Módulo 5

Desenvolvimento de Aplicações Móveis

Lição 6

Redes

Versão 1.0 - Set/2007

Autor
xxx

Equipe
Rommel Feria
John Paul Petines

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

Aécio Júnior
Alexandre Mori
Alexis da Rocha Silva
Allan Souza Nunes
Allan Wojcik da Silva
Anderson Moreira Paiva
Andre Neves de Amorim
Angelo de Oliveira
Antonio Jose R. Alves Ramos
Aurélio Soares Neto
Bruno da Silva Bonfim
Carlos Fernando Gonçalves
Denis Mitsuo Nakasaki
Fábio Bombonato
Fabrício Ribeiro Brigagão
Francisco das Chagas
Frederico Dubiel
Herivelto Gabriel dos Santos
Jacqueline Susann Barbosa
João Vianney Barrozo Costa
Kefreen Ryenz Batista Lacerda
Kleberth Bezerra G. dos Santos
Leandro Silva de Morais
Leonardo Ribas Segala
Lucas Vinícius Bibiano Thomé
Luciana Rocha de Oliveira
Luiz Fernandes de Oliveira Junior
Marco Aurélio Martins Bessa
Maria Carolina Ferreira da Silva
Massimiliano Giroldi
Mauro Cardoso Mortoni
Paulo Afonso Corrêa
Paulo Oliveira Sampaio Reis
Pedro Henrique Pereira de Andrade
Ronie Dotzlaw
Seire Pareja
Sergio Terzella
Vanessa dos Santos Almeida
Robson Alves Macêdo

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
- **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, iremos estudar como acessar redes utilizando *MIDlets*.

Ao final desta lição, o estudante será capaz de:

- Descrever o *Framework* Genérico de Conexão e como pode ser usado para suportar diferentes métodos de conexão
- Especificar argumentos de conexão usando o formato de endereço URL do GCF
- Criar conexões HTTP/HTTPS
- Criar *MIDlets* usando soquetes TCP e *datagramas* UDP

2. Framework Genérico de Conexão

O Framework Genérico de Conexão suporta as conexões baseadas em pacotes (através de *Soquetes*) e baseadas em *stream* (através de *Datagramas*). Como o nome diz, este *framework* provê uma API básica para conexões em CLDC. Determina uma base comum para conexões como HTTP, soquetes e *datagramas*. Mesmo *bluetooth* e serial I/O tem um lugar neste *framework*. Fornece um conjunto genérico e comum da API que abstrai todos os tipos de conexões. Deve-se notar que nem todos os tipos de conexão devem ser requeridos para ser implementados pelos dispositivos *MIDP*.

2.1. Hierarquia da Interface GCF

A hierarquia extensível da interface do GCF torna possível a generalização. Um novo tipo de conexão pode ser adicionado a este *framework* através da extensão desta hierarquia.

