

## Contents

- Changing path of local repository (.m2)
- CSV Parsing with Spring Boot
- What is Unit Testing ( Junit , Mockito )

## Changing path of local repository (.m2)

We can change the path of the .m2 folder by giving the new desired path inside the settings.xml file inside the .m2 folder . cd ~/.m2 . Here we create a new settings.xml .

By default the path of the .m2 folder is your root . Inside settings.xml , let's write :

```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1.00 xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0 https://maven.apache.org/xsd/settings-1.0.0.xsd">
  <localRepository>Users/pa/repo_23</localRepository>
</settings>
```

### Note :

We can get the parent element settings tag from <https://maven.apache.org/settings.html> and in localRepository tag we can give the absolute new path where we want the local repository to be created .

Finally , do mvn clean package in any maven project to start downloading dependencies from the central repository to your new local repository . Its name has changed from .m2 to repo\_23 in the above case .

**Note :** Path of local repository will change for all the projects and not for single project .

```
+ repo_23 ls
antlr          ch              commons-io      javax           org
aopalliance    classworlds     commons-logging junit           mysql
asm            com             io             jakarta         net
backport-util-concurrent commons-codec
+ repo_23
```

## CSV Parsing with Spring Boot

TASK : You will be receiving a csv/xls file which contains some employee info, you need to parse that file and save the employee info in the db . Clients will come to your app frontend and upload a csv file . This will create a POST request to your server with the file attachment .

**Note :** You will need Apache Commons IO and Apache Commons CSV dependencies.

The screenshot shows a Postman interface for a POST request to `localhost:8080/parse_csv`. The request is configured with the following details:

- Method:** POST
- URL:** localhost:8080/parse\_csv
- Body Type:** form-data (selected)
- Body Data:**

KEY	VALUE
<input checked="" type="checkbox"/> enterprise_name	geeksforgeeks
<input checked="" type="checkbox"/> gfg_csv	employeeCSV.csv
Key	Value

The interface also shows tabs for Params, Authorization, Headers (9), Body (selected), Pre-request Script, Tests, and Settings. Below the body data, there are tabs for Body (selected), Cookies (1), Headers (4), and Test Results. The Body tab is further divided into Pretty (selected), Raw, Preview, and Visualize. A 'Text' dropdown and a red icon are also visible.

Suppose we are sending two keys as form data from the postman . One is text and other is csv file .

We can easily retrieve the name of the fileKey , name of textkey and name of the real file attached (value) as below :

```
@PostMapping("/parse_csv")

public ResponseEntity<String> parseCSV(HttpServletRequest httpServletRequest) {

    Part textPart = httpServletRequest.getPart("enterprise_name");

    Part filePart = httpServletRequest.getPart("gfg_csv");

    System.out.println(filePart.getName()); //fileKey

    System.out.println(filePart.getSubmittedFileName()); //fileValue name

    System.out.println(textPart.getName()); // textKey name

    return new ResponseEntity<>(HttpStatus.OK);

}
```

But Inorder to retrieve the value of text we are sending from Postman , we need to first convert the textPart to InputStream and then convert the inputStream to String as below :

```
InputStream inputStream = textPart.getInputStream();

String result = IOUtils.toString(inputStream, StandardCharsets.UTF_8);
```

Now lets focus on parsing the file and onboarding the employees from csv to database.

We will convert the filePart to InputStream and then create a new CSVParser object and then get all the rows from our csv file in an a list of CSVRecords /

