APACHE POI
Tutorial
About the tutorial

Apache POI Tutorial
This tutorial provides a basic understanding of Apache POI library and its features.

Audience
This tutorial is designed for all enthusiastic readers working on Java and especially those who want to create, read, write, and modify Excel files using Java.

Prerequisites
A general awareness of Java programming with JDK1.5 or later versions and IO concepts in Java are the only prerequisites to understand this tutorial.

Copyright & Disclaimer
© Copyright 2014 by Tutorials Point (I) Pvt. Ltd.

All the content and graphics published in this e-book are the property of Tutorials Point (I) Pvt. Ltd. The user of this e-book is prohibited to reuse, retain, copy, distribute or republish any contents or a part of contents of this e-book in any manner without written consent of the publisher.

The API of Apache POI contains a number of methods and classes. In this tutorial, we have used only some of those for demonstration purpose. We encourage the readers to refer the complete API document for a comprehensive understanding.

We strive to update the contents of our website and tutorials as timely and as precisely as possible, however, the contents may contain inaccuracies or errors. Tutorials Point (I) Pvt. Ltd. provides no guarantee regarding the accuracy, timeliness or completeness of our website or its contents including this tutorial. If you discover any errors on our website or in this tutorial, please notify us at contact@tutorialspoint.com
# Apache POI Tutorial

## Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache POI Tutorial</td>
<td>3</td>
</tr>
<tr>
<td>Audience</td>
<td>3</td>
</tr>
<tr>
<td>Prerequisites</td>
<td>3</td>
</tr>
<tr>
<td>Copyright &amp; Disclaimer</td>
<td>3</td>
</tr>
<tr>
<td>Apache POI – Overview</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>What is Apache POI?</td>
</tr>
<tr>
<td></td>
<td>Components of Apache POI</td>
</tr>
<tr>
<td>Flavors of Java Excel API</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>Aspose Cells for Java</td>
</tr>
<tr>
<td></td>
<td>JXL</td>
</tr>
<tr>
<td></td>
<td>JExcel</td>
</tr>
<tr>
<td></td>
<td>Apache POI</td>
</tr>
<tr>
<td>Apache POI Installation</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>System Requirements</td>
</tr>
<tr>
<td></td>
<td>Step 1: Verify your Java Installation</td>
</tr>
<tr>
<td></td>
<td>Step 2: Set your Java Environment</td>
</tr>
<tr>
<td></td>
<td>Step 3: Install Apache POI Library</td>
</tr>
<tr>
<td>POI – Classes and Methods</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>Workbook</td>
</tr>
<tr>
<td></td>
<td>Sheet</td>
</tr>
<tr>
<td></td>
<td>Row</td>
</tr>
<tr>
<td></td>
<td>Cell</td>
</tr>
<tr>
<td></td>
<td>HSSFCColor</td>
</tr>
<tr>
<td></td>
<td>XSSFCColor</td>
</tr>
<tr>
<td></td>
<td>XSSFHyperlink</td>
</tr>
<tr>
<td>Workbooks</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>Create Blank Workbook</td>
</tr>
<tr>
<td></td>
<td>Open Existing Workbook</td>
</tr>
<tr>
<td>Spreadsheets</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td>Create a Spreadsheet</td>
</tr>
<tr>
<td></td>
<td>Rows on Spreadsheet</td>
</tr>
<tr>
<td>Cells</td>
<td>42</td>
</tr>
<tr>
<td></td>
<td>Create a Cell</td>
</tr>
</tbody>
</table>
Fonts and Text Direction ................................................................. 52
Fonts and Font Styles ................................................................. 52
Text Direction ............................................................................ 55
Formula ....................................................................................... 56
Hyperlink .................................................................................... 56
Print Area .................................................................................... 56
POI Interaction with Database ..................................................... 56
Many a time, a software application is required to generate reports in Microsoft Excel file format. Sometimes, an application is even expected to receive Excel files as input data. For example, an application developed for the Finance department of a company will be required to generate all their outputs in Excel.

Any Java programmer who wants to produce MS Office files as output must use a predefined and read-only API to do so.

What is Apache POI?
Apache POI is a popular API that allows programmers to create, modify, and display MS Office files using Java programs. It is an open source library developed and distributed by Apache Software Foundation to design or modify Microsoft Office files using Java program. It contains classes and methods to decode the user input data or a file into MS Office documents.

Components of Apache POI
Apache POI contains classes and methods to work on all OLE2 Compound documents of MS Office. The list of components of this API is given below.

- **POIFS** (Poor Obfuscation Implementation File System): This component is the basic factor of all other POI elements. It is used to read different files explicitly.

- **HSSF** (Horrible Spreadsheet Format): It is used to read and write **xls** format of MS-Excel files.

- **XSSF** (XML Spreadsheet Format): It is used for **xlsx** file format of MS-Excel.
Apache POI Tutorial

- **HPSF** (Horrible Property Set Format): It is used to extract property sets of the MS-Office files.

- **HWPF** (Horrible Word Processor Format): It is used to read and write doc extension files of MS-Word.

- **XWPF** (XML Word Processor Format): It is used to read and write docx extension files of MS-Word.

- **HSLF** (Horrible Slide Layout Format): It is used for read, create, and edit PowerPoint presentations.

- **HDGF** (Horrible DiaGram Format): It contains classes and methods for MS-Visio binary files.

- **HPBF** (Horrible PuBlisher Format): It is used to read and write MS-Publisher files.

This tutorial guides you through the process of working on Excel files using Java. Therefore the discussion is confined to HSSF and XSSF components.

**Note:** Older versions of POI support binary file formats such as doc, xls, ppt, etc. Version 3.5 onwards, POI supports OOOXML file formats of MS-Office such as docx, xlsx, pptx, etc.

Like Apache POI, there are other libraries provided by various vendors for Excel file generation. These include Aspose cells for Java by Aspose, JXL by Commons Libraries, and JExcel by Team Dev.
This chapter takes you through some of the flavors of Java Excel API and their features. There are many vendors who provide Java Excel related APIs; some of them are considered in this chapter.

**Aspose Cells for Java**

Aspose Cells for Java is a purely licensed Java Excel API developed and distributed by the vendor Aspose. The latest version of this API is 8.1.2, released in July 2014. It is a rich and heavy API (combination of plain Java classes and AWT classes) for designing the Excel component that can read, write, and manipulate spreadsheets.

The common uses of this API are as follows:

- Excel reporting, build dynamic Excel reports
- High-fidelity Excel rendering and printing
- Import and export data from Excel spreadsheets
- Generate, edit, and convert spreadsheets

**JXL**

JXL is a third-party framework designed for Selenium that supports data driven automation on web browsers (auto-update of data on web browsers). However it is also used as a common support library for JExcel API because it has basic features to create, read, and write spreadsheets.

The basic features are as follows:

- Generate Excel files
- Import data from workbooks and spreadsheets
- Obtain the total number of rows and columns.
**JExcel**

JExcel is a purely licensed API provided by Team Dev. Using this, programmers can easily read, write, display, and modify Excel workbooks in both .xls and .xlsx formats. This API can be easily embedded with Java Swing and AWT. The latest version of this API is Jexcel-2.6.12, released in 2009.

The main features are as follows.

- Automate Excel application, workbooks, spreadsheets, etc.
- Embed workbooks in a Java Swing application as ordinary Swing component
- Add event listeners to workbooks and spreadsheets
- Add event handlers to handle the behavior of workbook and spreadsheet events
- Add native peers to develop custom functionality

**Apache POI**

Apache POI is a 100% open source library provided by Apache Software Foundation. Most of the small and medium scale application developers depend heavily on Apache POI (HSSF + XSSF). It supports all the basic features of Excel libraries; however, rendering and text extraction are its main features.
Apache POI Tutorial

- Supports Multiple Format
- .xls
- .xlsx

Import/Export
- ResultSet
- Arrays
- Collections

Apache POI (HSSF+XSSF)

Data Processing
- Create Charts
- Apply Formats
- Apply Formulae

Rendering
- Set Print Area
- Print Spreadsheets
- Print Workbooks
- Print Formulae
This chapter takes you through the process of setting up Apache POI on Windows and Linux based systems. Apache POI can be easily installed and integrated with your current Java environment following a few simple steps without any complex setup procedures. User administration is required while installation.

### System Requirements

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>JDK</strong></td>
<td>Java SE 2 JDK 1.5 or above</td>
</tr>
<tr>
<td><strong>Memory</strong></td>
<td>1 GB RAM (recommended)</td>
</tr>
<tr>
<td><strong>Disk Space</strong></td>
<td>No minimum requirement</td>
</tr>
<tr>
<td><strong>Operating System Version</strong></td>
<td>Windows XP or above, Linux</td>
</tr>
</tbody>
</table>

Let us now proceed with the steps to install Apache POI.