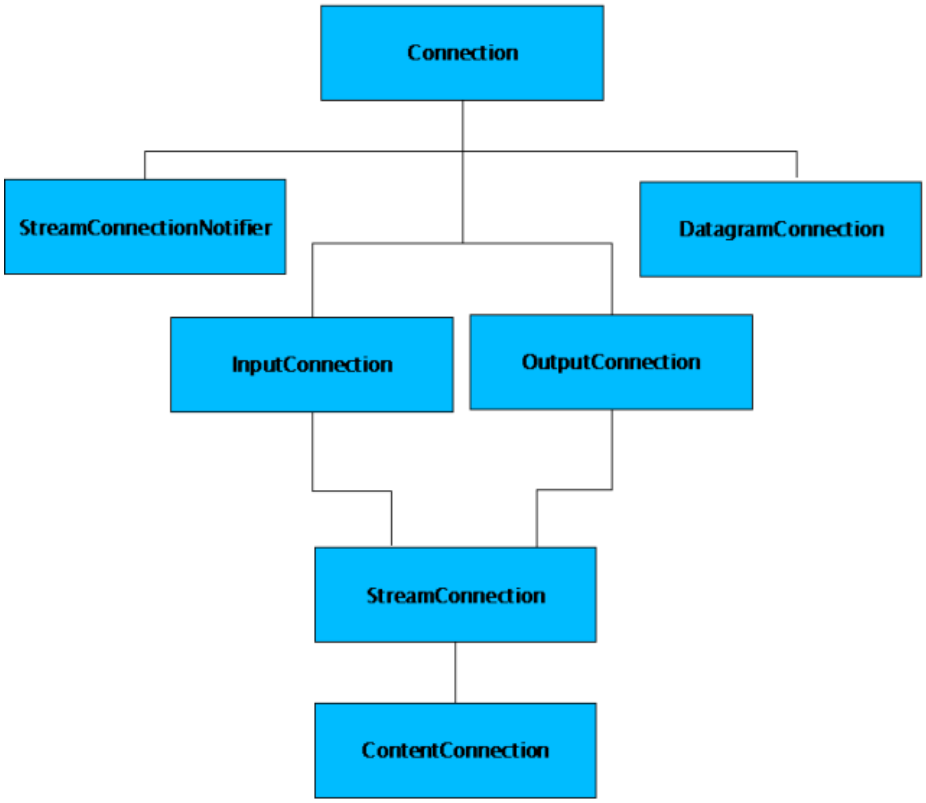


Figura 1: Hierarquia da Interface GCF

2.2. A URL de Conexão GCF

Argumentos de conexão são especificados usando o formato de endereçamento:

```
scheme://username:password@host:port/path;parameters
```

1. ***scheme*** (esquema) é o protocolo ou método de conexão. Exemplos de esquemas são: http, ftp e https
2. ***username*** (nome do usuário) é opcional, entretanto, caso seja especificado, um @ deve preceder o *host*
3. ***password*** (senha do usuário) é opcional e pode ser especificada somente se o ***username*** estiver presente. Se a ***password*** estiver presente, deve estar separada do username por dois pontos (:)

4. ***host*** (domínio) este argumento é obrigatório. Pode ser um *hostname*, um nome completo no domínio (FQDN) ou o endereço IP do *host* alvo.
5. ***port*** (porta) este argumento é opcional. Caso não seja especificado, a porta padrão para o esquema será utilizada.
6. **path** (caminho)
7. **parameters** (argumentos) este é opcional, mas deve estar precedido por ponto e vírgula quando presente

Se utilizarmos colchetes para definir os argumentos opcionais neste formato de endereçamento, poderemos expressar da seguinte forma:

```
scheme://[username[:password]@]host[:port]/path[;parameters]
```

O *Uniform Resource Indicator* (URI) está definido no RFC 2396, que é a base para este formato de endereçamento. No MIDP 2.0, somente os esquemas "http" e "https" devem ser implementados por estes dispositivos.

3. Conexão HTTP

3.1. O Protocolo HTTP

HTTP significa Protocolo de Transporte de Hiper Texto. Este é o protocolo utilizado para transferir páginas da WEB de seus servidores (ex. www.sun.com) para os *WEB Browsers*. O cliente (*WEB Browser*) requisita uma página, especificando o seu caminho com os modelos do tipo "*GET*" ou "*POST*".

Para o modelo "*GET*", argumentos são especificados e embutidos na *URL*. Por exemplo, para passar um atributo com o nome "id" e valor 100 a página "index.jsp", a URL é especificada da seguinte forma: "http://hostname/index.jsp?id=100". Argumentos adicionais são separados pelo simbolo &, por exemplo: "http://hostname/index.jsp?id=100&page=2".

Quando o modelo "*POST*" é utilizado, argumentos não fazem parte da *URL*, mas são enviados em linhas diferentes após o comando *POST*.

Cliente / Navegador WEB	Servidor HTTP

`GET /index.jsp?id=100 HTTP/1.1`	`HTTP/1.1 200 OK`
	`Server: Apache-Coyote/1.1`
	`Content-Type: text/html;charset=ISO-8859-1`
	`Date: Wed, 18 Jun 2005 14:09:31 GMT`
	`Connection: close`
	
	`<html>`
	`  <head>`
	`  <title>Test Page</title>`
	`  </head>`
	`  <body>`
	`  <h1 align="center">Test Page</h1>`
	`  </body>`
	`</html>`

