Fachhochschule Aargau
Nordwestschweiz

F | H | A

iText Class Library

Fachhochschule Nordwestschweiz FHA

University of Applied Sciences Aargau

Autor: Filipe Luis
f.luis@stud.fh-aargau.ch

Studiengang: Informatik
Fach: Seminar Enterprise Computing
Reporting in Java Web Applications
Dozent Prof. Dr. Dominik Gruntz
Datum: 07. Mai 2004

Abstract

iText ist eine offene und frei erhältliche Java-Bibliothek zur Erzeugung von PDF Dokumenten. Mit dieser Bibliothek können PDFs auch aus Web-Applikationen programmatisch erzeugt werden. Dieser Seminarbericht soll einen Überblick über die Library geben und dadurch die Möglichkeiten von iText aufzeigen. Im Besonderen sollen dabei die Möglichkeiten der PDF-Erzeugung mittels XML Dokumenten und aus Servlets aufgezeigt werden.

Inhaltsverzeichnis

1. Was ist iText **4**

2. Einführung: Einfache Dokumente **5**
2.1. Das erste Dokument 5

3. Elemente eines Dokuments **8**
3.1. Chunks 8
3.2. Phrase 9
3.3. Paragraphs 9
3.4. Sprungmarken 9
3.5. Listen 9
3.6. Tabellen 10

4. Positionierung & Writer **11**
4.1. Automatische Anordnung 11
4.2. PdfWriter 12
4.2.1. Interne Schichten 13
4.2.2. Externe Schichten 13
4.3. HtmlWriter 14
4.4. XmlWriter 15
4.5. RtfWriter 16

5. iText in Web-Applikationen **17**
5.1. Servlet 17

6. iText & XML **20**
6.1. Tagmap für eigene XML Dokumente 20

A. Source Code **22**
A.1. PdfCotnentByte 22
A.2. Listen 23
A.3. Servlets 24

1. Was ist iText

iText ist eine Bibliothek zur dynamischen Erzeugung und Manipulation von *Portable Document Format* (PDF) Dateien. Das Projekt wird von Bruno Lowagie vorangetrieben und entwickelt. Die iText Bibliothek bietet dem Entwickler umfangreiche Möglichkeiten in Java PDF Dokumente zu erstellen. Seit kurzer Zeit wurde iText auch auf die C# Plattform portiert (einige Klassen stehen allerdings noch nicht zur Verfügung, weiterführende Informationen dazu unter [4]).

Die iText Bibliothek umfasst im wesentlichen folgende Features:

- Unterschiedliche Seitengrössen (Beispiele: A0 bis A10, Legal, Letter uvm.
- Hoch- oder Querformat
- Seitenabstände
- Tabellen
- Silbentrennung
- Kopf- und Fusszeilen
- Seitennummerierung
- Barcodes
- Schrifttypen
- Dokument Verschlüsselung
- Integration von Bilddateien (Formate: JPEG, GIF, PNG und WMF)
- Geordnete und ungeordnete Listen
- Dokumenttemplates
- Erzeugung von PDF Dateien mittels XML

Die gesamte Bibliothek ist unter [3] erhältlich und offen.

2. Einführung: Einfache Dokumente

Ziel der Einführung ist es, die notwendigen Schritte zur Erzeugung einer PDF Datei aufzuzeigen und zu erläutern.

2.1. Das erste Dokument

Die Erstellung eines PDF Dokuments mit der iText Bibliothek gliedert sich in fünf Schritte:

1. Erzeugung einer `Document` Instanz.
2. Erzeugung eines Writers, welcher als Obeserver des `Document` Objekts agiert und zugefügten Inhalt in die PDF Datei schreibt (siehe Kapitel 4.1 auf Seite 11).
3. Das Dokument *muss*, um Inhalt hinzufügen zu können explizit geöffnet werden.
4. Hinzufügen des Inhaltes.
5. Schliessen des Dokumentes (damit die OutputStreams geleert werden).

Nachfolgend wurde der Programm-Code zur Erzeugung eines einfachen PDF Dokumentes aufgelistet. Im Code finden sich die oben beschriebenen Schritte wieder!

Listing `HelloWorld.java`:

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


public class HelloWorld
{

    public static void main(String[] args)
    {
        //-- 1. Schritt: Document Instanz
        Document doc = new Document();

        try
                {
                        //-- 2. Schritt: Writer erzeugen
                        PdfWriter.getInstance(doc, new FileOutputStream("HelloWorld.pdf"));

                        //-- 3. Schritt: Dokumentö ffnen
                        doc.open();
```

```
                //-- 4. Schritt: Inhalt ühinzufgen
                Font font = FontFactory.getFont(FontFactory.HELVETICA_BOLD, 34);
                Chunk c = new Chunk("Hello World",font);
                doc.add(c);

                //-- 5. Schritt: Dokument schliessen
                doc.close();
                System.out.println("HelloWorld.pdf created!");

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

Die erzeugte `HelloWorld.pdf` Datei:

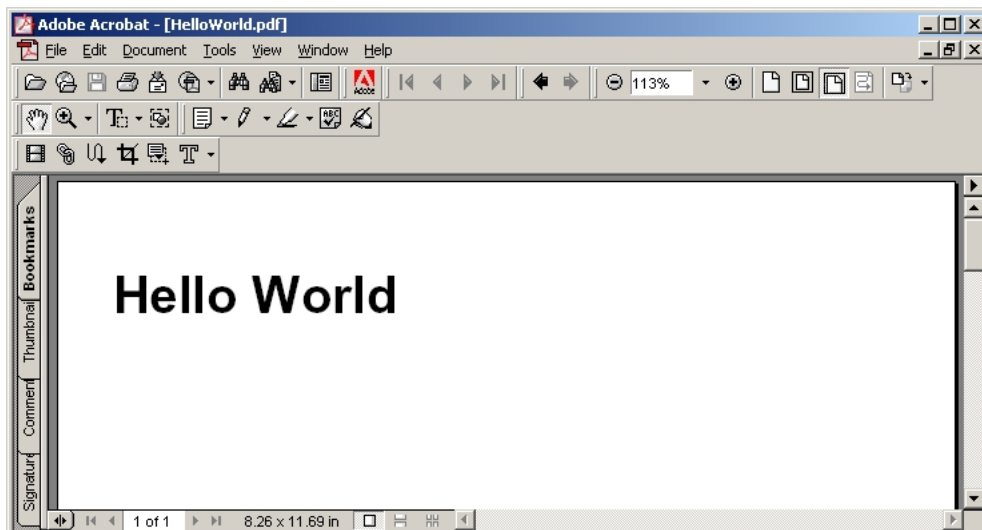

Abbildung 2.1.: Output des ersten Beispiels.

