Módulo 2

Introdução à Programação II

Lição 5

Aplicações Textuais

Versão 1.0 - Mar/2007

Autor
Rebecca Ong

Equipe
Joyce Avestro
Florence Balagtas
Rommel Feria
Rebecca Ong
John Paul Petines
Sun Microsystems
Sun Philippines

Necessidades para os Exercícios

Sistemas Operacionais Suportados

NetBeans IDE 5.5 para os seguintes sistemas operacionais:

- Microsoft Windows XP Profissional SP2 ou superior
- Mac OS X 10.4.5 ou superior
- Red Hat Fedora Core 3
- Solaris™ 10 Operating System (SPARC® e x86/x64 Platform Edition)

NetBeans Enterprise Pack, poderá ser executado nas seguintes plataformas:

- Microsoft Windows 2000 Profissional SP4
- Solaris™ 8 OS (SPARC e x86/x64 Platform Edition) e Solaris 9 OS (SPARC e x86/x64 Platform Edition)
- Várias outras distribuições Linux

Configuração Mínima de Hardware

Nota: IDE NetBeans com resolução de tela em 1024x768 pixel

Sistema Operacional	Processador	Memória	HD Livre
Microsoft Windows	500 MHz Intel Pentium III workstation ou equivalente	512 MB	850 MB
Linux	500 MHz Intel Pentium III workstation ou equivalente	512 MB	450 MB
Solaris OS (SPARC)	UltraSPARC II 450 MHz	512 MB	450 MB
Solaris OS (x86/x64 Platform Edition)	AMD Opteron 100 Série 1.8 GHz	512 MB	450 MB
Mac OS X	PowerPC G4	512 MB	450 MB

Configuração Recomendada de Hardware

Sistema Operacional	Processador	Memória	HD Livre
Microsoft Windows	1.4 GHz Intel Pentium III workstation ou equivalente	1 GB	1 GB
Linux	1.4 GHz Intel Pentium III workstation ou equivalente	1 GB	850 MB
Solaris OS (SPARC)	UltraSPARC IIIi 1 GHz	1 GB	850 MB
Solaris OS (x86/x64 Platform Edition)	AMD Opteron 100 Series 1.8 GHz	1 GB	850 MB
Mac OS X	PowerPC G5	1 GB	850 MB

Requerimentos de Software

NetBeans Enterprise Pack 5.5 executando sobre Java 2 Platform Standard Edition Development Kit 5.0 ou superior (JDK 5.0, versão 1.5.0_01 ou superior), contemplando a Java Runtime Environment, ferramentas de desenvolvimento para compilar, depurar, e executar aplicações escritas em linguagem Java. Sun Java System Application Server Platform Edition 9.

- Para **Solaris**, **Windows**, e **Linux**, os arquivos da JDK podem ser obtidos para sua plataforma em http://java.sun.com/j2se/1.5.0/download.html
- Para **Mac OS X**, Java 2 Plataform Standard Edition (J2SE) 5.0 Release 4, pode ser obtida diretamente da Apple's Developer Connection, no endereço: http://developer.apple.com/java (é necessário registrar o download da JDK).

Para mais informações: http://www.netbeans.org/community/releases/55/relnotes.html

