DESENVOLVIMENTO DE SOFTWARE PARA DISPOSITIVOS MÓVEIS

DEVELOPMENT OF SOFTWARE FOR MOBILE DEVICES

ROBERTO JUNDI FURUTANI [1]

JOSÉ LUIZ VIEIRA DE OLIVEIRA [2]

8º CICLO DE PALESTRAS DE INFORMÁTICA DA FAI

Resumo

A programação para dispositivos móveis com a tecnologia Java com foco no desenvolvimento de jogos para celulares, possibilitou grandes avanços nesta área, oferecendo muitas funcionalidades como a detecção de colisões, criação de animações, manipulação de imagens, criação de cenários e formulários para entrada de dados e é um passo fundamental para a padronização da programação para dispositivos móveis.

Palavras-chaves

Java – programação – celulares

Abstract

The programming to mobile devices with Java technology with focus on development games for cellular, made possible great advances in this area, it offers many functionalities as the detection of collisions, creation of animations, manipulation of images, creation of scenes and forms for data entry and is a fundamental step for the standardization of the programming for mobile devices.

Key words

java - programming – cell phones

Introdução

A linguagem Java foi lançada no segundo semestre de 1995, nesta época surgia a Internet, então os desenvolvedores da Sun decidiram utilizar uma linguagem para criar programas que executassem como parte das páginas Web no navegador. Esses programas se chamam applets, e podem ser executados em qualquer navegador habilitado com a Máquina Virtual Java (Java Virtual Machine, JVM). Com os applets foi possível dar mais interatividade as páginas da Internet e assim foi a primeira

[1] Aluno do 3º ano do Curso de Ciência da Computação das Faculdades Adamantinenses Integradas robertofurutani@yahoo.com.br

[2] Professor das Faculdades Adamantinenses Integradas joseluiz@fai.com.br.

programação interativa disponível para a Internet, com isso a linguagem Java foi difundida e atraiu muitos desenvolvedores em pouco tempo. A tecnologia Java (Figura 1) é dividida em quatro plataformas:

1. Plataforma Java ™ 2 Enterprise Edition, voltado para criação de programas do lado do servidor.
2. Plataforma Java 2 Standard Edition, usado para criar programas do lado do cliente.
3. Plataforma Java 2 Micro Edition, usado para criar softwares para dispositivos com baixa capacidade de processamento e pouca memória disponível, como os celulares e os assistentes pessoais digitais (PDAs).
4. Java Card, é um dispositivo chamado "smart card" (cartão inteligente), são cartões que podem ser programados e tem um pequeno processador acoplado.

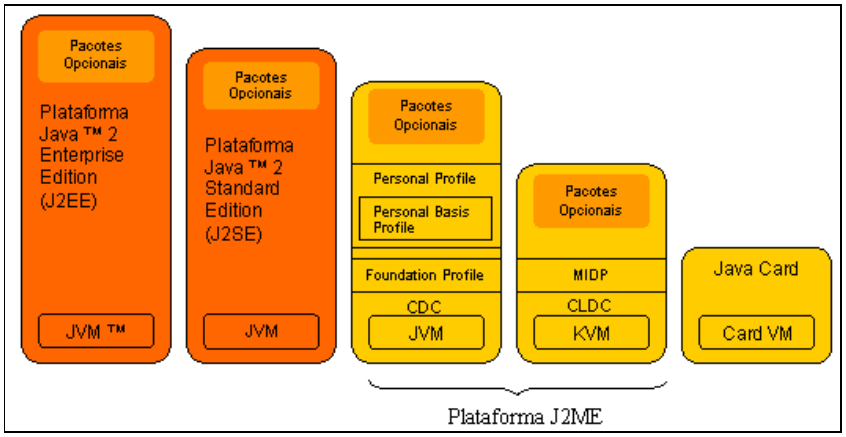

Figura 1. A tecnologia Java

A Plataforma Java™ 2, Micro Edition(J2ME)

A Java ™ 2 , Micro Edition (J2ME) é a plataforma Java para dispositivos pequenos, com baixa capacidade de memória e processamento, como celulares, Tv set-top boxes. A arquitetura J2ME é constituído de configurações, perfis e pacotes opcionais como elementos de construção do ambiente de execução Java.

