iTextPDF

tutorialspoint

SIMPLY EASY LEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia https://twitter.com/tutorialspoint

About the Tutorial

Apache iText is an open-source Java library that supports the development and conversion of PDF documents. In this tutorial, we will learn how to use iText to develop Java programs that can create, convert, and manipulate PDF documents.

Audience

This tutorial has been prepared for beginners to make them understand the basics of iText library. It will help the readers in building applications that involve creation, manipulation, and deletion of PDF documents.

Prerequisites

For this tutorial, it is assumed that the readers have a prior knowledge of Java programming language.

Copyright & Disclaimer

Table of Contents

About the Tutorial i
Audience i
Prerequisites i
Copyright & Disclaimer i
Table of Contents ii

ITEXT – INTRODUCTION 1

1. iText – Overview 2

2. iText – Creating a PDF Document 9
Creating an Empty PDF Document 9
Example 10

3. iText – Adding an AreaBreak 13
Creating an AreaBreak 13
Example 14

4. iText – Adding a Paragraph 17
Creating a Paragraph 17
Example 18

5. iText — Adding a List 21
Creating a List 21
Example 22

ITEXT – TABLES 25

6. iText – Adding a Table 26
Adding a Table to a Pdf 26
Example 28

7. iText – Formatting Cell Contents 30
Formatting the Cells in a Table 30
Example 32

8. iText – Formatting the Borders of a Cell 35
Formatting the Borders of a Cell 35
Example 37

9. iText – Adding Image to a Table 41
Adding an Image to a Table 41
Example 43

10. iText – Nested Table 47
Adding Nested Tables in a Pdf 47
Example 49

11. iText – Adding Lists to a Table 53
Adding Lists to a Table in a PDF 53
Example 55

ITEXT – IMAGES 58

12. iText – Adding Image to a PDF 59
Adding Image to a Pdf 59
Example 60

13. iText – Setting Position of the Image 63
Setting the Position of the Image 63
Example 65

14. iText – Scaling an Image 67
Scaling an Image in a PDF 67
Example 69

15. iText – Rotating an Image 71
Rotating an Image in a PDF 71
Example 73

ITEXT – ANNOTATIONS 75

16. iText – Text Annotation 76
Creating a Text Annotation in a PDF 76
Example 78

17. iText – Link Annotation 80
Creating a Link Annotation in a PDF 80
Example 82

18. iText – Line Annotation 85
Creating a Line Annotation in a Pdf 85
Example 87

19. iText – Markup Annotation 90
Creating a Markup Annotation in a PDF 90
Step 7: Adding the annotation to a page 91
Example 92

20. iText – Circle Annotation 94
Creating a Circle Annotation in a PDF 94
Example 96

ITEXT – CANVAS 98

21. iText – Drawing an Arc 99
Drawing an Arc on a PDF 99
Example 100

22. iText – Drawing a Line 103
Drawing a Line on a PDF 103
Example 104

23. iText – Drawing a Circle 107
Drawing a Circle on a Pdf 107
Example 108

ITEXT – MISCELLANEOUS 111

24. iText – Setting Font 112
Setting Font of the Text in a PDF 112
Example 114

25. iText – Shrinking the Content 117
Shrinking the Content in a PDF 117
Example 119

26. iText – Tiling PDF Pages 121

27. iText – N-up 124

iText – Introduction

1. iText – Overview

The Portable Document Format (PDF) is a file format that helps to present data in a manner that is independent of application software, hardware, and operating systems. Each PDF file holds description of a fixed-layout flat document, including text, fonts, graphics, and other information needed to display it.

There are several libraries available to create and manipulate PDF documents through programs, such as:

- **Adobe PDF Library**: This library provides API in languages such as C++, .NET and Java. Using this, we can edit, view, print, and extract text from PDF documents.
- **Formatting Objects Processor**: Open-source print formatter driven by XSL Formatting Objects and an output independent formatter. The primary output target is PDF.
- **PDF Box**: Apache PDFBox is an open-source Java library that supports the development and conversion of PDF documents.
- **Jasper Reports**: This is a Java reporting tool which generates reports in PDF document including Microsoft Excel, RTF, ODT, comma-separated values and XML files.

What is iText?

Similar to the above-listed software, iText is a Java PDF library using which, you can develop Java programs that create, convert, and manipulate PDF documents.