Figura 2: Exemplo de Transação HTTP GET

Cliente / Navegador WEB	Servidor HTTP

`GET /non-existent.html HTTP/1.0`	`HTTP/1.1 404 /non-existent.html`
	`Server: Apache-Coyote/1.1`
	`Content-Type: text/html;charset=utf-8`
	`Content-Length: 983`
	`Date: Mon, 11 Jul 2005 13:21:01 GMT`
	`Connection: close`
	
	`<html><head><title>Apache Tomcat/5.5.7 - Error report</title><style>...`
	`<body><h1>HTTP Status 404</h1>`
	`...`
	`The requested resource (non-existent.html) is not available.`
	`...`
	`</body></html>`

Figura 3: Exemplo de transação HTTP GET com um response erro

3.2. Criando uma conexão HTTP

Podemos abrir uma conexão HTTP usando o método *Connector.open*() e fazer um casting com uma das seguintes interfaces: *StreamConnection*, *ContentConnection* ou *HTTPConnection*. Entretanto, com as interfaces *StreamConnection* e *ContentConnection* podemos especificar e derivar parâmetros específicos do HTTP resultante.

Quando se utiliza a *StreamConnection*, o tamanho da resposta não pode ser determinado previamente. Com a interface *ContentConnection* ou *HTTPConnection* existe a possibilidade de se determinar o tamanho da resposta. Porém, o tamanho não está sempre disponível, então o

programa deverá que ser capaz de recorrer a outros meios para obter a resposta sem o conhecimento prévio do tamanho.

```
HttpConnection connection = null;
InputStream iStream = null;
byte[] data = null;

try {
  connection = (HttpConnection) Connector.open("http://www.sun.com/");
  int code = connection.getResponseCode();
  switch (code) {
    case HttpConnection.HTTP_OK:
      iStream = connection.openInputStream();
      int length = (int) connection.getLength();
      if (length > 0){
        data = new byte[length];
        int totalBytes = 0;
        int bytesRead = 0;
        while ((totalBytes < length) && (bytesRead > 0)) {
          bytesRead = iStream.read(
          data, totalBytes, length - totalBytes);
          if (bytesRead > 0){
            totalBytes += bytesRead;
          }
        }
      } else {
          // Tamanho não é conhecido, ler por caracter
          ...
      }
      break;
    default:
      break;
  }
  ...
```

3.3. Controlando Redirecionamentos HTTP

Algumas vezes o servidor redireciona o navegador do cliente para outras páginas WEB através de uma resposta que pode ser através de uma *HTTP_MOVED_PERM* (301), *HTTP_MOVED_TEMP* (302), *HTTP_SEE_OTHER* (303) ou *HTTP_TEMP_REDIRECT* (307) ao invés da resposta comum *HTTP_OK*. O programa terá de ser capaz de detectar isso utilizando o método *getResponseCode*(), obter a nova *URI* do cabeçalho utilizando o método *getHeaderField*("Localização") e recuperar esse documento na nova localização.

```
int code = connection.getResponseCode();
switch(code){
  case HttpConnection.HTTP_MOVED_PERM:
  case HttpConnection.HTTP_MOVED_TEMP:
  case HttpConnection.HTTP_SEE_OTHER:
  case HttpConnection.HTTP_TEMP_REDIRECT:
    String newUrl = conn.getHeaderField("Location");
  ...
```

4. Conexões HTTPS

HTTPS é um modelo HTTP sobre uma conexão segura de transporte. A abertura de uma conexão do tipo HTTPS é realizada de forma idêntica a abrir uma conexão do tipo HTTP. A única diferença é que a URL é passada para *Connector.open*() e o resultado moldado para uma variável da classe *HttpsConnection*.

Um tipo adicional de exceção pode ser lançada por um método *Connector.open*() ao invés das exceções comuns *IllegalArgumentException*, *ConnectionNotFoundException*, *java.io.IOException* e *SecurityException*. Uma exceção do tipo *CertificateException* pode ser disparada por causa de falhas no certificado.

```
import javax.microedition.io.*;

HttpsConnection connection = null;
InputStream iStream = null;
byte[] data = null;
try {
  connection = (HttpsConnection) Connector.open("https://www.sun.com/");
  int code = connection.getResponseCode();
  ...
} catch (CertificateException ce){
  switch (ce.getReason()){
    case CertificateException.EXPIRED:
      ...
  }
}
```

Todos as constantes listadas a seguir foram retiradas da especificação MIDP 2.0 – JSR 118 e são do tipo byte estáticas.