Configurações

As configurações são compostas de uma máquina virtual e o mínimo de bibliotecas de classes para o funcionamento. Elas provêem a base de funcionalidades para um tipo dispositivos que se assimilam em características como processador, conexão em rede e memória disponível. Atualmente há duas configurações J2ME: o Connected Limited Device Configuration (CLDC) e o Connected Device Configuration (CDC).

O CLDC é o menor das duas, pois é para ser usado com dispositivos com conexão esporádica a rede, processadores lentos e memória limitada, por exemplo os celulares. Estes dispositivos normalmente

tem processadores de 16 ou 32 bits, e memória disponível de 128KB até 512 KB para Java. Já o CDC requer processadores mais rápidos, mais memória e maior conexão com rede. Os processadores devem ter 32 bits e memória de no mínimo disponível 2MB para Java e suas aplicações.

Perfis

Fornece um ambiente de execução para uma determinada categoria de dispositivo.

MIDP (Mobile Information Device Profile): Este perfil é designado para celulares e PDAs, oferece muitos recursos para aplicações móveis, incluindo interface com usuário, conexão a rede, armazenamento de dados local.

1. *Foundation Profile:* É um perfil de baixo nível para CDC, oferece capacidade de implementar conexão com rede sem a necessidade de interface com usuário.
2. *Personal Profile:* É um perfil CDC para dispositivos que necessitam de interface gráfica com usuário avançado ou suporte a applets.
3. *Personal Basis Profile:* É um subconjunto de Personal Profile, e suporta uma interface gráfica com usuário básico é usado em TV set-top boxes, quiosques de informações, etc.
4. *Pacotes Opcionais*: A plataforma J2ME pode ser estendida com vários pacotes opcionais com CLDC, CDC, e seus perfis correspondentes. Esse pacotes podem permitir o acesso a redes Bluetooth, web services, banco de dados, etc.

Assim, com o J2ME é possível criar aplicativos que rodem em diferentes aparelhos celulares e de diferentes marcas, basta que o aparelho esteja habilitado com a máquina virtual Java, isto torna muito mais fácil o desenvolvimento e a distribuição de programas.

O uso da MIDP 2.0, a mais nova versão do perfil J2ME é direcionado principalmente para celulares (é possível usar também para PDAs (Personal Digital Assistant) como os Palms), um passo fundamental para uma maior padronização da programação Java para esses dispositivos. O MIDP 2.0 trouxe várias funcionalidades úteis para o desenvolvimento de jogos, como por exemplo, o suporte à transparência de pixels. A mais importante foi a inclusão da Game API, base para o desenvolvimento de jogos.

Estrutura Básica de um MIDlet

Um programa feito para a plataforma J2ME recebe o nome de MIDlet, assim como os programas para a plataforma J2EE recebem o nome de Servlets. A estrutura básica de um MIDlet é:

```
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;
public class HelloWorld extends MIDlet implements CommandListener {
       public HelloWorld () { } // Construtor da classe.
       // Método chamado quando é iniciado uma aplicação.
       public void startApp() { }
       // Método chamado quando a aplicação é interrompida, por exemplo, quando
chega uma ligação, ou uma mensagem sms.
```

```
        public void pauseApp() { }
        // Método chamado quando uma aplicação será fechada.
        public void destroyApp(boolean condicional) { }
        //Método onde será implementado quase toda aplicação, e recebe os comandos do
usuário.
        public void commandAction(Command c, Displayable d){} }
```

Figura 2. Estrutura mínima de um programa para Celular

Game API

O conjunto de cinco classes do pacote javax.microedition.lcdui.game formam o Game API, oferecendo poder e simplicidade a criação de jogos para celulares. Para exemplificar os conceitos mostrados aqui, será utilizado um jogo bem simples. É o FaiGame (Figura 3), um jogo bem simples.

