

JAVA DOCUMENTAION

NAME: P. SANJANA REDDY

EMP ID: MT4020

Table of Content

Topic	User Story	Page NO
1.	Convert string date to timestamp data type columns based on pattern	2-11
2.	Create an application that connects to database and performs CRUD operations using JDBC Querying	12-26
3.	Implement all types of Inheritance with real time examples	27-39
4.	Fetch data from table using JDBC, and convert data to specified file formats.	40-56
5.	Create a custom Array List Collection	57-61
6.	Insert data from specified file formats based on pattern from given folder path as input to table in JDBC	62-89
7.	Implement a Virtual Constructor using Factory Design Pattern	90-99
8.	Read files and write to JDBC using multi-Threading	100-111

USER STORY 2

Convert string date to timestamp data type columns based on pattern

Description: Load date as String (data type) into a column and convert them to various timestamp columns based on pattern/format

Add date along with time as String and parse them to below formats into each column

Format:

dd-MM-yyyy

- dd-MM-yyyy'T'HH:mm:ss*SSSZZZZ
- yyyy MMM dd HH:mm:ss.SSS zzz
- MMM dd HH:mm:ss ZZZZ yyyy
- dd/MMMM/yyyy:HH:mm:ss ZZZZ
- MMM dd yyyy HH:mm:ss
- HH:mm:ss.SSS
- dd/MMMM HH:mm:ss,SSS
- dd-MMM-yyyy HH:mm:ss.SSS

12th June 2021 10pm format data to be loaded into input column in the table and update the table with all the patterns mentioned into the same row with each pattern as column

Input: Can accept all types of date formats as string and should be able to parse it to date formats into different columns - Example Input: 12th June 2021 10pm

Output: Table containing the input row data with different date format and input format

Acceptance Criteria:

- String column to be converted into timestamp data type column where each column holds each patterned date along with time for few columns.
- Log the output to the file along with input and output (entire table)
- Update Azure Board on daily basis.

- Get the use case verified by Technical Lead.
- Complete use case documentation.

CODE:

Main class:

```
package DateFormatter;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import java.io.IOException;
import java.sql.Connection;
import java.sql.SQLException;
import java.text.ParseException;
import java.util.Scanner;

public class MainClass {
    private static final Logger logger = LogManager.getLogger(MainClass.class);
    public static void main(String[] args) throws ParseException, SQLException, IOException {
        Scanner sc = new Scanner(System.in);
        logger.info("Enter the date to convert into different formats:");
        String strDate = sc.nextLine();
        //Creation of object to DateFormatter.ConnectionClass
        ConnectionClass c = new ConnectionClass();
        //calling getConnection() Method which returns connection object
        Connection con = c.getConnection();
        //Creation of object to DateFormatter.DateFormatClass
        DateFormatClass dfc = new DateFormatClass();
        //calling convertDateFormat to convert into different formats
        dfc.convertDateFormat(strDate);
        //calling insertToDatabase method to insert data to database
        dfc.insertToDatabase(strDate, con);
    }
}
```

Explanation :

- Imported all the required packages.
- **java.io.IOException** : is an exception which programmers use in the code to throw a failure in Input & Output operations.
- **java.sql.Connection**: is an exception which programmers use in the code to throw a failure in sql connection.

- **java.util.Scanner** : To use Scanner class for taking input from user.
- **java.sql.SQLException** : It provides info on database access error.
- **java.text.ParseException** : It is used to throw exception when date is unable to parse into any format.
- The method getLogger() method belongs to LogManager class.
- Created a logger object which is used to log input and output to log file.
- The info() method of a Logger class is used to log an info message.
- Scanner class in Java is found in the java.util package.
- Java provides various ways to read input from the keyboard, the java.util.Scanner class is one of them.
- Used Scanner class to take input.
- `nextLine()` : It is used to get the input string from user.
- Created objects to classes and called required method using created object.

Output:

```
16:42:07.512 [main] INFO  DateFormatter.MainClass - Enter the date to convert into different formats:  
13th march 2009 4pm
```

Connection class :

```
package DateFormatter;

import org.json.simple.parser.JSONParser;
import org.json.simple.JSONObject;
import org.json.simple.parser.ParseException;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.sql.*;
import java.util.Base64;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

public class ConnectionClass {
    private static final Logger logger = LogManager.getLogger(ConnectionClass.class);
    //getConnection() method returns a connection object
    public Connection getConnection() throws IOException, SQLException {
        //Created a JSONParser object
        JSONParser jp = new JSONParser();
        //Creation of Connection Object
        Connection c = null;
        try {
            //Initialised JSONObject class object which is used to access values based on keys
            JSONObject jo = (JSONObject) jp.parse(new FileReader("src/test/config.json"));
            String url=(String)jo.get("Url");
            //username from config.json
            String username =(String) jo.get("username");
            //Encrypted password from config.json
            String pwd = (String)jo.get("Password");
            //Code to decrypt password
            Base64.Decoder d = Base64.getUrlDecoder();
            String password = new String(d.decode(pwd));
            try {
                //Initialization of connection object
                c = DriverManager.getConnection(url, username, password);
                logger.info("Connection successfully established with database. . .");
            }
            catch (SQLException e) {
                e.printStackTrace();
            }
        }
        catch (ParseException e) {
            e.printStackTrace();
        }
        return c;
    }
}
```

Explanation :

- **java.io.IOException** : is an exception which programmers use in the code to throw a failure in Input & Output operations.
- **java.sql.Connection**: is an exception which programmers use in the code to throw a failure in sql connection
- The method `getLogger()` method belongs to `LogManager` class.
- **java.util.Base64** :Has static methods for obtaining encoders and decoders for the Base64 encoding schema.
- Here user name and encrypted password are present in `config.json` file.
- `JSONParser` is used to parse json data.
- `JSONObject` is used to access values based on keys from json String.
- **Base64.Decoder** for decoding byte data using Base64 encoding scheme .
- The **`getConnection()`** method of `DriverManager` class is used to establish connection with the database.
- `getConnection()` method requires three parameters Connection url, Username, Password.
- Here url is `"jdbc:postgresql://w3.training5.modak.com:5432/training"`
 - "training" is database name.
 - "5432" is port number.
 - "w3.training5.modak.com" is the hostname on which postgresql is running.
- Username and password are taken from `config.json`.

Config.json File :

```
{  
  "Username": "mt4020",  
  "Password": "bXQ0MDIwQG0wMnkyMg",  
  "url": "jdbc:postgresql://w3.training5.modak.com:5432/training"  
}
```

Output:

```
16:42:11.043 [main] INFO  DateFormatter.ConnectionClass - Connection successfully established with database. . .
```

DateFormatterClass:

```
package DateFormatter;
```

```
import java.sql.*;
```

```
import java.text.ParseException;
```

```
import java.text.SimpleDateFormat;
```

```
import java.util.Date;
```

```
import org.antlr.stringtemplate.StringTemplate;
```

```
import org.apache.logging.log4j.LogManager;
```

```
import org.apache.logging.log4j.Logger;
```

```
import org.stringtemplate.v4.ST;
```

```
import org.stringtemplate.v4.STGroup;
```

```
import org.stringtemplate.v4.STGroupFile;
```

```
public class DateFormatClass {
```

```
    private static final Logger logger = LogManager.getLogger(DateFormatClass.class);
```

```
    String str1 = null, str2 = null, str3 = null, str4 = null, str5 = null, str6 = null, str7 = null,  
    str8 = null;
```

```
    //Method convert date to different formats
```

```
    void convertDateFormat(String strDate) throws ParseException {
```

```
//parse given string to date format
strDate=strDate.substring(0,2)+strDate.substring(4,strDate.length());

logger.info(strDate);

Date date = new SimpleDateFormat("dd MMMM yyyy hhaa").parse(strDate);
//parse date to the format "E, dd MMM yyyy HH:mm:ss Z"
SimpleDateFormat simpleformat1 =
new SimpleDateFormat("E, dd MMM yyyy HH:mm:ssZ");
str1 = String.valueOf(simpleformat1.format(date));
logger.info("Date parsed to format \"E, dd MMM yyyy HH:mm:ss Z\" is : " + str1);
//parse date to the format "dd/MMM/yyyy:HH:mm:ss ZZZZ"
SimpleDateFormat simpleformat2 =
new SimpleDateFormat("dd/MMM/yyyy:HH:mm:ssZZZZ");
str2 = String.valueOf(simpleformat2.format(date));
logger.info("Date parsed to format \"dd/MMM/yyyy:HH:mm:ss ZZZZ\" is : " + str2);
//parse date to the format "MMM dd HH:mm:ss ZZZZ yyyy"
SimpleDateFormat simpleformat3 =
new SimpleDateFormat("MMM dd HH:mm:ss ZZZZ yyyy");
str3 = String.valueOf(simpleformat3.format(date));
logger.info("Date parsed to format \"MMM dd HH:mm:ss ZZZZ yyyy\" is : " + str3);
//parse date to the format "yyyy MMM dd HH:mm:ss.SSS zzz"
SimpleDateFormat simpleformat4 =
new SimpleDateFormat("yyyy MMM dd HH:mm:ss.SSS zzz");
str4 = String.valueOf(simpleformat4.format(date));
logger.info("Date parsed to format \"yyyy MMM dd HH:mm:ss.SSS zzz\" is : " + str4);
//parse date to the format "dd/MMM HH:mm:ss,SSS"
SimpleDateFormat simpleformat5 = new SimpleDateFormat("dd/MMM HH:mm:ss,SSS");
str5 = String.valueOf(simpleformat5.format(date));
logger.info("Date parsed to format \"dd/MMM HH:mm:ss,SSS\" is : " + str5);
```



```

//parse date to the format "dd-MMM-yyyy HH:mm:ss.SSS"
SimpleDateFormat simpleformat6 =
new SimpleDateFormat("dd-MMM-yyyy HH:mm:ss.SSS");
str6 = String.valueOf(simpleformat6.format(date));
logger.info("Date parsed to format \"dd-MMM-yyyy HH:mm:ss.SSS\" is : " + str6);
//parse date to the format "HH:mm:ss.SSS"
SimpleDateFormat simpleformat7 = new SimpleDateFormat("HH:mm:ss.SSS");
str7 = String.valueOf(simpleformat7.format(date));
logger.info("Date parsed to format \"HH:mm:ss.SSS\" is : " + str7);
//parse date to the format "dd-MM-yyyy'T'HH:mm:ss*SSSZZZZ"
SimpleDateFormat simpleformat8 =
new SimpleDateFormat("dd-MM-yyyy'T'HH:mm:ss*SSSZZZZ");
str8 = String.valueOf(simpleformat8.format(date));
logger.info("Date parsed to format \"dd-MM-yyyy'T'HH:mm:ss*SSSZZZZ\" is : " + str8);}

//Code to insert each format into database
void insertToDatabase(String strDate,Connection c) throws SQLException {
    //creation of Statement Object to execute queriesL
    Statement stm=c.createStatement();
    // Load the file
    STGroup stGroup =
        new STGroupFile("src/main/java/DateFormatter/StringTemplates.stg");
    // Pick the correct template
    ST templateExample1 = stGroup.getInstanceOf("templateExample1");
    // Pass on values to use when rendering
    templateExample1.add("param1", "MT4020_UserStory_2");
    templateExample1.add("param2",str);
    String render1 = templateExample1.render();
    logger.info(render1);
    stm.executeUpdate(render1);
}

```

```
c.close();  
}  
}
```

StringTemplate.stg file:

```
templateExample1(param1,param2) ::= <<  
insert into <param1> values('<param2>');  
>>
```

Explanation :

- **java.util.date** : This class implements date and time in java.
- **org.stringtemplate.v4.STGroup**: This is used for implementing string templates
- **org.stringtemplate.v4.STGroupFile**: This is used for implementing string templates
- Added the dependencies required for String Templates.
- **SimpleDateFormat** : This class provides methods to format and parse date and time to different formats in java.
- First input String is converted into standard date and then by using SimpleDateFormat it is converted into different formats.
- Created a stg file with name "StringTemplates.stg".
- Named template as templateExample1 with two parameters.
- Wrote query for insertion into database.
- Then executed query using executeUpdate().

Output:

```
11:53:21.788 [main] INFO DateFormatter.DateFormatClass - 10 june 2022 10pm
11:53:21.797 [main] INFO DateFormatter.DateFormatClass - Date parsed to format "E, dd MMM yyyy HH:mm:ss Z" is : Fri, 10 Jun 2022 22:00:00 +0530
11:53:21.798 [main] INFO DateFormatter.DateFormatClass - Date parsed to format "dd/MM/yyyy:HH:mm:ss ZZZZ" is : 10/Jun/2022:22:00:00 +0530
11:53:21.798 [main] INFO DateFormatter.DateFormatClass - Date parsed to format "MMM dd HH:mm:ss ZZZZ yyyy" is : Jun 10 22:00:00 +0530 2022
11:53:21.799 [main] INFO DateFormatter.DateFormatClass - Date parsed to format "yyyy MMM dd HH:mm:ss.SSS zzz" is : 2022 Jun 10 22:00:00.000 IST
11:53:21.799 [main] INFO DateFormatter.DateFormatClass - Date parsed to format "dd/MMM HH:mm:ss,SSS" is : 10/Jun 22:00:00,000
11:53:21.799 [main] INFO DateFormatter.DateFormatClass - Date parsed to format "dd-MMM-yyyy HH:mm:ss.SSS" is : 10-Jun-2022 22:00:00.000
11:53:21.799 [main] INFO DateFormatter.DateFormatClass - Date parsed to format "HH:mm:ss.SSS" is : 22:00:00.000
11:53:21.800 [main] INFO DateFormatter.DateFormatClass - Date parsed to format "dd-MM-yyyy'T'HH:mm:ss*SSSZZZZ" is : 10-06-2022T22:00:00+000+0530
```

Data in database :

14th september 2000 7 pm	Thu, 14 Sep 2000 19:00:00 +0530	14/Sep/2000:19:00:00 +0530	Sep 14 19:00:00 +0530 2000	2000 Sep 14 19:00:00.000 IST	14/Sep 19:00:00,000	14-Sep-2000 19:00:00.000	19:00:00.000	14-09-2000T19:00:00+0530
16th july 2022 9pm	Sat, 16 Jul 2022 21:00:00 +0530	16/Jul/2022:21:00:00 +0530	Jul 16 21:00:00 +0530 2022	2022 Jul 16 21:00:00.000 IST	16/Jul 21:00:00,000	16-Jul-2022 21:00:00.000	21:00:00.000	16-07-2022T21:00:00+0530
03rd august 2000 4pm	Thu, 03 Aug 2000 16:00:00 +0530	03/Aug/2000:16:00:00 +0530	Aug 03 16:00:00 +0530 2000	2000 Aug 03 16:00:00.000 IST	03/Aug 16:00:00,000	03-Aug-2000 16:00:00.000	16:00:00.000	03-08-2000T16:00:00+0530
10th june 2022 5pm	Fri, 10 Jun 2022 17:00:00 +0530	10/Jun/2022:17:00:00 +0530	Jun 10 17:00:00 +0530 2022	2022 Jun 10 17:00:00.000 IST	10/Jun 17:00:00,000	10-Jun-2022 17:00:00.000	17:00:00.000	10-06-2022T17:00:00+0530
10th september 2022 3pm	Sat, 10 Sep 2022 15:00:00 +0530	10/Sep/2022:15:00:00 +0530	Sep 10 15:00:00 +0530 2022	2022 Sep 10 15:00:00.000 IST	10/Sep 15:00:00,000	10-Sep-2022 15:00:00.000	15:00:00.000	10-09-2022T15:00:00+0530
10th september 2022 3pm	Sat, 10 Sep 2022 15:00:00 +0530	10/Sep/2022:15:00:00 +0530	Sep 10 15:00:00 +0530 2022	2022 Sep 10 15:00:00.000 IST	10/Sep 15:00:00,000	10-Sep-2022 15:00:00.000	15:00:00.000	10-09-2022T15:00:00+0530

User Story 3

3.Create an application that connects to database and performs CRUD operations using JDBC Querying.

Description: Connect to the database and perform various Read, Write, Update and Delete based on the user input data where user has ability to pick the schema name and table name from the database.

Handle the transactions using last modified timestamp (insert time at which all the transaction happened) and last modified by user id (use case developer)

Input:

Schema name, table name in the database, the operation to perform (Read/Update/Delete/Write), and the data to perform the operation.