BAD_EXTENSIONS	Indicar que um certificado possui extensões críticas desconhecidas
BROKEN_CHAIN	Indicar que um certificado numa cadeia não foi gerado pela próxima autoridade na cadeia
CERTIFICATE_CHAIN_TOO_LONG	Indicar que o tamanho da cadeia de servidores certificados excedeu o tamanho permitido pela política do emissor
EXPIRED	Indicar que um certificado expirou
INAPPROPRIATE_KEY_USAGE	Indicar que a chave pública do certificado foi usada de maneira considerada inapropriada pelo emissor
MISSING_SIGNATURE	Indicar que um objeto certificado não contém uma assinatura
NOT_YET_VALID	Indicar que um certificado ainda não é válido
ROOT_CA_EXPIRED	Indicar que a chave pública da autoridade certificadora raiz (root CA) expirou
SITENAME_MISMATCH	Indicar que um certificado não contém o nome correto do site
UNAUTHORIZED_INTERMEDIATE_CA	Indicar que um certificado intermediário na cadeia não possui a permissão para ser uma autoridade certificadora intermediária
UNRECOGNIZED_ISSUER	Indicar que um certificado foi emitido por uma entidade desconhecida
UNSUPPORTED_PUBLIC_KEY_TYPE	Indicar que o tipo da chave pública no certificado não é suportado pelo dispositivo
UNSUPPORTED_SIGALG	Indicar que um certificado foi assinado utilizando um algoritmo não suportado
VERIFICATION_FAILED	Indicar uma verificação falha de certificado

Figura 4: Razões para uma exceção CertificateException

5. *Sockets* TCP

A maior parte das implementações de HTTP funcionam no topo da camada TCP. Ao enviar dados usando a camada TCP, estes podem ser divididos em pedaços menores chamado pacotes. A camada TCP garante que todos os pacotes enviados pelo transmissor serão recebidos pelo receptor na mesma ordem que eles foram enviados. Se um pacote não for recebido pelo receptor, ele será reenviado. Isso significa que quando uma mensagem é enviada, podemos ter certeza que ela será entregue para o receptor no mesmo formato na qual foi enviada, sem omissões ou inserções (exceto em circunstâncias extremas como o receptor sendo desconectado da rede).

A camada TCP é o responsável pela remontagem dos pacotes e retransmissão de forma transparente. Por exemplo, o protocolo HTTP não se preocupa com a montagem e desmontagem de pacotes porque isso deve ser tratados pela camada TCP.

Algumas vezes, o tamanho da mensagem é muito pequeno e torna-se muito ineficiente para ser enviado através de um único pacote (a sobrecarga do pacote é muito grande se comparada aos dados a serem enviado). Imagine muitos pacotes percorrendo a rede com apenas um byte de dados e muitos bytes de sobrecarga (digamos 16 bytes). Isso poderia fazer a rede ser muito ineficiente, muitos pacotes inundariam a rede com alguns poucos bytes de dados.

Nesses casos, a implementação do TCP deve aguardar mensagens subsequentes para serem enviadas. Então, poderá empacotar várias mensagens antes de enviar o pacote. Se isso ocorrer, poderá haver atraso ou latência na conexão. Se sua aplicação requer que a latência seja a menor possível, você deverá configurar a opção DELAY do socket para zero. Ou, se a aplicação pode conviver com perdas de pacotes ou pacotes desordenados, você pode querer testar uma conexão UDP ou *Datagrama*. Conexões UDP também possuem menos sobrecarga de pacotes.

5.1. Parte Cliente

Iremos na primeira parte do projeto construir as classes de comunicação do lado cliente para proceder à comunicação via *sockets*.

