LEARNING itext

#itext

Table of Contents

About

Chapter 1: Getting started with itext

Remarks

If you look at PDF creation, you'll find two different approaches:

- Graphical designers use desktop applications such as Adobe Acrobat or Adobe InDesign to create a document in a manual or semimanual process.
- In another context, PDF documents are created programmatically, using an API to produce PDFs directly from software applications, without —or with minimal— human intervention. Sometimes the document is created in an intermediary format first (e.g. XML, HTML,...), and then converted to PDF.

These different approaches demand different software products.

The same goes for PDF manipulation.

- You can update a PDF manually in tools such as Adobe Acrobat,
- There are also tools that allow forms to be filled out automatically based on information from a database.

iText is a tool that focuses on the automation side of things.

What is iText?

iText is an SDK that was developed to allow developers to do the following (and much more):

- Generate documents and reports based on data from an XML file or a database
- Create maps and books, exploiting numerous interactive features available in PDF
- Add bookmarks, page numbers, watermarks, and other features to existing PDF documents
- Split or concatenate pages from existing PDF files
- Fill out interactive forms
- Digitally sign PDF documents
- Serve dynamically generated or manipulated PDF documents to a web browser

iText is not an end-user tool. You have to build iText into your own applications so that you can automate the PDF creation and manipulation process.

When to use iText?

Typically, iText is used in projects that have one of the following requirements:

- The content isn't available in advance: it's calculated based on user input or real-time database information.

- The PDF files can't be produced manually due to the massive volume of content: a large number of pages or documents.
- Documents need to be created in unattended mode, in a batch process.
- The content needs to be customized or personalized; for instance, the name of the end user has to be stamped on a number of pages.

Often you'll encounter these requirements in web applications, where content needs to be served dynamically to a browser. Normally, you'd serve this information in the form of HTML, but for some documents, PDF is preferred over HTML for better printing quality, for identical presentation on a variety of platforms, for security reasons, to comply with specific industry standards (such as PAdES, PDF/A, or PDF/UA), or to reduce the file size.

Versions

Version	First release	Latest release	End-of-Life
0.30 - 0.99	2000-02-14	2003-05-01	2005-12-31
1.00 - 1.4.8	2003-06-25	2006-12-19	2009-12-31
2.00 - 2.1.7	2003-02-15	2009-07-07	2012-12-31
5.0.0 - 5.5.11	2009-12-07	2017-03-20	2018-12-31
7.0.0 - ...	2016-05-03	...	2025-12-31

Examples

Installation or Setup

iText for Java

Importing the iText jars from the Central Maven Repository is the best way to install iText 7. These simple videos explain how to do this using different IDEs:

- How to import iText 7 in Eclipse to create a Hello World PDF?
- How to import iText 7 in Netbeans to create a Hello World PDF?
- How to import iText 7 in IntelliJ IDEA to create a Hello World PDF?

In these tutorials, we only define the `kernel` and the `layout` projects as dependencies. Maven also automatically imports the `io` jar because the `kernel` packages depend on the `io` packages.

This is the basic list of dependencies for standard use of iText 7:

```
<dependencies>
    <dependency>
```

```
        <groupId>com.itextpdf</groupId>
        <artifactId>kernel</artifactId>
        <version>7.0.0</version>
        <scope>compile</scope>
    </dependency>
    <dependency>
        <groupId>com.itextpdf</groupId>
        <artifactId>io</artifactId>
        <version>7.0.0</version>
        <scope>compile</scope>
    </dependency>
    <dependency>
        <groupId>com.itextpdf</groupId>
        <artifactId>layout</artifactId>
        <version>7.0.0</version>
        <scope>compile</scope>
    </dependency>
    <dependency>
        <groupId>com.itextpdf</groupId>
        <artifactId>forms</artifactId>
        <version>7.0.0</version>
        <scope>compile</scope>
    </dependency>
    <dependency>
        <groupId>com.itextpdf</groupId>
        <artifactId>pdfa</artifactId>
        <version>7.0.0</version>
        <scope>compile</scope>
    </dependency>
    <dependency>
        <groupId>com.itextpdf</groupId>
        <artifactId>pdftest</artifactId>
        <version>7.0.0</version>
        <scope>compile</scope>
    </dependency>
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-log4j12</artifactId>
        <version>1.7.18</version>
    </dependency>
</dependencies>
```

Every dependency corresponds with a jar in Java and with a DLL in C#.

- `kernel` and `io`: contain low-level functionality.
- `layout`: contains high-level functionality.
- `forms`: needed for all the AcroForm examples.
- `pdfa`: needed for PDF/A-specific functionality.
- `pdftest`: needed for the examples that are also a test.

For more specific use of iText 7, you may need additional jars:

- `barcodes`: use this if you want to create bar codes.
- `hyph`: use this if you want text to be hyphenated.
- `font-asian`: use this is you need CJK functionality (Chinese / Japanese / Korean)
- `sign`: use this if you need support for digital signatures.

All the jars listed above are available under the AGPL license. You can also download these jars in a ZIP-file hosted on Github: https://github.com/itext/itext7/releases

If you want to use these jars, you have to add them to your CLASSPATH, just like you would add any other jar.

Additional iText 7 functionality is available through add-ons, which are delivered as jars under a commercial license. If you want to use any of these add-ons, or if you want to use iText 7 with your proprietary code, you need to obtain a commercial license key for iText 7 (see the legal section of the iText web site).

You can import such a license key using the license-key module. You can get the license-key jar like this:

```
<dependency>
    <groupId>com.itextpdf</groupId>
    <artifactId>itext-licensekey</artifactId>
    <version>2.0.0</version>
    <scope>compile</scope>
</dependency>
```

Some functionality in iText is closed source. For instance, if you want to use **PdfCalligraph**, you need the `typography` module. This module won't work without an official license key.

iText for C#

You can download a ZIP-file containing all the DLLs that are available under the AGPL. For more info about these DLLs, please read the Java documentation.

Hello World

This is a very simple program to create a PDF using iText 7 / Java:

```
//Initialize writer
PdfWriter writer = new PdfWriter(dest);

//Initialize document
PdfDocument pdfDoc = new PdfDocument(writer);
Document doc = new Document(pdfDoc);

//Add paragraph to the document
doc.add(new Paragraph("Hello World!"));

//Close document
doc.close();
```

(*Listing_01_01_HelloWorld.java*)

You can navigate to many other examples from that page.

And this is a very simple program to create a PDF using the precursor iText 5.5.x / Java:

```
// step 1
Document document = new Document();
// step 2
PdfWriter.getInstance(document, new FileOutputStream(filename));
// step 3
document.open();
// step 4
document.add(new Paragraph("Hello World!"));
// step 5
document.close();
```

(HelloWorld.java)

There are many more examples to navigate from this page, too.

These two examples look pretty similar. The advantages of the re-designed iText 7 API will become apparent, though, as soon as one starts to look closer at less trivial examples. Thus, simply navigate through the example source code from the links above and compare.

Read Getting started with itext online: https://riptutorial.com/itext/topic/3557/getting-started-with-itext

Chapter 2: Columns: iText 5 versus iText 7

Remarks

In iText 5, you can't use the `add()` method to add a `Paragraph` to a `Document` if you want to organize the content in columns. We can't reuse the code of the Text2Pdf.java (iText 5) example.

Instead we have to create a `ColumnText` object, we have to add all the `Paragraph` objects to this object, and once we've finished adding all the content, we can start rendering that content using the `go()` method. While doing so, we have to keep track of the columns, and create new pages when necessary.

What we fixed in iText 7:

With iText 7, we can copy and paste the code from the Text2Pdf.java (iText 7) example. We can continue using the `add()` method the same way we did before. If we want to render the content in two columns instead of in one, we simple have to change the document renderer:

```
Rectangle[] columns = {
    new Rectangle(36, 36, 254, 770),
    new Rectangle(305, 36, 254, 770)};
document.setRenderer(new ColumnDocumentRenderer(document, columns));
```

Want to know more?

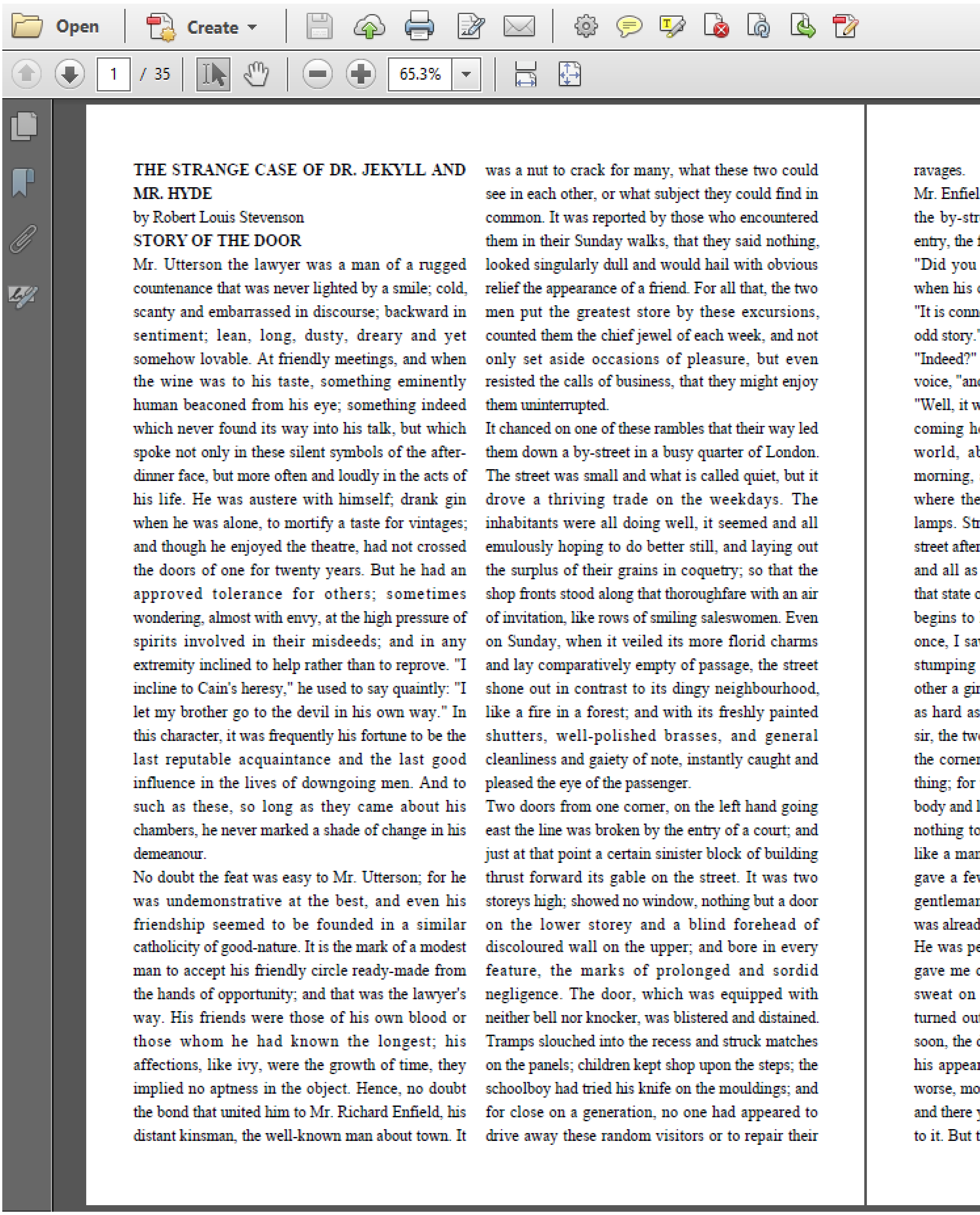
Read Working with the RootElement which is chapter 5 in the iText 7: Building Blocks tutorial. Get the free ebook!

Examples

Text2PdfColumns.java (iText 5)

Suppose that we have the following text file: jekyll_hyde.txt

How do we convert it to a PDF that looks like this:



THE STRANGE CASE OF DR. JEKYLL AND MR. HYDE
by Robert Louis Stevenson
STORY OF THE DOOR
Mr. Utterson the lawyer was a man of a rugged countenance that was never lighted by a smile; cold, scanty and embarrassed in discourse; backward in sentiment; lean, long, dusty, dreary and yet somehow lovable. At friendly meetings, and when the wine was to his taste, something eminently human beaconed from his eye; something indeed which never found its way into his talk, but which spoke not only in these silent symbols of the after-dinner face, but more often and loudly in the acts of his life. He was austere with himself; drank gin when he was alone, to mortify a taste for vintages; and though he enjoyed the theatre, had not crossed the doors of one for twenty years. But he had an approved tolerance for others; sometimes wondering, almost with envy, at the high pressure of spirits involved in their misdeeds; and in any extremity inclined to help rather than to reprove. "I incline to Cain's heresy," he used to say quaintly: "I let my brother go to the devil in his own way." In this character, it was frequently his fortune to be the last reputable acquaintance and the last good influence in the lives of downgoing men. And to such as these, so long as they came about his chambers, he never marked a shade of change in his demeanour.
No doubt the feat was easy to Mr. Utterson; for he was undemonstrative at the best, and even his friendship seemed to be founded in a similar catholicity of good-nature. It is the mark of a modest man to accept his friendly circle ready-made from the hands of opportunity; and that was the lawyer's way. His friends were those of his own blood or those whom he had known the longest; his affections, like ivy, were the growth of time, they implied no aptness in the object. Hence, no doubt the bond that united him to Mr. Richard Enfield, his distant kinsman, the well-known man about town. It was a nut to crack for many, what these two could see in each other, or what subject they could find in common. It was reported by those who encountered them in their Sunday walks, that they said nothing, looked singularly dull and would hail with obvious relief the appearance of a friend. For all that, the two men put the greatest store by these excursions, counted them the chief jewel of each week, and not only set aside occasions of pleasure, but even resisted the calls of business, that they might enjoy them uninterrupted.
It chanced on one of these rambles that their way led them down a by-street in a busy quarter of London. The street was small and what is called quiet, but it drove a thriving trade on the weekdays. The inhabitants were all doing well, it seemed and all emulously hoping to do better still, and laying out the surplus of their grains in coquetry; so that the shop fronts stood along that thoroughfare with an air of invitation, like rows of smiling saleswomen. Even on Sunday, when it veiled its more florid charms and lay comparatively empty of passage, the street shone out in contrast to its dingy neighbourhood, like a fire in a forest; and with its freshly painted shutters, well-polished brasses, and general cleanliness and gaiety of note, instantly caught and pleased the eye of the passenger.
Two doors from one corner, on the left hand going east the line was broken by the entry of a court; and just at that point a certain sinister block of building thrust forward its gable on the street. It was two storeys high; showed no window, nothing but a door on the lower storey and a blind forehead of discoloured wall on the upper; and bore in every feature, the marks of prolonged and sordid negligence. The door, which was equipped with neither bell nor knocker, was blistered and distained. Tramps slouched into the recess and struck matches on the panels; children kept shop upon the steps; the schoolboy had tried his knife on the mouldings; and for close on a generation, no one had appeared to drive away these random visitors or to repair their
ravages.
Mr. Enfiel
the by-str
entry, the f
"Did you
when his c
"It is conn
odd story."
"Indeed?"
voice, "and
"Well, it w
coming ho
world, ab
morning,
where the
lamps. Str
street after
and all as
that state o
begins to
once, I sa
stumping
other a gir
as hard as
sir, the tw
the corner
thing; for
body and l
nothing to
like a man
gave a fev
gentleman
was alread
He was pe
gave me o
sweat on
turned ou
soon, the
his appear
worse, mo
and there y
to it. But t

When using iText 5, you'd need code like this:

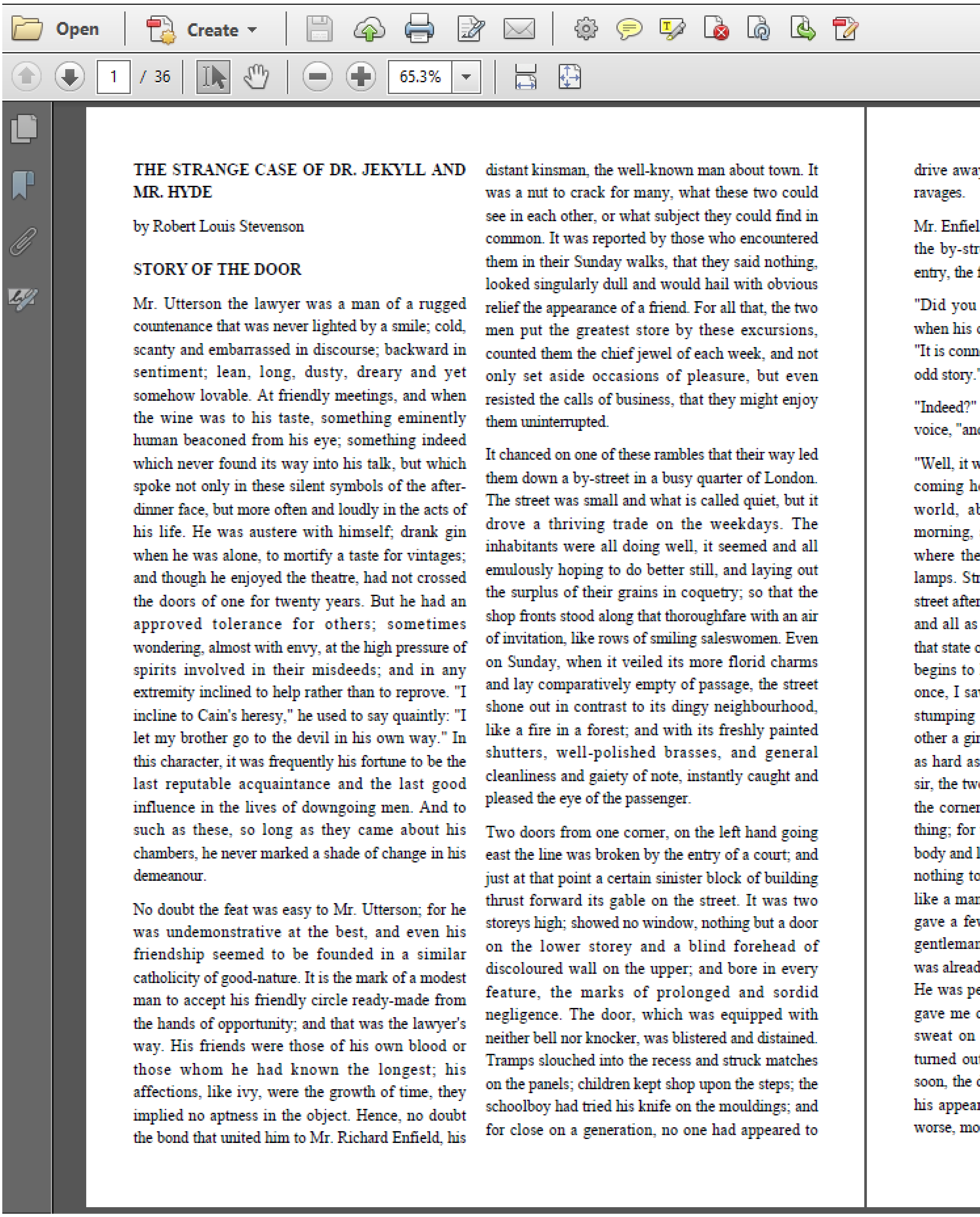
```
public void createPdf(String dest)
throws DocumentException, IOException {
    Document document = new Document();
    PdfWriter writer = PdfWriter.getInstance(document, new FileOutputStream(dest));
    document.open();
    ColumnText ct = new ColumnText(writer.getDirectContent());
    BufferedReader br = new BufferedReader(new FileReader(TEXT));
    String line;
    Paragraph p;
    Font normal = new Font(FontFamily.TIMES_ROMAN, 12);
    Font bold = new Font(FontFamily.TIMES_ROMAN, 12, Font.BOLD);
    boolean title = true;
    while ((line = br.readLine()) != null) {
        p = new Paragraph(line, title ? bold : normal);
        p.setAlignment(Element.ALIGN_JUSTIFIED);
        title = line.isEmpty();
        ct.addElement(p);
    }
    Rectangle[] columns = {
        new Rectangle(36, 36, 290, 806), new Rectangle(305, 36, 559, 806)
    };
    int c = 0;
    int status = ColumnText.START_COLUMN;
    while (ColumnText.hasMoreText(status)) {
        ct.setSimpleColumn(columns[c]);
        status = ct.go();
        if (++c == 2) {
            document.newPage();
            c = 0;
        }
    }
    document.close();
}
```

Source: developers.itextpdf.com

Text2PdfColumns.java (iText 7)

Suppose that you have the following text file: jekyll_hyde.txt

How do we convert it to a PDF that looks like this:



THE STRANGE CASE OF DR. JEKYLL AND MR. HYDE

by Robert Louis Stevenson

STORY OF THE DOOR

Mr. Utterson the lawyer was a man of a rugged countenance that was never lighted by a smile; cold, scanty and embarrassed in discourse; backward in sentiment; lean, long, dusty, dreary and yet somehow lovable. At friendly meetings, and when the wine was to his taste, something eminently human beaconed from his eye; something indeed which never found its way into his talk, but which spoke not only in these silent symbols of the after-dinner face, but more often and loudly in the acts of his life. He was austere with himself; drank gin when he was alone, to mortify a taste for vintages; and though he enjoyed the theatre, had not crossed the doors of one for twenty years. But he had an approved tolerance for others; sometimes wondering, almost with envy, at the high pressure of spirits involved in their misdeeds; and in any extremity inclined to help rather than to reprove. "I incline to Cain's heresy," he used to say quaintly: "I let my brother go to the devil in his own way." In this character, it was frequently his fortune to be the last reputable acquaintance and the last good influence in the lives of downgoing men. And to such as these, so long as they came about his chambers, he never marked a shade of change in his demeanour.

No doubt the feat was easy to Mr. Utterson; for he was undemonstrative at the best, and even his friendship seemed to be founded in a similar catholicity of good-nature. It is the mark of a modest man to accept his friendly circle ready-made from the hands of opportunity; and that was the lawyer's way. His friends were those of his own blood or those whom he had known the longest; his affections, like ivy, were the growth of time, they implied no aptness in the object. Hence, no doubt the bond that united him to Mr. Richard Enfield, his distant kinsman, the well-known man about town. It was a nut to crack for many, what these two could see in each other, or what subject they could find in common. It was reported by those who encountered them in their Sunday walks, that they said nothing, looked singularly dull and would hail with obvious relief the appearance of a friend. For all that, the two men put the greatest store by these excursions, counted them the chief jewel of each week, and not only set aside occasions of pleasure, but even resisted the calls of business, that they might enjoy them uninterrupted.

It chanced on one of these rambles that their way led them down a by-street in a busy quarter of London. The street was small and what is called quiet, but it drove a thriving trade on the weekdays. The inhabitants were all doing well, it seemed and all emulously hoping to do better still, and laying out the surplus of their grains in coquetry; so that the shop fronts stood along that thoroughfare with an air of invitation, like rows of smiling saleswomen. Even on Sunday, when it veiled its more florid charms and lay comparatively empty of passage, the street shone out in contrast to its dingy neighbourhood, like a fire in a forest; and with its freshly painted shutters, well-polished brasses, and general cleanliness and gaiety of note, instantly caught and pleased the eye of the passenger.

Two doors from one corner, on the left hand going east the line was broken by the entry of a court; and just at that point a certain sinister block of building thrust forward its gable on the street. It was two storeys high; showed no window, nothing but a door on the lower storey and a blind forehead of discoloured wall on the upper; and bore in every feature, the marks of prolonged and sordid negligence. The door, which was equipped with neither bell nor knocker, was blistered and distained. Tramps slouched into the recess and struck matches on the panels; children kept shop upon the steps; the schoolboy had tried his knife on the mouldings; and for close on a generation, no one had appeared to

drive awa
ravages.

Mr. Enfiel
the by-str
entry, the f

"Did you
when his c
"It is conn
odd story."

"Indeed?"
voice, "and

"Well, it w
coming ho
world, ab
morning,
where the
lamps. Str
street after
and all as
that state o
begins to
once, I sa
stumping
other a gir
as hard as
sir, the two
the corner
thing; for
body and l
nothing to
like a man
gave a few
gentleman
was alread
He was pe
gave me o
sweat on
turned out
soon, the c
his appear
worse, mo

When using iText 7, you'd need code like this:

```
public void createPdf(String dest) throws IOException {
    PdfDocument pdf = new PdfDocument(new PdfWriter(dest));
    Document document = new Document(pdf)
        .setTextAlignment(TextAlignment.JUSTIFIED);
    Rectangle[] columns = {
        new Rectangle(36, 36, 254, 770),
        new Rectangle(305, 36, 254, 770)};
    document.setRenderer(new ColumnDocumentRenderer(document, columns));
    BufferedReader br = new BufferedReader(new FileReader(TEXT));
    String line;
    PdfFont normal = PdfFontFactory.createFont(FontConstants.TIMES_ROMAN);
    PdfFont bold = PdfFontFactory.createFont(FontConstants.TIMES_BOLD);
    boolean title = true;
    while ((line = br.readLine()) != null) {
        document.add(new Paragraph(line).setFont(title ? bold : normal));
        title = line.isEmpty();
    }
    document.close();
}
```

Source: developers.itextpdf.com and the iText 7: Building Blocks tutorial.

Read Columns: iText 5 versus iText 7 online: https://riptutorial.com/itext/topic/5788/columns--itext-5-versus-itext-7

Chapter 3: Fonts: iText 5 versus iText 7

Remarks

In the first versions of iText, there was only one font class: `Font`.

With this font, you could create a `Font` object for fourteen fonts from five font families: Helvetica (regular, bold, oblique, bold-oblique), Times Roman (regular, bold, italic, bold-italic), Courier (regular, bold, oblique, bold-oblique), Symbol and Zapf Dingbats.

Such a `Font` object was created like this:

```
Font font = new Font(FontFamily.TIMES_ROMAN);
```

You also had to define the font size, for instance:

```
Font font14pt = new Font(FontFamily.TIMES_ROMAN, 14);
```

The default font was Helvetica; the default font size 12.

iText evolved and more fonts were supported. The `BaseFont` class was used to deal with these fonts internally. A `BaseFont` class was created like this:

```
BaseFont bf_russian = BaseFont.createFont(
    "resources/fonts/FreeSans.ttf",
    "CP1251",
    BaseFont.EMBEDDED);
```

The first parameter is the path to a font program, for instance a TTF file, the second parameter is the encoding, for instance CP1251 for Cyrillic characters, the third parameter indicates if a subset of the font needs to be embedded.

The `BaseFont` class is to be used when you add content at the lowest level, for instance when creating text objects in your code using `beginText()`, `setFontAndSize()`, `setTextMatrix()`, `showText()`, `endText()` sequences. Typically, you will only use this low-level approach if you are a PDF specialist. If you don't know anything of PDF syntax, you shouldn't use such a sequence.

You can also use the `BaseFont` class to create a `Font` object:

```
Font russian = new Font(bf_russian, 12);
```

Now we can use the `russian` font to create a `Paragraph` that contains Russian text.

There are some other ways in iText 5 to create `Font` objects, but this is the most common procedure. People were sometimes confused by the difference between `Font` and `BaseFont`, and they didn't always use the correct approach.

What we fixed in iText 7:

We made things more uniform. There is now a single `PdfFont` class, and you create a font using a `PdfFontFactory`:

```
PdfFont font = PdfFontFactory.createFont(FontConstants.TIMES_ROMAN);
PdfFont russian = PdfFontFactory.createFont(
    "src/main/resources/fonts/FreeSans.ttf", "CP1251", true);
```

You no longer need to create different font objects if you want to switch to another font size. Switching to a different font size can simply be done using the `setFontSize()` method:

```
Paragraph p = new Paragraph("Hello World! ")
    .add(new Text("Hallo Wereld! ").setFontSize(14))
    .add(new Text("Bonjour le monde! ").setFontSize(10));
```

The default font is still Helvetica and the default font size is still 12, but you can now define a font (and a font size) for the document:

```
document.setFont(font);
```

In this case `font` will be the default font when adding a building block (for instance a `Paragraph`) without specifying a font.

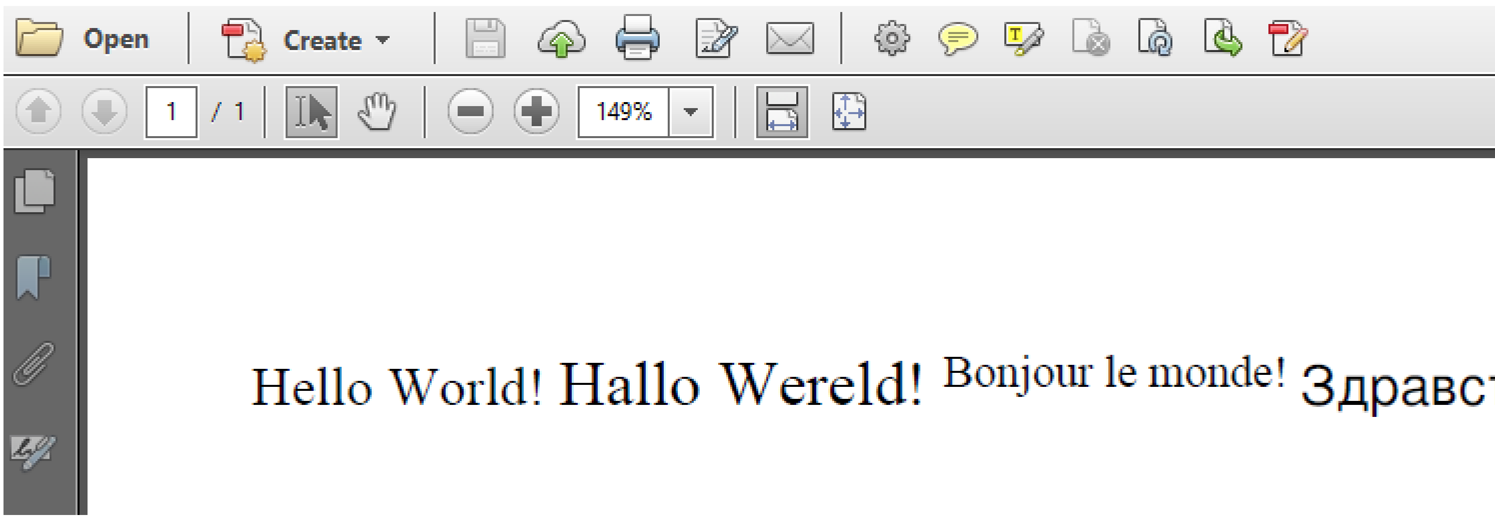
Want to know more?

Read Introducing the PdfFont class which is chapter 1 in the iText 7: Building Blocks tutorial. Get the free ebook!

Examples

HelloWorldInternational.java (iText 5)

In this iText 5 example, we will create a Hello World example in different languages, using different fonts:



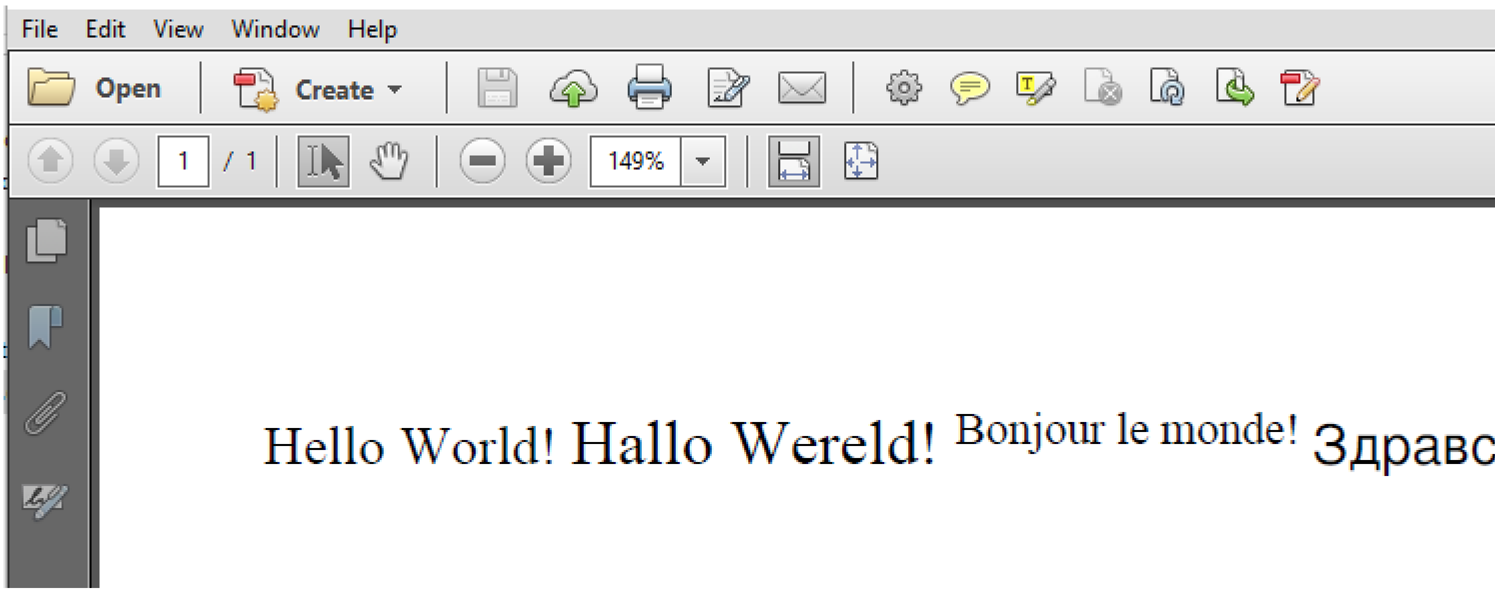
Hello World! Hallo Wereld! Bonjour le monde! Здравс

```
public void createPdf(String dest)
    throws DocumentException, IOException {
    Document document = new Document();
    PdfWriter.getInstance(document, new FileOutputStream(dest));
    document.open();
    Font font = new Font(FontFamily.TIMES_ROMAN);
    Font font14pt = new Font(FontFamily.TIMES_ROMAN, 14);
    Font font10pt = new Font(FontFamily.TIMES_ROMAN, 10);
    BaseFont bf_russian = BaseFont.createFont(
        "resources/fonts/FreeSans.ttf",
        "CP1251",
        BaseFont.EMBEDDED);
    Font russian = new Font(bf_russian, 12);
    BaseFont bf_cjk = BaseFont.createFont(
        "resources/fonts/NotoSansCJKsc-Regular.otf",
        BaseFont.IDENTITY_H,
        BaseFont.EMBEDDED);
    Font cjk = new Font(bf_cjk, 12);
    Paragraph p = new Paragraph("Hello World! ", font);
    Chunk chunk = new Chunk("Hallo Wereld! ", font14pt);
    p.add(chunk);
    chunk = new Chunk("Bonjour le monde! ", font10pt);
    chunk.setTextRise(4);
    p.add(chunk);
    chunk = new Chunk(
        "\u0417\u0434\u0440\u0430\u0432\u0441\u0442\u0432\u0443\u043b\u0442\u0435
\u043c\u0438\u0440! ",
        russian);
    p.add(chunk);
    p.add(new Chunk("\u4f60\u597d\u4e16\u754c! ", cjk));
    p.add(new Chunk("\uc5ec\ubcf4\uc138\uc694 \uc138\uacc4!", cjk));
    document.add(p);
    document.close();
}
```

