**UNIVERSIDADE REGIONAL DE BLUMENAU**

**CENTRO DE CIÊNCIAS EXATAS E NATURAIS**

**CURSO DE CIÊNCIAS DA COMPUTAÇÃO**

(Bacharelado)

# PROTÓTIPO DE SOFTWARE PARA DISPOSITIVOS MÓVEIS UTILIZANDO JAVA ME PARA CÁLCULO DE REGULARIDADE EM RALLY

TRABALHO DE CONCLUSÃO DE CURSO SUBMETIDO À UNIVERSIDADE REGIONAL DE BLUMENAU PARA A OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA COM NOME EQUIVALENTE NO CURSO DE CIÊNCIAS DA COMPUTAÇÃO — BACHARELADO

**FÁBIO MARCELO DEPINÉ**

BLUMENAU, DEZEMBRO/2002.

2002/2-25

# PROTÓTIPO DE SOFTWARE PARA DISPOSITIVOS MÓVEIS UTILIZANDO JAVA ME PARA CÁLCULO DE REGULARIDADE EM RALLY

**FÁBIO MARCELO DEPINÉ**

ESTE TRABALHO DE CONCLUSÃO DE CURSO, FOI JULGADO ADEQUADO PARA OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA DE TRABALHO DE CONCLUSÃO DE CURSO OBRIGATÓRIA PARA OBTENÇÃO DO TÍTULO DE:

**BACHAREL EM CIÊNCIAS DA COMPUTAÇÃO**

_____

Prof. Francisco Adell Péricas — Orientador na FURB

_____

Prof. José Roque Voltolini da Silva — Coordenador do TCC

**BANCA EXAMINADORA**

_____

Prof. Francisco Adell Péricas

_____

Prof. Sérgio Stringari

_____

Prof. Jomi Fred Hubner

Dedico este trabalho aos meus pais, Leandro e Vera, ao meu irmão Tiago,
e à minha namorada, Ana Lígia, pelo incentivo recebido durante
o desenvolvimento deste trabalho.

# AGRADECIMENTOS

Agradeço aos professores do curso de Bacharelado em Ciências da Computação da Universidade Regional de Blumenau, por todo conhecimento repassado no curso. Em especial ao professor Francisco Adell Péricas, que me orientou no desenvolvimento deste trabalho.

Aos meus familiares, que estiveram do meu lado o tempo todo e sempre me apoiaram e incentivaram durante toda a minha formação acadêmica.

Aos meus amigos, por todo tempo e ajuda oferecida.

À Ana Lígia, pela paciência e compreensão.

E finalmente agradeço à pessoa mais importante de todas, pelas oportunidades e desafios, e por estar sempre ao meu lado guiando meus passos. Obrigado Deus.

# SUMÁRIO

# LISTA DE FIGURAS

# LISTA DE QUADROS



# LISTA DE TABELAS

# RESUMO

Este trabalho apresenta um estudo sobre a tecnologia J2ME, objetivando a especificação e implementação de um protótipo de *software* para cálculo de uma planilha de rally de reguralidade em um dispositivo portátil. Para o funcionamento do protótipo, foi desenvolvido também uma classe em Java para emular cálculos com ponto-flutuante.

# ABSTRACT

This paper presents a study about J2ME tecnology, resulting in a specification and implementation of a prototype system to provide a rally spread sheet calculation in a portable device. In order to make such prototype work, a Java class to emulate floating-point calculations has also been developed.

# 1 INTRODUÇÃO

Num tempo não muito além dos dias atuais, possuir um aparelho de telefone celular era um fator de status para as pessoas. Somente uma fatia muito pequena da população, a qual fosse caracterizada por seu poder aquisitivo elevado, teria condições de exibir esses “aparelhinhos” móveis. Aliado a essa situação, surgia o advento das redes de computadores livres dos fios que as interconectavam. Era a chegada de uma inovação na maneira de se comunicar, deixando mais livre a localização desses terminais de acesso à rede.

Os dispositivos sem fio oferecem uma conectividade que outros dispositivos não possuem. Em poucos anos o desenvolvimento de aplicações para esses equipamentos tende a aumentar drasticamente, utilizando-se dos recursos que os mesmos têm a oferecer.

Entre esses recursos há o ambiente Java. Equipamentos que rodam Java estão cada vez mais presentes no dia a dia das pessoas, sejam produtos fixos ou móveis, de uso pessoal ou de consumo, e que apresentam freqüentemente, entre outras características, alguma forma de comunicação e um elevado grau de sofisticação em relação às tarefas que executam (dispositivos inteligentes). Geralmente são microprocessadores embutidos em algum dispositivo, como em telefones celulares e eletrodomésticos.

Segundo Hopson (1998), Java é uma linguagem de programação independente de plataforma desenvolvida pela Sun Microsystems. Originalmente feita para integrar circuitos de eletrodomésticos, ganhou a Internet, sendo utilizada largamente na Web com objetivo de dinamizar *sites* e integrar servidores.

A utilização de Java deve facilitar a integração de dispositivos móveis aos serviços da Internet - a execução de blocos de softwares baixados pelo aparelho durante a navegação e a utilização de aplicações gráficas são os argumentos para convencer o consumidor. Junte-se a isso a possibilidade de executar jogos multi-usuário, transmissão e visualização de figuras, bem como de fotografias digitais e acesso a *sites* interativos, como no caso de mapas e gráficos de cotação de ações.

A expectativa é grande, e com razão. O usuário poderá personalizar seu aparelho de acordo com as suas necessidades, fazendo a cópia de aplicativos. A vantagem evidente é a possibilidade de atualização permanente do software, não sendo mais necessário que o usuário

compre um novo aparelho a cada vez que uma nova aplicação ou serviço seja lançado. Sendo mais fácil de usar que o Wireless Application Protocol (WAP), utilizado nos celulares para navegar na Internet, uma vez que o Java permite interface gráfica personalizável, os fabricantes esperam um salto nas vendas.

O desenvolvimento de aplicações Java para celulares é possível utilizando-se o ambiente Java 2 Platform, Micro Edition (J2ME), basicamente uma versão compacta da linguagem padronizada da Sun, destinado ao desenvolvimento de aplicações para dispositivos móveis. A plataforma J2ME oferece para tanto, uma máquina virtual Java, chamada de KVM, pequena o bastante para ser suportada dentro das restrições de memória destes dispositivos. Essa máquina, por sua vez, não suporta todas as classes e funcionalidades do Java. O J2ME é formado basicamente por duas outras especificações.

Uma é a *Connected Limited Device Configuration* (CLDC), que fornece um conjunto de Java API para aplicações sem fio, ou seja, que sejam suportadas pelo dispositivo móvel. Essa especificação fornece as classes responsáveis pela conexão, entrada e saída de dados, classes de manipulações de *strings* e de operações matemáticas.

A outra é a *Mobile Information Device Profile* (MIDP), que oferece uma biblioteca de interface gráfica para o dispositivo móvel. Essa especificação provê ainda as classes para memória persistente e algumas classes que definem objetos de formulário.

O presente trabalho visa o desenvolvimento de um aplicativo que permita que se entre com as informações referentes a uma planilha de um rally de regularidade e obtenha o processamento dos tempos e deslocamentos do percurso, utilizando um celular que roda aplicações em Java. O protótipo a ser implementado neste trabalho será desenvolvido utilizando-se a linguagem Java para equipamentos portáteis (J2ME).

## 1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é desenvolver um protótipo de software para cálculo de planilha de rally de regularidade, utilizando a plataforma J2ME da Sun, para dispositivos móveis que suportem esta tecnologia.