### Step 1: Verify your Java Installation

First of all, you need to have Java Software Development Kit (SDK) installed on your system. To verify this, execute any of the two commands depending on the platform you are working on.

If the Java installation has been done properly, then it will display the current version and specification of your Java installation. A sample output is given in the following table.

<table>
<thead>
<tr>
<th>Platform</th>
<th>Command</th>
<th>Sample Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>Open command console and type:</td>
<td>Java version &quot;1.7.0_60&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Java (TM) SE Run Time Environment</td>
</tr>
</tbody>
</table>
We assume the readers of this tutorial have Java SDK version 1.7.0_60 installed on their system.

In case you do not have Java SDK, download its current version from http://www.oracle.com/technetwork/java/javase/downloads/index.html and have it installed.

Step 2: Set your Java Environment

Set the environment variable JAVA_HOME to point to the base directory location where Java is installed on your machine. For example,

<table>
<thead>
<tr>
<th>Platform</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>Set JAVA_HOME to C:\ProgramFiles\java\jdk1.7.0_60</td>
</tr>
<tr>
<td>Linux</td>
<td>Export JAVA_HOME=/usr/local/java-current</td>
</tr>
</tbody>
</table>

Append the full path of Java compiler location to the System Path.

<table>
<thead>
<tr>
<th>Platform</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>Append the String &quot;C:\Program Files\Java\jdk1.7.0_60\bin&quot; to the end of the system variable PATH.</td>
</tr>
<tr>
<td>Linux</td>
<td>Export PATH=$PATH:$JAVA_HOME/bin/</td>
</tr>
</tbody>
</table>
Execute the command `java -version` from the command prompt as explained above.

**Step 3: Install Apache POI Library**

Download the latest version of Apache POI from [http://poi.apache.org/download.html](http://poi.apache.org/download.html) and unzip its contents to a folder from where the required libraries can be linked to your Java program. Let us assume the files are collected in a folder on C drive.

The following images show the directories and the file structure inside the downloaded folder.
Add the complete path of the five **jars** as highlighted in the above image to the CLASSPATH.

<table>
<thead>
<tr>
<th>Platform</th>
<th>Description</th>
</tr>
</thead>
</table>
| **Windows** | Append the following strings to the end of the user variable CLASSPATH:  
  “C:\poi-3.9\poi-3.9-20121203.jar;”  
  “C:\poi-3.9\poi-ooxml-3.9-20121203.jar;”  
  “C:\poi-3.9\poi-ooxml-schemas-3.9-20121203.jar;”  
  “C:\poi-3.9\ooxml-lib\dom4j-1.6.1.jar;”  
  “C:\poi-3.9\ooxml-lib\xmlbeans-2.3.0.jar;”;”  |
| **Linux** | Export CLASSPATH=$CLASSPATH:  
  /usr/share/poi-3.9/poi-3.9-20121203.tar:  
  /usr/share/poi-3.9/poi-ooxml-schemas-3.9-20121203.tar:  
  /usr/share/poi-3.9/poi-ooxml-3.9-20121203.tar:  
  /usr/share/poi-3.9/ooxml-lib/dom4j-1.6.1.tar:  
  /usr/share/poi-3.9/ooxml-lib/xmlbeans-2.3.0.tar |
This chapter explains a few classes and methods under the Apache POI API that are critical to work on Excel files using Java programs.

Workbook

This is the super-interface of all classes that create or maintain Excel workbooks. It belongs to the `org.apache.poi.ss.usermodel` package. The two classes that implement this interface are as follows:

- **HSSFWorkbook**: This class has methods to read and write Microsoft Excel files in .xls format. It is compatible with MS-Office versions 97–2003.

- **XSSFWorkbook**: This class has methods to read and write Microsoft Excel and OpenOffice xml files in .xls or .xlsx format. It is compatible with MS-Office versions 2007 or later.

HSSFWorkbook

It is a high-level class under the `org.apache.poi.hssf.usermodel` package. It implements the Workbook interface and is used for Excel files in .xls format. Listed below are some of the methods and constructors under this class.

**Class Constructors**

<table>
<thead>
<tr>
<th>S. No.</th>
<th>Constructor and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>HSSFWorkbook()</td>
</tr>
<tr>
<td></td>
<td>Creates a new HSSFWorkbook object from scratch.</td>
</tr>
<tr>
<td></td>
<td>HSSFWorkbook(DirectoryNode directory, boolean preserveNodes)</td>
</tr>
<tr>
<td>---</td>
<td>-------------------------------------------------------------</td>
</tr>
<tr>
<td>2</td>
<td>Creates a new HSSFWorkbook object inside a specific directory.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>HSSFWorkbook(DirectoryNode directory, POIFSFileSystem fs, boolean preserveNodes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Given a POIFSFileSystem object and a specific directory within it, it creates an HSSFWorkbook object to read a specified workbook.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>HSSFWorkbook(java.io.InputStream s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Creates a new HSSFWorkbook object using an input stream.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>HSSFWorkbook(java.io.InputStream s, boolean preserveNodes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Constructs a POI file system around your input stream.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>HSSFWorkbook(POIFSFileSystem fs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>Constructs a new HSSFWorkbook object using a POIFSFileSystem object.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>HSSFWorkbook(POIFSFileSystem fs, boolean preserveNodes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Given a POIFSFileSystem object, it creates a new HSSFWorkbook object to read a specified workbook.</td>
</tr>
</tbody>
</table>

The frequently used parameters inside these constructors are:

- **directory** : It is the POI filesystem directory to process from.
- **fs** : It is the POI filesystem that contains the workbook stream.
- **preservenodes**: This is an optional parameter that decides whether to preserve other nodes like macros. It consumes a lot of memory as it stores all the POIFileSystem in memory (if set).

**Note:** The HSSFWorkbook class contains a number of methods; however they are compatible with xls format only. In this tutorial, the focus is on the latest version of Excel file formats. Hence, the class methods of HSSFWorkbook are not listed here. If you require these class methods, then refer POI-HSSFWorkbook class API at [https://poi.apache.org/apidocs/org/apache/poi/hssf/usermodel/HSSFWorkbook.html](https://poi.apache.org/apidocs/org/apache/poi/hssf/usermodel/HSSFWorkbook.html).

**XSSFWorkbook**

It is a class that is used to represent both high and low level Excel file formats. It belongs to the `org.apache.xssf.usermodel` package and implements the `Workbook` interface. Listed below are the methods and constructors under this class.

**Class Constructors**

<table>
<thead>
<tr>
<th>S. No.</th>
<th>Constructor and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>XSSFWorkbook()</code>&lt;br&gt;Creates a new XSSFWorkbook object from scratch.</td>
</tr>
<tr>
<td>2</td>
<td><code>XSSFWorkbook(java.io.File file)</code>&lt;br&gt;Constructs an XSSFWorkbook object from a given file.</td>
</tr>
<tr>
<td>3</td>
<td><code>XSSFWorkbook(java.io.InputStream is)</code>&lt;br&gt;Constructs an XSSFWorkbook object, by buffering the whole input stream into memory and then opening an OPCPackage object for it.</td>
</tr>
<tr>
<td>4</td>
<td><code>XSSFWorkbook(java.lang.String path)</code>&lt;br&gt;Constructs an XSSFWorkbook object given the full path of a file.</td>
</tr>
</tbody>
</table>
## Class Methods

<table>
<thead>
<tr>
<th>S. No.</th>
<th>Method and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>createSheet()</code></td>
</tr>
<tr>
<td></td>
<td>Creates an XSSFSheet for this workbook, adds it to the sheets, and returns the high level representation.</td>
</tr>
<tr>
<td>2</td>
<td><code>createSheet(java.lang.String sheetname)</code></td>
</tr>
<tr>
<td></td>
<td>Creates a new sheet for this Workbook and returns the high level representation.</td>
</tr>
<tr>
<td>3</td>
<td><code>createFont()</code></td>
</tr>
<tr>
<td></td>
<td>Creates a new font and adds it to the workbook's font table.</td>
</tr>
<tr>
<td>4</td>
<td><code>createCellStyle()</code></td>
</tr>
<tr>
<td></td>
<td>Creates a new XSSFFCellStyle and adds it to the workbook's style table.</td>
</tr>
<tr>
<td>5</td>
<td><code>createFont()</code></td>
</tr>
<tr>
<td></td>
<td>Creates a new font and adds it to the workbook's font table.</td>
</tr>
<tr>
<td>6</td>
<td><code>setPrintArea(int sheetIndex, int startColumn, int endColumn, int startRow, int endRow)</code></td>
</tr>
<tr>
<td></td>
<td>Sets the print area of a given sheet as per the specified parameters.</td>
</tr>
</tbody>
</table>

For the remaining methods of this class, refer the complete API document at: [http://poi.apache.org/apidocs/org/apache/poi/xssf/usermodel/XSSFWorkbook.html](http://poi.apache.org/apidocs/org/apache/poi/xssf/usermodel/XSSFWorkbook.html) for the complete list of methods.

