Apostila de J2ME

versão 1.0

por Juliano Carniel e Clóvis Teixeira

juliano@dainf.cefetpr.br
clv@dainf.cefetpr.br

Índice

Apostila de J2ME

- Introdução:

Comecemos com uma pergunta básica porém não menos importante. O que vem a ser esse J2ME? E para que serve?

Java 2 Micro Edition é uma API Java voltada para micro aplicativos que rodam em micro processadores assim como os dos celulares.

Neste Tutorial vamos tratar da Tecnologia J2ME MIDP 1.0 voltada para aplicações gráficas porém não vamos tratar de jogos aqui, embora após a leitura deste, você pode ter um embasamento para começar a desenvolver algo na área. As aplicações criadas usando-se MIDP são chamadas MIDlets (assim como Servlets e Applets).

Usamos os programas:

- Wireless Tollkit que é um software gratuito feito pela Sun para simplificar o ciclo de desenvolvimento do J2ME o qual pode ser encontrado em http://wireless.java.sun.com/allsoftware/ (é necessário que você já tenha o j2sdk instalado http://java.sun.com/j2se/1.4.1/download.html), você ainda pode baixar outros emuladores encontrados na mesma pagina do Wireless Toolkit para posteriores testes.

- E o editor GEL que é free, e é um ótimo editor facilitando muito na edição com auto-complete de métodos e muitas outras facilidades que este programa provê, e pode ser encontrado em www.gexperts.com

Caso você já tenha algum outro programa como o Eclipse por exemplo, pode-se usa-lo também fazendo as devidas modificações.

Como já citado trabalhamos com MIDP 1.0, e alguns podem se perguntar por que não a 2.0? Todos celulares existentes hoje, ou a grande maioria, usa a tecnologia 1.0 então ainda não se justifica o uso da versão 2.0, e iremos demonstrar posteriormente algumas diferenças, embora não sejam muitas, e nem gritantes.

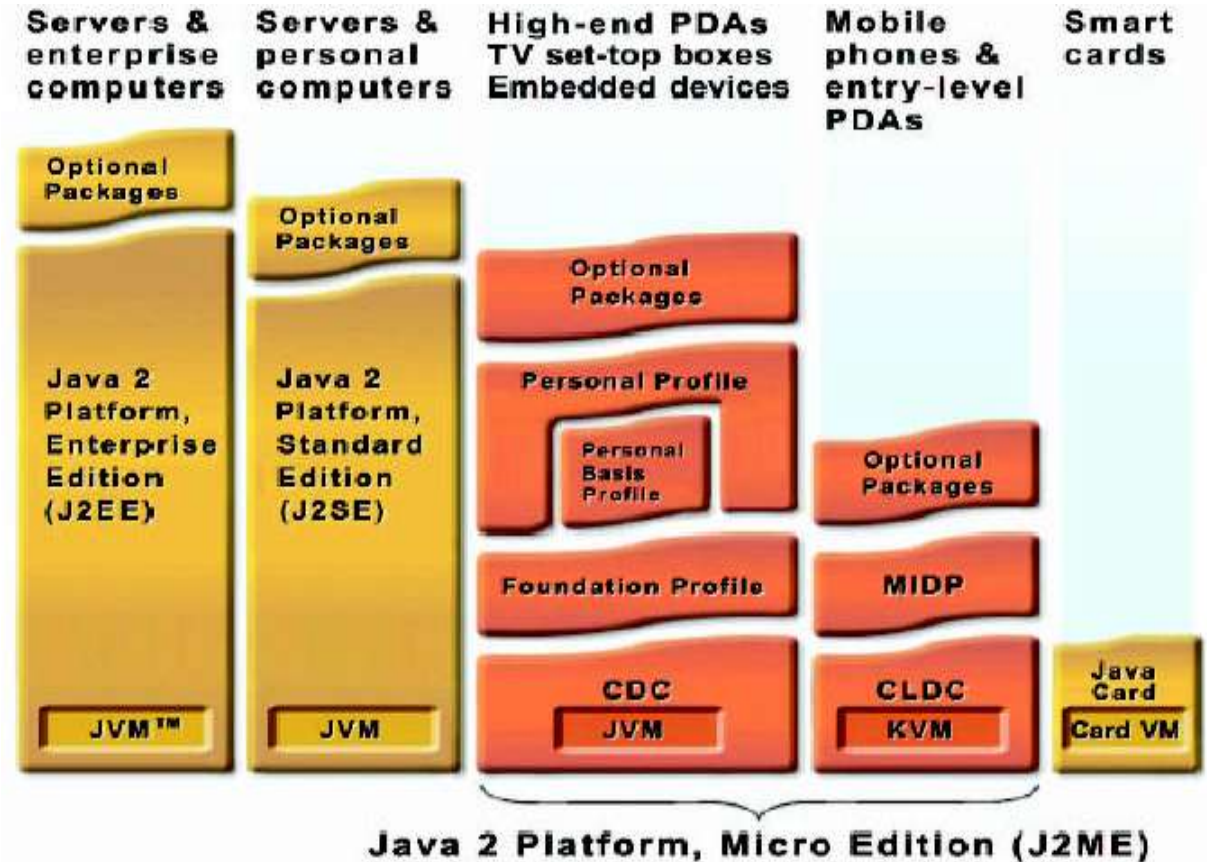
- Entendendo J2ME

J2ME é dividido em configurations, profiles e API's opcionais.

Para entendermos melhor, pensemos da seguinte maneira, Profiles são mais específicos que configurations, e fazendo analogia a um velho exemplo temos uma abstração sobre o que é um carro e como ele é fabricado (configuration) e como um Ford é fabricado (profile), mais tecnicamente falando profile é baseado em configuration e ainda acima dos profiles estão as API's que na nossa analogia seria um modelo específico da Ford.

Existem dois "configurations", um configuration é o CLDC (Connected, Limited Device Configuration), que rege as configurações para aparelhos bem pequenos como celulares ou PDA's, o qual fica acima das diretrizes J2ME juntamente com CDC (Connected Device Configuration) o que rege as configurações para aparelhos um pouco maiores, mas mesmo assim pequenos.

Podem haver vários Profiles vamos citar dois aqui os quais são os mais importantes para este estudo, MIDP (Mobile Information Device Profile) e também o PDAP (Personal Digital Assistant Profile) e ambos estão acima do CLDC.



CLDC como já vimos rege as configurações para aparelhos extremamente pequenos, ele foi desenvolvido para dispositivos de 160KB à 512KB com memória valida para Java. Você não leu errado e nem nós trocamos unidades de medidas, a memória está realmente certa, o que nos faz pensar muito em termos de aplicações que podem rodar neles e nos traz de volta a certas programações para DOS no que diz respeito à memória. E falando em memória relevemos também o processamento que é muito fraco,

mas muito mesmo não mais que 10Mhz e isto se tratando de aparelhos tops de linha (até a composição desta materia), o que nos faz analisar melhor códigos e métodos usados, os quais dispendem de muito processamento e uso de memória. E também a conexão lenta, tipicamente de 9.600bps.

MIDP tem as seguintes características:

-128kB de memória não-volátil para JAVA.

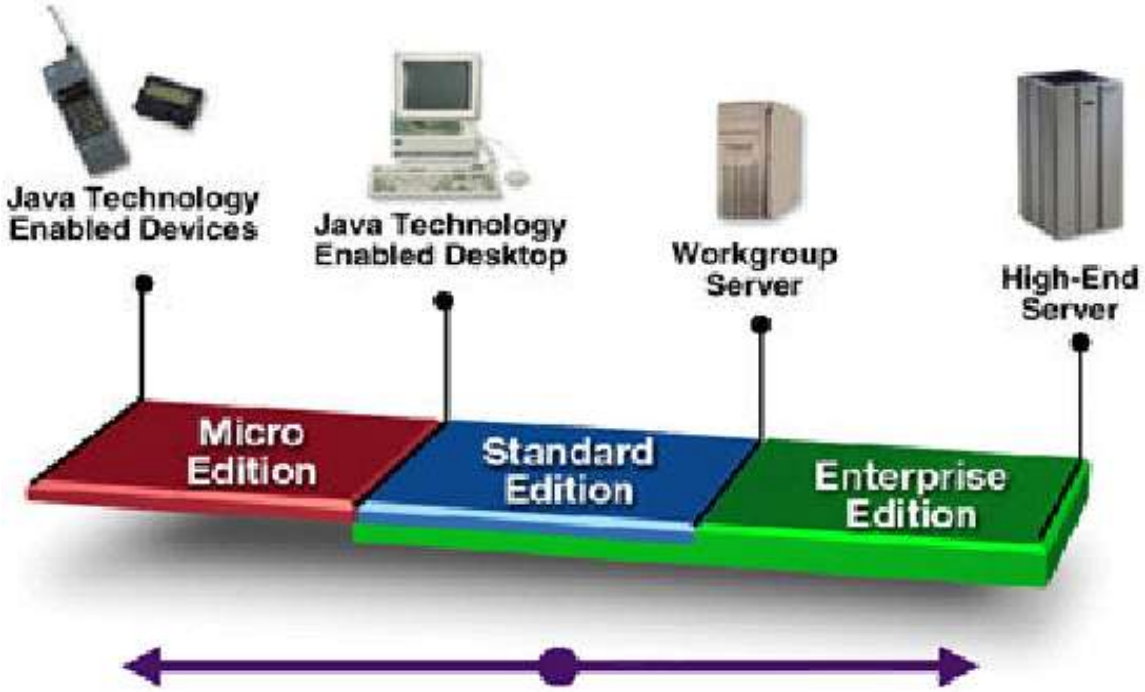
-32KB de memória volátil para tempo de execução.

-8KB de memória não-volátil de para armazenamento de dados.

-uma tela de pelo menos 96x54 pixels (aqui já cai por terra a propaganda de muitos celulares, os quais dizem ter uma tela ampla de 96x54, o que é o mínimo necessário).

-Capacidade de entrada de dados seja por teclado (do celular), teclado externo ou mesmo Touch-screen.

-Possibilidade de enviar e receber dados em conexão.



- Segurança

Uma pergunta freqüentemente feita é: Agora com programas rodando nos celulares não iremos ter problemas de vírus, ou programas maliciosos? A resposta é não, pois o Java não tem acesso a API do celular em si, ou seja, ele não poderá acessar funções especificas do celular como, por exemplo, a agenda telefônica, sendo assim não poderão ser apagados ou modificados seus dados, a não ser que o fabricante lhe disponibilize esta API, como é o caso dos jogos nos celulares, cada fabricante disponibiliza uma API específica para os jogos, podendo assim aproveitar melhor o desempenho do aparelho, porém cai por terra a forte característica de vida do JAVA, a (WORA) "Write Once, Run Anywhere!", apesar de ser mais seguro, perde-se um pouco e funcionalidade.