Os objetivos específicos do trabalho são:

a) demonstrar a tecnologia J2ME e verificar suas limitações quando aplicada à equipamentos portáteis;
b) especificar e implementar um protótipo de software para cálculo de planilha de rally para dispositivos móveis;
c) entrar com as informações sobre a planilha do rally no celular, retornando o resultado ao usuário.

## 1.2 ESTRUTURA DO TRABALHO

Este trabalho está divido em seis capítulos, estando assim distribuídos:

No primeiro capítulo, encontra-se a introdução e os objetivos a serem alcançados com o desenvolvimento do trabalho.

No segundo capítulo é apresentado um estudo sobre a tecnologia Java.

No terceiro capítulo, encontra-se fundamentos, conceitos e características de um rally de regularidade.

O quarto capítulo trata do desenvolvimento do protótipo, demonstrando as ferramentas utilizadas, a especificação e a implementação do mesmo.

No quinto capítulo, é apresentado as considerações finais sobre o trabalho e sugestões para trabalhos futuros.

Após o capítulo 5, encontra-se os apêndices e as referências bibliográficas utilizadas na elaboração desde trabalho.

# 2 JAVA

Java é uma linguagem computacional completa, adequada para o desenvolvimento de aplicações baseadas na rede Internet, redes fechadas ou ainda programas executáveis.

Segundo Nunes (2002), das tecnologias no campo dos computadores, a linguagem Java foi das que teve maior efeito no mercado. Desde páginas web usando applets Java, até aplicações desenvolvidas completamente nesta linguagem, a linguagem Java tem honrado o seu *slogan* "Write Once, Run Anywhere".

Foi desenvolvida na primeira metade da década de 90 nos laboratórios da Sun Microsystems com o objetivo de ser mais simples e eficiente do que suas predecessoras. O alvo inicial era a produção de software para produtos eletrônicos de consumo (fornos de microondas, agendas eletrônicas, etc.). Um dos requisitos para esse tipo de software é ter código compacto e de arquitetura neutra.

Segundo Indrusiak (2002), a linguagem obteve sucesso em cumprir os requisitos de sua especificação, mas apesar de sua eficiência não conseguiu sucesso comercial. Com a popularização da rede Internet, os pesquisadores da Sun Microsystems perceberam que aquele seria um nicho ideal para aplicar a recém criada linguagem de programação. A partir disso, adaptaram o código Java para que pudesse ser utilizado em microcomputadores conectados a rede Internet, mais especificamente no ambiente da World Wide Web.

Com isso, a linguagem conseguiu uma grande popularização, passando a ser usada amplamente na construção de documentos Internet que permitam maior interatividade. Os principais navegadores Internet disponíveis comercialmente passaram a dar suporte aos programas Java, e outras tecnologias em áreas como computação gráfica e banco de dados também buscaram integrar-se com o novo paradigma proposto pela linguagem: aplicações voltadas para o uso em redes de computadores.

Atualmente, a linguagem Java é a força propulsora por trás de alguns dos maiores avanços da computação mundial, como:

a) acesso remoto a bancos de dados;
b) bancos de dados distribuídos;
c) comércio eletrônico;

d) interatividade em páginas WWW e em ambientes de Realidade Virtual distribuídos;
e) ensino à distância;
f) jogos e entretenimento.

## 2.1 CÓDIGO INTERPRETADO E PORTÁVEL

Segundo Indrusiak (2002), as diversas linguagens de programação podem ser tanto compiladas como interpretadas. Quando se utiliza uma linguagem compilada, é necessário executar um programa para traduzir os arquivos fonte, legíveis em linguagem de alto nível, em código executável. As linguagens compiladas têm a vantagem de produzir código de alta performance, o qual está ajustado para o funcionamento em um tipo específico de processador ou arquitetura de processador. Aplicativos compilados, chamados de código binário, só podem rodar no tipo de computador para o qual foram compilados, uma vez que esses aplicativos consistem, na realidade, em instruções em linguagem de máquina, entendidas e executadas pelo microprocessador.

As linguagens interpretadas só existem em código fonte. Quando em execução, um programa chamado interpretador toma o código fonte e executa as ações indicadas pelos comandos no arquivo. O interpretador é, na realidade, o único aplicativo que está sendo executado. Entre os benefícios das linguagens interpretadas está o fato dos programas interpretados poderem rodar em uma variedade de plataformas diferentes, pois estes só existem em código fonte. Além disso são mais fáceis de depurar.

A linguagem Java é tanto compilada como interpretada. Após escrever um programa em Java, utilizando um editor de textos qualquer, salva-se o programa como código fonte. A seguir, pode-se compilar esse fonte, a fim de produzir um tipo de arquivo binário chamado de arquivo de classe. Esses arquivos não são executados diretamente pois eles não contêm instruções que são entendidas diretamente pelos processadores atualmente disponíveis no mercado.

Os programas Java são compilados em um formato intermediário chamado *bytecodes*. Assim, esses programas podem ser interpretados e executados em qualquer sistema através de um interpretador Java, ou Java Virtual Machine (JVM). Com isso, o código precisa ser escrito

e compilado apenas uma vez, pois os *bytecodes* gerados serão executados da mesma forma em qualquer plataforma de hardware e software.

## 2.2 SEGURANÇA

Por ter seu projeto voltado para a simplicidade de código, as possibilidades de erro de programação em Java são reduzidas. Apesar disso, a linguagem traz outros recursos para tornar seu código ainda mais eficiente. O processo de compilação - geração de *bytecodes* - é projetado para a detecção prévia dos possíveis erros, evitando que os mesmos se manifestem em tempo de execução.

Segundo Hoff (1996), além de diminuir as possibilidades de erro de programação, a linguagem tem um esquema de segurança para garantir a integridade de código, principalmente no caso do código originário de rede insegura. Esses recursos de segurança são notáveis principalmente dentro do ambiente do interpretador.

Após baixar um applet (*bytecodes* para serem interpretados e executados em um navegador Internet) da rede, o interpretador faz uma verificação do código, buscando alterações intencionais ou não. A seguir, o interpretador determina o formato de memória para execução. Em outras palavras, não é possível acessar informações diretamente da memória ou inserir código estranho ao código original.

## 2.3 JAVA API

Uma aplicação desenvolvida em Java pode utilizar várias bibliotecas nativas desenvolvidas pela Sun, e o conjunto dessas bibliotecas Java é conhecido como Java Application Programming Interface (API). Nessas bibliotecas estão várias classes, organizadas em pacotes. Cada um desses pacotes traz classes com funcionalidade básica e vital para um determinado ramo de programação Java.

Juntas, a JVM e a Java API constituem o ambiente de execução de Java ou ainda, a plataforma Java.

### 2.3.1 JCP

Segundo SouJava (2002), Java Community Process (JCP) é o processo aberto de padronização da tecnologia Java. É através desse processo que as novas APIs Java são especificadas e também que as APIs já existentes são evoluídas. O JCP define como a comunidade deve solicitar uma nova API, ou modificação, que ocorre através de uma Java Specification Request (JSR) e, uma vez aceita a solicitação, como essa solicitação deve ser encaminhada e atendida, qual o produto final (especificação + implementação de referência + teste de compatibilidade), quanto tempo o processo vai durar, etc. O JCP é o conjunto de normas que rege a evolução da plataforma Java.

