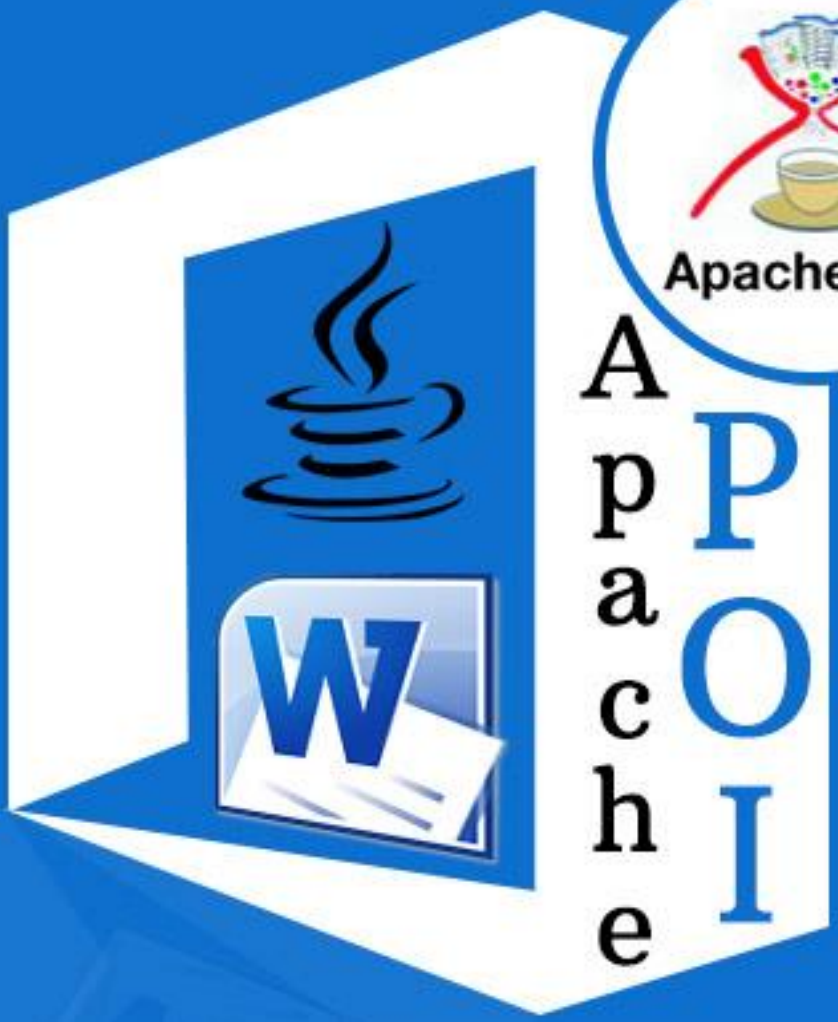


LEARN APACHE POI-WORD

java word library

tutorialspoint

SIMPLYEASYLEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia

https://twitter.com/tutorialspoint

About the Tutorial

This tutorial provides a basic understanding of Apache POI library and its features. Here we will learn how to read, write, and manage MS-Word documents using Java programs.

Audience

This tutorial is designed for the readers working on Java and especially those who want to create, read, write, and modify Word files using Java.

Prerequisites

A general awareness of Java programming with JDK 1.5 or later versions and IO concepts in Java are the only prerequisites to understand this tutorial.

Copyright & Disclaimer

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

Table of Contents

1. APACHE POI – OVERVIEW

Many a time, a software application is required to generate reference documents in Microsoft Word file format. Sometimes, an application is even expected to receive Word files as input data.

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 MS-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.
- **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 to 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 MS-Word files using Java. Therefore the discussion is confined to HWPF and XWPF components.

Note: Older versions of POI support binary file formats such as DOC, XLS, PPT, etc. Version 3.5 onwards, POI supports OOXML file formats of MS-Office such as DOCX, XLSX, PPTX, etc.

2. APACHE POI – INSTALLATION

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

JDK	Java SE 2 JDK 1.5 or above
Memory	1 GB RAM (recommended)
Disk Space	No minimum requirement
Operating System Version	Windows XP or above, Linux

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 mentioned below, 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:

Platform	Command	Sample Output
Windows	Open command console and type: **\>java –version**	Java version "1.7.0_60" Java (TM) SE Run Time Environment (build 1.7.0_60-b19) Java Hotspot (TM) 64-bit Server VM (build 24.60-b09,mixed mode)

Linux	Open command terminal and type: **$java –version**	java version "1.7.0_25" Open JDK Runtime Environment (rhel-2.3.10.4.el6_4-x86_64) Open JDK 64-Bit Server VM (build 23.7-b01, mixed mode)

- We assume that 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,

Platform	Description
Windows	Set JAVA_HOME to C:\ProgramFiles\java\jdk1.7.0_60
Linux	Export JAVA_HOME=/usr/local/java-current

