**Apache POI – Read and Write Excel File in Java**

Learn to read excel, write excel, evaluate formula cells and apply custom formatting to generated excel file using Apache POI library with examples.

If you are building a software for HR or finance domain, there is usually requirement for generating excel reports which are usually across management levels. Apart from reports, you can expect input data for application coming in form of excel sheets and application is expected to support it.

[**Apache POI**](https://poi.apache.org/) is well trusted library among many other open source libraries to handle such usecases involving excel files. Using POI, you can read and write MS Excel files using Java.

Please note that, in addition, you can read and write MS Word and MS PowerPoint files also using POI library.

In this **Apache POI tutorial**, We will discuss some common excel operations required to do in real life applications.

Table of Contents:

[Apache POI Dependencies](https://howtodoinjava.com/apache-commons/readingwriting-excel-files-in-java-poi-tutorial/#poi_runtime_dependencies)

[POI Classes](https://howtodoinjava.com/apache-commons/readingwriting-excel-files-in-java-poi-tutorial/#useful_common_classes)

[Write an excel file](https://howtodoinjava.com/apache-commons/readingwriting-excel-files-in-java-poi-tutorial/#writing_excel_file)

[Read an excel file](https://howtodoinjava.com/apache-commons/readingwriting-excel-files-in-java-poi-tutorial/#reading_excel_file)

[Using formulas in excel sheet](https://howtodoinjava.com/apache-commons/readingwriting-excel-files-in-java-poi-tutorial/#using_formulas)

[Formatting the cells](https://howtodoinjava.com/apache-commons/readingwriting-excel-files-in-java-poi-tutorial/#formatting_cells)

[Sourcecode download](https://howtodoinjava.com/apache-commons/readingwriting-excel-files-in-java-poi-tutorial/#sourcecode)

**Apache POI – Dependencies**

If you are working on a maven project, you can include the POI dependency in pom.xml file using this:

|  |
| --- |
| pom.xml |
| <dependency>      <groupId>org.apache.poi</groupId>      <artifactId>poi</artifactId>      <version>3.9</version>  </dependency> |

If you are not using maven, then you can download maven jar files from [**POI download**](https://poi.apache.org/download.html) page. Include following jar files to run the sample code given in this tutorial.

* dom4j-1.6.1.jar
* poi-3.9-20121203.jar
* poi-ooxml-3.9-20121203.jar
* poi-ooxml-schemas-3.9-20121203.jar
* xmlbeans-2.3.0.jar

**Apache POI – Classes**

1. **HSSF, XSSF and XSSF classes**

Apache POI main classes usually start with either **HSSF**, **XSSF** or **SXSSF**.

* + **HSSF** – is the POI Project’s pure Java implementation of the Excel ’97(-2007) file format. e.g. **[HSSFWorkbook](https://poi.apache.org/apidocs/org/apache/poi/hssf/usermodel/HSSFWorkbook.html" \o "HSSFWorkbook)**, **[HSSFSheet](https://poi.apache.org/apidocs/org/apache/poi/hssf/usermodel/HSSFSheet.html" \o "HSSFSheet)**.
  + **XSSF** – is the POI Project’s pure Java implementation of the Excel 2007 OOXML (.xlsx) file format. e.g. **[XSSFWorkbook](https://poi.apache.org/apidocs/org/apache/poi/xssf/usermodel/XSSFWorkbook.html" \o "XSSFWorkbook)**, **[XSSFSheet](https://poi.apache.org/apidocs/org/apache/poi/xssf/usermodel/XSSFSheet.html" \o "XSSFSheet)**.
  + **SXSSF** (since 3.8-beta3) – is an API-compatible streaming extension of XSSF to be used when very large spreadsheets have to be produced, and heap space is limited. e.g. **[SXSSFWorkbook](https://poi.apache.org/apidocs/org/apache/poi/xssf/streaming/SXSSFWorkbook.html" \o "SXSSFWorkbook)**, **[SXSSFSheet](https://poi.apache.org/apidocs/org/apache/poi/xssf/streaming/SXSSFSheet.html" \o "SXSSFSheet)**. SXSSF achieves its**low memory footprint by limiting access to the rows that are within a sliding window**, while XSSF gives access to all rows in the document.