A maquina virtual tem um espaço independente de memória, e não pode acessar

a memória correspondente às aplicações nativas do celular.

Outro ponto importante é que não é permitida a carga de classes definidas pelo usuário, ou seja, o usuário não tem acesso a outras funções que não seja as da VM, o que aumenta a segurança, porém restringe um pouco o desenvolvimento.

- Tecnologias Wireless

Muitas pessoas ainda tem dúvidas sobre as diferentes tecnologias de celulares existentes. Vamos tentar defini-las e diferencia-las em poucas palavras, sendo que este não é o enfoque principal deste Tutorial.

- *TDMA*: (time division multiple access, ou acesso múltiplo por divisão do tempo), ou seja, ele não diferencia voz e dados então para navegar na internet(dados) você paga o mesmo preço que pagaria por fazer uma ligação de voz. Quem tem algum celular tdma sabe o quão caro e lento é isso. Somente para estabelecer a conexão leva-se 8 segundos e a transferência de dados da-se a 9.600bps.

- *CDMA:* (code division multiple access, ou acesso múltiplo por divisão de códigos), ou seja, é separado voz de dados, e tanto esta tecnologia quando a gsm são conexão 100% ativa, ou seja, não existe este delay de conexão você requisita uma informação e ela vem diretamente. Além de a taxa de transferência que é maior chegando a 256Kbps usando CDMA 1xRTT, que é o que esta em vigor.

- *GSM:* (global system for communication, ou sistema global de comunicação), ou seja, tem as mesmas características da tecnologia CDMA, porém opera usando GPRS.

Caso haja maior interesse nas tecnologias de transmissão e na evolução e tudo mais que diz respeito recomendo estes sites http://www.sit.com.br/SeparataTELCO50.htm e http://idgnow.terra.com.br/idgnow/telecom/2002/09/0043 .

- **Diferença entre MIDP 1.0 e 2.0**

Existem algumas diferenças entre as versões, como todos sabemos a cada versão mais nova é sempre incrementado mais funções. Além desses métodos a mais, e algumas facilidades maiores da 2.0 como classes para jogos, e tratamento de sons, temos a diferença na comunicação de dados que é o mais interessante aqui como, por exemplo, as conexões são feitas através do protocolo http que no caso é inseguro no

envio de informações, já na 2.0 está implementado o https (ssl) conexão com criptografia que já é conhecida de todos.

Aqui listamos os Packages disponíveis para as versões.

- java.lang
- java.lang.ref (somente 1.1)
- java.io
- java.util
- javax.microedition.io
- javax.microedition.lcdui
- javax.microedition.lcdui.game (somente na 2.0)
- javax.microedition.media (somente na 2.0)
- javax.microedition.media.control (somente na 2.0)
- javax.microedition.midlet
- javax.microedition.rms
- javax.microedition.pki (somente na 2.0)

Opcionalmente, fabricantes podem fornecer API's JAVA para acesso a partes especificas de cada aparelho.

Com relação à segurança a MIDP 2.0 traz bastante recursos nessa parte, e já se aproxima mais do J2SE, com Permissions Types, Protection Domains baseados em IP's e PKI's.

- API's

A MIDP 1.0 não possui suporte a ponto flutuante, ou seja, se desejas trabalhar com contas, e valores reais terás que manuseá-los em código mesmo. A versão 1.1 do CLDC já possui suporte a ponto flutuante, porém mesmo com a versão 1.0 você pode baixar programas específicos pra isso como esse programa encontrado em http://home.rochester.rr.com/ohommes/MathFP .

Você tem ainda outras restrições como:

- Sem user classLoading
- Sem finalização de objetos (finallize() de java.lang.Object)
- Garbage collector existe, porém não executa método automático.
- Sem RMI e sem JINI
- Sem métodos nativos, a não ser os fornecidos pela JVM nativa.
- MultiThreading (sem interrupt(), pause(), resume() e stop())
- Sem thread groups e thread Deamons

- Criando projetos, Compilando e Executando

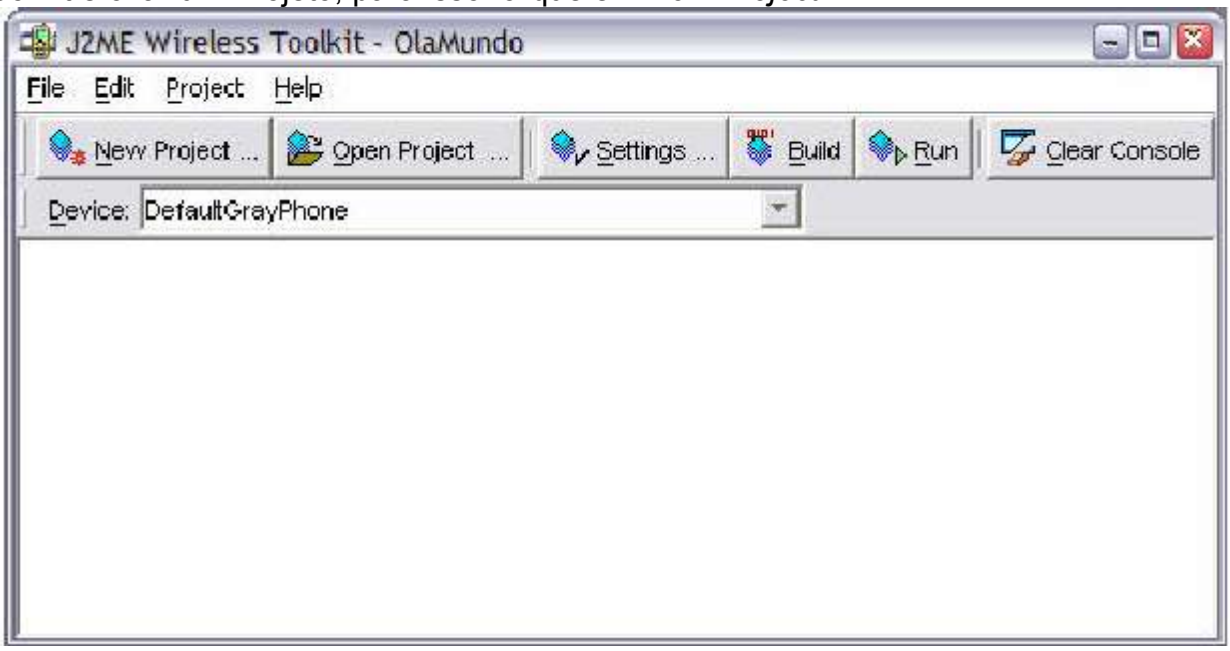
Bom depois de muita teoria, a qual nos deu um embasamento melhor para sabermos o que estamos fazendo, vamos a uma parte mais prática, e a mais esperada por todos.

O processo de desenvolvimento de MIDlet's é um pouco mais complexo do que de outros programas feitos em JAVA, pois é feito segundo estes passos:

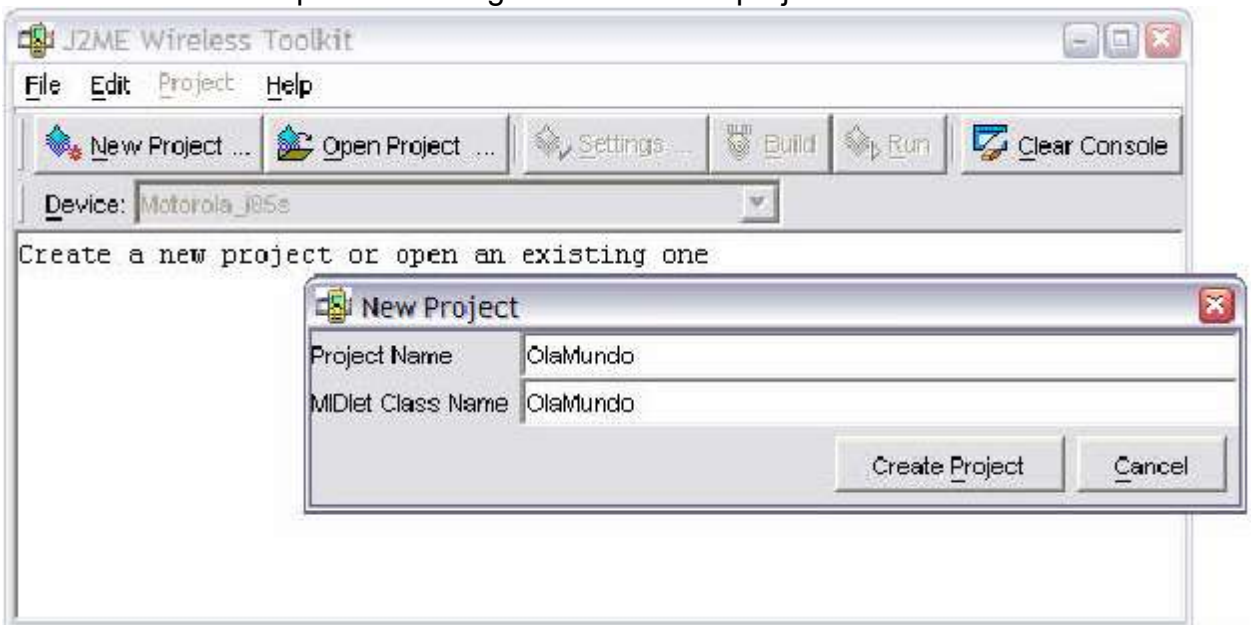
- Editar código fonte;
- Compilar;
- Pré-Verificar;

- Empacotar;
- Testar e/ou Instalar;

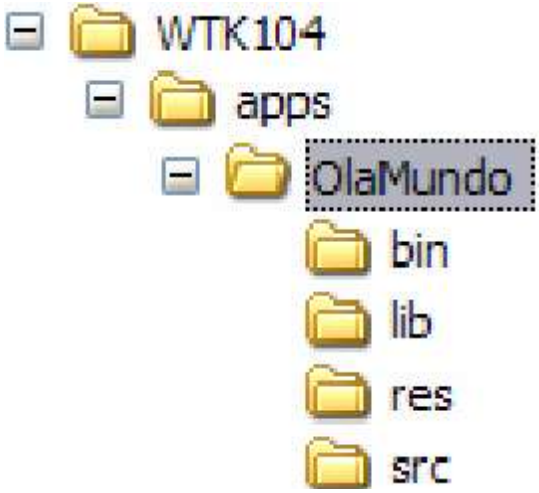
Depois de tudo instalado corretamente, devemos primeiramente abrir o Wireless ToolKit e criar um Projeto, para isso clique em *New Project*.