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

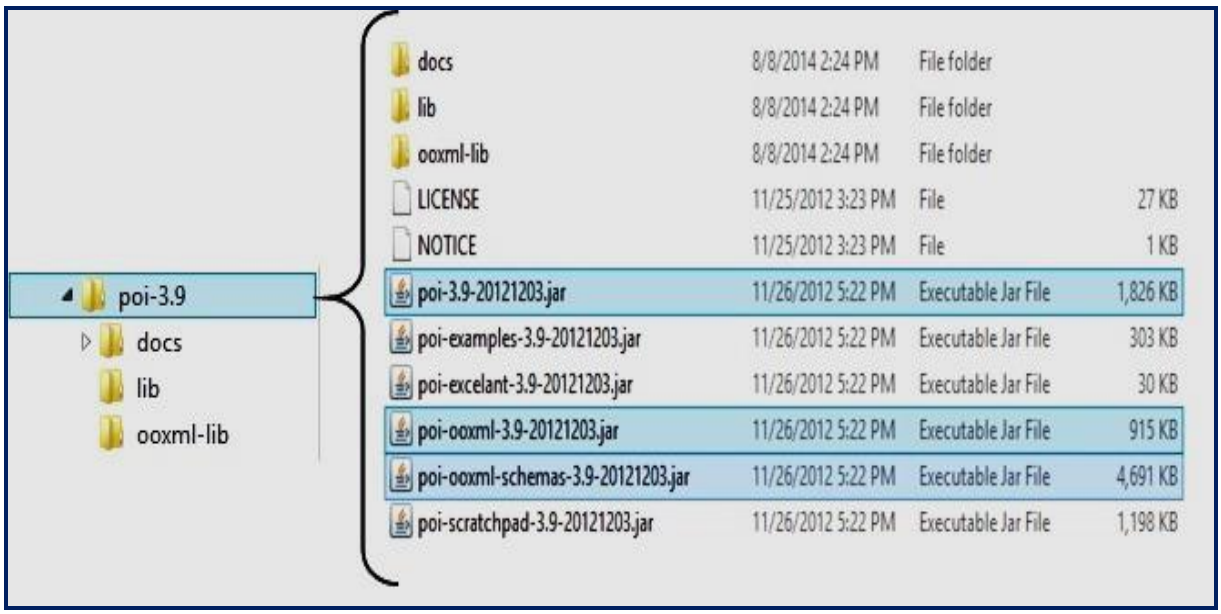
Platform	Description
Windows	Append the String "C:\Program Files\Java\jdk1.7.0_60\bin" to the end of the system variable PATH.
Linux	Export PATH=$PATH:$JAVA_HOME/bin/

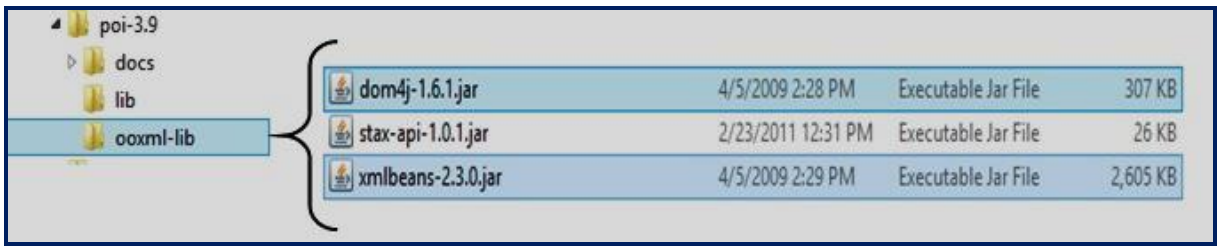
Execute the command **java -version** from the command prompt as explained above.

Step 3: Install Apache POI Library

Download the latest version of Apache POI from http://poi.apache.org/download.html 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 shows 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.

Platform	Description
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

3. CORE CLASSES

This chapter takes you through the classes and methods of Apache POI for managing a Word document.

Document

This is a marker interface (interface do not contain any methods), that notifies that the implemented class can be able to create a word document.

XWPFDocument

This is a class under **org.apache.poi.xwpf.usermodel** package. It is used to create MS-Word Document with .docx file format.

Class Methods:

S. No.	Method and Description
1	**commit()** Commits and saves the document.
2	**createParagraph()** Appends a new paragraph to this document.
3	**createTable()** Creates an empty table with one row and one column as default.
4	createTOC() Creates a table of content for Word document.
5	**getParagraphs()** Returns the paragraph(s) that holds the text of the header or footer.
6	**getStyle()** Returns the styles object used.

For the remaining methods of this class, refer the complete API document at:

https://poi.apache.org/apidocs/index.html?org/apache/poi/openxml4j/opc/internal/package-summary.html.

XWPFParagraph

This is a class under **org.apache.poi.xwpf.usermodel** package and is used to create paragraph in a word document. This instance is also used to add all types of elements into word document.

