iText

1. Wprowadzenie

2. Przykłady i zastosowania.

Wprowadzenie

Biblioteka iText (http://itextdocs.lowagie.com/) służy głównie do tworzenia dokumentów PDF przez programy napisane w Javie. Jej dodatkowe możliwości to obsługa formatów RTF i HTML.

PDF w pięciu krokach

Aby utworzyć dokument w formacie pdf należy wykonać następujące czynności:

- stworzyć instancje (obiekt) **`com.lowagie.text.Document`**:

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

- powiązać dokument ze strumieniem wyjściowym:

```
PdfWriter.getInstance(document,
                      new FileOutputStream("HelloWorld.pdf"));
```

- otworzyć dokument:

```
document.open();
```

- wprowadzić zawartość dokumentu:

```
document.add(new Paragraph("Hello World"));
```

- zamknąć dokument:

```
document.close();
```

PDF w pięciu krokach

```
import java.io.*;
import com.lowagie.text.*;
import com.lowagie.text.pdf.PdfWriter;

public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World");
        // krok 1
        Document document = new Document();
        try {
            // krok 2
            PdfWriter.getInstance(document,
                                  new FileOutputStream("HelloWorld.pdf"));
            // krok 3
            document.open();
            // krok 4
            document.add(new Paragraph("Hello World"));
        } catch (DocumentException de) {
            System.err.println(de.getMessage());
        } catch (IOException ioe) {
            System.err.println(ioe.getMessage());
        }
        // krok 5
        document.close();
    }
}
```

Krok 1: konstruktor `Document`

Klasa **`com.lowagie.text.Document`** posiada trzy konstruktory:

- **`public Document(Rectangle pageSize, int marginLeft, int marginRight, int marginTop, int marginBottom);`**
- **`public Document(Rectangle pageSize);`** równoważny wywołaniu
  `Document(Rectangle pageSize, 36, 36, 36, 36);`
- **`public Document();`** równoważny wywołaniu
  `Document(PageSize.A4);`

Krok 1: konstruktor `Document`

Rozmiary stron są typu **`Rectangle`**. W ramach klasy **`com.lowagie.text.PageSize`**: zdefiniowano większość podstawowych rozmiarów: **`A0-A10, LEGAL, LETTER, HALFLETTER, _11x17, LEDGER, NOTE, B0-B5, ARCH_A-ARCH_E, FLSA and FLSE.`**

Większość z powyższych formatów jest typu **`portrait`**. Aby utworzyć dokument typu **`landscape`** należy zdefiniować wysokość mniejszą od szerokości. Można również użyć metody **`rotate()`**:

```
Document document = new Document(PageSize.A4.rotate());
```

Krok 1: konstruktor `Document`

Można tworzyć także dokumenty mieszane.

```
Document document = new Document(PageSize.A4.rotate());
...
document.open();
document.add(new Paragraph("Ta strona jest typu landscape"));
document.setPageSize(PageSize.A4);
document.newPage();
document.add(new Paragraph("Ta natomiast znowu portrait"));
...
```

iText pozwala tworzyć strony o dowolnych rozmiarach, nie przekraczających 200" na 200". Należy pamiętać, ze Acrobat w wersji 3.0 dopuszcza maksymalnie 45" na 45".

Krok 1: konstruktor `Document`

Wraz z rozmiarem strony można określić także kolor tła:

```
...
Rectangle pageSize = new Rectangle(216, 720);
pageSize.setBackgroundColor(new java.awt.Color(0xFF, 0xFF, 0xDE));
Document document = new Document(pageSize);
...
document.open();
document.add(new Paragraph("Rozmiar strony to 216x720 punktów"));
document.add(new Paragraph("co daje 3 na 10 cali"));
document.add(new Paragraph("czyli 7.62 na 25.4 cm"));
document.add(new Paragraph("Użyty kolor tla to #FFFFDE."));
...
```

Krok 2: powiązanie dokumentu z plikiem `DocWriter`

Obecnie istnieją trzy formaty zapisu dokumentu:

- **`PdfWriter`** – generuje dokument w formacie PDF (*Portable Document Format*),
- **`RtfWriter`** – generuje dokument w formacie RTF (*Rich Text Format*),
- **`HtmlWriter`** – generuje dkolument w formacie HTML.

