

Java Starter

www.t2ti.com

Curso Java Starter

Apresentação

O Curso Java Starter foi projetado com o objetivo de ajudar àquelas pessoas que têm uma base de lógica de programação e desejam entrar no mercado de trabalho sabendo Java,

A estrutura do curso é formada por módulos em PDF e por mini-cursos em vídeo. O aluno deve baixar esse material e estudá-lo. Deve realizar os exercícios propostos. Todas as dúvidas devem ser enviadas para a lista de discussão que está disponível para inscrição na página do Curso Java Starter no site www.t2ti.com. As dúvidas serão respondidas pelos instrutores Albert Eije, Cláudio de Barros e Miguel Kojii, além dos demais participantes da lista.

Nosso objetivo é que após o estudo do Curso Java Starter o aluno não tenha dificuldades para acompanhar um curso avançado onde poderá aprender a desenvolver aplicativos para Web, utilizando tecnologias como Servlets e JSP e frameworks como Struts e JSF, além do desenvolvimento para dispositivos móveis.

Albert Eije trabalha com informática desde 1993. Durante esse período já trabalhou com várias linguagens de programação: Clipper, PHP, Delphi, C, Java, etc. Atualmente mantém o site www.alberteije.com.

Cláudio de Barros é Tecnólogo em Processamento de Dados.

Miguel Kojii é bacharel em Sistemas de Informação, profissional certificado Java (SCJP 1.5).

O curso Java Starter surgiu da idéia dos três amigos que trabalham juntos em uma instituição financeira de grande porte.

Módulo

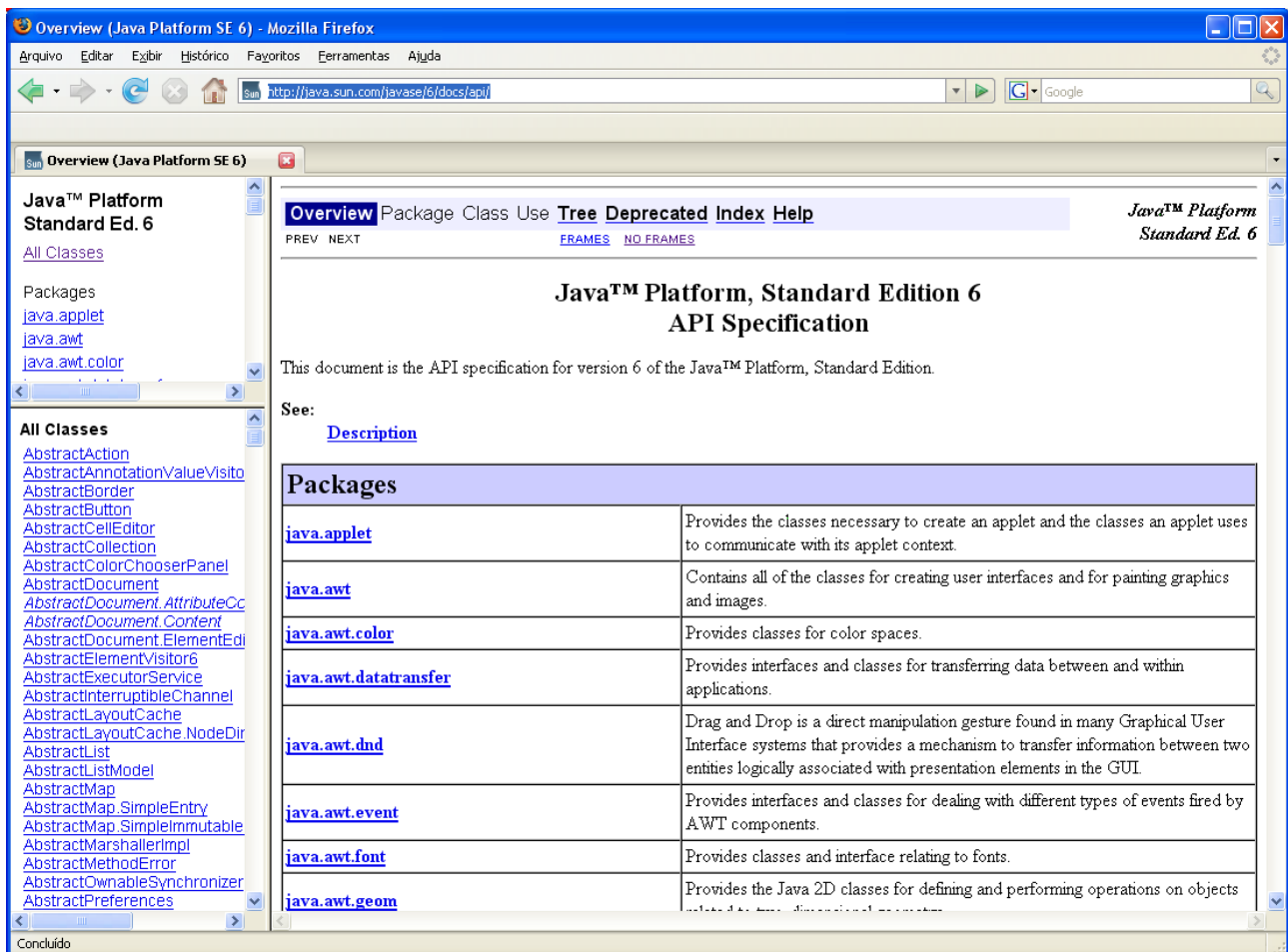
06

JAVADOC E ARQUIVOS "JAR"

JAVADOC

JAVADOC, é um utilitário para a geração da documentação das classes criadas em Java. Esta documentação é muito importante, pois é através dela que conhecemos melhor as classes, verificando quais métodos determinada classe possui e suas funcionalidades. Aprender utilizá-la é essencial.

A documentação da API Java 1.6 pode ser acessada no link: <http://java.sun.com/javase/6/docs/api/>