Seitengrösse. Das Objekt `com.lowagie.text.Document` besitzt mehrere Konstruktoren. Bei der Erzeugung einer `Document` Instanz wird ein `Rectangle` Objekt erwartet (Vorsicht: Ein `com.lowagie.text.Rectangle` und nicht ein `java.awt.Rectangle` Objekt), wird kein Parameter angegeben, so wird das statische `Rectangle` Objekt `PageSize.A4` verwendet (default).

Metadata. Über diverse `Document` Methoden können dem PDF Dokument Metadaten hinzugefügt werden. Folgende stehen zur Verfügung:

```
public boolean addTitle(String title)
public boolean addSubject(String subject)
```

```
public boolean addKeywords(String keywords)
public boolean addAuthor(String author)
public boolean addCreator(String creator)
```

Der Rückgabetyp `boolean` ist `true` wenn die entsprechende Operation erfolgreich ausgeführt werden konnte, sonst `false`.

Verschlüsselung. Der PdfWriter kann mit der Methode `public void setEncryption(byte[] userPassword, byte[] ownerPassword, int permissions, boolean strength128Bits)` angewiesen werden, das Dokument zu verschlüsseln. Mit Ausnahme der Passwörter können alle Parameter mit den statischen Felder der Klasse `PdfWriter` angegeben werden. Unter anderem steht dem Entwickler das Feld `PdfWriter.AllowPrinting` zur Verfügung. Wird dieser Wert nicht angegeben, so kann der User ohne Angabe des korrekten Passworts, die PDF Datei nicht mehr drucken (zumindest nicht ohne gewisse Bemühungen).

Einheiten. In der iText Bibliothek wird die typographische Einheit Punkt verwendet. 72 Punkte ergeben ein Inch oder $\sim$ 0.4cm. Eine A4 Seite hat somit 595 Punkte in der Breite und 842 Punkte in der Höhe.

3. Elemente eines Dokuments

Um einen Eindruck zu vermitteln was in iText dargestellt werden kann, sollen einige elementare Bausteine erläutert werden. Diese Liste ist bei weitem nicht vollständig, sie soll lediglich eine Übersicht vermitteln.

3.1. Chunks

Die ‘kleinste’ Einheit in iText ist das `Chunk` Objekt. Dieses Element besteht nur aus einem String mit einer bestimmten Schrift. Die weiteren Elemente `Phrase` und `Paragraph` bestehen aus lauter aneinander gehängten `Chunk` Objekten. Ein Beispiel für ein Chunk Objekt ist im Listing 2.1 auf Seite 5 zu sehen.

Formatierung eines Chunks. Verschiedene Möglichkeiten stehen dem Entwickler zur Verfügung um einen Chunk zu formatieren.

- Für ein Chunk Text der ‘Unterstrichen’ und ‘Kursiv’ gesetzt werden soll, genügt folgende Anweisung:

  ```
  Chunk chunk = new Chunk("Kursiv und unterschrichen",
  FontFactory.getFont(FontFactory.HELVETICA,12, Font.UNDERLINE | Font.ITALIC));
  ```

  Weitere Möglichkeiten wie z.B. Durchstreichen des Texts sind vorhanden.

- Mit der Methode `setTextRise(float f)` kann in Punkten definiert werden, ob der Text Hochgestellt oder Tiefgestellt gedruckt werden soll. Die Angabe erfolgt relativ zur Baseline, wobei ein positives Vorzeichen den Text nach oben verschiebt, ein negatives den Text nach unten versetzt.

- Die Hintergrundfarbe kann mit der Methode `setBackground(java.awt.Color color)` gesetzt werden.

3.2. Phrase

Das Phrase Objekt stellt eine Sammlung von einem oder mehreren Chunk Objekten dar. Zusätzlich definiert das Phrase Objekt ein sog. leading (Zeilenabstand) und eine Main Font welche für alle beinhalteten Chunk Elemente gilt (es ist möglich einzelnen Elementen ein anderes Layout zuzuweisen).

3.3. Paragraphs

Ein Absatz besteht aus einem oder mehreren Chunk und/oder Phrase Objekten. Ein Paragraph Objekt kann wie ein Phrase Objekt formatiert werden. Zusätzlich stehen jedoch noch die Möglichkeiten den Text einzurücken und auszurichten (links, rechts oder zentriert) zur Verfügung.

3.4. Sprungmarken

iText bietet die Möglichkeit, ähnlich wie in HTML, Links zu definieren. Als Sprungziele können interne (innerhalb des Dokumentes) sowie auch externe Quellen referenziert werden (z.B. eine Website). Nachfolgend ist ein Anchor Objekt dargestellt, welches auf die Website der Fachhochschule Aargau verweist, und ein Anchor-Paar, welches eine Sprung- und eine Zielmarke darstellt. Das Konzept der Anchors entspricht demjenigen der Links in HTML. Der Entwickler definiert also ein String, im nachfolgenden Beispiel jeweils goto, welches beim Anklicken den Betrachter des Dokuments zur Zielmarke springen lässt. Die Zielmarke besteht ebenfalls aus einem String das für den Benutzer sichtbar ist (hier: destination) und einem Namen (NextPage) welches intern als Markierung des Sprungsziels dient, für den Betrachter jedoch unsichtbar ist.

```
Anchor xref = new Anchor("goto");
xref.setReference("http://www.fh-aargau.ch/");  //-- extern
xref.setName("Website");

Anchor iref = new Anchor("goto", afont);
iref.setReference("#NextPage");                 //-- intern

Anchor zref = new Anchor("destination");
zref.setName("NextPage");
```

3.5. Listen

Mit dem List Objekt ist es möglich Listen zu erstellen. Der Kunstruktor erwartet ein boolean Flag, mit welchem angezeigt wird, ob die Liste geordnet oder ungeordnet ist. Verschachtelte Listen

sind ebenfalls möglich. Im Anhang A.2 auf Seite 23 ist ein Beispiel zu finden.