Features of iText

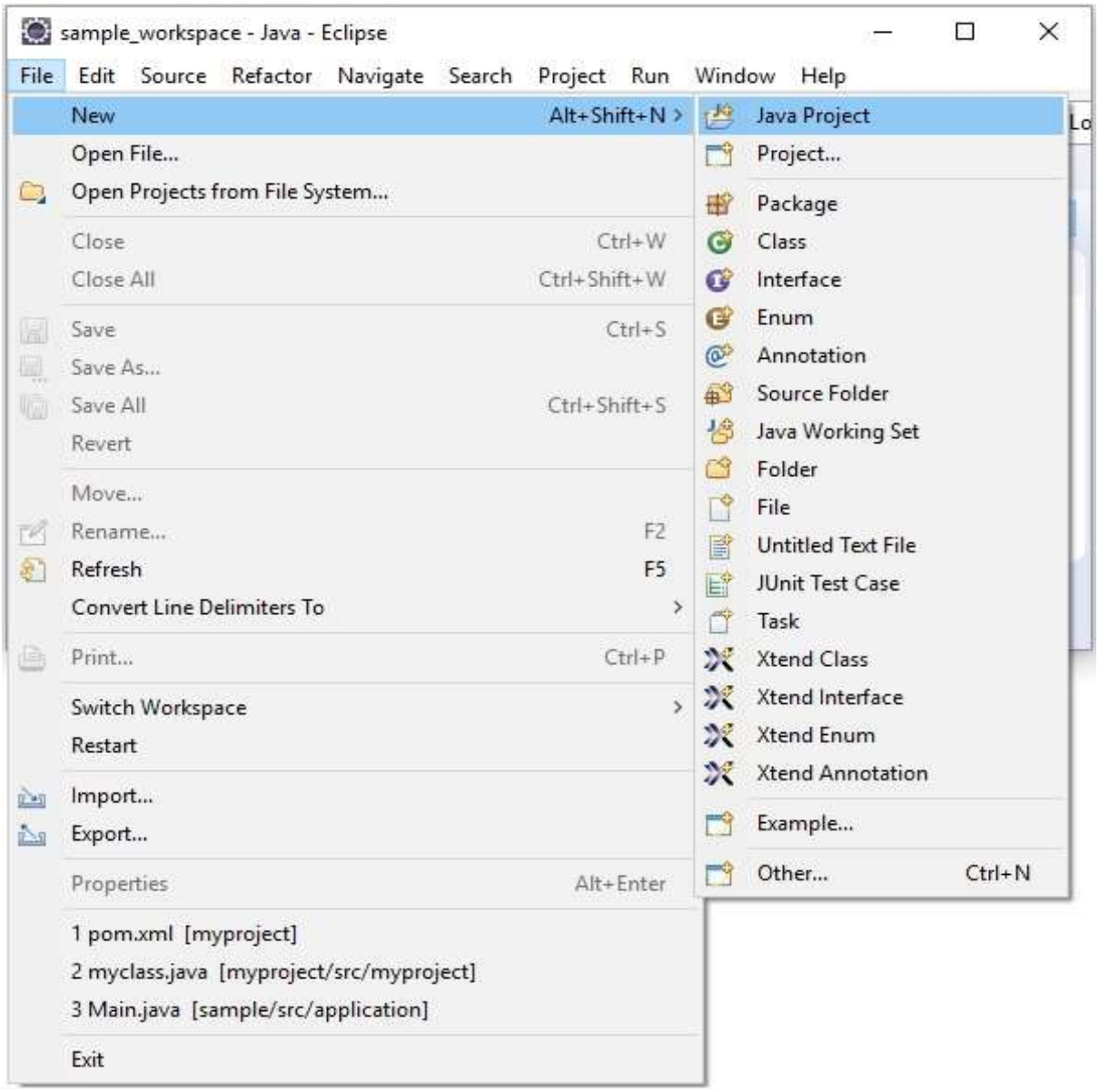
Following are the notable features of iText library:

- **Interactive:** iText provides you classes (API's) to generate interactive PDF documents. Using these, you can create maps and books.
- **Adding bookmarks, page numbers, etc.:** Using iText, you can add bookmarks, page numbers, and watermarks.
- **Split & Merge:** Using iText, you can split an existing PDF into multiple PDFs and also add/concatenate additional pages to it.
- **Fill Forms:** Using iText, you can fill interactive forms in a PDF document.
- **Save as Image:** Using iText, you can save PDFs as image files, such as PNG or JPEG.
- **Canvas:** iText library provides you a Canvas class using which you can draw various geometrical shapes on a PDF document like circle, line, etc.
- **Create PDFs:** Using iText, you can create a new PDF file from your Java programs. You can include images and fonts too.