Output:

Verify if the operations were performed as expected from the database.

To verify the update and delete use a last-modified timestamp and last modified by (user-id) columns to differentiate.

Acceptance Criteria:

- Log the output to the file along with input and output
- Load the tables with different values if required based on the use case
- Operations should perform as expected.
- Update Azure Board on daily basis.
- Get the use case verified by Technical Lead.
- Complete use case documentation.

CODE:

Main Class :

```
package PerformCURDOperations;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

import java.io.IOException;
import java.sql.*;
import java.util.Scanner;

public class Main {
    private static final Logger logger = LogManager.getLogger(Main.class);
    public static void main(String[] args) throws IOException, SQLException {
        //Taking Schema name from input
        Scanner sc = new Scanner(System.in);
        logger.info("Enter Schema name:");
        //Taking Table name from input
        String schemaName = sc.nextLine();
        logger.info("Enter Table name:");
        String tableName = sc.nextLine();
        ConnectionClass fw = new ConnectionClass();
        CurdOperations co = new CurdOperations();
        Connection cn = fw.getConnection();
        while (true){
            logger.info("which operation you want to perform:");
            String strMethod = sc.nextLine();
            switch (strMethod) {
```

```
case "insert":
    //For insert Operation
    co.insertMethod(cn, schemaName, tableName);
    logger.info("Insertion performed successfully");
    break;
case "update":
    //For Update Operation
    co.updateMethod(cn, schemaName, tableName);
    logger.info("Updation performed successfully");
    break;
case "delete":
    //For delete Operation
    co.deleteMethod(cn, schemaName, tableName);
    logger.info("Deletion performed successfully");
    break;
case "retrieve":
    //For Retrieve Operation
    co.retriveMethod(cn, schemaName, tableName);
    logger.info("Data Retrieved successfully");
    break;
default:
    logger.info("No Operation");
}
logger.info("If you want to continue then enter yes or else no");
String str1 = sc.nextLine();
if (str1.equals("no")){
    break;
}
```

```

else {
    continue;}
}
}
}

```

Explanation:

- **java.io.IOException** : is an exception which programmers use in the code to throw a failure in Input & Output operations.
- **java.sql.Connection**: is an exception which programmers use in the code to throw a failure in sql connection.
- **java.util.Scanner** : To use Scanner class for taking input from user.
- **java.sql.SQLException** : It provides info on database access error.
- **java.text.ParseException** : It is used to throw exception when date is unable to parse into any format.
- The method getLogger() method belongs to LogManager class.
- Created a logger object which is used to log input and output to log file.
- The info() method of a Logger class is used to log an info message.
- Scanner class in Java is found in the java.util package.
- Java provides various ways to read input from the keyboard, the java.util.Scanner class is one of them.
- Used Scanner class to take input.
- `nextLine()` : It is used to get the input string from user.
- Used switch statement to select anyone of the CRUD operation.
- When user input insert it go to insert operation in the same way other operations works.
- If user enter other than CURD operation it prints "No Operation doesn't exist".
- If we enter 'no' it exit the program or else it ask again to perform which operation .

Output :

```
19:10:06.068 [main] INFO PerformCURDOperations.Main - Enter Schema name:
public
19:10:09.733 [main] INFO PerformCURDOperations.Main - Enter Table name:
userstory03_1
```

Connection class :

```
package PerformCURDOperations;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;
import org.json.simple.parser.ParseException;

import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.Base64;

public class ConnectionClass {
    private static final Logger logger = LogManager.getLogger(ConnectionClass.class);
    //getConnection() method returns a connection object
    public Connection getConnection() throws IOException, SQLException {
        //Created a JSONParser object
        JSONParser jp = new JSONParser();
        //Creation of Connection Object
        Connection c = null;
        try {
            //Initialised JSONObject class object which is used to access values based on keys
            JSONObject jo = (JSONObject) jp.parse(new FileReader("src/test/config.json"));
            String Url = (String) jo.get("url");
            //username from config.json
```



```

//Encrypted password from config.json
String pwd = (String)jo.get("Password");
//Code to decrypt password
Base64.Decoder d = Base64.getURLDecoder();
String password = new String(d.decode(pwd));
try {
    //Initialization of connection object
    c = DriverManager.getConnection(Url, username, password);
    logger.info("Connection successfully established with database. . .");
}
catch (SQLException e) {
    e.printStackTrace();
}
}
catch (ParseException e) {
    e.printStackTrace();
}
}
return c;
}
}

```

Explanation :

- **java.io.IOException** : is an exception which programmers use in the code to throw a failure in Input & Output operations.
- **java.sql.Connection**: is an exception which programmers use in the code to throw a failure in sql connection
- The method getLogger() method belongs to LogManager class.
- **java.util.Base64** :Has static methods for obtaining encoders and decoders for the Base64 encoding schema.
- Here user name and encrypted password are present in config.json file.
- JSONParser is used to parse json data.
- JSONObject is used to access values based on keys from json String.
- **Base64.Decoder** for decoding byte data using Base64 encoding scheme .
- The **getConnection()** method of DriverManager class is used to establish connection with the database.

- getConnection() method requires three parameters Connection url, Username, Password.
- Here url is "jdbc:postgresql://w3.training5.modak.com:5432/training"
 - "training" is database name.
 - "5432" is port number.
 - "w3.training5.modak.com" is the hostname on which postgresql is running.
- Username and password are taken from config.json.

Config.json File :

```
{
  "Username": "mt4020",
  "Password": "bXQ0MDIwQ60wMnkyMg",
  "url": "jdbc:postgresql://w3.training5.modak.com:5432/training"
}
```

Output:

```
9:10:15.285 [main] INFO PerformCURDOperations.ConnectionClass - Connection successfully established with database. . .
```

CurdOperation Class:

Insert Operation:

```
public void insertMethod(Connection connection,String schemaName,String tableName )
```

```
throws SQLException {
```

```
    STGroup stGroup = new
```

```
STGroupFile("src/main/java/PerformCURDOperations/StringTemplate.stg");
```

```
    // Pick the correct template
```

```
    ST template1 = stGroup.getInstanceOf("templateForInsertion");
```

```
    ST template2 = stGroup.getInstanceOf("templateForSelection1");
```

```
    ST template3 = stGroup.getInstanceOf("templateForRetrieve");
```

```
    // Pass on values to use when rendering
```

```
    Statement stm = connection.createStatement();
```

```

template2.add("param1",tableName);
template3.add("param1",tableName);
String query1 = template2.render();
String query2 = template3.render();
logger.info(query1);
logger.info(query2);
ResultSet rs1 = stm.executeQuery(query1);
String str = "",strc="";
while (rs1.next()) {
    str = str + rs1.getString(2) + " ";
    strc=strc+rs1.getString(1)+" ";
}
ResultSet rs2 = stm.executeQuery(query2);
ResultSetMetaData rsmd2 = rs2.getMetaData();
int NumOfCol = rsmd2.getColumnCount();
logger.info(NumOfCol);
String arr[] = str.split(" ");
String arr1[]=strc.split(" ");
String col = "";
for (int i = 1; i <= NumOfCol - 3; i++) {
    logger.info("Name of [" + i + "] Column=" + rsmd2.getColumnName(i));
    String str1 = sc.nextLine();
    if (arr[i] == "integer") {
        int n = Integer.parseInt(str1);
        col = col + n + ",";
    }
    if (arr[i] == "double precision") {
        double d = Double.parseDouble(str1);
        col = col + d + ",";
    }
}

```

```

    } else {
        col = col + "" + str1 + "" + ",";
    }
}

col=col+"CURRENT_TIMESTAMP";
template1.add("param1", tableName);
template1.add("param2",col);
String query = template1.render();
logger.info(query);
stm.executeUpdate(query);
}

```

StringTemplate.stg:

```

templateForInsertion(param1,param2) ::= <<
insert into <param1> values(<param2>);
>>

```

Explanation:

- Here Insert Operation is performed when called insertMethod().
- The statement interface is used to create SQL basic statements in Java it provides methods to execute queries with the database.
- Used executeUpdate() to execute insert statement.
- Values for insertion are taken from input.
- Converted each values to appropriate datatype using parser.

Update Operation :

```

public void updateMethod(Connection connection,String schemaName,String tableName)
throws SQLException {
    Statement stm = connection.createStatement();
    STGroup stGroup = new

```

```

STGroupFile("src/main/java/PerformCURDOperations/StringTemplate.stg");

// Pick the correct template
ST template1 = stGroup.getInstanceOf("templateForSelection1");
ST template2 = stGroup.getInstanceOf("templateForRetrieve");
template1.add("param1",tableName);
template2.add("param1",tableName);
String query1 = template1.render();
String query2 = template2.render();
logger.info(query1);
logger.info(query2);
ResultSet rs1 = stm.executeQuery(query1);
String str = "",strc="";
while (rs1.next()) {
    str = str + rs1.getString(2) + " ";
    strc=strc+rs1.getString(1)+" ";
}
String arr[] = str.split(" ");
String arr1[]=strc.split(" ");
ResultSet rs = stm.executeQuery(query2);
ResultSetMetaData rsmd = rs.getMetaData();
int NumOfCol = rsmd.getColumnCount();
logger.info("Enter column name you want to update:");
for(int i=0;i<NumOfCol-3;i++){
    System.out.print(arr1[i]+" ");
}
String str1=sc.nextLine();
logger.info("Enter value to update:");
String str2=sc.nextLine();
Object n = null;

```

```

logger.info("Enter the id which you want to update:");
int uid = sc.nextInt();
logger.info("Enter user id:");
int usid = sc.nextInt();
n=(Object) str2;
ST template3 = stGroup.getInstanceOf("templateForUpdation");
template3.add("param1",tableName);
template3.add("param2",str1);
template3.add("param3",n);
template3.add("param4",usid);
template3.add("param5",uid);
String query3 = template3.render();
logger.info(query3);
stm.executeUpdate(query3);
}

```

StringTemplate.stg:

```

templateForUpdation(param1,param2,param3,param4,param5) ::= <<
update <param1> set <param2>='<param3>',modified_time=CURRENT_TIMESTAMP,modified_by=<param4> where id=<param5>;
>>

```

Explanation:

- Here Update Operation performed when updateMethod() is called.
- The statement interface is used to create SQL basic statements in Java it provides methods to execute queries with the database.
- Used executeUpdate() to execute update statement.
- Based on id update Operation is performed.
- When update Operation is performed, Updated the modified_time and modified_by columns.
- Used String Templates to write queries.

Retrieve Operation :

```
public void retrieveMethod(Connection connection,String schemaName,String tableName)
throws SQLException {
    Statement stm = connection.createStatement();
    STGroup stGroup = new
STGroupFile("src/main/java/PerformCURDOperations/StringTemplate.stg");
    ST template1 = stGroup.getInstanceOf("templateForRetrieve");
    template1.add("param1",tableName);
    String query1 = template1.render();
    logger.info(query1);
    ResultSet rs = stm.executeQuery(query1);
    ResultSetMetaData rsmd = rs.getMetaData();
    int NumOfCol = rsmd.getColumnCount();
    for (int i = 1; i <= NumOfCol; i++) {
        System.out.print(rsmd.getColumnName(i) + " ");
    }
    System.out.println();
    while (rs.next()) {
        for (int i = 1; i <= NumOfCol; i++) {
            System.out.print(rs.getObject(i) + " ");

        }
        System.out.println();
    }
}
```

StringTemplate.stg:

```
templateForRetrieve(param1) ::= <<
select * from <param1>;
>>
```

Explanation:

- Here Select Operation performed when retrieveMethod() is called.
- The statement interface is used to create SQL basic statements in Java it provides methods to execute queries with the database.
- Used executeQuery() to execute select statement.
- ResultSetMetaData: Used to get metadata of a table like total number of column, column name, column type etc. ,
- getColumnName(int): It returns the column name of the specified column index.
- Used String Templates to write query.

Delete Method:

```
public void deleteMethod(Connection connection,String schemaName,String table_Name)
throws SQLException {
    Statement stm = connection.createStatement();
    logger.info("Enter id of item you want to delete:");
    int id = sc.nextInt();
    STGroup stGroup = new
STGroupFile("src/main/java/PerformCURDOperations/StringTemplate.stg");
    ST template1 = stGroup.getInstanceOf("templateForDeletion");
    template1.add("param1",table_Name);
    template1.add("param2",id);
    String query1 = template1.render();
    logger.info(query1);
    stm.executeUpdate(query1);
}
}
```


StringTemplate.stg:

```
templateForDeletion(param1,param2) ::= <<
delete from <param1> where id=<param2>;
>>
```

Expalantion :

- Here delete Operation is performed when deleteMethod() is called.
- The statement interface is used to create SQL basic statements in Java it provides methods to execute queries with the database.
- Used executeUpdate() to execute delete statement.
- The row is deleted based on id which is entered by the user.
- Used String Templates to write query.

Output:

- Data in Database

select * from userstory03_1 | *Enter a SQL expression to filter results (use Ctrl+Space)*

	123 id	abc ename	123 sal	created_time	modified_time	123 modified_by
1	3	manu	3,000,000	2022-04-27 23:20:24.929	2022-04-28 11:48:07.002	12,345
2	1	sathwi reddy	200,000	2022-04-30 19:14:17.392	[NULL]	[NULL]

- Data in LogFiles:

```

2022-05-04 12:39:03.674 [main] INFO PerformCURDOperations.Main - Enter Schema name:
"2022-05-04 12:39:07.300 [main] INFO PerformCURDOperations.Main - Enter Table name:
"2022-05-04 12:39:17.609 [main] INFO PerformCURDOperations.ConnectionClass - Connection successfully established with dat
"2022-05-04 12:39:17.610 [main] INFO PerformCURDOperations.Main - which operation you want to perform:
"2022-05-04 12:39:23.122 [main] INFO PerformCURDOperations.ConnectionClass - select COLUMN_NAME,DATA_TYPE from INFORMATIO
"2022-05-04 12:39:23.122 [main] INFO PerformCURDOperations.ConnectionClass - select * from userstory03_1;
"2022-05-04 12:39:23.179 [main] INFO PerformCURDOperations.ConnectionClass - 6
"2022-05-04 12:39:23.179 [main] INFO PerformCURDOperations.ConnectionClass - Name of [1] Column=id
"2022-05-04 12:39:27.738 [main] INFO PerformCURDOperations.ConnectionClass - Name of [2] Column=ename
"2022-05-04 12:40:16.414 [main] INFO PerformCURDOperations.ConnectionClass - Name of [3] Column=sal
"2022-05-04 12:40:16.414 [main] INFO PerformCURDOperations.ConnectionClass - insert into userstory03_1 values('4','sathwa
"2022-05-04 12:40:16.431 [main] INFO PerformCURDOperations.Main - Insertion performed successfully
"2022-05-04 12:40:16.431 [main] INFO PerformCURDOperations.Main - If you want to continue then enter yes or else no
"2022-05-04 12:40:22.442 [main] INFO PerformCURDOperations.Main - which operation you want to perform:
"2022-05-04 12:40:26.326 [main] INFO PerformCURDOperations.ConnectionClass - select * from userstory03_1;
"2022-05-04 12:40:26.332 [main] INFO PerformCURDOperations.Main - Data Retrieved successfully
"2022-05-04 12:40:26.332 [main] INFO PerformCURDOperations.Main - If you want to continue then enter yes or else no
"2022-05-04 12:43:21.444 [main] INFO PerformCURDOperations.Main - Enter Schema name:
"2022-05-04 12:43:25.649 [main] INFO PerformCURDOperations.Main - Enter Table name:
"2022-05-04 12:43:31.665 [main] INFO PerformCURDOperations.ConnectionClass - Connection successfully established with dat
"2022-05-04 12:43:31.666 [main] INFO PerformCURDOperations.Main - which operation you want to perform:
"2022-05-04 12:43:35.562 [main] INFO PerformCURDOperations.ConnectionClass - select COLUMN_NAME,DATA_TYPE from INFORMATIO
"2022-05-04 12:43:35.563 [main] INFO PerformCURDOperations.ConnectionClass - select * from userstory03_1;
"2022-05-04 12:43:35.620 [main] INFO PerformCURDOperations.ConnectionClass - Enter column name you want to update:
"2022-05-04 12:43:44.843 [main] INFO PerformCURDOperations.ConnectionClass - Enter value to update:
"2022-05-04 12:43:54.304 [main] INFO PerformCURDOperations.ConnectionClass - Enter the id which you want to update:
"2022-05-04 12:43:58.531 [main] INFO PerformCURDOperations.ConnectionClass - Enter user id:
"2022-05-04 12:44:04.090 [main] INFO PerformCURDOperations.ConnectionClass - update userstory03_1 set ename='rama',modifi
"2022-05-04 12:44:04.098 [main] INFO PerformCURDOperations.Main - Updation performed successfully
"2022-05-04 12:44:04.098 [main] INFO PerformCURDOperations.Main - If you want to continue then enter yes or else no
"

```

User Story 4

4. Implement all types of Inheritance with real time examples

Description: Implement single, multiple, hierarchical and other inheritances

Acceptance Criteria:

- All types on Inheritance are implemented
- Understanding why multiple inheritance is not possible

CODE:

SINGLE INHERITANCE :

Code in Main class:

```
package InheritanceImplementation.SingleInheritance;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
public class Main {
    private static final Logger logger =
LogManager.getLogger(InheritanceImplementation.HierarchicalInheritance.Main.class);
    public static void main(String[] args) {
        PerformCalculation pc=new PerformCalculation();
        pc.perfrom();
        logger.info("Successfully implemented single Inheritance");
    }
}
```

Explanation:

- Single Inheritance : In Single Inheritance sub class inherit the features of one super class.
- Created object for child class and called method which is in child class.