Figura 3. Jogo rodando no emulador Nokia.

A Game API é composta pelas seguintes classes:

1. GameCanvas
2. Layer
3. LayerManager
4. Sprite
5. TiledLayer

A classe GameCanvas é responsável pelo ciclo básico do jogo (veja Figura 4), ou seja, verificar os estados das teclas (se elas foram ou não pressionadas), gerenciar o buffer gráfico e enviar as imagens para a tela.

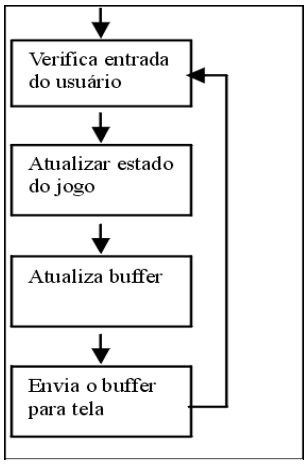

Figura 4. Ciclo Básico dos jogos.

Primeiro é verificado a entrada do usuário, depois é atualizado o estado do jogo de acordo com essa entrada e a lógica implementada no jogo, por exemplo, se o usuário pressionou a tecla para a direita e a lógica do jogo manda que o personagem mova-se para a direita. Em seguida o buffer gráfico é atualizado e enviado para a tela, depois o ciclo recomeça. No código do construtor da subclasse GameCanvas deve-se chamar o construtor da superclasse e informar se os eventos normais de teclas de jogo devem ser ignorados e depois inicializar os elementos gráficos e adicioná-los ao LayerManager, veja Figura 5.

```
public FaiGameCanvas() throws IOException {
        super(true); // Ignora os eventos das teclas de jogo
        fai = new Fai();
        aluno = new Aluno();
        //cria um gerenciador de layers
        layerManager = new LayerManager();
         //Insere os elementos no layer
        layerManager.append(aluno);
        layerManager.append(fai);
         //Cria um mapa das ruas
         ruas = criarRuas();
        layerManager.append(ruas);
    }
```

Figura 5. Adicionando elementos gráficos ao gerenciador de layout.

Para verificar a tecla que está pressionada é utilizado o método getKeyStates() e comparar com uma das constantes que representa uma tecla utilizando o operador "&". Exemplos, o primeiro (Figura 6)

verifica se a tecla para a direita foi pressionada e o segundo (Figura 7) se a tecla para cima foi pressionada.

```
int keyStates = getKeyStates();
    if ((keyStates & RIGHT_PRESSED) != 0){
}
```

Figura 6. Move o personagem para a direita

```
int keyStates = getKeyStates();
    if ((keyStates & UP_PRESSED) != 0){
}
```

Figura 7. Move o personagem para cima.

Toda a implementação do ciclo do jogo é feita dentro do método run(), implementado a interface Runnable, que vai fazer com que o jogo seja executado em uma thread independente. E no método startApp() da classe FaiGameMIDlet é iniciada a thread responsável pelo ciclo do jogo dessa forma:

```
canvas = new FaiGameCanvas();
canvas.start();
```

Figura 8. Iniciando a thread do jogo.

A classe Layer tem duas subclasses concretas, a classe Sprite e a TiledLayer. Em um jogo há muitos elementos gráficos, porém existe um tipo especial de gráfico chamado Sprite. Um Sprite pode ser um personagem ou um monstro e geralmente eles são animados e sofrem algum tipo de ação. No jogo exemplo o aluno e a fai são dois sprites. Com a classe Sprite é possível criar uma animação a partir de vários quadros e oferece métodos para a verificação de colisão, rotação e movimentação. A criação de um sprite é simples, basta criar uma classe que estenda a classe Sprite mostrada na Figura 9.

```
public class Aluno extends Sprite
```

Figura 9. Criando um sprite.

Depois na chamada do construtor da classe superior será passada por parâmetro a imagem (uma figura PNG) e a largura e a altura de cada quadro.

```
      super(Image.createImage("/boneco.png"), 16, 16);
```

Figura 10. Definindo a imagem e tamanho do Sprite.