3.6. Tabellen

Mit der `Table` Klasse können Tabellen erzeugt werden. Die Klasse stellt zwei Konstruktoren zur Verfügung:

- `Table(int spalten)`
- `Table(int spalten, int zeilen)`

Mit folgenden Parametern kann das Aussehen der Tabelle formatiert werden:

Cellspacing und Cellpadding: Mit der Methode `setSpacing(float value)` kann der Abstand vom Zelleninhalt und dem Zellenrand in Punkte angegeben werden. Um den Abstand der Zellen zum Tabellenrand (Cellpadding) zu definieren, wird die Methode `setPadding(float value)` aufgerufen.

Alignment: Um den Inhalt in den Zellen auszurichten können folgende Methoden verwendet werden.

```
cell.setHorizontalAlignment(Element.ALIGN_CENTER);
cell.setVerticalAlignment(Element.ALIGN_MIDDLE);
```

Color: Die Hintergrundfarbe wird mit der Methode `setBackgroundColor(Color color)` gesetzt.

Um der Tabelle Inhalt zuzufügen, werden Instanzen der `Cell` Klasse verwendet. Die `Cell` Objekte können ein beliebiges Element aufnehmen. Nachdem der Inhalt einer Zelle bestimmt ist, kann die Zelle über die Methode `addCell(...)` der Tabelle eingefügt werden. Die Zellen werden von links nach rechts eingefügt. Beim Konstruktor kann die Zeilenanzahl weggelassen werden, da die Tabelle automatisch Zeilen einfügt wenn mehr benötigt werden sollten (dies gilt auch für den zweiten Konstruktor, obwohl die Zeilenanzahl angegeben werden kann).

Informationen zum Umbruch der Tabellen bei einem Seitewechsel sind im Abschnitt 4.1 auf Seite 12 zu finden.

4. Positionierung & Writer

Die iText Bibliothek definiert fünf Writer: *DocWriter*, *PdfWriter*, *HtmlWriter*, *XmlWriter* und *RtfWriter*. Die Writer werden wie bereits im ersten Beispiel 2.1 auf Seite 5 gesehen im zweiten Schritt erzeugt. Sie agieren als *Listener* und werden über die `add()` Methoden über Änderungen am Dokument notifiziert. Jedes Element muss also über die `add(...)` Methode der `Document` Klasse hinzugefügt werden, da die Writer ansonsten nicht über die Änderung notifiziert werden. Die Writer entscheiden dann, je nach eingefügtem Element, wie sie ihn darstellen. Die Writer sind dabei auch an die Grenzen des jeweiligen Datenformats gebunden. Nicht alles was in PDF Dokumenten darstellbar ist, kann beispielsweise auch in HTML umgesetzt werden (z.B. `newPage()`). Der Writer 'kennt' also die Möglichkeiten des Formats und versucht diese entsprechend zu nutzen.

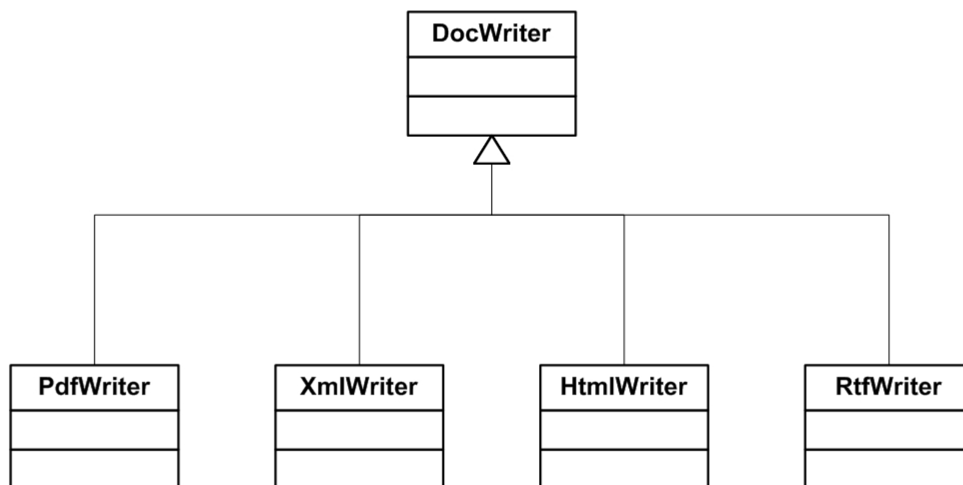

Abbildung 4.1.: Die iText Writers.

4.1. Automatische Anordnung

Die Writer sind auch für die Anordnung der Elemente zuständig, d.h. der Programmierer definiert nur die Struktur des Dokumentes (vergleichbar wie beim schreiben von HTML Seiten). Beim Entwickeln definiert der Programmierer ein Element nach dem anderen und fügt diesen dem Dokument mit der `add(...)` Methode hinzu. Die Writer werden von der `Document` Klasse über diese Änderung notifiziert. Der Writer nimmt diese Elemente entgegen und positioniert diese. Bei der Positionierung hängt die Anordnung u.a. vom Typ des Elementes ab: `Chunk` und `Phrase` Objekte werden beispielsweise nacheinander auf der gleichen Zeile von links nach rechts eingefügt

(Die Zeilenumbrüche werden dabei von iText selbst gesteuert). Die `Paragraph` Objekte haben ein `leading` Parameter was heisst, dass die Absätze immer auf einer neuen Zeile beginnen.