Code in Base class :

```
class Calculation {  
    //Implemented some methods in base class  
    int sum(int i , int j)  
    {  
        return i+j;  
    }  
    int subtract(int i , int j)  
    {  
        return i-j;  
    }  
    float division(int i,int j){  
        return i/j;  
    }  
    double multiplication(int i,int j){ return i*j;}  
}
```

Explanation :

- It is a base class with four methods sum(), subtract(), division(), multiplication().
- sum() and subtract() method has return type int.
- division() method has return type float.
- multiplication() has return type double.

Code in Sub class :

```
class PerformCalculation extends Calculation{  
    private static final Logger logger = LogManager.getLogger(PerformCalculation.class);  
    //called methods which are in base class by using inheritance  
    void perfrom(){  
        logger.info("Called sum method which is in base class");  
        logger.info( sum(1,2));  
    }  
}
```

```

        logger.info("Called sum subtract which is in base class");
        logger.info(subtract(1,2));
        logger.info("Called division method which is in base class");
        logger.info(division(1,2));
        logger.info("Called multiplication method which is in base class");
        logger.info(multiplication(1,2));
    }

```

Explanation :

- Called methods which are in base class.
- Extended Calculation class and called sum(), subtract(), division(), multiplication() methods in sub class PerformCalculation.
- Logger class is used to log messages for specific application.
- The getLogger() method is used to get the specified Logger in this LogManager instance.
- The info () method of a Logger class is used to Log an INFO message.
- After calling perform() method which is in subclass the output

Output:

```

"C:\Program Files\Java\jdk1.8.0_261\bin\java.exe" ...
15:37:54.192 [main] INFO InheritanceImplementation.SingleInheritance.PerformCalculation - Called sum method which is in base class
15:37:54.196 [main] INFO InheritanceImplementation.SingleInheritance.PerformCalculation - 3
15:37:54.196 [main] INFO InheritanceImplementation.SingleInheritance.PerformCalculation - Called sum subtract which is in base class
15:37:54.197 [main] INFO InheritanceImplementation.SingleInheritance.PerformCalculation - -1
15:37:54.197 [main] INFO InheritanceImplementation.SingleInheritance.PerformCalculation - Called division method which is in base class
15:37:54.197 [main] INFO InheritanceImplementation.SingleInheritance.PerformCalculation - 0.0
15:37:54.197 [main] INFO InheritanceImplementation.SingleInheritance.PerformCalculation - Called multiplication method which is in base class
15:37:54.198 [main] INFO InheritanceImplementation.SingleInheritance.PerformCalculation - 2.0
15:37:54.198 [main] INFO InheritanceImplementation.HierarchicalInheritance.Main - Successfully implemented single Inheritance

```

Multilevel Inheritance :

- In this Inheritance a class extends other class which is a child class of other class.
- The below figure shows multilevel inheritance.

Code in Main class :

```

package InheritanceImplementation.MultilevelInheritance;

import org.apache.logging.log4j.LogManager;

```

```

import org.apache.logging.log4j.Logger;

public class Main {

    private static final Logger logger = LogManager.getLogger(Main.class);

    public static void main(String[] args) {

        FerrariF8Tributo ff=new FerrariF8Tributo();

        logger.info("Features of Ferrari F8 Tributo");

        ff.FerrariF8tributoFeatures();

        ff.Ferrarifeatures();

        ff.carFeatures();

        ff.autoMobilesFeatures();

        ff.function();

        logger.info(ff.price);

        logger.info(ff.milleage);

        logger.info(ff.noofseats);

        logger.info(ff.fuelType);

        logger.info(ff.Steering);

        logger.info(ff.noofWhells);

    }

}

```

Explanation:

- Here created a object for FerrariF8Tributo class and accessed all the methods of its super classes.
- Here it getting properties from it base class, and its base class getting properties from its super class.
- Logger class is used to log messages for specific application.
- The getLogger() method is used to get the specified Logger in this LogManager instance.
- The info () method of a Logger class is used to Log an INFO message.

Code implementing Multilevel Inheritance:

```

package InheritanceImplementation.MultilevelInheritance;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

public class VechileClass{
    private static final Logger logger = LogManager.getLogger(VechileClass.class);
    void function(){
        logger.info("Vechiles are used for transpotation");
    }
}

class AutomobileVechile extends VechileClass{
    private static final Logger logger = LogManager.getLogger(AutomobileVechile.class);
    void autoMobilesFeatures(){
        logger.info("This are Motor driven type");
    }
}

class Car extends AutomobileVechile {
    private static final Logger logger = LogManager.getLogger(Car.class);
    int noofWhells=4;
    public void vehicleType()
    {
        logger.info("Vehicle Type: Car");
    }
    public void carFeatures(){
        logger.info("Cras have Leather seats , Bluetooth , Backup camera etc..");
    }
}

class Ferrari extends Car{

```

```

private static final Logger logger = LogManager.getLogger(Ferrari.class);

String Steering="power Steering";

String fuelType = "Petrol";

void Ferrarifeatures(){

    logger.info("Ferrari has features like Anti Lock Braking System,Air Conditioner,Automatic
Climate Control etc..");

}

}

class FerrariF8Tributo extends Ferrari{

    private static final Logger logger = LogManager.getLogger(FerrariF8Tributo.class);

    float milleage= 7.75f;

    String price="4.02cr";

    int noofseats=2;

    void FerrariF8tributoFeatures() {

        logger.info("It has features like Power Windows,Front Passenger Airbag,Alloy Wheels
etc..");

    }}

```

Explanation:

- Here AutomobileVechile extends vechileClass and then Car extends AutomobileVechile
- So, Car has features of both VechileClass and AutomobileVechile .
- Ferrari extends Car and then FerrariF8Tributo extends Ferrari.
- FerrariF8Tributo has features of all the classes.

Output :

```

15:53:08.061 [main] INFO InheritanceImplementation.MultilevelInheritance.Car - Cars have leather seats , Bluetooth , Backup camera etc
15:53:08.061 [main] INFO InheritanceImplementation.MultilevelInheritance.AutomobileVechile - This are Motor driven type
15:53:08.062 [main] INFO InheritanceImplementation.MultilevelInheritance.VechileClass - Vechiles are used for transpotation
15:53:08.062 [main] INFO InheritanceImplementation.MultilevelInheritance.Main - 4.02cr
15:53:08.064 [main] INFO InheritanceImplementation.MultilevelInheritance.Main - 7.75
15:53:08.064 [main] INFO InheritanceImplementation.MultilevelInheritance.Main - 2
15:53:08.064 [main] INFO InheritanceImplementation.MultilevelInheritance.Main - Petrol
15:53:08.064 [main] INFO InheritanceImplementation.MultilevelInheritance.Main - power Steering
15:53:08.065 [main] INFO InheritanceImplementation.MultilevelInheritance.Main - 4
15:53:08.065 [main] INFO InheritanceImplementation.MultilevelInheritance.Main - Successfully implemented Multilevel Inheritance.

```

Hierarchical Inheritance :

- The multiple child classes inherit the single class or the single class is inherited by multiple child class.

Code in Base Class :

```
class BirdsClass {
    static final Logger logger = LogManager.getLogger(BirdsClass.class);
    int noOfWings=2;
    int noOfLegs=2;
    String birdsClass="Aves";
    void method(){
        logger.info("Birds are endothermic and their lay eggs to reproduce");
    }
}
```

Explanation :

- Created a object for another class.
- Than called methods of that class using created object.
- Logger class is used to log messages for specific application.
- The getLogger() method is used to get the specified Logger in this LogManager instance.
- The info () method of a Logger class is used to Log an INFO message.
- Here BirdsClass is the base class and is inherited by two classes.
- Code in two child classes :
- Code in child class Pigeon.
- Pigeon class extends BirdsClass so it get all features of BirdsClass.

Code in child class Pigeon :

```
class Pigeon extends BirdsClass{
    String lengthOfBird= "34 to 35 cm";
    String sizeOfEggs="4 to 5cm";
    void pigeonEats(){
        logger.info("Pigeon eats seeds and grains");
    }
}
```

```

    }
}

```

Code in child class Parrot.

```

class Parrot extends BirdsClass{

    String lengthOfBird="8 to 100 cm";

    String sizeOfEggs="2.5 to 3.5";

    void parrotEats(){

        logger.info("Parrot eats nuts,flowers,fruit,seeds and insects");

    }

}

```

Output :

```

C:\Program Files\Java\jdk1.8.0_261\bin\java.exe ...
15:57:41.492 [main] INFO  InheritanceImplementation.HierarchialInheritance.Main - Pigeon bird characteristics
15:57:41.496 [main] INFO  InheritanceImplementation.HierarchialInheritance.Main - pigeon has 2 legs
15:57:41.497 [main] INFO  InheritanceImplementation.HierarchialInheritance.Main - pigeon has 2 wings
15:57:41.497 [main] INFO  InheritanceImplementation.HierarchialInheritance.Main - pigeon belongs to Avesclass
15:57:41.497 [main] INFO  InheritanceImplementation.HierarchialInheritance.BirdsClass - Pigeon eats seeds and grains
15:57:41.497 [main] INFO  InheritanceImplementation.HierarchialInheritance.BirdsClass - Birds are endothermic and their lay eggs to reproduce
15:57:41.497 [main] INFO  InheritanceImplementation.HierarchialInheritance.Main - Parrot bird characteristics
15:57:41.497 [main] INFO  InheritanceImplementation.HierarchialInheritance.Main - Parrot has 2 legs
15:57:41.497 [main] INFO  InheritanceImplementation.HierarchialInheritance.Main - Parrot has 2 wings
15:57:41.498 [main] INFO  InheritanceImplementation.HierarchialInheritance.Main - Parrot belongs to Avesclass
15:57:41.498 [main] INFO  InheritanceImplementation.HierarchialInheritance.BirdsClass - Parrot eats nuts,flowers,fruit,seeds and insects
15:57:41.498 [main] INFO  InheritanceImplementation.HierarchialInheritance.BirdsClass - Birds are endothermic and their lay eggs to reproduce
15:57:41.498 [main] INFO  InheritanceImplementation.HierarchialInheritance.Main - Successfully implemented Hierarchical Inheritance.

```

Hybrid Inheritance :

- A hybrid inheritance is a combination of more than one types of inheritance.
- It is combination of multiple and multi level inheritance.

Code in Main Class :

```

package InheritanceImplementation.HybridInheritance;

import org.apache.logging.log4j.LogManager;

import org.apache.logging.log4j.Logger;

public class Main {

```

```

private static final Logger logger = LogManager.getLogger(Main.class);
public static void main(String[] args) {
    Child c=new Child();
    c.getAge();
    c.looking();
    c.talking();
    c.walking();
    logger.info("My grand mother age"+c.mage);
    logger.info("My grand mother age"+c.fage);
    logger.info("My mother age:"+c.getAge());
    logger.info("Hybrid Inheritance implemented successfully");
}
}

```

Explanation :

- Created a object for child class.
- Then called methods of that class and parent classes using created object.
- Logger class is used to log messages for specific application.
- The getLogger() method is used to get the specified Logger in this LogManager instance.
- The info () method of a Logger class is used to Log an INFO message.

Code implementing Hybrid Inheritance :

```

package InheritanceImplementation.HybridInheritance;

```

```

public interface GrandFather {
    int fage=80;
    void talking();
}

```

```

public interface GrandMother {
    int mage=70;
}

```

```
void walking();  
}
```

Explanation :

- Code in two interfaces :
- Created two Interfaces GrandMother and GrandFather.
- An interface has static final variables and can't create an object.
- Interface methods are by default public and abstract.
- Here GrandMother Interface has method walking() and GrandFather has method talking().

Code in Mother Class and Child class:

```
package InheritanceImplementation.HybridInheritance;  
import org.apache.logging.log4j.LogManager;  
import org.apache.logging.log4j.Logger;  
public class Mother implements GrandMother,GrandFather{  
    private static final Logger logger = LogManager.getLogger(Mother.class);  
    int motherAge=40;  
    public void walking(){  
        logger.info("Mother Class Implementing GrandMother Interface...");  
        logger.info("Walking style from grand mother");  
    }  
    public void talking(){  
        logger.info("Mother Class Implementing GrandFather Interface...");  
        logger.info("Talking style from grand father");  
    }  
}  
class Child extends Mother{  
    private static final Logger logger = LogManager.getLogger(Child.class);  
    int getAge(){
```

```

        return super.motherAge;
    }

    void looking(){
        logger.info("Child Class Extending Mother class");
        logger.info("Child Looking like Mother");
    }
}

```

Explanation :

- Created a class which implements two interfaces.
- It gets all properties from two interfaces.
- Hence multiple inheritance achieved.
- Child class extended Parent class.
- It gets properties from Parent Class and two interfaces.
- Hence Multi level inheritance achieved.

Output :

```

16:26:51.472 [main] INFO InheritanceImplementation.HybridInheritance.Child - Child Class Extending Mother class
16:26:51.472 [main] INFO InheritanceImplementation.HybridInheritance.Child - Child Looking like Mother
16:26:51.472 [main] INFO InheritanceImplementation.HybridInheritance.Mother - Mother Class Implementing GrandFather Interface...
16:26:51.472 [main] INFO InheritanceImplementation.HybridInheritance.Mother - Talking style from grand father
16:26:51.472 [main] INFO InheritanceImplementation.HybridInheritance.Mother - Mother Class Implementing GrandMother Interface...
16:26:51.472 [main] INFO InheritanceImplementation.HybridInheritance.Mother - Walking style from grand mother
16:26:51.472 [main] INFO InheritanceImplementation.HybridInheritance.Main - My grand mother age70
16:26:51.472 [main] INFO InheritanceImplementation.HybridInheritance.Main - My grand mother age80
16:26:51.472 [main] INFO InheritanceImplementation.HybridInheritance.Main - My mother age:40
16:26:51.472 [main] INFO InheritanceImplementation.HybridInheritance.Main - Hybrid Inheritance implemented successfully

```

Multiple Inheritance :

- Multiple Inheritance means extending features from more than one class.
- In java, multiple inheritance is not supported because of ambiguity problem.
- In java Multiple Inheritance can be achieved in to ways.
- A class can implements multiple interfaces.
- An interface can extends multiple interfaces.

Code in Main Class :

```

package InheritanceImplementation.MultipleInheritance;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

public class Main{

    private static final Logger logger = LogManager.getLogger(Main.class);

    public static void main(String[] args) {

        Rectangle r=new Rectangle();

        r.show();

        logger.info("Area of rectangle is:"+r.calculateArea(2,3));

        logger.info("Perimeter of rectangle is:"+r.calculatePerimeter(3,2));

        logger.info("Can't implemented Multiple Inheritance.");

    }

}

```

Explanation :

- Created a object for child class.
- Then called methods of that class and its parent class using created object.
- Logger class is used to log messages for specific application.
- The getLogger() method is used to get the specified Logger in this LogManager instance.
- The info () method of a Logger class is used to Log an INFO message.

