FUNDAÇÃO DE ENSINO "EURÍPIDES SOARES DA ROCHA"
MANTENEDORA DO CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA – UNIVEM
CURSO DE CIÊNCIA DA COMPUTAÇÃO

RENATO BUENO DE CAMARGO JUNIOR

DESENVOLVIMENTO DE APLICATIVOS PARA DISPOSITIVOS PORTÁTEIS UTILIZANDO J2ME

MARÍLIA
2005

RENATO BUENO DE CAMARGO JUNIOR

DESENVOLVIMENTO DE APLICATIVOS PARA DISPOSITIVOS PORTÁTEIS UTILIZANDO J2ME

Monografia apresentada ao Curso de Ciência da Computação, da Fundação de Ensino "Eurípides Soares da Rocha", Mantenedora do Centro Universitário Eurípides de Marília – UNIVEM, como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Orientador:
Prof. Ms. André Luiz Satoshi Kawamoto

MARÍLIA
2005

CAMARGO, Renato Bueno Junior
Desenvolvimento de aplicativos para dispositivos portáteis utilizando j2me / Renato Bueno de Camargo Junior; orientador: André Luiz Satoshi Kawamoto. Marília, SP: [s.n.], 2003.

Monografia (Graduação em Ciência da Computação)- Curso de Ciência da Computação, Fundação de Ensino "Eurípides Soares da Rocha", Mantenedora do Centro Universitário Eurípides de Marília – UNIVEM.

1. Engenharia de software 2. Java 3. J2ME

CDD: 005.1151

RENATO BUENO DE CAMARGO JUNIUOR
RA Nº 287164

DESENVOLVIMENTO DE APLICATIVOS PARA DISPOSITIVOS PORTÁTEIS UTILIZANDO J2ME

BANCA EXAMINADORA DA MONOGRAFIA PARA
OBTENÇÃO DO GRAU DE BACHAREL EM CIÊNCIA DA COMPUTAÇÃO

CONCEITO FINAL: 8,0 (oito)

ORIENTADOR: _____
Prof. Ms. Andre Luiz Satoshi Kawamoto

1º EXAMINADOR: _____
Prof. Dr. Edmundo Sergio Spoto

2º EXAMINADOR: _____
Prof. Dr. Maria Istela Cognin

Marília, 01 de dezembro de 2005

À Deus, pelo esplendor da vida, presente em todas as atividades;

Aos amigos pelo incentivo e ajuda;

À minha família que me deu apoio, compreensão e carinho nas horas difíceis.

AGRADECIMENTOS

Agradeço as manifestações de carinho e apreço, recebidas de todos os familiares, os quais foram os artifícios e a luz inspiradora, para o sucesso deste trabalho.
Agradeço as manifestações de amizade e ajuda, recebidas de todos os amigos, que também compartilharam comigo os momentos de estudos e trabalhos, vocês foram fundamentais, sem as suas colaborações eu não teria conseguido.

Agradeço de modo particular:

À todas as pessoas que me ajudaram direta ou indiretamente em mais este trabalho.

E à Deus pela força de superar os obstáculos e vencer os desafios.

Ao prof. Ms. André Luiz Satoshi Kawamoto, pelo auxílio seguro e oportuno na orientação, aliados à experiência intelectual e profissional, que foram imprescindíveis para o desenvolvimento e conclusão deste trabalho.

“Os homens que decidem realizar o que os outros julgam impossível são os que fazem as descobertas, dão origem aos inventos e movem o mundo”.

Joel Barker

CAMARGO, Renato Bueno Junior. Desenvolvimento de aplicativos para dispositivos portáteis utilizando J2ME. Monografia (Bacharelado em Ciência da Computação) – Curso de

Ciência da Computação, Fundação de Ensino “Eurípides Soares da Rocha”, Mantenedora do Centro Universitário Eurípides de Marília – UNIVEM.

RESUMO

O propósito deste trabalho é investigar a utilização da plataforma J2ME (*Java 2 Micro Edition*) para o desenvolvimento de aplicativos para pequenos dispositivos, mais especificamente telefones celulares e PDAs (*Personal Digital Assistants*). Adicionalmente, este trabalho visa servir como uma referência básica sobre esta tecnologia, já que se trata de uma plataforma relativamente nova e que, até o momento, não dispõe de muita documentação em Português.

J2ME tem se tornado bastante utilizada porque, com o apoio da JCP (*Java Community Process*), que engloba diversos fabricantes de dispositivos, está sendo criado um padrão de desenvolvimento para esses equipamentos, os quais apresentam muitas diferenças entre si.

Procurou-se mostrar de forma clara como esta plataforma funciona, detalhando seus aspectos técnicos e sua interação com os dispositivos para os quais ela se destina.

A pesquisa realizada para este trabalho se deu, principalmente, em livros técnicos. Devido à carência de material sobre a tecnologia no Brasil, foram utilizados também *sites* da Internet, sobretudo documentos da *SUN Microsystems*, criadora da linguagem, e também de fabricantes de equipamentos móveis como Ericsson, Nokia, Motorola, Palm, HP e outros.

CAMARGO, Renato Bueno Junior. Desenvolvimento de aplicativos para dispositivos portáteis utilizando J2ME. Monografia (Bacharelado em Ciência da Computação) – Curso de Ciência da Computação, Fundação de Ensino "Eurípides Soares da Rocha", Mantenedora do Centro Universitário Eurípides de Marília – UNIVEM.

ABSTRACT

The purpose of this work is to investigate the utilization of the J2ME (Java 2 Micro Edition) platform for the development of applications for small devices, more specifically cell phones and PDAs (Personal Digital Assistants). Moreover, this work aims to stand as a basic reference about this technology, since it is a relatively new platform and, up to this moment, does not have much documentation in Portuguese.

J2ME has become widely used because, supported by the JCP (Java Community Process), which involves several device manufacturers, a development pattern for such devices, which present several differences among themselves, is being created.

One of the goals of this work was to show clearly how this platform works; detailing its technical aspects and its interaction with the devices which it is designed for.

The research for this work was accomplished, mainly, in technical books. Due to the lack of material about this technology in Brazil, Internet sites were also used, principally documents from the Sun Microsystems, creator of the language, and from mobile device manufacturers such as Ericsson, Nokia, Motorola, Palm, HP, and others.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURA E SIGLAS

API - Aplication Program Interface

AWT - Abstract Window Toolkit

CDC - Connected Device Configuration

CLDC - Connected, Limited Device Configuration

IXC - Inter-Xlet Communication

J2EE - Java 2 Enterprise Edition

J2ME - Java 2 Micro Edition

J2SE - Java 2 Standard Edition

JCA - Java Cryptography Architecture

JCP - Java Community Process

JDBC - Java Database Conectivity

JSR - Java Specification Request

JVM - Java Virtual Machine

MIDP - Mobile Information Device Profile

MMAPI - Mobile Media API

PDAP - Personal Digital Assistant Profile

RMI - Remote Invocation Method

WLAN - Wireless Local Area Network

WMAPI - Wireless Messaging API

WTK - Wireless Toolkit

SUMÁRIO

1. INTRODUÇÃO

Motivação

Com a evolução da tecnologia surgiu a necessidade de estarmos constantemente atualizados, devido a grande concorrência no mercado e mesmo por questões particulares. Isso gerou uma grande demanda por dispositivos que possam disponibilizar informações de forma prática e rápida. No cenário atual, os dispositivos mais usados que fornecem tal benefício são os celulares de 3ª Geração e PDAs (*Personal Digital Assistants*), que devido a grande e rápida evolução tecnológica estão convergindo para um único aparelho, capaz de se utilizar da crescente largura de banda para disponibilizar serviços de voz, multimídia, transmissão, armazenamento e processamento de dados.

Objetivo

Este trabalho tem o objetivo de mostrar a utilização da Linguagem J2ME para o desenvolvimento de vários tipos de aplicações desde, por exemplo, aplicativos integrados à banco de dados de grandes empresas, até a games para usuários domésticos.

Organização

Além da análise da plataforma J2ME, este trabalho apresenta um estudo de caso simples, e está organizado da seguinte maneira: o Capítulo 2 introduz a plataforma J2ME, apresentando sua arquitetura, configurações e perfis; o Capítulo 5 descreve a ferramenta *Wireless Tookit*, utilizada no desenvolvimento de aplicações J2ME; o Capítulo 6 mostra o estudo de caso com três diferentes aplicações cada um com um grau de complexidade; e finalmente o Capítulo 7 mostra as conclusões obtidas e os trabalhos futuros.

2. JAVA 2 MICRO EDITION (J2ME)

A plataforma Java definida pela Sun Microsystems é atualmente dividida em quatro grandes grupos: *Java 2 Standard Edition* (J2SE), destinado a computadores pessoais domésticos; *Java 2 Enterprise Edition* (J2EE), mais abrangente que o J2SE, e destinado a aplicações em servidores; *Java 2 Micro Edition* (J2ME), que tem o objetivo de disponibilizar aplicativos Java em dispositivos portáteis;.e finalmente, *JavaCard*, que é uma tecnologia destinada a rodar em *smart-cards* e outros dispositivos extremamente limitados. A família da plataforma Java atual é mostrada na Figura 1.

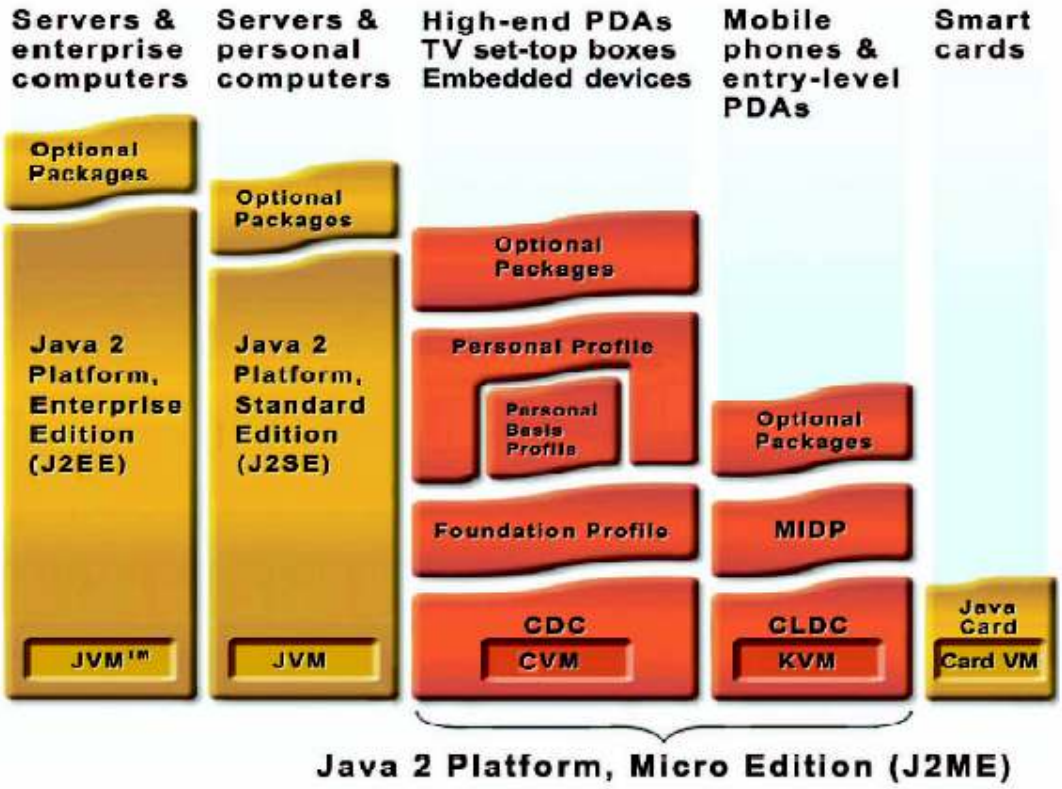


Figura 1: A Família Java[1].

Particularmente, o desenvolvimento da arquitetura da plataforma J2ME necessitou de maior atenção, pois os dispositivos para os quais foi criada possuíam características intrínsecas diferentes uns dos outros. Embora J2ME possua um escopo bem mais amplo do que o da J2SE e o da J2EE, essa plataforma é menos conhecida e estudada que as outras no meio da informática.

[1] Fonte: Sun Microsystem - Datasheet Java 2 Platform, Micro Edition http://java.sun.com/j2me/docs/j2me-ds.pdf

Isso tem mudado nos últimos anos, com o crescente estudo da computação móvel e a descoberta de novas tecnologias no âmbito dos sistemas embutidos, que deu a essa plataforma um enfoque bem maior, uma vez que J2ME é voltada tanto para o mercado de dispositivos de pequeno porte, tais como telefones celulares, PDAs, *Internet screenphones*, televisores digitais de nova geração, sistemas de navegação automotiva, comutadores e roteadores de rede, componentes para automação residencial, etc.

Esses dispositivos caracterizam-se pelo fato de que seu propósito original não é o processamento de dados, mas, por possuírem algum tipo de microprocessador, são potencialmente capazes de realizar operações computacionais.

J2ME possibilita criar aplicações especiais para tirar proveito das vantagens e particularidades desses aparelhos, ou seja, aplicações integradas ao serviço das operadoras, que tenham a interatividade esperada para um pequeno aparelho e que respondam de acordo com a situação e a intenção do usuário.

Finalmente, J2ME é um conjunto de especificações que define uma JVM (*Java Virtual Machine* - Máquina Virtual Java) simplificada, um conjunto de APIs (*Application Programming Interfaces* - Interfaces para Programação de Aplicações) especializadas para pequenos dispositivos e ferramentas direcionadas aos dispositivos com poder de processamento, memória e conectividade menor que um computador de mesa.

Essa plataforma possui vantagens em relação às demais tecnologias *wireless*, que são a portabilidade, ou seja, a capacidade de desenvolver aplicações que executem em qualquer dispositivo, independente de tecnologia de operadora ou fabricante; e a possibilidade de se utilizar os recursos gerais de Java, ou seja, programação orientada a objetos, suporte à manipulação da tela gráfica e teclado dos dispositivos, maior suporte à conectividade com outros dispositivos e programação em rede com o protocolo HTTP.

No passado, todo dispositivo era oferecido no mercado com um conjunto fixo de funcionalidades cuja programação era realizada exclusivamente pelo fabricante, sobre uma tecnologia altamente proprietária. J2ME possibilita desenvolver, atualizar e instalar novas aplicações segundo as necessidades particulares de cada usuário. O uso de Java na programação de pequenos aparelhos significa ganhar todas as vantagens que a tecnologia traz consigo: (SUN, 2004)

▪ **Dinamismo**, pois novas aplicações podem ser obtidas da rede e instaladas no dispositivo a qualquer momento;

▪ **Segurança**, pois, garantem a proteção das informações carregadas pelo dispositivo, ou seja, os dados de uma aplicação não são acessíveis por outras;

▪ **Portabilidade**, uma vez que aplicações podem ser portadas entre dispositivos de diferentes fabricantes e de diferentes tipos;

▪ **alto nível de abstração do código**, **modularização**, **reusabilidade**, características próprias da Orientação a Objetos;

▪ **Confiabilidade**, pois não é tolerado *reboots* ou *crashs* em dispositivos embarcados. Graças aos mecanismos de proteção e gerência de memória, Java atinge esta demanda.

2.1. Arquitetura de J2ME

A arquitetura hierárquica do J2ME é definida por um conjunto de especificações em camadas, onde cada especificação visa atender às necessidades específicas ou gerais de um dispositivo. Duas camadas compõem o J2ME: as Configurações (*Configurations*) e os Perfis (*Profiles*) conforme ilustrado na Figura 2.

