iTEXT TUTORIAL

Hot Recipes for the iText Library

CHANDAN SINGH

Java Code Geeks
Java 2 Java Developers Resource Center

iText Tutorial

Contents

Preface

iText is an open source library for creating and manipulating PDF files in Java.

iText provides support for most advanced PDF features such as PKI-based signatures, 40-bit and 128-bit encryption, color correction, Tagged PDF, PDF forms (AcroForms), PDF/X, color management via ICC profiles and barcodes, and is used by several products and services, including Eclipse BIRT, Jasper Reports, Red Hat JBoss Seam, Windward Reports, and pdftk. (Source: https://en.wikipedia.org/wiki/IText)

In this ebook, we provide a series of tutorials on how we can use Itext to create a PDF and perform basic operations. We will also cover PDFWriter and PDFReader and demonstrate the different operations we can perform on PDF using the same.

About the Author

Chandan holds a degree in Computer Engineering and is a passionate software programmer. He has good experience in Java/J2EE Web-Application development for Banking and E-Commerce Domains.

Chapter 1

iText Tutorial for Beginners

In this example, we will demonstrate how we can create and maintain PDF Files using ITEXT

1.1 Why use PDF?

PDF stands for **Portable Document Format**, was a proprietary file format owned by Adobe Systems. However, it was released as an open standard in July, 2008. The PDF format is independent of the underlying software and hardware of the system on which it is viewed. The electronic documents in general also have many advantages, some of which are listed below:

- Easy to access and search
- Secure
- Ease of store.

Itext provides a number of operations to create and effectively manage PDF documents. Itext is not a end-user tool and its utilities need to be used in Programming language to perform the operations. We will look in detail at the operations using JAVA as the programming language:

1.2 Implementation

Create a new Maven project as shown below and select create simple Maven project:

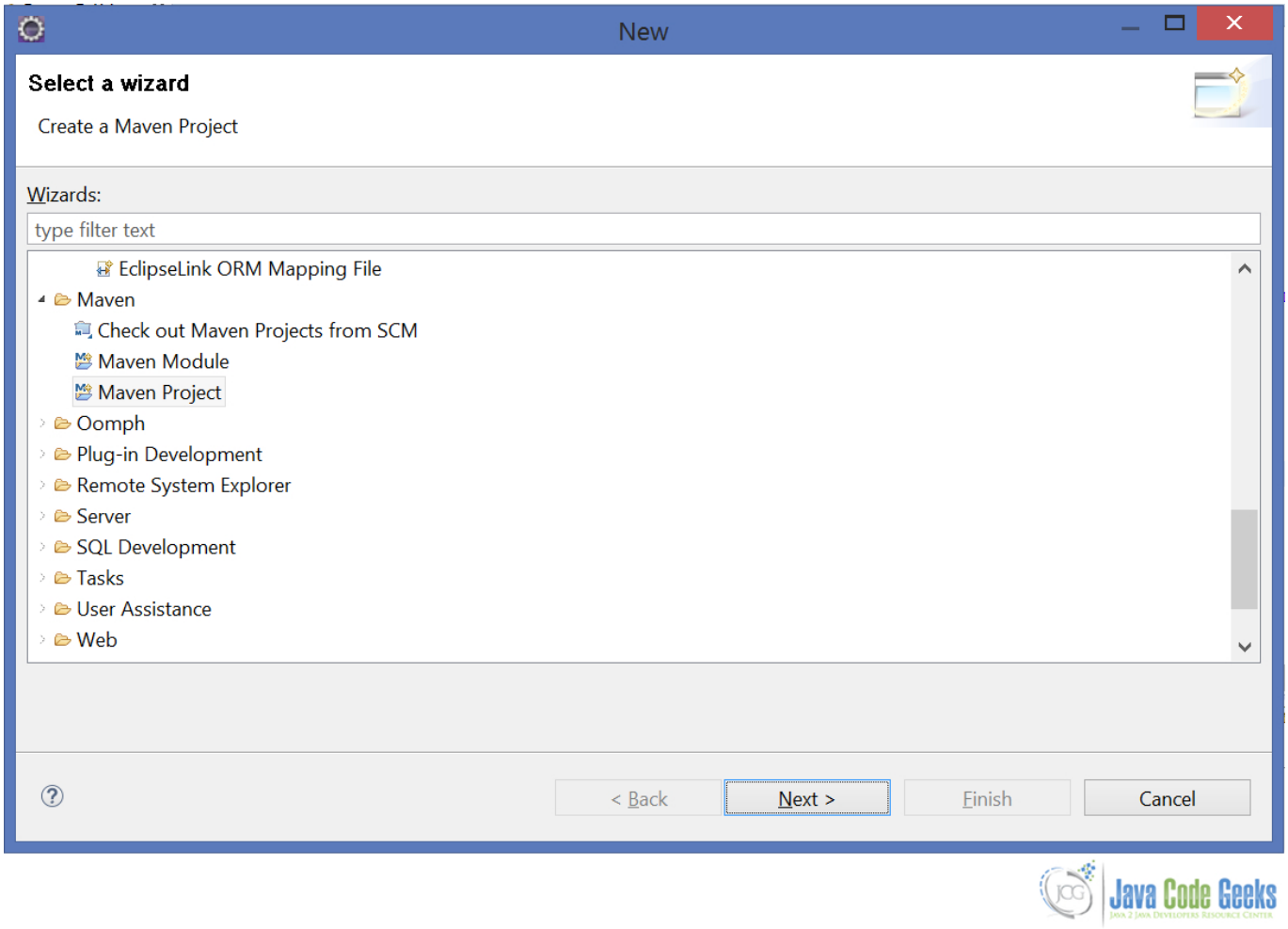

Figure 1.1: Create Maven Project

Here is the final project structure:

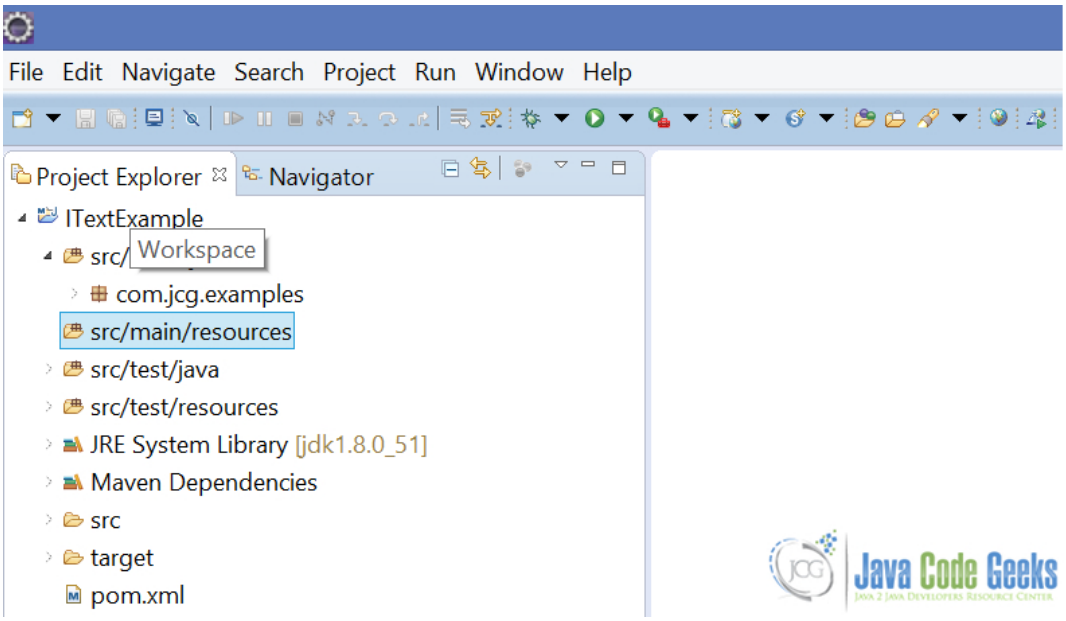

Figure 1.2: Project Structure

We will need to import the dependencies for the IText. Here is the `pom.xml`:

pom.xml

```
<project xmlns="https://maven.apache.org/POM/4.0.0" xmlns:xsi="https://www.w3.org/2001/ ↩
    XMLSchema-instance"
        xsi:schemaLocation="https://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd ↩
            /maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <groupId>ITextExample</groupId>
        <artifactId>ITextExample</artifactId>
        <version>0.0.1-SNAPSHOT</version>

        <dependencies>
                <dependency>
                        <groupId>com.itextpdf</groupId>
                        <artifactId>itextpdf</artifactId>
                        <version>5.5.6</version>
                </dependency>

        </dependencies>

</project>
```

This adds the dependency for `Itextpdf.jar` which contains the required class files to create the `PDF` document.

Now let's create a simple "Hello World" PDF:

CreatePdf.java

```
package com.jcg.examples;

import java.io.FileNotFoundException;
import java.io.FileOutputStream;

import com.itextpdf.text.Document;
import com.itextpdf.text.DocumentException;
import com.itextpdf.text.Paragraph;
```

```
import com.itextpdf.text.pdf.PdfWriter;

public class CreatePdf
{
                public static void main(String[] args) throws FileNotFoundException,  ↩
                    DocumentException
                {
                                Document document = new Document();
                                @SuppressWarnings("unused")
                                PdfWriter pdfWriter = PdfWriter.getInstance(document, new  ↩
                                    FileOutputStream("HelloWorld.pdf"));
                                document.open();

                                Paragraph paragraph = new Paragraph();
                                paragraph.add("Hello World!");

                                document.add(paragraph);
                                document.close();
                }

}
```

We create an object of `com.itextpdf.text.Document`. Then we pass this object to `com.itextpdf.text.pdf.PdfWriter` along-with the qualified file-name we want to be created. Next we create a `com.itextpdf.text.Paragraph` object and add a `Hello World!` String to it to be written to the PDF file. We append this paragraph to the document and close the document. Closing the document causes the content to be flushed and written to the file by the `PdfWriter` we created earlier. Note that, once the document has been closed, nothing can written to the document body.

Here's the output:

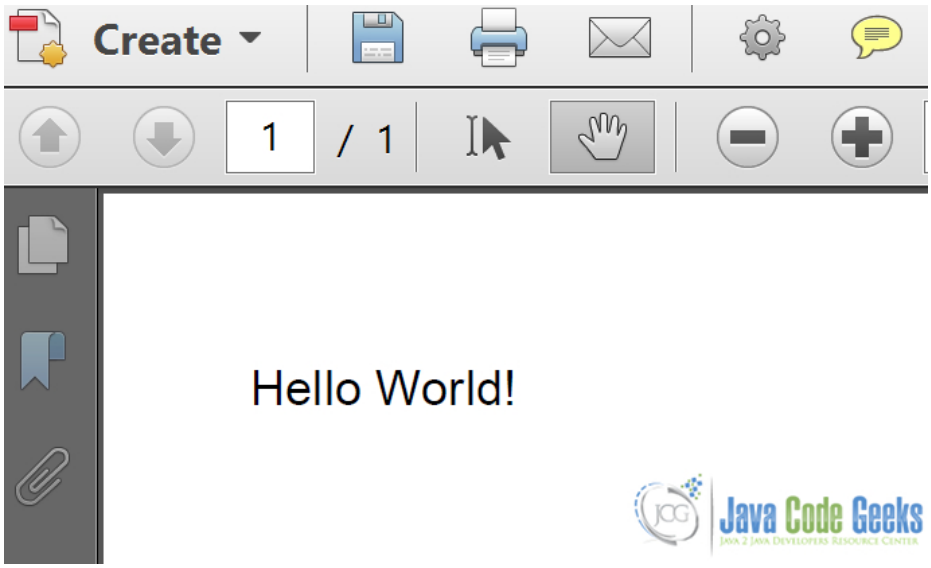

Figure 1.3: Hello World PDF

We can add any number of `com.itextpdf.text.Element` objects to the document. We will discuss a number of classes that implement the `Element` interface.

We will see an example where we will use different Elements to style our PDF document and understand the difference between them.

ChunkExample.java

```
package com.jcg.examples;

import java.io.FileNotFoundException;
import java.io.FileOutputStream;

import com.itextpdf.text.Chunk;
import com.itextpdf.text.Document;
import com.itextpdf.text.DocumentException;
import com.itextpdf.text.pdf.PdfWriter;

public class ChunkExample
{
		public static void main(String[] args) throws FileNotFoundException,  ↩
		    DocumentException
		{
				Document document = new Document();
				@SuppressWarnings("unused")
				PdfWriter pdfWriter = PdfWriter.getInstance(document, new  ↩
				    FileOutputStream("HelloWorld.pdf"));
				document.open();

				for(int count = 0;count<10;count++)
				{
						Chunk chunk = new Chunk("Hello World!!");
						document.add(chunk);
				}

				document.close();
		}

}
```

Here's how the output file looks like:

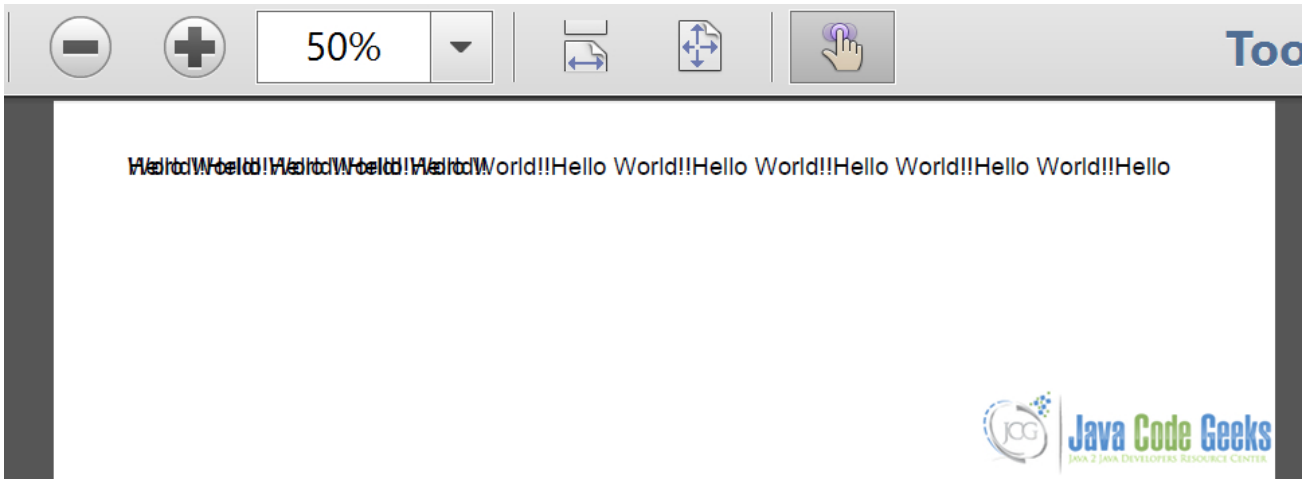

Figure 1.4: Chunk Example

As you can see the `com.itextpdf.text.Chunk` just keeps writing from right from left. If the line ends, it again starts writing on the same line. It does not know when to add a new line. That is why, it is considered better to wrap it in other elements like `Phrase` or `Paragraph` and then write to avoid overwriting as shown in the image.

we will see how we can do this using a Phrase.