Code in two parent classes and Child class :

```

public class Area {

    private static final Logger logger = LogManager.getLogger(Area.class);

    void show() {

        logger.info("This is the Area Class");

    }

    public int calculateArea(int a, int b) {

        return a*b;

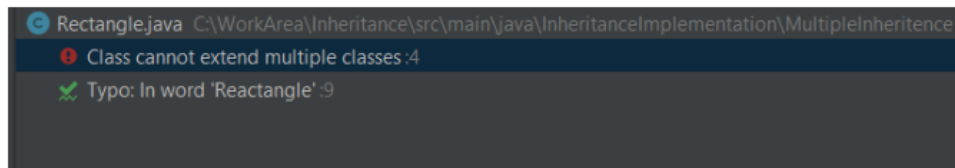
    }

}

```

```
    }  
}  
public class Perimeter {  
    private static final Logger logger = LogManager.getLogger(Perimeter.class);  
    void show() {  
        logger.info("This is the Perimeter class");  
    }  
  
    public int calculatePerimeter(int a, int b) {  
        return 2*(a+b);  
    }  
}  
public class Rectangle extends Area , Perimeter{  
    private static final Logger logger = LogManager.getLogger(Rectangle.class);  
    @Override  
    //implementing show() method which is in parent classes  
    public void show() {  
        logger.info("Calculating Area and Perimeter of Rectangle");  
    }  
}
```

Output :

A screenshot of an IDE's error console. The top bar shows the file path: 'Rectangle.java C:\WorkArea\Inheritance\src\main\java\InheritanceImplementation\MultipleInheritance'. Below this, there are two error messages. The first is a red error icon followed by the text 'Class cannot extend multiple classes :4'. The second is a green checkmark icon followed by the text 'Typo: In word 'Reactangle' :9'.

- Here it is showing that we cannot extend multiple class.
- Java does not support multiple inheritance.
- This means that a class cannot extend more than one class
- The reason why Java does not support multiple inheritance is that when there exist methods with same signature in both the super classes then compiler cannot determine which class method to be called.
- Because of ambiguity java doesn't support multiple inheritance.

User Story-5

5. Fetch data from table using JDBC, and convert data to specified file formats.

Description: Provide a database name as an input, load all the tables from each schema in the given database to file formats like CSV, TSV, JSON, Delimited file (| - pipe) into different folders with table name as file name with formatters.

Load data from String, Int, Long, Double, Date, Date Time from the source tables.

Input: Provide database name and fetch all the tables and load them to destination with file formats.

Acceptance Criteria: Log the output to the file along with input and output.

- Verify the count using the code along with data types matching at the destination files using the same code.

Load the tables with different values if required based on the use case

- Update Azure Board on daily basis.
- Get the use case verified by Technical Lead.
- Complete use case documentation.

CODE:

Main class:

```
package ConvertTables;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.stringtemplate.v4.ST;
```

```
import org.stringtemplate.v4.STGroup;
import org.stringtemplate.v4.STGroupFile;
import java.io.File;
import java.io.IOException;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Scanner;

public class Main {

    private static final Logger logger = LogManager.getLogger(Main.class);

    public static void main(String[] args) throws IOException, SQLException {

        Scanner sc=new Scanner(System.in);

        logger.info("Enter the database name:");

        String db=sc.nextLine();

        ConnectionClass cc = new ConnectionClass();

        Connection connection = cc.getConnection( );

        Statement stm = connection.createStatement();

        STGroup stGroup =

        new STGroupFile("src\\main\\java\\ConvertTables\\StringTemplates.stg");

        ST template1 = stGroup.getInstanceOf("templateForRetrive");

        template1.add("param1",db);

        String query1=template1.render();

        ResultSet rs = stm.executeQuery(query1);

        FormatsClass fc = new FormatsClass();

        while(rs.next()){

            // fc.fileWriter(connection,tablename);

            fc.csvFormat(connection, rs.getString(1),rs.getString(2));

            fc.tsvFormat(connection, rs.getString(1),rs.getString(2));
```

```

        fc.pipeFormat(connection, rs.getString(1),rs.getString(2));
        fc.jsonFormat(connection,rs.getString(1),rs.getString(2));
    }
    fc.countFiles();
}
}

```

StringTemplate.stg:

```

templateForRetrive(param1,param2) ::= <<
SELECT table_schema,table_name FROM INFORMATION_SCHEMA.TABLES where table_catalog='<param1>';
>>
templateForSelect(param1,param2) ::= <<
SELECT * from <param1>.<param2>;
>>

```

Explanation:

- **java.io.IOException** : is an exception which programmers use in the code to throw a failure in Input & Output operations.
- **java.sql.Connection**: is an exception which programmers use in the code to throw a failure in sql connection.
- **java.util.Scanner** : To use Scanner class for taking input from user.
- **java.sql.SQLException** : It provides info on database access error.
- **java.text.ParseException** : It is used to throw exception when date is unable to parse into any format.
- **java.util.regex.Matcher**: is used to search through a text for multiple occurrences of a regular expression
- **java.util.regex.Pattern**: It is used to define a pattern for the regex engine.
- The method getLogger() method belongs to LogManager class.
- Created a logger object which is used to log input and output to log file.
- The info() method of a Logger class is used to log an info message.
- Scanner class in Java is found in the java.util package.

- Java provides various ways to read input from the keyboard, the `java.util.Scanner` class is one of them.
- Used `Scanner` class to take input.
- `nextLine()` : It is used to get the input string from user.
- Created objects for classes.
- **Matcher**: It is used to search through a text for multiple occurrences of a regular expression
- **Pattern**: It is used to define a pattern for the regex engine.

Connection Class:

```

package ConvertTables;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;
import org.json.simple.parser.ParseException;

import java.io.FileReader;
import java.io.IOException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.Base64;

public class ConnectionClass {
    private static final Logger logger = LogManager.getLogger(ConnectionClass.class);
    //getConnection() method returns a connection object
    public Connection getConnection() throws IOException, SQLException {
        //Created a JSONParser object
        JSONParser jp = new JSONParser();
        //Creation of Connection Object
        Connection c = null;
        try {
            //Initialised JSONObject class object which is used to access values based on keys
            JSONObject jo = (JSONObject) jp.parse(new FileReader("src\\test\\config.json"));
            String url = (String)jo.get("Url");
            //username from config.json
            String username = (String) jo.get("Username");
            //Encrypted password from config.json
            String pwd = (String)jo.get("Password");

```

```

            //username from config.json
            String username = (String) jo.get("Username");
            //Encrypted password from config.json
            String pwd = (String)jo.get("Password");
            //Code to decrypt password
            Base64.Decoder d = Base64.getUrlDecoder();
            String password = new String(d.decode(pwd));
            try {
                //Initialization of connection object
                c = DriverManager.getConnection(url, username, password);
                logger.info("Connection established successfully...");
            }
            catch (SQLException e) {
                e.printStackTrace();
            }
        }
        catch (ParseException e) {
            e.printStackTrace();
        }
        return c;
    }
}

```

Explanation :

- **java.io.IOException** : is an exception which programmers use in the code to throw a failure in Input & Output operations.
- **java.sql.Connection**: is an exception which programmers use in the code to throw a failure in sql connection
- The method getLogger() method belongs to LogManager class.
- **java.util.Base64** :Has static methods for obtaining encoders and decoders for the Base64 encoding schema.
- Here user name and encrypted password are present in config.json file.
- JSONParser is used to parse json data.
- JSONObject is used to access values based on keys from json String.
- **Base64.Decoder** for decoding byte data using Base64 encoding scheme .
- The **getConnection()** method of DriverManager class is used to establish connection with the database.
- getConnection() method requires three parameters Connection url, Username, Password.
- Here url is "jdbc:postgresql://w3.training5.modak.com:5432/training"
 - "training" is database name.
 - "5432" is port number.
 - "w3.training5.modak.com" is the hostname on which postgresql is running.
- Username and password are taken from config.json.

Config.json File:

```
{
  "Username": "mt4020",
  "Password": "bXQ0MDIwQG0wMnkyMg",
  "url": "jdbc:postgresql://w3.training5.modak.com:5432/training"
}
```

Output:

```
"C:\Program Files\Java\jdk1.8.0_261\bin\java.exe" ...
19:25:37.211 [main] INFO ConvertTables.ConnectionClass - Connection established successfully...
```

Format Class :

Code to convert to .csv files :

```
package ConvertTables;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.postgresql.util.PSQLException;
import org.stringtemplate.v4.ST;
import org.stringtemplate.v4.STGroup;
import org.stringtemplate.v4.STGroupFile;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class FormatsClass {
    private static final Logger logger = LogManager.getLogger(ConnectionClass.class);
    public void csvFormat(Connection connection, String schemaName, String tableName)
throws SQLException, IOException {
        Statement stm = connection.createStatement();
```

```

try {
    STGroup stGroup =

new STGroupFile("src\\main\\java\\ConvertTables\\StringTemplates.stg");
    ST template1 = stGroup.getInstanceOf("templateForSelect");
    template1.add("param1",schemaName);
    template1.add("param2",tableName);
    String query1=template1.render();
    ResultSet rs = stm.executeQuery(query1);
    int columnCount = rs.getMetaData().getColumnCount();
    try {
        FileWriter fw =

        new FileWriter(src\\main\\resources\\userStories\\CSV\\" + tableName + ".csv");
        for (int i = 1; i < columnCount; i++) {
            fw.append(rs.getMetaData().getColumnName(i));
            fw.append(",");
        }
        fw.append(rs.getMetaData().getColumnName(columnCount));
        fw.append(System.getProperty("line.separator"));
        while (rs.next()) {
            for (int i = 1; i <= columnCount; i++) {
                if (i != columnCount) {
                    if (rs.getObject(i) != null) {
                        String data = rs.getObject(i).toString();
                        fw.append(data);
                        fw.append(",");
                    } else {
                        String data = "null";
                        fw.append(data);
                        fw.append(",");
                    }
                }
            }
        }
    }
}

```



```

        }
    } else {
        if (rs.getObject(i) != null) {
            String data = rs.getObject(i).toString();
            fw.append(data);
        } else {
            String data = "null";
            fw.append(data);
        }
    }
}
fw.append(System.getProperty("line.separator"));
}
fw.flush();
fw.close();
} catch (IOException ioe) {
    ioe.printStackTrace();
}
} catch (SQLException e) {
    System.out.println("permission denied");
}
}

```

Explanation:

- Created a stg file with name "StringTemplates.stg".
- Named template as templateForSelect with two parameters.
- Wrote query for to get data from the specified table.
- Then executed query using executeQuery().

- The `getMetaData()` method of `ResultSet` interface retrieves the `ResultSetMetaData` object of the current `ResultSet`.
- `MetaData` hold data about the tables and other information about the table.
- `getColumnName()` gives count of number of columns in the table.
- Wrote code to convert tables into CSV file format.
- Java `FileWriter` class is used to write character-oriented data to a file. It is character-oriented class which is used for file handling in java.

Code to convert to .tsv files :

`public void tsvFormat(Connection connection, String schemaName, String tableName) throws SQLException, IOException {`

```

    Statement stm = connection.createStatement();
    try {
        STGroup stGroup =
            new STGroupFile("src\\main\\java\\ConvertTables\\StringTemplates.stg");
        ST template1 = stGroup.getInstanceOf("templateForSelect");
        template1.add("param1",schemaName);
        template1.add("param2",tableName);
        String query1=template1.render();
        ResultSet rs = stm.executeQuery(query1);
        int columnCount = rs.getMetaData().getColumnCount();
        try {
            FileWriter fw =
                new FileWriter("src\\main\\resources\\userStories\\TSV\\" + tableName + ".tsv");
            for (int i = 1; i <= columnCount; i++) {
                fw.append(rs.getMetaData().getColumnName(i));
                fw.append("\t");
            }
            fw.append(System.getProperty("line.separator"));
        }
    }
}

```

```

while (rs.next()) {
    for (int i = 1; i <= columnCount; i++) {
        if (rs.getObject(i) != null) {
            String data = rs.getObject(i).toString();
            fw.append(data);
            fw.append("\t");
        } else {
            String data = "null";
            fw.append(data);
            fw.append("\t");
        }
    }
    fw.append(System.getProperty("line.separator"));
}
fw.flush();
fw.close();
} catch (IOException ioe) {
    ioe.printStackTrace();
}

} catch (SQLException e) {
    System.out.println("permission denied");
}
}

```

Explanation :

- Created a stg file with name "StringTemplates.stg".
- Named template as templateForSelect with two parameters.
- Wrote query for to get data from the specified table.
- Then executed query using executeQuery().

- The `getMetaData()` method of `ResultSet` interface retrieves the `ResultSetMetaData` object of the current `ResultSet`.
- `MetaData` hold data about the tables and other information about the table.
- `getColumnName()` gives count of number of columns in the table.
- Java `FileWriter` class is used to write character-oriented data to a file. It is character-oriented class which is used for file handling in java.
- Wrote code to convert tables into TSV file format.

Code to convert to .txt files :

```
public void pipeFormat(Connection connection, String schemaName, String tableName)
throws SQLException, IOException {
    Statement stm = connection.createStatement();
    try {
        STGroup stGroup =
            new STGroupFile("src\\main\\java\\ConvertTables\\StringTemplates.stg");
        ST template1 = stGroup.getInstanceOf("templateForSelect");
        template1.add("param1",schemaName);
        template1.add("param2",tableName);
        String query1=template1.render();
        ResultSet rs = stm.executeQuery(query1);
        int columnCount = rs.getMetaData().getColumnCount();
        try {
            FileWriter fw =
                new FileWriter("src\\main\\resources\\userStories\\PIPE\\" + tableName + ".txt");
            for (int i = 1; i < columnCount; i++) {
                fw.append(rs.getMetaData().getColumnName(i));
                fw.append("|");
            }
            fw.append(rs.getMetaData().getColumnName(columnCount));
        }
    }
}
```

```

fw.append(System.getProperty("line.separator"));
while (rs.next()) {
    for (int i = 1; i <= columnCount; i++) {
        if (i != columnCount) {
            if (rs.getObject(i) != null) {
                String data = rs.getObject(i).toString();
                fw.append(data);
                fw.append("|");
            } else {
                String data = "null";
                fw.append(data);
                fw.append("|");
            }
        } else {
            if (rs.getObject(i) != null) {
                String data = rs.getObject(i).toString();
                fw.append(data);
            } else {
                String data = "null";
                fw.append(data);
            }
        }
    }
    fw.append(System.getProperty("line.separator"));
}
fw.flush();
fw.close();
} catch (IOException ioe) {
    ioe.printStackTrace();
}

```

```

    }

    } catch (SQLException e) {
        System.out.println("permission denied");
    }
}

```

Explanation:

- Created a stg file with name "StringTemplates.stg".
- Named template as templateForSelect with two parameters.
- Wrote query for to get data from the specified table.
- Then executed query using executeQuery().
- The getMetaData() method of ResultSet interface retrieves the ResultSetMetaData object of the current ResultSet.
- MetaData hold data about the tables and other information about the table.
- getColumnNames() gives count of number of columns in the table

Code to convert to .json files :

```

public void jsonFormat(Connection connection, String schemaName, String tableName)
throws SQLException, IOException {
    Statement stm = connection.createStatement();
    try {
        STGroup stGroup = new
STGroupFile("C:\\WorkArea\\TableToFileFormats\\src\\main\\java\\ConvertTables\\StringTem
plates.stg");

        ST template1 = stGroup.getInstanceOf("templateForSelect");
        template1.add("param1",schemaName);
        template1.add("param2",tableName);
        String query1=template1.render();
        ResultSet rs = stm.executeQuery(query1);
    }
}

```

```

int columns = rs.getMetaData().getColumnCount();
try {
    FileWriter fw =

    new FileWriter("src\\main\\resources\\userStories\\JSON\\" + tableName + ".json");
    fw.append("[\n");
    fw.append("{ " + "\n");
    if (rs.next()) {
        for (int i = 1; i <= columns; i++) {
            fw.append("\\"" + rs.getMetaData().getColumnName(i) + "\"");
            fw.append(":");
            if (rs.getObject(i) != null) {
                String obj = rs.getObject(i).toString();
                fw.append("\\"" + obj + "\"");
                if (i != columns) fw.append(",");
            } else {
                String obj = "null";
                fw.append("\\"" + obj + "\"");
                if (i != columns) fw.append(",");
            }
        }
    }
}

while (rs.next()) {
    fw.append("\n" + '}');
    fw.append(', ' + "\n");
    fw.append("{ " + "\n");
    for (int i = 1; i <= columns; i++) {
        fw.append("\\"" + rs.getMetaData().getColumnName(i) + "\"");
        fw.append(":");
        if (rs.getObject(i) != null) {

```

```

        String obj = rs.getObject(i).toString();
        fw.append "\"" + obj + "\"";
        if (i != columns) fw.append(",");
    } else {
        String obj = "null";
        fw.append "\"" + obj + "\"";
        if (i != columns) fw.append(",");
    }
}
}
fw.append("\n"+"}");
fw.append("\n");
fw.append("]");
fw.flush();
fw.close();
}
catch(IOException e){
    e.printStackTrace();
}
}
}

```

Explanation:

- Created a stg file with name "StringTemplates.stg".
- Named template as templateForSelect with two parameters.
- Wrote query for to get data from the specified table.
- Then executed query using executeQuery().
- The getMetaData() method of ResultSet interface retrieves the ResultSetMetaData object of the current ResultSet.