## Sheet

Sheet is an interface under the `org.apache.poi.ss.usermodel` package and it is a super-interface of all classes that create high or low level spreadsheets with specific
names. The most common type of spreadsheet is worksheet, which is represented as a grid of cells.

**HSSFSheet**

This is a class under the `org.apache.poi.hssf.usermodel` package. It can create excel spreadsheets and it allows to format the sheet style and sheet data.

**Class Constructors**

<table>
<thead>
<tr>
<th>S. No.</th>
<th>Constructor and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>HSSFSheet(HSSFWorkbook workbook)</code>&lt;br&gt;Creates new HSSFSheet called by HSSFWorkbook to create a sheet from scratch.</td>
</tr>
<tr>
<td>2</td>
<td><code>HSSFSheet(HSSFWorkbook workbook, InternalSheet sheet)</code>&lt;br&gt;Creates an HSSFSheet representing the given sheet object.</td>
</tr>
</tbody>
</table>

**XSSFSheet**

This is a class which represents high level representation of excel spreadsheet. It is under `org.apache.poi.hssf.usermodel` package.

**Class Constructors**

<table>
<thead>
<tr>
<th>S. No.</th>
<th>Constructor and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>XSSFSheet()</code>&lt;br&gt;Creates new XSSFSheet - called by XSSFWorkbook to create a sheet from scratch.</td>
</tr>
<tr>
<td>2</td>
<td><code>XSSFSheet(PackagePart part, PackageRelationship rel)</code>&lt;br&gt;Creates an XSSFSheet representing the given package part and relationship.</td>
</tr>
</tbody>
</table>
Class Methods

<table>
<thead>
<tr>
<th>S. No.</th>
<th>Methods and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>addMergedRegion(CellRangeAddress region)</code>&lt;br&gt;Adds a merged region of cells (hence those cells form one).</td>
</tr>
<tr>
<td>2</td>
<td><code>autoSizeColumn(int column)</code>&lt;br&gt;Adjusts the column width to fit the contents.</td>
</tr>
<tr>
<td>3</td>
<td><code>iterator()</code>&lt;br&gt;This method is an alias for <code>rowIterator()</code> to allow foreach loops</td>
</tr>
<tr>
<td>4</td>
<td><code>addHyperlink(XSSFHyperlink hyperlink)</code>&lt;br&gt;Registers a hyperlink in the collection of hyperlinks on this sheet</td>
</tr>
</tbody>
</table>

For the remaining methods of this class, refer the complete API at: [https://poi.apache.org/apidocs/org/apache/poi/xssf/usermodel/XSSFSheet.html](https://poi.apache.org/apidocs/org/apache/poi/xssf/usermodel/XSSFSheet.html).

Row

This is an interface under the `org.apache.poi.ss.usermodel` package. It is used for high-level representation of a row of a spreadsheet. It is a super-interface of all classes that represent rows in POI library.

XSSFRow

This is a class under the `org.apache.poi.xssf.usermodel` package. It implements the Row interface, therefore it can create rows in a spreadsheet. Listed below are the methods and constructors under this class.

Class Methods

<table>
<thead>
<tr>
<th>S. No.</th>
<th>Methods and Description</th>
</tr>
</thead>
</table>
1. **createCell**(int columnIndex)
   Creates new cells within the row and returns it.

2. **setHeight**(short height)
   Sets the height in short units.

For the remaining methods of this class, follow the given link


**Cell**

This is an interface under the org.apache.poi.ss.usermodel package. It is a super-interface of all classes that represent cells in the rows of a spreadsheet.

Cells can take various attributes such as blank, numeric, date, error, etc. Cells should have their own numbers (0 based) before being added to a row.

**XSSFCell**

This is a class under the org.apache.poi.xssf.usermodel package. It implements the Cell interface. It is a high-level representation of cells in the rows of a spreadsheet.

**Field Summary**

Listed below are some of the fields of the XSSFCell class along with their description.

<table>
<thead>
<tr>
<th>Cell Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CELL_TYPE_BLANK</td>
<td>Represents blank cell</td>
</tr>
<tr>
<td>CELL_TYPE_BOOLEAN</td>
<td>Represents Boolean cell (true or false)</td>
</tr>
<tr>
<td>CELL_TYPE_ERROR</td>
<td>Represents error value on a cell</td>
</tr>
<tr>
<td>CELL_TYPE_FORMULA</td>
<td>Represents formula result on a cell</td>
</tr>
<tr>
<td>S. No.</td>
<td>Methods and Description</td>
</tr>
<tr>
<td>-------</td>
<td>-------------------------</td>
</tr>
</tbody>
</table>
| 1     | setCellStyle(CellStyle style)  
Sets the style for the cell. |
| 2     | setCellType(int cellType)  
Sets the type of cells (numeric, formula, or string). |
| 3     | setCellValue(boolean value)  
Sets a boolean value for the cell. |
| 4     | setCellValue(java.util.Calendar value)  
Sets a date value for the cell. |
| 5     | setCellValue(double value)  
Sets a numeric value for the cell. |
| 6     | setCellValue(java.lang.String str)  
Sets a string value for the cell. |
| 7     | setHyperlink(Hyperlink hyperlink)  
Assigns a hyperlink to this cell. |

For the remaining methods and fields of this class, visit the following link:

XSSFCellStyle

This is a class under the `org.apache.poi.xssf.usermodel` package. It will provide possible information regarding the format of the content in a cell of a spreadsheet. It also provides options for modifying that format. It implements the CellStyle interface.

Field Summary

The following table lists a few fields that are inherited from the CellStyle interface.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALIGN_CENTER</td>
<td>Center align the cell contents</td>
</tr>
<tr>
<td>ALIGN_CENTER_SELECTION</td>
<td>Center-selection horizontal alignment</td>
</tr>
<tr>
<td>ALIGN_FILL</td>
<td>Cell fit to the content size</td>
</tr>
<tr>
<td>ALIGN_JUSTIFY</td>
<td>Fit cell contents to its width</td>
</tr>
<tr>
<td>ALIGN_LEFT</td>
<td>Left align the cell contents</td>
</tr>
<tr>
<td>ALIGN_RIGHT</td>
<td>Right align the cell contents</td>
</tr>
<tr>
<td>BORDER_DASH_DOT</td>
<td>Cell style with dash and dot</td>
</tr>
<tr>
<td>BORDER_DOTTED</td>
<td>Cell style with dotted border</td>
</tr>
<tr>
<td>BORDER_DASHED</td>
<td>Cell style with dashed border</td>
</tr>
<tr>
<td>BORDER_THICK</td>
<td>Cell style with thick border</td>
</tr>
<tr>
<td>BORDER_THIN</td>
<td>Cell style with thin border</td>
</tr>
<tr>
<td>VERTICAL_BOTTOM</td>
<td>Align the cell contents vertical bottom</td>
</tr>
<tr>
<td>VERTICAL_CENTER</td>
<td>Align the cell contents vertical center</td>
</tr>
<tr>
<td>S. No.</td>
<td>Constructor and Description</td>
</tr>
<tr>
<td>------</td>
<td>-----------------------------</td>
</tr>
</tbody>
</table>
| 1    | XSSFCelStyle(int cellXfId, int cellStyleXfId, StylesTable stylesSource, ThemesTable theme)  
      | Creates a cell style from the supplied parts |
| 2    | XSSFCelStyle(StylesTable stylesSource)  
      | Creates an empty cell Style |

### Class Methods

<table>
<thead>
<tr>
<th>S. No.</th>
<th>Method and Description</th>
</tr>
</thead>
</table>
| 1    | setAlignment(short align)  
      | Sets the type of horizontal alignment for the cell |
| 2    | setBorderBottom(short border)  
      | Sets the type of border for the bottom border of the cell |
| 3    | setBorderColor(XSSFCelBorder.BorderSide side, XSSFCelColor color)  
      | Sets the color for the selected border |
| 4    | setBorderLeft(Short border)  
      | Sets the type of border for the left border of the cell |
| 5    | setBorderRight(short border)  
      | Sets the type of border for the right border of the cell |
For the remaining methods and fields in this class, go through the following link:


**HSSFColor**

This is a class under the `org.apache.poi.hssf.util` package. It provides different colors as nested classes. Usually these nested classes are represented by using their own indexes. It implements the Color interface.

