# Spring batch

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

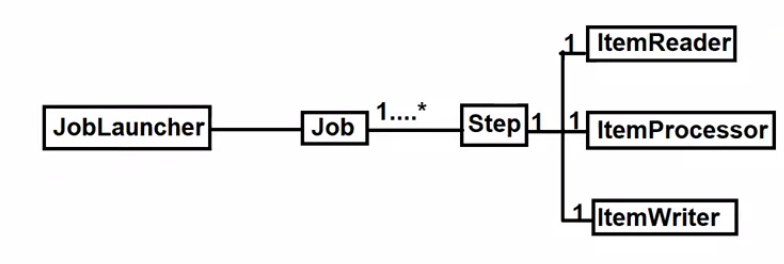
We can restart if that chunk fails

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

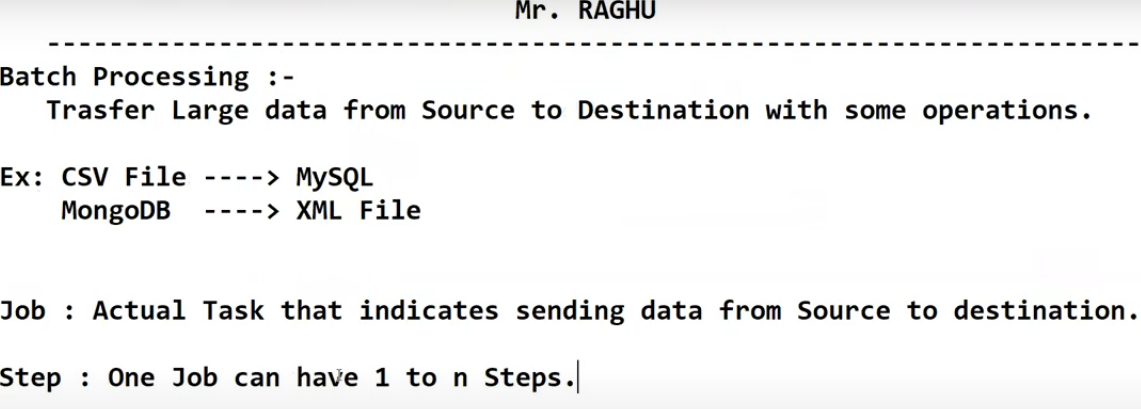
If you are handling with huge volume of data then use spring batch – because it can process chunk by chunk

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

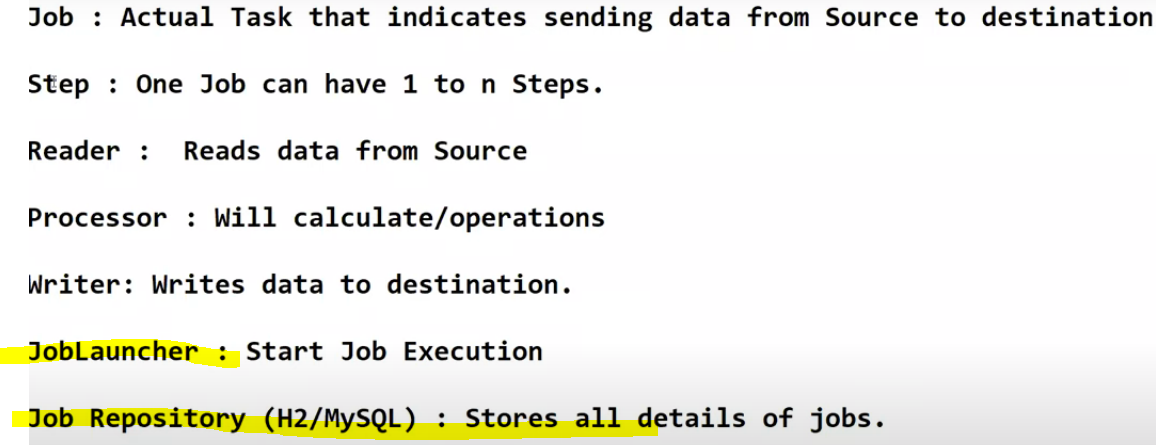


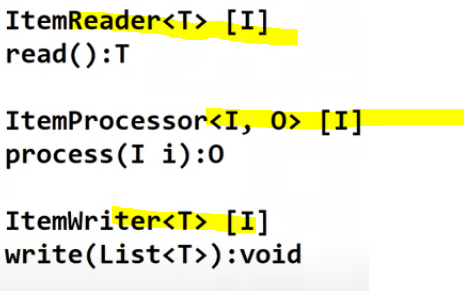
All the job status will be stored in job repository.- jan ran or failed or the status ..



