Java Streams

- ‘Stream’ is basically a ‘process sequence’ through which a data can either be written in a file or can be read from a file.
- ‘Stream’ can be imagined as a ‘Channel’ through which data bytes are either flows from program to a destination file or from a destination file to a program.
- In order to write ‘characters’ to a file we need an ‘outputStream’.
- In order to read ‘characters’ from a file we need an ‘inputStream’.

Writing a text file using java

- The steps to write a simple text file are:

1. Use a output stream which will selects the directory in which our text file will be save.
2. Use a output stream to create a file and chain it to previous step.
3. Use ‘BufferedWriter’ to write ‘characters’ in the file and chain it to previous step.
4. Close the last stream.

```
public class Test {

public static void main(String[] args) throws IOException
{
//output stream to select directory
File f = new File("c://new.txt");
//output stream to write files, and chaining to previous step
FileWriter fw = new FileWriter(f);
//Chaining Buffered writer to write in the text file
BufferedWriter writer = new BufferedWriter(fw);
//writing in the text file
String name = "nandan";
String surname = "singh";
String para = " once upon a time there was a king, simba, who is " +
"taken care by 'Timon' and 'pumba";
writer.write(name);
//creating a new line using method of BufferedWriter
writer.newLine();
writer.write(surname);
writer.newLine();
writer.write(para);
//closing the last stream
writer.close();
//end of the process for our acknowledge
System.out.println("file has been created");
```

Reading a text file using java

- The steps to read a simple text file are:

1. Use a input which will selects the directory in which our text file is already saved.
2. Use a input stream to use that file and chain it to previous step.
3. Use ‘BufferedReader’ to read ‘characters’ or ‘lines’ from the file and chain it to previous step.
4. Close the last stream.

```
public class FileReading {

public static void main(String[] args) throws IOException
{
//input stream for selecting files from a directory
File f = new File("c:\\new.txt");
//input stream to read the file and chaining it to previous step
FileReader fr = new FileReader(f);
//BufferedReader to read character and chaining it to previous step
BufferedReader reader = new BufferedReader(fr);

String line = null;
while((line=reader.readLine())!= null)
{
System.out.println(line);
}
//closing the last stream
reader.close();

}

}
```

Apache Poi

- Apache POI (poor Obfuscation implementation) is basically a JAVA API which is used to handle Microsoft .xls and xlsx files effectively using java codes.
- For dealing with excel (97 – 2003) file format, HSSF API is used i.e. for .xls files.
- For dealing with .xlsx file format , XSSF API is used i.e. for .xlsx files
- How to use Apache poi:

1. Download Apache Poi API by Apache website.
2. Configure build path and add external jars to the projects build path

Writing .xls file using POI

1. Create excel workbook using POI.
2. Create sheet (or sheets) in above workbook.
3. Create Row (or rows) in above sheet.
4. Create Cell (or cells) in above Row.
5. Use a output stream which will selects the directory in which our .xls file will be save.
6. Use a output stream to create a file and chain it to previous step.
7. Use ‘WorkBook.writer’ to write ‘characters’ in the file and chain it to previous step.
8. Close the output stream

```
public class LearningPoi {
public static void main(String[] args) throws IOException
{
//creating work book
XSSFWorkbook workbook = new XSSFWorkbook();

//create sheet on the workbook , the HSSFSheet have private constructor
XSSFSheet sheet1 = workbook.createSheet("first sheet");
XSSFSheet sheet2 = workbook.createSheet("second sheet");

//create row in sheet1
Row row0 = sheet1.createRow(0);
Row row1 = sheet1.createRow(1);
Row row2 = sheet1.createRow(2);

//create cell in row 0
Cell cellA = row0.createCell(0);

//setting cell value
cellA.setCellValue("Name");
```

```
//repeating above step for other cells
Cell cellB = row0.createCell(1);
cellB.setCellValue("email");
Cell cellC = row0.createCell(2);
cellC.setCellValue("mobile Number");
Cell cellD = row0.createCell(3);

//creating file stream
File f =  new File("c:/new.xlsx");
//chaining output stream to path
FileOutputStream fo = new FileOutputStream(f);
//Writing workbook to output stream
workbook.write(fo);
//closing stream
fo.close();
System.out.println("excel file is writtern");
}}
```

Reading excel file using POI

1. Use a input which will selects the directory in which our excel file is already saved.
2. Use a input stream to use that file and chain it to previous step.
3. Use ‘WorkbookFactory’ to read ‘characters’ or ‘lines’ from the file and chain it to previous step.
4. Read the excel file with appropriate logic.

```
public class ReadingPoi {

public static void main(String[] args) throws InvalidFormatException, IOException
{
//input file stream
File f = new File("c:\\new.xls");
//connecting to input stream
FileInputStream fi = new FileInputStream(f);
//connecting workbook to input stream
Workbook workbook = WorkbookFactory.create(fi);

//get the first sheet
org.apache.poi.ss.usermodel.Sheet sheet0 = workbook.getSheetAt(0);
//get the first row
Row row0 = sheet0.getRow(0);

//get the first cell
Cell cell0 = row0.getCell(0);
Cell cell1 = row0.getCell(1);
Cell cell2 = row0.getCell(2);

System.out.println("cell 0 is=--> "+cell0+" cell 1 is -->"+cell1
+" cell 2 is -->" + cell2);

}
```

Mysql connectivity

Reading data from database

Writing data to the database