- MetaData hold data about the tables and other information about the table.
- getColumnNames() gives count of number of columns in the table.
- Wrote code to convert tables into JSON file format.
- Java FileWriter class is used to write character-oriented data to a file. It is character-oriented class which is used for file handling in java.

Code to count number of files in each folder :

```
void countFiles(){
    File folder0 = new File("src\\main\\resources\\userStories\\CSV");
    File[] listOfFiles1 = folder0.listFiles();
    logger.info("No.of files in CSV folder are:"+ listOfFiles1.length);
    File folder1 = new File("src\\main\\resources\\userStories\\TSV");
    File[] listOfFiles2 = folder1.listFiles();
    logger.info("No.of files in TSV folder are:"+ listOfFiles2.length);
    File folder2 = new File("src\\main\\resources\\userStories\\PIPE");
    File[] listOfFiles3 = folder2.listFiles();
    logger.info("No.of files in PIPE folder are:"+ listOfFiles3.length);
    File folder3 = new File("src\\main\\resources\\userStories\\JSON");
    File[] listOfFiles4 = folder3.listFiles();
    logger.info("No.of files in JSON folder are:"+ listOfFiles4.length);
}
}
```

Output:

```
| 2022-05-04 12:48:10.881 [main] INFO ConvertTables.Main - Enter the database name:
| 2022-05-04 12:48:14.643 [main] INFO ConvertTables.ConnectionClass - Connection established successfully...
| 2022-05-04 12:48:54.195 [main] INFO ConvertTables.ConnectionClass - No.of files in CSV folder are:357
| 2022-05-04 12:48:54.197 [main] INFO ConvertTables.ConnectionClass - No.of files in TSV folder are:357
| 2022-05-04 12:48:54.200 [main] INFO ConvertTables.ConnectionClass - No.of files in PIPE folder are:357
| 2022-05-04 12:48:54.202 [main] INFO ConvertTables.ConnectionClass - No.of files in JSON folder are:357
|
```

User Story – 6

6. Create a custom ArrayList Collection

Description: Create a new class CustomArrayList by extending List Interface and create implement methods with a similar logic and return the required output in every method.

Acceptance Criteria:

- When list is implemented.
- Understood concepts of return types, methods

CODE :

Main class :

```
package CustomArray;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.example.CustomArrayList;
import java.util.ArrayList;
import java.util.Arrays;

public class Main {

    private static final Logger logger = (Logger) LogManager.getLogger(Main.class);

    public static void main(String[] args) {

        CustomArrayList1 cal1=new CustomArrayList1();

        //Adding elements
        cal1.add(1);
        cal1.add(2);
        cal1.add("sanju reddy");
        logger.info("calling size() method "+cal1.size());
        logger.info("printing Elements:");
        for(int i=0;i<cal1.size();i++){
            logger.info(cal1.get(i));
```

```

    }
    logger.info("calling contains() method "+cal1.contains(7));
    logger.info("calling isEmpty() method "+cal1.isEmpty());
    logger.info("calling get() method "+cal1.get(1));
    logger.info("calling indexOf() method "+cal1.indexOf(1));
    logger.info("calling lastIndexOf() method "+cal1.lastIndexOf(2));
}
}

```

Explanation :

- Created object for CustomArrayList and called methods.
- size() returns the number of elements present in it.
- contains() return true if it has the given element.
- get() will return element at specific index.
- indexOf() will return index of specific element.
- isEmpty() will return true if it is empty

CustomArrayList class:

- Implementation of methods
- Initially initialised some variables.

```

int capacity=10;
Object[] arr=new Object[capacity];
int c=0;

```

Implementation of size() method :

- Counting number of elements by using for loop.
- Then returned the count.

```

public int size() {
    // "Implementation of size method"
}

```

```
    return this.c;
}
```

Implementation of isEmpty() method :

- If the size() is zero then it is empty.
- If it is empty it return true.
- If it is not empty it return false.

```
public boolean isEmpty() {
    //"Implementation of isEmpty method"
    if (arr.length == 0) {
        return true;
    }
    return false;
}
```

Implementation of contains() method :

- It searches for element by going to each and every element.
- If element got found it returns true.

```
public boolean contains(Object o) { //logger.info("Implementation of contains method");
    for (int i = 0; i < arr.length; i++) {
        if (arr[i] == o) {
            return true;
        }
    }
    return false;
}
```

Implementation of add() method :

- Here is the code to add element.

```

//logger.info("Implementation of add method");
if(c==capacity){
    Object[] arr1=new Object[c++];
    for(int i=0;i<arr.length;i++){
        arr1[i]=arr[i];
    }
    this.arr=arr1;
}
arr[c]=o;
c++;
return true;
}

```

Implementation of indexOf() method :

- It searches for element by going to each and every element.
- If element got found it returns index.

```

public int indexOf(Object o) {
    // logger.info("Implementation of indexOf method");
    for (int i = 0; i < arr.length; i++) {
        if (arr[i]==o) {
            return i;
        }
    }
    return -1;
}

```

Implementation of lastIndexOf() method :

- It searches for element by going to each and every element.
- If element got found it continues and return the last index.

```

public int lastIndexOf(Object o) {
    // logger.info("Implementation of lastIndexOf method");
    int k = -1;
    for (int i = 0; i < arr.length; i++) {
        if (arr[i] == o) {
            k = i;
        }
    }
    return k;
}

```

Output:

```

"C:\Program Files\Java\jdk1.8.0_261\bin\java.exe" ...
19:26:19.516 [main] INFO CustomArray.Main - calling size() method 3
19:26:19.516 [main] INFO CustomArray.Main - printing Elements:
19:26:19.516 [main] INFO CustomArray.Main - 1
19:26:19.516 [main] INFO CustomArray.Main - 2
19:26:19.516 [main] INFO CustomArray.Main - sanju reddy
19:26:19.516 [main] INFO CustomArray.Main - calling contains() method false
19:26:19.516 [main] INFO CustomArray.Main - calling isEmpty() method false
19:26:19.516 [main] INFO CustomArray.Main - calling get() method 2
19:26:19.516 [main] INFO CustomArray.Main - calling indexOf() method 0
19:26:19.516 [main] INFO CustomArray.Main - calling lastIndexOf() method 1

```

User Story – 7

7. Insert data from specified file formats based on pattern from given folder path as input to table in JDBC

Description: Read data from specified file formats (TSV, CSV, JSON, delimited (| pipe) based on a pattern from given folder path as input to tables in JDBC

- Provide Folder path as input which contains data to be loaded.
- Read all files from the folder based on pattern provided by end-user.
- Use regular expression to find file format
- Read the data from the file
- Perform required transformations
- Connect to Postgres SQL table using JDBC
- Get the database name as input and maintain the table name to be same as file name (Standard names).
- Create database, table
- Insert/Update data from each file in batches of about 10000 records at once
- Bulk Insert records into JDBC for each collection
- Insert all files into the PostgreSQL database

Input: Folder path and database name

Output: <PostgreSQL Database>

Acceptance Criteria:

- Log the output to the file along with input and output
- Use files with different file formats as required based on the use case
- Use Encrypted passwords for both connections

- Create files as required if needed. Make sure to attach files in the user story.
- Create each table for a file in specified folder path
- Update Azure Board on daily basis.
- Get the use case verified by Technical Lead.
- Complete use case documentation.

CODE :

Main Class :

```
package FileToTableConversion;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.json.simple.parser.ParseException;
import java.io.*;
import java.sql.Connection;
import java.sql.SQLException;
import java.util.Scanner;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class Main {
    private static final Logger logger = LogManager.getLogger(Main.class);

    public static void main(String[] args) throws IOException, SQLException, ParseException {
        Scanner sc = new Scanner(System.in);
        logger.info("Enter database name:");
        String db=sc.nextLine();
        logger.info("Enter the folder path :");
        String fpath=sc.nextLine();
        ConnectionClass con = new ConnectionClass();
        Connection cc = con.getConnection();
        //Creating a new file instance
        File folder = new File(fpath);
        //Object for JsonClass
        JsonClass jc=new JsonClass(cc);
```



```

//Object for JsonClass
JsonClass jc=new JsonClass(cc);
//Object for TsvClass
TsvClass tc=new TsvClass(cc);
//Object for PipeClass
PipeClass pc=new PipeClass(cc);
//Object for CsvClass
CsvClass ccl=new CsvClass(cc);
//here method is used to return all the files in the form of an
File[] listOfFiles = folder.listFiles();
Pattern pattern = Pattern.compile("PIPE");
Matcher matcher = pattern.matcher(fpath);
Pattern pattern1 = Pattern.compile("CSV");
Matcher matcher1 = pattern1.matcher(fpath);
Pattern pattern2 = Pattern.compile("TSV");
Matcher matcher2 = pattern2.matcher(fpath);
Pattern pattern3 = Pattern.compile("JSON");
Matcher matcher3 = pattern3.matcher(fpath);
if(matcher.find()){
    for(int i=0;i<10;i++){
        pc.insertCreate(listOfFiles[i]);
    }
}
if(matcher1.find()){
    for(int i=0;i<10;i++){
        ccl.insertCreate(listOfFiles[i]);
    }
}

```

```

}
if(matcher1.find()){
    for(int i=0;i<10;i++){
        ccl.insertCreate(listOfFiles[i]);
    }
}
if(matcher2.find()){
    for(int i=0;i<10;i++){
        tc.insertCreate(listOfFiles[i]);
    }
}
if(matcher3.find()){
    for(int i=0;i<10;i++){
        jc.insertCreate(listOfFiles[i]);
    }
}
}
}

```

Explanation :

- **java.io.IOException** : is an exception which programmers use in the code to throw a failure in Input & Output operations.
- **java.sql.Connection**: is an exception which programmers use in the code to throw a failure in sql connection.
- **java.util.Scanner** : To use Scanner class for taking input from user.
- **java.sql.SQLException** : It provides info on database access error.
- **java.text.ParseException** : It is used to throw exception when date is unable to parse into any format.
- **java.util.regex.Matcher**: is used to search through a text for multiple occurrences of a regular expression
- **java.util.regex.Pattern**: It is used to define a pattern for the regex engine.
- The method getLogger() method belongs to LogManager class.
- Created a logger object which is used to log input and output to log file.
- The info() method of a Logger class is used to log an info message.
- Scanner class in Java is found in the java.util package.
- Java provides various ways to read input from the keyboard, the java.util.Scanner class is one of them.
- Used Scanner class to take input.
- `nextLine()` : It is used to get the input string from user.
- Created objects for classes.
- **Matcher**: It is used to search through a text for multiple occurrences of a regular expression
- **Pattern**: It is used to define a pattern for the regex engine.
- If path has CSV then it called insertData() method of Csv Class.
- If path has TSV then it called insertData() method of Tsv Class.
- If path has PIPE then it called insertData() method of Pipe Class.
- If path has JSON then it called insertData() method of Json Class.

Output:

```
"C:\Program Files\Java\jdk1.8.0_261\bin\java.exe" ...  
00:28:24.883 [main] INFO FileToTableConversion.Main - Enter database name:  
training  
00:28:29.163 [main] INFO FileToTableConversion.Main - Enter the folder path :  
src/main/resources/TSV|
```

Connection Class :

```
package FileToTableConversion;  
  
import org.apache.logging.log4j.LogManager;  
import org.apache.logging.log4j.Logger;  
import org.json.simple.JSONObject;  
import org.json.simple.parser.JSONParser;  
import org.json.simple.parser.ParseException;  
import java.io.FileReader;  
import java.io.IOException;  
import java.sql.Connection;  
import java.sql.DriverManager;  
import java.sql.SQLException;  
import java.util.Base64;  
  
public class ConnectionClass {  
    private static final Logger logger = LogManager.getLogger(ConnectionClass.class);  
    //getConnection() method returns a connection object  
    public Connection getConnection() throws IOException, SQLException {  
        //Created a JSONParser object  
        JSONParser jp = new JSONParser();  
        //Creation of Connection Object  
        Connection c = null;  
        try {  
  
            JSONObject jo = (JSONObject) jp.parse(new FileReader("src/test/config.json"));  
            //username from config.json
```

```

String username =(String) jo.get("Username");
//Encrypted password from config.json
String pwd = (String)jo.get("Password");
//Code to decrypt password
Base64.Decoder d = Base64.getUrlDecoder();
String password = new String(d.decode(pwd));
String Url=(String) jo.get("url");
try {
    //Initialization of connection object
    c = DriverManager.getConnection(Url, username, password);
    logger.info("Connection established successfully...");
}
catch (SQLException e) {
    e.printStackTrace();
}
}
catch (ParseException e) {
    e.printStackTrace();
}
return c;
}
}

```

Explanation :

- **java.io.IOException** : is an exception which programmers use in the code to throw a failure in Input & Output operations.
- **java.sql.Connection:** is an exception which programmers use in the code to throw a failure in sql connection
- The method *getLogger()* method belongs to *LogManager* class.

- **java.util.Base64** :Has static methods for obtaining encoders and decoders for the Base64 encoding schema.
- Here user name and encrypted password are present in config.json file.
- JSONParser is used to parse json data.
- JSONObject is used to access values based on keys from json String.
- **Base64.Decoder** for decoding byte data using Base64 encoding scheme .
- The **getConnection()** method of DriverManager class is used to establish connection with the database.
- getConnection() method requires three parameters Connection url, Username, Password.
- Here url is "jdbc:postgresql://w3.training5.modak.com:5432/training"
 - "training" is database name.
 - "5432" is port number.
 - "w3.training5.modak.com" is the hostname on which postgresql is running.
- Username and password are taken from config.json.

