iText - Quick Guide

Previous Page Next Page

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. Using this library, you can develop Java programs that create, convert and manipulate 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 above listed software's 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.

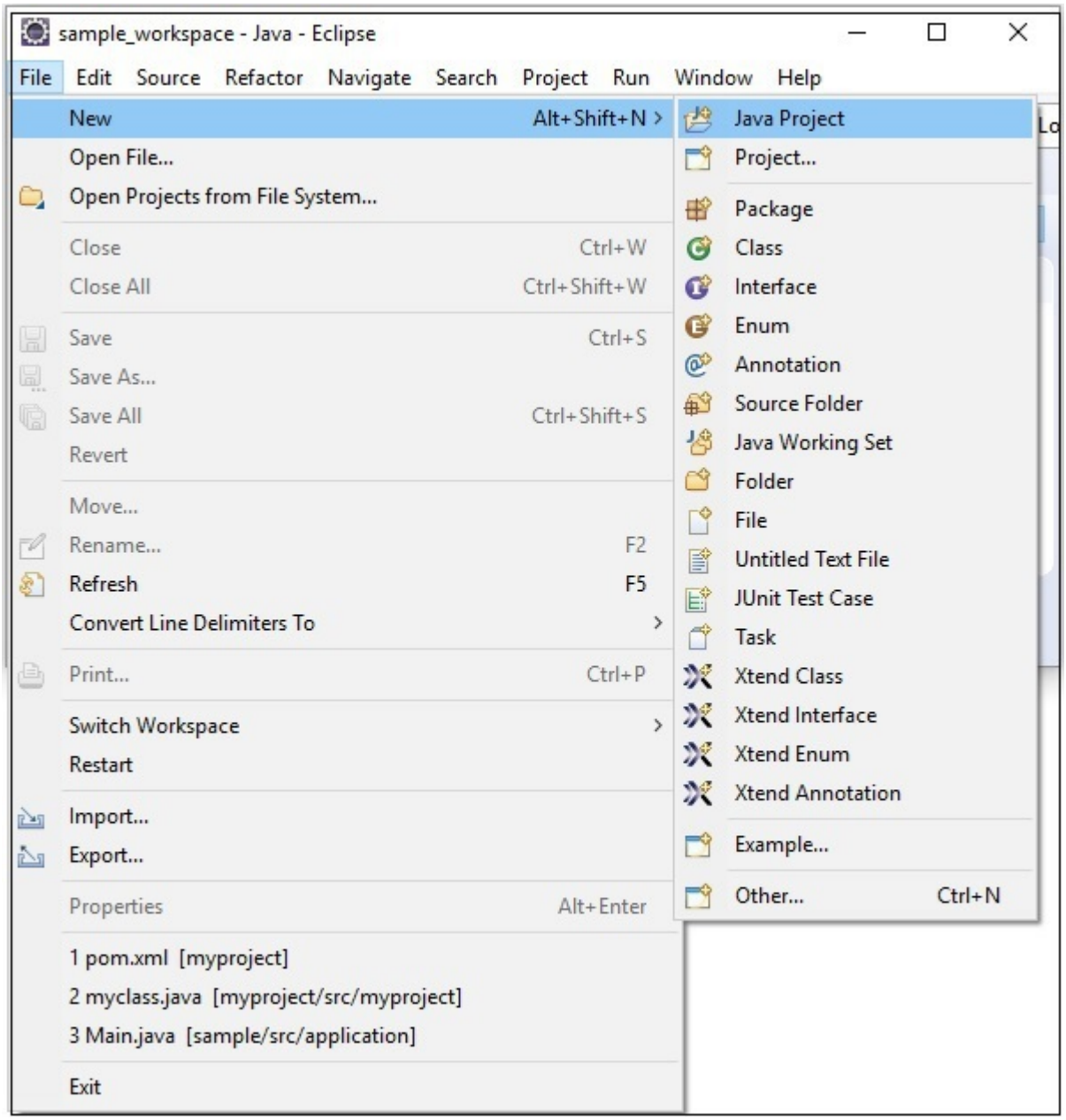
Canvas – iText library provides you a Canvas class using which you can draw various geometrical shapes on a PDF document like circle, line, etc.

Create PDFs – Using iText, you can create a new PDF file from your Java programs. You can include images and fonts too.

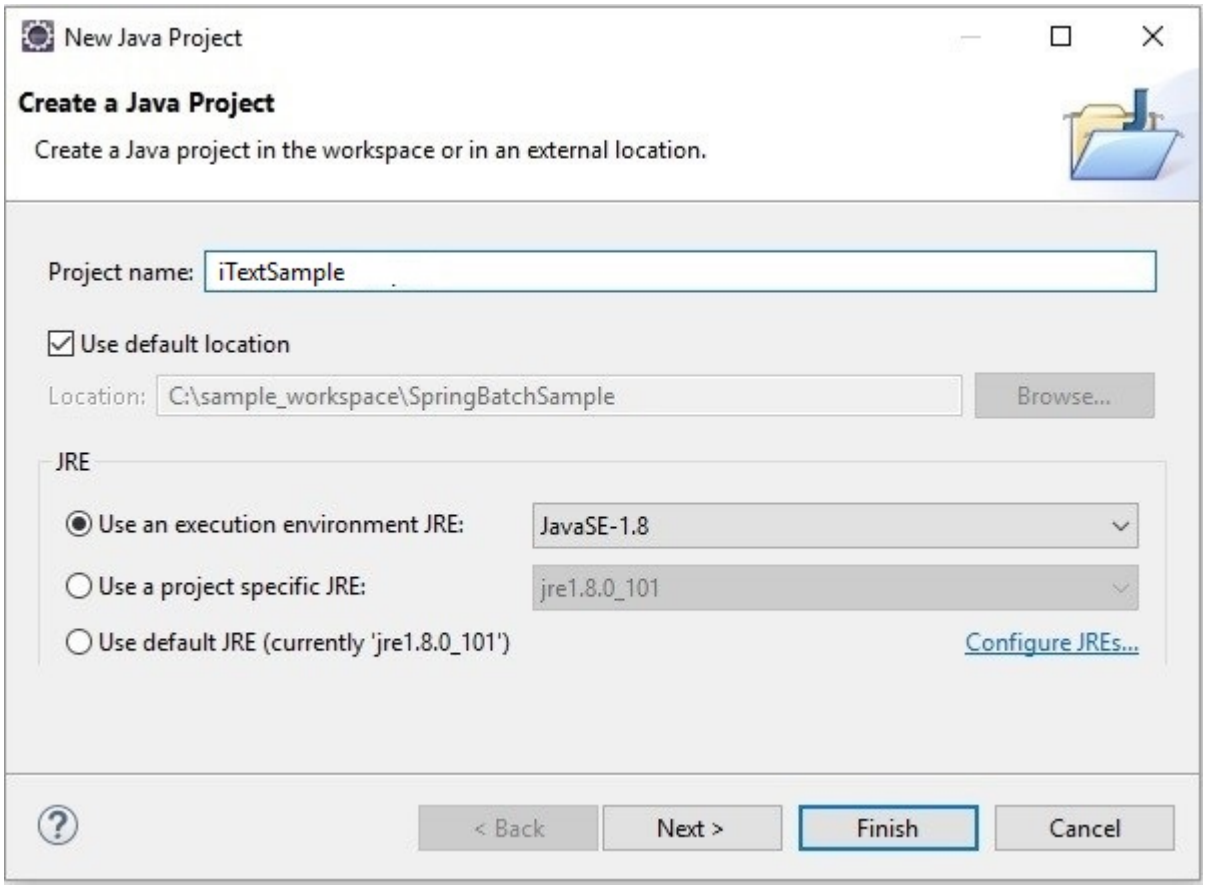
IText Environment

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

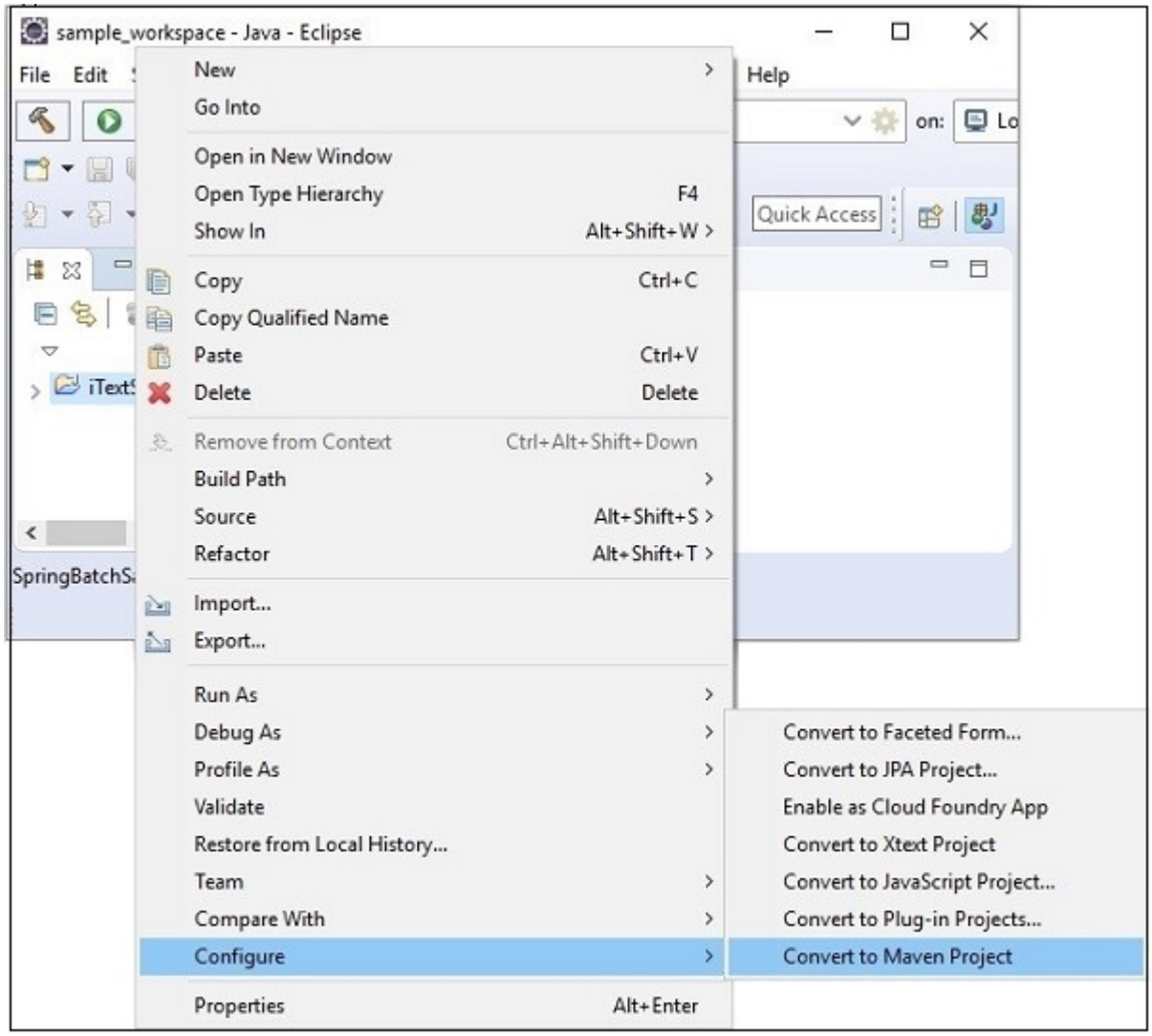
Step 1 – Install Eclipse and open a new project in it as shown below.



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

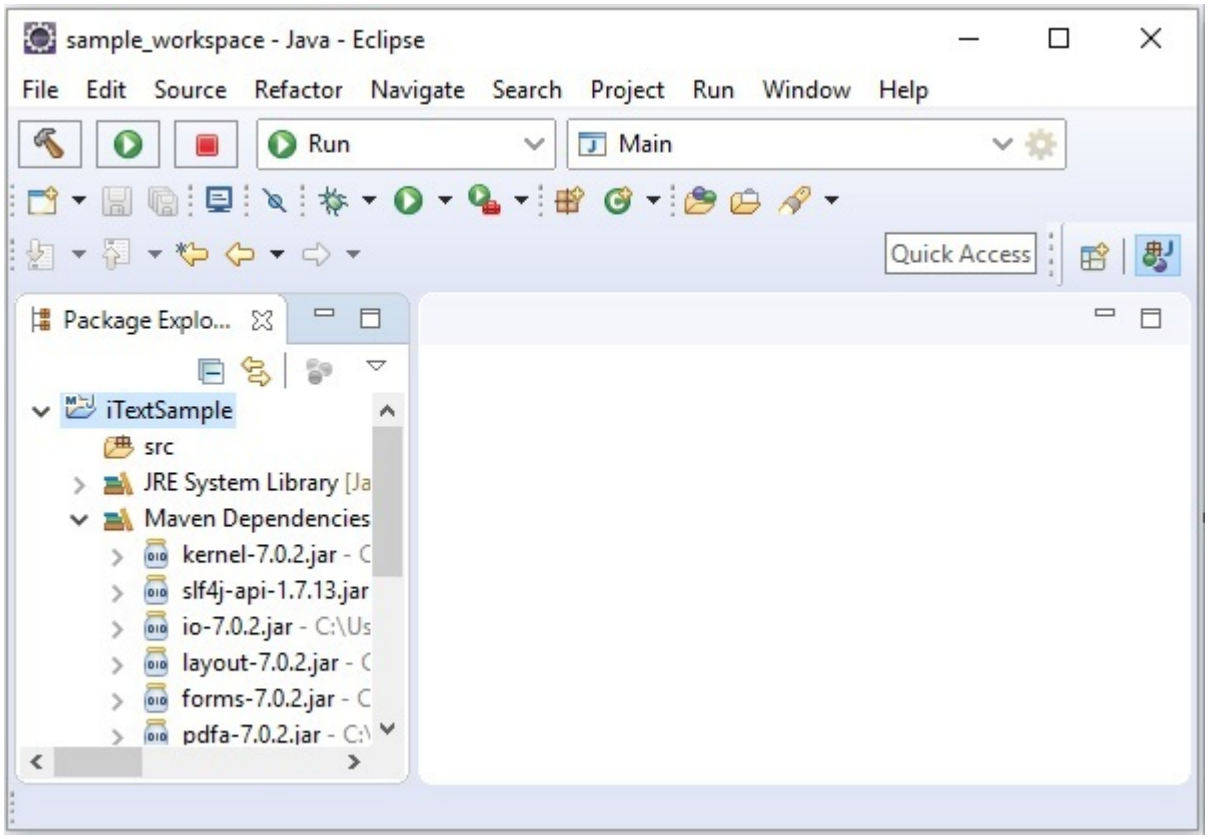
Convert the project into Maven project and add the following content to its **pom.xml**.

```
<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.



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");
   }
}
```

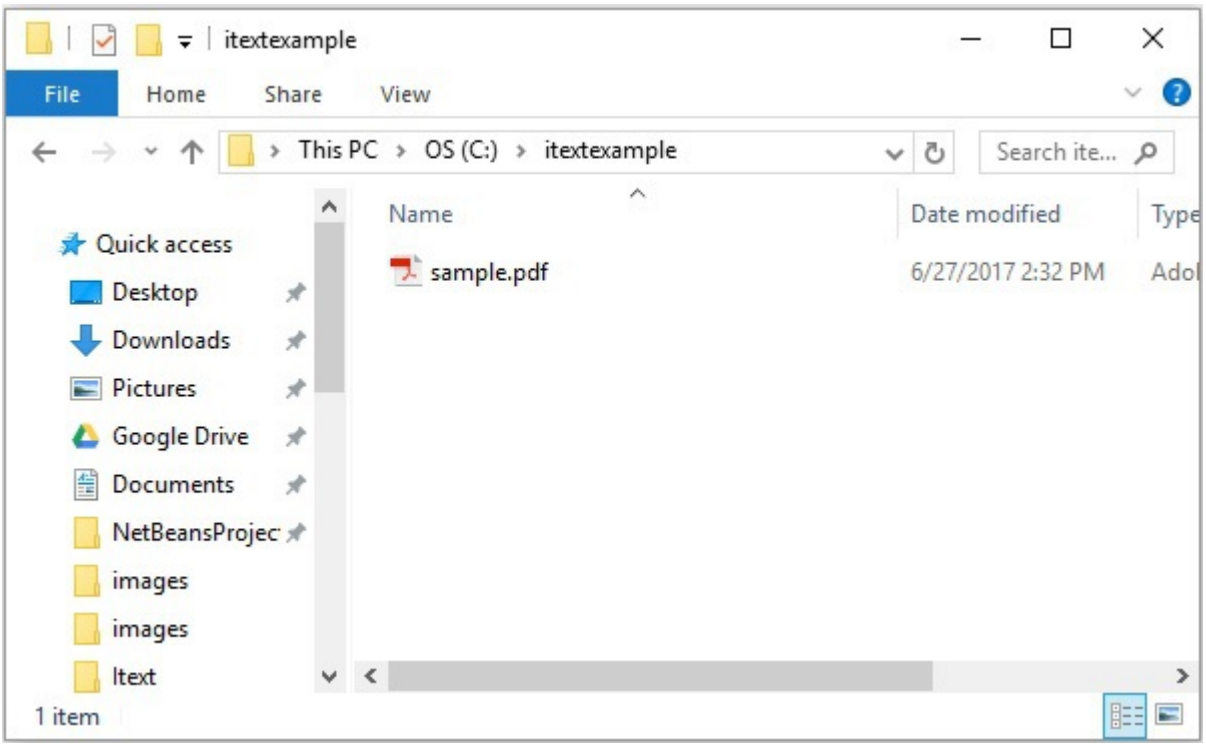
Compile and execute the saved Java file from the Command prompt using the following commands –

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

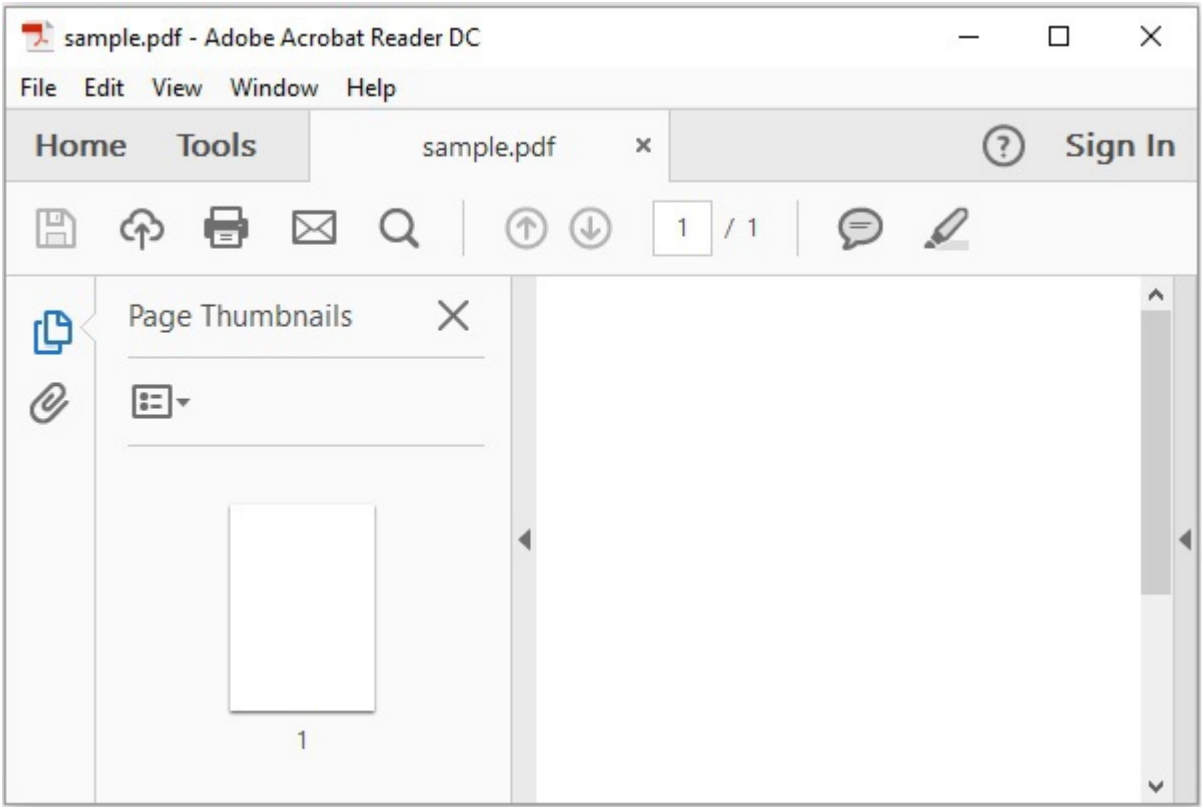
Upon execution, the above program creates a PDF document, displaying the following message.

```
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.



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");
   }
}
```