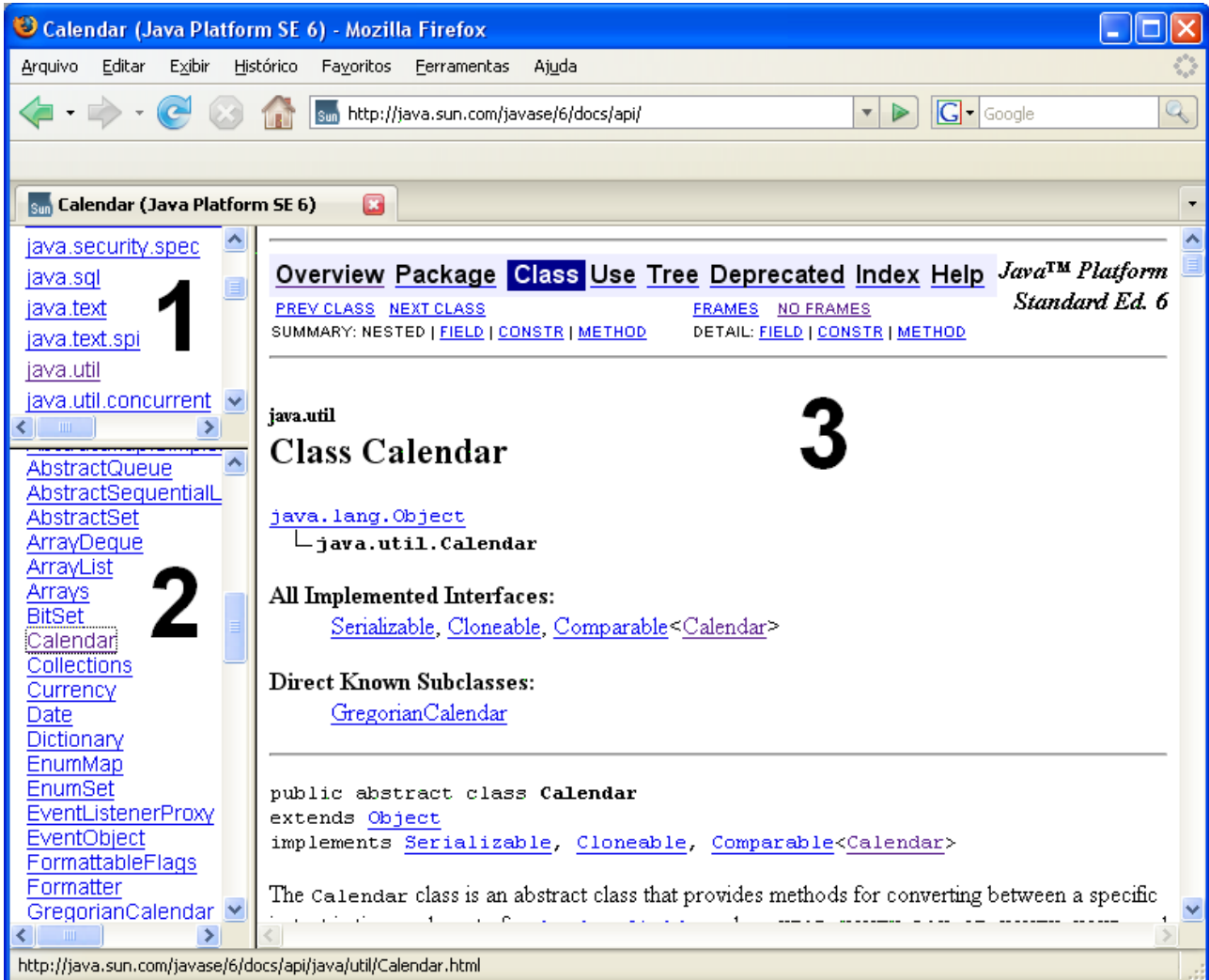


The screenshot shows a Mozilla Firefox browser window displaying the Java Platform SE 6 API Specification page. The browser's address bar shows the URL <http://java.sun.com/javase/6/docs/api/>. The page content includes a navigation menu with links for Overview, Package, Class, Use, Tree, Deprecated, Index, and Help. The main heading is "Java™ Platform, Standard Edition 6 API Specification". Below the heading, there is a description of the document and a "See:" section with a link to "Description". A "Packages" table is also visible, listing various Java packages and their descriptions.

Packages	
java.applet	Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context.
java.awt	Contains all of the classes for creating user interfaces and for painting graphics and images.
java.awt.color	Provides classes for color spaces.
java.awt.datatransfer	Provides interfaces and classes for transferring data between and within applications.
java.awt.dnd	Drag and Drop is a direct manipulation gesture found in many Graphical User Interface systems that provides a mechanism to transfer information between two entities logically associated with presentation elements in the GUI.
java.awt.event	Provides interfaces and classes for dealing with different types of events fired by AWT components.
java.awt.font	Provides classes and interface relating to fonts.
java.awt.geom	Provides the Java 2D classes for defining and performing operations on objects related to two-dimensional geometry.

Curso Java Starter

Na tela seguinte, estamos acessando a API da classe `java.util.Calendar`:



O campo representado por "1" contém a lista de pacotes. O campo representado por "2" contém a lista de classes e interfaces do pacote selecionado em "1". O campo representado por "3" contém a descrição da classe selecionada em "2".

Todo o detalhamento da classe vai estar no campo representado por "3".

No cabeçalho da página encontramos links para:

Overview – Mostra todos os pacotes da API Java.

Package – Mostra a descrição de todas as classes do pacote atual. Neste

Curso Java Starter

caso, java.util.

Class – classe que está sendo acessada.

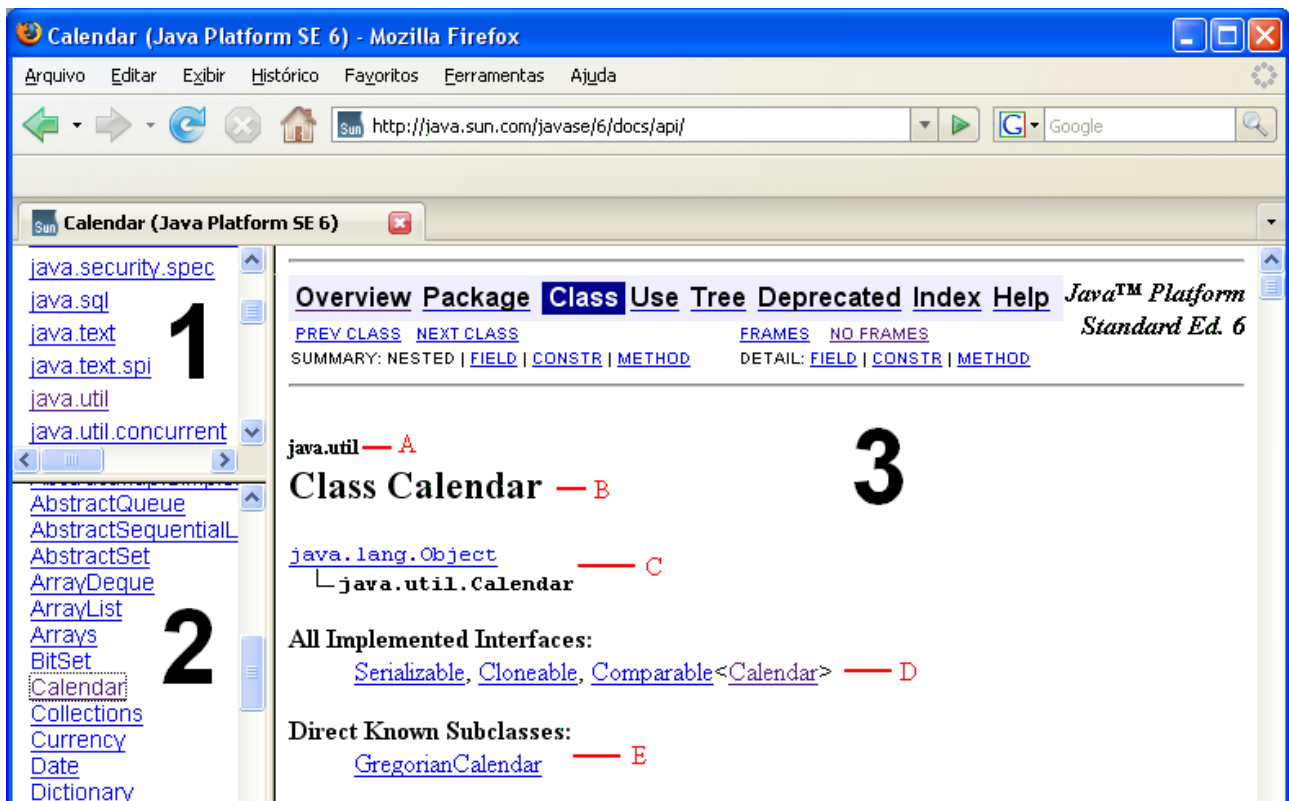
Use – Lista dos pacotes que a classe atual utiliza.