Wszystkie te klasy są wyprowadzone z klasy abstrakcyjnej **`DocWriter`**. Konstruktor jest chroniony więc jedyna możliwość związania dokumentu z plikiem to wywołanie metody:

```
public static xxxWriter getInstance(Document document,
                    OutputStream os) throws DocumentException
```

(**`xxx`** oznacza **`Pdf`**, **`Rtf`** lub **`Html`**).

Krok 2: powiązanie dokumentu z plikiem `DocWriter`

```
PdfWriter writer = PdfWriter.getInstance(document,
                        new FileOutputStream("HelloWorld.pdf"));
```

Obiekt **`writer`** w typowych zastosowaniach nie jest nigdy używany.

Drugi parametr jest dowolnym strumieniem wyjściowym (**`java.io.OutputStream`**). Najczęściej używane:

- **`java.io.FileOutputStream`** – zapis do pliku,
- **`java.io.ByteArrayOutputStream`** – tworzenie dokumentu w pamięci operacyjnej,
- **`javax.servlet.ServletOutputStream`** – generowanie pdf'a przez serwer www,
- **`System.out`** – niezbyt ciekawe rozwiązanie.

Krok 2: powiązanie dokumentu z plikiem `DocWriter`

Do jednego dokumentu można utworzyć wiele **`DocWriter`**'ów.

```
Document document = new Document();

...

PdfWriter pdf = PdfWriter.getInstance(document,
                                      new FileOutputStream("HelloWorldPdf.pdf"));

RtfWriter2 rtf = RtfWriter2.getInstance(document,
                                      new FileOutputStream("HelloWorldRtf.rtf"));

HtmlWriter html = HtmlWriter.getInstance(document,
                                      new FileOutputStream("HelloWorldHtml.html"));

document.open();

document.add(new Paragraph("Hello World")); // pojawi sie we wszystkich
plikach

pdf.pause();

rtf.pause();

// dalsze rzeczy pojawia sie tylko w HTML'u
```

Krok 2: powiązanie dokumentu z plikiem `DocWriter`

```
Anchor pdfRef = new Anchor("see Hello World in PDF.");
pdfRef.setReference("./HelloWorldPdf.pdf");
Anchor rtfRef = new Anchor("see Hello World in RTF.");
rtfRef.setReference("./HelloWorldRtf.rtf");
document.add(pdfRef);
document.add(Chunk.NEWLINE);
document.add(rtfRef);
pdf.resume();
rtf.resume();
// dalsze rzeczy pojawia sie znów we wszystkich plikach
...
```

Krok 3: otwarcie dokumentu

Istnieje szereg czynności, które można zrobić wyłącznie przed otwarciem dokumentu. Najczęściej wykorzystywane to:

• szyfrowanie dokumentu:

```
...

Document document = new Document();

PdfWriter writer = PdfWriter.getInstance(document,
                                    new FileOutputStream("HelloEncrypted.pdf"));

writer.setEncryption(PdfWriter.STRENGTH128BITS, "Hello", "World",
                                PdfWriter.AllowCopy | PdfWriter.AllowPrinting);

document.open();

document.add(new Paragraph("Hello World"));

...
```