Config.json File:

```
{  
  "Username": "mt4020",  
  "Password": "bXQ0MDIwQ60wMnkyMg",  
  "url": "jdbc:postgresql://w3.training5.modak.com:5432/training"  
}
```

Csv Class:

```

package FileToTableConversion;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;
import org.postgresql.util.PSQLException;
import org.stringtemplate.v4.ST;
import org.stringtemplate.v4.STGroup;
import org.stringtemplate.v4.STGroupFile;

import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Scanner;

public class CsvClass {
    private static final Logger logger = LogManager.getLogger(Main.class);
    Connection connection=null;
    int count=0;
    int batchSize=10000;
    CsvClass(Connection cc) throws SQLException {
        connection=cc;
    }
    void insertCreate(File f1) throws IOException, SQLException {
        Scanner dataReader5 = new Scanner(f1);
        String str = dataReader5.nextLine();
        String[] arr = str.split(" ");
        String fileName = f1.getName();

```

```

        String[] arr = str.split(" ");
        String fileName = f1.getName();
        fileName = fileName.replace(" .csv", "");
        STGroup stGroup = new STGroupFile("src/main/java/FileToTableConversion/StringTemplates.stg");
        //dropTable(fileName);
        String cstr = "";
        for (int i = 0; i < arr.length; i++) {
            if (i != arr.length - 1) {
                cstr = cstr + arr[i] + " varchar(100),";
            } else {
                cstr = cstr + arr[i] + " varchar(100)";
            }
        }
        createTable(cstr, fileName);
        Statement stm=connection.createStatement();
        while (dataReader5.hasNextLine()) {
            String str1 = dataReader5.nextLine();
            String[] iarr = str1.split(" ");
            String istr = "";
            for (int i = 0; i < arr.length; i++) {
                if (i != iarr.length - 1) {
                    istr = istr + " " + iarr[i] + " " + ",";
                } else {
                    istr = istr + " " + iarr[i] + " ";
                }
            }
            ST template1 = stGroup.getInstanceOf("templateForInsertionCsv");
            template1.add("param1", fileName);
            template1.add("param2", istr);
            String query1 = template1.render();
            count++;

```

```

void createTable(String cstr, String tableName) throws IOException, SQLException {
    try {
        Statement stm = connection.createStatement();
        STGroup stGroup = new STGroupFile( fileName: "src/main/java/FileToTableConversion/StringTemplates.stg");
        // Pick the correct template
        ST template1 = stGroup.getInstanceOf( name: "templateForCreationCsv");
        template1.add( name: "param1", tableName);
        template1.add( name: "param2", cstr);
        String query1 = template1.render();
        logger.info(query1);
        stm.executeUpdate(query1);
    } catch (SQLException e) {
        e.printStackTrace();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

```

```

void dropTable(String tableName) throws SQLException, IOException {
    try {
        Statement stm = connection.createStatement();
        STGroup stGroup = new STGroupFile( fileName: "src/main/java/FileToTableConversion/StringTemplates.stg");
        // Pick the correct template
        ST template1 = stGroup.getInstanceOf( name: "templateForDropCsv");
        template1.add( name: "param1", tableName);
        String query1 = template1.render();
        logger.info(query1);
        stm.executeUpdate(query1);
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

```

Explanation :

- Here initialized connection object.
- Insertion of records with batches of size 10000.
- Used replace() function to replace ".csv" with "".
- Here executing insert queries when count reaches to 10000
- Remaining insert queries are executed at last.
- If we execute by batches execution time reduces.
- Here create Operation is performed when called createTable().
- The statement interface is used to create SQL basic statements in Java it provides methods to execute queries with the database.

- Used String Templates to write query.
- Used executeUpdate() to execute create statement.
- Here wrote dropTable() method to drop tables for executing code next time.
- Used String Templates to write query.

StringTemplate.stg :

```
templateForCreationCsv(param1,param2) ::= <<
Create table <param1>_csv(<param2>);
>>

templateForInsertionCsv(param1,param2) ::= <<
insert into <param1>_csv values(<param2>);
>>

templateForDropCsv(param1) ::= <<
drop table <param1>_csv;
>>
```

Output :

```
"C:\Program Files\Java\jdk1.8.0_261\bin\java.exe" ...
11:46:17.495 [main] INFO FileToTableConversion.Main - Enter database name:
abc@abc
11:46:20.805 [main] INFO FileToTableConversion.Main - Enter the folder path :
C:\Users\abc\workspace\CSV
11:46:22.477 [main] INFO FileToTableConversion.ConnectionClass - Connection established successfully...
11:46:22.589 [main] INFO FileToTableConversion.Main - Create table employee11_csv(id varchar(100),fname varchar(100),sname varchar(100),lname varchar(100),salary varchar(100));
11:46:22.734 [main] INFO FileToTableConversion.Main - Create table employeeedetails1_csv(empid varchar(100),fullname varchar(100),managerid varchar(100),dateofjoining varchar(100));
11:46:22.858 [main] INFO FileToTableConversion.Main - Create table empp_csv(eid varchar(100),ename varchar(100),sal varchar(100),doj varchar(100));
11:46:22.988 [main] INFO FileToTableConversion.Main - Create table mt4020tsv_us7_1_csv(voter_id varchar(100),house_no varchar(100),street_name varchar(100),landmark varchar(100));
11:46:23.096 [main] INFO FileToTableConversion.Main - Create table mt4020tsv_us7_3_csv(grantee varchar(100),role_name varchar(100),is_granttable varchar(100));
11:46:23.202 [main] INFO FileToTableConversion.Main - Create table mt4020tsv_us7_7_csv(character_set_catalog varchar(100),character_set_schema varchar(100),character_set_name varchar(100));
11:46:23.313 [main] INFO FileToTableConversion.Main - Create table mt4020tsv_us7_8_csv(constraint_catalog varchar(100),constraint_schema varchar(100),constraint_name varchar(100));
11:46:23.428 [main] INFO FileToTableConversion.Main - Create table mt4020_us7_0_csv(table_catalog varchar(100),table_schema varchar(100),table_name varchar(100),constraint_catalog varchar(100));
11:46:23.572 [main] INFO FileToTableConversion.Main - Create table mt4020_userstory_1_csv(id varchar(100),fname varchar(100),lname varchar(100),sal varchar(100),time_before_mod varchar(100));
11:46:23.705 [main] INFO FileToTableConversion.Main - Create table pg_extension_csv(extname varchar(100),extowner varchar(100),extnamespace varchar(100),extrelocatable varchar(100));
Process finished with exit code 0
```

- Created tables in Database

	ABC table_name
1	employee11_csv
2	employeeedetails1_csv
3	empp_csv
4	mt4020tsv_us7_1_csv
5	mt4020tsv_us7_3_csv
6	mt4020tsv_us7_7_csv
7	mt4020tsv_us7_8_csv
8	mt4020_us7_0_csv
9	mt4020_userstory_1_csv
10	pg_extension_csv

Tsv Class :

```
package FileToTableConversion;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.postgresql.util.PSQLException;
import org.stringtemplate.v4.ST;
import org.stringtemplate.v4.STGroup;
import org.stringtemplate.v4.STGroupFile;

import java.io.File;
import java.io.IOException;
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Scanner;

public class TsvClass {
    private static final Logger logger = LogManager.getLogger(Main.class);
    Connection connection=null;
    Statement stm=null;
    int count=0;
    int batchSize=10000;
    TsvClass(Connection cc) throws SQLException {
        connection=cc;
    }
    void insertCreate(File f1) throws IOException, SQLException {
        Scanner dataReader5 = new Scanner(f1);
        String str = dataReader5.nextLine();
```

```

String[] arr=str.split("\t");

String fileName=f1.getName();
fileName=fileName.replace(".tsv","");
// dropTable(fileName);

STGroup stGroup = new
STGroupFile("src/main/java/FileToTableConversion/StringTemplates.stg");
String cstr="";
for(int i=0;i<arr.length;i++) {
    if (i != arr.length - 1) {
        cstr = cstr + arr[i] + " varchar(100),";
    } else {
        cstr = cstr + arr[i] + " varchar(100)";
    }
}
createTable(cstr,fileName);

Statement stm= connection.createStatement();
while (dataReader5.hasNextLine()) {
    String str1= dataReader5.nextLine();
    String[] iarr = str1.split("\t");
    String istr = "";
    for (int i = 0; i < iarr.length; i++) {
        if (i != iarr.length - 1) {
            istr = istr + "" + iarr[i] + "" + ",";
        } else {
            istr = istr + "" + iarr[i] + "";
        }
    }
    ST template1 = stGroup.getInstanceOf("templateForInsertionTsv");
    template1.add("param1", fileName);
    template1.add("param2", istr);
    String query1 = template1.render();
    count++;
}

```

```

        stm.addBatch(query1);
        if (count % batchSize==0) {
            stm.executeBatch();
        }
    }
    stm.executeBatch();
}

```

```

void createTable(String cstr,String tableName) throws IOException, SQLException {
    try{
        Statement stm= connection.createStatement();
        STGroup stGroup = new STGroupFile("src/main/java/FileToTableConversion/StringTemplates.stg");
        // Pick the correct template
        ST template1 = stGroup.getInstanceOf("templateForCreationTsv");
        template1.add("param1",tableName);
        template1.add("param2",cstr);
        String query1 = template1.render();
        stm.executeUpdate(query1);
        logger.info(query1);
    }
    catch(PSQLException e){
        e.printStackTrace();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

void dropTable(String tableName) throws SQLException, IOException {
    try {
        Statement stm= connection.createStatement();

```

```

        STGroup          stGroup          =          new
STGroupFile("src/main/java/FileToTableConversion/StringTemplates.stg");
    // Pick the correct template
    ST template1 = stGroup.getInstanceOf("templateForDropTsv");
    template1.add("param1",tableName);
    String query1 = template1.render();
    logger.info(query1);
    stm.executeUpdate(query1);
}
catch(PSQLException e){
    e.printStackTrace();
}
}
}

```

Explanation :

- Here initialized connection object.
- Insertion of records with batches of size 10000.
- Used replace() function to replace ".txt" with "".
- Here executing insert queries when count reaches to 10000
- Remaining insert queries are executed at last.
- If we execute by batches execution time reduces.
- Here create Operation is performed when called createTable().
- The statement interface is used to create SQL basic statements in Java it provides methods to execute queries with the database.
- Used String Templates to write query.
- Used executeUpdate() to execute create statement.

StringTemplates.stg:

```
templateForCreationTsv(param1,param2) ::= <<
Create table <param1>_tsv(<param2>);
>>

templateForInsertionTsv(param1,param2) ::= <<
insert into <param1>_tsv values(<param2>);
>>

templateForDropTsv(param1) ::= <<
drop table <param1>_tsv;
>>
```

Output:

```
"2022-05-05 16:18:41.767 [main] INFO FileToTableConversion.Main - drop table customer_dataset5_tsv;
"2022-05-05 16:19:12.357 [main] INFO FileToTableConversion.Main - Enter database name:
"2022-05-05 16:19:21.867 [main] INFO FileToTableConversion.Main - Enter the folder path :
"2022-05-05 16:19:23.809 [main] INFO FileToTableConversion.ConnectionClass - Connection established successfully...
"2022-05-05 16:19:23.933 [main] INFO FileToTableConversion.Main - Create table address_lookup_mt4020_tsv(voter_id varchar(100),house_no varchar(100),street_name varchar(100));
"2022-05-05 16:19:23.975 [main] INFO FileToTableConversion.Main - Create table applicable_roles_tsv(grantee varchar(100),role_name varchar(100),is_grantable varchar(100));
"2022-05-05 16:19:23.975 [main] INFO FileToTableConversion.Main - Create table attributes_tsv(udt_catalog varchar(100),udt_schema varchar(100),udt_name varchar(100),attribut
"2022-05-05 16:19:23.988 [main] INFO FileToTableConversion.Main - Create table collations_tsv(collation_catalog varchar(100),collation_schema varchar(100),collation_name var
"2022-05-05 16:19:24.210 [main] INFO FileToTableConversion.Main - Create table columns_tsv(table_catalog varchar(100),table_schema varchar(100),table_name varchar(100),colu
"2022-05-05 16:19:25.023 [main] INFO FileToTableConversion.Main - Create table column_privileges_tsv(grantor varchar(100),grantee varchar(100),table_catalog varchar(100),tab
"2022-05-05 16:19:25.786 [main] INFO FileToTableConversion.Main - Create table constraint_table_usage_tsv(table_catalog varchar(100),table_schema varchar(100),table_name var
"2022-05-05 16:19:25.748 [main] INFO FileToTableConversion.Main - Create table customer_dataset1_tsv(rn varchar(100),house_no___flat_no varchar(100),city_district varchar(10
"2022-05-05 16:19:27.899 [main] INFO FileToTableConversion.Main - Create table customer_dataset3_tsv(house_no___flat_no varchar(100),city_district varchar(100),state_ut vard
"2022-05-05 16:19:28.516 [main] INFO FileToTableConversion.Main - Create table customer_dataset5_tsv(house_no___flat_no varchar(100),city_district varchar(100),state_ut vard
```

- Tables in database :

SELECT table_name FROM INFO	
Grid	ABC table_name
1	attributes_tsv
2	collations_tsv
3	columns_tsv
4	address_lookup_mt4020.
5	applicable_roles_tsv
6	column_privileges_tsv
7	constraint_table_usage_t
8	customer_dataset1_tsv
9	customer_dataset3_tsv
10	customer_dataset5_tsv

Pipe Class:

```
package FileToTableConversion;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.postgresql.util.PSQLException;
import org.stringtemplate.v4.ST;
import org.stringtemplate.v4.STGroup;
import org.stringtemplate.v4.STGroupFile;

import java.io.File;
import java.io.IOException;
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Scanner;

public class PipeClass {
    private static final Logger logger = LogManager.getLogger(Main.class);
    Connection connection=null;
    int count=0;
    int batchSize=10000;
    PipeClass(Connection cc) throws SQLException {
        connection=cc;
    }
    void insertCreate(File f1) throws IOException, SQLException {
        STGroup stGroup =
```

```

new STGroupFile("src/main/java/FileToTableConversion/StringTemplates.stg");
Scanner dataReader5 = new Scanner(f1);
String str = dataReader5.nextLine();
String[] arr = str.split("\\\\|");
String cstr = "";
String fileName = f1.getName();
fileName = fileName.replace(".txt", "");
//dropTable(fileName);
for (int i = 0; i < arr.length; i++) {
    if (i != arr.length - 1) {
        cstr = cstr + arr[i] + " varchar(100),";
    } else {
        cstr = cstr + arr[i] + " varchar(100)";
    }
}
createTable(cstr, fileName);
Statement stm = connection.createStatement();
while (dataReader5.hasNextLine()) {
    String str1 = dataReader5.nextLine();
    String[] iarr = str1.split("\\\\|");
    String istr = "";
    for (int i = 0; i < iarr.length; i++) {
        if (i != iarr.length - 1) {
            istr = istr + "'" + iarr[i] + "'" + ",";
        } else {
            istr = istr + "'" + iarr[i] + "'";
        }
    }
}

```

```

        ST template1 = stGroup.getInstanceOf("templateForInsertionPipe");
        template1.add("param1", fileName);
        template1.add("param2", istr);
        String query1 = template1.render();
        count++;
        stm.addBatch(query1);
        if (count % batchSize==0) {
            stm.executeBatch();
        }
    }
    stm.executeBatch();
}

void createTable(String cstr,String tableName) throws IOException, SQLException {
    try{
        Statement stm= connection.createStatement();
        STGroup stGroup =

new STGroupFile("src/main/java/FileToTableConversion/StringTemplates.stg");
        // Pick the correct template
        ST template1 = stGroup.getInstanceOf("templateForCreationPipe");
        template1.add("param1",tableName);
        template1.add("param2",cstr);
        String query1 = template1.render();
        stm.executeUpdate(query1);
        logger.info(query1);
    }
    catch(PSQLException e){
        e.printStackTrace();
    } catch (SQLException e) {

```



```

        e.printStackTrace();
    }
}

void dropTable(String tableName) throws SQLException, IOException {
    try {
        Statement stm = connection.createStatement();
        STGroup stGroup =

        new STGroupFile("src/main/java/FileToTableConversion/StringTemplates.stg");
        // Pick the correct template
        ST template1 = stGroup.getInstanceOf("templateForDropPipe");
        template1.add("param1", tableName);
        String query1 = template1.render();
        logger.info(query1);
        stm.executeUpdate(query1);
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
}

```

Explanation :

- Here initialized connection object.
- Insertion of records with batches of size 10000.
- Used replace() function to replace ".txt" with "".
- Here executing insert queries when count reaches to 10000
- Remaining insert queries are executed at last.
- If we execute by batches execution time reduces.
- Here create Operation is performed when called createTable().

- The statement interface is used to create SQL basic statements in Java it provides methods to execute queries with the database.
- Used String Templates to write query.
- Used executeUpdate() to execute create statement.
- Here wrote dropTable() method to drop tables for executing code next time.
- Used String Templates to write query.

