Blog da Softblue

Este artigo foi criado por **Carlos Eduardo Gusso Tosin**.
Conheça o currículo completo do instrutor **clicando aqui**.

Gerando PDF em Java

Publicado em 24/05/2010 às 14:17:55 horas.

O formato PDF é muito utilizado porque se trata de uma forma fácil de distribuir documentos, além de ser possível visualizá-los em qualquer plataforma.
Como documentos PDF são amplamente utilizados, existem diversas maneiras de gerá-los. E é claro que o Java não poderia ficar de fora! Através do uso da API **iText**, você pode criar seus próprios arquivos PDF através do Java.

O objetivo deste artigo é mostrar como gerar documentos PDF em Java, usando a API iText. Você aprenderá como configurar os elementos da página do documento, além de como inserir textos, imagens e tabelas.

A API iText

O iText é a API utilizada para gerar PDFs. Ela é gratuita e open source. Para utilizar o iText em sua aplicação Java, entre no site http://itextpdf.com e baixe o arquivo .JAR referente ao iText 5.0.2. Depois basta adicionar o iText ao classpath da sua aplicação e utilizar as funcionalidades desta API, que vai possibilitar que você gere os seus documentos PDF. O iText funciona com a versão 5 do Java ou superior.

Documento

É hora de começarmos a aprendar a usar o iText! E a primeira classe que você deve conhecer é a classe `Document`. Ela representa o documento PDF, e para criar um documento PDF do tamanho A4 no modo retrato, o código fica assim:

```
1 Document doc = new Document(PageSize.A4);
```

A classe `PageSize` possui diversas constantes para tamanhos comuns de páginas (LETTER, NOTE, A0, A1, A2, etc.). É possível também definir o tamanho manualmente, desta forma:

```
1 Rectangle rect = new Rectangle(595, 841);
2 Document doc = new Document(rect);
```

Neste caso, é usada a classe `Rectangle`, que representa as dimensões de um retângulo.

Sistema de medidas

É importante destacar como funcionam as medidas utilizadas pelo iText. Se você quiser especificar as medidas das páginas dos documentos ou das margens, você deve ter em mente que você deve utilizar pontos. Como estamos mais acostumados a trabalhar na escala métrica, você pode escrever um código que faça a conversão da escala métrica para pontos. A relação é: **2,54cm = 72 pontos**.

No exemplo anterior, o uso de 595x841 pontos define as páginas do documento como tendo tamanho 21,0x29,7 cm, o que equivale ao tamanho A4.

Margens das Páginas

Para definir margens nas páginas do documento, basta que elas sejam especificadas no construtor da classe `Document`, como é feito a seguir:

```
1 Document doc = new Document(PageSize.A4, 72, 72, 72, 72);
```

Lembre-se de especificar as margens em pontos, e não no sistema métrico! O código acima define que o tamanho das páginas será A4 com margens de 2,54cm na esquerda, direita, em cima e embaixo.

Preparando o documento para receber dados

Após a criação do objeto que representa o documento, é necessário seguir mais alguns passos antes de iniciarmos a inserção dos dados. O código que representa este passo é mostrado a seguir:

```
1 OutputStream os = new FileOutputStream("out.pdf");
2 PdfWriter.getInstance(doc, os);
3 doc.open();
```

Vamos tentar entender o que é mostrado neste código. Primeiramente, devemos criar uma stream de saída, indicando o destino do documento PDF. Neste caso estamos interessados em gerar o documento em um arquivo, portanto criamos um `FileOutputStream`, indicando qual será o nome do arquivo de saída.

Na sequência, é necessário chamar o método `getInstance()` da classe `PdfWriter`, para que ele associe o documento à stream de saída.

O último passo antes de iniciarmos a criação do documento, é abri-lo. Isto é feito através da chamada ao método `open()`. É importante que, ao final da inserção dos dados no documento, o mesmo seja fechado através do método `close()`. E outro detalhe importante: todas as modificações no documento (como configuração de margens e tamanho de páginas) devem ser feitas antes da chamada ao método `close()`.