Colaboradores que auxiliaram no processo de tradução e revisão

Alexandre Mori
Alexis da Rocha Silva
Aline Sabbatini da Silva Alves
Allan Wojcik da Silva
André Luiz Moreira
Andro Márcio Correa Louredo
Antoniele de Assis Lima
Antonio Jose R. Alves Ramos
Aurélio Soares Neto
Bruno da Silva Bonfim
Bruno dos Santos Miranda
Bruno Ferreira Rodrigues
Carlos Alberto Vitorino de Almeida
Carlos Alexandre de Sene
Carlos André Noronha de Sousa
Carlos Eduardo Veras Neves
Cleber Ferreira de Sousa
Cleyton Artur Soares Urani
Cristiano Borges Ferreira
Cristiano de Siqueira Pires
Derlon Vandri Aliendres
Fabiano Eduardo de Oliveira
Fábio Bombonato
Fernando Antonio Mota Trinta
Flávio Alves Gomes
Francisco das Chagas
Francisco Marcio da Silva
Gilson Moreno Costa
Givailson de Souza Neves
Gustavo Henrique Castellano
Hebert Julio Gonçalves de Paula
Heraldo Conceição Domingues
Hugo Leonardo Malheiros Ferreira
Ivan Nascimento Fonseca
Jacqueline Susann Barbosa
Jader de Carvalho Belarmino
João Aurélio Telles da Rocha
João Paulo Cirino Silva de Novais
João Vianney Barrozo Costa
José Augusto Martins Nieviadonski
José Leonardo Borges de Melo
José Ricardo Carneiro
Kleberth Bezerra G. dos Santos
Lafaiete de Sá Guimarães
Leandro Silva de Morais
Leonardo Leopoldo do Nascimento
Leonardo Pereira dos Santos
Leonardo Rangel de Melo Filardi
Lucas Mauricio Castro e Martins
Luciana Rocha de Oliveira
Luís Carlos André
Luís Octávio Jorge V. Lima
Luiz Fernandes de Oliveira Junior
Luiz Victor de Andrade Lima
Manoel Cotts de Queiroz
Marcello Sandi Pinheiro
Marcelo Ortolan Pazzetto
Marco Aurélio Martins Bessa
Marcos Vinicius de Toledo
Maria Carolina Ferreira da Silva
Massimiliano Giroldi
Mauricio Azevedo Gamarra
Mauricio da Silva Marinho
Mauro Cardoso Mortoni
Mauro Regis de Sousa Lima
Namor de Sá e Silva
Néres Chaves Rebouças
Nolyanne Peixoto Brasil Vieira
Paulo Afonso Corrêa
Paulo José Lemos Costa
Paulo Oliveira Sampaio Reis
Pedro Antonio Pereira Miranda
Pedro Henrique Pereira de Andrade
Renato Alves Félix
Renato Barbosa da Silva
Reyderson Magela dos Reis
Ricardo Ferreira Rodrigues
Ricardo Ulrich Bomfim
Robson de Oliveira Cunha
Rodrigo Pereira Machado
Rodrigo Rosa Miranda Corrêa
Rodrigo Vaez
Ronie Dotzlaw
Rosely Moreira de Jesus
Seire Pareja
Sergio Pomerancblum
Silvio Sznifer
Suzana da Costa Oliveira
Tásio Vasconcelos da Silveira
Thiago Magela Rodrigues Dias
Tiago Gimenez Ribeiro
Vanderlei Carvalho Rodrigues Pinto
Vanessa dos Santos Almeida
Vastí Mendes da Silva Rocha
Wagner Eliezer Roncoletta

Auxiliadores especiais

Revisão Geral do texto para os seguintes Países:

- **Brasil** – Tiago Flach
- **Guiné Bissau** – Alfredo Cá, Bunene Sisse e Buon Olossato Quebi – ONG Asas de Socorro

Coordenação do DFJUG

- **Daniel deOliveira** – JUGLeader responsável pelos acordos de parcerias
- **Luci Campos -** Idealizadora do DFJUG responsável pelo apoio social
- **Fernando Anselmo -** Coordenador responsável pelo processo de tradução e revisão, disponibilização dos materiais e inserção de novos módulos
- **Regina Mariani -** Coordenadora responsável pela parte jurídica
- **Rodrigo Nunes -** Coordenador responsável pela parte multimídia
- **Sérgio Gomes Veloso -** Coordenador responsável pelo ambiente JEDI™ (Moodle)

Agradecimento Especial

John Paul Petines – Criador da Iniciativa JEDI™
Rommel Feria – Criador da Iniciativa JEDI™

1. Objetivos

Nesta lição, será mostrada uma revisão sobre como utilizar os argumentos de linha de comando. Além disso, serão mostradas mais informações sobre como utilizar fluxos de dados para obter entrada do usuário durante a execução da sua aplicação para manipular arquivos.