## 2.4 PLATAFORMA JAVA 2

De acordo com Sun Wireless (2002), Java 2 é a nova versão da plataforma Java da Sun Microsystems. Percebendo que um só produto não conseguiria abranger todas as necessidades do mercado, a Sun projetou a Java 2 em três edições: Java 2 Platform, Standard Edition (J2SE); Java 2 Platform, Enterprise Edition (J2EE); e Java 2 Platform, Micro Edition (J2ME). Cada edição define uma tecnologia e um conjunto de ferramentas com propósitos diferentes.

A J2SE foi a primeira edição criada da Java 2 Platform. Suporta o desenvolvimento de aplicações para desktop e estações de trabalho. J2SE é a versão básica do Java com a API padrão.

Pouco tempo depois do lançamento da J2SE, uma nova edição foi criada, a J2EE, suportando aplicações mais robustas e destinada a servidores de grande porte.

Segundo SUN Brasil (2002), a API da J2EE fornece uma linha completa de funcionalidades e um ambiente integrado para a criação de aplicativos Java multi-camadas no servidor.

Segundo SouJava (2002), a plataforma J2EE, que expande a J2SE, é uma plataforma completa, robusta, estável, segura e de alta performance, voltada para o desenvolvimento de soluções corporativas. O principal objetivo da Plataforma J2EE é reduzir a complexidade e o tempo (e portanto o custo) do desenvolvimento de aplicações corporativas.

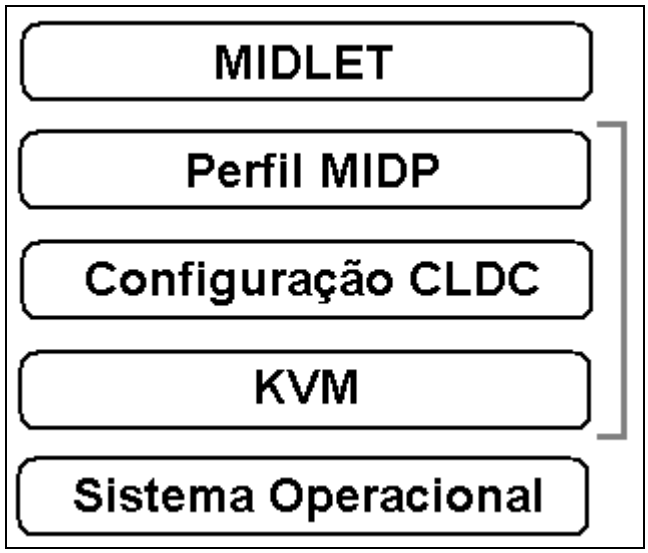
### 2.4.1 J2ME

Com o aumento expressivo de equipamentos eletrônicos móveis, como telefones celulares e PDAs, a Sun, de olho nesse mercado, criou a última integrante da plataforma Java 2, a Micro Edition.

De acordo com Fonseca (2002), essa tecnologia é uma versão compacta da linguagem padronizada pela Sun, e direcionada ao mercado de pequenos dispositivos. J2ME na verdade é um conjunto de especificações que tem por objetivo disponibilizar uma JVM, API e ferramentas para equipamentos portáteis e qualquer dispositivo com poder de processamento menor que os atuais computadores de mesa.

Segundo SUN (2002), a arquitetura J2ME é modular e escalável. Esta modularidade e escalabilidade são definidas pela tecnologia J2ME em um modelo de 3 camadas embutidas sobre o sistema operacional do dispositivo: a camada de perfil, a camada de configuração, e a camada do interpretador. A aplicação encontra-se acima da camada de perfil. Podemos observar essas camadas na FIGURA 1.

**FIGURA 1: CAMADAS DA J2ME**




#### 2.4.1.1 CAMADA DE CONFIGURAÇÃO

De acordo com MicroJava (2002), devido à grande quantidade de dispositivos existentes, variando desde placas com sistema operacional embutido até PDAs com grande poder de processamento, e funcionalidades diferentes, com conexões de redes velozes ou não,

foi necessário definir um mínimo de funcionalidades (JVM e API) para uma faixa de equipamentos que tivessem características semelhantes (memória, processamento, conectividade).

Logo, uma mesma configuração atende vários dispositivos diferentes, não possuindo detalhes específicos do dispositivo. Uma configuração serve como a base de execução e desenvolvimento das aplicações J2ME.

A API de uma configuração pode ter subconjuntos da API de J2SE. No momento existem duas Configurações: Connected Device Configuration (CDC) e Connected Limited Device Configuration (CLDC). A configuração CDC possui um conjunto de APIs que suportam equipamentos fixos de porte médio, tal como televisores. A configuração CLDC é um conjunto de APIs destinadas a aparelhos cujo poder de processamento, display e memória são limitados.

A CLDC é direcionada aos aparelhos com processamento de 16-32 Mhz ou maior, 128 a 512 Kb de memória total, seja ela RAM, Flash ou ROM, disponíveis para a plataforma Java, fornecimento de energia limitado, normalmente por baterias, conectividade com algum tipo de rede, normalmente sem fio, conexões intermitentes e com largura de banda pequena (9600 bps ou menos), interfaces de usuário com recursos bastante limitados, ou mesmo sem interface visual, ou com variados níveis de sofisticação baseados em LCD, leds, etc.

Existem algumas limitações na configuração CLDC em relação à edição J2SE. As principais são:

a) não há suporte para números de ponto flutuante, portanto os tipos *float* e *double* não podem ser usados. Isso ocorre principalmente por causa do hardware que não suporta, pois operações com ponto flutuante requerem um maior processamento;
b) sem suporte a finalização (object.finalize());
c) suporte limitado a erros. Existem apenas três classes de erros: java.lang.Error, java.lang.OutOfMemoryError, e java.lang.VirtualMachineError. Erros de Runtime são manipulados de maneira dependente de implementação, que pode terminar a aplicação ou efetuar um *reset* no dispositivo. Exceções de aplicação são utilizadas normalmente, como no J2SE;

d) não há suporte à Java Native Interface (JNI), por questões de segurança e performance.

Segundo SUN J2ME (2002), a resposta do porquê dessas limitações está na memória e processamento limitados, características dos equipamentos portáteis. Se comparadas as classes java.io e java.net do J2SE com as classes javax.microedition.io e java.io do J2ME, pode-se observar que a economia de espaço é grande: 28 Kbytes para J2ME contra 316 Kbytes para J2SE.

#### 2.4.1.2 CAMADA DE PERFIL

Segundo SouJava (2002), é necessário uma especialização da JVM e API para determinados dispositivos (ou grupos deles), para se obter performance e outros detalhes específicos. A implementação tem liberdade para estender a API e JVM para utilizar recursos do equipamento, o que causa a dependência da implementação.

Um Perfil funciona como o complemento da configuração, em que esta define o básico de funcionalidades e o perfil atende detalhes específicos do dispositivo. Alguns dos perfis existentes são: Mobile Information Device Profile (MIDP), RMI Profile, Foundation Profile, PDA Profile, PersonalJava, etc.

O MIDP é um perfil CLDC voltado especificamente aos dispositivos portáteis. O perfil MIDP especifica interface com o usuário, entrada e persistência de dados, manipulação de eventos, modelo de armanezamento orientado a registro, rede, mensagens, e aspectos relacionados ao ambiente de programação para esses dispositivos.

Uma aplicação Java baseada na configuração CLDC e no perfil MIDP, e portanto voltada a dispositivos portáveis, é conhecida como Midlet.

#### 2.4.1.3 CAMADA DO INTERPRETADOR

Esta camada é uma implementação da máquina virtual Java customizada para dispositivos específicos.

A Kilobyte Virtual Machine (KVM) é o interpretador Java otimizado para dispositivos portáteis com recursos limitados. A KVM é extremamente pequena e eficiente, garantindo ótima performance.