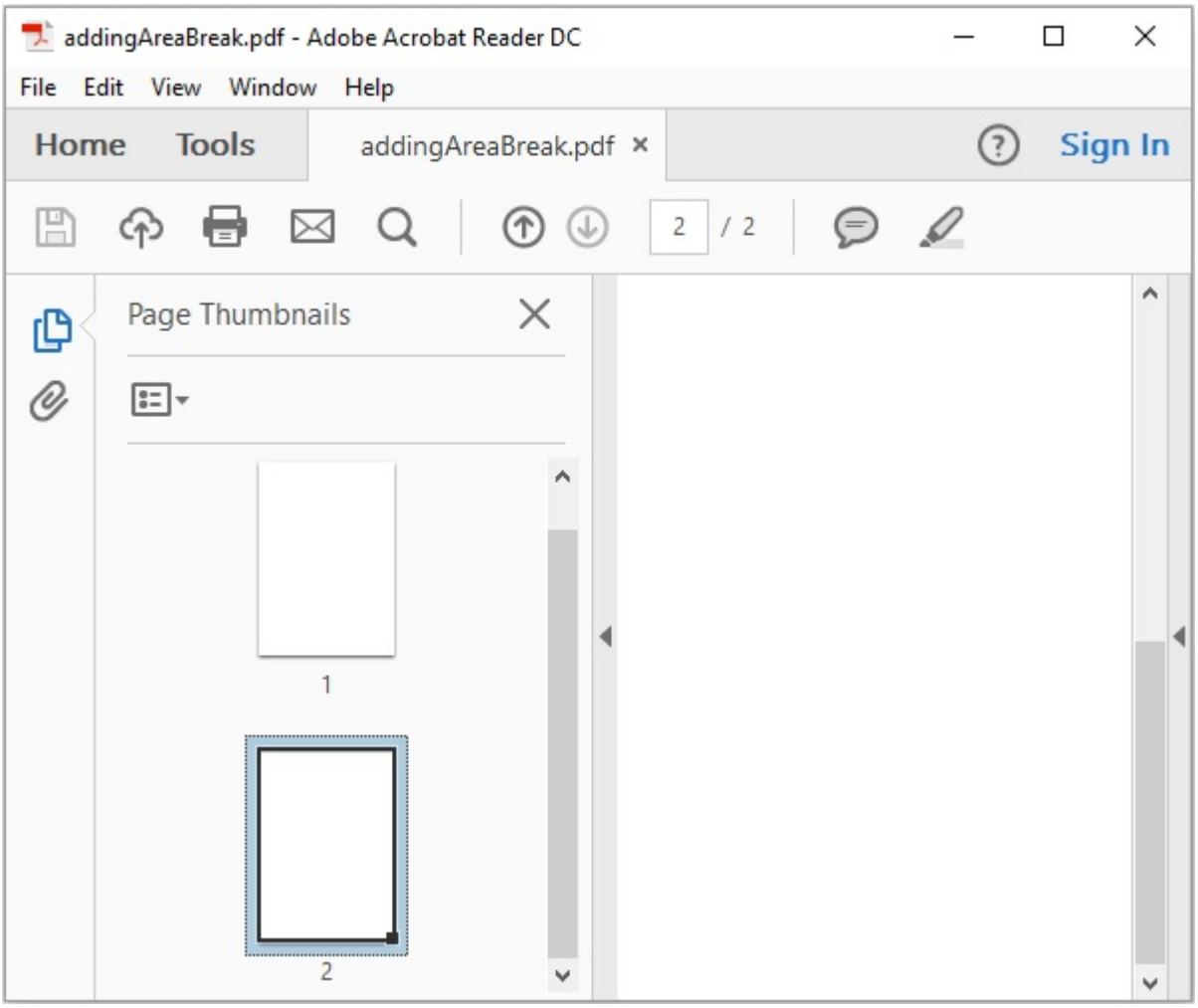
Compile and execute the saved Java file from the Command prompt using the following commands –

```
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.



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();
```

Example

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

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

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

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

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

      // Creating a Document
      Document document = new Document(pdf);
      String para1 = "Tutorials Point originated from the idea that there exists
      a class of readers who respond better to online content and prefer to learn
      new skills at their own pace from the comforts of their drawing rooms.";

      String para2 = "The journey commenced with a single tutorial on HTML in 2006
      and elated by the response it generated, we worked our way to adding fresh
      tutorials to our repository which now proudly flaunts a wealth of tutorials
      and allied articles on topics ranging from programming languages to web designing
      to academics and much more.";

      // Creating Paragraphs
      Paragraph paragraph1 = new Paragraph(para1);
      Paragraph paragraph2 = new Paragraph(para2);

      // Adding paragraphs to document
      document.add(paragraph1);
      document.add(paragraph2);

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

```
        System.out.println("Paragraph added");
    }
}
```

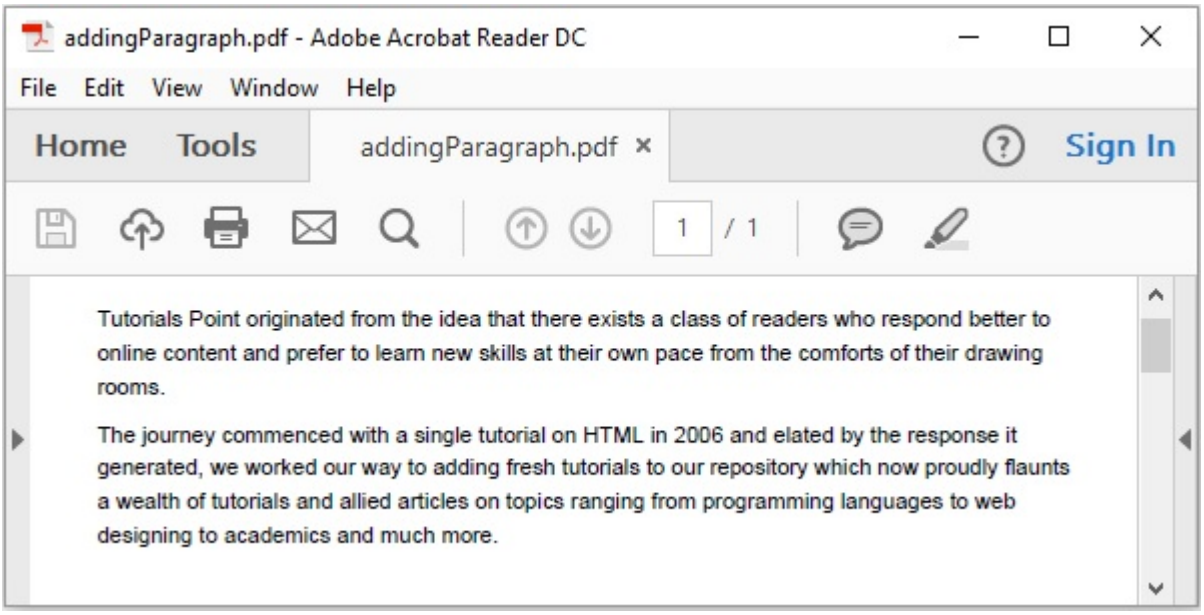
Compile and execute the saved Java file from the Command prompt using the following commands –

```
javac AddingParagraph.java
java AddingParagraph
```

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

```
Paragraph added
```

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



iText - Adding a List

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

Creating a List

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 list to the document, you need to instantiate the **List** class and add this object to the document using the **add()** method.

Following are the steps to create a PDF document and add a List in it.

Step 1: Creating a PdfWriter object

The **PdfWriter** class represents the DocWriter 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/addingList.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 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: Creating the 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 a List object

The **List** class represents a series of objects that are vertically outlined. It belongs to the package **com.itextpdf.layout.element**.

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

```
// Creating a list
List list = new List();
```

Step 5: Adding elements to the list

Add contents to the **list** object using the **add()** method of the **List** class by passing String values, as shown below.

```
//  Add elements to the list
list.add("Java");
list.add("JavaFX");
list.add("Apache Tika");
list.add("OpenCV");
```

Step 6: Adding List to the document

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

```
// Adding list to the document
document.add(list);
```

Step 7: 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 and add a list to it using the iText library. It creates a PDF document with the name **addingList.pdf**, adds a list to it, and saves it in the path **C:/itextExamples/**.

Save this code in a file with name **AddingList.java**.

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

import com.itextpdf.layout.Document;
import com.itextpdf.layout.element.List;
import com.itextpdf.layout.element.Paragraph;

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

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

      // Creating a Document
```

```
      Document document = new Document(pdf);

      // Creating a Paragraph
      Paragraph paragraph = new Paragraph("Tutorials Point provides the following tutorials");

      // Creating a list
      List list = new List();

      // Add elements to the list
      list.add("Java");
      list.add("JavaFX");
      list.add("Apache Tika");
      list.add("OpenCV");
      list.add("WebGL");
      list.add("Coffee Script");
      list.add("Java RMI");
      list.add("Apache Pig");

      // Adding paragraph to the document
      document.add(paragraph);

      // Adding list to the document
      document.add(list);

      // Closing the document
      document.close();
      System.out.println("List added");
   }
}
```

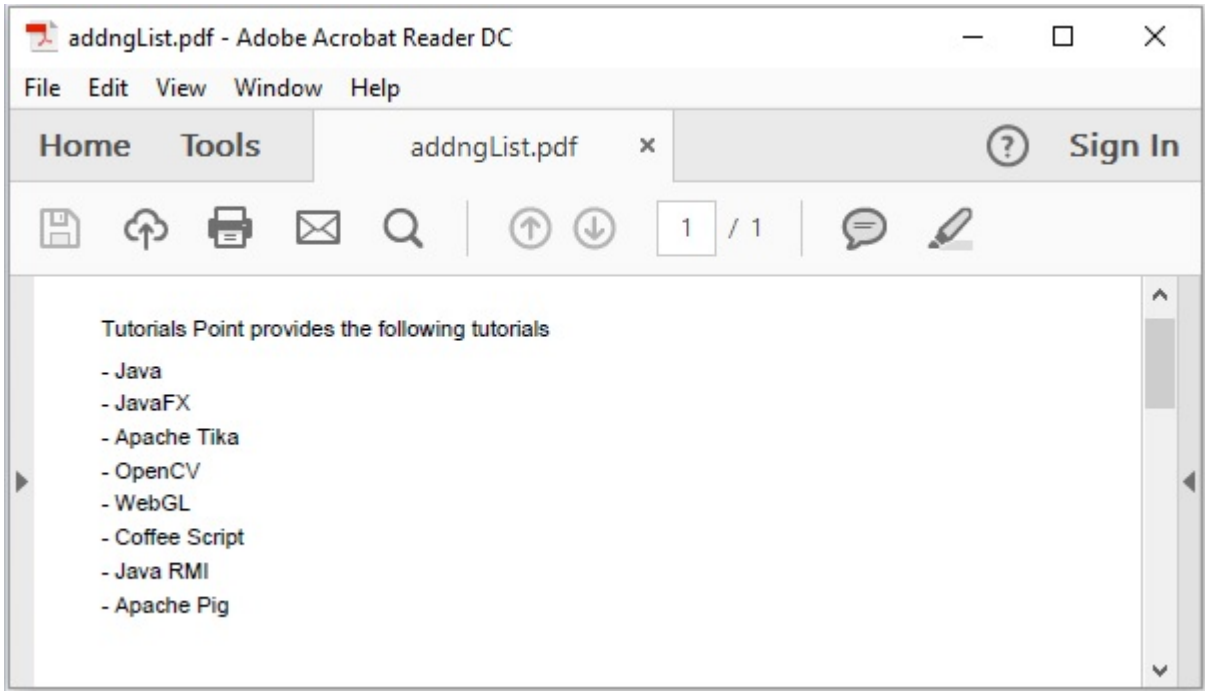
Compile and execute the saved Java file from the Command prompt using the following commands –

```
javac AddingList.java
java AddingList
```

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

```
List added
```

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



iText - Adding a Table

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

Adding a Table to a Pdf

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 table to the document, you need to instantiate the **Table** class and add this object to the document using the **add()** method.

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

Step 1: Creating a PdfWriter object

The **PdfWriter** class represents the DocWriter 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/addingTable.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 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: Creating the 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 a Table object

The **Table** class represents a two-dimensional grid filled with cells ordered in rows and columns. It belongs to the package **com.itextpdf.layout.element**.

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

```
// Creating a table object
float [] pointColumnWidths = {150F, 150F, 150F};
Table table = new Table(pointColumnWidths);
```

Step 5: Adding cells to the table

Create a **cell** object by instantiating the **Cell** class of the package **com.itextpdf.layout.element**. Add the contents of the cell using the **add()** method of this class.

Finally, to add this cell to the table, call the **addCell()** method of the **Table** class and pass the **cell** object as a parameter to this method, as shown below.

```
// Adding cell 1 to the table
Cell cell1 = new Cell();   // Creating a cell
cell1.add("Name");         // Adding content to the cell
table.addCell(cell1);      // Adding cell to the table
```

```
// Adding cell 2 to the table Cell
cell2 = new Cell();       // Creating a cell
cell2.add("Raju");        // Adding content to the cell
table.addCell(cell2);     // Adding cell to the table
```

Step 6: Adding table to the document

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

```
// Adding list to the document
document.add(table);
```

Step 7: 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 and add a table to it using the iText library. It creates a PDF document with the name **addingTable.pdf**, adds a table to it, and saves it in the path **C:/itextExamples/**

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

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

import com.itextpdf.layout.Document;
import com.itextpdf.layout.element.Cell;
import com.itextpdf.layout.element.Table;

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

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

      // Creating a Document object
      Document doc = new Document(pdf);

      // Creating a table
      float [] pointColumnWidths = {150F, 150F, 150F};
      Table table = new Table(pointColumnWidths);

      // Adding cells to the table
```

```
        table.addCell(new Cell().add("Name"));
        table.addCell(new Cell().add("Raju"));
        table.addCell(new Cell().add("Id"));
        table.addCell(new Cell().add("1001"));
        table.addCell(new Cell().add("Designation"));
        table.addCell(new Cell().add("Programmer"));

        // Adding Table to document
        doc.add(table);

        // Closing the document
        doc.close();
        System.out.println("Table created successfully..");
    }
}
```

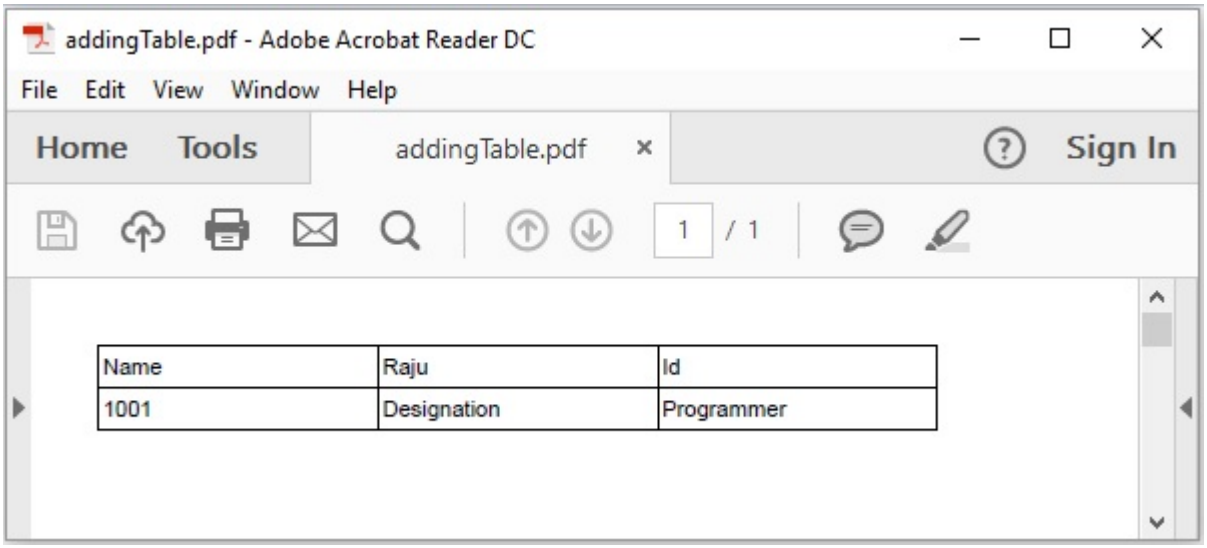
Compile and execute the saved Java file from the Command prompt using the following commands –

```
javac AddingTable.java
java AddingTable
```

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

```
Table created successfully..
```

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



iText - Formatting Cell Contents

In this chapter, we will see how to create a PDF document and add a table and format the contents of a cell in the table using the iText library.

Formatting the Cells in a Table

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 table to the document, you need to instantiate the **Table** class and add this object to the document using the **add()** method. You can format the contents of a cell in a table using the methods of the **Cell** class.

Following are the steps to format the contents of a cell in a table.

Step 1: Creating a PdfWriter object

The **PdfWriter** class represents the DocWriter 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/addingBackground.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 PDFDocument 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 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 a Table object

The **Table** class represents a two-dimensional grid filled with cells, ordered in rows and columns. It belongs to the package **com.itextpdf.layout.element**.

Instantiate the Table class as shown below.

```
// Creating a table
float [] pointColumnWidths = {200F, 200F};
Table table = new Table(pointColumnWidths);
```

Step 5: Creating cells

Create a **cell** object by instantiating the **Cell** class of the package **com.itextpdf.layout.element**. Add the contents of the cell using the **add()** method of the **Cell** class, as shown below.

```
// Adding cell 1 to the table
Cell cell1 = new Cell();     // Creating a cell
cell1.add("Name");           // Adding content to the cell

// Adding cell 2 to the table
Cell cell2 = new Cell();     // Creating a cell
cell2.add("Raju");           // Adding content to the cell
```

Step 6: Adding Background to the cell

Once you created the cell and added contents to it, you can format the cell. For example, you can set its background, align the text inside the cell, change the text color, etc., using different methods of the cell class such as **setBackgroundColor(), setBorder(), setTextAlignment()**.

You can set the background color, border, and text alignment to the cell created in the previous step, as shown below.

```
c1.setBackgroundColor(Color.DARK_GRAY);    // Setting background color to cell1
c1.setBorder(Border.NO_BORDER);            // Setting border to cell1
c1.setTextAlignment(TextAlignment.CENTER); // Setting text alignment to cell1
```

Step 7: Adding cell to the table

Finally, to add this cell to the table, call the **addCell()** method of the **Table** class and pass the **cell** object as a parameter to this method, as shown below.

```
table.addCell(c1);
```

Step 8: Adding table to the document

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

```
// Adding list to the document
document.add(table);
```

Step 9: 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 format the contents of a cell in a table using the iText library. It creates a PDF document with the name **addingBackground.pdf**, adds a table to it, formats the contents of its cells, and saves it in the path **C:/itextExamples/**

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

```
import com.itextpdf.kernel.color.Color;
import com.itextpdf.kernel.pdf.PdfDocument;
import com.itextpdf.kernel.pdf.PdfWriter;

import com.itextpdf.layout.Document;
import com.itextpdf.layout.border.Border;
import com.itextpdf.layout.element.Cell;
import com.itextpdf.layout.element.Table;
import com.itextpdf.layout.property.TextAlignment;

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

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

      // Creating a Document object
      Document doc = new Document(pdfDoc);

      // Creating a table
      float [] pointColumnWidths = {200F, 200F};
      Table table = new Table(pointColumnWidths);

      // Populating row 1 and adding it to the table
      Cell c1 = new Cell();                          // Creating cell 1
      c1.add("Name");                                // Adding name to cell 1
      c1.setBackgroundColor(Color.DARK_GRAY);        // Setting background color
      c1.setBorder(Border.NO_BORDER);                // Setting border
      c1.setTextAlignment(TextAlignment.CENTER);     // Setting text alignment
      table.addCell(c1);                             // Adding cell 1 to the table

      Cell c2 = new
      Cell();
```

```
      c2.add("Raju");
      c2.setBackgroundColor(Color.GRAY);
      c2.setBorder(Border.NO_BORDER);
      c2.setTextAlignment(TextAlignment.CENTER);
      table.addCell(c2);

      // Populating row 2 and adding it to the table
      Cell c3 = new Cell();
      c3.add("Id");
      c3.setBackgroundColor(Color.WHITE);
      c3.setBorder(Border.NO_BORDER);
      c3.setTextAlignment(TextAlignment.CENTER);
      table.addCell(c3);

      Cell c4 = new Cell();
      c4.add("001");
      c4.setBackgroundColor(Color.WHITE);
      c4.setBorder(Border.NO_BORDER);
      c4.setTextAlignment(TextAlignment.CENTER);
      table.addCell(c4);

      // Populating row 3 and adding it to the table
      Cell c5 = new Cell();
      c5.add("Designation");
      c5.setBackgroundColor(Color.DARK_GRAY);
      c5.setBorder(Border.NO_BORDER);
      c5.setTextAlignment(TextAlignment.CENTER);
      table.addCell(c5);

      Cell c6 = new Cell();
      c6.add("Programmer");
      c6.setBackgroundColor(Color.GRAY);
      c6.setBorder(Border.NO_BORDER);
      c6.setTextAlignment(TextAlignment.CENTER);
      table.addCell(c6);

      // Adding Table to document
      doc.add(table);

      // Closing the document
      doc.close();

      System.out.println("Background added successfully..");
   }
}
```

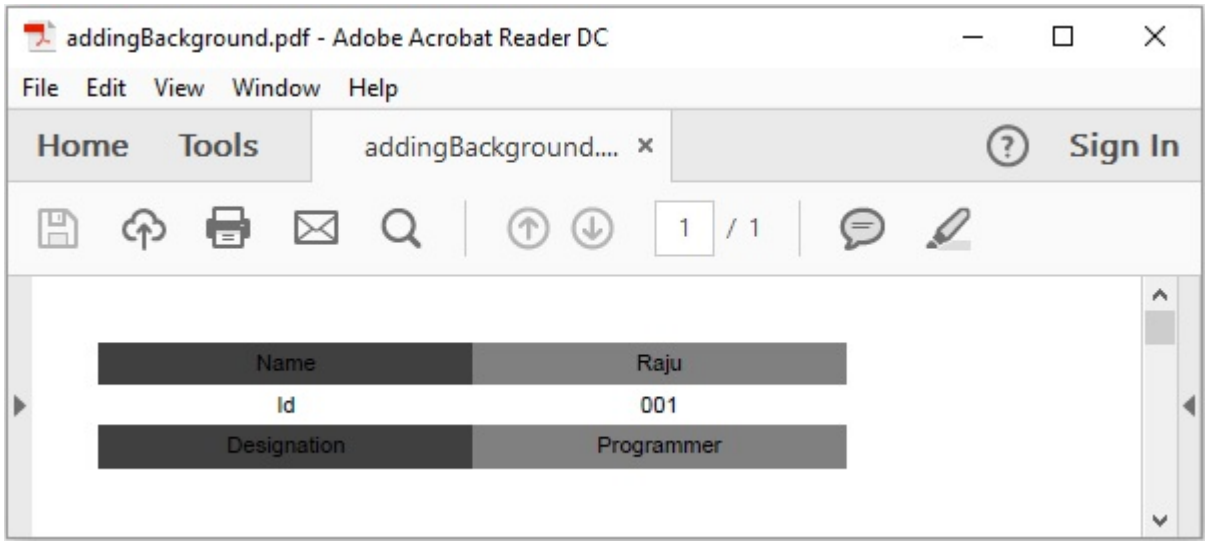
Compile and execute the saved Java file from the Command prompt using the following commands –

```
javac BackgroundToTable.java
java BackgroundToTable
```

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

```
Background added successfully..
```

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



iText - Formatting the Borders of a Cell

In this chapter, we will see how to format the borders of a cell in a table using iText library.

