# Arcabouços de Software para Desenvolvimento de Aplicações Embarcadas

Herman Martins Gomes - hmg@dsc.ufcg.edu.br

# Ementa

- Introdução a Dispositivos Móveis Portáteis, Aplicações Embarcadas e J2ME, Interface com o Usuário, Armazenamento Persistente de Dados, Redes e Serviços Web, Visão Geral dos Pacotes J2ME, Otimização de Aplicações e Estudos de Caso

# Objetivos

GERAIS

– Apresentar ao aluno uma visão geral da arquitetura e ambientes para o desenvolvimento de aplicações embarcadas. Capacitar o aluno ao desenvolvimento de aplicações utilizando a linguagem Java 2, Micro Edition (J2ME).

ESPECÍFICOS

– Fornecer o conceito e metodologia de desenvolvimento da plataforma J2ME/MIDP. Familiarizar o aluno com a biblioteca API e diferentes ambientes de desenvolvimento para J2ME. Treinar o aluno no desenvolvimento e otimização de software embarcado orientado a objetos de qualidade (documentação, legibilidade, funcionalidade).

# Avaliação

- Duas provas (P1, P2) e duas listas de exercícios (L1,L2). A média na disciplina será a média aritmética entre as quatro notas anteriores.
- Reposição de Prova apenas com justificativa formal apresentada na aula seguinte à realização da prova.

# Horários

- Turma1:
  - Quarta-feira 19:00-21:00
  - Sábado 08:00-10:00
- Turma2:
  - Quinta-feira 19:00-21:00
  - Sábado 10:00-12:00

# Programa

1ª UNIDADE – INTRODUÇÃO
- Dispositivos Móveis, Aplicações Embarcadas e Redes sem Fio
- Visão Geral de J2ME
- Arquitetura e Ambientes de Desenvolvimento para J2ME
- Programação para Dispositivos Portáteis – MIDP e MIDlets
- Melhores Práticas e Padrões para J2ME

2ª UNIDADE – INTERFACE COM O USUÁRIO
- Comandos, Itens e Processamento de Eventos
- Interface Gráfica

3ª UNIDADE: ARMAZENAMENTO PERSISTENTE DE DADOS
- Sistema de Gerenciamento de Registros: RecordStore
- Bancos de Dados

4a. UNIDADE: REDE E SERVIÇOS WEB
- Arcabouço genérico de conexão
- Serviços Web
- Noções de Criptografia e Segurança de Rede

5a. UNIDADE: VISÃO GERAL DOS PACOTES E CLASSES J2ME
- Diferenças entre MIDP 1.0 e 2.0
- Game API
- Mobile Media API
- Demais APIs

6a. UNIDADE OTIMIZAÇÃO DE APLICAÇÕES
- Benchmarking
- Ferramentas de Diagnóstico
- Uso de Memória
- Velocidade de Processamento

7a. UNIDADE: EXEMPLOS DE APLICAÇÕES J2ME

# Programação de aulas – Turma1

| Aula | Data | Assunto |
|---|---|---|
| 1 | 15/09/2004 (Quarta-feira) | Dispositivos Móveis, Aplicações Embarcadas e Redes sem Fio, Visão Geral de J2ME |
| 2 | 18/09/2004 (Sábado) | Arquitetura e Ambientes de Desenvolvimento para J2ME, Melhores Práticas e Padrões para J2ME |
| 3 | 22/09/2004 (Quarta-feira) | Comandos, Itens e Processamento de Eventos |
| 4 | 25/09/2004 (Sábado) | Comandos, Itens e Processamento de Eventos, Interface Gráfica. 1ª Lista de Exercícios será distribuída. |
| 5 | 16/10/2004 (Sábado) | Interface Gráfica. 1ª Lista de Exercícios deverá ser entregue ao professor. Comentários sobre a 1ª Lista de Exercícios em sala de aula. |
| 6 | 20/10/2004 (Quarta-feira) | Divulgação das notas da 1ª Lista de Exercícios. Realização da 1ª Prova. |
| 7 | 23/10/2004 (Sábado) | Sistema de Gerenciamento de Registros: RecordStore. Conexão a Bancos de Dados. |
| 8 | 27/10/2004 (Quarta-feira) | Arcabouço genérico de conexão de rede, Serviços Web |

| Aula | Data | Assunto |
|---|---|---|
| 9 | 30/10/2004 (Sábado) | Noções de Criptografia e Segurança de Rede |
| 10 | 03/11/2004 (Quarta-feira) | Diferenças entre MIDP 1.0 e 2.0, Game API |
| 11 | 06/11/2004 (Sábado) | Mobile Media API, Demais APIs |
| 12 | 10/11/2004 (Quarta-feira) | Demais APIs. 2ª Lista de Exercícios será distribuída. |
| 13 | 13/11/2004 (Sábado) | Benchmarking, Ferramentas de Diagnóstico, Uso de Memória, Velocidade de Processamento |
| 14 | 17/11/2004 (Quarta-feira) | J2ME e Symbian OS. Exemplos de Aplicações J2ME. 2ª Lista de Exercícios deverá ser entregue ao professor. Comentários sobre a 2ª Lista de Exercícios em sala de aula. |
| 15 | 20/11/2004 (Sábado) | Realização da 2ª Prova. |
| 16 | 24/11/2004 (Quarta-feira) | Buffer de contingência. Reposições de Provas. |

# Programação de aulas – Turma2

| Aula | Data | Assunto |
| --- | --- | --- |
| 1 | 16/09/2004 (Quinta-feira) | Dispositivos Móveis, Aplicações Embarcadas e Redes sem Fio, Visão Geral de J2ME |
| 2 | 18/09/2004 (Sábado) | Arquitetura e Ambientes de Desenvolvimento para J2ME, Melhores Práticas e Padrões para J2ME |
| 3 | 23/09/2004 (Quinta-feira) | Comandos, Itens e Processamento de Eventos |
| 4 | 25/09/2004 (Sábado) | Comandos, Itens e Processamento de Eventos, Interface Gráfica. 1ª Lista de Exercícios será distribuída. |
| 5 | 16/10/2004 (Sábado) | Interface Gráfica. 1ª Lista de Exercícios deverá ser entregue ao professor. Comentários sobre a 1ª Lista de Exercícios em sala de aula. |
| 6 | 21/10/2004 (Quinta-feira) | Divulgação das notas da 1ª Lista de Exercícios. Realização da 1ª Prova. |
| 7 | 23/10/2004 (Sábado) | Sistema de Gerenciamento de Registros: RecordStore. Conexão a Bancos de Dados. |
| 8 | 28/10/2004 (Quinta-feira) | Arcabouço genérico de conexão de rede, Serviços Web |

| Aula | Data | Assunto |
| --- | --- | --- |
| 9 | 30/10/2004 (Sábado) | Noções de Criptografia e Segurança de Rede |
| 10 | 04/11/2004 (Quinta-feira) | Diferenças entre MIDP 1.0 e 2.0, Game API |
| 11 | 06/11/2004 (Sábado) | Mobile Media API, Demais APIs |
| 12 | 11/11/2004 (Quinta-feira) | Demais APIs. 2ª Lista de Exercícios será distribuída. |
| 13 | 13/11/2004 (Sábado) | Benchmarking, Ferramentas de Diagnóstico, Uso de Memória, Velocidade de Processamento |
| 14 | 18/11/2004 (Quinta-feira) | J2ME e Symbian OS. Exemplos de Aplicações J2ME. 2ª Lista de Exercícios deverá ser entregue ao professor. Comentários sobre a 2ª Lista de Exercícios em sala de aula. |
| 15 | 20/11/2004 (Sábado) | Realização da 2ª Prova. |
| 16 | 25/11/2004 (Quinta-feira) | Buffer de contingência. Reposições de Provas. |

# Bibliografia


- *Enterprise J2ME: Developing Mobile Java Applications*, Michael Juntao Yuan, Prentice Hall, 2004.
- *The Complete Reference: J2ME*, James Keogh, McGrawHill, 2003.
- *Mobile and Wireless Design Essentials*, Martyn Mallick, Wiley Publishing, 2003.
- *Wireless Java: Developing with J2ME*, Jonathan Knudsen, Apress, 2003.
- *MIDP 2.0 Style Guide for the Java 2 Platform, Micro Edition*, Cynthia Bloch and Annette Wagner, Addison-Wesley, 2003.
- *J2ME in a Nutshell: a Desktop Quick Reference*, Kim Topley, O'Reilly&Associates Inc, 2002.
- *Java 2 Platform, Micro Edition (J2ME)*, Sun Microsystems, http://java.sun.com/j2me/

# 1ª UNIDADE – INTRODUÇÃO

- Dispositivos Móveis, Aplicações Embarcadas e Redes sem Fio
- Visão Geral de J2ME
- Arquitetura e Ambientes de Desenvolvimento para J2ME
- Programação para Dispositivos Portáteis – MIDP e MIDlets
- Melhores Práticas e Padrões para J2ME

# Dispositivos Móveis, Aplicações Embarcadas e Redes sem Fio

- A indústria de dispositivos móveis portáteis e redes sem fio cresce a passos largos
- Novos dispositivos são constantemente lançados e redes sem fio estão ampliando suas facilidades de acesso a comunicação de dados

# Dispositivos Móveis, Aplicações Embarcadas e Redes sem Fio

- ***Dispositivos móveis*** são quaisquer aparelhos de comunicação e/ou computação que podem ser usados enquanto se está em movimento
- O termo "***sem fio***" (ou *wireless*) se refere à transmissão de voz e/ou dados através de ondas de rádio
- Uma rede sem fio pode ser acessada tanto a partir de um dispositivo móvel como a partir de uma localização fixa

# Dispositivos Móveis, Aplicações Embarcadas e Redes sem Fio

- Uma aplicação ***embarcada*** é toda aquela que roda em um dispositivo de propósito específico, desempenhando alguma tarefa que seja útil para o dispositivo
- Assim, as aplicações que executam em uma câmera digital, um telefone celular, ou no controle de um elavador, podem ser vista como aplicações embarcadas

# Dispositivos Móveis, Aplicações Embarcadas e Redes sem Fio

- Para uma aplicação ser considerada móvel ou sem fio, ela deve ser adaptada às características do dispositivo em que será executada
- Recursos limitados, baixas taxas de transferência de rede e conexões intermitentes são todos fatores que devem ser levados em conta no projeto dessas aplicações

# Dispositivos Móveis, Aplicações Embarcadas e Redes sem Fio

- Aplicações sem fio que não são móveis utilizam redes sem fio fixas

  Exemplos?

- Há também as aplicações móveis que não se caracterizam como sem fio.

  Exemplos?

# Dispositivos Móveis, Aplicações Embarcadas e Redes sem Fio

- Classes de aplicações
  - Comércio móvel (ou *m-commerce*) se refere à compra de produtos ou serviços a partir de um terminal móvel. Exemplos?
  - Negócios móveis (ou *m-business*) relaciona-se a *e-business* da mesma forma que *m-commerce* relaciona-se com *e-commerce*. Tipicamente, são utilizados por corporações para fornecer acesso móvel seguro a dados da empresa a partir de qualquer localização. Exemplos?

# Dispositivos Móveis, Aplicações Embarcadas e Redes sem Fio

- Desafios
  - São problemas normalmente associados a redes sem fio: pouca cobertura e penetração, largura da banda limitada, tempos latência elevados, baixa confiabilidade, custos elevados, ausência de padrões
  - A escolha da classe certa de dispositivos é de fundamental importância para o sucesso de uma aplicação móvel
  - É crítico que os dispositivos forneçam os recursos de hardware necessários aos requisitos da aplicação

# Dispositivos Móveis, Aplicações Embarcadas e Redes sem Fio

- Propulsionadores da Área
  - Redes Sem Fio - aumento da banda de transmissão
  - Conexão indefinida - “allways-on”
  - Custos mais baixos – redes “packet switched” vs “circuit switched”
  - Novos serviços
  - Interoperabilidade entre operadoras
  - Redes sem fio públicas

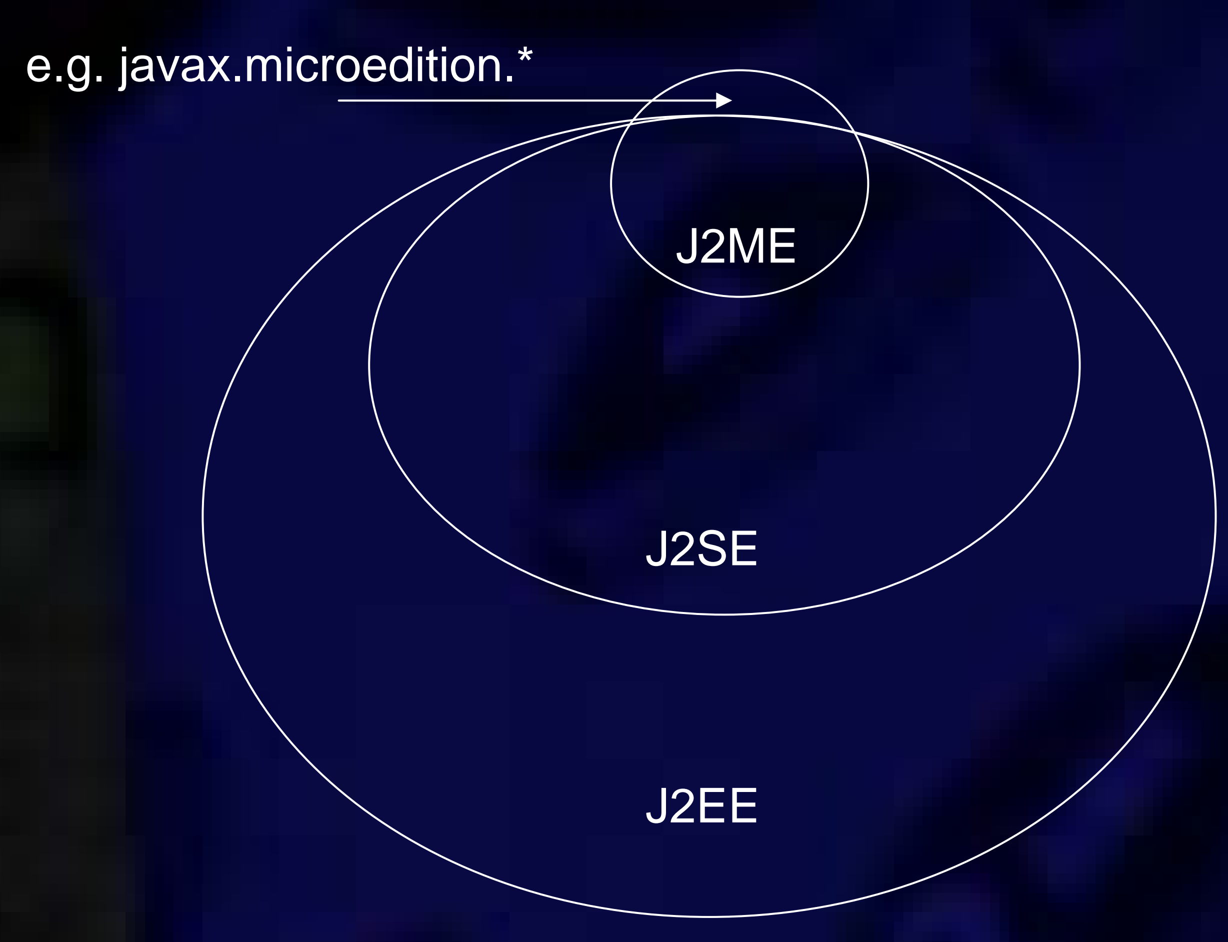
# Visão Geral de J2ME

- Por quê JAVA?
  - Portabilidade: desenvolver e manter uma única versão de aplicação para uma classe de vários dispositivos gera economia
  - Robustez: bytecodes são verificados antes de execução e a memória não mais necessária é reciclada pelo coletor de lixo, falhas na aplicação afetam apenas a máquina virtual
  - Segurança: APIs podem fornecer facilidades de segurança de dados e programas
  - Orientada a objetos: favorece reuso de código e o desenvolvimento modular

# Visão Geral de J2ME

- Apesar da portabilidade ser um conceito-chave da filosofia JAVA, ela tem limites:
  - → só funciona entre sistemas operacionais e plataformas de hardware semelhantes
- Assim, JAVA foi particionada em 3 edições, todas com papéis importantes em aplicações móveis
  - J2SE: é a base da plataforma, define as JVMs e bibliotecas que rodam em PCs e estações de trabalho
  - J2EE: adiciona um grande número de APIs e ferramentas propícias ao desenvolvimento de servidores de aplicações complexas, e aplicações corporativas
  - J2ME: é projetada para pequenos dispositivos, contendo máquinas virtuais otimizadas e um mínimo de bibliotecas e adaptações simplificadas para bibliotecas J2SE padrão

# Visão Geral de J2ME



# Visão Geral de J2ME

- Plataforma de desenvolvimento voltada para 2 tipos de dispositivos:
  - *High-end consumer devices:*
    - CDC (*Connected Device Configuration*)
    - TV interativa, Videofones, dispositivos *sem fio*
    - Grande variedade de interfaces com usuário
    - Memória típica de 2 a 4 Mb.
    - Conexão persistente e de banda larga, normalmente TCP/IP

# Visão Geral de J2ME

- Plataforma de desenvolvimento voltada para 2 tipos de dispositivos:
  - *Low-end consumer devices:*
    - *CLDC (Connected Limited Device Configuration)*
    - Telefones Celulares, pagers bi-direcionais, PDAs,...
    - Processador lento (8 a 32 MHz)
    - Memória da ordem de KBytes
    - Conexão lenta, intermitente (9600bps) e normalmente não baseada em TCP/IP
    - Ligados através de baterias

# Visão Geral de J2ME

- Profiles: complementam uma configuração através da adição de classes que forneçam características apropriadas a um tipo particular de dispositivo ou segmento específico de mercado
  - MIDP (Mobile Information Device Profile): adiciona rede, componentes de interface armazenamento local a CLDC (telefones celulares e dispositivos similares)
  - PDA Profile: similar a MIDP, mas dirigida a PDAs
  - Foundation Profile: extende CDC incluindo a maioria das bibliotecas Java2 1.3
  - Personal Basis and Personal Profiles: correntemente sendo especificadas, adicionan interface a Foundation Profile
  - RMI Profile: Remote Method Invocation adicionadas a Foundation Profile
  - Game Profile: fornece uma plataforma para desenvolvimento de jogos em CDC

# Visão Geral de J2ME

MIDP Profile

PDA Profile

CLDC

Personal

Personal basis

RMI Profile

Game Profile

Foundation Profile

CDC

Configuration

Profile

# Visão Geral de J2ME

- Algumas diferenças entre J2SE e J2ME
  - Apesar de J2ME ser uma versão reduzida de J2SE não se deve esperar que uma aplicação J2SE execute num ambiente J2ME sem precisar sofrer modificações importantes de código
  - Há classes e recursos acessados por uma aplicação J2SE que não necessariamente estão disponíveis num ambiente J2ME
  - A forma de chamar um programa J2ME é diferente de J2SE, uma vez que dispositivos portáteis em geral não possuem prompt de comando

# Visão Geral de J2ME

- Até quando o mercado terá dispositivos pequenos o bastante para justificar J2ME?
  - Ainda por algum tempo, apesar do 3G:
    - Grande parte da carga da bateria normalmente é necessária para o transmissor de rádio e o processamento de RF
    - Produção em massa e a baixos custos normalmente implicam em menos memória e processadores mais lentos

# Requerimentos de MIDP ao Software Básico

- Ambiente de execução protegida, oferecido pelo SO
- Algum tipo de suporte à rede
- Acesso ao teclado e outros dispositivos de entrada (como telas *touchscreen*)
- Acesso à tela do sistema como um *array* retangular de *pixels*
- Acesso a algum tipo de armazenamento persistente

# MIDP e MIDlets

- Aplicações JAVA que executam em dispositivos MIDP são conhecidas como MIDlets
- Um MIDlet consiste de no mínimo uma classe JAVA que deve ser derivada da classe abstrata javax.microedition.midlet.MIDlet
- MIDlets utilizam um ambiente de execução dentro da JAVA VM, o qual fornece um ciclo de vida controlado a partir de um conjunto de métodos que todo MIDlet deve implementar
- MIDlets podem também usar métodos para obter serviços de seu ambiente, e deverão usar apenas as APIs definidas na especificação MIDP

# MIDP e MIDlets

- Um grupo de MIDlets relacionadas podem ser reunidas em uma MIDlet *suite*
- Todas os MIDlets de uma *suite* são empacotados e instalados em (ou removidos de) um dispositivo como sendo uma única entidade
- MIDlets numa *suite* compartilham todos os recursos estáticos e dinâmicos de seu ambiente:
  - Dados de execução podem ser compartilhados entre MIDlets e as primitivas usuais de sincronização JAVA podem ser usadas para controlar o acesso aos dados
  - Dados persistentes também podem ser acessados por todos os MIDlets de uma *suite*

# MIDP e MIDlets

- Exemplo do compartilhamento de classes e dados entre MIDlets de uma mesma *suite*: suponha que a *suite* contenha a classe Contador, que será usada para contar quantas instâncias de MIDlets da *suite* estão executando

```
public class Contador {
        private int instancias;
        public static synchronized void incremente() {
            instancias++;
        }
        public static synchronized void decremente() {
            instancias--;
        }
        public static int getInstances() {
            return instances;
        }
}
```

- Apenas uma única instância de Contador será carregada na JAVA VM, não importa quantos MIDlets da *suite* estejam em execução. Isto significa que a contagem será de fato global.