Krok 3: otwarcie dokumentu

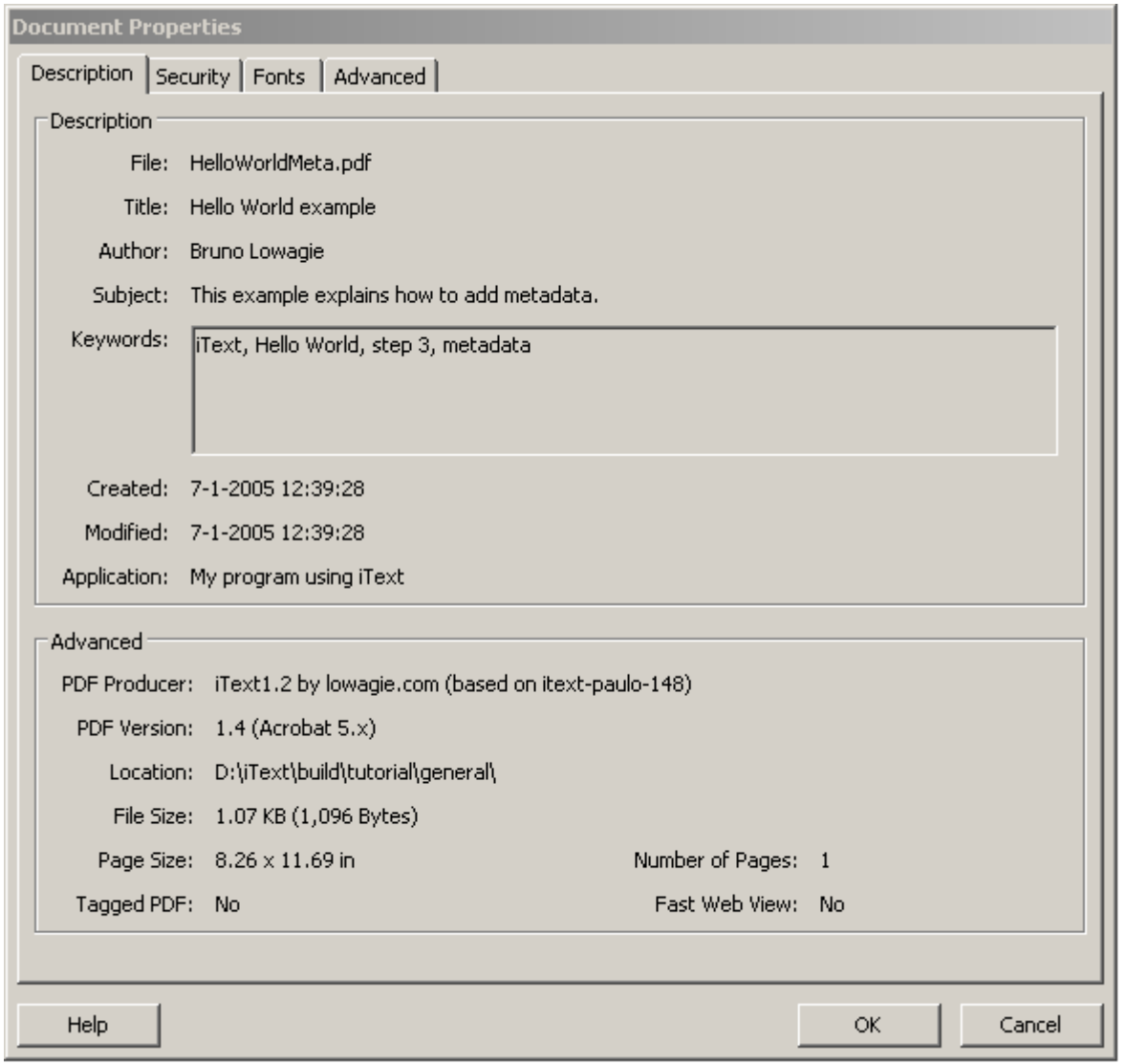
• dodatkowe informacje o dokumencie – można ustawić za pomocą odpowiednich metod, np:

Tytuł:	**`addTitle(java.lang.String)`**
Autor:	**`addAuthor(java.lang.String)`**
Temat:	**`addSubject(java.lang.String)`**
Słowa kluczowe:	**`addKeywords(java.lang.String)`**
Application:	**`addCreator(java.lang.String)`**
Producent:	**`addProducer(java.lang.String)`**
Data utworzenia:	**`addCreationDate(java.util.Date)`**

Krok 3: otwarcie dokumentu

```
...
document.addTitle("Hello World example");
document.addAuthor("Bruno Lowagie");
document.addSubject("This example explains how to add metadata.");
document.addKeywords("iText, Hello World, step 3, metadata");
document.addCreator("My program using iText");
...
```