Class Methods:

S. No.	Method and Description
1	**createRun()** Appends a new run to this paragraph.
2	**getAlignment()** Returns the paragraph alignment which shall be applied to the text in this paragraph.
3	**setAlignment(ParagraphAlignment align)** Specifies the paragraph alignment which shall be applied to the text in this paragraph.
4	**setBorderBottom(Borders border)** Specifies the border which shall be displayed below a set of paragraphs, which have the same set of paragraph border settings.
5	**setBorderLeft(Borders border)** Specifies the border which shall be displayed on the left side of the page around the specified paragraph.
6	**setBorderRight(Borders border)** Specifies the border which shall be displayed on the right side of the page around the specified paragraph.
7	**setBorderTop(Borders border)** Specifies the border which shall be displayed above a set of paragraphs which have the same set of paragraph border settings.

For the remaining methods of this class, refer the complete API document at:

https://poi.apache.org/apidocs/index.html?org/apache/poi/openxml4j/opc/internal/package-summary.html.

XWPFRun

This is a class under **org.apache.poi.xwpf.usermodel** package and is used to add a region of text to the paragraph.

Class Methods:

S. No.	Method and Description
1	**addBreak()** Specifies that a break shall be placed at the current location in the run content.
2	**addTab()** Specifies that a tab shall be placed at the current location in the run content.
3	**setColor(java.lang.String rgbStr)** Sets text color.
4	**setFontSize(int size)** Specifies the font size which shall be applied to all noncomplex script characters in the content of this run when displayed.
5	**setText(java.lang.String value)** Sets the text of this text run.
6	**setBold(boolean value)** Specifies whether the bold property shall be applied to all non-complex script characters in the content of this run when displayed in a document.

For the remaining methods of this class, refer the complete API document at:

https://poi.apache.org/apidocs/index.html?org/apache/poi/openxml4j/opc/internal/package-summary.html.

XWPFStyle

This is a class under **org.apache.poi.xwpf.usermodel** package and is used to add different styles to the object elements in a word document.

Class Methods:

S. No.	Method and Description
1	**getNextStyleID()** It is used to get StyleID of the next style.
2	**getStyleId()** It is used to get StyleID of the style.

3	**getStyles()** It is used to get styles.
4	**setStyleId(java.lang.String styleId)** It is used to set styleID.

For the remaining methods of this class, refer the complete API document at:

https://poi.apache.org/apidocs/index.html?org/apache/poi/openxml4j/opc/internal/package-summary.html.

XWPFTable

This is a class under **org.apache.poi.xwpf.usermodel** package and is used to add table data into a word document.

Class Methods:

S. No.	Method and Description
1	**addNewCol()** Adds a new column for each row in this table.
2	**addRow(XWPFTableRow row, int pos)** Adds a new Row to the table at position pos.
3	**createRow()** Creates a new XWPFTableRow object with as many cells as the number of columns defined in that moment.
4	setWidth**(int width)** Sets the width of the column.

For the remaining methods of this class, refer the complete API document at:

https://poi.apache.org/apidocs/index.html?org/apache/poi/openxml4j/opc/internal/package-summary.html.

XWPFWordExtractor

This is a class under **org.apache.poi.xwpf.extractor** package. It is a basic parser class used to extract the simple text from a Word document.

Class Methods:

S. No.	Method and Description
1	**getText()** Retrieves all the text from the document.

For the remaining methods of this class, refer the complete API document at:

https://poi.apache.org/apidocs/index.html?org/apache/poi/openxml4j/opc/internal/package-summary.html.

4. DOCUMENT

Here the term 'document' refers to a MS-Word file. After completion of this chapter, you will be able to create new documents and open existing documents using your Java program.

Create Blank Document

The following simple program is used to create a blank MS-Word document:

```
import java.io.File;
import java.io.FileOutputStream;
import org.apache.poi.xwpf.usermodel.XWPFDocument;

public class CreateDocument
{
      public static void main(String[] args)throws Exception
      {
         //Blank Document
         XWPFDocument document= new XWPFDocument();
         //Write the Document in file system
         FileOutputStream out = new FileOutputStream(
                  new File("createdocument.docx"));
         document.write(out);
         out.close();
         System.out.println(
                  "createdocument.docx written successully");
      }
}
```

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

```
$javac  CreateDocument.java
$java CreateDocument
```

If your system environment is configured with the POI library, it will compile and execute to generate a blank Excel file named **createdocument.docx** in your current directory and display the following output in the command prompt:

```
createdocument.docx written successfully
```

5. PARAGRAPH

In this chapter you will learn how to create a Paragraph and how to add it to a document using Java. Paragraph is a part of a page in a Word file.

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

Create a Paragraph

First of all, let us create a Paragraph using the referenced classes discussed in the earlier chapters. By following the previous chapter, create a Document first, and then we can create a Paragraph.

The following code snippet is used to create a spreadsheet:

```
//Create Blank document
  XWPFDocument document= new XWPFDocument();
//Create a blank spreadsheet
  XWPFParagraph paragraph = document.createParagraph();
```

Run on Paragraph

You can enter the text or any object element, using **Run**. Using Paragraph instance you can create **run**.