Ao final desta lição, o estudante será capaz de:

- Obter entrada através da linha de comando
- Explicar como manipular propriedades do sistema
- Ler através da entrada padrão
- Ler e escrever em arquivos

2. Argumentos de Linha de Comando e Propriedades do Sistema

Como foi visto no módulo anterior, Java permite ao usuário enviar dados na linha de comando para a classe que está sendo executada. Por exemplo, passar os argumentos 1 e 2 para uma classe nomeada como *Calculate*, pode-se digitar a seguinte linha de comando:

```
java Calculate 1 2
```

Neste exemplo, o valor 1 é armazenado na posição args[0] do argumento *args*, enquanto que o valor 2 é armazenado na posição args[1] do argumento *args*. Deste modo, a finalidade de declarar "String args[]" como um argumento no método *main* fica clara.

Estes são os passos para passar argumentos na linha de comando no NetBeans:

1. Procurar o nome do projeto no painel dos projetos à esquerda
2. Clicar com o botão direito do mouse sobre o nome ou o ícone do projeto
3. Selecionar a opção *Properties*
4. Clicar em *Run* no painel de Categories à esquerda
5. Entrar os argumentos separados por espaço no campo texto do item *Arguments:*
6. Clicar no botão *OK*
7. Compilar e executar a classe.

Para compreender mais facilmente estas etapas, observe as imagens a seguir.