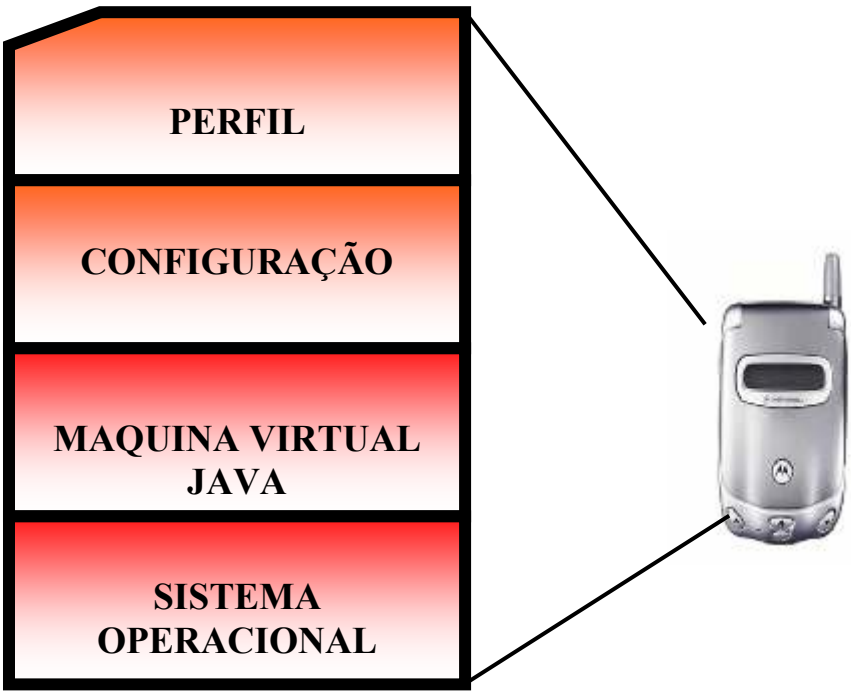


Figura 2: As camadas do J2ME[2]

As configurações definem uma plataforma mínima para atender a uma família de dispositivos que compartilham os mesmos requisitos, como poder de processamento e

[2] Fonte Sun Microsystem TM - "J2ME Building Blocks for Mobile Devices" – White,Paper on KVM and the Connected, Limited Device Configuration (CLDC), http://java.sun.com/products/cldc/wp/KVMwp.pdf

memória. Para tanto, são definidas as especificações da Máquina Virtual Java e da API em execução nesse grande grupo de dispositivos.

Os perfis, por outro lado, estendem uma determinada configuração e definem as necessidades específicas de uma categoria de dispositivo, como telefones celulares ou PDAs. Um perfil é, portanto, composto de uma API específica para cada dispositivo, mantendo interoperabilidade a uma mesma família de dispositivos.

As especificações de configurações e perfis são desenvolvidas e administradas por um grupo de usuários de dispositivos eletrônicos e produtores através da *Java Community Process* (JCP) (SUN, 2004). Atualmente, a plataforma J2ME tem especificado duas configurações: *Connected Device Configuration* (CDC) e *Connected Limited Device Configuration* (CLDC) ; quatro perfis: *Mobile Information Device Profile* (MIDP), *Personal Profile,PersonalBasisProfile* e *Foundantion Profile*.

3. Configurações

Uma configuração define uma plataforma mínima para um grupo de dispositivos com características similares, tanto na memória quanto no poder de processamento. Sendo assim, uma configuração define as características suportadas tanto pela própria linguagem de programação Java, pela máquina virtual e suas bibliotecas de classes e APIs, as quais um determinado fabricante pode esperar que estarão disponíveis em todos os dispositivos de uma mesma categoria.

Atualmente, existem duas configurações definidas para o J2ME, o CLDC e o CDC (SUN, 2004). O relacionamento entre essas duas configurações e o ambiente padrão de desenvolvimento Java (J2SE) está ilustrado na Figura 3.

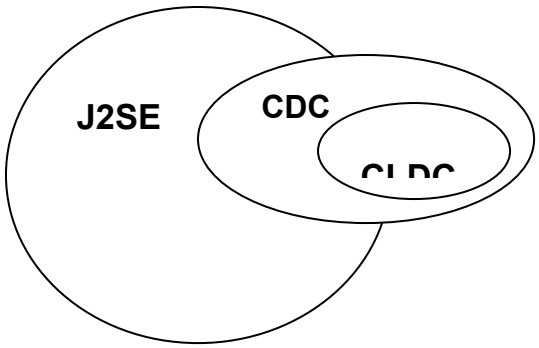


Figura 3. Relacionamento entre as Configurações do J2ME e J2SE

Conforme mostrado na Figura 3, as configurações do J2ME herdam a maioria de sua funcionalidade do J2SE e um pequeno número de características são introduzidas para atender as necessidades de dispositivos com escassez de recursos.A grande razão para estruturação em camadas do J2ME e o pequeno número de configurações é diminuir o esforço e o custo em manutenção de uma grande variedade de máquinas virtuais e ao mesmo tempo fornecer implementações comuns aos mais diversos perfis. Uma mudança pequena em uma configuração pode refletir em um conjunto de alterações complexas na máquina virtual, o que inviabilizaria um grande número de configurações.

3.1. CLDC

O objetivo do CLDC é padronizar uma plataforma Java mínima de desenvolvimento de aplicações, altamente portável, para dispositivos conectados com restrições de recursos CLDC. Os telefones celulares, PDAs, televisores, dentre outros, são alguns dispositivos que suportam a especificação do CLDC (SUN, 2004).

Um dos objetivos secundários do CLDC é a redução dos recursos fornecidos na API, de forma a manter a margem atual de preços dos dispositivos mencionados. Outro objetivo é focar no desenvolvimento de aplicações ao invés do desenvolvimento de sistemas, em decorrência da redução das APIs de alto-nível fornecidas pelo CLDC. Finalmente, é desejável a distribuição dinâmica de aplicações pela Internet, portanto é preciso definir um ambiente seguro para o *download* dessas aplicações através de diversas redes e para diversos dispositivos.

O CLDC foi desenvolvido para atender a dispositivos com uma série de restrições em diversos níveis, por conseguinte, o CLDC deve atender a uma série de exigências básicas de hardware, software e da plataforma J2ME.

Em nível de hardware, o CLDC assume que os dispositivos possuam ao menos 160 Kbytes de memória não-volátil para a máquina virtual e as bibliotecas Java e, ao menos 32 Kbytes de memória volátil para a execução da máquina virtual. Também são necessários um processador de 16 ou 32-bits e baixo consumo de bateria; e conectividade a algum tipo de rede com limitações de largura de banda e conexões intermitentes.

Telefones celulares, *pagers*, PDAs, aparelhos domésticos e terminais de vendas são alguns dos dispositivos que podem suportar essa especificação.

Outro fator importante é a capacidade do software, que pode variar entre os dispositivos. Alguns sistemas operacionais, por exemplo, oferecem suporte a operações concorrentes e *multithreading*, enquanto outros podem nem possuir um sistema de arquivos. O CLDC requer um sistema operacional ou um *kernel* que possa, pelo menos, manipular o *hardware* e conceder um processo de execução a máquina virtual.

A plataforma J2ME deve definir um "mínimo denominador comum" para ser portável a amplo escopo de dispositivos (SUN, 2004). Para manter a interoperabilidade, o CLDC não especifica nenhuma característica exclusiva a um determinado dispositivo, um dos papéis dos pacotes opcionais e dos perfis do J2ME. Estes últimos, no CLDC, são desenvolvidos para atender aos requisitos específicos dos dispositivos. O único perfil existente para o CLDC é o MIDP usado em telefones celulares. Outro perfil do CLDC é o PDAP *(Personal Digital Assistant Profile*), ainda em estágio de desenvolvimento, que irá endereçar os PDAs em geral.

Esse conjunto mínimo de exigências e os objetivos do CLDC estabelecem a base através da qual é projetada a arquitetura do CLDC e são definidos os aspectos de segurança da plataforma. Outras considerações estão voltadas à aderência da API à especificação da linguagem Java e da máquina virtual, e as bibliotecas específicas do CLDC.

3.1.1. O Modelo de Segurança do CLDC

A reduzida disponibilidade de recursos de memória para o CLDC impede que o *framework* de segurança do J2SE possa ser incorporado em sua totalidade, havendo conseqüentes simplificações. Busca-se a segurança no CLDC em três diferentes níveis: na máquina virtual, nas aplicações e na comunicação fim-a-fim.

O primeiro nível de segurança é na máquina virtual (baixo-nível) que impede que uma aplicação em execução cause danos no dispositivo hospedeiro. A segurança neste nível é realizada pelo *Class File Verifier*, responsável por evitar uma ordem de execução incorreta dos *bytecodes* e por verificar se os objetos não referenciam áreas inválidas de memória.

O *Class File Verifier* somente pode analisar, *grosso modo*, se uma aplicação Java é válida ou não. O acesso a recursos externos está além do escopo do primeiro nível de segurança. O conceito de segurança em nível de aplicação significa que "uma aplicação Java pode somente acessar aquelas bibliotecas, recursos de sistema e outros componentes que o dispositivo e o ambiente Java permite acessar" (SUN, 2004). O acesso a impressoras

compartilhadas, a dispositivos infravermelhos ou a própria rede, é gerenciado por esse nível de segurança. As medidas de segurança realizadas nesse nível são: modelagem da aplicação em uma *(sandbox*), a proteção de classes do sistema e adição de restrições na carga dinâmica de classes.

O modelo em *sandbox* busca limitar os recursos destinados para a aplicação à somente aqueles internos ao *sandbox*, de forma a impedir que aplicações errôneas possam obter controle a estes recursos.

A proteção de classes do sistema impede que projetistas possam sobrecarregar ou modificar algumas das classes protegidas do sistema, como o pacote `javax.microedition`.

Uma nova restrição é adicionada na carga dinâmica de classes no CLDC, a qual define que uma classe só pode carregar outra classe interna ao seu pacote JAR, impedindo que ela interfira na execução de uma outra aplicação.

O último nível de segurança assume que o dispositivo é uma entidade de uma comunicação fim-a-fim, como uma WLAN (*wireless Local Area Network – rede local sem fio*). Uma das medidas de segurança neste nível mais comuns é a criptografia dos dados. Em particular, este nível de segurança está fora do escopo do CLDC e é dependente da implementação da aplicação.

3.1.2. Restrições da API Java do CLDC

A máquina virtual do CLDC deve ser compilável com as especificações da linguagem Java, levando-se em consideração as limitações da API CLDC. As diferenças entre as especificações da linguagem Java são:

- Ausência do método `finalize` nas classes;
- Limitação nas exceções e tratamento de erros.

As exceções da linguagem Java, exceto as exceções assíncronas, têm suporte do CLDC. Por outro lado, o tratamento de erros é minimizado porque a recuperação de cada erro é específica a um dado dispositivo e seu custo é muito elevado causando um *overhead* desnecessário. Outras diferenças da API do CLDC são decorrentes das mudanças na máquina virtual listadas abaixo:

▪ Formatos de arquivos de classe e cargas de classe;

▪ Verificação dos arquivos de classe;

▪ Redução das funções da máquina virtual.

As funções da máquina virtual são reduzidas e não permitem grupos de *threads* e *threads daemon*.

O modelo de verificação dos arquivos de classe realizado do CLDC deve ser diferente do J2SE, porque este último requer grande quantidade de memória e elevado índice de processamento. O processo de verificação do CLDC executa uma leitura linear do *bytecode* e é dividido em dois estágios.

No primeiro estágio, uma ferramenta de pré-verificação é aplicada sobre o *bytecode* e adiciona atributos do tipo `StackMap` nas classes. Esses atributos aumentam o desempenho do segundo estágio de verificação que ocorre em tempo de execução no dispositivo hospedeiro da aplicação.

3.1.3. Bibliotecas do CLDC

O conjunto de bibliotecas do CLDC busca fornecer um conjunto mínimo de funcionalidades para o desenvolvimento de aplicações e para a definição dos perfis na camada superior. As bibliotecas da especificação do CLDC pertencem a um subconjunto das classes do J2SE, ou são classes específicas.

As classes derivadas do J2SE devem manter a mesma semântica de sua especificação original, e conseqüentemente devem manter o mesmo conjunto de atributos e métodos. Não devem ser adicionadas quaisquer outras especificações na classe. Os pacotes e as respectivas classes herdados do J2SE para o J2ME estão listados na Tabela 1.

Tabela 1. Classes Derivadas do J2SE no CLDC

Tipo	Pacote	Classes e Interfaces
Sistema	`Java.lang`	`Object, Class, Runtime, System, Thread, Runnable, String, StringBuffer, Throwable`
Tipos de Dados	`Java.lang`	`Boolean, Byte, Short, Integer, Long, Float, Double, Character`
Coleções	`Java.util`	`Vector, Stack, Hastable, Enumeration`
Entrada e Saída	`Java.io`	`InputStrem, OutputStream, ByteArrayInputStream, ByteArrayOutputStream, DataInput, DataOutput, DataInputStream, DataOutputStream, Reader, Writer, InputStreamReader, OutputStreamWriter, PrintStream`
Data e Hora	`Java.util`	`Calendar, Date, TimeZone`
Utilidade Geral	`java.util` `java.lang`	`Math, Random`

Exceção e Erro	`java.lang`
	`java.io`
	`Java.util`
		`Exception, Error (e respectivas subclasses)`

Referências Fracas	`Java.lang.ref`	`Reference, WeakReference`

3.1.4. Pacotes Opcionais do CLDC

A plataforma J2ME pode ser estendida pela adição de pacotes adicionais de desenvolvimento para as configurações e os perfis. O objetivo destes pacotes é endereçar requisitos específicos de algumas aplicações fornecendo uma API especializada para o tratamento de novas tecnologias. Os pacotes opcionais presentes no CLDC são: *Mobile Media;Wireless Messaging API;BlueTooth* e *Location API*.

3.1.4.1. *Mobile Media*

O *Mobile Media* (MMAPI), *Java Specification Request 135* (JSR), (JCP, 2004) especifica uma pequena API multimídia para dispositivos embutidos. Este pacote fornece acesso e controle de mídias de áudio e de vídeo e é escalável e extensível para outras funcionalidades multimídia sofisticadas.

O MMAPI foi desenvolvido para ser flexível ao formato de arquivo de mídia e ao protocolo deixando a cargo do projetista do perfil a especificação dos mesmos. Uma vantagem do MMAPI é que ele é favorável à configuração CDC.

3.1.4.2. *Wireless Messaging API*

O *Wireless Messaging* (WMAPI), o JSR 120, (JCP, 2004) define uma API para acesso padrão a recursos de comunicação móvel que poderá oferecer a terceiros um ambiente de desenvolvimento de aplicações Java com ampla conectividade. O conjunto de componentes fornecidos pela API do WMAPI pode ser usado isoladamente ou em conjunto com qualquer perfil J2ME.Algumas das tecnologias que possuirão suporte no WMAPI são: *Short Message Service* (SMS), *Unstructered Supplementary Service Data* (USSD) e *Cell Broadcast Service* (CBS).

3.1.4.3. *Bluetooth*

A interface de comunicações sem fio *Bluetooth* foi desenvolvida como uma especificação aberta para comunicação de voz e dados e originalmente projetada para comunicação em uma direção entre um dispositivo de rádio e um computador. Ela funciona como alternativa à interface de comunicação unidirecional com radiação infravermelha, oferecida pela *Infrared Data Association* (IrDA). A utilização do infravermelho na transmissão limita o raio de ação do dispositivo a 1 m, na ausência de qualquer obstrução [ALENCAR, 2001].

A interface incorpora alguma inteligência, permitindo a troca de dados automaticamente, sem intervenção do usuário. As especificações dessa tecnologia são

coordenadas pelo *Bluetooh Special Interest Group*, que tem entre seus fundadores empresas como Ericsson, IBM, Intel, Nokia e Toshiba.

Bluetooth usa uma forma lenta da técnica de espalhamento espectral (saltos em freqüência), com um total de 79 faixas de freqüências disponíveis e 1600 saltos em freqüência a cada segundo, de acordo com um padrão de codificação pseudo-aleatório. Caso haja colisão, um salto para uma faixa já ocupada, o pacote relevante é retransmitido na próxima freqüência. A modulação é FM digital (FSK), a transmissão é *full-duplex* com duplexação por divisão no tempo (TDD) e o sistema inclui criptografia e verificação de privacidade [ALENCAR, 2001].

A taxa de dados utilizável é de 456 kbit/s em cada direção, com a operação simétrica, e até 751 kbit/s para transmissão assimétrica, com 56 kbit/s na outra direção. A potência de transmissão típica é 1 mW (0 dBm), para um raio de 10 m, mas a potência pode ser elevada, se necessário, a 100 mW (20 dBm), para estender o raio de ação a 100 m. É possível formar piconets com um máximo de oito dispositivos, em que um deles é o mestre e os outros escravos. Piconets podem se comunicar, formando uma rede maior (*scatternet*) [ALENCAR, 2001].