Formatting the Borders of a Cell

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 table to the document, you need to instantiate the **Table** class and add this object to the document using the **add()** method.

You can add various types of borders like **DashedBorder, SolidBorder, DottedBorder, DoubleBorder, RoundDotsBorder**, etc. with various colors using the **setBorder()** method of the **Cell** class.

Following are the steps to format the borders of a cell in a table.

Step 1: Creating a PdfWriter object

The **PdfWriter** class represents the DocWriter 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/coloredBorders.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 PDFDocument 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 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 a Table object

The **Table** class represents a two-dimensional grid filled with cells ordered in rows and columns. It belongs to the package **com.itextpdf.layout.element**.

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

```
// Creating a table
float [] pointColumnWidths = {200F, 200F};
Table table = new Table(pointColumnWidths);
```

Step 5: Creating cells

Create a cell object by instantiating the **Cell** class of the package **com.itextpdf.layout.element** add the contents of the cell using the **add()** method of the **Cell** class, as shown below.

```
// Adding cell 1 to the table
Cell cell1 = new Cell(); // Creating a cell
cell1.add("Name");       // Adding content to the cell
```

Step 6: Formatting the border of the cell

The iText library provides various classes representing the border such as **DashedBorder, SolidBorder, DottedBorder, DoubleBorder, RoundDotsBorder**, etc.

The constructors of these classes accept two parameters: a **color** object representing the color of the border and an **integer** representing the width of the border.

Choose one of this border types and instantiate the respective border by passing the **color** object and an **integer** representing the width, as shown below.

```
Border b1 = new DashedBorder(Color.RED, 3);
```

Now, set the border of the cell using the **setBorder()** method of the **cell** class. This method accepts an object of the type **Border** as a parameter.

Set the border of the cell by passing the above created **Border** object as a parameter to the **setBorder()** method as shown below.

```
c1.setBorder(b1)
```

Finally, to add this cell to the table, call the **addCell()** method of the **Table** class and pass the **cell** object as a parameter to this method, as shown below.

```
table.addCell(c1);
```

Step 7: Adding table to the document

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

```
// Adding list to the document
document.add(table);
```

Step 8: 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 format the border of a cell in a table using the iText library. It creates a PDF document with the name **coloredBorders.pdf**, adds a table to it, formats the contents of its cells, and saves it in the path **C:/itextExamples/**

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

```
import com.itextpdf.kernel.color.Color;
import com.itextpdf.kernel.pdf.PdfDocument;
```

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

import com.itextpdf.layout.Document;
import com.itextpdf.layout.border.Border;
import com.itextpdf.layout.border.DashedBorder;
import com.itextpdf.layout.border.DottedBorder;
import com.itextpdf.layout.border.DoubleBorder;
import com.itextpdf.layout.border.RoundDotsBorder;
import com.itextpdf.layout.border.SolidBorder;

import com.itextpdf.layout.element.Cell;
import com.itextpdf.layout.element.Table;
import com.itextpdf.layout.property.TextAlignment;

public class FormatedBorders {
   public static void main(String args[]) throws Exception {
      // Creating a PdfWriter object
      String dest = "C:/itextExamples/coloredBorders.pdf";

      PdfWriter writer = new
      PdfWriter(dest);

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

      // Creating a Document object
      Document doc = new Document(pdfDoc);

      // Creating a table
      float [] pointColumnWidths = {200F, 200F};
      Table table = new Table(pointColumnWidths);

      // Adding row 1 to the table
      Cell c1 = new Cell();

      // Adding the contents of the cell
      c1.add("Name");

      // Setting the back ground color of the cell
      c1.setBackgroundColor(Color.DARK_GRAY);

      // Instantiating the Border class
      Border b1 = new DashedBorder(Color.RED, 3);

      // Setting the border of the cell
      c1.setBorder(b1);

      // Setting the text alignment
      c1.setTextAlignment(TextAlignment.CENTER);

      // Adding the cell to the table
      table.addCell(c1);
      Cell c2 = new Cell();
      c2.add("Raju");
      c1.setBorder(new SolidBorder(Color.RED, 3));
      c2.setTextAlignment(TextAlignment.CENTER);
      table.addCell(c2);

      // Adding row 2 to the table
      Cell c3 = new Cell();
      c3.add("Id");
```

```
      c3.setBorder(new DottedBorder(Color.DARK_GRAY, 3));
      c3.setTextAlignment(TextAlignment.CENTER);
      table.addCell(c3);

      Cell c4 = new Cell();
      c4.add("001");
      c4.setBorder(new DoubleBorder(Color.DARK_GRAY, 3));
      c4.setTextAlignment(TextAlignment.CENTER);
      table.addCell(c4);

      // Adding row 3 to the table
      Cell c5 = new Cell();
      c5.add("Designation");
      c5.setBorder(new RoundDotsBorder(Color.RED, 3));
      c5.setTextAlignment(TextAlignment.CENTER);
      table.addCell(c5);

      Cell c6 = new Cell();
      c6.add("Programmer");
      c6.setBorder(new RoundDotsBorder(Color.RED, 3));
      c6.setTextAlignment(TextAlignment.CENTER);
      table.addCell(c6);

      // Adding Table to document
      doc.add(table);

      // Closing the document
      doc.close();

      System.out.println("Borders added successfully..");
   }
}
```

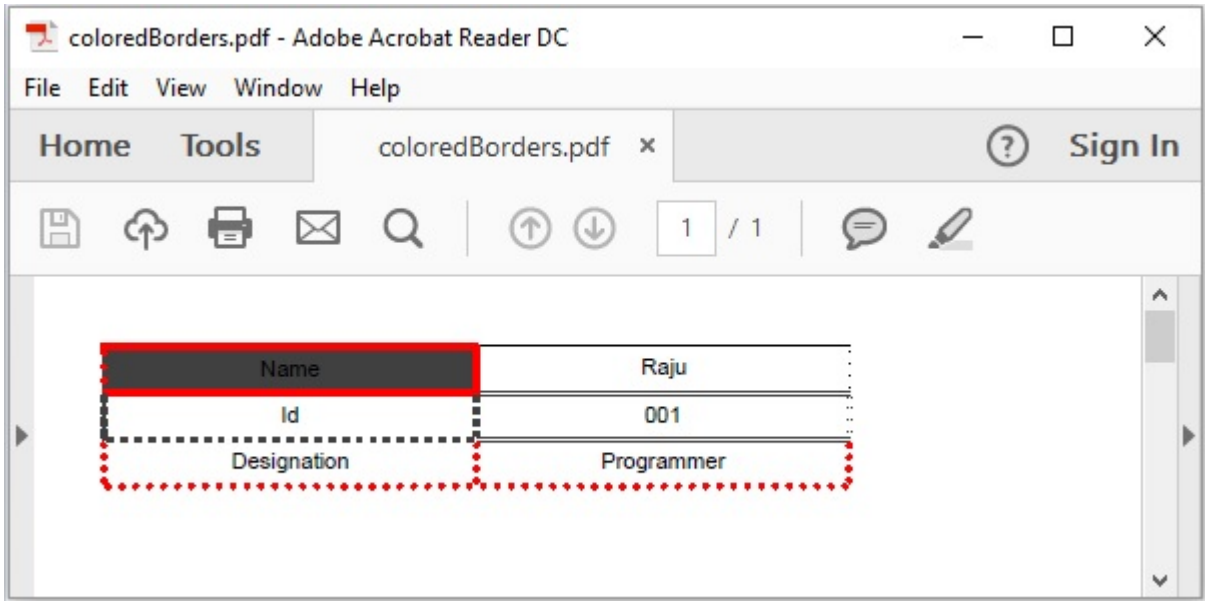
Compile and execute the saved Java file from the Command prompt using the following commands –

```
javac FormatedBorders.java
java FormatedBorders
```

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

```
Borders added successfully
```

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



iText - Adding Image to a Table

In this chapter, we will see how to add an image to a table in a PDF document using the iText library.

Adding an Image to a Table

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 table to the document, you need to instantiate the **Table** class and add this object to the document using the **add()** method.

To add an image to this table, you need to instantiate the **Cell** class, create and an object of the image that is required to be added, add the image to the **cell** object using the **add()** method of the **Cell** class.

Following are the steps to insert an image into the cell of a table.

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/addingImage.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: Creating the 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 a Table object

The **Table** class represents a two-dimensional grid filled with cells, ordered in rows and columns. It belongs to the package **com.itextpdf.layout.element**.

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

```
// Creating a table
float [] pointColumnWidths = {200F, 200F};
Table table = new Table(pointColumnWidths);
```

Step 5: Creating the cell

Create a **cell** object by instantiating the **Cell** class of the package **com.itextpdf.layout**, as shown below.

```
// Adding cell to the table
Cell cell = new Cell();  // Creating a cell
```

Step 6: Creating an Image

To create the **image** object, first of all, create an **ImageData** object using the **create()** method of the **ImageDataFactory** class. As a parameter of this method, pass a string

parameter representing the path of the image, as shown below.

```
// Creating an ImageData object
String imageFile = "C:/itextExamples/javafxLogo.jpg";
ImageData data = ImageDataFactory.create(imageFile);
```

Now, instantiate the **Image** class of the **com.itextpdf.layout.element** package. While instantiating, pass the **ImageData** object created above, as a parameter to its constructor, as shown below.

```
// Creating an Image object
Image img = new Image(data);
```

Add the **image** object to the cell using the **add()** method of the cell class, as shown below.

```
// Adding image to the cell
cell.add(img.setAutoScale(true));
```

Step 7: Adding cell to the table

Finally, to add this cell to the table, call the **addCell()** method of the **Table** class and pass the **cell** object as a parameter to this method, as shown below.

```
table.addCell(cell);
```

Step 8: Adding table to the document

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

```
// Adding list to the document
document.add(table);
```

Step 9: 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 add an image to a cell of a table in a PDF document using the iText library. It creates a PDF document with the name **addingImage.pdf**, adds a table to it, inserts an image (javafxLogo.jpg) to one of its cells, and saves it in the path **C:/itextExamples/**.

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

```
import com.itextpdf.io.image.ImageData;
import com.itextpdf.io.image.ImageDataFactory;

import com.itextpdf.kernel.pdf.PdfDocument;
import com.itextpdf.kernel.pdf.PdfWriter;

import com.itextpdf.layout.Document;
import com.itextpdf.layout.element.Cell;
import com.itextpdf.layout.element.Image;
import com.itextpdf.layout.element.Table;

public class a3AddingImageToTable {
   public static void main(String args[]) throws Exception {
      // Creating a PdfWriter object
      String dest = "C:/itextExamples/addingImage.pdf";
      PdfWriter writer = new PdfWriter(dest);

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

      // Creating a Document object
      Document doc = new Document(pdfDoc);

      // Creating a table
      float [] pointColumnWidths = {150f, 150f};
      Table table = new Table(pointColumnWidths);

      // Populating row 1 and adding it to the table
      Cell cell1 = new Cell();
      cell1.add("Tutorial ID");
      table.addCell(cell1);

      Cell cell2 = new Cell();
      cell2.add("1");
      table.addCell(cell2);

      // Populating row 2 and adding it to the table
      Cell cell3 = new Cell();
      cell3.add("Tutorial Title");
      table.addCell(cell3);

      Cell cell4 = new Cell();
      cell4.add("JavaFX");
      table.addCell(cell4);

      // Populating row 3 and adding it to the table
      Cell cell5 = new Cell();
      cell5.add("Tutorial Author");
      table.addCell(cell5);

      Cell cell6 = new Cell();
      cell6.add("Krishna Kasyap");
      table.addCell(cell6);

      // Populating row 4 and adding it to the table
      Cell cell7 = new Cell();
      cell7.add("Submission date");
      table.addCell(cell7);

      Cell cell8 = new Cell();
```

```
      cell8.add("2016-07-06");
      table.addCell(cell8);

      // Populating row 5 and adding it to the table
      Cell cell9 = new Cell();
      cell9.add("Tutorial Icon");
      table.addCell(cell9);

      // Creating the cell10
      Cell cell10 = new Cell();

      // Creating an ImageData object
      String imageFile = "C:/itextExamples/javafxLogo.jpg";
      ImageData data = ImageDataFactory.create(imageFile);

      // Creating the image
      Image img = new Image(data);

      // Adding image to the cell10
      cell10.add(img.setAutoScale(true));

      // Adding cell110 to the table
      table.addCell(cell10);

      // Adding Table to document
      doc.add(table);

      // Closing the document
      doc.close();

      System.out.println("Image added to table successfully..");
   }
}
```

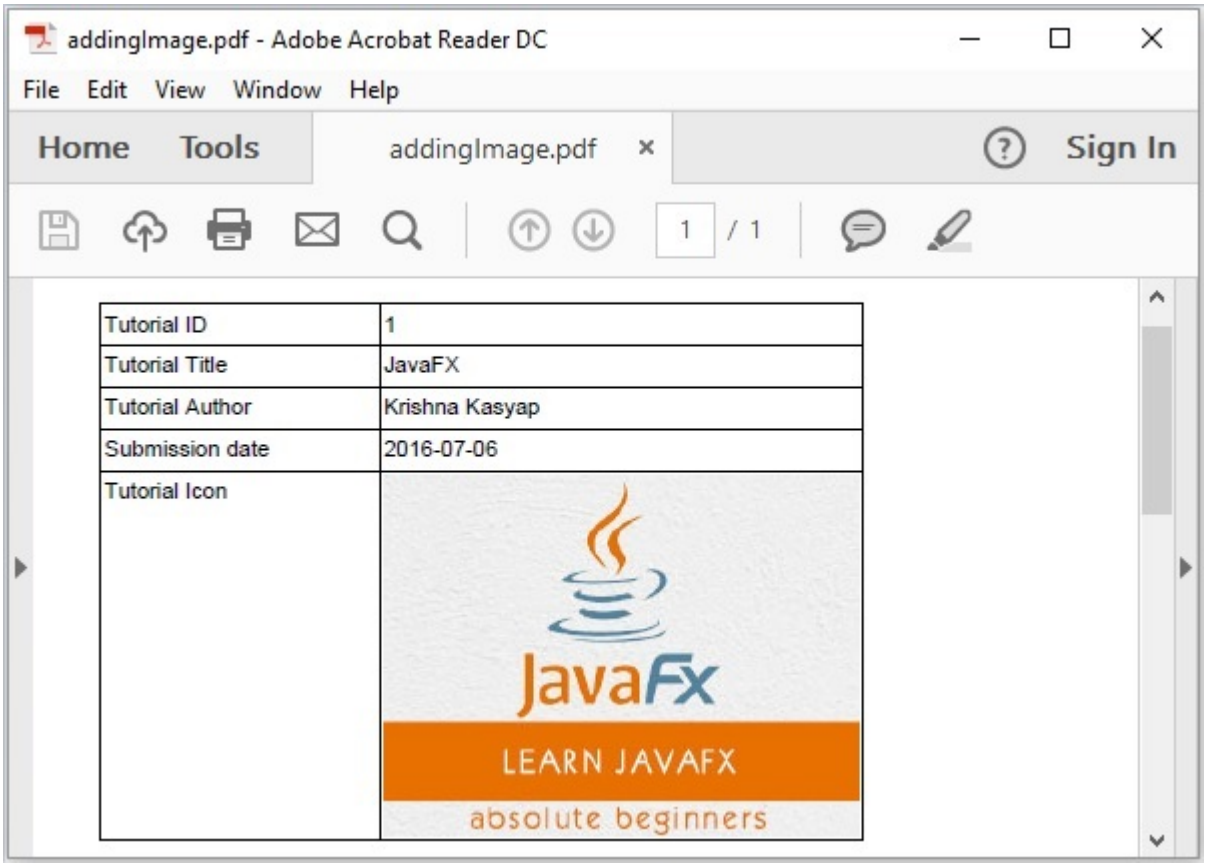
Compile and execute the saved Java file from the Command prompt using the following commands –

```
javac AddingImageToTable.java
java AddingImageToTable
```

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

```
Image added to table successfully..
```

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



iText - Nested Table

In this chapter, we will see how to add a nested table to a table in a PDF document using the iText library.

Adding Nested Tables in a Pdf

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 table to the document, you need to instantiate the **Table** class and add this object to the document using the **add()** method.

To add a table to this table, you need to create another table (nested table), and pass it to the cell object using the **add()** method of the **Cell** class.

Following are the steps to insert a table into the cell of a table.

Step 1: Creating a PdfWriter object

The **PdfWriter** class represents the DocWriter 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/addingNestedTable.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: Creating the 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 a Table object

The **Table** class represents a two-dimensional grid filled with cells, ordered in rows and columns. It belongs to the package **com.itextpdf.layout.element**.

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

```
// Creating a table
float [] pointColumnWidths = {200F, 200F};
Table table = new Table(pointColumnWidths);
```

Step 5: Creating the cell

Create a **cell** object by instantiating the **Cell** class of the package **com.itextpdf.layout**, as shown below.

```
// Adding cell to the table
Cell contact = new Cell();    // Creating a cell
```

Step 6: Creating Nested table

After creating the **cell**, create a nested table, and populate its cells, as shown below.

```
// Creating nested table for contact
float [] pointColumnWidths2 = {150f, 150f};
Table nestedTable = new Table(pointColumnWidths2);

// Populating row 1 and adding it to the nested table
Cell nested1 = new Cell();
nested1.add("Phone");
nestedTable.addCell(nested1);

Cell nested2 = new Cell();
nested2.add("9848022338");
nestedTable.addCell(nested2);

// Populating row 2 and adding it to the nested table
Cell nested3 = new Cell();
nested3.add("email");
nestedTable.addCell(nested3);

Cell nested4 = new Cell();
nested4.add("Raju123@gmail.com");
nestedTable.addCell(nested4);

// Populating row 3 and adding it to the nested table
Cell nested5 = new Cell();
nested5.add("Address");
nestedTable.addCell(nested5);

Cell nested6 = new Cell();
nested6.add("Hyderabad");
nestedTable.addCell(nested6);
```

Step 7: Adding Nested table to the cell

Now, add the above created nested table to the cell of the parent (container) table using the **add()** method of the **Cell** class. And, add this cell to the parent table using the **addCell()** method of the **Table** class, as shown below.