Gerando seu primeiro PDF em Java

Agora você já sabe tudo o que precisa para começar a escrever o seu primeiro arquivo PDF usando o iText. O código abaixo mostra um exemplo completo da geração de um arquivo PDF simples, usando o que você viu nesta parte inicial:

```
01 import java.io.FileOutputStream;
02 import java.io.OutputStream;
03 import com.itextpdf.text.Document;
04 import com.itextpdf.text.PageSize;
05 import com.itextpdf.text.Paragraph;
06 import com.itextpdf.text.pdf.PdfWriter;
07
08 public class PrimeiroPDF {
09     public static void main(String[] args) throws Exception {
10         Document doc = null;
11         OutputStream os = null;
12
13         try {
14             //cria o documento tamanho A4, margens de 2,54cm
15             doc = new Document(PageSize.A4, 72, 72, 72, 72);
16
17             //cria a stream de saída
18             os = new FileOutputStream("out.pdf");
19
20             //associa a stream de saída ao
21             PdfWriter.getInstance(doc, os);
22
23             //abre o documento
24             doc.open();
25
26             //adiciona o texto ao PDF
27             Paragraph p = new Paragraph("Meu primeiro arquivo PDF!");
28             doc.add(p);
29
30         } finally {
31             if (doc != null) {
32                 //fechamento do documento
33                 doc.close();
34             }
35             if (os != null) {
36                //fechamento da stream de saída
37                os.close();
38             }
39         }
40     }
41 }
```

Este é o esqueleto básico de código que você irá utilizar ao criar seus arquivos PDF, mudando apenas o conteúdo. Na sequência você verá mais detalhes sobre a classe `Paragraph`.

Textos

A forma mais fácil de adicionar textos ao PDF é utilizar a classe `Paragraph`. Esta classe permite que você adicione blocos de texto ao documento. O texto é inserido na página como se você estivesse utilizando um editor de textos. O iText respeita as margens que você especificou para o documento e também gerencia automaticamente as quebras de linha. Observe o código abaixo, que gera dois blocos de texto como saída:

```
1 Paragraph p1 = new Paragraph("Meu primeiro arquivo PDF!");
2 doc.add(p1);
3
4 Paragraph p2 = new Paragraph("Estou utilizando a classe Paragraph para criar um
  bloco de texto na geração do meu primeiro arquivo PDF.");
5 doc.add(p2);
```

O construtor da classe `Paragraph` recebe o texto como parâmetro. Depois basta chamar o método `add()` para adicionar o texto ao PDF.

Alterando a fonte

Nem sempre a fonte utilizada por padrão pelo iText é a fonte que você deseja nos seus documentos. Logo, é possível que você altere a fonte dos textos que você adiciona ao PDF.

A fonte é representada pela classe `com.itextpdf.text.Font`. Basta criar um objeto desta classe com a fonte desejada e associá-lo ao texto. Observe o exemplo:

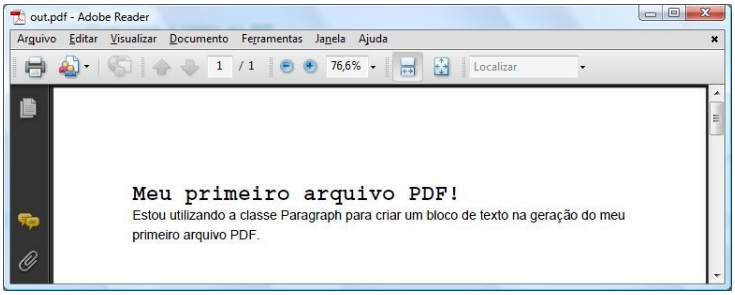
```
1 Font f = new Font(FontFamily.COURIER, 20, Font.BOLD);
```

O código acima cria uma fonte da família Courier, tamanho 20, itálica. O enum `FontFamily` e a classe `Font` possuem também outras constantes para você explorar.