Krok 3: otwarcie dokumentu



Krok 4: zawartość dokumentu

iText zawiera zestaw własnych "obiektów wysokiego poziomu" które są odwzorowywane na konkretne elementy PDF'a. Najpopularniejsze z nich to:

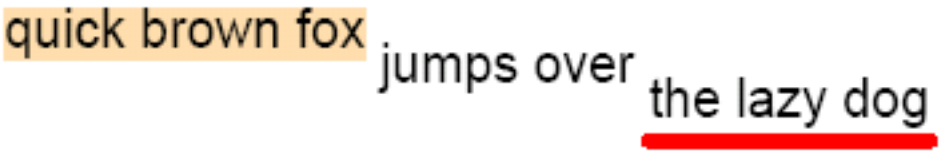
• **Chunk** – najmniejszy fragment (jednostka) dokumentu:

```
...
Chunk fox = new Chunk("quick brown fox");
float superscript = 8.0f;
fox.setTextRise(superscript);
fox.setBackground(new Color(0xFF, 0xDE, 0xAD));
Chunk jumps = new Chunk(" jumps over ");
Chunk dog = new Chunk("the lazy dog");
float subscript = -8.0f;
dog.setTextRise(subscript);
dog.setUnderline(new Color(0xFF, 0x00, 0x00), 3.0f, 0.0f, -5.0f + subscript,
                                              0.0f, PdfContentByte.LINE_CAP_ROUND);
```

Krok 4: zawartość dokumentu

```
document.add(fox);
document.add(jumps);
document.add(dog);
...
```

Efekt:



Krok 4: zawartość dokumentu

- **`Paragraph`** – zbiór (kontener) dla innych elementów dokumentu:

```
...
Paragraph p = new Paragraph();
p.add(new Chunk("This text is in Times Roman. This is ZapfDingbats: ",
                                        new Font(Font.TIMES_ROMAN, 12)));
p.add(new Chunk("abcdefghijklmnopqrstuvwxyz",
                                        new Font(Font.ZAPFDINGBATS, 12)));
p.add(new Chunk(". This is font Symbol: ", new Font(Font.TIMES_ROMAN, 12)));
p.add(new Chunk("abcdefghijklmnopqrstuvwxyz", new Font(Font.SYMBOL, 12)));
document.add(new Paragraph(p));
...
```

This text is in Times Roman. This is ZapfDingbats: ❁✲❃❉❆❄✺✹✻✼❋●○■❑❒❑❒▲▼◆❖◗❙❚. This is font Symbol: αβχδεφγηιφκλμνοπθρστυϖωξψζ

Krok 4: zawartość dokumentu

W ramach paragrafu można ponadto określić m. in.:

• justowanie – **`Element.ALIGN_LEFT, Element.ALIGN_CENTER, Element.ALIGN_RIGHT, Element.ALIGN_JUSTIFIED`**,

• odstępy między znakami – **`setSpaceCharRatio()`** (domyślnie 2.5):

```
writer.setSpaceCharRatio(PdfWriter.NO_SPACE_CHAR_RATIO);
```

• wcięcia:

`setIndentationLeft(float indentation)`,

`setIndentationRight(float indentation)`,

`setSpacingBefore(float spacing)`,

`setSpacingAfter(float spacing)`.

Krok 4: zawartość dokumentu

• **`Phrase`** – może być używana zamiennie z **`Paragraph`**:

```
...
Phrase phrase0 = new Phrase();

Phrase phrase1 = new Phrase("(1) this is a phrase\n");

Phrase phrase2 = new Phrase(24, "(2) this is a phrase with leading 24.
    You can only see the difference if the line is long enough. Do you see it?
    There is more space between this line and the previous one.\n");

Phrase phrase3 = new Phrase("(3) this is a phrase with a red, normal font
    Courier, size 20. As you can see the leading is automatically changed.\n",
    FontFactory.getFont(FontFactory.COURIER, 20, Font.NORMAL,
    new Color(255, 0, 0)));

Phrase phrase4 = new Phrase(new Chunk("(4) this is a phrase\n"));

Phrase phrase5 = new Phrase(18, new Chunk("(5) this is a phrase in Helvetica,
    bold, red and size 16 with a given leading of 18 points.\n",
    FontFactory.getFont(FontFactory.HELVETICA, 16, Font.BOLD,
    new Color(255, 0, 0))));

Phrase phrase6 = new Phrase("(6)");

Chunk chunk = new Chunk(" This is a font: ");

phrase6.add(chunk);
```