O Grupo de Interesse Especial (*Special Interest Group* – SIG), com mais de dois mil membros, entre empresas de computação e telecomunicações, está convencido que a tecnologia *Bluetooth* é prática para ser usada como alternativa às ligações com fio em casas e pequenos escritórios. O mercado mundial de circuitos integrados (CI) *Bluetooth* é estimado em 500 milhões de unidades nos próximos cinco anos, e a idéia é incorporar o CI *Bluetooth* a todos os produtos já disponíveis no mercado, incluindo computadores (laptops e *notebooks*), periféricos de computadores (impressoras e teclados), telefones celulares, assistentes digitais pessoais e aparelhos de projeção [ALENCAR, 2001].

Com base nesta tecnologia, a Motorola desenvolveu uma API que permite aos usuários desenvolverem aplicações portáteis e interoperáveis. Esta API é conhecida unicamente como *Blueetooth API* e especificação JSR 82 (JCP , 2004).

3.1.4.4. *Location API for J2ME*

O pacote *Location API for* J2ME, JSR 179 (JCP, 2004) atende a necessidade crescente de aplicações baseadas em localidade. Esta API produz informação sobre a localização corrente do dispositivo em um contexto geo-espacial possibilitando aos projetistas desenvolverem aplicativos baseados em localização, como por exemplo, os Sistemas de

Posicionamento Global (GPS). Essa API, associada aos Sistemas de Informações Geográficos (GIS), busca fornecer ao usuário da tecnologia uma poderosa ferramenta de direcionamento em quaisquer localidades.

3.2. CDC

O CDC é uma configuração do J2ME ligeiramente mais rica em funcionalidade do que o CLDC. Seu objetivo é atender a uma ampla gama de consumidores de dispositivos embutidos, tais como PDAs e televisores. Em geral, os dispositivos que executam o CDC devem possuir, em nível de hardware, um processador de 32-bits e cerca de 2 MB de RAM e 2,5 MB de ROM para o ambiente Java.

Os usuários desta configuração incluem empresas, usuários de serviços, vendedores de dispositivos, projetistas de aplicações e provedores de serviços. Para cada uma dessas classes, o CDC fornece alguns benefícios. Às empresas, o CDC provê um amplo suporte ao conceito de mobilidade com segurança e a fácil integração no ambiente empresarial. Aos projetistas de aplicações, há uma rica API compatível com ferramentas de desenvolvimento do J2SE e que permite a reutilização de código escrito em J2SE.

O CDC permite o desenvolvimento de uma grande variedade de modelos de aplicativos para se adequar a diferentes ambientes, inclusive a Web. Esses modelos de aplicativos podem ser aplicações *standalone*, *applets* e *xlets*.

Uma aplicação *standalone* é o modelo mais básico de aplicativo Java, voltada ao desenvolvimento de utilitários e ambientes estáticos com pequena variação. Neste modelo de aplicação, a máquina virtual Java busca o método **`main`** e a partir dele instancia as classes usadas pela aplicação. O aplicativo encerra ao término do `main`.

Os *applets* permitem aos usuários interagir com páginas Web dinâmicas que contenham um ambiente de execução Java embutido. O sistema fornece recursos de segurança e uma interface gráfica. A vantagem destes aplicativos é o encapsulamento do ciclo de vida dos *applets*, cujos métodos são implementados pela ambiente de execução.

Os *xlets* possuem suporte nos perfis *Personal Basis* e *Personal*, e assemelham-se em propósito aos *applets*. A diferença está em nível de desenvolvimento: os *xlets* têm um ciclo de vida menor que os *applets* e não possuem dependência com conteúdo AWT, ou seja, são independentes da natureza da interface do ambiente.

3.2.1. O Modelo de Segurança do CDC

O modelo de segurança do CDC é mais completo que o do CLDC e fornece cinco níveis de segurança, em vez de três.

Semelhante ao CLDC, o primeiro nível de segurança ocorre na máquina virtual e é realizada pelo *Class File Verifier*. Este componente realiza a verificação de classes e algumas características da linguagem.

O nível seguinte é realizado na aplicação, e restringe seu acesso somente aos seus recursos disponíveis *(sandbox*). Outra política realizada neste nível é o controle de permissões regulado por uma política de acesso aos dados definida na aplicação. O CDC adiciona a criptografia para realizar o tráfego seguro de informações entre aplicações e dispositivos usando oo *framework* JCA *(Java Cryptography Architecture*), para acessar informações criptografadas e desenvolver funcionalidades de criptografia.

A administração de certificados de segurança e o policiamento da identidade da rede são os outros dois níveis de segurança. O certificado de segurança é um mecanismo que estende a segurança a um certificador confiável através de um sistema de autenticação. O policiamento das redes provê a administração da confiabilidade da rede de forma interoperável e descentralizada.

3.2.2. A API do CDC

A apresentação da API do CDC deve ser realizada levando em consideração e comparando os três perfis mencionados anteriormente, o *Foundation*, o *Personal Basis* e o *Personal*. Na Tabela 2 ilustram-se os pacotes suportados por cada perfil no CDC.

Tabela 2. API dos Perfis do CDC

Perfis	Pacotes

Foundation	`java.io`
	`java.lang`
	`java.lang.ref`
	`java.lang.reflect`
	`java.net`
	`java.security`
	`java.security.acl`
	`java.security.cert`
	`java.security.interfaces`
	`java.security.spec`
	`java.text`
	`java.util`
	`java.util.jar`
	`java.util.zip`
	`javax.microedition.io`

Personal Basis	`java.awt`
	`java.awt.color`
	`java.awt.image`
	`java.awt.event`
	`java.beans`
	`java.rmi`
	`java.rmi.registry`
	`javax.microedition.xlet`
	`javax.microedition.xlet.ixc`

Personal	`java.applet`
	`java.awt`
	`java.awt.datatransfer`

O perfil *Foundation* fornece algumas das funcionalidades do J2SE e incluem a classe `javax.microedition.io` para obter compatibilidade com o *framework* de conexão do CLDC. Enquanto isso, o perfil *Personal Basis* estende a API do *Foundation* e adiciona suporte a alguns componentes AWT, a *xlets*, a comunicação via RMI e a *beans*. Por fim, o perfil *Personal* contém as funcionalidades completas dos objetos AWT e suporte a *applets*.

3.2.3. Pacotes Opcionais do CDC

Alguns dos pacotes do CDC possuem a mesma especificação dos pacotes opcionais do CLDC, como o *Bluetooth API* e o *Wireless Messaging API*. Outros pacotes usam a funcionalidade adicional do CDC para atender aos requisitos de aplicações mais específicas. Esses pacotes são o *RMI Optional Package* e o *JDBC Optional Package*.

3.2.3.1. RMI Optional Package

A API RMI OP, JSR 66, (Chamada Remota de Procedimento - *Remote Method Invocation*) (JCP, 2004) insere no contexto do CDC aplicações distribuídas. A função do RMI OP é prover um mecanismo de desenvolvimento de protocolos de acesso a objetos distribuídos em sistemas remotos.

O relacionamento do RMI OP é bi-direcional, ou seja, as aplicações em dispositivos CDC podem manipular objetos nos servidores e vice-versa. Contudo, os dispositivos deverão implementar o perfil *Foundation*, ou algum que incorpore toda sua funcionalidade, e prover conectividade TCP/IP.

O RMI OP é uma especialização do pacote `java.rmi` do J2SE com suporte a muitas características deste pacote, como a semântica completa do protocolo RMI, a exportação de objetos remotos e coletores de lixo distribuídos. Por outro lado, RMI OP não possibilita o uso do protocolo através de *firewalls* usando *proxy* HTTP nem o protocolo de multiplexação do RMI.

3.2.3.2. JDBC OP

A tecnologia JDBC *(Java Database Conectivity*) permite acessar virtualmente qualquer fonte de informações através de linguagens de programação Java. Em geral, essa tecnologia é utilizada para acessar Sistemas Gerenciadores de Bancos de Dados (SGBD), contudo, a API JDBC OP, JSR 169, (JCP, 2004) permite acesso a qualquer componente que implemente uma interface JDBC, inclusive arquivos textuais.

De forma semelhante ao RMI OP, o JDBC OP é uma especialização do pacote `java.sql` do J2SE cujas funcionalidades foram adaptadas a um ambiente com recursos mais reduzidos e ausência de métodos depreciados (*deprecated*). O JDBC OP também requer que o dispositivo CDC implemente o perfil *Foundation*, ou outro que incorpore todas as suas funcionalidades.

4. Perfis

Um perfil estende uma configuração e especializa-a para um determinado segmento do mercado, ou um domínio de aplicação ou um grupo particular de dispositivos. Isso se dá pelo fato de haver uma ampla quantidade de dispositivos que possam ser atendidos pelo J2ME.

O J2ME provê o conceito de perfil para a definição de uma plataforma Java para um grupo de dispositivos específicos. O significado dessa divisão em perfis permite que um telefone celular possua um perfil enquanto um PDA possua outro perfil, e por outro lado, ambos compartilhem a mesma configuração.

Os perfis estão acima das configurações na hierarquia da arquitetura do J2ME e usufruem os serviços providos por ela. A configuração CLDC possui apenas um perfil: o *Mobile Information Device Profile* (MIDP), outro perfil do CLDC é o PDAP *(Personal Digital Assistant Profile*), ainda em estágio de desenvolvimento, que irá endereçar os PDAs em geral. Enquanto isso, a configuração CDC possui três perfis: *Foundation Profile* (FP), *Personal Basis Profile* (PBP) e *Personal Profile* (PP) (SUN, 2004).

4.1. Perfis do CLDC

Conforme vimos o único perfil do CLDC é o MIDP, a seguir é explicado como funciona os dois tipos de MIDPs e suas aplicações os *MIDlets* e o *MIDlets Suite*.

4.1.1. MIDP 1.0

O MIDP é um perfil da configuração CLDC endereçado a uma classe de dispositivos conhecidos como Dispositivos de Informação Móveis *(mobile information device*, ou MIDs), como por exemplo, telefones celulares e *pagers*.

Os dispositivos que suportem MIDP devem atender a alguns requisitos mínimos: memória suficiente para execução de aplicativos MIDP; um visor monocromático ou colorido

com resolução mínima de 96 pixels de largura por 56 pixels de altura; um teclado ou *touch screen*; e conectividade sem fio. Em geral, todos os dispositivos sem fio se encaixam nos requisitos do MIDP, inclusive PDAs. Contudo, essa última classe de dispositivos contém uma variedade de funções e recursos adicionais que são mais bem explorados com o uso de outros perfis, como o PDAP *(Personal Digital Assistant Profile*), atualmente em estágio de desenvolvimento.

Atualmente, estão definidas duas versões do MIDP, a 1.0 e a 2.0, ambas aprovadas pelo JCP. Ambas as versões fornecem amplo suporte na industria das telecomunicações. O MIDP adiciona algumas características ao CLDC:

- Suporte a administração do ciclo de vida de uma aplicação similar aos *applets*;
- Armazenamento de dados persistente;
- Conectividade em redes baseada no *framework* de conexão do CLDC;
- Suporte a simples interface com usuário.

Conforme mencionado, o MIDP define um novo grupo de aplicações similares aos *applets*, chamadas de *midlets*. Os *midlets* devem seguir regras rígidas de inicialização e de destruição especificadas em sua API.

4.1.2. MIDP 2.0

O padrão ***MIDP 2.0 New Generation*** oferece novos recursos como aumento de performance, melhorias nas APIs para interfaces gráficas , maior segurança e APIs para jogos (funcionalidades 2D) e conectividade, tornando possível o desenvolvimento e a utilização de uma maior variedade de aplicações.

A segurança neste padrão está mais confiável. Através da assinatura criptográfica a *Suite* de *MIDlets* a ser baixada para o telefone fica mais confiável, validando a origem dos *MIDlets*, podendo ser assinados e tratados semelhante à J2SE.

O conceito de *sandbox* também foi melhorado. As aplicações confiáveis *(trusted)* tem acesso a recursos específicos, enquanto as não-confiáveis *(untrusted)* precisam de confirmação do usuário para acessar os protocolos HTTP e HTTPS.

Quanto à rede, além do HTTP, há suporte a cinco protocolos: HTTPS, Comunicação Serial, *Sockets*, *Server Sockets* e *Datagramas*, através do *Generic Connection Framework*.

Também suporta a tecnologia *Push*: MIDlets podem registrar um dispositivo para receber eventos (conexões de rede, mensagens enviadas), mesmo quando o dispositivo não esteja em modo de execução de aplicações Java.

É requerida a implementação do OTA *Recommended Practice* (*Over The Air*), padronizando o *download* e a instalação de *MIDlets* através do *browser* embutido. A padronização inclui notificações para o servidor quando as aplicações são instaladas, gerenciamento de atualizações de *MIDlets* e o suporte a sessões através de *URL-Rewriting*.

4.1.3. *Midlets*

Os *midlets* são as unidades básicas de execução no MIDP. Um *midlet* é uma classe que herda a classe `javax.microedition.MIDlet`**.** Os *midlets* possuem três diferentes estados cujas transições entre eles são condicionados por três métodos abstratos definidos na classe `MIDlet`: `startApp`, `pauseApp` e `destroyApp`. Os estados e as transições de estado de um *midlet* que estão ilustrados na Figura 4.

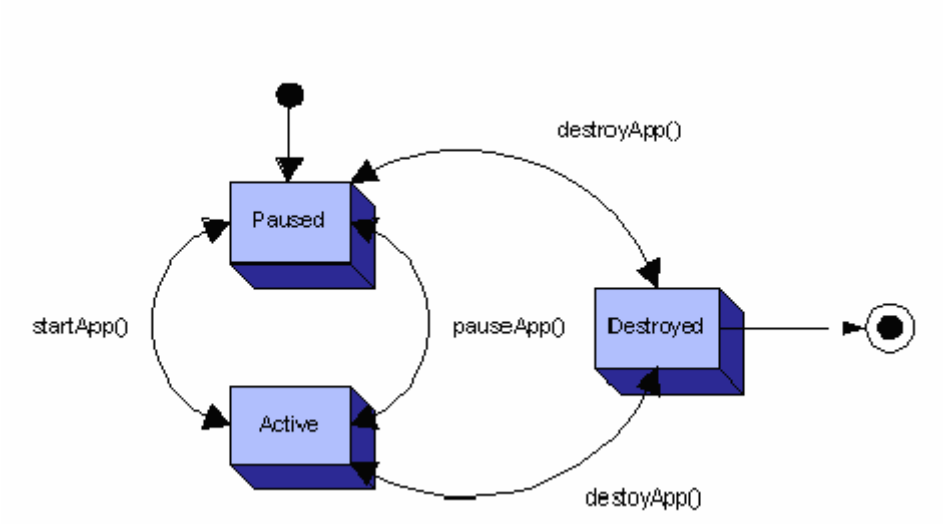


Figura 4: O ciclo de vida de um MIDLet

Os três estados possíveis de um *midlet* são: pausado *(paused*), ativado *(active*) e destruído *(destroyed*).

No estado pausado, o *midlet* aloca somente os recursos indispensáveis para sua hibernação. O *midlet* encontra-se pausado se não tiver sido invocado o método `startApp` ou como o resultado de um método `pauseApp`. As notificações assíncronas, como a sincronização do *timer*, podem ser recebidas neste estado.

Um *midlet* está ativado como resultado do método `startApp`. Todos os recursos para melhor desempenho de execução do *midlet* podem ser alocados neste estado.

O estado destruído é alcançado quando é invocado o método `destroyApp`. Após ter sido destruído, um *midlet* não pode transitar por nenhum outro estado. Esse estado substitui a tradicional saída de uma aplicação, `System.exit`.

As transições de estado podem ser disparadas pelo próprio *midlet* ou pelo programador da aplicação. Todos os métodos responsáveis pelas transições de estados são síncronos, portanto, um *midlet* só altera seu estado atual ao término bem sucedido da execução do método.

O método `startApp` é chamado uma única vez para ativar o *midlet* e também após a saída do estado de pausa. Em geral, os recursos opcionais que otimizem a execução do *midlet* são alocados durante a execução deste método.

Os dispositivos móveis podem se deparar com insuficiência de energia ou desconectividade transitória, nestes casos, o sistema invoca `pauseApp` o que leva o *midlet* a abrir mão da maioria de seus recursos e manter somente aqueles necessários para sua existência.

Já `destroyApp` é a forma tradicional através da qual o *midlet* encerra sua execução normal e é enviado para a coleta de lixo do sistema. Esse método pode ser condicional ou incondicional. Os *midlets* destruídos incondicionalmente, podem continuar em seu estado corrente do sistema e disparar uma exceção `MidletStateChangeExcpetion`. Por outro lado, os *midlets* destruídos condicionalmente abortam automat icamente, salvando qualquer dado persistente e liberando todos os recursos alocados.