```
contact.add(nestedTable);
table.addCell(contact);
```

Step 8: Adding table to the document

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

```
// Adding list to the document
document.add(table);
```

Step 9: 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 add a table to a cell of a table (nested table) in a PDF document using the iText library. It creates a PDF document with the name **addingNestedTable.pdf**, adds a table to it, inserts another table to one of its cells, and saves it in the path **C:/itextExamples/**.

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

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

import com.itextpdf.layout.Document;
import com.itextpdf.layout.element.Cell;
import com.itextpdf.layout.element.Table;

public class a4AddNestedTablesPdf {
   public static void main(String args[]) throws Exception {
      // Creating a PdfWriter object
      String dest = "C:/itextExamples/addingNestedTable.pdf";
      PdfWriter writer = new PdfWriter(dest);

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

      // Creating a Document object
      Document doc = new Document(pdfDoc);

      // Creating a table
      float [] pointColumnWidths1 = {150f, 150f};
      Table table = new Table(pointColumnWidths1);

      // Populating row 1 and adding it to the table
      Cell cell1 = new Cell();
      cell1.add("Name");
      table.addCell(cell1);

      Cell cell2 = new Cell();
      cell2.add("Raju");
      table.addCell(cell2);

      // Populating row 2 and adding it to the table
      Cell cell3 = new Cell();
      cell3.add("Id");
      table.addCell(cell3);

      Cell cell4 = new Cell();
      cell4.add("1001");
      table.addCell(cell4);
```

```
        // Populating row 3 and adding it to the table
        Cell cell5 = new Cell();
        cell5.add("Designation");
        table.addCell(cell5);

        Cell cell6 = new Cell();
        cell6.add("Programmer");
        table.addCell(cell6);

        // Creating nested table for contact
        float [] pointColumnWidths2 = {150f, 150f};
        Table nestedTable = new Table(pointColumnWidths2);

        // Populating row 1 and adding it to the nested table
        Cell nested1 = new Cell();
        nested1.add("Phone");
        nestedTable.addCell(nested1);

        Cell nested2 = new Cell();
        nested2.add("9848022338");
        nestedTable.addCell(nested2);

        // Populating row 2 and adding it to the nested table
        Cell nested3 = new Cell();
        nested3.add("email");
        nestedTable.addCell(nested3);

        Cell nested4 = new Cell();
        nested4.add("Raju123@gmail.com");
        nestedTable.addCell(nested4);

        // Populating row 3 and adding it to the nested table
        Cell nested5 = new Cell();
        nested5.add("Address");
        nestedTable.addCell(nested5);

        Cell nested6 = new Cell();
        nested6.add("Hyderabad");
        nestedTable.addCell(nested6);

        // Adding table to the cell
        Cell cell7 = new Cell();
        cell7.add("Contact");
        table.addCell(cell7);

        Cell cell8 = new Cell();
        cell8.add(nestedTable);
        table.addCell(cell8);

        // Adding table to the document
        doc.add(table);

        // Closing the document
        doc.close();
        System.out.println("Nested Table Added successfully..");
    }
}
```

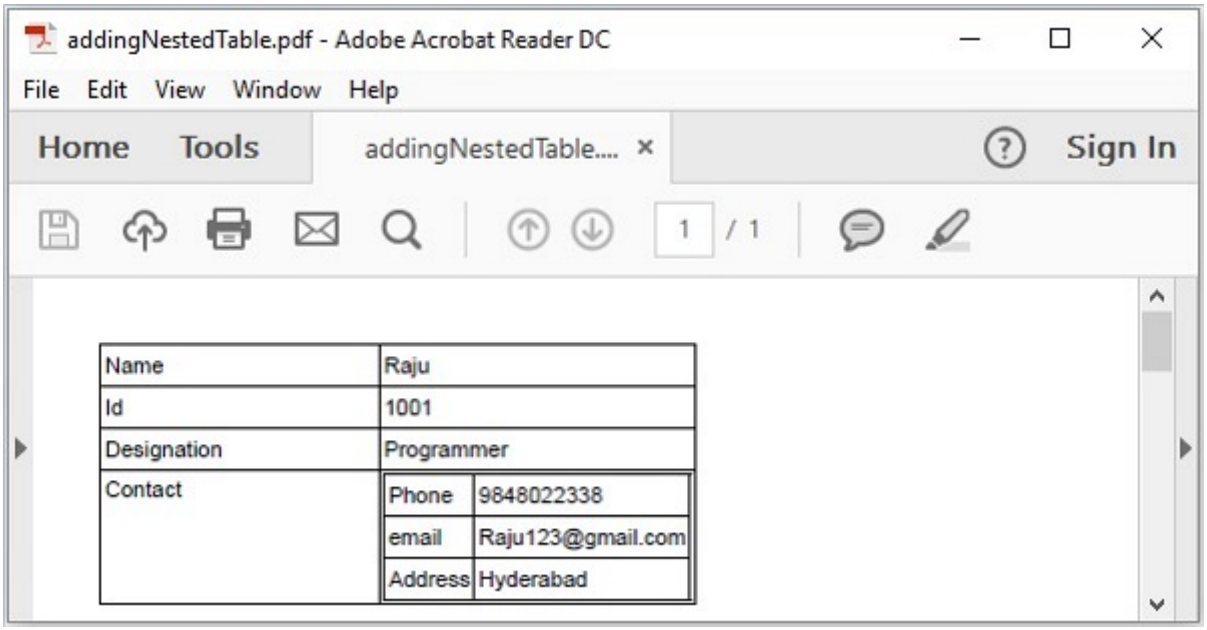
Compile and execute the saved Java file from the Command prompt using the following commands –

```
javac AddNestedTable.java
java AddNestedTable
```

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

```
Nested Table Added successfully..
```

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



iText - Adding Lists to a Table

In this chapter, we will see how to add a list to a table in a PDF document using the iText library.

Adding Lists to a Table in a PDF

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 table to the document, you need to instantiate the **Table** class and add this object to the document using the **add()** method.

To add a **list** to the table, you need to instantiate the **List** class of the **com.itextpdf.layout.element** package and insert it into the **cell** object using the **add()** method of the **Cell** class.

Following are the steps to add a list to the cell of a table.

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/addingObjects.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 c**om.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 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 a Table object

The **Table** class represents a two-dimensional grid filled with cells, ordered in rows and columns. It belongs to the package **com.itextpdf.layout.element**.

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

```
// Creating a table
float [] pointColumnWidths = {200F, 200F};
```

```
Table table = new Table(pointColumnWidths);
```

Step 5: Creating the cell

Create a **cell** object by instantiating the **Cell** class of the package **com.itextpdf.layout**, as shown below.

```
// Adding cell to the table
Cell listCell = new Cell(); // Creating a cell
```

Step 6: Creating List object

After creating the cell, create a **list** object by instantiating the **List** class of the package **com.itextpdf.layout.element**. Create the list items by instantiating the **ListItem** class and add the created items using the **add()** method of the **List** class, as shown below.

```
List list = new List();
ListItem item1 = new ListItem("JavaFX");
ListItem item2 = new ListItem("Java");
ListItem item3 = new ListItem("Java Servlets");
list.add(item1);
list.add(item2);
list.add(item3);
```

Step 7: Adding list to the cell of a table

Now, add the above created list to the cell of the table using the **add()** method of the Cell class. And, add this cell to the table using the **addCell()** method of the **Table** class, as shown below.

```
listCell.add(list);
table.addCell(listCell);
```

Step 8: Adding table to the document

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

```
// Adding list to the document
document.add(table);
```

Step 9: 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 add a list to a cell of a table in a PDF document using the iText library. It creates a PDF document with the name **addingObjects.pdf**, adds a table to it, inserts a list to one of its cells, and saves it in the path **C:/itextExamples/**

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

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

import com.itextpdf.layout.Document;
import com.itextpdf.layout.element.Cell;
import com.itextpdf.layout.element.List;
import com.itextpdf.layout.element.ListItem;
import com.itextpdf.layout.element.Table;
import com.itextpdf.layout.property.TextAlignment;

public class AddingListsToTable {
   public static void main(String args[]) throws Exception {
      // Creating a PdfWriter object
      String file = "C:/itextExamples/addingObjects.pdf";
      PdfDocument pdfDoc = new PdfDocument(new PdfWriter(file));

      // Creating a Document object
      Document doc = new Document(pdfDoc);

      // Creating a table
      float [] pointColumnWidths = {300F, 300F};
      Table table = new Table(pointColumnWidths);

      // Adding row 1 to the table
      Cell c1 = new Cell();
      c1.add("Java Related Tutorials");
      c1.setTextAlignment(TextAlignment.LEFT);
      table.addCell(c1);

      List list1 = new List();
      ListItem item1 = new ListItem("JavaFX");
      ListItem item2 = new ListItem("Java");
      ListItem item3 = new ListItem("Java Servlets");
      list1.add(item1);
      list1.add(item2);
      list1.add(item3);

      Cell c2 = new Cell();
      c2.add(list1);
      c2.setTextAlignment(TextAlignment.LEFT);
      table.addCell(c2);

      // Adding row 2 to the table
      Cell c3 = new Cell();
      c3.add("No SQL Databases");
      c3.setTextAlignment(TextAlignment.LEFT);
      table.addCell(c3);
```

```
        List list2 = new List();
        list2.add(new ListItem("HBase"));
        list2.add(new ListItem("Neo4j"));
        list2.add(new ListItem("MongoDB"));

        Cell c4 = new Cell();
        c4.add(list2);
        c4.setTextAlignment(TextAlignment.LEFT);
        table.addCell(c4);

        // Adding Table to document
        doc.add(table);

        // Closing the document
        doc.close();
        System.out.println("Lists added to table successfully..");
    }
}
```

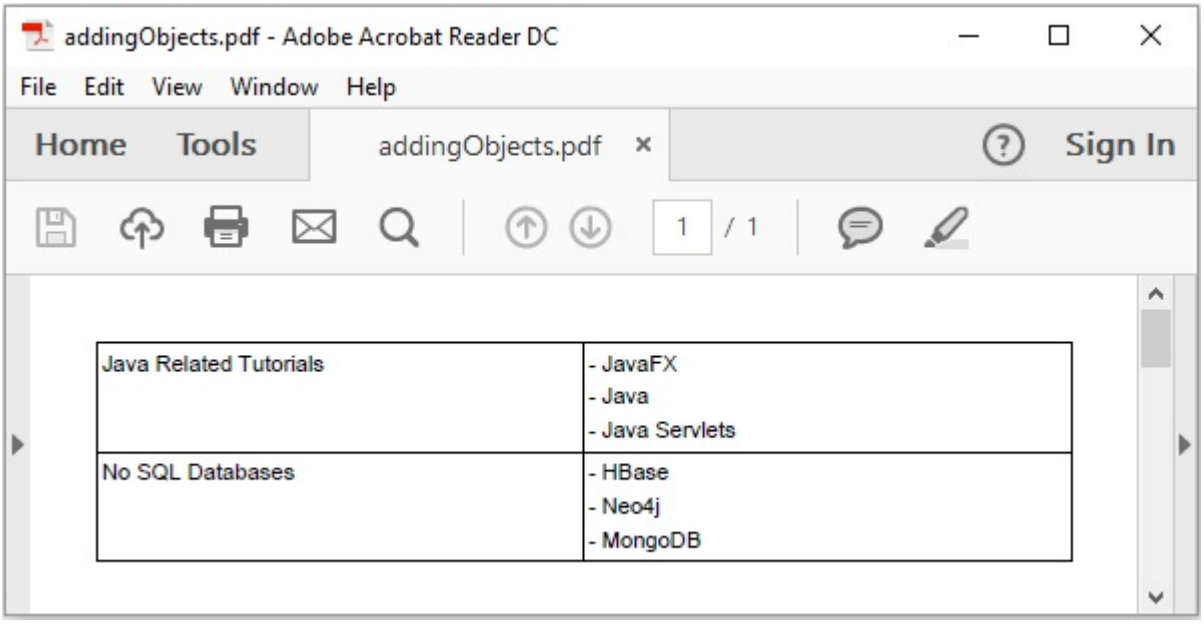
Compile and execute the saved Java file from the Command prompt using the following commands –

```
javac AddingListsToTable.java
java AddingListsToTable
```

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

```
Lists added to table successfully..
```

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



iText - Adding Image to a PDF

In this chapter, we will see how to add an image to a PDF document using the iText library.

Adding Image to a Pdf

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. To add image to the PDF, create an object of the image that is required to be added and add it using the **add()** method of the **Document** class.

Following are the steps to add an image to the PDF document.

Step 1: Creating a PdfWriter object

The **PdfWriter** class represents the DocWriter 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/addingImage.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: Creating the 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 Image object

To create the **image** object, first of all, create an **ImageData** object using the **create()** method of the **ImageDataFactory** class. As a parameter of this method, pass a string parameter representing the path of the image, as shown below.

```
// Creating an ImageData object
String imageFile = "C:/itextExamples/javafxLogo.jpg";
ImageData data = ImageDataFactory.create(imageFile);
```

Now, instantiate the **Image** class of the **com.itextpdf.layout.element** package. While instantiating, pass the above created **ImageData** object as a parameter to its constructor, as shown below.

```
// Creating an Image object
Image img = new Image(data);
```

Step 5: Adding image to the document

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

```
// Adding image to the document
document.add(img);
```

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 add an image to a PDF document using the iText library. It creates a PDF document with the name **addingImage.pdf**, adds an image to it, and saves it in the path **C:/itextExamples/**.

Save this code in a file with name **AddingImage.java**.

```
import com.itextpdf.io.image.ImageData;
import com.itextpdf.io.image.ImageDataFactory;

import com.itextpdf.kernel.pdf.PdfDocument;
import com.itextpdf.kernel.pdf.PdfWriter;

import com.itextpdf.layout.Document;
import com.itextpdf.layout.element.Image;

public class AddingImage {
```

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

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

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

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

      // Creating an ImageData object
      String imFile = "C:/itextExamples/logo.jpg";
      ImageData data = ImageDataFactory.create(imFile);

      // Creating an Image object
      Image image = new Image(data);

      // Adding image to the document
      document.add(image);

      // Closing the document
      document.close();

      System.out.println("Image added");
   }
}
```

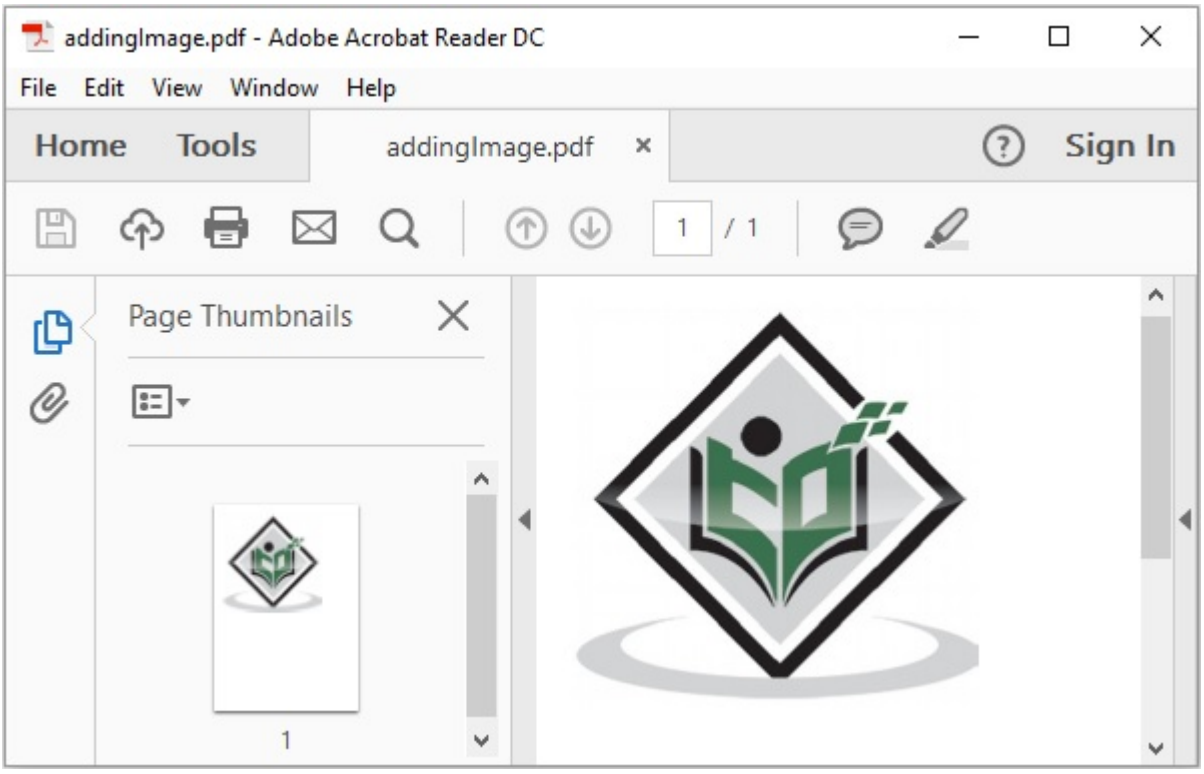
Compile and execute the saved Java file from the Command prompt using the following commands –

```
javac AddingImage.java
java AddingImage
```

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

```
Image added
```

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



iText - Setting Position of the Image

In this chapter, we will see how to set the position of an image in a PDF document using the iText library.

Setting the Position of the Image

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.

To add an image to the pdf, create and an object of the image that is required to be added and add it using the **add()** method of the **Document** class. You can insert the image in a desired position on the document using the method **setFixedPosition()** of the **Image** class.

Following are the steps to set the position of an image in the PDF document.

Step 1: Creating a PdfWriter object

The **PdfWriter** class represents the DocWriter 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/positionOfImage.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: Creating the 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 Image object

To create the image object, first of all, create an **ImageData** object using the **create()** method of the **ImageDataFactory** class. As a parameter of this method, pass a string parameter representing the path of the image, as shown below.

```
// Creating an ImageData object
String imageFile = "C:/itextExamples/javafxLogo.jpg";
ImageData data = ImageDataFactory.create(imageFile);
```

Now, instantiate the **Image** class of the **com.itextpdf.layout.element** package. While instantiating, pass the **ImageData** object as a parameter to its constructor, as shown below.

```
// Creating an Image object
Image img = new Image(data);
```

Step 5: Setting the position of the image

You can set the position of the image in a PDF document using the **setFixedPosition()** method of the Image. Set the position of the image to the coordinates (100, 250) on the document using this method, as shown below.

```
// Setting the position of the image to the center of the page
image.setFixedPosition(100, 250);
```

Step 6: Adding image to the document

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

```
// Adding image to the document
document.add(img);
```

Step 7: 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 set an image at a desired position on a PDF document using the iText library. It creates a PDF document with the name **positionOfImage.pdf**, adds an image to it, sets it nearer to the center of the page, and saves it in the path **C:/itextExamples/**

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

```
import com.itextpdf.io.image.ImageData;
import com.itextpdf.io.image.ImageDataFactory;

import com.itextpdf.kernel.pdf.PdfDocument;
import com.itextpdf.kernel.pdf.PdfWriter;

import com.itextpdf.layout.Document;
import com.itextpdf.layout.element.Image;

public class SettingPosition {
   public static void main(String args[]) throws Exception {
      // Creating a PdfWriter
      String dest = "C:/EXAMPLES/itextExamples/3images/positionOfImage.pdf";
      PdfWriter writer = new PdfWriter(dest);

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

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

      // Creating an ImageData object
      String imFile = "C:/EXAMPLES/itextExamples/3images/logo.jpg";
      ImageData data = ImageDataFactory.create(imFile);

      // Creating an Image object
      Image image = new Image(data);

      // Setting the position of the image to the center of the page
      image.setFixedPosition(100, 250);

      // Adding image to the document
      document.add(image);

      // Closing the document
      document.close();

      System.out.println("Image added");
   }
}
```

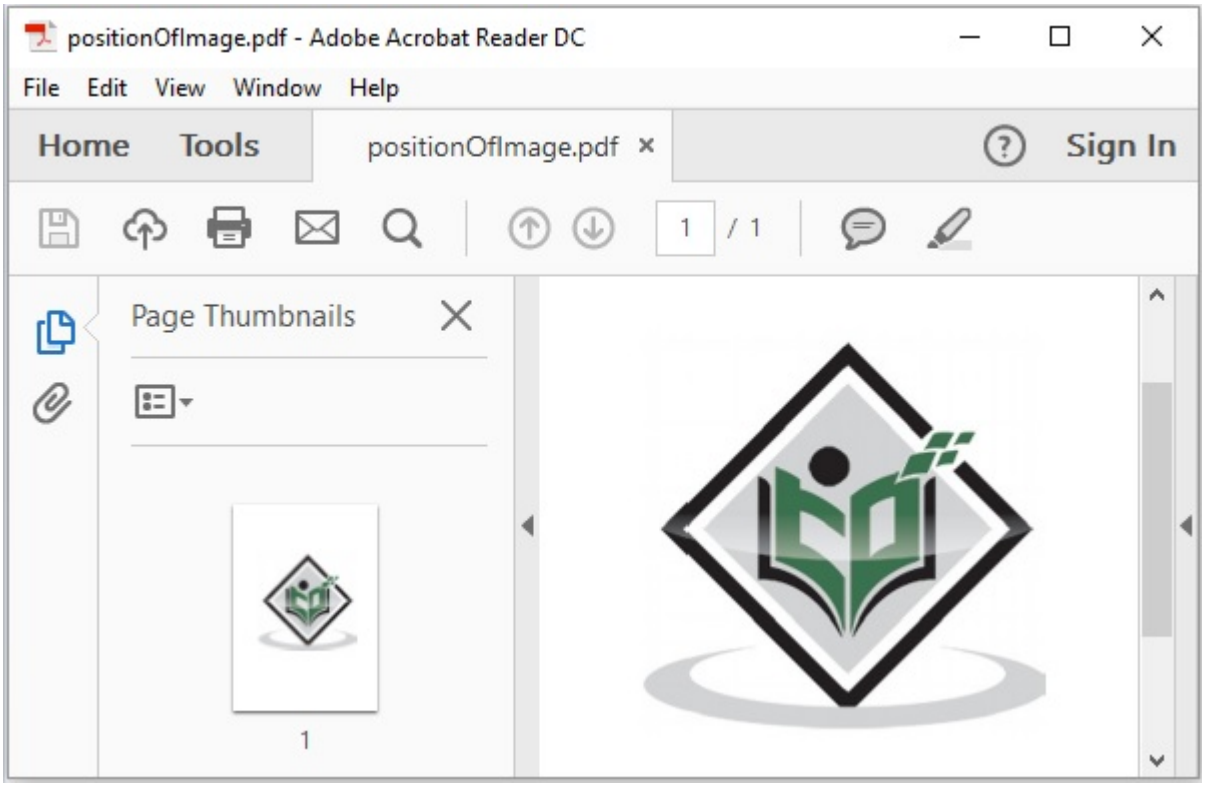
Compile and execute the saved Java file from the command prompt using the following commands.

```
javac SettingPosition.java
java SettingPosition
```

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

```
Image added
```

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



iText - Scaling an Image

In this chapter, we will see how to scale an image in a PDF document using the iText library.

Scaling an Image in a PDF

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.

To add image to the PDF, create an object of the image that is required to be added and add it using the **add()** method of the **Document** class. You can scale an image using the **setAutoScale()** method.

Following are the steps to scale an image that exists on the PDF document.