# MIDP e MIDlets

- Todos os arquivos de uma MIDlet *suite* devem estar contidos dentro de um pacote JAR
- Estes pactes contém as classes do MIDlet, outros recursos (como imagens) e um arquivo de manifesto
- O arquivo de manifesto contém uma lista de atributos e definições que serão usados pelo gerenciador de aplicações para instalar os arquivos do JAR no dispositivo

# MIDP e MIDlets

- Atributos do arquivo de manifesto

| | |
|---|---|
| MIDlet-Name | Nome da *suite* |
| MIDlet-Version | Número da versão do MIDlet |
| MIDlet-Vendor | Nome do fornecedir do MIDlet |
| MIDlet-n | Um para cada MIDlet. Contém Nome do MIDlet, icon opcional e nome da classe MIDlet |
| MicroEdition-Profile | J2ME profile necessária para executar o MIDlet |
| MicroEdition-Configuration | Configuração necessária para executar o MIDlet |
| MIDlet-Icon | Icone associado com o MIDlet (PNG), opcional |
| MIDlet-Description | Descrição opicional |
| MIDlet-Info-URL | URL contendo mais informações sobre o MIDlet |

# MIDP e MIDlets

- Segurança de MIDlets
  - O modelo de segurança JAVA em J2SE é muito caro em termos de recursos de memória e requer conhecimentos de configuração que não estão presentes no usuário típico de um dispositivo móvel
  - Assim, nem CLDC nem MIDP incluem estas funcionalidades

# MIDP e MIDlets

- Segurança de MIDlets
  - Criptografia de chave pública e certificados não estão disponíveis como padrão
  - É necessário ter cuidado ao instalar MIDlets e preferencialmente só aceitar software de fontes confiáveis
  - Em MIDP 2.0, foi incluído o protocolo https que ajuda a aliviar estes problemas

# Ambiente de Execução e Ciclo de Vida de um MIDlet

- Todo MIDlet deve ser derivado da classe abstrata javax.microedition.midlet.MIDlet

```
public class MinhaMIDlet extends MIDlet {
    //Construtor opcional
    MinhaMIDlet() {
    }
    protected void StartApp() throws MIDletStateChangeException {
    }
    protected void pauseApp() {
    }
    protected void destroyApp(boolean unconditional)
                                    throws MIDletStateChangeException {
    }
}
```

- Em qualquer instante de tempo um MIDlet deverá estar em 1 dentre 3 estados: Pausado, Ativo ou Destruído

# Ambiente de Execução e Ciclo de Vida de um MIDlet

- Quando um MIDlet é carregado, inicialmente fica no estado Pausado
- As inicializações da classe são efetuadas (inicializações de variáveis e chamada ao construtor)
- Se ocorre uma exceção durante a execução do construtor, o MIDlet é destruída
- Caso contrário, o MIDlet é escalonado para execução em algum tempo futuro, quando seu estado muda de Pausado para Ativo e o método StartApp() é invocado

# Ambiente de Execução e Ciclo de Vida de um MIDlet

- startApp()
  - Na inicialização e reinicialização do Midlet
  - Deve obter os recursos necessários para execução (timers, conexões de rede) e fazer as inicializações de objetos
  - Isto pode também ser feito no método construtor
  - Entretanto, na especificação MIDP, o *Display* só estará disponível após a primeira invocação de startApp()
  - Portanto, a portabilidade pode ficar comprometida se o MIDlet acessar o *Display* no construtor
  - Pode falhar de 2 maneiras:
    - transient: falha temporária. Leva o MIDlet de volta ao estado pausado e avisa ao sistema lançando uma MIDletStateChangeException
    - non-transient: Erro fatal. Pode ser tratado ou lançado para o sistema que irá invocar o método destroyApp

# Ambiente de Execução e Ciclo de Vida de um MIDlet

- pauseApp()
  - No estado Ativado, o MIDlet irá executar até que seja pausado ou destruído
  - A qualquer tempo a plataforma MIDP pode colocar o MIDlet no estado Pausado, através do método pauseApp()
  - Num telefone celular isto pode ocorrer quando uma chamada telefônica precisa ser recebida e os recursos de tela precisam ser liberados
  - Ao ser notificado, o MIDlet deve liberar tantos recursos quanto possível
  - O sistema pode optar por terminar a execução de MIDlets que não obedecerem a requisição
  - Se o ambiente MIDP decide mover o MIDlet para o estado Ativo novamente, por exemplo, após a finalização da ligação telefônica, este invoca o método startApp() do MIDlet

# Ambiente de Execução e Ciclo de Vida de um MIDlet

- destroyApp(boolean)
  - Chamado quando a plataforma de execução precisa finalizar o MIDlet, o qual precisará liberar todos os recursos que foram alocados
  - Se o boolean for true, significa que o MIDlet deve ser finalizado incondicionalmente
  - Caso contrário, o MIDlet pode indicar que precisa continuar executando através do lançamento de uma exceção

# Ambiente de Execução e Ciclo de Vida de um MIDlet

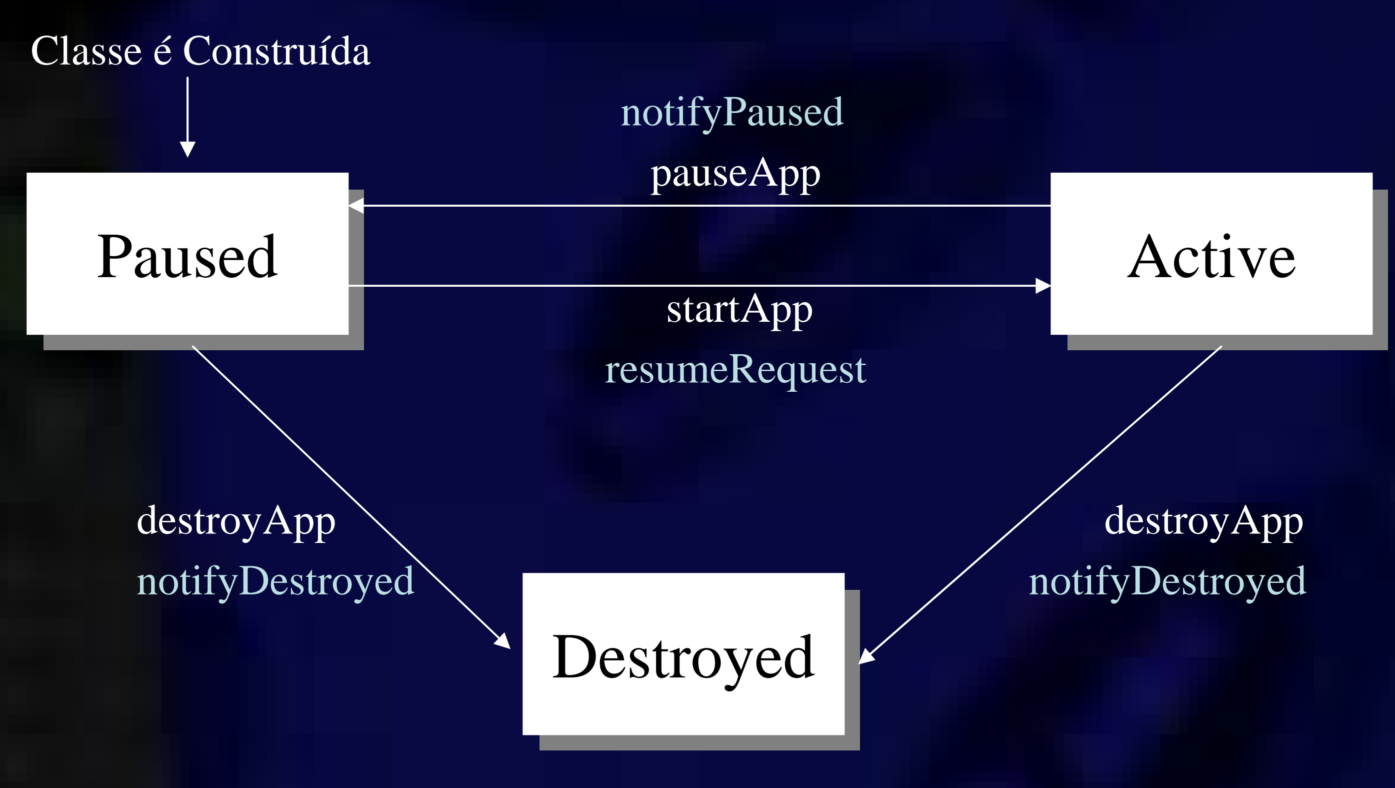
- destroyApp(boolean)
  - O seguinte código pode ser usado para responder a um botão de Exit na interface com o usário de um MIDlet:

```
try {
        //chama destroyApp para liberar recursos
        destroyApp(false);
        //Arranja para o MIDlet ser destruído
        notifyDestroyed();
} catch (MIDletStateChangeException ex) {
                //O MIDlet não quer ser fechado
}
```

# Ambiente de Execução e Ciclo de Vida de um MIDlet

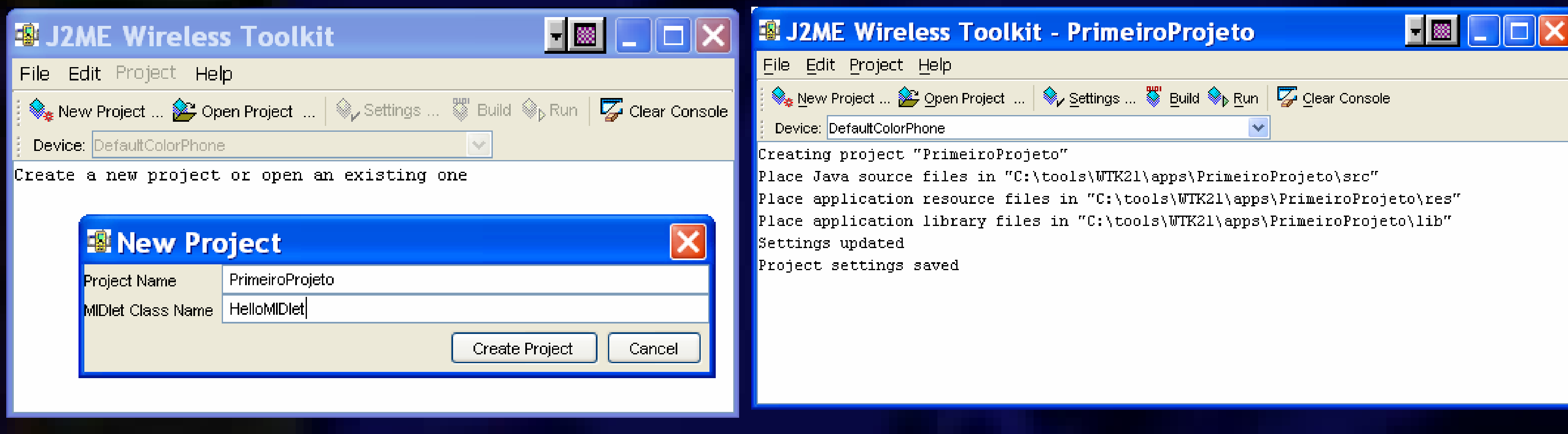
- Relação entre os métodos destroyApp() e notifyDestroyed()
  - Quando o MIDlet está sendo destruído pela plataforma, é provável que o usuário tenha solicitado isto através de um destroyApp(true). Neste caso não é necessário invocar o método notifyDestroyed()
  - Quanto o MIDlet quer terminar por conta própria, tipicamente porque não há mais trabalho para fazer ou o usuário pressionou um botão Exit, pode fazer isto invocando seu método notifyDestroyed(), o qual informa a plataforma de que deve ser destruído. Neste caso a plataforma não chama o método destroyApp(), pois assume que o MIDlet já está pronto paraser terminado.

# Ciclo de Vida de uma MIDlet



# Primeiro Programa

- O que é necessário
  - Editor de texto
  - SUN J2ME Wireless Toolkit
- Etapas
  - Criar um projeto no J2ME Wireless Toolkit (KToolbar), o nome do projeto será PrimeiroProjeto e o nome da classe HelloMIDlet



# Primeiro Programa

- Salvar o seguinte código com o nome: <WTK>\PrimeiroProjeto\apps\src\HelloMIDlet.java em que <WTK> é o diretório onde o toolkit foi instalado

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

public class HelloMIDlet
    extends MIDlet
    implements CommandListener {
  private Form mMainForm;

  public HelloMIDlet() {
    mMainForm = new Form("HelloMIDlet");
    mMainForm.append(new StringItem(null, "Hello, MIDP!"));
    mMainForm.addCommand(new Command("Exit", Command.EXIT, 0));
    mMainForm.setCommandListener(this);
  }

  public void startApp() {
    Display.getDisplay(this).setCurrent(mMainForm);
  }

  public void pauseApp() {}

  public void destroyApp(boolean unconditional) {}

  public void commandAction(Command c, Displayable s) {
    notifyDestroyed();
  }
}
```

# Primeiro Programa

- Compilar e executar o programa criado, através das opções *build* e *run* do wireless toolkit

# Exercícios

- Exercício1: experimente mudar o device (dispositivo emulado) e executar seu programa novamente para ver o resultado
- Exercício2: verifique na API do toolkit que outros comandos padrão estão disponíveis (além de Command.EXIT), modifique a sua aplicação para tratar diferentes comandos. Teste o resultado
- Exercício3: compile e execute alguns dos projetos de demonstração do toolkit. Experimente-os nos diferentes dispositivos disponíveis

# Ambientes de Desenvolvimento para J2ME

- Sun Java2SE SDK – free
  http://java.sun.com/j2se/

- J2ME Toolkits
  - Sun Wireless Toolkit
    http://java.sun.com/products/j2mewtoolkit/
  - Nokia Developer´s Suite
    http://www.forum.nokia.com/main/0,6566,034-2,00.html
  - Siemens Mobility Toolkit
    http://www.siemens-mobile.com/
  - ......

# Ambientes de Desenvolvimento para J2ME

- Ambientes Integrados de Desenvolvimento
  - Borland JBuilder – comercial
    http://www.borland.com/jbuilder/
  - Sun One Studio (anteriorment Forte Java) -> não mais disponível
  - Sun Java Studio Creator - comercial
    http://wwws.sun.com/software/products/jscreator
  - Eclipse - free
    http://www.eclipse.org
  - EclipseME plugin - free
    http://eclipseme.sourceforge.net/
  - Mais ferramentasm para JAVA disponíveis em:
    http://www.mhavila.com.br/link/prog/java/java-tool.html

# Ambientes de Desenvolvimento para J2ME

- Exercício1: Instalar o plugin Eclipseme e configurar o Eclipse para trabalhar com o Sun Wireless Toolkit (WTK20, WTK21 ou WTK22)

- Exercício2: Instalar e configurar o Nokia Developer´s Toolkit para funcionar com Eclipe]se

- Exercício3: Criar um projeto no Eclipse contendo o MIDlet escrito na aula anterior e testá-lo utilizando os toolkits da Sun e da Nokia

# Melhores Práticas de Programação J2ME

- Manter o projeto das aplicações simples
  - O projeto de uma aplicação tipicamente envolve a divisão do problema em objetos que possuem dados e métodos associados
  - A melhor prática é limitar o projeto à funcionalidade mínima requerida para atender às expectativas do usuário
  - Colocar cada componente funcional em seu próprio MIDlet sempre que possível, e empacotar os MIDlets da aplicação numa mesma MIDlet *suite*
  - Isto irá facilitar o gerenciador de aplicações no dispositivo a controlar melhor os MIDlets e seus recursos

# Melhores Práticas de Programação J2ME

- Manter o tamanho das aplicações pequeno
  - O tamanho de uma aplicação J2ME é crítico
  - A melhor prática é remover todo e qualquer componente desnecessário a fim de reduzir o tamanho final da aplicação
  - Abusar do uso de recursos multimídia (muito comuns da plataforma PC) pode aumentar drasticamente o tamanho de uma aplicação

# Melhores Práticas de Programação J2ME

- Manter o tamanho das aplicações pequeno (continuação)
  - Mesmo seguindo a recomendação acima, sua aplicação ainda pode ser muito grande para ser executada em um dispositivo de pequeno porte
  - Neste caso, a solução é dividir a aplicação em vários MIDlets e então combiná-los em uma *suite*, conforme descrito no slide anterior

# Melhores Práticas de Programação J2ME

- Limitar o uso de memória
  - Projetar a sua aplicação de forma a gerenciar a memória de forma eficiente
  - Há basicamente 2 tipos de gerenciamento de memória que podem ser usados em uma aplicação J2ME
    - Gerenciamento global
    - Gerenciamento de pico

# Melhores Práticas de Programação J2ME

- Limitar o uso de memória (continuação)
  - Gerenciamento global – dar preferência a usar tipos primitivos, usar o tipo mínimo de dado para a tarefa
  - Gerenciamento de pico – gerenciar a coleta de lixo, apesar da coleta ser automatizada (como em J2SE), você não sabe quando vai ocorrer, assim é critico liberar toda a memória desnecessária a qualquer instante de tempo. Recomendações:
    - Alocar objetos apenas imediatamente antes de serem usados, ao invés de fazer isto no início da aplicação
    - Atribuir null aos objetos não mais necessários
    - Procurar reusar objetos ao invés de criar novos objetos, isto reduz tanto alocação de memória quanto tempo de processamento
    - Reduzir a possibilidade de exceções também economiza memória
    - Liberar recursos sempre que não sejam mais necessários

# Melhores Práticas de Programação J2ME

- Computações massivas devem ser enviadas ao servidor
  - Quando o tipo de computação a ser executada pelo dispositivo móvel portátil excede suas capacidades, a saída é construir uma aplicação J2ME cliente-serviço ou que faça uso de um *web-service*
  - Nesta situação, a aplicação é dividida em 2 partes:
    - a parte do cliente, que fornece a camada de apresentação da aplicação, envia as requisições do cliente ao servidor e recupera os resultados para apresentação no dispositivo e
    - a parte do serviço, que recebe e processa as requisições do cliente e retorna as respostas de volta ao cliente, praticamente todo o processamento ocorre neste nível

# Melhores Práticas de Programação J2ME

- Computações massivas devem ser enviadas ao servidor
  - Exemplo:
    - Suponha um funcionário de um serviço de remessa noturna que está em dúvida sobre o endereço do destinatário de uma encomenda
    - Ele pode usar uma aplicação-cliente em seu telefone celular para consultar o banco de dados da empresa sobre o número do telefone do destinatário
    - A consulta é capturada pela aplicação-cliente e enviada através de uma conexão de rede sem fio para a parte da aplicação encarregada da lógica do negócio (*business logic*), que está executando num servidor corporativo
    - A lógica do negócio acessa que serviços Web são necessários para atender à requisição e prossegue invocando esses serviços com os parâmetros necessários

# Melhores Práticas de Programação J2ME

- Computações massivas devem ser enviadas ao servidor
  - Exemplo (continuação):
    - Neste exemplo, o software da lógica do negócio determina qual SGBD é necessário para localizar o telefone do destinatário, o SGBD (camada de processamento) por sua vez realiza a computação necessária e encaminha a resposta para a camada de lógica do negócio, a qual, por sua vez, envia o telefone encontrado para o telefone celular do funcionário que a solicitou
    - O processamento na parte cliente da aplicação se limita a apresentar a interface com o usuário, capturar a requisição do usuário, abrir e manter as conexões de rede com os servidores da empresa, enviar o pedido, receber a resposta e apresentá-la na tela do dispositivo móvel portátil

# Melhores Práticas de Programação J2ME

- Gerenciar o uso da conexão de rede
  - Problemas na transição entre células (delays e interrupções)
  - *dead zones* – o celular está fora da área de cobertura da operadora
  - Tipicamente a conexão com a operadora é quebrada e algumas vezes não é possível restabelecê-la automaticamente
  - Os problemas de conexão normalmente ocorrem sem aviso prévio
  - Apesar de não ser possível evitar eventuais quebras de conexão, é possível reduzir seu impacto para a aplicação e seu usuário

# Melhores Práticas de Programação J2ME

- Gerenciar o uso da conexão de rede (cont...)
  - Tornar as conexões curtas, ou seja, transferir o mínimo de informação necessária para executar uma tarefa
  - Por exemplo, numa aplicação para recuperar emails de um servidor, ao invés de recuperar todo o conteúdo, recuperar apenas os seus cabeçalhos (From, Subject, Data)
  - Essa informação resumida é apresentada na tela do celular com opções para navegar pela lista de emails, ver um conteúdo particular, deletar emails etc (como ocorre com Webmail)

# Melhores Práticas de Programação J2ME

- Gerenciar o uso da conexão de rede (cont..)
  - Considerar o uso de uma tecnologia do tipo *store-forwarding* e um agente do lado do servidor (*server-side agent*) toda vez que sua aplicação precisar solicitar volumes muito grandes de informação
  - Um *server-side agent* é um programa executando no servidor que é encarregado de receber requisições de dispositivos móveis e recuperar a informação a partir de uma fonte de dados, o que é muito similar à camada de lógica do negócio em *Web services*
  - Os resultados da consulta são então guardados pelo agente até que o dispositivo móvel solicite a informação, quando então o conteúdo é despachado para o dispositivo móvel

# Melhores Práticas de Programação J2ME

- Gerenciar o uso da conexão de rede (cont..)
  - O pedido do dispositivo em geral consiste de uns poucos bytes
  - Exemplo
    - Um cliente encontra um carro de entrega de encomendas à noite próximo de sua casa e pergunta ao funcionário qual o status do pacote que despachou na semana passada
    - O cliente não tem muita informação sobre o pacote despachado, exceto seu nome, endereço e destino
    - O sistema de rastreamento móvel rodando no celular do funcionário pode responder à consulta do usuário através das tecnologias *store-forwarding e server-side agent* acima descritas

# Melhores Práticas de Programação J2ME

- Gerenciar o uso da conexão de rede (cont...)
  - É importante incorporar nas aplicações móveis sem fio um mecanismo de recuperação de quedas de transmissão
  - Por exemplo, é possível reter no dispositivo móvel alguma informação-chave sobre uma requisição até que a mesma seja completamente atendida
  - A aplicação móvel pode então usar esta chave para resubmeter automaticamente a requisição ou a pedido do usuário, na ocorrência de uma quebra de conexão

# Melhores Práticas de Programação J2ME

- Simplificar a interface com o usuário
  - Grandes diferenças entre Interface de PCs e interfaces de dispositivos móveis portáteis
  - Fazer bom uso das características encontradas em cada dispositivo
  - Em alguns casos, quando muita informação precisa ser digitada no dispositivo, pode ser mais interessante digitá-la num PC e depois transferi-la para a sua aplicação executanto no dispositivo
  - Cuidado com a escolha e uso de teclas de atalho, nem sempre essas teclas são de fácil acesso no dispositivo final
  - Evite o desenvolvimento de interfaces em que muita informação textual precisa ser digitada, use menus sempre que for possível
  - Tenha em mente que muitos usuários de aplicações para dispositivos móveis portáteis gostariam de utilizá-las utilizando apenas 1 dedo enquanto segura o aparelho com a mesma mão

# Melhores Práticas de Programação J2ME

- Usar variáveis locais
  - É possível aumentar a velocidade de processamento de uma aplicação se os passos extras para acessar os componentes de uma classe forem eliminados através da utilização de variáveis locais
  - Evidentemente, é necessário ponderar os ganhos de processamento versus os benefícios do encapsulamento de dados em classes

- Evitar concatenar *Strings*
  - A concatenação de *Strings* consome memória e valiosos ciclos de processamento
  - Uma alternativa para reduzir o uso de memória (mas não o tempo extra de processamento) é utilizar um objeto da classe *StringBuffer*

# Melhores Práticas de Programação J2ME

- Evitar sincronizações
  - É muito comum disparar um ou múltiplos *threads* numa determinada operação
  - Por exemplo, uma rotina de classificação pode ser compartilhada simultaneamente por múltiplas operações que necessitam classificar dados
  - Cada operação chama a rotina de classificação de forma independente das demais operações
  - *Deadlocks* e outros conflitos podem surgir nessa situação
  - Os problemas podem ser evitados através de primitivas de sincronização
  - É interessante usar *threads* sempre que uma operação vai levar mais do que 0.1s para executar
  - Evitar o uso de sincronização sempre que não exista a possibilidade de que conflitos entre operações

# Melhores Práticas de Programação J2ME

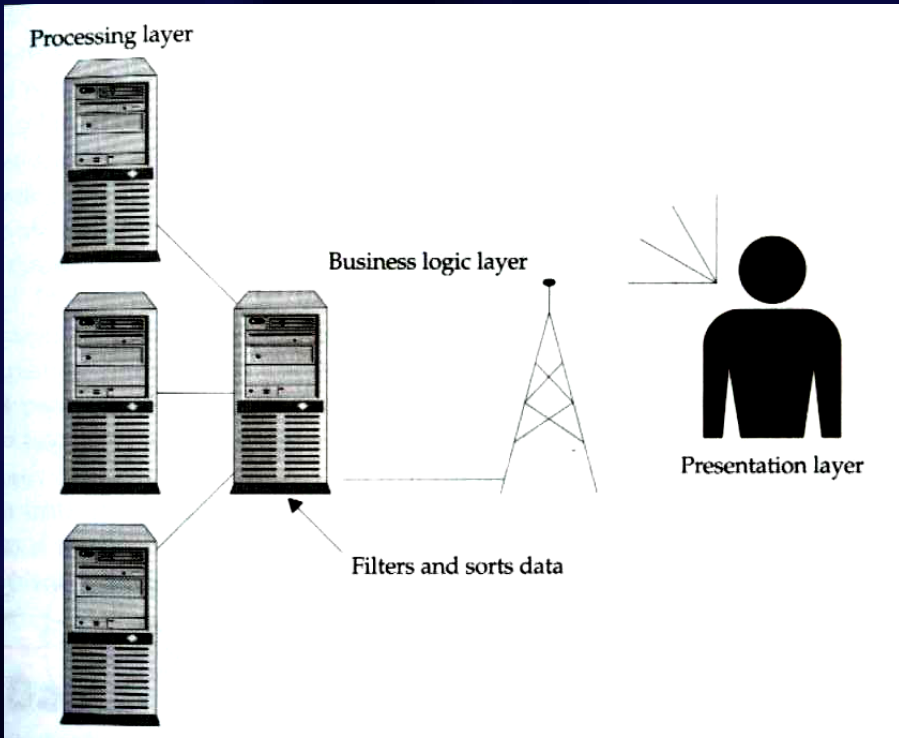
- Salvar e executar código a partir de um servidor Web
  - A vantagem aqui é eliminar problemas associados com múltiplas versões de uma mesma aplicação
  - Exemplo:

```
midp –transient http://www.minhaempresa.com/aplicacao.jad
```

  - Ao invés de executar um arquivo JAD local, a opção –transient especifica que o arquivo JAD está localizado em um servidor *WEB*
  - Desta forma, o desenvolvedor precisa apenas atualizar uma cópia da aplicação, e os fornecedores precisam apenas deixar a versão mais atual no servidor *WEB*
  - Esta tecnologia é ideal para decodificadores de TV por assinatura

# Melhores Práticas de Programação J2ME

- Minimizar o tráfego de rede
  - Ao desenvolver uma aplicação para um dispositivo móvel portátil, é importante balancear quanto do processamento vai ocorrer no dispositivo e quanto vai ocorrer num servidor remoto
  - Coletar de uma única vez toda a informação do usuário que é requerida pelo processo



# Melhores Práticas de Programação J2ME

- Lidar com aspectos temporais
  - Aplicações J2ME que dependem da informação de data/hora podem sofrer de problemas
  - Em computadores de mesa e servidores a informação de tempo é local e depende do *time-zone*
  - Entretanto, o mesmo não é tão simples em um dispositivo móvel portátil que pode rapidamente se mover através de múltiplas *time-zones*
  - Outro problema é que há métodos diferentes para determinar a data/hora de um dispositivo móvel portátil: GPS ou a célula que o aparelho se encontra
  - A melhor prática é sempre armazenar o tempo em GMT

# Melhores Práticas de Programação J2ME

- Sincronização automática de dados
  - Dados que podem ser usados tanto num dispositivo portátil quanto numa aplicação executando num *PC*
  - Nesta situação, uma boa prática é fornecer um mecanismo de sincronização automática de dados
  - A atualização de dados pode ser incremental, em *batch* ou completa
  - A forma incremental requer que os dados sejam atualizados toda vez que uma modificação ocorrer
  - A forma *batch* permite atualizações sob demanda ou periódicas sejam executadas
  - A forma completa normalmente será utilizada em situações de emergência para recuperar dados quando as atualizações incremental ou em *batch* não estiverem sincronizadas

# Melhores Práticas de Programação J2ME

- Ter cuidado com a implementação de startAPP()
  - Declarações que devem ser executadas apenas durante a inicialização do MIDlet devem ser colocados no construtor e não no método startAPP(), que pode ser chamado várias vezes dentro do ciclo de vida de um MIDlet

# Melhores Práticas de Programação J2ME

- Considerações finais para o desenvolvimento de aplicações J2ME
  - Aplicações são tipicamente um único *thread*
  - Um aplicação roda por vez
  - Aplicações são dirigidas por eventos
  - Usuários mudam de uma aplicação para outra ao invés de terminar uma aplicação
  - Dispositivos móveis portáteis são utilizados intermitantemente
  - Aplicações utilizam múltiplas janelas, cada qual num tempo diferente
  - A média de uso de uma aplicação para dispositivo móvel portátil é de 2 minutos 30 vezes por dia

# Melhores Práticas de Programação J2ME

- Considerações finais para o desenvolvimento de aplicações J2ME (continuação)
  - Aplicações devem realizar uma tarefa dentro de 2 minutos; caso contrário é provável que o usuário irá desligar o dispositivo
  - Limite a interação com o usuário de forma que ele precise teclar poucas seqüências de teclas, use uma aplicação no PC para a entrada de muitos dados e depois transfira para o dispositivo
  - Usuários querem uma resposta instantânea de uma aplicação
  - O processamento pesado pode ser feito num servidor ou computador de maior capacidade
  - Minimize/otimize as tarefas que envolvem consumo de energia, como comunicações, animações e sons
  - Reduza a comunicação de dados, já que em alguns casos o usuário irá pagar pelo tráfego

- Comandos, Itens e Processamento de Eventos
- Interface Gráfica

# Comandos Itens e Processamento de Eventos

- Visão geral das interfaces com o usuário em J2ME
  - Uma interface é um conjunto de rotinas que apresentam informação na tela, solicita ao usuário qual a tarefa a executar e então processa a tarefa
  - Exemplo: uma aplicação de email
  - É possível fazer uso de três tipos de interface: *Command*, *Form* ou *Canvas*
  - Uma interface do tipo *Command* é basicamente um botão que o usuário pressiona no dispositivo para desempenhar uma função específica
  - *Exit* é uma instância de *Command* que é associada com um botão Sair do teclado, que pode indicar saí da aplicação. Da mesma, forma, temos o comando *Help*

# Comandos Itens e Processamento de Eventos

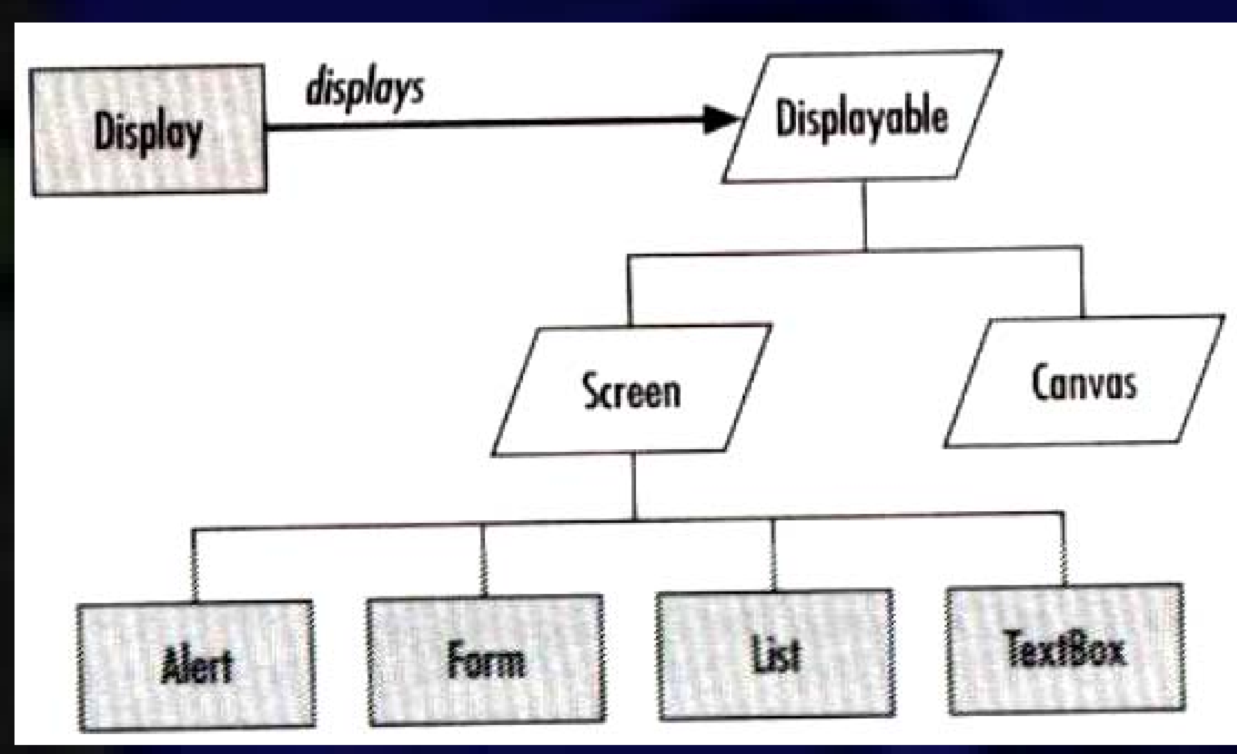
- Visão geral das interfaces com o usuário em J2ME
  - Uma interface com o usuário baseada em *Forms* irá receber vários outros itens de interface dentro dela, como caixas de texto, botões de rádio, listas, dentre outros
  - Um *Form* é simular a um *Form* HTML
  - Uma interface baseada em *Canvas* consiste de instâncias da classe *Canvas,* a partir das quais o desenvolvedor cria images e manipula pixels

# Comandos Itens e Processamento de Eventos

- Visão geral das interfaces com o usuário em J2ME
  - A classe *Display* representa a uma janela lógica no dispositivo, na qual um MIDlet pode apresentar sua interface com o usuário
  - Cada MIDlet tem acesso a uma única instância dessa classe
  - É possível obter uma referência para o display através do método: public static Display getDisplay(MIDlet midlet);
  - É possível também desenhar diretamente em uma classe derivada da classe abstrata *Displayable*
  - Um *Displayable* não é visível até que seja associada ao objeto *Display* do MIDlet através do método setCurrent()
  - Há 2 tipos pré-definidos de *Displayable*: *Screen* (alto nível) e *Canvas (*baixo nível)

# Comandos Itens e Processamento de Eventos

- Visão geral das interfaces com o usuário em J2ME



# Comandos Itens e Processamento de Eventos

- Determinando o atributo de cor de um dispositivo

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class CheckColor extends MIDlet implements CommandListener {
    private Display display;
    private Form form;
    private TextBox textbox;
    private Command exit;
    public CheckColor(){
        display = Display.getDisplay(this);
        exit = new Command("Exit",Command.SCREEN,1);
        String message=null;
        if(display.isColor()){
            message="Color display.";
        }
        else{
            message="No color display.";
        }
        textbox = new TextBox("Check Colors",message,17,0);
        textbox.addCommand(exit);
        textbox.setCommandListener(this);
    }
```

# Comandos Itens e Processamento de Eventos

- Determinando o atributo de cor de um dispositivo (continuação)

```
    public void startApp(){
        display.setCurrent(textbox);
    }
    public void pauseApp(){
    }
    public void destroyApp(boolean unconditional){
    }
    public void commandAction(Command command, Displayable displayable){
        if (command==exit){
            destroyApp(true);
            notifyDestroyed();
        }
    }
}
```

# Comandos Itens e Processamento de Eventos

- Exercícios práticos:

  1. Execute o programa anterior utilizando diferentes dispositivos e verifique se o mesmo funciona corretamente.

  2. Modifique o programa para imprimir a quantidade de cores ou tons de cinza (quando pertinente) disponíveis na tela gráfica do dispositivo.

# Comandos Itens e Processamento de Eventos

- A Classe Command

Exemplo:

Command cancel = new Command("Cancel",Command.CANCEL,1);

O 1º argumento indica o texto que irá aparecer na tela do dispositivo para identificar o comando

O 2º argumento indica o tipo de comando pré-definido, que irá ser mapeado automaticamente para uma tecla no dispositivo. Apesar disso, é a aplicação do usuário quem precisa dar o tratamento adequado ao comando

O 3º argumento indica a prioridade (quanto menor o valor, maior a prioridade). O gerenciador da aplicação pode usar a prioridade para determinar a ordem na qual os rótulos dos comandos irão aparecer na tela, mas o programador normalmente não tem qualquer controle sobre isto.

# Comandos Itens e Processamento de Eventos

- A Classe CommandListener

Exemplos:

```
public void commandAction(Command command, Displayable displayable){
  if (command=cancel){
     destroyApp(false);
     notifyDestroyed();
   }
}
```

Ou

```
public void commandAction(Command command, Displayable displayable){
  if (command.getCommandType()==Command.CANCEL){
     destroyApp(false);
     notifyDestroyed();
   }
}
```

# Comandos Itens e Processamento de Eventos

- A Classe Item e a Interface ItemListener

Items podem ser campos de texto, imagens, campos de dados, botões de rádio, caixas de opção, dentre outras características comuns em interfaces gráficas.

O usuário interage com sua aplicação através da mudança do status de instâncias derivadas da classe Item, exceto para instâncias da classe ImageItem e StringItem, que podem apenas serem mostradas na tela, mas não alteradas pelo usuário.

Uma mudança de estado uma instância da classe Item é processada através do método itemStateChanged() (definido na interface ItemStateListener), o qual é chamado automaticamente quando o estado muda.

O método só é invocado se o estado muda através da interação do usuário e não através de alteração causada pela aplicação.

# Comandos Itens e Processamento de Eventos

- A Classe Item e a Interface ItemListener, Exemplo:

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

public class RadioButtons extends MIDlet implements ItemStateListener, CommandListener {
        private Display display;
        private Form form;
        private Command exit;
        private Item selection;
        private ChoiceGroup radioButtons;
        private int defaultIndex;
        private int radioButtonsIndex;
        public RadioButtons() {
                display=Display.getDisplay(this);
                radioButtons=new ChoiceGroup("Select your color",Choice.EXCLUSIVE);
                radioButtons.append("Red",null);
                radioButtons.append("White",null);
                radioButtons.append("Blue",null);
                radioButtons.append("Green",null);
                defaultIndex=radioButtons.append("All",null);
                radioButtons.setSelectedIndex(defaultIndex,true);
                exit=new Command("Exit",Command.EXIT,1);
                form=new Form("");
                radioButtonsIndex=form.append(radioButtons);
                form.addCommand(exit);
                form.setCommandListener(this);
                form.setItemStateListener(this);
        }
```

# Comandos Itens e Processamento de Eventos

- A Classe Item e a Interface ItemListener, Exemplo (continuação):

```
public void startApp() throws MIDletStateChangeException {
     display.setCurrent(form);
          }
          public void pauseApp() {
          }

          public void destroyApp(boolean arg) {
          }

          public void commandAction(Command command, Displayable displayable){
                         if (command==exit){
                                        destroyApp(false);
                                        notifyDestroyed();
                         }
          }
          public void itemStateChanged(Item item){
                         if(item==radioButtons){
                                        StringItem msg = new StringItem("Your color is ",
                                                       radioButtons.getString(radioButtons.getSelectedIndex()));
                                        form.append(msg);
                         }
          }
}
```

# Comandos Itens e Processamento de Eventos

- Exercício prático:

Modifique o programa RadioButtons para aceitar múltiplas opções selecionadas de uma só vez (MULTIPLE ao invés de EXCLUSIVE). Remova a opção ALL da lista de escolhas do programa. Consulte a documentação da classe ChoiceGroup e encontre um método para obter o status de um grupo de escolha em que múltiplas opções possam estar selecionadas ao mesmo tempo. Em seguida, modifique o método itemStateChanged para imprimir na tela quais as opções que foram selecionadas.

# Comandos Itens e Processamento de Eventos

- Gerenciamento de Exceções
  - O gerenciador de aplicações do dispositivo chama os métodos startApp(), pauseApp() e destroyApp() para controlar o ciclo de vida de uma aplicação J2ME
  - Entretanto há situações em que interromper o fluxo normal de processamento de uma aplicação pode causar danos irreparáveis.
  - Por exemplo, um MIDlet pode estar no meio de uma sessão de comunicação ou salvando um dado persistente quando o método destroyApp() é chamado.

# Comandos Itens e Processamento de Eventos

- Gerenciamento de Exceções
  - É possível recuperar um certo controle da operação do MIDlet através do lançamento de uma exceção do tipo MIDletStateChangeException.
  - Isto é usado para temporariamente rejeitar uma solicitação tanto de término (destroyAPP()) quanto de re-início (startAPP()) da aplicação.
  - É comum colocar código que lança exceções desse tipo dentro do método destroyAPP(), uma vez que terminar o MIDlet durante algum processamento crítico pode ter um efeito fatal em uma comunicação ou em um dado armazenado.

# Comandos Itens e Processamento de Eventos

- Gerenciamento de Exceções – Exemplo

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class ThrowException extends MIDlet  implements CommandListener{
private Display display;
private Form form;
private Command exit;
private boolean isSafeToQuit;
        public ThrowException() {
                isSafeToQuit=false;
                display=Display.getDisplay(this);
                exit = new Command("Exit",Command.SCREEN,1);
                form = new Form("ThrowException");
                form.addCommand(exit);
                form.setCommandListener(this);
        }
        protected void startApp(){
                display.setCurrent(form);
        }

        protected void pauseApp() {
        }
```

# Comandos Itens e Processamento de Eventos

- Gerenciamento de Exceções – Exemplo

```
protected void destroyApp(boolean unconditional) throws MIDletStateChangeException {
        if(unconditional==false){
                throw new MIDletStateChangeException();
        }
}

public void commandAction(Command command, Displayable displayable){
        try{
                if(isSafeToQuit==false){
                StringItem msg=new StringItem("Busy", "Please try again");
                form.append(msg);
                destroyApp(false);
                }
        else{
                destroyApp(true);
                notifyDestroyed();
        }
        }
        catch (MIDletStateChangeException exception){
                isSafeToQuit = true;
        }
}
}
```

# Interface Gráfica de Alto Nível

- ## Hierarquia de Classes

```
public class Display
          public abstract class Displayable
                    public abstract class Screen extends Displayable
                              public class Alert extends Screen
                              public class Form extends Screen
                              public class List extends Screen implements Choice
                              public abstract class Item
                                        public class ChoiceGroup extends Item implements Choice
                                        public class DateField extends Item
                                        public class TextField extends Item
                                        public class Gauge extends Item
                                        public class ImageItem extends Item
                                        public class StringItem extends Item
                              public class TextBox extends Screen
                    public abstract class Canvas extends Displayable
public class Command
public class Ticker
public class Graphics
public interface Choice
```

# Interface Gráfica de Alto Nível

- A classe Alert
  - Um alerta é uma caixa de diálogo mostrada pelo programa para indicar um erro potencial ou crítico tais como perda de comunicação com um computador remoto
  - Um alerta também pode ser usado para mostrar qualquer outro tipo de mensagem, mesmo se a mensagem não é relacionada com um erro
  - Um alerta não deve ser usado para receber entrada de um usuário, a não ser a seleção do botão OK para fechá-lo
  - Isto significa que objetos Displayable (como ChoiceGroup e TextBox) não podem ser usados dentro de uma caixa de alerta

# Interface Gráfica de Alto Nível

- A classe Alert
  - Por exemplo, uma alerta é a forma ideal de apresentar um lembrete na tela:

```
Alert = new Alert(“Falha”, “Perda na Comunicação”, null, null);
Display.setCurrent(alert);
```

  - O primeiro argumento é o título da caixa de diálogo, o segundo argumento é a mensagem mostrada na caixa de diálogo, o terceiro argumento é uma imagem que pode ser mostrada na caixa, e, finalmente, o quarto argumento representa do tipo de alerta

| Tipo | Descrição |
| --- | --- |
| ALARM | A solicitação foi atendida |
| CONFIRMATION | Um evento ou processamento foi concluído |
| ERROR | Um erro foi detectado |
| INFO | Um alerta não associado a erro ocorreu |
| WARNING | Um erro potencial pode ocorrer |

# Interface Gráfica de Alto Nível

- A classe Alert
  - Um alerta pode reagir de 2 formas dependendo do valor default do atributo timeout do alert. Uma forma seria o alerta permanecer visível até que o usuário pressione o botão de OK, e a outra seria ficar visível durante um intervalo de tempo em milisegundos
  - Exemplo:

```
Alert = new Alert(“Falha”, “Perda de Comunicação”, null,null);
alert.setTimeout(Alert.FOREVER);
Display.setCurrent(alert);

Ou

Alert = new Alert(“Falha”, “Perda de Comunicação”, null,null);
alert.setTimeout(2000);
Display.setCurrent(alert);
```

  - Ver exemplo mais completo: DisplayAlert.java

# Interface Gráfica de Alto Nível

- Criando e Manipulando uma instância da classe Gauge
  - A classe Gauge cria uma barra de progresso animada, a qual graficamente representa o status de um processo
  - Entretanto, é o programador quem precisa implementar código para atualizar a indicação de progresso de um Gauge
  - Ver exemplo1: GaugeNonInteractive.java
  - Ver exemplo2: GaugeInteractive.java

# Interface Gráfica de Alto Nível

- Entrada de dados textuais – classe TextField
  - A classe TextField é usada para capturar uma ou mais linhas de texto digitadas pelo usuário
  - Exemplo: textfield = new TextField(“First Name:”, “”, 30, TextField.ANY);

| Restrição do TextField | Descrição |
|---|---|
| CONSTRAINT_MASK | Usada para determinar o valor corrente da restrição |
| ANY | Qualquer caracter é aceito |
| EMAILADDR | Aceita apenas um endereço de email |
| NUMERIC | Apenas números positivos ou negativos |
| PASSWORD | Esconde a entrada |
| PHONENUMBER | Apenas número de telefone, normalmente depende da configuração do dispositivo |
| URL | Apenas caracteres correspondendo a uma URL válida |

# Interface Gráfica de Alto Nível

- Entrada de dados textuais – classe TextField

  – Ver exemplo: TextFieldCapture.java

  – Exercício: modificar a classe TextFieldCapture.java de tal forma que simule um login do usuário no sistema (as informações de usuários válidos e suas respectivas senhas serão guardadas em variáveis internas ao programa – atentar que esta não é a forma correta de fazer a autenticação, mas para efeito do exercício é aceita). O programa ficará num loop de autenticação. Caso o login e a password sejam reconhecidos, o programa deverá mostrar um alerta de sucesso durante 5 segundos e em seguida sair.

# Interface Gráfica de Alto Nível

- A classe ImageItem

  – Ver exemplo: ImmutableImage.java

  – Exercício: modificar a classe acima de tal forma que o nome da imagem a ser mostrada seja fornecido pelo teclado. Lembre-se de copiar os arquivos img1.png e img2.png para o diretório de seu projeto no eclipse. Ao concluir o programa, experimente fornecer um nome de imagem inexistente.

# Interface Gráfica de Baixo Nível

- A classe Canvas
  - Um canvas é organizado como uma matriz de pixels
  - As coordenadas x e y representam as colunas e linhas dessa matriz, respectivament
  - A coordenada (0,0) é localizada no canto superior esquerdo do display

# Interface Gráfica de Baixo Nível

- A classe Canvas
  - O tamanho do canvas depende do dispositivo uma vez que o tamanho do display é idêntico ao tamanho do canvas
  - É importante, como programador, verificar as dimensões do canvas do dispositivo antes de desenhar qualquer coisa na tela
  - O tamanho do canvas é medido em pixels e pode ser obtido a partir dos métodos getWidth() e getHeight da classe Canvas

# Interface Gráfica de Baixo Nível

- A classe Canvas
  - Os valores retornados por getWidth() e getHeight() podem ser usados para desenhar uma imagem numa dada localização, de tal forma que seja proporcional ao tamanho do canvas.
  - Os componentes usados para criar uma imagem em um canvas são desenhados quando o método paint() declarado na classe Displayable é chamado.

# Interface Gráfica de Baixo Nível

- O método paint()
  - paint() é um método abstrato usado tanto por instâncias e subclasses da classe Screen quando da classe Canvas
  - Entretanto, o desenvolvedor não precisa se preocupar com a implementação de paint() quando faz uso de interface de alto nível, pois já está definido nas subclasses da classe Screen
  - Só é necessário dar uma implementação para paint() em aplicações baseadas em Canvas

# Interface Gráfica de Baixo Nível

- O método paint()
  - O método paint() requer apenas um parâmetro que é uma referência para a instância da classe Graphics criada pela a aplicação
  - O seguinte exemplo, mostra como desenhar um retângulo iniciando nas coordenadas 12,6 (mais ao topo e à esquerda) e tendo dimensões de 40x20 pixels

```
protected void paint (Graphics graphics){
    graphics.drawRect(12,6,40,20);
}
```

# Interface Gráfica de Baixo Nível

- O método paint()
  - O método paint() não é chamado explicitamente, ao invés disso, ele é chamado automaticamente pelo setCurrent() assim que a aplicação se inicia
  - O método repaint() deve ser chamado toda vez que o canvas inteiro (nenhum parâmetro necessário) ou uma porção dele (quatro parâmetros necessários) deve ser redesenhado

# Interface Gráfica de Baixo Nível

- Os métodos showNotify() e hideNotify()
  - O gerenciador de aplicações do dispositivo chama o método showNotify() imediatamente antes de mostrar o vanvas. Pode-se implementar esse método com código que prepara o canvas para ser apresentado, como inicializando recursos, threads, variáveis etc.
  - O método hideNotify() é chamado pelo gerenciador de aplicações do dispositivo após o canvas ser removido da tela. O programador pode definir esse método para conter código que libera recursos que foram alocados no método showNotify(), o que pode incluir desativar threads e resetar os valores associados a variáveis

# Interface Gráfica de Baixo Nível

- Interações com o usuário
  - Há 2 técnicas para receber entrada do usuário em aplicações J2Me com interface de baixo nível
  - A primeira técnica consiste em criar uma ou mais instâncias da classe Command e associá-la à classe Canvas através do método addCommand()
  - A outra técnica consiste em usar componentes que geram eventos do usuário de baixo nível.
  - Esses componentes são os “key codes”, “game actions” e “pointers”

# Interface Gráfica de Baixo Nível

- Interações com o usuário
  - Um “key code” é um valor numérico enviado pelo dispositivo sob a detecção de uma tecla particular. Cada tecla é identificada por um “key code” único.
  - Um “game action” é uma tecla associada a controladores comuns de jogos, como as setas direcionais.
  - Um “pointer” é uma entrada recebida através de um dispositivo apontador, como uma tela sensível ao toque (touch screen) ou um mouse

# Interface Gráfica de Baixo Nível

- Interações com o usuário
  - Existem três métodos que são chamados quando um evento de tecla particular ocorre enquanto o seu MIDlet está executando: keyPressed(), keyReleased() e keyRepeated().
    - keyPressed() é chamado quando uma tecla é pressionada pelo usuário.
    - keyReleased() é chamado quando uma tecla é pressionada pelo usuário é liberada.
    - keyRepeated() é chamado quando o usuário mantém uma tecla pressionada causando uma repetição. Observação: nem todo dispositivo dá suporte à repetição de teclas, para saber se há suporte, chamar o método hasRepeatEvents().