1. **Row and Cell**

Apart from above classes, [**Row**](https://poi.apache.org/apidocs/org/apache/poi/ss/usermodel/Row.html) and [**Cell**](https://poi.apache.org/apidocs/org/apache/poi/ss/usermodel/Cell.html) are used to interact with a particular row and a particular cell in excel sheet.

1. **Style Classes**

A wide range of classes like [**CellStyle**](https://poi.apache.org/apidocs/org/apache/poi/ss/usermodel/CellStyle.html), [**BuiltinFormats**](https://poi.apache.org/apidocs/org/apache/poi/ss/usermodel/BuiltinFormats.html), [**ComparisonOperator**](https://poi.apache.org/apidocs/org/apache/poi/ss/usermodel/ComparisonOperator.html), [**ConditionalFormattingRule**](https://poi.apache.org/apidocs/org/apache/poi/ss/usermodel/ConditionalFormattingRule.html), [**FontFormatting**](https://poi.apache.org/apidocs/org/apache/poi/ss/usermodel/FontFormatting.html), [**IndexedColors**](https://poi.apache.org/apidocs/org/apache/poi/ss/usermodel/IndexedColors.html), [**PatternFormatting**](https://poi.apache.org/apidocs/org/apache/poi/hssf/record/cf/PatternFormatting.html), [**SheetConditionalFormatting**](https://poi.apache.org/apidocs/org/apache/poi/ss/usermodel/SheetConditionalFormatting.html) etc. are used when you have to add formatting in a sheet, mostly based on some rules.

1. **FormulaEvaluator**

Another useful class **FormulaEvaluator** is used to evaluate the formula cells in excel sheet.

**Apache POI – Write an excel file**

I am taking this example first so that we can reuse the excel sheet created by this code to read back in next example.

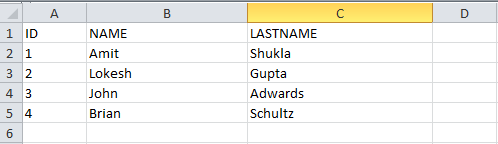
**Writing excel using POI** is very simple and involve following steps:

1. Create a workbook
2. Create a sheet in workbook
3. Create a row in sheet
4. Add cells in sheet
5. Repeat step 3 and 4 to write more data

It seems very simple, right? Lets have a look at the code doing these steps.

Java program to write excel file using apache POI library.

|  |
| --- |
| package com.howtodoinjava.demo.poi;  //import statements  public class WriteExcelDemo  {      public static void main(String[] args)      {          //Blank workbook          XSSFWorkbook workbook = new XSSFWorkbook();            //Create a blank sheet          XSSFSheet sheet = workbook.createSheet("Employee Data");            //This data needs to be written (Object[])          Map<String, Object[]> data = new TreeMap<String, Object[]>();          data.put("1", new Object[] {"ID", "NAME", "LASTNAME"});          data.put("2", new Object[] {1, "Amit", "Shukla"});          data.put("3", new Object[] {2, "Lokesh", "Gupta"});          data.put("4", new Object[] {3, "John", "Adwards"});          data.put("5", new Object[] {4, "Brian", "Schultz"});            //Iterate over data and write to sheet          Set<String> keyset = data.keySet();          int rownum = 0;          for (String key : keyset)          {              Row row = sheet.createRow(rownum++);              Object [] objArr = data.get(key);              int cellnum = 0;              for (Object obj : objArr)              {                 Cell cell = row.createCell(cellnum++);                 if(obj instanceof String)                      cell.setCellValue((String)obj);                  else if(obj instanceof Integer)                      cell.setCellValue((Integer)obj);              }          }          try          {              //Write the workbook in file system              FileOutputStream out = new FileOutputStream(newFile("howtodoinjava\_demo.xlsx"));              workbook.write(out);              out.close();              System.out.println("howtodoinjava\_demo.xlsx written successfully on disk.");          }          catch (Exception e)          {              e.printStackTrace();          }      }  } |



**Apache POI – Read an excel file**

**Reading an excel file using POI** is also very simple if we divide this in steps.

1. Create workbook instance from excel sheet
2. Get to the desired sheet
3. Increment row number
4. iterate over all cells in a row
5. repeat step 3 and 4 until all data is read

Lets see all above steps in code. I am writing the code to read the excel file created in above example. It will read all the **column names** and the values in it – cell by cell.