Neste caso cada quadro vai ter 16x16 pixels, ou seja, a classe Sprite divide automaticamente a imagem boneco.png de tamanho32x32 pixels em quadros de 16x16 pixels para compor a animação.

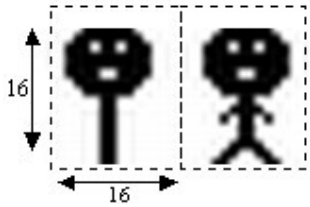

Figura 11. Imagem dividida em quadros de 16x16 pixels

São formados quadros para dar a ilusão de que o aluno está andando. Para alternar entre os quadros use o método nextFrame(), quando se tiver um número grande quadros é possível determinar a seqüência que vai ser seguida. Agora será necessária definir o centro da imagem para que se possa fazer rotação em torno de si mesma, para isso existe o método defineReferencePixel(8,8), que define o pixel central. Se não for setado um valor, o valor assumido é 0,0 por padrão. Por exemplo, para rotacionar (veja figura 12 e 13) o aluno para cima, deve-se girar ele 270 graus:

```
setTransform(TRANS_ROT270);
```

*Figura 12.*Método responsável pela rotação da imagem.

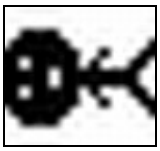

Figura 13. Imagem rotacionada em 270 graus.

Constante	Valor
TRANS_NONE	0
TRANS_MIRROR_ROT180	1
TRANS_MIRROR	2
TRANS_ROT180	3
TRANS_MIRROR_ROT270	4
TRANS_ROT90	5
TRANS_ROT270	6
TRANS_MIRROR_ROT90	7

Tabela 1. Tipos de rotações possíveis.

Para mover o aluno na tela usa-se o método move(int pixel), recebe como parâmetro a quantidade pixels que deve movimentar. Se for para direita e para cima o número deve ser positivo e se for para baixo e para esquerda negativo. A detecção de colisão é uma das mais essenciais funções em um jogo, com a Game API é simples fazer essa detecção. O método collidesWith() é que faz esse serviço, ele verifica colisões com imagens dentro do TiledLayer, com um objeto Image ou com outro Sprite. Na classe FaiGameCanvas é utilizado as seguintes versões:

- aluno.collidesWith(ruas, true))

- aluno.collidesWith(fai, true))

A primeira detecta a colisão do aluno com um TiledLayer e a segunda com um Sprite. O true serve para indicar que deve ser considerada a transparência do pixel, ou seja, partes transparentes não serão levadas em conta para detectar a colisão. O programador pode determinar através do método defineCollisionRectangle() uma área especifica de colisão. Usando o TiledLayer é possível criar imagens maiores mapeando pedaços de outra imagem. A construção dos quarteirões do jogo foi feito através desse recurso, como mostrada na Figura 14.

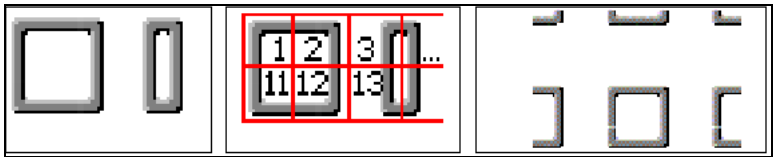

Figura 14. Demostração das ações do TiledLayer para criar um cenário.

No lado esquerdo é descrito a imagem original de 160x32 pixels, para dividir é usado a classe TiledLayer com a sintaxe TiledLayer(10, 10, imagem, 16, 16), no centro a imagem já foi dividida em quadros de 16x16 pixels e foi atribuído um índices para cada quadro. A partir desse índices criamos um mapa. Na imagem do lado direito é mostrada um pedaço do cenário criado com o mapa mostrado na Figura 15:

```
final int[] mapa = {
        16,  0, 15, 16,  0, 15, 16,  0, 15, 16,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        2,  0,  1,  2,  0,  1,  2,  0,  1,  2,
        12,  0, 11, 12,  0, 11, 12,  0, 11, 12,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        2,  0,  1,  2,  0,  0,  0,  0,  1,  2,
        12,  0, 11, 12,  0,  0,  0,  0, 11, 12,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        2,  0,  1,  2,  0,  1,  2,  0,  1,  2,
        12,  0, 11, 12,  0, 11, 12,  0, 11, 12};
```

Figura 15. Vetor com os índices dos quadros da imagem para montar o cenário.

