(/)

Creating PDF Files in Java

Last modified: August 16, 2019

by baeldung (https://www.baeldung.com/author/baeldung/)

Data (https://www.baeldung.com/category/data/)

1. Introduction

In this quick article, we'll focus on creating PDF document from scratch based on popular iText and PdfBox library.

2. Maven Dependencies

Let's take a look at the Maven dependencies, which needs to be included in our project:

```
<dependency>
    <groupId>com.itextpdf</groupId>
    <artifactId>itextpdf</artifactId>
    <version>5.5.10</version>
</dependency>
<dependency>
    <groupId>org.apache.pdfbox</groupId>
    <artifactId>pdfbox</artifactId>
    <version>2.0.4</version>
</dependency>
```

The latest version of the libraries can be found here: iText (https://search.maven.org/classic/#search%7Cga%7C1%7Ca%3A%22itextpdf%22) and PdfBox (https://search.maven.org/classic/#search%7Cga%7C1%7Ca%3A%22pdfbox%22).

One extra dependency is necessary to add, in case our file will need to be encrypted. The Bounty Castle Provider package contains implementations of cryptographic algorithms and is required by both libraries:

```
<dependency>
    <groupId>org.bouncycastle</groupId>
    <artifactId>bcprov-jdk15on</artifactId>
    <version>1.56</version>
</dependency>
```

The latest version of the library can be found here: The Bounty Castle Provider (https://mvnrepository.com/artifact/org.bouncycastle/bcprov-jdk15on/1.56).

3. Overview

Both, the iText and PdfBox are java libraries used for creation/manipulation of pdf files. Although the final output of the libraries is the same, they operate in a bit different manner. Let's take a look at them.

4. Create Pdf in IText

4.1. Insert Text in Pdf

Let's have a look, at the way a new file with “Hello World” text is inserted in pdf file

```
Document document = new Document();
PdfWriter.getInstance(document, new
FileOutputStream("iTextHelloWorld.pdf"));

document.open();
Font font = FontFactory.getFont(FontFactory.COURIER, 16,
BaseColor.BLACK);
Chunk chunk = new Chunk("Hello World", font);

document.add(chunk);
document.close();
```

Creating a pdf with a use of the iText library is based on manipulating objects implementing *Elements* interface in *Document* (in version 5.5.10 there are 45 of those implementations).

The smallest element which can be added to the document and used is called *Chunk*, which is basically a string with applied font.

Additionally, *Chunk*'s can be combined with other elements like *Paragraphs*, *Section* etc. resulting in nice looking documents.

4.2. Inserting Image

The iText library provides an easy way to add an image to the document. We simply need to create an *Image* instance and add it to the *Document*.

```
Path path =
Paths.get(ClassLoader.getSystemResource("Java_logo.png").toURI());

Document document = new Document();
PdfWriter.getInstance(document, new
FileOutputStream("iTextImageExample.pdf"));
document.open();
Image img = Image.getInstance(path.toAbsolutePath().toString());
document.add(img);

document.close();
```

4.3. Inserting Table

We might face a problem when we would like to add a table to our pdf. Luckily iText provides out-of-the-box such functionality.

First what we need to do is to create a *PdfTable* object and in constructor provide a number of columns for our table. Now we can simply add new cell by calling

Now we can simply add new cell by calling the *addCell* method on the newly created table object. iText will create table rows as long as all necessary cells are defined, what it means is that once you create a table with 3 columns and add 8 cells to it, only 2 rows with 3 cells in each will be displayed.

Let's take a look at the example:

```
Document document = new Document();
PdfWriter.getInstance(document, new FileOutputStream("iTextTable.pdf"));

document.open();

PdfPTable table = new PdfPTable(3);
addTableHeader(table);
addRows(table);
addCustomRows(table);

document.add(table);
document.close();
```

We create a new table with 3 columns and 3 rows. The first row we will treat as a table header with a changed background color and border width:

```
private void addTableHeader(PdfPTable table) {
    Stream.of("column header 1", "column header 2", "column header 3")
      .forEach(columnTitle -> {
        PdfPCell header = new PdfPCell();
        header.setBackgroundColor(BaseColor.LIGHT_GRAY);
        header.setBorderWidth(2);
        header.setPhrase(new Phrase(columnTitle));
        table.addCell(header);
    });
}
```

The second row will be composed of three cells just with text, no extra formatting.

```
private void addRows(PdfPTable table) {
    table.addCell("row 1, col 1");
    table.addCell("row 1, col 2");
    table.addCell("row 1, col 3");
}
```

We can include not only text in cells but also images. Additionally, each cell might be formatted individually, in the example presented below we apply horizontal and vertical alignment adjustments:

```
private void addCustomRows(PdfPTable table)
  throws URISyntaxException, BadElementException, IOException {
    Path path =
Paths.get(ClassLoader.getSystemResource("Java_logo.png").toURI());
    Image img = Image.getInstance(path.toAbsolutePath().toString());
    img.scalePercent(10);

    PdfPCell imageCell = new PdfPCell(img);
    table.addCell(imageCell);

    PdfPCell horizontalAlignCell = new PdfPCell(new Phrase("row 2, col
2"));
    horizontalAlignCell.setHorizontalAlignment(Element.ALIGN_CENTER);
    table.addCell(horizontalAlignCell);

    PdfPCell verticalAlignCell = new PdfPCell(new Phrase("row 2, col
3"));
    verticalAlignCell.setVerticalAlignment(Element.ALIGN_BOTTOM);
    table.addCell(verticalAlignCell);
}
```

4.4. File Encryption

In order to apply permission using iText library, we need to have already created pdf document. In our example, we will use our *iTextHelloWorld.pdf* file generated previously.