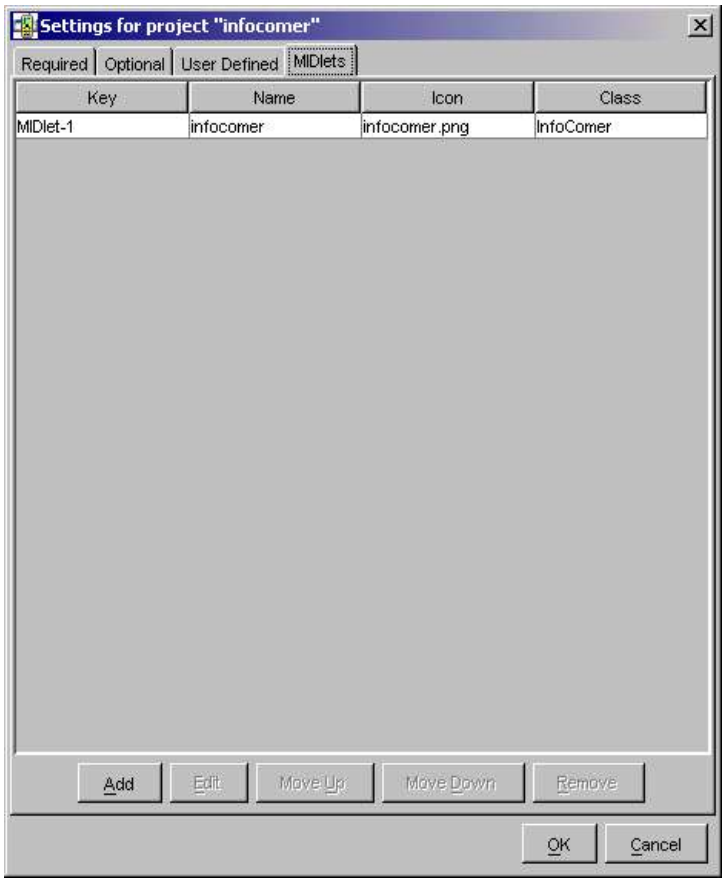
Depois em Project Name defina o nome do seu projeto, e logo abaixo o nome da classe de sua midlet que deve ser igual ao nome do projeto.



Feito isso ele terá criado vários diretórios dentro de path_WTK/apps/Sua_Aplicacao/, para cada aplicação é criado uma árvore de diretórios conforme ilustrado abaixo.



Em seguida abrirá uma tela que contém informações sobre a sua aplicação, aconselho a não mudar o que está lá, pois é por esse arquivo que o celular reconhecerá antes de ser baixado, se ele é apto a rodar ou não seu programa, a única coisa que poderias mudar seria o *Vendor Name*, o qual você pode colocar o nome de sua empresa, ou o que desejares. Se vocês verificarem a última aba desta tela (aba MIDlets), tem um link para uma figura .png, que é o icone que aparecerá no celular de sua aplicação você pode modificá-lo colocando o nome de uma figura personalizada, apenas lembre-se que a figura deve ficar no devido diretório (path_WTK/apps/Sua_Aplicacao/res), e sempre para qualquer aplicação deve-se por dentro do diretório /res.



Existem dois botões que serão muito utilizados por nos, BUILD, e RUN, os quais

são auto-explicativos.

Após a criação do projeto, estamos aptos, e finalmente, a criar nossos códigos.

Obfuscator

Como vimos dispositivos celulares possuem memória muito reduzida, e quanto menor sua aplicação for, melhor será para o celular. Para isso usamos o Obfuscator que é usado para diminuir o tamanho dos arquivos eliminando redundância e também para evitar engenharia reversa.

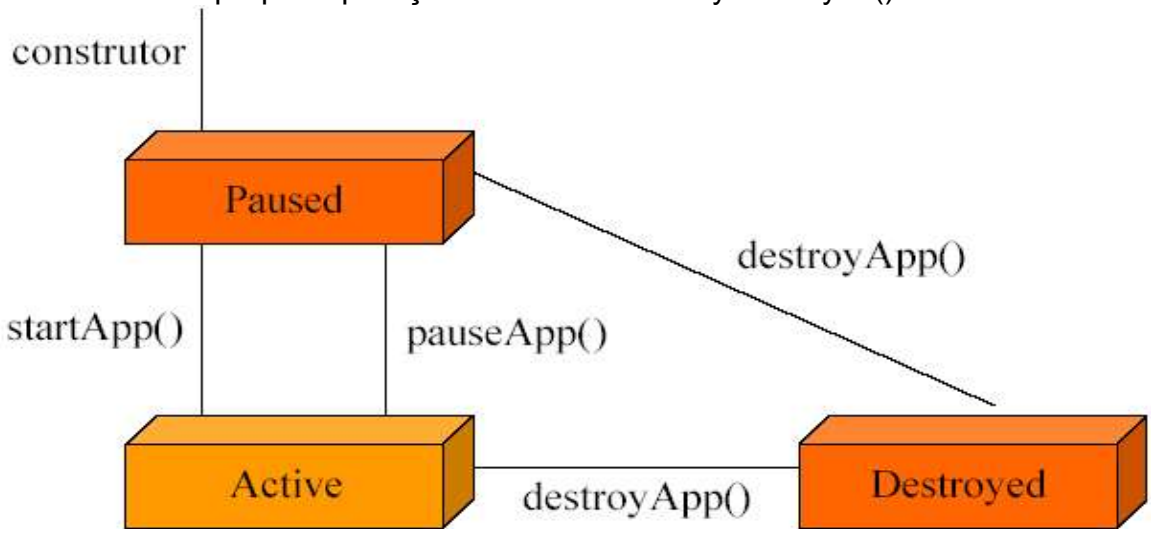
Este programa, renomeia classes, variáveis e métodos para nomes mais compactos, removem classes, métodos não usados e ainda inserem dados ilegais ou questionáveis para confundir Descompiladores.

Para criar um pacote obfuscado você precisa de algum programa Obfuscator, um bom programa, o Retroguard, pode ser encontrado em www.retrologic.com, depois de baixado você deve copiar o arquivo retroguard.jar dentro do diretório bin do wirelessToolkit. (Ex: c:\WTK104\bin)

- Ciclo de Vida de uma MIDlet

O Aplication Manager (AM) de cada dispositivo é quem vai controlar os aplicativos a serem instalados, onde e como serão armazenados e como serão executados. As classes de cada aplicativo estão em um arquivo JAR, o qual vem acompanhado de um descritor JAD, que terá todas as informações as quais já vimos anteriormente.

Assim que a MIDlet é invocada, o AM invoca o método startApp(), o qual coloca a midlet no estado Active. Enquanto ela estiver executando o AM pode pausar ela invocando o método pauseApp() no caso de uma chamada sendo recebida, ou SMS chegando. A aplicação pode pausar a si mesma, bastando invocar notifyPaused(). Assim como a AM pode pausar a aplicação e esta a si mesma, ocorre o mesmo com o DestroyApp() que é invocado pela AM para fechar a aplicação ou até mesmo pode ser fechada através da própria aplicação invocando o notifyDestroyed().



- Interface

As MIDlets devem poder ser executadas em qualquer dispositivo sem alterações, contendo a VM, porém isso torna-se bastante difícil na parte de Interface com usuário, pois dispositivos variam de tamanho de tela, cores, teclados, touch-Screens e outros aspectos.

As aplicações são desenvolvidas com uma certa abstração de tela, pois os comandos e insersão de dados são feitos através dos botões do celular, e isto não sabemos previamente. As aplicações descobrem isto em Runtime e se comportam de maneira apropriada a cada celular. Já no desenvolvimento de jogos a aplicação é bem mais específica, pois o desenvolvedor precisa conhecer o dispositivo previamente para melhor aproveitamento de recursos, como disposição em tela por exemplo.

A tela do dispositivo é representada por uma instância da classe Display, a qual é obtida pelo método getDisplay(), geralmente contida no método startApp(), pois o método getDisplay(), somente fica disponível após o inicio da Aplicação (startApp()). Nessa instancia de Display são inseridos heranças de Displayable.

```
public void startApp()
{
    Display d = Display.getDisplay(this);
    ...
}
```

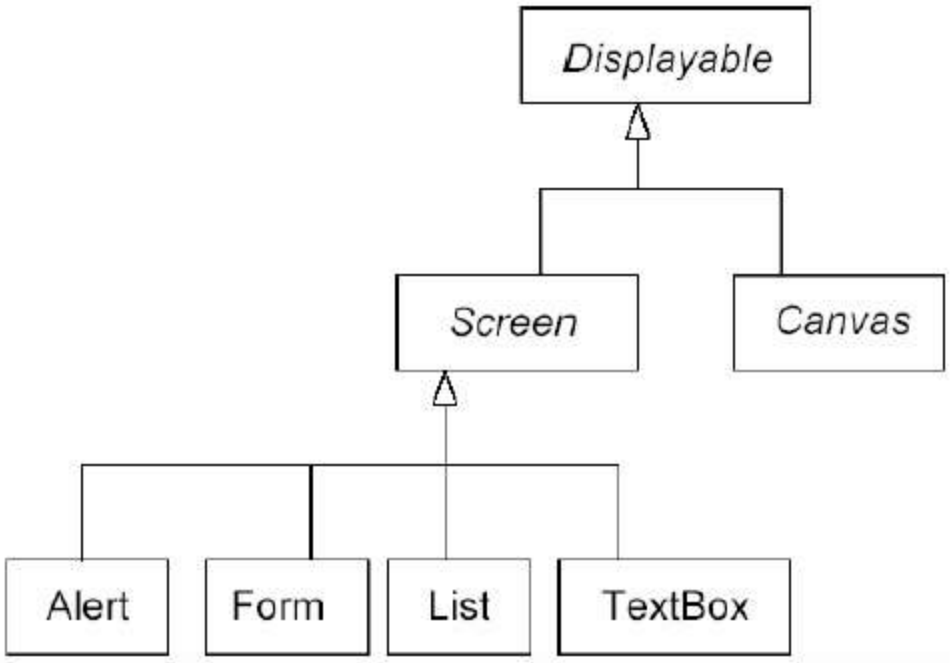
Na tela são mostrados componentes, objetos já instanciados, os quais são chamados pelo método setCurrent() o qual seta o objeto que será mostrado.

O ciclo básico de uma MIDlet pode ser definido por:

- Mostrar um Displayable;
- Esperar por Ação do Usuário;
- Decidir qual Displyable mostrar em seguida;
- Repetir;

As ações do usuário são gerenciadas por comandos (commands), os quais são adicionados a componentes visuais. E também temos as classes Screen e Canvas as quais podemos dividir em duas classes:

- High-level API's que engloba a classe Screen e suas heranças, pois são classificados como objetos de Interface.
- Low-level API's que engloba a classe Canvas e suas heranças, as quais proporcionam uma área livre para gráficos.