The following code snippet is used to create a Run.

```
XWPFRun run=paragraph.createRun();
```

Write into a Paragraph

Let us try entering some text into a document. Consider the below text data:

```
At tutorialspoint.com, we strive hard to provide quality tutorials
```

```
for self-learning purpose in the domains of Academics, Information
Technology, Management and Computer Programming Languages.
```

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

```
import java.io.File;
import java.io.FileOutputStream;
import org.apache.poi.xwpf.usermodel.XWPFDocument;
import org.apache.poi.xwpf.usermodel.XWPFParagraph;
import org.apache.poi.xwpf.usermodel.XWPFRun;

public class CreateParagraph
{
      public static void main(String[] args)throws Exception
      {
        //Blank Document
         XWPFDocument document= new XWPFDocument();
         //Write the Document in file system
         FileOutputStream out = new FileOutputStream(
                  new File("createparagraph.docx"));

         //create Paragraph
         XWPFParagraph paragraph = document.createParagraph();
         XWPFRun run=paragraph.createRun();
         run.setText("At tutorialspoint.com, we strive hard to " +
            "provide quality tutorials for self-learning " +
            "purpose in the domains of Academics, Information " +
            "Technology, Management and Computer Programming
                    Languages.");
         document.write(out);
         out.close();
         System.out.println("createparagraph.docx written
successfully");
      }
}
```

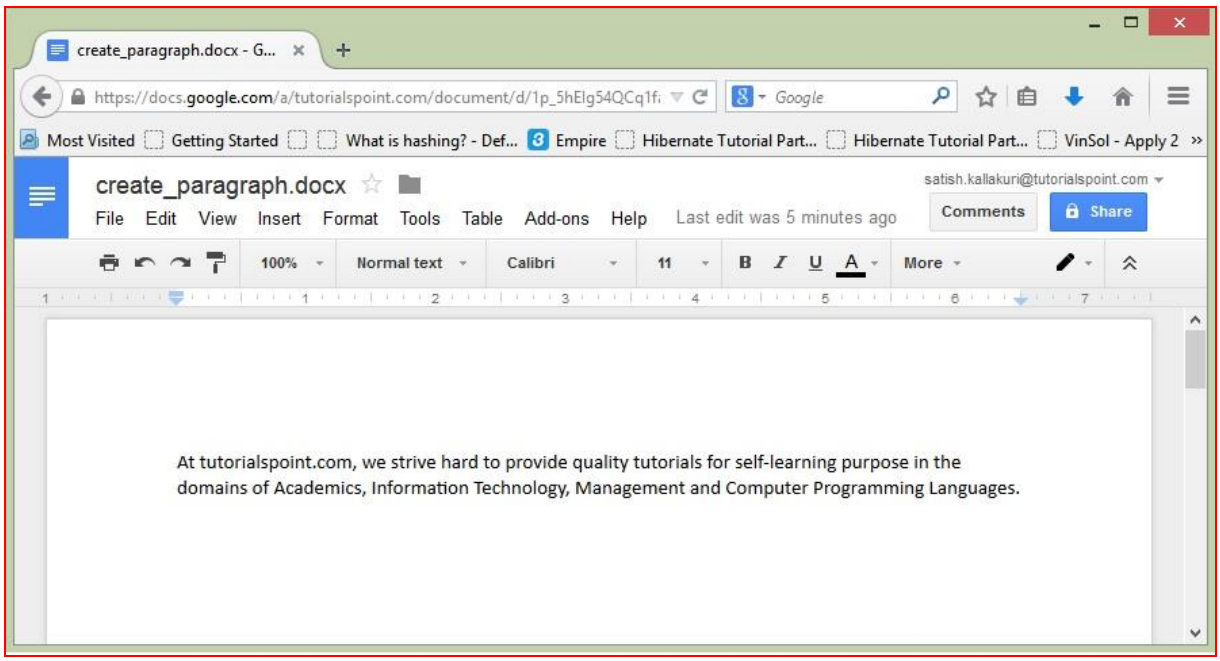
Save the above Java code as **CreateParagraph.java**, and then compile and run it from the command prompt as follows:

```
$javac CreateParagraph.java
$java CreateParagraph
```

It will compile and execute to generate a Word file named **createparagraph.docx** in your current directory and you will get the following output in the command prompt:

```
createparagraph.docx written successfully
```

The **createparagraph.docx** file looks as follows.



6. BORDERS

In this chapter, you will learn how to apply border to a paragraph using Java programing.

Applying Border

The following code is used to apply Borders in a Document:

```
import java.io.File;
import java.io.FileOutputStream;
import org.apache.poi.xwpf.usermodel.Borders;
import org.apache.poi.xwpf.usermodel.XWPFDocument;
import org.apache.poi.xwpf.usermodel.XWPFParagraph;
import org.apache.poi.xwpf.usermodel.XWPFRun;

public class ApplyingBorder
{
      public static void main(String[] args)throws Exception
      {
        //Blank Document
         XWPFDocument document= new XWPFDocument();

         //Write the Document in file system
         FileOutputStream out = new FileOutputStream(
                  new File("applyingborder.docx"));

         //create paragraph
         XWPFParagraph paragraph = document.createParagraph();

         //Set bottom border to paragraph
         paragraph.setBorderBottom(Borders.BASIC_BLACK_DASHES);

         //Set left border to paragraph
         paragraph.setBorderLeft(Borders.BASIC_BLACK_DASHES);
```

```
        //Set right border to paragraph
        paragraph.setBorderRight(Borders.BASIC_BLACK_DASHES);

        //Set top border to paragraph
        paragraph.setBorderTop(Borders.BASIC_BLACK_DASHES);

        XWPFRun run=paragraph.createRun();
        run.setText("At tutorialspoint.com, we strive hard to " +
           "provide quality tutorials for self-learning " +
           "purpose in the domains of Academics, Information " +
           "Technology, Management and Computer Programming " +
                "Languages.");

        document.write(out);
        out.close();
        System.out.println("
            applyingborder.docx written successully");
     }
}
```