Segundo SUN (2002), o nome Kilobyte provém do fato de que o seu consumo de memória é tão pequeno que é medido em kilobytes. A quantidade mínima de memória requerida pela KVM é 128 Kb, incluíndo o intepretador, um mínimo de bibliotecas especificadas pela configuração e algum espaço para rodar os aplicativos.

Rodando com as bibliotecas da plataforma J2ME, da configuração CLDC e do perfil MIDP, a KVM fornece um ambiente de execução específico para pequenas aplicações.

#### 2.4.1.4 J2ME X WAP

O Wireless Application Protocol (WAP) é um conjunto de protocolos e definições destinado a disponibilizar conteúdo da Internet em dispositivos de comunicação móvel.

De acordo com Anuff, a linguagem de programação utilizada para a criação do conteúdo dos sites WAP é a Wireless Markup Language (WML). Por meio de linguagens de script (ASP, PHP, JSP) é possível criar páginas com conteúdo dinâmico.

A WML possui muitas desvantagens se comparada com a J2ME, como interação limitada, ambiente gráfico pobre, não permite processamento local e não funciona sem cobertura de rede. Em uma página WML a lógica fica do lado do servidor, o que não acontece com uma aplicação Java, que permite adicionar funcionalidade aos clientes. Na J2ME o ambiente gráfico é melhor e a interação com o usuário é mais simples e rápida, além de customizável. A J2ME também suporta falhas esporádicas na rede, conservando o estado da informação e terminando a transação quando a ligação é restabelecida, e também é independente da pilha protocolar (WAP, TPC/IP, CDMA / TDMA / i-Mode / iDEN).

A interface de um browser WAP é conveniente para acessar informações estáticas, como notícias, ou exibir uma lista de aplicações e serviços, mas o uso da tecnologia Java é mais adequada a serviços e aplicações interativas ou orientadas a transações, que o usuário pode continuar interagindo mesmo desconectado da rede.

De acordo com Motorola (2002), o WAP pode ser utilizado para fazer o download de aplicações J2ME para o dispositivo móvel. Mas não é a única opção, pois o fabricante do dispositivo pode fornecer uma aplicação específica para esse propósito.

#### 2.4.1.5 FERRAMENTAS DE DESENVOLVIMENTO

Existem várias ferramentas no mercado para desenvolver e testar aplicações J2ME, como por exemplo o JBuilder Mobile Set e o Zucotto Whiteboard.

Especificamente para a família Palm, a própria Palm disponibiliza o PalmOS Emulator (POSE). É possível escolher a versão do PalmOS, a pele (Palm III, Palm V, Palm Vx, Palm m500), e o tamanho da memória (2 MB, 8 MB).

A Motorola disponibiliza o Motorola Software Development Kit (SDK) para J2ME, que vem acompanhado de várias ferramentas para desenvolvimento, teste, depuração, emulação.

A Sun oferece o J2ME Wireless Toolkit, que vem com um emulador que permite escolher qual o dispositivo a ser emulado, e pode também ser customizado em área de pintura de tela, cores, teclas, podendo ser criado um novo dispositivo baseado nas customizações. De acordo com Nextel (2002), o J2ME Wireless Toolkit conta com várias ferramentas de apoio ao desenvolvedor, como *profiling*, *trace* de execução de código, velocidade de conexão, monitoramento de atividades de rede e memória, tamanho da *heap* e armazenamento.

Todas as ferramentas analisadas para desenvolvimento de aplicações J2ME vêm com o J2ME Wireless Toolkit integrado.

# 3 RALLY DE REGULARIDADE

O Rally de Regularidade é uma modalidade esportiva segura e barata, se comparada a outras dentro do automobilismo. É um esporte de aventura que explora o sincronismo e a comunicação entre os componentes da equipe, exigindo atenção e agilidade dos dois competidores. Cada veículo possui dois integrantes, um piloto e um navegador.

A competição é praticada em estradas normais de trânsito, onde a perícia do piloto exige ainda a ajuda de um navegador para interpretar as orientações sobre o roteiro a ser percorrido. No momento da largada é entregue aos participantes o livro de bordo que contém o percurso a ser seguido, etapa por etapa, a distância aproximada de cada trecho, média de velocidade a ser mantida e referências de modo a permitir o acerto do roteiro determinado.

O objetivo principal de uma prova de Rally de Regularidade é percorrer o roteiro correto mantendo a melhor regularidade possível, ou seja, o mais próximo das médias horárias definidas pelo livro de bordo.

## 3.1 LIVRO DE BORDO

O Livro de Bordo ou Planilha é um boleto aonde constam as informações referentes ao roteiro a ser seguido, com todas as orientações sobre a prova. O percurso a ser seguido é dividido em vários trechos, podendo existir três tipos de trechos:

a) trecho neutralizado ou neutro: é o trecho utilizado para fazer uma pausa na competição, como por exemplo, um trecho neutro de 5 minutos, onde os competidores terão que esperar esse tempo antes de prosseguir;
b) trecho de deslocamento: é aquele utilizado para o deslocamento através de cidades, vilas e locais de grande movimento. É fornecido um tempo máximo para percorrer esse trecho, sem uma velocidade determinada;
c) trecho de regularidade: é o trecho onde o participante deverá cumprir a velocidade determinada no Livro de Bordo, buscando cumprir o roteiro corretamente mantendo esta velocidade.

A dupla de competidores só terá as informações de distância e as médias horárias a serem cumpridas, e o trabalho do navegador, além de orientar o piloto no roteiro, é calcular o tempo ideal da dupla, baseado nas informações do odômetro e da velocidade.

O roteiro da prova é descrito basicamente através de quatro colunas no Livro de Bordo:

a) medição parcial ou odômetro, que informa a distância de cada referência em relação ao início do trecho. A cada início de trecho esta medição reinicia;
b) referência, é a coluna aonde consta uma referência na estrada;
c) velocidade média, informa qual velocidade deve ser respeitada. Cada trecho tem uma velocidade média, que só pode ser alterada no início de um novo trecho;
d) tempo ideal, coluna essa que mostra o tempo exato para passagem na referência. Este campo deve ser calculado pelo Navegador;

A FIGURA 2 mostra um exemplo de uma página de livro de bordo.

**FIGURA 2: EXEMPLO DE UMA PÁGINA DE LIVRO DE BORDO**

| Odômetro | Referência | Velocidade | Tempo |
|---|---|---|---|
| 0,00 | Placa | 40 | 00:15:00 |
| 2,02 | Cruzamento | | |
| 3,52 / 0,00 | Pegue à direita | 30 | 00:20:17 |
| 0,21 | Ponto de ônibus | | |
| 0,24 / 0,00 | Deslocamento | D | 00:20:46 |
| 0,78 | Pegue à esquerda | | |
| 1,48 / 0,00 | Placa à direita | 45 | 00:23:12 |
| 1,02 | Cuidado: preferencial | | |

### 3.1.1 LÓGICA DO LIVRO DE BORDO

O livro de bordo deve obedecer determinadas regras, que são:

a) o trecho inicial não pode ser neutro;
b) será iniciado um novo trecho quando o odômetro for zerado;
c) a velocidade média poderá ser alterada somente no início de um novo trecho;
d) o tempo ideal será informado no início de cada trecho;

e) o tempo ideal no início de um novo trecho é igual ao último tempo do trecho anterior, pois o ponto referencial é o mesmo, e por isso dividem a mesma linha no Livro de Bordo;
f) um trecho neutro é identificado pela velocidade média igual a N ou zero, e um trecho de deslocamento pela velocidade média igual a D ou um;
g) as linhas de um trecho neutro ou de deslocamento terão o tempo ideal nulo ou igual a zero (00:00:00), com exceção da primeira linha do trecho, pois não é necessário respeitar o tempo nesses trechos.