Der Entwickler kann mit gewissen Methoden auf die automatische Positionierung Einfluss nehmen (dies hängt allerdings auch vom Format ab, wie bereits erwähnt sind nicht alle Writer in der Lage alle Elemente korrekt darzustellen). Beispiele:

- `newPage()` Methode der `Document` Klasse: Erwingt in jedem Falle einen Seitenumbruch.
- Mit dem `leading` Parameter im `Paragraph` Konstruktor (Abstand zur vorangehende Zeile).
- Mit der Methode `setKeepTogether()` der Paragraph Klasse, wird iText einen Absatz nicht durch einen Seitenumbruch trennen. In diesem Falle wird der ganze Absatz auf die nächste Seite verschoben.

Seitenumbrüche & Tabellen. iText bestimmt die Seitenumbrüche automatisch. Dabei werden auch Tabellen problemlos umgebrochen. Es besteht sogar die Möglichkeit den Header der Tabelle auf jeder Seite zu wiederholen.

Der Entwickler kann auf die Art wie die Tabelle umgebrochen wird mit folgenden zwei Methoden Einfluss nehmen:

- `setTableFitsPage(true)`: Die Tabelle wird, wenn möglich, auf eine Seite gesetzt. Dabei wird versucht, die Tabelle auf der aktuellen Seite unterzubringen, ansonsten wird mit `newPage()` die Tabelle auf die nächste Seite gesetzt. Falls diese Versuche scheitern, weil der Umfang zu gross für eine Seite ist, wird sie dennoch umgebrochen.
- `hasToFitPageCells(true)`: Die Tabelle wird so umgebrochen, dass ganze Zeilen umgebrochen werden. Ist diese Option nicht gesetzt, so kann es geschehen, dass ein Seitenumbruch innerhalb einer Zeile stattfindet.

Daten Stream. Während dem Anordnen der Elemente erzeugen die Writers fortlaufend einen Daten Stream (darum sollte das Dokument auch immer explizit mit `close()` geschlossen werden, damit die Buffer geleert werden). Dies stellt vor allem bei der Erzeugung von dynamischen Inhalt (Inhaltsverzeichnisse, Referenzen usf.) eine Problematik dar und setzt der iText Bibliothek auch gewisse Grenzen.

4.2. PdfWriter

Der PdfWriter unterscheidet sich in einer Eigenschaft von den anderen Writern: Der Entwickler besitzt die Möglichkeit, Elemente auch manuell an absolute Positionen zu setzten. Dazu wurde in der iText Library ein Modell mit vier hierarchischen Layers eingeführt. Die Hierarchie besteht aus vier `PdfContentByte` Objekten, welche je eine Schicht repräsentieren.

Externer Layer oberhalb von Text und Grafik `PdfContentByte cb = writer.getDirectContent()`
Interner Layer für Text
Interner Layer für Grafik
Externer Layer unterhalb von Text und Grafik `PdfContentByte cb = writer.getDirectContentUnder()`

4.2.1. Interne Schichten

Zwei dieser vier Layers werden vom `PdfWriter` intern für die automatische Positionierung verwendet, auf diese hat der Programmierer keinen direkten Zugriff.

Die Unterteilung der Layers in Text- und Grafik-Schichten dient u.a. der Darstellung von Tabellen: Die Ränder und Hintergrundfarben der Zellen werden im Grafik Layer gesetzt, der eigentliche Inhalt der Zellen wird im Text-Layer abgelegt.

Alle Elemente, welche über die Methode `add( Element element )` der `Document` Klasse dem Dokument hinzugefügt werden, werden über diese interne `PdfContentByte` Objekte gesetzt.

4.2.2. Externe Schichten

Um auf die externen Schichten Zugriff zu erhalten, kann der Entwickler die Methoden `getDirectContent()` und `getDirectContentUnder()` der `PdfWriter` Klasse verwenden. Alle Schichten arbeiten völlig unabhängig voneinander, d.h. die Textfragmente können sich überlappen! Wird der Text in der obersten Schicht eingefügt, so überlappt er die restlichen Elemente, wird er in der untersten Schicht eingefügt so verschwindet er hinter den oberen Elementen.

Absolute Positionierung von Text

Nachfolgend wird ein Beispiel aufgezeigt wie man einen Text absolut positioniert. Beim Positionieren ist besonders der Ursprung des Koordinatensystems zu erwähnen, dieser befindet sich in der linken unteren Ecke.

```
PdfContentByte cb = writer.getDirectContent();
BaseFont bf = BaseFont.createFont(
                    BaseFont.HELVETICA,
                    BaseFont.CP1252,
                    BaseFont.NOT_EMBEDDED);

cb.beginText();
cb.setFontAndSize(bf, 14);
cb.setTextMatrix(0, 0);
```

```
cb.showText("Urspung Koodrinatensystem (0,0).");
cb.endText();
```

Bemerkungen. Zusammengefasst müssen folgende Schritte beachtet werden:

1. Instanz eins `PdfContentByte` Objektes erzeugen.
2. Erzeugen einer `BaseFont`.
3. Den Text-Bereich mit Aufruf der Methode `beginText()` eröffnen.
4. Die Schrift und die Grösse setzen.
5. Eine Text-Matrix an einer bestimmten Stelle öffnen.
6. Mit der Methode `showText(...)` den Text einfügen.
7. Den Text-Bereich durch Aufruf von `endText()` schliessen.

Ein ausführliches Beispiel, in welchem u.a. auch Text unter bestehende Elemente gelegt wird ist im Anhang A.1 auf Seite 22 zu finden.

Rotationen. Die iText Library bietet noch eine weitere Methode Text zu setzen an. Diese Methode `setTextMatrix(float a, float b, float c, float d, float x, float y)` erhält als Parameter die Koeffizienten einer Rotationsmatrix. Auch hierzu findet man im erwähnten Beispiel PdfContentByte im Anhang eine Anwendung.