Source: developers.itextpdf.com

HelloWorldInternational.java (iText 7)

In this iText 7 example, we will create a Hello World example in different languages, using different fonts:


File Edit View Window Help
Open
Create
1 / 1
149%
Hello World! Hallo Wereld! Bonjour le monde! Здравс

```
public void createPdf(String dest) throws IOException {
    PdfDocument pdf = new PdfDocument(new PdfWriter(dest));
    try (Document document = new Document(pdf)) {
        PdfFont font = PdfFontFactory.createFont(FontConstants.TIMES_ROMAN);
        PdfFont russian = PdfFontFactory.createFont(
            "src/main/resources/fonts/FreeSans.ttf",
            "CP1251", true);
        PdfFont cjk = PdfFontFactory.createFont(
            "src/main/resources/fonts/NotoSansCJKsc-Regular.otf",
            PdfEncodings.IDENTITY_H, true);
        document.setFont(font);
        Paragraph p = new Paragraph("Hello World! ")
            .add(new Text("Hallo Wereld! ").setFontSize(14))
            .add(new Text("Bonjour le monde! ").setFontSize(10).setTextRise(4))
            .add(
                new
Text("\u0417\u0434\u0440\u0430\u0432\u0441\u0442\u0432\u0443\u043b\u0442\u0435
\u043c\u0438\u0440! ")
                    .setFont(russian))
            .add(new Text("\u4f60\u597d\u4e16\u754c! ")
                .setFont(cjk))
            .add(new Text("\uc5ec\ubcf4\uc138\uc694 \uc138\uacc4!")
                .setFont(cjk));
        document.add(p);
    }
}
```

Source: developers.itextpdf.com and the iText 7: Building Blocks tutorial.

Read Fonts: iText 5 versus iText 7 online: https://riptutorial.com/itext/topic/5780/fonts--itext-5-versus-itext-7

Chapter 4: Forms: iText 5 vs iText 7

Remarks

iText 5 is a library that has grown organically. Many developers contributed code. For instance: one developer contributed code to create form fields from scratch, using classes such as `TextField` and `PdfFormField`; another developer contributed code to change existing form fields, using the `AcroField` class and a series of `setFieldProperty()` methods.

In iText 5, the classes used to create form fields cannot be used to change form fields, and vice-versa. There is no relationship whatsoever between the two sets of classes. That's confusing for many users. For instance: some users discover the `TextField` class, and assume they can use that class to change the properties of an existing text field. This isn't the case, they need to use the `AcroFields` class instead.

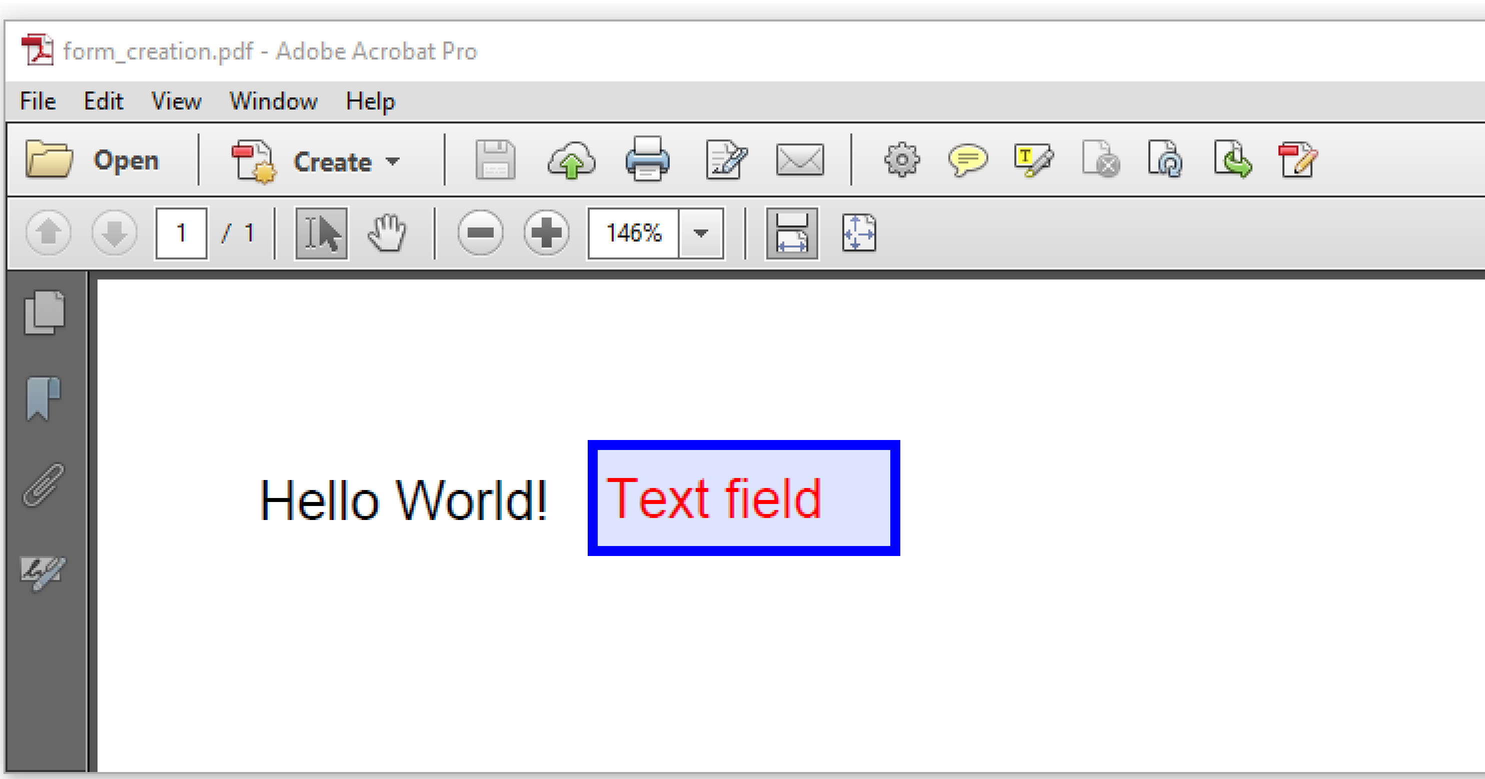
All of this is fixed in iText 7. We created a new set of classes such as `PdfFormField` and its subclass `PdfTextField` that can be used to create a new field, as well as to update an existing form field.

The iText 7 form field methods can be chained to make your code more compact, and they are much more intuitive than the corresponding methods in iText 5. Making the form functionality more elegant was one of the key reasons to rewrite iText from scratch.

Examples

FormCreation.java (iText 5)

In this iText 5 example, we'll create a text field and we'll add it to a PDF:

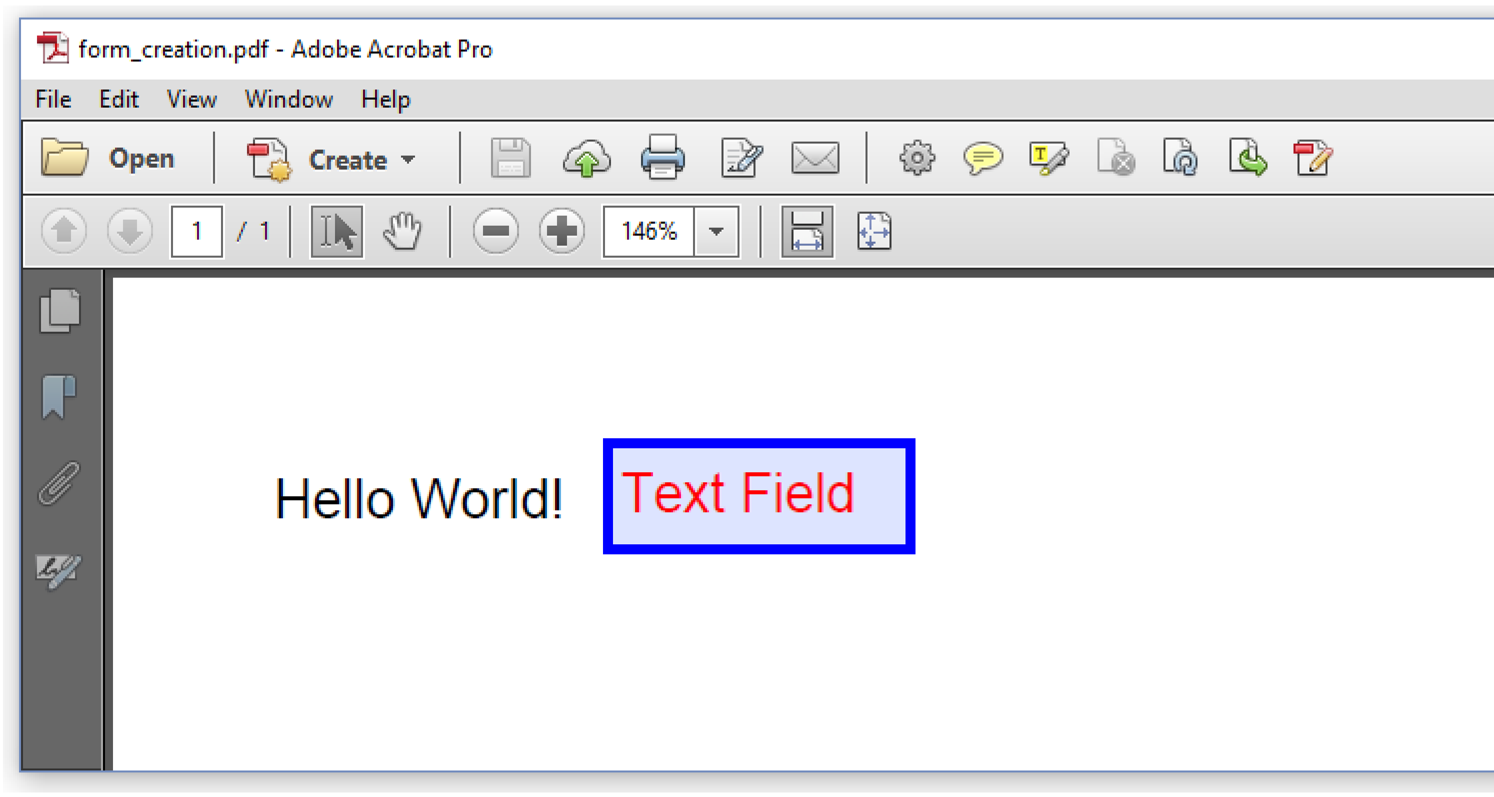

form_creation.pdf - Adobe Acrobat Pro
File Edit View Window Help
Open
Create
1 / 1
146%
Hello World!
Text field

```
public void manipulatePdf(String src, String dest) throws DocumentException, IOException {
    PdfReader reader = new PdfReader(src);
    PdfStamper stamper = new PdfStamper(reader, new FileOutputStream(dest));
    TextField tf = new TextField(stamper.getWriter(),
        new Rectangle(110, 780, 180, 806), "text");
    tf.setBorderColor(BaseColor.BLUE);
    tf.setBorderWidth(2);
    tf.setTextColor(BaseColor.RED);
    tf.setFontSize(12);
    tf.setText("Text field");
    PdfFormField field = tf.getTextField();
    stamper.addAnnotation(field, 1);
    stamper.close();
    reader.close();
}
```

Source: developers.itextpdf.com

FormCreation.java (iText 7)

In this iText 7 example, we'll create a text field and we'll add it to a PDF:

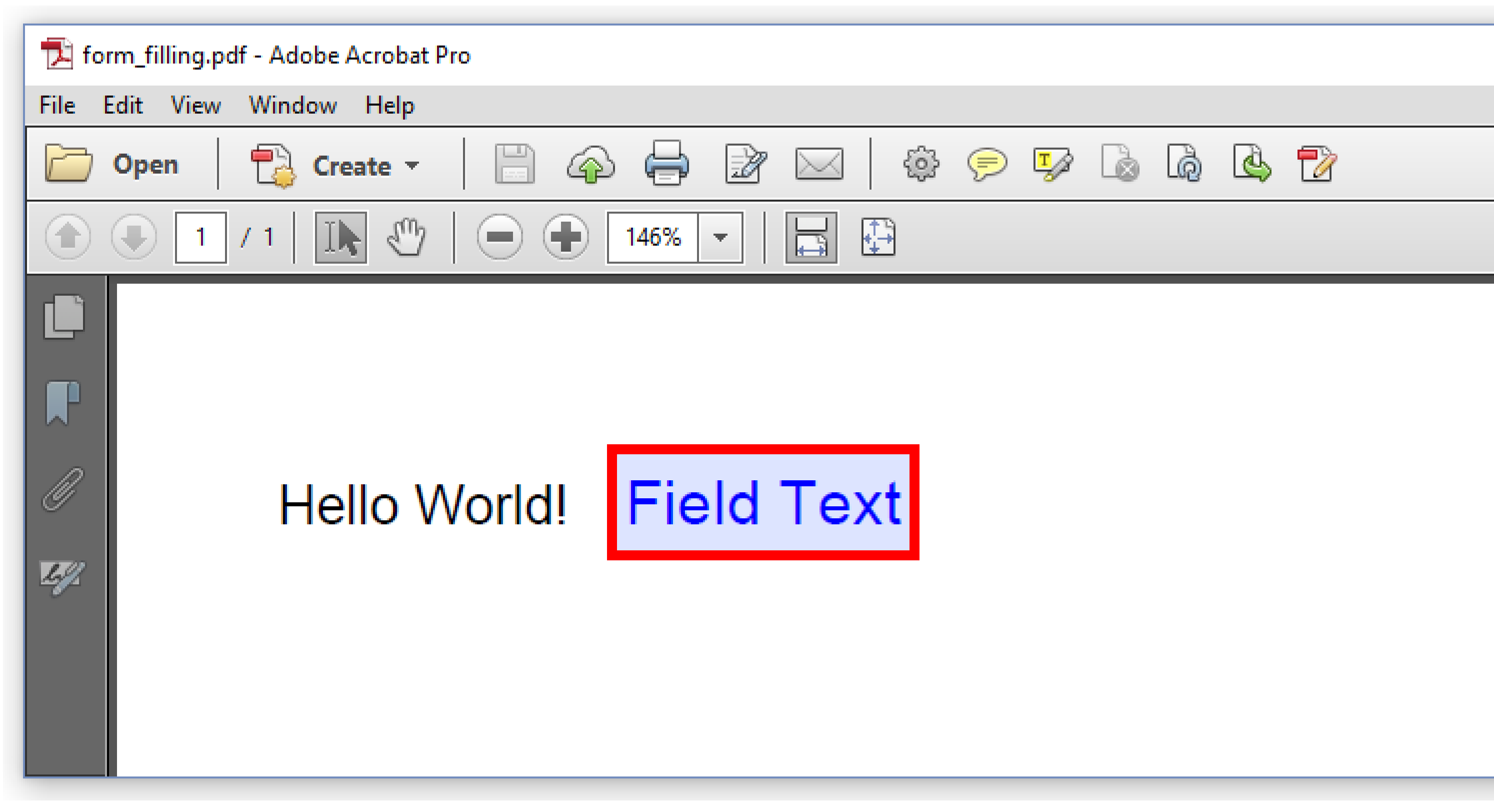

form_creation.pdf - Adobe Acrobat Pro
File Edit View Window Help
Open
Create
1 / 1
146%
Hello World!
Text Field

```
public void manipulatePdf(String src, String dest) throws IOException {
    PdfReader reader = new PdfReader(src);
    PdfDocument pdf = new PdfDocument(reader, new PdfWriter(dest));
    PdfAcroForm form = PdfAcroForm.getAcroForm(pdf, true);
    PdfFormField tf = PdfTextFormField.createText(
        pdf, new Rectangle(110, 780, 70, 26), "text", "Text Field")
        .setBorderColor(Color.BLUE)
        .setBorderWidth(2)
        .setColor(Color.RED)
        .setFontSize(12);
    form.addField(tf);
    pdf.close();
}
```