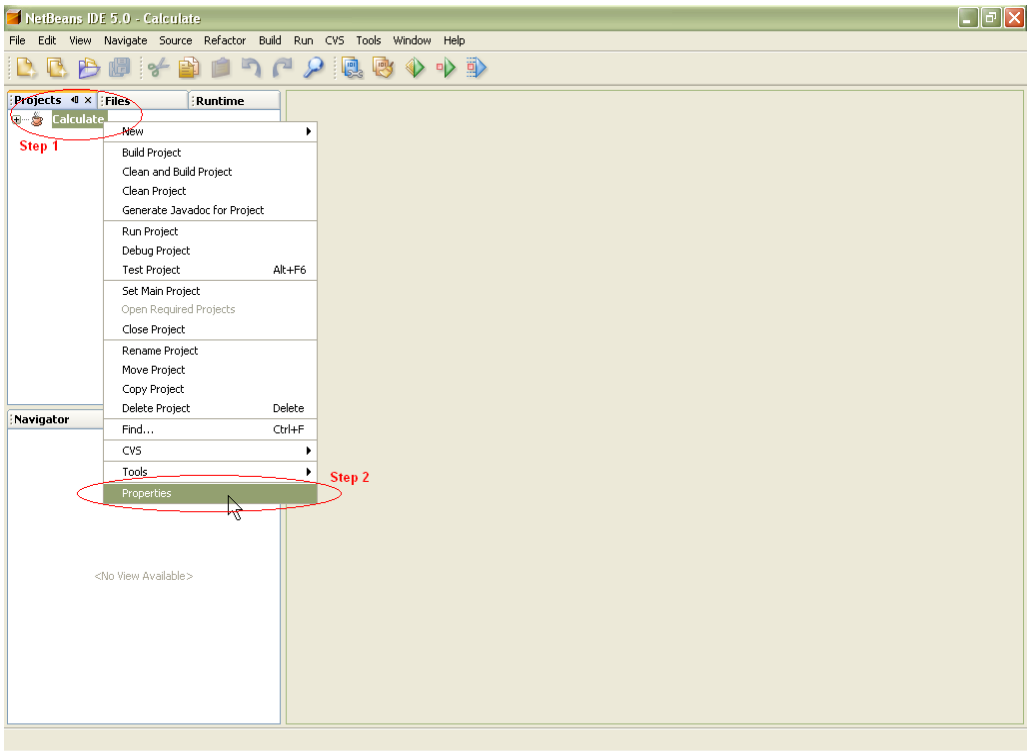

Figura 1: Passando Argumentos na Linha de Comando com o NetBeans – Passos 1 e 2

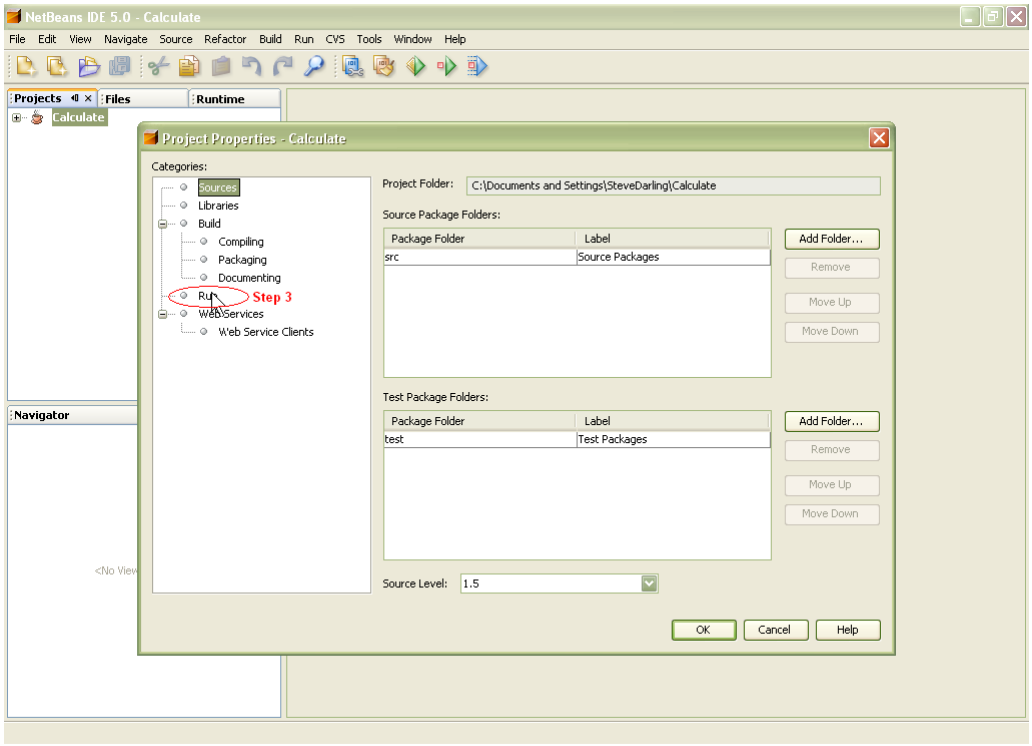

Figura 2: Passando Argumentos na Linha de Comando com o NetBeans - Passo 3

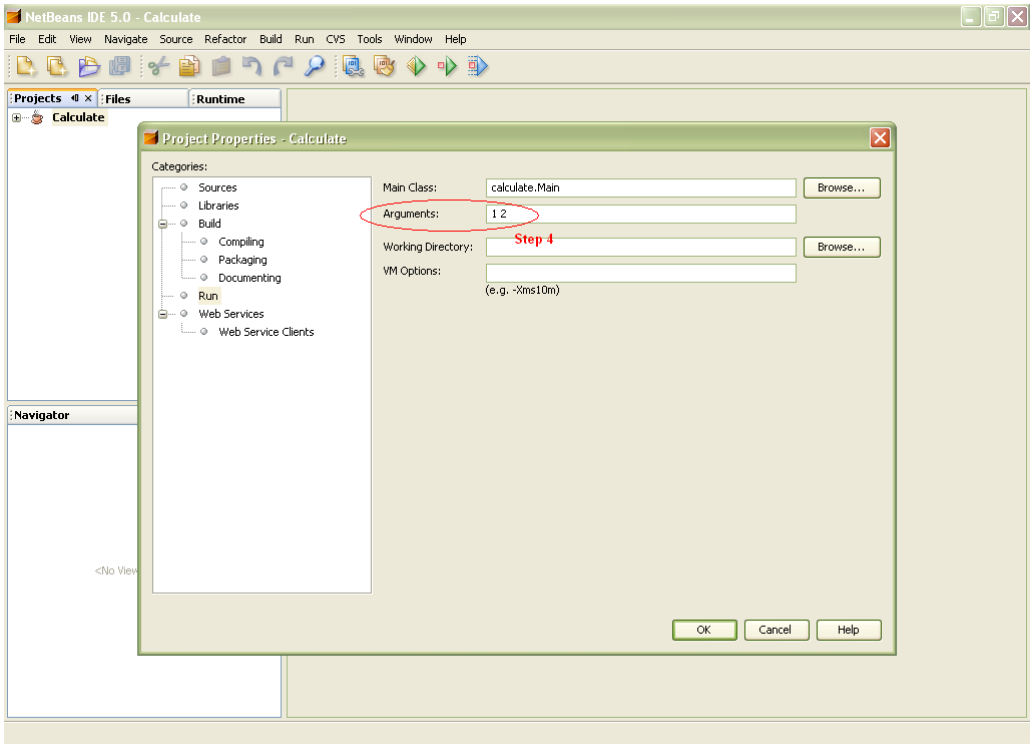

Figura 3: Passando Argumentos na Linha de Comando com o NetBeans – Passo 4

Além de passar argumentos ao método *main*, também é possível manipular propriedades do sistema a partir da linha de comando.