# Interface Gráfica de Baixo Nível

- Interações com o usuário
  - Todos os 3 métodos anteriores requerem um parâmetro inteiro para representar o código de tecla pressionada.
  - Exemplo: KeyCodeExample.java
  - Exercício1: modifique o programa anterior para usar as teclas detectadas (2,4,6 e 8) para alterar o posicionamento na tela de um retângulo azul de dimensões 10x10 localizado inicialmente no centro da tela.
  - Implemente a facilidade de processar a repetição de teclas (note que este recurso só funcionará com o emulador da nokia)
  - Modifique o seu programa para, ao invés de usar o teclado numérico, usar as teclas associadas a ação de jogos (constantes Canvas.UP, Canvas.DOWN etc), para isso será necessário usuar o método getGameAction(key) para obter um código associado a teclas de jogo antes de compará-la com as constantes acima.

# Interface Gráfica de Baixo Nível

- Trabalhando com dispositivos apontadores
  - Esta classe de dispositivos inclui mice e painéis touch screen
  - Existem 3 tipos de eventos de apontador que um MIDlet precisa processar
    - Pressionar o dispositivo sobre a superfície sensitiva (tela, tablet etc)
    - Soltar o dispositivo (ao remover a pressão sobre a tela ou liberar um botão do mouse
    - Arrastar (transladar) o dispositivo enquanto pressionado

# Interface Gráfica de Baixo Nível

- Trabalhando com dispositivos apontadores
  - Os métodos responsáveis por processar os eventos acima são: pointerPressed(), pointerReleased() e pointerDragged()
  - Esses métodos precisam ser implementados pelo programador da aplicação, todos os 3 métodos requerem 2 parâmetros: as coordenadas x,y do dispositivo apontador

# Interface Gráfica de Baixo Nível

- Trabalhando com dispositivos apontadores
  - Ao usar um emulador (como o SUN wireless toolkit) é necessário habilitar o recurso de touch screen a fim de testar suas aplicações.
  - Para fazer isso defina como **true** o parâmetro *touch_screen* no arquivo de configurações do dispositivo emulado escolhido.
    - Exemplo:
      WTK21\wtklib\devices\DefaultColorPhone\DefaultColorPhone.properties

# Interface Gráfica de Baixo Nível

- Trabalhando com dispositivos apontadores
  - Discutir Exemplo: PointerExample.java
  - Exercício: Modifique o exemplo acima para, ao invés de desenhar linhas ao longo da trajetória do apontador, desenhar um círculo de raio 20 nas coordenadas do apontador. O círculo deve acompanhar o apontador ao longo da tela, porém sem deixar rastro.
  - Para desenhar o círculo, utilize o método fillArc() da classe Graphics.

# Interface Gráfica de Baixo Nível

- Mostrando imagens num Canvas
  - Ver Exemplo: MutableImageExample.java
  - Exercício: modifique o exemplo anterior para apresentar uma imagem imutável (lida de um arquivo) ao invés de uma image mutável.

# 3ª UNIDADE: ARMAZENAMENTO PERSISTENTE DE DADOS

- Sistema de Gerenciamento de Registros
- Conexão a Bancos de Dados

# Sistema de Gerenciamento de Registros

- Persistência pode ser definida como a retenção de dados durante a operação de um MIDlet, dados estes que podem ser recuperados em um momento posterior
- Persistência é comum a muitas aplicações JAVA escritas em J2SE e J2EE
- Entretanto, a forma como a persistência é mantida em uma aplicação J2ME é diferente em função dos recursos limitados nos dispositivos portáteis

# Sistema de Gerenciamento de Registros

- O Sistema de Gerenciamento de Registros (RMS) fornece uma espécie de combinação entre sistema de arquivos e gerenciador de banco de dados
- RMS habilita o programador a armazenar dados em colunas e linhas, de forma similar à organização de uma tabela num banco de dados

# Sistema de Gerenciamento de Registros

- Usando RMS, o programador pode implementar métodos para inserir, recuperar, buscar e classificar registros
- Apesar disso, RMS não fornece funcionalidades de um sistema gerenciador de banco de dados, nem tampouco é um banco de dados relacional, e, portanto, não possível interagir com os dados usando uma linguagem como SQL

# Sistema de Gerenciamento de Registros

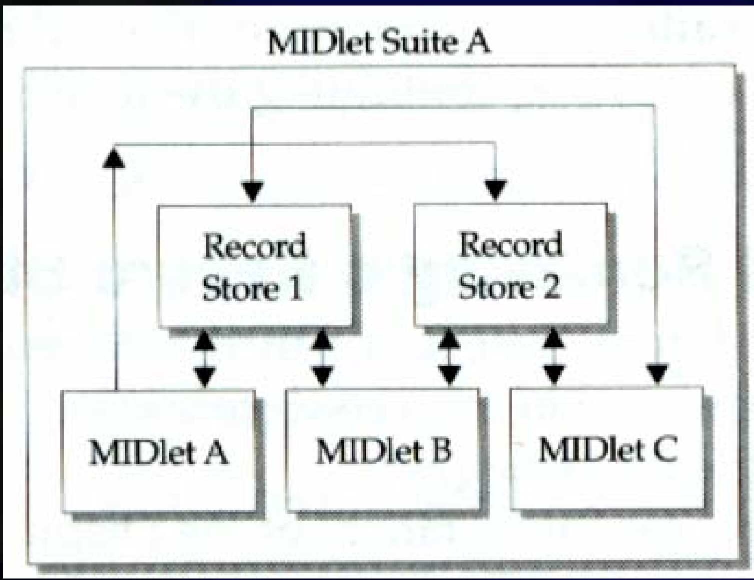
- RMS armazena informação em um *record store*, que é semelhante a um arquivo ou uma tabela de um banco de dados
- Um *record store* contém informação referenciada através de um nome único (*record* ID, ou chave primária)
- Contém uma coleção de registros organizados em linhas (registros) e colunas (campos)

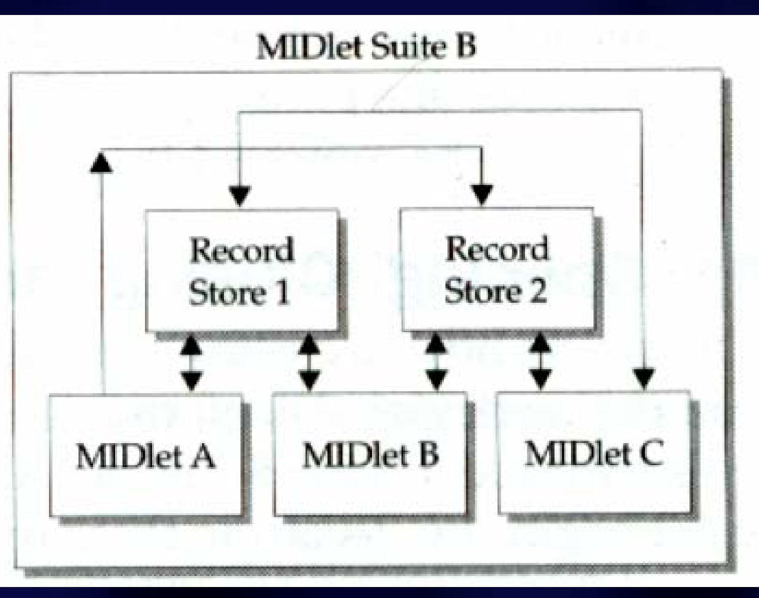
# Sistema de Gerenciamento de Registros

- Apesar de conceitualmente ser possível imaginar um *record store* como contendo linhas e colunas, tecnicamente só possui 2 colunas
- A primeira coluna é o *record* ID e a segunda é um array de bytes que contém os dados persistentes
- É possível criar múltiplos *record stores* em um MIDlet, desde que tenham nomes únicos

# Sistema de Gerenciamento de Registros

- Um *record store* pode ser compartilhado entre MIDlets de um mesmo MIDlet suite
- Duplicações de nomes de *record store* podem ocorrer desde que pertençam a diferentes MIDlet suites



# Sistema de Gerenciamento de Registros

- O método openRecordStore() pode ser usado para criar um novo registro ou abrir um registro existente
- Requer 2 parâmetros: o primeiro é um string contendo o nome do *record store*, o segundo é um boolean que, quando **true,** indica que um novo *record store* deve ser criado na hipótese dele não existir

# Sistema de Gerenciamento de Registros

- A fim de liberar os recursos usados durante a operação com um *record store*, deve-se chamar o método closeRecordStore()
- Após um closeRecordStore(), os dados vão estar salvos de forma persistente, porém inácessíveis ao MIDlet, até que o *record* store seja aberto novamente através de um openRecordStore()
- Para remover um *record store*, usa-se o método deleteRecordStore(), que recebe um string contendo o nome do registro que deve ser removido do dispositivo
- Ver exemplo: RecordStoreExample.java

# Sistema de Gerenciamento de Registros

- Há duas técnicas para leitura e gravação de registros num *record store*
  - Ler e gravar apenas 1 coluna de dados na forma de string
  - Ler e gravar múltiplas colunas de dados de tipos diferentes
- Inicialmente, vamos estudar a primeira técnica:

```
String output = "Test";
byte[] bytes=output.getBytes();
recordstore.addRecord(bytes,0, bytes.length);
```

# Sistema de Gerenciamento de Registros

- Ver Exemplo: WriteReadExample.java
- Exercício: Modificar o exemplo anterior para implementar 3 opções de comando: a primeira para receber um string digitado no teclado e adicioná-lo ao *record store*, a segunda para listar na tela o conteúdo de todos os registros armazenados e a terceira para sair do programa. Verifique se os registros são preservados ao reiniciar o MIDlet. Tente localizar na instalação do emulador onde os *record stores* ficam armazenados.

# Sistema de Gerenciamento de Registros

- Para gravar e ler registros contendo tipos de dados mistos

```
// Gravação
byte[] outputRecord;
String outputString = "First Record";
int outputInteger = 15;
boolean outputBoolean = true;
ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
DataOutputStream outputDataStream =
new DataOutputStream(outputStream);
outputDataStream.writeUTF(outputString);
outputDataStream.writeBoolean(outputBoolean);
outputDataStream.writeInt(outputInteger);
outputDataStream.flush();
outputRecord = outputStream.toByteArray();
recordstore.addRecord(outputRecord, 0, outputRecord.length);
outputStream.reset();
outputStream.close();
outputDataStream.close();
```

# Sistema de Gerenciamento de Registros

- Para gravar e ler registros contendo tipos de dados mistos

```
// Leitura
String inputString = null;
int inputInteger = 0;
boolean inputBoolean = false;
byte[] byteInputData = new byte[100];
ByteArrayInputStream inputStream = new ByteArrayInputStream(byteInputData);
DataInputStream inputDataStream =
new DataInputStream(inputStream);
for (int x = 1; x <= recordstore.getNumRecords(); x++)
{
recordstore.getRecord(x, byteInputData, 0);
inputString = inputDataStream.readUTF();
inputBoolean = inputDataStream.readBoolean();
inputInteger = inputDataStream.readInt();
inputStream.reset();
}
inputStream.close();
inputDataStream.close();
```

# Sistema de Gerenciamento de Registros

- Ver exemplo: WriteReadMixedDataTypesExample.java

# Sistema de Gerenciamento de Registros – Record Enumeration

- Um *record* store se parece mais com um arquivo simples do que com um sistema gerenciamento de banco de dados
- Faltam muitas características sofisticadas de um SGBD, como por exemplo, inexistência de consultas SQL para armazenar, recuperar, classificar ou filtrar registros

# Sistema de Gerenciamento de Registros – Record Enumeration

- Entretanto ainda é possível realizar buscas e ordenações em J2ME através da interface RecordEnumeration
- Uma enumeração fornece um mecanismo para navegar através dos dados, gerenciando como os dados são recuperados de um *record* store
- Mudanças no *record* store são refletidas quando  o seu conteúdo é iterado através da enumeração

# Sistema de Gerenciamento de Registros – Record Enumeration

- É possível obter uma enumeração através do método enumerateRecords(), o qual requer 3 parâmetros
- O primeiro parâmetro é o filtro de registros a ser usado para excluir da resposta alguns dos registros retornados
- O segundo parâmetro é uma referência para o comparador de registros
- O terceiro parâmetro é um boolean que indica se a enumeração deve ser automaticamente atualizada quando mudanças ocorrerem no *record store* associado

# Sistema de Gerenciamento de Registros – Record Enumeration

- Exemplo:

```
RecordEnumeration recordEnumeration = recordstore.enumerateRecords(null, null,
false);
```

- A partir da criação da enumeração, o próximo passo é utilizar métodos para interagir com os registros
- Uma das formas mais comuns de interação é percorrer sequencialmente todos os registros de uma enumeração
- O código seguinte ilustra como isso pode ser feito:

```
while ( recordEnumeration.hasNextElement())
{
//do something
}
```

# Sistema de Gerenciamento de Registros – Record Enumeration

- Para recuperar um registro de uma enumeração pode-se usar os métodos nextRecord() ou previousRecord()
- Quando a enumeração é criada, esta fica posicionada no primeiro registro
- Antes de se mover em uma dada direção da enumeração é imprescindível verificar se há registros na direção pretendida (métodos hasNextElement() ou hasPreviousElement())
- Exemplo:

```
String string = new String(recordEnumeration.nextRecord());
```

# Sistema de Gerenciamento de Registros – Record Enumeration

- Leitura de um tipos de dados heterogêneos dentro de uma RecordEnumeration
- Ver exemplo: MixedRecordEnumerationExample.java

# Sistema de Gerenciamento de Registros – Record Enumeration

- Classificando registros
  - Registros dentro de uma RecordEnumeration podem ser ordenados ou classificadors através de uma classe comparadora que implementa a interface RecordComparator
  - O método compare() precisa ser implementado nessa classe de tal forma que indique se o seu primeiro parâmetro deve preceder ou não o segundo parâmetro, ambos arrays de bytes

# Sistema de Gerenciamento de Registros – Record Enumeration

- Classificando registros
  - O método compare() retorna um valor de comparação que pode ser:
    - RecordComparator.EQUIVALENT
    - RecordComparator.PRECEDES ou
    - RecordComparator.FOLLOW
  - para indicar se o primeiro parâmetro é equivalente, deve vir antes ou depois do segundo parâmetro, respectivamente
  - Ver exemplo: SortExample.java

# Sistema de Gerenciamento de Registros – Record Enumeration

- Buscando registros
  - A interface RecordFilter é usada em buscas por registros
  - A interface obriga a implementação de 2 métodos: matches() e filterClose()
  - O método matches() contém a lógica necessária para determinar se uma coluna de dados atende ao critério de busca e retorna um boolean que é true toda vez que um casamento ocorre
  - O método filterClose() é usado, ao final da busca, para liberar recursos requisitados pela implementação da interface RecordFilter

# Sistema de Gerenciamento de Registros – Record Enumeration

- Buscando registros
  - Ver exemplo: SearchExample.java

# Sistema de Gerenciamento de Registros – Record Enumeration

- Monitorando automaticamente um *record store* através da interface RecordListener
  - Toda vez que mudanças são efetuadas no *record store* uma instância da interface é notificada
  - A classe que implementa RecordListener deve conter 3 métodos: recordAdded(), recordChanged(), and recordDeleted()
  - Todos esses métodos requerem 2 parâmetros: o primeiro contém uma referência para o *record store* que foi modificado e o segundo contém um inteiro para o recordID que foi adicionado, modificado ou removido

# Sistema de Gerenciamento de Registros – Record Enumeration

- Monitorando automaticamente um *record store* através da interface RecordListener

```
recordstore.addRecordListener( new MyRecordListener());
class MyRecordListener implements RecordListener
{
public void recordAdded(RecordStore recordstore, int recordid)
{
try
{
//do something
}
catch (Exception error)
{
//do something
}
}
```

# Sistema de Gerenciamento de Registros – Record Enumeration

- Monitorando automaticamente um *record store* através da interface RecordListener

```
public void recordDeleted(RecordStore recordstore, int recordid)
{
try
{
//do something
}
catch (Exception error)
{
//do something
}
}
public void recordChanged(RecordStore recordstore, int recordid)
{
try
{
//do something
}
catch (Exception error)
{
//do something
}
}
}
```

# Serialização de Objetos em CLDC
## (Connected Limited Device Configuration)

- CLDC não dá suporte a serialização de objetos nem a reflexão
- Isto significa que não há suporte para fazer a persistência de objetos em um fluxo de bytes para perfis (profiles) baseados em CLDC
- Aplicações que precisam de persistência de objetos deverão implementer seus próprios mecanismos

# Serialização de Objetos em CLDC
## (Connected Limited Device Configuration)

- Isto será necessário sempre que se deseje gravar objetos em uma facilidade de armazenamento persistente, como RMS, ou enviar objetos através de uma conexão de rede
- É difícil construir um mecanismo genérico de suporte à persistência, que funcione com objetos arbitrários

# Serialização de Objetos em CLDC
## (Connected Limited Device Configuration)

- Entretanto, escrever mecanismos de persistência específicos para uma determinada tarefa é muito simples e indiscutivelmente mais apropriado para as limitações da plataforma CLDC.
- É fácil implementar persistência através da cooperação da classe que precisa ser armazenada

# Serialização de Objetos em CLDC (Connected Limited Device Configuration)

- Num nível mais básico, consiste em definir uma interface como esta e fazer com que as classes com suporte a persistência a implementem:

```
import java.io.*;
/**
 * A simple interface for building persistent objects on
 * platforms where serialization is not available.
 */
public interface Persistent {
    /**
     * Called to persist an object.
     */
    byte[] persist() throws IOException;

    /**
     * Called to resurrect a persistent object.
     */
    void resurrect( byte[] data ) throws IOException;
}
```

# Serialização de Objetos em CLDC (Connected Limited Device Configuration)

- Considere uma classe simples descrevendo um empregado:

```
// Non-persistent version
public class Employee {
    private int      employeeID;
    private String   firstName;
    private String   lastName;
    private int      managerID;
    public Employee( int employeeID, String firstName,
                     String lastName, int managerID ){
        this.employeeID = employeeID;
        this.firstName = firstName;
        this.lastName = lastName;
        this.managerID = managerID;
    }
    public int getID() { return employeeID; }
    public String getFirstName() {
        return firstName != null ? firstName : "";
    }
    public String getLastName() {
        return lastName != null ? lastName : "";
    }
    public int getManagerID() { return managerID; }
}
```

# Serialização de Objetos em CLDC
## (Connected Limited Device Configuration)

- A versão persistente adiciona um construtor vazio e implementa a interface Persistent:

```
// Persistent version
import java.io.*;
public class Employee implements Persistent {
    private int       employeeID;
    private String   firstName;
    private String   lastName;
    private int       managerID;
    public Employee(){ }
    public Employee( int employeeID, String firstName,
                    String lastName, int managerID ){
        this.employeeID = employeeID;
        this.firstName = firstName;
        this.lastName = lastName;
        this.managerID = managerID;
    }
```

# Serialização de Objetos em CLDC (Connected Limited Device Configuration)

- Classe Employee persistente (continuação)

```
    public int getID() { return employeeID; }
    public String getFirstName() { return firstName != null ? firstName : ""; }
    public String getLastName() { return lastName != null ? lastName : ""; }
    public int getManagerID() { return managerID; }
    public String toString() {
        StringBuffer b = new StringBuffer();
        b.append( '{' );
        b.append( employeeID );
        b.append( ',' );
        b.append( firstName );
        b.append( ',' );
        b.append( lastName );
        b.append( ',' );
        b.append( managerID );
        b.append( '}' );
        return b.toString();
    }
```

# Serialização de Objetos em CLDC (Connected Limited Device Configuration)

- Classe Employee persistente (continuação)

```
public byte[] persist() throws IOException {
  ByteArrayOutputStream bout =  new ByteArrayOutputStream();
  DataOutputStream        dout = new DataOutputStream( bout );
  dout.writeInt( getID() );
  dout.writeUTF( getFirstName() );
  dout.writeUTF( getLastName() );
  dout.writeInt( getManagerID() );
  dout.flush();
  return bout.toByteArray();
}
```

# Serialização de Objetos em CLDC
## (Connected Limited Device Configuration)

- Classe Employee persistente (continuação)

```
public void resurrect( byte[] data )  throws IOException {
        ByteArrayInputStream bin = new ByteArrayInputStream( data );
        DataInputStream      din = new DataInputStream( bin );
        employeeID = din.readInt();
        firstName = din.readUTF();
        lastName = din.readUTF();
        managerID = din.readInt();
    }
}
```

# Serialização de Objetos em CLDC (Connected Limited Device Configuration)

- A persistência é na verdade conseguida através das classesw DataOutputStream e DataInputStream, as quais permitem facilmente ler e escrever tipos primitivos Java, além de Strings
- No exemplo da classe Employee, um objeto dessa classe pode ser preparado para armazenamento persistente da seguinte forma:

```
Employee emp = .....; // an employee instance
 try {
     byte[] persisted = emp.persist();
 }
 catch( java.io.IOException e ){
     // do something here
 }

// RMS storage of the “persisted” byte array here
```

# Serialização de Objetos em CLDC (Connected Limited Device Configuration)

- O seguinte trecho de código ilustra como recuperar um funcionário

```
byte[] persisted = ....; // presistence info (RMS recovery of byte array)
Employee emp = new Employee();

try {
    emp.resurrect( persisted );
}
catch( java.io.IOException e ){
    // do something here
}
```

# Serialização de Objetos em CLDC
## (Connected Limited Device Configuration)

- Note que a interface Persistent usa array de bytes ao invés de streams
- Em aplicações MIDP é comum implementar a persistência através de RMS, que tem uma API baseada em arrays, logo a interface Persistence é apropriada a esse tipo de armazenamento
- A persistência fica mais complicada quando os objetos a serem serializados contém referências para outros objetos persistentes

# Serialização de Objetos em CLDC
## (Connected Limited Device Configuration)

- O exemplo anterior evita armazenar uma referência para o empregador (Manager, que também é um empregado) na classe empregado (Employee)
- Isto iria complicar a persistência pois ao invés de tratar um único objeto, seria necessário tratar objetos com referências cruzadas ou cíclicas
- A melhor abordagem é evitar esses problemas, tornando os objetos persistentes completamente auto-contidos, e usar chaves (identificadores únicos) para fazer a ligação entre objetos

# Serialização de Objetos em CLDC
## (Connected Limited Device Configuration)

- Classes que não implementam a interface Persistent podem serem também “persistidas”, mas apenas se elas expuserem informação suficiente em sua interface pública
- A classe java.util.Vector, por examplo, pode receber suporte à persistência através de métodos estáticos definidos numa classe auxiliar
- Ver exemplo: VectorHelper.java