Source: developers.itextpdf.com

FormFilling.java (iText 5)

In this iText 5 example, we'll change the properties and the value of a text field:


form_filling.pdf - Adobe Acrobat Pro
File Edit View Window Help
Open
Create
1 / 1
146%

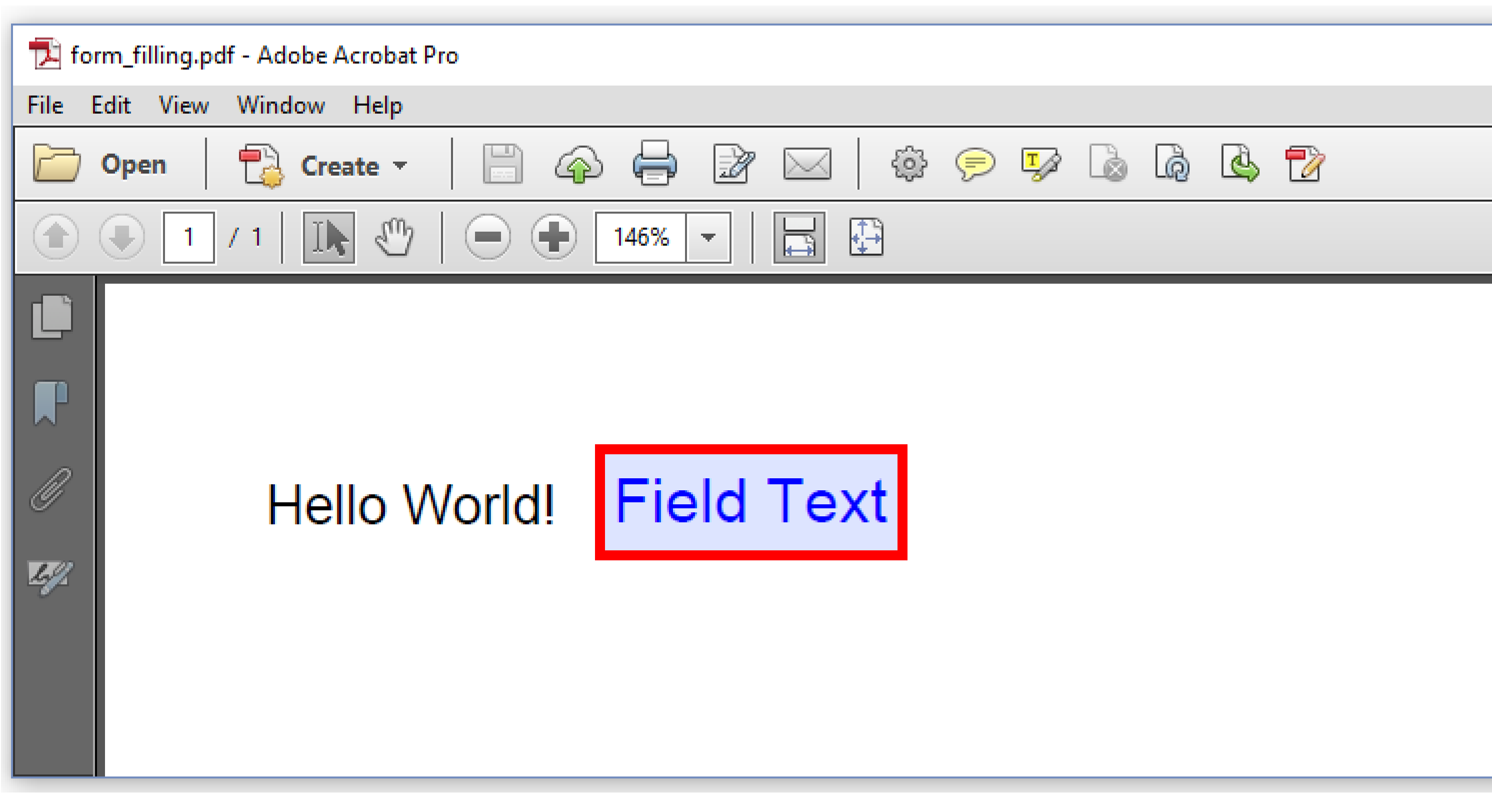
Hello World! Field Text

```
public void manipulatePdf(String src, String dest) throws DocumentException, IOException {
    PdfReader reader = new PdfReader(src);
    PdfStamper stamper = new PdfStamper(reader, new FileOutputStream(dest));
    AcroFields fields = stamper.getAcroFields();
    fields.setFieldProperty("text", "textcolor", BaseColor.BLUE, null);
    fields.setFieldProperty("text", "bordercolor", BaseColor.RED, null);
    fields.setFieldProperty("text", "fontsize", 14, null);
    fields.setField("text", "Field Text");
    stamper.close();
    reader.close();
}
```

Source: developers.itextpdf.com

FormFilling.java (iText 7)

In this iText 7 example, we'll change the properties and the value of a text field:


form_filling.pdf - Adobe Acrobat Pro
File Edit View Window Help
Open
Create
1 / 1
146%

Hello World! Field Text

```
public void manipulatePdf(String src, String dest) throws IOException {
    PdfReader reader = new PdfReader(src);
    PdfDocument pdf = new PdfDocument(reader, new PdfWriter(dest));
    PdfAcroForm form = PdfAcroForm.getAcroForm(pdf, true);
    PdfFormField tf = form.getFormFields().get("text");
    tf.setBorderColor(Color.RED)
        .setColor(Color.BLUE)
        .setFontSize(14)
        .setValue("Field Text");
    pdf.close();
}
```

Source: developers.itextpdf.com

Read Forms: iText 5 vs iText 7 online: https://riptutorial.com/itext/topic/7005/forms--itext-5-vs-itext-7

Chapter 5: Page events (iText 5) versus Event handlers and Renderers (iText 7)

Remarks

In iText 5, we introduced the concept of page events to allow developers to add specific behavior when a document is opened, when a new page is opened, when a page ends, and when a document is closed.

In the documentation, we made it very clear that it was **forbidden** to add content in the `onStartPage()` method; content can only be added in the `onEndPage()` method. We also made it very clear that the `Document` object passed to the page event methods was passed for **read-only** purposes only. It was **forbidden** to use `document.add()` even in the `onEndPage()` method.

Unfortunately, many developers completely ignore the documentation, which led to problems such as:

- "'System.StackOverflowException" in "OnEndPage" event handler
- Getting a Stack Overflow Exception generating a PDF
- iTextSharp line separator disappears with PdfPageEventHelper
- iTextSharp error document has no page before open it
- ...

I can't remember how many times I got agitated because yet another developer posted a duplicate of these questions. People often wonder why they get a harsh answer, but they don't realize that a minimum of effort from their side would have saved everyone, including themselves, plenty of time. All of these questions could have been answered by saying "Read The (you-know-which) Manual."

Another option was a complete overhaul of iText so that these kind of problems can be avoided.

Due to the organic growth of iText, the page event class had also been extended with functionality that was unrelated to page events. It contained *generic chunk* functionality, it registered the start and the end of paragraphs, and so on.

What we fixed in iText 7:

We removed the page event functionality.

For all events with respect to pages, we now implement the `IEventHandler` interface, and we use the `addEventHandler` to add this handler as a `PdfDocumentEvent` to the `PdfDocument`. In the example, we use an `END_PAGE` event, but we could also have used a `START_PAGE` event. It doesn't matter any more whether you add content at the start or at the end. You can read more about this in Handling events; setting viewer preferences and writer properties which is chapter 7 in the iText 7: Building Blocks tutorial.

We improved the building blocks in the sense that we made them more hierarchical (see Before we start: Overview of the classes and interfaces which is the introduction of the iText 7: Building Blocks tutorial). We also introduced a set of Renderer classes, one for each building block, and we allow developers to adapt these renderers so that a building block shows a different behavior when rendered. See for instance the renderer example in Adding AbstractElement objects (part 1) which is chapter 7 in the iText 7: Building Blocks tutorial.

These changes simplify the functionality for developers who don't (want to) know much about PDF and iText, while at the same time offering an abundance of flexibility to those developers who aren't afraid to dig deep into the iText code to create a PDF exactly the way they want it.

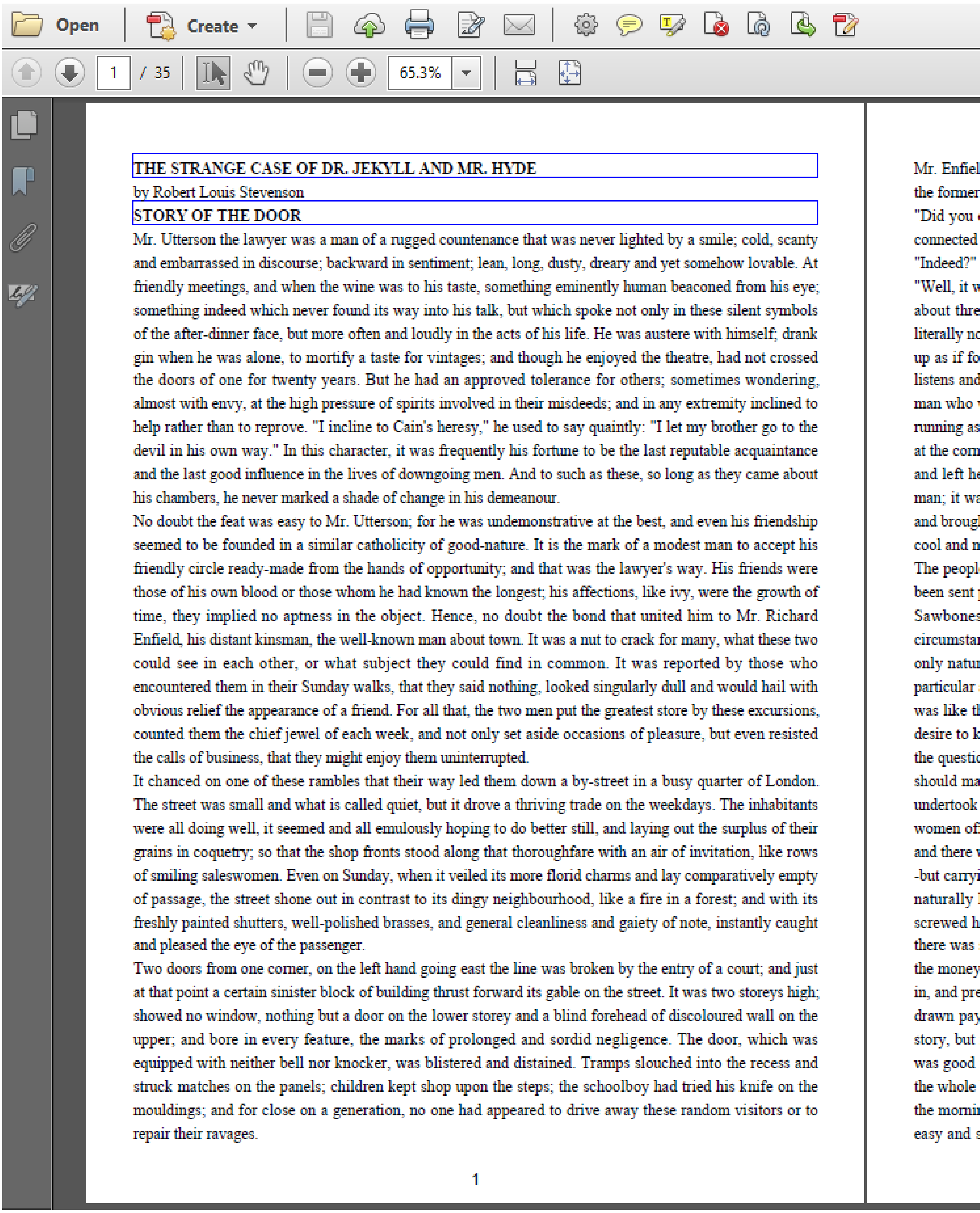
Want to know more? Get the free ebook!

Examples

Text2PdfPageEvents.java (iText 5)

Suppose that we have the following text file: jekyll_hyde.txt

How do we convert it to a PDF that looks like this:



THE STRANGE CASE OF DR. JEKYLL AND MR. HYDE
by Robert Louis Stevenson
STORY OF THE DOOR
Mr. Utterson the lawyer was a man of a rugged countenance that was never lighted by a smile; cold, scanty and embarrassed in discourse; backward in sentiment; lean, long, dusty, dreary and yet somehow lovable. At friendly meetings, and when the wine was to his taste, something eminently human beaconed from his eye; something indeed which never found its way into his talk, but which spoke not only in these silent symbols of the after-dinner face, but more often and loudly in the acts of his life. He was austere with himself; drank gin when he was alone, to mortify a taste for vintages; and though he enjoyed the theatre, had not crossed the doors of one for twenty years. But he had an approved tolerance for others; sometimes wondering, almost with envy, at the high pressure of spirits involved in their misdeeds; and in any extremity inclined to help rather than to reprove. "I incline to Cain's heresy," he used to say quaintly: "I let my brother go to the devil in his own way." In this character, it was frequently his fortune to be the last reputable acquaintance and the last good influence in the lives of downgoing men. And to such as these, so long as they came about his chambers, he never marked a shade of change in his demeanour.
No doubt the feat was easy to Mr. Utterson; for he was undemonstrative at the best, and even his friendship seemed to be founded in a similar catholicity of good-nature. It is the mark of a modest man to accept his friendly circle ready-made from the hands of opportunity; and that was the lawyer's way. His friends were those of his own blood or those whom he had known the longest; his affections, like ivy, were the growth of time, they implied no aptness in the object. Hence, no doubt the bond that united him to Mr. Richard Enfield, his distant kinsman, the well-known man about town. It was a nut to crack for many, what these two could see in each other, or what subject they could find in common. It was reported by those who encountered them in their Sunday walks, that they said nothing, looked singularly dull and would hail with obvious relief the appearance of a friend. For all that, the two men put the greatest store by these excursions, counted them the chief jewel of each week, and not only set aside occasions of pleasure, but even resisted the calls of business, that they might enjoy them uninterrupted.
It chanced on one of these rambles that their way led them down a by-street in a busy quarter of London. The street was small and what is called quiet, but it drove a thriving trade on the weekdays. The inhabitants were all doing well, it seemed and all emulously hoping to do better still, and laying out the surplus of their grains in coquetry; so that the shop fronts stood along that thoroughfare with an air of invitation, like rows of smiling saleswomen. Even on Sunday, when it veiled its more florid charms and lay comparatively empty of passage, the street shone out in contrast to its dingy neighbourhood, like a fire in a forest; and with its freshly painted shutters, well-polished brasses, and general cleanliness and gaiety of note, instantly caught and pleased the eye of the passenger.
Two doors from one corner, on the left hand going east the line was broken by the entry of a court; and just at that point a certain sinister block of building thrust forward its gable on the street. It was two storeys high; showed no window, nothing but a door on the lower storey and a blind forehead of discoloured wall on the upper; and bore in every feature, the marks of prolonged and sordid negligence. The door, which was equipped with neither bell nor knocker, was blistered and distained. Tramps slouched into the recess and struck matches on the panels; children kept shop upon the steps; the schoolboy had tried his knife on the mouldings; and for close on a generation, no one had appeared to drive away these random visitors or to repair their ravages.