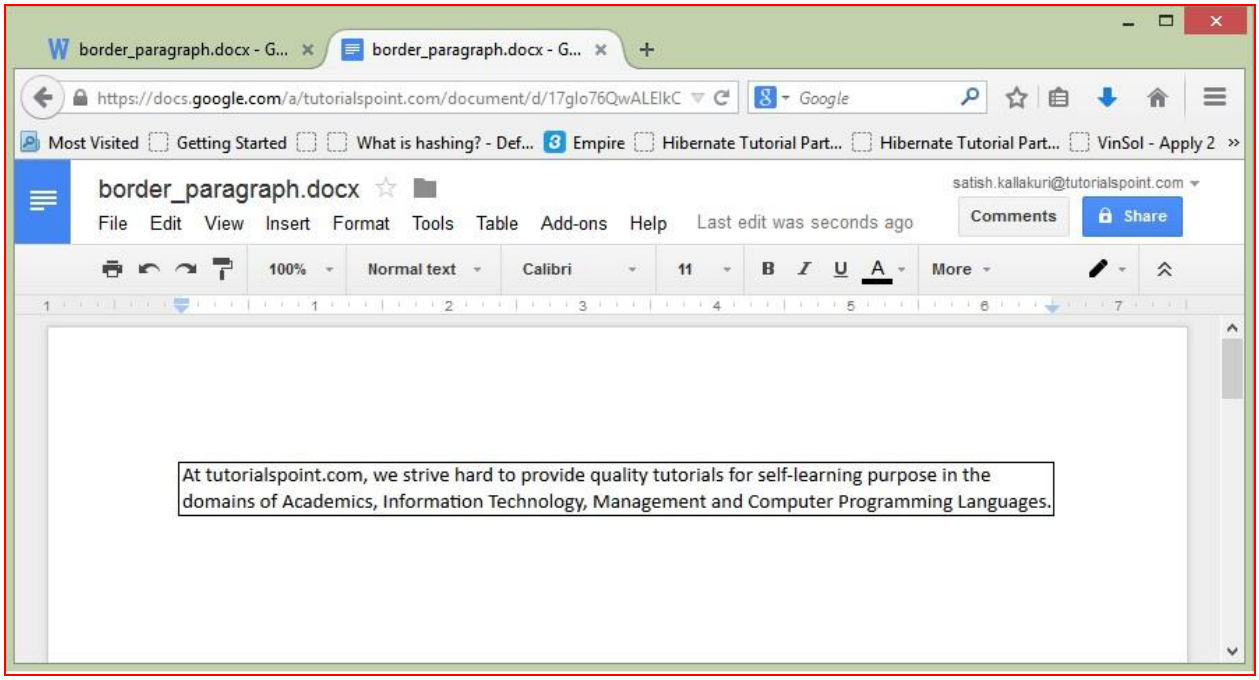
Save the above code in a file named **ApplyingBorder.java,** compile and execute it from the command prompt as follows:

```
$javac ApplyingBorder.java
$java ApplyingBorder
```

If your system is configured with the POI library, then it will compile and execute to generate a Word document named **applyingborder.docx** in your current directory and display the following output:

```
applyingborder.docx written successfully
```

The **applyingborder.docx** file looks as follows:


border_paragraph.docx - G...
border_paragraph.docx - G...
https://docs.google.com/a/tutorialspoint.com/document/d/17glo76QwALElkC
Most Visited
Getting Started
What is hashing? - Def...
Empire
Hibernate Tutorial Part...
Hibernate Tutorial Part...
VinSol - Apply 2
border_paragraph.docx
satish.kallakuri@tutorialspoint.com
File Edit View Insert Format Tools Table Add-ons Help
Last edit was seconds ago
Comments
Share
100%
Normal text
Calibri
11
More
At tutorialspoint.com, we strive hard to provide quality tutorials for self-learning purpose in the domains of Academics, Information Technology, Management and Computer Programming Languages.

7. TABLES

In this chapter, you will learn how to create a table of data in a document. You can create a table data by using **XWPFTable** class. By adding each **Row** to table and adding each **cell** to **Row,** you will get table data.