Java program to read excel file using apache POI library.

|  |
| --- |
| package com.howtodoinjava.demo.poi;  //import statements  public class ReadExcelDemo  {      public static void main(String[] args)      {          try          {              FileInputStream file = new FileInputStream(newFile("howtodoinjava\_demo.xlsx"));                //Create Workbook instance holding reference to .xlsx file              XSSFWorkbook workbook = new XSSFWorkbook(file);                //Get first/desired sheet from the workbook              XSSFSheet sheet = workbook.getSheetAt(0);                //Iterate through each rows one by one              Iterator<Row> rowIterator = sheet.iterator();              while (rowIterator.hasNext())              {                  Row row = rowIterator.next();                  //For each row, iterate through all the columns                  Iterator<Cell> cellIterator = row.cellIterator();                    while (cellIterator.hasNext())                  {                      Cell cell = cellIterator.next();                      //Check the cell type and format accordingly                      switch (cell.getCellType())                      {                          case Cell.CELL\_TYPE\_NUMERIC:                              System.out.print(cell.getNumericCellValue() + "t");                              break;                          case Cell.CELL\_TYPE\_STRING:                              System.out.print(cell.getStringCellValue() + "t");                              break;                      }                  }                  System.out.println("");              }              file.close();          }          catch (Exception e)          {              e.printStackTrace();          }      }  }    Output:    ID      NAME        LASTNAME  1.0     Amit        Shukla  2.0     Lokesh      Gupta  3.0     John        Adwards  4.0     Brian       Schultz |

**Apache POI – Add and evaluate formula cells**

When working on complex excel sheets, we encounter many cells which have formula to calculate their values. These are formula cells. Apache POI has excellent support for adding formula cells and evaluating already present formula cells also.

Les see one example of how to **add formula cells in excel**?

In this code, there are four cells in a row and fourth one in multiplication of all previous 3 rows. So the formula will be : A2\*B2\*C2 (in second row)

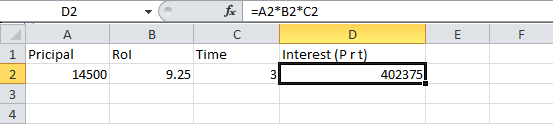
Java program to add formula in excel file using apache POI library.

|  |
| --- |
| public static void main(String[] args)  {      XSSFWorkbook workbook = new XSSFWorkbook();      XSSFSheet sheet = workbook.createSheet("Calculate Simple Interest");        Row header = sheet.createRow(0);      header.createCell(0).setCellValue("Pricipal");      header.createCell(1).setCellValue("RoI");      header.createCell(2).setCellValue("T");      header.createCell(3).setCellValue("Interest (P r t)");        Row dataRow = sheet.createRow(1);      dataRow.createCell(0).setCellValue(14500d);      dataRow.createCell(1).setCellValue(9.25);      dataRow.createCell(2).setCellValue(3d);      dataRow.createCell(3).setCellFormula("A2\*B2\*C2");        try {          FileOutputStream out =  new FileOutputStream(new File("formulaDemo.xlsx"));          workbook.write(out);          out.close();          System.out.println("Excel with foumula cells written successfully");        } catch (FileNotFoundException e) {          e.printStackTrace();      } catch (IOException e) {          e.printStackTrace();      }  } |

Similarly, I you want to read a file which have formula cells in it, use following logic to **evaluate formula cells**.

Java program to eveluate formula in excel file using apache POI library.