**Nested classes**

All nested classes of this class are static and each class has its index. These nested color classes are used for cell formatting such as cell content, border, foreground, and background. Listed below are some of the nested classes.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>setBorderTop(short border)</code></td>
<td>Sets the type of border for the top border of the cell</td>
</tr>
<tr>
<td><code>setFillBackgroundColor(XSSFColor color)</code></td>
<td>Sets the background fill color represented as an XSSFColor value.</td>
</tr>
<tr>
<td><code>setFillForegroundColor(XSSFColor color)</code></td>
<td>Sets the foreground fill color represented as an XSSFColor value.</td>
</tr>
<tr>
<td><code>setFillPattern(short fp)</code></td>
<td>Specifies the cell fill information for pattern and solid color cell fills.</td>
</tr>
<tr>
<td><code>setFont(Font font)</code></td>
<td>Sets the font for this style.</td>
</tr>
<tr>
<td><code>setRotation(short rotation)</code></td>
<td>Sets the degree of rotation for the text in the cell.</td>
</tr>
<tr>
<td><code>setVerticalAlignment(short align)</code></td>
<td>Sets the type of vertical alignment for the cell.</td>
</tr>
<tr>
<td>S. No.</td>
<td>Class names (colors)</td>
</tr>
<tr>
<td>-------</td>
<td>---------------------</td>
</tr>
<tr>
<td>1</td>
<td>HSSFCOLOR.AQUA</td>
</tr>
<tr>
<td>2</td>
<td>HSSFCOLOR.AUTOMATIC</td>
</tr>
<tr>
<td>3</td>
<td>HSSFCOLOR.BLACK</td>
</tr>
<tr>
<td>4</td>
<td>HSSFCOLOR.BLUE</td>
</tr>
<tr>
<td>5</td>
<td>HSSFCOLOR.BRIGHT_GREEN</td>
</tr>
<tr>
<td>6</td>
<td>HSSFCOLOR.BRIGHT_GRAY</td>
</tr>
<tr>
<td>7</td>
<td>HSSFCOLOR.CORAL</td>
</tr>
<tr>
<td>8</td>
<td>HSSFCOLOR.DARK_BLUE</td>
</tr>
<tr>
<td>9</td>
<td>HSSFCOLOR.DARK_GREEN</td>
</tr>
<tr>
<td>10</td>
<td>HSSFCOLOR.SKY_BLUE</td>
</tr>
<tr>
<td>11</td>
<td>HSSFCOLOR.WHITE</td>
</tr>
<tr>
<td>12</td>
<td>HSSFCOLOR.YELLOW</td>
</tr>
</tbody>
</table>

**Class Methods**

Only one method of this class is important and that is used to get the index value.

<table>
<thead>
<tr>
<th>S. No.</th>
<th>Method and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>getIndex()</code></td>
</tr>
<tr>
<td></td>
<td>This method is used to get the index value of a nested class.</td>
</tr>
</tbody>
</table>

For the remaining methods and nested classes, refer the following link:
XSSFColor

This is a class under the `org.apache.poi.xssf.usermodel` package. It is used to represent color in a spreadsheet. It implements the Color interface. Listed below are some of its methods and constructors.

### Class Constructors

<table>
<thead>
<tr>
<th>S. No.</th>
<th>Constructor and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>XSSFColor()</code>&lt;br&gt;Creates a new instance of XSSFColor.</td>
</tr>
<tr>
<td>2</td>
<td><code>XSSFColor(byte[] rgb)</code>&lt;br&gt;Creates a new instance of XSSFColor using RGB.</td>
</tr>
<tr>
<td>3</td>
<td><code>XSSFColor(java.awt.Color clr)</code>&lt;br&gt;Creates a new instance of XSSFColor using the Color class from the awt package.</td>
</tr>
</tbody>
</table>

### Class Methods

<table>
<thead>
<tr>
<th>S. No.</th>
<th>Method and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>setAuto(boolean auto)</code>&lt;br&gt;Sets a boolean value to indicate that the ctColor is automatic and the system ctColor is dependent.</td>
</tr>
<tr>
<td>2</td>
<td><code>setIndexed(int indexed)</code>&lt;br&gt;Sets indexed ctColor value as system ctColor.</td>
</tr>
</tbody>
</table>
For the remaining methods, visit the following link:


**XSSFFont**
This is a class under the `org.apache.poi.xssf.usermodel` package. It implements the Font interface and therefore it can handle different fonts in a workbook.

### Class Constructor

<table>
<thead>
<tr>
<th>S. No.</th>
<th>Constructor and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>XSSFFont()</td>
</tr>
<tr>
<td></td>
<td>Creates a new XSSFont instance.</td>
</tr>
</tbody>
</table>

### Class Methods

<table>
<thead>
<tr>
<th>S. No.</th>
<th>Method and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>setBold(boolean bold)</td>
</tr>
<tr>
<td></td>
<td>Sets a Boolean value for the 'bold' attribute.</td>
</tr>
<tr>
<td>2</td>
<td>setColor(short color)</td>
</tr>
<tr>
<td></td>
<td>Sets the indexed color for the font.</td>
</tr>
<tr>
<td>3</td>
<td>setColor(XSSFColor color)</td>
</tr>
<tr>
<td></td>
<td>Sets the color for the font in Standard Alpha RGB color value.</td>
</tr>
<tr>
<td>4</td>
<td>setFontHeight(short height)</td>
</tr>
<tr>
<td></td>
<td>Sets the font height in points.</td>
</tr>
<tr>
<td>5</td>
<td>setFontName(java.lang.String name)</td>
</tr>
<tr>
<td></td>
<td>Sets the name for the font.</td>
</tr>
</tbody>
</table>
6. **setItalic(boolean italic)**
   Sets a Boolean value for the 'italic' property.

For the remaining methods, go through the following link:


---

**XSSFHyperlink**

This is a class under the org.apache.poi.xssf.usermodel package. It implements the Hyperlink interface. It is used to set a hyperlink to the cell contents of a spreadsheet.

**Fields**

The fields of this class are as follows. Here, fields mean the types of hyperlinks used.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LINK_DOCUMENT</td>
<td>Used to link any other document</td>
</tr>
<tr>
<td>LINK_EMAIL</td>
<td>Used to link email</td>
</tr>
<tr>
<td>LINK_FILE</td>
<td>Used to link any other file in any format</td>
</tr>
<tr>
<td>LINK_URL</td>
<td>Used to link a web URL</td>
</tr>
</tbody>
</table>

**Class Methods**

<table>
<thead>
<tr>
<th>S. No.</th>
<th>Method and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>setAddress(java.lang.String address)</td>
</tr>
<tr>
<td></td>
<td>Hyperlink address.</td>
</tr>
</tbody>
</table>
For the remaining methods, visit the following link:


XSSFCreationHelper
This is a class under the `org.apache.poi.xssf.usermodel` package. It implements the CreationHelper interface. It is used as a support class for formula evaluation and setting up hyperlinks.

Class methods

<table>
<thead>
<tr>
<th>S. No.</th>
<th>Method and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>createFormulaEvaluator()</code>&lt;br&gt;Creates an XSSFFormulaEvaluator instance, the object that evaluates formula cells.</td>
</tr>
<tr>
<td>2</td>
<td><code>createHyperlink(int type)</code>&lt;br&gt;Creates a new XSSFHyperlink.</td>
</tr>
</tbody>
</table>

For the remaining methods, refer the following link:


XSSFPrintSetup
This is a class under the `org.apache.poi.xssf.usermodel` package. It implements the `PrintSetup` interface. It is used to set print page size, area, options, and settings.
### Class Methods

<table>
<thead>
<tr>
<th>S. No.</th>
<th>Method and Description</th>
</tr>
</thead>
</table>
| 1      | `setLandscape(boolean ls)`
|        | Sets a boolean value to allow or block landscape printing. |
| 2      | `setLeftToRight(boolean ltor)`
|        | Sets whether to go left to right or top down in ordering while printing. |
| 3      | `setPaperSize(short size)`
|        | Sets the paper size. |

For the remaining methods, visit the following link:

Here the term 'Workbook' means Microsoft Excel file. After completion of this chapter, you will be able to create new Workbooks and open existing Workbooks with your Java program.

**Create Blank Workbook**

The following simple program is used to create a blank Microsoft Excel Workbook.

```java
import java.io.*;
import org.apache.poi.xssf.usermodel.*;

public class CreateWorkBook {
    public static void main(String[] args) throws Exception {
        // Create Blank workbook
        XSSFWorkbook workbook = new XSSFWorkbook();
        // Create file system using specific name
        FileOutputStream out = new FileOutputStream(new File("createworkbook.xlsx"));
        // Write operation workbook using file out object
        workbook.write(out);
        out.close();
        System.out.println("createworkbook.xlsx written successfully");
    }
}
```
Let us save the above Java code as `CreateWorkBook.java`, and then compile and execute it from the command prompt as follows:

```shell
$javac  CreateWorkBook.java
$java CreateWorkBook
```

If your system environment is configured with the POI library, it will compile and execute to generate the blank Excel file named `createworkbook.xlsx` in your current directory and display the following output in the command prompt.

```text
createworkbook.xlsx written successfully
```

### Open Existing Workbook

Use the following code to open an existing workbook.

```java
import java.io.*;
import org.apache.poi.xssf.usermodel.*;

public class OpenWorkBook {
    public static void main(String args[]) throws Exception {
        File file = new File("openworkbook.xlsx");
        FileInputStream fIP = new FileInputStream(file);

        //Get the workbook instance for XLSX file
        XSSFFormat workbook = new XSSFFormat(fIP);
    }
}
```
if (file.isFile() && file.exists())
{
    System.out.println("openworkbook.xlsx file open successfully.");
}
else
{
    System.out.println("Error to open openworkbook.xlsx file.");
}

Save the above Java code as **OpenWorkBook.java**, and then compile and execute it from the command prompt as follows:

```bash
$javac OpenWorkBook.java
$java OpenWorkBook
```

It will compile and execute to generate the following output.

```bash
openworkbook.xlsx file open successfully.
```

After opening a workbook, you can perform read and write operations on it.
This chapter explains how to create a spreadsheet and manipulate it using Java. Spreadsheet is a page in an Excel file; it contains rows and columns with specific names.