## 3.2 TEMPO IDEAL

O Tempo Ideal de passagem é o tempo exato em que os competidores devem passar pela referência citada no Livro de Bordo. Esse tempo deve ser calculado pelo navegador, utilizando os dados do odômetro e da velocidade média.

A fórmula para o cálculo do Tempo Ideal é a seguinte:

Tempo em minutos = (Distância em quilômetros / Velocidade média horária) * 60

## 3.3 POSTO DE CRONOMETRAGEM OU CONTROLE

Ao longo do percurso são distribuídos pela organização, vários Postos de Cronometragem (PCs) para medir o desempenho de cada competidor, sendo sua localização desconhecida dos competidores. Para cada carro que passa em frente ao PC é registrado seu tempo exato de passagem (hh:mm:ss).

## 3.4 PONTUAÇÃO E CLASSIFICAÇÃO

A medida de tempo utilizada no Rally é o segundo. Para cada segundo que o competidor passar adiantado ou atrasado em relação ao seu tempo ideal de passagem ele perderá pontos, geralmente dois pontos por segundo adiantado e um ponto por segundo atrasado.

Baseada nos tempos de passagem registrados pelos PCs e comparando com o tempo ideal de passagem de cada carro, a organização gera o número de pontos que cada competidor perdeu em cada PC ao longo da prova. Como o objetivo é fazer o menor número possível de

pontos, a pontuação é mais conhecida por penalização. O vencedor do Rally será o participante que perder o menor número de pontos na somatória de todos os Postos de Cronometragem.

# 4 DESENVOLVIMENTO DO PROTÓTIPO

O protótipo de *software* desenvolvido neste trabalho é uma aplicação para cálculo de planilha de rally de regularidade. Esta aplicação deve ser utilizada em dispositivos móveis que suportem a tecnologia J2ME, com a configuração CLDC e o perfil MIDP.

Ao desenvolver o protótipo, sempre se teve em mente possibilitar ao usuário da tecnologia J2ME, uma forma rápida e eficiente de obter todos os cálculos referentes à uma planilha de rally de regularidade, que normalmente teriam que ser feitos manualmente com o auxílio de uma calculadora, o que consome muito tempo porque a planilha geralmente é extensa. Tempo este que nem sempre o competidor de rally tem disponível, pois as vezes a planilha é entregue ao competidor minutos antes do início da prova.

## 4.1 REQUISITOS PRINCIPAIS DO PROTÓTIPO

Alguns requisitos devem estar presentes no protótipo para ser realizada a especificação. Estes requisitos correspondem à algumas características que o protótipo precisa ter para alcançar o resultado esperado.

Os requisitos principais do protótipo são:

a) será desenvolvido para operar sobre um ambiente J2ME com a configuração CLDC e o perfil MIDP;
b) irá armazenar em memória as informações de cada linha dos trechos da planilha;
c) irá obedecer a lógica do livro de bordo;
d) o tempo ideal será informado em horas, minutos e segundos;
e) o usuário poderá alterar o tempo ideal calculado pelo protótipo;
f) após armazenar as informações da planilha em memória, o usuário poderá visualizar todos os tempos calculados;

## 4.2 ESPECIFICAÇÃO DO PROTÓTIPO

O processo principal e os cinco sub-processos que formam o protótipo foram representados utilizando a metodologia de especificação denominada fluxogramação. A ferramenta Microsoft Visio 2000 foi utilizada para auxiliar na construção dos fluxogramas.

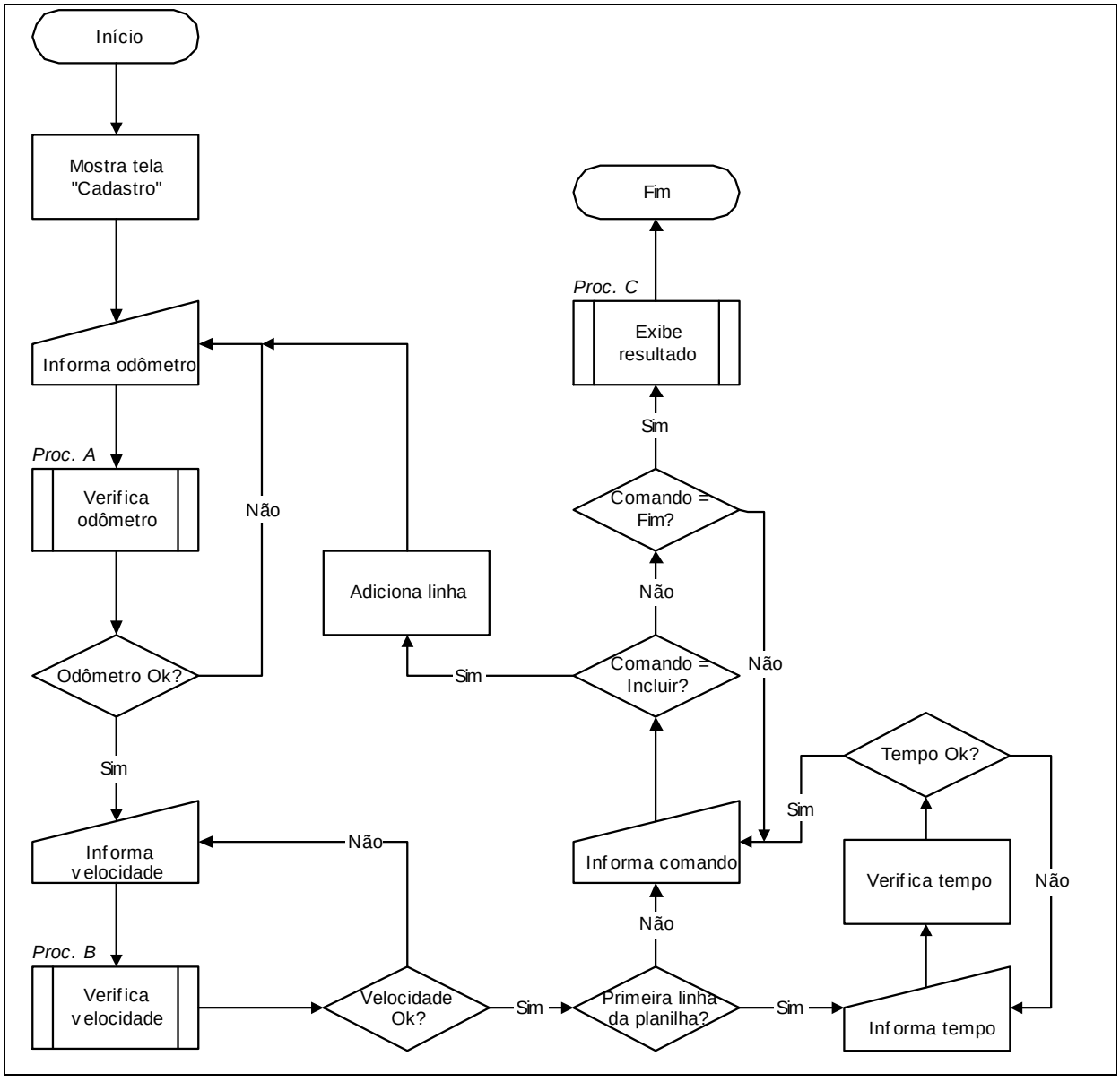
Na TABELA 1 são enumerados os sub-processos juntamente com a descrição de cada um deles.