Create Table

The following code is used to creating table in a document:

```
import java.io.File;
import java.io.FileOutputStream;
import org.apache.poi.xwpf.usermodel.XWPFDocument;
import org.apache.poi.xwpf.usermodel.XWPFTable;
import org.apache.poi.xwpf.usermodel.XWPFTableRow;

public class CreateTable
{
      public static void main(String[] args)throws Exception
      {
        //Blank Document
         XWPFDocument document= new XWPFDocument();

         //Write the Document in file system
         FileOutputStream out = new FileOutputStream(
                  new File("create_table.docx"));

         //create table
        XWPFTable table = document.createTable();

         //create first row
         XWPFTableRow tableRowOne = table.getRow(0);
         tableRowOne.getCell(0).setText("col one, row one");
         tableRowOne.addNewTableCell().setText("col two, row one");
         tableRowOne.addNewTableCell().setText("col three, row
```

```
                one");
        //create second row
        XWPFTableRow tableRowTwo = table.createRow();
        tableRowTwo.getCell(0).setText("col one, row two");
        tableRowTwo.getCell(1).setText("col two, row two");
        tableRowTwo.getCell(2).setText("col three, row two");

        //create third row
        XWPFTableRow tableRowThree = table.createRow();
        tableRowThree.getCell(0).setText("col one, row three");
        tableRowThree.getCell(1).setText("col two, row three");
        tableRowThree.getCell(2).setText("col three, row three");

        document.write(out);
        out.close();
        System.out.println("
             create_table.docx written successully");
     }
}
```

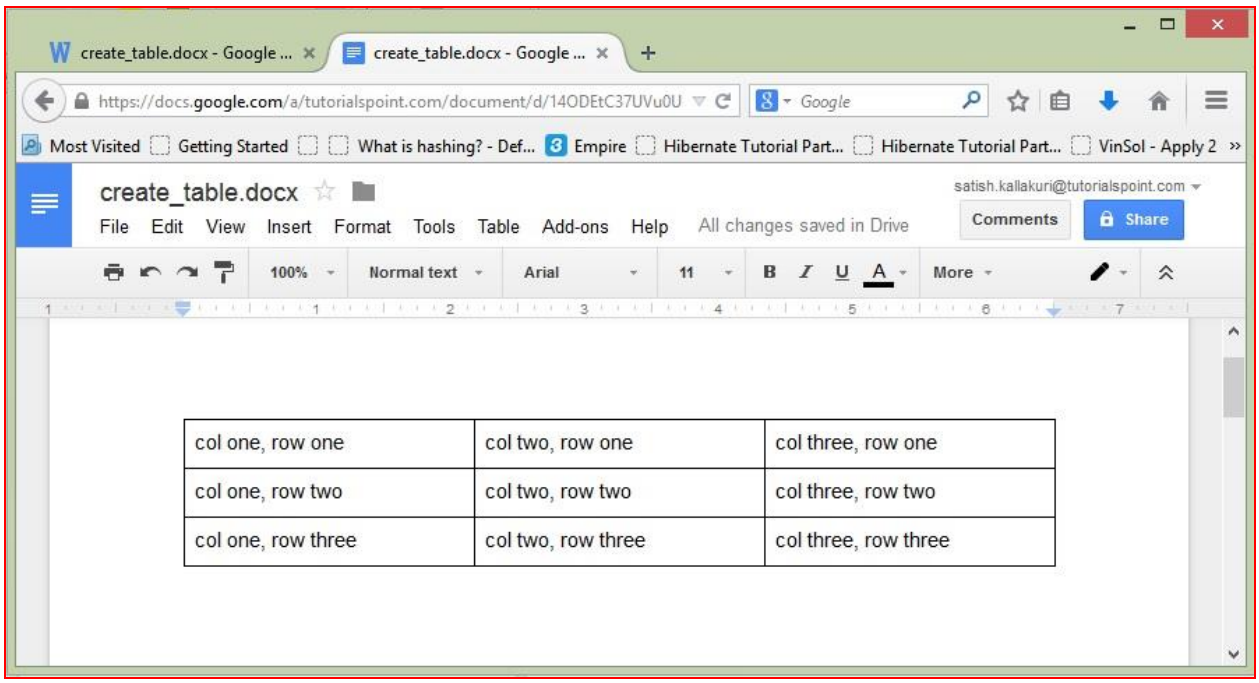
Save the above code in a file named **CreateTable.java**. Compile and execute it from the command prompt as follows:

```
$javac CreateTable.java
$java CreateTable
```

It generates a Word file named **createtable.docx** in your current directory and display the following output on the command prompt:

```
createtable.docx written successfully
```

The **createtable.docx** file looks as follows:



8. FONT STYLE AND ALIGNMENT

This chapter shows how to apply different font styles and alignments in a Word document using Java. Generally, Font Style contains: Font size, Type, Bold, Italic, and Underline. And Alignment is categorized into left, center, right, and justify.

