APLICACIÓN JAVA CON SPRING FRAMEWORK, SPRING DATA SPRING BATCH PARA EL DISEÑO Y DESARROLLO DE METODO PARA LEER UN CODIGO POSTAL EN ESPECIFICO

Para realizar el programa primero se debe crear una aplicación, aunque se puede utilizar diferentes ide de Java como Eclipse, INTELLIJ Idea, en este caso yo use spring tools suite 4 y se crea un proyecto de spring boot, le ponemos un nombre a la aplicación y elegimos las dependencias que usaremos en este caso spring web, batch, data JPA, dev tools y h2 database.

Primero debemos saber de dónde obtendremos los datos en mi caso es un archivo .csv que tiene los siguientes campos

d_codigo	d_asenta	d_tipo_asen	D_mnpio	d_estado	d_ciudad	d_CP	c_estado	c_oficina	c_CP	c_tipo_asent	c_mnpio	id_asenta_cp	d_zona	c_cve_ciudad
1000	San Ángel	Colonia	Álvaro Obre	Ciudad de M	Ciudad de M	1001	9	1001		9	10	1	Urbano	1
1010	Los Alpes	Colonia	Álvaro Obre	Ciudad de M	Ciudad de M	1001	9	1001		9	10	5	Urbano	1
1020	Guadalupe I	Colonia	Álvaro Obre	Ciudad de M	Ciudad de M	1001	9	1001		9	10	6	Urbano	1
1030	Axotla	Pueblo	Álvaro Obre	Ciudad de M	Ciudad de M	1001	9	1001		28	10	9	Urbano	1
1030	Florida	Colonia	Álvaro Obre	Ciudad de M	Ciudad de M	1001	9	1001		9	10	10	Urbano	1
1040	Campestre	Colonia	Álvaro Obre	Ciudad de M	Ciudad de M	1001	9	1001		9	10	12	Urbano	1
1049	Tlacopac	Pueblo	Álvaro Obre	Ciudad de M	Ciudad de M	1001	9	1001		28	10	14	Urbano	1
1050	Ex-Hacienda	Colonia	Álvaro Obre	Ciudad de M	Ciudad de M	1001	9	1001		9	10	16	Urbano	1
1060	Altavista	Colonia	Álvaro Obre	Ciudad de M	Ciudad de M	1001	9	1001		9	10	17	Urbano	1
1060	San Ángel In	Colonia	Álvaro Obre	Ciudad de M	Ciudad de M	1001	9	1001		9	10	18	Urbano	1

Pero para leer nuestro código postal no se leerá cada campo uno a uno ya que seria demasiado tardado y además una perdida de tiempo, suponiendo por ejemplo que el que buscamos este en la última posición.

Así que optaremos otra estrategia que se especificara a continuación

CREACION DEL MODELO

Crearemos una clase nueva que se encargara de obtener los campos en mi caso la clase se llama ZipCodes y tiene el siguiente diseño

```
@Entity
@Table(name = "Zip_Codes")
public class ZipCodes implements Serializable{

    private static final long serialVersionUID = 7291345612921641755L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer d_codigo;
    private String d_asenta;
    private String d_tipo_asenta;
    private String D_mnpio;
    private String d_estado;
    private String d_ciudad;
    private Integer d_CP;
    private Integer c_estado;
    private Integer c_oficina;
    private Integer c_CP;
    private Integer c_tipo_asenta;
    private Integer c_mnpio;
    private Integer id_asenta_cpcons;
    private String d_zona;
    private Integer c_cve_ciudad;
```

Como se menciono no leeremos los datos uno a uno si que lo incrustaremos esos datos en una Base de datos que aparte de simplificar el trabajo lo volverá mucho más rápido y seguro.

Agregamos Entity para decirle al programa que necesitamos una tabla que siga el diseño de esta clase, le ponemos un nombre y especificamos un id, el valor generado para ese id será identity por lo que no tendremos que preocuparnos por asignar un id en cada insert a la base de datos si no que se hará solo.

PASAR DATOS DE CSV A SQL

Pero de que nos sirve tener el modelo si este no contiene datos es la pregunta, bueno se necesita crear un método que tome los datos y los pase a la tabla que nosotros especificamos.

Para ello haremos uso de spring batch

Spring Batch es un framework ligero enfocado específicamente en la creación de procesos batch. Además de marcar unas directrices para el diseño de procesos, Spring Batch proporciona con una gran cantidad de componentes que intentan dar soporte a las diferentes necesidades que suelen surgir a la hora de crear estos programas: trazas, transaccionalidad, contingencia, estadísticas, paralelismo, particionamiento, lectura y escritura de datos, etc…

Para lograr esto crearemos una clase llamada BatchConfiguration y debemos definir lo siguiente:

ItemReader: Elemento responsable de leer datos de una fuente de datos (BBDD, fichero, cola de mensajes, etc…)

ItemWriter: Elemento responsable guardar la información leída por el reader. Si hay un reader debe haber un writer.