PhraseExample.java

```
package com.jcg.examples;

import java.io.FileNotFoundException;
import java.io.FileOutputStream;

import com.itextpdf.text.Chunk;
import com.itextpdf.text.Document;
import com.itextpdf.text.DocumentException;
import com.itextpdf.text.Phrase;
import com.itextpdf.text.pdf.PdfWriter;

public class PhraseExample
{
		public static void main(String[] args) throws FileNotFoundException,  ↩
		    DocumentException
		{
				Document document = new Document();
				@SuppressWarnings("unused")
				PdfWriter pdfWriter = PdfWriter.getInstance(document, new  ↩
				    FileOutputStream("HelloWorld.pdf"));
				document.open();

				Phrase phrase = new Phrase();
				for(int count = 0;count<10;count++)
				{
						Chunk chunk = new Chunk("Hello World!!");
						phrase.add(chunk);

				}
				document.add(phrase);
				document.close();
		}

}
```

Here is the output:

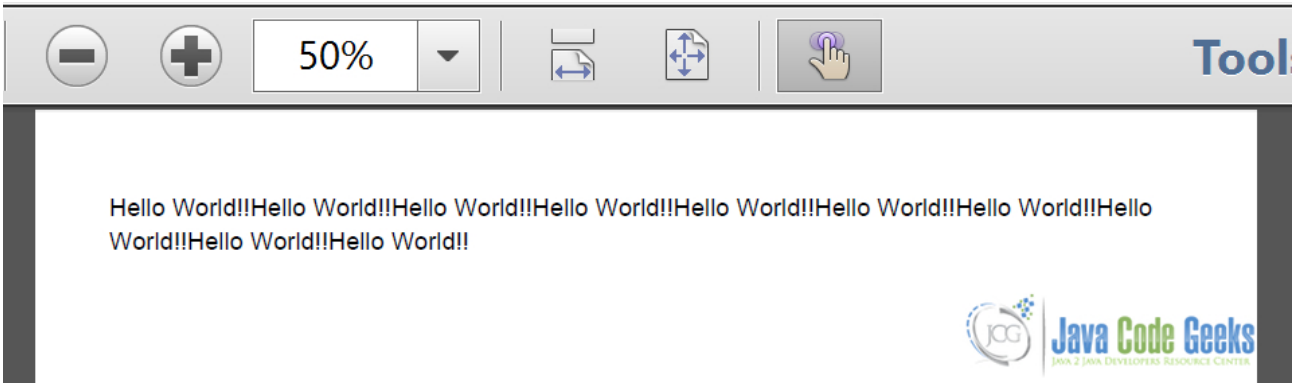

Figure 1.5: Phrase Example

As you can see in the generated PDF file, the `com.itextpdf.text.Phrase` element knows to insert a new line when it reaches end on the right side of the current line. However, Phrase cannot be used to to adjust indentation, alignment etc in the document. We will use `com.itextpdf.text.Paragraph` for adjusting indentation, alignment , spacing between two paragraphs of the same document.

Here's how a Paragraph can be formatted:

ParagraphExample.java

```
package com.jcg.examples;

import java.io.FileNotFoundException;
import java.io.FileOutputStream;

import com.itextpdf.text.Document;
import com.itextpdf.text.DocumentException;
import com.itextpdf.text.Element;
import com.itextpdf.text.Paragraph;
import com.itextpdf.text.Phrase;
import com.itextpdf.text.pdf.PdfWriter;

public class ParagraphExample
{
		public static void main(String[] args) throws FileNotFoundException, ↩
		    DocumentException
		{
				Document document = new Document();
				@SuppressWarnings("unused")
				PdfWriter pdfWriter = PdfWriter.getInstance(document, new ↩
				    FileOutputStream("HelloWorld.pdf"));
				document.open();

				Paragraph paragraph1 = new Paragraph("This is Paragraph 1") ↩
				    ;
				Paragraph paragraph2 = new Paragraph("This is Paragraph 2") ↩
				    ;
				paragraph1.setIndentationLeft(80);
				paragraph1.setIndentationRight(80);
				paragraph1.setAlignment(Element.ALIGN_CENTER);
				paragraph1.setSpacingAfter(15);
				paragraph2.setSpacingBefore(15);
				paragraph2.setAlignment(Element.ALIGN_LEFT);
				Phrase phrase = new Phrase("This is a large sentence.");
				for(int count = 0;count<10;count++)
				{
						paragraph1.add(phrase);
						paragraph2.add(phrase);
				}

				document.add(paragraph1);
				document.add(paragraph2);

				document.close();
		}

}
```

We create a couple of objects of `com.itextpdf.text.Paragraph` Element with different indentations and spacing. We indent the first on the left and right side by 80 units and align it to the centre. The second paragraph element is aligned to the left.

Personally, I think it would have been better to use java.lang.Enum to provide Alignment information instead of using `public final` integers. Using `Enums` makes for more readable and type-safe code.

Let's have a look how the paragraph looks like in a document:

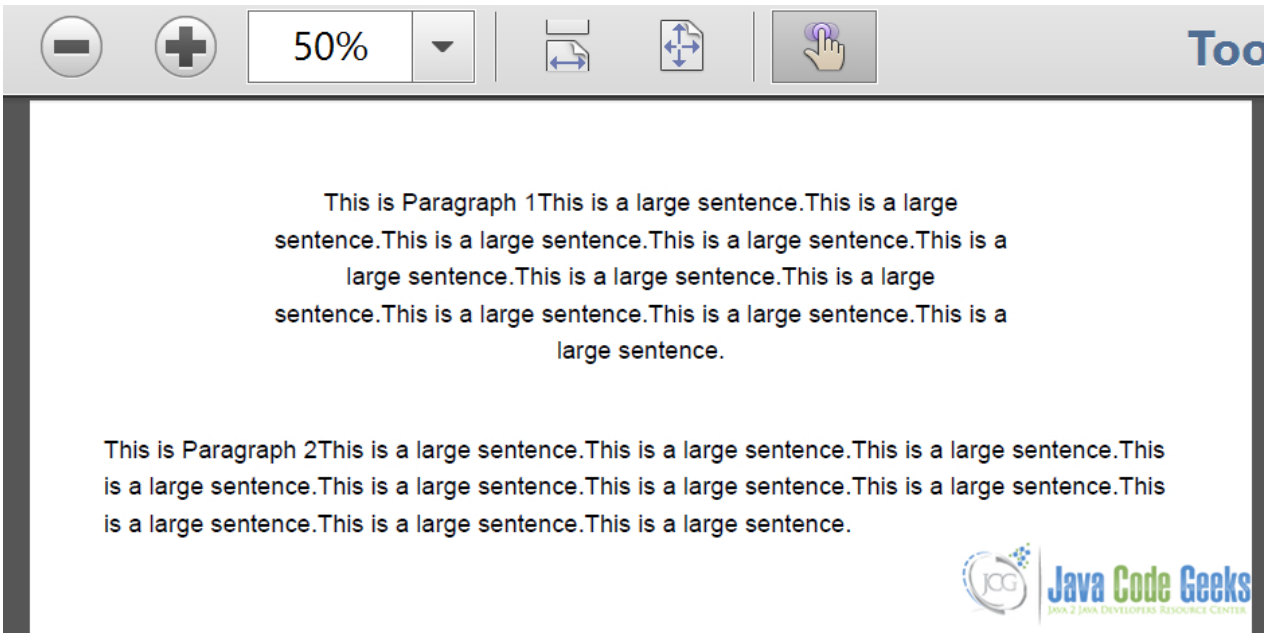

Figure 1.6: Paragraph Example

We now have a good idea of how the various text elements in a Itext look like and how they are used. Let's see now, as to how we can modify fonts in the document using the utilities offered by `IText`

FontExample.java

```
package com.jcg.examples;

import java.io.FileNotFoundException;
import java.io.FileOutputStream;

import com.itextpdf.text.Chunk;
import com.itextpdf.text.Document;
import com.itextpdf.text.DocumentException;
import com.itextpdf.text.Element;
import com.itextpdf.text.Font;
import com.itextpdf.text.Paragraph;
import com.itextpdf.text.Phrase;
import com.itextpdf.text.pdf.PdfWriter;

public class FontExample
{
                public static void main(String[] args) throws FileNotFoundException,  ↩
                    DocumentException
                {
                                Document document = new Document();
                                @SuppressWarnings("unused")
                                PdfWriter pdfWriter = PdfWriter.getInstance(document, new  ↩
                                    FileOutputStream("HelloWorld.pdf"));
                                document.open();

                                Phrase phrase = new Phrase();
                                Paragraph paragraph = new Paragraph();

                                Font timesRomanfont = new Font(Font.FontFamily.TIMES_ROMAN ↩
                                    ,16,Font.BOLDITALIC);
                                Chunk timesRomanChunk = new Chunk("Java Code Geeks", ↩
                                    timesRomanfont);
```

```
                                phrase.add(timesRomanChunk);
                                phrase.add(Chunk.NEWLINE);

                                Font strikeThruFont = new Font(Font.FontFamily.COURIER,20, ↩
                                    Font.STRIKETHRU);
                                Chunk strikeThruChunk = new Chunk("Strike Through", ↩
                                    strikeThruFont);
                                phrase.add(strikeThruChunk);
                                phrase.add(Chunk.NEWLINE);

                                Font underlineFont = new Font(Font.FontFamily.HELVETICA,20, ↩
                                    Font.UNDERLINE);
                                Chunk underlineChunk = new Chunk("This is underLined", ↩
                                    underlineFont);
                                phrase.add(underlineChunk);

                                paragraph.add(phrase);
                                paragraph.setAlignment(Element.ALIGN_CENTER);
                                document.add(paragraph);

                                document.close();
                }

}
```

Here's how the different fonts used in the `FontExample` class look like when used in a document:

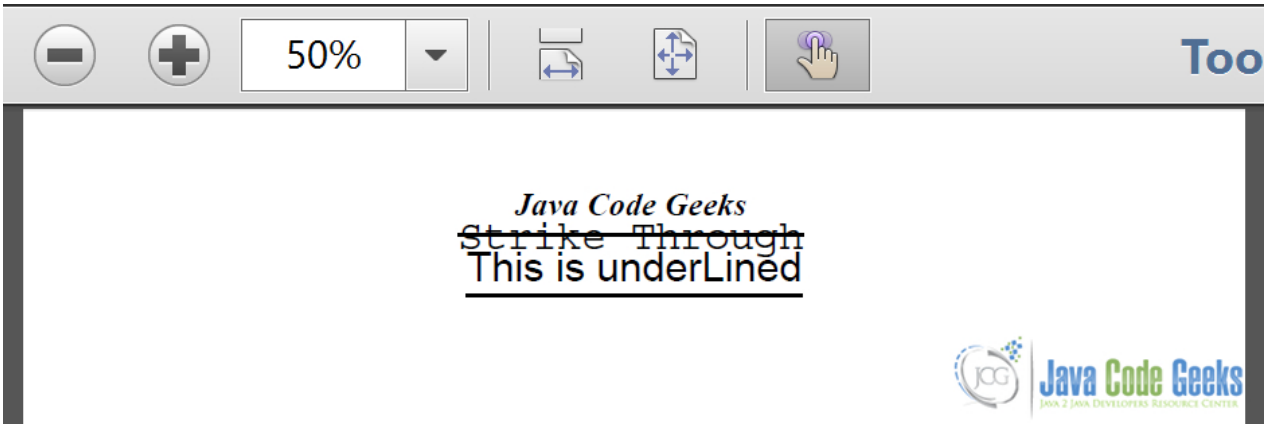

Figure 1.7: Font Example

This was all about creating the PDF. Now, we will see how we can modify the existing PDF to add the contents.

We will modify the PDF shown in Figure 1.6 to add a `JavaCodeGeeks` phrase with Roman Italic font **below** the existing content.Here's the code:

ModifyPdf.java

```
package com.jcg.examples;

import java.io.FileOutputStream;
import java.io.IOException;

import com.itextpdf.text.DocumentException;
import com.itextpdf.text.pdf.BaseFont;
import com.itextpdf.text.pdf.PdfContentByte;
import com.itextpdf.text.pdf.PdfReader;
```

```
import com.itextpdf.text.pdf.PdfStamper;

public class ModifyPdf
{
public static void main(String[] args)
{
                try
                {
                                PdfReader pdfReader = new PdfReader("HelloWorld.pdf");
                                PdfStamper pdfStamper = new PdfStamper(pdfReader, new  ↩
                                    FileOutputStream("HelloWorldModified.pdf"));
                                PdfContentByte content = pdfStamper.getUnderContent(1);

                                BaseFont bf = BaseFont.createFont(BaseFont.TIMES_ITALIC,  ↩
                                    BaseFont.CP1250, BaseFont.EMBEDDED);
        content.beginText();
        content.setFontAndSize(bf, 18);
        content.showTextAligned(PdfContentByte.ALIGN_CENTER, "JavaCodeGeeks", 250,590,0);
        content.endText();

                                pdfStamper.close();
                                pdfReader.close();
                }
                catch (IOException e)
                {
                                e.printStackTrace();
                }
                catch (DocumentException e)
                {
                                e.printStackTrace();
                }

}
}
```

And here's the output PDF:

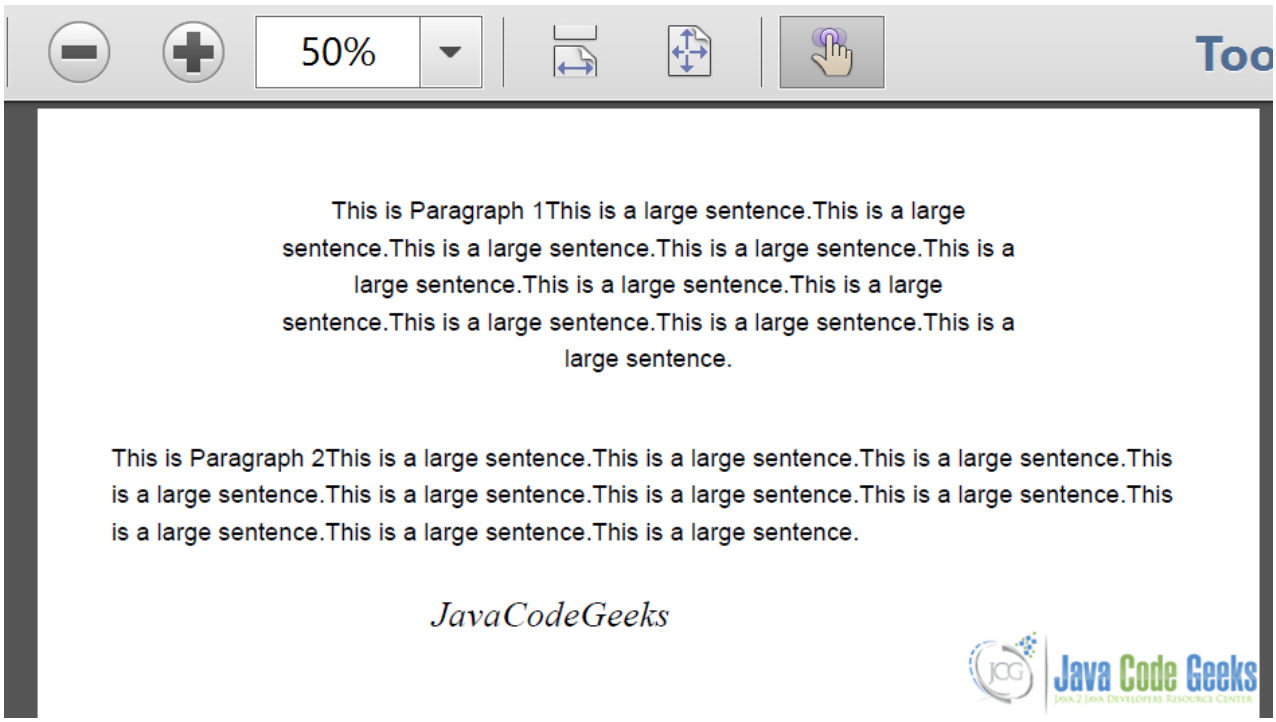

Figure 1.8: Modified PDF

We use the `com.itextpdf.text.pdf.PdfReader` class to read the existing PDF Document. The `com.itextpdf.text.pdf.PdfStamper` is then used to write extra content to the PDF Document using `com.itextpdf.text.pdf.PdfContentByte` class. The `PdfStamper#getUnderContent` is used to write the content under the existing text elements of the PDF. For writing above the existing content we use `PdfStamper#getOverContent` method.

Many a times while creating a PDF report or document, we need to put some water-mark in the pages of the document. We will put the **JavaCodeGeeks** logo to the PDF Document shown in the Figure 1.6.

AddWaterMark.java

```
package com.jcg.examples;


import java.io.FileOutputStream;
import java.io.IOException;

import com.itextpdf.text.DocumentException;
import com.itextpdf.text.Image;
import com.itextpdf.text.pdf.PdfContentByte;
import com.itextpdf.text.pdf.PdfReader;
import com.itextpdf.text.pdf.PdfStamper;


public class AddWaterMark
{
                public static void main(String[] args)
                {
                                try
                                {
                                                PdfReader pdfReader = new PdfReader(" ↩
                                                    WaterMarked.pdf");
                                                PdfStamper pdfStamper = new PdfStamper( ↩
                                                    pdfReader, new FileOutputStream(" ↩
                                                    HelloWorldModified1.pdf"));
```

```
                                        PdfContentByte content = pdfStamper. ↩
                                            getUnderContent(1);

                                        Image background = Image.getInstance("src ↩
                                            \\\\main\\\\resources\\\\JavaCodeGeeks- ↩
                                            water3.png");

                                        background.setAbsolutePosition(150f, 650f);
                                        content.addImage(background);

                                        pdfStamper.close();
                                        pdfReader.close();
                                }
                                catch (IOException e)
                                {
                                        e.printStackTrace();
                                }
                                catch (DocumentException e)
                                {
                                        e.printStackTrace();
                                }
                        }

}
```

Here's how the water-marked document looks like:

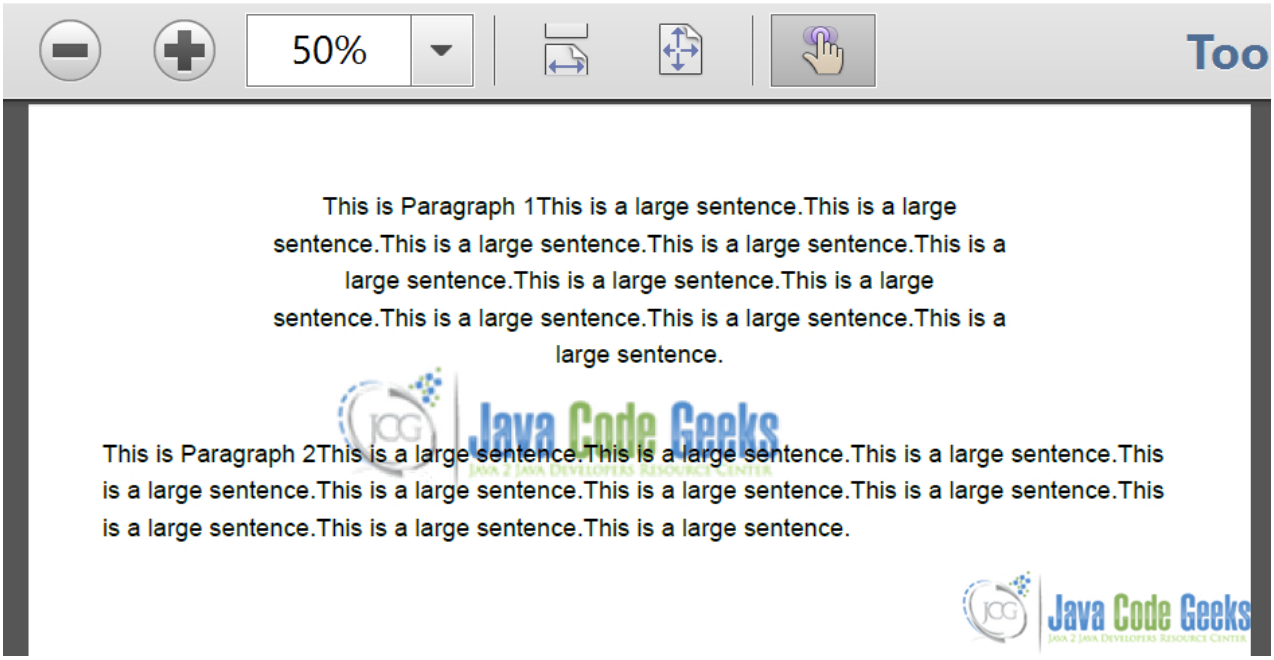

Figure 1.9: Background

In this example, we read the PDF document and add the image at a location on the document. When the requirement is to add the watermark at the creation time , we have to implement the `com.itextpdf.text.pdf.PdfPageEventHelper` and override the `onEndPage` method. The logic to add the watermark goes in this overridden method. Then set the instance of the class implementing the above interface to the `pageEvent` property of the `PdfWriter`. This adds the water-mark when each page in the PDF Document ends.

Here's the relevant snippet that demonstrates this:

```
PdfWriter writer = PdfWriter.getInstance(document, new FileOutputStream(File_Name));
writer.setPageEvent(new AddWaterMarkEvent());
//rest of the code
```

And the `AddWaterMarkEvent` would look something like this:

```
package com.jcg.examples;


import java.io.IOException;
import java.net.MalformedURLException;

import com.itextpdf.text.BadElementException;
import com.itextpdf.text.Document;
import com.itextpdf.text.DocumentException;
import com.itextpdf.text.Image;
import com.itextpdf.text.pdf.PdfPageEventHelper;
import com.itextpdf.text.pdf.PdfWriter;


class PDFBackground extends PdfPageEventHelper
{

                @Override
                public void onEndPage(PdfWriter writer, Document document)
                {
                                try
                                {
                                                Image background = Image.getInstance("src ↩
                                                    \\\\main\\\\resources\\\\JavaCodeGeeks- ↩
                                                    water3.png");
                                                float width = background.getWidth();
                                                float height = background.getHeight();
                                                writer.getDirectContentUnder().addImage( ↩
                                                    background, width, 0, 0, height, 0, 0);
                                }
                                catch (BadElementException e)
                                {
                                                e.printStackTrace();
                                }
                                catch (MalformedURLException e)
                                {
                                                e.printStackTrace();
                                }
                                catch (IOException e)
                                {
                                                e.printStackTrace();
                                }
                                catch (DocumentException e)
                                {
                                                e.printStackTrace();
                                }
                }

}
```

1.3 Download the Source Code

Here we studied how we can use IText to create and/or update our PDF Files using JAVA as the programming language.

Download

You can download the source code of this example here: **ITextExample.zip**

Chapter 2

iText PDFwriter Example

In the previous example, we studied about how we can use `Itext` to create a PDF and perform basic operations. In this example, we will dive deeper into the `https://api.itextpdf.com/itext/com/itextpdf/text/pdf/PdfWriter.html[PDFWriter]` and demonstrate the different operations we can perform on PDF using the same.

So let's get started without much ado. Here's the `pom.xml`

pom.xml

```
<project xmlns="https://maven.apache.org/POM/4.0.0" xmlns:xsi="https://www.w3.org/2001/ ↩
    XMLSchema-instance"
        xsi:schemaLocation="https://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd ↩
            /maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <groupId>ITextExample</groupId>
        <artifactId>ITextExample</artifactId>
        <version>0.0.1-SNAPSHOT</version>

        <dependencies>
                <dependency>
                        <groupId>com.itextpdf</groupId>
                        <artifactId>itextpdf</artifactId>
                        <version>5.5.6</version>
                </dependency>
                <dependency>
                        <groupId>org.bouncycastle</groupId>
                        <artifactId>bcprov-jdk15on</artifactId>
                        <version>1.52</version>
                </dependency>


        </dependencies>

</project>
```

This pom.xml has one more dependency for the `bouncycastle.jar`. We need this Jar for encrypting the PDF Files that we have created. The Jar provides implementation of the encryption algorithms we will use, to encrypt the PDF document we generate. We will see in a while how we can achieve this, let's have a look at the table of contents.

2.1 Creation of PDF

We will look at a simple code that will create a PDF with "Hello-World".

CreatePDF.java

```
package com.jcg.examples;

import java.io.FileNotFoundException;
import java.io.FileOutputStream;

import com.itextpdf.text.Document;
import com.itextpdf.text.DocumentException;
import com.itextpdf.text.Element;
import com.itextpdf.text.Paragraph;
import com.itextpdf.text.pdf.PdfWriter;

public class CreatePdf
{
                public static void main(String[] args) throws FileNotFoundException,  ↩
                    DocumentException
                {
                        Document document = new Document();
                        @SuppressWarnings("unused")
                        PdfWriter pdfWriter = PdfWriter.getInstance(document, new  ↩
                            FileOutputStream("HelloWorld.pdf"));
                        document.open();

                        Paragraph paragraph = new Paragraph();
                        paragraph.add("Hello World!");
                        paragraph.setAlignment(Element.ALIGN_CENTER);

                        Paragraph otherParagraph = new Paragraph();
                        otherParagraph.add("Welcome to JCG!");
                        otherParagraph.setAlignment(Element.ALIGN_CENTER);

                        document.add(paragraph);
                        document.add(otherParagraph);
                        document.close();
                }

}
```

The code is pretty straight-forward. We start by instantiating the `com.itextpdf.text.Document` class. Then pass this `Document` instance to the `PDFWriter` class along-with the `FileOutputStream` object. Then we add a `Paragraph` element to the document and indent it to the center. Adding an element to the document is enough to get it written to the PDF. When we close the document(line 31), all the elements written to the document get flushed and thus written to the PDF File. Once the `Document` is closed, nothing can be written to the body anymore. Doing so throws `DocumentException`.

Here's a snapshot of the document thus created:

Figure 2.1: Sample PDF

2.2 Add a Water-Mark

When generating PDF documents, many-a-times it is important to add water-mark of your organization or your client's organization to the document. We will see how we can add water-mark to our HelloWorld PDF generated above.

To add a water-mark to the PDF generated, we shall create a `event-listener` class. This event listener class will listen for the page end event and add the water-mark when it encounters such an event.