Step 1: Creating a PdfWriter object

The **PdfWriter** class represents the DocWriter 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/autoScale.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: Creating the 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 Image object

To create an image object, first of all, create an **ImageData** object using the **create()** method of the **ImageDataFactory** class. As a parameter of this method, pass a string parameter representing the path of the image, as shown below.

```
// Creating an ImageData object
String imageFile = "C:/itextExamples/javafxLogo.jpg";
ImageData data = ImageDataFactory.create(imageFile);
```

Now, instantiate the **Image** class of the **com.itextpdf.layout.element** package. While instantiating, pass the **ImageData** object as a parameter to its constructor, as shown below.

```
// Creating an Image object
Image img = new Image(data);
```

Step 5: Scaling an image

You can scale an image using the **setAutoScale()** method.

```
// Setting the position of the image to the center of the page
image.setFixedPosition(100, 250);
```

Step 6: Adding image to the document

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

```
// Adding image to the document
document.add(img);
```

Step 7: 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 scale an image with respective to the document size on a PDF document using the iText library. It creates a PDF document with the name **autoScale.pdf**, adds an image to it, scales it with respect to the page dimensions, saves it in the path **C:/itextExamples/**.

Save this code in a file with name **SettingAutoScale.java**.

```
import com.itextpdf.io.image.ImageData;
import com.itextpdf.io.image.ImageDataFactory;

import com.itextpdf.kernel.pdf.PdfDocument;
import com.itextpdf.kernel.pdf.PdfWriter;

import com.itextpdf.layout.Document;
import com.itextpdf.layout.element.Image;

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

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

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

      // Creating an ImageData object
      String imFile = "C:/itextExamples/logo.jpg";
      ImageData data = ImageDataFactory.create(imFile);
```

```
        // Creating an Image object
        Image image = new Image(data);

        // Setting the position of the image to the center of the page
        image.setFixedPosition(100,250);

        // Adding image to the document
        document.add(image);

        // Closing the document
        document.close();
        System.out.println("Image Scaled");
    }
}
```

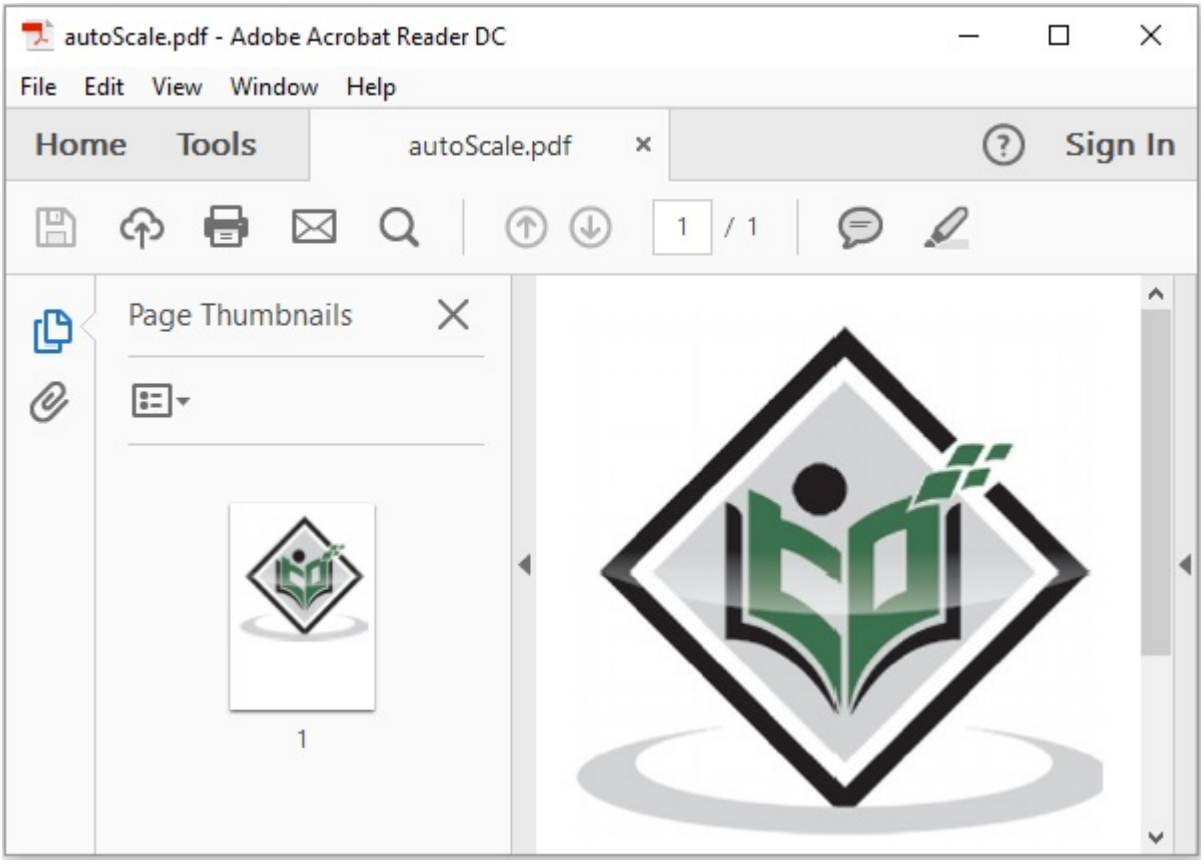
Compile and execute the saved Java file from the command prompt using the following commands.

```
javac SettingAutoScale.java
java SettingAutoScale
```

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

```
Image Scaled
```

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



In this chapter, we will see how to insert an image in a PDF document and how to rotate that image, using the functions available in the iText library.

Rotating an Image in a PDF

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.

To add image to the PDF, create an object of the image that is required to be added and add it using the **add()** method of the **Document** class. You can rotate an image using the **setRotationAngle()** method.

Following are the steps to rotate an image in a PDF document.

Step 1: Creating a PdfWriter object

The **PdfWriter** class represents the DocWriter 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/rotatingImage.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 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 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 Image object

To create an image object, first of all, create an **ImageData** object using the **create()**method of the **ImageDataFactory** class. As a parameter of this method, pass a string parameter representing the path of the image, as shown below.

```
// Creating an ImageData object
String imageFile = "C:/itextExamples/javafxLogo.jpg";
ImageData data = ImageDataFactory.create(imageFile);
```

Now, instantiate the **Image** class of the **com.itextpdf.layout.element** package. While instantiating, pass the **ImageData object**, as a parameter to its constructor, as shown below.

```
// Creating an Image object
Image img = new Image(data);
```

Step 5: rotating an image

You can rotate an image using the **setRotationAngle()** method. To this method, you need to pass an integer representing the rotation angle by which you want to rotate the image.

```
// Rotating the image
image.setRotationAngle(45);
```

Step 6: Adding image to the document

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

```
// Adding image to the document
document.add(img);
```

Step 7: 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 rotate an image by a given angle on a PDF document using the iText library.

It creates a PDF document with the name **rotatingImage.pdf**, adds an image to it, rotates it, and saves it in the path **C:/itextExamples/**.

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

```
import com.itextpdf.io.image.ImageData;
import com.itextpdf.io.image.ImageDataFactory;

import com.itextpdf.kernel.pdf.PdfDocument;
import com.itextpdf.kernel.pdf.PdfWriter;

import com.itextpdf.layout.Document;
import com.itextpdf.layout.element.Image;

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

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

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

      // Creating an ImageData object
      String imFile = "C:/itextExamples/logo.jpg";
      ImageData data = ImageDataFactory.create(imFile);

      // Creating an Image object
      Image image = new Image(data);

      // Rotating the image
      image.setRotationAngle(45);

      // Adding image to the document
      document.add(image);

      // Closing the document
      document.close();

      System.out.println("Image rotated");
   }
}
```

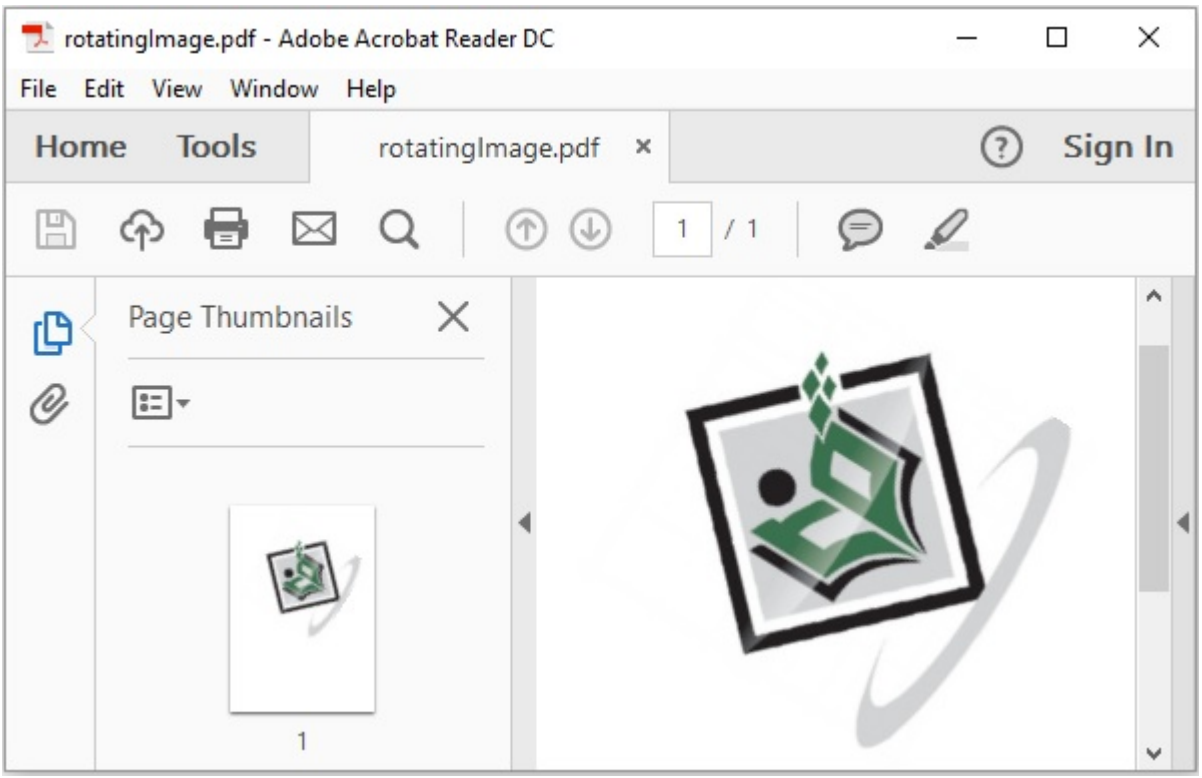
Compile and execute the saved Java file from the Command prompt using the following commands –

```
javac RotatingImage.java
java RotatingImage
```

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

```
Image Rotated
```

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



iText - Text Annotation

In this chapter, we will see how to add text annotation to a PDF document using iText library.

Creating a Text Annotation in a PDF

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.

To use text annotation in your PDF document, you need to create an object of **PdfTextAnnotation** class and add this to the **PdfPage**.

Following are the steps to use text annotation in the PDF document.

Step 1: Creating a PdfWriter object

The **PdfWriter** class represents the DocWriter 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/textAnnotation.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 **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 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 PdfAnnotation object

The **PdfAnnotation** class of the package **com.itextpdf.kernel.pdf.annot** represents the superclass of all the annotations.

Among its derived classes, **PdfTextAnnotation** class represents the text annotation. Create an object of this class as shown below.

```
// Creating PdfAnnotation
Rectangle rect = new Rectangle(20, 800, 0, 0);
PdfAnnotation ann = new PdfTextAnnotation(rect);
```

Step 5: Setting the color of the annotation

Set color to the annotation using the **setColor()** method of the **PdfAnnotation** class. To this method, pass the **color** object representing the color of the annotation as a

parameter.

```
// Setting color to the annotation
ann.setColor(Color.GREEN);
```

Step 6: Setting the title and contents of the annotation

Set the title and contents of the annotation using the **setTitle()** and **setContents()** methods of the **PdfAnnotation** class respectively, as shown below.

```
// Setting title to the annotation
ann.setTitle(new PdfString("Hello"));

// Setting contents of the annotation
ann.setContents("Hi welcome to Tutorialspoint.");
```

Step 7: Adding the annotation to a page

Create a new **PdfPage** class using the **addNewPage()** method of the PdfDocument class and add the above annotation using the **addAnnotation()** method of **PdfPage** class, as shown below.

```
// Creating a new page PdfPage page =
pdf.addNewPage();

// Adding annotation to a page in a PDF
page.addAnnotation(ann);
```

Step 8: 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 add text annotation to a PDF document using the iText library. It creates a PDF document with the name **textAnnotation.pdf**, adds a text annotation to it, and saves it in the path **C:/itextExamples/**

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

```
import com.itextpdf.kernel.color.Color;
import com.itextpdf.kernel.geom.Rectangle;
import com.itextpdf.kernel.pdf.PdfDocument;
import com.itextpdf.kernel.pdf.PdfPage;
import com.itextpdf.kernel.pdf.PdfString;
import com.itextpdf.kernel.pdf.PdfWriter;
```

```
import com.itextpdf.kernel.pdf.annot.PdfAnnotation;
import com.itextpdf.kernel.pdf.annot.PdfTextAnnotation;
import com.itextpdf.layout.Document;

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

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

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

      // Creating PdfTextAnnotation object
      Rectangle rect = new Rectangle(20, 800, 0, 0);
      PdfAnnotation ann = new PdfTextAnnotation(rect);

      // Setting color to the annotation
      ann.setColor(Color.GREEN);

      // Setting title to the annotation
      ann.setTitle(new PdfString("Hello"));

      // Setting contents of the annotation
      ann.setContents("Hi welcome to Tutorialspoint.");

      // Creating a new page
      PdfPage page =  pdf.addNewPage();

      // Adding annotation to a page in a PDF
      page.addAnnotation(ann);

      // Closing the document
      document.close();

      System.out.println("Annotation added successfully");
   }
}
```

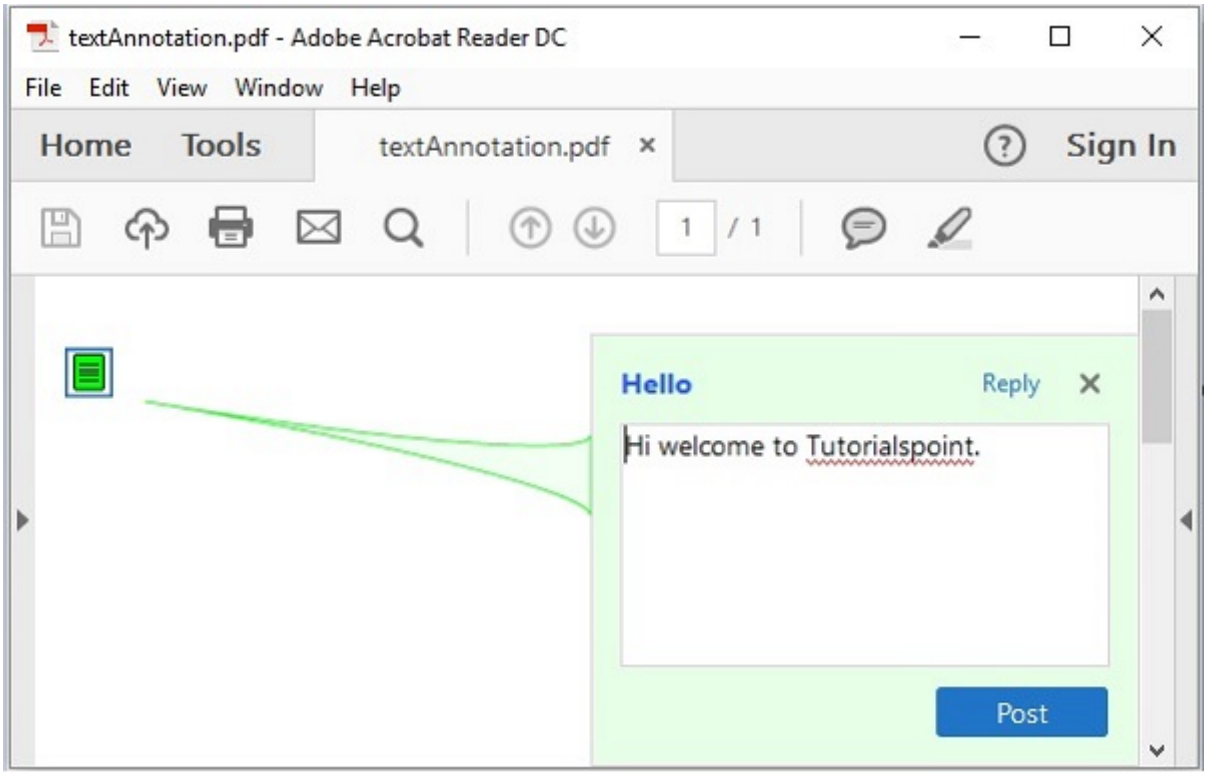
Compile and execute the saved Java file from the command prompt using the following commands.

```
javac TextAnnotation.java
java TextAnnotation
```

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

```
Annotation added successfully
```

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



iText - Link Annotation

In this chapter, we will see how to add link annotation to a PDF document using iText library.

Creating a Link Annotation in a PDF

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.

To use text annotation in your PDF document, you need to create an object of PdfTextAnnotation class and add this to the PdfPage.

Following are the steps to use text annotation in a PDF document.

Step 1 : Creating a PdfWriter object

The **PdfWriter** class represents the DocWriter 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/linkAnnotation.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 **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 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 PdfAnnotation object

The **PdfAnnotation** class of the package **com.itextpdf.kernel.pdf.annot** represents the superclass of all the annotations.

Among its derived classes, **PdfLinkAnnotation** class represents the link annotation. Create an object of this class, as shown below.

```
// Creating a PdfLinkAnnotation object
Rectangle rect = new Rectangle(0, 0);
PdfLinkAnnotation annotation = new PdfLinkAnnotation(rect);
```

Step 5: Setting the action of the annotation

Set action to the annotation using the **setAction()** method of the **PdfLinkAnnotation** class, as shown below.

```
// Setting action of the annotation
PdfAction action = PdfAction.createURI("http: // www.tutorialspoint.com/");
annotation.setAction(action);
```

Step 6: Creating a link

Create a link by instantiating the **Link** class of the package **com.itextpdf.layout.element**, as shown below.

```
// Creating a link
Link link = new Link("Click here", annotation);
```

Step 7: Adding the link annotation to a paragraph

Create a new paragraph by instantiating the **Paragraph** class and add the link created in the previous step using the **add()** method of this class, as shown below.

```
// Creating a paragraph
Paragraph paragraph = new Paragraph("Hi welcome to Tutorialspoint ");

// Adding link to paragraph
paragraph.add(link.setUnderline());
```

Step 8: Adding paragraph to the document

Add the paragraph to the document using the **add()** method of the **Document** class, as shown below.

```
// Adding paragraph to document
document.add(paragraph);
```

Step 9: 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 add link annotation to a PDF document using the iText library.

It creates a PDF document with the name **linkAnnotation.pdf**, adds a link annotation to it, and saves it in the path **C:/itextExamples/**

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

```
import com.itextpdf.kernel.geom.Rectangle;
import com.itextpdf.kernel.pdf.PdfDocument;
import com.itextpdf.kernel.pdf.PdfWriter;
import com.itextpdf.kernel.pdf.action.PdfAction;
import com.itextpdf.kernel.pdf.annot.PdfLinkAnnotation;
```

```
import com.itextpdf.layout.Document;
import com.itextpdf.layout.element.Link;
import com.itextpdf.layout.element.Paragraph;

public class LinkAnnotation {
   public static void main(String args[]) throws Exception {
      // Creating a PdfWriter
      String dest = "C:/itextExamples/linkAnnotation.pdf";

      PdfWriter writer = new
      PdfWriter(dest);

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

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

      // Creating a PdfLinkAnnotation object
      Rectangle rect = new Rectangle(0, 0);
      PdfLinkAnnotation annotation = new PdfLinkAnnotation(rect);

      // Setting action of the annotation
      PdfAction action = PdfAction.createURI("http:// www.tutorialspoint.com/");
      annotation.setAction(action);

      // Creating a link
      Link link = new Link("Click here", annotation);

      // Creating a paragraph
      Paragraph paragraph = new Paragraph("Hi welcome to Tutorialspoint ");

      // Adding link to paragraph
      paragraph.add(link.setUnderline());

      // Adding paragraph to document
      document.add(paragraph);

      // Closing the document
      document.close();

      System.out.println("Annotation added successfully");
   }
}
```

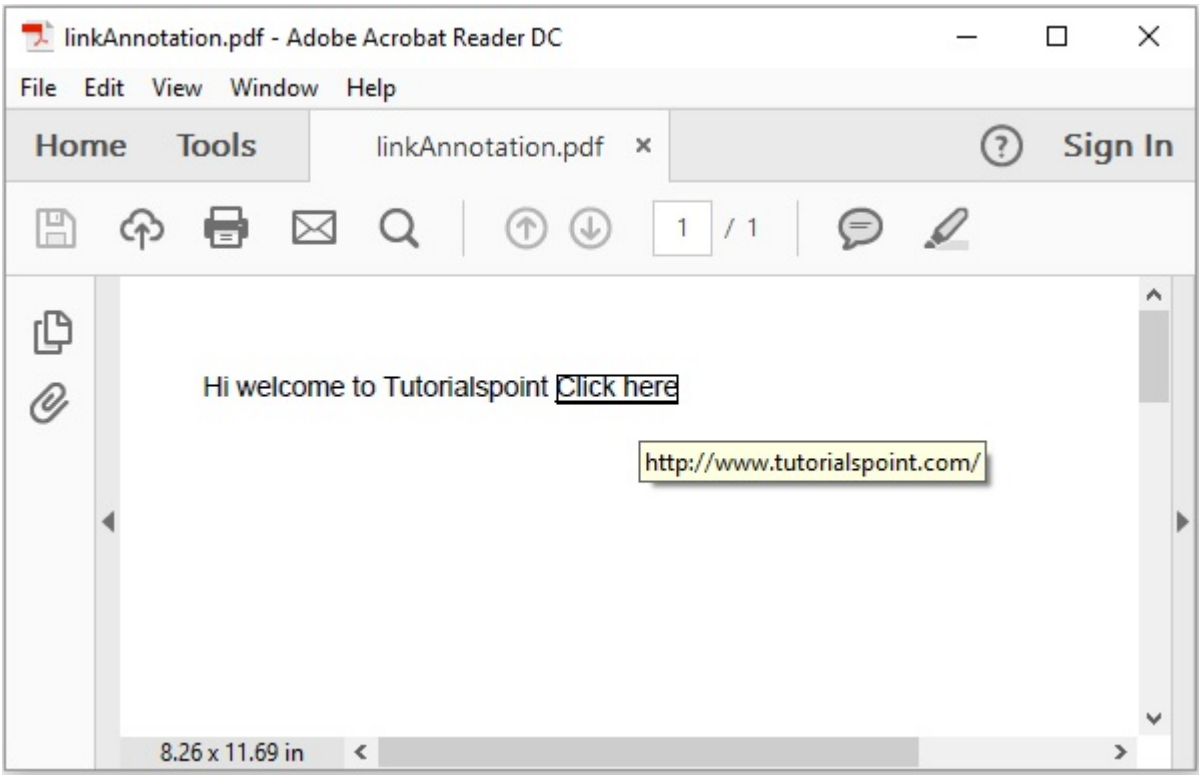
Compile and execute the saved Java file from the Command prompt using the following commands –

```
javac LinkAnnotation.java
java LinkAnnotation
```

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

```
Annotation added successfully
```

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



iText - Line Annotation

In this chapter, we will see how to add line annotation to a PDF document using iText library.