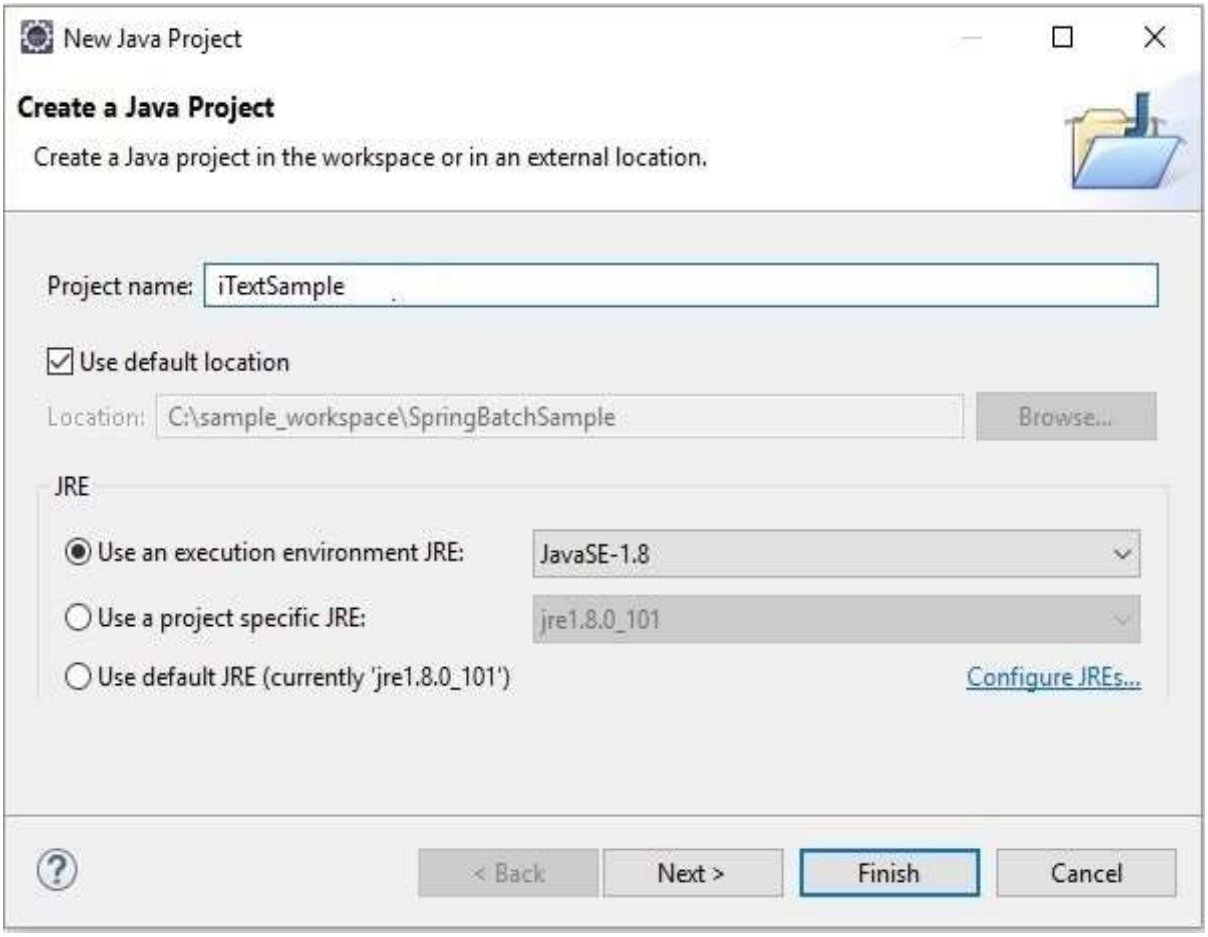
IText Environment

Follow the steps given below to set the iText environment on Eclipse.

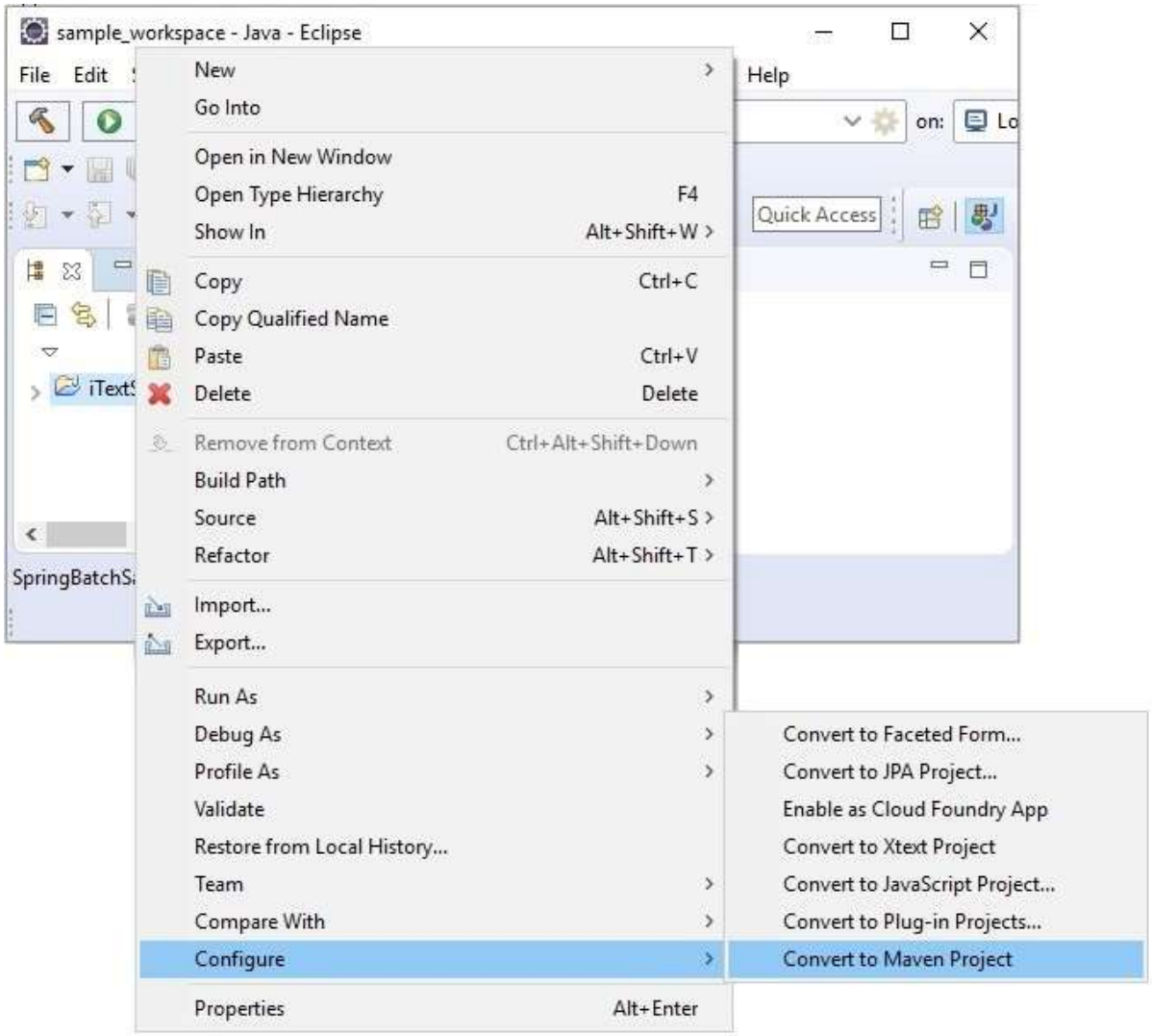
Step 1: Install Eclipse and open a new project in it as shown below.