After completing this chapter, you will be able to create a spreadsheet and perform read operations on it.

Create a Spreadsheet

First of all, let us create a spreadsheet using the referenced classes discussed in the earlier chapters. By following the previous chapter, create a workbook first and then we can go on and create a sheet.

The following code snippet is used to create a spreadsheet.

```java
//Create Blank workbook
XSSFWorkbook workbook = new XSSFWorkbook();
//Create a blank spreadsheet
XSSFSheet spreadsheet = workbook.createSheet("Sheet Name");
```

Rows on Spreadsheet

Spreadsheets have a grid layout. The rows and columns are identified with specific names. The columns are identified with alphabets and rows with numbers.

The following code snippet is used to create a row.
Write into a Spreadsheet
Let us consider an example of employee data. Here the employee data is given in a tabular form.

<table>
<thead>
<tr>
<th>Emp Id</th>
<th>Emp Name</th>
<th>Designation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tp01</td>
<td>Gopal</td>
<td>Technical Manager</td>
</tr>
<tr>
<td>Tp02</td>
<td>Manisha</td>
<td>Proof Reader</td>
</tr>
<tr>
<td>Tp03</td>
<td>Masthan</td>
<td>Technical Writer</td>
</tr>
<tr>
<td>Tp04</td>
<td>Satish</td>
<td>Technical Writer</td>
</tr>
<tr>
<td>Tp05</td>
<td>Krishna</td>
<td>Technical Writer</td>
</tr>
</tbody>
</table>

The following code is used to write the above data into a spreadsheet.

```java
import java.io.File;
import java.io.FileOutputStream;
import java.util.Map;
import java.util.Set;
import java.util.TreeMap;
import org.apache.poi.ss.usermodel.Cell;
import org.apache.poi.xssf.usermodel.XSSFRow;
import org.apache.poi.xssf.usermodel.XSSFSheet;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;

public class Writesheet
{
    public static void main(String[] args) throws Exception
    {
```
//Create blank workbook
XSSFWorkbook workbook = new XSSFWorkbook();

//Create a blank sheet
XSSFSheet spreadsheet = workbook.createSheet(" Employee Info ");

//Create row object
XSSFRow row;

//This data needs to be written (Object[])
Map < String, Object[] > empinfo =
new TreeMap < String, Object[] >();
empinfo.put("1", new Object[] { "EMP ID", "EMP NAME", "DESIGNATION" });
empinfo.put("2", new Object[] { "tp01", "Gopal", "Technical Manager" });
empinfo.put("3", new Object[] { "tp02", "Manisha", "Proof Reader" });
empinfo.put("4", new Object[] { "tp03", "Masthan", "Technical Writer" });
empinfo.put("5", new Object[] { "tp04", "Satish", "Technical Writer" });
empinfo.put("6", new Object[] { "tp05", "Krishna", "Technical Writer" });

//Iterate over data and write to sheet
Set < String > keyid = empinfo.keySet();
int rowid = 0;
for (String key : keyid)
{
    row = spreadsheet.createRow(rowid++);
    Object [] objectArr = empinfo.get(key);
    int cellid = 0;
    for (Object obj : objectArr)
Save the above Java code as **Writesheet.java**, and then compile and run it from the command prompt as follows:

```
$javac Writesheet.java
$java Writesheet
```

It will compile and execute to generate an Excel file named **Writesheet.xlsx** in your current directory and you will get the following output in the command prompt.

```
Writesheet.xlsx written successfully
```

The **Writesheet.xlsx** file looks as follows.
Read from a Spreadsheet
Let us consider the above excel file named *Writsheet.xlsx* as input. Observe the following code; it is used for reading the data from a spreadsheet.

```java
import java.io.File;
import java.io.FileInputStream;
import java.util.Iterator;
import org.apache.poi.ss.usermodel.Cell;
import org.apache.poi.ss.usermodel.Row;
import org.apache.poi.xssf.usermodel.XSSFRow;
import org.apache.poi.xssf.usermodel.XSSFSheet;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;

public class Readsheet
{
    static XSSFRow row;
```
public static void main(String[] args) throws Exception {
    FileInputStream fis = new FileInputStream(new File("WriteSheet.xlsx");
    XSSFWorkbook workbook = new XSSFWorkbook(fis);
    XSSFSheet spreadsheet = workbook.getSheetAt(0);
    Iterator<Row> rowIterator = spreadsheet.iterator();
    while (rowIterator.hasNext())
    {
        row = (XSSFRow) rowIterator.next();
        Iterator<Cell> cellIterator = row.cellIterator();
        while (cellIterator.hasNext())
        {
            Cell cell = cellIterator.next();
            switch (cell.getCellType())
            {
                case Cell.CELL_TYPE_NUMERIC:
                    System.out.print(
                        cell.getNumericCellValue() + " \t \t ");
                    break;
                case Cell.CELL_TYPE_STRING:
                    System.out.print(
                        cell.getStringCellValue() + " \t \t ");
                    break;
            }
        }
        System.out.println();
    }
    fis.close();
}

Let us keep the above code in Readsheet.java file, and then compile and run it from the command prompt as follows:
$javac Readsheet.java
$java Readsheet

If your system environment is configured with the POI library, it will compile and execute to generate the following output in the command prompt.

<table>
<thead>
<tr>
<th>EMP ID</th>
<th>EMP NAME</th>
<th>DESIGNATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>tp01</td>
<td>Gopal</td>
<td>Technical Manager</td>
</tr>
<tr>
<td>tp02</td>
<td>Manisha</td>
<td>Proof Reader</td>
</tr>
<tr>
<td>tp03</td>
<td>Masthan</td>
<td>Technical Writer</td>
</tr>
<tr>
<td>tp04</td>
<td>Satish</td>
<td>Technical Writer</td>
</tr>
<tr>
<td>tp05</td>
<td>Krishna</td>
<td>Technical Writer</td>
</tr>
</tbody>
</table>
Any data that you enter into a spreadsheet is always stored in a cell. We use the labels of rows and columns to identify a cell. This chapter describes how to manipulate data in cells in a spreadsheet using Java programming.

Create a Cell
You need to create a row before creating a cell. A row is nothing but a collection of cells.

The following code snippet is used for creating a cell.

```java
//create new workbook
XSSFWorkbook workbook = new XSSFWorkbook();
//create spreadsheet with a name
XSSFSheet spreadsheet = workbook.createSheet("new sheet");
//create first row on a created spreadsheet
XSSFRow row = spreadsheet.createRow(0);
//create first cell on created row
XSSFCell cell = row.createCell(0);
```

Types of Cells
The cell type specifies whether a cell can contain strings, numeric value, or formulas. A string cell cannot hold numeric values and a numeric cell cannot hold strings. Given below are the types of cells, their values, and type syntax.
<table>
<thead>
<tr>
<th>Type of cell value</th>
<th>Type Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank cell value</td>
<td>XSSFCell.CELL_TYPE_BLANK</td>
</tr>
<tr>
<td>Boolean cell value</td>
<td>XSSFCell.CELL_TYPE_BOOLEAN</td>
</tr>
<tr>
<td>Error cell value</td>
<td>XSSFCell.CELL_TYPE_ERROR</td>
</tr>
<tr>
<td>Numeric cell value</td>
<td>XSSFCell.CELL_TYPE_NUMERIC</td>
</tr>
<tr>
<td>String cell value</td>
<td>XSSFCell.CELL_TYPE_STRING</td>
</tr>
</tbody>
</table>

The following code is used to create different types of cells in a spreadsheet.

```java
import java.io.File;
import java.io.FileOutputStream;
import java.util.Date;
import org.apache.poi.xssf.usermodel.XSSFCell;
import org.apache.poi.xssf.usermodel.XSSFRow;
import org.apache.poi.xssf.usermodel.XSSFSheet;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;

public class TypesofCells
{
    public static void main(String[] args) throws Exception
    {
        XSSFWorkbook workbook = new XSSFWorkbook();
        XSSFSheet spreadsheet = workbook.createSheet("cell types");
        XSSFRow row = spreadsheet.createRow((short) 2);
        row.createCell(0).setCellValue("Type of Cell");
        row.createCell(1).setCellValue("cell value");
        row = spreadsheet.createRow((short) 3);
        row.createCell(0).setCellValue("set cell type BLANK");
        row.createCell(1);
    }
}```
Apache POI Tutorial

Save the above code in a file named `TypesofCells.java`, compile and execute it from the command prompt as follows.