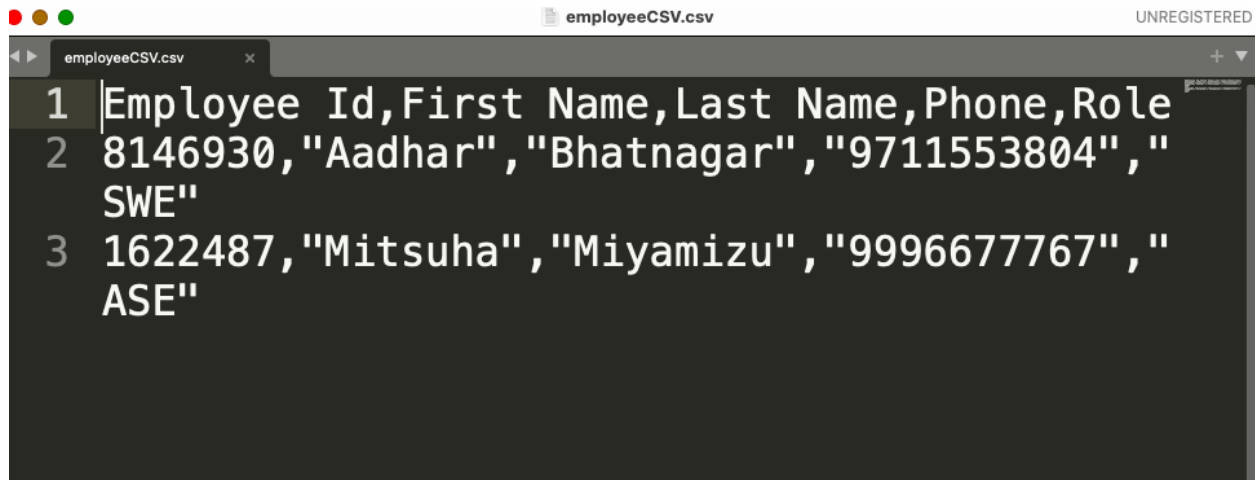
```
Part filePart = httpRequest.getPart("gfg_csv");  
  
InputStream fileInputStream = filePart.getInputStream();  
  
CSVFormat csvFormat = CSVFormat.DEFAULT;  
  
CSVParser csvParser = new CSVParser(new  
    InputStreamReader(fileInputStream),csvFormat);  
  
List<CSVRecord> rows = csvParser.getRecords();
```

Now it's simple . Create an Entity class , Service class and Repository class . In the controller , we will get each row and simply need to extract the columns from row one by one and build an object . We will save this object to our Employee table . Same will be done for all rows we get from the list of CSVRecord .

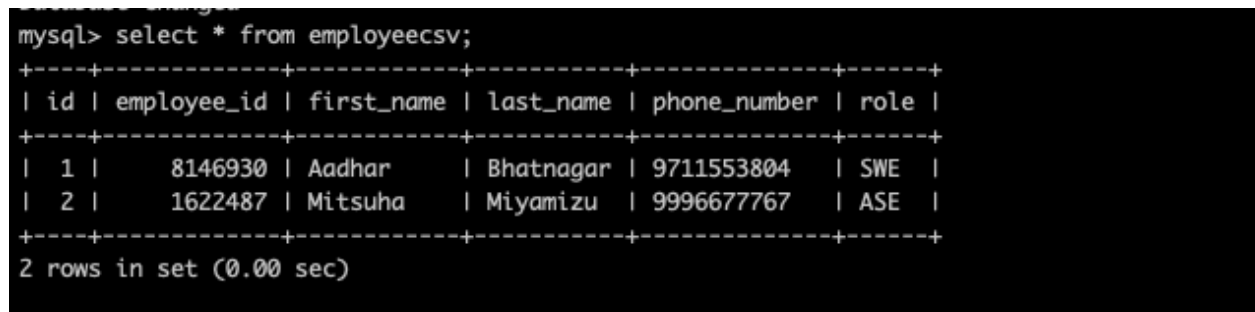
```
for(int i=1;i< rows.size();i++){  
  
    CSVRecord row = rows.get(i);  
  
    EmployeeCSV employeeCSV = EmployeeCSV.builder()  
  
        .employeeId(Integer.parseInt(row.get(0)))  
  
        .firstName(row.get(1))  
  
        .lastName(row.get(2))  
  
        .phoneNumber(row.get(3))  
  
        .role(row.get(4))  
  
        .build();  
  
    employeeCSVService.createEmployeeFromCSV(employeeCSV);
```

```
}