PDFEventListener.java

```
package com.jcg.examples;


import java.io.IOException;
import java.net.MalformedURLException;

import com.itextpdf.text.BadElementException;
import com.itextpdf.text.Document;
import com.itextpdf.text.DocumentException;
import com.itextpdf.text.Image;
import com.itextpdf.text.pdf.PdfPageEventHelper;
import com.itextpdf.text.pdf.PdfWriter;


class PDFEventListener extends PdfPageEventHelper
{

                @Override
                public void onEndPage(PdfWriter writer, Document document)
                {
                                try
                                {
                                                Image background = Image.getInstance("src\\\\main ↩
                                                    \\\\resources\\\\JavaCodeGeeks-water3.png");
                                                float width = background.getWidth();
                                                float height = background.getHeight();
                                                writer.getDirectContentUnder().addImage(background, ↩
                                                     width, 0, 0, height, 0, 0);
                                }
```

```
                                catch (BadElementException e)
                                {
                                        e.printStackTrace();
                                }
                                catch (MalformedURLException e)
                                {
                                        e.printStackTrace();
                                }
                                catch (IOException e)
                                {
                                        e.printStackTrace();
                                }
                                catch (DocumentException e)
                                {
                                        e.printStackTrace();
                                }
                        }

}
```

Next we need to register this event listener with the `PdfWriter` class.

```
PdfWriter pdfWriter = PdfWriter.getInstance(document, new FileOutputStream("HelloWorld.pdf" ↩
    ));
pdfWriter.setPageEvent(new PDFEventListener());
```

Here's how the book-marked document looks like :

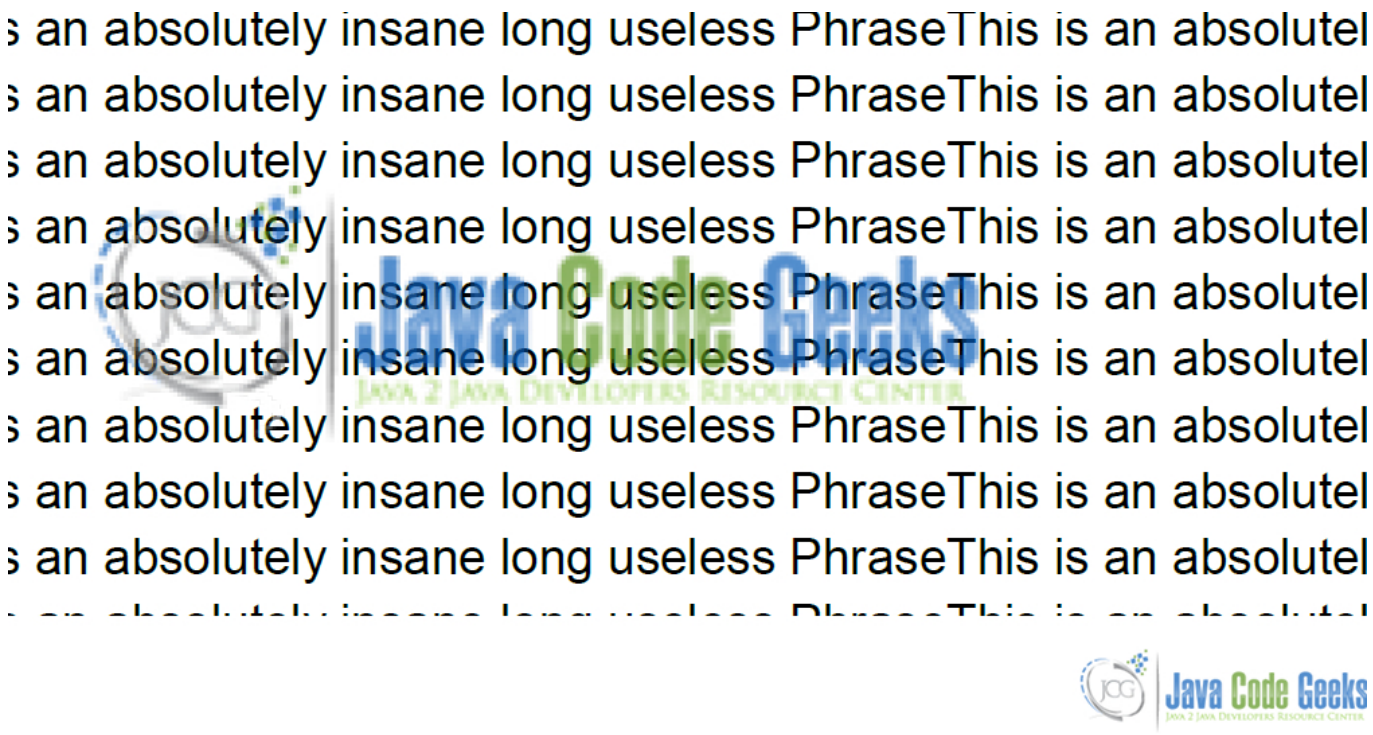

Figure 2.2: WaterMark Document

Note: Decreasing the opacity of the image prior to using it in the document will improve the aesthetics of the document.

2.3 Attach a File to PDF

In this section we will demonstrate how we can attach a file to a PDF document while creating it. Let's see the code :

AddAttachment.java

```
package com.jcg.examples;


import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;

import com.itextpdf.text.Document;
import com.itextpdf.text.DocumentException;
import com.itextpdf.text.Element;
import com.itextpdf.text.Paragraph;
import com.itextpdf.text.pdf.PdfFileSpecification;
import com.itextpdf.text.pdf.PdfWriter;


public class AddAttachment
{
                public static void main(String[] args) throws FileNotFoundException,  ↩
                    DocumentException
                {
                                Document document = new Document();
                                PdfWriter pdfWriter = PdfWriter.getInstance(document, new  ↩
                                    FileOutputStream("HelloWorld.pdf"));
                                document.open();
                                File file  = new File("HelloWorld1.pdf");
                                if(file.exists())
                                try
                                {
                                                PdfFileSpecification fileSpecification =  ↩
                                                    PdfFileSpecification.fileEmbedded( ↩
                                                    pdfWriter, file.getAbsolutePath(),file. ↩
                                                    getName(), null);

                                                pdfWriter.addFileAttachment("Sample  ↩
                                                    Attachment", fileSpecification);
                                }
                                catch (IOException e)
                                {
                                                e.printStackTrace();
                                }


                                Paragraph paragraph = new Paragraph();
                                paragraph.add("Hello World!");
                                paragraph.add("Welcome to JCG!");
                                paragraph.setAlignment(Element.ALIGN_CENTER);

                                document.add(paragraph);
                                document.close();
                }

}
```

We get the `com.itextpdf.text.pdf.PdfFileSpecification` instance by passing the source of the file and the description which we want to see in the attachment section of the PDF. Then we attach the Specification to the writer using

`PdfWriter#addFileAttachment` method which in turn attaches to the `Document` body. One important thing to note here is that the document should be open for writing before get the instance of the `PdfFileSpecification`, else the document body is null. This throws `NullPointerException` when the attachment object is added to the body.

Here's the attached document snapshot:

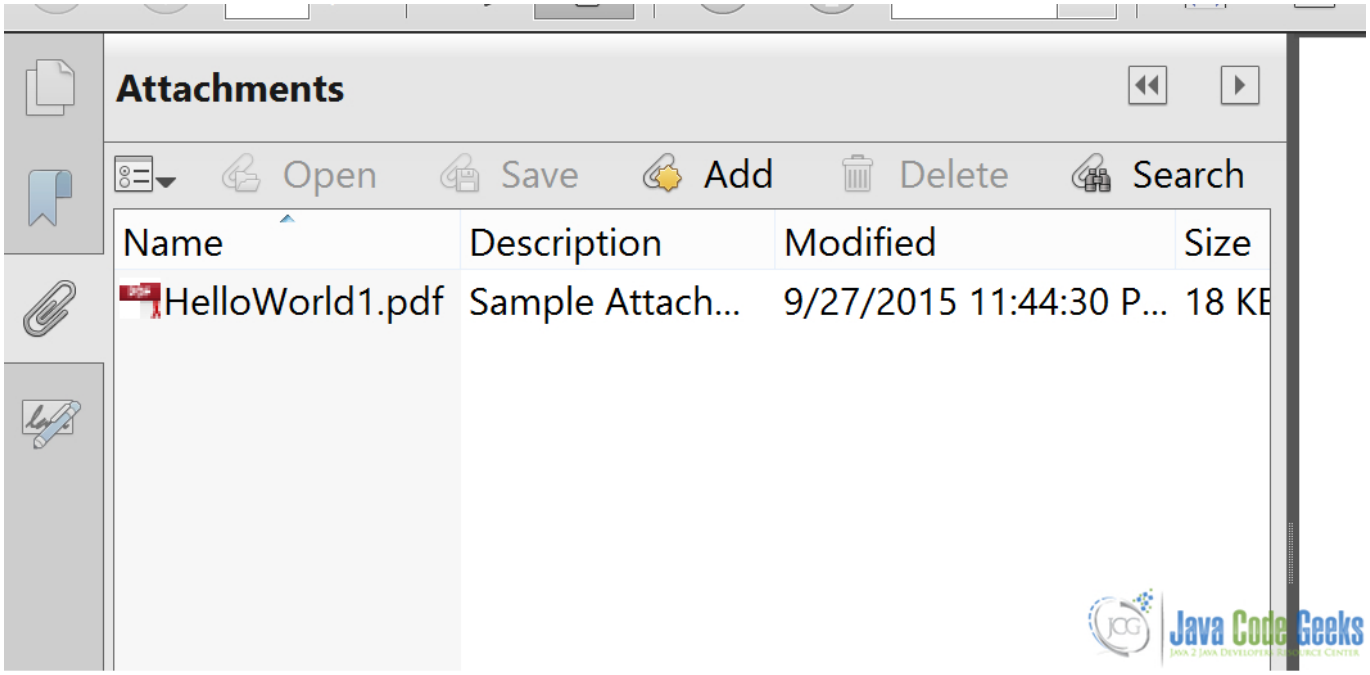

Figure 2.3: Attachment to a PDF

2.4 Encryption of the PDF

One of the most common operation we may want to perform on the PDF we generate is to protect them from unauthorized eyes.The best way to achieve this is to password protect the file. In this section, we will see how we can encrypt the PDF Files we generate using the `PdfWriter` class.

PdfEncrytpion.java

```
package com.jcg.examples;


import java.io.FileNotFoundException;
import java.io.FileOutputStream;

import com.itextpdf.text.Document;
import com.itextpdf.text.DocumentException;
import com.itextpdf.text.Paragraph;
import com.itextpdf.text.pdf.PdfWriter;


public class PdfEncrytpion
{
                public static void main(String[] args)
                {
                                try
                                {
                                                Document document = new Document();
```

```
                                        PdfWriter pdfWriter = PdfWriter.getInstance ↩
                                            (document, new FileOutputStream(" ↩
                                            HelloWorld.pdf"));
                                        pdfWriter.setEncryption("chandan".getBytes ↩
                                            (), "chandansingh".getBytes(), PdfWriter ↩
                                            .ALLOW_ASSEMBLY, PdfWriter. ↩
                                            ENCRYPTION_AES_256);
                                        document.open();

                                        Paragraph paragraph = new Paragraph();
                                        paragraph.add("Hello World!");

                                        document.add(paragraph);
                                        document.close();
                                }
                                catch (FileNotFoundException | DocumentException e)
                                {
                                        e.printStackTrace();
                                }
                        }

}
```

To encrypt a pdf file we need to set the encryption option for the document using the `PdfWriter` class. We use the `PdfWriter.setEncryption` method to do this. Here's is its signature:

```
void com.itextpdf.text.pdf.PdfWriter.setEncryption(byte[] userPassword, byte[]  ↩
    ownerPassword, int permissions, int encryptionType) throws DocumentException
```

The user password recognizes the consumers(by default readers) of the document. The owner password recgnizes the creater or the owner of the document. The creator may also choose the operations that are to be permitted once the document is opened. The operations can be multi-selected by ORing the different operations provided by the `PdfWriter` class. We chose the encryption Type to `AES 256 bytes`.

Here's the password prompt when you attempt to open the PDF :

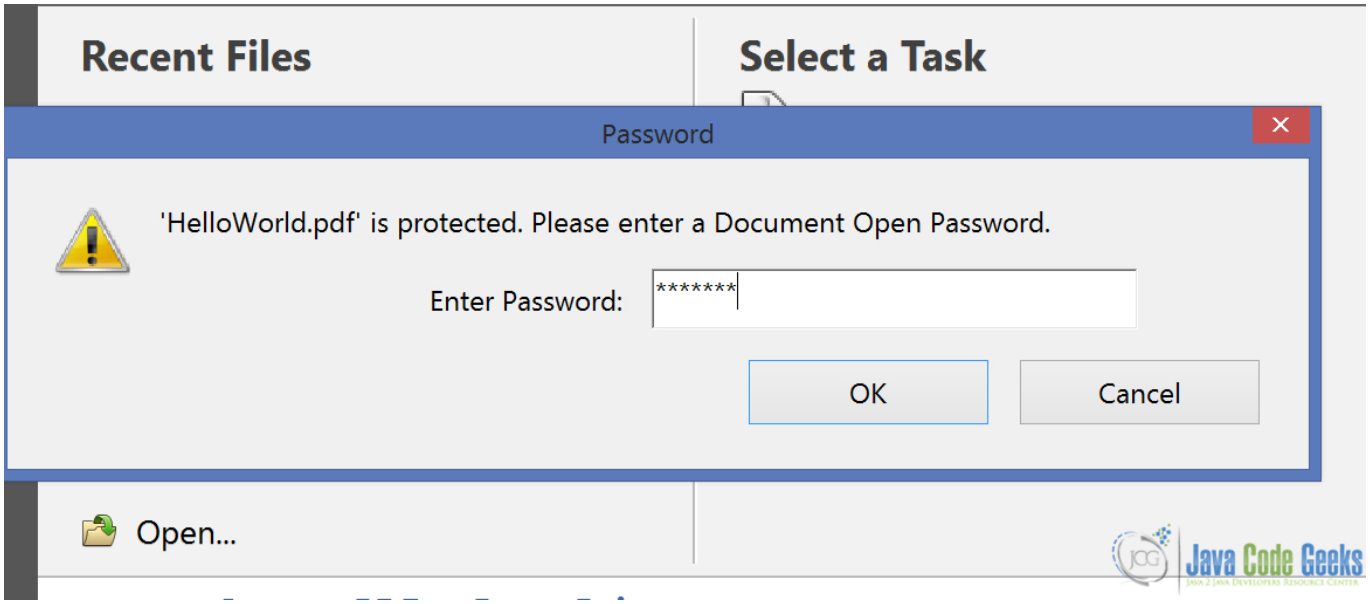

Figure 2.4: Encrypted PDF

2.5 Download the Source Code

We studied how we can create PDF using `PDFWriter` class from IText and the different operations supported by it.

Download

You can download the source code of this example here: **PDFWriterExample.zip**

Chapter 3

iText PDFReader Example

In the previous example, we studied how we can use `IText` to create and manage PDF files. In this example, we will see how we can use IText to read the PDF files in our application.

We will use the PDF files created in the previous examples to read and decrypt. The reader may download the source files from the previous example.

3.1 Project Set-up

Let's get started by creating a simple Maven project. Now, import the maven dependencies using the below `pom.xml` :

pom.xml

```
<project xmlns="https://maven.apache.org/POM/4.0.0" xmlns:xsi="https://www.w3.org/2001/ ↩
    XMLSchema-instance"
        xsi:schemaLocation="https://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd ↩
            /maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <groupId>ITextExample</groupId>
        <artifactId>ITextExample</artifactId>
        <version>0.0.1-SNAPSHOT</version>

        <dependencies>
                <dependency>
                        <groupId>com.itextpdf</groupId>
                        <artifactId>itextpdf</artifactId>
                        <version>5.5.6</version>
                </dependency>
                <dependency>
                        <groupId>org.bouncycastle</groupId>
                        <artifactId>bcprov-jdk15on</artifactId>
                        <version>1.52</version>
                </dependency>


        </dependencies>

</project>
```

Now the project setup is complete and we can start with reading the PDF files.

3.2 Read a simple PDF

Here's a simple class that reads the PDF File and prints it out in the console and also writes to a separate PDF File.