Creating a Line Annotation in a Pdf

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.

To use text annotation in your PDF document, you need to create an object of **PdfTextAnnotation** class and add this to the **PdfPage**.

Following are the steps to use text annotation in the PDF document.

Step 1: Creating a PdfWriter object

The **PdfWriter** class represents the DocWriter 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/lineAnnotation.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 PDFDocument 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 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 the 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 PdfAnnotation object

The **PdfAnnotation** class of the package **com.itextpdf.kernel.pdf.annot** represents is the superclass of all the annotations.

Among its derived classes, **PdfLineAnnotation** class represents the line annotation. Create an object of this class as shown below.

```
// Creating PdfAnnotation
Rectangle rect = new Rectangle(20, 800, 0, 0);
PdfAnnotation annotation = new PdfLineAnnotation(rect);
```

Step 5: Setting the color of the annotation

Set color to the annotation using the **setColor()** method of the **PdfAnnotation** class. To this method, pass the color object representing the color of the annotation as a parameter.

```
// Setting color to the annotation
annotation.setColor(Color.BLUE);
```

Step 6: Setting the title and contents of the annotation

Set the title and contents of the annotation using the **setTitle()** and **setContents()** methods of the **PdfAnnotation** class respectively, as shown below.

```
// Setting title to the PdfLineAnnotation
annotation.setTitle(new PdfString("iText"));

// Setting contents of the PdfLineAnnotation
annotation.setContents("Hi welcome to Tutorialspoint");
```

Step 7: Adding the annotation to a page

Create a new **PdfPage** class using the **addNewPage()** method of the **PdfDocument** class and add the above created annotation using the addAnnotation() method of PdfPage class, as shown below.

```
// Creating a new page
PdfPage page = pdf.addNewPage();

// Adding annotation to a page in a PDF
page.addAnnotation(annotation);
```

Step 8: 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 add line annotation to a PDF document using the iText library. It creates a PDF document with the name **lineAnnotation.pdf**, adds a line annotation to it, and saves it in the path **C:/itextExamples/**.

Save this code in a file with name **LineAnnotation.java**.

```
import com.itextpdf.kernel.color.Color;
import com.itextpdf.kernel.geom.Rectangle;
import com.itextpdf.kernel.pdf.PdfDocument;
import com.itextpdf.kernel.pdf.PdfPage;
import com.itextpdf.kernel.pdf.PdfString;
import com.itextpdf.kernel.pdf.PdfWriter;
import com.itextpdf.kernel.pdf.annot.PdfAnnotation;
import com.itextpdf.kernel.pdf.annot.PdfLineAnnotation;
import com.itextpdf.layout.Document;

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

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

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

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

      // Creating a PdfPage
      PdfPage page = pdf.addNewPage();

      // creating PdfLineAnnotation object
      Rectangle rect = new Rectangle(0, 0);
      float[] floatArray  = new float[]{
         20, 790, page.getPageSize().getWidth() - 20, 790
      };
      PdfAnnotation annotation = new PdfLineAnnotation(rect, floatArray);

      // Setting color of the PdfLineAnnotation
      annotation.setColor(Color.BLUE);

      // Setting title to the PdfLineAnnotation
      annotation.setTitle(new PdfString("iText"));

      // Setting contents of the PdfLineAnnotation
      annotation.setContents("Hi welcome to Tutorialspoint");

      // Adding annotation to the page
      page.addAnnotation(annotation);

      // Closing the document
      document.close();

      System.out.println("Annotation added successfully");
   }
}
```

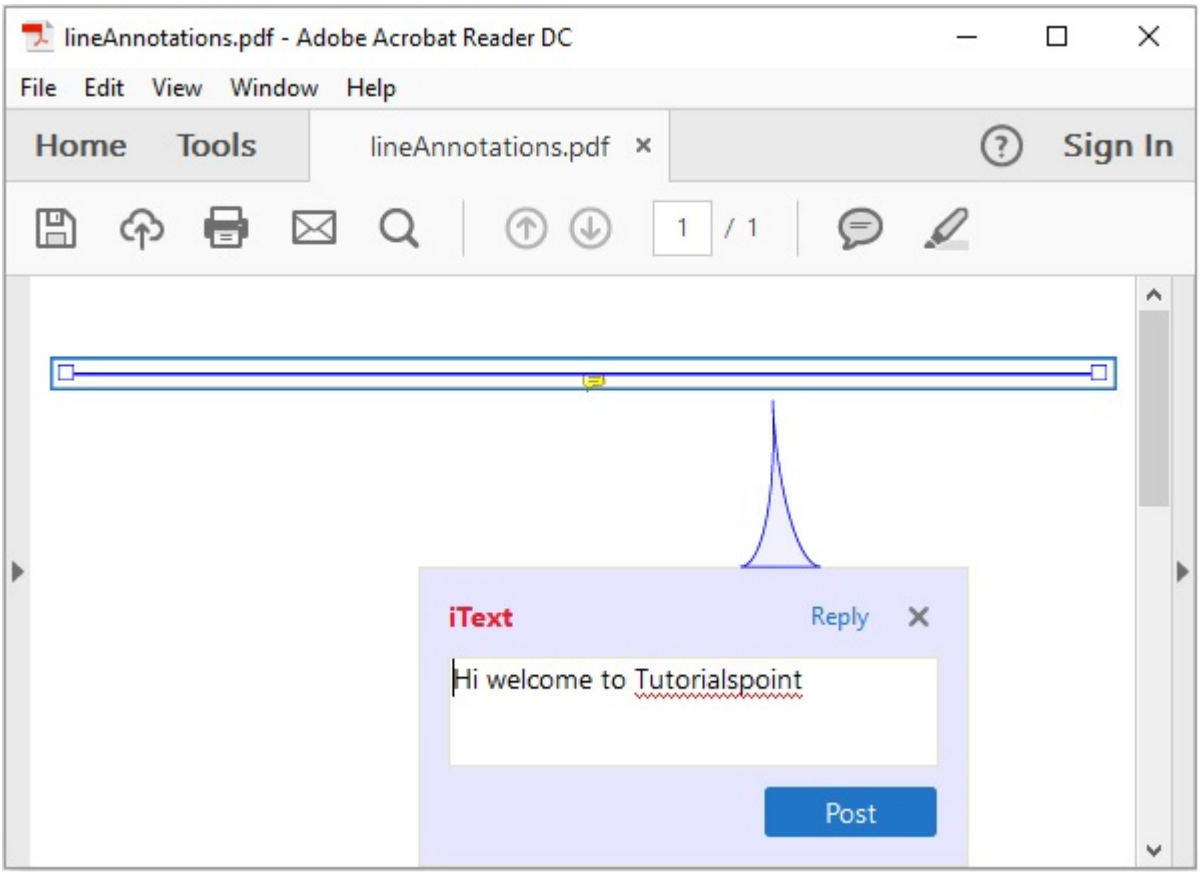
Compile and execute the saved Java file from the command prompt using the following commands –

```
javac LineAnnotation.java
java LineAnnotation
```

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

```
Annotation added successfully
```

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



iText - Markup Annotation

In this chapter, we will see how to add text markup annotation to a PDF document using iText library.

Creating a Markup Annotation in a PDF

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. To use text annotation in your PDF document, you need to create an object of **PdfTextAnnotation** class and add this to the **PdfPage**.

Following are the steps to use text annotation in the PDF document.

Step 1: Creating a PdfWriter object

The **PdfWriter** class represents the DocWriter 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/markupAnnotation.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 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 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 PdfAnnotation object

The **PdfAnnotation** class of the package **com.itextpdf.kernel.pdf.annot** represents the superclass of all the annotations.

Among its derived classes, **PdfTextMarkupAnnotation** class represents the text markup annotation. Create an object of this class as shown below.

```
// Creating a PdfTextMarkupAnnotation object
Rectangle rect = new Rectangle(105, 790, 64, 10);
float[] floatArray = new float[]{169, 790, 105, 790, 169, 800, 105, 800};
PdfAnnotation annotation = PdfTextMarkupAnnotation.createHighLight(rect,floatArray);
```

Step 5: Setting the color of the annotation

Set color to the annotation using the **setColor()** method of the **PdfAnnotation** class. To this method, pass the color object representing the **color** of the annotation as a parameter.

```
// Setting color to the annotation
annotation.setColor(Color.YELLOW);
```

Step 6: Setting the title and contents of the annotation

Set the title and contents of the annotation using the **setTitle()** and **setContents()** methods of the **PdfAnnotation** class respectively.

```
// Setting title to the annotation
annotation.setTitle(new PdfString("Hello!"));

// Setting contents to the annotation
annotation.setContents(new PdfString("Hi welcome to Tutorialspoint"));
```

Step 7: Adding the annotation to a page

Create a new **PdfPage** class using the **addNewPage()** method of the **PdfDocument** class and add the above created annotation using the **addAnnotation()** method of PdfPage class, as shown below.

```
// Creating a new Pdfpage
PdfPage pdfPage = pdfDoc.addNewPage();

// Adding annotation to a page in a PDF
pdfPage.addAnnotation(annotation);
```

Step 8: 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 add text markup annotation to a PDF document using the iText library. It creates a PDF document with the name **markupAnnotation.pdf**, adds a text markup annotation to it, and saves it in the path **C:/itextExamples/**

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

```
import com.itextpdf.kernel.color.Color;
import com.itextpdf.kernel.geom.Rectangle;
import com.itextpdf.kernel.pdf.PdfDocument;
import com.itextpdf.kernel.pdf.PdfPage;
import com.itextpdf.kernel.pdf.PdfString;
import com.itextpdf.kernel.pdf.PdfWriter;
import com.itextpdf.kernel.pdf.annot.PdfAnnotation;
```

```
import com.itextpdf.kernel.pdf.annot.PdfTextMarkupAnnotation;
import com.itextpdf.layout.Document;

public class MarkupAnnotation {
   public static void main(String args[]) throws Exception {
      // Creating a PdfDocument object
      String file = "C:/itextExamples/markupAnnotation.pdf";
      PdfDocument pdfDoc = new PdfDocument(new PdfWriter(file));

      // Creating a Document object
      Document doc = new Document(pdfDoc);

      // Creating a PdfTextMarkupAnnotation object
      Rectangle rect = new Rectangle(105, 790, 64, 10);
      float[] floatArray = new float[]{169, 790, 105, 790, 169, 800, 105, 800};
      PdfAnnotation annotation =
         PdfTextMarkupAnnotation.createHighLight(rect,floatArray);

      // Setting color to the annotation
      annotation.setColor(Color.YELLOW);

      // Setting title to the annotation
      annotation.setTitle(new PdfString("Hello!"));

      // Setting contents to the annotation
      annotation.setContents(new PdfString("Hi welcome to Tutorialspoint"));

      // Creating a new Pdfpage
      PdfPage pdfPage = pdfDoc.addNewPage();

      // Adding annotation to a page in a PDF
      pdfPage.addAnnotation(annotation);

      // Closing the document
      doc.close();

      System.out.println("Annotation added successfully");
   }
}
```

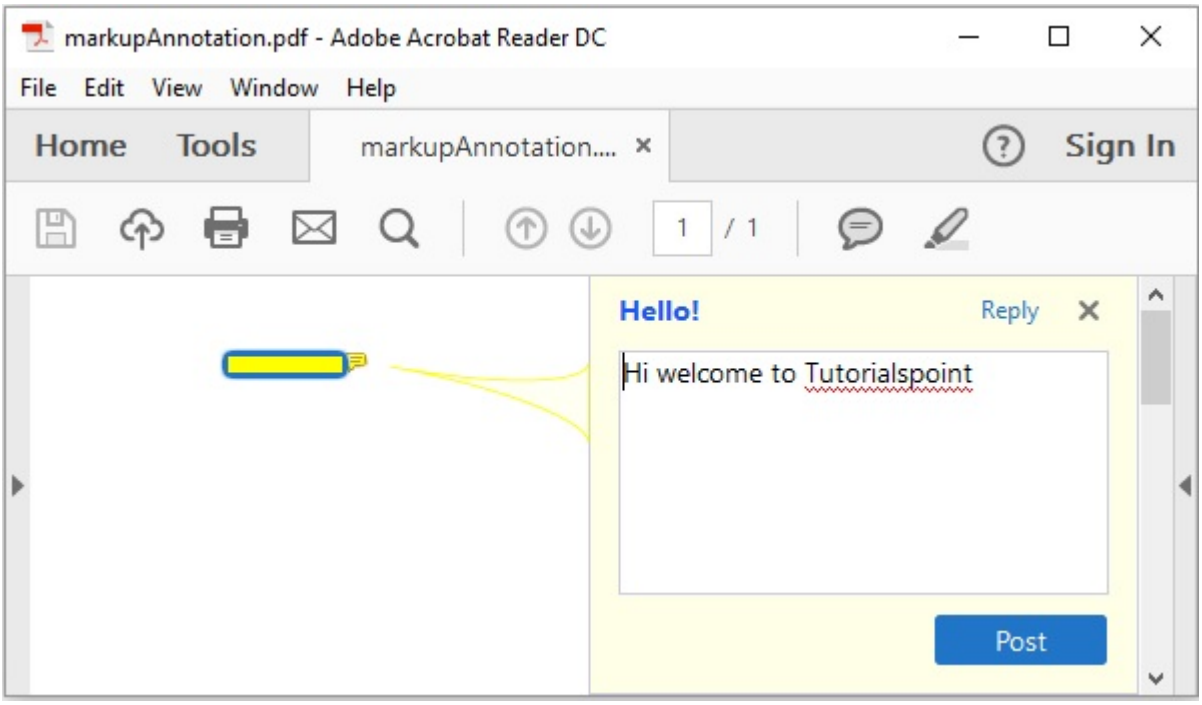
Compile and execute the saved Java file from the Command prompt using the following commands –

```
javac MarkupAnnotation.java
java MarkupAnnotation
```

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

```
Annotation added successfully
```

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



iText - Circle Annotation

In this chapter, we will see how to add circle annotation to a PDF document using iText library.

Creating a Circle Annotation in a PDF

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.

To use text annotation in your PDF document, you need to create an object of PdfTextAnnotation class and add this to the **Pdfpage**.

Following are the steps to use text annotation in the PDF document.

Step 1: Creating a PdfWriter object

The **PdfWriter** class represents the DocWriter 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/circleAnnotation.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 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 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 PdfAnnotation object

The **PdfAnnotation** class of the package **com.itextpdf.kernel.pdf.annot** represents the superclass of all the annotations.

Among its derived classes, **PdfCircleAnnotation** class represents the circle annotation. Create an object of this class as shown below.

```
// Creating a PdfCircleAnnotation object Rectangle
rect = new Rectangle(150, 770, 50, 50);
PdfAnnotation annotation = new PdfCircleAnnotation(rect);
```

Step 5: Setting the color of the annotation

Set color to the annotation using the **setColor()** method of the **PdfAnnotation** class. To this method, pass the color object representing the color of the annotation as a parameter.

```
// Setting color to the annotation
annotation.setColor(Color.YELLOW);
```

Step 6: Setting the title and contents of the annotation

Set the title and contents of the annotation using the **setTitle()** and **setContents()** methods of the **PdfAnnotation** class respectively.

```
// Setting title to the annotation
annotation.setTitle(new PdfString("circle annotation"));

// Setting contents of the annotation
annotation.setContents(new PdfString("Hi welcome to Tutorialspoint"));
```

Step 7: Adding the annotation to a page

Create a new **PdfPage** class using the **addNewPage()** method of the **PdfDocument** class and add the above created annotation using the **addAnnotation()** method of PdfPage class, as shown below.

```
// Creating a new page
PdfPage page = pdf.addNewPage();

// Adding annotation to a page in a PDF
page.addAnnotation(ann);
```

Step 8: 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 add circle annotation to a PDF document using the iText library. It creates a PDF document with the name **circleAnnotation.pdf**, adds a circle annotation to it, and saves it in the path **C:/itextExamples/**

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

```
import com.itextpdf.kernel.color.Color;
import com.itextpdf.kernel.geom.Rectangle;
import com.itextpdf.kernel.pdf.PdfDocument;
import com.itextpdf.kernel.pdf.PdfPage;
import com.itextpdf.kernel.pdf.PdfString;
import com.itextpdf.kernel.pdf.PdfWriter;
import com.itextpdf.kernel.pdf.annot.PdfAnnotation;
import com.itextpdf.kernel.pdf.annot.PdfCircleAnnotation;
import com.itextpdf.layout.Document;

public class CircleAnnotation {
   public static void main(String args[]) throws Exception {
      // Creating a PdfDocument object
      String file = "C:/itextExamples// circleAnnotation.pdf";
```

```
      PdfDocument pdf = new PdfDocument(new PdfWriter(file));

      // Creating a Document object
      Document doc = new Document(pdf);

      // Creating a PdfCircleAnnotation object
      Rectangle rect = new Rectangle(150, 770, 50, 50);
      PdfAnnotation annotation = new PdfCircleAnnotation(rect);

      // Setting color to the annotation
      annotation.setColor(Color.YELLOW);

      // Setting title to the annotation
      annotation.setTitle(new PdfString("circle annotation"));

      // Setting contents of the annotation
      annotation.setContents(new PdfString("Hi welcome to Tutorialspoint"));

      // Creating a new page
      PdfPage page = pdf.addNewPage();

      // Adding annotation to a page in a PDF
      page.addAnnotation(annotation);

      // Closing the document
      doc.close();

      System.out.println("Annotation added successfully");
   }
}
```

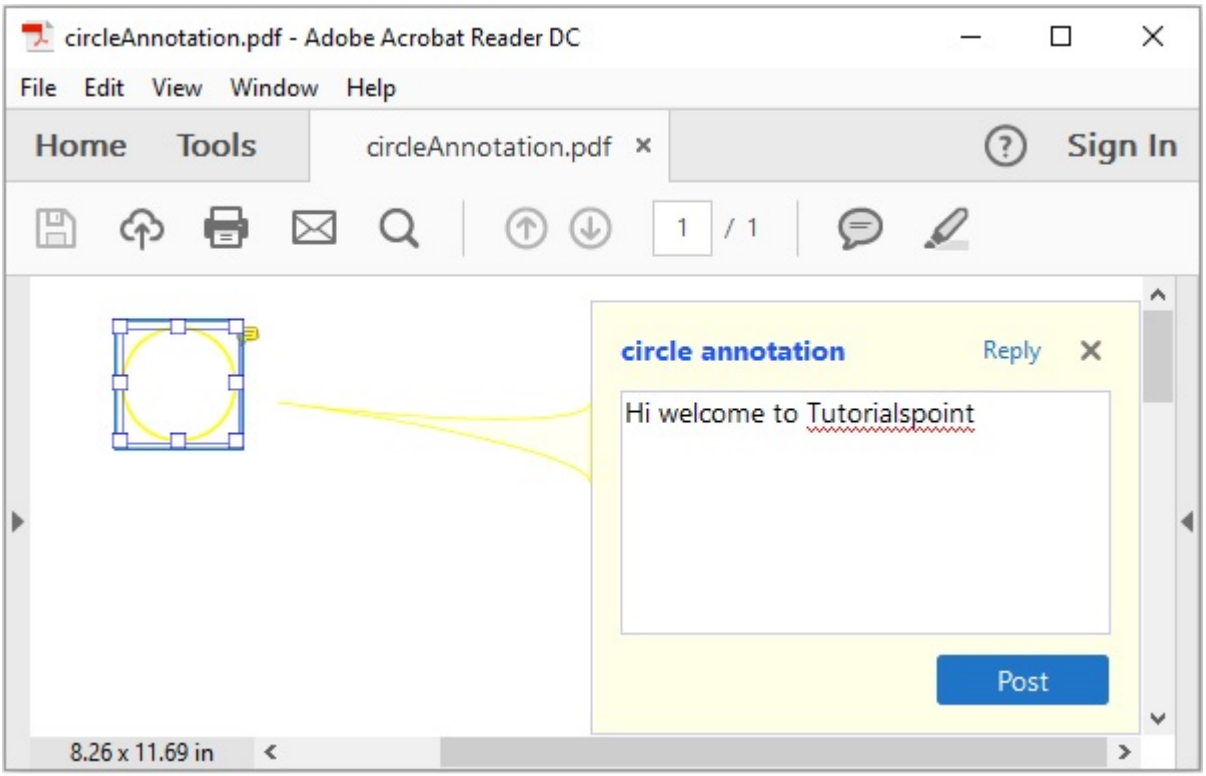
Compile and execute the saved Java file from the Command prompt using the following commands –

```
javac PdfCircleAnnotation.java
java PdfCircleAnnotation
```

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

```
Annotation added successfully
```

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



iText - Drawing an Arc

In this chapter, we will see how to draw an arc on a PDF document using iText library.