Step 2: Create an **iTextSample** project as shown below.



Step 3: Right-click on the project and convert it into a Maven project as shown below. Once you convert it into Maven project, it will give you a **pom.xml** where you need to mention the required dependencies. Thereafter, the **jar** files of those dependencies will be automatically downloaded into your project.



Step 4: Now, in the **pom.xml** of the project, copy and paste the following content (dependencies for iText application) and refresh the project.

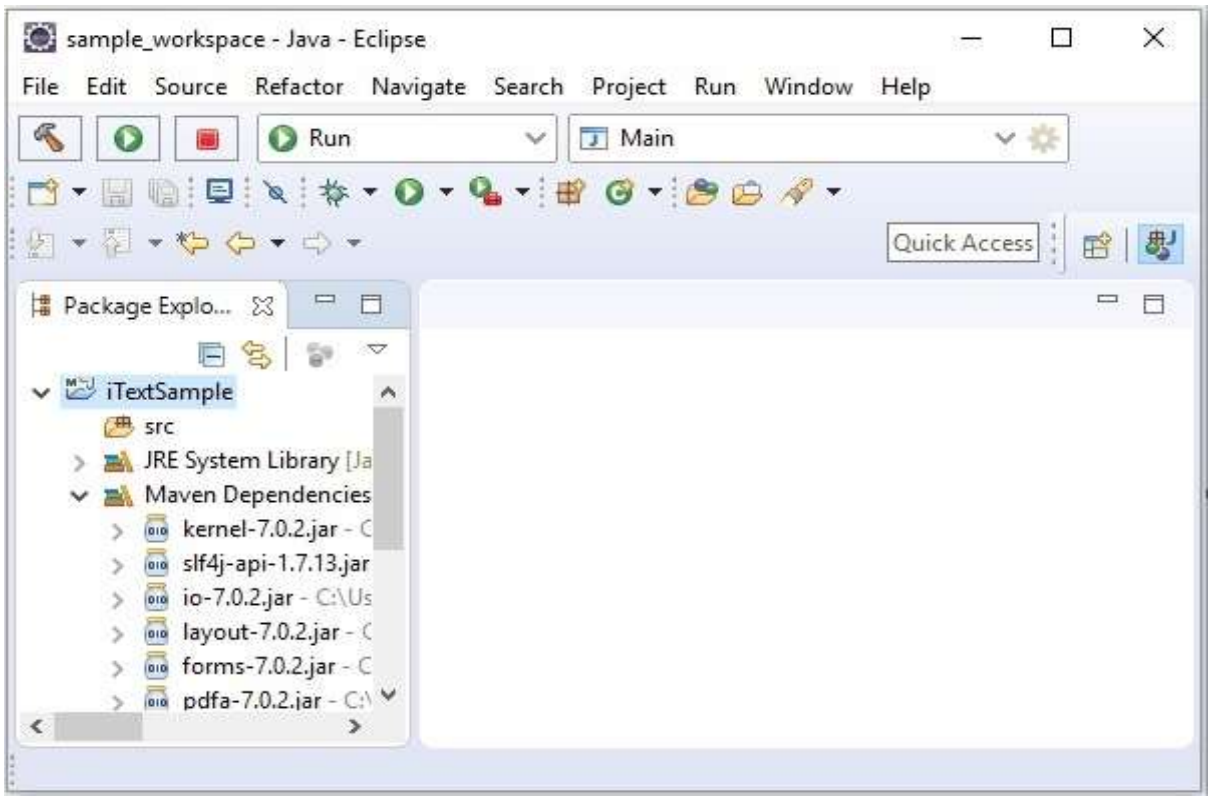
Using pom.xml

Convert the project into Maven project and add the following content to its **pom.xml**.

```
<project xmlns="http:// maven.apache.org/POM/4.0.0" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://
maven.apache.org/POM/4.0.0 http:// maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>SanthoshExample</groupId>
  <artifactId>SanthoshExample</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <build>
    <sourceDirectory>src</sourceDirectory>
    <plugins>
      <plugin>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.5.1</version>
        <configuration>
          <source>1.8</source>
          <target>1.8</target>
        </configuration>
      </plugin>
    </plugins>
  </build>
    <dependencies>
    <!-- always needed -->
    <dependency>
        <groupId>com.itextpdf</groupId>
        <artifactId>kernel</artifactId>
        <version>7.0.2</version>
    </dependency>

    <dependency>
        <groupId>com.itextpdf</groupId>
        <artifactId>io</artifactId>
        <version>7.0.2</version>
    </dependency>

    <dependency>
        <groupId>com.itextpdf</groupId>
        <artifactId>layout</artifactId>
        <version>7.0.2</version>
```

```
    </dependency>

    <dependency>
        <groupId>com.itextpdf</groupId>
        <artifactId>forms</artifactId>
        <version>7.0.2</version>
    </dependency>

    <dependency>
        <groupId>com.itextpdf</groupId>
        <artifactId>pdfa</artifactId>
        <version>7.0.2</version>
    </dependency>

    <dependency>
        <groupId>com.itextpdf</groupId>
        <artifactId>sign</artifactId>
        <version>7.0.2</version>
    </dependency>

    <dependency>
        <groupId>com.itextpdf</groupId>
        <artifactId>barcodes</artifactId>
        <version>7.0.2</version>
    </dependency>

    <dependency>
        <groupId>com.itextpdf</groupId>
        <artifactId>font-asian</artifactId>
        <version>7.0.2</version>
    </dependency>

    <dependency>
        <groupId>com.itextpdf</groupId>
        <artifactId>hyph</artifactId>
        <version>7.0.2</version>
    </dependency>
</dependencies>
</project>
```