ReadPdf.java

```
package com.jcg.examples;

import java.io.FileOutputStream;
import java.io.IOException;

import com.itextpdf.text.DocumentException;
import com.itextpdf.text.pdf.BaseFont;
import com.itextpdf.text.pdf.PdfContentByte;
import com.itextpdf.text.pdf.PdfReader;
import com.itextpdf.text.pdf.PdfStamper;

public class ReadPdf
{
                public static void main(String[] args)
                {
                                try
                                {
                                                PdfReader pdfReader = new PdfReader("HelloWorld.pdf ↩
                                                    ");
                                                PdfStamper pdfStamper = new PdfStamper(pdfReader,  ↩
                                                    new FileOutputStream("Rewritten HelloWorld.pdf") ↩
                                                    );
                                                PdfContentByte content = pdfStamper.getUnderContent ↩
                                                    (1);//1 for the first page
                                                BaseFont bf = BaseFont.createFont(BaseFont. ↩
                                                    TIMES_ITALIC, BaseFont.CP1250, BaseFont.EMBEDDED ↩
                                                    );
                                                content.beginText();
                                                content.setFontAndSize(bf, 18);
                                                content.showTextAligned(PdfContentByte.ALIGN_CENTER ↩
                                                    , "JavaCodeGeeks", 250,650,0);
                                                content.endText();

                                                pdfStamper.close();
                                                pdfReader.close();
                                }
                                catch (IOException e)
                                {
                                                e.printStackTrace();
                                }
                                catch (DocumentException e)
                                {
                                                e.printStackTrace();
                                }
                }
}
```

We create an instance of `com.itextpdf.text.pdf.PdfReader` class by passing the Filename of the PDF we wish to read. Then we pass the instance of this class to `com.itextpdf.text.pdf.PdfStamper` which creates a new PDF file and adds the content of the existing file along-with the extra Text we added. It is possible to add images and files in a similar fashion. The `com.itextpdf.text.pdf.PdfContentByte` class is used to get the exact location where the file is to be modified like `page number` , under the existing content, over the existing content, x & y pivot positions etc. It also applies proper encoding to the Fonts we have selected which are to be written to the PDF file.

Here is the sample output of the modified PDF :

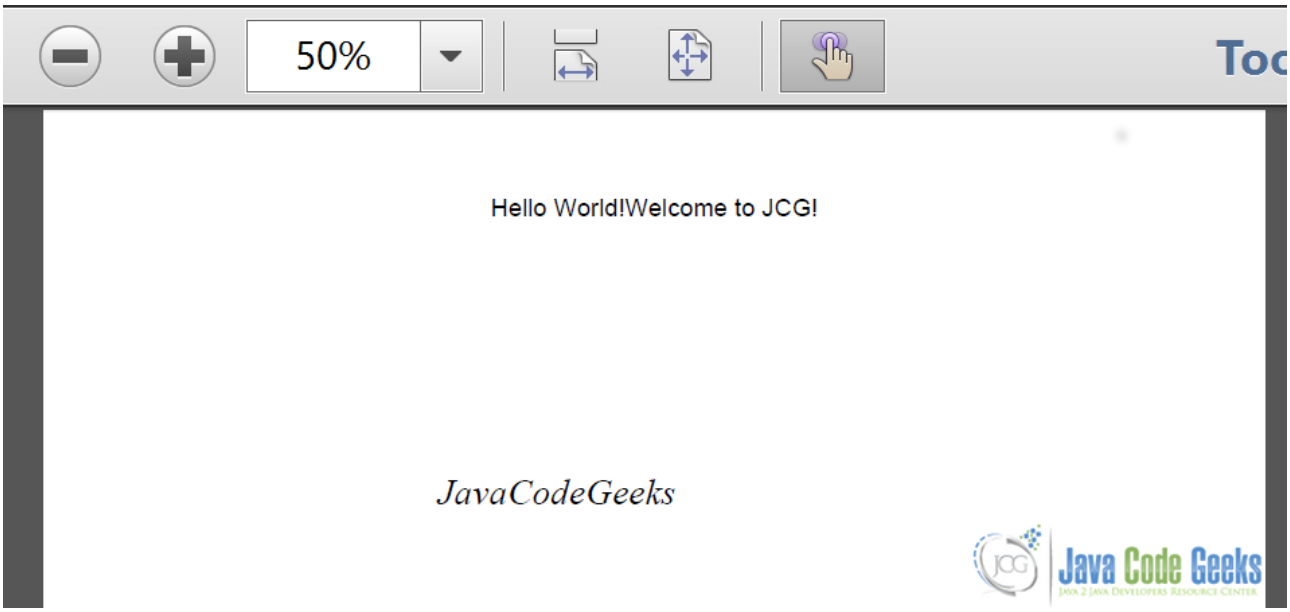

Figure 3.1: Read PDF Using Itext

3.3 Extract a File from PDF

In the previous example, we saw how we can attach a file to the PDF document. In this section we will see how we can extract an attached file from the PDF.

Here's the code for it:

ExtractAttachment.java

```
package com.jcg.examples;


import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.Set;

import com.itextpdf.text.pdf.PRStream;
import com.itextpdf.text.pdf.PdfArray;
import com.itextpdf.text.pdf.PdfDictionary;
import com.itextpdf.text.pdf.PdfName;
import com.itextpdf.text.pdf.PdfReader;


public class ExtractAttachment
{

                private static final String FILE_NAME = "HelloWorld.pdf";

                public static void main(String[] args)
                {
                                try
                                {
                                                PdfReader pdfReader = new PdfReader(FILE_NAME);
                                                PdfDictionary catalog = pdfReader.getCatalog();
```

```
                                        PdfDictionary names = catalog.getAsDict(PdfName. ↩
                                            NAMES);
                                        PdfDictionary embeddedFiles = names.getAsDict( ↩
                                            PdfName.EMBEDDEDFILES);
                                        PdfArray embeddedFilesArray = embeddedFiles. ↩
                                            getAsArray(PdfName.NAMES);
                                        extractFiles(pdfReader, embeddedFilesArray);
                                }
                                catch (IOException e)
                                {
                                        e.printStackTrace();
                                }
                        }

                        private static void extractFiles(PdfReader pdfReader, PdfArray filespecs)
                        {
                                PdfDictionary files = filespecs.getAsDict(1);
                                PdfDictionary refs = files.getAsDict(PdfName.EF);
                                PRStream prStream = null;
                                FileOutputStream outputStream = null;
                                String filename = "";
                                Set keys= refs.getKeys();
                                try
                                {
                                        for (PdfName key : keys)
                                        {
                                                prStream = (PRStream) PdfReader. ↩
                                                    getPdfObject(refs.getAsIndirectObject( ↩
                                                    key));
                                                filename = files.getAsString(key).toString ↩
                                                    ();
                                                outputStream = new FileOutputStream(new  ↩
                                                    File(filename));
                                                outputStream.write(PdfReader.getStreamBytes ↩
                                                    (prStream));
                                                outputStream.flush();
                                                outputStream.close();
                                        }
                                }
                                catch (FileNotFoundException e)
                                {
                                        e.printStackTrace();
                                }
                                catch (IOException e)
                                {
                                        e.printStackTrace();
                                }
                                finally
                                {
                                        try
                                        {
                                                if (outputStream != null)
                                                        outputStream.close();
                                        }
                                        catch (IOException e)
                                        {
                                                e.printStackTrace();
                                        }
                                }
                        }
}
```

We start of extraction by creating the reading the PDF in the `PdfReader` class. Then we extract the catalog of the document via the reader in the `com.itextpdf.text.pdf.PdfDictionary` object. From the document catalog, we extract the array of attached documents and pass-on the `pdfreader` and attached document array to the `extractFiles` method.

This method gets a `java.util.Set` object from the array and creates a new file with the same name as the attached file. We iterate over this Set i.e. once for each file in the attachment Set. We get the content of the attached file in the `com.itextpdf.text.pdf.PRStream` object using the `PdfReader#getPdfObject` method. We pass the `key` as the current element taken from the Set.

3.4 Read an encrypted PDF

Reading an encrypted pdf is almost similar as reading a plain, non-encypted PDF. We just need to use a different version of `PdfReader` constructor.

```
com.itextpdf.text.pdf.PdfReader.PdfReader(String filename, byte[] ownerPassword) throws
    IOException
```

In this constructor, we pass the owner password we used while creating the PDF document as a byte array.

ReadEncryptedPdf.java

```
package com.jcg.examples;

import java.io.IOException;

import com.itextpdf.text.pdf.PdfReader;

public class ReadEncryptedPdf
{
                public static void main(String[] args)
                {
                                try
                                {
                                                byte[] ownerPassword = "ownerPassword".
                                                    getBytes();
                                                PdfReader pdfReader = new PdfReader("
                                                    EncryptedHelloWorld.pdf",ownerPassword);
                                                System.out.println("Is the PDF Encrypted "+
                                                    pdfReader.isEncrypted());
                                                System.out.println("File is opened with
                                                    full permissions : "+pdfReader.
                                                    isOpenedWithFullPermissions());
                                                System.out.println("File length is : "+
                                                    pdfReader.getFileLength());
System.out.println("File is tampered? "+pdfReader.isTampered());

                                                pdfReader.close();
                                }
                                catch (IOException e)
                                {
                                                e.printStackTrace();
                                }
                }
}
```

The `pdfReader#isEncrypted()` method returns true if the Document opened by this instance is encrypted.

The `isOpenedWithFullPermissions` is used to check if the document is opened with full permission i.e to read write and modify. If the document is not encrypted one, this method returns `true` . The `isTampered()` method is used to check if the file was modified.

Note: If the tampered flag is set to true, it cannot be used in a `com.itextpdf.text.pdf.PdfStamper`.

Opening such a tampered reader in a `Pdfstamper` will throw a `com.itextpdf.text.DocumentException` with message as `"the original document was reused read it again from file"`. Providing a wrong password will lead to `com.itextpdf.text.BadPasswordException` when passing the reader to Pdfstamper class.

Here's the output of the attempt :

```
Is the PDF Encrypted : true
File is opened with full permissions : true
File length is : 1393
File is tampered? false
```

3.5 Download the Source Code

We studied how we can read a PDF using `PdfReader` class from IText and the different operations that could be performed on the PDF document.

Download

You can download the source code of this example here: **ItextPdfReaderExample.zip**

Chapter 4

iText Rectangle Example

In the past examples,we studied about the PDFWriter and PDFReader examples. In this example, we will demonstrate how we can create an `Itext Rectangle` and use it in our PDF document.

4.1 Setup the Project

Let's setup the project by creating a simple Maven project and selecting the skip archetype selection. Update the contents of `pom.xml` with the contents of the file below:

pom.xml

```
<project xmlns="https://maven.apache.org/POM/4.0.0" xmlns:xsi="https://www.w3.org/2001/ ↩
    XMLSchema-instance"
        xsi:schemaLocation="https://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd ↩
            /maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <groupId>ITextExample</groupId>
        <artifactId>ITextExample</artifactId>
        <version>0.0.1-SNAPSHOT</version>

        <dependencies>
                <dependency>
                        <groupId>com.itextpdf</groupId>
                        <artifactId>itextpdf</artifactId>
                        <version>5.5.6</version>
                </dependency>
                <dependency>
                        <groupId>org.bouncycastle</groupId>
                        <artifactId>bcprov-jdk15on</artifactId>
                        <version>1.52</version>
                </dependency>

        </dependencies>

</project>
```

Now that the project is setup, let's create a Rectangle in the PDF Document using the `IText` library.

CreateRectangle.java

```
package com.jcg.examples;

import java.io.FileNotFoundException;
import java.io.FileOutputStream;
```

```
import com.itextpdf.text.BaseColor;
import com.itextpdf.text.Document;
import com.itextpdf.text.DocumentException;
import com.itextpdf.text.Rectangle;
import com.itextpdf.text.pdf.PdfContentByte;
import com.itextpdf.text.pdf.PdfWriter;

public class CreateRectangle
{
                public static void main(String[] args)
                {
                                try
                                {
                                                Document document = new Document();
                                                PdfWriter writer = PdfWriter.getInstance( ↩
                                                    document, new FileOutputStream(" ↩
                                                    Rectagled.pdf"));
                                                document.open();

                                                PdfContentByte contentByte = writer. ↩
                                                    getDirectContent();
                                                Rectangle rectangle = new Rectangle(36, 36, ↩
                                                     559, 806);
                                                rectangle.setBorder(Rectangle.BOX);
                                                contentByte.setColorStroke(BaseColor.BLACK) ↩
                                                    ;
                                                rectangle.setBorderWidth(2);
                                                contentByte.rectangle(rectangle);
                                                document.close();
                                }
                                catch (FileNotFoundException e)
                                {
                                                e.printStackTrace();
                                }
                                catch (DocumentException e)
                                {
                                        e.printStackTrace();
                                }
                }
}
```

We create an instance of `com.itextpdf.text.Document` class. Then we create a reference to the underlying `com.itextpdf.text.pdf.PdfContentByte` object and create a `com.itextpdf.text.Rectangle` Object on it. The `Rectangle` element can be customized like setting its border color, filing it with a particular color, adjusting text to fit inside the rectangle. In the example above, I have set the border Color to Black. We can create specific colors by using the `com.itextpdf.text.BaseColor` class and passing the specified values `R-G-B` color values.

Here's how the Rectangle looks like in a document.