- Commands

Os commands com vimos, são usados para interação do usuário com a aplicação. Sua aplicação básica é na troca de Displayables, os quais são acionados por elementos reais do dispositivo, que podem ser: softkeys, itens de menu, botões e dispositivos de acionamento.

Esses commands são entidades abstratas e podem ser adicionados arbitrariamente para cada elemento Screen ou Canvas.

Os commands tem três propriedades:

- Label: texto mostrado ao usuário na interface;
- Type: tipo do comando;
- Priority: prioridade dos comandos que são utilizados para mostrar os comandos mais utilizados com maior facilidade.

```
Command(String Label, int Type, int Priority);
Ex:
ComandoSair = new Command("Sair", Command.EXIT, 0);
```

As ações dos commands são totalmente dependente do código, ou seja, você terá que implementar a ação de cada command selecionado, os quais ficarão dentro de um método chamado commandAction, que vem como herança quando a MIDlet herda CommandListener. Existem os métodos addCommand(Command) e removeCommand (Command), os quais são auto-explicativos. Os commands são geralmente relacionados a SoftKeys ou SoftButtons, que são botões existentes nos celulares os quais são usados para menus, e navegação em geral, sem uma função única. Geralmente possui-se dois botões de SoftKeys, porém isso não significa que você precisa só ter dois commands sendo implementados, pois se houver mais commands que botões eles são agrupados e é aberto um menu. Para isso que serve a prioridade setada nos botões, ou seja, a prioridade 0 (mais prioritário) indica que esse command deve ser mostrado nas softKeys, ou em primeiro lugar no menu.

O tratamento dos commands é feito com o conceito de Listeners, ou seja, comandos são adicionados a Displayables, cada displayable pode possuir um CommandListener o qual é invocado assim que um command é acionado.

```
public void setCommandListener(CommandListener)
```

Exemplo:

```
TextBox t1 = new TextBox("Texto", "", 10,TextField.ANY);
t1.setCommandListener(this);
t1.addCommand(ComandoSair); //comando feito acima
```

Método commandAction

```
public void commandAction(Command c, Displayable d)
{
     if (c==comando1)
     {
          ...
     }
}
```

- TextBox

É um componente básico que serve para entrada de texto(string) em celulares. Todos sabemos o quão entediante é a entrada de dados em celulares, e por isso temos que aprimorar essa técnica e manter o bom padrão de uma MIDlet, e de qualquer outro programa que é deixar o usuário feliz e menos cansado e entediado.

O construtor de um textBox é:

public TextBox(String titulo, String texto, int tam_Max, int constraints);

onde:

- Titulo é mostrado no topo da tela.
- Texto é mostrado dentro do textbox, como string inicial.
- tam_Max tamanho máximo que será alcançado pelo textBox.
- Constraints é são as constantes da classe TextField, as quais veremos logo abaixo.

Exemplos:

```
TextBox t1 = new TextBox("Nome: ", "", 30, TextField.ANY);
TextBox t2 = new TextBox("Nome: ", "", 20, TextField.ANY |
TextField.PASSWORD);
```

- Tickers

Ticker nada mais é que um texto que corre no topo da tela, semelhante as tickers de mercados de ações, e semelhante ao <marquee> de html. O recurso de Ticker serve para algum texto informativo que possa ser importante, ficar lembrando toda hora, ou algo de enfeite que você queira colocar. Mas lembre-se, a tela do celular já é pequena para sua aplicação, e o ticker tomara mais uma linha para si, a qual não poderá ser usada. Sua criação é simples.

```
     // Displayable d = ...
  Ticker ticker = new Ticker("Esta mensagem deve passar na tela");
     d.setTicker(ticker);
```

E os métodos para manuseio dessas Tickers não poderiam ser outros senão:

```
public void setTicker(Ticker ticker)
public Ticker getTicker
```

- Forms

Form é uma herança de Screen e Displayable, o qual pode conter um número arbitrário de controles de interface chamados *Itens.* Um form tem a mesma função de um Container em AWT, mas com as limitações da MIDlet é claro, ele serve para poder-se colocar mais de um componente na tela. Apesar de haver rolagem automática quando todos os objetos não cabem na tela, cuidado para não deixar um form muito grande com muitos componentes, pois a rolagem dos forms tende a ser confuso nos dispositivos, pois cada um implementa isso da sua maneira, e por isso não deve se tornar uma prática corriqueira, além de tornar seu aplicativo esteticamente mal feito, pois o usuário muitas vezes não irá saber se há mais objetos a serem preenchidos.

Para criarmos um form temos duas sobrecargas:

- ***public Form(String Titulo);***
- ***public Form(String Titulo, Item[] itens);***

Como vimos os forms recebem itens os quais são divididos em diversos tipos cada um com sua finalidade.

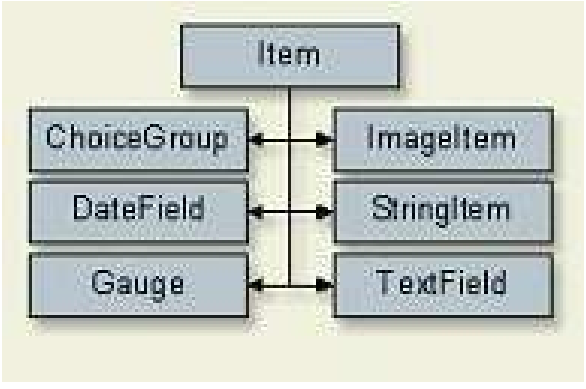
Os itens podem ser adicionados e removidos mesmo com os forms sendo mostrados. Os métodos de trabalho com os forms são:

- append(Item item);
- set(int Indice, Item item);
- insert(int Indice, Item item);
- delete(int Indice);

Lembre-se que o método *append* adiciona um novo item no final da lista, e o método *insert* adiciona um item no índice indicado movendo os outros existentes para baixo.

- Item

Como já foi falado existem vários tipos de itens, e cada um com sua determinada função. Esses diversos tipos de itens estendem da classe Item. Esses itens são inseridos em forms, eles podem ou não ter um Label, podem ou não ser mostrados, eles podem ter comandos associados a eles.



Vamos relata-los aqui.

A partir de agora vamos usar o mesmo form que criaremos agora, para todos os exemplos abaixo:

```
Form form = new Form("Form de teste");
```

- ***StringItem*** – nada mais é que um simples texto.
  Para este Item, existem os métodos de manipulação: getLabel(), setLabel(), getText(), setText();

  StringItem("Label: ","Text");

  Ex:
  ```
  StringItem strItem = new StringItem("Nome: ","Juliano");
  form.append(strItem);
  ```

- ***TextField*** – é uma string editável, ou melhor exemplificando um campo para preencher.
  Para este item existem os métodos de manipulação: getString(); setString();

  TextField(String Label, String Text, int TamMax, int Constraints)

  Ex:
  ```
  TextField tf = new TextField("Digite seu Nome: ", "", 30, TextField.ANY);
  form.append(tf);
  ```

  As constraints são as mesmas do TextBox (ANY, PASSWORD, EMAILADDR, NUMERIC, PHONENUMBER, URL).

- ***ImageItem*** – mostra a instância de uma Imagem dentro de um form, ou seja, você precisa já de um objeto Image. Este item possui um label, um texto alternativo e pode ser posicionado segundo um layout.

  ImageItem(String Label, Image imagem, int Layout, String texto_alt);

  Os layouts são constantes da classe, e podem ser: LAYOUT_DEFAULT, LAYOUT_LEFT, LAYOUT_CENTER, LAYOUT_RIGHT, *LAYOUT_NEWLINE_BEFORE*, LAYOUT_NEWLINE_AFTER**).
  **somente existentes no MIDP 2.0*

  Ex:
  ```
  Image img = new Image.createImage("/foto.png");
  ImageItem imgItem = new ImageItem("Foto", img, ImageItem.LAYOUT_DEFAULT, "Alt Text");
  form.append(imgItem);
  ```

- ***DateField*** – É uma interface usada para datas, horas ou ambos, como irá aparecer é por conta do Celular e o programador não tem controle sobre isso. No construtor deve ser especificado a Label e o tipo de dado, que pode ser:
  - DateField.DATE

- DateField.TIME
- DateField.DATE_TIME

public DateField(String Label, int modo);

Ex: `DateField data = new DateField("Data: ", DateField.DATE );`

- ***Gauges*** – É uma representação gráfica de um valor inteiro, e como ele será mostrado depende da implementação de cada aparelho. Existem dois tipos de gauges, o interativo, que permite a ação do usuário sobre seu valor, e Não interativo controlado somente pelo programa. Os valores podem ser alterados em tempo de execução, pelos comandos getMaxValue(), setMaxValue(); setValue(), getValue().

public Gauge(String Label, boolean Interativo, int ValorMaximo, int ValorInicial);

Ex: `Gauge g = new Gauge("Gauge Interativo", true, 20, 1);`

- ***ChoiceGroup*** – é uma lista de escolhas semelhante a List, ambos implementam a interface Choice. Os tipo de listas são Exclusive e Multiple, porém não tem o tipo Implicit. Já na MIDP 2.0 é implementado o tipo POPUP, que deixa a lista semelhante a um menu DropDown.

```
public ChoiceGroup(String label, int tipo)
public ChoiceGroup(String label, int tipo, String[] elementos,
Image[] imagens)
```

Eventos de Itens
Os Itens possuem eventos de alteração de Estado, que podem ser manejados para.
• Aplicação pode tratar eventos de alteração de estado em Itens
• Interface ItemStateListener
– método `public void itemStateChanged(Item item)`
• Registra-se o listener com o método:
`setItemStateListener(ItemStateListener listener)`

- Lists

Como o próprio nome diz são listas, as quais permitem ao usuário selecionar itens (elements) de uma lista. Esses elementos podem ser representados tanto por Strings quanto por Imagens.

Existem três tipos de listas, Implicit, Exclusive e Multiple. Implicit deixa você escolher uma opção e clicar no botão padrão de seleção ou em um command que você adicionar, Exclusive, que lembra uma lista de Radio, permite somente a seleção de uma opção e para seleciona-la deve-se usar um botão com um Command setado. E ainda tem a list Multiple, que funciona como uma CheckList já conhecida de todos, que nos permite selecionar várias opções.

Quando é selecionado um item em uma lista Implicit, o método CommandAction é chamado, sendo que para esta List já foi anteriormente adicionando um CommandListener.