StringTemplates.stg:

```
templateForCreationPipe(param1,param2) ::= <<
Create table <param1>_txt(<param2>);
>>
templateForInsertionPipe(param1,param2) ::= <<
insert into <param1>_txt values(<param2>);
>>
templateForDropPipe(param1) ::= <<
drop table <param1>_txt;
>>
```

Output:

```
"2022-05-05 16:03:52.401 [main] INFO FileToTableConversion.Main - Enter database name:
"2022-05-05 16:03:55.471 [main] INFO FileToTableConversion.Main - Enter the folder path :
"2022-05-05 16:04:07.319 [main] INFO FileToTableConversion.ConnectionClass - Connection established successfully...
"2022-05-05 16:04:07.416 [main] INFO FileToTableConversion.Main - Create table character_sets_txt(character_set_catalog varchar(100),character_set_schema varchar(100),charac
"2022-05-05 16:04:07.440 [main] INFO FileToTableConversion.Main - Create table check_constraints_txt(constraint_catalog varchar(100),constraint_schema varchar(100),constrain
"2022-05-05 16:04:07.462 [main] INFO FileToTableConversion.Main - Create table collations_txt(collation_catalog varchar(100),collation_schema varchar(100),collation_name var
"2022-05-05 16:04:07.627 [main] INFO FileToTableConversion.Main - Create table columns_txt(table_catalog varchar(100),table_schema varchar(100),table_name varchar(100),colum
"2022-05-05 16:04:08.541 [main] INFO FileToTableConversion.Main - Create table column_domain_usage_txt(domain_catalog varchar(100),domain_schema varchar(100),domain_name var
"2022-05-05 16:04:08.563 [main] INFO FileToTableConversion.Main - Create table column_options_txt(table_catalog varchar(100),table_schema varchar(100),table_name varchar(100)
"2022-05-05 16:04:08.580 [main] INFO FileToTableConversion.Main - Create table constraint_column_usage_txt(table_catalog varchar(100),table_schema varchar(100),table_name va
"2022-05-05 16:04:08.610 [main] INFO FileToTableConversion.Main - Create table customer_dataset1_txt(rn varchar(100),house_no___flat_no varchar(100),city_district varchar(10
"2022-05-05 16:04:10.053 [main] INFO FileToTableConversion.Main - Create table customer_dataset3_txt(house_no___flat_no varchar(100),city_district varchar(100),state_ut vard
"2022-05-05 16:04:11.333 [main] INFO FileToTableConversion.Main - Create table customer_dataset5_txt(house_no___flat_no varchar(100),city_district varchar(100),state_ut vard
```

- Tables in database :

SELECT table_name FROM INFOR

	ABC table_name
1	character_sets_txt
2	check_constraints_txt
3	collations_txt
4	columns_txt
5	column_domain_usage_t
6	column_options_txt
7	constraint_column_usage
8	customer_dataset1_txt
9	customer_dataset3_txt
10	customer_dataset5_txt

Json Class:

```
package FileToTableConversion;

import com.google.gson.Gson;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.json.simple.JSONArray;
import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;
import org.json.simple.parser.ParseException;
import org.postgresql.util.PSQLException;
import org.stringtemplate.v4.ST;
import org.stringtemplate.v4.STGroup;
import org.stringtemplate.v4.STGroupFile;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.sql.Connection;
import java.sql.SQLException;
```

```

import java.sql.Statement;
import java.util.Map;
import java.util.Scanner;
public class JsonClass {
    private static final Logger logger = LogManager.getLogger(Main.class);
    Connection connection=null;
    Statement stm=null;
    int count=0;
    int batchSize=10000;
    JsonClass(Connection cc) throws SQLException {
        connection=cc;
    }
    void insertCreate(File f1) throws IOException, SQLException, ParseException {
        STGroup stGroup =

        new STGroupFile("src/main/java/FileToTableConversion/StringTemplates.stg");
        JSONParser parser = new JSONParser();
        Object obj = parser.parse(new FileReader(f1));
        String fileName = f1.getName();
        fileName = fileName.replace(".json", "");
        //dropTable(fileName);
        JSONArray jsonArray = (JSONArray) obj;
        logger.info(jsonArray.size());
        try {
            Statement stm = connection.createStatement();
            for (int i = 0; i < jsonArray.size(); i++) {
                String fields = "";
                String rows = "";
                JSONObject jsonObject = (JSONObject) jsonArray.get(i);

```

```

String mp = jsonObject.toJSONString();
Map<String, Object> map = new Gson().fromJson(mp, Map.class);
// print map keys and values
if (i == 0) {
    for (Map.Entry<String, Object> entry : map.entrySet()) {
        fields = fields + entry.getKey() + " varchar(100),";
    }
    fields = fields.substring(0, fields.length() - 1);
    createTable(fields, fileName);

}
for (Map.Entry<String, Object> entry : map.entrySet()) {
    rows = rows + "" + entry.getValue() + ",";
}
rows = rows.substring(0, rows.length() - 1);
ST template1 = stGroup.getInstanceOf("templateForInsertionJson");
template1.add("param1", fileName);
template1.add("param2", rows);
String query1 = template1.render();
count++;
stm.addBatch(query1);
if (count % batchSize == 0) {
    stm.executeBatch();
}
}
stm.executeBatch();
}
catch(StringIndexOutOfBoundsException e){
    System.out.println("empty");
}

```

```

    }
}

void createTable(String cstr,String tableName) throws IOException, SQLException {
    try{
        Statement stm= connection.createStatement();
        STGroup stGroup =

new STGroupFile("src/main/java/FileToTableConversion/StringTemplates.stg");
        // Pick the correct template
        ST template1 = stGroup.getInstanceOf("templateForCreationJson");
        template1.add("param1",tableName);
        template1.add("param2",cstr);
        String query1 = template1.render();
        logger.info(query1);
        stm.executeUpdate(query1);
    }
    catch(PSQLException e){
        e.printStackTrace();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

void dropTable(String tableName) throws SQLException, IOException {
    try {
        Statement stm= connection.createStatement();
        STGroup stGroup =

        new STGroupFile("src/main/java/FileToTableConversion/StringTemplates.stg");
        // Pick the correct template

```

```

    ST template1 = stGroup.getInstanceOf("templateForDropJson");
    template1.add("param1",tableName);
    String query1 = template1.render();
    logger.info(query1);
    stm.executeUpdate(query1);
}
catch(PSQLException e){
    e.printStackTrace();
}
}
}

```

Explanation :

- Here initialized connection object.
- Insertion of records with batches of size 10000.
- **JSONArray** : It is an ordered sequence of values. It provides methods to access values by index and to put values.
- **JSONObject** : It is a unordered collection of key-value pair. Provide methods to access values by keys.
- **Map**: It is an interface that maps keys to values.
- Here executing insert queries when count reaches to 10000
- Remaining insert queries are executed at last.
- If we execute by batches execution time reduces.
- Here create Operation is performed when called createTable().
- The statement interface is used to create SQL basic statements in Java it provides methods to execute queries with the database.
- Used String Templates to write query.

StringTemplate.stg:

```
templateForCreationJson(param1,param2) ::= <<
Create table <param1>_json(<param2>);
>>

templateForInsertionJson(param1,param2) ::= <<
insert into <param1>_json values(<param2>);
>>

templateForDropJson(param1) ::= <<
drop table <param1>_json;
>>
```

Output:

```
ersion.Main - Enter database name:
ersion.Main - Enter the folder path :
ersion.ConnectionClass - Connection established successfully...
ersion.Main - 5
ersion.Main - Create table address_lookup_mt4020_json(country varchar(100),pincode varchar(100),city varchar(100),voter_id varchar(100),house_no varchar(100),state varchar(100)
ersion.Main - 4
ersion.Main - Create table constraint_table_usage_json(constraint_catalog varchar(100),constraint_name varchar(100),table_schema varchar(100),table_catalog varchar(100),table_
ersion.Main - 11074
ersion.Main - Create table customer_dataset1_json(gender varchar(100),nationality varchar(100),house_no____flat_no varchar(100),postal_code_1 varchar(100),dob varchar(100),pn v
ersion.Main - 2
ersion.Main - Create table enabled_roles_json(role_name varchar(100));
ersion.Main - 1
ersion.Main - Create table mt4020_csv_us7_8_json(constraint_catalog varchar(100),constraint_name varchar(100),constraint_schema varchar(100),check_clause varchar(100));
ersion.Main - 3
ersion.Main - Create table san_json(n1 varchar(100),n2 varchar(100),n3 varchar(100),id varchar(100));
ersion.Main - 5
ersion.Main - Create table schemata_json(default_character_set_name varchar(100),catalog_name varchar(100),schema_owner varchar(100),sql_path varchar(100),default_character_se
ersion.Main - 12
ersion.Main - Create table sql_implementation_info_json(character_value varchar(100),comments varchar(100),implementation_info_name varchar(100),implementation_info_id varchar
ersion.Main - 4
ersion.Main - Create table sql_languages_json(sql_language_implementation varchar(100),sql_language_source varchar(100),sql_language_conformance varchar(100),sql_language_bind
ersion.Main - 2
ersion.Main - Create table userstory03_1_json(created_time varchar(100),ename varchar(100),modified_time varchar(100),modified_by varchar(100),id varchar(100),sal varchar(100))
```

- Tables in database :

SELECT table_name FROM SYS	
	asc table_name
1	mt4020_csv_us7_8_json
2	customer_dataset1_json
3	enabled_roles_json
4	san_json
5	schemata_json
6	sql_implementation_info
7	sql_languages_json
8	userstory03_1_json
9	address_lookup_mt4020
10	constraint_table_usage_j

User Story-8

8. Implement a Virtual Constructor using Factory Design Pattern for a real time use case

Description: Using Factory Design pattern concept, implement a real time approach of using this with multiple classes, interfaces, POJO's, constructors and make sure that you achieve 100% abstraction.

Acceptance Criteria:

- Design Pattern is implemented.
- 100% Abstraction is achieved.
- Understood the concepts of using design patterns.

CODE :

Main Class :

```
package AbstarctFactoryPatterns;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class Main {

    private static final Logger logger = (Logger) LogManager.getLogger(Main.class);

    public static void main(String[] args) throws IOException {

        BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

        logger.info("Enter the name of type of Sim you want to recharge: ");

        String simName=br.readLine();

        logger.info("Enter number of months you want to recharge : ");

        String numOfMonths=br.readLine();
```

```

Sim s= RechargeSim.getSimName(simName);
Months m= GetMonths.getMonths(numOfMonths);
RateClass r=new RateClass();
logger.info("Interest rate for "+s.getSimName()+" sim is " +r.getRate(s.getSimName()));
double rate=r.getRate(s.getSimName());
int months=m.getMonths();
GetRateMonth amount=new GetRateMonth();
amount.getRateMonths(rate,months);
amount.calculatePayment();
}
}

```

Explanation :

- Import statements:
- **java.io.IOException** : is an exception which programmers use in the code to throw a failure in Input & Output operations.
- **java.sql.Connection**: is an exception which programmers use in the code to throw a failure in sql connection.
- **java.util.BufferedReader** : Used buffer reader for taking input from user.
- Here created objects for classes and called methods of that classes.
- The readLine () method of Console class in Java is used to read a single line of text from the console

Sim Interface :

```

package                                AbstarctFactoryPatterns;
public                                interface                                Sim                                {
    String                                getSimName();
}

```

Explanation :

- **Interface** : Like a class, an interface can have methods and variables, but the methods declared in interface are by default abstract.
- This interface has one method getSimName().

Months Interface :

```
package AbstarctFactoryPatterns;  
  
public interface Months {  
  
    int getMonths();  
  
}
```

Explanation :

- **Interface** : Like a class, an interface can have methods and variables, but the methods declared in interface are by default abstract.
- This interface has one method getMonths().

JIO class:

```
package AbstarctFactoryPatterns;  
  
public class JIO implements Sim{  
  
    private final String simName;  
  
    JIO(){  
  
        simName="JIO";  
  
    }  
  
    @Override  
  
    public String getSimName(){  
  
        return simName;  
  
    }  
  
}
```

```
}  
  
}
```

Explanation :

- JIO class implements Sim interface.
- And implemented getSimName method().

Airtel class :

```
package AbstarctFactoryPatterns;
```

```
public class Airtel implements Sim{
```

```
    private final String simName;
```

```
    Airtel(){
```

```
        simName="Airtel";
```

```
    }
```

```
    @Override
```

```
    public String getSimName(){
```

```
        return simName;
```

```
    }
```

```
}
```

Explanation :

- Airtel class implements Sim interface.
- And implemented getSimName method().

Idea class :

```
package AbstarctFactoryPatterns;
```

```
public class Idea implements Sim{

    private final String simName;

    Idea(){

        simName="Idea";

    }

    @Override

    public String getSimName(){

        return simName;

    }

}
```

Explanation :

- Idea class implements Sim interface.
- And implemented getSimName method().

OneMonth Class :

```
package AbstarctFactoryPatterns;

public class OneMonth implements Months {

    private final int months;

    OneMonth(){

        months=1;

    }

    @Override
```

```
public int getMonths() {  
    return months;  
}  
}
```

Explanation :

- It is implementing Months interface.
- Wrote body for getMonths() method.

TwoMoths Class :

package AbstarctFactoryPatterns;

```
public class TwoMonths implements Months {  
    private final int months;  
  
    TwoMonths(){  
        months=2;  
    }  
  
    @Override  
    public int getMonths() {  
        return months;  
    }  
}
```

Explanation :

- It is implementing Months interface.
- Wrote body for getMonths() method.

ThreeMonths Class :

```
package AbstarctFactoryPatterns;

public class ThreeMonths implements Months {

    private final int months;

    ThreeMonths(){

        months=3;

    }

    @Override

    public int getMonths() {

        return months;

    }

}
```

Explanation :

- It is implementing Months interface.
- Wrote body for getMonths() method.

SixMonths class:

```
package AbstarctFactoryPatterns;

public class SixMonths implements Months{

    int months;

    SixMonths(){

        months=6;

    }

    @Override
```

```
public int getMonths() {  
  
    return months;  
  
}  
  
}
```

Explanation :

- It is implementing Months interface.
- Wrote body for getMonths() method.

RechargeSim Class :

```
package AbstarctFactoryPatterns;  
  
class RechargeSim{  
  
    public static Sim getSimName(String bank){  
  
        if(bank == null){  
  
            return null;  
  
        }  
  
        if(bank.equalsIgnoreCase("JIO")){  
  
            return new JIO();  
  
        } else if(bank.equalsIgnoreCase("Airtel")){  
  
            return new Airtel();  
  
        } else if(bank.equalsIgnoreCase("Idea")){  
  
            return new Idea();  
  
        }  
  
        return null;  
  
    }  
  
}
```



```
}
```

Explanation :

- If we want to recharge JIO sim then it returns reference of that class.
- If we want to recharge Airtel sim then it returns reference of that class.
- If we want to recharge Idea sim then it returns reference of that class.
- Else it will return null.

GetMonths Class :

```
package AbstarctFactoryPatterns;
```

```
public class GetMonths {
```

```
    public static Months getMonths(String months){
```

```
        if(months == null){
```

```
            return null;
```

```
        }
```

```
        if(months.equalsIgnoreCase("one") || months.equalsIgnoreCase("1")){
```

```
            return new OneMonth();
```

```
        } else if(months.equalsIgnoreCase("two") || months.equalsIgnoreCase("2")){
```

```
            return new TwoMonths();
```

```
        } else if(months.equalsIgnoreCase("three") || months.equalsIgnoreCase("3")){
```

```
            return new ThreeMonths();
```

```
        }
```

```
        else if(months.equalsIgnoreCase("three") || months.equalsIgnoreCase("6")){
```

```
            return new SixMonths();
```

```
        }
```

```
        return null;
    }
}
```

Explanation :

- If we want to recharge for one month then it returns reference of that class.
- If we want to recharge for two month then it returns reference of that class.
- If we want to recharge for three month then it returns reference of that class.
- If we want to recharge for six month then it returns reference of that class.
- Else it will return null.

Amount Class :

```
package AbstarctFactoryPatterns;

public abstract class Amount {

    protected double rate;

    protected int months;

    abstract void getRateMonths(double rate, int months);

    public void calculatePayment(){

        double amount=150*months+(150*rate);

        System.out.println("Total amount for "+months+ " is "+amount);

    }

}
```

GetRateMonth class:

```
package AbstarctFactoryPatterns;

public class GetRateMonth extends Amount{
```

```

void getRateMonths(double r, int m) {

    rate=r;

    months=m;

}

}

}

```

Explanation :

- Here Amount has two methods one abstract method and calculatePayment() method.
- calculatePayment() method contain calculation of total amount.
- getRateMonths() method in GetRateMonth class as implementation.

Rate Class :

```

package AbstarctFactoryPatterns;

public class RateClass {

    double getRate(String simName) {

        if ("JIO".equalsIgnoreCase(simName)) {

            return 1.2;

        }

        if ("Airtel".equalsIgnoreCase(simName)) {

            return 1.5;

        }

        if ("Idea".equalsIgnoreCase(simName)) {

            return 1.0;

```

```

    }

    return 0.0;

}

}

```

Explaantion:

- Here each sim has different rate.
- When given JIO sim it returned 1.2 as rate.
- When given Airtel sim it returned 1.5 as rate.
- When given Idea sim it returned 1.0 as rate.