Tree – Hierarquia de classes no pacote atual.

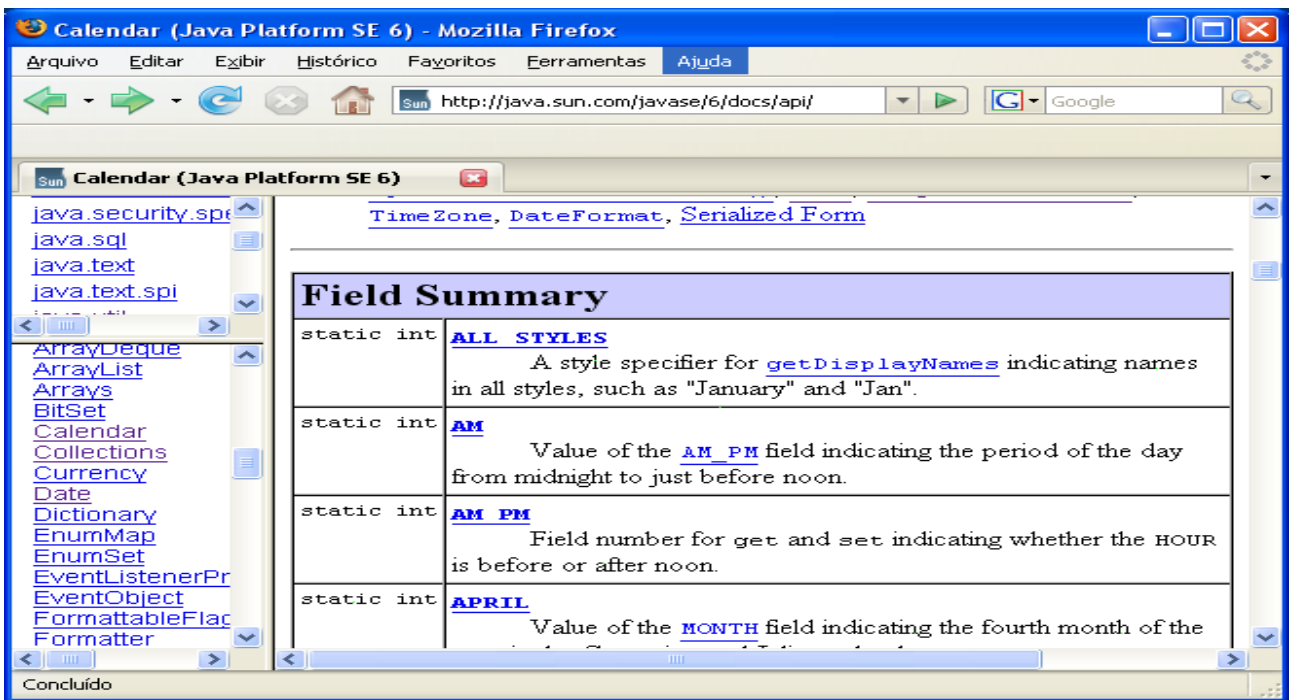
Deprecated – classes e métodos que não são recomendados utilizar.

Index – índice de métodos e campos. Muito útil quando não se sabe a classe.

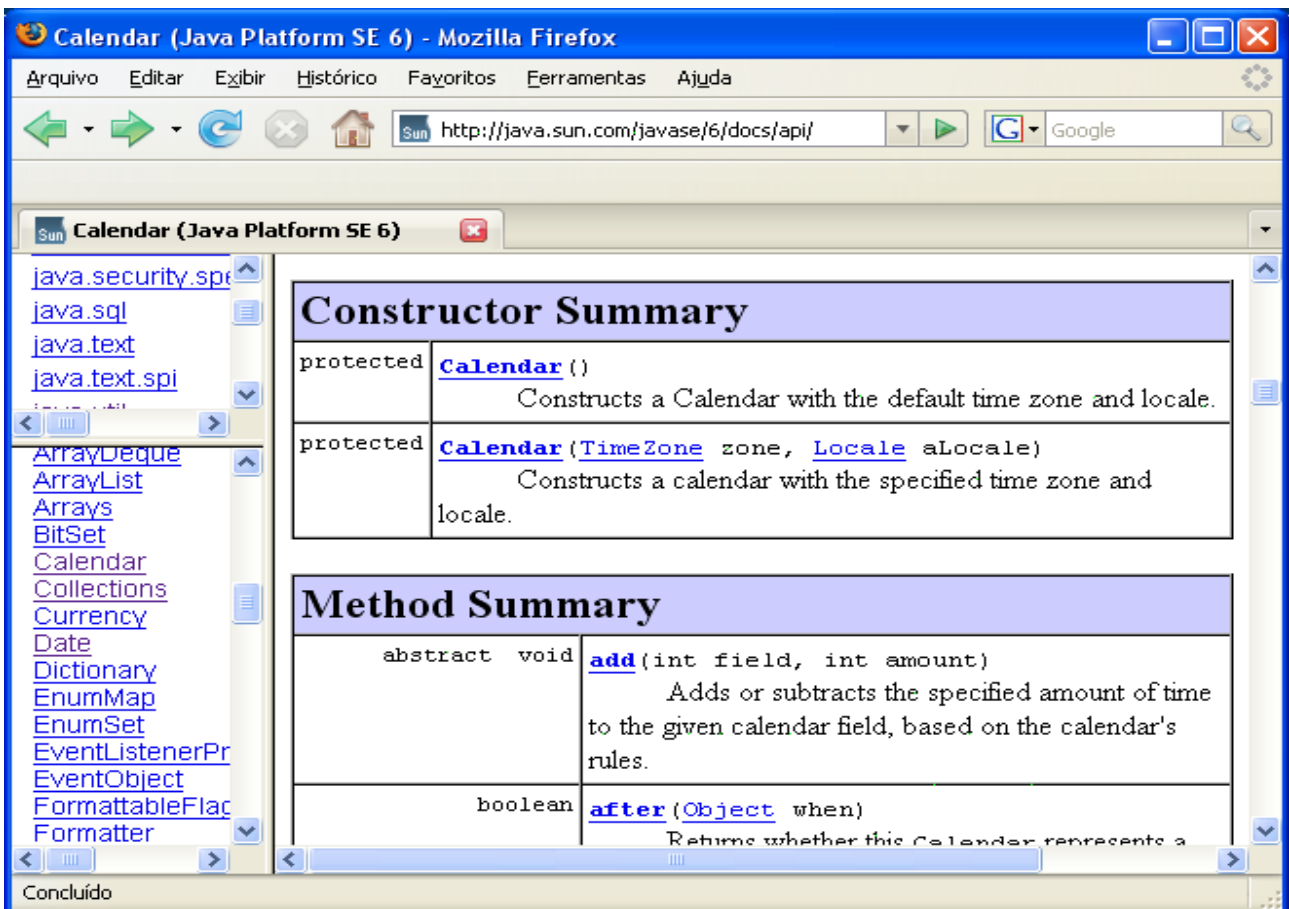
Help – ajuda de como utilizar a documentação.