**TABELA 1: SUB-PROCESSOS DO PROTÓTIPO**

| Nome | Descrição |
|---|---|
| Processo A | Verifica odômetro |
| Processo B | Verifica velocidade |
| Processo C | Exibe resultado |
| Processo D | Calcula tempo total |
| Processo E | Mostra tempo formatado |

A FIGURA 3 apresenta uma especificação genérica do protótipo.

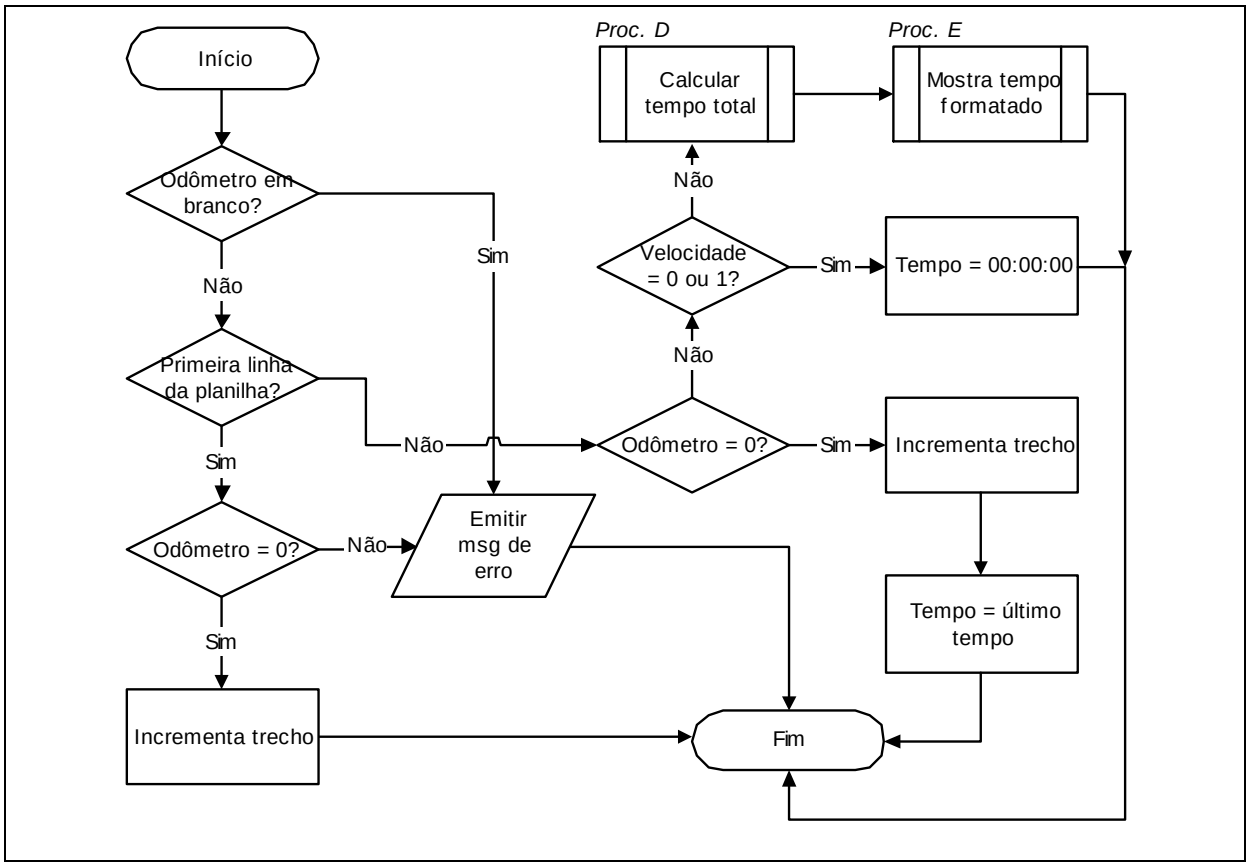
**FIGURA 3: ESPECIFICAÇÃO GENÉRICA DO PROTÓTIPO**



A seguir, são apresentados os diagramas correspondentes aos sub-processos do protótipo com suas respectivas descrições.

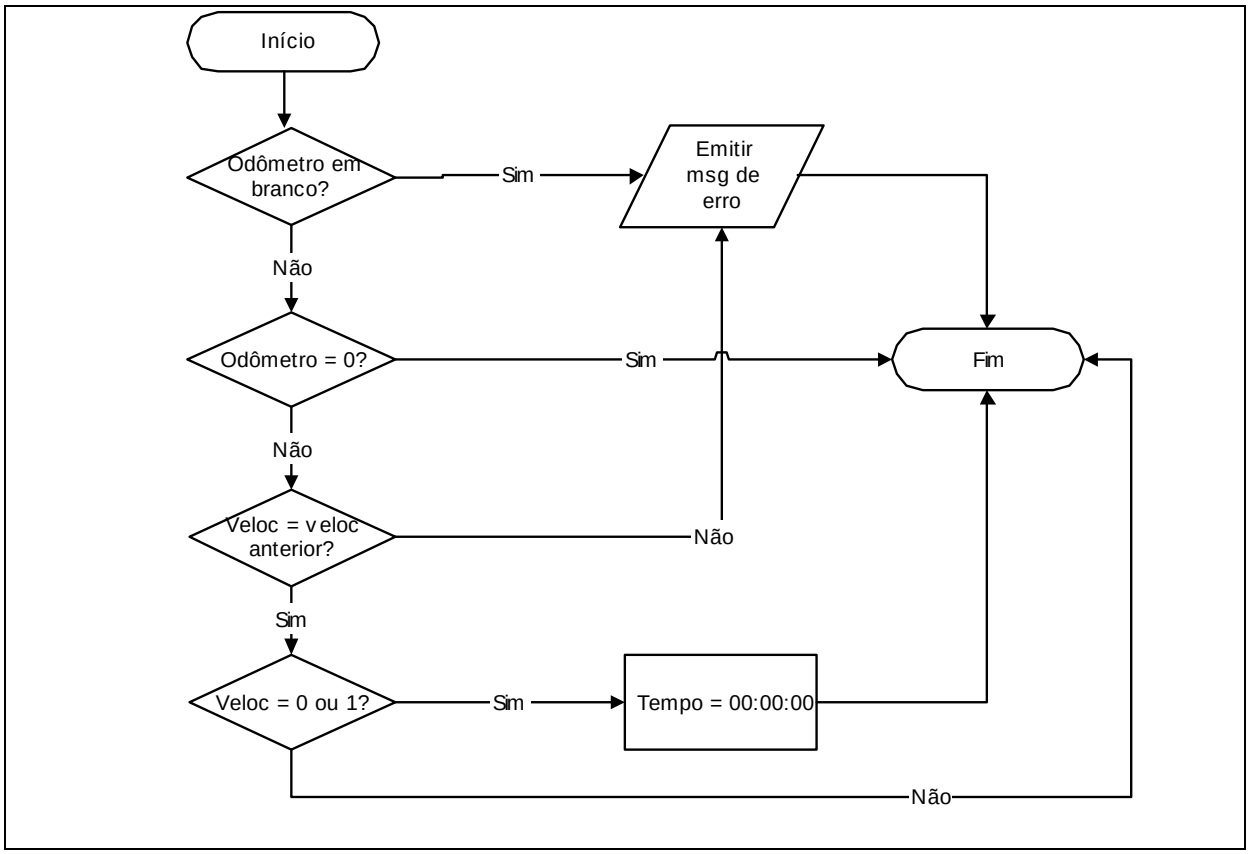
Na FIGURA 4, observa-se o detalhamento do processo A, que verifica se o odômetro informado pelo usuário está logicamente correto. O processo A faz a chamada para os processos D e E, que também são detalhados nesse capítulo.

