```
                     Date : 17/04/2021
                     Spring Boot 9AM
                       Mr. RAGHU
         ----------------------------------------
            Spring Boot Batch API:CSV File TO MySQL

--Core Java -------------
class A {

}

1. A a = new A();
//creating object

2. A a = new A() { }
//creating anynymous inner class + object

3. A a = new A() {{  }}
// creating anynymous inner class + instance block + object
new A(){

  {
    //instance block

  }

}
***) When we are writing anonymouse class then we can
   not define constrcutor. So, use instance block.

4. A a = new A() {{{ }}}
//creating anynymous inner class + instance block
    + local scope block + object

-----------------------------------------------
new A() {
  {

  }
};
For this code internally one sub class is created
without any name(name less/anonymous). But Java gives
numbers while accessing only. ($1, $1212..etc)

Internally code looks like:

class $1 extends A {
 ...
}
and object created at same time
new $1();

---Test.java------------------
package in.nareshit.raghu;

class A{
  A(){
          System.out.println("FROM A");
  }
}

public class Test {

        public static void main(String[] args) {
            A  a = new A();
            System.out.println(a.getClass().getName());
            A  a1 = new A() {
                 {
                         System.out.println("FROM SUB TYPE");
                 }
            };
            System.out.println(a1.getClass().getName());

        }
}
```

```
*) Anynomyous code executes faster compared with normal
   sub class, but coding is bit complex compared to normal
   one.

CSV- Comma Separated Values (Excel File)
Tokenize: Convert one String into multiple Strings using
         one separator symbol (delimeter , . - / +)
Ex: "Hello-World-Welcome-To-Nit"
    "Hello","World","Welcome","To","Nit".

FlatFileItemReader(C):-
  This file will convert given data (Text file data)
  into Required Object Format. It Reads data line by line

a. Load File with name and location
    use method setResource(..)

Resource(I) [org.springframework.core.io]

=> /src/main/resources folder
Resource r1 = new ClassPathResource("abcd.csv");
=> In Computer Drives
Resource r2 = new FileSystemResource("d:/mydata/abcd.csv");
=> in internet location
Resource r3 = new
UrlResource("http://s3.amazon.bucket/sourcenit/abcd.csv");

b. Read Data Line by Line From File
    setLineMapper(..) Here LineMapper(I) So, we use
    Impl class: DefaultLineMapper.

*) one Line in File is one String object internally
      String s="10,PEN,300.0";

c. One Line Data should be converted into multiple values
   which can be done using 'LineTokenizer(I)'
   (DelimitedLineTokenizer(C)). Default Delimeter is
   COMMA (,). We can even use any other char like - . /
   ..etc

d. Provide Names to values (like variable names)
    ex:
       pid   = 10;
       pcode = PEN
       pcost = 300.0

e. Convert above variables data into one object
    using 'FieldSetMapper(I)' Impl class
        BeanWrapperFieldSetMapper(C)

 ie create object and set data based on variable names
    Ex:
        Product p = new Product();
        p.setPid(pid);
        p.setPcode(pcode);
        p.setPcost(pcost);

-----Sample code-------------------------
@Bean
public ItemReader<String> reader() {
 FlatFileItemReader<String> reader = new FlatFileItemReader<>();
 reader.setResource(new ClassPathResource("abcd.csv"));
 reader.setLineMapper(new DefaultLineMapper<>() {{
        setLineTokenizer(new DelimitedLineTokenizer() {{
                setDelimiter(DELIMITER_COMMA);
                setNames("pid","pcode","pcost");
        }});
        setFieldSetMapper(new BeanWrapperFieldSetMapper<>() {{
                setTargetType(Product.class);
        }});
  }});
  return reader;
}
--------------------------------------------------
JdbcBatchItemWriter(C):-
 This is used to execute multiple INSERT/UPDATE
 SQLs to Database at a time using single network call.

a. Create Database Connection (as we are using any
   JDBC/JPA AutoConfiguration)

 @Bean
 public DataSource ds() {
   setDriver, setUrl...
 }

b. Create one INSERT SQL query using named Parameters

SQL="INSERT INTO PRODUCT(PID,PCODE,PCOST,GST,DISCOUNT)
      VALUES(:prodId,:prodCode,:prodCost,:prodGst,:prodDisc)";

c. Read data from Object and place values inside named
   parameter using variable names in given object
```

```
--sample code---
@Bean
public ItemWriter<String> writer() {
  JdbcBatchItemWriter<String> writer = new JdbcBatchItemWriter<>();
   writer.setDataSource(dataSource());
   writer.setSql("INSERT INTO PRODUCT ....");
   writer.setItemSqlParameterSourceProvider(new
BeanPropertyItemSqlParameterSourceProvider<>());
   return writer;
}
BeanProperty     = object variable
ItemSqlParameter = input to SQL
SourceProvider   = Data location

ie Read data to SQL Parameter from variables getMethod
   using Given Object
```