Classe ClientSocketMidlet.java:

```
package socket;

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class ClientSocketMidlet extends MIDlet {

  private static Display display;
  private boolean isPaused;

  public void startApp() {
    isPaused = false;
    display = Display.getDisplay(this);
    Client client = new Client(this);
    client.start();
  }
  public static Display getDisplay() {
    return display;
  }
  public boolean isPaused() {
    return isPaused;
  }
  public void pauseApp() {
    isPaused = true;
  }
  public void destroyApp(boolean unconditional) {
  }
}
```

Classe Client.java:

```
package socket;

import java.io.*;
import javax.microedition.io.*;
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;

public class Client implements Runnable, CommandListener {
  private ClientSocketMidlet parent;
  private Display display;
  private Form f;
  private StringItem si;
  private TextField tf;
  private boolean stop;
  private Command sendCommand = new Command("Send", Command.ITEM, 1);
  private Command exitCommand = new Command("Exit", Command.EXIT, 1);
  InputStream is;
  OutputStream os;
  SocketConnection sc;
  Sender sender;

  public Client(ClientSocketMidlet m) {
    parent = m;
    display = Display.getDisplay(parent);
    f = new Form("Socket Client");
    si = new StringItem("Status:", " ");
    tf = new TextField("Send:", "", 30, TextField.ANY);
    f.append(si);
    f.append(tf);
    f.addCommand(exitCommand);
    f.addCommand(sendCommand);
    f.setCommandListener(this);
    display.setCurrent(f);
  }
  public void start() {
    Thread t = new Thread(this);
    t.start();
  }
  public void run() {
    try {
      sc = (SocketConnection) Connector.open("socket://localhost:5000");
      si.setText("Connected to server");
      is = sc.openInputStream();
      os = sc.openOutputStream();
      sender = new Sender(os);
      while (true) {
        StringBuffer sb = new StringBuffer();
        int c = 0;
        while (((c = is.read()) != '\n') && (c != -1)) {
          sb.append((char) c);
        }
        if (c == -1) {
          break;
        }
        si.setText("Message received - " + sb.toString());
      }
      stop();
      si.setText("Connection closed");
      f.removeCommand(sendCommand);
    } catch (ConnectionNotFoundException cnfe) {
      Alert a = new Alert("Client",
        "Please run Server MIDlet first", null, AlertType.ERROR);
      a.setTimeout(Alert.FOREVER);
      a.setCommandListener(this);
      display.setCurrent(a);
    } catch (IOException ioe) {
      if (!stop) {
```

```
          ioe.printStackTrace();
        }
      } catch (Exception e) {
        e.printStackTrace();
      }
    }
    public void commandAction(Command c, Displayable s) {
      if ((c == sendCommand) && !parent.isPaused()) {
        sender.send(tf.getString());
      }
      if ((c == Alert.DISMISS_COMMAND) || (c == exitCommand)) {
        parent.notifyDestroyed();
        parent.destroyApp(true);
      }
    }
    public void stop() {
      try {
        stop = true;
        if (sender != null) {
          sender.stop();
        }
        if (is != null) {
          is.close();
        }
        if (os != null) {
          os.close();
        }
        if (sc != null) {
          sc.close();
        }
      } catch (IOException ioe) {
      }
    }
  }
```

Classe Sender.java:

```
  package socket;

  import java.io.*;
  import javax.microedition.io.*;
  import javax.microedition.lcdui.*;
  import javax.microedition.midlet.*;

  public class Sender extends Thread {
    private OutputStream os;
    private String message;

    public Sender(OutputStream os) {
      this.os = os;
      start();
    }
    public synchronized void send(String msg) {
      message = msg;
      notify();
    }
    public synchronized void run() {
      while (true) {
        if (message == null) {
          try {
            wait();
          } catch (InterruptedException e) {
          }
        }
        if (message == null) {
          break;
        }
```

```
        try {
          os.write(message.getBytes());
          os.write("\r\n".getBytes());
        } catch (IOException ioe) {
          ioe.printStackTrace();
        }
        message = null;
      }
    }
    public synchronized void stop() {
      message = null;
      notify();
    }
  }
```

6. Server Sockets

No modelo cliente-servidor, o servidor continuamente espera pela conexão de um cliente que deverá conhecer o número da porta deste. Devemos utilizar o método *Connector.open()* para criar uma conexão do tipo *socket*. A URL passada para o método possui um formato semelhante ao um *Socket TCP*, com um *hostname* passado em branco (isto é, *socket://:5000*).

6.1. Parte Servidora

Iremos agora complementar as classes de comunicação via *socket* criando as classes que resolverão a parte do servidor.