Propriedades de sistema são bastante semelhantes às variáveis de ambiente. Porém, não são

dependentes de plataforma. Uma propriedade é simplesmente um mapeamento entre o nome da propriedade e o seu valor correspondente. Isto é representado em Java com a classe *Properties*. A classe *System* provê alguns métodos para determinar as propriedades correntes do sistema, o método *getProperties* retorna um objeto do tipo *Properties*. O método *getProperty* possui duas formas:

`public static String getProperty(String key)`
Esta versão retorna o valor de uma propriedade do sistema, indicada pela chave especificada. Retorna nulo se não houver nenhuma propriedade com a chave especificada.
`public static String getProperty(String key, String def)`
Esta versão retorna também o valor de uma propriedade do sistema, indicada pela chave especificada. Retorna def, valor padrão, se não houver nenhuma propriedade com a chave especificada.

Tabela 1: método getProperty() da classe System

Não enfatizaremos os detalhes das propriedades do sistema. Iremos manipular diretamente as propriedades do sistema.

É possível utilizar a opção -D, como argumento da linha de comando Java, para incluir uma nova propriedade.

```
java -D<name>=value
```

Por exemplo, na propriedade de sistema *user.home* iremos inserir o valor *philippines*, vamos digitar o seguinte comando:

```
java -Duser.home=philippines
```

Para exibir a lista de propriedades do sistema disponíveis e seus valores correspondentes, deve-se utilizar o método *getProperties*, como segue:

```
System.getProperties().list(System.out);
```

Aqui temos uma amostra da lista de propriedades do sistema:

```
-- listing properties --
java.runtime.name=Java(TM) 2 Runtime Environment, Stand...
sun.boot.library.path=C:\Program Files\Java\jdk1.5.0_06\jre...
java.vm.version=1.5.0_06-b05
java.vm.vendor=Sun Microsystems Inc.
java.vendor.url=http://java.sun.com/
path.separator=;
java.vm.name=Java HotSpot(TM) Client VM
file.encoding.pkg=sun.io
user.country=US
sun.os.patch.level=Service Pack 2
java.vm.specification.name=Java Virtual Machine Specification
user.dir=C:\Documents and Settings\becca\Neste...
java.runtime.version=1.5.0_06-b05
java.awt.graphicsenv=sun.awt.Win32GraphicsEnvironment
java.endorsed.dirs=C:\Program Files\Java\jdk1.5.0_06\jre...
os.arch=x86
java.io.tmpdir=C:\DOCUME~1\becca\LOCALS~1\Temp\
line.separator=
java.vm.specification.vendor=Sun Microsystems Inc.
user.variant=
```

```
os.name=Windows XP
sun.jnu.encoding=Cp1252
java.library.path=C:\Program Files\Java\jdk1.5.0_06\jre...
java.specification.name=Java Platform API Specification
java.class.version=49.0
sun.management.compiler=HotSpot Client Compiler
os.version=5.1
user.home=C:\Documents and Settings\becca
user.timezone=
java.awt.printerjob=sun.awt.windows.WPrinterJob
file.encoding=Cp1252
java.specification.version=1.5
user.name=becca
java.class.path=C:\Documents and Settings\becca\Neste...
java.vm.specification.version=1.0
sun.arch.data.model=32
java.home=C:\Program Files\Java\jdk1.5.0_06\jre
java.specification.vendor=Sun Microsystems Inc.
user.language=en
awt.toolkit=sun.awt.windows.WToolkit
java.vm.info=mixed mode, sharing
java.version=1.5.0_06
java.ext.dirs=C:\Program Files\Java\jdk1.5.0_06\jre...
sun.boot.class.path=C:\Program Files\Java\jdk1.5.0_06\jre...
java.vendor=Sun Microsystems Inc.
file.separator=\
java.vendor.url.bug=http://java.sun.com/cgi-bin/bugreport...
sun.cpu.endian=little
sun.io.unicode.encoding=UnicodeLittle
sun.desktop=windows
sun.cpu.isalist=
```