1

Note the blue border that is added to the titles, and the page number at the bottom of each page. In iText 5, these elements are added using page events:

```
class MyPageEvents extends PdfPageEventHelper {

    protected float startpos = -1;
    protected boolean title = true;

    public void setTitle(boolean title) {
        this.title = title;
    }

    @Override
    public void onEndPage(PdfWriter writer, Document document) {
        Rectangle pagesize = document.getPageSize();
        ColumnText.showTextAligned(
            writer.getDirectContent(),
            Element.ALIGN_CENTER,
            new Phrase(String.valueOf(writer.getPageNumber())),
            (pagesize.getLeft() + pagesize.getRight()) / 2,
            pagesize.getBottom() + 15,
            0);
        if (startpos != -1)
            onParagraphEnd(writer, document,
                pagesize.getBottom(document.bottomMargin()));
        startpos = pagesize.getTop(document.topMargin());
    }

    @Override
    public void onParagraph(PdfWriter writer, Document document,
        float paragraphPosition) {
        startpos = paragraphPosition;
    }

    @Override
    public void onParagraphEnd(PdfWriter writer, Document document,
        float paragraphPosition) {
        if (!title) return;
        PdfContentByte canvas = writer.getDirectContentUnder();
        Rectangle pagesize = document.getPageSize();
        canvas.saveState();
        canvas.setColorStroke(BaseColor.BLUE);
        canvas.rectangle(
            pagesize.getLeft(document.leftMargin()),
            paragraphPosition - 3,
            pagesize.getWidth() - document.leftMargin() - document.rightMargin(),
            startpos - paragraphPosition);
        canvas.stroke();
        canvas.restoreState();
    }
}
```

We can reuse the code to convert a text file to a PDF from the Text2Pdf.java (iText 5) example, and introduce the page event to the `PdfWriter`:

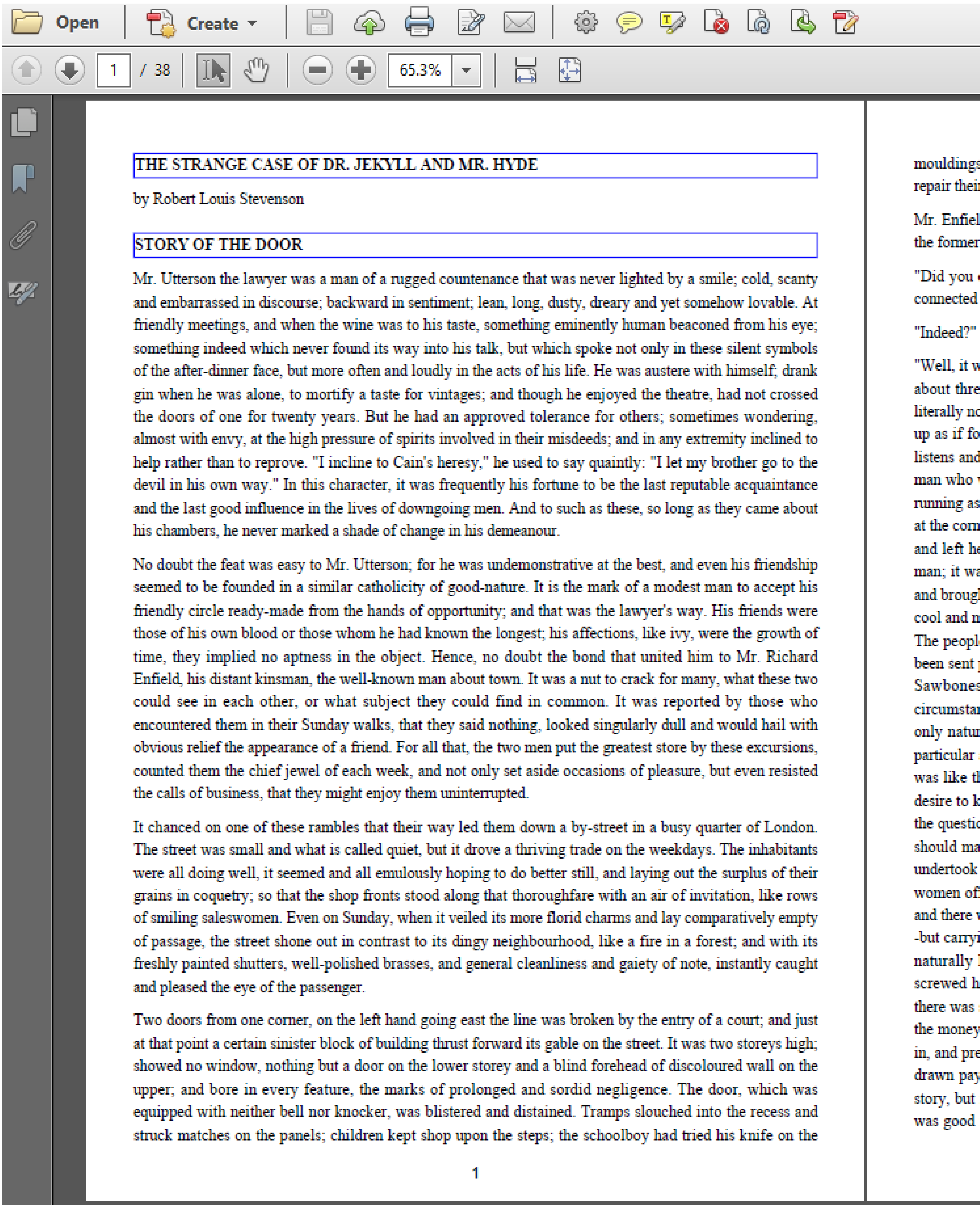
```
public void createPdf(String dest)
throws DocumentException, IOException {
    Document document = new Document();
    PdfWriter writer = PdfWriter.getInstance(document, new FileOutputStream(dest));
    MyPageEvents events = new MyPageEvents();
    writer.setPageEvent(events);
    document.open();
    BufferedReader br = new BufferedReader(new FileReader(TEXT));
    String line;
    Paragraph p;
    Font normal = new Font(FontFamily.TIMES_ROMAN, 12);
    Font bold = new Font(FontFamily.TIMES_ROMAN, 12, Font.BOLD);
    boolean title = true;
    while ((line = br.readLine()) != null) {
        p = new Paragraph(line, title ? bold : normal);
        p.setAlignment(Element.ALIGN_JUSTIFIED);
        events.setTitle(title);
        document.add(p);
        title = line.isEmpty();
    }
    document.close();
}
```

Source: developers.itextpdf.com

Text2PdfPageEvents1.java (iText 7)

Suppose that you have the following text file: jekyll_hyde.txt

How do we convert it to a PDF that looks like this:



THE STRANGE CASE OF DR. JEKYLL AND MR. HYDE

by Robert Louis Stevenson

STORY OF THE DOOR

Mr. Utterson the lawyer was a man of a rugged countenance that was never lighted by a smile; cold, scanty and embarrassed in discourse; backward in sentiment; lean, long, dusty, dreary and yet somehow lovable. At friendly meetings, and when the wine was to his taste, something eminently human beaconed from his eye; something indeed which never found its way into his talk, but which spoke not only in these silent symbols of the after-dinner face, but more often and loudly in the acts of his life. He was austere with himself; drank gin when he was alone, to mortify a taste for vintages; and though he enjoyed the theatre, had not crossed the doors of one for twenty years. But he had an approved tolerance for others; sometimes wondering, almost with envy, at the high pressure of spirits involved in their misdeeds; and in any extremity inclined to help rather than to reprove. "I incline to Cain's heresy," he used to say quaintly: "I let my brother go to the devil in his own way." In this character, it was frequently his fortune to be the last reputable acquaintance and the last good influence in the lives of downgoing men. And to such as these, so long as they came about his chambers, he never marked a shade of change in his demeanour.

No doubt the feat was easy to Mr. Utterson; for he was undemonstrative at the best, and even his friendship seemed to be founded in a similar catholicity of good-nature. It is the mark of a modest man to accept his friendly circle ready-made from the hands of opportunity; and that was the lawyer's way. His friends were those of his own blood or those whom he had known the longest; his affections, like ivy, were the growth of time, they implied no aptness in the object. Hence, no doubt the bond that united him to Mr. Richard Enfield, his distant kinsman, the well-known man about town. It was a nut to crack for many, what these two could see in each other, or what subject they could find in common. It was reported by those who encountered them in their Sunday walks, that they said nothing, looked singularly dull and would hail with obvious relief the appearance of a friend. For all that, the two men put the greatest store by these excursions, counted them the chief jewel of each week, and not only set aside occasions of pleasure, but even resisted the calls of business, that they might enjoy them uninterrupted.

It chanced on one of these rambles that their way led them down a by-street in a busy quarter of London. The street was small and what is called quiet, but it drove a thriving trade on the weekdays. The inhabitants were all doing well, it seemed and all emulously hoping to do better still, and laying out the surplus of their grains in coquetry; so that the shop fronts stood along that thoroughfare with an air of invitation, like rows of smiling saleswomen. Even on Sunday, when it veiled its more florid charms and lay comparatively empty of passage, the street shone out in contrast to its dingy neighbourhood, like a fire in a forest; and with its freshly painted shutters, well-polished brasses, and general cleanliness and gaiety of note, instantly caught and pleased the eye of the passenger.

Two doors from one corner, on the left hand going east the line was broken by the entry of a court; and just at that point a certain sinister block of building thrust forward its gable on the street. It was two storeys high; showed no window, nothing but a door on the lower storey and a blind forehead of discoloured wall on the upper; and bore in every feature, the marks of prolonged and sordid negligence. The door, which was equipped with neither bell nor knocker, was blistered and distained. Tramps slouched into the recess and struck matches on the panels; children kept shop upon the steps; the schoolboy had tried his knife on the

1

mouldings
repair thei

Mr. Enfiel
the former

"Did you
connected

"Indeed?"

"Well, it w
about thre
literally no
up as if fo
listens and
man who
running as
at the corn
and left he
man; it wa
and broug
cool and n
The peopl
been sent
Sawbones
circumsta
only natur
particular
was like t
desire to k
the questio
should ma
undertook
women of
and there v
-but carryi
naturally l
screwed h
there was
the money
in, and pre
drawn pay
story, but
was good

Note the page numbers at the bottom of each page. These are added using an `IEventHandler` implementation:

```
protected class Footer implements IEventHandler {

    @Override
    public void handleEvent(Event event) {
        PdfDocumentEvent docEvent = (PdfDocumentEvent) event;
        PdfDocument pdf = docEvent.getDocument();
        PdfPage page = docEvent.getPage();
        Rectangle pageSize = page.getPageSize();
        PdfCanvas pdfCanvas = new PdfCanvas(
            page.getLastContentStream(), page.getResources(), pdf);
        Canvas canvas = new Canvas(pdfCanvas, pdf, pageSize);
        float x = (pageSize.getLeft() + pageSize.getRight()) / 2;
        float y = pageSize.getBottom() + 15;
        canvas.showTextAligned(
            String.valueOf(pdf.getPageNumber(page)),
            x, y, TextAlignment.CENTER);
    }
}
```

We can reuse the Text2Pdf.java (iText 7) example with only two minor changes:

```
public void createPdf(String dest) throws IOException {
    PdfDocument pdf = new PdfDocument(new PdfWriter(dest));
    pdf.addEventHandler(PdfDocumentEvent.END_PAGE, new Footer());
    Document document = new Document(pdf)
        .setTextAlignment(TextAlignment.JUSTIFIED);
    BufferedReader br = new BufferedReader(new FileReader(TEXT));
    String line;
    PdfFont normal = PdfFontFactory.createFont(FontConstants.TIMES_ROMAN);
    PdfFont bold = PdfFontFactory.createFont(FontConstants.TIMES_BOLD);
    boolean title = true;
    Border border = new SolidBorder(Color.BLUE, 1);
    while ((line = br.readLine()) != null) {
        document.add(new Paragraph(line)
            .setFont(title ? bold : normal)
            .setBorder(title ? border : Border.NO_BORDER));
        title = line.isEmpty();
    }
    document.close();
}
```

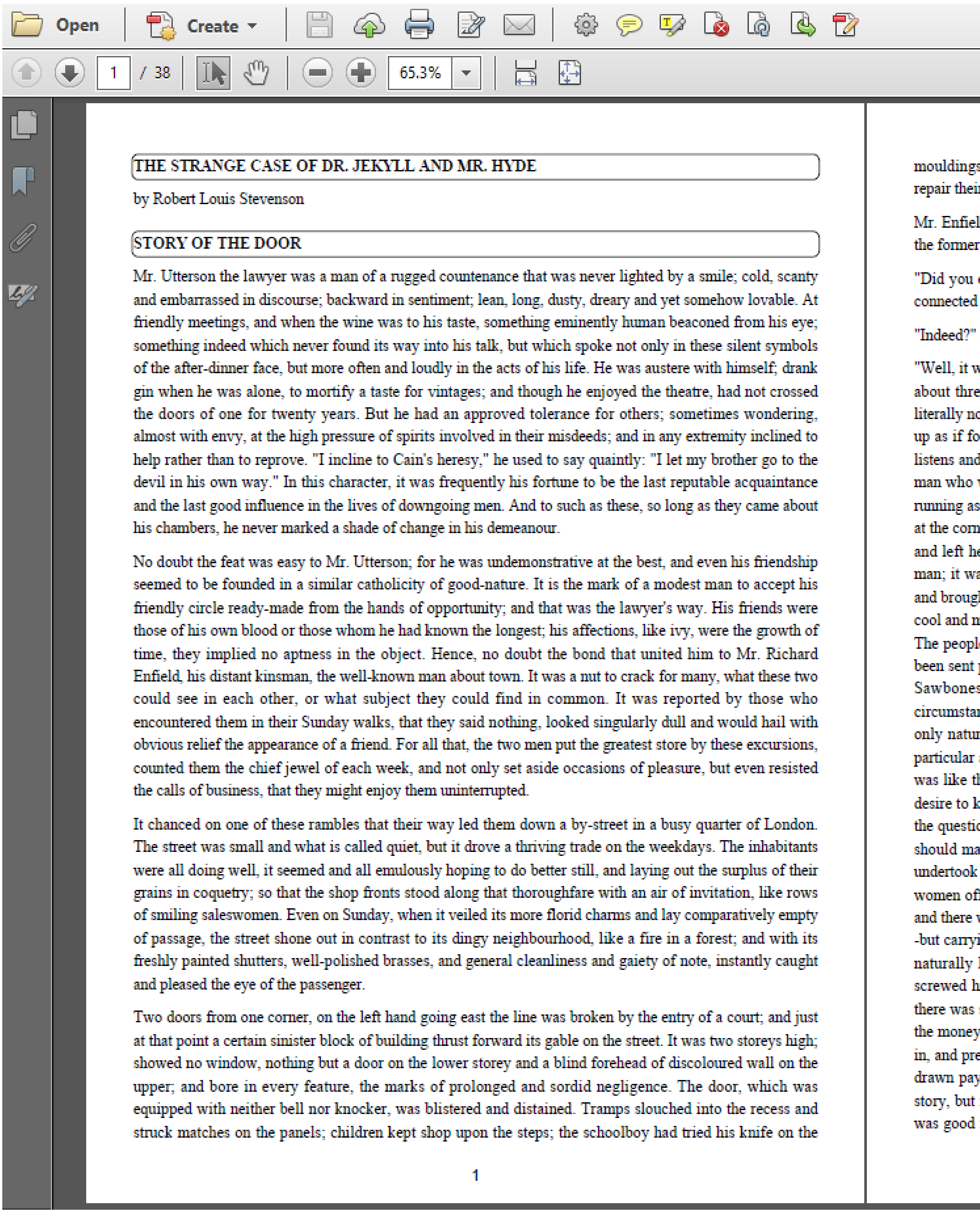
We add an event handler that will trigger the `handleEvent()` method of the `Footer` class every time a page ends. We also define a border for the `Paragraph` objects that are used for a title.

Source: developers.itextpdf.com and the iText 7: Building Blocks tutorial.

Text2PdfPageEvents2.java

Suppose that you have the following text file: jekyll_hyde.txt

How do we convert it to a PDF that looks like this:



THE STRANGE CASE OF DR. JEKYLL AND MR. HYDE

by Robert Louis Stevenson

STORY OF THE DOOR

Mr. Utterson the lawyer was a man of a rugged countenance that was never lighted by a smile; cold, scanty and embarrassed in discourse; backward in sentiment; lean, long, dusty, dreary and yet somehow lovable. At friendly meetings, and when the wine was to his taste, something eminently human beaconed from his eye; something indeed which never found its way into his talk, but which spoke not only in these silent symbols of the after-dinner face, but more often and loudly in the acts of his life. He was austere with himself; drank gin when he was alone, to mortify a taste for vintages; and though he enjoyed the theatre, had not crossed the doors of one for twenty years. But he had an approved tolerance for others; sometimes wondering, almost with envy, at the high pressure of spirits involved in their misdeeds; and in any extremity inclined to help rather than to reprove. "I incline to Cain's heresy," he used to say quaintly: "I let my brother go to the devil in his own way." In this character, it was frequently his fortune to be the last reputable acquaintance and the last good influence in the lives of downgoing men. And to such as these, so long as they came about his chambers, he never marked a shade of change in his demeanour.

No doubt the feat was easy to Mr. Utterson; for he was undemonstrative at the best, and even his friendship seemed to be founded in a similar catholicity of good-nature. It is the mark of a modest man to accept his friendly circle ready-made from the hands of opportunity; and that was the lawyer's way. His friends were those of his own blood or those whom he had known the longest; his affections, like ivy, were the growth of time, they implied no aptness in the object. Hence, no doubt the bond that united him to Mr. Richard Enfield, his distant kinsman, the well-known man about town. It was a nut to crack for many, what these two could see in each other, or what subject they could find in common. It was reported by those who encountered them in their Sunday walks, that they said nothing, looked singularly dull and would hail with obvious relief the appearance of a friend. For all that, the two men put the greatest store by these excursions, counted them the chief jewel of each week, and not only set aside occasions of pleasure, but even resisted the calls of business, that they might enjoy them uninterrupted.