Finally, if you observe the Maven dependencies, you can observe that all the required **jar** files were downloaded.



2. iText – Creating a PDF Document

Let us now understand how to create a PDF document using the iText library.

Creating an Empty PDF Document

You can create an empty PDF Document by instantiating the **Document** class. While instantiating this class, you need to pass a **PdfDocument** object as a parameter to its constructor.

Following are the steps to create an empty PDF document.

Step 1: Creating a PdfWriter object

The **PdfWriter** class represents the Doc Writer for a PDF. This class belongs to the package **com.itextpdf.kernel.pdf**. The constructor of this class accepts a string, representing the path of the file where the PDF is to be created.

Instantiate the PdfWriter class by passing a string value (representing the path where you need to create a PDF) to its constructor, as shown below.

```
// Creating a PdfWriter
String dest = "C:/itextExamples/sample.pdf";
PdfWriter writer = new PdfWriter(dest);
```

When an object of this type is passed to a PdfDocument (class), every element added to this document will be written to the file specified.

Step 2: Creating a PdfDocument object

The **PdfDocument** class is the class that represents the PDF Document in iText. This class belongs to the package **com.itextpdf.kernel.pdf**. To instantiate this class (in writing mode), you need to pass an object of the class **PdfWriter** to its constructor.

Instantiate the PdfDocument class by passing the above created PdfWriter object to its constructor, as shown below.

```
// Creating a PdfDocument
PdfDocument pdfDoc = new PdfDocument(writer);
```

Once a PdfDocument object is created, you can add various elements like page, font, file attachment, and event handler using the respective methods provided by its class.

Step 3: Adding an empty page

The **addNewPage()** method of the **PdfDocument** class is used to create an empty page in the PDF document.

Add an empty page to the PDF document created in the previous step as shown below.

```
// Adding an empty page
pdfDoc.addNewPage();
```

Step 4: Creating a Document object

The **Document** class of the package **com.itextpdf.layout** is the root element while creating a self-sufficient PDF. One of the constructors of this class accepts an object of the class PdfDocument.

Instantiate the **Document** class by passing the object of the class **PdfDocument** created in the previous steps as shown below.

```
// Creating a Document
Document document = new Document(pdfDoc);
```

Step 5: Closing the Document

Close the document using the **close()** method of the **Document** class as shown below.

```
// Closing the document
document.close();
```

Example

Following is the Java program which demonstrates the creation of a PDF Document. It creates a PDF document with the name **sample.pdf**, adds an empty page to it, and saves it in the path **C:/itextExamples/**

Save this code in a file with the name **create_PDF.java.**

```
import com.itextpdf.kernel.pdf.PdfDocument;
import com.itextpdf.kernel.pdf.PdfWriter;
import com.itextpdf.layout.Document;

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

      // Creating a PdfWriter
      String dest = "C:/itextExamples/sample.pdf";
      PdfWriter writer = new PdfWriter(dest);

      // Creating a PdfDocument
      PdfDocument pdfDoc = new PdfDocument(writer);

      // Adding a new page
```

```
        pdfDoc.addNewPage();

        // Creating a Document
        Document document = new Document(pdfDoc);

        // Closing the document
        document.close();

        System.out.println("PDF Created");
    }
}
```