Existem duas sobrecargas para este construtor, no qual você pode iniciar os Arrays de elementos e de Imagens ou não. O array de elementos, se utilizado, não poderá ser nulo, mas poderá conter elementos nulos, os quais aparecerão em branco na tela.

```
public List(String titulo, int tipo)
public List(String titulo, int tipo, String[] elementos, Image[]
imagens)
```

Quando uma lista tornar-se grande demais para a tela, será criado uma rolagem, a qual não cabe a nós implementa-la, ela é por conta da VM.

Cada elemento da list possui um índice. Esse índice inicia em 0. Para editar uma list, existem alguns métodos que valem a pena ser comentados aqui. Que são:

- **set:** altera o elemento do qual o índice foi fornecido;
- **insert:** insere um elemento em uma posição indicada por um índice. Se este índice estiver no meio da lista, ele força os outros elementos uma posição à frente;
- **Append:** inclui um elemento ao final da lista;
- **delete:** remove o elemento indicado pelo índice;

EX:

```
public void set(int indice, String elemento, Image imagem);
public void insert(int indice, String elemento, Image imagem);
public int append(String elemento, Image imagem);
public void delete(int indice);
```

Dois métodos bastante utilizados que valem a pena ser comentados são:

- ***public boolean isSelected(int indice):*** que me diz se o índice esta selecionado atualmente.
- ***public int getSelectedIndex():*** que me retorna o índice que foi escolhido, somente para listas Implicit e Exclusive.
-

- Imagens

A especificação de MIDP determina que a implementação deverá ser capaz de manipular Imagens PNG, o que não ocorre em todos os celulares. Alguns trabalham com JPG, outros com GIF, BMP, e até mesmo PNG sem compactação, porém é muito raro você conseguir encontrar a especificação de cada aparelho para isso, o que torna muito difícil a programação com imagens, quando não é usado o padrão.

```
public static Image createImage(String nome)
public static Image createImage(byte[] dados, int offset, int tamanho)
```

O primeiro construtor busca o arquivo indicado pelo nome na pasta /res, de sua aplicação no caso de uso do emulador, quando ele é compilado e empacotado, a imagem é empacotada junto com o programa. O segundo construtor obtém a imagem de um Arrray de Bytes, partindo da posição offset, até o tamanho indicado. Essas imagens podem ser Mutáveis ou Imutáveis. As mutáveis podem ser modificadas pelo Objeto Graphics, obtido pelo método getGraphics(). Porém, esses dois construtores citados acima geram imagens imutáveis. Para se criar imagens mutáveis deve-se usar o seguinte método.

```
public static Image createImage(int largura, int altura)
```

Lembre-se que as imagens passadas para Alerts, ChoiceGroups, ImgeItems, ou Lists devem ser Imutáveis. Imagens Imutáveis podem ser criadas a partir de outras imagens Mutáveis.

```
public static Image createImage(Image imagem)
```

Somente um comentário à parte. Imagens em MIDP 2.0 podem ser criadas à partir de porções de outras imagens já existentes.

```
public static Image createImage(Image image, int x, int y, int
largura, int altura, int transformacao)
```

- Alerts

Um alert nada mais é que uma mensagem informativa ao usuário, tem a mesma idéia de um alert de javascript, ou VB, ou qualquer outra linguagem, ou seja, ele é basicamente uma "telinha" que mostra uma mensagem e logo depois sai da tela.

Esses alerts podem ser tanto Timed Alerts, ou Modal Alert. No Timed Alert, você pode setar um tempo ou não o qual receberá o tempo padrão do aparelho. E o alerta modal que aparece e fica esperando uma intervenção do usuário, e possui vários tipos, como Alarm, Confirmation, Error, Info, Warning, cada qual com seu ícone e som, os quais também podem ser setados pelo desenvolvedor.

```
public Alert()
public Alert(String titulo, String texto, Image icone, AlertType
tipo)
```

**Qualquer um ou todos os parâmetros do segundo construtor podem ser nulos.*

Alerts são criados com um tempo default para desaparecer, tempo qual é implementado na VM, e pode ser obtido por getDefaultTimeout(), e também configurado por setTimeout(). Quando criamos um Timed Alert, podemos transforma-lo em modal adicionando ao parâmetro AlertType para Alert.FOREVER.

Ex:

```
Alert alerta;
alerta = new Alert("Alerta","Acesso não autorizado.", null, null);
alerta.setTimeout(5000);// timeout para 5 segundos
```

O comportamento padrão de um Alert é mostar a próxima tela setada, se usado o método *setCurrent(atual, próximo),* ou ele irá voltar para a tela atual se for usado, setCurrent(Alert).

- Conexões

Sistemas móveis geralmente oferecem acesso a dados via conexões OTA(Over The Air), porém essas conexões ainda são lentas, pouco confiáveis e intermitentes, o que nos dificulta um pouco essa transmissão de dados, e na pior das hipóteses entedia o usuário, este por estar já acostumado com uma conexão rápida e confiável.

A MIDP possui mecanismos genéricos para acesso a rede, as quais são definidas pela CLDC que é bem flexível. Para isso ela usa o Package javax.microedition.io e é

baseada na interface Connection. O link real entre a interface Connection é feito através da classe javax.microedition.io.Connector, que funciona passando-se uma string de conexão para um método estático de Connector e recebe uma Connection com resultado.

A conexão na MIDP 1.0 é feita por HTTP, através de HttpConnection, porém algumas implementações particulares de celulares podem oferecer HttpsConnection, SocketConnection entre outros. Já na MIPD 2.0 a obrigatoriedade á passada para a implementação do HttpsConnection. Ambos os métodos tratam das particularidades do protocolo Http como Headers, métodos e outros(RFC 2616).

O Http trabalha com o processo de response/request incluindo Headers no processo. Os parâmetros passados na url são codificados antes da transmissão o que garante integridade da informação e padronização, para isso utiliza-se de URLEncoder, que transforma espaços em sinais de +, a...z, A...Z, 0...9, ponto(.), Hífen(-), Asterisco(*) e Underscore(_) são mantidos, e todo qualquer outro caractere é convertido em %xy, onde xy é o valor Hexadecimal que representa os 8 bits menos significativos do caractere.

Revisando um pouco Http, temos os métodos GET, POST e HEAD, onde GET e POST buscam paginas, GET enviando parâmetros na URL, enquanto POST envia os parâmetro no header, e o HEAD que é idêntico ao GET, porém o servidor só responde com Headers.

A conexão GET funciona, passando-se uma URL para o método open() da classe connector, da qual você irá receber o resultado como um HttpConnection ou um InputConnection (trataremos disso depois), então você deverá obter os streams de conexão InputStream e OutputStream, e então ler os dados que foram retornados para você do Servidor e tratá-los conforme o que você precisa, e para isso você deve antecipadamente conhecer o formato de retorno da informação vinda do servidor.

Ex: `String url = "servidor.com.br/suaPagina.jsp";` *(aqui pode ser qualquer pagina de qualquer extensão, inclusive Servlets que são geralmente utilizados como resposta para MIDlets)*

```
InputConnection ic = (InputConnection) Connector.open(url);
InputStream is = ic.openInputStream();
```

//aqui vai todo o processo de leitura (Veja nos Exemplos), que será guardado em um array de bytes dependendo do tipo de dado que você espera do servidor, não esqueça que se você espera um dado que não seja puro texto depois terá que converte-lo para seu estado original, e caso você esteja esperando Strings como resposta, use StringBuffer.

```
ic.close();
```

Sempre que você estabelece uma conexão Http, é aconselhável que verifique o retorno dos Headers para saber se o retorno está ok, através de HttpConnection.HTTP_OK.

Ex:

```
int status = -1;
status = con.getResponseCode();
if (status == HttpConnection.HTTP_OK)
{
     // processa resultado
}
```

Dicas:

- Sempre que for fazer uma conexão tenha uma thread separada para isso.

- Acesso sempre toma tempo e pode ser demorado.
- Mantenha algum tipo de indicação de progresso para a interface com o usuário.

• Use GET ao invés de POST
- Mais simples e sem a preocupação com request headers
• Não use URLs hard-coded
- Mantenha em application descriptors, o que permite alterar a URL sem recompilação
• Tenha certeza de que as exceções são tratadas adequadamente

– Redes wireless não são muito confiáveis, prepare-se para o pior.
– Capture todas as exceções e proceda de forma razoável, com mensagens e indicações.

• Libere memória e recursos assim que possível

– Em dispositivos portáteis, tais recursos são muito escassos

- SMS via servidor

O Suporte a SMS só é dado usando-se MIDP 2.0 o qual ainda não está em vigor em nenhum celular no Brasil, ou ao menos a grande maioria usa MIDP1.0 (até esta publicação). Sendo assim, será muito difícil você conseguir desenvolver alguma aplicação com relação a isso, sendo que com a mipd1.0 já em vigor esta sendo bem difícil de se encontrar material e é uma tecnologia que a comunidade esta começando a interessar-se agora.

Bom em alguns fóruns já houve algumas perguntas sobre esse tópico e por isso vou comentar aqui algumas coisas. Bom, alguém pediu como fazer para mandar sms de servidores remotos, o que é necessário é um contrato com uma operadora a qual lhe fornecerá passagem livre do seu IP pela rede dela, lhe permitindo o envio dessas mensagens nada tem o que ver com a tecnologia J2ME. E obviamente não há a necessidade de se construir uma aplicação pra isso sendo que os celulares já possuem isso nativo.

- Armazenamento em celulares (RecordStores)

A persistência em celulares é tratada com RecordStores que são conjuntos de registros e instâncias de ***javax.microedition.rms.RecordStore.***

Cada recordStore é identificado por um nome que deve ser único, sua criação é feita de forma simples e bem sugestiva, da seguinte forma:

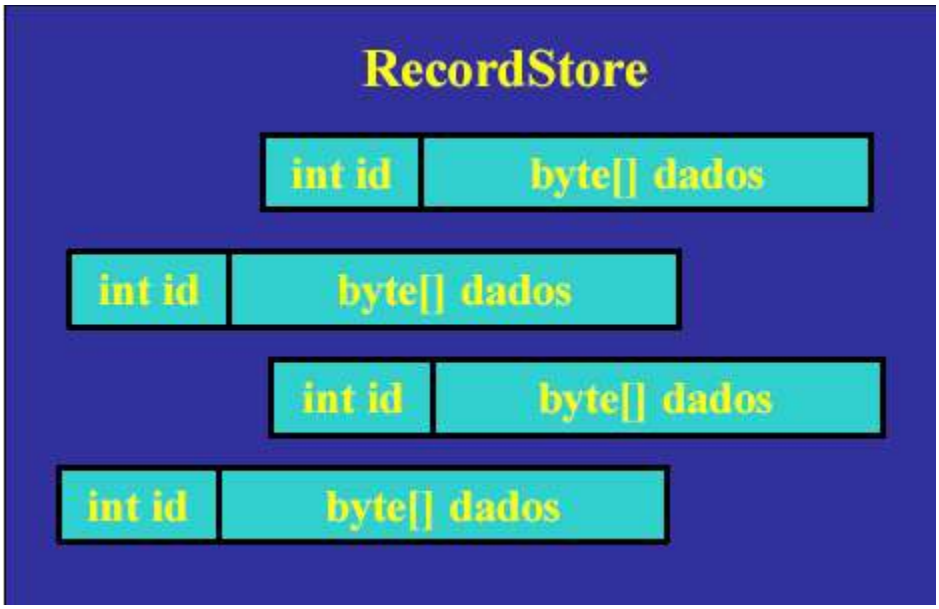
```
RecordStore.openRecordStore("NomeQualquer", true);
```

O parâmetro *boolean* da chamada do método indica que se *true* o RecordStore será aberta e se não existir será criada. O *false* apenas abre uma RecordStore existente. Lembre-se também que sempre que for manusear um RecordStore você precisa usar ***try{} catch{},*** pois estas operações geram exceções como: RecordStoreException, RecordStoreFullException, RecordStoreNotFoundException.