```



```
1 Employee Id,First Name,Last Name,Phone,Role
2 8146930,"Aadhar","Bhatnagar","9711553804","SWE"
3 1622487,"Mitsuha","Miyamizu","9996677767","ASE"
```



```
mysql> select * from employeecsv;
+----+-----+-----+-----+-----+-----+
| id | employee_id | first_name | last_name | phone_number | role |
+----+-----+-----+-----+-----+-----+
| 1 | 8146930 | Aadhar | Bhatnagar | 9711553804 | SWE |
| 2 | 1622487 | Mitsuha | Miyamizu | 9996677767 | ASE |
+----+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

**UNIT TESTING :** Unit Testing is a software testing technique by means of which individual units of software i.e. group of computer program modules, usage procedures and operating procedures are tested to determine whether they are suitable for use or not. It is a testing method using which every independent module is tested to determine if there are any issues by the developer himself. It is correlated with functional correctness of the independent modules. Unit tests are supposed to be written by developers .

### TYPES OF UNIT TESTING :

There are two ways to perform unit testing:

#### 1) Manual Testing :

If you execute the test cases manually without any tool support, it is known as manual testing. It is time consuming and less reliable.

#### 2) Automated Testing :

If you execute the test cases by tool support, it is known as automated testing. It is fast and more reliable.

References :

<https://www.geeksforgeeks.org/unit-testing-software-testing/>

<https://www.geeksforgeeks.org/types-software-testing/>

<https://www.geeksforgeeks.org/what-is-unit-testing-and-why-developer-should-learn-it/>

<https://www.geeksforgeeks.org/difference-between-unit-testing-and-system-testing/>

**JUNIT** : JUnit is an open source testing framework which is used to write and run repeatable automated tests, so that we can be ensured that our code works as expected. JUnit is widely used in industry and can be used as stand alone Java program (from the command line) or within an IDE such as Eclipse.

References :

<https://www.geeksforgeeks.org/difference-between-junit-and-testng/>

**MOCKITO** : Mockito is used along with Junit for writing unit tests . It is responsible for mocking .Mocking is a testing technique where real components are replaced with objects that have a predefined behavior (mock objects) only for the test/tests that have been created for. In other words, a mock object is an object that is configured to return a specific output for a specific input, without performing any real action. Used to mock your component calls .

References :

<https://www.geeksforgeeks.org/software-engineering-mock-introduction/>

### Challenge 1 : What is Mocking ?

HINT : Mocking is a process of developing the objects that act as the mock or clone of the real objects. In other words, mocking is a testing technique where mock objects are used instead of real objects for testing purposes. Mock objects provide a specific (dummy) output for a particular (dummy) input passed to it.

The mocking technique is not only used in Java but also used in any object-oriented programming language. There are many frameworks available in Java for mocking, but Mockito is the most popular framework among them.

To mock objects, you need to understand the three key concepts of mocking, i.e., stub, fake, and mock. Some of the unit tests involve only stubs, whereas some involve fake and mocks.

The brief description of the mocking concepts is given below:

**Stub**: Stub objects hold predefined data and provide it to answer the calls during testing. They are referred to as a dummy object with a minimum number of methods required for a test. It also provides methods to verify other methods used to access the internal state of a stub, when necessary. Stub object is generally used for state verification.

**Fake**: Fake are the objects that contain working implementations but are different from the production one. Mostly it takes shortcuts and also contains the simplified version of the production code.



Mock: Mock objects act as a dummy or clone of the real object in testing. They are generally created by an open-source library or a mocking framework like Mockito, EasyMock, etc. Mock objects are typically used for behavior verification.

### **HOMEWORK ( BONUS )**

Challenge 1 : Lets add the dependencies for Junit and Mockito in our pom.xml of the Minor Project

Challenge 2 : What is the significance of <scope>test</scope> under <dependency> in pom.xml ?

**HINT** : <https://stackoverflow.com/questions/26975818/what-is-scope-under-dependency-in-pom-xml-for>

Challenge 3 : Let us try to write the unit tests for TransactionService from our Minor Project . Example : let's write a unit test for the below method from Transaction Service .

*public String createTransaction(Request request)*

References :

<https://www.geeksforgeeks.org/how-to-write-test-cases-in-java-application-using-mockito-and-junit/>

<https://www.geeksforgeeks.org/unit-testing-in-spring-boot-project-using-mockito-and-junit/>