4.3. HtmlWriter

Der `HtmlWriter` arbeitet im wesentlichen gleich wie der `PdfWriter`. Bruno Lowagie empfiehlt den professionellen Einsatz dieses Packages nicht, da der HTML Code nicht sehr 'schön' ist aber vor allem noch nicht alle Features unterstützt werden. Für das Debugging von PDF Dokumenten hingegen, wird dieses Paket sehr empfohlen, da der HTML Quellcode wesentlich einfacher zu verstehen ist.

Das Einbinden des HtmlWriters ist relativ einfach:

```
Document document = new Document();

PdfWriter.getInstance(document, new FileOutputStream("beispiel.pdf"));
HtmlWriter.getInstance(document, new FileOutputStream("beispiel.html"));
```

Dadurch wird nun bei jedem `add( Element element )` Aufruf gleich auch HTML Code erzeugt. Wenn man dem ersten Beispiel 2.1 auf Seite 5 noch einen `HtmlWriter` hinzufügen würde, erhält man folgenden HTML Code:

```
<html>
    <head>
        <!-- Producer: iTextXML by lowagie.com -->
        <!-- CreationDate: Thu Apr 22 17:00:50 CEST 2004 -->
    </head>
    <body leftmargin="36.0" rightmargin="36.0" topmargin="36.0" bottommargin="36.0">
        <span>Hello World</span>
    </body>
</html>
```

Weiterführende Informationen sind in der API Dokumentation unter [2, com.lowagie.text.html] zu finden.

4.4. XmlWriter

Der `XmlWriter` wird im Kapitel 6 auf Seite 20 eingehender betrachtet und hier nur der Vollständigkeit halber erwähnt.

Das Einbinden des `XmlWriters` erfolgt analog zur Einbindung des HtmlWriters. Zur Erzeugung des XML Dokuments wird die Document Type Declaration von iText (`http://itext.sourceforge.net/itext.dtd`) verwendet. Im wesentlichen stellt die DTD eine Abbildung der iText Elemente dar. So kann jedes Element wie z.B. ein `Chunk` Objekt in XML dargestellt werden.

Wenn man dem ersten Beispiel 2.1 auf Seite 5 noch einen `XmlWriter` hinzufügen würde, erhält man folgenden XML Code:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE ITEXT SYSTEM "http://itext.sourceforge.net/itext.dtd">
<itext creationdate="Thu Apr 22 17:16:57 CEST 2004" producer="iTextXML by lowagie.com">
    <chunk font="Helvetica" size="34.0">Hello World</chunk>
</itext>
```

4.5. RtfWriter

Mit der iText Library kommt ebenfalls noch ein `RtfWriter` mit. Das Einbinden verläuft gleich wie bei den vorangehenden Writer.

Bemerkungen. Einige Features werden im RTF Format noch nicht unterstützt: z.B. Phrasen mit einem Leading, verschachtelte Tabellen und das Rotieren von Bildern.

5. iText in Web-Applikationen

Um Dokumente aus einem Servlet heraus zu erstellen, gibt es zwei verschiedene Ansätze:

1. Das PDF Dokument wird auf dem Server erzeugt und gespeichert. Die Applikation verwendet ein `java.io.FileOutputStream` Objekt um das Dokument auf dem Filesystem abzulegen. In einem nächsten Schritt kann der Webuser das Dokument über das Hypertext Transfer Protocol (HTTP) anfordern (GET).
2. Das PDF Dokument wird auf dem Server erzeugt aber nicht gespeichert. Über die Angabe des Content Types (Type: Application, Subtype: pdf) kann dem Client direkt ein PDF Dokument über das HTTP Protokoll verschickt werden.

Im Rahmen dieses Semniarberichtes wird nur auf die zweite Variante eingegangen, da sie gegenüber der ersten gleich mehrere Vorteile bietet:

Für die erste Variante muss auf dem Server eine Datei gespeichert und dies macht eigentlich nur dann Sinn, wenn sie nach der Erzeugung für längere Zeit zur Verfügung stehen muss. Damit der Client auf das abgespeicherte Dokument zugreifen kann, muss ihm ein Link angegeben werden. Zusätzlich entsteht die Problematik des Wegräumens: das abgespeicherte Dokument muss wieder entfernt werden. All diese Probleme können mit der zweiten Variante vermieden werden.

5.1. Servlet

Bevor irgend welche Daten an den Client zurückgeschickt werden, müssen die HTTP Response Headers gesetzt werden. In diesem Fall unterscheidet sich vor allem der `Content-Type` von den 'herkömmlichen' Servlets.

`Content-type` Falls der `Content-type` nicht gesetzt wird, muss der Client selber herausfinden wie er das File handhaben will. Mit den `Content-type`s kann das Format festgelegt werden. In diesem Fall wird der `Content-type` wie folgt gesetzt:
`response.setContentType("application/pdf");`.

`Content-disposition` Mit dieser Angabe kann dem Client der Filename des Dokuments mitgeteilt werden. Zusätzlich kann der Client angewiesen werden, das Dokument im Browser anzuzeigen

oder aber eine externe Applikation zu verwenden. Hier:
`response.setHeader("Content-disposition", "inline; filename=response.pdf");`

`Content-length` Die Grösse des Dokumentes muss in Bytes angegeben werden. Dieser Parameter muss stimmen, da der Browser sonst das Dokument nicht darstellen kann.

Beispiel ITextServlet. Nachfolgend soll ein Beispiel eines Servlets besprochen werden. Den gesamten Code findet man im Anhang A.3 auf Seite 24. Dieses Beispiel greift auf eine PostgreSQL Datenbank zu. Die Verbindung zur Datenbank wird in der `init(...)` Methode aufgebaut, dieses Thema wird jedoch an dieser Stelle nicht ausführlicher besprochen (weitere Informationen dazu findet man auf den Webseiten [6] und [5]).

In der `doGet(...)` Methode des Servlets wird ein `ByteArrayOutputStream` deklariert, dieser soll den Daten Stream, welcher vom `PdfWriter` erzeugt wird, entgegennehmen. Die Methode `createPDFByteArrayStream()` führt dann die entsprechenden Queries auf der Datenbank aus, um einige Daten für das PDF Dokument zu erhalten. Im weiteren wird in den bereits bekannten fünf Schritten ein PDF Dokument erzeugt. Dabei wird dieses mal nicht wie bisher in ein `FileOutputStream` geschrieben (es soll ja keine Datei erzeugt werden). Statt dessen lääst man den `PdfWriter` in ein `ByteArrayOutputStream` schreiben.