Assim como abrimos os recordStores temos que fecha-los, e sempre que abrir um recordStore, e assim que ele não for mais necessário, fecha-lo, pois lembremos que trabalhamos com uma memória bem reduzida, não esqueça que o método pauseApp, também deve fechar os recordStores. O método para fechar o recordStore é nada mais que:

```
ObjetoRecordStore.closeRecordStore();
```

A estrutura de Armazenamento do um RecordStore é bem simplória, possui apenas um id e um array de Bytes como local para armazenagem de dados. Se você está acostumado com banco de dados, e com uma estrutura de linhas e colunas, esqueça! Se quiseres guardar mais que uma informação no array de Bytes, terás que trabalhar com este array, colocando algum caractere de controle para seu armazenamento (Ex: Nome | Telefone). Somente como exemplo, nós trabalhamos com uma aplicação onde era preciso guardar um nome, e uma fotografia, e então colocávamos o nome +"carc_de_Controle"+Foto[ ], e então na hora de lista essas fotos tínhamos que trabalhar com cada registro percorrendo o array inteiro para primeiro listar o nome e depois a foto, a qual tinhamos que pegar o array e então criar um Image, e então colocávamos em um ImageItem.



Cada registro pode ser lido ou modificado, segundo seu ID, que é retornado a cada inserção.

```
public byte[] getRecord(int recordID)
```

Nos retorna um array de bytes para podermos trabalhar com ele.

Para deletar records, usamos o método:

```
public void deleteRecord(int ID)
```

Quando deletamos um record, o seu id, continua existindo, não sendo reorganizada a lista, ou seja, quando você quiser listar um RecordStore deve-se testar se ainda existe aquele record, por comparação de Arrays, através do método, `getRecordSize()`.

Os records podem ser "listados" a partir de RecordEnumeration. Na Classe Recorstore temos um método que lista todos os records de determinada recordStore, retornando um `RecordEnumeration`. Este método é o:

```
RecordStore.enumerateRecords();
```

Isso serve para que seja possível percorrer todos os elementos do recordStore a

partir dos metodos da classe RecordEnumeration, nextRecord(), previousRecord().

Para controlar o laço de listagem desses records, usa-se o metodo `hasNextElement() do RecordEnumeration.`

Importante saber que quando um record é excluído, os índices(IDs) dos elementos permanecem como estão, não há reorganização dos IDs. Apenas é possível reorganizar o Enumeration desse RecordStore.

Este método pede parâmetros de comparação que não abordaremos agora, porém o método pode ser usado como descrito abaixo:

```
recEnum=rsExcluir.enumerateRecords(null,null,false);
```

Ou seja, retorna todos os elementos e o "false" informa que esse Enumeration não será reorganizado automaticamente, caso haja alguma alteração na RecordStore. Para eliminar um Enumeration usa-se o método `destroy()`.

- Canvas

Como já vimos, Canvas é uma classe extendida de Displayable, que nos permite trabalhar livremente com gráficos e figuras em mais baixo nível.

A partir da classe Graphics é que podemos desenhar figuras primitivas, imagens e textos em qualquer lugar dentro do espaço de resolução da tela do dispositivo.

Também é a partir do Canvas que podemos desenvolver animações como jogos e apresentações, usando-se recursos de Threads (processamento concorrente), porém não trataremos disso nesse artigo.

- Codificando

A estrutura básica de uma MIDlet dá-se por:

```
/********** inicio do código **********/
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;

public class HelloWorld extends MIDlet implements CommandListener
{

public HelloWorld ()  { } //metodo construtor da classe;

public void startApp()  { } //método chamado sempre ao início de uma aplicação

public void pauseApp() { } //método chamado quando a aplicação é interrompida, por exemplo, quando chega uma ligação, ou uma mensagem sms.

public void destroyApp(boolean condicional) { }  //método chamado
```

```
quando uma aplicação será fechada, ela fica aguardando o Garbage
Collector

public void commandAction(Command c, Displayable d)
{
      //método onde será implementado quase toda aplicação, e os
comandos chamados pelas softKeys
}

}
      /****************** Fim do Código **********************/
```

Com isso podemos ver que a estrutura de uma aplicação é simples, porém, por ser assim tão simples às vezes restringe a implementação.

Códigos Exemplos

Obs: Comentários já feitos não serão repetidos;

ExemploCommandsAndTicker

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

public class ExemploCommandsAndTicker extends MIDlet implements
CommandListener
{

   Display tela;
   TextBox texto;
   Ticker ticker;
   Command sair, opcao1, opcao2, opcao3, opcao4, opcao5;


public ExemploCommandsAndTicker()
{

// instanciar TextBox
this.texto = new TextBox("Commands", "Aplicação iniciada", 20,
TextField.ANY);

this.ticker = new Ticker("Exemplo de ticker que deve rodar em cima da
tela");
//seta o ticker na tela
this.texto.setTicker(this.ticker);

// comandos do TextBox
this.sair = new Command("Sair", Command.EXIT, 0);
this.opcao1 = new Command("Opção 1", Command.SCREEN, 1);
this.opcao2 = new Command("Opção 2", Command.SCREEN, 2);
this.opcao3 = new Command("Opção 3", Command.SCREEN, 3);
this.opcao4 = new Command("Opção 4", Command.SCREEN, 4);
this.opcao5 = new Command("Opção 5", Command.SCREEN, 5);

// relacionar Commands com TextBox
this.texto.addCommand(sair);
this.texto.addCommand(opcao1);
this.texto.addCommand(opcao2);
this.texto.addCommand(opcao3);
this.texto.addCommand(opcao4);
this.texto.addCommand(opcao5);

// registrar TextBox com o CommandListener
this.texto.setCommandListener(this);
}

public void startApp()
{
```

```
   // obter tela do dispositivo
   this.tela = Display.getDisplay(this);

   // setar Displayable corrente para a tela
   this.tela.setCurrent(this.texto);
}

public void pauseApp()
{

}

public void destroyApp(boolean i)
{

}

//aqui precisamos de um command e um displayable
//o displyable será o componinte ativo na tela
//porque ele extende de Display
public void commandAction(Command c, Displayable d)
{
   if (c == this.sair)
   {
       // sair da aplicação
       this.destroyApp(true);
       this.notifyDestroyed();
   }

   if (c == this.opcao1)
   {
       // alterar texto do TextBox
       this.texto.setString("Opção 1 selecionada.");
       this.ticker.setString("Opção 1 Selecionada");
   }

   //...
   //pode-se implementar as outras opcoes aqui
   //use a imaginacao
   //ou espere aprender mais algumas coisas e entao acrescente
}

}
```

ExemploListAndAlert

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

public class ExemploListAndAlert extends MIDlet implements
CommandListener
{
    Display tela;
    TextBox t1;
    Command sair, prox, prox1;
    Alert alarme, confirmacao, erro, info, aviso;
    List menu, exclusiva, multipla;


public ExemploListAndAlert()
{
//criamos os arrays de opcoes que irao aparecer nas lists
String[] menuElementos = { "Alarme", "Confirmação", "Erro", "Info",
"Aviso", "Exclusiva", "Multipla" };
String[] exclusivaElementos = { "Opcao1", "Opcao2", "Opcao3", "Opcao4",
"Opcao5", "Opcao6" };
String[] multiplaElementos = { "Opcao1", "Opcao2", "Opcao3", "Opcao4",
"Opcao5", "Opcao6" };


    //instancia todos os alerts
    //vamos deixar todos eles com null no campo de icone
    this.alarme = new Alert("Alarme","Alerta de
Alarme",null,AlertType.ALARM);

    //setamos o tempo que ele ficará na tela antes de sumir
    this.alarme.setTimeout(5000);
    this.confirmacao = new Alert("Confirmacao","Alerta de
confirmacao",null,AlertType.CONFIRMATION);
    this.confirmacao.setTimeout(4000);
    this.erro = new Alert("Erro","Alerta de Erro",null,AlertType.ERROR);

    //fizemos deste alert, um alert modal
    //que esperará uma intervencao do usuario
    this.erro.setTimeout(Alert.FOREVER);
    this.info = new Alert("Info","Alerta de Info",null,AlertType.INFO);
this.aviso = new Alert("Aviso","Alerta de Aviso",null,AlertType.WARNING);


    //instancia os comandos
    this.prox = new Command("Prox",Command.SCREEN,1);
    this.prox1 = new Command("Prox", Command.SCREEN,1);
    this.sair = new Command("Sair",Command.EXIT,0);

    //instancia os textbox
    this.t1 = new TextBox("Result","",200,TextField.ANY);

    //relacionar comandos as textboxes
    this.t1.addCommand(sair);
```

```
    this.t1.setCommandListener(this);


    //instanciar lists
    //criando a lista exclusiva com o array de elementos criado acima
    this.exclusiva = new List("Exclusiva", Choice.EXCLUSIVE,
exclusivaElementos, null);
    this.exclusiva.addCommand(this.sair);
    this.exclusiva.addCommand(this.prox);

    //criando a lista multipla com o array de elementos criado acima
    this.multipla = new List("Multipla", Choice.MULTIPLE,
multiplaElementos, null);
    this.multipla.addCommand(this.sair);
    this.multipla.addCommand(this.prox1);

    //criando a lista implicita com o array de elementos criado acima
    //esta lista sera noss menu principal
    this.menu = new List("Menu",Choice.IMPLICIT,menuElementos,null);
    this.menu.addCommand(this.sair);
    this.menu.setCommandListener(this);
}

public void startApp()
{
    this.tela = Display.getDisplay(this);
    this.tela.setCurrent(this.menu);
}

public void destroyApp(boolean i)
{
}

public void pauseApp()
{
}

public void commandAction(Command c, Displayable d)
{
       String opcao = "";

       if (c == this.sair)
       {
           this.destroyApp(true);
           this.notifyDestroyed();
       }

       if (c == this.prox)
       {
           //pega a String do item selecionado para por no textbox
           opcao = this.exclusiva.getString
(this.exclusiva.getSelectedIndex());
           this.t1.setString(opcao);
           this.tela.setCurrent(this.t1);
       }
```

```
        if (c == this.prox1)
        {
            //usamos esse laço para verificar quais opções
            //foram selecionadas para entao apresentalas
            for(int count=0; count<6; count++)
            {
                if (this.multipla.isSelected(count))
                {
                     //adicionamos as opcoes em uma string só
                     opcao = opcao+this.multipla.getString(count)+"\n";
                }
            }
            this.t1.setString(opcao);
            this.tela.setCurrent(this.t1);
        }

        //aqui comparamos se o comando veio de uma lista
        //e tambem se a lista era a lista menu
        if ((c == List.SELECT_COMMAND) && (d == this.menu))
        {
            int selecionado = this.menu.getSelectedIndex();
            //aqui verificamos se foi selecionado um dos Alerts
            //entao ele eh acionado, e apos ser mostrado
            //a tela anterior que é o menu
            //eh mostrado novamente
            switch (selecionado) {
                case 0:
                     this.tela.setCurrent(this.alarme);
                     break;
                case 1:
                     this.tela.setCurrent(this.confirmacao);
                     break;
                case 2:
                     this.tela.setCurrent(this.erro);
                     break;
                case 3:
                     this.tela.setCurrent(this.info);
                     break;
                case 4:
                     this.tela.setCurrent(this.aviso);
                     break;
               //aqui temos as outras listas...
               case 5:
                    this.exclusiva.setCommandListener(this);
                    this.tela.setCurrent(this.exclusiva);
                    break;
               case 6:
                    this.multipla.setCommandListener(this);
                    this.tela.setCurrent(this.multipla);
                    break;
            }
        }
}

}
```