Curso Java Starter

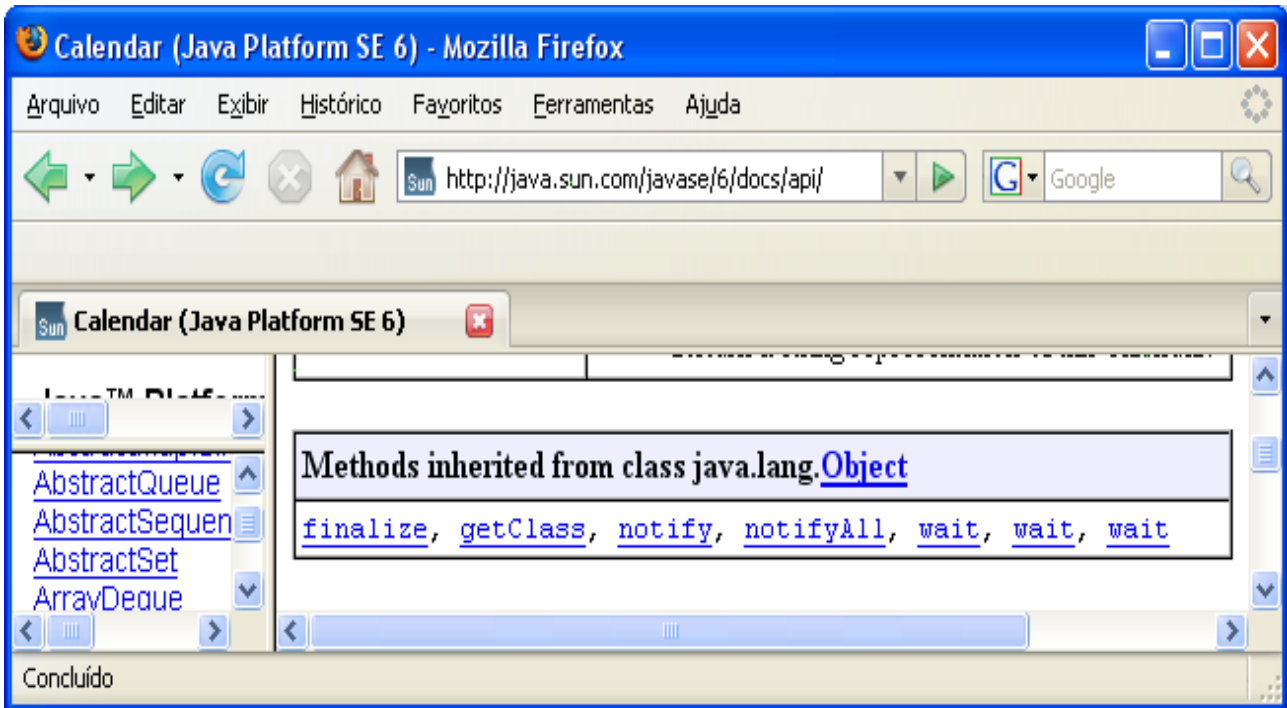


Nesta parte da tela temos a lista de campos da classe. Na primeira coluna temos o tipo(int, String, etc.) e na segunda temos o nome e a descrição do campo.



Curso Java Starter

Aqui temos a lista de construtores e a lista de métodos. Na lista de construtores, a primeira coluna mostra o modificador de acesso(public, protected, etc) e na segunda coluna mostra o nome e a descrição do construtor. Na lista de métodos, a primeira coluna mostra o tipo de retorno(void, boolean, etc.) e na segunda coluna mostra o nome e a descrição do método.



Nesta tela temos os métodos que são herdados da super-classe. Neste caso, a classe *java.util.Calendar* só herda os métodos da classe *java.lang.Object*.

Em cada nome de campo, construtor, método mostrados nas telas que vimos, tem um link direcionando para a sua respectiva descrição detalhada. Na tela abaixo vemos o detalhamento do método *getInstance(TimeZone)*.



Podemos observar aqui o modificador de acesso (**public**), o tipo de retorno (*Calendar*) o nome do método (*getInstance*) e o parâmetro (*TimeZone*).

Gerando o JAVADOC

Quando instalamos o JDK, o gerador de javadoc também é instalado. O mesmo fica na pasta "%JAVA_HOME%/bin/javadoc.exe".

Agora vamos criar um javadoc da classe a seguir:

```
package veiculo

public class CarroPasseio {

    public void acelerar() {
        //código...
    }

    public void frear(int intensidade) {
        //código
    }
}
```