**FIGURA 4: FLUXOGRAMA DO PROCESSO A (VERIFICA ODÔMETRO)**



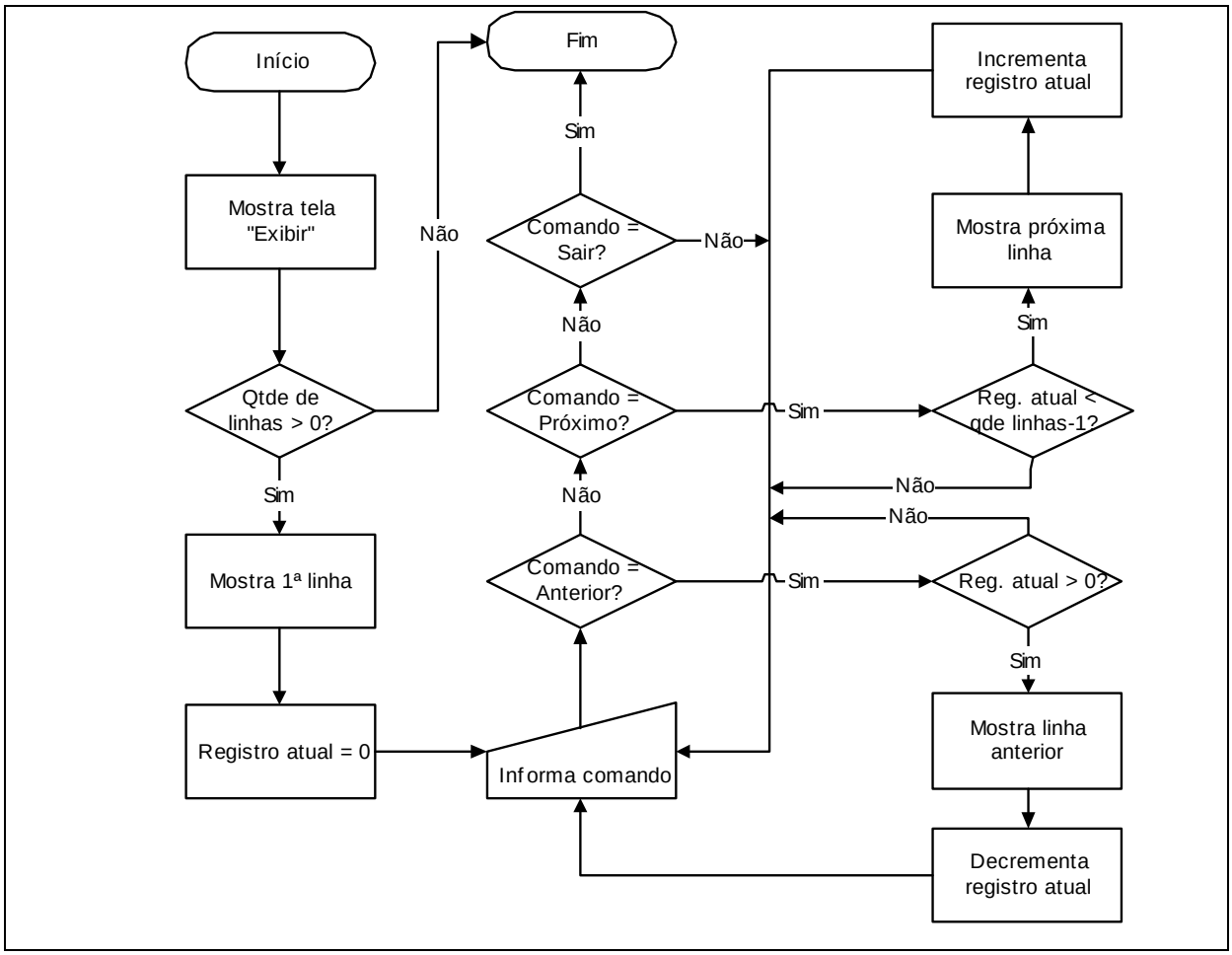

Na FIGURA 5, observa-se o detalhamento do processo B. O processo B verifica se a velocidade informada pelo usuário está logicamente correta.

**FIGURA 5: FLUXOGRAMA DO PROCESSO B (VERIFICA VELOCIDADE)**



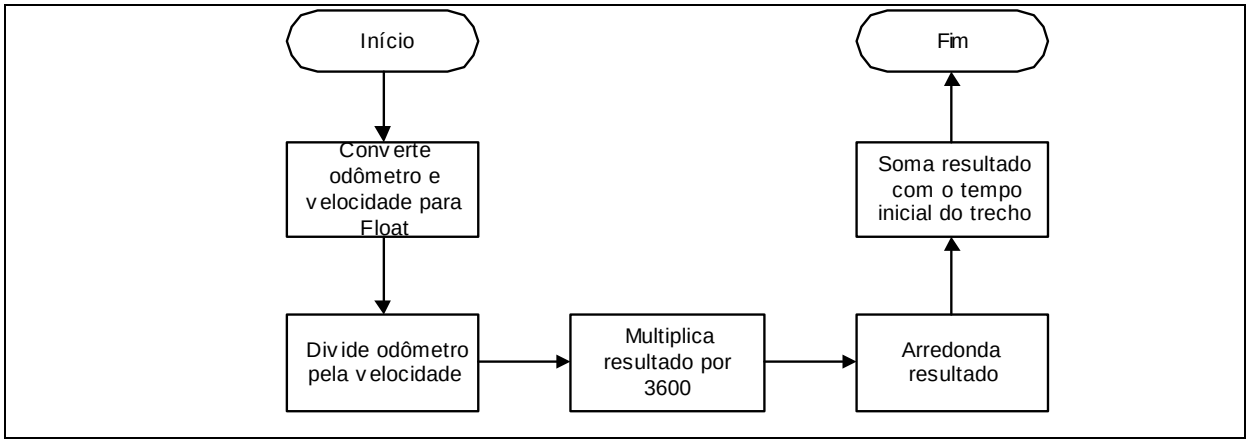

Na FIGURA 6, observa-se o detalhamento do processo C. Este processo exibe todas as linhas de cada trecho da planilha cadastradas pelo usuário.