Job: El Job es la representación del proceso. Un proceso, a su vez, es un contenedor de pasos (steps).

Step: Un step (paso) es un elemento independiente dentro de un Job (un proceso) que representa una de las fases de las que está compuesto dicho proceso. Un proceso (Job) debe tener, al menos, un step.

Primero empecemos con la creación de nuestro ItemReader:

```
@Bean
public FlatFileItemReader<ZipCodes> reader() {
    return new FlatFileItemReaderBuilder<ZipCodes>()
      .name("ZipCodesItemReader")
      .resource(new ClassPathResource("CDMX.csv"))
      .delimited()
      .names(new String[]{ "d_codigo", "d_asenta", "d_tipo_asenta", "D_mnpio", "d_estado", "d_ciudad", "d_CP",
              "c_estado", "c_oficina", "c_CP", "c_tipo_asenta", "c_mnpio", "id_asenta_cpcons", "d_zona",
              "c_cve_ciudad"})
      .fieldSetMapper(new BeanWrapperFieldSetMapper<ZipCodes>() {{
          setTargetType(ZipCodes.class);
      }})
      .linesToSkip(1)
      .build();
}
```

Así que creamos un componente que contenga un método que nos regresara un FlatFileItemReader con el podremos tratar nuestro archivo de texto que es el .csv, le ponemos un nombre y especificamos la ruta donde se encuentra el archivo, ahora hay que delimitar los campos es decir su forma de separación en caso obviamente especificamos que se separen por el nombre de la columna y especificamos un mapper que se encargara de procesar las líneas para tratarlas como objetos.

Ya con nuestro reader debemos especificar el writer que es a donde queremos que se encuentren los datos

```
@Bean
public JdbcBatchItemWriter<ZipCodes> writer() {
    return new JdbcBatchItemWriterBuilder<ZipCodes>()
            .itemSqlParameterSourceProvider(new BeanPropertyItemSqlParameterSourceProvider<ZipCodes>())
            .sql("INSERT INTO Zip_Codes(d_asenta, d_tipo_asenta, D_mnpio,"
                    + "d_estado, d_ciudad, d_CP, c_estado, c_oficina, c_CP, c_tipo_asenta,"
                    + "c_mnpio, id_asenta_cpcons, d_zona, c_cve_ciudad) VALUES(:d_asenta, :d_tipo_asenta, "
                    + ":D_mnpio, :d_estado, :d_ciudad, :d_CP, :c_estado, :c_oficina, :c_CP, :c_tipo_asenta, "
                    + ":c_mnpio, :id_asenta_cpcons, :d_zona, :c_cve_ciudad)")
            .dataSource(dataSource)
            .build();
}
```

Aquí solo debemos decir que acción debe implementar la base de datos, como queremos insertar los datos que leímos del csv hacemos un insert into con el nombre de la tabla que creamos anteriormente y mandamos nuestro datasource que es la base de datos que contendrá estos datos.

Para ello hay especificar que tipo de BD usaremos

```
spring.application.name=zip-codes
spring.datasource.url=jdbc:h2:tcp://localhost/~/test
spring.datasource.driver-class-name=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.jpa.hibernate.ddl-auto=create
```

En este caso estoy usando una base de datos de tipo h2 que se encuentra localizada en memoria, hay que especificar su url, el driver, nombre, password y la plataforma, en este caso le digo que crea la tabla, pero elimina los datos cada vez que se vuelve a iniciar el programa. La estoy usando porque es bastante intuitiva y no requiere instalar nada, aunque lo ideal seria tener una base de datos montada en un servidor web para el acceso de todos los usuarios.

```
@Bean
public Job createEmployeeJob(Step step1) {
     return jobBuilderFactory
       .get("createZipCodesJob")
       .incrementer(new RunIdIncrementer())
       .flow(step1)
       .end()
       .build();
}

@Bean
public Step step1(ItemReader<ZipCodes> reader, ItemWriter<ZipCodes> writer) {
      return stepBuilderFactory
        .get("step1")
        .<ZipCodes, ZipCodes>chunk(10)
        .reader(reader)
        .writer(writer)
        .build();
}
```

Como ya se mencionó estos métodos será leer y traspasar los datos especificados, pero ¿qué es un chunk?

Supongamos que tenemos un fichero en texto plano, donde debemos tratar cada línea para luego persistir cierta información en una base de datos. Si configuramos nuestro step con un intervalo de commit fuese igual a 10, lo que haría sería leer una línea del fichero, luego tratarla, leer otra línea del fichero, volver a tratarla, así hasta 10 veces. Una vez que ya hemos leído y tratado 10 líneas, el writer recibe esa información (los 10 chunks) y los persiste en base de datos. Este proceso se repetiría hasta terminar con todas las líneas el fichero.