```
$javac TypesofCells.java
$java TypesofCells
```
If your system is configured with the POI library, then it will compile and execute to generate an Excel file named `typesofcells.xlsx` in your current directory and display the following output.

```
typesofcells.xlsx written successfully
```

The `typesofcells.xlsx` file looks as follows.

**Cell Styles**
Here you can learn how to do cell formatting and apply different styles such as merging adjacent cells, adding borders, setting cell alignment and filling with colors.

The following code is used to apply different styles to cells using Java programming.
import java.io.File;
import java.io.FileOutputStream;
import org.apache.poi.hssf.util.HSSFColor;
import org.apache.poi.ss.usermodel.IndexedColors;
import org.apache.poi.ss.usermodel.CellRangeAddress;
import org.apache.poi.xssf.usermodel.XSSFCell;
import org.apache.poi.xssf.usermodel.XSSFCellStyle;
import org.apache.poi.xssf.usermodel.XSSFRow;
import org.apache.poi.xssf.usermodel.XSSFSheet;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;

public class CellStyle
{
    public static void main(String[] args) throws Exception
    {
        XSSFWorkbook workbook = new XSSFWorkbook();
        XSSFSheet spreadsheet = workbook.createSheet("cellstyle");
        XSSFRow row = spreadsheet.createRow((short) 1);
        row.setHeight((short) 800);
        XSSFCell cell = (XSSFCell) row.createCell((short) 1);
        cell.setCellValue("test of merging");

        //MERGING CELLS
        //this statement for merging cells
        spreadsheet.addMergedRegion(new CellRangeAddress(
            1, //first row (0-based)
            1, //last row  (0-based)
            1, //first column (0-based)
            4  //last column  (0-based)
        ));

        //CELL Alignment
row = spreadsheet.createRow(5);
cell = (XSSFCell) row.createCell(0);
row.setHeight((short) 800);

// Top Left alignment
XSSFCellStyle style1 = workbook.createCellStyle();
spreadsheet.setColumnWidth(0, 8000);
style1.setAlignment(XSSFCellStyle.ALIGN_LEFT);
style1.setVerticalAlignment(XSSFCellStyle.VERTICAL_TOP);
cell.setCellValue("Top Left");
cell.setCellStyle(style1);

row = spreadsheet.createRow(6);
cell = (XSSFCell) row.createCell(1);
row.setHeight((short) 800);

// Center Align Cell Contents
XSSFCellStyle style2 = workbook.createCellStyle();
style2.setAlignment(XSSFCellStyle.ALIGN_CENTER);
style2.setVerticalAlignment(XSSFCellStyle.VERTICAL_CENTER);
cell.setCellValue("Center Aligned");
cell.setCellStyle(style2);

row = spreadsheet.createRow(7);
cell = (XSSFCell) row.createCell(2);
row.setHeight((short) 800);

// Bottom Right alignment
XSSFCellStyle style3 = workbook.createCellStyle();
style3.setAlignment(XSSFCellStyle.ALIGN_RIGHT);
style3.setVerticalAlignment(XSSFCellStyle.VERTICAL_TOP);
XSSFCellStyle.VERTICAL_BOTTOM);
cell.setCellValue("Bottom Right");
cell.setCellStyle(style3);

row = spreadsheet.createRow(8);
cell = (XSSFCell) row.createCell(3);

// Justified Alignment
XSSFCellStyle style4 = workbook.createCellStyle();
style4.setAlignment(XSSFCellStyle.ALIGN_JUSTIFY);
style4.setVerticalAlignment(XSSFCellStyle.VERTICAL_JUSTIFY);
cell.setCellValue("Contents are Justified in Alignment");
cell.setCellStyle(style4);

//CELL BORDER
row = spreadsheet.createRow((short) 10);
row.setHeight((short) 800);
cell = (XSSFCell) row.createCell((short) 1);
cell.setCellValue("BORDER");
XSSFCellStyle style5 = workbook.createCellStyle();
style5.setBorderBottom(XSSFCellStyle.BORDER_THICK);
style5.setBottomBorderColor(IndexedColors.BLUE.getIndex());
style5.setBorderLeft(XSSFCellStyle.BORDER_DOUBLE);
style5.setLeftBorderColor(IndexedColors.GREEN.getIndex());
style5.setBorderRight(XSSFCellStyle.BORDER_HAIR);
style5.setRightBorderColor(IndexedColors.RED.getIndex());
style5.setBorderTop(XSSFCellStyle.BIG_SPOTS);
style5.setTopBorderColor(IndexedColors.ORANGE.getIndex());
IndexedColors.CORAL.getIndex());

    cell.setCellStyle(style5);

    // Fill Colors
    //background color
    row = spreadsheet.createRow((short) 10);
    cell = (XSSFCell) row.createCell((short) 1);
    XSSFCelStyle style6 = workbook.createCellStyle();
    style6.setFillBackgroundColor(HSSFColor.LEMON_CHIFFON.index);
    style6.setFillPattern(XSSFCellStyle.LESS_DOTS);
    style6.setAlignment(XSSFCellStyle.ALIGN_FILL);
    spreadsheet.setColumnWidth(1, 8000);
    cell.setCellValue("FILL BACKGROUND/FILL PATTERN");
    cell.setCellStyle(style6);

    //Foreground color
    row = spreadsheet.createRow((short) 12);
    cell = (XSSFCell) row.createCell((short) 1);
    XSSFCelStyle style7 = workbook.createCellStyle();
    style7.setFillForegroundColor(HSSFColor.BLUE.index);
    style7.setFillPattern(XSSFCellStyle.LESS_DOTS);
    style7.setAlignment(XSSFCellStyle.ALIGN_FILL);
    cell.setCellValue("FILL FOREGROUND/FILL PATTERN");
    cell.setCellStyle(style7);
    FileOutputStream out = new FileOutputStream(new File("cellstyle.xlsx"));
    workbook.write(out);
    out.close();
    System.out.println("cellstyle.xlsx written successfully");
}
Save the above code in a file named `CellStyle.java`, compile and execute it from the command prompt as follows.

```bash
$javac CellStyle.java
$java CellStyle
```

It will generate an Excel file named `cellstyle.xlsx` in your current directory and display the following output.

```
cellstyle.xlsx written successfully
```

The `cellstyle.xlsx` file looks as follows.
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>test of merging</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Top Left</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Center Aligned</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Bottom Right</td>
<td>Contents</td>
<td>are</td>
<td>Justified</td>
</tr>
<tr>
<td></td>
<td></td>
<td>justified</td>
<td>in</td>
<td>alignment</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>contents</td>
<td>are</td>
<td>justi</td>
</tr>
<tr>
<td>8</td>
<td>BORDER</td>
<td>borders</td>
<td>are</td>
<td>justified</td>
</tr>
<tr>
<td>9</td>
<td>FILL BACKGROUND/FILL PATTERN</td>
<td>borders</td>
<td>are</td>
<td>justified</td>
</tr>
<tr>
<td>10</td>
<td>FILL FOREGROUND/FILL PATTERN</td>
<td>borders</td>
<td>are</td>
<td>justified</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>borders</td>
<td>are</td>
<td>justified</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td>borders</td>
<td>are</td>
<td>justified</td>
</tr>
<tr>
<td>13</td>
<td></td>
<td>borders</td>
<td>are</td>
<td>justified</td>
</tr>
<tr>
<td>14</td>
<td></td>
<td>borders</td>
<td>are</td>
<td>justified</td>
</tr>
</tbody>
</table>
This chapter explains how to set different fonts, apply styles, and display text in different angles of direction in an Excel spreadsheet.

Every system comes bundled with a huge collection of fonts such as Arial, Impact, Times New Roman, etc. The collection can also be updated with new fonts, if required. Similarly there are various styles in which a font can be displayed, for example, bold, italic, underline, strike through, etc.

Fonts and Font Styles

The following code is used to apply a particular font and style to the contents of a cell.