It chanced on one of these rambles that their way led them down a by-street in a busy quarter of London. The street was small and what is called quiet, but it drove a thriving trade on the weekdays. The inhabitants were all doing well, it seemed and all emulously hoping to do better still, and laying out the surplus of their grains in coquetry; so that the shop fronts stood along that thoroughfare with an air of invitation, like rows of smiling saleswomen. Even on Sunday, when it veiled its more florid charms and lay comparatively empty of passage, the street shone out in contrast to its dingy neighbourhood, like a fire in a forest; and with its freshly painted shutters, well-polished brasses, and general cleanliness and gaiety of note, instantly caught and pleased the eye of the passenger.

Two doors from one corner, on the left hand going east the line was broken by the entry of a court; and just at that point a certain sinister block of building thrust forward its gable on the street. It was two storeys high; showed no window, nothing but a door on the lower storey and a blind forehead of discoloured wall on the upper; and bore in every feature, the marks of prolonged and sordid negligence. The door, which was equipped with neither bell nor knocker, was blistered and distained. Tramps slouched into the recess and struck matches on the panels; children kept shop upon the steps; the schoolboy had tried his knife on the

1

Note that this is very similar to what we had before, but the border of the titles now has rounded corners. We created a custom `ParagraphRenderer` to achieve this, and we created a `TitleParagraph` object that uses that renderer:

```
public class TitleParagraph extends Paragraph {

    public TitleParagraph(String line) {
        super(line);
        try {
            setFont(PdfFontFactory.createFont(FontConstants.TIMES_BOLD));
        }
        catch (IOException ioe) {
        }
    }

    @Override
    protected IRenderer makeNewRenderer() {
        return new ParagraphRenderer(this) {
            @Override
            public void drawBorder(DrawContext drawContext) {
                Rectangle occupiedAreaBBox = getOccupiedAreaBBox();
                float[] margins = getMargins();
                Rectangle rectangle = applyMargins(occupiedAreaBBox, margins, false);
                PdfCanvas canvas = drawContext.getCanvas();
                canvas.roundRectangle(rectangle.getX() - 1, rectangle.getY() - 1,
                rectangle.getWidth() + 2, rectangle.getHeight() + 2, 5).stroke();
            }
        };
    }
}
```

Our code to convert text to PDF is very simple now. We no longer have to set the font to bold for the titles and we no longer have to define a border:

```
public void createPdf(String dest) throws IOException {
    PdfDocument pdf = new PdfDocument(new PdfWriter(dest));
    pdf.addEventHandler(PdfDocumentEvent.END_PAGE, new Footer());
    Document document = new Document(pdf)
        .setTextAlignment(TextAlignment.JUSTIFIED);
    BufferedReader br = new BufferedReader(new FileReader(TEXT));
    String line;
    PdfFont normal = PdfFontFactory.createFont(FontConstants.TIMES_ROMAN);
    boolean title = true;
    Border border = new SolidBorder(Color.BLUE, 1);
    while ((line = br.readLine()) != null) {
        if (title)
            document.add(new TitleParagraph(line));
        else
            document.add(new Paragraph(line).setFont(normal));
        title = line.isEmpty();
    }
    document.close();
}
```

Source: developers.itextpdf.com and the iText 7: Building Blocks tutorial.

Read Page events (iText 5) versus Event handlers and Renderers (iText 7) online: https://riptutorial.com/itext/topic/5790/page-events--itext-5--versus-event-handlers-and-renderers--itext-7-

Chapter 6: Pdf Creation: iText 5 versus iText 7

Remarks

In the original design for iText, it was possible to create a high-level `Document` object, and then have different `DocListener` objects listening to that `Document` object. This was achieved by using different writers: a `PdfWriter`, an `HTMLWriter`, and an `RtfWriter`. When using a `PdfWriter`, a `PdfDocument` was created internally. This low-level class took care of all PDF-related structures. More or less the same was true for the other formats.

Over the years, iText specialized and it became a pure PDF library. The creation of HTML and RTF was abandoned, hence it was no longer necessary to create a `Document` before creating a `PdfWriter`, but we had to stick to the original architecture because we weren't ready to break the API.

Over the years, we added more and more PDF functionality to iText, and the fact that `PdfDocument` was a class *for internal use only* became problematic. We used workarounds so that we could introduce new PDF features that belonged in the `PdfDocument` class up until the point that we reached the ceiling of what we considered acceptable as workarounds.

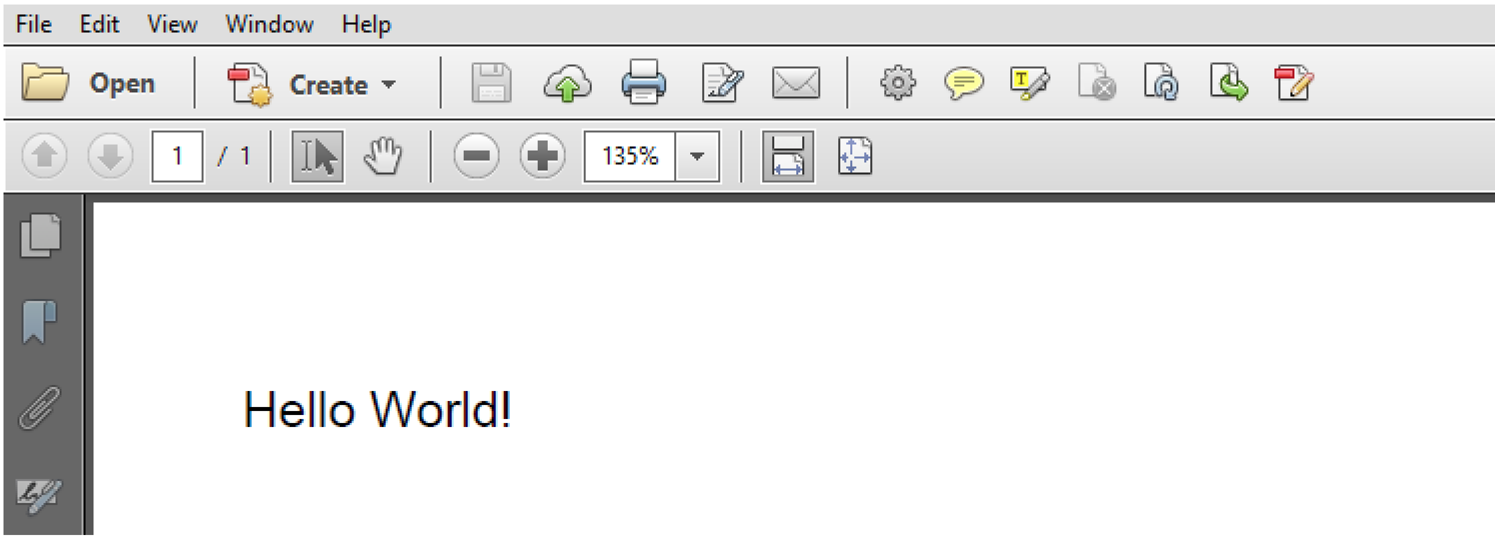
That's when we decided to rewrite iText from scratch and to create a completely new architecture for iText. Now we have a clear distinction between the `PdfDocument` (for low-level operations) and the `Document` (for high-level functionality). We no longer have to open the document, and if we use the *try-with-resources* approach, we don't even have to close it ourselves.

Want to know more? Get the free ebook!

Examples

HelloWorld.java (iText 5)

Suppose that we want to create a simple Hello World document:



Hello World!

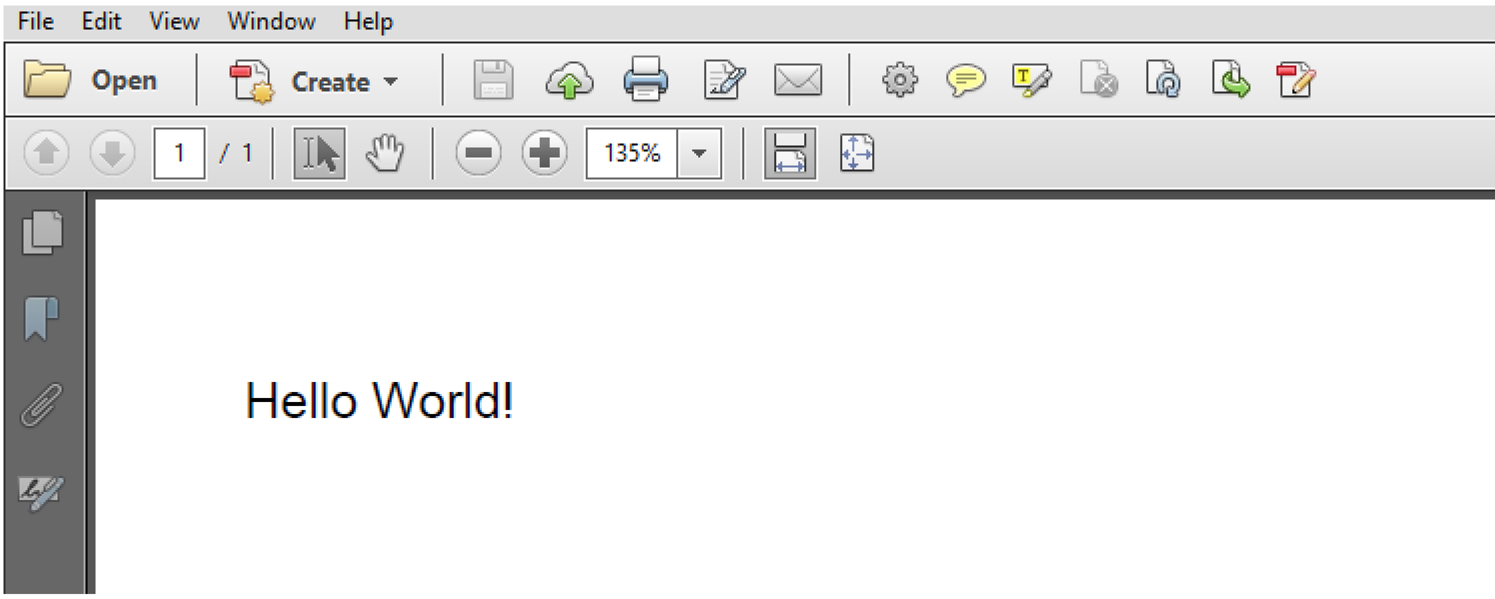
In iText 5, this would be done like this:

```
public void createPdf(String dest)
    throws DocumentException, IOException {
    Document document = new Document();
    PdfWriter.getInstance(
        document, new FileOutputStream(dest));
    document.open();
    document.add(new Paragraph("Hello World!"));
    document.close();
}
```

Source: developers.itextpdf.com

HelloWorld1.java and HelloWorld2.java (iText 7)

Suppose that we wanted to create a simple Hello World document:



Hello World!

In iText 7, we could do that like this:

```
public void createPdf(String dest) throws IOException {
    PdfDocument pdf = new PdfDocument(new PdfWriter(dest));
    Document document = new Document(pdf);
    document.add(new Paragraph("Hello World!"));
    document.close();
}
```

Or, we could even do it like this:

```
public void createPdf(String dest) throws IOException {
    PdfDocument pdf = new PdfDocument(new PdfWriter(dest));
    try (Document document = new Document(pdf)) {
        document.add(new Paragraph("Hello World!"));
    }
}
```

Source: developers.itextpdf.com and the iText 7: Building Blocks tutorial.

Read Pdf Creation: iText 5 versus iText 7 online: https://riptutorial.com/itext/topic/5792/pdf-creation--itext-5-versus-itext-7

Chapter 7: Q & A about versions

Introduction

Some frequently asked questions about the version numbers in iText.

Remarks

Why do the version numers jump from 2 to 5, and from 5 to 7? There are several reasons for skipping version numbers. In 2009, the version number of iText (Java) and iTextSharp (C#) were not in sync. The Java version was at version 2.1.7; the C# version was at version 4.1.6. A decision was taken to move to Java 5 for the Java version and to harmonize the version numbers of iText and iTextSharp.

What's the deal with iText 4? Third parties have released iText 4.2.z versions, but those aren't official versions. They aren't supported by anyone and shouldn't be used because no one really knows what's inside.

Is iText 7 backward compatible? iText 7 is a totally new version of iText, written from scratch by the iText team. Backward compatibility is broken in favor of a more intuitive interface. The Java version of iText has moved from Java 5 to Java 7, which was one of the reasons to jump from iText 5 to iText 7.

Examples

itext 2 vs iText 5 vs iText 7

iText 2 and earlier:

import com.lowagie.text.*;

iText 5:

import com.itextpdf.text.*;

iText 7:

import com.itextpdf.kernel.*;

import com.itextpdf.layout.*;

...

Read Q & A about versions online: https://riptutorial.com/itext/topic/9945/q---a-about-versions

Chapter 8: Styles: iText 5 versus iText 7

Remarks

Creating a document in which you have to switch between styles frequently tends to be tedious in iText 5. You need to create a lot of `Chunk` objects and you always have to make a trade-off between applying the styles directly to every new `Chunk` or creating a helper method that creates the `Chunk` for you.

What we fixed in iText 7:

It is now possible to chain methods. The `setFont()`, `setFontSize()`, `addStyle()`, and other methods all return the object on which they are invoked. Adding a `Paragraph` involving different styles can now be done in one line:

```
document.add(
    new Paragraph()
        .add("In this example, named ")
        .add(new Text("HelloWorldStyles").addStyle(style))
        .add(", we experiment with some text in ")
        .add(new Text("code style").addStyle(style))
        .add("."));
```

Using the `Style` object, you can now also apply different properties (font, font color, background color, font size,...) in one go with the `addStyle()` method.

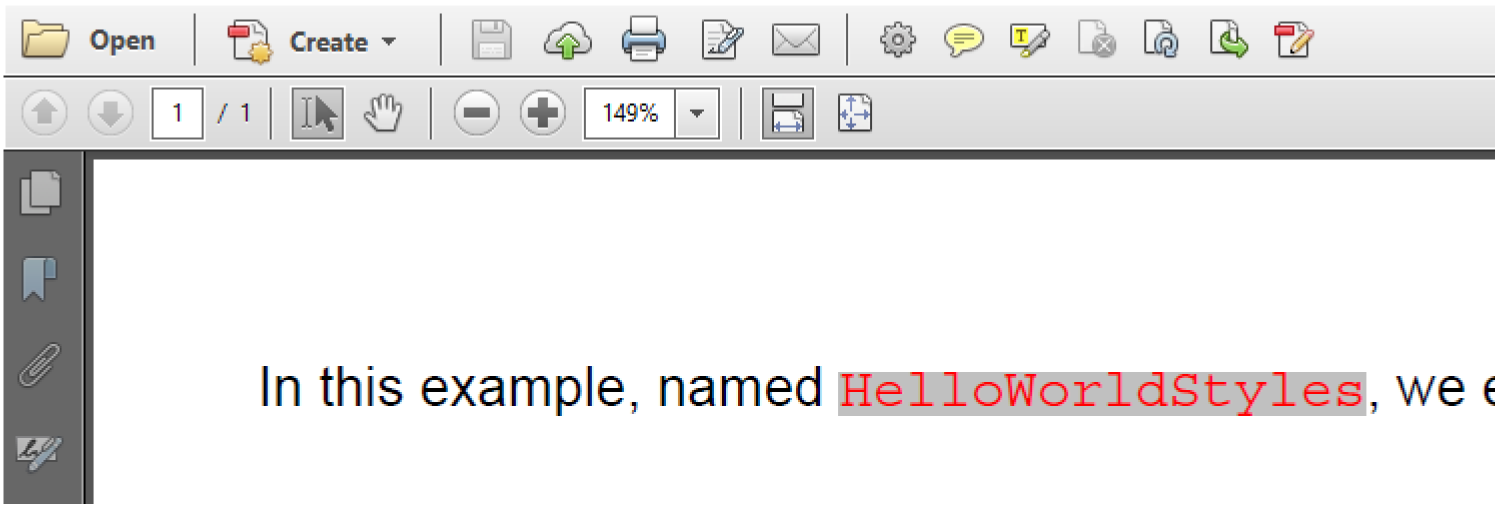
Want to know more?

Read Introducing the PdfFont class which is chapter 1 in the iText 7: Building Blocks tutorial. Get the free ebook!