LECTURA DE DATOS CON SPRING DATA

Para leer nuestros datos usaremos spring data y JPA, que nos ayudara a obtener los valores cargadas en la base datos, primero creamos una interfaz que contiene los métodos que implementaremos uno obtiene todos los correos postales y el siguiente obtiene un correo en base al id especificado que es nuestro objetivo final

```
public interface ZipCodesRepo extends CrudRepository<ZipCodes, Integer>
```

Creamos una interfaz que extiende de CrudRepository que pertenece a spring data que nos permitirá usar distintos métodos que nos serán de gran utilidad

Ahora en una clase nueva llamaremos a los métodos que nos otorga CrudRepository

```
@Override
@Transactional(readOnly = true)
public List<ZipCodes> findAll() {
    return (List<ZipCodes>) zipCodesRepo.findAll();
}

@Override
@Transactional(readOnly = true)
public ZipCodes findById(Integer zip_code) {
    // TODO Auto-generated method stub
    return zipCodesRepo.findById(zip_code).orElse(null);
}
```

El primer método que usaremos es findAll que como se puede deducir obtiene básicamente todos los elementos guardados en la table, mientras que el siguiente método será findById que realiza un select a la base de datos en base al parámetro establecido en nuestro caso será el id del código postal en caso de encontrarlo lo desplegara en caso contrario desplegara un null.

Finalmente creamos una clase que será nuestro Controller donde implementaremos un API Rest EndPoint

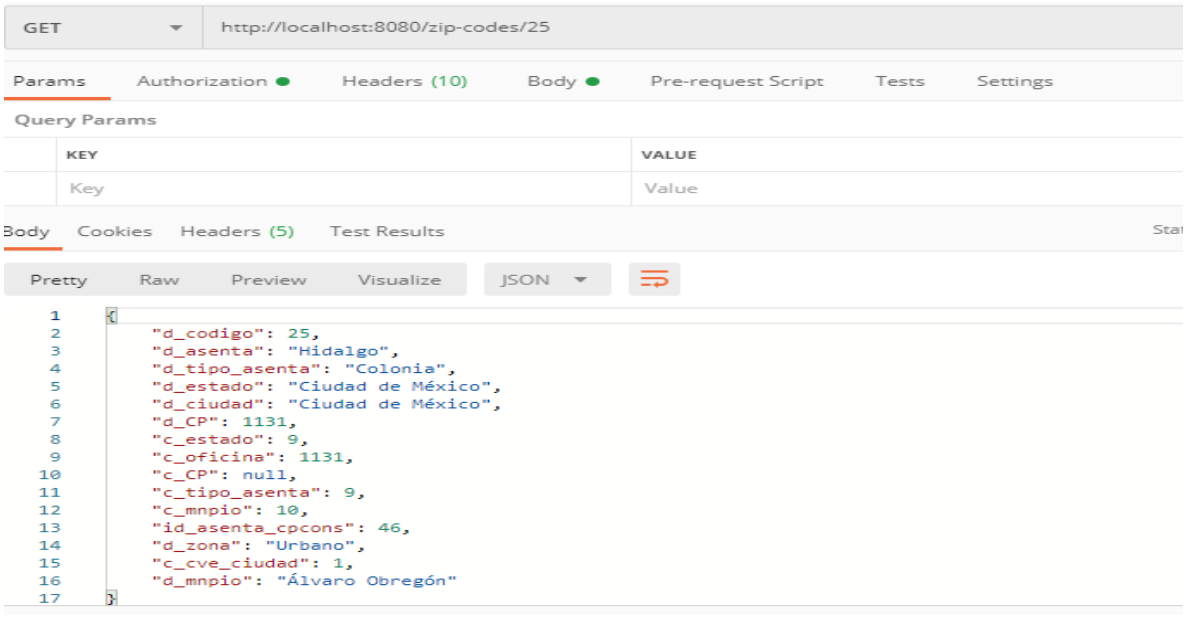
REST es cualquier interfaz entre sistemas que use HTTP para obtener datos o generar operaciones sobre esos datos en todos los formatos posibles, como XML y JSON, en nuestro caso es JSON.

Asi que básicamente es usar los métodos que creamos para mapearlos en un servicio web especificado por una ruta de acceso es decir nuestro endpoint

```java
@GetMapping("/zip-codes/{zip_code}")
public ResponseEntity<ZipCodes> getZipCode(@PathVariable Integer zip_code) {
    ZipCodes zipCode = zipCodeService.findById(zip_code);
    if (zipCode == null) {
        return new ResponseEntity<ZipCodes>(zipCode, HttpStatus.NOT_FOUND);
    }
    return new ResponseEntity<ZipCodes>(zipCode, HttpStatus.OK);
}
```

Básicamente buscamos si encontramos desplegamos en caso de que no se encuentre se informara.

Solo falta ver su funcionamiento así que levantamos el servicio, entonces clic derecho al proyecto, run as y spring boot app por defecto el servicio se levanta en el puerto 8080 así que a menos que lo cambiemos ahí estará, para ver los resultados podríamos abrir una pestaña en nuestro navegador web y escribir la ruta y veríamos el funcionamiento en mi caso usare postman



Listo podemos ver que efectivamente nos regresó nuestro código con el id 25.