Drawing an Arc on a PDF

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.

To draw an arc on a PdfDocument, instantiate the **PdfCanvas** class of the package **com.itextpdf.kernel.pdf**.canvas and create an arc using the **arc()** method of this class.

Following are the steps to draw an arc on a PDF document.

Step 1: Creating a PdfWriter object

The **PdfWriter** class represents the DocWriter 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/drawingArc.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 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 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 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 a PdfCanvas object

Create a new **PdfPage** class using the **addNewPage()** method of the **PdfDocument** class.

Instantiate the **PdfCanvas** object of the package **com.itextpdf.kernel.pdf.canvas** by passing the above created **PdfPage** object to the constructor of this class, as shown below.

```
// Creating a new page
PdfPage pdfPage = pdfDoc.addNewPage();

// Creating a PdfCanvas object
PdfCanvas canvas = new PdfCanvas(pdfPage);
```

Step 5: Drawing the arc

Draw the arc using the **arc()** method of the **Canvas** class and fill it using the **fill()** method, as shown below.

```
// Drawing an arc
canvas.arc(50, 50, 300, 545, 0, 360);

// Filling the arc
canvas.fill();
```

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 draw an arc in a PDF document using the iText library.

It creates a PDF document with the name **drawingArc.pdf**, draws an arc in it, and saves it in the path **C:/itextExamples/**

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

```
import com.itextpdf.kernel.color.Color;
import com.itextpdf.kernel.pdf.PdfDocument;
import com.itextpdf.kernel.pdf.PdfPage;
import com.itextpdf.kernel.pdf.PdfWriter;
import com.itextpdf.kernel.pdf.canvas.PdfCanvas;
import com.itextpdf.layout.Document;

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

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

      // Creating a Document object
      Document doc = new Document(pdfDoc);

      // Creating a new page
      PdfPage pdfPage = pdfDoc.addNewPage();

      // Creating a PdfCanvas object
      PdfCanvas canvas = new PdfCanvas(pdfPage);

      // Drawing an arc
      canvas.arc(50, 50, 300, 545, 0, 360);

      // Filling the arc
      canvas.fill();
```

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

        System.out.println("Object drawn on pdf successfully");
    }
}
```

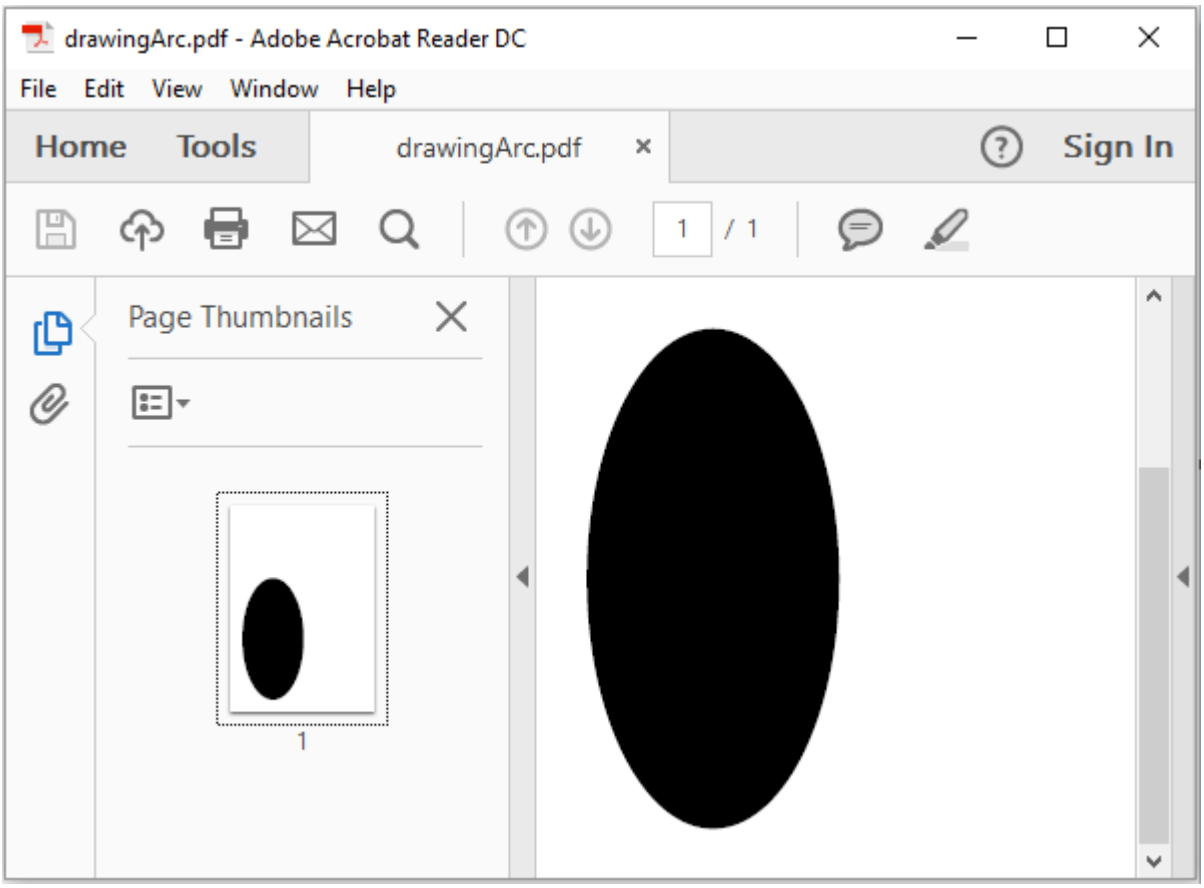
Compile and execute the saved Java file from the Command prompt using the following commands –

```
javac DrawingArc.java
java DrawingArc
```

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

```
Object drawn on pdf successfully
```

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



iText - Drawing a Line

In this chapter, we will see how to draw a line on a PDF document using iText library.

Drawing a Line on a PDF

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.

To draw a line on a PdfDocument Instantiate the **PdfCanvas** class of the package **com.itextpdf.kernel.pdf.canvas** and create a line using the **moveTo()** and **lineTO()** methods of this class.

Following are the steps to draw a line on the pdf document.

Step 1: Creating a PdfWriter object

The **PdfWriter** class represents the DocWriter 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/drawingLine.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 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 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 a PdfCanvas object

Create a new **PdfPage** class using the **addNewPage()** method of the **PdfDocument** class.

Instantiate the **PdfCanvas** object of the package **com.itextpdf.kernel.pdf.canvas** by passing the above created **PdfPage** object to the constructor of this class, as shown below.

```
// Creating a new page
PdfPage pdfPage = pdfDoc.addNewPage();

// Creating a PdfCanvas object
PdfCanvas canvas = new PdfCanvas(pdfPage);
```

Step 5: Drawing the line

Set the initial point of the line using the **moveTO()** method of the **Canvas** class, as shown below.

```
// Initial point of the line
canvas.moveTo(100, 300);
```

Now, draw a line from this point to another point using the **lineTo()** method, as shown below.

```
// Drawing the line
canvas.lineTo(500, 300);
```

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 draw a line in a PDF document using the iText library. It creates a PDF document with the name **drawingLine.pdf**, draws an arc in it, and saves it in the path **C:/itextExamples/**

Save this code in a file with name **DrawingLine.java**.

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

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

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

      // Creating a Document object
      Document doc = new Document(pdfDoc);

      // Creating a new page
      PdfPage pdfPage = pdfDoc.addNewPage();

      // Creating a PdfCanvas object
      PdfCanvas canvas = new PdfCanvas(pdfPage);

      // Initial point of the line
      canvas.moveTo(100, 300);

      // Drawing the line
      canvas.lineTo(500, 300);

      // Closing the path stroke
      canvas.closePathStroke();

      // Closing the document
      doc.close();

      System.out.println("Object drawn on pdf successfully");
   }
}
```

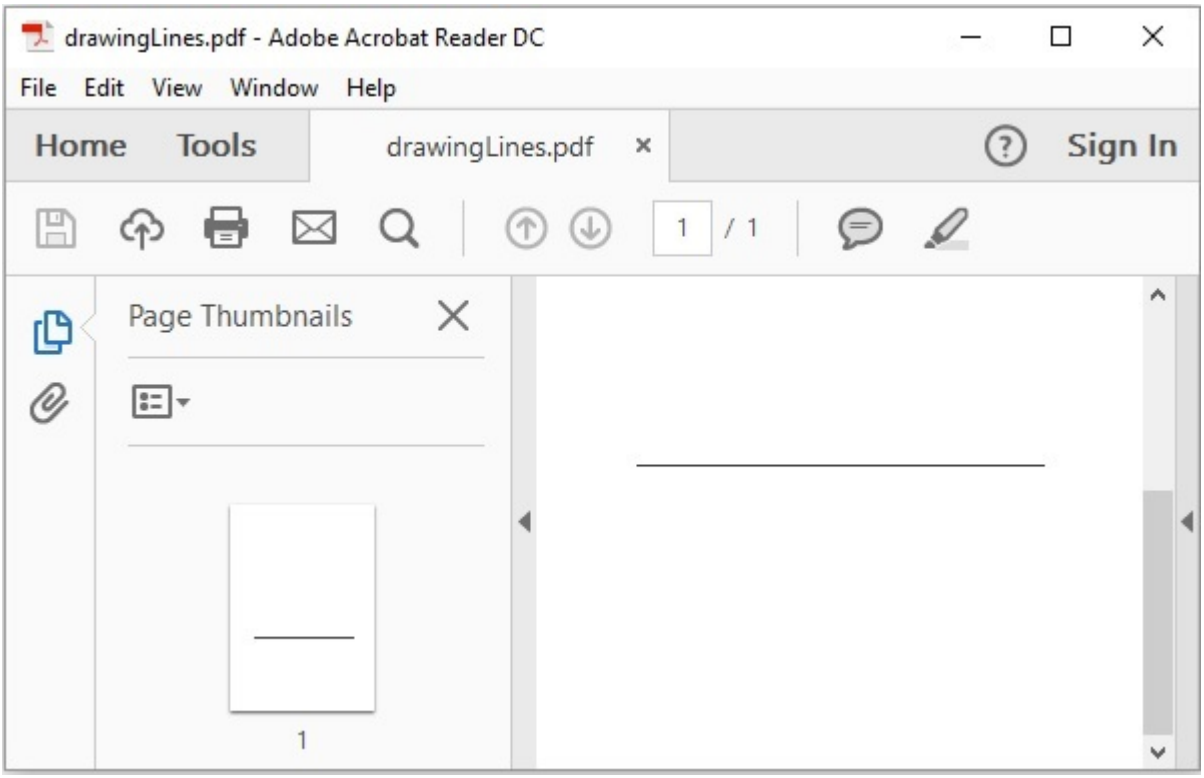
Compile and execute the saved Java file from the Command prompt using the following commands –

```
javac DrawingLine.java
java DrawingLine
```

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

```
Object drawn on pdf successfully
```

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



iText - Drawing a Circle

In this chapter, we will see how to draw a circle on a PDF document using iText library.

Drawing a Circle on a Pdf

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.

To draw a circle on a PdfDocument, instantiate the **PdfCanvas** class of the package **com.itextpdf.kernel.pdf**.canvas and invoke the **circle()** method of this class.

Following are the steps to draw a circle on a PDF document.

Step 1: Creating a PdfWriter object

The **PdfWriter** class represents the DocWriter 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/drawingCircle.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 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 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 a PdfCanvas object

Create a new **PdfPage** class using the **addNewPage()** method of the **PdfDocument** class. Instantiate the **PdfCanvas** object of the package **com.itextpdf.kernel.pdf.canvas** by passing the **PdfPage** object to the constructor of this class, as shown below.

```
// Creating a new page
PdfPage pdfPage = pdfDoc.addNewPage();

// Creating a PdfCanvas object
PdfCanvas canvas = new PdfCanvas(pdfPage);
```

Step 5 Setting the color

Set the color of the circle using the **setColor()** method of the **Canvas** class, as shown below.

```
// Setting color to the circle
Color color = Color.GREEN;
```

```
canvas.setColor(color, true);
```

Step 6: Drawing the Circle

Draw a circle by invoking the **circle()** method of the **Canvas**, as shown below.

```
// creating a circle
canvas.circle(300, 400, 200);
```

Step 7: 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 draw a circle on a pdf document using the iText library. It creates a PDF document with the name **drawingCircle.pdf**, draws a circle in it, and saves it in the path **C:/itextExamples/**

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

```
import com.itextpdf.kernel.color.Color;
import com.itextpdf.kernel.pdf.PdfDocument;
import com.itextpdf.kernel.pdf.PdfPage;
import com.itextpdf.kernel.pdf.PdfWriter;
import com.itextpdf.kernel.pdf.canvas.PdfCanvas;
import com.itextpdf.layout.Document;

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

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

      // Creating a Document object
      Document doc = new Document(pdfDoc);

      // Creating a new page
      PdfPage pdfPage = pdfDoc.addNewPage();

      // Creating a PdfCanvas object
      PdfCanvas canvas = new PdfCanvas(pdfPage);

      // Setting color to the circle
      Color color = Color.GREEN;
      canvas.setColor(color, true);

      // creating a circle
      canvas.circle(300, 400, 200);
```

```
        // Filling the circle
        canvas.fill();

        // Closing the document
        doc.close();

        System.out.println("Object drawn on pdf successfully");
    }
}
```

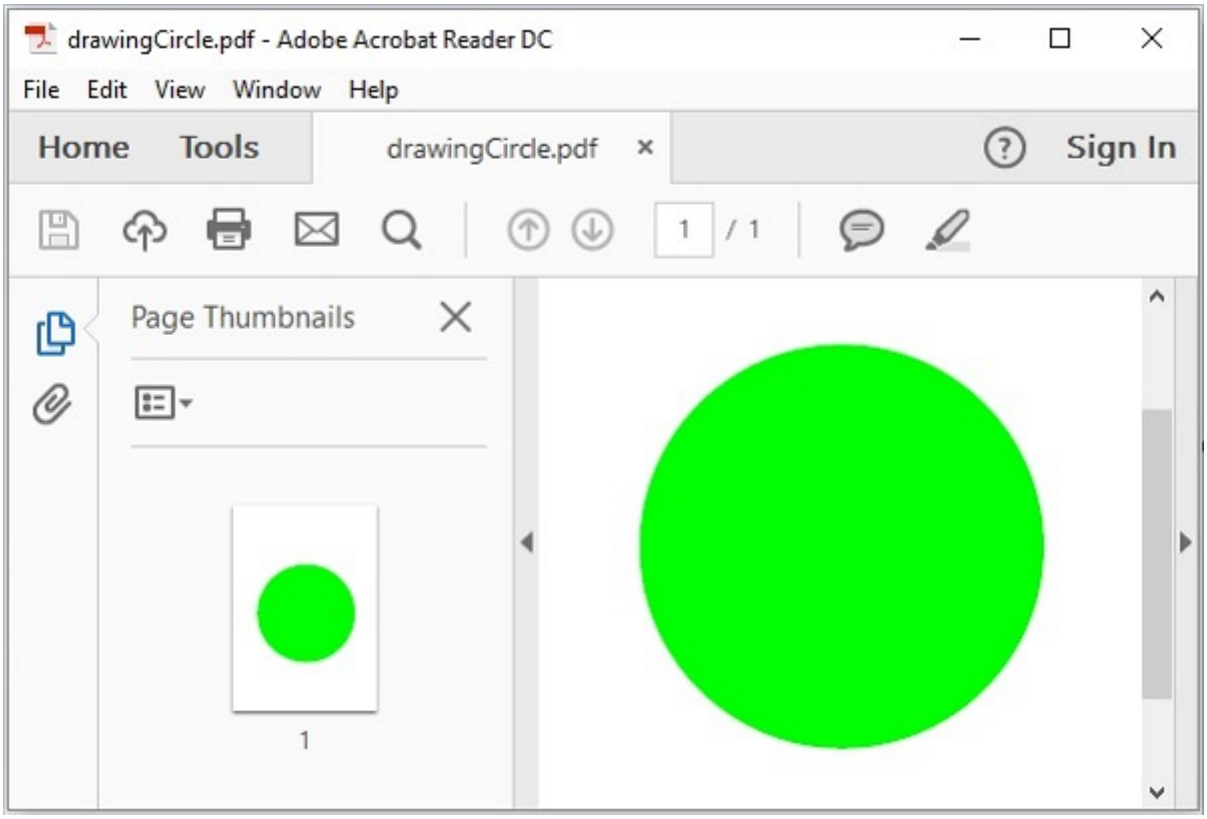
Compile and execute the saved Java file from the Command prompt using the following commands.

```
javac DrawingCircle.java
java DrawingCircle
```

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

```
Object drawn on pdf successfully
```

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



iText - Setting Font

In this chapter, we will see how to set color and font to text in a PDF document using the iText library.

Setting Font of the Text in a PDF

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.

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. You can set color and font to the text using the methods **setFontColor()** and **setFont()** respectively.

Following are the steps to set color and font to text in a pdf document.

Step 1: Creating a PdfWriter object

The **PdfWriter** class represents the DocWriter 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/fonts.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

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 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 Text

Create the text by instantiating the **Text** class of the package **com.itextpdf.layout.element** as shown below.

```
// Creating text object
Text text = new Text("Tutorialspoint");
```

Step 5: Setting the font and color to the text

Create the **PdfFont** object using the **createFont()** method of the class **PdfFontFactory** of the package **com.itextpdf.kernel.font** as shown below

```
// Setting font of the text PdfFont
font = PdfFontFactory.createFont(FontConstants.HELVETICA_BOLD);
```

Now, set font to the text using the **setFont()** method of the **Text** class to this method. Pass the **PdfFont** object as a parameter, as shown below.

```
text1.setFont(font);
```

To set the color to the text invoke the **setFontColor()** method of the Text class, as shown below.

```
// Setting font color
text.setFontColor(Color.GREEN);
```

Step 6: Adding text to the paragraph

Create a **Paragraph** class object and add the above created text using its **add()** method, as shown below.

```
// Creating Paragraph
Paragraph paragraph = new Paragraph();

// Adding text to the paragraph
paragraph.add(text);
```

Step 7: Adding paragraph to the document

Add the paragraph to the document using the **add()** method of the **Document** class, as shown below.

```
doc.add(paragraph1)
```

Step 8: 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 set color and font to text in a PDF using the iText library. It creates a PDF document with the name **fonts.pdf**, formats the text, and saves it in the path **C:/itextExamples/**

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

```
import com.itextpdf.io.font.FontConstants;
import com.itextpdf.kernel.color.Color;
import com.itextpdf.kernel.font.PdfFontFactory;
import com.itextpdf.kernel.font.PdfFont;
import com.itextpdf.kernel.pdf.PdfDocument;
import com.itextpdf.kernel.pdf.PdfWriter;

import com.itextpdf.layout.Document;
import com.itextpdf.layout.element.Paragraph;
import com.itextpdf.layout.element.Text;

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

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

      // Creating a Document object
      Document doc = new Document(pdf);

      // Creating text object
      Text text1 = new Text("Tutorialspoint");

      // Setting font of the text
      PdfFont font = PdfFontFactory.createFont(FontConstants.HELVETICA_BOLD);
      text1.setFont(font);

      // Setting font color
      text1.setFontColor(Color.GREEN);

      // Creating text object
      Text text2 = new Text("Simply Easy Learning");
      text2.setFont(PdfFontFactory.createFont(FontConstants.HELVETICA));

      // Setting font color
      text2.setFontColor(Color.BLUE);

      // Creating Paragraph
      Paragraph paragraph1 = new Paragraph();

      // Adding text1 to the paragraph
```

```
        paragraph1.add(text1);
        paragraph1.add(text2);

        // Adding paragraphs to the document
        doc.add(paragraph1);
        doc.close();

        System.out.println("Text added to pdf ..");
    }
}
```

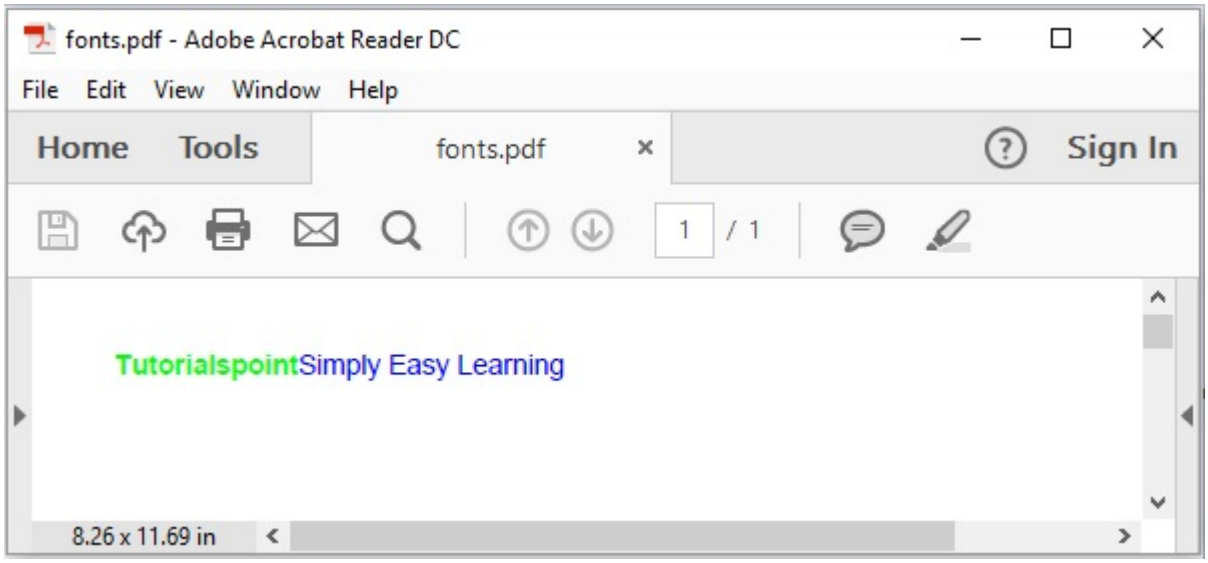
Compile and execute the saved Java file from the Command prompt using the following commands –

```
javac FormatingTheText.java
java FormatingTheText
```

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

```
Text added to pdf ..
```

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



iText - Shrinking the Content

In this chapter, we will see how to scale an image on a PDF document using the iText library.