```
Document doc = new Document();
ByteArrayOutputStream pdf_data;
PdfWriter.getInstance(doc, pdf_data);
```

Mit den Daten aus der Datenbank wird eine Tabelle zeilenweise erzeugt. Nachdem der Inhalt eingefügt wurde, wird mit `doc.close()` sichergestellt, dass der `PdfWriter` seine Daten vollständig im `ByteArrayOutputStream` abgelegt hat. Dieser wird nun an die `doGet(...)` Methode zurück gegeben.

Die `doGet(...)` Methode definiert die HTTP Response Headers. Der erzeugte PDF Daten Stream wird danach wie folgt an den Client versendet.

```
ServletOutputStream sos;
sos = response.getOutputStream();

bstream.writeTo(sos);
sos.flush();
```

Der Client erhält somit das PDF Dokument und wie in den HTTP Headers angegeben, wird das Dokument inline geöffnet:

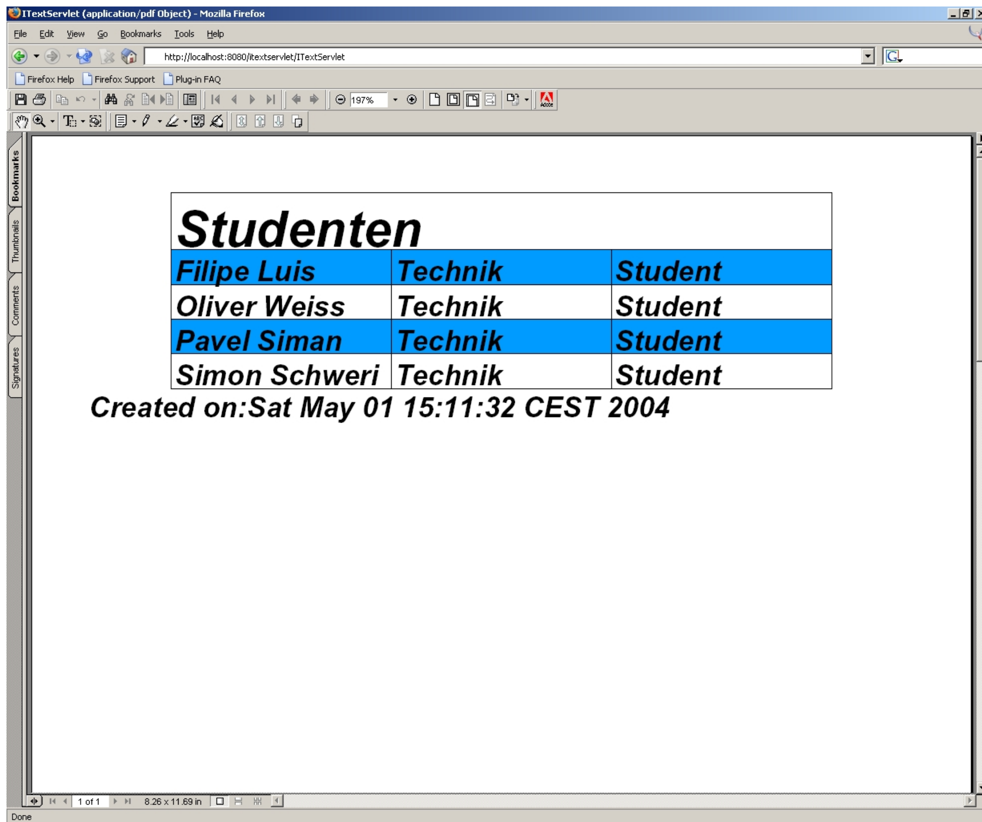

Abbildung 5.1.: Ein durch ein Servlet erzeugtes PDF Dokument.

Weiterführende Informationen zur dynamischen Erstellung von PDFs aus Web Applikationen sind auf der Website [1] zu finden.

6. iText & XML

Die iText Library liefert im Paket `com.lowagie.text.xml` diverse Klassen für die Erzeugung von PDF und RTF Dokumenten mit. Die Idee dabei ist recht einfach: Die Java Objekte wie zum Beispiel `Paragraph`, `List` oder `Chunk` sollen in XML abgebildet werden. Diese Abbildung der Elemente ist in der DTD `http://itext.sourceforge.net/itext.dtd` zu finden. Aus jedem XML Dokument welches diese `itext.dtd` referenziert, ist somit direkt ein PDF Dokument erzeugbar.

6.1. Tagmap für eigene XML Dokumente

In der Regel möchte ein Entwickler jedoch ein spezifisches XML Dokument verwenden, welches eine eigene DTD referenziert. In diesem Fall muss der Programmierer eine Tagmap schreiben, welche die eigenen DTD Elemente in die itext DTD Elemente abbildet. Die Tagmap muss dem Parser angegeben werden (`tagmap.xml`).

