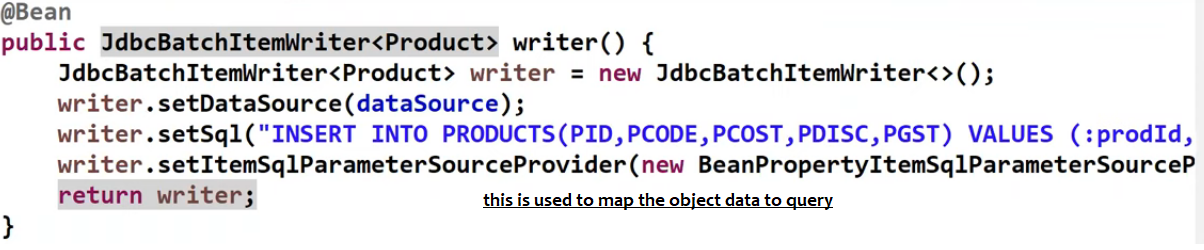
### ItemWriter

Here use predefined writer, it will receive the chunk of data at once, & it will write that full chunk at once to database

Ex:- JDBCBatchItemWriter – this is the predefined class given by batch to write data to database

ItemWriter will be invoked only once for that chunk –if chunk size is 10, it will be invoked once for that chunk

For that 10 records, because if item writer is writing to that db, it should not write each record once, it should write only once for that chunk



### Testing purpose

Use H2 database and to create a table using below query , if u want that query to be automatically be executed , place that query in src/main/resources/data.sql

create table PRODUCTS (PID INT ,PCODE VARCHAR(20),PCOST DOUBLE,PDISC DOUBLE,PGST DOUBLE );

### Strategy

To process millions of records , we don’t process them all at a time,

We process them in chunks and stores in database

### Coding

Jars

<dependency>  
 <groupId>com.mysql</groupId>  
 <artifactId>mysql-connector-j</artifactId>  
 <scope>runtime</scope>  
</dependency>

To work with my sql we should add above jar

#### App properties

spring.batch.jdbc.initialize-schema =always

means everytime it wont create tables – if tables are already present it will leave as it is, but is absent then it will create tables required for spring batch

Annotations

1. @EnableBatchProcessing