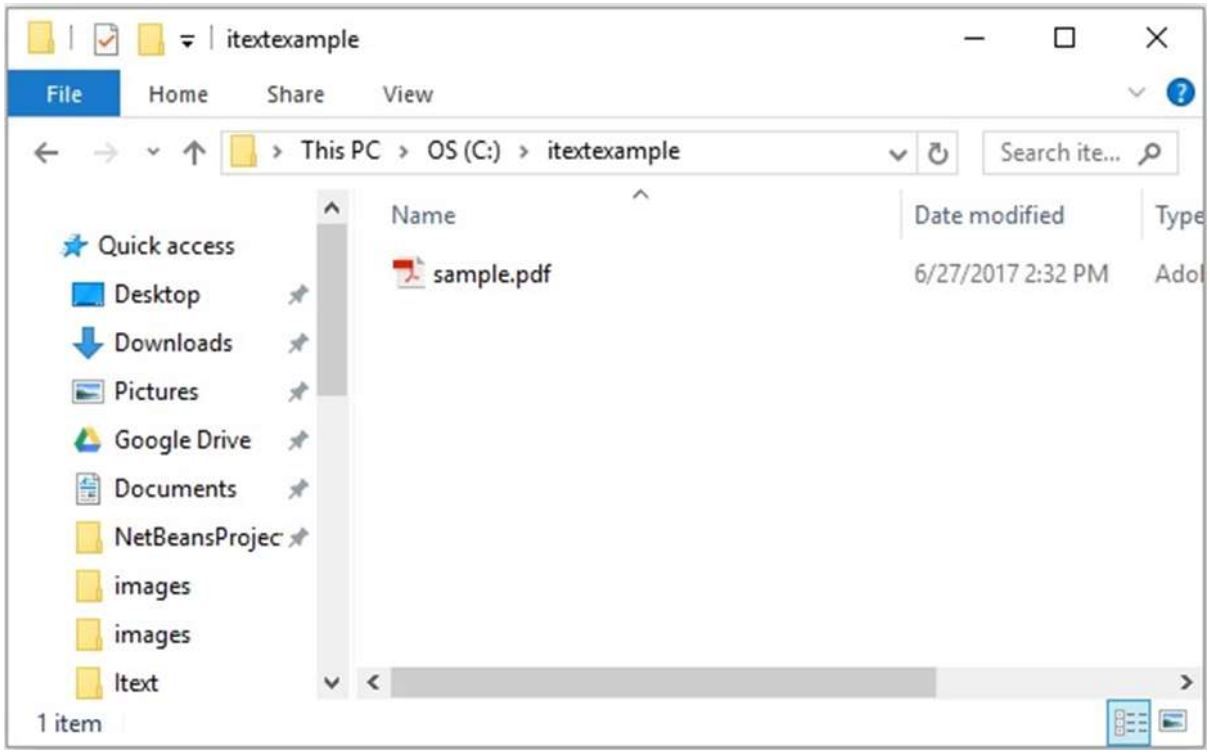
Compile and execute the saved Java file from the Command prompt using the following commands:

```
javac create_PDF.java
java create_PDF
```

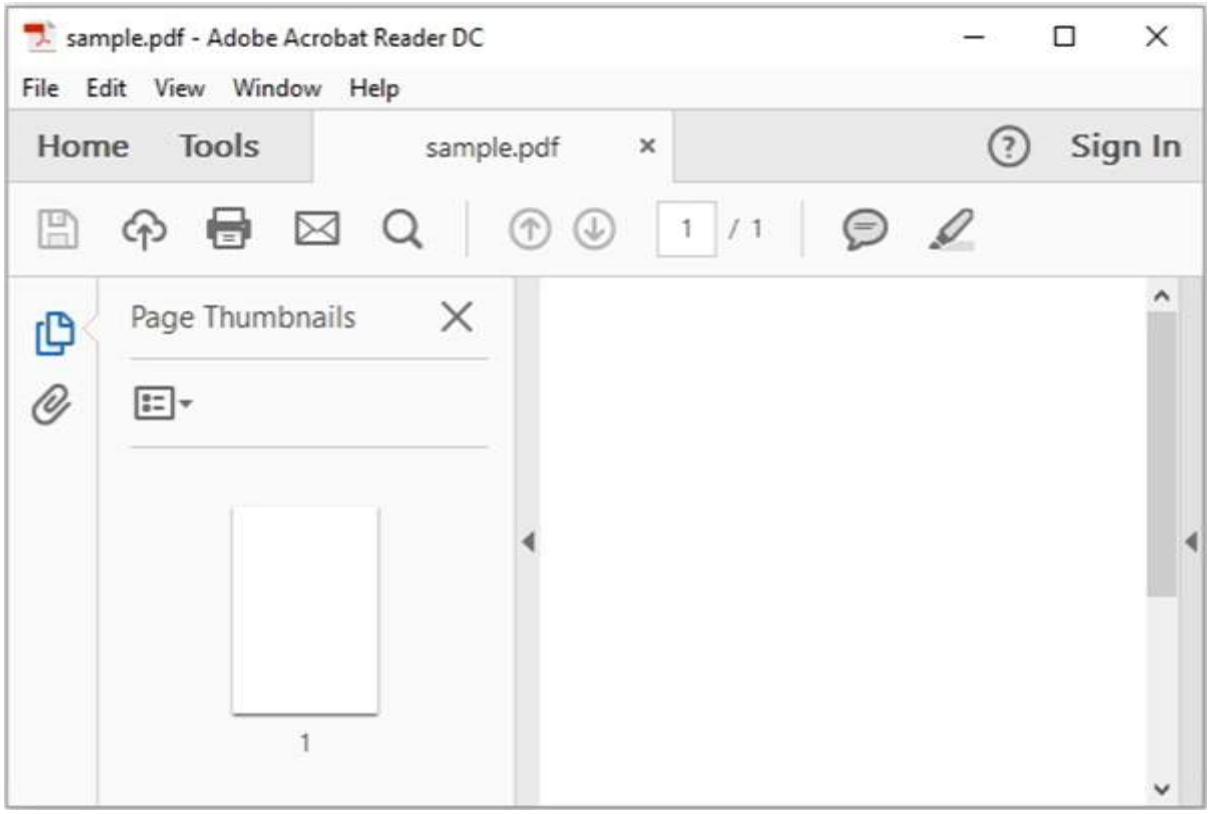
Upon execution, the above program creates a PDF document, displaying the following message.

```
PDF created
```

If you verify the specified path, you can find the created PDF document as shown below.



Since this is an empty document, if you try to open this document, it will display an error message, as shown in the following screenshot.



3. iText – Adding an AreaBreak

In this chapter, we will see how to create a PDF document with AreaBreak using the iText library.