## Job launcher

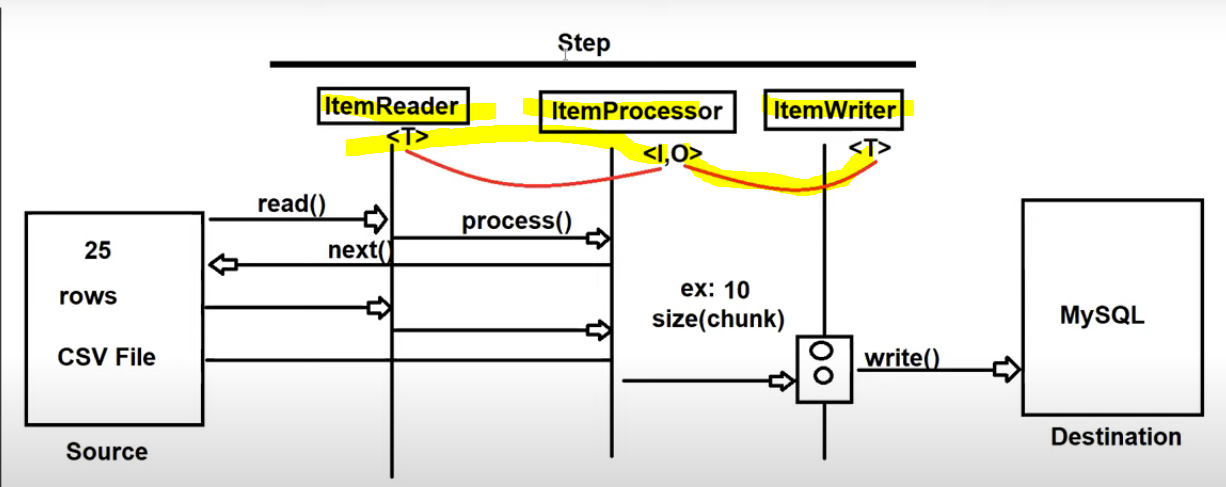
Job launcher will take inputs from job parameters



all are interfaces

## Item Reader or Item Processor vs Item Writer

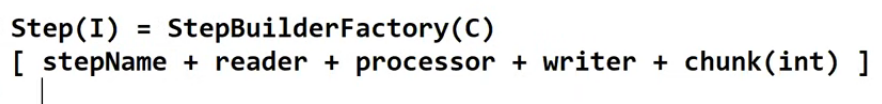
Item is nothing but an individual record.

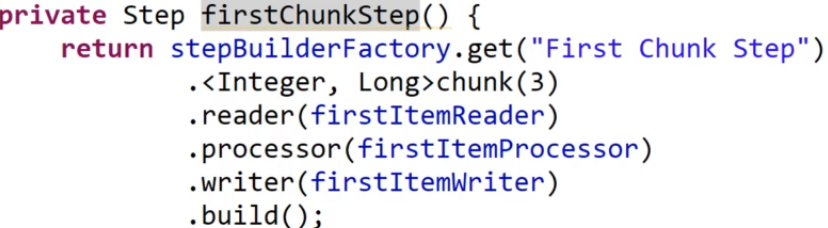


If 25 times reader executed , then 25 times Processor will be executed ,Whereas only 3 times the writer will be executed. Because 25 times the writer should not hit the database, we will mention the chunk size as 10 , only when it has 10 records then it will hit the ItemWriter .