3. Lendo da Entrada Padrão

Ao invés de obter uma entrada do usuário através da linha de comando, a maioria dos usuários prefere entrar com os dados quando requisitados pela classe enquanto ela estiver em execução. Um modo de realizar isto é com o uso dos fluxos. Um fluxo é uma abstração de um arquivo ou de um dispositivo que permite ler ou escrever uma série de itens. Eles são conectados com dispositivos físicos, tais como: teclados, consoles e arquivos. Há dois tipos gerais de fluxo de dados: fluxo de *bytes* e fluxo de caracteres. Os fluxos de bytes são para os dados binários, enquanto os fluxos de caracteres são para os caracteres *Unicode*. *System.in* e *System.out* são dois exemplos de objetos de fluxos de bytes pré-definidos em Java. O primeiro, por padrão, refere-se ao teclado e o último, ao monitor.

Para ler caracteres do teclado, utilize o *System.in*, que é um fluxo de *bytes* traduzido em um objeto do tipo *BufferedReader*. A linha seguinte mostra como realizar isto:

```
BufferedReader br = new BufferedReader(
   new InputStreamReader(System.in));
```

O método *read* do objeto do tipo *BufferedReader* é utilizado para ler do dispositivo de entrada.

```
ch = (int)br.read();    // método read retorna um inteiro
```

Vejamos esta classe como exemplo:

```
import java.io.*;

class FavoriteCharacter {
   public static void main(String args[]) throws IOException {
      System.out.println("Hi, what's your favorite character?");
      char favChar;
      BufferedReader br = new BufferedReader(new
         InputStreamReader(System.in));
      favChar = (char) br.read();
      System.out.println(favChar + " is a good choice!");
   }
}
```

Executando esta classe e informando o caractere "a" como entrada, obteremos a seguinte saída:

```
Hi, what's your favorite character?
a
a is a good choice!
```

Para ler uma linha inteira, em vez de um caractere de cada vez, é possível utilizar o método *readLine*.

```
str = br.readLine();
```

Vejamos uma classe similar ao exemplo anterior. Esta classe lê uma linha inteira em vez de um único caractere.

```
import java.io.*;

class GreetUser {
   public static void main(String args[]) throws IOException {
      System.out.println("Hi, what's your name?");
      String name;
      BufferedReader br = new BufferedReader(new
                                 InputStreamReader(System.in));
```

```
            name = br.readLine();
            System.out.println("Nice to meet you, " + name + "! :)");
        }
    }
```

A saída esperada da classe *GreetUser*, quando o usuário entrar com a palavra *Rebecca* é:

```
Hi, what's your name?
Rebecca
Nice to meet you, Rebecca! :)
```

Ao utilizar fluxos, não esqueça de importar o pacote *java.io* como mostrado:

```
import java.io.*;
```

Ao ler fluxos é possível que ocorram exceções. Não se deve esquecer de tratar estas exceções usando o bloco *try-catch* ou de indicá-las na cláusula *throws* do método.

4. Manipulação de Arquivo

Em alguns casos, as entradas de dados são armazenadas nos arquivos. Além disso, há também situações na qual queremos armazenar a saída de uma determinada classe para um arquivo.

Em um sistema informatizado de matrícula, os dados do estudante que podem ser utilizados como uma entrada para o sistema são, provavelmente, armazenados em um arquivo. Então, uma possível saída do sistema é a informação sobre os assuntos registrados pelos estudantes. Outra vez, a saída, neste caso, pode ser armazenada em um arquivo. Como visto nesta aplicação, há necessidade de ler e escrever em um arquivo. Aprenderemos sobre entrada e saída de arquivo ainda nesta seção.

4.1. Lendo um Arquivo

Para ler um arquivo, pode-se utilizar a classe *FileInputStream*. Este é um dos construtores desta classe:

```
FileInputStream(String filename)
```

O construtor cria uma conexão para um arquivo real, cujo o nome será especificado como um argumento. Uma exceção do tipo *FileNotFoundException* é lançada quando o arquivo não existe ou não pode ser aberto para leitura.