A associação entre a fonte e o texto é feita no construtor da classe `Paragraph`, desta forma:

```
1 Paragraph p1 = new Paragraph("Meu primeiro arquivo PDF!", f);
```

Pronto! Você já tem um título para o seu PDF! Observe como ficou o resultado:

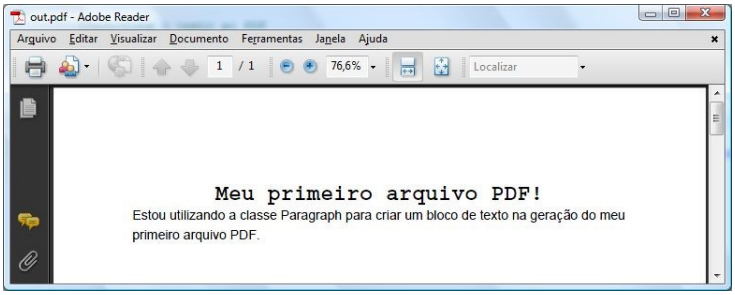


Alinhamento

Muitas vezes o alinhamento à esquerda não é o suficiente. O iText proporciona outras formas de alinhamento do texto. Observe:

```
1 p1.setAlignment(Element.ALIGN_CENTER);
```

O método `setAlignment()` permite especificar um alinhamento para o bloco. Neste caso, `p1` será alinhado no centro da linha. Observe:

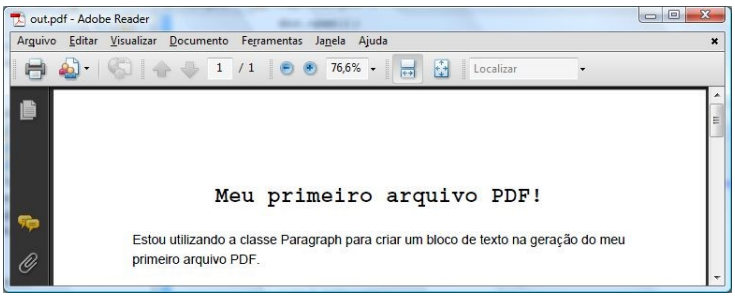


Espaçamento

Caso você deseje ter um controle maior sobre o espaçamento utilizado antes e depois de um bloco de texto, você pode utilizar os métodos `setSpacingBefore()` e `setSpacingAfter()`. Observe:

```
1 p1.setSpacingAfter(20);
```

Neste exemplo, após `p1` ser adicionado ao documento, será deixado um espaço em branco extra de 20 pontos. Observe:



Imagens

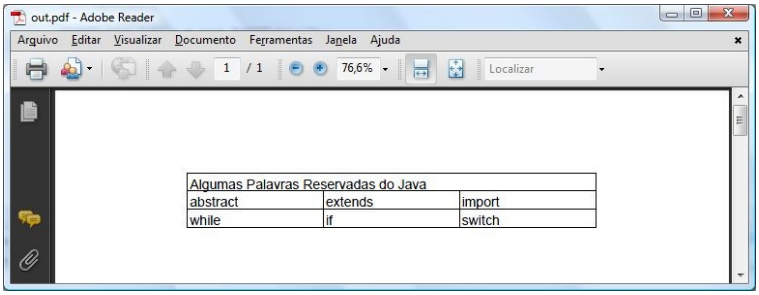
Da mesma forma que textos, imagens também podem ser inseridas no documento PDF. A maneira mais fácil de fazer isto é através da classe `com.itextpdf.text.Image`. Ela é utilizada desta forma:

```
Image img = Image.getInstance("softblue.jpg");
img.setAlignment(Element.ALIGN_CENTER);
doc.add(img);
```

Basta chamar o método `getInstance()` da classe, passando o caminho da imagem que deve ser carregada, e depois adicionar a imagem ao documento através do método `add()`. Os métodos `setAlignment()`, `setSpacingBefore()` e `setSpacingAfter()` já vistos anteriormente também podem ser aplicados às imagens. Observe como fica o resultado:

Tabelas

O iText possui também suporte à criação de tabelas. Observe uma tabela simples abaixo que contém algumas palavras reservadas do Java:



O código que gera a tabela acima é o seguinte:

```
PdfPTable table = new PdfPTable(3);
PdfPCell header = new PdfPCell(new Paragraph("Algumas Palavaras Reservadas do Java"));
header.setColspan(3);
table.addCell(header);
table.addCell("abstract");
table.addCell("extends");
table.addCell("import");
table.addCell("while");
table.addCell("if");
table.addCell("switch");
doc.add(table);
```

Vamos tentar entender o que significa este código. Ao trabalhar com tabelas no iText, você basicamente vai usar duas classes: `PdfPTable` e `PdfPCell`. A primeira representa a tabela, enquanto a última representa uma célula da tabela.