```
PdfWriter.getInstance(doc, new FileOutputStream("XML_doc.pdf"));
SAXParser parser = SAXParserFactory.newInstance().newSAXParser();
parser.parse("XML_Document.xml", new SAXmyHandler(doc, new TagMap("tagmap.xml")));
```

Für ein Zeitungsartikel wurde folgende DTD geschrieben (`itext_newspaper.dtd`):

```
<!ELEMENT NEWSPAPER (ARTICLE+)>
<!ELEMENT ARTICLE (AUTHOR,HEADLINE, BYLINE, BODY, NOTES)>
<!ELEMENT HEADLINE (#PCDATA)>
<!ELEMENT BYLINE (#PCDATA)>
<!ELEMENT BODY (#PCDATA)>
<!ELEMENT NOTES (LINES)*>
<!ELEMENT LINES (#PCDATA)>
<!ELEMENT AUTHOR (#PCDATA)>


<!ATTLIST ARTICLE
        EDITOR CDATA #IMPLIED
        DATE CDATA #IMPLIED
        EDITION CDATA #IMPLIED
>
```

Ein XML Dokument (`XML_Document.xml`) welches diese DTD referenziert:

```
ï»¿
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE NEWSPAPER SYSTEM "itext_newspaper.dtd">
<NEWSPAPER>
        <ARTICLE>
```

```
                <AUTHOR>Filipe Luis</AUTHOR>
                <HEADLINE>Schlagzeile</HEADLINE>
                <BYLINE>Untertitel</BYLINE>
                <BODY>Hier folgt der Artikel...</BODY>
                <NOTES>
                        <LINES>Notiz 1</LINES>
                        <LINES>Notiz 2</LINES>
                </NOTES>
        </ARTICLE>
</NEWSPAPER>
```

Und die Faltung der `itext_newspaper.dtd` auf die `itext` Elemente in der Tagmap:

```
<tagmap>
        <tag name="itext" alias="NEWSPAPER"/>
        <tag name="newline" alias="ARTICLE"/>
        <tag name="paragraph" alias="AUTHOR" content="Ã¼Krzel Autor: ">
                <attribute name="size" value="12"/>
                <attribute name="align" value="Left"/>
        </tag>
        <tag name="paragraph" alias="HEADLINE">
                <attribute name="leading" value="36"/>
                <attribute name="size" value="18"/>
                <attribute name="align" value="Center"/>
        </tag>
        <tag name="paragraph" alias="BYLINE">
                <attribute name="leading" value="20"/>
                <attribute name="size" value="14"/>
                <attribute name="align" value="Center"/>
                <attribute name="style" value="italic"/>
        </tag>
        <tag name="paragraph" alias="BODY">
                <attribute name="leading" value="18"/>
                <attribute name="size" value="12"/>
                <attribute name="align" value="Left"/>
        </tag>
        <tag name="list" alias="NOTES">
                <attribute name="numbered" value="false"/>
                <attribute name="symbolindent" value="0"/>
                <attribute name="listsymbol" value="o"/>
                <attribute name="symbolindent" value="15"/>
        </tag>
        <tag name="listitem" alias="LINES">
                <attribute name="size" value="11"/>
                <attribute name="align" value="Left"/>
                <attribute name="style" value="italic"/>
        </tag>
</tagmap>
```

A. Source Code

A.1. PdfCotnentByte

Listing PdfContentByteDemo.java:

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

import com.lowagie.text.*;
import com.lowagie.text.pdf.BaseFont;
import com.lowagie.text.pdf.PdfContentByte;
import com.lowagie.text.pdf.PdfWriter;

/*
 * Filipe Luis, IA01
 *
 * Fachhochschule Aargau, 16.04.2004
 *
 */

public class ContentByteDemo
{

        public static void main(String[] args)
        {
                //-- 1. Schritt: Document Instanz
                Document doc = new Document();

                try
                {
                        //-- 2. Schritt: Writer erzeugen
                        PdfWriter writer =
                                PdfWriter.getInstance(
                                        doc,
                                        new FileOutputStream("ContentByteDemo.pdf"));

                        //-- 3. Schritt: Dokumentö ffnen

                        HeaderFooter header =
                                new HeaderFooter(new Phrase("Beispiel ContentByteDemo"), false);

                        doc.setHeader(header);

                        doc.open();

                        //-- 4. Schritt: Inhalt ühinzufgen
                        PdfContentByte cb = writer.getDirectContent();
                        PdfContentByte cbu = writer.getDirectContentUnder();
                        BaseFont bf =
                                BaseFont.createFont(
                                        BaseFont.HELVETICA,
                                        BaseFont.CP1252,
                                        BaseFont.NOT_EMBEDDED);

                        cb.beginText();

                        cb.setFontAndSize(bf, 14);
                        cb.setTextMatrix(0, 0);
```

```
                        cb.showText("Urspung Koodrinatensystem (0,0).");

                        cb.setTextMatrix(100, 700);
                        cb.showText("Text at position 100,700.");

                        final float pi_4 = (float) (Math.sqrt(2) / 2.0);

                        cb.setTextMatrix(2, 0, 0, 2, 200, 500);
                        cb.showText("Hello Audience!");

                        cb.setTextMatrix(1.5f, pi_4, -pi_4, 1.5f, 200, 500);
                        cb.showText("Hello Audience!");

                        cb.setTextMatrix(0, 1, -1, 0, 200, 500);
                        cb.showText("Hello Audience!");

                        cb.setTextMatrix(100, 450);
                        cb.showText("Tessin, Oktober 2003");
                        //-- Bild soll unmittelbar nach Text erscheinen cb.getYTLM()...
                        Image image = Image.getInstance("absolute.jpg");
                        image.scalePercent(20);

                        float x = cb.getXTLM();
                        float y = cb.getYTLM() - (6 + image.plainHeight());
                        System.out.println(
                                "Absolute Position of Picture is: (" + x + "," + y + ")");
                        image.setAbsolutePosition(x, y);
                        doc.add(image);

                        cbu.setFontAndSize(bf, 16);
                        cbu.setTextMatrix(10,400);
                        cbu.showText("Hinter dem Bild (Tessin, Oktober 2003)");

                        cb.endText();

                        Font pfont = new Font(Font.HELVETICA, 14, Font.ITALIC);
                        Paragraph p = new Paragraph(new Phrase("Durch interner Layer gesetzt", pfont));

                        p.setLeading(12);
                        doc.add(p);

                        //-- 5. Schritt: Dokument schliessen
                        doc.close();
                        System.out.println("ContentByteDemo.pdf created!");

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

A.2. Listen

Listing ListDemo.java:

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

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

```
/*
 * Filipe Luis, IA01
 *
 * Fachhochschule Aargau, 16.04.2004
 *
 */

public class ListDemo
{

        public static void main(String[] args)
        {
                //-- 1. Schritt: Document Instanz
                Document doc = new Document();

                try
                {
                        //-- 2. Schritt: Writer erzeugen
                        PdfWriter.getInstance(doc, new FileOutputStream("ListDemo.pdf"));