## Step

Step is an interface created using stepBuilderFactory

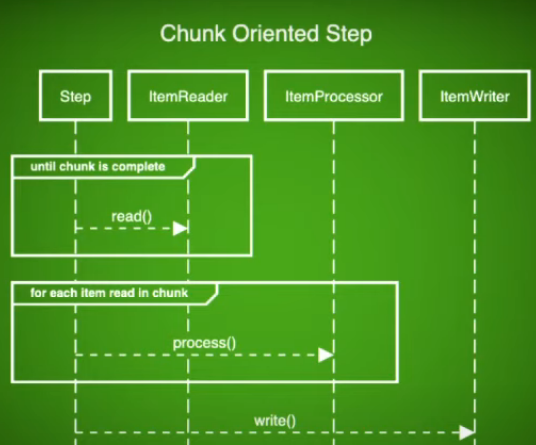




### Chunk oriented step

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

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



### Job

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

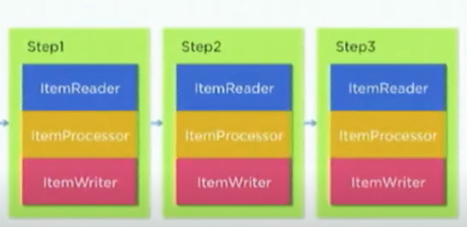
Job is nothing but a process which run from start to finishes without any interuption

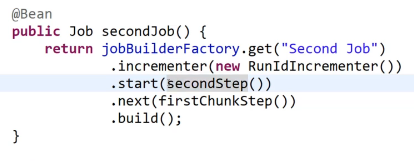
Job is also created by job BuilderFactory.

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

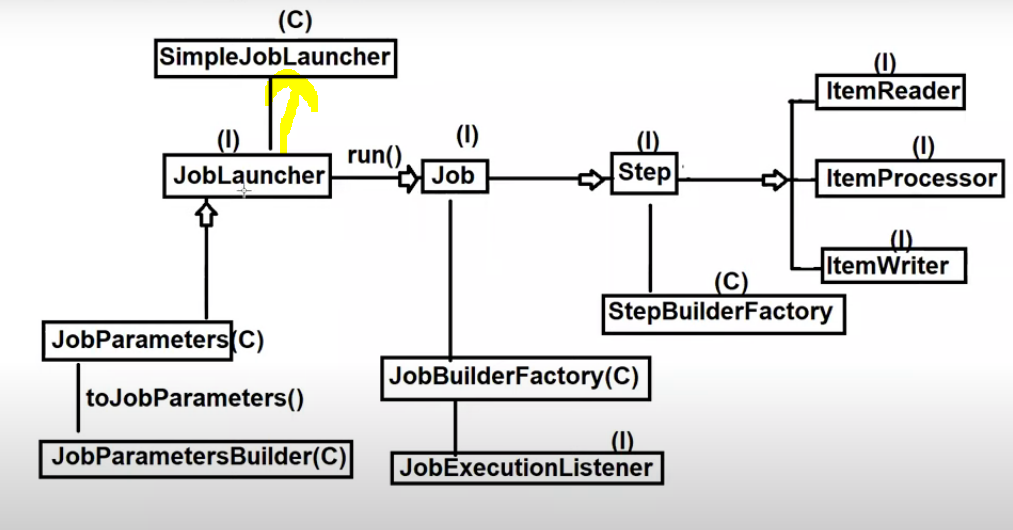
Job means lot of steps – like if u want to become IAS officer u have lot of steps

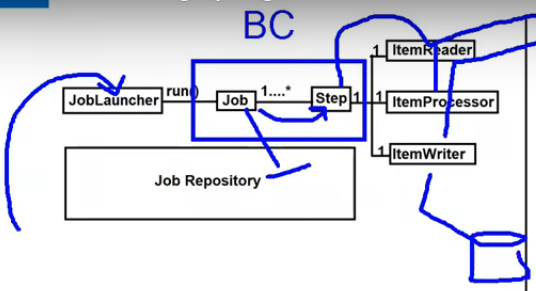
* We can have a job with 🡪 Tasklet and chunk oriented step combo