O número zero indica espaço em branco. Depois é um laço for (Figura 16) para montar o cenário de acordo com o mapa e então é usado o método setCell() da classe TiledLayer.

```
for (int i = 0; i < mapa.length; i++) {
        int coluna = i % 10;
        int linha = (i - coluna) / 10;
        ruas.setCell(coluna, linha, mapa[i]); }
```

Figura 16. Adicionando os quadros nas células respectivas.

A classe LayerManager facilita a renderização de objetos Sprite e TiledLayer. Com ela, em vez do programador chamar o método paint() de cada objeto individualmente, ele adiciona ao LayerManager e chamar o método paint() desta classe, que fica responsável por desenhar as imagens na ordem determinada. No jogo de exemplo, primeiro é inserido o sprite aluno, depois o sprite fai e depois o TiledLayer, mostrado na Figura 17:

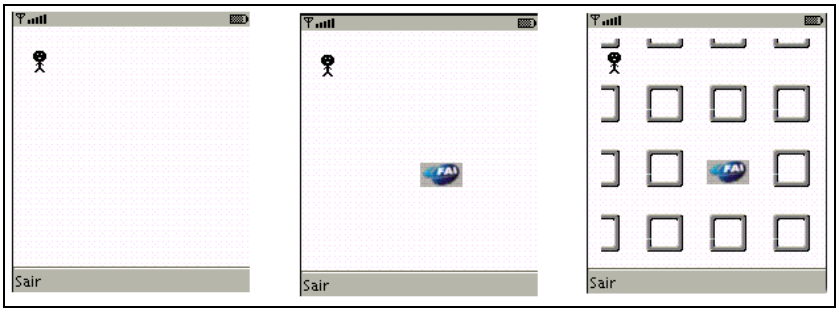

Figura 17. Diversas camadas são inseridas ao LayerManager, criando um cenário completo.

Além de jogos com J2ME é possível desenvolver programas normais, como um programa de cadastro, um sistema de estoque, etc. Pois o MIDP nos oferece componentes para que o usuário entre com seus dados e processe as informações. Um exemplo simples é o da Figura 18 abaixo:

Figura 18. Exemplo de formulário

É mostrado na tela dois TextField, que são usados para que o usuário entre com uma informação do tipo String, e um command, que é usado para interação do usuário com o aplicativo, neste caso é o comando exit para que o usuário feche o programa.

Commands: É usado para que o usuário envie comandos ao programa. Para criar um command deve-se informa três parâmetros.

1. Label: texto que será mostrado na tela;
2. Type: tipo do comando;
3. Priority: prioridade do comando.

Um exemplo é mostrado na Figura 19:

```
Sair = new Command("Sair",Command.Exit,0);
```

Figura 19. Criando um comando para fecha a aplicação.

Forms: Utilizado para montar formulários, pode ter um número qualquer de controles de interface chamados Items. Um form pode ser criado de 2 maneiras veja a Figura 20:

```
Form(String titulo);
Form(String titulo, Item[] item);
```

Figura 20. Maneiras de criar um formulário.

Os Items podem ser adicionados e removidos com os métodos: append(Item item), set(int Indice, Item item), insert(int Indice, Item item) e delete(int Indice).

1. StringItem – um texto simples.
2. TextField – uma caixa de texto. O TextField é utilizado para inserção de dados como um texto qualquer, só números, email, etc.
3. DateField – Usado para data, hora ou ambos.
4. Gauge – Representação gráfica de um número inteiro (Figura 21). Existem dois tipos de gauges, um interativo que o usuário possa modificar e outro controlado por programa.