                        //-- 3. Schritt: Dokumentö ffnen
                        doc.open();

                        //-- 4. Schritt: Inhalt ühinzufgen

                        List list = new List(false, 15);
                        list.setListSymbol("o");
                        list.add(new ListItem("Erstes Element"));

                        List sublist = new List(true, 20);
                        sublist.add(new ListItem("Erstes Subelement"));
                        sublist.add(
                                new ListItem("Zweites Subelement: Besonders "
                                + "lang damit wir sehen, wie die Zeile in Listen umgebrochen wird."));
                        sublist.add(new ListItem("Drittes Subelement"));

                        list.add(sublist);
                        list.add(new ListItem("Zweites und letztes Element"));
                        doc.add(list);

                        //-- 5. Schritt: Dokument schliessen
                        doc.close();
                        System.out.println("ListDemo.pdf created!");

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

A.3. Servlets

Listing `ITextServlet.java`:

```
import java.awt.Color;
import java.io.*;
import java.sql.*;

import javax.servlet.*;
import javax.servlet.http.*;
```

```
import com.lowagie.text.Document;
import com.lowagie.text.DocumentException;
import com.lowagie.text.Font;
import com.lowagie.text.FontFactory;
import com.lowagie.text.Paragraph;
import com.lowagie.text.Phrase;
import com.lowagie.text.pdf.PdfPCell;
import com.lowagie.text.pdf.PdfPTable;
import com.lowagie.text.pdf.PdfWriter;

public class ITextServlet extends HttpServlet
{

	private Connection dbcon; // Connection for scope of ShowBedrock

	//-- Init Methode: Aufbau einer Datenbankverbindung
	public void init(ServletConfig config) throws ServletException
	{
		//-- Login angaben
		String loginUser = "postgres";
		String loginPasswd = "itext";
		String loginUrl =
			"jdbc:postgresql://da8260i.cs.fh-aargau.ch:5432/itext";

		//-- Laden des PostgreSQL Treibers
		try
		{
			Class.forName("org.postgresql.Driver");
			dbcon =
				DriverManager.getConnection(loginUrl, loginUser, loginPasswd);
		}
		catch (ClassNotFoundException ex)
		{
			System.out.println("ClassNotFoundEx");
			System.err.println("ClassNotFoundException: " + ex.getMessage());
			throw new ServletException("Class not found Error");
		}
		catch (SQLException ex)
		{
			System.out.println("SQLEx");
			System.err.println("SQLException: " + ex.getMessage());
		}
	}

	//-- Implementation des GET Requests
	public void doGet(HttpServletRequest request, HttpServletResponse response)
		throws IOException, ServletException
	{

		ByteArrayOutputStream bstream = createPDFByteArrayStream();

		response.setContentType("application/pdf"); //-- Response Mime Typ
		response.setHeader("Content-disposition", "inline;students.pdf");
		response.setContentLength(bstream.size());

		ServletOutputStream sos;
		sos = response.getOutputStream();

		bstream.writeTo(sos);
		sos.flush();
	}

	private ByteArrayOutputStream createPDFByteArrayStream()
	{
		ByteArrayOutputStream bstream = null;

		try
		{
			//-- Statement vorbereitne
			Statement statement = dbcon.createStatement();

			String query = "SELECT *";
			query += "FROM   employee ";
```

```
                //-- Query üausfhren
                ResultSet rs = statement.executeQuery(query);

                //-- PDF Dokument erzeugen
                Document doc = new Document();
                doc.addAuthor("Filipe Luis");

                bstream = new ByteArrayOutputStream();
                PdfWriter writer = PdfWriter.getInstance(doc, bstream);

                doc.open();
                Phrase phrase =
                        new Phrase(
                                "Studenten",
                                FontFactory.getFont(
                                        FontFactory.HELVETICA,
                                        32,
                                        Font.BOLD | Font.ITALIC));
                PdfPTable table = new PdfPTable(3);
                PdfPCell cell = new PdfPCell(phrase);
                cell.setColspan(3);
                table.addCell(cell);

                int count = 0;
                String[] s = { "name", "dept", "jobtitle" };
                Font font = new Font(Font.HELVETICA, 18, Font.BOLDITALIC);
                //-- DB äEintrge in Tabelle üeinfgen

                        while (rs.next())
                        {

                                for (int i = 0; i < s.length; i++)
                                {
                                        String entry = rs.getString(s[i]);
                                        phrase = new Phrase(entry, font);
                                        cell = new PdfPCell(phrase);
                                        if (count % 2 == 0)
                                                cell.setBackgroundColor(new Color(0, 155, 255));
                                        table.addCell(cell);
                                }
                                count++;
                        }

                doc.add(table);
                doc.add(
                        new Paragraph(18, "Created on:" + new java.util.Date(), font));
                doc.close();
        }
        catch (DocumentException e)
        {
                System.err.println(e.getMessage());
        }
        catch (Exception ex)
        {
                System.err.println(ex.getMessage());
        }
        return bstream;
    }
}
```

Literaturverzeichnis

[1] Dynamically Crating PDFs in a Web Application. Article on http://www.onjava.com/, Zuletzt geprüft im Mai 2004.

[2] iText Tutorial. http://itext.sourceforge.net/docs/, Zuletzt geprüft im Mai 2004.

[3] iText Website. http://www.lowagie.com/iText/download.html, Zuletzt geprüft im Mai 2004.

[4] iTextSharp. http://itextsharp.sourceforge.net, Zuletzt geprüft im Mai 2004.

[5] Java Servlets, JSP, Jakarta-Tomcat, Database, Apache and Linux. Article on http://www.yolinux.com/TUTORIALS/, Zuletzt geprüft im Mai 2004.

[6] The PostgreSQL Database and Linux. Article on http://www.yolinux.com/TUTORIALS/, Zuletzt geprüft im Mai 2004.