```java
import java.io.File;
import java.io.FileOutputStream;
import org.apache.poi.hssf.util.HSSFColor;
import org.apache.poi.xssf.usermodel.XSSFCell;
import org.apache.poi.xssf.usermodel.XSSFCellStyle;
import org.apache.poi.xssf.usermodel.XSSFFont;
import org.apache.poi.xssf.usermodel.XSSFRow;
import org.apache.poi.xssf.usermodel.XSSFSheet;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;

public class FontStyle
```
public static void main(String[] args) throws Exception {

    XSSFWorkbook workbook = new XSSFWorkbook();
    XSSFSheet spreadsheet = workbook.createSheet("Fontstyle");
    XSSFRow row = spreadsheet.createRow(2);

    // Create a new font and alter it.
    XSSFFont font = workbook.createFont();
    font.setFontHeightInPoints((short) 30);
    font.setFontName("IMPACT");
    font.setItalic(true);
    font.setColor(HSSFColor.BRIGHT_GREEN.index);

    // Set font into style
    XSSFCellStyle style = workbook.createCellStyle();
    style.setFont(font);

    // Create a cell with a value and set style to it.
    XSSFFCell cell = row.createCell(1);
    cell.setCellValue("Font Style");
    cell.setCellStyle(style);

    FileOutputStream out = new FileOutputStream(new File("fontstyle.xlsx"));
    workbook.write(out);
    out.close();
    System.out.println("fontstyle.xlsx written successfully");
}
}
Let us save the above code in a file named `FontStyle.java`. Compile and execute it from the command prompt as follows.

```bash
$javac FontStyle.java
$java FontStyle
```

It generates an Excel file named `fontstyle.xlsx` in your current directory and display the following output on the command prompt.

`fontstyle.xlsx written successfully`

The `fontstyle.xlsx` file looks as follows.
Text Direction

Here you can learn how to set the text direction in different angles. Usually cell contents are displayed horizontally, from left to right, and at $0^\circ$ angle; however you can use the following code to rotate the text direction, if required.

```java
import java.io.File;
import java.io.FileOutputStream;
import org.apache.poi.xssf.usermodel.XSSFCell;
import org.apache.poi.xssf.usermodel.XSSFCellStyle;
import org.apache.poi.xssf.usermodel.XSSFRow;
import org.apache.poi.xssf.usermodel.XSSFSheet;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;

public class TextDirection {
    public static void main(String[] args) throws Exception {
        XSSFWorkbook workbook = new XSSFWorkbook();
        XSSFSheet spreadsheet = workbook.createSheet("Text direction");
        XSSFRow row = spreadsheet.createRow(2);
        XSSFCCellStyle myStyle = workbook.createCellStyle();
        myStyle.setRotation((short) 0);
        XSSFCell cell = row.createCell(1);
        cell.setCellValue("0D angle");
        cell.setCellStyle(myStyle);

        // 30 degrees
        myStyle = workbook.createCellStyle();
        myStyle.setRotation((short) 30);
        cell = row.createCell(3);
```
<table>
<thead>
<tr>
<th>Angle (Degrees)</th>
<th>Cell Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>&quot;30D angle&quot;</td>
</tr>
<tr>
<td>90</td>
<td>&quot;90D angle&quot;</td>
</tr>
<tr>
<td>120</td>
<td>&quot;120D angle&quot;</td>
</tr>
<tr>
<td>270</td>
<td>&quot;270D angle&quot;</td>
</tr>
<tr>
<td>360</td>
<td>&quot;360D angle&quot;</td>
</tr>
</tbody>
</table>

```java
cell.setCellValue("30D angle");
cell.setCellStyle(myStyle);

//90 degrees
myStyle=workbook.createCellStyle();
myStyle.setRotation((short) 90);

cell = row.createCell(5);
cell.setCellValue("90D angle");
cell.setCellStyle(myStyle);

//120 degrees
myStyle=workbook.createCellStyle();
myStyle.setRotation((short) 120);

Cell = row.createCell(7);
cell.setCellValue("120D angle");
cell.setCellStyle(myStyle);

//270 degrees
myStyle = workbook.createCellStyle();
myStyle.setRotation((short) 270);

cell = row.createCell(9);
cell.setCellValue("270D angle");
cell.setCellStyle(myStyle);

//360 degrees
myStyle=workbook.createCellStyle();
myStyle.setRotation((short) 360);

cell = row.createCell(12);
cell.setCellValue("360D angle");
cell.setCellStyle(myStyle);

FileOutputStream out = new FileOutputStream(
```
new File("textdirection.xlsx");
workbook.write(out);
out.close();
System.out.println(
    "textdirection.xlsx written successfully");
}
}

Keep the above code in TextDirection.java file, then compile and execute it from the command prompt as follows.

$javac TextDirection.java
$java TextDirection

It will compile and execute to generate an Excel file named textdirection.xlsx in your current directory and display the following output on the command prompt.

textdirection.xlsx written successfully

The textdirection.xlsx file looks as follows.
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>J</th>
<th>K</th>
<th>L</th>
<th>M</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0°</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0° angle</td>
<td>30° angle</td>
<td>90° angle</td>
<td>120° angle</td>
<td>150° angle</td>
<td>180° angle</td>
<td>210° angle</td>
<td>240° angle</td>
<td>270° angle</td>
<td>300° angle</td>
<td>330° angle</td>
<td>360° angle</td>
<td></td>
</tr>
</tbody>
</table>

Test direction: [Some text]
This chapter takes you through the process of applying different formulas on cells using Java programming. The basic purpose of Excel application is to maintain numerical data by applying formulas on it.

In a formula, we pass dynamic values or locations of the values in the Excel sheet. On executing this formula, you get the desired result. The following table lists a few basic formulas that are frequently used in Excel.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adding multiple numbers</td>
<td>=SUM(Loc1:Locn) or =SUM(n1,n2,)</td>
</tr>
<tr>
<td>Count</td>
<td>=COUNT(Loc1:Locn) or =COUNT(n1,n2,)</td>
</tr>
<tr>
<td>Power of two numbers</td>
<td>=POWER(Loc1,Loc2) or =POWER(number, power)</td>
</tr>
<tr>
<td>Max of multiple numbers</td>
<td>=MAX(Loc1:Locn) or =MAX(n1,n2,)</td>
</tr>
<tr>
<td>Product</td>
<td>=PRODUCT(Loc1:Locn) or =PRODUCT(n1,n2,)</td>
</tr>
<tr>
<td>Factorial</td>
<td>=FACT(Locn) or =FACT(number)</td>
</tr>
<tr>
<td>Absolute number</td>
<td>=ABS(Locn) or =ABS(number)</td>
</tr>
<tr>
<td>Today date</td>
<td>=TODAY()</td>
</tr>
<tr>
<td>Converts lowercase</td>
<td>=LOWER(Locn) or =LOWER(text)</td>
</tr>
<tr>
<td>Square root</td>
<td>=SQRT(Locn) or =SQRT(number)</td>
</tr>
</tbody>
</table>
The following code is used to add formulas to a cell and execute it.

```java
import java.io.File;
import java.io.FileOutputStream;
import org.apache.poi.xssf.usermodel.XSSFCell;
import org.apache.poi.xssf.usermodel.XSSFRow;
import org.apache.poi.xssf.usermodel.XSSFSheet;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;

public class Formula
{
    public static void main(String[] args) throws Exception
    {
        XSSFWorkbook workbook = new XSSFWorkbook();
        XSSFSheet spreadsheet = workbook.createSheet("formula");
        XSSFRow row = spreadsheet.createRow(1);
        XSSFCell cell = row.createCell(1);
        cell.setCellValue("A =");
        cell = row.createCell(2);
        cell.setCellValue(2);
        row = spreadsheet.createRow(2);
        cell = row.createCell(1);
        cell.setCellValue("B =");
        cell = row.createCell(2);
        cell.setCellValue(4);
        row = spreadsheet.createRow(3);
        cell = row.createCell(1);
        cell.setCellValue("Total =");
        cell = row.createCell(2);
        cell.setCellValue(6);
    }
}
```
// Create SUM formula
   cell.setCellType(XSSFCell.CELL_TYPE_FORMULA);
   cell.setCellFormula("SUM(C2:C3)");
   cell = row.createCell(3);
   cell.setCellValue("SUM(C2:C3)");

   row = spreadsheet.createRow(4);
   cell = row.createCell(1);
   cell.setCellValue("POWER =");
   cell = row.createCell(2);
   // Create POWER formula
   cell.setCellType(XSSFCell.CELL_TYPE_FORMULA);
   cell.setCellFormula("POWER(C2,C3)");
   cell = row.createCell(3);
   cell.setCellValue("POWER(C2,C3)");

   row = spreadsheet.createRow(5);
   cell = row.createCell(1);
   cell.setCellValue("MAX =");
   cell = row.createCell(2);
   // Create MAX formula
   cell.setCellType(XSSFCell.CELL_TYPE_FORMULA);
   cell.setCellFormula("MAX(C2,C3)");
   cell = row.createCell(3);
   cell.setCellValue("MAX(C2,C3)");

   row = spreadsheet.createRow(6);
   cell = row.createCell(1);
   cell.setCellValue("FACT =");
   cell = row.createCell(2);
// Create FACT formula
    cell.setCellType(XSSFCell.CELL_TYPE_FORMULA);
    cell.setCellFormula("FACT(C3)");
    cell = row.createCell(3);
    cell.setCellValue("FACT(C3)");

    row = spreadsheet.createRow(7);
    cell = row.createCell(1);
    cell.setCellValue("SQRT =");
    cell = row.createCell(2);

    // Create SQRT formula
    cell.setCellType(XSSFCell.CELL_TYPE_FORMULA);
    cell.setCellFormula("SQRT(C5)");
    cell = row.createCell(3);
    cell.setCellValue("SQRT(C5)");

    workbook.getCreationHelper()
        .createFormulaEvaluator()
        .evaluateAll();

    FileOutputStream out = new FileOutputStream(new File("formula.xlsx"));
    workbook.write(out);
    out.close();
    System.out.println("fromula.xlsx written successfully");}
$javac Formula.java
$java Formula

It will generate an Excel file named \texttt{formula.xlsx} in your current directory and display the following output on the command prompt.

\texttt{formula.xlsx written successfully}

The \texttt{formula.xlsx} file looks as follows.
This chapter explains how to add hyperlinks to the contents in a cell. Usually hyperlinks are used to access any web URL, email, or an external file.

The following code shows how to create hyperlinks on cells.

