Módulo 2

Introdução à Programação II

Lição 10

Redes

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

A linguagem Java permite que facilmente sejam criadas aplicações que desempenham diversas tarefas através de uma rede. Esse é um dos principais benefícios de Java, uma vez que a linguagem foi criada tendo em vista seu uso para a Internet. Antes de aprender sobre o uso e a funcionalidade Java em redes, devemos ter conhecimento de alguns conceitos básicos de redes.

Ao final desta lição, o estudante será capaz de:

- Explicar os conceitos básicos de redes
  - Endereço IP
  - Protocolo
  - Portas
  - Paradigma Cliente/Servidor
  - Sockets
- Criar aplicações usando o pacote de Java para redes
  - *As classes ServerSocket* e *Socket*
  - *As classes MulticastSocket* e *DatagramPacket*

2. Conceitos Básicos sobre Redes

A Internet é uma rede mundial que reúne diferentes tipos de computadores que são interligados de diversas maneiras. Apesar da diversidade de hardware e software interconectados, é extraordinário como a Internet permanece interconectada (funcional). Isso é possível devido aos padrões de comunicação definidos e respeitados. Esses padrões asseguram a compatibilidade e a confiabilidade de comunicação entre uma ampla diversidade de sistemas na Internet. Vejamos alguns desses padrões.

2.1. Endereço IP

Cada um dos computadores conectados à Internet tem um único endereço IP. Do ponto de vista lógico, o endereço IP é similar ao tradicional endereço para correspondência enviada pelos correios, conhecido como CEP, no sentido de que um endereço identifica unicamente um determinado objeto. O endereçamento IP é um número de 32 bits usado para identificar exclusivamente cada computador conectado à Internet. O número *192.1.1.1* é um exemplo de um endereço IP. Esses endereços podem também ser expressos em formato símbólico, como por exemplo: *docs.rinet.ru*.

2.2. Protocolo

Uma vez que existem muitos tipos de comunicações que ocorrem ativamente na Internet, é necessário existir um número igual de mecanismos para dar conta deles. Cada tipo de comunicação exige um protocolo específico e único.

Um protocolo refere-se a um conjunto de regras e padrões que definem um certo tipo de comunicação via Internet. O protocolo descreve o formato dos dados enviados através da Internet, juntamente com as informações de como e quando está sendo enviado.

Na realidade, o conceito de protocolo não é novo. Considere quantas vezes estivemos envolvidos neste tipo de conversação:

"Alô"
"Alô. Eu poderia falar com Joana?"
"Aguarde um momento, por favor"
"Obrigado"

Esse é um protocolo social usado quando se trata de uma conversa telefônica. Esse tipo de protocolo nos proporciona confiança e familiaridade sobre como nos comportarmos em determinadas situações.

Examinemos agora alguns protocolos importantes usados na Internet. Sem dúvida, o Hypertext Transfer Protocol (HTTP, ou Protocolo de Transferência de Hipertexto) é um dos protocolos mais comumente usados. Ele é usado para transferir documentos HTML na Web. Existe também o File Transfer Protocol (FTP, ou Protocolo de Transferência de Arquivos), que geralmente é conhecido, em comparação com o HTTP; e permite que seja transferido arquivos binários pela Internet. Os dois protocolos têm seu próprio conjunto de regras e padrões de como os dados sobre transferência de dados. Java dá suporte a ambos os protocolos.

2.3. Portas

Note que os protocolos somente fazem sentido quando usados no contexto de um serviço. Por exemplo, o protocolo HTTP é usado quando se está disponibilizando conteúdo de Web através de um serviço HTTP. Cada computador na Internet pode disponibilizar uma diversidade de serviços

por meio dos diversos protocolos suportados. O problema, porém, é que o tipo de serviço que precisa ser conhecido, antes que as informações possam ser transferidas. É nesse contexto que as portas são relevantes.

Uma porta é um número de 16 bits que identifica cada serviço oferecido por um servidor de rede. Para usar determinado serviço, e portanto estabelecer uma linha de comunicação e conexão por meio de um protocolo específico, é necessário conectar-se à porta apropriada. As portas são associadas a um número, e alguns desses números são especificamente associados a um determinado tipo de serviço. Essas portas às quais são alocados serviços específicos são denominadas portas padrão. Por exemplo, o serviço FTP está localizado na porta 21, o passo que, enquanto o serviço HTTP está localizado na porta 80. Ao necessitar executar uma transferência de arquivo via FTP, é imprescindível conectar-se à porta 21 do servidor. Todos os serviços alocados de forma padrão são atribuídos a valores de portas abaixo de 1024. Os valores de portas acima de 1024 são disponíveis para comunicações personalizadas. Caso um valor de porta acima de 1024 já esteja em uso por alguma comunicação personalizada, deve-se procurar outros valores ociosos.