Creating an AreaBreak

You can create an empty PDF Document by instantiating the **Document** class. While instantiating this class, you need to pass a **PdfDocument** object as a parameter, to its constructor. Then, to add an areabreak to the document, you need to instantiate the **AreaBreak** class and add this object to document using the **add()** method.

Following are the steps to create an empty PDF document with AreaBreak.

Step 1: Creating a PdfWriter object

The **PdfWriter** class represents the Doc Writer for a PDF, this class belongs to the package **com.itextpdf.kernel.pdf**. The constructor of this class accepts a string, representing the path of the file where the PDF is to be created.

Instantiate PdfWriter class by passing a string value representing the path where you need to create a PDF, to its constructor, as shown below.

```
// Creating a PdfWriter
String dest = "C:/itextExamples/addingAreaBreak.pdf";
PdfWriter writer = new PdfWriter(dest);
```

When an object of this type is passed to a PdfDocument (class), then every element added to this document will be written to the file specified.

Step 2: Creating a PdfDocument object

The **PdfDocument** class is the class that represents the PDF Document in iText, this class belongs to the package **com.itextpdf.kernel.pdf.** To instantiate this class (in writing mode) you need to pass an object of the class **PdfWriter** to its constructor.

Instantiate the PdfDocument class by passing above created PdfWriter object to its constructor, as shown below.

```
// Creating a PdfDocument
PdfDocument pdfDoc = new PdfDocument(writer);
```

Once a PdfDocument object is created you can add various elements like page, font, file attachment, event handler using the respective methods provided by its class.

Step 3: Creating a Document object

The **Document** class of the package **com.itextpdf.layout** is the root element while creating a self-sufficient PDF. One of the constructors of this class accepts an object of the class PdfDocument.

Instantiate the **Document** class by passing the object of the class **PdfDocument** created in the previous steps, as shown below.

```
// Creating a Document
Document document = new Document(pdfDoc);
```

Step 4: Creating an Area Break object

The **AreaBreak** class belongs to the package **com.itextpdf.layout.element**. On instantiating this class, the current context area will be terminated and a new one will be created with the same size (in case we use default constructor).

Instantiate the **AreaBreak** class as shown below.

```
// Creating an Area Break
AreaBreak aB = new AreaBreak();
```

Step 5: Adding AreaBreak

Add the **areabreak** object created in the previous step using the **add()** method of the **Document** class, as shown below.

```
// Adding area break to the PDF
document.add(aB);
```

Step 6: Closing the Document

Close the document using the **close()** method of the **Document** class as shown below.

```
// Closing the document
document.close();
```

Example

The following Java program demonstrates how to create a PDF document with AreaBreak using the iText library. It creates a PDF document with the name **addingAreaBreak.pdf**, adds an **areabreak** to it, and saves it in the path **C:/itextExamples/**

Save this code in a file with the name **AddingAreaBreak.java**.

```
import com.itextpdf.kernel.pdf.PdfDocument;
import com.itextpdf.kernel.pdf.PdfWriter;
import com.itextpdf.layout.Document;
import com.itextpdf.layout.element.AreaBreak;

public class AddingAreaBreak {
   public static void main(String args[]) throws Exception{
      // Creating a PdfWriter
      String dest = "C:/itextExamples/addingAreaBreak.pdf";
      PdfWriter writer = new PdfWriter(dest);
```

```
      // Creating a PdfDocument
      PdfDocument pdf = new PdfDocument(writer);

      // Creating a Document by passing PdfDocument object to its constructor
      Document document = new Document(pdf);

      // Creating an Area Break
      AreaBreak aB = new AreaBreak();

      // Adding area break to the PDF
      document.add(aB);

      // Closing the document
      document.close();

      System.out.println("Pdf created");
   }
}
```