ExemploTextBox

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

public class ExemploTextBox extends MIDlet implements CommandListener
{

       TextBox nome, mail, resultado, senha;
       Command sair, proximo, login, fim;
       Display tela;

public ExemploTextBox()
{

       //construtores dos atributos defindos acima
       this.nome = new TextBox("Nome","", 20, TextField.ANY);
       this.senha = new TextBox("Senha","", 20, TextField.PASSWORD);
       this.mail = new TextBox("Mail", "", 20, TextField.EMAILADDR);
       this.resultado = new TextBox("Resultado", "", 30, TextField.ANY);

       //construtores dos commands que irao ser adicionados aos objetos
       this.sair = new Command("Sair", Command.EXIT, 0);
       this.proximo = new Command("Proximo", Command.SCREEN, 1);
       this.login = new Command("Login", Command.SCREEN, 1);
       this.fim = new Command("Fim", Command.SCREEN, 1);

       //adicao dos commands aos aos objetos, e são setados como
commandListener
       this.nome.addCommand(this.sair);
       this.nome.addCommand(this.proximo);
       this.nome.setCommandListener(this);

       this.senha.addCommand(this.sair);
       this.senha.addCommand(this.login);
       this.senha.setCommandListener(this);

       this.mail.addCommand(this.sair);
       this.mail.addCommand(this.fim);
       this.mail.setCommandListener(this);

       this.resultado.addCommand(this.sair);
       this.resultado.setCommandListener(this);
}

//método executado ao inicio de Execuçao da MIDlet
public void startApp()
{
    this.tela = Display.getDisplay(this);
    this.tela.setCurrent(this.nome);

}
//método executado se a MIDlet foir pausada por ela mesma
//ou por um evento externo
public void pauseApp()
```

```
{

}

//método chamado quando a aplicação for encerrada,
//não há necessidade de ser implementada
public void destroyApp(boolean condicional)
{

}

//método chamado assim que um command for adicionado.
public void commandAction(Command c, Displayable d)
{
    //definiçao do que será feito quando um command for acionado
    if (c == this.sair)
    {
        this.notifyDestroyed(); //notifica que será destruido
        this.destroyApp(true); //chama o destrutor
    }

    if (c == this.proximo)
    {
        this.tela.setCurrent(this.senha); //seta como ativo na tela
    }

    if (c == this.login)
    {
        this.tela.setCurrent(this.mail);
    }

    if (c == this.fim)
    {
        //seta a String do objeto resultado pegando o que foi inserido
        //nos campos anteriores.
        this.resultado.setString(this.nome.getString()+"
"+this.mail.getString());
        this.tela.setCurrent(this.resultado);
    }

}
}
```

ExemploForm1

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

public class ExemploForm1 extends MIDlet implements CommandListener
{

	Display tela;
	Form login, resultado;
	TextField nome, senha;
	StringItem resultadoMsg;
	Command sair, proximo;

public ExemploForm1()
{
	// inicializar Commands
	this.sair = new Command("Sair", Command.EXIT, 0);
	this.proximo = new Command("Prox", Command.SCREEN, 1);

	// form de login
	this.login = new Form("Login");
	this.nome = new TextField("Nome:", "", 20, TextField.ANY);
	this.senha = new TextField("Senha:", "", 20, TextField.ANY |
TextField.PASSWORD);

	//adiciona-se os componentes ao Form Login
	this.login.append(this.nome);
	this.login.append(this.senha);
	this.login.addCommand(this.sair);
	this.login.addCommand(this.proximo);
	this.login.setCommandListener(this);

	// form de resultado
	this.resultado = new Form("Resultado");
	this.resultadoMsg = new StringItem("","");

	//adiciona-se o componente ao Form Resultado
	this.resultado.append(this.resultadoMsg);
	this.resultado.addCommand(this.sair);
	this.resultado.setCommandListener(this);

}

public void startApp()
{
	this.tela = Display.getDisplay(this);
	this.tela.setCurrent(this.login);
}

public void pauseApp()
{

}
```

```
public void destroyApp(boolean condicional)
{

}

public void commandAction(Command c, Displayable d)
{
    if (c == this.sair) {
        this.destroyApp(true);
        this.notifyDestroyed();
    }
    if (c == this.proximo) {
        //O Label sempre aparecerá antes do Text não importando
        //a ordem que vc adicione ele ao componente
        //faça o teste trocando de ordem as chamdas.
        this.resultadoMsg.setLabel(this.nome.getString()+" ");
        this.resultadoMsg.setText(this.senha.getString());
        this.tela.setCurrent(this.resultado);
    }

}

}
```

ExemploGauge

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

//neste exemplo vamos implementar mais duas interfaces
//o ItemStateListener vai nos ajudar a reconhecer as Setas direcionais
//do teclado do dispositivo.
//e a interface Runnable desempenha o papel de uma Thread
//onde poderiamos ter usado uma classe extra extendendo de Thread
//no entanto preferimos simplificar a implementação.
//O processamente paralelo será usado afim de mostrar o
//movimento do gauge nao interativo.
//usamos o recurso de Thread por que não haveria possibilidade
//de incrementar nosso gauge, e mostra-lo na tela
//na mesma linha de execuçao.

public class ExemploGauge extends MIDlet implements CommandListener,
ItemStateListener, Runnable
{

        Display tela;
        List menu;
        Form gauge1, gauge2;
        Gauge inter, ninter;
        Command sairCommand, menuCommand;
        Thread minhaThread;

public ExemploGauge()
{
        // inicializar Commands
        this.sairCommand = new Command("Sair", Command.EXIT, 0);
        this.menuCommand = new Command("Menu", Command.SCREEN, 1);

        // menu principal
        String[] menuElementos = {"Interativo", "Não Interativo"};
        this.menu = new List("Menu Principal", List.IMPLICIT,
menuElementos, null);
        this.menu.addCommand(this.sairCommand);
        this.menu.setCommandListener(this);

        //declararemos um for para cada gauge.
        // gauge interativo
        this.gauge1 = new Form("Interativo");
        //setamos o label inicial do gauge
        //true para torna-lo interativo
        //50 é o numero de toques qeu ele necessitará para ser completado
        //0 para indicar seu inicio.
        this.inter = new Gauge("0 %", true, 50, 0);
        this.gauge1.append(this.inter);
        this.gauge1.addCommand(this.sairCommand);
        this.gauge1.addCommand(this.menuCommand);
        this.gauge1.setCommandListener(this);
        //preparamos este form para "ouvir" os direcionais do teclado.
        this.gauge1.setItemStateListener(this);
```

```
        // gauge nao-interativo
        this.gauge2 = new Form("Não Interativo");
        //mesma coisa que o gauge anterior,
        //somente do false indica qeu este gauge esta somente
        //sobre controle do programa
        this.ninter = new Gauge("0%", false, 100, 0);
        this.gauge2.append(this.ninter);
        this.gauge2.addCommand(this.sairCommand);
        this.gauge2.addCommand(this.menuCommand);
        this.gauge2.setCommandListener(this);

}

public void startApp()
{
        this.tela = Display.getDisplay(this);
        this.tela.setCurrent(this.menu);
}

public void pauseApp()
{

}

public void destroyApp(boolean condicional)
{

}

public void commandAction(Command c, Displayable d)
{
    if (c == this.sairCommand)
    {
        this.destroyApp(true);
        this.notifyDestroyed();
    }
    if (c == this.menuCommand)
    {
        this.tela.setCurrent(this.menu);
    }

    //verifica qual item da lista do menu principal foi selecionado
    if (c == List.SELECT_COMMAND && d == this.menu)
    {
        //se for o primeiro item que é o gauge interativo
        //ele apresenta o gauge e espera por comandos.
        if (this.menu.getSelectedIndex() == 0)
        {
            this.tela.setCurrent(this.gauge1);
        }

 //se for o gauge nao interativo, ele inicial nossa "thread"
//a qual controlará nosso gauge fazendo-o aumentar até onde determinarmos
        if (this.menu.getSelectedIndex() == 1)
        {
```

```
            this.tela.setCurrent(this.gauge2);
            this.minhaThread = new Thread(this);
            minhaThread.start();
        }
    }
}

//controle do gauge interativo conforme o valor vai mudando ele
//vai aprensentando na tela o valor correspondente
//e como setamos ele para que fosse somente ateh 50
//multiplicamos por 2 os valores para supostamente
//chegar a 100%
public void itemStateChanged(Item i)
{
       if (i == this.inter)
       {
           this.inter.setLabel("    "+(this.inter.getValue()*2)+"%");
       }
}