Examples

HelloWorldStyles.java (iText 5)

In this iText 5 example, we need to switch between different styles in the same document:


In this example, named HelloWorldStyles, we e

The best way to do this in iText 5, is to create a convenience method that creates a `Chunk` in the style that needs to be used frequently; see the `createBgChunk()` method:

```
public Chunk createBgChunk(String s, Font font) {
    Chunk chunk = new Chunk(s, font);
    chunk.setBackground(BaseColor.LIGHT_GRAY);
    return chunk;
}
```

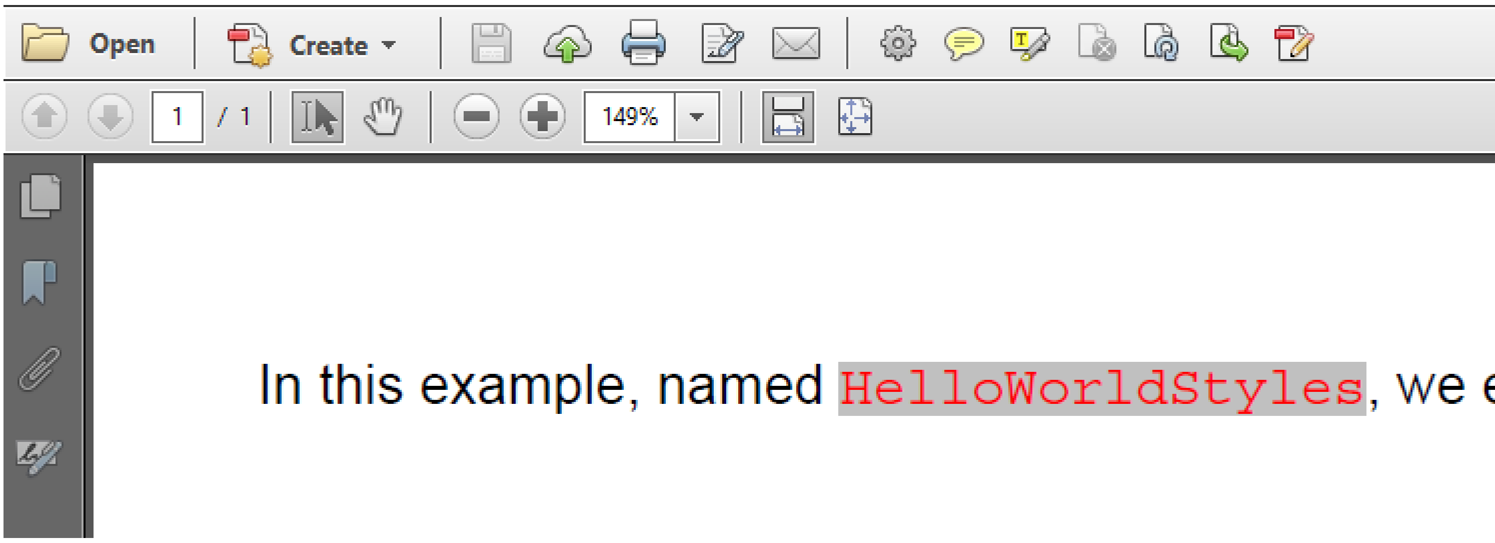
We can now use this method in the code that creates the PDF:

```
public void createPdf(String dest)
throws DocumentException, IOException {
    Document document = new Document();
    PdfWriter.getInstance(document, new FileOutputStream(dest));
    document.open();
    Font code = new Font(FontFamily.COURIER, 12, Font.NORMAL, BaseColor.RED);
    Paragraph p = new Paragraph("In this example, named ");
    p.add(createBgChunk("HelloWorldStyles", code));
    p.add(", we experiment with some text in ");
    p.add(createBgChunk("code style", code));
    p.add(".");
    document.add(p);
    document.close();
}
```

Source: developers.itextpdf.com

HelloWorldStyles.java (iText 7)

In this iText 7 example, we need to switch between different styles in the same document:


In this example, named HelloWorldStyles, we e

The best way to achieve this in iText 7, is to create a `Style` object, and to apply that `Style` to a `Text` object:

```
public void createPdf(String dest) throws IOException {
    PdfDocument pdf = new PdfDocument(new PdfWriter(dest));
    PdfFont code = PdfFontFactory.createFont(FontConstants.COURIER);
    Style style = new Style()
        .setFont(code)
        .setFontSize(12)
        .setFontColor(Color.RED)
        .setBackgroundColor(Color.LIGHT_GRAY);
    try (Document document = new Document(pdf)) {
        document.add(
            new Paragraph()
                .add("In this example, named ")
                .add(new Text("HelloWorldStyles").addStyle(style))
                .add(", we experiment with some text in ")
                .add(new Text("code style").addStyle(style))
                .add("."));
    }
}
```

Source: developers.itextpdf.com and the iText 7: Building Blocks tutorial.

Read Styles: iText 5 versus iText 7 online: https://riptutorial.com/itext/topic/5782/styles--itext-5-versus-itext-7

Chapter 9: Tables: iText 5 versus iText 7

Remarks

The iText 5 class names `PdfPTable` and `PdfPCell` were chosen because we already had classes named `Table` and `Cell` to create table and cell objects at the highest programming level. There was also a class named `PdfTable` to be used by iText internally. Those classes had a lot of flaws and they were deprecated in favor of `PdfPTable` and `PdfPCell`. They have been removed a long time ago.

Over the years, `PdfPTable` and `PdfPCell` also received some criticism from users. For instance: users didn't understand the difference between **text mode** and **composite** mode.

Text mode is used when you create a `PdfPCell` like this:

```
cell = new PdfPCell(new Phrase("Cell with rowspan 2"));
```

In this case, you define properties like the horizontal alignment on the level of the `PdfPCell`.

Composite mode kicks in the moment you use the `addElement()` method:

```
cell = new PdfPCell();
cell.addElement(new Phrase("Cell 1.2"));
```

In this case, some properties defined at the level of the `PdfPCell` (such as the horizontal alignment) are ignored. The horizontal alignment is to be defined at the level of the elements added to the cell. For instance: if you want to create a cell in which different paragraphs need to have a different horizontal alignment, you will switch to composite mode.

If you look at the screen shot of the table created with the iText 5 example, you will notice that the cells with content **Cell 1.1** (added in text mode) and **Cell 1.2** (added in composite mode) are aligned quite differently.

In answer to the criticism on the odd alignment, we introduced methods to use ascender and descender information. We use these methods for the cells with content **Cell 2.1** (added in text mode) and **Cell 2.2** (added in composite mode). We also introduced a padding of 5 for these cells.

Now the result is much better.

What we fixed in iText 7:

Since we created iText 7 from scratch, we had no legacy classes with names we couldn't reuse. We introduced a new `Table` and a new `Cell` class.

There is no more text mode and no more composite mode. A `Cell` is created either without parameters, or with parameters that define the rowspan and the colspan. All content is added the same way: using the `add()` method.

Our customers were also asking to provide a means to distinguish a margin and a padding. In the iText 7 example, we added a gray background to show the difference. In the cell with content **Cell 2.1**, we define a margin of 5 user units. The default padding is 2. In the cell with content **Cell 2.2**, we define a padding of 5 user units, the default margin in 0.

As you can tell from the screen shots, the cells are rendered quite nicely. We didn't have to use methods to set the ascender or descender. The default behavior is much closer to the behavior a developer would expect.

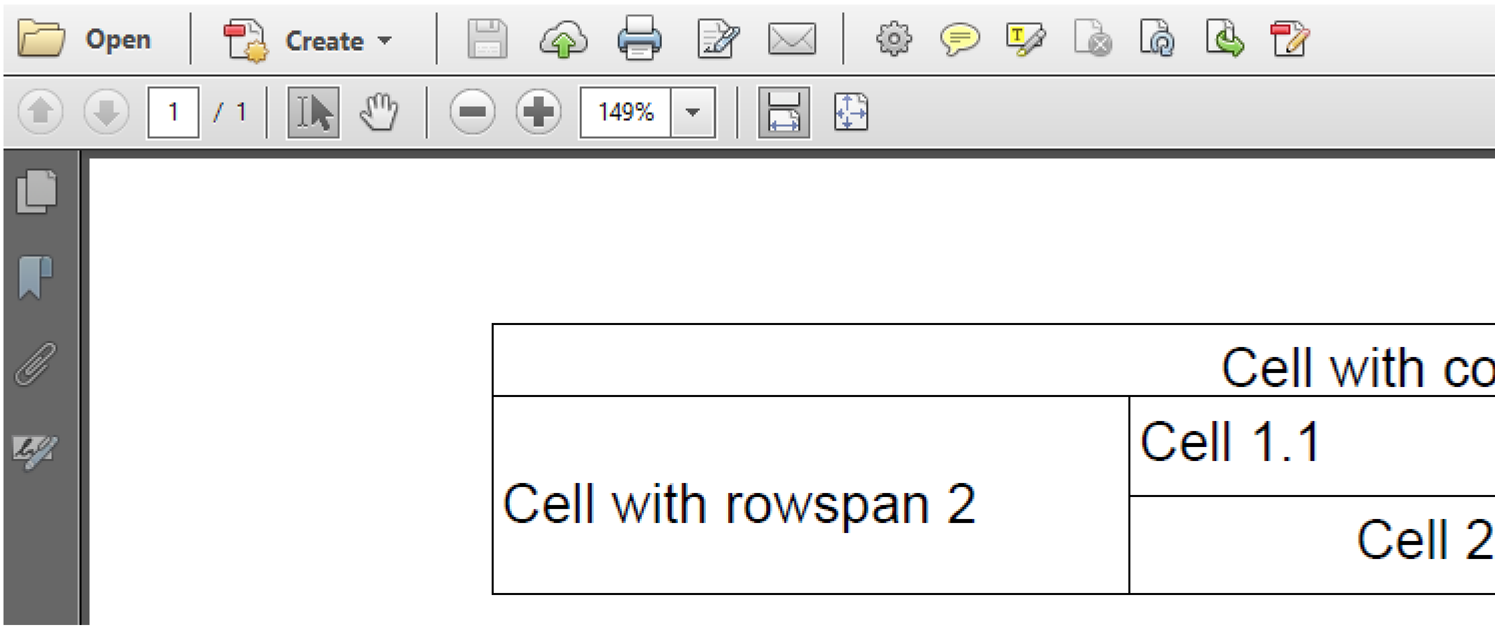
Want to know more about tables and cells in iText 7?

Read Adding AbstractElement objects (part 2) which is chapter 5 in the iText 7: Building Blocks tutorial. Get the free ebook!

Examples

HelloWorldTable.java (iText 5)

In this example, we'll create the following table using iText 5:



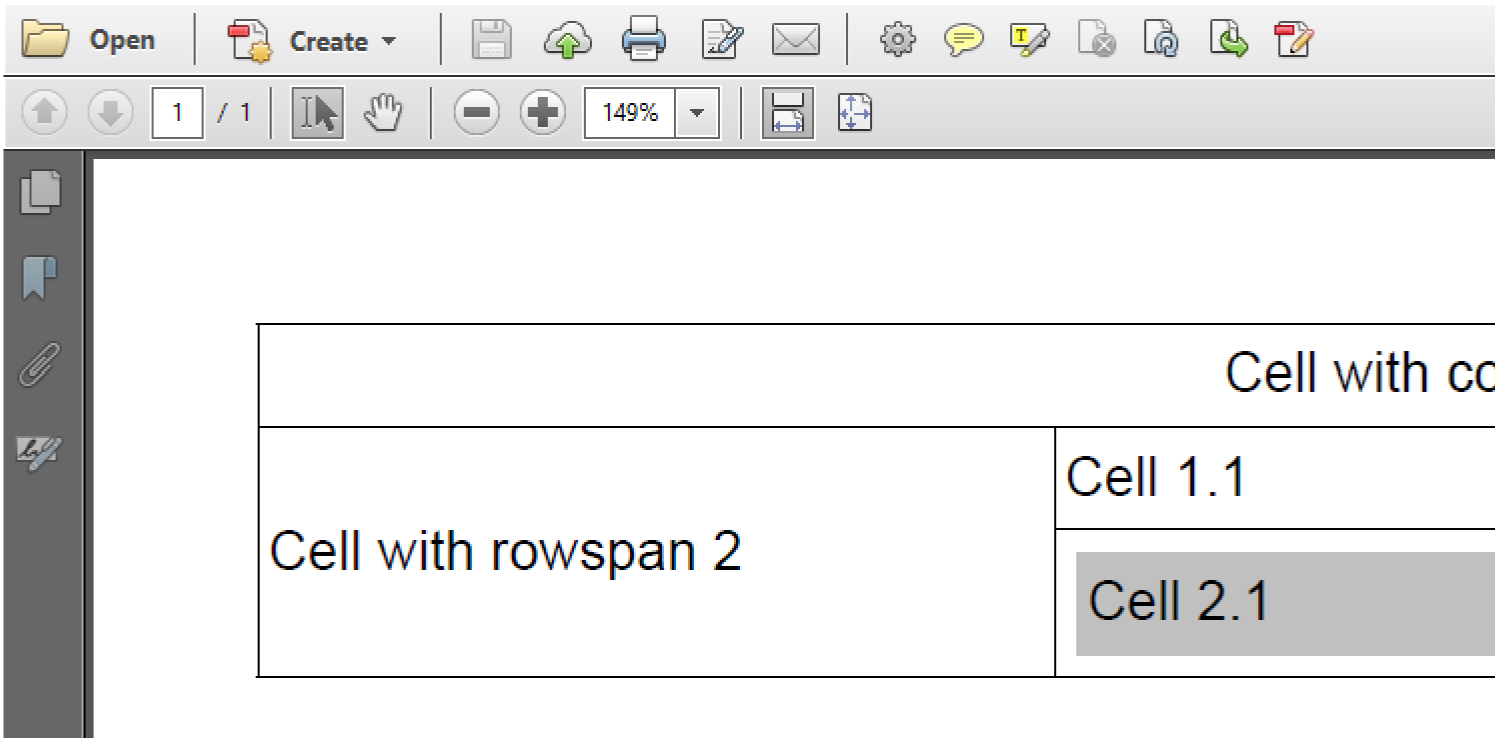
We need the `PdfPTable` and `PdfPCell` class to achieve this:

```
public void createPdf(String dest) throws IOException, DocumentException {
    Document document = new Document();
    PdfWriter.getInstance(document, new FileOutputStream(dest));
    document.open();
    PdfPTable table = new PdfPTable(3);
    PdfPCell cell = new PdfPCell(new Phrase("Cell with colspan 3"));
    cell.setColspan(3);
    cell.setHorizontalAlignment(Element.ALIGN_CENTER);
    table.addCell(cell);
    cell = new PdfPCell(new Phrase("Cell with rowspan 2"));
    cell.setRowspan(2);
    cell.setVerticalAlignment(Element.ALIGN_MIDDLE);
    table.addCell(cell);
    table.addCell("Cell 1.1");
    cell = new PdfPCell();
    cell.addElement(new Phrase("Cell 1.2"));
    table.addCell(cell);
    cell = new PdfPCell(new Phrase("Cell 2.1"));
    cell.setPadding(5);
    cell.setUseAscender(true);
    cell.setUseDescender(true);
    cell.setHorizontalAlignment(Element.ALIGN_CENTER);
    table.addCell(cell);
    cell = new PdfPCell();
    cell.setPadding(5);
    cell.setUseAscender(true);
    cell.setUseDescender(true);
    Paragraph p = new Paragraph("Cell 2.2");
    p.setAlignment(Element.ALIGN_CENTER);
    cell.addElement(p);
    table.addCell(cell);
    document.add(table);
    document.close();
}
```

Source: developers.itextpdf.com

HelloWorldTable.java (iText 7)

In this example, we'll create the following table using iText 7:



Cell with co

Cell with rowspan 2

Cell 1.1

Cell 2.1

We'll need the `Table` and `Cell` class to achieve this:

```
public void createPdf(String dest) throws IOException {
    PdfDocument pdf = new PdfDocument(new PdfWriter(dest));
    try (Document document = new Document(pdf)) {
        Table table = new Table(3);
        Cell cell = new Cell(1, 3)
            .setTextAlignment(TextAlignment.CENTER)
            .add("Cell with colspan 3");
        table.addCell(cell);
        cell = new Cell(2, 1)
            .add("Cell with rowspan 2")
            .setVerticalAlignment(VerticalAlignment.MIDDLE);
        table.addCell(cell);
        table.addCell("Cell 1.1");
        table.addCell(new Cell().add("Cell 1.2"));
        table.addCell(new Cell()
            .add("Cell 2.1")
            .setBackgroundColor(Color.LIGHT_GRAY)
            .setMargin(5));
        table.addCell(new Cell()
            .add("Cell 1.2")
            .setBackgroundColor(Color.LIGHT_GRAY)
            .setPadding(5));
        document.add(table);
    }
}
```

Source: developers.itextpdf.com and the iText 7: Building Blocks tutorial.

Read Tables: iText 5 versus iText 7 online: https://riptutorial.com/itext/topic/5784/tables--itext-5-versus-itext-7

Chapter 10: Text to PDF: iText 5 versus iText 7

Remarks

The code to convert a plain text file to a PDF document is pretty simple whether you use iText 5 or iText 7. In iText 7, you have the advantage that you can define the alignment at the level of the document. In iText 5, you have to set the alignment for every separate `Paragraph` object.

To understand the real difference between iText 5 and iText 7 in this pair of examples, we have to take a look at the resulting PDF. In iText 5, we end up with 35 pages of text. In iText 7, we have the same text distributed over 38 pages.

The text is easier to read when created by iText 7 because different defaults are used when creating the layout. You could get the same result from iText 5 code, but then you'd have to change some values with respect to spacing.

In iText 7, the default values were chosen based on 16 years of experience with iText. This way, you get a better result with less code.

Want to know more?