Outros métodos podem causar transições de estados e estão no controle do programador da aplicação. O método `resumeRequest` indica que o *midlet* deve retornar ao estado ativo, por exemplo, após a expiração de um timer do sistema. Os métodos `notifyPaused` e `notifyDestroyed` informam ao sistema que o *midlet* entrou, respectivamente, no estado pausado e no estado de destruído.

4.1.4. Midlets Suites

Os *midlets suítes* definem uma coleção de *midlets* encapsulados em um arquivo comprimido JAR, de forma a atender alguns requisitos de segurança oriundos da interatividade entre os *midlets*.

Para preservar o modelo seguro do CLDC, os *midlets suites* tornam-se a unidade mínima de instalação, remoção e atualização de aplicações MIDP. Isso quer dizer que nenhum *midlet* pode ser, por exemplo, instalado ou atualizado ou removido sem que o seu *midlet suite* também o seja conjuntamente.

O conteúdo de um arquivo JAR descrevendo um *midlet suite* deve conter: as classes que implementam os *midlets*, os recursos usados pelos *midlets* e um arquivo de manifesto descrevendo o conteúdo do arquivo JAR. Em especial, o manifesto descreve uma forma de codificação dos atributos do *midlet suite* que descrevem o conteúdo da própria aplicação. Estes atributos estão mostrados na Tabela 3.

Tabela 3. Atributos dos *midlets*

Atributo	**Descrição**
MIDlet-Name	Identificação para o usuário
MIDlet-Version	Versão
MIDlet-Vendor	Nome do provedor
MIDlet-Icon	Ícone descrito por um arquivo PNG
MIDlet-Description	Descrição
MIDlet-Info-URL	URL com a documentação
MIDlet-<n>	Nome, ícone e classe dos *midlets* no arquivo JAR
MIDlet-Jar-URL	URL de origem do arquivo JAR
MIDlet-Jar-Size	Tamanho do arquivo JAR
MIDlet-Data-Size	Número minimo de dados persistentes
MicroEdition-Profile	Perfil do J2ME
MicroEdition-Configuration	Configuração do J2ME

4.2. Perfis do CDC

O conjunto de perfis implementado sobre a API do CDC, diferentemente dos perfis do CLDC, não visa atender aos requisitos específicos de cada dispositivo, e sim atender as características dos usuários dos dispositivos e as necessidades dos projetistas de aplicações. A

atual especificação do J2ME define os perfis *Foundation, Personal Basis* e*, Personal* para o CDC.

4.2.1. *Foundation Profile*

O *Foundation* é o perfil mais genérico do CDC e funciona como uma fundação ou base para o desenvolvimento de perfis mais sofisticados, como o *Personal* e o *Personal Basis*. A API do *Foudation* estende toda a funcionalidade do CDC e adiciona características específicas em nível de segurança e em nível de utilidade. Em primeiro lugar, a API do *Foundation* exclui o suporte a interfaces gráficas, como o AWT e o SWING, tornando-o específico para dispositivos embutidos conectados sem requisitos gráficos, como roteadores.

Por outro lado, o *framework* de segurança do *Foundation* inclui suporte para listas de controle, classes e interfaces para interpretar e administrar certificados de segurança e gerar chaves de criptografia RSA PKCS#1 e DAS NIST's FIPS-186.

O *Foundation* também inclui classes para leitura e escrita de arquivos comprimidos JAR e ZIP e suporte a conectividade baseada no *framework* de conexão genérica do CLDC, incluindo todos os protocolos CDC, soquete e HTTP.

4.2.2. *Personal Basis Profile*

O *Personal Basis* é o perfil intermediário do CDC e é um superconjunto do *Foundation* com três novas características adicionais: o modelo de aplicação *xlet*; um subconjunto da API AWT do J2SE composta de componentes de interface gráfica leves; e comunicação entre *xlets (Inter-Xlet Communication* – IXC) utilizando um subconjunto da API do RMI. Em suma, estas características facilitam o desenvolvimento de aplicações gráficas interativas com ciclos de vida bem definidos.

A API do *Personal Basis* inclui suporte aos pacotes AWT: `java.awt`, `java.awt.color`, `java.awt.event` e `java.awt.image`. Os pacotes `java.rmi` e `java.rmi.registry` são incluídos para suporte no modelo de comunicação entre *xlets*, entretanto, a comunicação via RMI é implementada pelo pacote opcional RMIOP.

O modelo *xlet* é adaptado da API Java TV e contém dois elementos básicos: as interfaces `Xlet` e `XletContext`, ambas incluídas no pacote `javax.microedition.xlet`. A classe principal do programa implementa a interface **`Xlet`** que define os eventos invocados pelo sistema, e a interface `XletContext` define as *callbacks* através das quais é possível obter informações do ambiente de operação do *xlet*. Os estados que um *xlet* pode assumir são os mesmo que um *midlet* pode assumir (ativo, pausado e destruído).

A criação de um *xlet* é composta da instanciação do *xlet*, em geral a partir de um construtor vazio, e da chamada ao método `initXlet` que possui como argumento uma referência ao `XletContext`. Se o *xlet* não puder ser inicializado, ele irá disparar uma `XletStateChangeException` para notificar o sistema que deverá ser destruído e removido da lista de aplicações.

Em seguida, o *xlet* pode entrar em seu estado ativo após uma chamada ao método `startXlet`. É no estado ativo que o *xlet* pode exibir sua interface gráfica para o usuário e alocar quaisquer recursos que possa necessitar. `Xlet` também inclui métodos para pausar um *xlet* (`pauseXlet`) e destruí-lo **(**`destroyXlet`). Usualmente, é o sistema quem realiza as transições de estado, porém o programador poderá invocar explicitamente os métodos `notifyPaused`, `notifyDestroyed` e `resumeRequest`, para respectivamente, mudar os estados do *xlet* para pausado, destruído e ativo.

Outra adição do *Personal Basis* é o IXC, que permite que dois ou mais *xlets* em execução na mesma máquina virtual possam compartilhar objetos e executar código em um contexto diferente. O IXC é baseado em RMI, ainda que a comunicação seja realizada interna em uma mesma máquina virtual, e não entre máquinas virtuais.

O compartilhamento de objetos no IXC é realizado por um processo chamado *binding* ou exportação, e ocorre quando um *xlet* escreve no registro do IXC todos os objetos disponíveis para compartilhamento. O registro é implementado pela classe `javax.microedition.xlet.ixc.Registry` e é um *singleton*, ou seja, único para todos os *xlets*. Este registro funciona como uma tabela *hash* compartilhada onde um *xlet* exporta um objeto com o método *bind* e busca um objeto com o método *lookup*. Após encontrar o objeto desejado, a máquina virtual retorna um *stub* que implementa a mesma interface do objeto real e funciona como um *proxy*.Por exemplo, quando uma aplicação invoca um *stub*, o sistema invoca o mesmo método na tabela de registro, ainda que seja em uma *thread* distinta.

4.2.3 *Personal Profile*

O *Personal* é o perfil mais específico e completo do CDC e é um superconjunto do *Personal Basis* com um ambiente semelhante ao J2SE. A API do *Personal* adiciona suporte a *applets* e estende seu suporte a todos os componentes gráficos incluídos no pacote AWT. Os pacotes adicionados são `java.applet` e `java.awt`, do J2SE. Os *applets* são uma nova alternativa para o desenvolvimento de aplicativos. Semelhantemente aos *midlets* e *xlets*, o *applet* possui um ciclo de vida bem definido com estados semelhantes e transições realizadas pelo sistema ou explicitamente pelo programador. Os *applets* executam em um ambiente seguro (*sandbox*) criados pelo navegador WEB no qual o *applet* está embutido.

5. Wireless Toolkit

Existem muitas ferramentas disponíveis no mercado para a construção de aplicativos J2ME, sendo em sua grande maioria distribuídas gratuitamente, como exemplo, as ferramentas dos fabricantes de celulares (Nokia, SonyEricsson, Motorola, etc.).

As IDEs (*integrated development environment*) são opcionais, pois o código pode ser feito em qualquer editor de texto e a compilação e pré-verificação através de linha de comando se o J2ME *Wireless Toolkit* da Sun *Microsystems* estiver instalado. Entretanto, a utilização de IDEs é bastante útil, pois torna a codificação mais visual e facilita a depuração e correção de erros.

A ferramenta utilizada nesse trabalho para desenvolvimento foi o *Wireless Toolkit* (WTK), que é uma ferramenta de desenvolvimento de aplicações para dispositivos com o perfil MIDP, em geral, telefones celulares fornecida pela *Sun Microsystems* (*WIRELESS TOOLKIT*, 2004).

Essa ferramenta oferece suporte para desenvolvimento nos pacotes WMAPI e MMAPI, e encontra-se atualmente na versão 2.2, compilável com a especificação MIDP 2.0. Adicionalmente, o WTK fornece ferramentas visuais para a pré-verificação, conversão de formatos de arquivos, testes e emuladores, além da possibilidade de integração de outros emuladores, adquiridos nos sites dos fabricantes e integrados ao ambiente de desenvolvimento.

5.1. Ambiente da Ferramenta

O ambiente mínimo de desenvolvimento incluso no WTK é o *KToolBar*, composto de uma interface gráfica para compilação, empacotamento e execução de *midlet*s. A execução é implementada através de emuladores, conforme ilustrado na Figura 5.

Figura 5. Emuladores do KtoolBar

O *KToolBar* é responsável por realizar a compilação, a depuração e a pré-verificação dos arquivos fontes dos *midlets* a partir do compilador J2SE SDK. Em particular, a depuração é realizada através dos emuladores disponibilizados, enquanto a pré-verificação é realizada pelo componente Pré-verificador, que é invocado logo após a compilação. O ambiente de trabalho do *KToolBar* é apresentado na Figura 6.

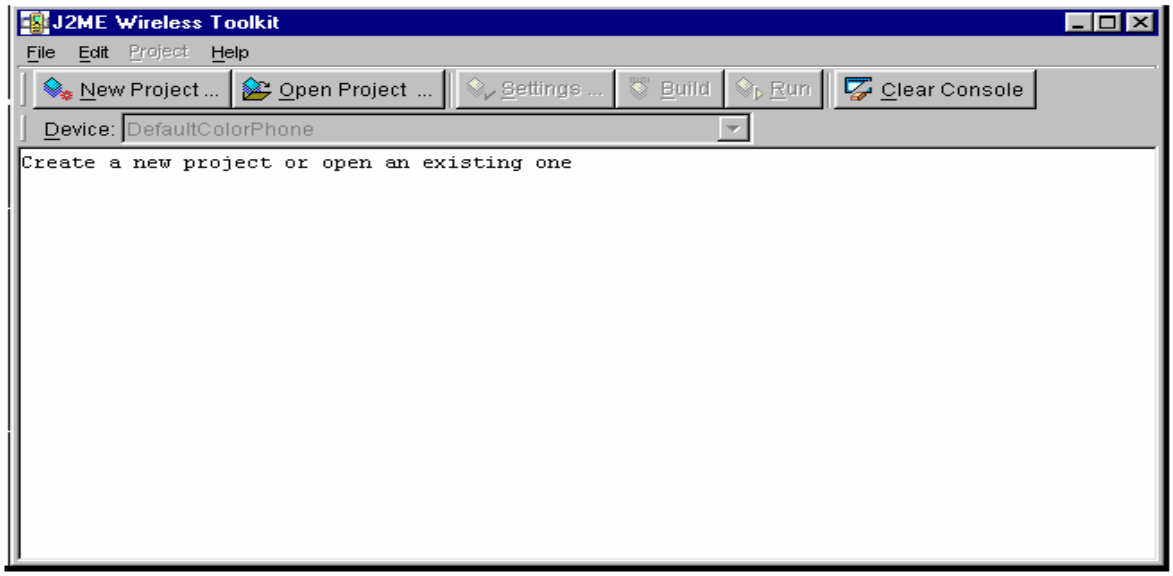


Figura 6 - Ambiente de Trabalho da KTOOLBAR

Outra funcionalidade do *KToolBar* é a o empacotamento dos *midlets* em *midlets suite*s. O processo de empacotamento distingue dois tipos de pacotes: os pacotes padrões e os

pacotes ofuscados, estes últimos reduzem o tamanho dos arquivos JAR através do ofuscamento das classes. Depois de empacotados nos formatos JAR ou JAD, o *midlets suite* pode ser testado no componente *OTA Provisioning (Over-the-Air Provisionin*g), que permite o acompanhamento de todo o processo de execução da aplicação, desde o servidor WEB até o dispositivo.

O *KToolBar* também fornece todos os procedimentos necessários para a criação e a administração de um *midlet suit*e. Por exemplo, a edição dos atributos e das permissões do *midlet suite* pode ser realizada através da interface exibida na Figura 7.

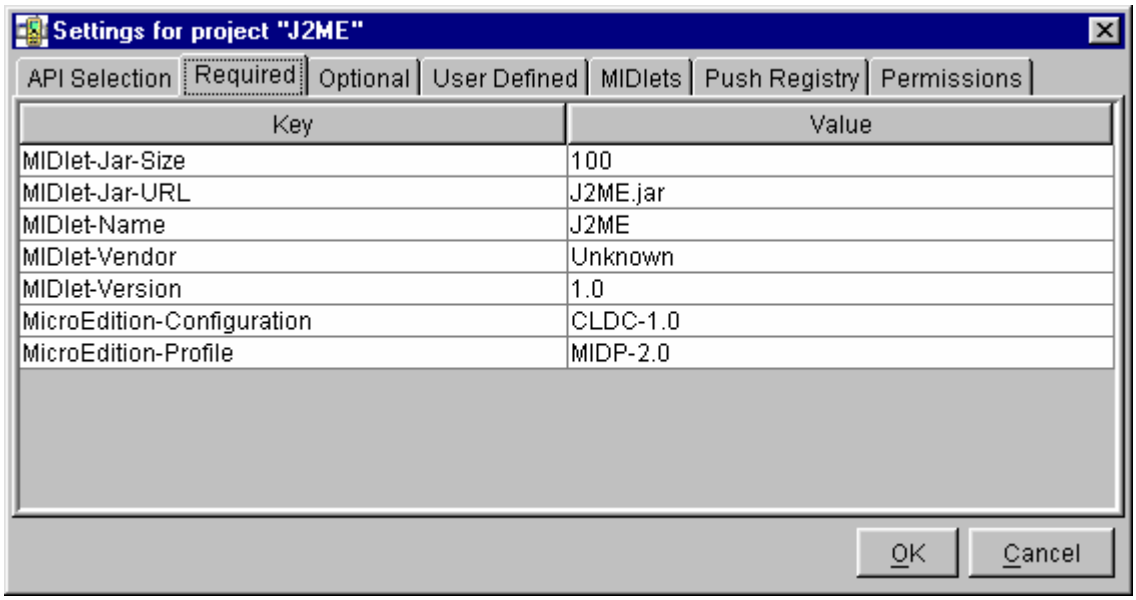


Figura 7. Edição dos Atributos de um Midlet Suíte

Outras funções para o gerenciamento e para a análise do *midlet suite* podem ser realizadas através de um pacote de utilitários *(Utilitie*s) que acompanha o WTK. Este pacote é ilustrado na Figura 8.

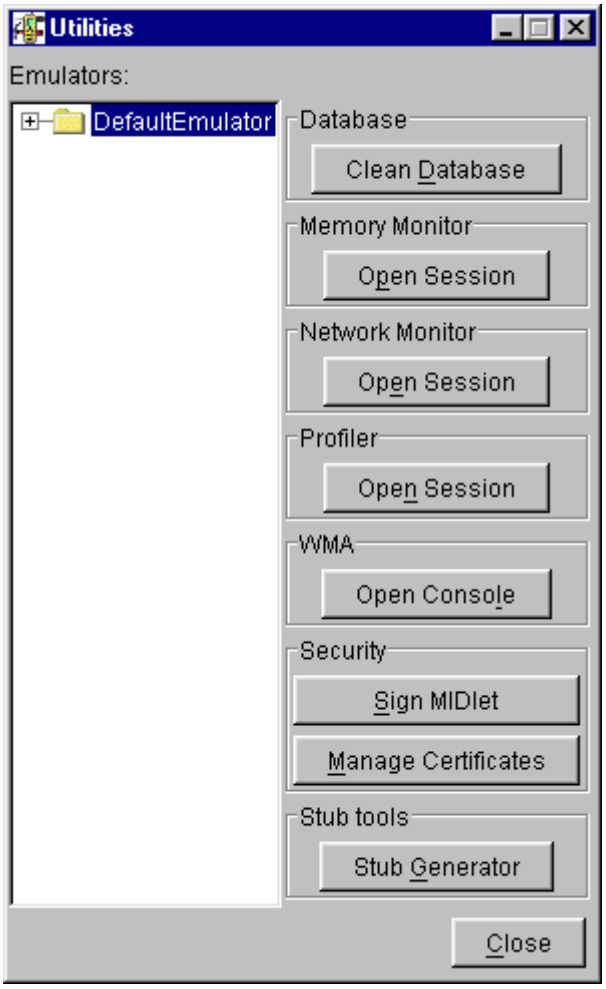


Figura 8. Utilitários do WTK

A lista da esquerda, conforme exibida na Figura 8 , permite a seleção do emulador desejado para a execução do *midlet suit*e. O utilitário também possui outros componentes que desempenham diversas funções conforme descrito na Tabela 4.