# Serialização de Objetos em CLDC (Connected Limited Device Configuration)

- Note que a classe VectorHelper apenas lida com vetores cujos elementos são do tipo Integer ou String, ou objetos que implementam a interface Persistent
- A classe VectorHelper pode ser facilmente estendida para acomodar outros tipos de dados básicos como Long e Boolean
- A classe é razoavelmente genérica, assim ela armazena a informação de tipo como parte dos dados de persistência a fim de corretamente recriar o conteúdo do vetor
- A maioria dos vetores tipicamente armazenam elementos de um mesmo tipo, neste caso, pode ser interessante remover toda a informação extra e tratar o vetor como simplesmente um array de objetos

# Serialização de Objetos em CLDC (Connected Limited Device Configuration)

- Ver exemplo: PersistentTest.java
  - A primeira vez que o MIDlet é executado, ele salva os objetos num *RecordStore*
  - Nas próximas execuções, o MIDlet lê os objetos e apresenta seus valores
- Exercício: Modificar o programa anterior para incluir uma opção (comando) de exclusão do *RecordStore* criado

# Conexão a Bancos de Dados

- Um aplicação J2ME (executando tanto num dispositivo CLCD quanto CDC) podem armazenar e recuperar dados localmente utilizando RMS
- Aplicações J2ME que executam em dispositivos CDC também são capazes de utilizar um Sistema de Gerenciamento de Banco de Dados (SGBD)
- O SGBD é tipicamente localizado num servidor conectado ao dispositivo através de uma rede, apesar de alguns dispositivos CDC poderem também acessar um SGBD localmente

# Conexão a Bancos de Dados

- As razões para um dispositivo CLCD não ter acesso a conexões de banco de dados estão relacionadas principalmente à complexidade envolvida nesse tipo de conexão: requer-se muitos recursos de processamento e de memória dos dispositivos clientes.
- Dispositivos CLDC possuem entre160KB e 512KB de memória disponível e são alimentados por baterias
- Eles também utilizam conexões de rede sem fio pouco confiáveis e podem não ter uma interface gráfica com o usuário
- Esta classe de dispositivos engloba principalmente telefones celulares e PDAs, e usa uma Java Virtual Machine (KVM), que é uma versão reduzida de uma JVM

# Conexão a Bancos de Dados

- Por outro lado, dispositivos CDC usam uma arquitetura de 32 bits e possuem pelo menos 2MB de memória disponível, implementando uma JVM completamente funcional
- Dispositivos CDC incluem aparelhos domésticos, decodificadores de TV digital, sistemas de navegação, terminais inteligentes de vendas e *smartphones*

# Conexão a Bancos de Dados

- Um banco de dados é uma coleção de dados gerenciada por um SGBD
- Há uma grande variedade de SGBDs comerciais disponíveis no mercado atualmente: Oracle, DB2, Sybase, Microsoft Access, etc
- Há também muitos SGBDs distribuídos com licença de software livre, a exemplo de MySQL e PostgreSQL
- Uma aplicação J2ME executando num dispositivo CDC interage com um SGBD através de uma combinação de objetos de dados Java que são definidos na especificação Java Database Connection (JDBC) e utilizando uma linguagem de consulta SQL (Structured Query Language)
- A interface JDBC dá suporte ao link de comunicação com o SGBD enquanto que SQL é a linguagem utilizada para formular as mensagens (também conhecidas como consultas) que são enviadas ao SGBD para consultar e manipular dados no SGBD

# Conexão a Bancos de Dados

- Um driver JDBC é um tradutor que converte mensagens proprietárias do SGBD para mensagens de baixo-nível que são entendidas pela API JDBC e vice-versa
- Isto significa que programadores Java podem escrever código que faz uso de interfaces JDBC de alto-nível para interagir com um SGBD
- A interface JDBC converte uma rotina em mensagens de baixo-nível que são conformantes com o driver JDBC
- O driver então traduz a rotina em mensagens que são entendidas e processadas pelo SGBD

# Conexão a Bancos de Dados

- Os drivers JDBC são criados pelos fabricantes de SGBDs e precisam ser capazes de:
  - Abrir um canal de conexão entre o SGBD e a aplicação J2ME
  - Traduzir declarações SQL, que são enviadas pela aplicação J2ME, em mensagens que podem ser processadas pelo SGBD
  - Retornar dados seguindo a especificação JDBC
  - Retornar informações, como mensagens de erro, seguindo a especificação JDBC
  - Fornecer rotinas de gerenciamento de transações
  - Fechar a conexão entre o SGBD e a aplicação J2ME

# Conexão a Bancos de Dados

- A especificação de um driver JDBC pode ser classificada em 4 grupos:
  - Driver tipo 1 - JDBC to ODBC (Microsoft Open Database Connectivity) – traduz entre a especificação JDBC e a especificação ODBC - lento
  - Driver tipo 2 - Java/Native Code Driver – API específica para um SGBD particular
  - Driver JDBC tipo 3 – código SQL é traduzido em declarações JDBC que, por sua vez, são convertidos para o protocolo do SGBD
  - Driver JDBC tipo 4 – consultas SQL são traduzidas diretamente para o formato requerido pelo SGBD – mais rápido

# Conexão a Bancos de Dados

- A API JDBC API está contida em 2 pacotes:
  - O primeiro pacote, java.sql, que é parte de J2SE, contém as interfaces JDBC de base, que incluem as facilidades para conexão ao SGBD e interação com os dados armazenados
  - O outro pacote chama-se javax.sql, que é uma extensão de java.sql e está em J2ME. Uma parte do pacote javax.sql trata da interação com a interface de nomes e diretórios de Java (JNDI) e gerencia as conexões, entre outras características avançadas de JDBC

# Conexão a Bancos de Dados

- A interação de uma aplicação j2ME com um SGBD, utilizando uma conexão JDBC, envolve basicamente um conjunto de 5 rotinas:
  - Inicialização do driver JDBC
  - Conexão com o SGBD
  - Criação e execução de uma consulta
  - Processamento dos dados retornados pelo SGBD e
  - Fechamento da conexão com o SGBD

# Conexão a Bancos de Dados

- Leitura do driver JDBC

```
try {
  Class.forName( "sun.jdbc.odbc.JdbcOdbcDriver");
}
catch (ClassNotFoundException error) {
  System.err.println("Unable to load the JDBC/ODBC bridge." +
  error.getMessage());
  System.exit(1);
}
```

# Conexão a Bancos de Dados

- Conexão com o SGBD

```
String url = "jdbc:odbc:CustomerInformation";
String userID = "jim";
String password = "keogh";
Statement DataRequest;
Connection Db;
try {
  Class.forName( "sun.jdbc.odbc.JdbcOdbcDriver");
  Db = DriverManager.getConnection(url,userID,password);
}
catch (ClassNotFoundException error) {
  System.err.println("Unable to load the JDBC/ODBC bridge." + error);
  System.exit(1);
}
catch (SQLException error) {
  System.err.println("Cannot connect to the database." + error);
  System.exit(2);
}
```

# Conexão a Bancos de Dados

- Criação e execução de uma consulta SQL

```
Statement DataRequest;
ResultSet Results;
try {
  String query = "SELECT * FROM Customers";
  DataRequest = Database.createStatement();
  DataRequest = Db.createStatement();
  Results = DataRequest.executeQuery (query);
  DataRequest.close();
}
catch ( SQLException error ){
  System.err.println("SQL error." + error);
  System.exit(3);
}
```

# Conexão a Bancos de Dados

- Processamento dos dados retornados pelo SGBD

```
ResultSet Results;
String FirstName;
String LastName;
String printrow;
boolean Records = Results.next();
if (!Records ) {
  System.out.println( "No data returned");
  return;
}
else
{
  do {
    FirstName = Results.getString (FirstName) ;
    LastName = Results.getString (LastName) ;
    printrow = FirstName + " " + LastName;
    System.out.println(printrow);
  } while ( Results.next() );
}
```

# Conexão a Bancos de Dados

- Fechamento da conexão com o SGBD

```
Db.close();
```

# Arcabouço de Conexão em Rede

- Em MDP, fluxos de entrada/saída são o mecanismo primário disponível para aplicações lerem e escreverem fluxos de dados
- Tanto J2SE quanto J2ME possuem um pacote java.io que contém essas classes
- Adicionalmente, MIDP define o pacote javax.microedition.io, o qual dá suporte a rede e comunicações para aplicaçãoes MIDP
- Este pacote é equivalente ao pacote java.net em J2SE

# Arcabouço de Conexão em Rede

- Aplicações MIDP utilizam os tipos definidos em javax.microedition.io para criar e manipular vários tipos de conexão de rede
- Para se ler e escrever nessas conexões, utilizam-se tipos do pacote java.io, que contém um subconjunto de classes e interfaces da sua versão J2SE
- O arcabouço genérico para conexão MIDP define uma infraestrutura que abstrai ou esconde das aplicações os detalhes de mecanismos de rede específicos e suas implementações

# Arcabouço de Conexão em Rede

- Neste modelo genérico de conexão, uma aplicação faz uma requisição a um conector, que retorna uma conexão para um recurso-alvo na rede
- A fim de criar uma conexão, usam-se endereços num formato especial padronizado
- No modelo de conexão MIDP, o programador identifica o recurso e obtém uma conexão para ele em um único passo

# Arcabouço de Conexão em Rede

- O modelo de conexão J2ME contrasta com o modelo J2SE, no qual a aplicação deve criar dois objetos: um que representa o recurso de rede alvo e outro objeto como sendo o fluxo ou conexão para esse objeto
- Por exemplo, para acessar uma URL em J2SE, uma aplicação constrói um objeto da classe java.net.URL, o qual representa o recurso de rede (URL) real
- A partir desse objeto, a aplicação então explicitamente abre uma conexão para o recurso URL, que leva à criação de um objeto URLConnection
- Nesse momento, a aplicação pode então objter um fluxo de entrada da conexão que fornece o conteúdo do recurso de rede

# Arcabouço de Conexão em Rede

- A classe URL sabe como acessar o recurso físico, o objeto de conexão, por outro lado, não sabe nada sobre como abrir ou localizar uma URL, mas ele sabe como interfacear como um objeto URL
- Em geral, o modelo J2SE requer que o programador construa um tipo de strem que seja compatível com o tipo de recurso sendo acessado, que pode ser uma URL, um socket de rede, um datagrama e assim por diante
- O modelo J2SE não abstrai esses detalhes da aplicação

# Arcabouço de Conexão em Rede

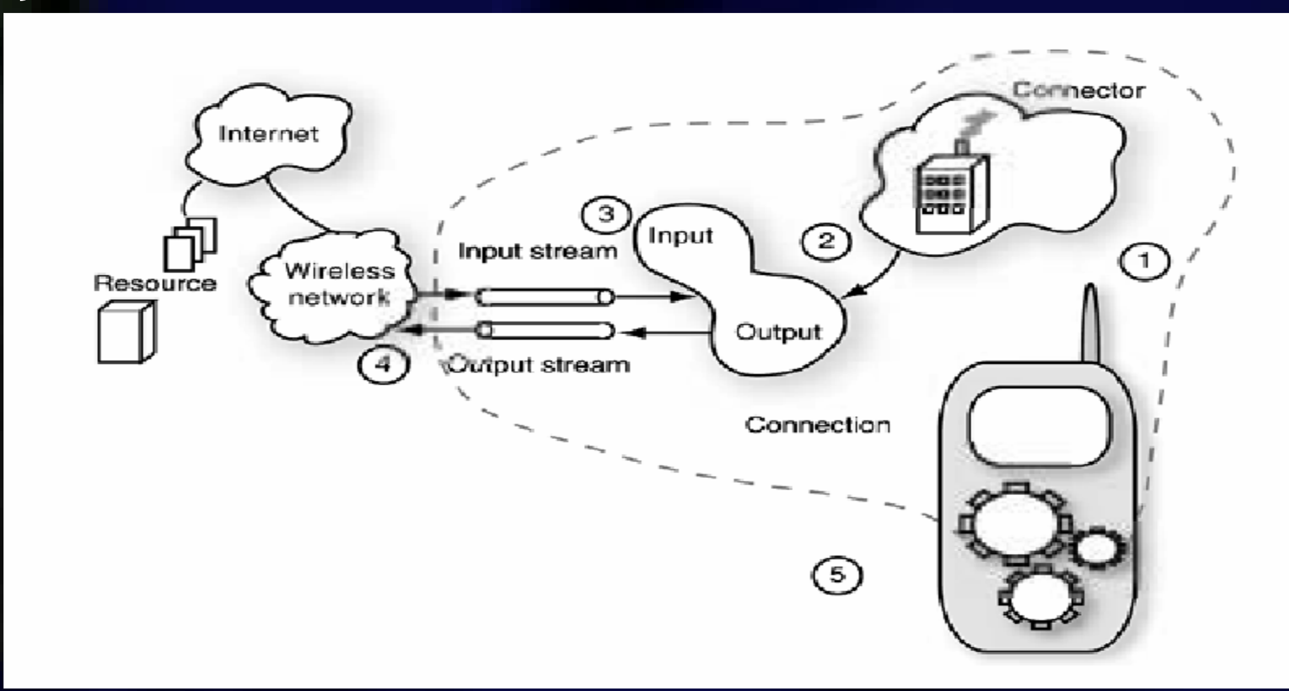
- No modelo MIDP, fluxos se comportam da mesma forma como no modelo J2SE, eles ainda não sabem nada sobre o recurso físico de rede real, simplesmente sabem como manipular o seu conteúdo
- O conector, entretanto, esconde da aplicação os detalhes de interfaceamento do stream como recurso real de rede
- Há duas vantagens principais para esse modelo genérico de conexão: primeiro ele esconde da aplicação os detalhes do estabelecimento de uma conexão; e segundo, esta abstração torna o modelo extensível
- Ao usar um mecanismo extensível para referenciar recursos de rede, implementações na plataforma MIDP pode ser melhoradas para dar suporte a protocolos adicionais, enquanto mantém um único mecanismo padronizado para acessar todo tipo de recursos . Além disso, a lógica do negócio permanece desacoplada dos mecanismos de rede

# Arcabouço de Conexão em Rede

- A fim de usar esse modelo genérico de conexão, aplicações MIDP especificam o recurso de rede que precisa ser acessado utilizando um identificador de recurso universal (URI), o qual segue uma sintaxe bem definida
- Um URI dá suporte à identificação de recursos na internet. Uma forma genérica de um URI é: <esquema>://<endereço>;<parametros>
- O esquema representa o protocolo a ser usado na conexão (file , datagram , socket , serversocket , http , ftp, dentre outros)

# Arcabouço de Conexão em Rede

1) A aplicação requisita a classe Connector para abrir e retornar uma conexão para um recurso de rede
2) O método Connector.open() analisa a URI e retorna um objeto representando a conexão, esse objeto mantém referências para fluxos de saída e de entrada para a rede
3) A aplicação obtém o objeto de conexão InputStream ou OutputStream
4) A aplicação lê a partir do InputStream ou escreve no OutputStream como parte de seu processamento
5) A aplicação, ao seu término, fecha a conexão



# Arcabouço de Conexão em Rede

- Ver exemplos:
  - HttpExample.java
  - LoadImageFromWeb.java
- Exercício: Adaptar os exemplos anteriores para construir uma aplicação que dá 3 opções ao usuário:

  1) Fornecer um endereço web (http) para ser visitado;
  2) Visualizar na tela do dispositivo todo o código fonte html do endereço visitado. Note que se o código for grande, serão necessárias várias telas de visualização, neste caso o programa deverá dar opções para o usuário navegar pelas telas (próxima, anterior), mais uma opção para sair da visualização;
  3) Visualizar apenas as imagens .png encontradas no endereço (utilize o google para encontrar páginas com arquivos png). Para isso, você deve localizar as linhas que contém o tag <img src=“imagem" …>, isolar o nome e a extensão da imagem, determinar se é .png, e, em seguida, criar a imagem para visualização na tela. Opcionalmente, você pode tentar redimensionar as imagens para o tamanho da tela do dispositivo (ou seja seria mostrada 1 imagem por tela). As observações sobre navegação do item 2) também se aplicam aqui.

# J2ME e BlueTooth

- Há diferentes formas de comunicação entre dispositivos: cabos de par trançado, coaxial, fibra óptica, ondas de rádio, raios infravermelho, dentre outros
- Nesse contexto, Bluetooth é uma nova modalidade de conexão para dispositivos que estão próximos entre si.
- Pode ser vista como uma tecnologia que substitui a necessidade de cabos, mas suas aplicações são inúmeras
- Bluetooth faz muito mais do que simplesmente substituir cabos, trata-se de uma tecnologia de rádio freqüência que utiliza a banda de 2.4 GHz (faixa reservada para aplicações industriais, medicas e científicas, como abertura de portões, intercomunicadores dentre outros)

# J2ME e BlueTooth

- Histórico
  - Bluetooth surgiu em 1994 como iniciativa da Ericsson, como uma forma de dispositivos se comunicarem entre si quando separados por pequenas distânticas (até 10 metros)
  - Em 1998, o consórcio Bluetooth Special Interest Group (BSIG), foi formado entre Ericsson, IBM, Intel, Nokia e Toshiba
  - Hoje em dia, mas de 2000 empresas fazem parte do BSIG
- Infravermelho
  - Infravermelho é barato e confiável, porém tem suas desvantagens: permite apenas comunicações em linha reta, permite apenas comunicação de um-para-um

# J2ME e BlueTooth

- 802.11b
  - Bluetooth e IEEE 802.11b são protocolos de comunicação sem fio operando na banda de 2.4GHz
  - Entretanto, não se deve considerar Bluetooth como um substituto para as redes locais sem fio 802.11
  - O protocolo 802.11b é projetado para conectar muitos dispositivos (como desktops, laptops, etc) ao mesmo tempo, com velocidades de até 11Mb/s e a distâncias de até 100m
  - Em contraste, Bluetooth é projetado para conectar poucos dispositivos de pequeno porte (como PDAs, celulares e periféricos) a velocidadesde a´te 1Mb/s distantes a não mais de 10m uns dos outros
  - 802.11b constitui uma LAN enquanto que Bluetooth constitui uma PAN

# J2ME e BlueTooth

- Características de Bluetooth
  - Sem fio e automático: os dispositivos se descobrem e estabelecem comunicação automaticamente
  - Tecnologia barata: hoje custa $20 para embutir num dispositivo, mas há previsões desse custo cair para $5
  - Como a banda ISM é não licenciada (porém regulamentada) permite o uso de dispositovos bluetooth em qualquer lugar sem a necessidade de permissões especiais
  - Permite lidar tanto com voz e dados simultaneamente
  - A comunicação é omnidirecional e pode atravessar objetos, não havendo necessidade de alinhamento entre os dispositivos
  - Utiliza *frequency hopping*, o que reduz o risco da interceptação de comunicações

# J2ME e BlueTooth

- Aplicações de Bluetooth
  - Transferência de arquivos
  - Formação de redes ad-hoc
  - Sincronização entre dispositivos
  - Conectividade entre periféricos
  - Kits para veículos
  - Pagamentos móveis

# J2ME e BlueTooth

- Topologia de Redes Bluetooth
  - Dispositivos bluethooth são organizados em grupos denominados *piconets*
  - Uma *piconet* consiste e um dispositivo master e até sete escravos ativos
  - Um master e um único escravo usam comunicação ponto-a-ponto; se há múltiplos escravos, uma comunicação ponto-multiponto é utilizada
  - Um dispositivo master é aquele que inicia a comunicação
  - Um dispositivo em uma *piconet* pode se comunicar com outro dispositivo de uma *piconet* diferente, formando assim uma *scatternet*

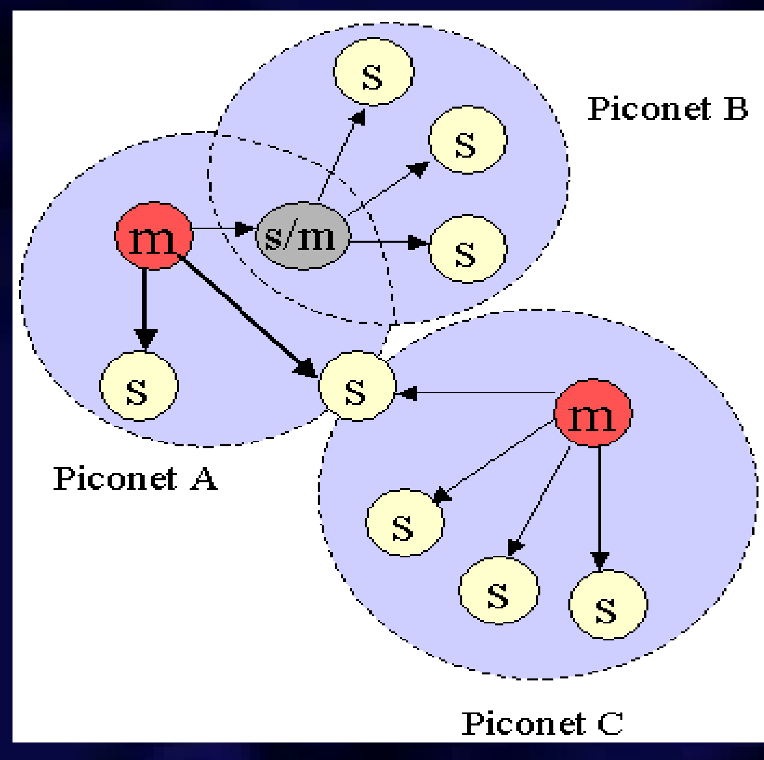
# J2ME e BlueTooth

- Topologia de Redes Bluetooth

•A duração normal de uma transmissão é um slot

•Um pacote pode ocupar até 5 *time slots*

•Para permitir comunicações full duplex, Bluetooth usa um esquema TDM (time-division multiplexing), no qual um master sempre usa um slot par quando transmite e um escravo usa um slot ímpar

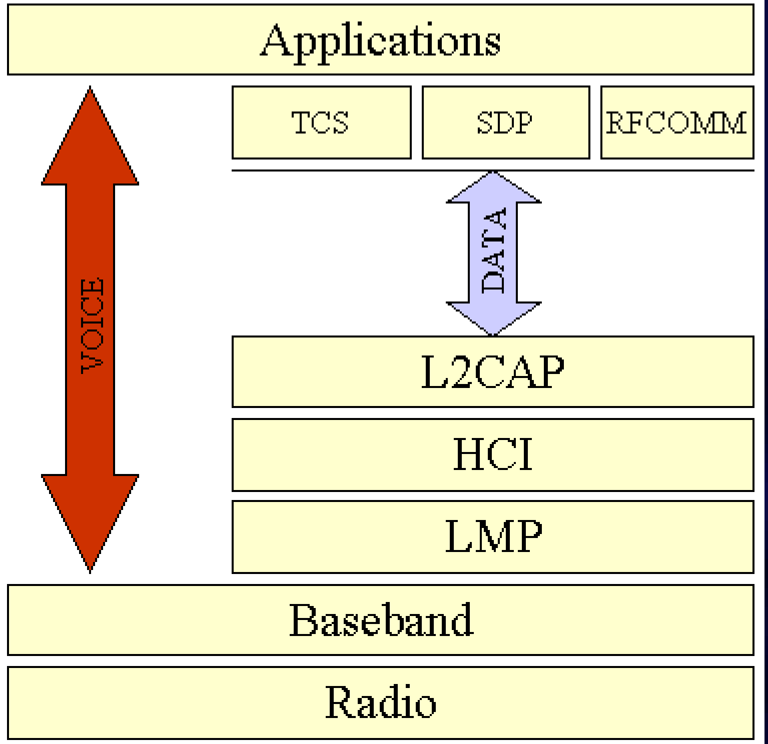


# J2ME e BlueTooth

- Modos de operação de baixo consumo
  - Bluetooth fornece suporte para três modos de operação de baixo consumo de energia: *sniff*, *hold* e *park,* em ordem decrescente de consumo de energia
  - No modo *sniff*, um escravo escuta a rede porém não desempenha um papel ativo na *piconet*
  - Um dispositivo em *hold* não transmite dados, mas seu clock continua a funcionar e escravos permanecem em sincronização com o mestre. O dispositivo não é um membro ativo da *piconet*, mas retém seu endereço de membro ativo
  - O modo *park* é como o modo *hold* na medida em que o escravo é sincronizado com o mestre, mas não é parte do tráfego. Nesse modo, entretanto, o escravo não mantém seu endereço ativo.