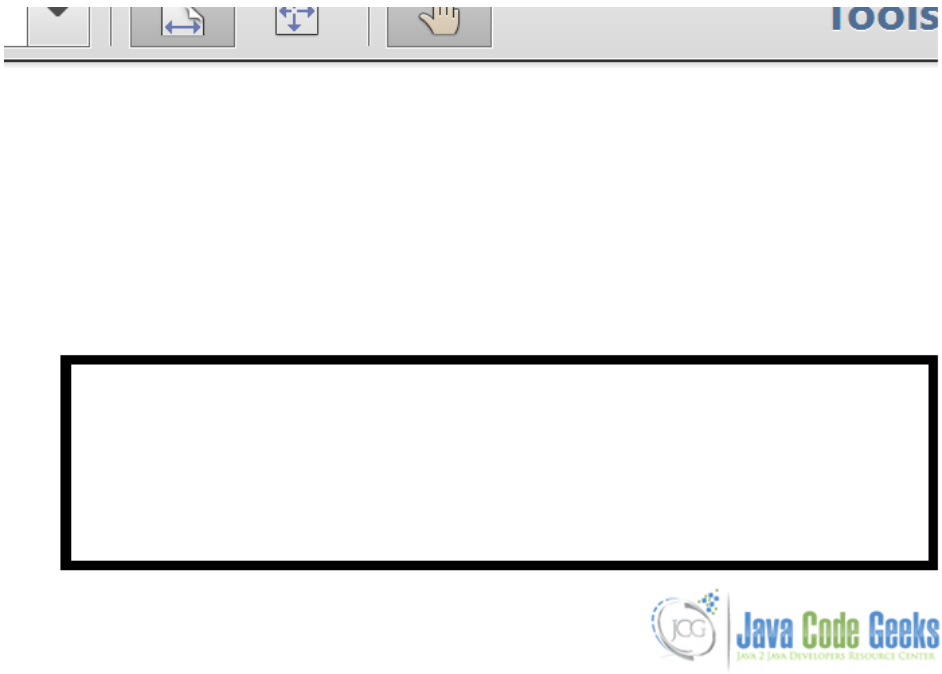

Figure 4.1: Rectangle

Here's another code snippet that fills up the rectangle with the specified color :

```
package com.jcg.examples;

import java.io.FileNotFoundException;
import java.io.FileOutputStream;

import com.itextpdf.text.BaseColor;
import com.itextpdf.text.Document;
import com.itextpdf.text.DocumentException;
import com.itextpdf.text.pdf.PdfContentByte;
import com.itextpdf.text.pdf.PdfWriter;

public class FillRectangle
{
                public static void main(String[] args)
                {
                                try
                                {
                                                Document document = new Document();
                                                PdfWriter writer = PdfWriter.getInstance( ↩
                                                    document, new FileOutputStream(" ↩
                                                    Rectangled.pdf"));
                                                document.open();
                                                PdfContentByte contentByte = writer. ↩
                                                    getDirectContent();
                                                contentByte.rectangle(186, 186, 159, 150);
                                                contentByte.setColorFill(BaseColor.CYAN);
                                                contentByte.fill();
                                                document.close();
                                }
                                catch (FileNotFoundException e)
                                {
                                                e.printStackTrace();
                                }
```

```
                                catch (DocumentException e)
                                {
                                        e.printStackTrace();
                                }
                }
}
```

To fill a rectangle we use the `PdfContentByte#setColorFill` method to set the Color to fill the background and then call the `PdfContentByte#fill()` method to actually fill up the method.

Here's how the output looks like :

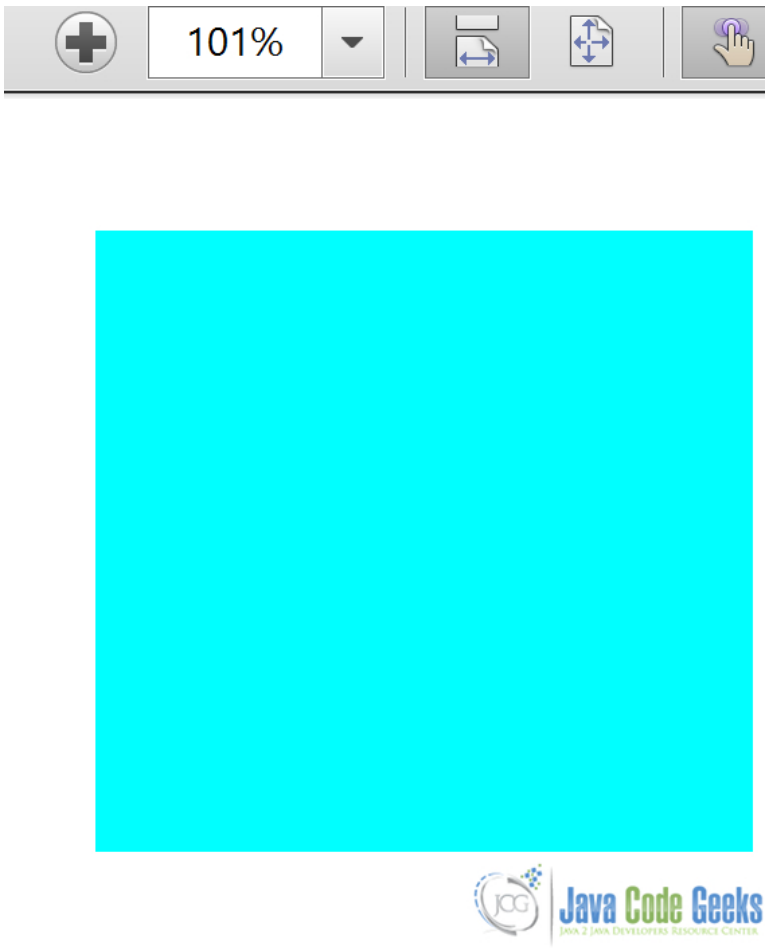

Figure 4.2: ColorFill Rectangle

Similarly, we can use the `com.itextpdf.text.pdf.ColumnText#showTextAligned` method to place fonts and other PDF Objects into the Rectangle.

4.2 Download the Source Code

Here we studied how we can create an `IText Rectangle` and modify it to fit our requirements for the document.

Download

You can download the source code of this example here: **ItextRectangleExample.zip**

Chapter 5

iText PDFstamper Example

In the previous examples we have seen how the PdfReader and Pdfwriter classes in the IText library work. In this example, we will demonstrate the working of another important class, `https://api.itextpdf.com/itext/com/itextpdf/text/pdf/PdfStamper.html[PDFStamper]`.

`PDFStamper` class is be used to modify existing PDF document by adding extra content to the pages. The extra content are the objects supported by the `PdfContentByte`. We will see how the objects can be added using the PDFStamper.

5.1 Setup the Project

Let's setup the project by creating a simple Maven project and selecting the skip archetype selection. Update the contents of `pom.xml` with the contents of the file below:

pom.xml

```
<project xmlns="https://maven.apache.org/POM/4.0.0" xmlns:xsi="https://www.w3.org/2001/ ↩
    XMLSchema-instance"
        xsi:schemaLocation="https://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd ↩
            /maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <groupId>ITextExample</groupId>
        <artifactId>ITextExample</artifactId>
        <version>0.0.1-SNAPSHOT</version>

        <dependencies>
                <dependency>
                        <groupId>com.itextpdf</groupId>
                        <artifactId>itextpdf</artifactId>
                        <version>5.5.6</version>
                </dependency>
                <dependency>
                        <groupId>org.bouncycastle</groupId>
                        <artifactId>bcprov-jdk15on</artifactId>
                        <version>1.52</version>
                </dependency>
        </dependencies>

</project>
```

This will import the Maven dependencies for the project. Now we are all set for the example.

Here's a simple HelloWorld PDF document :

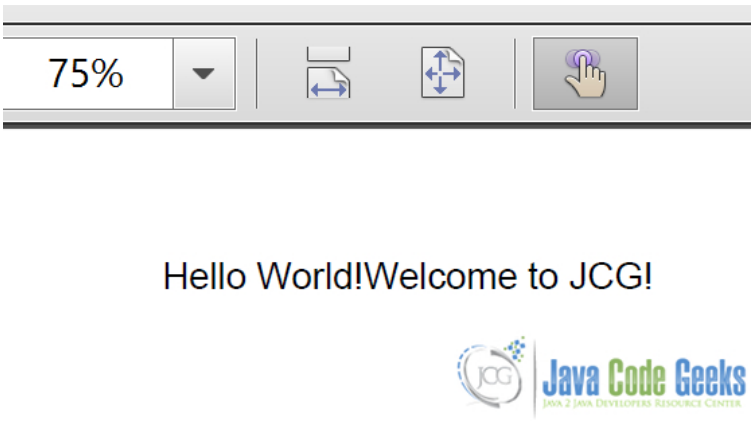

Figure 5.1: Original Document

We will modify this PDF to include the `phrase` "Hello JCGians!!". Let's write a program for this:

PDFStamperExample.java

```
package com.jcg.examples;


import java.io.FileOutputStream;
import java.io.IOException;

import com.itextpdf.text.DocumentException;
import com.itextpdf.text.Element;
import com.itextpdf.text.Phrase;
import com.itextpdf.text.pdf.ColumnText;
import com.itextpdf.text.pdf.PdfContentByte;
import com.itextpdf.text.pdf.PdfReader;
import com.itextpdf.text.pdf.PdfStamper;


class PDFStamperExample
{
                public static void main(String[] args)
                {
                                try
                                {
                                                PdfReader pdfReader = new PdfReader(" ↩
                                                    HelloWorld.pdf");
                                                PdfStamper pdfStamper = new PdfStamper( ↩
                                                    pdfReader, new FileOutputStream(" ↩
                                                    HelloWorldModified.pdf"));
                                                PdfContentByte canvas = pdfStamper. ↩
                                                    getOverContent(1);
                                                ColumnText.showTextAligned(canvas, Element. ↩
                                                    ALIGN_LEFT, new Phrase("Hello people!"), ↩
                                                     250, 750, 0);
                                                pdfStamper.close();
                                                pdfReader.close();
                                }
                                catch (IOException e)
                                {
                                                e.printStackTrace();
                                }
                                catch (DocumentException e)
                                {
```

```
                                        e.printStackTrace();
                                    }
                                }
}
```

We create an object of `com.itextpdf.text.pdf.PdfReader` and pass the path of the PDF we wish to modify. A `FileOutputStream` object is also created, which has the path to the new modified file to be created by the progrmamme. Next, we create an instance of `com.itextpdf.text.pdf.PdfStamper` class by passing the `pdfReader` and a `FileOutputStream` objects created earlier. Next, we need to access the `com.itextpdf.text.pdf.PdfContentByte` object to add the PDF objects like `Phrase` , `Paragraph` etc. The `PdfStamper#getOverContent()` method returns the reference to the underlying `PdfContentByte` objects. Adding the objects to the document via this object adds them over the layer above this document. We will discuss about this in detail later. In case, we need to add to more than one page we will have to iterate over the pages using the `PdfReader#getNumberOfPages()` . Then we close the `PDFStamper` and `PdfReader` . Not closing the objects leads to the generated PDF being corrupted.

Here's the snapshot of the output PDF Document:

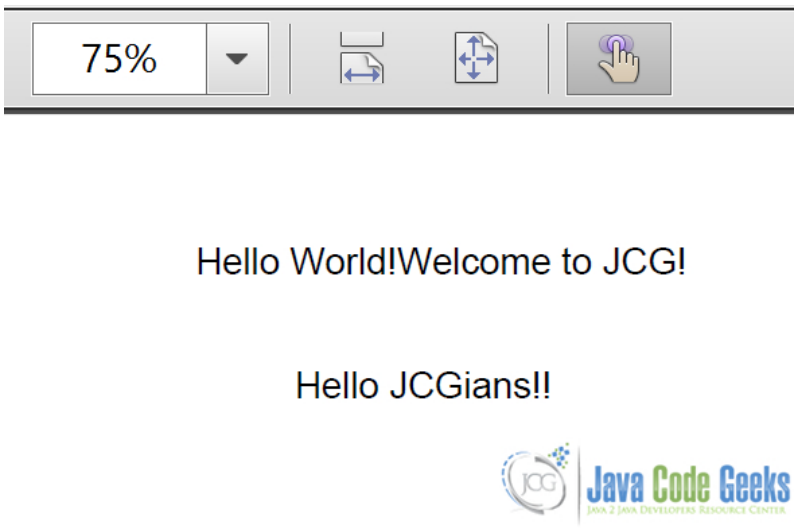

Figure 5.2: Modified Document

One difference that the user needs to understand is about `PdfStamper#getOverContent()` vs `PdfStamper#getUnderContent()` methods. The getOverContent() method returns the access to the `ContentByte` over the existing document and getUnderContent() method returns access to the canvas under the existing document.

Similarly, we can add any type of PDF Objects to the existing document via the `PdfStamper` class like image, phrase, paragraph etc.

5.2 Download the Source Code

Here we demonstrated how we can use the `PDFStamper` class in `IText` Library to modify the contents of an existing PDF Document.

Download

You can download the source code of this example here: **PdfStamperExample.zip**

Chapter 6

iText PDFtable example

In the previous example, we demonstrated the use of PDFStamper class in the IText library. In this example, we will demonstrate how we can use the `https://api.itextpdf.com/itext/com/itextpdf/text/pdf/PdfPTable.html[PDFTable]` to improve the design of the PDF Document and to customize the Document layout with a tabular structure.

Let's start by setting up the project. We create a simple Maven project in the Eclipse. Replace the content of `pom.xml` with the below given contents:

6.1 Project Set-Up

pom.xml

```
<project xmlns="https://maven.apache.org/POM/4.0.0" xmlns:xsi="https://www.w3.org/2001/ ↩
    XMLSchema-instance" xsi:schemaLocation="https://maven.apache.org/POM/4.0.0 https://maven ↩
    .apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>PDFTableExample</groupId>
  <artifactId>PDFTableExample</artifactId>
  <version>0.0.1-SNAPSHOT</version>

        <dependencies>
                <dependency>
                        <groupId>com.itextpdf</groupId>
                        <artifactId>itextpdf</artifactId>
                        <version>5.5.6</version>
                </dependency>
                <dependency>
                        <groupId>org.bouncycastle</groupId>
                        <artifactId>bcprov-jdk15on</artifactId>
                        <version>1.52</version>
                </dependency>


        </dependencies>

</project>
```

This will load the required `Itext` libraries and its corresponding dependencies into the project. Now that the project is setup, lets start with the demonstration and actual code writing.

Here's the class that will create a new `PdfTable` in the PDF Document.

CreatePDFTable.java

```
package com.jcg.examples;

import java.io.FileNotFoundException;
import java.io.FileOutputStream;

import com.itextpdf.text.BaseColor;
import com.itextpdf.text.Document;
import com.itextpdf.text.DocumentException;
import com.itextpdf.text.Phrase;
import com.itextpdf.text.pdf.PdfPCell;
import com.itextpdf.text.pdf.PdfPTable;
import com.itextpdf.text.pdf.PdfWriter;

public class CreatePDFTable
{
		public static void main(String[] args)
		{
			Document document = new Document();
			try
			{
				@SuppressWarnings("unused")
				PdfWriter pdfWriter = PdfWriter.getInstance ↩
				    (document, new FileOutputStream(" ↩
				    CreateTable.pdf"));
				document.open();

				PdfPTable pdfPTable =new PdfPTable(2);
				PdfPCell pdfCell1 = new PdfPCell(new Phrase ↩
				    ("Cell-1"));
				PdfPCell pdfCell2 = new PdfPCell(new Phrase ↩
				    ("Cell-12"));
				pdfPTable.addCell(pdfCell1);
				pdfPTable.addCell(pdfCell2);
				PdfPCell pdfCell3 = new PdfPCell(new Phrase ↩
				    ("Cell-21"));
				pdfCell3.setColspan(2);
				pdfCell3.setBackgroundColor(BaseColor. ↩
				    DARK_GRAY);
				pdfCell3.setBorderColor(BaseColor.RED);
				pdfCell3.setRotation(90);
				pdfPTable.addCell(pdfCell3);

				pdfPTable.setWidthPercentage(70);

				document.add(pdfPTable);
				document.close();
			}
			catch (FileNotFoundException e)
			{
				e.printStackTrace();
			}
			catch (DocumentException e)
			{
				e.printStackTrace();
			}
			document.open();
		}
}
```

We start by creating an instance of `com.itextpdf.text.Document` . We pass this instance to the `PDFWriter` along-with the path and name of the Document to be created. Next, we create an instance of `com.itextpdf.text.pdf.PdfPTable`

. This table may have multiple cells, with each cell having its own properties, which can be managed at an elemental level.