Font Style

The following code is used to set different styles of font:

```
import java.io.File;
import java.io.FileOutputStream;
import org.apache.poi.xwpf.usermodel.VerticalAlign;
import org.apache.poi.xwpf.usermodel.XWPFDocument;
import org.apache.poi.xwpf.usermodel.XWPFParagraph;
import org.apache.poi.xwpf.usermodel.XWPFRun;

public class FontStyle
{
      public static void main(String[] args)throws Exception
      {
        //Blank Document
         XWPFDocument document= new XWPFDocument();

         //Write the Document in file system
         FileOutputStream out = new FileOutputStream(
                  new File("fontstyle.docx"));

         //create paragraph
         XWPFParagraph paragraph = document.createParagraph();

         //Set Bold an Italic
        XWPFRun paragraphOneRunOne = paragraph.createRun();
```

```
        paragraphOneRunOne.setBold(true);
        paragraphOneRunOne.setItalic(true);
        paragraphOneRunOne.setText("Font Style");
        paragraphOneRunOne.addBreak();

        //Set text Position
        XWPFRun paragraphOneRunTwo = paragraph.createRun();
        paragraphOneRunTwo.setText("Font Style two");
        paragraphOneRunTwo.setTextPosition(100);

        //Set Strike through and Font Size and Subscript
        XWPFRun paragraphOneRunThree = paragraph.createRun();
        paragraphOneRunThree.setStrike(true);
        paragraphOneRunThree.setFontSize(20);
        paragraphOneRunThree.setSubscript(
                VerticalAlign.SUBSCRIPT);
        paragraphOneRunThree.setText(" Different Font Styles");

        document.write(out);
        out.close();
        System.out.println("fontstyle.docx written successully");
    }
}
```

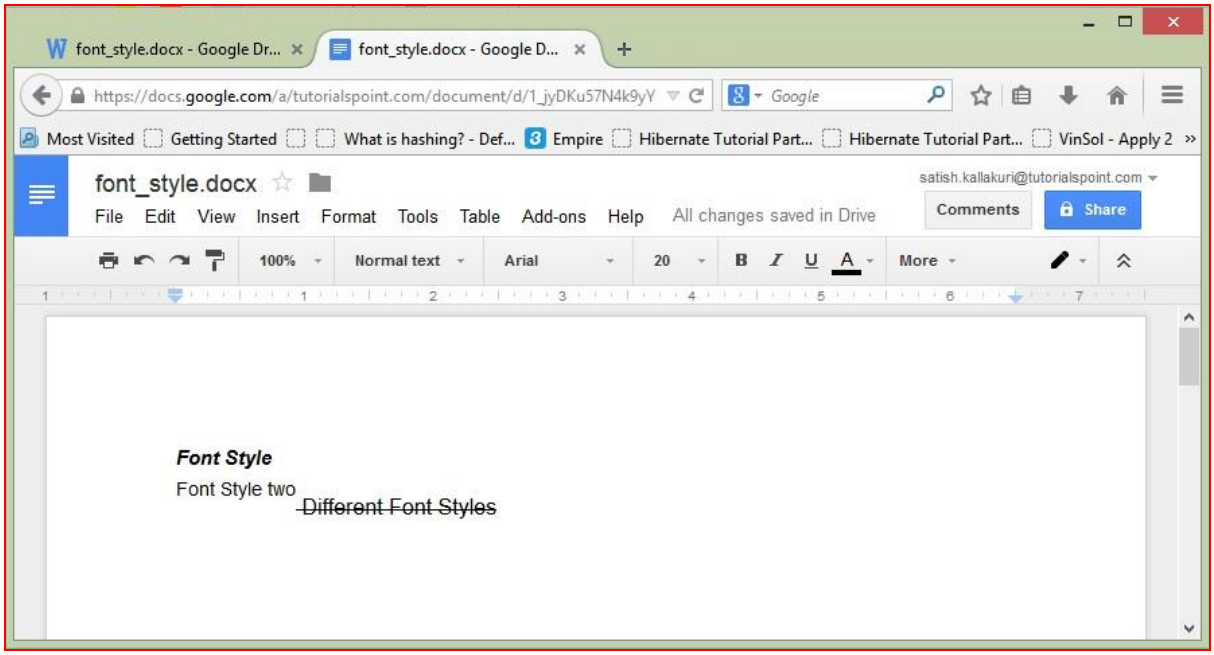
Save the above code as **FontStyle.java** and then compile and execute it from the command prompt as follows:

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

It will generate a Word file named **fontstyle.docx** in your current directory and display the following output on the command prompt:

```
fontstyle.docx written successfully
```

The **fontstyle.docx** file looks as follows.



Alignment

The following code is used to set alignment to the paragraph text:

```
import java.io.File;
import java.io.FileOutputStream;
import org.apache.poi.xwpf.usermodel.ParagraphAlignment;
import org.apache.poi.xwpf.usermodel.XWPFDocument;
import org.apache.poi.xwpf.usermodel.XWPFParagraph;
import org.apache.poi.xwpf.usermodel.XWPFRun;

public class AlignParagraph
{
      public static void main(String[] args)throws Exception
      {
         //Blank Document
          XWPFDocument document= new XWPFDocument();

          //Write the Document in file system
          FileOutputStream out = new FileOutputStream(
```

```
                new File("alignparagraph.docx"));

        //create paragraph
        XWPFParagraph paragraph = document.createParagraph();

        //Set alignment paragraph to RIGHT
        paragraph.setAlignment(ParagraphAlignment.RIGHT);
        XWPFRun run=paragraph.createRun();
        run.setText("At tutorialspoint.com, we strive hard to " +
           "provide quality tutorials for self-learning " +
           "purpose in the domains of Academics, Information " +
           "Technology, Management and Computer Programming " +
                "Languages.");

        //Create Another paragraph
        paragraph=document.createParagraph();

        //Set alignment paragraph to CENTER
        paragraph.setAlignment(ParagraphAlignment.CENTER);
        run=paragraph.createRun();
        run.setText("The endeavour started by Mohtashim, an AMU " +
          "alumni, who is the founder and the managing director " +
          "of Tutorials Point (I) Pvt. Ltd. He came up with the " +
          "website tutorialspoint.com in year 2006 with the help" +
          "of handpicked freelancers, with an array of tutorials" +
          " for computer programming languages. ");
        document.write(out);
        out.close();
        System.out.println("
              alignparagraph.docx written successfully");
     }
}
```