Output:

```

C:\Program Files\Java\jdk1.8.0_261\bin\java.exe ...
19:42:50.280 [main] INFO  AbstarctFactoryPatterns.Main - Enter the name of type of Sim you want to recharge:
JIO
19:42:58.480 [main] INFO  AbstarctFactoryPatterns.Main - Enter number of months you want to recharge :
Two
19:43:02.069 [main] INFO  AbstarctFactoryPatterns.Main - Interest rate for JIO sim is 1.2
Total amount for 2 is 480.0

Process finished with exit code 0

```

User Story 9

9. Read files and write to JDBC using multi-Threading

Description: Using multi-threading, spin 5 threads and each thread should read data from a different file and should insert data into a JDBC destination

Input:

Enter name of the folder:

Output:

Data from file <filename_1> is inserted into table using thread <thread_no 1>

Data from file <filename_2> is inserted into table using thread <thread_no 2>

Data from file <filename_3> is inserted into table using thread <thread_no 3>

Data from file <filename_4> is inserted into table using thread <thread_no 4>

Data from file <filename_5> is inserted into table using thread <thread_no 5>

Acceptance Criteria:

- Data is ingested into destination successfully
- Understood concepts of Files IO, JDBC, Threads

CODE :

Main Class:

```
package MultiThreading;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import java.io.File;
import java.io.IOException;
import java.sql.Connection;
import java.sql.SQLException;
import java.util.Scanner;

public class Main {

    private static final Logger logger = LogManager.getLogger(Main.class);

    public static void main(String[] args) throws SQLException, IOException {

        Scanner sc=new Scanner(System.in);

        logger.info("Enter the folder name:");

        String str= sc.nextLine();

        File folder = new File("src\\main\\resources\\"+str);

        File[] listOfFiles = folder.listFiles() ;

        for(int i=0;i<5;i++) {

            ReaderClass rc = new ReaderClass(i,listOfFiles[i]);
```

```
        rc.start();
    }
}
}
```

Explanation:

- **java.io.IOException** : is an exception which programmers use in the code to throw a failure in Input & Output operations.
- **java.util.Scanner** : To use Scanner class for taking input from user.
- The method getLogger() method belongs to LogManager class.
- **java.io.File** : To import File class.
- The File class contains several methods for working with the pathname, deleting and renaming files, creating new file.
- Created a logger object which is used to log input and output to log file.
- The info() method of a Logger class is used to log an info message.
- Scanner class in Java is found in the java.util package.
- Java provides various ways to read input from the keyboard, the java.util.Scanner class is one of them.
- Used Scanner class to take input.
- `nextLine()` : It is used to get the input string from user.
- The `java.io.File.listFiles ()` method returns the array of pathnames defining the files in the directory denoted by this abstract pathname.
- Created a object for ReaderClass.Then started a thread.
- Here five threads are created for reading each file.

Output:



Connection Class :

```

package FileToTableConversion;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;
import org.json.simple.parser.ParseException;
import java.io.FileReader;
import java.io.IOException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.Base64;

public class ConnectionClass {

    private static final Logger logger = LogManager.getLogger(ConnectionClass.class);
    //getConnection() method returns a connection object
    public Connection getConnection() throws IOException, SQLException {
        //Created a JSONParser object
        JSONParser jp = new JSONParser();
        //Creation of Connection Object
        Connection c = null;
        try {

            JSONObject jo = (JSONObject) jp.parse(new FileReader("src/test/config.json"));
            //username from config.json
            String username =(String) jo.get("Username");
            //Encrypted password from config.json
            String pwd = (String)jo.get("Password");
            //Code to decrypt password
            Base64.Decoder d = Base64.getUrlDecoder();

```

```

String password = new String(d.decode(pwd));
String Url=(String) jo.get("url");
try {
    //Initialization of connection object
    c = DriverManager.getConnection(Url, username, password);
    logger.info("Connection established successfully...");
}
catch (SQLException e) {
    e.printStackTrace();
}
}
catch (ParseException e) {
    e.printStackTrace();
}
return c;
}
}

```

Explanation :

- **java.io.IOException** : is an exception which programmers use in the code to throw a failure in Input & Output operations.
- **java.sql.Connection**: is an exception which programmers use in the code to throw a failure in sql connection
- The method getLogger() method belongs to LogManager class.
- **java.util.Base64** :Has static methods for obtaining encoders and decoders for the Base64 encoding schema.
- Here user name and encrypted password are present in config.json file.
- JSONParser is used to parse json data.
- JSONObject is used to access values based on keys from json String.

- **Base64.Decoder** for decoding byte data using Base64 encoding scheme .
- The **getConnection()** method of DriverManager class is used to establish connection with the database.
- getConnection() method requires three parameters Connection url, Username, Password.
- Here url is "jdbc:postgresql://w3.training5.modak.com:5432/training"
 - "training" is database name.
 - "5432" is port number.
 - "w3.training5.modak.com" is the hostname on which postgresql is running.
- Username and password are taken from config.json.

Config.json File:

```
{
  "Username": "mt4020",
  "Password": "bXQ0MDIwQ60wMnkyMg",
  "url": "jdbc:postgresql://w3.training5.modak.com:5432/training"
}
```

Reader class:

```
package MultiThreading;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.postgresql.util.PSQLException;
import org.stringtemplate.v4.ST;
import org.stringtemplate.v4.STGroup;
import org.stringtemplate.v4.STGroupFile;

import java.io.File;
import java.io.IOException;
import java.sql.Connection;
import java.sql.SQLException;
```

```
import java.sql.Statement;
```

```
import java.util.Scanner;
```

```
class ReaderClass extends Thread {
```

```
    private static final Logger logger = LogManager.getLogger(ReaderClass.class);
```

```
    int i;
```

```
    File f1;
```

```
    ReaderClass(int i, File f1) {
```

```
        this.i = i;
```

```
        this.f1 = f1;
```

```
    }
```

```
    public void run() {
```

```
        try {
```

```
            logger.info("Data from file "+ f1.getName() +" is inserted into table using thread  
"+Thread.currentThread().getId());
```

```
            createInsertOperations(f1);
```

```
        }
```

```
        catch (Exception e) {
```

```
            logger.info("Exception is caught");
```

```
        }
```

```
    }
```

```
    void createInsertOperations(File f1) throws IOException, SQLException {
```

```
        Scanner dataReader5 = new Scanner(f1);
```

```
        String str = dataReader5.nextLine();
```

```
        String[] arr=str.split("\t");
```

```
        String cstr="";
```

```
        for(int i=0;i<arr.length;i++) {
```

```
            if (i != arr.length - 1) {
```

```

        cstr = cstr + arr[i] + " varchar(100),";
    } else {
        cstr = cstr + arr[i] + " varchar(100)";
    }
}

createTable(cstr);
while (dataReader5.hasNextLine()) {
    String str1= dataReader5.nextLine();
    String[] iarr = str1.split("\t");
    String istr = "";
    for (int i = 0; i < arr.length; i++) {
        if (i != iarr.length - 1) {
            istr = istr + "\"" + iarr[i] + "\"" + ",";
        } else {
            istr = istr + "\"" + iarr[i] + "\"";
        }
    }
    insertData(istr);
}

}

void createTable(String cstr) throws IOException, SQLException {
    try{
        ConnectionClass c = new ConnectionClass();
        Connection cnt = c.getConnection();
        Statement stm= cnt.createStatement();
        STGroup stGroup = new STGroupFile("src\\main\\java\\StringTemplate.stg");
        ST template1 = stGroup.getInstanceOf("templateForCreation");
        template1.add("param1",i);
    }
}

```

```

        template1.add("param2",cstr);
        String query1=template1.render();
        stm.executeUpdate(query1);
        cnt.close();}
    catch(PSQLException e){
        e.printStackTrace();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

void insertData(String istr){
    try {
        ConnectionClass c = new ConnectionClass();
        Connection cnt = c.getConnection();
        Statement stm = cnt.createStatement();
        STGroup stGroup = new STGroupFile("src\\main\\java\\StringTemplate.stg");
        ST template1 = stGroup.getInstanceOf("templateForInsertion");
        template1.add("param1",i);
        template1.add("param2",istr);
        String query1=template1.render();
        stm.executeUpdate(query1);
        cnt.close();
    }

    catch(PSQLException e){
        e.printStackTrace();
    } catch (SQLException | IOException e) {
        e.printStackTrace();
    }
}

```

```
}  
}
```

Explanation:

- **java.io.IOException** : is an exception which programmers use in the code to throw a failure in Input & Output operations.
- **java.util.Scanner** : To use Scanner class for taking input from user.
- The method getLogger() method belongs to LogManager class.
- **java.io.File** : To import File class.
- The File class contains several methods for working with the pathname, deleting and renaming files, creating new file.
- ReaderClass() constructor has two parameters file path and index.
- Each thread starts and insert data to table from different files.
- Here we are calling createInsertOperation() which create table and perform insert operation.
- Each thread takes different file path and perform create and insert operation .
- Called createTable() method and insertData() method.
- Created a object for connection class and called getConnection() method.
- Created a stg file with name "StringTemplates.stg".
- Named template as templateForCreation with two parameters.
- Wrote query for insertion into database.
- Then executed query using executeUpdate().
- Created a object for connection class and called getConnection() method.
- Created a stg file with name "StringTemplates.stg".
- Named template as templateForInsertion with two parameters.
- Wrote query for insertion into database.
- Then executed query using executeUpdate().

Output :

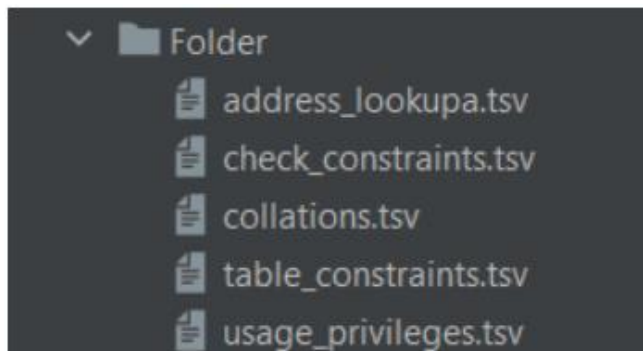
```

"2022-05-04 17:01:46.948 [main] INFO MultiThreading.Main - Enter the folder name:
"2022-05-04 17:01:54.169 [Thread-2] INFO MultiThreading.ReaderClass - Data from file check_constraints.tsv is inserted into table using thread 15
"2022-05-04 17:01:54.169 [Thread-1] INFO MultiThreading.ReaderClass - Data from file address_lookupa.tsv is inserted into table using thread 14
"2022-05-04 17:01:54.169 [Thread-3] INFO MultiThreading.ReaderClass - Data from file collations.tsv is inserted into table using thread 16
"2022-05-04 17:01:54.169 [Thread-4] INFO MultiThreading.ReaderClass - Data from file table_constraints.tsv is inserted into table using thread 17
"2022-05-04 17:01:54.169 [Thread-5] INFO MultiThreading.ReaderClass - Data from file usage_privileges.tsv is inserted into table using thread 18

```

Data in database :

- Data from different files stored in different table by threads.



- Data in table multiThreading_us9_table0

multithreading_us9_table0 1 ×									
	abc voter_id	abc house_no	abc street_name	abc landmark	abc city	abc state	abc country	abc	abc pincode
1	1	1-5/4	hid-45	bankofbaroda	hyderabad	telangana	india		505327
2	2	1-5/3	BIT-89	vnrhostel	warangal	AP	india		505320
3	3	1-5/5	st-09	pubg_hotel	nizambad	UP	india		505327
4	4	1-5/6	hi-05	hdfca	jagital	WB	india		505397
5	5	1-5/7	hy-65	babaroda	knr	Delhi	india		567327

- Data in multiThreading_us9_table1

multithreading_us9_table1				
	abc constraint_catalog	abc constraint_schema	abc constraint_name	abc check_clause
1	training	public	2200_266514_2_not_null	ename IS NOT NULL
2	training	public	2200_276077_2_not_null	ename1 IS NOT NULL
3	training	public	2200_276077_1_not_null	eid IS NOT NULL
4	training	public	2200_268157_1_not_null	voter_id IS NOT NULL
5	training	public	2200_266514_1_not_null	eid IS NOT NULL
6	training	public	2200_276048_2_not_null	ename IS NOT NULL
7	training	public	2200_276048_1_not_null	eid IS NOT NULL

- Data in table multiThreading_us9_table2

select * from multiThreading_us9_table2

	collation_catalog	collation_schema	collation_name	pad_attribute
1	training	pg_catalog	default	NO PAD
2	training	pg_catalog	C	NO PAD
3	training	pg_catalog	POSIX	NO PAD
4	training	pg_catalog	ucs_basic	NO PAD
5	training	pg_catalog	aa_DJ.utf8	NO PAD
6	training	pg_catalog	aa_ER	NO PAD
7	training	pg_catalog	aa_ER@saaho	NO PAD
8	training	pg_catalog	aa_ER.utf8	NO PAD
9	training	pg_catalog	aa_ER.utf8@saaho	NO PAD
10	training	pg_catalog	aa_ET	NO PAD
11	training	pg_catalog	aa_ET.utf8	NO PAD
12	training	pg_catalog	af_ZA.utf8	NO PAD
13	training	pg_catalog	am_ET	NO PAD

- Data in table multiThreading_us9_table3

select * from multiThreading_us9_table3

	constraint_catalog	constraint_schema	constraint_name	table_catalog	table_schema	table_name	constraint_type	is_deferrable
1	training	public	voter_details_mt4020_pkey	training	public	voter_details_mt4020	PRIMARY KEY	NO
2	training	public	n_tablename_pkey	training	public	n_tablename	PRIMARY KEY	NO
3	training	public	empp_pkey	training	public	empp	PRIMARY KEY	NO
4	training	public	empp1_pkey	training	public	empp1	PRIMARY KEY	NO
5	training	public	2200_268157_1_not_null	training	public	voter_details_mt4020	CHECK	NO
6	training	public	2200_266514_1_not_null	training	public	n_tablename	CHECK	NO
7	training	public	2200_266514_2_not_null	training	public	n_tablename	CHECK	NO
8	training	public	2200_276048_1_not_null	training	public	empp	CHECK	NO
9	training	public	2200_276048_2_not_null	training	public	empp	CHECK	NO
10	training	public	2200_276077_1_not_null	training	public	empp1	CHECK	NO
11	training	public	2200_276077_2_not_null	training	public	empp1	CHECK	NO

- Data in table multiThreading_us9_table4

multithreading_us9_table4 1

select * from multiThreading_us9_table4

	grantor	grantee	object_catalog	object_schema	object_name	object_type	privilege_type	is_grantable
1	postgres	PUBLIC	training	pg_catalog	default	COLLATION	USAGE	NO
2	postgres	PUBLIC	training	pg_catalog	C	COLLATION	USAGE	NO
3	postgres	PUBLIC	training	pg_catalog	POSIX	COLLATION	USAGE	NO
4	postgres	PUBLIC	training	pg_catalog	ucs_basic	COLLATION	USAGE	NO
5	postgres	PUBLIC	training	pg_catalog	aa_DJ.utf8	COLLATION	USAGE	NO
6	postgres	PUBLIC	training	pg_catalog	aa_ER	COLLATION	USAGE	NO
7	postgres	PUBLIC	training	pg_catalog	aa_ER@saaho	COLLATION	USAGE	NO
8	postgres	PUBLIC	training	pg_catalog	aa_ER.utf8	COLLATION	USAGE	NO
9	postgres	PUBLIC	training	pg_catalog	aa_ER.utf8@saaho	COLLATION	USAGE	NO
10	postgres	PUBLIC	training	pg_catalog	aa_ET	COLLATION	USAGE	NO
11	postgres	PUBLIC	training	pg_catalog	aa_ET.utf8	COLLATION	USAGE	NO
12	postgres	PUBLIC	training	pg_catalog	af_ZA.utf8	COLLATION	USAGE	NO
13	postgres	PUBLIC	training	pg_catalog	am_ET	COLLATION	USAGE	NO