2.4. O Paradigma Cliente/Servidor

O Paradigma Cliente/Servidor é à base das aplicações de redes em Java. Evidentemente, esse esquema compreende dois elementos principais, o cliente e o servidor. O termo cliente refere-se à máquina que necessita algum tipo de informação, ao passo que o servidor é a máquina onde essa informação está armazenada, aguardando para ser fornecida.

O paradigma descreve um cenário simples. Normalmente, um cliente conecta-se a um servidor para consultar determinada informação. O servidor então analisa a consulta e retorna a informação nele disponível para o cliente.

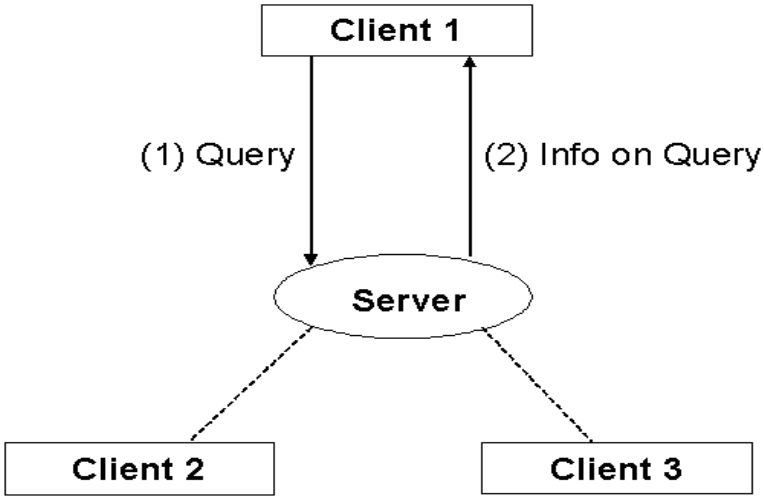

Figure 1: Client/Server model

2.5. Sockets

O último conceito geral de redes que iremos abordar antes de mergulhar em redes em Java relaciona-se com a classe *sockets*. A maior parte da programação em Java para redes é utilizado um determinado tipo de comunicação em rede denominada *sockets*.

S*ocket* é uma abstração, na forma de software, para um meio de entrada ou saída de comunicação. É por meio do uso de *sockets* que a linguagem Java executa toda a sua comunicação de baixo nível em redes. Os sockets são canais de comunicação que permitem a transferência de dados através de uma determinada porta; em resumo, um socket refere-se a uma porta-fim *(endpoint)* para comunicação entre duas máquinas.

3. O Pacote Java para Redes

O *java.net* package oferece classes especificas para o desenvolvimento de aplicações para redes. Para obter uma lista completa de classes e interfaces relacionadas com redes, por favor, consulte a documentação da API de Java. Vamos nos concentrar nestas quatro classes: *ServerSocket*, *Socket*, *MulticastSocket*, e *DatagramPacket*.

3.1. As Classes ServerSocket e Socket

A Classe *ServerSocket* disponibiliza as funcionalidades básicas de um servidor. A tabela a seguir descreve dois dos quatro construtores da classe *ServerSocket*:

Construtores da classe *ServerSocket*
`ServerSocket(int port)`
Instancia um servidor que é vinculado à porta especificada. Uma porta 0 aloca o servidor a qualquer porta ociosa. Por default, o comprimento máximo de fila para conexão (chegando) é fixado (como igual a) em 50.
`ServerSocket(int port, int backlog)`
Parâmetro *backlog (fila acumulada)*.

Table 1: ServerSocket constructors

Abaixo segue alguns dos métodos desta classe:

Métodos da classe *ServerSocket*
`public Socket accept()`
Faz com que o servidor aguarde e escute conexões de clientes, e então as aceite.
`public void close()`
Fecha o socket do servidor. Uma vez fechado o socket, os clientes não podem mais conectar-se ao servidor, a menos que o socket seja aberto novamente.
`public int getLocalPort()`
Retorna a porta à qual o socket está vinculado.
`public boolean isClosed()`
Indica se o socket está fechado ou não.