A chamada `new PdfPTable(3)` cria uma tabela com 3 colunas, de forma que a largura das colunas é a mesma. Veremos na sequência como especificar larguras diferentes para cada coluna.

Depois criamos um objeto da classe `PdfPCell`, que irá representar o cabeçalho da nossa tabela. No construtor do `PdfPCell` você pode passar um objeto `Paragraph` contendo o texto que será colocado na célula. Uma característica importante do cabeçalho é que ele deve ocupar as 3 colunas da tabela. Por este motivo nós usamos o método `setColspan(3)`.

Toda vez que desejamos adicionar uma célula à tabela, chamamos o método `addCell()`. Neste método podemos passar tanto objetos `PdfPCell` como os textos de cada célula diretamente (neste caso o próprio iText gera o objeto `PdfPCell` internamente). Perceba que em nenhum momento você diz em qual linha ou coluna da tabela a célula será inserida. O iText vai inserindo as células de cima para baixo e da esquerda para a direita. O gerenciamento da criação de novas linhas é feito de forma automática.

Depois de preparar a tabela, basta adicioná-la ao documento através da chamada `doc.add(table)`.

Definindo a Largura das Colunas

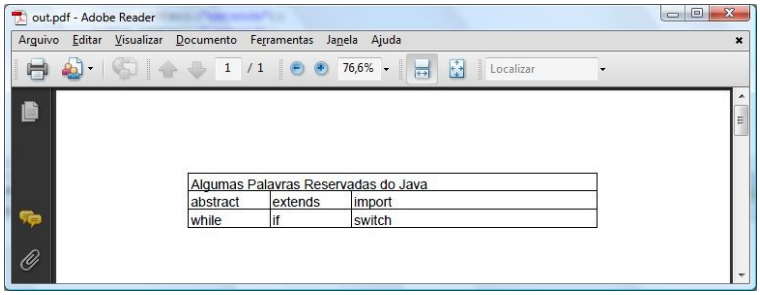
Como vimos anteriormente, ao construirmos uma tabela informando apenas o número de colunas, o iText deixa todas as colunas com o mesmo tamanho, o que nem sempre é o ideal.

A melhor forma de especificar a largura das colunas da tabela é usar a forma relativa. Ela funciona assim: você indica qual o tamanho da coluna com relação à largura total da tabela. Observe:

```
PdfPTable table = new PdfPTable(new float[] { 0.2f, 0.2f, 0.6f });
```

Ao passar um array de números no construtor da `PdfPTable`, você está indicando um tamanho relativo para cada coluna. Como o array tem 3 elementos, a tabela terá três colunas. Agora você deve pensar da seguinte

forma: todas as colunas da tabela juntas representam 100%. E, neste caso, a primeira coluna representa 20%, a segunda 20% e a terceira 60%. Perceba que as duas primeiras colunas terão o mesmo tamanho, e a terceira coluna será três vezes maior que cada coluna anterior. O resultado final fica assim:

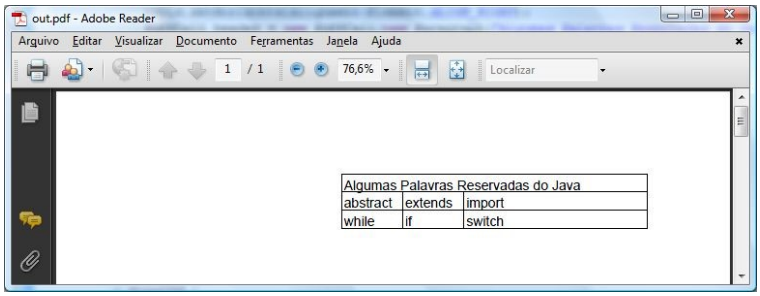


Mudando o tamanho e alinhamento da tabela

O iText permite também que você especifique qual o tamanho e o alinhamento da tabela com relação do documento. Isto é feito através dos métodos `setWidthPercentage()` e `setHorizontalAlignment()`. Observe:

```
1 table.setWidthPercentage(60.0f);
2 table.setHorizontalAlignment(Element.ALIGN_RIGHT);
```