Classe ServerSocketMidlet.java:

```
package socket;

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class ServerSocketMidlet extends MIDlet {
  private static Display display;
  private boolean isPaused;

  public void startApp() {
    isPaused = false;
    display = Display.getDisplay(this);
    Server server = new Server(this);
    server.start();
  }
  public static Display getDisplay() {
    return display;
  }
  public boolean isPaused() {
    return isPaused;
  }
  public void pauseApp() {
    isPaused = true;
  }
  public void destroyApp(boolean unconditional) {
  }
}
```

Classe Server.java:

```
package socket;

import java.io.*;
import javax.microedition.io.*;
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;

public class Server implements Runnable, CommandListener {
  private ServerSocketMidlet parent;
  private Display display;
  private Form f;
  private StringItem si;
  private TextField tf;
  private boolean stop;
  private Command sendCommand = new Command("Send", Command.ITEM, 1);
  private Command exitCommand = new Command("Exit", Command.EXIT, 1);
  InputStream is;
  OutputStream os;
  SocketConnection sc;
  ServerSocketConnection scn;
  Sender sender;
```

```
    public Server(ServerSocketMidlet m) {
      parent = m;
      display = Display.getDisplay(parent);
      f = new Form("Socket Server");
      si = new StringItem("Status:", " ");
      tf = new TextField("Send:", "", 30, TextField.ANY);
      f.append(si);
      f.append(tf);
      f.addCommand(exitCommand);
      f.setCommandListener(this);
      display.setCurrent(f);
    }
    public void start() {
      Thread t = new Thread(this);
      t.start();
    }
    public void run() {
      try {
        si.setText("Waiting for connection");
        scn = (ServerSocketConnection) Connector.open("socket://:5000");
        sc = (SocketConnection) scn.acceptAndOpen();
        si.setText("Connection accepted");
        is = sc.openInputStream();
        os = sc.openOutputStream();
        sender = new Sender(os);
        f.addCommand(sendCommand);
        while (true) {
          StringBuffer sb = new StringBuffer();
          int c = 0;
          while (((c = is.read()) != '\n') && (c != -1)) {
            sb.append((char) c);
          }
          if (c == -1) {
            break;
          }
          si.setText("Message received - " + sb.toString());
        }
        stop();
        si.setText("Connection is closed");
        f.removeCommand(sendCommand);
      } catch (IOException ioe) {
        if (ioe.getMessage().equals("ServerSocket Open")) {
          Alert a = new Alert("Server", "Port 5000 is already taken.", null,
            AlertType.ERROR);
          a.setTimeout(Alert.FOREVER);
          a.setCommandListener(this);
          display.setCurrent(a);
        } else {
          if (!stop) {
            ioe.printStackTrace();
          }
        }
      } catch (Exception e) {
        e.printStackTrace();
      }
    }
    public void commandAction(Command c, Displayable s) {
      if ((c == sendCommand) && !parent.isPaused()) {
        sender.send(tf.getString());
      }
      if ((c == Alert.DISMISS_COMMAND) || (c == exitCommand)) {
        parent.notifyDestroyed();
        parent.destroyApp(true);
      }
    }
    public void stop() {
```

```
      try {
        stop = true;
        if (is != null) {
          is.close();
        }
        if (os != null) {
          os.close();
        }
        if (sc != null) {
          sc.close();
        }
        if (scn != null) {
          scn.close();
        }
      } catch (IOException ioe) {
      }
    }
  }
```

7. Datagramas

Conexões por soquetes *TCP* são seguras. Ao contrário, entregas de pacotes *UDP* não são garantidas. Não há garantias de que pacotes enviados utilizando conexões por *datagramas* sejam recebidas por quem as solicitou. A ordem em que os pacotes são recebidos não é segura. A ordem em que os pacotes são enviados não é a mesma ordem em que são recebidos.

Datagramas ou pacotes UDP são utilizados quando a aplicação pode se sustentar (continuar em operação) mesmo quando um pacote está perdido ou quando não são entregues em ordem.

7.1. Parte Servidora

Iniciaremos o projeto de comunicação por datagramas através da parte servidora.