Shrinking the Content in a PDF

Following are the steps to shrink the contents of a PDF page using iText library.

Step 1: Creating a PdfWriter and PdfReader object

The **PdfWriter** class represents the DocWriter 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 object
String dest = "C:/itextExamples/shrinking.pdf";
PdfWriter writer = new PdfWriter(dest);
```

To read data from an existing pdf, create a **PdfReader** object as shown below.

```
// Creating a PdfReader
String src = "C:/itextExamples/pdfWithImage.pdf";
PdfReader reader = new PdfReader(src);
```

Step 2: Creating a PdfDocument object(s)

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.

Create source and destination PDF documents by passing the **PdfWriter** and **PdfReader** objects to the constructors, as shown below.

```
// Creating a PdfDocument objects
PdfDocument destpdf = new PdfDocument(writer);
PdfDocument srcPdf = new PdfDocument(reader);
```

Step 3: Opening a page from the existing PDF

Get a page from the source PDF using the **getPage()** method of the **PdfPage** class. Using this object, get the size of the page of the source document, as shown below.

```
// Opening a page from the existing PDF
PdfPage origPage = srcPdf.getPage(1);

// Getting the page size
Rectangle orig = origPage.getPageSizeWithRotation();
```

Step 4: Shrinking the contents of the source pdf

Using the **getScaleInstance()** method of the **AffineTransform** class, shrink the contents of a page of the source document, as shown below.

```
// Shrink original page content using transformation matrix
AffineTransform transformationMatrix = AffineTransform.getScaleInstance(
   page.getPageSize().getWidth()/ orig.getWidth()/2,
   page.getPageSize().getHeight()/ orig.getHeight()/2);
```

Step 5: Copying the page

Concatenate the **affine transform matrix**, created in the previous step, to the matrix of the **canvas** object of the destination PDF document, as shown below.

```
// Concatenating the affine transform matrix to the current matrix
PdfCanvas canvas = new PdfCanvas(page);
canvas.concatMatrix(transformationMatrix);
```

Now, add the page copy to the **canvas** object of the destination PDF to the source document, as shown below.

```
// Add the object to the canvas
PdfFormXObject pageCopy = origPage.copyAsFormXObject(destpdf);
canvas.addXObject(pageCopy, 0, 0);
```

Step 6: Creating the 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**, as shown below.

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

Step 7: 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 shrink contents of a PDF page using the iText library. It creates a PDF document with name **shrinkingPDF.pdf**, shrinks the image in the pdf, and saves it in the path **C:/itextExamples/**

Save this code in a file with name **ShrinkingPDF.java**.

```
import com.itextpdf.kernel.geom.AffineTransform;
import com.itextpdf.kernel.geom.Rectangle;

import com.itextpdf.kernel.pdf.PdfDocument;
import com.itextpdf.kernel.pdf.PdfPage;
import com.itextpdf.kernel.pdf.PdfReader;
import com.itextpdf.kernel.pdf.PdfWriter;
```

```
import com.itextpdf.kernel.pdf.canvas.PdfCanvas;

import com.itextpdf.kernel.pdf.xobject.PdfFormXObject;
import com.itextpdf.layout.Document;

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

      // Creating a PdfReader
      String src = "C:/itextExamples/pdfWithImage.pdf";
      PdfReader reader = new PdfReader(src);

      // Creating a PdfDocument objects
      PdfDocument destpdf = new PdfDocument(writer);
      PdfDocument srcPdf = new PdfDocument(reader);

      // Opening a page from the existing PDF
      PdfPage origPage = srcPdf.getPage(1);

      // Getting the page size
      Rectangle orig = origPage.getPageSizeWithRotation();

      // Adding a page to destination Pdf
      PdfPage page = destpdf.addNewPage();

      // Scaling the image in a Pdf page
      AffineTransform transformationMatrix = AffineTransform.getScaleInstance(
         page.getPageSize().getWidth()/orig.getWidth()/2,
         page.getPageSize().getHeight()/ orig.getHeight()/2);

      // Shrink original page content using transformation matrix
      PdfCanvas canvas = new PdfCanvas(page);
      canvas.concatMatrix(transformationMatrix);

      // Add the object to the canvas
      PdfFormXObject pageCopy = origPage.copyAsFormXObject(destpdf);
      canvas.addXObject(pageCopy, 0, 0);

      // Creating a Document object
      Document doc = new Document(destpdf);

      // Closing the document
      doc.close();

      System.out.println("Table created successfully..");
   }
}
```

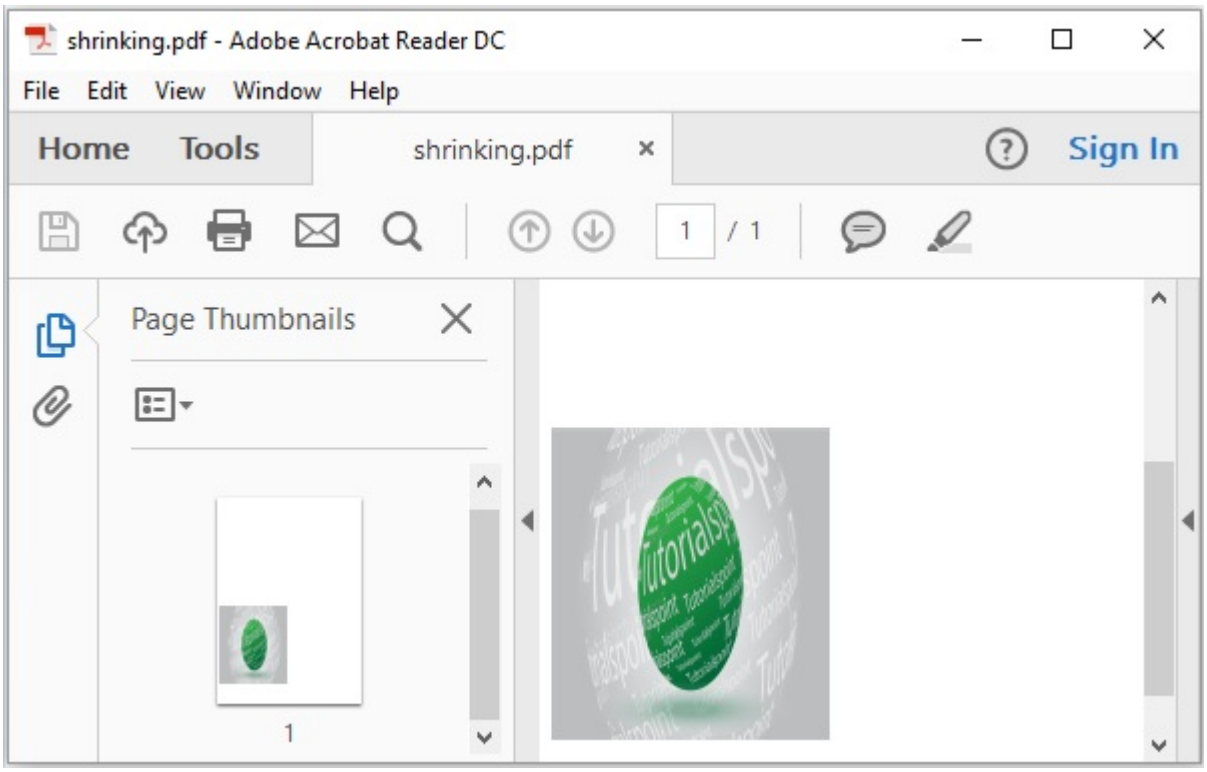
Compile and execute the saved Java file from the command prompt using the following commands –

```
javac ShrinkingPDF.java
java ShrinkingPDF
```

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

```
Table created successfully..
```

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



iText - Tiling PDF Pages

The following Java program demonstrates how to tile the contents of a PDF page to different pages using the iText library. It creates a PDF document with the name **tilingPdfPages.pdf** and saves it in the path **C:/itextExamples/**.

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

```
import com.itextpdf.kernel.geom.AffineTransform;
import com.itextpdf.kernel.geom.PageSize;
import com.itextpdf.kernel.geom.Rectangle;

import com.itextpdf.kernel.pdf.PdfDocument;
import com.itextpdf.kernel.pdf.PdfPage;
import com.itextpdf.kernel.pdf.PdfReader;
import com.itextpdf.kernel.pdf.PdfWriter;
import com.itextpdf.kernel.pdf.canvas.PdfCanvas;
import com.itextpdf.kernel.pdf.xobject.PdfFormXObject;

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

      // Creating a PdfReader
      String src = "C:/itextExamples/pdfWithImage.pdf";
      PdfReader reader = new PdfReader(src);
```

```
      // Creating a PdfDocument objects
      PdfDocument destpdf = new PdfDocument(writer);
      PdfDocument srcPdf = new PdfDocument(reader);

      // Opening a page from the existing PDF
      PdfPage origPage = srcPdf.getPage(1);

      // Getting the page size
      Rectangle orig = origPage.getPageSizeWithRotation();

      // Getting the size of the page
      PdfFormXObject pageCopy = origPage.copyAsFormXObject(destpdf);

      // Tile size
      Rectangle tileSize = PageSize.A4.rotate();
      AffineTransform transformationMatrix =
         AffineTransform.getScaleInstance(tileSize.getWidth() / orig.getWidth() *
         2f, tileSize.getHeight() / orig.getHeight() * 2f);

      // The first tile
      PdfPage page =
      destpdf.addNewPage(PageSize.A4.rotate());

      PdfCanvas canvas = new PdfCanvas(page);
      canvas.concatMatrix(transformationMatrix);
      canvas.addXObject(pageCopy, 0, -orig.getHeight() / 2f);

      // The second tile
      page = destpdf.addNewPage(PageSize.A4.rotate());
      canvas = new PdfCanvas(page);
      canvas.concatMatrix(transformationMatrix);
      canvas.addXObject(pageCopy, -orig.getWidth() / 2f, -orig.getHeight() / 2f);

      // The third tile
      page = destpdf.addNewPage(PageSize.A4.rotate());
      canvas = new PdfCanvas(page);
      canvas.concatMatrix(transformationMatrix);
      canvas.addXObject(pageCopy, 0, 0);

      // The fourth tile
      page = destpdf.addNewPage(PageSize.A4.rotate());
      canvas = new PdfCanvas(page);
      canvas.concatMatrix(transformationMatrix);
      canvas.addXObject(pageCopy, -orig.getWidth() / 2f, 0);

      // closing the documents
      destpdf.close();
      srcPdf.close();

      System.out.println("PDF created successfully..");
   }
}
```

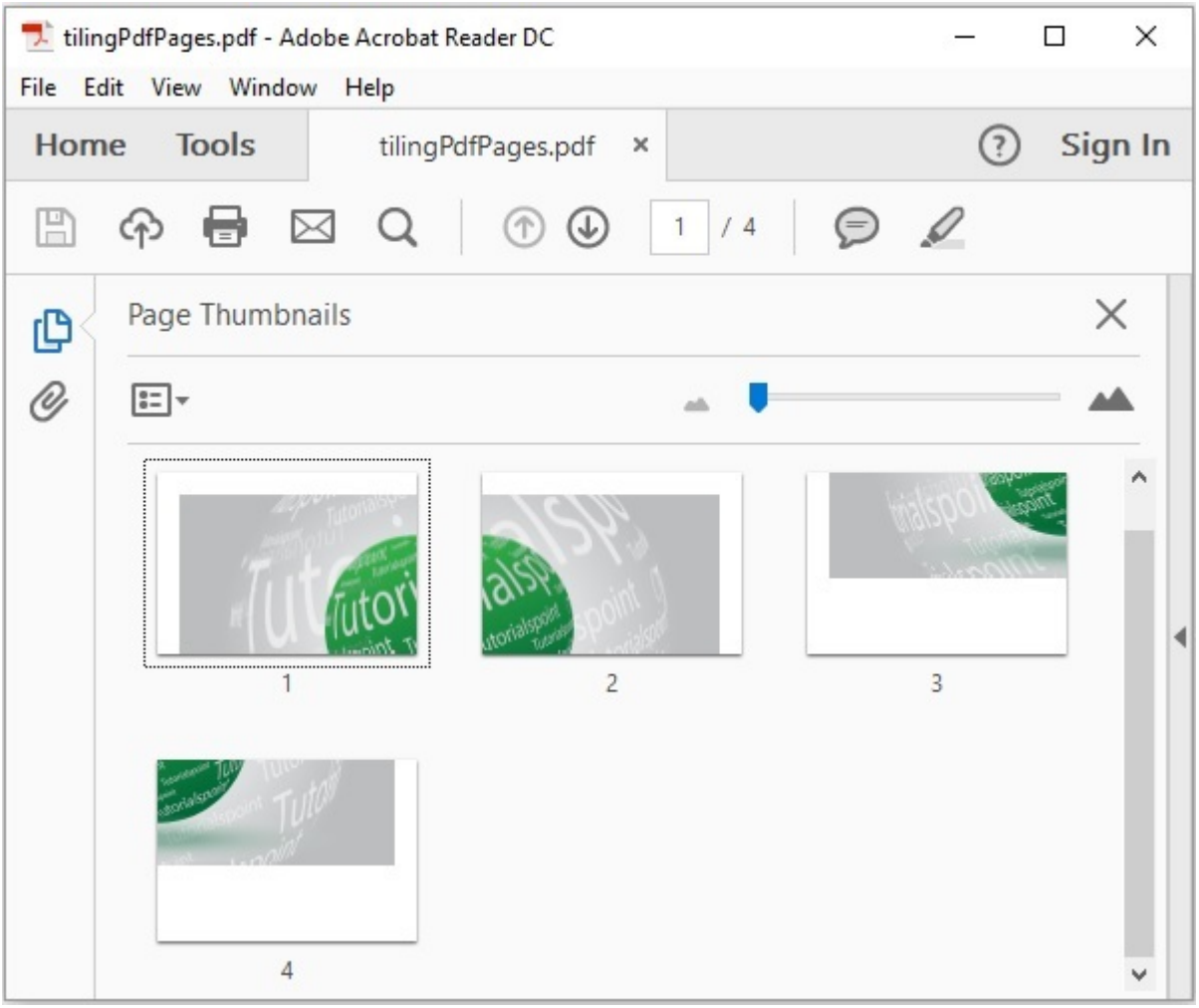
Compile and execute the saved Java file from the Command prompt using the following commands –

```
javac TilingPDFPages.java
java TilingPDFPages
```

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

```
PDF created successfully..
```

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



iText - N-up

The following Java program demonstrates how to perform N-up on a PDF page using the iText library. It creates a PDF document with the name **nUppingPDF.pdf** and saves it in the path **C:/itextExamples/**

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

```
import com.itextpdf.kernel.geom.AffineTransform;
import com.itextpdf.kernel.geom.PageSize;
import com.itextpdf.kernel.geom.Rectangle;

import com.itextpdf.kernel.pdf.PdfDocument;
import com.itextpdf.kernel.pdf.PdfPage;
import com.itextpdf.kernel.pdf.PdfReader;
import com.itextpdf.kernel.pdf.PdfWriter;
import com.itextpdf.kernel.pdf.canvas.PdfCanvas;
import com.itextpdf.kernel.pdf.xobject.PdfFormXObject;
```

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

      // Creating a PdfReader
      String src = "C:/itextExamples/pdfWithImage.pdf";
      PdfReader reader = new PdfReader(src);

      // Creating a PdfDocument objects
      PdfDocument destpdf = new PdfDocument(writer);
      PdfDocument srcPdf = new PdfDocument(reader);

      // Opening a page from the existing PDF
      PdfPage origPage = srcPdf.getPage(1);
      Rectangle orig = origPage.getPageSize();
      PdfFormXObject pageCopy = origPage.copyAsFormXObject(destpdf);

      // N-up page
      PageSize nUpPageSize = PageSize.A4.rotate();
      PdfPage page = destpdf.addNewPage(nUpPageSize);
      PdfCanvas canvas = new PdfCanvas(page);

      // Scale page
      AffineTransform transformationMatrix = AffineTransform.getScaleInstance(
         nUpPageSize.getWidth() / orig.getWidth() /
         2f, nUpPageSize.getHeight() / orig.getHeight() / 2f);
      canvas.concatMatrix(transformationMatrix);

      // Add pages to N-up page
      canvas.addXObject(pageCopy, 0, orig.getHeight());
      canvas.addXObject(pageCopy, orig.getWidth(), orig.getHeight());
      canvas.addXObject(pageCopy, 0, 0);
      canvas.addXObject(pageCopy, orig.getWidth(), 0);

      // closing the documents
      destpdf.close();
      srcPdf.close();

      System.out.println("PDF created successfully..");
   }
}
```

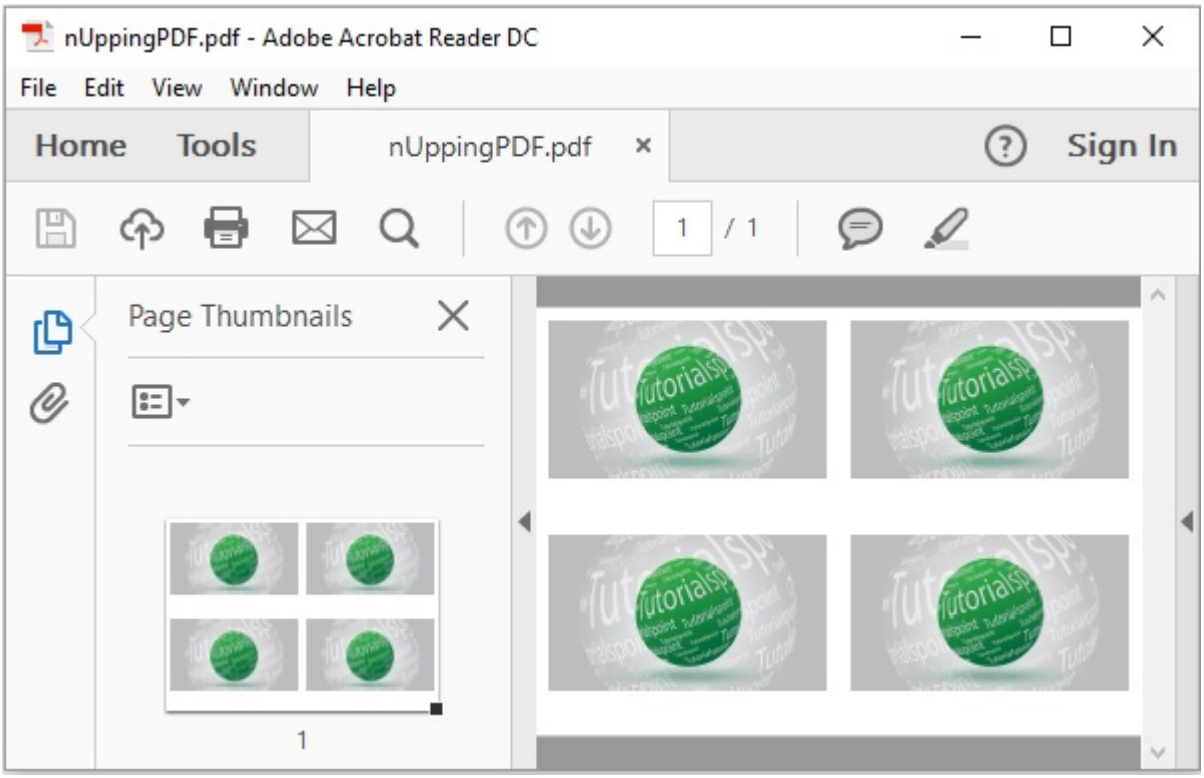
Compile and execute the saved Java file from the Command prompt using the following commands –

```
javac NUppingPDF.java
java NUppingPDF
```

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

```
PDF created successfully..
```

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



Previous Page Next Page

Enter email for newsletter go