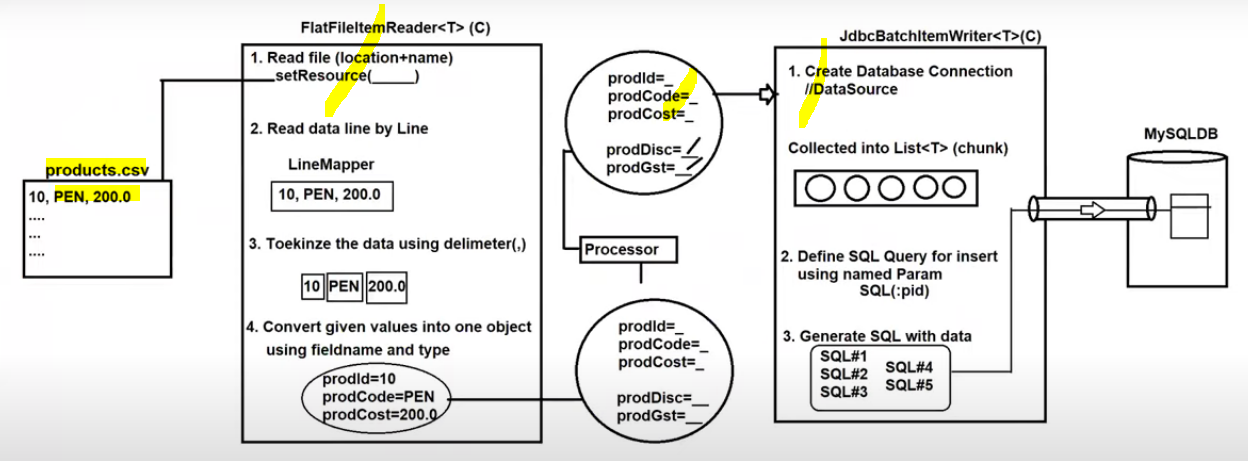


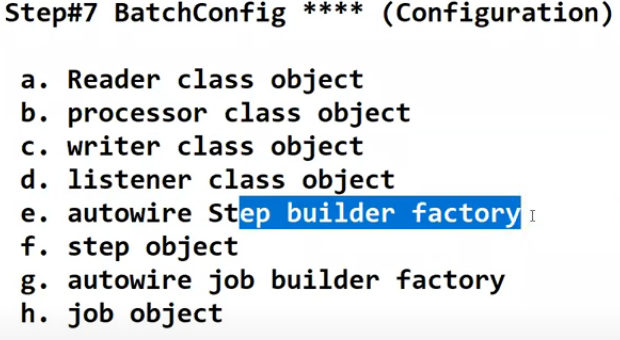
 

## Full diagram









### ItemReader

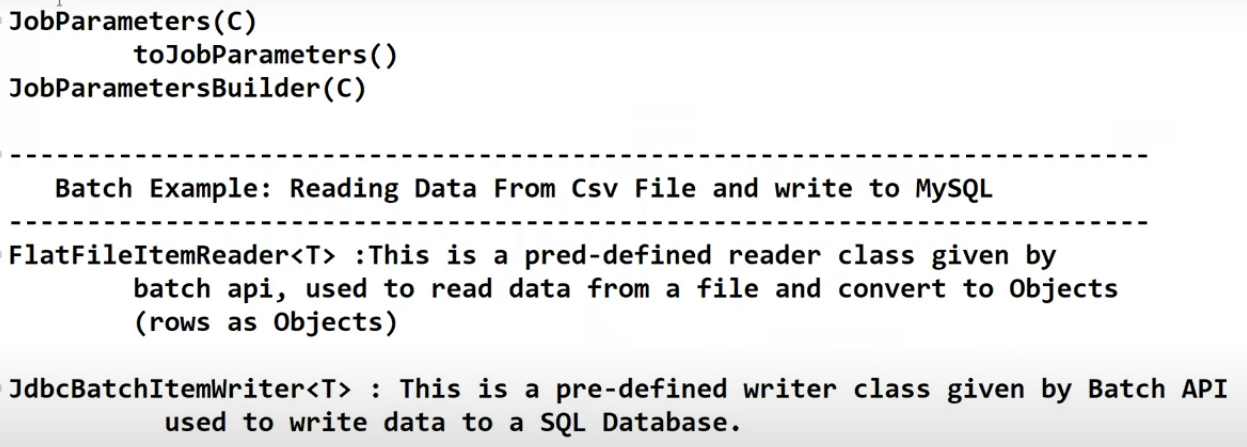
In most of the scenarios we will use predefined readers only

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

If chunk size is 20 , that item reader read() method will be executed 20 times continously,

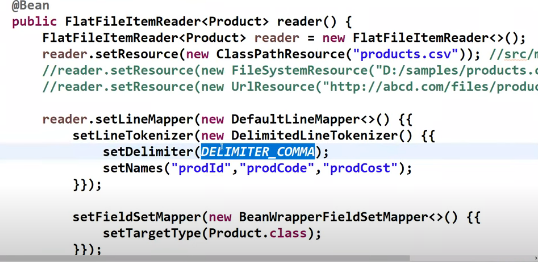
And it will keep on running/executing the reader until last element is null.