Classe ServerDatagramMidlet.java:

```
package datagram;

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class ServerDatagramMidlet extends MIDlet {
  private static Display display;
  private boolean isPaused;

  public void startApp() {
    isPaused = false;
    display = Display.getDisplay(this);
    Server server = new Server(this);
    server.start();
  }
  public static Display getDisplay() {
    return display;
  }
  public boolean isPaused() {
    return isPaused;
  }
  public void pauseApp() {
    isPaused = true;
  }
  public void destroyApp(boolean unconditional) {
  }
}
```

Classe Server.java:

```
package datagram;

import java.io.*;
import javax.microedition.io.*;
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;

public class Server implements Runnable, CommandListener {
  private ServerDatagramMidlet parent;
  private Display display;
  private Form f;
  private StringItem si;
  private TextField tf;
  private Command sendCommand = new Command("Send", Command.ITEM, 1);
  private Command exitCommand = new Command("Exit", Command.EXIT, 1);
  private Sender sender;
  private String address;

  public Server(ServerDatagramMidlet m) {
```

```
      parent = m;
      display = Display.getDisplay(parent);
      f = new Form("Datagram Server");
      si = new StringItem("Status:", " ");
      tf = new TextField("Send:", "", 30, TextField.ANY);
      f.append(si);
      f.append(tf);
      f.addCommand(sendCommand);
      f.addCommand(exitCommand);
      f.setCommandListener(this);
      display.setCurrent(f);
    }
    public void start() {
      Thread t = new Thread(this);
      t.start();
    }
    public void run() {
      try {
        si.setText("Waiting for connection");
        DatagramConnection dc = (DatagramConnection)
          Connector.open("datagram://:5555");
        sender = new Sender(dc);
        while (true) {
          Datagram dg = dc.newDatagram(100);
          dc.receive(dg);
          address = dg.getAddress();
          si.setText("Message received - " + new String(dg.getData(), 0,
            dg.getLength()));
        }
      } catch (IOException ioe) {
        Alert a = new Alert("Server", "Port 5000 is already taken.", null,
          AlertType.ERROR);
        a.setTimeout(Alert.FOREVER);
        a.setCommandListener(this);
        display.setCurrent(a);
      } catch (Exception e) {
        e.printStackTrace();
      }
    }
    public void commandAction(Command c, Displayable s) {
      if ((c == sendCommand) && !parent.isPaused()) {
        if (address == null) {
          si.setText("No destination address");
        } else {
          sender.send(address, tf.getString());
        }
      }
      if ((c == Alert.DISMISS_COMMAND) || (c == exitCommand)) {
        parent.destroyApp(true);
        parent.notifyDestroyed();
      }
    }
  }
```

7.2. Classe de Comunicação

Esta classe é aproveitada tanto pelo lado servidor quanto pelo lado cliente para o controle do envio das mensagens.

Classe Sender.java:

```
package datagram;

import java.io.*;
import javax.microedition.io.*;
import javax.microedition.lcdui.*;
```

```
import javax.microedition.midlet.*;

public class Sender extends Thread {
  private DatagramConnection dc;
  private String address;
  private String message;

  public Sender(DatagramConnection dc) {
    this.dc = dc;
    start();
  }
  public synchronized void send(String addr, String msg) {
    address = addr;
    message = msg;
    notify();
  }
  public synchronized void run() {
    while (true) {
      if (message == null) {
        try {
          wait();
        } catch (InterruptedException e) {
        }
      }
      try {
        byte[] bytes = message.getBytes();
        Datagram dg = null;
        if (address == null) {
          dg = dc.newDatagram(bytes, bytes.length);
        } else {
          dg = dc.newDatagram(bytes, bytes.length, address);
        }
        dc.send(dg);
      } catch (Exception ioe) {
        ioe.printStackTrace();
      }
      message = null;
    }
  }
}
```

7.3. Parte Cliente

Complementaremos o projeto com as classes que definem a parte cliente.