|  |
| --- |
| public static void readSheetWithFormula()  {      try      {          FileInputStream file = new FileInputStream(new File("formulaDemo.xlsx"));            //Create Workbook instance holding reference to .xlsx file          XSSFWorkbook workbook = new XSSFWorkbook(file);            FormulaEvaluator evaluator = workbook.getCreationHelper().createFormulaEvaluator();            //Get first/desired sheet from the workbook          XSSFSheet sheet = workbook.getSheetAt(0);            //Iterate through each rows one by one          Iterator<Row> rowIterator = sheet.iterator();          while (rowIterator.hasNext())          {              Row row = rowIterator.next();              //For each row, iterate through all the columns              Iterator<Cell> cellIterator = row.cellIterator();                while (cellIterator.hasNext())              {                  Cell cell = cellIterator.next();                  //Check the cell type after eveluating formulae                  //If it is formula cell, it will be evaluated otherwise no change will happen                  switch (evaluator.evaluateInCell(cell).getCellType())                  {                      case Cell.CELL\_TYPE\_NUMERIC:                          System.out.print(cell.getNumericCellValue() + "tt");                          break;                      case Cell.CELL\_TYPE\_STRING:                          System.out.print(cell.getStringCellValue() + "tt");                          break;                      case Cell.CELL\_TYPE\_FORMULA:                          //Not again                          break;                  }              }              System.out.println("");          }          file.close();      }      catch (Exception e)      {          e.printStackTrace();      }  }    Output:    Pricipal        RoI         T       Interest (P r t)  14500.0         9.25        3.0     402375.0 |



**Apache POI – Formatting the cells**

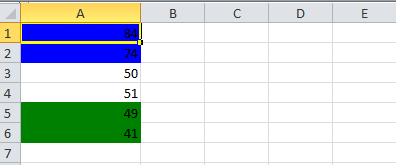
So for we have seen the examples of reading / writing and excel file using apache POI. But, when we are creating a report in excel file and it becomes utmost important to add formatting on cells which fit into any per-determined criteria. This formatting can be a different coloring based on certain value range, based on expiry date limit etc.

In below examples, I am taking couple of such **cell formatting** examples for various purposes.

**1) Cell value is in between a certain range**

This piece of code will color any cell in range whose value is between a configured range. [e.g. between 50 and 70]

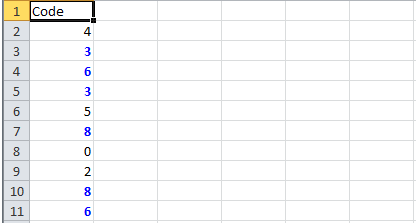
|  |
| --- |
| static void basedOnValue(Sheet sheet)  {      //Creating some random values      sheet.createRow(0).createCell(0).setCellValue(84);      sheet.createRow(1).createCell(0).setCellValue(74);      sheet.createRow(2).createCell(0).setCellValue(50);      sheet.createRow(3).createCell(0).setCellValue(51);      sheet.createRow(4).createCell(0).setCellValue(49);      sheet.createRow(5).createCell(0).setCellValue(41);        SheetConditionalFormatting sheetCF = sheet.getSheetConditionalFormatting();        //Condition 1: Cell Value Is   greater than  70   (Blue Fill)      ConditionalFormattingRule rule1 = sheetCF.createConditionalFormattingRule(ComparisonOperator.GT, "70");      PatternFormatting fill1 = rule1.createPatternFormatting();      fill1.setFillBackgroundColor(IndexedColors.BLUE.index);      fill1.setFillPattern(PatternFormatting.SOLID\_FOREGROUND);        //Condition 2: Cell Value Is  less than      50   (Green Fill)      ConditionalFormattingRule rule2 = sheetCF.createConditionalFormattingRule(ComparisonOperator.LT, "50");      PatternFormatting fill2 = rule2.createPatternFormatting();      fill2.setFillBackgroundColor(IndexedColors.GREEN.index);      fill2.setFillPattern(PatternFormatting.SOLID\_FOREGROUND);        CellRangeAddress[] regions = {              CellRangeAddress.valueOf("A1:A6")      };        sheetCF.addConditionalFormatting(regions, rule1, rule2);  } |



**2) Highlight duplicate values**

Highlight all cells which have duplicate values in observed cells.