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



Classpath means src/main/resources

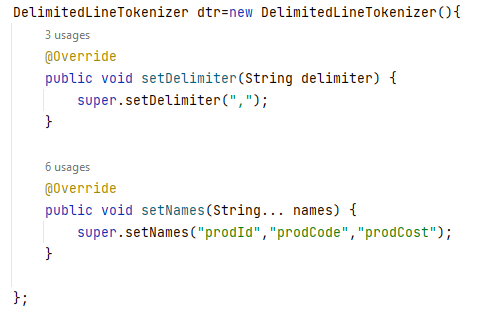
Full diagram



### Detailed ItemReader

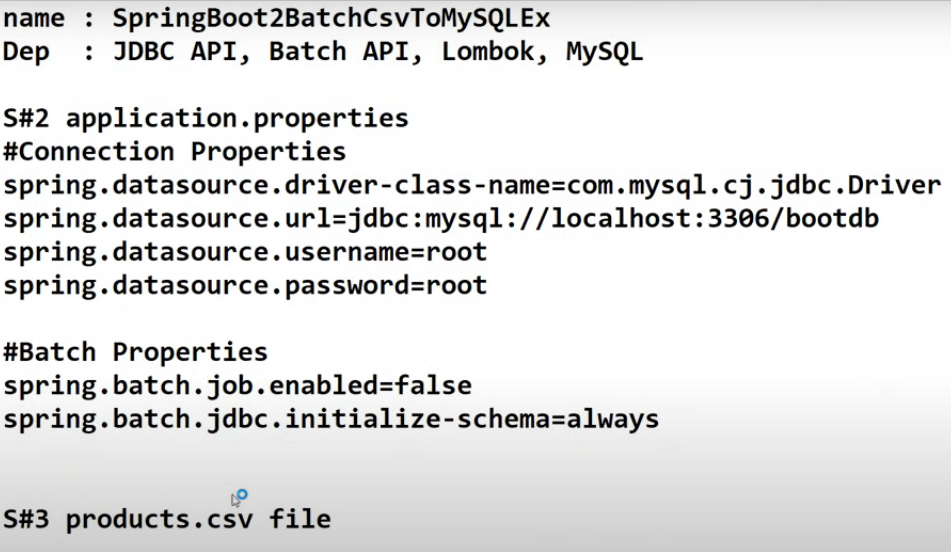
Create a line tokenizer object – Break the line and form the fields later map those to a bean.

it says, first split with comma, and consider those fields with these given field names and later map these fields to a bean