We have used `PdfPTable#setWidthPercentage` to set the table width relative to the document. Cells are created by creating instances of `com.itextpdf.text.pdf.PdfPCell` . We can set the cell border color using the `PdfPCell#setBorderColor` . Similarly we can set the background color using the `PdfPCell#setBackgroundColor` .

The width of the cells can be adjusted using the `setWidths` method. We can increase the size of a `colspan` using the `setColspan` method.

The text in the cell can also be rotated if needed by using `setRotation` method. Similarly, we can set the cell-padding, cell-aligment, cell-indentation etc.

Once the cell instance is ready, it can be attached to the parent table. This table is then added to the document. When the document is closed the table is flushed to PDF.

We can create a nested table by creating a cell in a table and adding a new table in the cell.

Here's how the `PDFPTable` looks in a document when the above class is run:

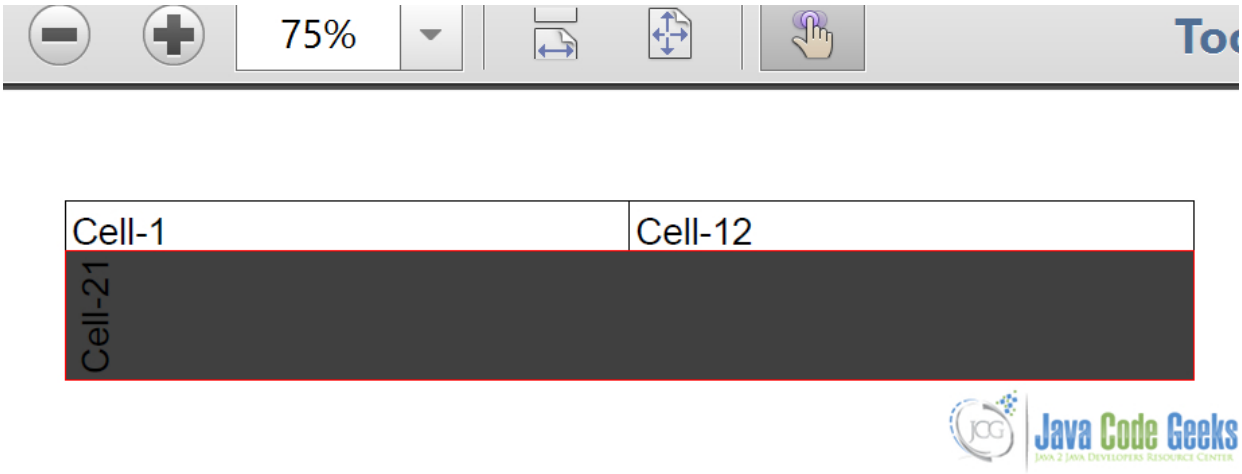

Figure 6.1: PDFTable in a Document

6.2 Download the Source Code

In this example we demonstrated how we can use the Itext PDFPTable class to improve the design and precision of the PDF document.

Download

You can download the source code of this example here: **PDFTableExample.zip**

Chapter 7

iText HTML to PDF Example

In the previous examples, we have studied about various Itext Classes like PDFTable, PDFStamper,PDFRectangle etc. that help us in creation of the PDF document. In this example, we will demonstrate when we already have a document in HTML format and need to convert it to a PDF Document.

7.1 Project Set-Up

We shall use Maven to setup our project. Open eclipse and create a simple Maven project and check the skip archetype selection checkbox on the dialogue box that appears. Replace the content of the existing `pom.xml` with the `pom.xml` below:

pom.xml

```
<project xmlns="https://maven.apache.org/POM/4.0.0" xmlns:xsi="https://www.w3.org/2001/ ↩
    XMLSchema-instance" xsi:schemaLocation="https://maven.apache.org/POM/4.0.0 https://maven ↩
    .apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>ItextHtmlToPDFExample</groupId>
  <artifactId>ItextHtmlToPDFExample</artifactId>
  <version>0.0.1-SNAPSHOT</version>

  <dependencies>
                <dependency>
                        <groupId>com.itextpdf</groupId>
                        <artifactId>itextpdf</artifactId>
                        <version>5.5.6</version>
                </dependency>
                <dependency>
                        <groupId>org.bouncycastle</groupId>
                        <artifactId>bcprov-jdk15on</artifactId>
                        <version>1.52</version>
                </dependency>

                <dependency>
                        <groupId>com.itextpdf.tool</groupId>
                        <artifactId>xmlworker</artifactId>
                        <version>5.5.7</version>
                </dependency>

        </dependencies>
</project>
```

In this example, we have added one more `dependency` for the `Xmlworker` JAR. That's all from setting-up project point of view, let's start with the actual code writing now:

7.2 Implementation

We will convert the below HTML document into a PDF document:

```
<html><body align='center'> This is my Project
<table border='2' align='center'>
<tr>
        <td>
            JavaCodeGeeks
        </td>
        <td>
            <a href='examples.javacodegeeks.com'>JavaCodeGeeks</a>
        </td>
    </tr>
    <tr>
        <td>
            Google Here
        </td>
        <td>
            <a href='www.google.com'>Google</a>
        </td>
    </tr>
</table>
```

Figure 7.1: Itext Html Pdf Table

Here's the how the document looks like in a browser(CHROME here):

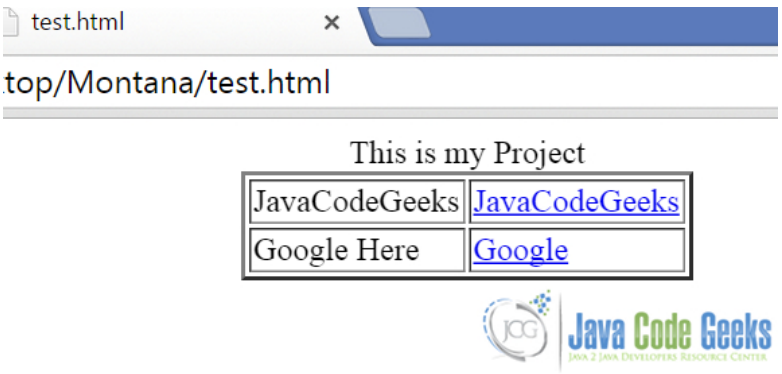

Figure 7.2: HTML Document

The `com.itextpdf.tool.xml.XMLWorkerHelper` converts the `XHTML` code to PDF. The `https://en.wikipedia.org/wiki/XHTML[Xhtml]` is a stricter version of HTML which ensures the document are well-formed and hence can be parsed efficiently by the standard XML parsers. Not closing the tags or any other syntax errors can lead to exception like :

```
com.itextpdf.tool.xml.exceptions.RuntimeWorkerException: Invalid nested tag html found,  ↩
    expected closing tag body.
```

Now that we are clear with the basics let's write the code for the actual conversion:

ItextHtmlToPDFExample.java

```
package com.jcg.examples;


import java.io.ByteArrayInputStream;
```

```
import java.io.File;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.io.OutputStream;

import com.itextpdf.text.Document;
import com.itextpdf.text.pdf.PdfWriter;
import com.itextpdf.tool.xml.XMLWorkerHelper;


public class ItextHtmlToPDFExample
{
        public static void main(String[] args)
        {
                try
                {
               OutputStream file = new FileOutputStream(new File("HTMLtoPDF.pdf"));
                        Document document = new Document();
                        PdfWriter writer = PdfWriter.getInstance(document, file);
                        StringBuilder htmlString = new StringBuilder();
                        htmlString.append(new String("<html><body> This is HMTL to PDF  ↩
                            conversion Example|======================== "));
                        htmlString.append(new String("|JavaCodeGeeks|<a href='examples. ↩
                            javacodegeeks.com'>JavaCodeGeeks</a> "));
                        htmlString.append(new String("| Google Here |<a href='www.google. ↩
                            com'>Google</a> |========================</body></html>"));

                        document.open();
                        InputStream is = new ByteArrayInputStream(htmlString.toString(). ↩
                            getBytes());
                        XMLWorkerHelper.getInstance().parseXHtml(writer, document, is);
                        document.close();
                        file.close();
                }
                catch (Exception e)
                {
                        e.printStackTrace();
                }

        }

}
```

We create an instance of the `Document` and `FileOutputStream` and pass it the `PDFWriter`. Now we create a `StringBuilder` object which holds the HTML source code. The `XMLWorker` class accepts the Byte Array of the HTML source code. `XMLWorkerHelper.getInstance().parseXHtml()` method parses the HTML source code and writes to the document created earlier via the `PDFWriter` instance.

Here's how the converted PDF document looks like:

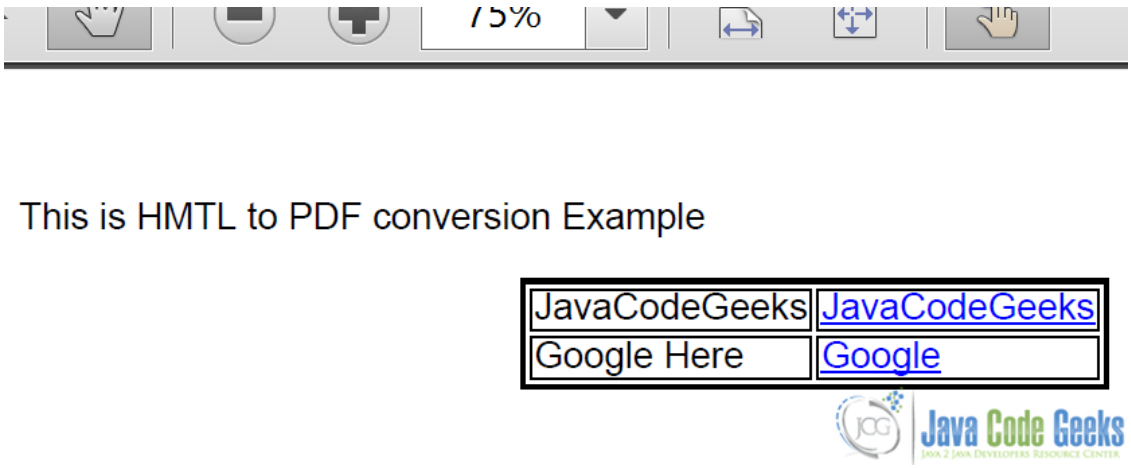

Figure 7.3: Html to PDF Document

7.3 Download the Source Code

Here, we demonstrated how we can convert a HTML Document to PDF format using the Itext library.

Download

You can download the source code of this example here: **ItextHtmlToPDFExample.zip**

Chapter 8

iText Watermark Example

In the previous example we learnt how we can convert a HTML document to a PDF Document using the `ITEXT` library. In this example we will demonstrate how we can add watermark to a PDF Document using `Itext` .

Watermark are usually added to a document to prevent counterfeiting or to mark the name of the maker or the organization, for advertising the name of the organization in the document. Whatever, be the reason, let's find out how we can achieve that in a PDF Document.

8.1 Project Set-Up

We shall use Maven to setup our project. Open eclipse and create a simple Maven project and check the skip archetype selection checkbox on the dialogue box that appears. Replace the content of the existing `pom.xml`

pom.xml

```
<project xmlns="https://maven.apache.org/POM/4.0.0" xmlns:xsi="https://www.w3.org/2001/ ↩
    XMLSchema-instance"
        xsi:schemaLocation="https://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd ↩
            /maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <groupId>ITextExample</groupId>
        <artifactId>ITextExample</artifactId>
        <version>0.0.1-SNAPSHOT</version>

        <dependencies>
                <dependency>
                        <groupId>com.itextpdf</groupId>
                        <artifactId>itextpdf</artifactId>
                        <version>5.5.6</version>
                </dependency>
                <dependency>
                        <groupId>org.bouncycastle</groupId>
                        <artifactId>bcprov-jdk15on</artifactId>
                        <version>1.52</version>
                </dependency>


        </dependencies>

</project>
```

That's all from setting-up project point of view, let's start with the actual code implementation now:

8.2 Implementation

`com.itextpdf.text.pdf.PdfPageEventHelper` class is used to listen for the page end event of the document via the `onEndPage` method. Whenever the page gets filled up with the contents it can accomodate, the `onEndPage` method is invoked and the watermark is added to the page of the document. This helps in all the pages of the document having the watermark.

We create a simple Water-Mark for **JavaCodeGeeks** with the letters JCG. The font color is Grey so it does not obscure the actual content of the document and is visible only as a background.

PDFEventListener.java

```
package com.jcg.examples;


import com.itextpdf.text.BaseColor;
import com.itextpdf.text.Document;
import com.itextpdf.text.Element;
import com.itextpdf.text.Font;
import com.itextpdf.text.Font.FontFamily;
import com.itextpdf.text.Phrase;
import com.itextpdf.text.pdf.ColumnText;
import com.itextpdf.text.pdf.PdfContentByte;
import com.itextpdf.text.pdf.PdfPageEventHelper;
import com.itextpdf.text.pdf.PdfWriter;


public class PDFEventListener extends PdfPageEventHelper
{

                @Override
                public void onEndPage(PdfWriter writer, Document document)
                {
                                PdfContentByte canvas = writer.getDirectContentUnder();
                                Phrase watermark = new Phrase("JCG", new Font(FontFamily. ↩
                                    TIMES_ROMAN, 190, Font.NORMAL, BaseColor.LIGHT_GRAY));
                                ColumnText.showTextAligned(canvas, Element.ALIGN_CENTER,  ↩
                                    watermark, 337, 500, 45);
                }
}
```

Now we will use an instance of this `PDFEventListener` class to the `com.itextpdf.text.pdf.PdfWriter` . To do so we need to register the instance with the `Pdfwriter` instance by `pdfWriter.setPageEvent` method.