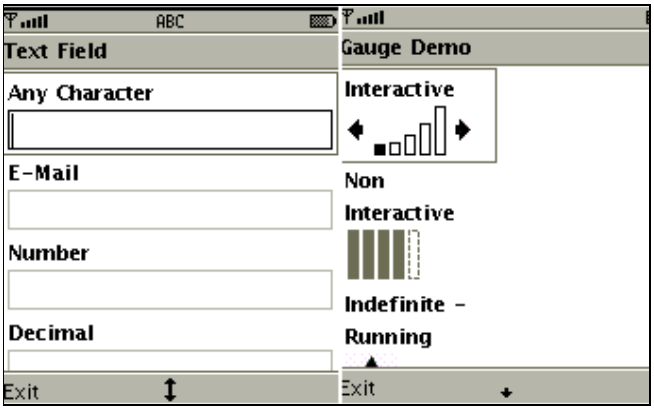

Figura 21. Demonstração de TextField e Gauge.

A ferramenta utilizada para executar o jogo no computador é um emulador chamado Wireless ToolKit (WTK) está disponível gratuitamente para download no site *http://java.sun.com/products/j2mewtoolkit* para vários sistemas operacionais. Clicando no botão new project, digitar FaiGame em project name e faigame.FaiGameMIDlet em MIDLet class Name (Figura 22). Depois clicar no botão Create Project, será mostrada uma janela, não alterar os valores sugeridos e clicar em Ok. Isso irá criar uma estrutura de diretórios em <Diretório de instalação do WTK>/apps/FaiGame. Descompactar o arquivo zip dentro deste diretório, mantendo a estrutura dos diretórios. Para compilar os fontes clicar em Build, para rodar em um dos emuladores existentes basta clicar em Run.

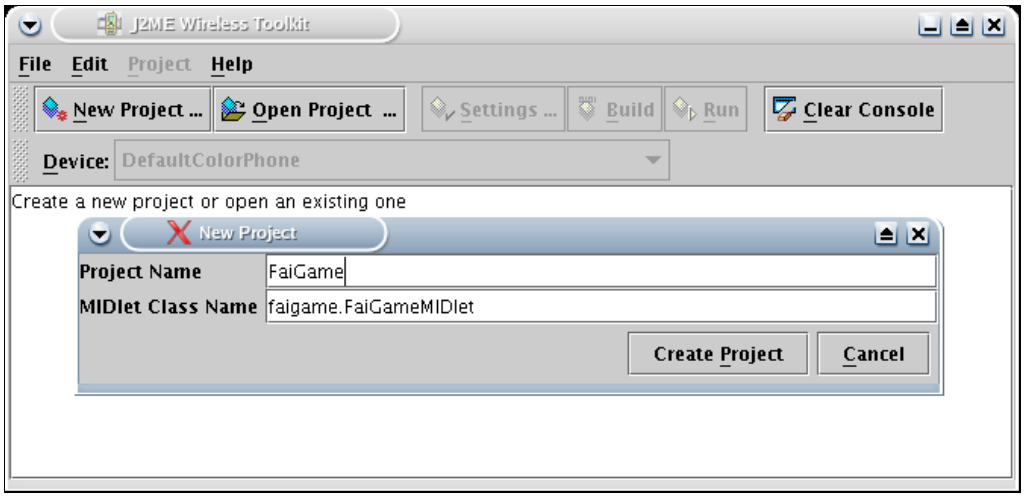

Figura 22. Criando um projeto para rodar o jogo exemplo.

Conclusões

A plataforma J2ME proporciona produtividade, versatilidade e robustez para desenvolvimento de jogos e aplicativos em geral para pequenos dispositivos e com o mercado de desenvolvimento de aplicações para celulares crescendo dia a dia, se torna um mercado muito grande para empresas que queiram desenvolver softwares para essa área, sendo o retorno financeiro altamente compensador.