Tabela 4. Componentes da Ferramenta Utility

Componente	**Função**
Memory Monitor	Monitora o consumo de memória da aplicação e analisa o uso de memória por objeto.
Network Monitor	Examina as transmissões e a comunicação entre o dispositivo e a rede.
Profiler	Possibilita aperfeiçoar o desempenho de um *midlet suite* a partir da análise dos gargalos detectados na execução da aplicação.
WMA	Permite utilização do pacote opcional WMA *(Wireless Messaging AP*I) para envio de SMS e CBS.
Security	Administra certificado de segurança e permite registrar um *midlet* seguro.

6. ESTUDOS DE CASO

A fim de validar o estudo da linguagem feito neste trabalho, foram desenvolvidos alguns estudos de caso, com complexidade variável: O primeiro foi um simulador de despertador, o segundo foi um jogo da memória e o terceiro uma agenda telefônica, todos esses aplicativos foram implementados com a ferramenta J2ME *Wireless Toolkit* e estão descritos nas seções seguintes.

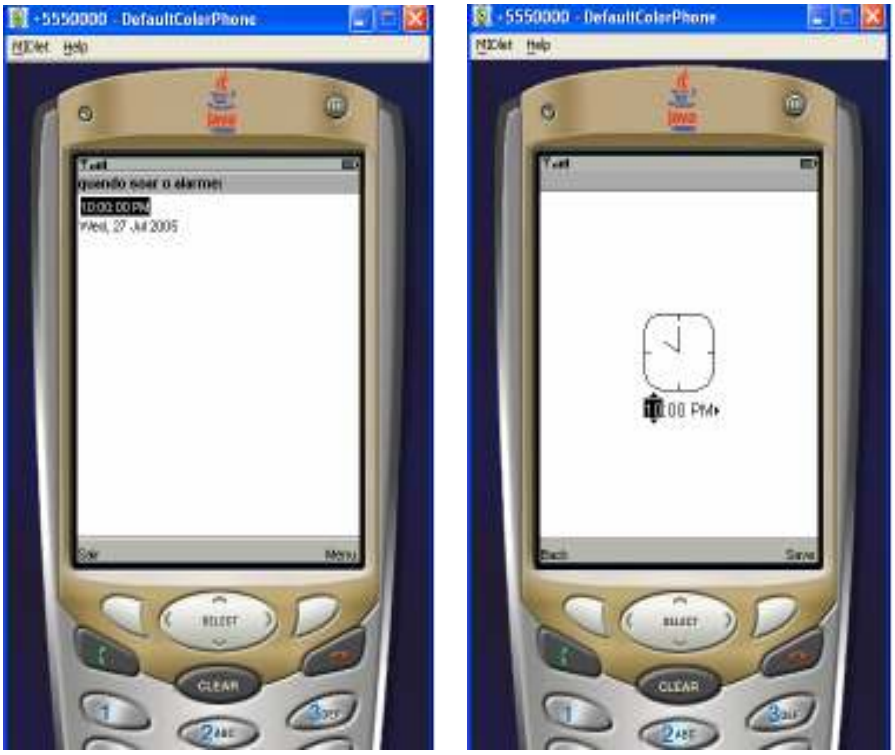
6.1. Despertador

O primeiro estudo de caso é um programa que simula o funcionamento de um despertador.

6.1.1. Descrição da Aplicação do Despertador

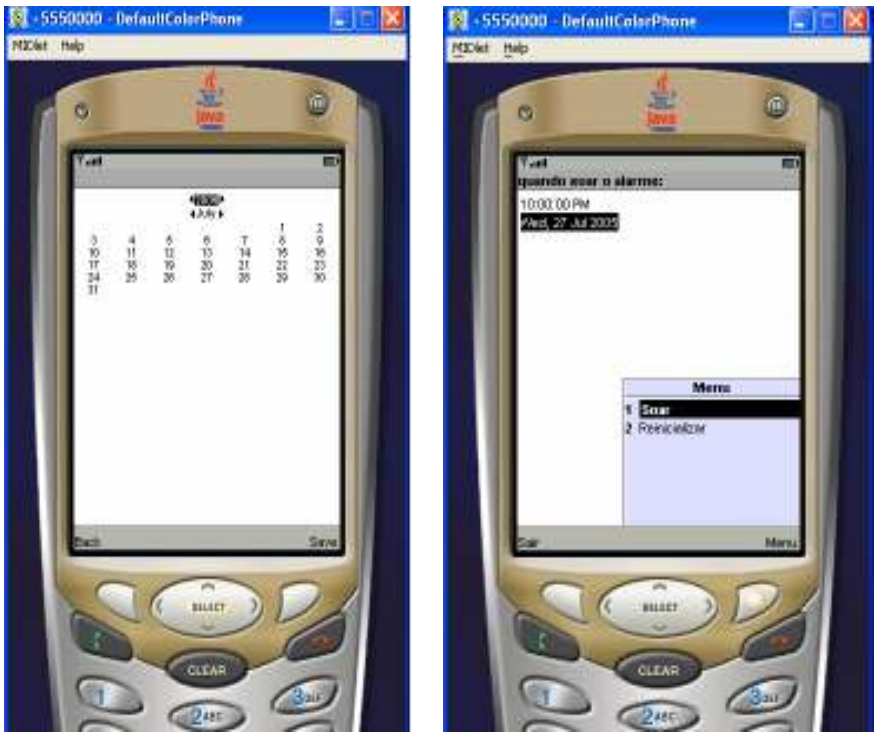
Para seu desenvolvimento, foi criada uma MIDlet que permite ao usuário especificar uma data e uma hora para soar um alarme, confirmar esses dados e, caso sejam validados, um alarme soará e uma mensagem será mostrada na tela quando o momento estabelecido pelo usuário chegar.

A tela principal, apresentada na Figura 9(a), será mostrada ao usuário quando iniciar a aplicação. Essa tela contém a hora e a data que deverão ser especificadas pelo usuário. Quando este desejar especificar a hora, ele deverá selecionar a hora e clicar no botão *select*, como é mostrado na Figura 9(b). Caso o usuário deseje especificar uma data, deve-se utilizar o mesmo processo, substituindo a hora pela data, como é mostrado na Figura 10(a). Também temos dois botões: o botão de sair que é utilizado para sair da aplicação e o botão do menu que pode ser utilizado tanto para iniciar o despertador, escolhendo a opção “soar”, quanto para reiniciar o despertador, selecionando a opção “reiniciar”, como mostrado na Figura 10(b).



(a) (b)

Figura 9: Telas principal e para ajustar a hora.



(a) (b)

Figura 10: Ajustar a data e tela do menu

6.1.2. Detalhes da Implementação do Despertador

A implementação dessa aplicação utilizou dois componentes importantes: o *Timer* e o *Alert*. Com um objeto *Timer*, é possível agendar tarefas para que ocorram em algum momento no futuro (nesse caso, a exibição da mensagem). A mensagem exibida é parte de um componente *Alert*. O objeto *Alert* é uma mensagem informativa ao usuário, ou seja, é uma "caixa de dialogo" que mostra uma mensagem e logo depois sai da tela, isto ocorre quando é estabelecido um tempo pelo usuário.

Ainda, foi utilizado um objeto *DateField,* iniciado com a data e a hora atuais, na implementação desse estudo de caso. Na Figura 11 mostra-se o trecho de código correspondente à inicialização desse objeto.

```
currentTime = new Date();
   //Campo DateField com a data de hoje como default
   dfSonecaTime = new DateField("", DateField.DATE_TIME);
   dfSonecaTime.setDate(currentTime);
```

Figura 11: Inicialização da hora e da data atuais

O objeto *dfSonecaTime*, da classe *DateField* é criado passando *DateField.Date_Time*, como parâmetro. Em seguida, o usuário tem a opção de alterar a data e a hora. Para validar o alarme, é necessário assegurar que a data estabelecida para despertar seja maior que a data atual. Para isso, o tempo estabelecido pelo usuário é comparado com o tempo atual, armazenado na variável *currentTime*. Se for válido, a variável *booleana dateOK* recebe valor verdadeiro, caso contrário, essa variável recebe o valor falso. Na Figura 12 mostra-se o trecho de código correspondente a essa operação.

```
if(dfSonecaTime.getDate().getTime()< currentTime.getTime())
             dateOK = false;
         else
            dateOK = true;
```

Figura 12: Validação do horário do alarme

Como mostrado na Figura 12, para obter o tempo do componente `dfSonecaTime` é necessário obter uma referência para um objeto `Date`, usando o método `dfSonecaTime.getDate()`. A obtenção do tempo a partir do objeto `Date`, por sua vez, é feita através do método `dfSonecaTime.getDate().getTime()`. Em seguida, basta comparar o tempo selecionado pelo usuário com o tempo atual, armazenado na variável `currentTime`.

Quando a opção *soar* do menu é selecionada, o método *commandAction()* é invocado. É necessário verificar se a data e a hora selecionadas são válidas, se não forem, cria-se um objeto *Alert* para avisar ao usuário que coloque uma data e hora diferente. Se tudo estiver configurado começamos a programar o cronômetro e determinar quanto tempo demorará a despertar, como mostra o trecho de código apresentado na Figura 13.

```
//cria um novo cronometro
tmSoneca = new Timer();
ttSoneca = new SonecaTimer();

//quantidade de tempo de espera
long amout = dfSonecaTime.getDate().getTime() -
currentTime.getTime();
tmSoneca.schedule(ttSoneca,amout);
```

Figura 13: Criação de um novo cronômetro e configuração do tempo de espera

O método `getTime()` foi usado para determinar quanto tempo passará até que o alarme soe. A quantidade de tempo é a diferença entre a hora selecionada pelo usuário e a hora atual.

Na hora selecionada pelo usuário é chamado o método *run()* dentro da classe `SonecaTimer`, neste método é criado um novo objeto *Alert*, e também é configurada a tela atual onde será mostrada as modificações, e o cancelamento do cronômetro, como mostra o trecho de código na Figura 14.

```
public final void run()
                {
  Alert al = new Alert ("hora de acordo");
  al.setTimeout(Alert.FOREVER);
  al.setType(AlertType.ALARM);
  AlertType.ERROR.playSound(display);
  display.setCurrent(al);
  //cancela essa tarefa do cronometro
  cancel();
                                    }
```

Figura 14: Método run

6.2. Jogo da Memória

O segundo estudo de caso a ser descrito é o jogo da memória. Trata-se de uma aplicação com um grau de complexidade mais elevado do que o despertador.

6.2.1. Descrição da Aplicação do Jogo da Memória

O jogo da memória funciona da seguinte forma: inicialmente, são apresentadas trinta e duas figuras ao usuário. Utilizando as teclas de ação (*up*, *down*, *right* e *left*) o usuário movimenta-se para uma direção escolhida. Quando posicionado sobre uma figura desejada, ele pressiona a tecla *select,* que faz com que uma outra figura, previamente oculta, seja revelada. O objetivo do jogo é descobrir pares de figuras iguais. Quando todos os pares forem descobertos o jogo acaba.

Na figura 15(a) mostra-se a tela principal do jogo e na figura 15(b) mostra-se a tela que representa o jogo em andamento.

(a) (b)

Figura 15: Telas do Jogo da Memória

6.2.2. Detalhes da Implementação do Jogo da Memória

Esta aplicação utiliza as classes `Canvas` e `Graphics` para criação de elementos gráficos personalizados, sistemas de coordenadas, criação de objetos na tela e tratamento de eventos.

A aplicação foi desenvolvida de acordo com os seguintes critérios: foi criado um vetor de imagens com trinta e duas figuras, destas, dezesseis são figuras do jogo (as que deverão formar pares iguais), enquanto as outras dezesseis são aquelas que se sobrepõem às figuras do jogo. O código para a criação deste vetor é mostrado na Figura 16.

```
  //Cria vetor de imagem
      lights = new Image[32];
      try
      {
      //Cria as imagens
       lights[0] = Image.createImage("/imagem_r.jpg");
       lights[1] = Image.createImage("/imagem_r.jpg");
       lights[2] = Image.createImage("/imagem_r.jpg");
       lights[3] = Image.createImage("/imagem_r.jpg");
       lights[4] = Image.createImage("/imagem_r.jpg");
       lights[5] = Image.createImage("/imagem_r.jpg");
       lights[6] = Image.createImage("/imagem_r.jpg");
       lights[7] = Image.createImage("/imagem_r.jpg");
       lights[8] = Image.createImage("/imagem_1.jpg");
       lights[9] = Image.createImage("/imagem_c.jpg");
       lights[10] = Image.createImage("/imagem_4.jpg");
       lights[11] = Image.createImage("/imagem_d.jpg");
       lights[12] = Image.createImage("/imagem_2.jpg");
       lights[13] = Image.createImage("/imagem_j.jpg");
       lights[14] = Image.createImage("/imagem_n.jpg");
       lights[15] = Image.createImage("/imagem_5.jpg");
       ...
        }
  catch(Exception exception)
  {
  System.out.println("Error: Não é possivel localizar ou
abrir as imagens : " +exception.toString());
  }
```

Figura 16: Código de criação do vetor de imagem

Uma vez criadas, essas imagens precisam ser desenhadas na tela. Para isso, foi utilizado o método `Paint` do componente `Graphics`. O método `Paint` invoca o componente *`drawImage`*, que serve para inserir imagens na tela especificando suas coordenadas, sendo que a altura e a largura passadas como parâmetro devem ser correspondentes às dimensões das imagens. Esse método também pode ser utilizado para manipular cores da tela, fontes e *strings*.

Na Figura 17 mostra-se o trecho do código, onde e como as imagens são alocadas na tela.

```
public void paint(Graphics g)
{
 g.setColor(0, 0, 0);
 g.fillRect(0, 0, width, height);

 if (r[0] == 0)
 {g.drawImage(lights[0], center_width - 42, center_height -
42,0);}
else
 {g.drawImage(lights[8], center_width - 42, center_height -
42,0);}
 if (r[1] == 0)
 { g.drawImage(lights[1], center_width, center_height - 42, 0);
else
{g.drawImage(lights[9 ], center_width, center_height - 42, 0);}
 ...
 }
```

Figura 17: Trecho do código que desenha as imagens na tela

Observando o trecho de código apresentado na Figura 17, verifica-se que o componente *drawImage* é invocado várias vezes. Cada condição estabelecida serve para desenhar duas imagens na tela, uma que será utilizada para formar os pares e a outra é a que será sobreposta a ela, e suas respectivas coordenada. Concluindo que ao clicar numa imagem outra aparecerá sobreposta a ela, como mostrado na Figura 15.

Todas as classes do jogo da memória descritas acima estão apresentadas no diagrama de classes, como mostrado na Figura 18.