No exemplo acima, estamos definindo que a tabela deve ocupar apenas 60% da largura do documento. Também estamos definindo que ela deve ser alinhada à direita. Observe o resultado:

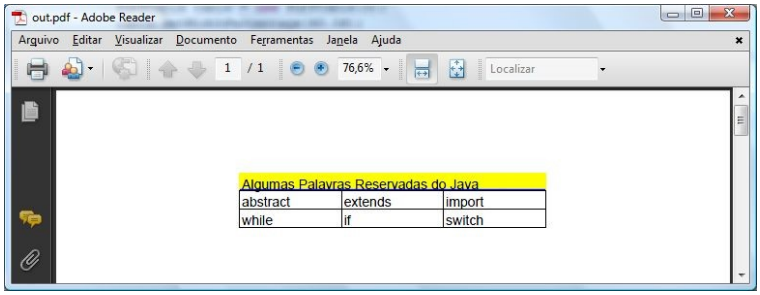


Customizando as bordas e cores

É possível também customizarmos as bordas das células e as cores das bordas e das células. Observe o código abaixo:

```
1 header.setBackgroundColor(BaseColor.YELLOW);
2 header.setBorderWidthBottom(2.0f);
3 header.setBorderColorBottom(BaseColor.BLUE);
4 header.setBorder(Rectangle.BOTTOM);
```

Ele gera como resultado:



O método `setBackgroundColor()` define a cor de fundo da célula. O método `setBorderWitdhBottom()` define a largura da linha da borda inferior. O método `setBorderColorBottom()` define a cor da borda inferior. E o método `setBorder()` define em que parte das célula a borda deve existir. Usando `Rectangle.BOTTOM`, estamos definindo que existe apenas uma borda inferior na célula.

É importante que você saiba que existem variantes dos métodos que definem a largura e cor da linha da borda. Utilizamos neste exemplo os método `setBorderWidthBottom()` e `setBorderColorBottom()`, para a borda inferior, mas é possível também usar os métodos equivalentes `top`, `left` e `right` (para borda superior, esquerda e direita, respectivamente.

Conclusão

Este artigo introduziu a API iText, usada para gerar documentos PDF em Java. O que está descrito aqui é apenas uma visão inicial de algumas funcionalidades importantes do iText. Mas muita coisa além do que está descrito aqui pode ser feita. Consulte a página oficial da API para ter maiores informações.

Comentários

bommesmo

Enviado em 30/12/2012 às 14:58:20 horas, por **bom**

Parabéns, ajudou-me muito no meu trabalho da faculdade!! Obrigado!

Enviado em 14/11/2012 às 19:55:05 horas, por **José Cordeiro**

André, não sei o que pode estar acontecendo, pois nunca tive esse problema. Sugiro que você procure na documentação do iText, mais precisamente na classe `Image`, que deve ter alguma coisa por lá. Abraço!

Enviado em 01/11/2012 às 13:31:12 horas, por **Carlos Tosin**

Bom dia a todos.

Ao adicionar uma imagem no pdf notei que a mesma está redimensionando (não ficando no tamanho original) como faço para deixá-la no tamanho original ?

E parabéns pelo tutorial muito útil.

Enviado em 01/11/2012 às 10:44:53 horas, por **André Souza**

Bom artigo, mas lembrando que existe a opção de utilizar plugins nas IDE's como o Jasper e IReport que facilitam muito a parte visual.

Enviado em 01/10/2012 às 15:01:56 horas, por **Henrique**

Muito bom mesmo o artigo. É tudo o que eu precisava. Parabéns pela apresentação e pela didática.

Enviado em 05/09/2012 às 20:21:58 horas, por **GALVANI LUPPI**

Ótimo tópico e perfeita explicação, ajudaram muito.
Obrigado.