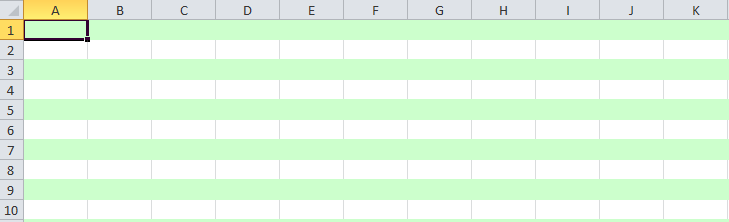
|  |
| --- |
| static void formatDuplicates(Sheet sheet) {      sheet.createRow(0).createCell(0).setCellValue("Code");      sheet.createRow(1).createCell(0).setCellValue(4);      sheet.createRow(2).createCell(0).setCellValue(3);      sheet.createRow(3).createCell(0).setCellValue(6);      sheet.createRow(4).createCell(0).setCellValue(3);      sheet.createRow(5).createCell(0).setCellValue(5);      sheet.createRow(6).createCell(0).setCellValue(8);      sheet.createRow(7).createCell(0).setCellValue(0);      sheet.createRow(8).createCell(0).setCellValue(2);      sheet.createRow(9).createCell(0).setCellValue(8);      sheet.createRow(10).createCell(0).setCellValue(6);        SheetConditionalFormatting sheetCF = sheet.getSheetConditionalFormatting();        // Condition 1: Formula Is   =A2=A1   (White Font)      ConditionalFormattingRule rule1 = sheetCF.createConditionalFormattingRule("COUNTIF($A$2:$A$11,A2)>1");      FontFormatting font = rule1.createFontFormatting();      font.setFontStyle(false, true);      font.setFontColorIndex(IndexedColors.BLUE.index);        CellRangeAddress[] regions = {              CellRangeAddress.valueOf("A2:A11")      };        sheetCF.addConditionalFormatting(regions, rule1);        sheet.getRow(2).createCell(1).setCellValue("<== Duplicates numbers in the column are highlighted.  " +              "Condition: Formula Is =COUNTIF($A$2:$A$11,A2)>1   (Blue Font)");  } |



**3) Color alternate rows in different colors**

A simple code to color each alternate row in a different color.

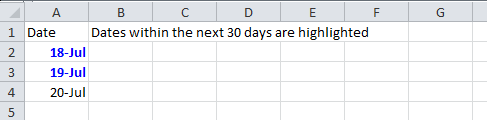
|  |
| --- |
| static void shadeAlt(Sheet sheet) {      SheetConditionalFormatting sheetCF = sheet.getSheetConditionalFormatting();        // Condition 1: Formula Is   =A2=A1   (White Font)      ConditionalFormattingRule rule1 = sheetCF.createConditionalFormattingRule("MOD(ROW(),2)");      PatternFormatting fill1 = rule1.createPatternFormatting();      fill1.setFillBackgroundColor(IndexedColors.LIGHT\_GREEN.index);      fill1.setFillPattern(PatternFormatting.SOLID\_FOREGROUND);        CellRangeAddress[] regions = {              CellRangeAddress.valueOf("A1:Z100")      };        sheetCF.addConditionalFormatting(regions, rule1);        sheet.createRow(0).createCell(1).setCellValue("Shade Alternating Rows");      sheet.createRow(1).createCell(1).setCellValue("Condition: Formula Is  =MOD(ROW(),2)   (Light Green Fill)");  } |



**4) Color amounts which are going to expire in next 30 days**

A very useful code for financial projects which keep track of dead lines.

|  |
| --- |
| static void expiryInNext30Days(Sheet sheet)  {      CellStyle style = sheet.getWorkbook().createCellStyle();      style.setDataFormat((short)BuiltinFormats.getBuiltinFormat("d-mmm"));        sheet.createRow(0).createCell(0).setCellValue("Date");      sheet.createRow(1).createCell(0).setCellFormula("TODAY()+29");      sheet.createRow(2).createCell(0).setCellFormula("A2+1");      sheet.createRow(3).createCell(0).setCellFormula("A3+1");        for(int rownum = 1; rownum <= 3; rownum++) sheet.getRow(rownum).getCell(0).setCellStyle(style);        SheetConditionalFormatting sheetCF = sheet.getSheetConditionalFormatting();        // Condition 1: Formula Is   =A2=A1   (White Font)      ConditionalFormattingRule rule1 = sheetCF.createConditionalFormattingRule("AND(A2-TODAY()>=0,A2-TODAY()<=30)");      FontFormatting font = rule1.createFontFormatting();      font.setFontStyle(false, true);      font.setFontColorIndex(IndexedColors.BLUE.index);        CellRangeAddress[] regions = {              CellRangeAddress.valueOf("A2:A4")      };        sheetCF.addConditionalFormatting(regions, rule1);        sheet.getRow(0).createCell(1).setCellValue("Dates within the next 30 days are highlighted");  } |



I am ending this **apache poi tutorial** here for keeping the post in limit.