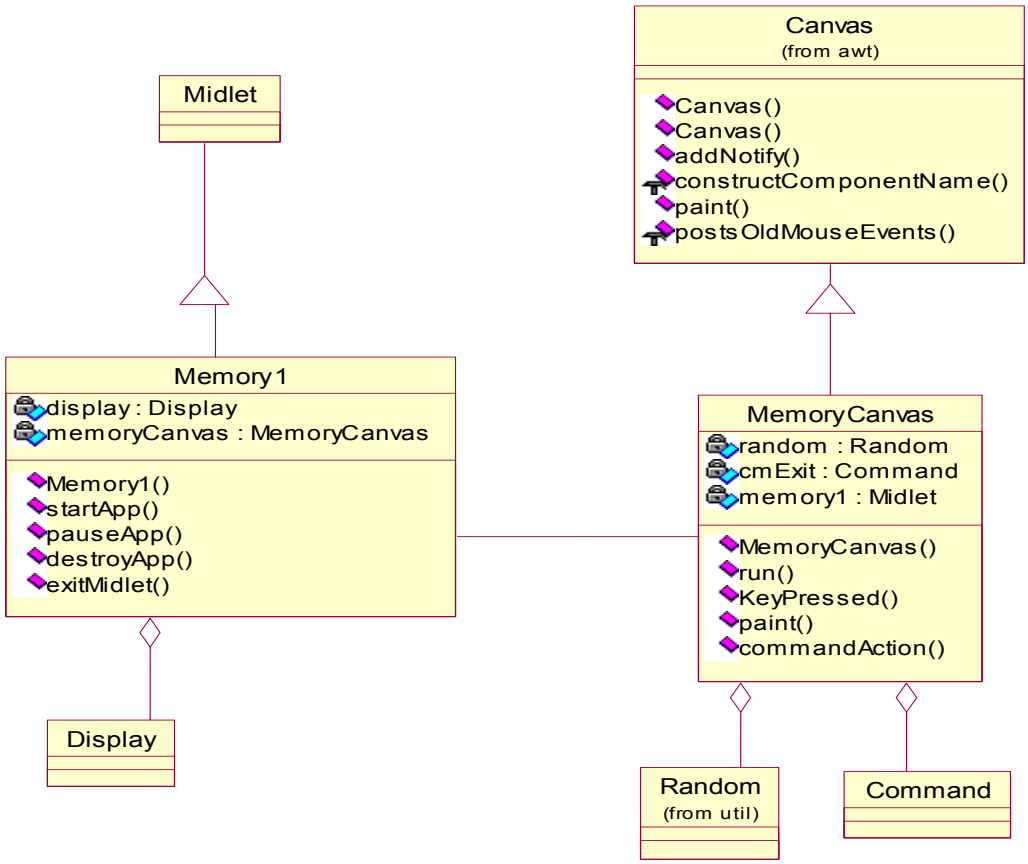


Figura 18: Diagrama de classe do jogo da memória

6.3. Agenda Telefônica

O terceiro estudo de caso é uma agenda telefônica, que pode ser utilizada em *pagers* ou em telefones celulares, e que apresenta as seguintes funcionalidades:

- Exibir detalhes de um contato;
- Incluir um contato;
- Alterar um contato;
- Excluir um contato;
- Sair da Agenda.

Essas funcionalidades são apresentadas, utilizando notação UML, pelo diagrama de casos de uso da Figura 19.

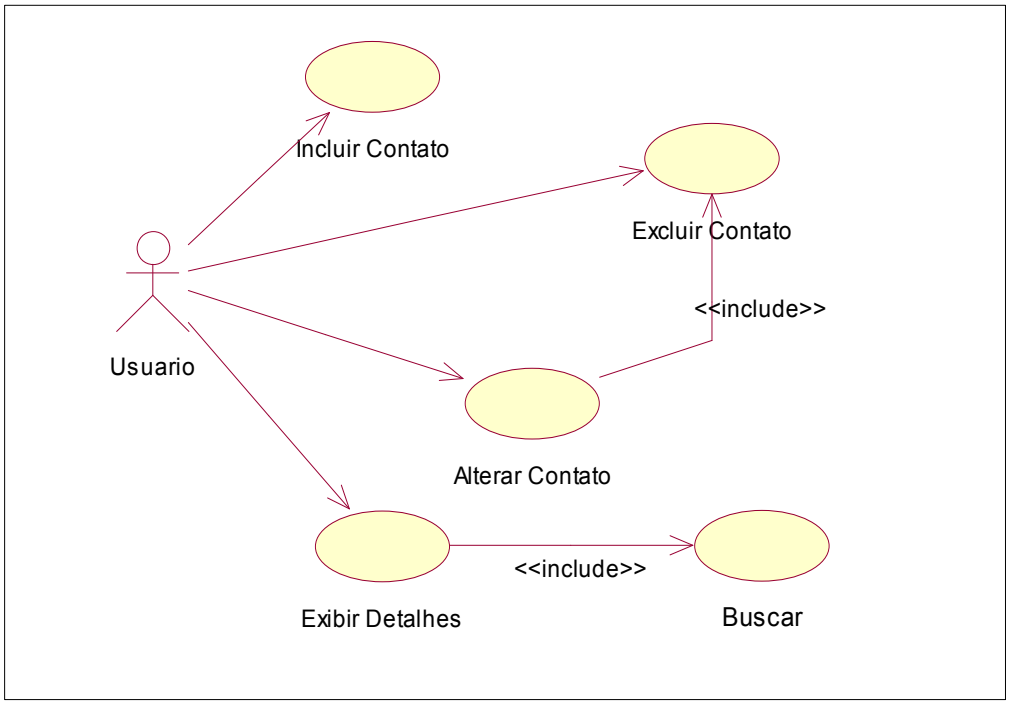


Figura 19: Diagrama de casos de uso

Esta aplicação é mais complexa do que o despertador e o jogo da memória, pois sua implementação utiliza objetos de diversas classes diferentes, tais como `Form`, `TextField`, `List`, `Ticker` e o `RMS` *(Record Management System* - Sistema de Gerenciamento de Registro).

6.3.1. Descrição da Aplicação Agenda Telefônica

A agenda telefônica funciona da seguinte forma: quando o usuário desejar incluir um novo contato na agenda, este deve clicar na opção “Menu” e selecionar o item “incluir”. Essa seleção faz com que apareçam os campos para serem preenchidos pelo usuário. Uma vez digitadas as informações, o usuário deve escolher a opção “salvar”. Caso não ocorra nenhum erro, será mostrada uma mensagem informando que a inclusão foi efetuado com sucesso, como é mostrado na Figura 20.

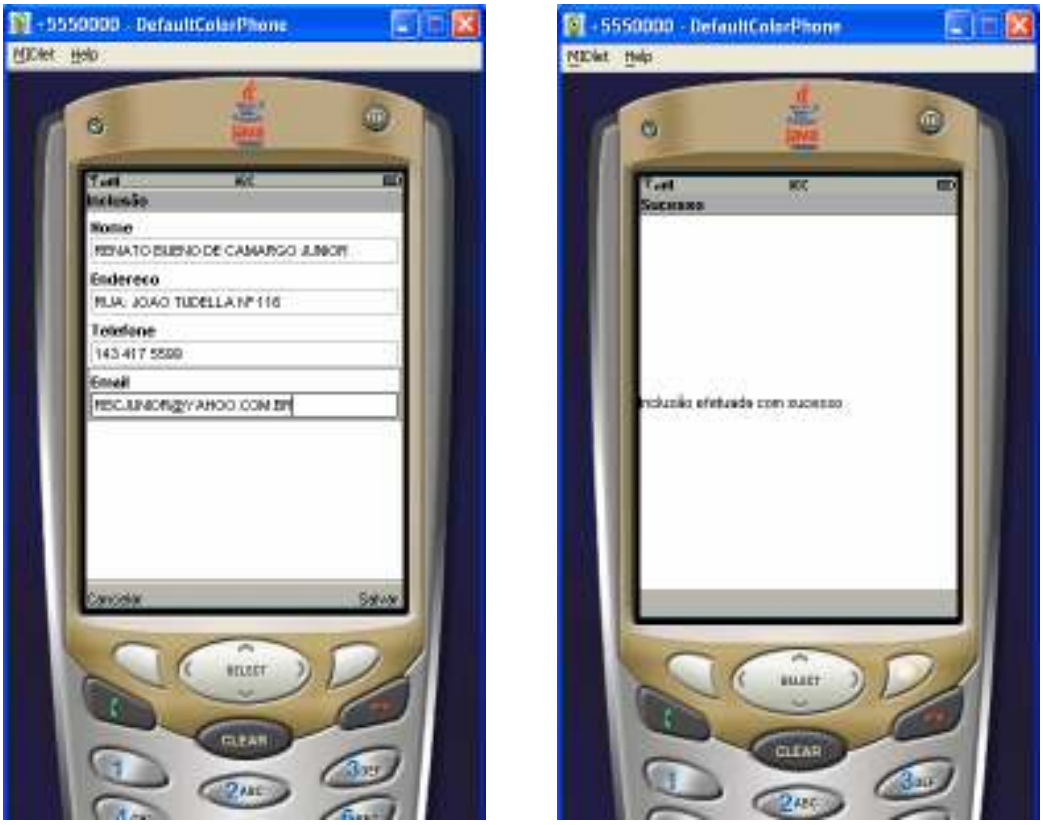


Figura 20: Telas de inclusão

Caso o usuário tenha digitado alguma informação incorreta e pretende corrigi-la, é necessário que ele escolha o item “alterar” do “Menu”. Uma vez corrigida a informação, deve-se optar por “salvar”. Será mostrada uma mensagem que a alteração foi efetuada com sucesso, como mostrado na Figura 21.

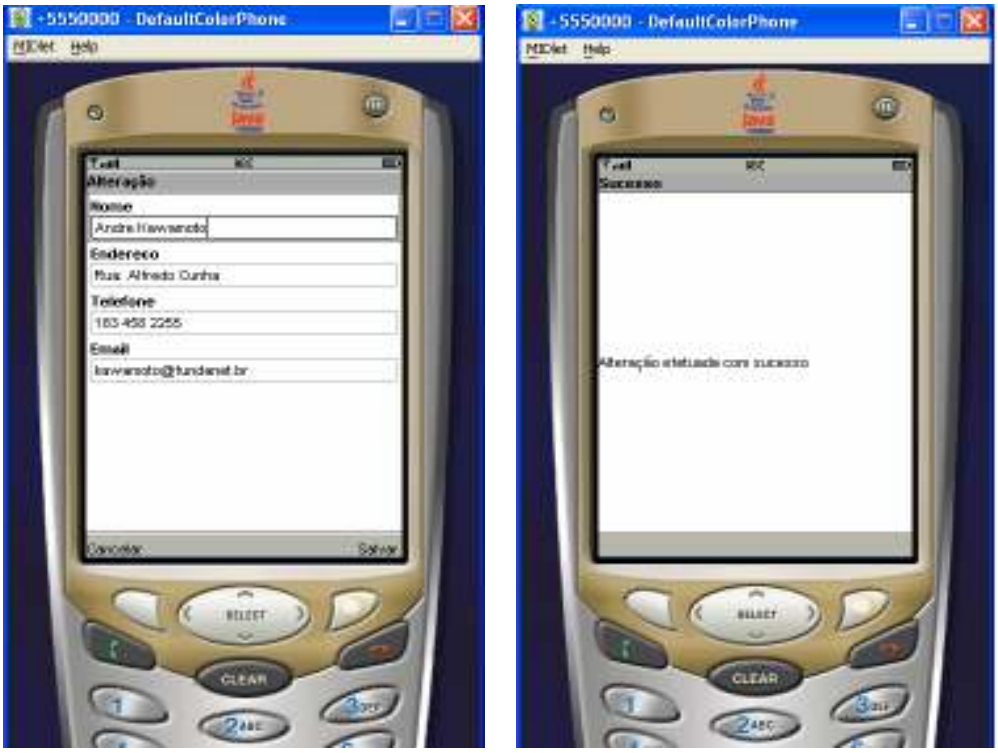


Figura 21: Telas de alteração

Quando o usuário desejar excluir alguns contatos, ele deve navegar até que se encontre no contato que ele deseja excluir, então pressionar a tecla “Menu” e selecionar o item “excluir”. Isso faz com que esse contato seja eliminado da memória, liberando aquele espaço. Será mostrada, então, uma mensagem de que a exclusão foi efetuada com sucesso, como mostrado na Figura 22

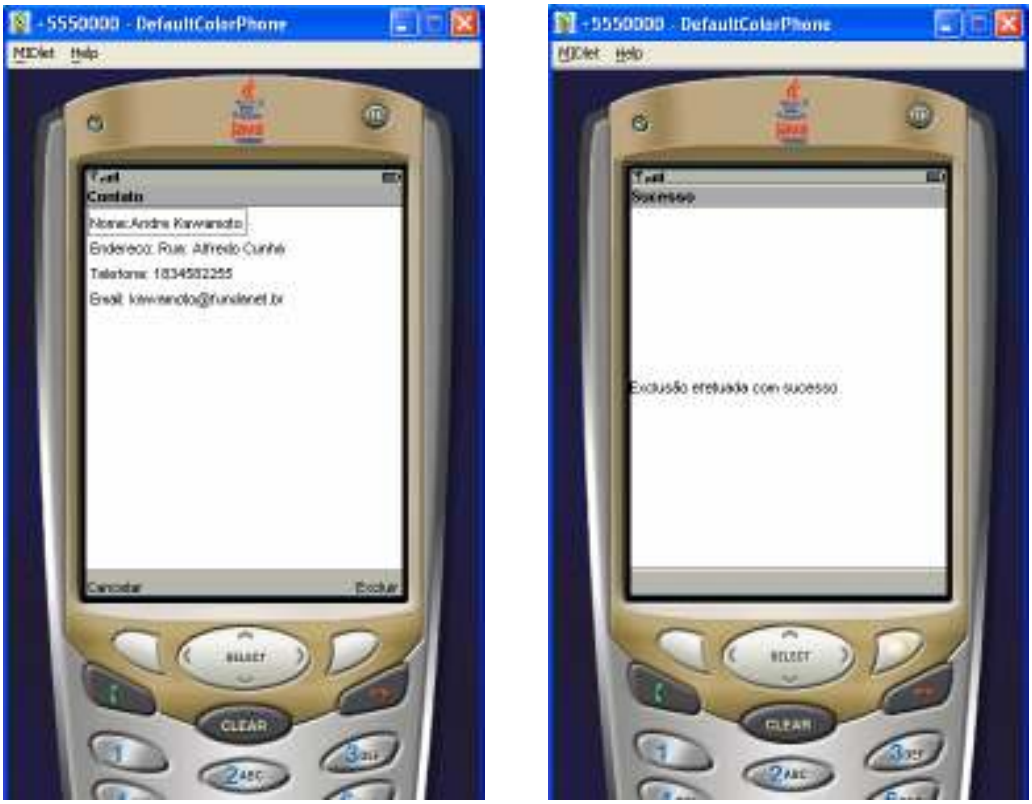


Figura 22: Telas de exclusão

Na Figura 23 mostra-se a tela de detalhes de um contato. Essa tela aparece quando o usuário clica na tecla “Menu” e seleciona o item “detalhes”. Uma vez nesta tela, é possível, clicando na tecla Menu outra vez, que o usuário altere, exclua um contato, ou simplesmente retorne para a tela principal.