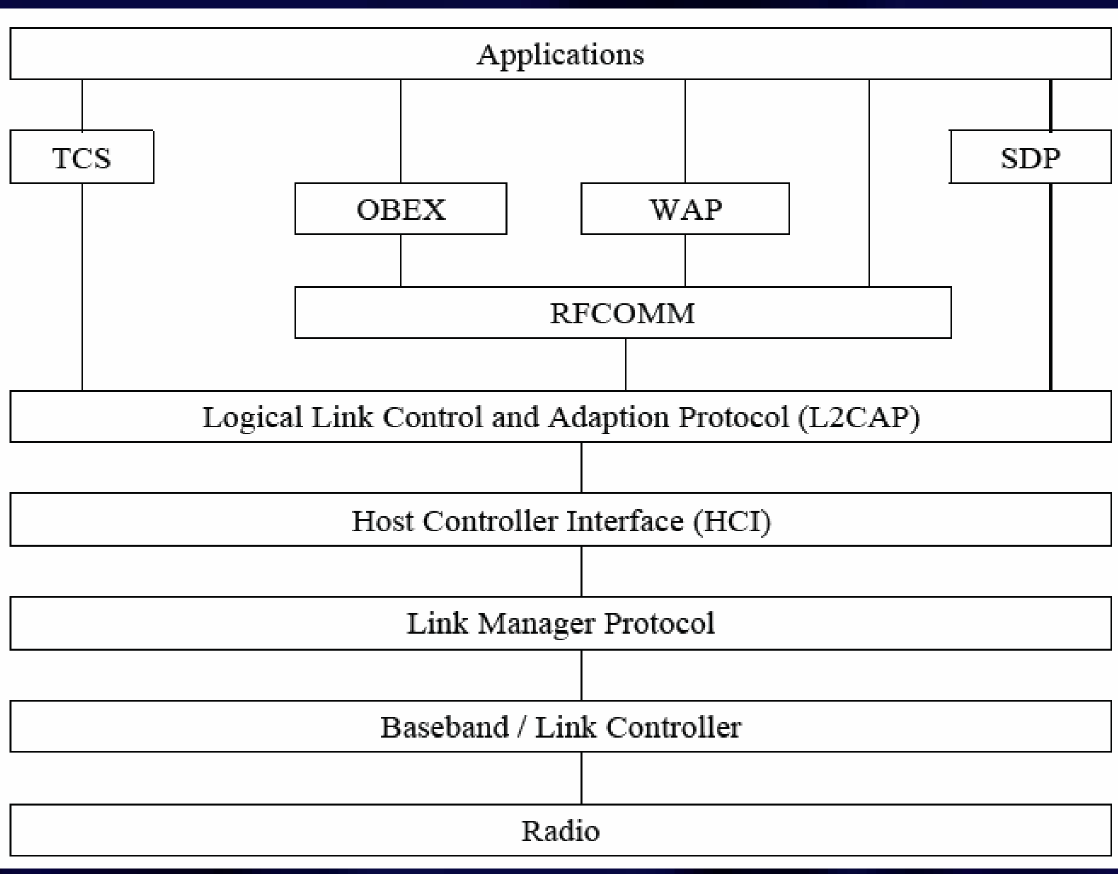
# J2ME e BlueTooth

- A pilha do protocolo Bluetooth
  - A especificação Bluetooth tem mais de 1500 páginas, contendo toda a informação necessária para garantir que diversos dispositivos que dão suporte à essa tecnologia possam se comunicar



# J2ME e BlueTooth

- A pilha do protocolo Bluetooth (outra figura)



# J2ME e BlueTooth

- A pilha do protocolo Bluetooth
  - A camada de *radio* representa a conexão sem fio pripriamente dita
    - Para evitar interferência com outros dispositivos que usam a banda ISM, a modulação é baseada em *frequency hopping*, que consiste em dividir a banda de 2.4GHz em 79 canais espaçados por 1MHz (de 2.402 a 2.480 GHz).
    - Esta camada pula de um canal para outro com uma frequencia de até 1600 vezes por segundo.
    - O alcance vai de 10cm a 10m, mas pode ser extendido para 100m, aumentando a potência de transmissão

# J2ME e BlueTooth

- A pilha do protocolo Bluetooth
  - A camada *baseband* é responsável por controlar e enviar pacotes de dados sobre o link de rádio
    - Fornece canais de transmissão tanto para dados quanto para voz
    - Esta camada mantém links do tipo SCO (Synchronous Connection-Oriented) para voz e do tipo ACO (Asynchronous Connectionless) para dados. Pacotes SCO nunca são retransmitidos, mas pacotes ACL são, para garantir integridade

# J2ME e BlueTooth

- A pilha do protocolo Bluetooth
  - Links SCO
    - são conexões ponto-a-ponto simétricas, nas quais os slots de tempo são reservados. Um escravo pode responder apenas no slot de tempo imediatamente posterior a uma transmissão SCO do master
    - Um master pode dar suporte a até três links SCO para um mesmo escravo ou para múltiplos escravos, e um único escravo pode suportar até dois links SCO de diferentes escravos

# J2ME e BlueTooth

- A pilha do protocolo Bluetooth
  - Links ACL
    - Diferentemente de links SCO, links ACL são estabelecidos usando slots que não foram reservados para links SCO
    - Suportam transmissões ponto-para-multiponto
    - Após uma transmissão ACL do master, apenas um escravo endereçado especificamente pode responder durante o próximo slot de tempo, se nenhum dispositivo não for endereçado, então a mensagem é tratada como broadcast

# J2ME e BlueTooth

- A pilha do protocolo Bluetooth
  - O *Link Manager Protocol – LMP* (Protocolo de gerenciamento do Link) usa os links estabelecidos pela camada anterior para estabelecer conexões e gerenciar *piconets*
  - Outras responsabilidades do *LMP* incluem serviços de autenticação e de segurança, além do monitoramento da qualidade do serviço

# J2ME e BlueTooth

- A pilha do protocolo Bluetooth
  - O *Host Controller Interface – HCI* (Controlador da Interface do Host) é a linha divisória entre software e hardware
  - A camada L2CAP e camadas superiores são componentes de software enquanto que as camadas LMP  e inferiores estão em hardware
  - HCI é a interface de driver para o barramento físico que conecta esses dois componentes

# J2ME e BlueTooth

- A pilha do protocolo Bluetooth
  - O *Logical Link Control and Adaptation Protocol – L2CAP* (Protocolo de Adaptação e de Controle do Link Lógico) recebe dados da aplicação e adapta esses dados para o formato Bluetooth
  - Além disso, parâmetros de qualidade do serviço (QoS) são trocados nessa camada
  - O *Host Controller Interface – HCI* (Controlador da Interface do Host) é a linha divisória entre software e hardware

# J2ME e BlueTooth

- Estabelecendo uma conexão
  - Quando um dispositivo não está conectado a uma *piconet*, ele está em modo *standby*
  - Neste modo, o dispositivo escuta mensagens a cada 1.28 segundos em 32 freqüências diferentes
  - Quando um dispositivo deseja estabelecer uma conexão com outro, este envia 16 mensagens de página (*page messages)* idênticas em 16 freqüências diferentes
  - Se o escravo não responder, o mestre retransmite a mensagem nas outras 16 freqüências
  - Se o mestre não conhece o endereço do escravo, este deve preceder a mensagem de página com uma mensagem de *inquire,* a qual requer uma resposta extra do escravo
  - Quando o escravo responde a uma mensagem de página, o mestre pode então começar a transmitir os dados ou voz

# J2ME e BlueTooth

- Estabelecendo uma conexão
  - Exemplo: um usuário de telefone celular com bluetooth deseja acessar seus emails no saguão de um aeroporto. Ao clicar no ícone da aplicação de email, o seguinte processamento é efetuado pela aplicação:
    - *Inquire*: em um novo ambiente, o dispositivo automaticamente inicia uma pesquisa para encontrar um ponto de acesso. Todos os pontos de acesso nas proximidades respondem com seus endereços, e o dispositivo escolhe um deles
    - *Page*: o procedimento de paginação sincroniza o dispositivo com seu ponto de acesso
    - Estabelecimento de um link: *LMP* estabelece um link com o ponto de acesso
    - Serviços de descoberta: *LMP* utiliza o protocolo de descoberta de serviços (*SDP*) para encontrar quais serviços estão disponíveis a partir do ponto de acesso. Suponhamos que o serviço de emails está disponível

# J2ME e BlueTooth

- Estabelecendo uma conexão
  - Exemplo (continuação)
    - Criar um canal *L2CAP*: *LMP* utiliza a informação obtida do *SDP* para criar um canal para o ponto de acesso. A aplicação pode utilizar este canal diretamente ou utilizar um protocolo, como *RFCOMM* (*Radio Frequency Communications Protocol*) que pode estar executando sobre *L2CAP*. O protocolo *RFCOMM* emula uma conexão serial
    - Criar um canal *RFCOMM*: dependendo das necessidades da aplicação, um canal *RFCOMM* ou outro pode ser criado sobre o canal L2CAP.

# J2ME e BlueTooth

- Estabelecendo uma conexão
  - Exemplo (continuação)
    - Autenticar: este é o único passo que requer a interação com o usuário. Se o ponto de acesso requer autenticação, este envia um pedido de autenticação e o usuário precisará informar a senha para acesso ao serviço. Na prática apenas uma chave gerada a partir da senha é enviada pelo link sem fio
    - *Log-in*: se o dispositivo usa o protocolo PPP sobre *RFCOMM* então o usuário poderá efetuar o login para a leitura de seus emails
    - Enviar e receber dados: neste ponto, o cliente de email e o ponto de acesso irão utilizar protocolos de rede padrão, como TCP/IP, para enviar e receber dados.

# J2ME e BlueTooth

- Perfis (*Profiles*) Bluetooth
  - O objetivo de um perfil é garantir interoperabilidade entre dispositivos e aplicações de diferentes fabricantes e vendedores
  - Um perfil define os papéis e capacidades de tipos de aplicações específicos
  - Dispositivos bluetooth só podem interagir se obedecerem a um perfil particular

# J2ME e BlueTooth

- Perfis (*Profiles*) Bluetooth
  - *Gegeric Access Profile* (Perfil de acesso genérico) define procedimentos de conexão, descoberta de dispositivos, e gerenciamento de links. Também define procedimentos relativos a segurança e formatação de parâmetros. Todos os dispositivos bluetooth precisar dar suporte a este perfil
  - *Service Discovery Application and Profile* (Perfil e aplicação de descoberta de serviços) define características e procedimentos para uma aplicação executando num dispositivo bluetooth descobrir serviços registrados em outros dispositivos, além de recuperar informação relacionada aos serviços

# J2ME e BlueTooth

- Perfis (*Profiles*) Bluetooth
  - *Serial Port Profile* (Perfil de porta serial) define os requerimentos para dispositivos bluetooth que precisam configurar conexões que emulam cabos seriais e usam o protocolo RFCOMM
  - *LAN Access Profile* (Perfil de acesso à rede local) define como dispositivos bluetooth podem acessar os serviços de uma LAN usando PPP, além de mostrar como mecanismos PPP podem ser usados para constituir uma rede de dispositivos bluetooth
  - *Synchronization Profile* (Perfil de sincronização) define os requerimentos da aplicação para dispositivos bluetooth que precisam sincronizar dados com um ou mais dispositivos

# J2ME e BlueTooth

- Bluetooth e Segurança
  - Segurança é fornecida de três formas: *frequency hopping* pseudo-randômica, autenticação e encriptação. Saltos entre diferentes frequências torna difícil a escuta da rede. A autenticação permite limitar a conectividade para dispositivos específicos. A encriptação usa chaves secretas para tornar tados inteligéveis apenas para agentes autorizados
  - Todos os dispositivos bluetooth precisam implementar o *Generic Access Profile*, o qual contém todos os protocolos bluetooth e define um modelo de segurança que inclui três modos: Modo 1, Modo 2 e Modo 3.

# J2ME e BlueTooth

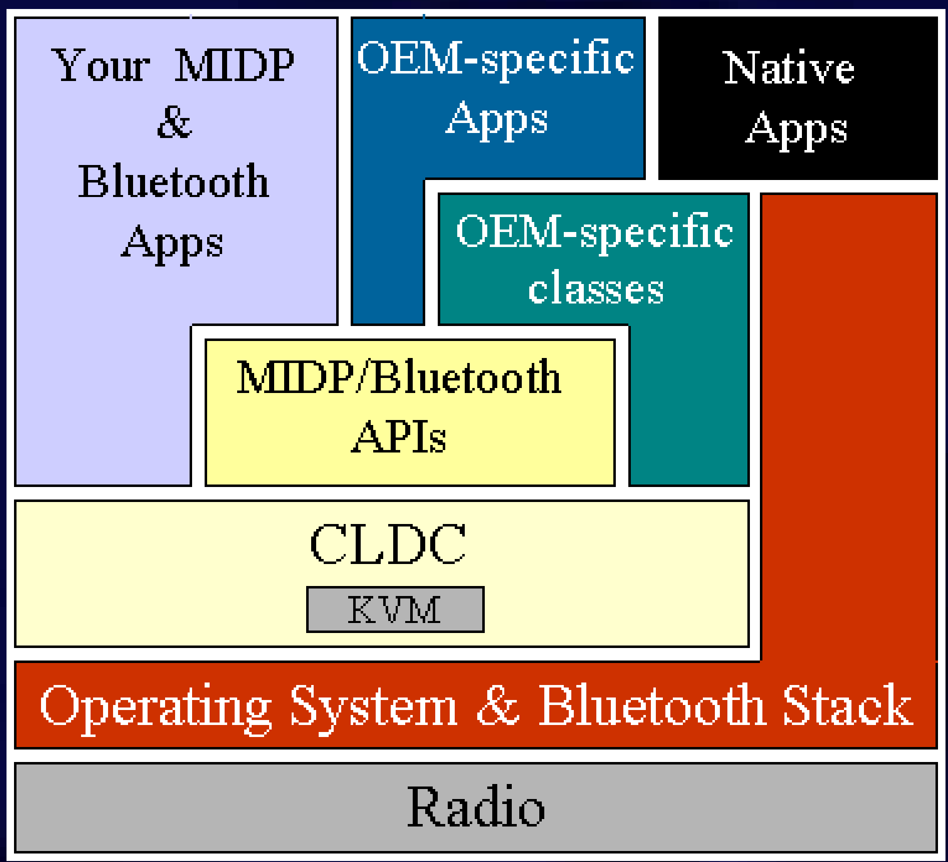
- Bluetooth e Segurança
  - Modo 1: é um modo de operação não segura, nenhum procedimento de segurança é iniciado.
  - Modo 2: é conhecido como securança imposta a nível de serviço (*service-level enforced security*). Quando um dispositivo opera nesse modo, os procedimentos de segurança só são iniciados quando canais de comunicação são estabelecidos
  - Modo 3: é conhecido como segurança imposta a nível de link (*link-level enforced security*). Nesse modo, os procedimentos de segurança são iniciados antes mesmo da configuração do link ter sido concluída

# J2ME e BlueTooth

- APIs Java para Bluetooth
  - JSR82 é o primeiro padrão aberto, não proprietário para o desenvolvimento de aplicações Bluetooth usando Java
  - JSR82 esconde a complexidade da pilha de protocolos bluetooth por trás de um conjunto de APIs java que permitem o programador focar no desenvolvimento de suas aplicações e não nos detalhes de baixo nível da especificação bluetooth
  - Projetadas para dispositivos com as seguintes características: mínimo de 512KB de memória, conectividade Bluetooth, J2ME CLDC

# J2ME e BlueTooth

- APIs Java para Bluetooth



# J2ME e BlueTooth

- Pacotes
  - javax.bluetooth: núcleo da API bluetooth
  - javax.obex: APIs para o protocolo de transferência de objetos (OBEX)
- Programação de Aplicações
  - Inicialização da pilha do protocolo bluetooth
    - Dependente do fabricante do dispositivo
    - No caso na Nokia, pode ser feito através de um diálogo de configuração do próprio SDK
    - Na solução Atinav para bluetooth, a pilha pode ser inicializada através de comandos como esses:

```
...
// set the port number
BCC.setPortNumber("COM1");
// set the baud rate
BCC.setBaudRate(50000);
// set the connectable mode
BCC.setConnectable(true);
// set the discovery mode to Limited Inquiry Access Code
BCC.setDiscoverable(DiscoveryAgent.LIAC);
...
```

# J2ME e BlueTooth

- Gerenciamento de dispositivos
  - Realizado através das classes LocalDevice e RemoteDevice
  - Exemplo 1, obtendo o dispositivo local:

```
...
// retrieve the local Bluetooth device object
LocalDevice local = LocalDevice.getLocalDevice();
// retrieve the Bluetooth address of the local device
String address = local.getBluetoothAddress();
// retrieve the name of the local Bluetooth device
String name = local.getFriendlyName();
...
```

# J2ME e BlueTooth

- Gerenciamento de dispositivos
  - Exemplo 2, obtendo o dispositivo remoto:

```
...
// retrieve the device that is at the other end of
// the Bluetooth Serial Port Profile connection,
// L2CAP connection, or OBEX over RFCOMM connection
RemoteDevice remote =
    RemoteDevice.getRemoteDevice(
        javax.microedition.io.Connection c);
// retrieve the Bluetooth address of the remote device
String remoteAddress = remote.getBluetoothAddress();
// retrieve the name of the remote Bluetooth device
String remoteName = local.getFriendlyName(true);
...
```

# J2ME e BlueTooth

- Descoberta de dispositivos
  - Conseguida através da classe DiscoveryAgent e da interface DiscoveryListener
  - Exemplo 1:

```
...
// retrieve the discovery agent
DiscoveryAgent agent = local.getDiscoveryAgent();
// place the device in inquiry mode
boolean complete = agent.startInquiry();
...
```

# J2ME e BlueTooth

- Descoberta de dispositivos
  - Exemplo 2:

```
...
// retrieve the discovery agent
DiscoveryAgent agent = local.getDiscoveryAgent();
// return an array of pre-known devices
RemoteDevice[] devices =
   agent.retrieveDevices(DiscoveryAgent.PREKNOWN);
...
```

# J2ME e BlueTooth

- Descoberta de dispositivos
  - Exemplo 3:

```
...
// retrieve the discovery agent
DiscoveryAgent agent = local.getDiscoveryAgent();
// return an array of devices found in a previous inquiry
RemoteDevice[] devices =
    agent.retrieveDevices(DiscoveryAgent.CACHED);
...
```

# J2ME e BlueTooth

- Descoberta de serviços
  - Considerando que um dispositivo local descobriu pelo menos um dispositivo remoto, ele pode iniciar a busca por serviços disponíveis
  - A classe DiscoveryAgente também fornece métodos para descobrir serviços em um dispositivo bluetooth servidor, bem como pode iniciar transações de descoberta de serviços

# J2ME e BlueTooth

- Registro de Serviços
  - Antes que um serviço possa ser descoberto, ele precisa ser primeiro registrato, ou anunciado num dispositivo bluetooth servidor
  - O servidor é responsável por:
    - Criar um registro que descreve o serviço oferecido
    - Adicionar esse registro ao SDDB (*Service Discovery DataBase*), de forma que se torne visível para potenciais clientes
    - Aceitar conexões de clientes
    - Atualizar registros no SDDB, quando os seus atributos mudam
    - Remover registros do SDDB quando estes não forem mais disponíveis

# J2ME e BlueTooth

- Registro de Serviços
  - Passo 1: para criar um novo serviço:

```
...
StreamConnectionNotifier service =
    (StreamConnectionNotifier) Connector.open("someURL");
```

  - Passo 2: para obter o registro do serviço:

```
ServiceRecord sr = local.getRecord(service);
```

  - Passo 3: indicar que o serviço está pronto para receber uma conexão de cliente (acceptAndOpen fica bloqueado até que um cliente se conecte):

```
StreamConnection connection =
    (StreamConnection) service.acceptAndOpen();
```

  - Quando o servidor está pronto para terminar, fecha a conexão e remove o registro do serviço:

```
service.close();
...
```

# J2ME e BlueTooth

- Comunicação
  - Para um dispositivo local usar um serviço num dispositivo remoto, os dois dispositivos precisam compartilhar um protocolo de comunicação comum
  - Assim, as aplicações podem acessar uma grande variedade de serviços Bluetooth
  - As APIs Java para bluetooth fornecem mecanismos que permitem conexões para qualquer serviço que usa RFCOMM, L2CAP ou OBEX como seu protocolo

# J2ME e BlueTooth

- Comunicação
  - Se um serviço utilizar algum outro protocolo (como TCP/IP, por exemplo) sobre algum dos protocolos de mais baixo nível, a aplicação poderá acessar o serviço, entretanto precisará implementar o protocolo adicional na aplicação, usando o *Generic Connection Framework* de CLDC
  - Devido ao fato do protocolo OBEX poder ser usado sobre diferentes meios de transmissão (com fio, infravermelho, radio bluetooth, dentre outros), a API OBEX é implementada separadamente, como pacote adicional

# J2ME e BlueTooth

- Protocolo RFCOMM
  - Emula uma conexão seria RS-232
  - O perfil de porta serial (SPP) facilita a comunicação entre dispositivos bluetooth fornecendo uma interface baseada em fluxos para o protocolo RFCOMM
  - Dois dispositivos podem compartilhar apenas uma sessão RFCOMM por vez
  - Até no máximo 60 conexões seriais lógicas podem ser multiplexadas numa dada sessão
  - Um único dispositivo Bluetooth pode ter no máximo 30 serviços RFCOMM ativos
  - Um dispositivo pode suportar apenas uma conexão cluente de cada vez para um dado serviço

# J2ME e BlueTooth

- Comunicação SPP entre servidor e cliente, lado do servidor

```
...
// assuming the service UID has been retrieved
String serviceURL =
    "btspp://localhost:"+serviceUID.toString());
// more explicitly:
String ServiceURL =
    "btspp://localhost:10203040607040A1B1C1DE100;name=SPP
        Server1";
try {
    // create a server connection
    StreamConnectionNotifier notifier =
      (StreamConnectionNotifier) Connector.open(serviceURL);
    // accept client connections
    StreamConnection connection = notifier.acceptAndOpen();
    // prepare to send/receive data
    byte buffer[] = new byte[100];
    String msg = "hello there, client";
    InputStream is = connection.openInputStream();
    OutputStream os = connection.openOutputStream();
    // send data to the client
    os.write(msg.getBytes());
    // read data from client
    is.read(buffer);
    connection.close();
} catch(IOException e) {
  e.printStackTrace();
}
...
```

# J2ME e BlueTooth

- Comunicação SPP entre servidor e cliente, lado do cliente

```
...
// (assuming we have the service record)
// use record to retrieve a connection URL
String url =
    record.getConnectionURL(
        record.NOAUTHENTICATE_NOENCRYPT, false);
// open a connection to the server
StreamConnection connection =
    (StreamConnection) Connector.open(url);
// Send/receive data
try {
    byte buffer[] = new byte[100];
    String msg = "hello there, server";
    InputStream is = connection.openInputStream();
    OutputStream os = connection.openOutputStream();
    // send data to the server
    os.write(msg.getBytes);
    // read data from the server
    is.read(buffer);
    connection.close();
} catch(IOException e) {
  e.printStackTrace();
}
...
```

# J2ME e BlueTooth

- Demos
  - Ver \WTK22\apps\BluetoothDemo
  - Executar 1 cliente e 1 servidor na mesma máquina usando o Wireless toolkit
  - Exercício: modificar para o cliente e servidor executarem em máquinas separadas