Table 2: ServerSocket methods

O exemplo a seguir é uma implementação de um servidor simples, que simplesmente ecoa a informação enviada pelo cliente.

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

public class EchoingServer {
   public static void main(String[] args) {
      ServerSocket server = null;
      Socket client;
      try {
         server = new ServerSocket(1234);
         // 1234 é um número de porta não utilizado
      } catch (IOException ie) {
         System.out.println("O socket não pode ser aberto.");
         System.exit(1);
```

```
            }
            while(true) {
               try {
                  client = server.accept();
                  OutputStream clientOut = client.getOutputStream();
                  PrintWriter pw = new PrintWriter(clientOut, true);
                  InputStream clientIn = client.getInputStream();
                  BufferedReader br = new BufferedReader(new
                     InputStreamReader(clientIn));
                  pw.println(br.readLine());
               } catch (IOException ie) {
               }
            }
         }
      }
```

A classe *ServerSocket* implementa *socket* de servidor e a classe *Socket* implementa um *socket* de cliente. A classe *Socket* possui oito construtores, dos quais dois já foram descartados. Examinemos brevemente dois desses construtores.

Construtores da classe *Socket*
`Socket(String host, int port)`
Cria um socket de cliente que se conecta ao número fornecido de porta no host especificado.
`Socket(InetAddress address, int port)`
Cria um socket de cliente que se conecta ao número fornecido de porta, no endereço IP especificado.

Table 3: Socket constructors

Segue alguns métodos da classe:

Métodos da classe *Socket*
`public void close()`
Fecha o socket cliente.
`public InputStream getInputStream()`
Recupera o fluxo de entrada associado a esse socket.
`public OutputStream getOutputStream()`
Recupera o fluxo de saída associado a esse socket.
`public InetAddress getInetAddress()`
Retorna o endereço de IP ao qual o socket está conectado
`public int getPort()`
(Returna) Retorna a porta remota à qual este socket está conectado.
`public boolean isClosed()`
Indica se o socket está fechado ou não.

Table 4: Socket methods

O exemplo a seguir é uma implementação de um cliente simples, que apenas envia dados para um servidor.

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

public class MyClient {
   public static void main(String args[]) {
      try {
         Socket client = new Socket(InetAddress.getLocalHost(), 1234);
         InputStream clientIn = client.getInputStream();
         OutputStream clientOut = client.getOutputStream();
         PrintWriter pw = new PrintWriter(clientOut, true);
         BufferedReader br = new BufferedReader(new
            InputStreamReader(clientIn));
         BufferedReader stdIn = new BufferedReader(new
         InputStreamReader(System.in));
         System.out.println("Digite uma mensagem para o servidor: ");
         pw.println(stdIn.readLine());
         System.out.println("Mensagem do Servidor: ");
         System.out.println(br.readLine());
         pw.close();
         br.close();
         client.close();
      } catch (ConnectException ce) {
         System.out.println("Não foi possível se conectar ao servidor.");
      } catch (IOException ie) {
         System.out.println("Erro de I/O.");
      }
   }
}
```

A execução de *EchoingServer* deixa-o pronto para aceitar mensagens do cliente. Depois que um cliente, como *MyClient*, envia uma mensagem ao servidor, o servidor devolve a mensagem de volta para o cliente. Abaixo, um exemplo da execução de *MyClient* depois da inicialização de *EchoingServer*:

```
Digite uma mensagem para o servidor:
Primeira mensagem para o servidor
Mensagem do Servidor:
Primeira mensagem para o servidor
```

3.2. As Classes MulticastSocket e DatagramPacket

A classe *MulticastSocket* é útil para aplicações que implementam comunicações em grupos. Os endereços IP para um grupo multicast ficam na faixa de 224.0.0.0 a 239.255.255.255. Entretanto, o endereço 224.0.0.0 é reservado, e não deve ser usado. Essa classe tem três construtores, mas consideraremos, aqui, apenas um desses construtores.

Construtores da classe *MulticastSocket*
`MulticastSocket(int port)`
Cria um multicast socket vinculado ao número de porta fornecido.

Tabela 1.5: Construtor de MulticastSocket

A tabela abaixo dá uma descrição de alguns métodos *de MulticastSocket*.

Métodos da classe *MulticastSocket*
`public void joinGroup(InetAddress mcastaddr)`
Entrar em um grupo *multicast* no endereço especificado.