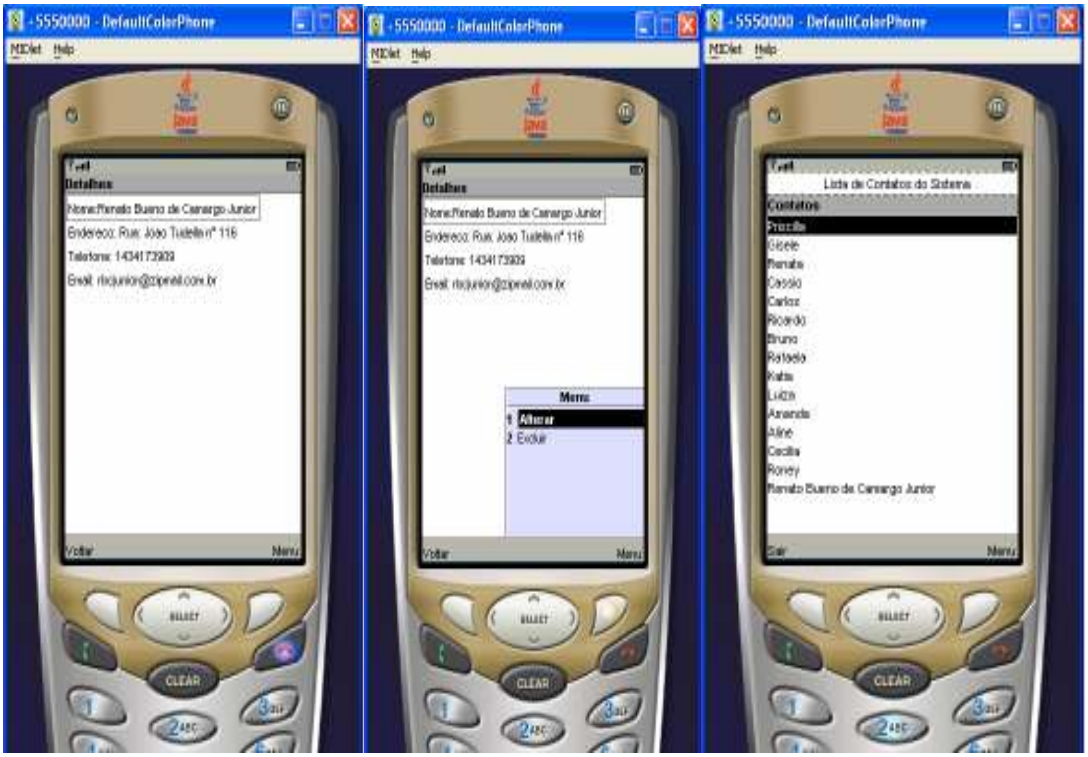


Figura 23: Telas de detalhes

6.3.2. Detalhes da Implementação da Agenda Telefônica

Um objeto da classe `Form` foi utilizado para criar cada um dos formulários da agenda e os anexar na tela. Dentro dos formulários de incluir e alterar são mostrados os componentes `TextField`, para o preenchimento dos campos da agenda, que são nome, endereço, telefone e email, tendo cada um deles suas restrições, por exemplo, a restrição ANY indica que esse campo pode ser preenchido com qualquer tipo, um campo com restrição PHONENUMBER só pode ser preenchido com números. O trecho de código mostra os campos do formulário na Figura 24.

```
//cria as caixas de texto em branco para o usuário digitar
tfNome = new TextField("Nome","",30,TextField.ANY);
fEndereco = new TextField("Endereco","",30,TextField.ANY);
tfTelefone  = new
TextField("Telefone","",30,TextField.PHONENUMBER);
tfEmail = new TextField("Email","",30,TextField.ANY);

//adiciono componentes no formulário inclusão
   formInc.append(tfNome);
   formInc.append(tfEndereco);
   formInc.append(tfTelefone);
   formInc.append(tfEmail);
```

Figura 24: Trecho do código da criação de caixa de textos na tela

Para permitir ao usuário selecionar itens (elementos) de uma lista foi utilizado um objeto da classe `List`. Na implementação dessa aplicação, os elementos dessa lista eram *strings*, porém essa classe permite ainda que sejam utilizadas imagens.

Lista Implicit deixa você escolher uma opção e clicar no botão padrão de seleção. Na Figura 25 mostra-se como uma lista *implicit* é criada.

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

```
//Logica para criar e montar a lista
listaNomes = new List("Contatos",List.IMPLICIT);
```

Figura 25: Cria e monta uma lista implicit

O *Alert* está sendo utilizado em todas as telas da agenda, quando é feita uma inclusão, uma alteração, ou exclusão, aparece uma mensagem informando ao usuário que a inclusão, a alteração e a exclusão foram efetuadas com sucesso. Como é mostrado na Figura 26.

```
//Exibe Alert indicando a adição do registro
Alert salvar = new Alert("Sucesso","Inclusão efetuada com
sucesso",null, null);
salvar.setTimeout(3000);
```

Figura 26: Exibindo o funcionamento do alert

A classe *Ticker* foi usado para escrever um texto de rolagem contínua na parte superior da tela. Mostrado na Figura 27.

```
//Incluir uma mensagem na primeira linha da tela
 Ticker msg = new Ticker("Lista de Contatos do Sistema");
 listaNomes.setTicker(msg);
```

Figura 27: Trecho de código que inclui mensagem na tela

Esta aplicação utiliza o pacote **`javax.microedition.rms`,** RMS, que contém as classes necessárias para implementar uma base de dados de armazenamento persistente no dispositivo. Essa base de dados é chamada de persistente, porque mesmo quando o dispositivo sofre *reboots*, troca de bateria ou desligamentos, os dados gravados permanecem inalterados. A base é limitada em suas potencialidades para armazenar e recuperar a informações devido às limitações do dispositivo.

Como uma alternativa ao uso de um sistema de arquivo, o RMS utiliza memória não-volátil para armazenar informações. Esse banco de dados orientados para registro, frequentemente referido como arquivo puro, pode ser imaginado como uma série de fileiras em uma tabela, com um identificador exclusivo para cada fileira. Como é mostrado na Figura 28.

ID de Registro	Dados
1	Array de bytes
2	Array de bytes
3	Array de bytes
.....	.....

Figura 28: Como os dados ficam armazenados

Cada registro consiste em um ID de registro, um valor inteiro que desempenha o papel de chave principal para o banco de dados e um *array* de *bytes* para armazenar os dados do registro.

Na Figura 29 apresenta-se a parte do código responsável por efetuar a busca dos dados de cada contato armazenado. (A)Para isso é aberto o armazém de registro, logo em seguida é usado um `RecordEnumeration` para percorrer os registros do RMS, assim que

os registros forem percorridos o número do identificador é armazenado e passa para o próximo id, depois recolhe os detalhes do registro atual e transforma os dados em *string*. (B)Entrando em *loop* até achar o nome igual ao selecionado na lista, depois é feito o tratamento para separar os quatro campos, e depois de tudo feito o banco de dados é fechado, como mostra a Figura 29.

```
private void buscarDados(String s){

try {
RecordStore rsContato =
RecordStore.openRecordStore("rms1",false);

byte [] dados = null;
//Cria um RecordEnumeration para percorrer os registros do
RMS
RecordEnumeration reContato =
rsContato.enumerateRecords(null,null,false);
//armazena o RecordId
id = reContato.nextRecordId();                          (A)
//pega detalhes do registro atual
dados = rsContato.getRecord(id);
//transforma os dados em String
String aux = new String(dados);

//Entra em looping até achar o nome igual ao Selecionado no
List
While(reContato.hasNextElement()&&!s.equals(aux.substring(0
,aux.indexOf("|")))) {

id = reContato.nextRecordId();
dados = rsContato.getRecord(id);
aux = new String(dados);                                (B)
}

//faz tratamento para separar os quatro campos
int i = aux.indexOf("|");
nome = aux.substring(0,i);
endereco = aux.substring(i+1,aux.indexOf("|",i+1));
i = aux.indexOf("|",i+1);
telefone= aux.substring(i+1,aux.indexOf("|",i+1));
i = aux.indexOf("|",i+1);
email = aux.substring(i+1,aux.length());

//fecha o banco de dados
rsContato.closeRecordStore();

} catch (Exception e) {
listaNomes.append("Lista Vazia",null);
}
}
```

Figura 29: Trecho de código do buscarDados

Para adicionar registros ao `recordStore` aberto, o método utilizado é o `addRecord`. O valor retornado pelo método `addRecord` é um identificador único para o registro adicionado. Os identificadores de registros começam em 1 e são incrementados a cada registro adicionado. Depois que os registros foram adicionados o `recordStore` é fechado. O código dessa operação é mostrado na Figura 30.

```
Adicionar os registro no banco de dados
private void tratarSalvar() {

try {

RecordStore rsContato =
RecordStore.openRecordStore("rms1",true);
String registro = nome + "|" + endereco + "|"+ telefone+ "|" +
email;
rsContato.addRecord(registro.getBytes(),0,registro.length());
rsContato.closeRecordStore();

} catch (Exception e) {
System.out.println(e.getMessage());
}

//Exibe Alert indicando a adição do registro
Alert salvar = new Alert("Sucesso","Inclusão efetuada com
sucesso",null, null);
salvar.setTimeout(3000);
//Monta a lista inicial
montarLista();
//redireciona a tela
display.setCurrent(salvar,listaNomes);
}
```

Figura 30: Trecho de código de inclusão

Para alterar um contato é necessário excluir o registro anterior na base de dados, que o usuário desejar alterar, para isso é utilizado o método `tratarexcluir` que está sendo explicada em seguida. Com a base de dados limpa os campos serão reconstruídos e assim podem ser preenchidos corretamente, adicionando o registro alterado na base de dados `tratarSalvarAlt`. Será mostrada uma mensagem que a operação de alteração foi efetuada com sucesso, depois disso monta-se uma nova lista e a redireciona a tela, como mostra a Figura 31.

```
private void tratarSalvarAlt(){

//exclui o registro na base de dados (Anterior)
tratarExcluir();

//alterar os dados
nome = tfNome.getString();
endereco = tfEndereco.getString();
telefone= tfTelefone.getString();
email = tfEmail.getString();

//adiciona registro alterado na base de dados
tratarSalvar();

//Exibe um alert indicando a alteração
Alert salvar = new Alert("Sucesso","Alteração efetuada com
sucesso",null, null);
salvar.setTimeout(3000);
//monta uma nova lista
montarLista();
//redireciona a tela
display.setCurrent(salvar,listaNomes);
}
```

Figura 31: Trecho do código da alteração

Para excluir um registro do `recordStore`, o método `deleteRecord` é chamado. É necessário informar o id do registro a ser excluído como parâmetro para esse método. Depois que o registro é excluído o `recordStore` é fechado, como mostra a Figura 32.

```
private void tratarExcluir(){

try {

RecordStore rsContato =
RecordStore.openRecordStore("rms1",false);
//Exclui o registro
rsContato.deleteRecord(id);

rsContato.closeRecordStore();

} catch (Exception e) {
System.out.println("erro na inclusão");
}

//Exibe um alert indicando a exclusão do registro
Alert excluir = new Alert("Sucesso","Exclusão efetuada com
sucesso",null, null );
excluir.setTimeout(3000);

//Monta uma nova Lista
montarLista();
//Redireciona a tela
display.setCurrent(excluir,listaNomes);
}
```

Figura 32: Trecho do código da exclusão

O diagrama de classes da Figura 33 mostra a classe AgendaTelefonica, com seus atributos e métodos principais, e as classes Java associadas.

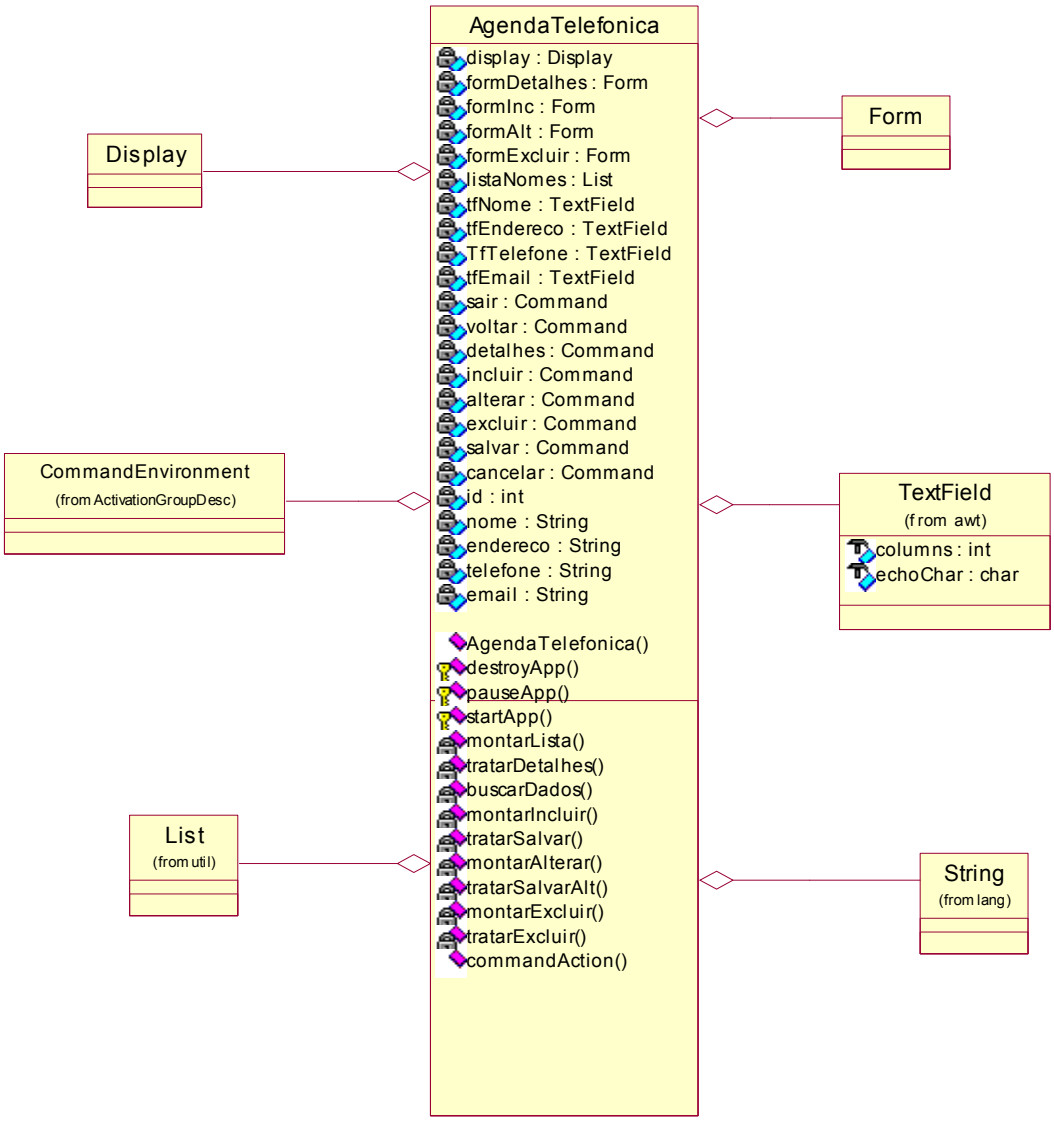


Figura 33: Diagrama de classe da Agenda Telefonica

7 Conclusão

O crescimento do mercado da informática atingiu, dentre outros dispositivos, àqueles com baixos recursos de memória e processamento e com funcionalidades específicas. Para este conjunto de dispositivos a *Sun Microsystems* desenvolveu a plataforma *Java 2 Micro Edition* (J2ME).

Este trabalho analisou em detalhes a arquitetura do J2ME e como ela é construída para atender a uma variedade de dispositivos que vão desde os telefones celulares e PDAs até sistemas embutidos, como fornos elétricos, por exemplo. Basicamente, os conceitos principais do J2ME são as configurações e os perfis.

As configurações agrupam as características similares a um conjunto de dispositivos horizontalmente compatíveis. Essas características incluem a quantidade de memória disponível, a capacidade de processamento do dispositivo e ainda o nível de conectividade com o meio externo. O J2ME especifica atualmente apenas duas configurações, o *Connected, Limited Device Configuration* (CLDC) e o *Connected Device Configuration* (CDC).

Os perfis especializam as configurações no sentido em que atendem a um conjunto de dispositivos verticalmente compatíveis, ou seja, dispositivos com características de alto nível semelhantes. Isso abrange, por exemplo, a capacidade de exibição de conteúdo gráfico do dispositivo e os protocolos que permitem que estes dispositivos se comuniquem com o meio externo. De posse disso, a Sun desenvolveu quatro perfis no contexto das duas configurações já existentes. O *Mobile Information Device Profile* (MIDP) é o único perfil da configuração CLDC, enquanto o *Foundation Profil*e, o *Personal Basis Profile* e o *Personal Profile* constituem uma hierarquia de perfis disponíveis para o CDC.

Adicionalmente, este trabalho apresentou o *Wireless Toolkit* (WTK) da Sun. Esta ferramenta fornece um ambiente de desenvolvimento de aplicações no perfil MIDP, os *midlet*s, e inclui componentes para realização de testes (emuladores) e para distribuição dos *midlet suites* desenvolvidos.

O uso da ferramenta mostrou-se vantajoso por fornecer uma interface gráfica para compilação, empacotamento e execução. Uma desvantagem do WTK a ser apontada é que essa ferramenta não possui um IDE (Ambiente de desenvolvimento integrado), com editor, depurador de código, etc.

Como estudo de caso, foram implementados três aplicativos, cada um com grau de complexidade diferente: despertador, jogo da memória e agenda telefônica. O despertador foi o aplicativo mais simples, e utilizou o método `date` para retornar a hora atual e os

componentes *Alert* e *Time,* para disparar o aviso sonoro e agendar tarefas futuras. A implementação do jogo da memória, por sua vez, foi mais complexa do que a do despertador, e utilizou as classes *Canvas* e *Graphics* para elementos gráficos personalizados. A maior dificuldade encontrada na implementação desse aplicativo foi ajustar as dimensões das figuras na tela e adequar o funcionamento das teclas de ação (*left*, *right*, *up* e *down*). Finalmente, a agenda telefônica foi o aplicativo mais complexo desenvolvido, e utilizou objetos de diversas classes que foram estudadas, tais como: *Form*, *TextField*, *List*, *Ticker* e o *RMS*. Neste aplicativo a dificuldade foi encontrar uma maneira de armazenar as informações em memória não volátil. Para isso, foi utilizado o RMS, que é um banco de dados orientado a registros.