Após ter criado um fluxo de entrada, é possível utilizá-lo para ler um arquivo associado através do método *read* que retorna um valor do tipo inteiro e o valor -1 quando o fim do arquivo for encontrado.

Aqui está um exemplo:

```
import java.io.*;

class ReadFile {
   public static void main(String args[]) throws IOException {
      System.out.println(
         "What is the name of the file to read from?");
      String filename;
      BufferedReader br = new BufferedReader(
         new InputStreamReader(System.in));
      filename = br.readLine();
      System.out.println(
         "Now reading from " + filename + "...");
      FileInputStream fis = null;
      try {
         fis = new FileInputStream(filename);
      } catch (FileNotFoundException ex) {
         System.out.println("File not found.");
      }
      char data;
      int temp;
      do {
         temp = fis.read();
         data = (char) temp;
         if (temp != -1)
            System.out.print(data);
      } while (temp != -1);
      System.out.println("Problem in reading from the file.");
   }
}
```

Assumindo que *temp.txt* existe e seja o arquivo de texto, está é uma possível saída desta classe:

```
What is the name of the file to read from?
temp.txt
Now reading from temp.txt...
temporary file
```

4.2. Escrevendo em um Arquivo

Para escrever em um arquivo, é possível utilizar a classe *FileOutputStream*. Este é um dos construtores desta classe:

```
FileOutputStream(String filename)
```

O construtor vincula um fluxo de saída a um arquivo real para escrita. Uma exceção do tipo *FileNotFoundException* é lançada quando o arquivo não puder ser aberto para escrita.

Uma vez que o fluxo de saída é criado, pode-se utilizá-lo para escrever no arquivo vinculado através do método *write*, que possui a seguinte assinatura:

```
void write(int b)
```

O argumento **b** refere-se ao dado a ser escrito para o arquivo real, associado ao fluxo de saída.

A classe seguinte demonstra a escrita em um arquivo:

```
import java.io.*;

class WriteFile {
   public static void main(String args[]) throws IOException {
      System.out.println(
         "What is the name of the file to be written to?");
      String filename;
      BufferedReader br = new BufferedReader(
         new InputStreamReader(System.in));
      filename = br.readLine();
      System.out.println(
         "Enter data to write to " + filename + "...");
      System.out.println("Type q$ to end.");
      FileOutputStream fos = null;
      try {
         fos = new FileOutputStream(filename);
      } catch (FileNotFoundException ex) {
         System.out.println("File cannot be opened for writing.");
      }
      try {
         boolean done = false;
         int data;
         do {
            data = br.read();
            if ((char)data == 'q') {
               data = br.read();
               if ((char)data == '$') {
                  done = true;
               } else {
                  fos.write('q');
                  fos.write(data);
               }
            } else {
```

```
                    fos.write(data);
                  }
               } while (!done);
            } catch (IOException ex) {
               System.out.println("Problem in reading from the file.");
            }
         }
      }
```

Exemplo de execução da classe *WriteFile*:

```
What is the name of the file to be written to?
temp.txt
Enter data to write to temp.txt...
Type q$ to end.
what a wonderful world
1, 2, step
q$
```

O arquivo **temp.txt** deverá conter os seguintes dados:

```
what a wonderful world
1, 2, step
```

Parceiros que tornaram JEDI™ possível

Instituto CTS
Patrocinador do DFJUG.

Sun Microsystems
Fornecimento de servidor de dados para o armazenamento dos vídeo-aulas.

Java Research and Development Center da Universidade das Filipinas
Criador da Iniciativa JEDI™.

DFJUG
Detentor dos direitos do JEDI™ nos países de língua portuguesa.

Banco do Brasil
Disponibilização de seus *telecentros* para abrigar e difundir a Iniciativa JEDI™.

Politec
Suporte e apoio financeiro e logístico a todo o processo.

Borland
Apoio internacional para que possamos alcançar os outros países de língua portuguesa.

Instituto Gaudium/CNBB
Fornecimento da sua infra-estrutura de hardware de seus servidores para que os milhares de alunos possam acessar o material do curso simultaneamente.