Once we load the file using *PdfReader*, we need to create a *PdfStamper* which is used to apply additional content to file like metadata, encryption etc:

```
PdfReader pdfReader = new PdfReader("HelloWorld.pdf");
PdfStamper pdfStamper
  = new PdfStamper(pdfReader, new FileOutputStream("encryptedPdf.pdf"));

pdfStamper.setEncryption(
  "userpass".getBytes(),
  ".getBytes(),
  0,
  PdfWriter.ENCRYPTION_AES_256
);

pdfStamper.close();
```

In our example, we encrypted the file with two passwords. The user password ("userpass") where a user has only read-only right with no possibility to print it, and owner password ("ownerpass") that is used as master key allowing a person to have full access to pdf.

If we want to allow the user to print pdf, instead of 0 (third parameter of *setEncryption*) we can pass:

```
PdfWriter.ALLOW_PRINTING
```

Of course, we can mix different permissions like:

```
PdfWriter.ALLOW_PRINTING | PdfWriter.ALLOW_COPY
```

Keep in mind that using iText to set access permissions, we are also creating a temporary pdf which should be deleted and if not it could be fully accessible to anybody.

5. Create Pdf in PdfBox

5.1. Insert Text in Pdf

As opposite to the *iText*, the *PdfBox* library provides API which is based on stream manipulation. There are no classes like *Chunk/ Paragraph* etc. The *PDDocument* class is an in-memory Pdf representation where the user writes data by manipulating *PDPageContentStream* class.

Let's take a look at the code example:

```
PDDocument document = new PDDocument();
PDPage page = new PDPage();
document.addPage(page);

PDPageContentStream contentStream = new PDPageContentStream(document,
page);

contentStream.setFont(PDType1Font.COURIER, 12);
contentStream.beginText();
contentStream.showText("Hello World");
contentStream.endText();
contentStream.close();

document.save("pdfBoxHelloWorld.pdf");
document.close();
```

5.2. Inserting Image

Inserting images is straightforward.

First we need to load a file and create a *PDImageXObject*, subsequently draw it on the document (need to provide exact x,y coordinates).

That's all:

```
PDDocument document = new PDDocument();
PDPage page = new PDPage();
document.addPage(page);

Path path =
Paths.get(ClassLoader.getSystemResource("Java_logo.png").toURI());
PDPageContentStream contentStream = new PDPageContentStream(document,
page);
PDImageXObject image
  = PDImageXObject.createFromFile(path.toAbsolutePath().toString(),
document);
contentStream.drawImage(image, 0, 0);
contentStream.close();

document.save("pdfBoxImage.pdf");
document.close();
```

5.3. Inserting a Table

Unfortunately, *PdfBox* does not provide any out-of-box methods allowing creating tables. What we can do in such situation is to draw it manually – literally, draw each line until our drawing resembles our dreamed table.

5.4. File Encryption

PdfBox library provides a possibility to encrypt, and adjust file permission for the user. Comparing to *iText*, it does not require to use an already existing file, as we simply use *PDDocument*. Pdf file permissions are handled by *AccessPermission* class, where we can set if a user will be able to modify, extract content or print a file.

Subsequently, we create a *StandardProtectionPolicy* object which adds password-based protection to the document. We can specify two types of password. The user password, after which person will be able to open a file with applied access permissions and owner password (no limitations to the file):

```
PDDocument document = new PDDocument();
PDPage page = new PDPage();
document.addPage(page);

AccessPermission accessPermission = new AccessPermission();
accessPermission.setCanPrint(false);
accessPermission.setCanModify(false);

StandardProtectionPolicy standardProtectionPolicy
  = new StandardProtectionPolicy("ownerpass", "userpass",
accessPermission);
document.protect(standardProtectionPolicy);
document.save("pdfBoxEncryption.pdf");
document.close();
```

Our example presents a situation that if a user provides user password, the file cannot be modified and printed.

6. Conclusions

In this tutorial, we discussed ways of creating a pdf file in two popular Java libraries.

Full examples can be found in the Maven based project over on GitHub (https://github.com/eugenp/tutorials/tree/master/pdf).

1 COMMENT Oldest

View Comments

Comments are closed on this article!

COURSES

ALL COURSES (/ALL-COURSES)
ALL BULK COURSES (/ALL-BULK-COURSES)
THE COURSES PLATFORM (HTTPS://COURSES.BAELDUNG.COM)

SERIES

JAVA "BACK TO BASICS" TUTORIAL (/JAVA-TUTORIAL)
JACKSON JSON TUTORIAL (/JACKSON)
HTTPCLIENT 4 TUTORIAL (/HTTPCLIENT-GUIDE)
REST WITH SPRING TUTORIAL (/REST-WITH-SPRING-SERIES)
SPRING PERSISTENCE TUTORIAL (/PERSISTENCE-WITH-SPRING-SERIES)
SECURITY WITH SPRING (/SECURITY-SPRING)
SPRING REACTIVE TUTORIALS (/SPRING-REACTIVE-GUIDE)

ABOUT

ABOUT BAELDUNG (/ABOUT)
THE FULL ARCHIVE (/FULL_ARCHIVE)
WRITE FOR BAELDUNG (/CONTRIBUTION-GUIDELINES)
EDITORS (/EDITORS)
JOBS (/TAG/ACTIVE-JOB/)
OUR PARTNERS (/PARTNERS)
ADVERTISE ON BAELDUNG (/ADVERTISE)

TERMS OF SERVICE (/TERMS-OF-SERVICE)
PRIVACY POLICY (/PRIVACY-POLICY)
COMPANY INFO (/BAELDUNG-COMPANY-INFO)
CONTACT (/CONTACT)