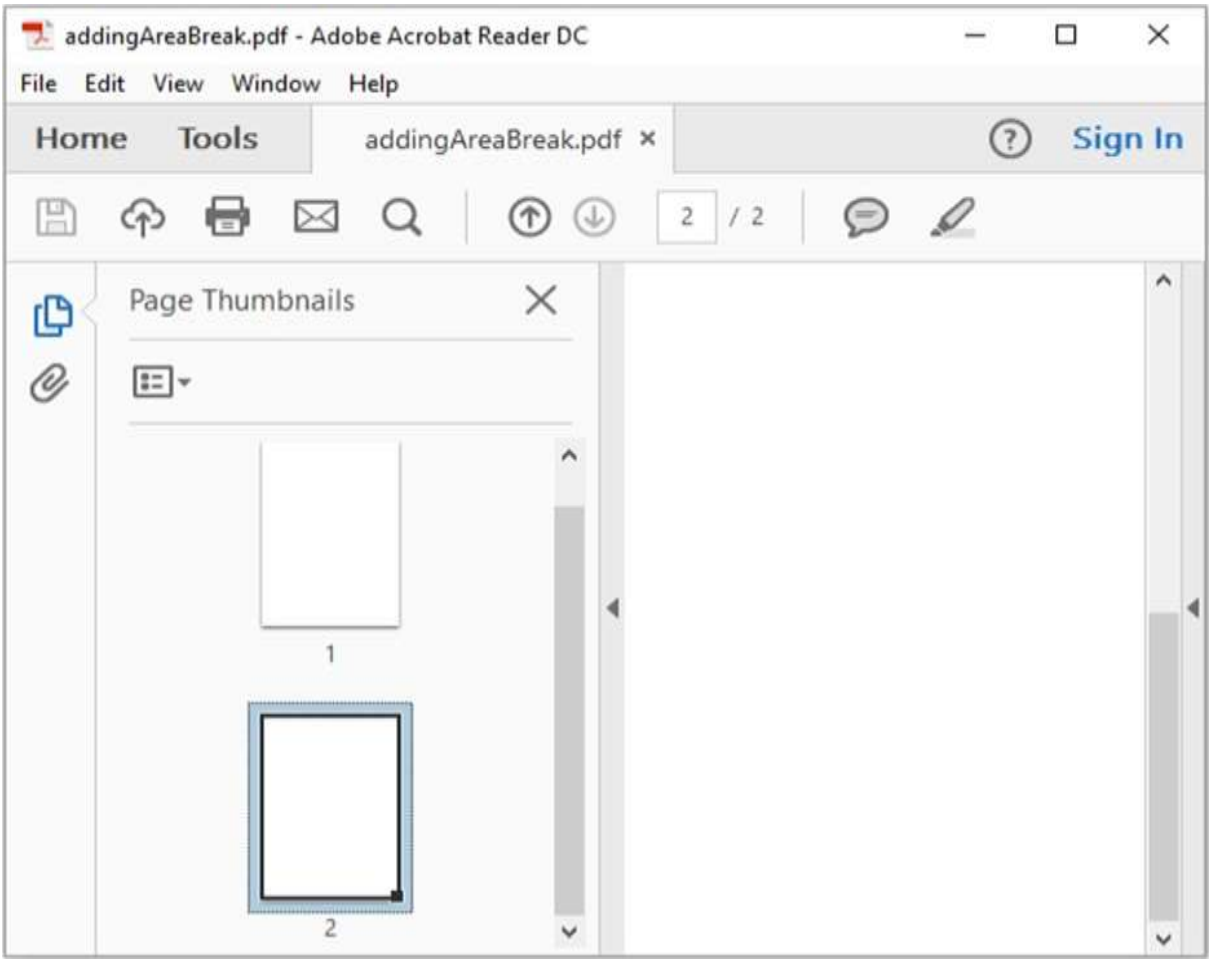
Compile and execute the saved Java file from the Command prompt using the following commands:

```
javac AddingAreaBreak.java
java AddingAreaBreak
```

Upon execution, the above program creates a PDF document, displaying the following message.

```
Pdf Created
```

If you verify the specified path, you can find the created PDF document, as shown below.


addingAreaBreak.pdf - Adobe Acrobat Reader DC
File Edit View Window Help
Home Tools
addingAreaBreak.pdf
Sign In
2 / 2
Page Thumbnails
1
2

4. iText – Adding a Paragraph

In this chapter, we will see how to create a PDF document and add a paragraph to it using the iText library.

Creating a Paragraph

You can create an empty PDF Document by instantiating the **Document** class. While instantiating this class, you need to pass a **PdfDocument** object as a parameter, to its constructor. Then, to add a paragraph to the document, you need to instantiate the **Paragraph** class and add this object to the document using the **add()** method.

Following are the steps to create a PDF document with a paragraph in it.

Step 1: Creating a PdfWriter object

The **PdfWriter** class represents the Doc Writer for a PDF. This class belongs to the package **com.itextpdf.kernel.pdf**. The constructor of this class accepts a string, representing the path of the file where the PDF is to be created.

Instantiate the PdfWriter class by passing a string value (representing the path where you need to create a PDF) to its constructor, as shown below.

```
// Creating a PdfWriter
String dest = "C:/itextExamples/addingParagraph.pdf";
PdfWriter writer = new PdfWriter(dest);
```

When the object of this type is passed to a PdfDocument (class), every element added to this document will be written to the file specified.

Step 2: Creating a PdfDocument

The **PdfDocument** class is the class that represents the PDF Document in iText. This class belongs to the package **com.itextpdf.kernel.pdf.** To instantiate this class (in writing mode), you need to pass an object of the class **PdfWriter** to its constructor.

Instantiate the PdfDocument class by passing the above created PdfWriter object to its constructor, as shown below.

```
// Creating a PdfDocument
PdfDocument pdfDoc = new PdfDocument(writer);
```

Once a PdfDocument object is created, you can add various elements like page, font, file attachment, and event handler using the respective methods provided by its class.

Step 3: Creating the Document class

The **Document** class of the package **com.itextpdf.layout** is the root element. While creating a self-sufficient PDF. One of the constructors of this class accepts an object of the class PdfDocument.

Instantiate the Document class by passing the object of the class **PdfDocument** created in the previous steps as shown below.

```
// Creating a Document
Document document = new Document(pdfDoc);
```

Step 4: Creating a Paragraph object

The **Paragraph** class represents a self-contained block of textual and graphical information. It belongs to the package **com.itextpdf.layout.element.**

Instantiate the **Paragraph** class by passing the text content as a string to its constructor, as shown below.

```
String para = "Welcome to Tutorialspoint.";
// Creating an Area Break
Paragraph para = new Paragraph (para);
```

Step 5: Adding Paragraph

Add the **Paragraph** object created in the previous step using the **add()** method of the **Document** class, as shown below.

```
// Adding area break to the PDF
document.add(para);
```

Step 6: Closing the Document

Close the document using the **close()** method of the **Document** class, as shown below.

```
// Closing the document
document.close();
```