```java
import java.io.File;
import java.io.FileOutputStream;
import org.apache.poi.common.usermodel.Hyperlink;
import org.apache.poi.hssf.util.HSSFCOLOR;
import org.apache.poi.ss.usermodel.CreationHelper;
import org.apache.poi.xssf.usermodel.XSSFCell;
import org.apache.poi.xssf.usermodel.XSSFCSS;
import org.apache.poi.xssf.usermodel.XSSFFont;
import org.apache.poi.xssf.usermodel.XSSFHyperlink;
import org.apache.poi.xssf.usermodel.XSSFSheet;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;

public class HyperlinkEX {
    public static void main(String[] args) throws Exception {
        XSSFWorkbook workbook = new XSSFWorkbook();
        XSSFSheet spreadsheet = workbook
            .createSheet("Hyperlinks");
        XSSFCell cell;
    }
}
```
CreationHelper createHelper = workbook.getCreationHelper();
XSSFCellStyle hlinkstyle = workbook.createCellStyle();
XSSFFont hlinkfont = workbook.createFont();
hlinkfont.setUnderline(XSSFFont.U_SINGLE);
hlinkfont.setColor(HSSFColor.BLUE.index);
hlinkstyle.setFont(hlinkfont);

    //URL Link
    cell = spreadsheet.createRow(1)
            .createCell((short) 1);
    cell.setCellValue("URL Link");
    XSSFFormulaHyperlink link = (XSSFFormulaHyperlink)createHelper.createHyperlink(Hyperlink.LINK_URL);
    link.setAddress("http://www.tutorialspoint.com/");
    cell.setHyperlink((XSSFHyperlink) link);
    cell.setCellStyle(hlinkstyle);

    //Hyperlink to a file in the current directory
    cell = spreadsheet.createRow(2)
            .createCell((short) 1);
    cell.setCellValue("File Link");
    link = (XSSFFormulaHyperlink)createHelper.createHyperlink(Hyperlink.LINK_FILE);
    link.setAddress("cellstyle.xlsx");
    cell.setHyperlink(link);
    cell.setCellStyle(hlinkstyle);

    //e-mail link
    cell = spreadsheet.createRow(3)
            .createCell((short) 1);
    cell.setCellValue("Email Link");
link = (XSSFHyperlink)createHelper.createHyperlink(Hyperlink.LINK_EMAIL);
link.setAddress("mailto:contact@tutorialspoint.com?subject=Hyperlink");
cell.setHyperlink(link);
cell.setCellStyle(hlinkstyle);

FileOutputStream out = new FileOutputStream(new File("hyperlink.xlsx"));
workbook.write(out);
out.close();
System.out.println("hyperlink.xlsx written successfully");
}
The **hyperlink.xlsx** file looks as follows.

![Image of hyperlink.xlsx file]

- **URL Link**
- **File Link**
- **Email Link**
This chapter explains how to set the print area on a spreadsheet. The usual print area is from left top to right bottom on Excel spreadsheets. Print area can be customized according to your requirement. It means you can print a particular range of cells from the whole spreadsheet, customize the paper size, print the contents with the grid lines turned on, etc.

The following code is used to set up the print area on a spreadsheet.

```java
import java.io.File;
import java.io.FileOutputStream;
import org.apache.poi.xssf.usermodel.XSSFPrintSetup;
import org.apache.poi.xssf.usermodel.XSSFSheet;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;

public class PrintArea {
    public static void main(String[] args) throws Exception {
        XSSFWorkbook workbook = new XSSFWorkbook();
        XSSFSheet spreadsheet = workbook.createSheet("Print Area");

        //set print area with indexes
        workbook.setPrintArea(0, //sheet index
```

```java
    0, 11)
```
Let us save the above code as PrintArea.java. Compile and execute it from the command prompt as follows.

```bash
javac PrintArea.java
java PrintArea
```
It will generate a file named `printarea.xlsx` in your current directory and display the following output on the command prompt.

```
printarea.xlsx written successfully
```

In the above code, we have not added any cell values. Hence `printarea.xlsx` is a blank file. But you can observe in the following figure that the print preview shows the print area with grid lines.
This chapter explains how the POI library interacts with a database. With the help of JDBC, you can retrieve data from a database and insert that data into a spreadsheet using the POI library. Let us consider MySQL database for SQL operations.

### Write into Database

Let us assume the following employee data table called `emp_tbl` is to be retrieved from the MySQL database `test`.

<table>
<thead>
<tr>
<th>EMP ID</th>
<th>EMP NAME</th>
<th>DEG</th>
<th>SALARY</th>
<th>DEPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1201</td>
<td>Gopal</td>
<td>Technical manager</td>
<td>45000</td>
<td>IT</td>
</tr>
<tr>
<td>1202</td>
<td>Manisha</td>
<td>Proof reader</td>
<td>45000</td>
<td>Testing</td>
</tr>
<tr>
<td>1203</td>
<td>Masthanvali</td>
<td>Technical writer</td>
<td>40000</td>
<td>IT</td>
</tr>
<tr>
<td>1204</td>
<td>Krian</td>
<td>Hr Admin</td>
<td>40000</td>
<td>HR</td>
</tr>
<tr>
<td>1205</td>
<td>Kranthi</td>
<td>Op Admin</td>
<td>30000</td>
<td>Admin</td>
</tr>
</tbody>
</table>

Use the following code to retrieve data from a database and insert the same into a spreadsheet.

```java
import java.io.File;
import java.io.FileOutputStream;
import java.sql.Connection;
```
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import org.apache.poi.xssf.usermodel.XSSFCell;
import org.apache.poi.xssf.usermodel.XSSFRow;
import org.apache.poi.xssf.usermodel.XSSFSheet;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;

public class ExcelDatabase {
    public static void main(String[] args) throws Exception {
        Class.forName("com.mysql.jdbc.Driver");
        Connection connect = DriverManager.getConnection(
            "jdbc:mysql://localhost:3306/test",
            "root",
            "root"
        );
        Statement statement = connect.createStatement();
        ResultSet resultSet = statement.executeQuery("select * from emp_tbl");

        XSSFWorkbook workbook = new XSSFWorkbook();
        XSSFSheet spreadsheet = workbook.createSheet("employee db");
        XSSFRow row = spreadsheet.createRow(1);
        XSSFCell cell;
        cell = row.createCell(1);
        cell.setCellValue("EMP ID");
        cell = row.createCell(2);
        cell.setCellValue("EMP NAME");
        cell = row.createCell(3);
cell.setCellValue("DEG");
cell=row.createCell(4);
cell.setCellValue("SALARY");
cell=row.createCell(5);
cell.setCellValue("DEPT");

int i=2;
while(resultSet.next())
{
    row=spreadsheet.createRow(i);
    cell=row.createCell(1);
    cell.setCellValue(resultSet.getInt("eid"));
    cell=row.createCell(2);
    cell.setCellValue(resultSet.getString("ename"));
    cell=row.createCell(3);
    cell.setCellValue(resultSet.getString("deg"));
    cell=row.createCell(4);
    cell.setCellValue(resultSet.getString("salary"));
    cell=row.createCell(5);
    cell.setCellValue(resultSet.getString("dept"));
    i++;
}

FileOutputStream out = new FileOutputStream(
    new File("exceldatabase.xlsx"));
workbook.write(out);
out.close();
System.out.println(  
    "exceldatabase.xlsx written successfully");
}
Let us save the above code as `ExcelDatabase.java`. Compile and execute it from the command prompt as follows.

```bash
$javac ExcelDatabase.java
$java ExcelDatabase
```

It will generate an Excel file named `exceldatabase.xlsx` in your current directory and display the following output on the command prompt.

```
exceldatabase.xlsx written successfully
```

The `exceldatabase.xlsx` file looks as follows.

![Excel Database Excel Sheet](image-url)