TRABALHOS FUTUROS

A partir dos resultados obtidos com este trabalho, abre-se a possibilidade de implementação de outros aplicativos, como por exemplo, a utilização de classes *Bluetooth* para a transmissão do conteúdo da agenda para outros dispositivos que utilizam essa tecnologia.

Além disso, em outras áreas, outras aplicações podem ser destacadas. Na área médica são diversas as aplicações onde a mobilidade torna-se essencial. Uma aplicação interessante seria a de monitoramento de pacientes diabéticos através de um "sensor" conectado ao paciente. Esse sensor avisaria, através de um alerta, uma mensagem no seu celular ou uma mensagem em números cadastrados, quando o nível de diabetes exigisse nova aplicação de insulina. Poderia ainda armazenar esses alertas e utilizar esse histórico para administrar melhor o tratamento.

Na área de comunicação, dispositivos móveis poderiam ser usados, por exemplo, para que médicos ajustassem tratamentos a qualquer hora, estudantes transferissem exercícios da rede para estudar no ônibus ou no *shopping*, montassem apresentações enquanto a caminho da sala de apresentações, vendedores tivessem acesso a estoques em tempo real, usuários de transporte urbano fossem informados de qual linha de ônibus presta serviços a um determinado bairro e quais os horários dos coletivos.

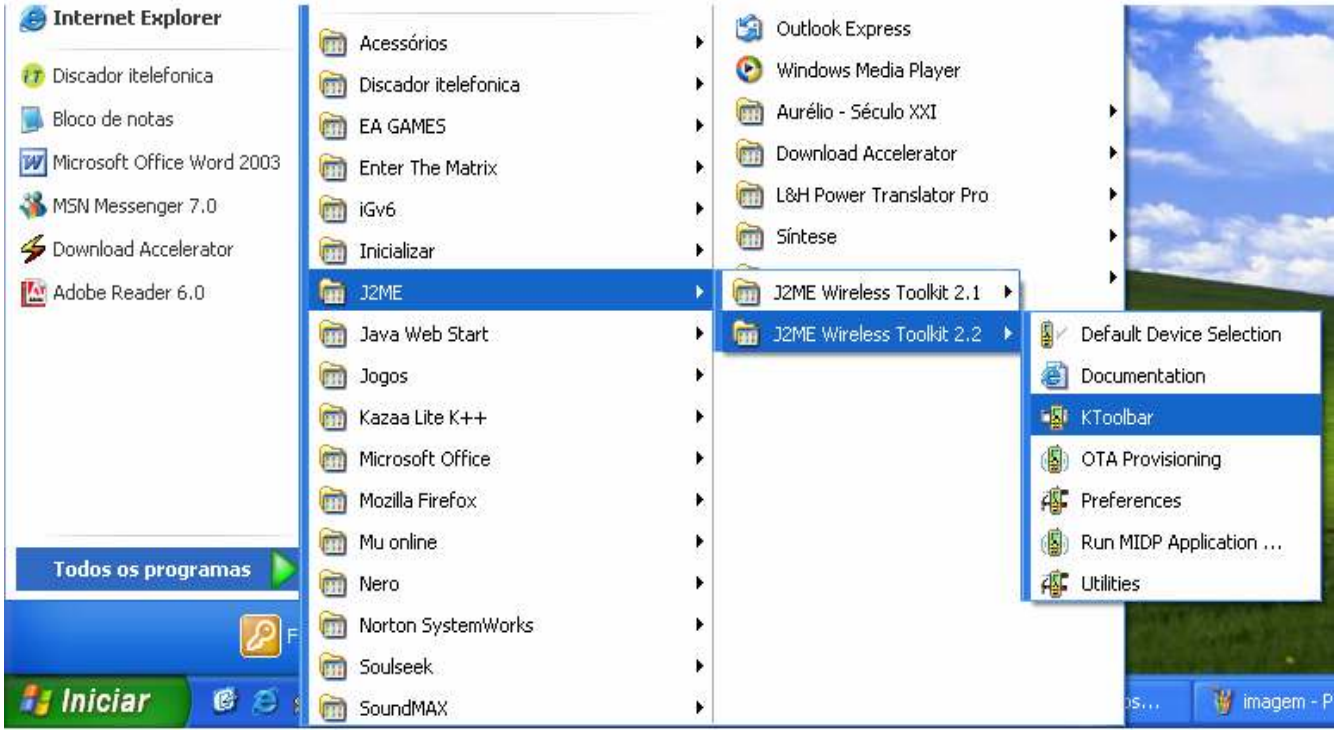
Ainda, destacam-se aplicações como acesso a banco de dados remotos, entretenimento, desenvolvimento de interfaces para jogos, que podem ser baixados ou atualizados da Internet diretamente para os dispositivos de informação móveis, entre outros.

ANEXO I – INSTALAÇÃO DO WIRELESS TOOLKIT

No site da SUN Microsystems podem ser encontradas ferramentas que permitem gerar os arquivos necessários para rodar aplicações em aparelhos compatíveis com o J2ME. Entre as ferramentas há um gerenciador de projetos bem simples e um emulador genérico de terminais com MIDP, que é o J2ME Wireless ToolKit.Porém, para facilitar as coisas, a SUN disponibilizou um IDE livre chamado Forte for Java ou SUN One Studio, nome que atualmente é usado. Existem outras opções tais como o NetBeans IDE ou a versão comercial do JBuilder da Borland ou editar o código em bloco de notas e salva-lo com extensão java. Para instalar um ambiente de desenvolvimento para J2ME, o primeiro passo é baixar e instalar o J2SE SDK e JRE da SUN. Sem os quais não é possível se fazer nada. É necessário ter o sistema operacional Windows NT, 2000 e XP. Existem versões para Linux e Solaris, caso se prefira ambiente UNIX. Para obter o Java2 SDK, Standard Edition (J2SE SDK) e/ou o Java 2, Standard Edition Runtime Environment (J2SE JRE) na versão mínima - 1.4.2 que ests disponível em: http://java.sun.com/j2se/1.4.2/download.html Em seguida baixe o J2ME Wireless Toolkit versão 1.0.4 _01 pelo link abaixo: http://java.sun.com/products/j2mewtoolkit/download.htm

Wireless Toolkit

Uma vez instalado conforme as instruções descritas no site da SUN, podemos iniciar o desenvolvimento de aplicações baseadas em J2ME. Vamos demonstrar o uso do Wireless Toolkit através do programa "Hello World!". O primeiro passo é iniciar o KToolbar que é onde criaremos um projeto. A figura a seguir ilustra onde pode ser localizada a aplicação em ambientes windows.



Uma vez iniciado o aplicativo KToolbar, o mesmo se apresentará como a seguir.

Após iniciar o KToobar devemos abrir um novo projeto. Na tela da aplicação KToolbar existe o botão "New Project …". Aperte esse botão para que seja aberto um formulário que solicitará que seja dado um nome de projeto (qualquer) e o nome da classe MIDlet que rodará o programa J2ME. No nosso exemplo, o nome do projeto é "Hello World!" e conforme descrito no código fonte do programa, a classe é chamada "HelloJ2ME".

Ao apertar o botão "Create Project" o sistema automaticamente criará uma estrutura de diretórios referentes ao projeto dentro do diretório de instalação do J2ME Wireless Toolkit.

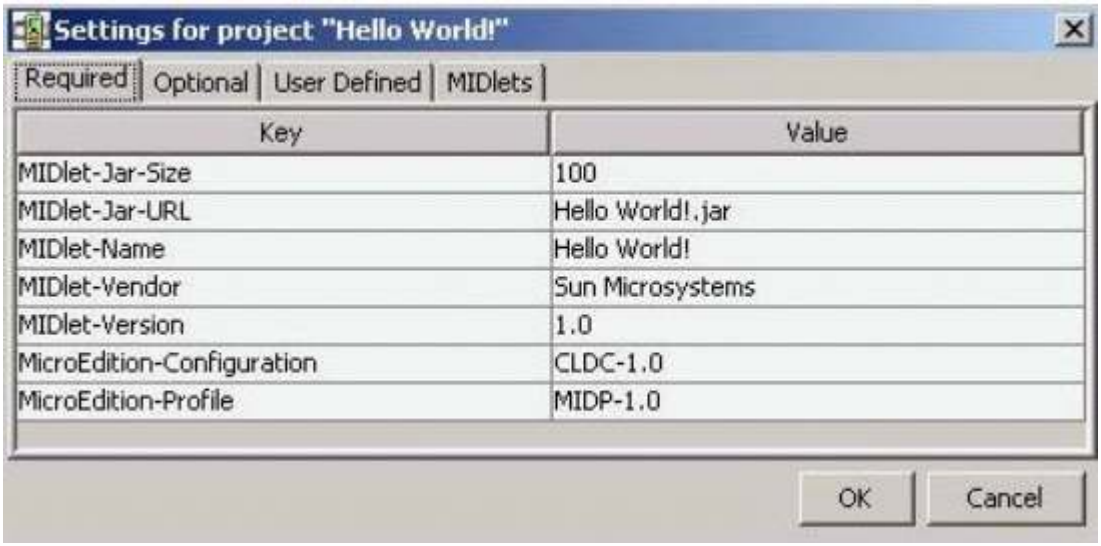
Essa estrutura estará abaixo do diretório "Apps\<Nome do Projeto>" que fica dentro da estrutura de diretórios do Toolkit (por exemplo "C:\J2MEwtk\Apps\Hello World!\"). Nela temos os seguintes diretórios.

· \Src - nele colocaremos o código fonte de nossa aplicação, ou seja, o arquivo "HelloJ2ME.java".

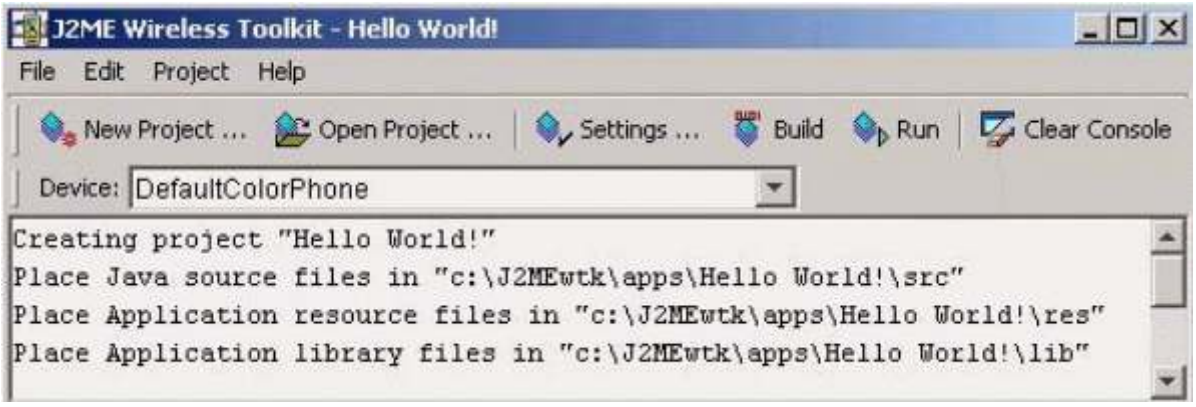
· \Bin - nele o Toolkit criará os arquivos .JAR e .JAD que são necessários para poder executar a aplicação em um celularou mesmo no emulador.

· \Res - nele são colocados todos os arquivos adicionais que são necessários para que a aplicação seja executada. Um exemplo típico é uma imagem no formato PNG com o ícone da aplicação. Todos os arquivos que forem colocados nesse diretório serão empacotados dentro do .JAR e serão acessíveis pela aplicação J2ME. No nosso exemplo não colocaremos nada lá.

Ao pressionar o botão "Create Project", o KToolbar criará os diretórios acima descritos e também o próprio diretório do projeto "Hello World!". Porém antes o Toolkit pedirá para que sejam confirmadas as informações do descritivo do Projeto/aplicação. Essas informações ou atributos serão o conteúdo do arquivo .JAD (Java Descriptor) a ser criado. No nosso caso não nos preocuparemos com esses dados e simplesmente apertaremos o botão "OK". O resultado aparece a seguir.



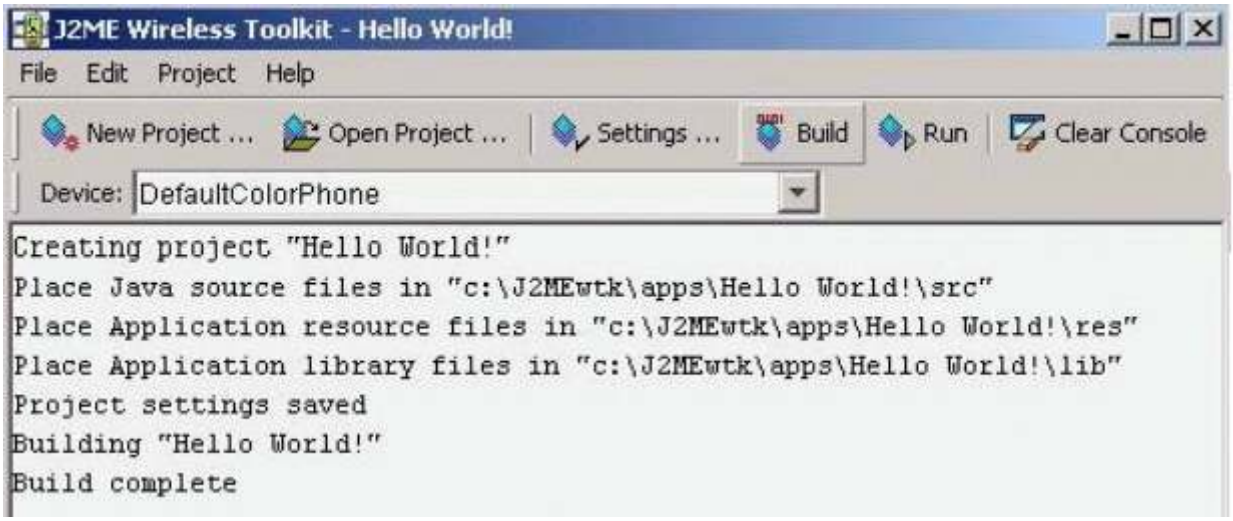
Após apertar o "OK", o Toolkit indicará no seu console que os diretórios do projeto foram criados conforme a figura abaixo.



Nesse momento o Toolkit nos sugere que os arquivos fonte sejam colocados no diretório "…\apps\Hello World!\src\". Nesse momento devemos copiar o arquivo "HelloJ2ME.java" para esse diretório. A partir de agora estamos prontos para construir a aplicação!

Basta apenas apertar o botão "Build" no KToolbar que o Toolkit compilará os arquivos fonte e criará os arquivos .class que serão usados para rodar o simulador. Abaixo segue a ilustração da construção do aplicativo "Hello World!".

Basta apenas apertar o botão "Build" no KToolbar que o Toolkit compilará os arquivos fonte e criará os arquivos .class que serão usados para rodar o simulador. Abaixo segue a ilustração da construção do aplicativo "Hello World!".



Caso não ocorra nenhum problema de compilação teremos a mensagem "Build Complete", ao final das mensagens, indicando que está tudo pronto para executar a aplicação "Hello World!" no simulador. Caso ocorram erros de compilação os mesmos serão apresentados no console do Toolkit.

No nosso caso, o programa está depurado e o código fonte está correto. Portanto não há nenhum problema de compilação. Neste caso estamos prontos para executar a aplicação.

Para isso basta apertar o botão "Run" no KToolbar. Ao fazer isso será aberta uma nova janela com o simulador escolhido. No nosso exemplo é o simulador genérico colorido, conforme a figura abaixo.

A figura acima mostra o estágio inicial do simulador que permite que seja escolhido qual Midlet executar. Nosso caso só há um Midlet especificado no arquivo .JAD, portanto temos apenas que pressionar o SoftKey para lançar a aplicação "Hello World!" (botão em destaque logo abaixo da palavra "Launch"). O resultado da execução do "Hello World!" segue abaixo.

Uma vez que a aplicação tenha executado a contento no simulador, é possível criar o arquivo .JAR (Java Archive) que é, na verdade, um arquivo compactado no formato PkZip com todas as classes e arquivos do diretório "…\res" (resources).

Para criar-se o .JAR basta selecionar "Project à Package" no menu do KToolbar. Esse arquivo somente é necessário para executar o programa no próprio aparelho celular.