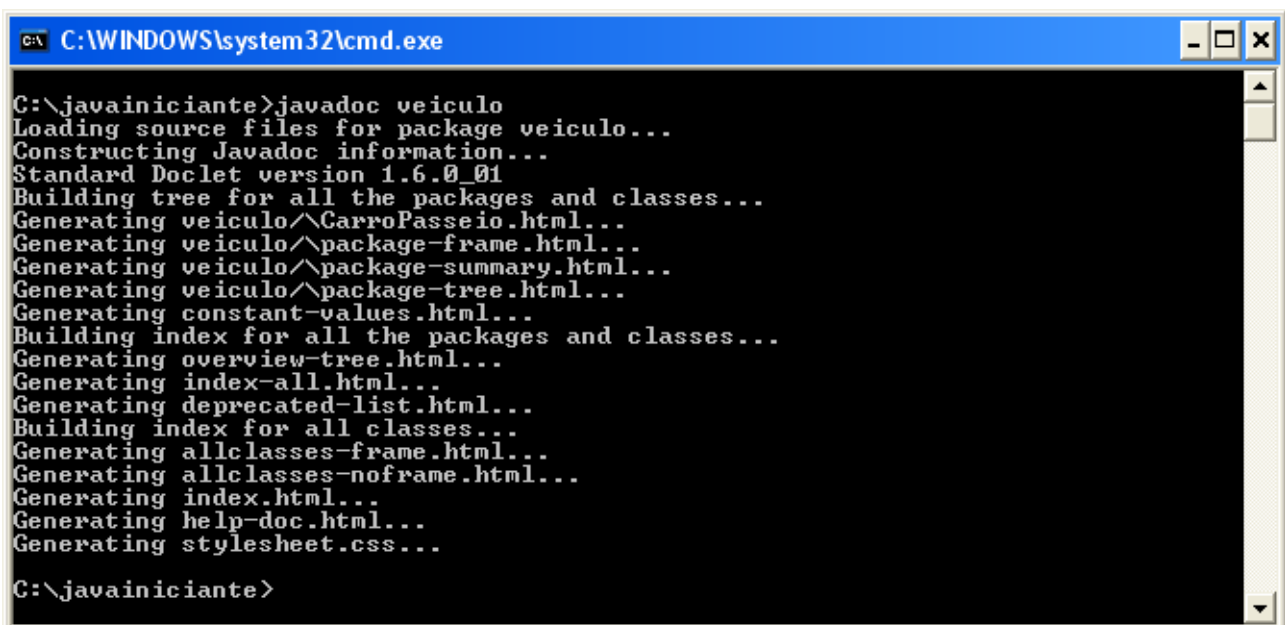

Curso Java Starter

```
}  
  
public int getVelocidade() {  
    //código...  
    return 0;  
}  
}
```

No prompt de comando digitamos o seguinte comando:

```
javadoc veiculo
```

veiculo --> nome do pacote



```
C:\WINDOWS\system32\cmd.exe  
  
C:\javainiciante>javadoc veiculo  
Loading source files for package veiculo...  
Constructing Javadoc information...  
Standard Doclet version 1.6.0_01  
Building tree for all the packages and classes...  
Generating veiculo\CarroPasseio.html...  
Generating veiculo\package-frame.html...  
Generating veiculo\package-summary.html...  
Generating veiculo\package-tree.html...  
Generating constant-values.html...  
Building index for all the packages and classes...  
Generating overview-tree.html...  
Generating index-all.html...  
Generating deprecated-list.html...  
Building index for all classes...  
Generating allclasses-frame.html...  
Generating allclasses-noframe.html...  
Generating index.html...  
Generating help-doc.html...  
Generating stylesheet.css...  
  
C:\javainiciante>
```

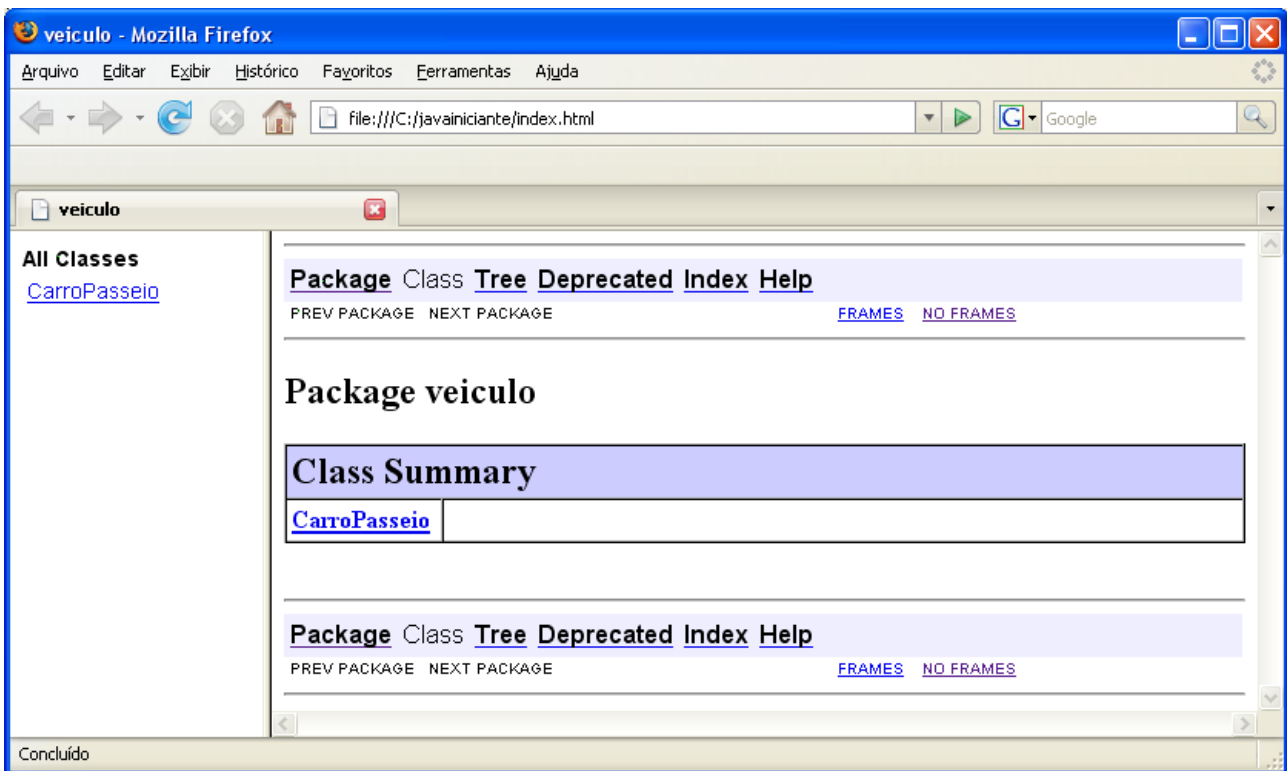
Será gerado javadoc para todas as classes que estiverem no pacote "veiculo".