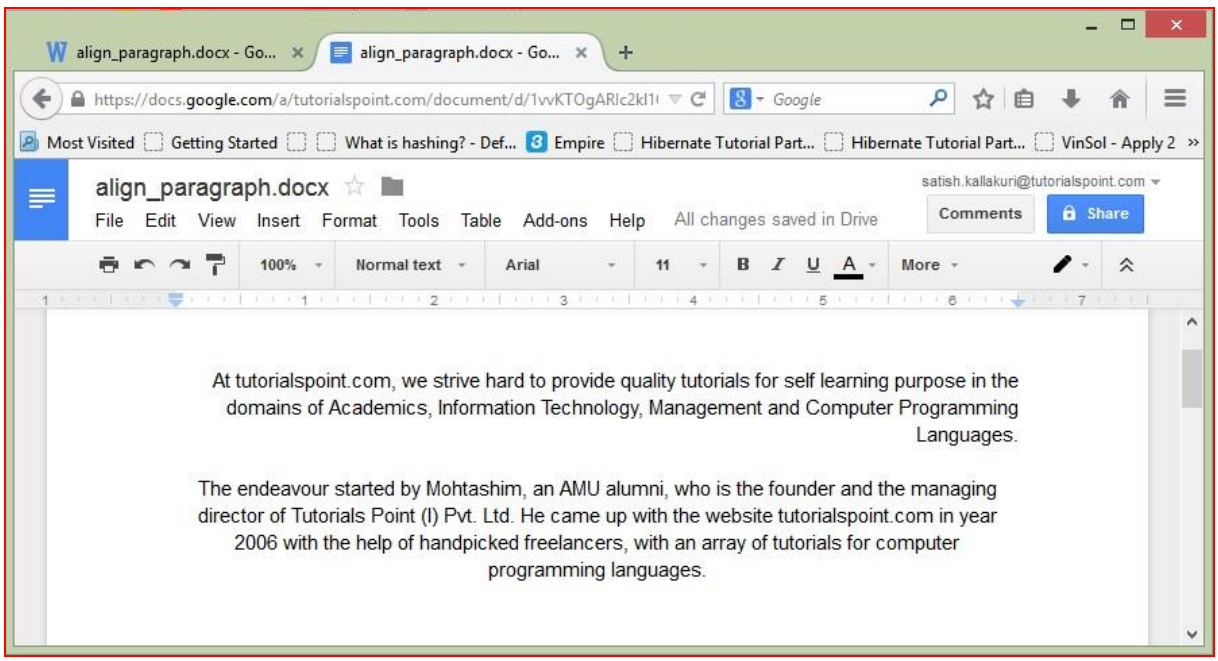
Save the above code as **AlignParagraph.java** and then compile and execute it from the command prompt as follows:

```
$javac AlignParagraph.java
$java AlignParagraph
```

It will generate a Word file named **alignparagraph.docx** in your current directory and display the following output in the command prompt:

```
alignparagraph.docx written successfully
```

The **alignparagraph.docx** file looks as follows:



9. TEXT EXTRACTION

This chapter explains how to extract simple text data from a Word document using Java. In case you want to extract metadata from a Word document, make use of Apache Tika.

For .docx files, we use the class org.apache.poi.xwpf.extractor.XPFFWordExtractor that extracts and returns simple data from a Word file. In the same way, we have different methodologies to extract headings, footnotes, table data, etc. from a Word file.

The following code shows how to extract simple text from a Word file:

```
import java.io.FileInputStream;
import org.apache.poi.xwpf.extractor.XWPFWordExtractor;
import org.apache.poi.xwpf.usermodel.XWPFDocument;

public class WordExtractor
{
      public static void main(String[] args)throws Exception
      {
            XWPFDocument docx = new XWPFDocument(
               new FileInputStream("create_paragraph.docx"));
            //using XWPFWordExtractor Class
            XWPFWordExtractor we = new XWPFWordExtractor(docx);
            System.out.println(we.getText());
      }
}
```

Save the above code as **WordExtractor.java**. Compile and execute it from the command prompt as follows:

```
$javac WordExtractor.java
$java WordExtractor
```

It will generate the following output:

```
At tutorialspoint.com, we strive hard to provide quality tutorials for
self-learning purpose in the domains of Academics, Information Technology,
Management and Computer Programming Languages.
```