Enviado em 03/09/2012 às 05:45:05 horas, por **Maurício Cayres**

Danilo, não sei o que pode estar acontecendo. Apenas com estas informações é difícil saber. Se com o debug você não conseguiu descobrir, recomendo que você grave algumas mensagens de log em arquivo para que o código que você executa a partir do JAR dê a você mais informações sobre o que está acontecendo. Abraço!

Enviado em 30/08/2012 às 16:42:07 horas, por **Carlos Tosin**

---

ola, estou com um problema dentro da ide que no caso e o netbeans funciona direitinho mas quando eu construo gero o .jar ele chega ate a gerar o pdf
mas fica como se nao tivesse o documento em si eu ja depurei o cod ele trava quando chega na parte do doc.close() os.close()

Enviado em 29/08/2012 às 11:19:08 horas, por **danilo**

Excelente tutorial, me ajudou, estava pesquisando sobre isso para o meu TCC e basicamente tudo que eu precisava sobre salvamento de arquivos em PDF achei aqui, muito obrigado.

Enviado em 27/08/2012 às 20:18:04 horas, por **Felipe**

Bruno, essa questão da geração do arquivo não está relacionada com o iText, mas sim com a própria API de I/O do Java. Se você usar o `FileOutputStream`, o arquivo será sobrescrito toda vez que você executar o código. Existem outras opções também, então recomendo que você tente se familiarizar mais com a API de I/O. Abraço!

Enviado em 13/05/2012 às 22:01:29 horas, por **Carlos Tosin**

---

olá,
Criei um metodo atravez dessa classe, porem se o pdf foi criado uma vez, ele nao cria outro, tenho que exclui-lo para assim conseguir gerar outro PDF.
Tem alguma solucao ?
Obrigado

Enviado em 13/05/2012 às 13:14:27 horas, por **Bruno**

Thiago, não existe uma forma certa ou errada de fazer. O que existe é a forma que mais se adapta à sua necessidade, lembrando sempre que é importante deixar o seu código bem estruturado. Você deve analisar isso quando for integrar sua aplicação com o iText. Abraço!

Enviado em 25/04/2012 às 11:52:32 horas, por **Carlos Tosin**

Muito Obrigado pelo Tópico, Professor!

Estou Aprendendo a fundo mesmo a Linguagem java e Delphi , mas tenho uma dúvida é a seguinte : onde posso colocar o código acima dentro de um Evento de Um Botão em Aplicativo java , como por exemplo ActionPerformed ?

Exemplo:

private void jButton1.ActionPerformed(java.awt.event.actionEvent evt) {

//o código do iText aqui

}

Seria assim ou de outra maneira?

Enviado em 24/04/2012 às 19:51:31 horas, por **Thiago Vinícius dos Santos - Rio de Janeiro**

Marcos, na hora de criar o `FileOutputStream` basta você fornecer o caminho completo do arquivo como parâmetro (ex: `C:\\Arquivos\\arquivo.pdf`). Abraço!

Enviado em 14/04/2012 às 16:32:32 horas, por **Carlos Tosin**

---

muito bom seu artigo! mas eu queria saber como faço pra direcionar o pdf para uma pasta especifica?

Enviado em 14/04/2012 às 14:27:29 horas, por **marcos**

---

alguem sabe ou tem um exemplo de gerar relatorio com o itext com valores do banco de dados?? por favor, obrigado!

Enviado em 31/10/2011 às 17:04:40 horas, por **Nathan**

CARLOS, esta API faz apenas a geração de PDFs. Recomendo que você dê uma procurada na internet atrás de APIs que fazem o inverso. Abraço!

Enviado em 28/10/2011 às 13:49:36 horas, por **Carlos Tosin**

E como eu converto do pdf gerado para texto novamente?

Enviado em 28/10/2011 às 11:43:09 horas, por **CARLOS**

---

como eu posso deixar os elementos de uma tabela centrrializados.

Enviado em 05/10/2011 às 12:24:05 horas, por **Julio Machado**

Muito boa explicação. Valeu mesmo!!!