`public void leaveGroup(InetAddress mcastaddr)`
Sair de um a grupo *multicast* no endereço especificado.
`public void send(DatagramPacket p)`
Um método herdado da classe *DatagramSocket*. Envia um pacote a partir desse socket.

Tabela 5: Métodos de MulticastSocket

Antes que alguém possa enviar uma mensagem para um grupo, é preciso primeiro ser um membro do grupo *multicast* usando o método *joinGroup*. Um membro pode então enviar mensagens usando o método *send*. Depois que você tiver terminado de falar com o grupo, poderá usar o método *leaveGroup* para encerrar sua participação no grupo.

Antes de examinar um exemplo de uso da classe *MulticastSocket*, vamos antes dar uma olhada rápida na classe *DatagramPacket*. Observe que no método *send* da classe *MulticastSocket*, o parâmetro necessário é um objeto *DatagramPacket*. Assim, precisamos compreender esse tipo de (objetos) objeto, antes de usar o método *send*.

A classe *DatagramPacket* é usada para enviar dados por intermédio de um protocolo sem conexão, como um multicast. Um problema nisso é que o envio de pacotes não é garantido. Consideremos, agora, dois de seus seis construtores.

Construtores da classe *DatagramPacket*
`DatagramPacket(byte[] buf, int length)`
Constrói um pacote datagrama para receber pacotes com um comprimento *length*. *length* precisa ser menor ou igual ao tamanho do buffer *buf*.
`DatagramPacket(byte[] buf, int length, InetAddress address, int port)`
Constrói um pacote datagrama para enviar pacotes com um comprimento *length* para a porta de número especificado, no hospedeiro especificado.

Tavela 6: Construtores de datagramPacket

Segue alguns métodos interessantes da classe *DatagramPacket*.

Métodos da classe *DatagramPacket*
`public byte[] getData()`
Retorna o buffer no qual dados foram armazenados.
`public InetAddress getAddress()`
Retorna o endereço IP da máquina para onde o pacote está sendo enviado ou de onde foi recebido.
`public int getLength()`
Retorna o comprimento dos dados que estejam sendo enviados ou recebidos.
`public int getPort()`
Retorna o número da porta no hospedeiro remoto para onde os pacotes estão sendo enviados ou de onde foram recebidos.

Table 7: Métodos deDatagramPacket

Nosso exemplo de *multicast* também consiste em duas classes, uma servidora e uma cliente. A servidora recebe mensagens do cliente e imprime essas mensagens.

Aqui está a classe servidora.

```
import java.net.*;

public class ChatServer {
   public static void main(String args[]) throws Exception {
      MulticastSocket server = new MulticastSocket(1234);
      InetAddress group = InetAddress.getByName("234.5.6.7");
      //getByName - retorna o endereço IP dado para o host
      server.joinGroup(group);
      boolean infinite = true;
      /* O servidor recebe continuamente os dados e imprimindo-os */
      while(infinite) {
         byte buf[] = new byte[1024];
         DatagramPacket data = new DatagramPacket(buf, buf.length);
         server.receive(data);
         String msg = new String(data.getData()).trim();
         System.out.println(msg);
      }
      server.close();
   }
}
```

Aqui está a classe cliente.

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

public class ChatClient {
   public static void main(String args[]) throws Exception {
      MulticastSocket chat = new MulticastSocket(1234);
      InetAddress group = InetAddress.getByName("234.5.6.7");
      chat.joinGroup(group);
      String msg = "";
      System.out.println("Digite uma mensagem ao servidor:");
      BufferedReader br = new BufferedReader(new
         InputStreamReader(System.in));
      msg = br.readLine();
      DatagramPacket data = new DatagramPacket(msg.getBytes(),
         0, msg.length(), group, 1234);
      chat.send(data);
      chat.close();
   }
}
```

Aqui está uma amostra de execução das classes *ChatServer* e *ChatClient*, assumindo que ChatServer foi executada antes de rodarmos a classe cliente:

```
/* Executando a classe ChatServer para mostrar as mensagens aceitas do
cliente */

/* Executando ChatClient – simplesmente passe as mensagens para o
servidor */
Digite a mensagem para o Servidor:
Primeira mensagem para o Servidor

/* A classe ChatServer recebe e mostra as mensagens dos clientes */
Primeira mensagem do Servidor

/* Executando ChatClient novamente */
Digite a mensagem para o Servidor:
Segunda mensagem para o Servidor
```

```
/* A classe ChatServer recebe e mostra a mensagem do cliente */
Segunda mensagem do Servidor
```

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.