Se tiver mais de um pacote, faça assim:

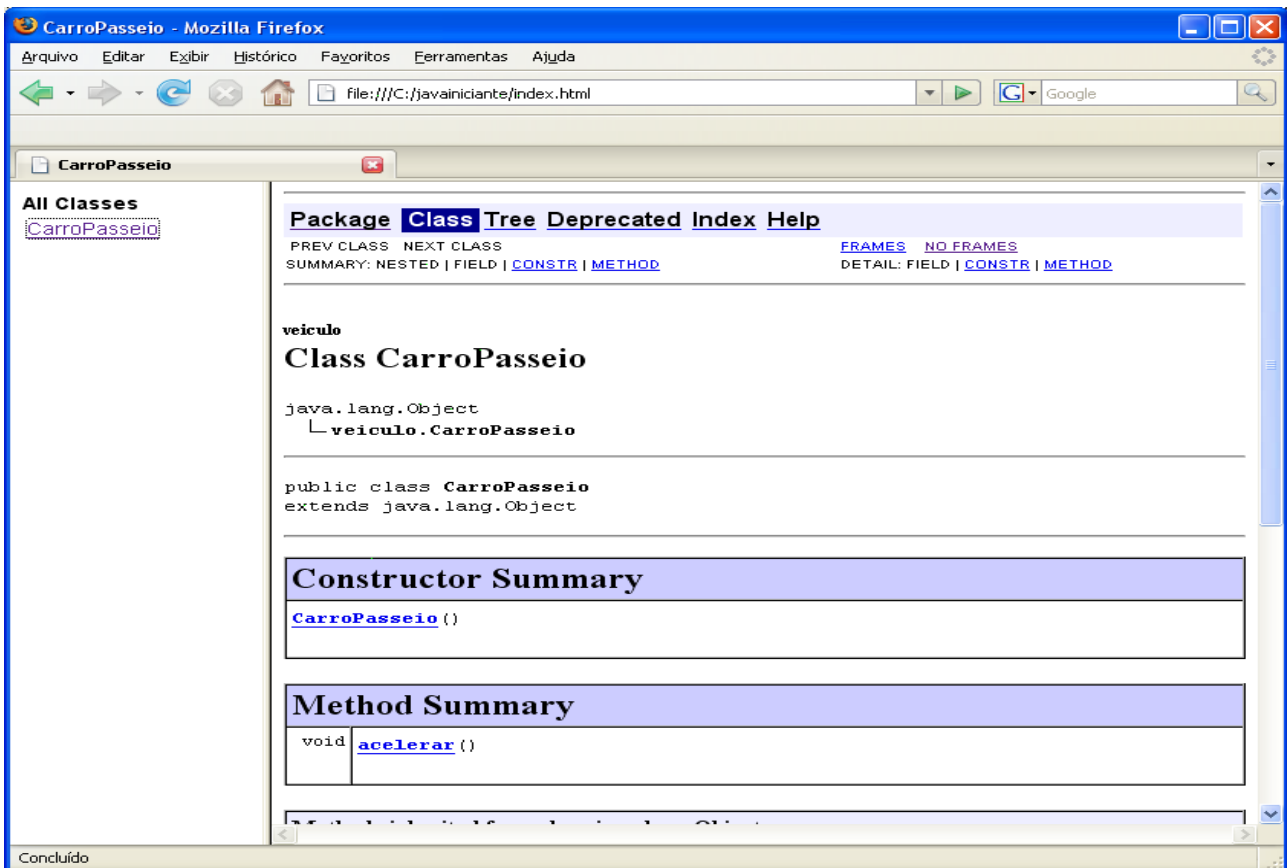
```
javadoc pacote1 pacote2 pacote3 ...
```

Serão gerados vários arquivos. Abra o arquivo "index.html" e temos o javadoc do pacote "veiculo":

Curso Java Starter



Acessando a classe "CarroPasseio" temos o seguinte:



Geramos o javadoc, mas vemos que só aparecem o nome do construtor e dos

Curso Java Starter

métodos sem nenhuma descrição. Como implementar isto?

Para inserirmos texto na documentação devemos adicionar ao código como se fossem comentários. Para abrir o texto utilize `/**` , para cada linha adicional utilize `*/` e para fechar utilize `*/`. Neste texto também pode-se definir o autor, parâmetros, versão, retorno, etc. Para isto, utilizamos o caractere `@` seguido do comando javadoc. Ex.:

- @author** – quem é o autor da classe, método, etc
- @param** – parâmetro que o método recebe
- @return** – o que o método retorna
- @version** – versão da classe
- @throws** – quais as exceções que podem ser geradas
- @since** – a partir de qual versão está disponível

Vamos documentar a nossa classe "CarroPasseio":

```
package veiculo;

/**
 * Classe que define o comportamento de um Carro de Passeio
 * @author Cláudio
 */
public class CarroPasseio {

    /**
     * Este método acelera o carro
     */
    public void acelerar(){
        //código...
    }

    /**
     * Frea o carro de acordo com a intensidade informada
     * @param intensidade - intensidade da freada.
     */
    public void frear(int intensidade){
        //código
    }

    /**
     * Retorna a velocidade atual do veículo
     * @return velocidade do veículo
     */
    public int getVelocidade(){
        //código...
        return 0;
    }
}
```

Após gerar o javadoc com o mesmo comando utilizado anteriormente, vemos o resultado nas duas figuras seguintes:

Curso Java Starter

veiculo - Mozilla Firefox

Arquivo Editar Exibir Histórico Favoritos Ferramentas Ajuda

file:///C:/javainiciante/index.html

veiculo

All Classes
[CarroPasseio](#)

Package [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)
PREV PACKAGE NEXT PACKAGE [FRAMES](#) [NO FRAMES](#)

Package veiculo

Class Summary

CarroPasseio	Classe que define o comportamento de um Carro de Passeio
------------------------------	--

Package [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)
PREV PACKAGE NEXT PACKAGE [FRAMES](#) [NO FRAMES](#)

Concluído

CarroPasseio - Mozilla Firefox

Arquivo Editar Exibir Histórico Favoritos Ferramentas Ajuda

file:///C:/javainiciante/index.html

CarroPasseio

All Classes
[CarroPasseio](#)

void	frear (int intensidade) Frea o carro de acordo com a intensidade informada
int	getVelocidade () Retorna a velocidade atual do veiculo

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

CarroPasseio

Concluído

Perceba, acima, que agora temos as descrições do métodos.

Arquivos JAR

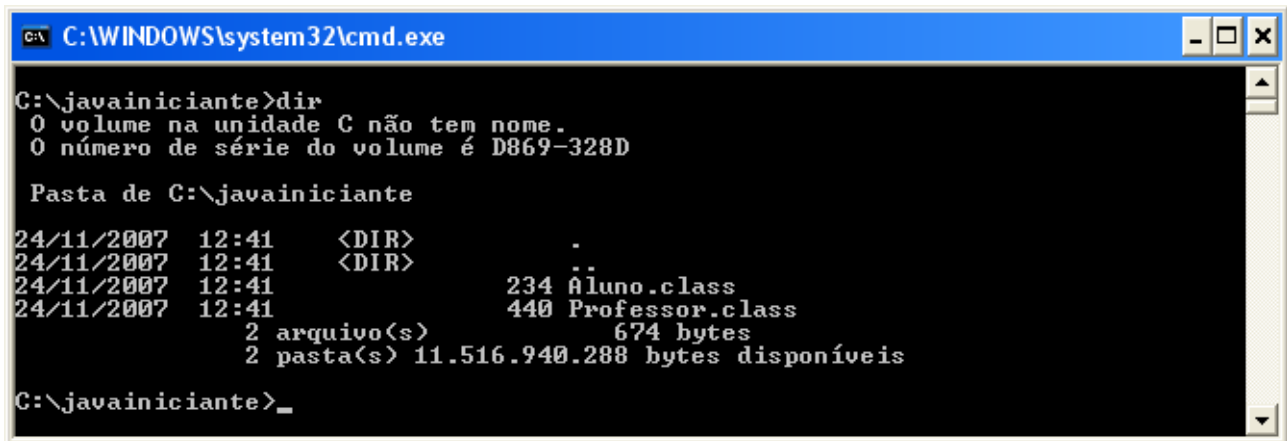
Depois que implementamos uma aplicação, vamos ter que distribuí-la. Mas já imaginou seu sistema com dezenas de pacotes e classes, e ter que mandar todos esses arquivos para o cliente? Fica meio complicado não é?