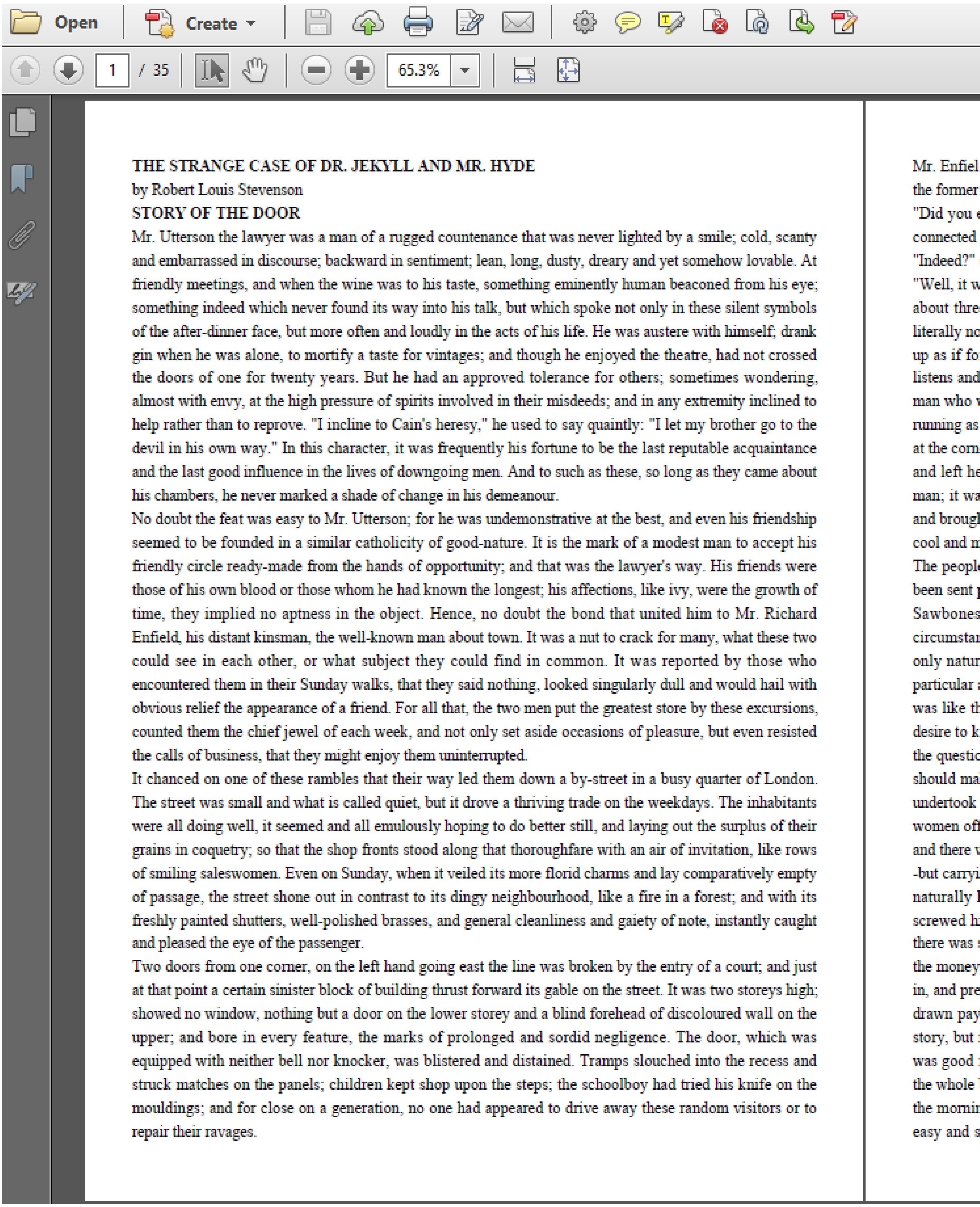
Read Working with the RootElement which is chapter 5 in the iText 7: Building Blocks tutorial. Get the free ebook!

Examples

Text2Pdf.java (iText 5)

Suppose that we have the following text file: jekyll_hyde.txt

How do we convert it to a PDF that looks like this:



THE STRANGE CASE OF DR. JEKYLL AND MR. HYDE
by Robert Louis Stevenson
STORY OF THE DOOR
Mr. Utterson the lawyer was a man of a rugged countenance that was never lighted by a smile; cold, scanty and embarrassed in discourse; backward in sentiment; lean, long, dusty, dreary and yet somehow lovable. At friendly meetings, and when the wine was to his taste, something eminently human beaconed from his eye; something indeed which never found its way into his talk, but which spoke not only in these silent symbols of the after-dinner face, but more often and loudly in the acts of his life. He was austere with himself; drank gin when he was alone, to mortify a taste for vintages; and though he enjoyed the theatre, had not crossed the doors of one for twenty years. But he had an approved tolerance for others; sometimes wondering, almost with envy, at the high pressure of spirits involved in their misdeeds; and in any extremity inclined to help rather than to reprove. "I incline to Cain's heresy," he used to say quaintly: "I let my brother go to the devil in his own way." In this character, it was frequently his fortune to be the last reputable acquaintance and the last good influence in the lives of downgoing men. And to such as these, so long as they came about his chambers, he never marked a shade of change in his demeanour.
No doubt the feat was easy to Mr. Utterson; for he was undemonstrative at the best, and even his friendship seemed to be founded in a similar catholicity of good-nature. It is the mark of a modest man to accept his friendly circle ready-made from the hands of opportunity; and that was the lawyer's way. His friends were those of his own blood or those whom he had known the longest; his affections, like ivy, were the growth of time, they implied no aptness in the object. Hence, no doubt the bond that united him to Mr. Richard Enfield, his distant kinsman, the well-known man about town. It was a nut to crack for many, what these two could see in each other, or what subject they could find in common. It was reported by those who encountered them in their Sunday walks, that they said nothing, looked singularly dull and would hail with obvious relief the appearance of a friend. For all that, the two men put the greatest store by these excursions, counted them the chief jewel of each week, and not only set aside occasions of pleasure, but even resisted the calls of business, that they might enjoy them uninterrupted.
It chanced on one of these rambles that their way led them down a by-street in a busy quarter of London. The street was small and what is called quiet, but it drove a thriving trade on the weekdays. The inhabitants were all doing well, it seemed and all emulously hoping to do better still, and laying out the surplus of their grains in coquetry; so that the shop fronts stood along that thoroughfare with an air of invitation, like rows of smiling saleswomen. Even on Sunday, when it veiled its more florid charms and lay comparatively empty of passage, the street shone out in contrast to its dingy neighbourhood, like a fire in a forest; and with its freshly painted shutters, well-polished brasses, and general cleanliness and gaiety of note, instantly caught and pleased the eye of the passenger.
Two doors from one corner, on the left hand going east the line was broken by the entry of a court; and just at that point a certain sinister block of building thrust forward its gable on the street. It was two storeys high; showed no window, nothing but a door on the lower storey and a blind forehead of discoloured wall on the upper; and bore in every feature, the marks of prolonged and sordid negligence. The door, which was equipped with neither bell nor knocker, was blistered and distained. Tramps slouched into the recess and struck matches on the panels; children kept shop upon the steps; the schoolboy had tried his knife on the mouldings; and for close on a generation, no one had appeared to drive away these random visitors or to repair their ravages.

When using iText 5, we'd use the following code:

```
public void createPdf(String dest)
```

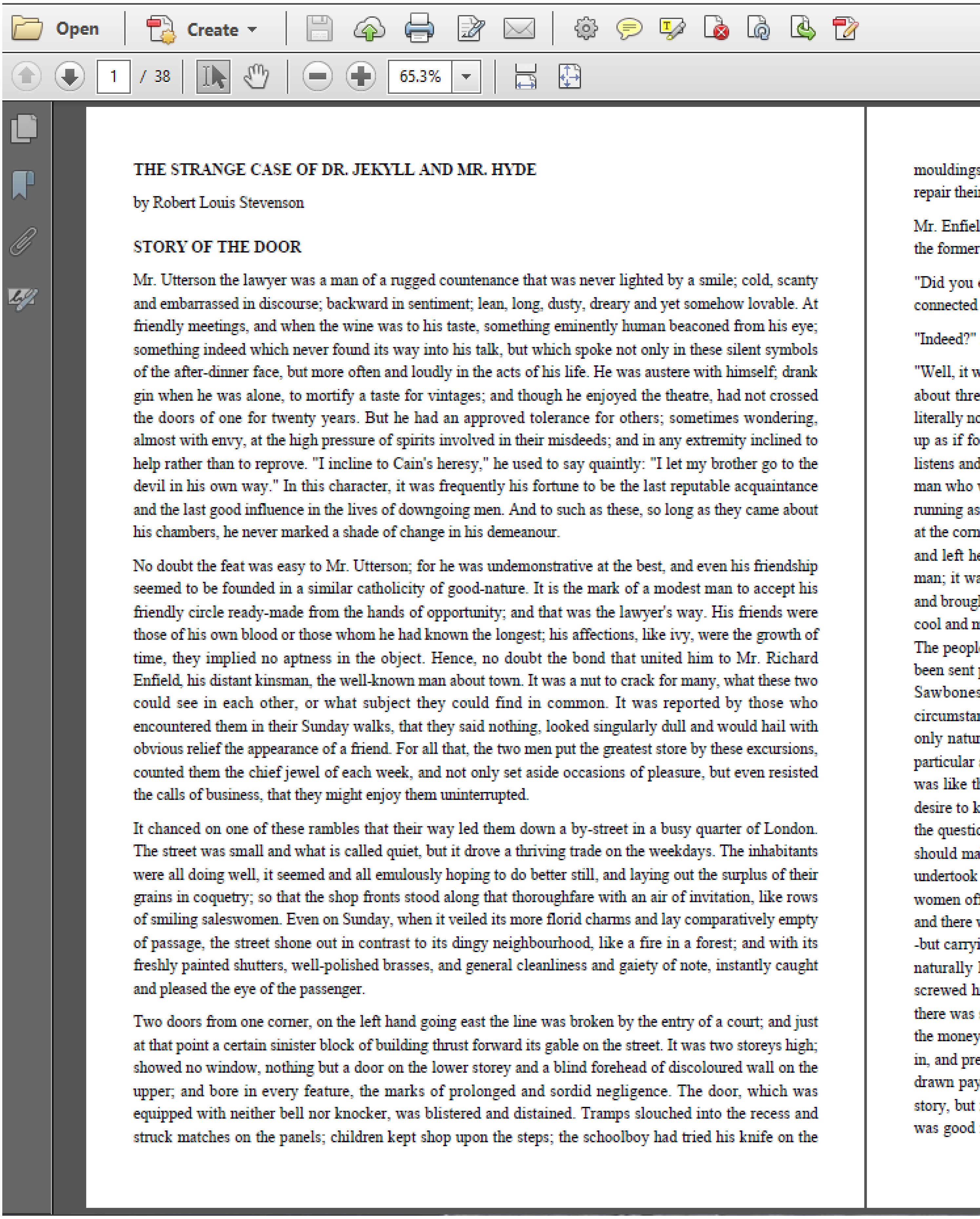
```
throws DocumentException, IOException {
    Document document = new Document();
    PdfWriter.getInstance(document, new FileOutputStream(dest));
    document.open();
    BufferedReader br = new BufferedReader(new FileReader(TEXT));
    String line;
    Paragraph p;
    Font normal = new Font(FontFamily.TIMES_ROMAN, 12);
    Font bold = new Font(FontFamily.TIMES_ROMAN, 12, Font.BOLD);
    boolean title = true;
    while ((line = br.readLine()) != null) {
        p = new Paragraph(line, title ? bold : normal);
        p.setAlignment(Element.ALIGN_JUSTIFIED);
        title = line.isEmpty();
        document.add(p);
    }
    document.close();
}
```

Source: developers.itextpdf.com

Text2Pdf.java (iText 7)

Suppose that you have the following text file: jekyll_hyde.txt

How do we convert it to a PDF that looks like this:


Open Create 1 / 38 65.3%

THE STRANGE CASE OF DR. JEKYLL AND MR. HYDE

by Robert Louis Stevenson

STORY OF THE DOOR

Mr. Utterson the lawyer was a man of a rugged countenance that was never lighted by a smile; cold, scanty and embarrassed in discourse; backward in sentiment; lean, long, dusty, dreary and yet somehow lovable. At friendly meetings, and when the wine was to his taste, something eminently human beaconed from his eye; something indeed which never found its way into his talk, but which spoke not only in these silent symbols of the after-dinner face, but more often and loudly in the acts of his life. He was austere with himself; drank gin when he was alone, to mortify a taste for vintages; and though he enjoyed the theatre, had not crossed the doors of one for twenty years. But he had an approved tolerance for others; sometimes wondering, almost with envy, at the high pressure of spirits involved in their misdeeds; and in any extremity inclined to help rather than to reprove. "I incline to Cain's heresy," he used to say quaintly: "I let my brother go to the devil in his own way." In this character, it was frequently his fortune to be the last reputable acquaintance and the last good influence in the lives of downgoing men. And to such as these, so long as they came about his chambers, he never marked a shade of change in his demeanour.

No doubt the feat was easy to Mr. Utterson; for he was undemonstrative at the best, and even his friendship seemed to be founded in a similar catholicity of good-nature. It is the mark of a modest man to accept his friendly circle ready-made from the hands of opportunity; and that was the lawyer's way. His friends were those of his own blood or those whom he had known the longest; his affections, like ivy, were the growth of time, they implied no aptness in the object. Hence, no doubt the bond that united him to Mr. Richard Enfield, his distant kinsman, the well-known man about town. It was a nut to crack for many, what these two could see in each other, or what subject they could find in common. It was reported by those who encountered them in their Sunday walks, that they said nothing, looked singularly dull and would hail with obvious relief the appearance of a friend. For all that, the two men put the greatest store by these excursions, counted them the chief jewel of each week, and not only set aside occasions of pleasure, but even resisted the calls of business, that they might enjoy them uninterrupted.

It chanced on one of these rambles that their way led them down a by-street in a busy quarter of London. The street was small and what is called quiet, but it drove a thriving trade on the weekdays. The inhabitants were all doing well, it seemed and all emulously hoping to do better still, and laying out the surplus of their grains in coquetry; so that the shop fronts stood along that thoroughfare with an air of invitation, like rows of smiling saleswomen. Even on Sunday, when it veiled its more florid charms and lay comparatively empty of passage, the street shone out in contrast to its dingy neighbourhood, like a fire in a forest; and with its freshly painted shutters, well-polished brasses, and general cleanliness and gaiety of note, instantly caught and pleased the eye of the passenger.

Two doors from one corner, on the left hand going east the line was broken by the entry of a court; and just at that point a certain sinister block of building thrust forward its gable on the street. It was two storeys high; showed no window, nothing but a door on the lower storey and a blind forehead of discoloured wall on the upper; and bore in every feature, the marks of prolonged and sordid negligence. The door, which was equipped with neither bell nor knocker, was blistered and distained. Tramps slouched into the recess and struck matches on the panels; children kept shop upon the steps; the schoolboy had tried his knife on the

mouldings
repair thei
Mr. Enfiel
the former
"Did you
connected
"Indeed?"
"Well, it w
about thre
literally no
up as if fo
listens and
man who
running as
at the corn
and left h
man; it wa
and broug
cool and n
The peopl
been sent
Sawbones
circumsta
only natur
particular
was like t
desire to k
the questi
should ma
undertook
women of
and there
-but carryi
naturally l
screwed h
there was
the money
in, and pre
drawn pay
story, but
was good

When using iText 7, we'd need the following code:

```
public void createPdf(String dest) throws IOException {
    PdfDocument pdf = new PdfDocument(new PdfWriter(dest));
    Document document = new Document(pdf)
        .setTextAlignment(TextAlignment.JUSTIFIED);
    BufferedReader br = new BufferedReader(new FileReader(TEXT));
    String line;
    PdfFont normal = PdfFontFactory.createFont(FontConstants.TIMES_ROMAN);
    PdfFont bold = PdfFontFactory.createFont(FontConstants.TIMES_BOLD);
    boolean title = true;
    while ((line = br.readLine()) != null) {
        document.add(new Paragraph(line).setFont(title ? bold : normal));
        title = line.isEmpty();
    }
    document.close();
}
```

Source: developers.itextpdf.com and the iText 7: Building Blocks tutorial.

Read Text to PDF: iText 5 versus iText 7 online: https://riptutorial.com/itext/topic/5786/text-to-pdf--itext-5-versus-itext-7

Credits

S. No	Chapters	Contributors
1	Getting started with itext	Alexis Pigeon, Bruno Lowagie, Community, Egl, mkl
2	Columns: iText 5 versus iText 7	Bruno Lowagie, Egl
3	Fonts: iText 5 versus iText 7	Bruno Lowagie, Egl
4	Forms: iText 5 vs iText 7	Bruno Lowagie, Egl
5	Page events (iText 5) versus Event handlers and Renderers (iText 7)	Bruno Lowagie, Egl
6	Pdf Creation: iText 5 versus iText 7	Bruno Lowagie, Egl
7	Q & A about versions	Bruno Lowagie
8	Styles: iText 5 versus iText 7	Bruno Lowagie, Egl
9	Tables: iText 5 versus iText 7	Bruno Lowagie, Egl
10	Text to PDF: iText 5 versus iText 7	Bruno Lowagie, Egl