Enviado em 04/03/2011 às 22:17:57 horas, por **Vítor**

---

muito bom este artigo, só tenho uma pergunta, se eu quiser pegar vários componentes de tela que es´tão dentro de um JPanel, também é possivel ?

Enviado em 11/01/2011 às 08:21:39 horas, por **Diogo**

Simplesmente juntar os bytes dos arquivos para formar um PDF só não é possível. Mas você pode dar uma investigada na API se existe a possibilidade de criar um novo documento e ir adicionando novas páginas, de forma que cada página seja dos PDFs que você quer juntar.

Enviado em 19/11/2010 às 17:03:13 horas, por **Carlos Tosin**

Tem como eu juntar todos esse pdf's gerados em apenas um? Me ajuda por favor.
Obrigado.

Enviado em 19/11/2010 às 12:49:13 horas, por **Cassio**

Ricardo, o método `getInstance()` da classe `Image` pode receber um nome de uma imagem (como mencionado no artigo), mas também pode receber um array de bytes. Basta você fazer `Image img = Image.getInstance(bytes)`. O array de bytes pode ser a imagem armazernada no banco de dados, como você mencionou.

Enviado em 18/11/2010 às 23:33:35 horas, por **Carlos Tosin**

Como eu posso adicionar uma imagem em um pdf que está no Banco de dados? e é recuperado em byte[]?

Enviado em 18/11/2010 às 18:42:41 horas, por **Ricardo rosa**

O que foi explicado neste artigo gera o arquivo PDF no sistema de arquivos (no local desejado). Então o arquivo fica gravado lá. Depois basta você ir até ele e carregá-lo.

Enviado em 08/11/2010 às 20:53:04 horas, por **Carlos Tosin**

Muito boa a explicação, só queria saber,se quando você usa este método, ele já abre o arquivo PDF ao salvá-lo?

Enviado em 08/11/2010 às 19:57:59 horas, por **Janser Lemes**

Professor Carlos, achei muito legal a dica de gerar PDF, e aproveitando eu deu uma olha no curso oferecido, e assisti a aula de demonstração e achei Show-de-bola... Estou muito interessado em adquirir os Cursos... Aproveitando quero deixar uma dica de assunto a ser abordado pelo Blog: Digitalização de imagens ou documentos com Scanner.

Enviado em 20/07/2010 às 20:05:14 horas, por **Márcio**

Olá estou a procura de um aplicativo que le pfd em celulares para Java, se alguem souber de algum ou tenha desenvolvido algum favo e-mail para ricsurfpro@hotmail.com

Enviado em 24/05/2010 às 22:06:25 horas, por **Ricardo**

Olá Julio Cesar. Você pode pegar dados sim de um banco de dados e gerar uma nota fiscal em PDF. Quanto ao código de barras, você nem precisa da fonte do Windows que você mencionou. O iText já tem suporte à criação de códigos de barras. Você diz o número e ele renderiza como código de barras no PDF. Consulte a documentação do iText para mais informações sobre essa funcionalidade. Já usei na prática num sistema real e funciona bem!

Enviado em 24/05/2010 às 18:44:05 horas, por **Carlos Tosin**

Muito Bom Professor. Posso também pegar arquivos do banco de dados e colocar no pdf. Exemplo uma nota fiscal....

Pode ser chamado por um servlet para gerar na web... Ele pega as fontes do windows... Por exemplo se eu por a fonte de barras..dá pra gerar o codigo de barras...Se sim resolverá meus problemas..

Enviado em 24/05/2010 às 18:07:19 horas, por **Julio Cesar**

Usando apenas o iText, não é possível. Existe uma API chamada POI, feita pela Apache (http://poi.apache.org) que permite a leitura e escrita de documentos do Microsoft Word. Daí dá para tentar criar alguma aplicação Java que use o POI para ler um .doc e use o iText para gerar o PDF.

Enviado em 24/05/2010 às 16:29:10 horas, por **Carlos Tosin**

Muito boa a explicação. Parabéns.
Uma pergunta com esta API não é possível buscar um documento .doc por exemplo, e transformá-lo em PDF?

Enviado em 24/05/2010 às 16:10:40 horas, por **Mélory Zolino**