Normalmente as aplicações Java são distribuídas em arquivos "JAR", que é um arquivo compactado no formato "ZIP", mas com a extensão ".jar".

Vamos criar duas classes e criar um arquivo JAR contendo essas classes:

```
public class Professor {  
  
    public static void main(String args[]){  
        System.out.println("Primeiro arquivo JAR criado!");  
    }  
}  
  
public class Aluno {  
  
    public void aprender(){  
        //código...  
    }  
}
```

Depois de compilada as classes, temos elas como mostra a figura abaixo:



```
C:\WINDOWS\system32\cmd.exe  
C:\javainiciante>dir  
O volume na unidade C não tem nome.  
O número de série do volume é D869-328D  
  
Pasta de C:\javainiciante  
24/11/2007 12:41 <DIR> .  
24/11/2007 12:41 <DIR> ..  
24/11/2007 12:41 234 Aluno.class  
24/11/2007 12:41 440 Professor.class  
                2 arquivo(s) 674 bytes  
                2 pasta(s) 11.516.940.288 bytes disponíveis  
C:\javainiciante>_
```

Agora vamos criar o arquivo JAR com as duas classes. Usamos o seguinte comando:

```
jar cvf PrimeiroJar.jar Aluno.class Professor.class
```

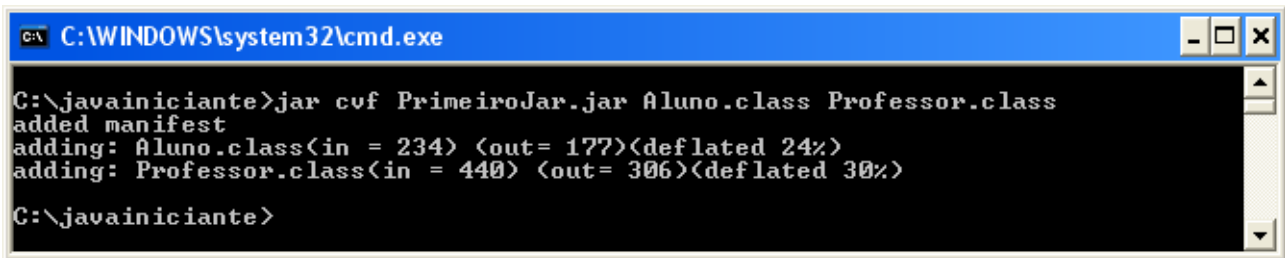
Curso Java Starter

onde:

cvf --> opções do comando jar (c = criar novo arquivo, v = mostrar o andamento do arquivamento, f = especificar o nome do arquivo gerado)

PrimeiroJar.jar --> nome do arquivo que será gerado

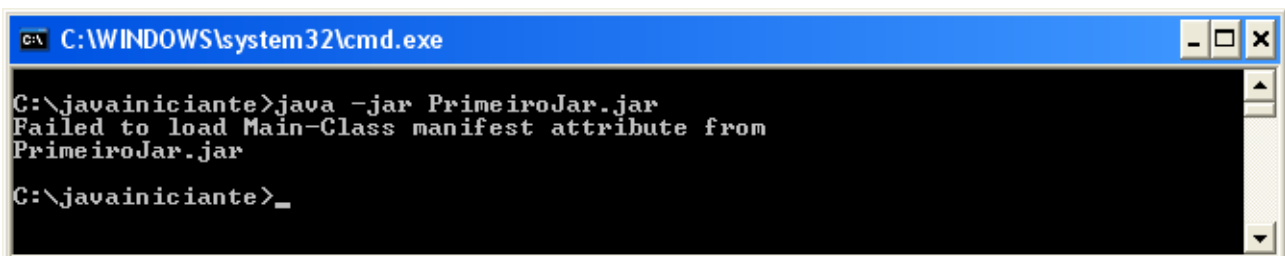
Aluno.class e **Professor.class** --> classes que serão arquivadas.



```
C:\WINDOWS\system32\cmd.exe
C:\javainiciante>jar cvf PrimeiroJar.jar Aluno.class Professor.class
added manifest
adding: Aluno.class(in = 234) (out= 177)(deflated 24%)
adding: Professor.class(in = 440) (out= 306)(deflated 30%)
C:\javainiciante>
```

Ok, agora as duas classes foram arquivadas no arquivo "PrimeiroJar.jar". Fica bem melhor distribuir a aplicação assim. Mas como vamos executar esta aplicação?

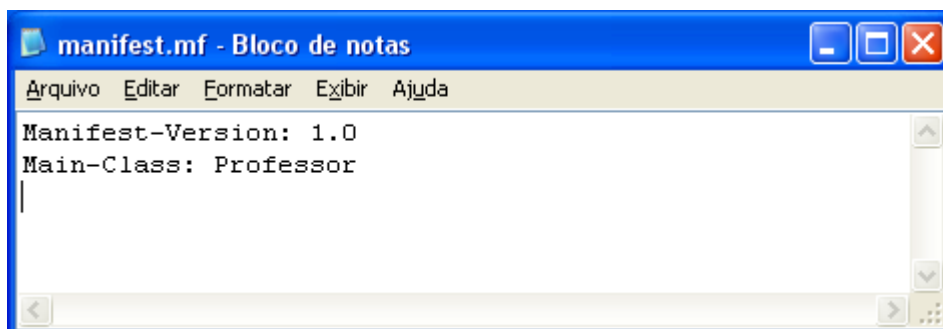
Para isso vamos usar a opção "-jar" do comando "java".



```
C:\WINDOWS\system32\cmd.exe
C:\javainiciante>java -jar PrimeiroJar.jar
Failed to load Main-Class manifest attribute from
PrimeiroJar.jar
C:\javainiciante>_
```

Opa, o que aconteceu? Nossa aplicação não funcionou. Por que?