# J2ME e BlueTooth

- Exemplo de Descoberta e Conexão entre dispositivos bluetooth
  - Ver os arquivos que estão em http://www.dsc.ufcg.edu.br/~hmg/disciplinas/nokia/suporte/DiscoverConnect.zip
  - DiscoverConnect.java - MIDlet
  - BluetoothDiscovery.java - Executa busca de dispositivos e serviços além do registro de serviço
  - BluetoothConnection.java - Objeto de conexão que é retornado pela classe BluetoothDiscovery
  - ErrorScreen.java - Utilizado para mostrar mensagens de error
  - html/*.* - documentação javadoc

# J2ME e BlueTooth

- Exemplo de Descoberta e Conexão entre dispositivos bluetooth
  - O MIDlet DiscoverConnect é um exemplo de como usar dispositivos bluetooth e descoberta de dispositivos
  - Após um dispositivo com um serviço ter sido encontrado, uma conexão é estabelecida
  - Cada tecla é transmitida e mostrada no outro dispositivo até que a conexão seja interrompida

# J2ME e BlueTooth

- Exemplo de Descoberta e Conexão entre dispositivos bluetooth
  - Como no exemplo anterior, é necessário escolher o papel do dispositivo (cliente ou servidor)
  - O MIDlet é também capaz de conexões ponto-multiponto
  - Os arquivos BluetoothDiscovery.java e BluetoothConnection.java são independentes do MIDlet DiscoverConnect e, portanto, pode ser facilmente acoplados para funcionamento em outras aplicações

# Mobile Media API

- Trata-se de um pacote opcional que dá suporte a aplicações multimídia em dispositivos J2ME (CLDC e CDC)
- A especificação JSR 135 é bastante flexível e foi projetada para acomodar qualquer protocolo e formado, ou seja, não requer que a implementação MMAPI precise dar suporte a protocolos de transporte em particular (como HTTP, ou RTP), ou formatos específicos (como MP3, MIDI ou MPEG)

# Mobile Media API

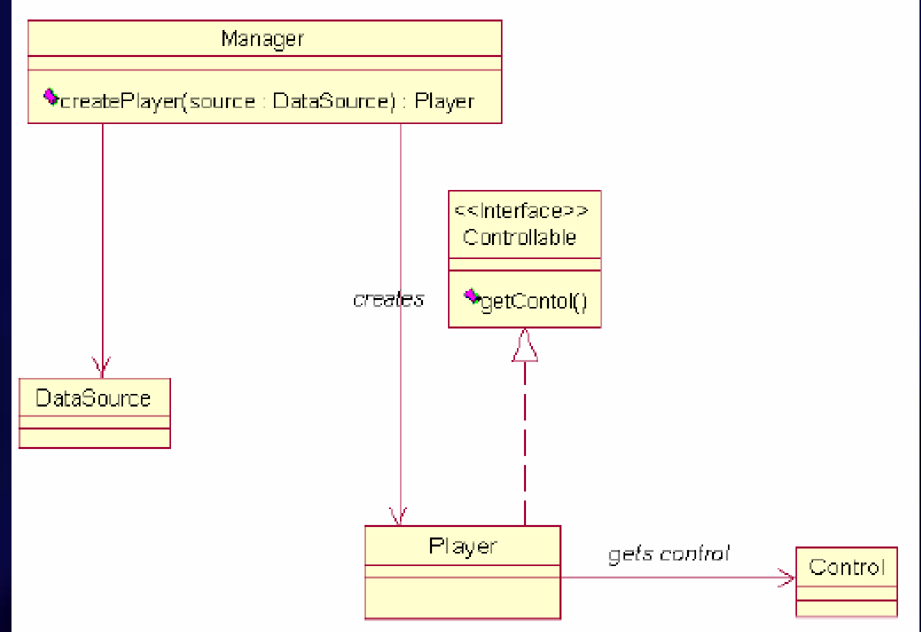
- Características
  - Geração de Tons, gravação e reprodução de mídia baseada em tempo
  - Extensível na medida em que novas funcionalidades (formatos de dados, por exemplo) podem ser adicionadas sem quebrar funcionalidades prévias
- Processamento multimídia
  - Gerenciamento do Protocolo: para ler dados de uma fonte, como um arquivo ou um servidor de streams, em um sistema de processamento de mídia
  - Gerenciamento de Conteúdo: analisar ou decodificar os dados da mídia e reproduzí-lo num dispositivo de saída como um auto-falante ou um display

# Mobile Media API

- Há duas classes de objetos de alto-nível para dar suporte  ao Processamento Multimídia:
  - DataSource, para encapsular o gerenciamento de protocolos escondendo os detalhes de como os dados são lidos da fonte de dados. Os métodos utilitários dessa classe habilitam o Player a manipular o conteúdo multimídia
  - Player, para ler dados do DataSource, efetuar seu processamento, e reproduzi-lo num dispositivo de saída. Esta classe possui métodos para controlar parâmetros de reprodução de tipos de mídia específicos

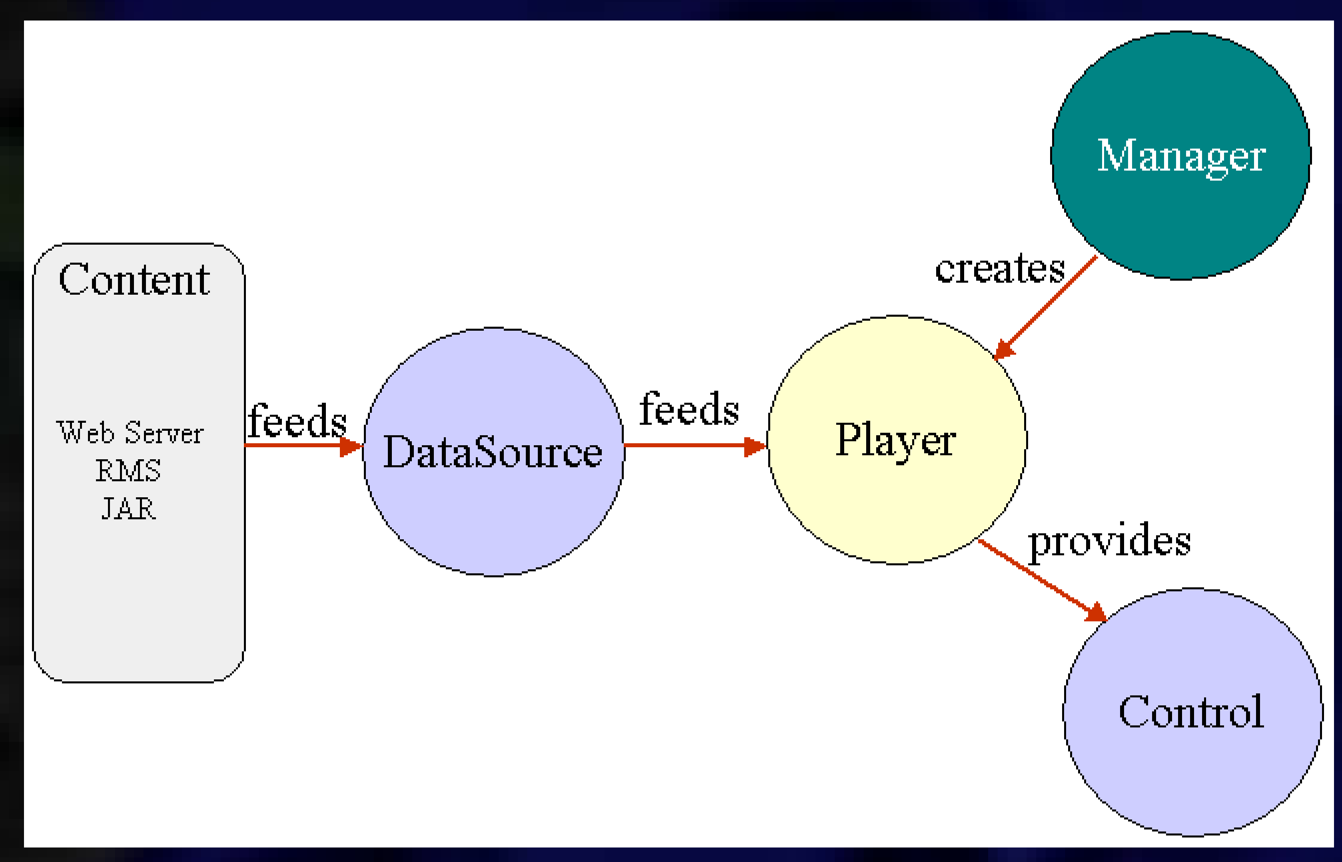
# Mobile Media API

- Há também uma terceira classe, Manager, que habilita a aplicação a criar Players a partir de DataSources, e também a partir de InputStreams
- Arquitetura geral de MMAPI



# Mobile Media API

- Outra forma de ver a Arquitetura



# Mobile Media API

- A classe Manager, possui o método createPlayer(), que é o ponto de entrada na API

```
...
Player player = Manager.createPlayer(String url);
...
```

- A URL especifica o protocolo e o conteúdo, no formato <protocolo>:<localização do conteúdo>
- O ciclo de vida de um Player inclui cinco estados distintos: UNREALIZED, REALIZED, PREFETCHED, STARTED, and CLOSED
- Os métodos realize(), prefetch(), start(), stop(), deallocate() e close() permitem realizar transições entre esses estados

# Mobile Media API

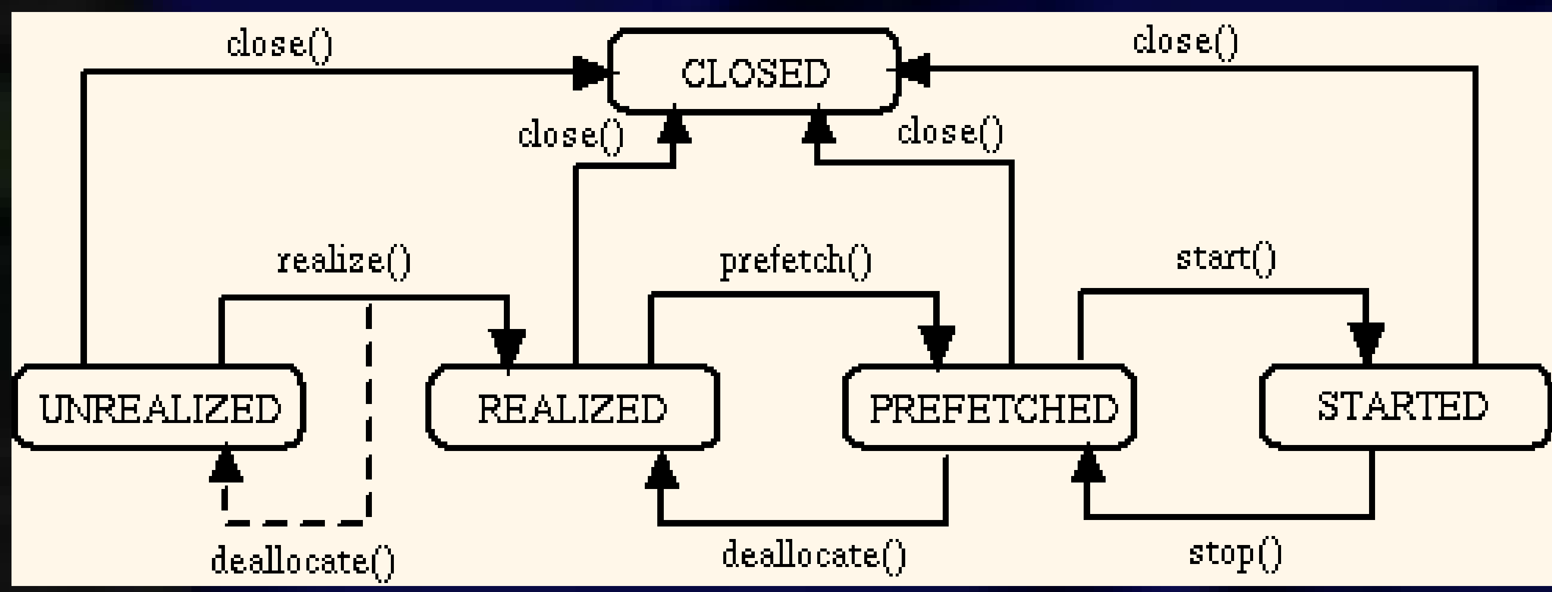
- Quando um Player é criado, este fica no estado UNREALIZED
- Chamando o método realize(), faz o Player mover para o estado REALIZED, quando a informação sobre como adquirir os dados multimídia é inicializada
- Chamando o método prefetch(), move o player oara o estado PREFETCHED, o qual estabelece conexões de rede para os dados e outras tarefas de inicialização

# Mobile Media API

- O método start() causa uma transição para o estado STARTED, momento em que o player pode processar os dados
- Quando o processamento é concluído o Player retorna para o estado PREFETCHED
- Finalmente, o método close() move o Player para o estado CLOSED

# Mobile Media API

- Diagrama de estados



# Mobile Media API

- Os pacotes MMAPI
  javax.microedition.media fornece algumas interfaces, uma exceção e a classe Manager, que é o ponto de acesso para recursos dependentes do sistema como Players para processamento multimidia
  javax.microedition.media.control define os tipos de controle específicos que podem ser usados em conjunção com um Player: VolumeControl, VideoControl, e outros
  javax.microedition.media.protocol define os protocolos para manipulação de controles customiuzzados. Por exemplo, esse pacote inclui a classe DataSource, que é uma abstração para manipuladores de controle de mídia

# Mobile Media API

- Geração de tons
  - Um tom é caracterizado por uma freqüência e uma duração. Este tipo de midia é importante para jogos e outras aplicações de audio, especialmente em dispositivos portáteis, nos quais pode ser a única modalidade multimidia disponível
  - O método Manager.playTone() gera tons

```
...
try {
   // play a tone for 4000 milliseconds at volume 100
   Manager.playTone()(ToneControl.C4, 4000, 100);
}
catch(MediaException me) {
}
...
```

# Mobile Media API

- Reprodução MP3 sem controles específicos

```
...
try {
   Player p = Manager.createPlayer
     ("http://server/somemusic.mp3");
   p.setLoopCount(5);
   p.start();
}
catch(IOException ioe) {
}
catch(MediaException e) {
}
...
```

# Mobile Media API

- Reprodução MP3 com algum controle (volume)

```
...
Player p;
VolumeControl vc;
try {
   p = Manager.createPlayer("http://server/somemusic.mp3");
   p.realize();
   // get volume control for player and set volume to max
   vc = (VolumeControl) p.getControl("VolumeControl");
   if(vc != null) {
      vc.setVolume(100);
   }
   // the player can start with the smallest latency
   p.prefetch();
   // non-blocking start
   p.start();
}
catch(IOException ioe) {
}
catch(MediaException e) {
}
...
```

# Mobile Media API

- Reproduzindo Mídia a partir de um RecordStore

```
...
RecordStore store;
int id;
// play back from a record store
try {
   InputStream is = new ByteArrayInputStream
     (store.getRecord(id));
   Player player = Manager.createPlayer(is, "audio/X-wav");
   p.start();
}
catch (IOException ioe) {
}
catch (MediaException me) {
}
...
```

# Mobile Media API

- Reproduzindo Mídia a partir de recursos encapsulados num pacote (JAR)

```
...
try {
   InputStream is =
     getClass().getResourceAsStream("audio.wav");
   Player player = Manager.createPlayer(is, "audio/X-wav");
   p.start();
}
catch(IOException ioe) {
}
catch(MediaException me) {
}
...
```

# Mobile Media API

- Reproduzindo Video

```
...
Player p;
VideoControl vc;
try {
   p = Manager.createPlayer("http://server/somemovie.mpg");
   p.realize();
   // get video control
   vc = (VideoControl) p.getControl("VideoControl");
   ....
   p.start();
}
catch(IOException ioe) {
}
catch(MediaException me) {
}
...
```

# Mobile Media API

- Exemplo:
  - Ver classe PlayerMIDlet na área de suporte na homepage do curso
  - Usar as seguintes urls para testar a classe

    http://java.sun.com/products/java-media/mma/media/test-wav.wav

    http://java.sun.com/products/java-media/mma/media/test-mpeg.mpg

# Mobile Media API

- Captura de Video
  - Obter um Player para captura de video

```
mPlayer = Manager.createPlayer("capture://video");
mPlayer.realize();
```

  - Mostrar as imagens da camera de video

```
mVideoControl = (VideoControl)
    mPlayer.getControl("VideoControl");
```

- Em um Canvas:

```
public CameraCanvas(SnapperMIDlet midlet,
    VideoControl videoControl) {
  int width = getWidth();
  int height = getHeight();

  mSnapperMIDlet = midlet;

  videoControl.initDisplayMode(
      VideoControl.USE_DIRECT_VIDEO, this);

  try {
    videoControl.setDisplayLocation(2, 2);
    videoControl.setDisplaySize(width - 4, height - 4);
  }
  catch (MediaException me) {
    try { videoControl.setDisplayFullScreen(true); }
    catch (MediaException me2) {}
  }
  videoControl.setVisible(true);
}
```

# Mobile Media API

- Captura de Video
  - Mostrar as imagens da camera de video
    - Em um Form:

```
Form form = new Form("Camera form");
Item item = (Item)mVideoControl.initDisplayMode(
    GUIControl.USE_GUI_PRIMITIVE, null);
form.append(item);
```

  - Capturar uma imagem

```
byte[] raw = mVideoControl.getSnapshot(null);
Image image = Image.createImage(raw, 0, raw.length);
```

# Mobile Media API

- Captura de Video
  - Criar um ícone a partir da imagem

```
private Image createThumbnail(Image image) {
  int sourceWidth = image.getWidth();
  int sourceHeight = image.getHeight();

  int thumbWidth = 64;
  int thumbHeight = -1;

  if (thumbHeight == -1)
    thumbHeight = thumbWidth * sourceHeight / sourceWidth;

  Image thumb = Image.createImage(thumbWidth, thumbHeight);
  Graphics g = thumb.getGraphics();

  for (int y = 0; y < thumbHeight; y++) {
    for (int x = 0; x < thumbWidth; x++) {
      g.setClip(x, y, 1, 1);
      int dx = x * sourceWidth / thumbWidth;
      int dy = y * sourceHeight / thumbHeight;
      g.drawImage(image, x - dx, y - dy,
          Graphics.LEFT | Graphics.TOP);
    }
  }

  Image immutableThumb = Image.createImage(thumb);

  return immutableThumb;
}
```

- Captura de Video
  - Ver exemplo http://www.dsc.ufcg.edu.br/~hmg/disciplinas/nokia/suporte/Snapper.zip

# Game API

- Game API é parte de MIDP 2.0, contendo apenas 5 classes no pacote javax.microedition.lcdui.game
- A nova classe GameCanvas torna possível desenhar na tela e responder a entrada do usuário
- Adicionalmente, há classes de gerenciamento de camadas (layers) para a construção eficiente de cenas complexas

# Game API

- A classe GameCanvas
  - GameCanvas é um tipo de Canvas com capacidades adicionais
  - Fornece métodos eficientes para desenho na tela e captura de eventos de entrada
  - Essas novas funcionalidades tornam possível encapsular toda a funcionalidade do jogo num único loop, sob o controle de uma única thread

# Game API

- A classe GameCanvas
  - Exemplo usando Canvas:

```
public void MicroTankCanvas extends Canvas
        implements Runnable {
  public void run() {
    while (true) {
      // Update the game state.
      repaint();
      // Delay one time step.
    }
  }
  public void paint(Graphics g) {
    // Painting code goes here.
  }
  protected void keyPressed(int keyCode) {
    // Respond to key presses here.
  }
}
```

# Game API

- A classe GameCanvas
  - Analisando o trecho de código anterior:
    - O método run(), que executa numa thread, atualiza o jogo uma única vez a cada passo
    - Tarefas típicas seriam atualizar a posição de um objeto na tela ou animar personagens ou veículos
    - Cada iteração do loop, o método repaint() é chamado para atualizar a tela
    - O sistema encaminha eventos de teclas para o método keyPressed(), o qual atualiza o estado do jogo apropriadamente

# Game API

- A classe GameCanvas
  - Problemas com o trecho de código anterior:
    - O programa está espalhado em threads diferenntes
    - O código está espalhado ao longo de 3 métodos diferentes
    - Quando o loop de animação principal (no método run()) chama repaint(), não existe uma forma de saber o que está ocorrendo com as outras partes da aplicação

# Game API

- A classe GameCanvas
  - Problemas com o trecho de código anterior (continuação):
    - Se o código no método keyPressed() está fazendo atualizações no estado do jogo ao mesmo tempo que a tela está sendo desenhada através do método paint(), problemas podem ocorrer
    - Se o desenho da tela levar mais que um único passo do loop em run(), a animação pode ficar com defeitos ou quebrada

# Game API

- A classe GameCanvas
  - Essa classe também contém um método para obter o estado corrente das teclas do dispositivo (*polling*)
  - Ao invés de esperar pelo sistema chamar o método keyPressed(), é possível determinar imediatamente quais as teclas são pressionadas através de chamadas ao método getKeyStates()

# Game API

- A classe GameCanvas
  - Exemplo usando GameCanvas:

```
public void MicroTankCanvas extends GameCanvas
        implements Runnable {
  public void run() {
    Graphics g = getGraphics();
    while (true) {
      // Update the game state.
      int keyState = getKeyStates();
      // Respond to key presses here.
      // Painting code goes here.
      flushGraphics();
      // Delay one time step.
    }
  }
}
```

- A classe GameCanvas
  - Ver exemplo: SimpleGameCanvas.zip na área de suporte da disciplina

# Game API

- Utilizando camadas (layers)
  - Jogos de ação típicos consistem de um background e vários personagens animados
  - Apesar de ser possível de programar diretamente este tipo de comportamento, na Game API é possível construir cenas utilizando camadas
  - Exemplo: ao desenhar uma cena contendo um carro num background de cidade, utilizando camadas é possível manipular o carro de forma independente do background

# Game API

- Utilizando camadas (layers)
  - A classe Layer é uma superclasse abstrata, definindo os atributos básicos de uma camada, como posição, tamanho, visibilidade
  - Cada subclasse de Layer precisa definir um método paint()
  - Há também duas subclasses concretas correlacionadas: TiledLayer e Sprite

# Game API

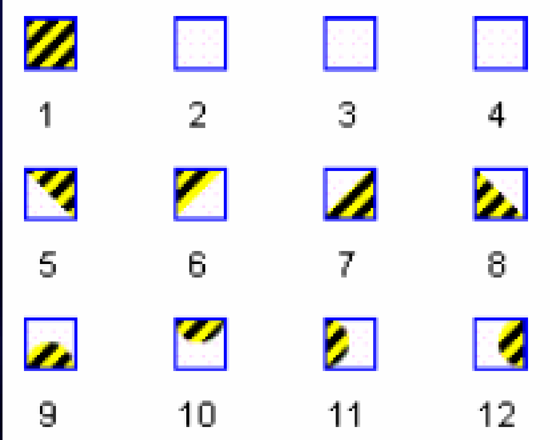
- Utilizando camadas (layers)
  - TiledLayer é útil na criação de imagens de backgrounds, é possível usar um pequeno conjunto de recortes de imagen para criar imagens maiores de forma eficiente
  - Sprite é uma camada animada. Fornecem-se os quadros (frames) de origem e adquire-se tltal controle sobre a animação

# Game API

- Utilizando camadas (layers)
  - A classe Sprite também oferece a habilidade de espelhar e rotacionar os quadros de origem em múltiplos de 90º
  - A classe LayerManager mantém um registro de todas as camadas na cena
  - Uma única chamada para o método paint() dessa classe é suficiente para redesenhar todas as camadas

# Game API

- Utilizando a classe TiledLayer
  - A imagem de origem fornece um conjunto de retalhos que podem ser arranjados para formar uma cena mais ampla
  - Exemplo: retalhos de 16x16 pixels