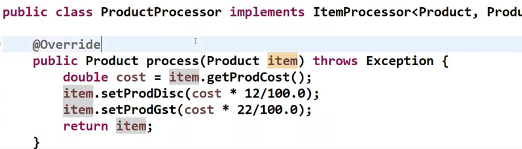
### Program



### 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.

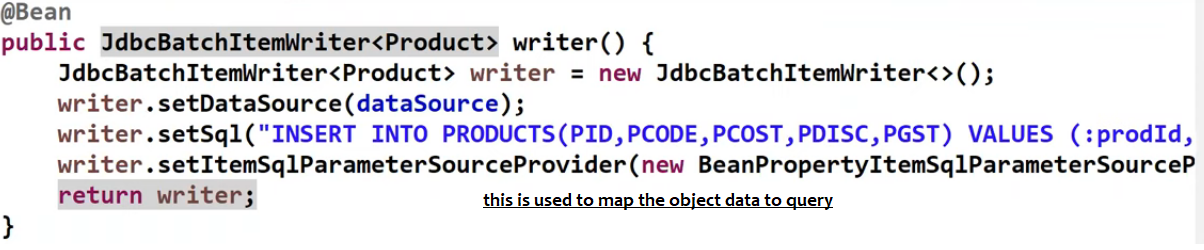


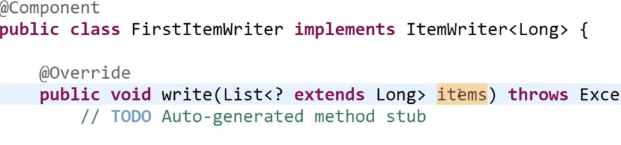
### ItemWriter

Here use predefined writer, it will receive the chunk of data at once, until item processor ,

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

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





### Step object

#### StepExecutionContext

For each step there will be separate step execution context. And each step execution context data stored in table called dtepeexecution\_context

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

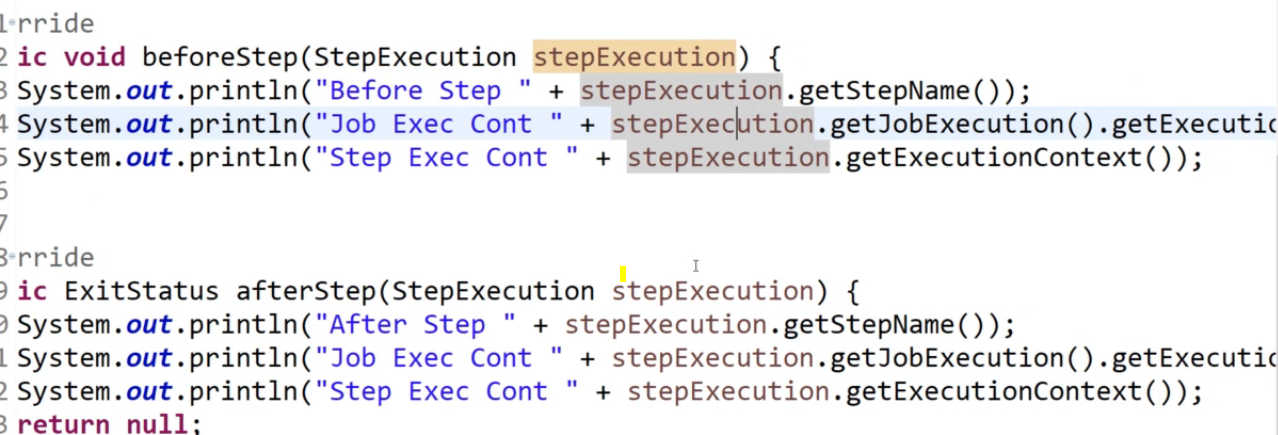
This is the memory available for that entire step.

This is also a map, which will be stored and is 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

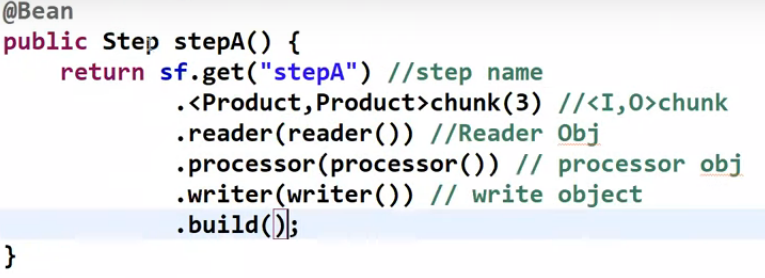
StepExecutionListener

This will be executed before and after executing the step



Create step using stepBuilder Factory

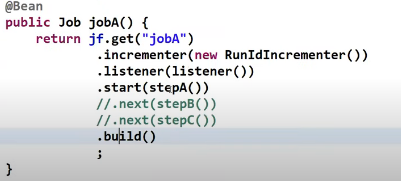
Dividing into chunks is mandatory, else memory out of bounds exceptions only, it says how many records u want to process at a time