CreateWatermarkedPDF.java

```
package com.jcg.examples;

import java.io.FileNotFoundException;
import java.io.FileOutputStream;

import com.itextpdf.text.BaseColor;
import com.itextpdf.text.Document;
import com.itextpdf.text.DocumentException;
import com.itextpdf.text.Font;
import com.itextpdf.text.Phrase;
import com.itextpdf.text.Font.FontFamily;
import com.itextpdf.text.pdf.PdfWriter;


public class CreateWatermarkedPDF
{
        public static void main(String[] args)
```

```
        {
                try
                {
                        Document document = new Document();
                        PdfWriter pdfWriter = PdfWriter.getInstance(document, new  ↩
                            FileOutputStream("WaterMarkedDocument.pdf"));
                        document.open();
                        pdfWriter.setPageEvent(new PDFEventListener());
                        Font font = new Font(FontFamily.TIMES_ROMAN, 20, Font.NORMAL,  ↩
                            BaseColor.BLACK);
                        document.add(new Phrase("Hi People!! This is an exaple to  ↩
                            demostrate Watermark in using Itext",font));
                        document.close();
                }
                catch (FileNotFoundException e)
                {
                        e.printStackTrace();
                }
                catch (DocumentException e)
                {
                        e.printStackTrace();
                }
        }
}
```

Upon execution of `CreateWatermarkedPDF` class a PDF document is created with the Water-Marked letters JCG in the background.

Here's how the Water-Marked document looks like:

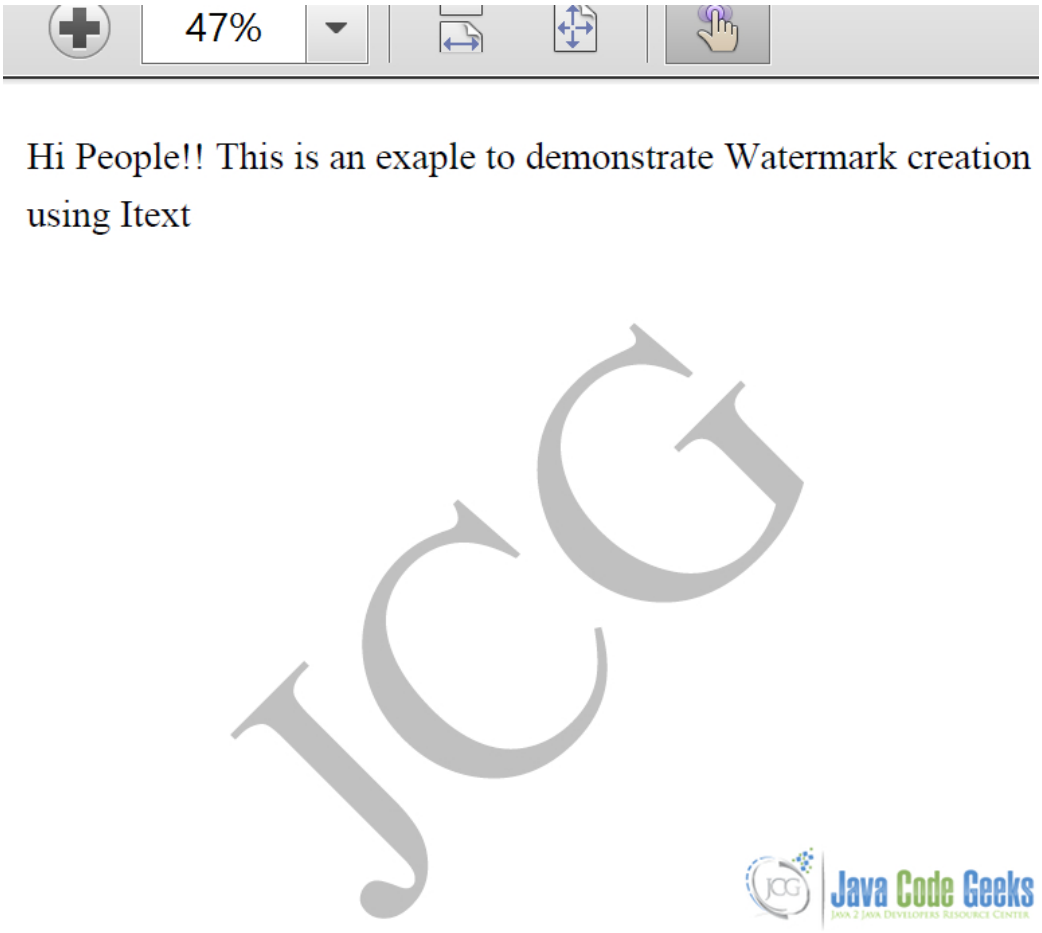

Figure 8.1: WaterMarked PDF Document

8.3 Download the Source Code

Here, we demonstrated how we can add water-mark while creating new document or to an existing document using the `Itext` library.

Download

You can download the source code of this example here: **ItextWatermarkExample.zip**

Chapter 9

iText Barcode Example

In the previous example we studied how to generate watermark in the PDF documents using Itext. In this example, we will learn how we can add Barcode to the PDF using Itext.

Barcodes are now-a-days omnipresent due to their ease of use and reliability. Barcodes are cost effective, take less time to read and versatile. We will have a look at how we can encode data into a Barcode in a PDF Document.

9.1 1. Project Setup

We shall use Maven to setup our project. Open eclipse and create a simple Maven project and check the skip archetype selection checkbox on the dialogue box that appears. Replace the content of the existing `pom.xml`

pom.xml

```
<project xmlns="https://maven.apache.org/POM/4.0.0" xmlns:xsi="https://www.w3.org/2001/ ↩
    XMLSchema-instance"
        xsi:schemaLocation="https://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd ↩
            /maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <groupId>ITextBarcodeExample</groupId>
        <artifactId>ITextBarcodeExample</artifactId>
        <version>0.0.1-SNAPSHOT</version>

        <dependencies>
                <dependency>
                        <groupId>com.itextpdf</groupId>
                        <artifactId>itextpdf</artifactId>
                        <version>5.5.6</version>
                </dependency>
                <dependency>
                        <groupId>org.bouncycastle</groupId>
                        <artifactId>bcprov-jdk15on</artifactId>
                        <version>1.52</version>
                </dependency>


        </dependencies>

</project>
```

That's all from setting-up project point of view, let's start with the actual code implementation now:

9.2 Implementation

CreateBarcode.java

```
package com.jcg.examples;


import java.io.FileNotFoundException;
import java.io.FileOutputStream;

import com.itextpdf.text.Document;
import com.itextpdf.text.DocumentException;
import com.itextpdf.text.Image;
import com.itextpdf.text.pdf.Barcode128;
import com.itextpdf.text.pdf.BarcodeEAN;
import com.itextpdf.text.pdf.BarcodeQRCode;
import com.itextpdf.text.pdf.PdfContentByte;
import com.itextpdf.text.pdf.PdfWriter;


public class CreateWatermarkedPDF
{
                public static void main(String[] args)
                {
                                try
                                {
                                                Document document = new Document();
                                                PdfWriter pdfWriter = PdfWriter.getInstance ↩
                                                    (document, new FileOutputStream(" ↩
                                                    HelloWorld.pdf"));

                                                document.open();
                                                PdfContentByte pdfContentByte = pdfWriter. ↩
                                                    getDirectContent();

                                                Barcode128 barcode128 = new Barcode128();
                                                barcode128.setCode("examples.javacodegeeks. ↩
                                                    com/author/chandan-singh");
                                                barcode128.setCodeType(Barcode128.CODE128);
                                                Image code128Image = barcode128. ↩
                                                    createImageWithBarcode(pdfContentByte,  ↩
                                                    null, null);
                                                code128Image.setAbsolutePosition(10, 700);
                                                code128Image.scalePercent(100);
                                                document.add(code128Image);

                                                BarcodeEAN barcodeEAN = new BarcodeEAN();
                                                barcodeEAN.setCodeType(BarcodeEAN.EAN13);
                                                barcodeEAN.setCode("1234523453323");
                                                Image codeEANImage = barcodeEAN. ↩
                                                    createImageWithBarcode(pdfContentByte,  ↩
                                                    null, null);
                                                codeEANImage.setAbsolutePosition(20, 600);
                                                codeEANImage.scalePercent(100);
                                                document.add(codeEANImage);

                                                BarcodeQRCode barcodeQrcode = new  ↩
                                                    BarcodeQRCode("examples.javacodegeeks. ↩
                                                    com/author/chandan-singh", 1, 1, null);
                                                Image qrcodeImage = barcodeQrcode.getImage ↩
                                                    ();
                                                qrcodeImage.setAbsolutePosition(20, 500);
```

```
                                        qrcodeImage.scalePercent(100);
                                        document.add(qrcodeImage);

                                        document.close();
                                }
                                catch (FileNotFoundException e)
                                {
                                        e.printStackTrace();
                                }
                                catch (DocumentException e)
                                {
                                        e.printStackTrace();
                                }
                        }
}
```

We start off by creating instances of `com.itextpdf.text.Document` and `com.itextpdf.text.pdf.PdfWriter` classes. Then we create different types of barcode and pass the data we wish to be encoded in the Barcode.

We start off with the type `Barcode128` . We create an instance of `com.itextpdf.text.pdf.Barcode128` and set the appropriate Code type and then create an image from the barcode which is embedded in to the document. Barcode 128 is typically used only for numeric or alpha-numeric data.

Next we create Barcode of the type `EAN-13` . EAN-13(European/International Article Number) Barcode format is usually compact and hence is used extensively on products with limited surface area. We use `com.itextpdf.text.pdf.BarcodeEAN` class to generate EAN/IAN Barcodes.

Next is the QR(Quick Response) Barcode. QR code can encapsulate large amounts of data compared to other UPC codes and is very fast. We use `com.itextpdf.text.pdf.BarcodeQRCode` class to generate QR Barcode

Here's how the different Barcode formats look like in a PDF document : .Barcode Types in a Document Barcode Types in a Document

Itext supports different variants of types of Barcodes that are described above like the EAN-8, CODE 128 RAW, CODE 128 UCC etc

9.3 Download the Source Code

Here we studied how to generate and embed `Barcode` into the PDF Document using `Itext` library.

Download

You can download the source code of this example here: **ItextBarcodeExample.zip**

Chapter 10

iText Merge PDF Example

In the previous example we saw how we can create and add Barcode to the PDF document. In this example we will demonstrate how we can merge multiple PDF Documents into one.

We often face a situation where we need to merge some of the PDF Documents in our applications. Itext provides us with a way to merge different PDF documents into a single PDF document. Let's see how we can achieve this :

10.1 Project Setup

We shall use Maven to setup our project. Open eclipse and create a simple Maven project and check the skip archetype selection checkbox on the dialogue box that appears. Replace the content of the existing `pom.xml`

pom.xml

```
<project xmlns="https://maven.apache.org/POM/4.0.0" xmlns:xsi="https://www.w3.org/2001/ ↩
    XMLSchema-instance"
        xsi:schemaLocation="https://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd ↩
            /maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <groupId>ItextPDFMergeExample</groupId>
        <artifactId>ItextPDFMergeExample</artifactId>
        <version>0.0.1-SNAPSHOT</version>

        <dependencies>
                <dependency>
                        <groupId>com.itextpdf</groupId>
                        <artifactId>itextpdf</artifactId>
                        <version>5.5.6</version>
                </dependency>
                <dependency>
                        <groupId>org.bouncycastle</groupId>
                        <artifactId>bcprov-jdk15on</artifactId>
                        <version>1.52</version>
                </dependency>


        </dependencies>

</project>
```

That's all from setting-up project point of view, let's start with the actual code implementation for this demonstration now:

10.2 Implementation

MergePDFExample.java

```
package com.examples.jcg;


import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.ArrayList;
import java.util.List;

import com.itextpdf.text.Document;
import com.itextpdf.text.DocumentException;
import com.itextpdf.text.pdf.PdfContentByte;
import com.itextpdf.text.pdf.PdfImportedPage;
import com.itextpdf.text.pdf.PdfReader;
import com.itextpdf.text.pdf.PdfWriter;


public class MergePDFExample
{
		public static void main(String[] args)
		{
			try
			{
				List<InputStream> list = new ArrayList<InputStream>();

				InputStream inputStreamOne = new FileInputStream(new File(" ↩
				    HelloWorld.pdf"));
				list.add(inputStreamOne);
				InputStream inputStreamTwo = new FileInputStream(new File(" ↩
				    HelloWorld1.pdf"));
				list.add(inputStreamTwo);

				OutputStream outputStream = new FileOutputStream(new File(" ↩
				    Merger.pdf"));
				mergePdf(list, outputStream);
			}
			catch (FileNotFoundException e)
			{
				e.printStackTrace();
			}
			catch (DocumentException e)
			{
				e.printStackTrace();
			}
			catch (IOException e)
			{
				e.printStackTrace();
			}
		}
		private static void mergePdf(List<InputStream> list, OutputStream ↩
		    outputStream) throws DocumentException, IOException
		{
			Document document = new Document();
```

```
                        PdfWriter pdfWriter = PdfWriter.getInstance(document, outputStream) ↩
                            ;
                        document.open();
                        PdfContentByte pdfContentByte = pdfWriter.getDirectContent();

                        for (InputStream inputStream : list)
                        {
                                PdfReader pdfReader = new PdfReader(inputStream);
                                for (int i = 1; i <= pdfReader.getNumberOfPages(); i++)
                                {
                                        document.newPage();
                                        PdfImportedPage page = pdfWriter.getImportedPage( ↩
                                            pdfReader, i);
                                        pdfContentByte.addTemplate(page, 0, 0);
                                }
                        }

                        outputStream.flush();
                        document.close();
                        outputStream.close();
                }
}
```

We start off by creating `FileInputStream` objects for each of the PDF Files we have to merge and add these `InputStream` Objects to the `List<InputStream>` instance. Next, we create an `OutputStream` instance, which will create the file we wish to be the final output document.

The `mergePdf` method creates an instance of PdfWriter using the `OutputStream` object we created earlier. Next we iterate over the `List` of `InputStream` objects we created and create a `com.itextpdf.text.pdf.PdfReader` object from each `FileInputStream` instances we extract from the list.

Next, we iterate over all the pages in each of the document using the `PdfReader.getNumberOfPages()` method. We import the page from the document using the `PdfWriter.getImportedPage()` and then add the page to the new `Document` by using the `PdfContentByte.addTemplate` method.

We repeat this procedure for each page in each of the PDF Document we have to merge to get all the PDF Files in a single Document.

10.3 Download the source code

Here we studied how we can merge existing PDF Documents into a single PDF document using Itext.

Download

You can download the source code of this example here: **ItextPDFMergeExample.zip**