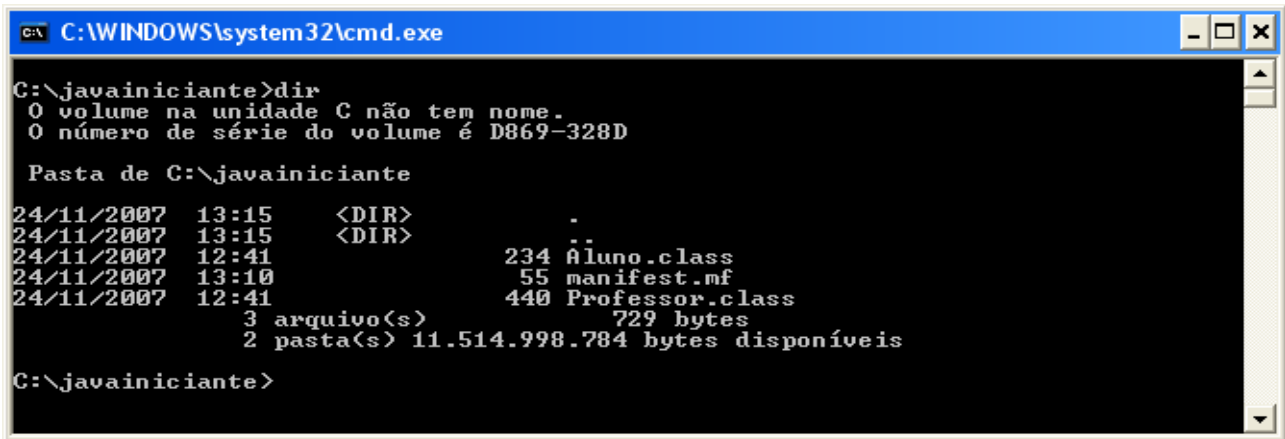
Quando a JVM executa um arquivo JAR, ela procura no arquivo chamado "manifest.mf" para saber qual a classe ela deve executar primeiro(a que contém o método "main"). Esse é um arquivo txt que além de informar qual a classe "main", tem a versão, bibliotecas, etc. Vamos criar este arquivo para a nossa aplicação:



```
manifest.mf - Bloco de notas
Arquivo Editar Formatar Exibir Ajuda
Manifest-Version: 1.0
Main-Class: Professor
|
```

Curso Java Starter

A primeira linha do arquivo contém a versão do arquivo manifest. Na segunda linha definimos a classe que contém o método "main". A última linha do arquivo deve estar em branco. Salvamos esse texto com o nome "manifest.mf". Agora temos os seguintes arquivos:

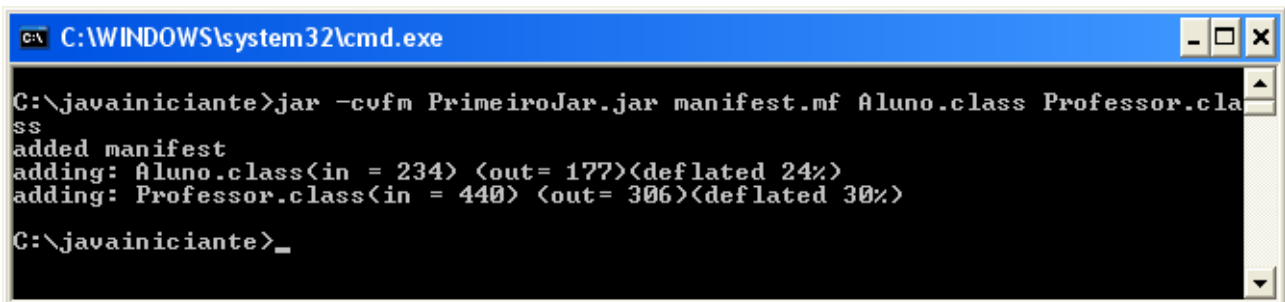


```
C:\WINDOWS\system32\cmd.exe
C:\javainiciante>dir
0 volume na unidade C não tem nome.
0 número de série do volume é D869-328D

Pasta de C:\javainiciante
24/11/2007  13:15    <DIR>          .
24/11/2007  13:15    <DIR>          ..
24/11/2007  12:41                234 Aluno.class
24/11/2007  13:10                55 manifest.mf
24/11/2007  12:41                440 Professor.class
              3 arquivo(s)              729 bytes
              2 pasta(s) 11.514.998.784 bytes disponíveis

C:\javainiciante>
```

Vamos agora incluir o "manifest" em nosso arquivo JAR:



```
C:\WINDOWS\system32\cmd.exe
C:\javainiciante>jar -cvfm PrimeiroJar.jar manifest.mf Aluno.class Professor.class
added manifest
adding: Aluno.class(in = 234) (out= 177)(deflated 24%)
adding: Professor.class(in = 440) (out= 306)(deflated 30%)

C:\javainiciante>_
```

Perceba que agora incluímos a opção "m" onde temos que indicar o arquivo "manifest" e o nome do arquivo "manifest.mf".

Vamos executar nossa aplicação para ver o que acontece:



```
C:\WINDOWS\system32\cmd.exe
C:\javainiciante>java -jar PrimeiroJar.jar
Primeiro arquivo JAR criado!

C:\javainiciante>
```

Beleza, agora sim, nossa aplicação funcionou e poderemos distribuí-la em um

único arquivo.

Pode ser que em um arquivo JAR não tenha nenhuma classe que contenha o método "main". Este arquivo pode ser um framework, um driver JDBC, uma biblioteca, etc.

Por exemplo, quando vamos utilizar o "hibernate" temos que adicionar ao CLASSPATH o arquivo JAR que contém as bibliotecas do mesmo.

Agora você deve tentar resolver a lista de exercícios abaixo. Qualquer dificuldade envie para a lista de discussão.

Exercícios

Aprenda com quem também está aprendendo, veja e compartilhe as suas respostas no nosso [Fórum](#):

[Exercícios – Módulo 06 – Documentação Java – Consulta e JavaDOC](#)

01 – Para ir se acostumando e aprender um pouco mais sobre as classes do Java, consulte no link <http://java.sun.com/javase/6/docs/api/> as especificações das classes "java.util.GregorianCalendar" e "java.lang.Math".

02 – Crie uma classe chamada "ImportarTexto" que esteja no pacote "texto.util" e que contenha os métodos "importarArquivo" e "filtrarDados". Gere a documentação dessa classe, incluindo os comandos javadoc "@param" e "@return".

03 – No prompt de comando digite "javadoc -help" e veja as opções deste utilitário.

04 – No prompt de comando digite "jar" e veja as opções deste utilitário.

05 – Gere um arquivo JAR que contenha a classe criada no exercício 2.

06 – Implemente na classe criada no exercício 2 o método "main" contendo a linha `javax.swing.JOptionPane.showMessageDialog(null, "Teste de arquivo JAR");`. Crie um arquivo "manifest.mf" definindo a classe "ImportarTexto" como a classe principal e gere o arquivo JAR.

Curso Java Starter

07 – Se estiver utilizando o “windows”, abra o “windows explorer” e navegue até a pasta onde está o arquivo JAR gerado no exercício anterior. Clique duas vezes neste arquivo e veja o que acontece.