Classe ClientDatagramMidlet.java

```
package datagram;

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class ClientDatagramMidlet extends MIDlet {
  private static Display display;
  private boolean isPaused;

  public void startApp() {
    isPaused = false;
    display = Display.getDisplay(this);
    Client client = new Client(this);
    client.start();
  }
  public static Display getDisplay() {
    return display;
  }
  public boolean isPaused() {
```

```
    return isPaused;
  }
  public void pauseApp() {
    isPaused = true;
  }
  public void destroyApp(boolean unconditional) {
  }
}
```

Classe Client.java:

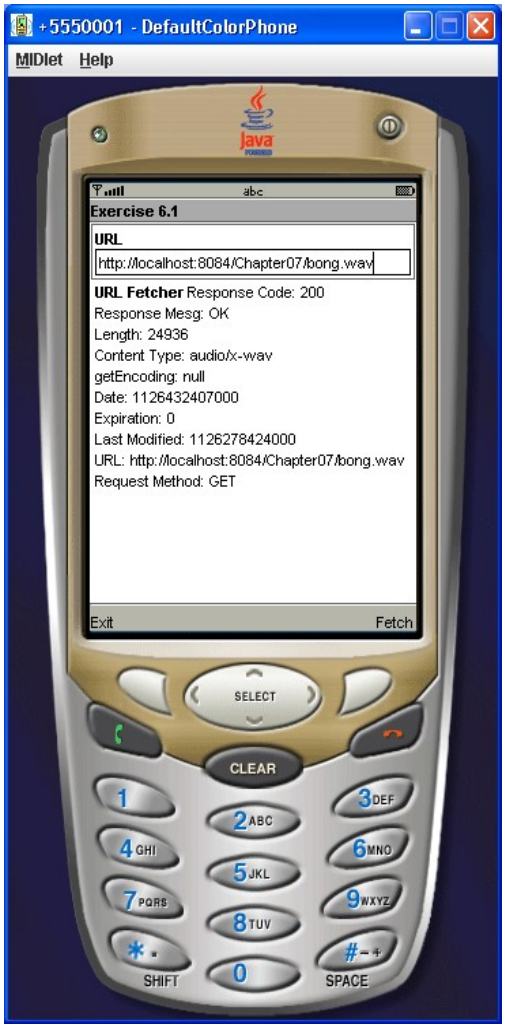
```
package datagram;

import java.io.*;
import javax.microedition.io.*;
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;

public class Client implements Runnable, CommandListener {

  private ClientDatagramMidlet parent;
  private Display display;
  private Form f;
  private StringItem si;
  private TextField tf;
  private Command sendCommand = new Command("Send", Command.ITEM, 1);
  private Command exitCommand = new Command("Exit", Command.EXIT, 1);
  private Sender sender;

  public Client(ClientDatagramMidlet m) {
    parent = m;
    display = Display.getDisplay(parent);
    f = new Form("Datagram Client");
    si = new StringItem("Status:", " ");
    tf = new TextField("Send:", "", 30, TextField.ANY);
    f.append(si);
    f.append(tf);
    f.addCommand(sendCommand);
    f.addCommand(exitCommand);
    f.setCommandListener(this);
    display.setCurrent(f);
  }
  public void start() {
    Thread t = new Thread(this);
    t.start();
  }
  public void run() {
    try {
      DatagramConnection dc = (DatagramConnection) Connector.open(
        "datagram://localhost:5555");
      si.setText("Connected to server");
      sender = new Sender(dc);
      while (true) {
        Datagram dg = dc.newDatagram(100);
        dc.receive(dg);
        if (dg.getLength() > 0) {
          si.setText("Message received - " + new String(dg.getData(), 0,
            dg.getLength()));
        }
      }
    } catch (ConnectionNotFoundException cnfe) {
      Alert a = new Alert("Client", "Please run Server MIDlet first", null,
        AlertType.ERROR);
      a.setTimeout(Alert.FOREVER);
      display.setCurrent(a);
    } catch (IOException ioe) {
      ioe.printStackTrace();
```

```
      }
    }
    public void commandAction(Command c, Displayable s) {
      if ((c == sendCommand) && !parent.isPaused()) {
        sender.send(null, tf.getString());
      }
      if (c == exitCommand) {
        parent.destroyApp(true);
        parent.notifyDestroyed();
      }
    }
  }
```

8. Exercícios

8.1. Buscar dados da URL

Criar um *MIDlet* que detalhe um endereço *HTTP*. Pesquisar a *URL* utilizando o método *GET* e mostrar as propriedades desta conexão (caso esteja disponível), tais como: código da resposta, descrição da mensagem, tamanho, tipo, modo de codificação, se está finalizado e data da última modificação.



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.