### Job

Here we will configure all the steps



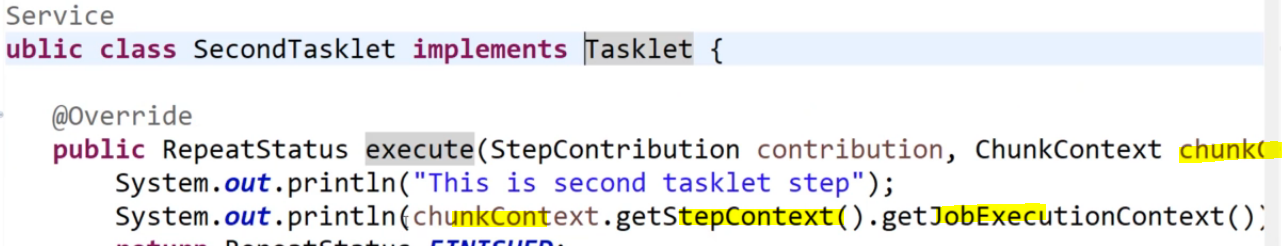
#### JobExecutionContext

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

Using below, if u set something to jobContext



Then this is available as per above

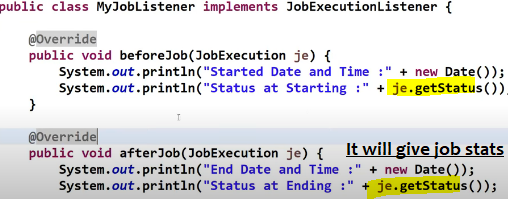


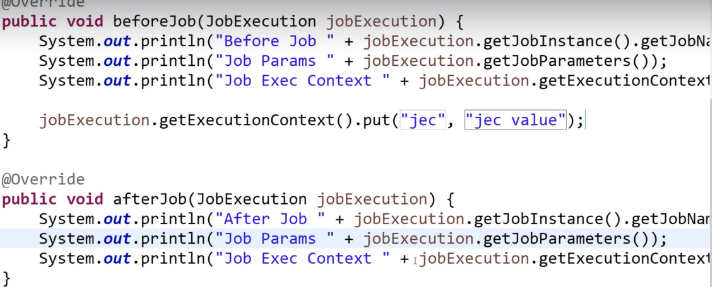
#### JobExecutionListener

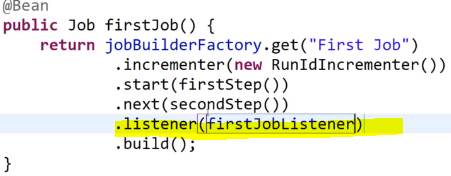


This must be a spring bean , so better mark it as a spring bean, marking is not enough

We need to attach to the job

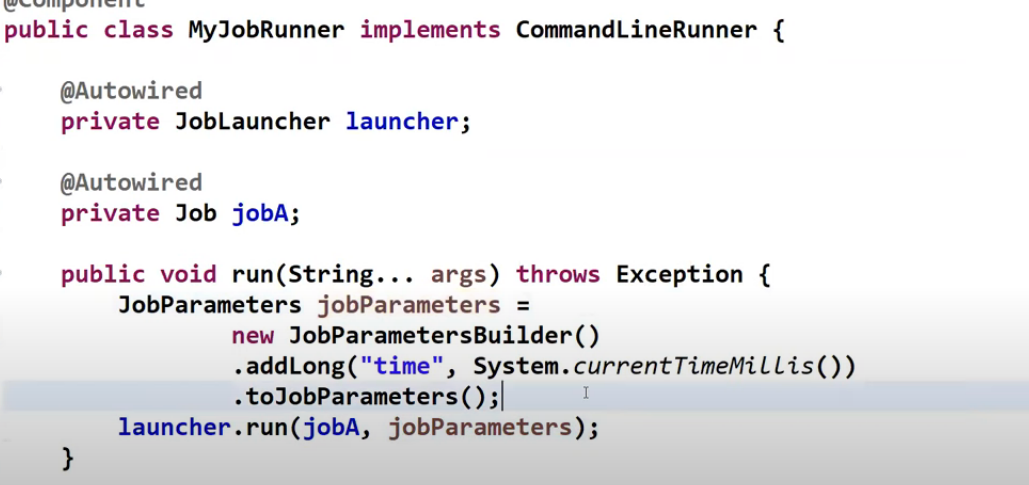






### Run job using jobParametersBuilder

And job parameters is created by job parameterBuilder



### 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

## jobRepository

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