# Game API

- Utilizando a classe TiledLayer
  - Para criar uma TiledLayer formada por uma matriz de 10x10 retalhos, em que a dimensão de cada retalho é de 16x16 pixels:

```
Image image = Image.createimage("/board.png");
TiledLayer tiledLayer = new Tiled Layer(10,10,imagem16,16);
```

# Game API

- Utilizando a classe TiledLayer
  - Para associar um retalho para uma célula, basta chamar o método setCell(coluna, linha, retalho)
  - Um valor 0 (default) no terceiro parâmetro indica que a célula deve ser vazia

```
Image image = Image.createimage("/board.png");
TiledLayer tiledLayer = new Tiled Layer(10,10,imagem16,16);
```

# Game API

- Utilizando a classe TiledLayer
  - Exemplo:

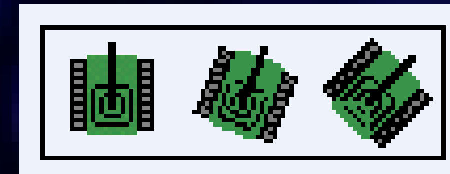
```
private TiledLayer createBoard() {
    Image image = null;
    try { image = Image.createImage("/board.png"); }
    catch (IOException ioe) { return null; }
    TiledLayer tiledLayer = new TiledLayer(10, 10, image, 16, 16);
    int[] map = {
            1, 1, 1, 1, 11, 0, 0, 0, 0, 0,
            0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
            0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
            0, 0, 0, 0, 9, 0, 0, 0, 0, 0,
            0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
            0, 0, 0, 7, 1, 0, 0, 0, 0, 0,
            1, 1, 1, 1, 6, 0, 0, 0, 0, 0,
            0, 0, 0, 0, 0, 0, 0, 7, 11, 0,
            0, 0, 0, 0, 0, 0, 7, 6, 0, 0,
            0, 0, 0, 0, 0, 7, 6, 0, 0, 0
    };
    for (int i = 0; i < map.length; i++) {
            int column = i % 10;
            int row = (i - column) / 10;
            tiledLayer.setCell(column, row, map[i]);
    }
    return tiledLayer;
}
```

# Game API

- Utilizando Sprites para animação de personagens
  - Enquanto que uma TiledLayer usa um conjunto de retalhos de imagem para criar uma imagem maior, um Sprite usa uma sequência de quadros para animação
  - Tudo o que é necessário é a imagem de origem e o tamanho de cada quadro (frame)

# Game API

- Utilizando Sprites para animação de personagens
  - Exemplo:



```
private MicroTankSprite createTank() {
   Image image = null;
   try { image = Image.createImage("/tank.png"); }
   catch (IOException ioe) { return null; }
   return new MicroTankSprite(image, 32, 32);
}
```

# Game API

- Utilizando Sprites para animação de personagens
  - A sequência default de apresentação dos frames é aquela presente na própria imagem
  - Para alterar esta sequência, utilizar o método setFrameSequence(), que recebe um array de inteiros
  - Para mover-se entre a sequência de frames, pode-se usar os métods prevFrame(), nextFrame() ou setFrame()
  - As mudanças só irão se materializar após o método paint() ter sido chamado

# Game API

- Utilizando Sprites para animação de personagens
  - A classe Sprite também fornece vários métodos colidesWith() para detectar colisões com outros Sprites, TiledLayers ou Imagens
  - É possível detectar colisões utilizando retângulos (método rápido, mas grosseiro), ou a nível de pixels (lento, mas preciso)
  - Ver exemplo: muTank

# Otimização de Aplicações J2ME

- Para realizar a otimização sugere-se utilizar algumas estratégias e ferramentas
- Considerar investir em Obfuscação de código (usando Proguard ou JAX, por exemplo) e em estratégias de coleta de lixo
- Deve-se também tomar cuidado para não decidir otimizar muito cedo no processo de desenvolvimento

# Otimização de Aplicações J2ME

- É possível utilizar uma estratégia de temporização muito simples para analisar trechos de código suspeitos de problemas de performance

- Exemplo:

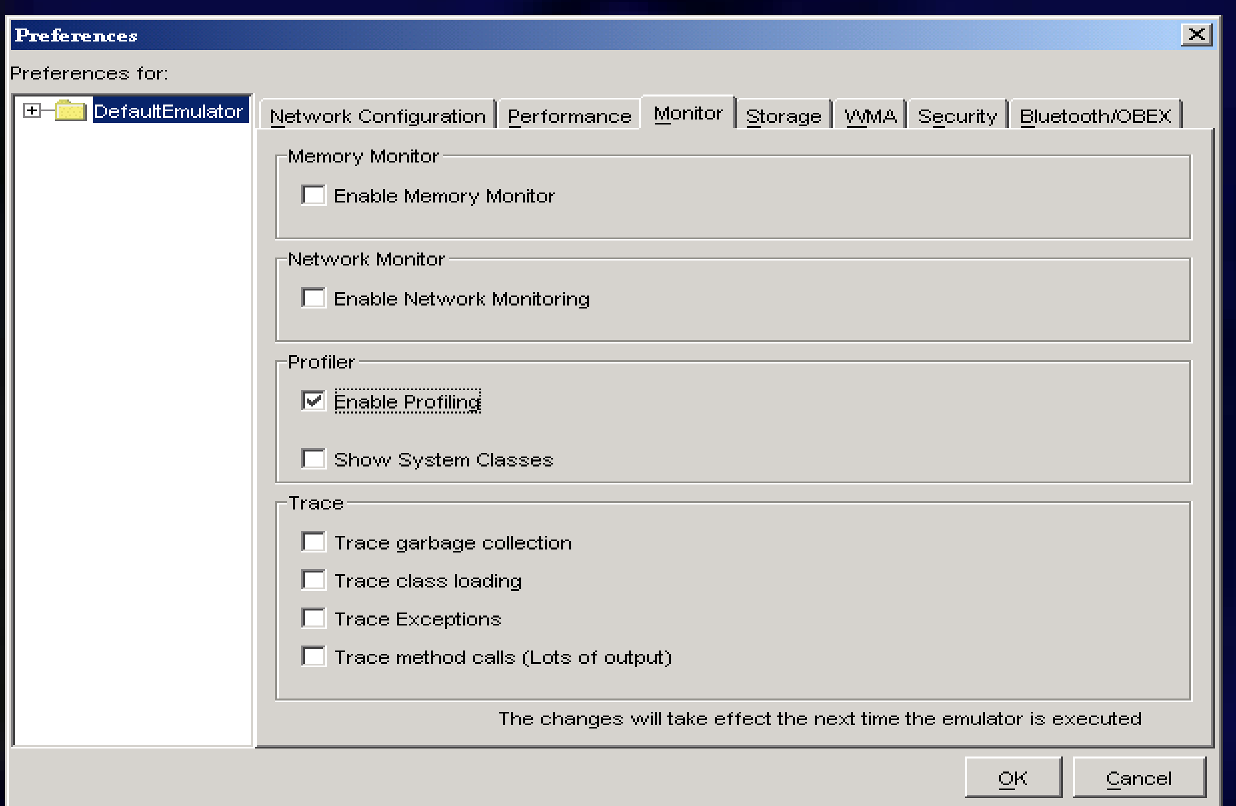
```
long t1 = System.currentTimeMillis();
fillCanvas();
long t2 = System.currentTimeMillis();
System.out.println("Time: " + (t2-t1) + "ms");
```

# Otimização de Aplicações J2ME

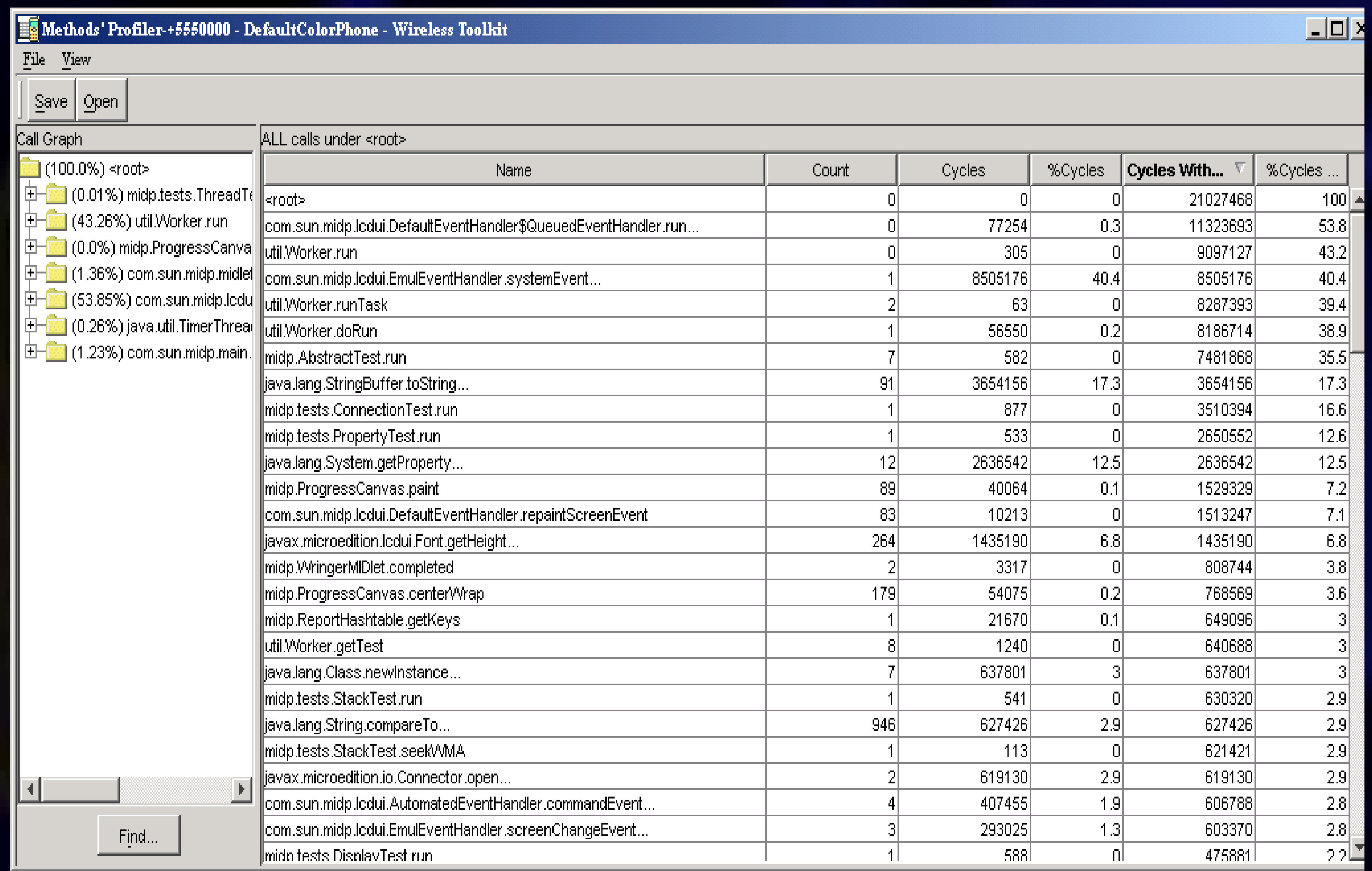
- Entertanto, a estratégia anterior não é apropriada para determinar precisamente quais métodos possuem problemas de performance, dentro de uma aplicação muito grande
- A solução para este problema está no uso de ferramentas para análise de performance (Profiler), monitoramento de memória e monitoramento do tráfego de rede

# Otimização de Aplicações J2ME

- Exemplo no KtoolBar do Sun Wireless toolkit



# Otimização de Aplicações J2ME




| Name | Count | Cycles | %Cycles | Cycles With... | %Cycles ... |
|---|---|---|---|---|---|
| <root> | 0 | 0 | 0 | 21027468 | 100 |
| com.sun.midp.lcdui.DefaultEventHandler$QueuedEventHandler.run... | 0 | 77254 | 0.3 | 11323693 | 53.8 |
| util.Worker.run | 0 | 305 | 0 | 9097127 | 43.2 |
| com.sun.midp.lcdui.EmulEventHandler.systemEvent... | 1 | 8505176 | 40.4 | 8505176 | 40.4 |
| util.Worker.runTask | 2 | 63 | 0 | 8287393 | 39.4 |
| util.Worker.doRun | 1 | 56550 | 0.2 | 8186714 | 38.9 |
| midp.AbstractTest.run | 7 | 582 | 0 | 7481868 | 35.5 |
| java.lang.StringBuffer.toString... | 91 | 3654156 | 17.3 | 3654156 | 17.3 |
| midp.tests.ConnectionTest.run | 1 | 877 | 0 | 3510394 | 16.6 |
| midp.tests.PropertyTest.run | 1 | 533 | 0 | 2650552 | 12.6 |
| java.lang.System.getProperty... | 12 | 2636542 | 12.5 | 2636542 | 12.5 |
| midp.ProgressCanvas.paint | 89 | 40064 | 0.1 | 1529329 | 7.2 |
| com.sun.midp.lcdui.DefaultEventHandler.repaintScreenEvent | 83 | 10213 | 0 | 1513247 | 7.1 |
| javax.microedition.lcdui.Font.getHeight... | 264 | 1435190 | 6.8 | 1435190 | 6.8 |
| midp.WringerMIDlet.completed | 2 | 3317 | 0 | 808744 | 3.8 |
| midp.ProgressCanvas.centerWrap | 179 | 54075 | 0.2 | 768569 | 3.6 |
| midp.ReportHashtable.getKeys | 1 | 21670 | 0.1 | 649096 | 3 |
| util.Worker.getTest | 8 | 1240 | 0 | 640688 | 3 |
| java.lang.Class.newInstance... | 7 | 637801 | 3 | 637801 | 3 |
| midp.tests.StackTest.run | 1 | 541 | 0 | 630320 | 2.9 |
| java.lang.String.compareTo... | 946 | 627426 | 2.9 | 627426 | 2.9 |
| midp.tests.StackTest.seekWMA | 1 | 113 | 0 | 621421 | 2.9 |
| javax.microedition.io.Connector.open... | 2 | 619130 | 2.9 | 619130 | 2.9 |
| com.sun.midp.lcdui.AutomatedEventHandler.commandEvent... | 4 | 407455 | 1.9 | 606788 | 2.8 |
| com.sun.midp.lcdui.EmulEventHandler.screenChangeEvent... | 3 | 293025 | 1.3 | 603370 | 2.8 |
| midp.tests.DisplayTest.run | 1 | 588 | 0 | 475881 | 2.2 |

# Otimização de Aplicações J2ME

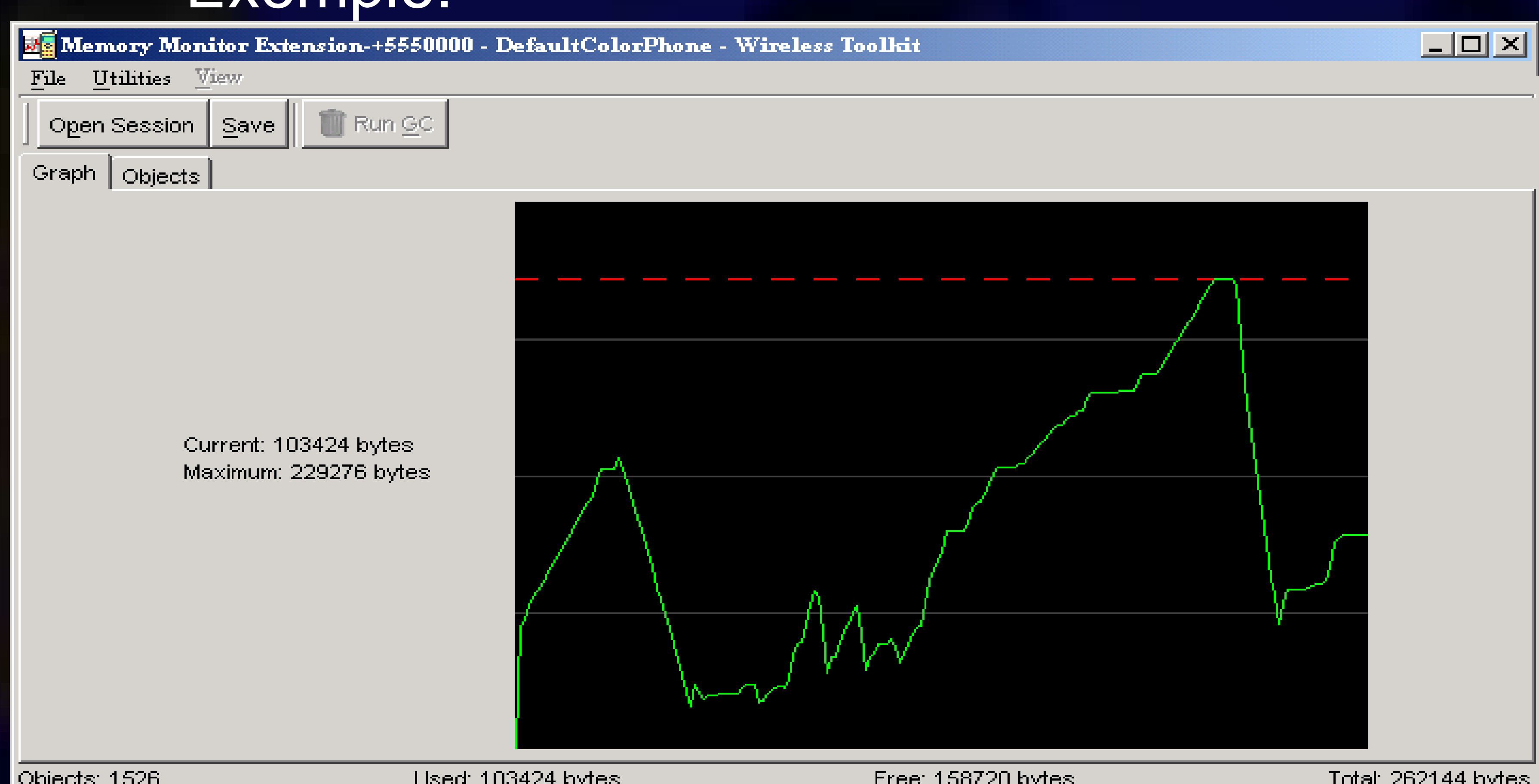
- Monitoramento de Memória
  - Memória é um recurso difícil em muitos dispositivos MIDP
  - Monitorando de dentro do programa:

```
Runtime rt = java.lang.Runtime.getRuntime();
System.out.println("Total heap: " + rt.totalMemory());
System.out.println("Total free: " + rt.freeMemory());
```

  - Para ativar o monitoramento de memória no wireless toolkit da Sun, utilizar a opção Edit->Preferences/Monitor da ferramenta KtoolBar

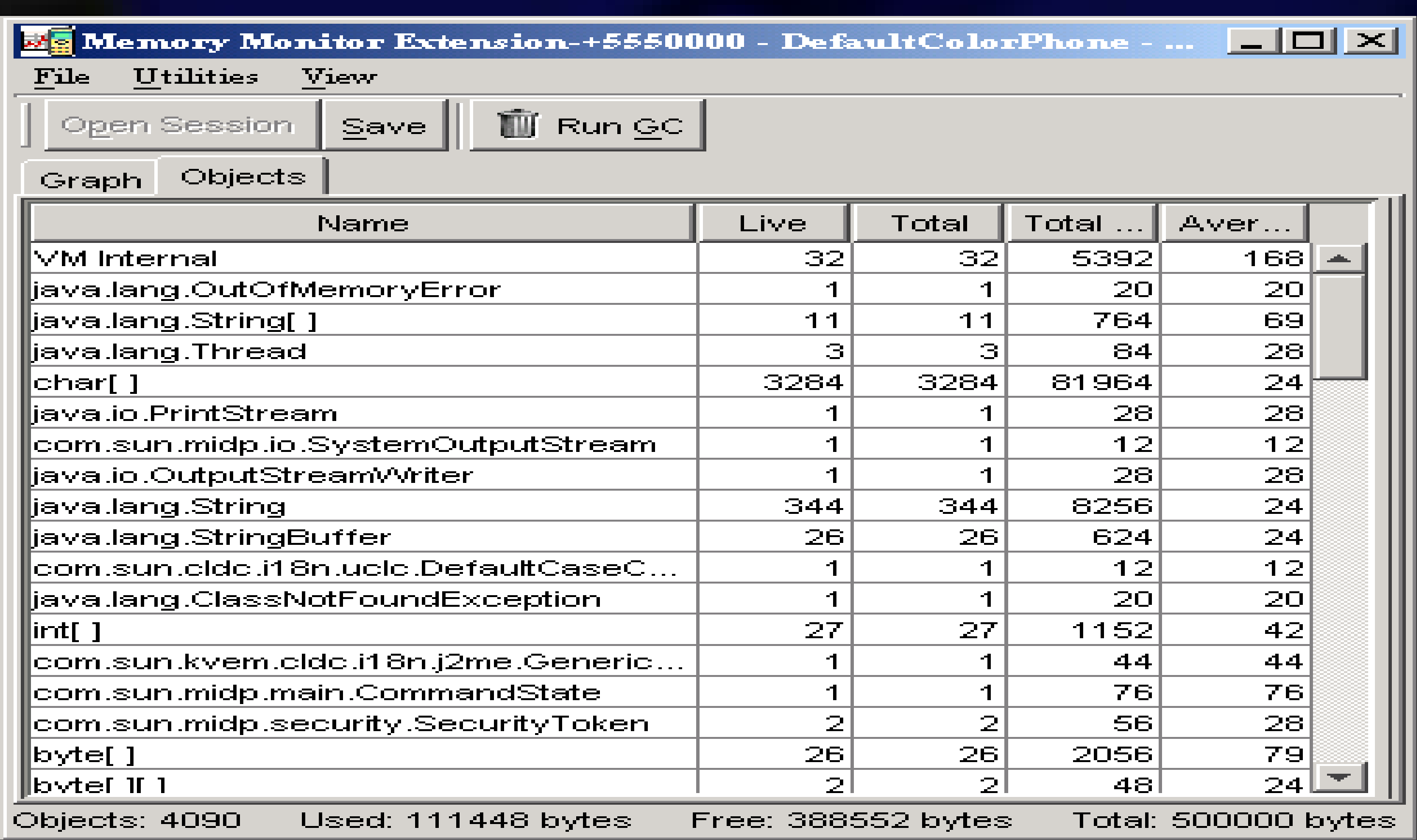
# Otimização de Aplicações J2ME

- Monitoramento de Memória
  - Exemplo:



# Otimização de Aplicações J2ME

- Monitoramento de Memória
  - Detalhes dos Objetos:




| Name | Live | Total | Total ... | Aver... |
|---|---|---|---|---|
| VM Internal | 32 | 32 | 5392 | 168 |
| java.lang.OutOfMemoryError | 1 | 1 | 20 | 20 |
| java.lang.String[ ] | 11 | 11 | 764 | 69 |
| java.lang.Thread | 3 | 3 | 84 | 28 |
| char[ ] | 3284 | 3284 | 81964 | 24 |
| java.io.PrintStream | 1 | 1 | 28 | 28 |
| com.sun.midp.io.SystemOutputStream | 1 | 1 | 12 | 12 |
| java.io.OutputStreamWriter | 1 | 1 | 28 | 28 |
| java.lang.String | 344 | 344 | 8256 | 24 |
| java.lang.StringBuffer | 26 | 26 | 624 | 24 |
| com.sun.cldc.i18n.uclc.DefaultCaseC... | 1 | 1 | 12 | 12 |
| java.lang.ClassNotFoundException | 1 | 1 | 20 | 20 |
| int[ ] | 27 | 27 | 1152 | 42 |
| com.sun.kvem.cldc.i18n.j2me.Generic... | 1 | 1 | 44 | 44 |
| com.sun.midp.main.CommandState | 1 | 1 | 76 | 76 |
| com.sun.midp.security.SecurityToken | 2 | 2 | 56 | 28 |
| byte[ ] | 26 | 26 | 2056 | 79 |
| byte[ ][ ] | 2 | 2 | 48 | 24 |

Objects: 4090 Used: 111448 bytes Free: 388552 bytes Total: 500000 bytes