//controle do gauge Nao interativo
//execução da nossa Thread (Runnable)
public void run()
{
      for (int cont = 1; cont <= 100; cont++)
      {
     // usamos este try-catch porque precisamos usar o sleep
     //para dar uma pausa pequena para melhor vizualizacao
     //do incremento.
     //por curiosidade, auemnte e dimunua o valor do sleep
     //lembre-se o valor é dado em milisegundos.
            try
            {
                Thread.sleep(30);
            } catch (Exception e)
            {
            }
            this.ninter.setValue(cont);
            this.ninter.setLabel("   "+cont+"%");
      }
}

}
```

ExemploConnectionImage

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

public class ConnectionImage extends MIDlet implements CommandListener
{
   Display tela;
   //form onde será exibida a imagem
   Form imageForm;

   //ImageItem onde a foto que virá do servidor será salva.
   ImageItem foto;

   //comandos
   Command buscarCommand, sairCommand;

   //string com a url para a imagem
   //colocamos a url diretamente aqui somente para exemplo
   //uma idéia seria vc deixar o usuario setar esse link
   //em um textbox ou algo do genero.

   String url = "http://www.seusite.com.br/suafoto.png";

   public ConnectionImage()
   {

       this.imageForm = new Form("");

       //imageItem onde será posta a imagem que vier do server
       //inicializamos ela com zero.
       //o qual poderia ser feito na hora de postar.
       this.foto = new ImageItem("foto",null,0,"");

       this.sairCommand = new Command("Sair", Command.EXIT, 0);
       this.buscarCommand = new Command("Buscar", Command.SCREEN, 1);

       this.imageForm.addCommand(this.sairCommand);
       this.imageForm.addCommand(this.buscarCommand);

       this.imageForm.setCommandListener(this);
   }

   public void startApp()
   {
       this.tela = Display.getDisplay(this);
       this.tela.setCurrent(this.imageForm);
   }

   public void pauseApp()
   {

   }
```

```
public void destroyApp(boolean bool)
{

}

public void commandAction(Command c, Displayable d)
{
    if (c == this.sairCommand)
    {
          this.destroyApp(true);
          this.notifyDestroyed();
    }
    else if(c == buscarCommand)
    {
     //se já houver um ImageItem no form
     //ele irá deleta-lo antes de jogar uma nova imagem
        if (imageForm.size() > 0){

            for (int i = 0; i < imageForm.size(); i++)
                imageForm.delete(i);
        }

        try
        {
               Image im;
               //chama o método que busca a imagem
               im = getImage(this.url);
               this.foto.setImage(im);
               imageForm.append(this.foto);
               // Display the form with the image
               this.tela.setCurrent(imageForm);
        }catch(Exception e)
        {

        }
    }
}

/*-------------------------------------------------
* Abre uma conexão http e baixa um arquivo png
* em um array de bytes
*-------------------------------------------------*/
private Image getImage(String url) throws IOException
{
    //usamos aqui ao invés de httpconnection
    //a conexao padrao, o qual nao nos garante nada.
    //embora a conexao http tambem nao seja muito confiavel.
ContentConnection connection = (ContentConnection) Connector.open(url);
    DataInputStream iStrm = connection.openDataInputStream();

    Image im = null;

    try
    {
        byte imageData[];
```

```
            int length = (int) connection.getLength();
            if (length != -1)
            {
                    imageData = new byte[length];

                    // Le um png em um array
                    iStrm.readFully(imageData);
            }
            else  //se o tamanho não está disponível
            {
                ByteArrayOutputStream bStrm = new ByteArrayOutputStream();

                    int ch;
                    while ((ch = iStrm.read()) != -1)
                          bStrm.write(ch);

                    imageData = bStrm.toByteArray();
                    bStrm.close();
            }

            //Cria a imagem de um array de bytes
            im = Image.createImage(imageData, 0, imageData.length);
        }
        finally
        {
           // Clean up
           if (iStrm != null)
                    iStrm.close();
           if (connection != null)
                    connection.close();
        }
        return (im == null ? null : im);
    }
}
```

ExemploHttpConnection

```
/*Este exemplo faz uma conexao http com um server, o qual retorna
um lista de usuarios dos quais nos podemos entao fazer uma
consulta apenas selecionando ele da lista e a outra url nos
retornará os dados, da pessoa.
Isto é só um exemplo, a parte ServerSide fica de sua
responsabilidade, visto que nao é este nosso objetivo.
As paginas podem tanto ser .jsp, como Servlets.
*/

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

public class ExemploConnect extends MIDlet implements CommandListener
{

    Display tela;
    List registros, result;
    Command sair, listar, voltar;
    String url = "www.seusite.com.br/suaPagina.jsp";

    public ExemploConnect()
    {
        this.sair = new Command("Sair",Command.EXIT, 0);
        this.listar = new Command("Listar",Command.SCREEN, 1);
        this.voltar = new Command("Voltar", Command.SCREEN, 0);

        this.registros = new List("Clientes",List.IMPLICIT);
        this.registros.addCommand(this.sair);
        this.registros.addCommand(this.listar);
        this.registros.setCommandListener(this);

        this.result = new List("Dados Cliente", List.IMPLICIT);
        this.result.addCommand(this.voltar);
        this.result.setCommandListener(this);

    }

    public void startApp()
    {
        this.tela = Display.getDisplay(this);
        this.tela.setCurrent(this.registros);

    }

    public void pauseApp()
    {

    }

    public void destroyApp(boolean b)
```

```
    {

    }

    public void commandAction(Command c, Displayable d)
    {
        if(c == this.sair)
        {
            this.destroyApp(true);
            this.notifyDestroyed();
        }
        //chama a função que faz a conexao com a pagina.
        if(c == this.listar)
        {
            this.conecta(this.registros);
        }

        //quando chamar o voltar limpa a lista para evitar
        //que sejam readicionados os mesmo itens na proxima conexao
        if(c == this.voltar)
        {
            int tamanho = this.result.size();
            for(int i= 0; i < tamanho; i++)
            {
                this.result.delete(i);
            }
            this.tela.setCurrent(this.registros);
        }

        //verifica o item escolhido e entao pega o codigo
        //e o junta com a url para fazer a busca no servidor
        if(c == List.SELECT_COMMAND)
        {
            String linha = this.registros.getString
(this.registros.getSelectedIndex());
            String cod = linha.substring(0,linha.indexOf("-"));
            String urlBusca = "www.seusite.com.br/BuscaCodigo.jsp?cod=";
            this.url = urlBusca+cod;

            //passamos a lista como parametro
            //pois é onde sera montado o resultado da busca
            this.conecta(result);
            this.tela.setCurrent(this.result);
        }
    }

    public void conecta(List lista)
    {
        try
            {
                //estabelecemos uma conexao http com a url.
          HttpConnection con = (HttpConnection) Connector.open(this.url);
                int status = -1;
                status = con.getResponseCode();

             //como a conexao foi http, é verificado o status da conexao
```

```
                if(status == HttpConnection.HTTP_OK)
                {
                    InputStream is = con.openInputStream();
                    StringBuffer sb = new StringBuffer();

                    int lido = is.read();

                   //como podemos perceber abaixo
            //é lido byte a byte do servidor.
                   while (lido != -1)
                    {
                        byte b1 = (byte)lido;

                        if(b1 == (byte)'\n')
                        {
                            lista.append(sb.toString(),null);
                            sb.setLength(0);
                        }
                        else
                        {
                            sb.append((char)b1);
                        }
                        lido = is.read();
                    }
                }
                //se nao conseguir estabelecer conexao,
                //colocamos esta mensagem como padrao.
                else
                {
                    lista.append("Nenhum Registro Encontrado!", null);
                }
            }
            catch (IOException io1)
            {

            }
    }
}
```

ExemploCanvas

```
/*
  Vamos construir um exemplo do uso do Canvas em J2ME.
  Canvas é o uso de graficos em nivel baixo.
  Vamos a ele...
*/

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


// Toda a aplicacao tem que "extender" de MIDlet, e implementar pelo
menos a
// interface CommandListener, caso vc queira ter interagir com a
aplicação!

public class ExemploCanvas extends MIDlet implements CommandListener
{
       Display tela;
       Command sair;
       // para desenhar na tela precisamos da Classe Canvas
       // como nossa classe principal não pode extender a classe Canvas
       // criaremos uma outra Classe que extenda de Canvas
       NossaCanvas canvas; //classe criada mais a baixo

       public ExemploCanvas()
       {
              this.tela=Display.getDisplay(this);

              this.sair=new Command("Sair",Command.EXIT,0);

              // inicia o NossoCanvas
              this.canvas=new NossaCanvas();

              //adiciona o comando sair ao canvas
              this.canvas.addCommand(this.sair);

              // prepara o canvas para "escutar" os comandos
              this.canvas.setCommandListener(this);
       }

       public void startApp()
       {
             //define a tela para o objeto NossoCanvas que vai desenhar
na tela
            this.tela.setCurrent(this.canvas);
       }

       public void pauseApp()
       {
        // não precisa executar nada se não quiser
       }
```

```
        public void destroyApp(boolean b)
        {
         // não precisa executar nada se não quiser
        }


        // metodo abstrato da interface CommandListener, é ativado quando
algum comando é acionado
        public void commandAction(Command c, Displayable d)
        {
             // reconhece que o comando foi o "sair" e executa.
             if(c==this.sair)
             {
                 this.destroyApp(true);
                 this.notifyDestroyed();
             }
        }
}

//classe criada extendendo de Canvas
class NossaCanvas extends Canvas
{
      public NossaCanvas()
      {

      }

// metodo abstrato de Canvas que pinta na tela
//ele é acionado na criacao do objeto e quando usa-se o metodo repaint()
      public void paint(Graphics g)
      {
       g.setColor(255,255,255); //define uma cor RGB

g.fillRect(0,0,96,64); //cria um retangulo ja pintado com a cor definida

g.setColor(128); //existe também uma sobrecarga para este método com cor
de 8 bits
       g.drawRect(5,5,88,59); //desenha um retangulo sem preenchimento
       g.setColor(192);
//cria-se uma variavel String porque o método drawString() nao aceita o
uso de "lala" diretamente no método.
       String teste="NoSSo Canvas";
       // define o texto as coordenadas, e onde as coordenadas devem ser
consideradas na string
       g.drawString(teste,7,7,Graphics.LEFT|Graphics.TOP);
      }
}
```

Referências Bibliográficas

Keogh, James - J2ME – The Complete Reference – Editora Osborne

Knudsen, Jonathan – Wireless Java – Developing with J2ME – Editora Apress

Almeida, Leandro Batista de - Material de Apoio

Copyright