Krok 4: zawartość dokumentu

```
phrase6.add(new Chunk("Helvetica", FontFactory.getFont(
    FontFactory.HELVETICA, 12)));

phrase6.add(chunk);

phrase6.add(new Chunk("Times New Roman", FontFactory.getFont(
    FontFactory.TIMES_ROMAN, 12)));

phrase6.add(chunk);

phrase6.add(new Chunk("Courier", FontFactory.getFont(
    FontFactory.COURIER, 12)));

phrase6.add(chunk);

phrase6.add(new Chunk("Symbol", FontFactory.getFont(FontFactory.SYMBOL, 12)));

phrase6.add(chunk);

phrase6.add(new Chunk("ZapfDingBats", FontFactory.getFont(
    FontFactory.ZAPFDINGBATS, 12)));

Phrase phrase7 = new Phrase("(7) if you don't add a newline yourself, all
    phrases are glued to eachother!");

document.add(phrase1);

document.add(phrase2);

document.add(phrase3);
```

Krok 4: zawartość dokumentu

```
document.add(phrase5);
document.add(phrase6);
document.add(phrase7);
...
```

(1) this is a phrase
(2) this is a phrase with leading 24. You can only see the difference if the line is long enough. Do you see it? There is more space between this line and the previous one.
(3) this is a phrase with a red, normal font Courier, size 20. As you can see the leading is automatically changed.
(4) this is a phrase
(5) this is a phrase in Helvetica, bold, red and size 16 with a given leading of 18 points.
(6) This is a font: Helvetica This is a font: Times New Roman This is a font: Courier This is a font: Σψμβολ This is a font: ❁❃❑❄⌘✻■❋✣❁▼▲(7) if you don't add a newline yourself, all phrases are glued to eachother!

Krok 4: zawartość dokumentu

```
...

document.add(new Phrase(16, "\n\n\n"));

document.add(new Phrase(-16, "Hello, this is a very long phrase to show you
    the somewhat odd effect of a negative leading. You can write from bottom
    to top. This is not fully supported. It's something between a feature and
    a bug."));

...
```

bug.
can write from bottom to top. This is not fully supported. It's something between a feature and a
Hello, this is a very long phrase to show you the somewhat odd effect of a negative leading. You

Krok 4: zawartość dokumentu

```
...

document.add(Phrase.getInstance("What is the " + (char) 945 + "-coefficient of
     the " + (char) 946 + "-factor in the " + (char) 947 + "-equation?\n"));

for (int i = 913; i < 970; i++) {

    document.add(Phrase.getInstance(" " + String.valueOf(i) + ": " +
        (char) i));

}

...
```

What is the α-coefficient of the β-factor in the γ-equation?
913: Α 914: Β 915: Γ 916: Δ 917: Ε 918: Ζ 919: Η 920: Θ 921: Ι 922: Κ 923: Λ 924: Μ 925: Ν 926: Ξ 927: Ο 928: Π 929: Ρ 930: 931: Σ 932: Τ 933: Υ 934: ϑ 935: Χ 936: Ψ 937: Ω 938: 939: 940: 941: 942: 943: 944: 945: α 946: β 947: γ 948: δ 949: ε 950: ζ 951: η 952: θ 953: ι 954: κ 955: λ 956: μ 957: ν 958: ξ 959: ο 960: π 961: ρ 962: σ 963: σ 964: τ 965: υ 966: φ 967: χ 968: ψ 969: ω

Krok 5: zamknięcie dokumentu

Zamknięcie dokumentu (`document.close()`) powoduje wyczyszczenie i zapisanie strumieni wyjściowych związanych z dokumentem. Metoda close() jest także wywoływana automatycznie podczas usuwania obiektu, jednak powinno się ją wywoływać samodzielnie.

Zamiast podsumowania

http://itextdocs.lowagie.com/tutorial/