**FIGURA 6: FLUXOGRAMA DO PROCESSO C (EXIBE RESULTADO)**



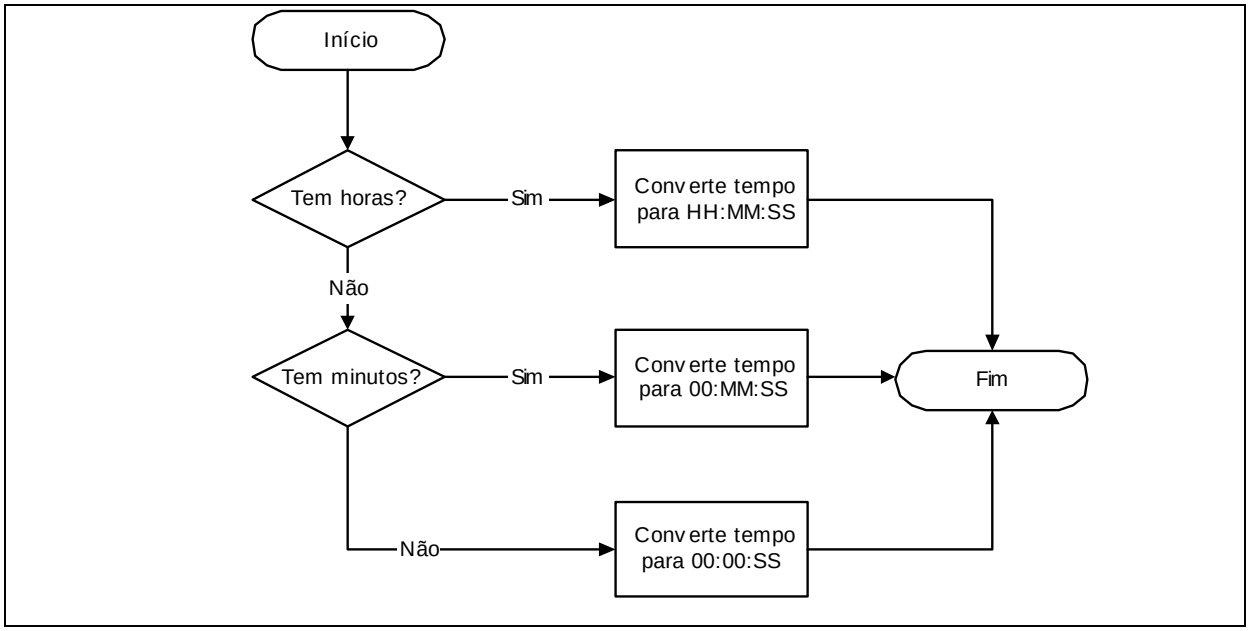

Na FIGURA 7, observa-se o detalhamento do processo D, que é um sub-processo do processo A. Neste processo é feito o cálculo do tempo ideal para a referência. Primeiramente obtem-se o tempo entre a linha inicial do trecho e a linha atual, depois soma-se com o tempo incial do trecho, resultando assim no tempo ideal para a referência.

**FIGURA 7: FLUXOGRAMA DO PROCESSO D (CALCULA TEMPO TOTAL)**



Na FIGURA 8, observa-se o detalhamento do processo E, que é um sub-processo do processo A. A entrada para este processo é o tempo total da referência em segundos, e a saída é esse tempo formatado em horas, minutos e segundos, de maneira a facilitar a leitura pelo usuário.

**FIGURA 8: FLUXOGRAMA DO PROCESSO E (MOSTRA TEMPO FORMATADO)**




## 4.3 IMPLEMENTAÇÃO DO PROTÓTIPO

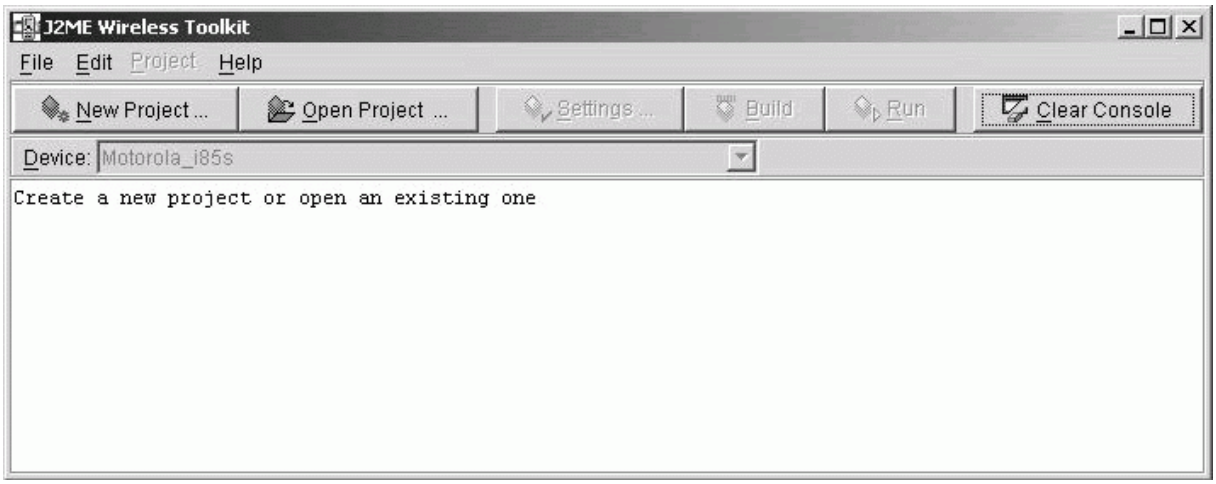
Nos tópicos a seguir serão descritas as ferramentas que foram utilizadas na implementação e o funcionamento do protótipo.

### 4.3.1 FERRAMENTAS UTILIZADAS NA IMPLEMENTAÇÃO

Para a implementação do protótipo foi utilizada a linguagem J2ME, com a configuração CLDC 1.0 e o perfil MIDP 1.0, e o ambiente de desenvolvimento J2ME Wireless Toolkit v.1.0.4_01.

O J2ME Wireless Toolkit requer o J2 SDK, Standard Edition, instalado. A versão utilizada foi a v.1.3.1_05. Ele vem com emulador para o celular i85s da Motorola, para o qual o protótipo foi desenvolvido, e não dispõe de nenhum editor integrado, sendo necessário utilizar um editor de textos convencional ou algum produto comercial para escrever o código. Na FIGURA 9 observa-se o J2ME Wireless Toolkit.

**FIGURA 9: J2ME WIRELESS TOOLKIT**




O celular Motorola i85s é um dispositivo móvel que possibilita executar aplicações J2ME. Esse dispositivo vem com algumas aplicações pré-instaladas, porém é possível transferir outras para o celular através de cabo serial. Na FIGURA 10 observa-se o celular i85s da Motorola.

**FIGURA 10: CELULAR I85S DA MOTOROLA**

### 4.3.2 PRINCIPAIS FUNÇÕES DO PROTÓTIPO

As funções apresentadas nesta seção correspondem aos cinco sub-processos do protótipo.

No QUADRO 1 pode-se visualizar o código do processo A. A especificação deste código-fonte pode ser vista na Figura 4.

**QUADRO 1: IMPLEMENTAÇÃO DO PROCESSO A (VERIFICA ODÔMETRO)**

```
  public void itemStateChanged(Item item)
    {
    if (item == edOdometro)
      {
      if (edOdometro.getString().equals(""))
        display.setCurrent(alert03);
      else
        if ((fPlanilha.fLinhas.size() == 0) && (edOdometro.getString().equals("0")
== false))
          display.setCurrent(alert05);
        else
          if (edOdometro.getString().equals("0"))
            {
            siTrecho.setText(Integer.toString(
fPlanilha.PrimeiraLinhaDoTrechoAtual().fTrecho + 1 ));
            if (fPlanilha.UltimaLinha() != null)
              edTempo.setString(
MostraTempoFormatado(fPlanilha.UltimaLinha().fTempo) ); // Tempo = último tempo
            }
          else
            // Se odômetro=0, então é a 1ª linha de um trecho de Desloc. ou Neutro,
            // e o tempo = último tempo do trecho anterior, então não deve zerar.
            if ((edVelocidade.getString().equals("0")) ||
(edVelocidade.getString().equals("1")))
              {
              if (edOdometro.getString().equals("0") == false)
                edTempo.setString("00:00:00");
              }
            else
              if (fPlanilha.PrimeiraLinhaDoTrechoAtual() != null)
                {
                int xTempoFinalCalculadoEmSegundos = CalculaTempoTotal(
edOdometro.getString(), edVelocidade.getString(),
fPlanilha.PrimeiraLinhaDoTrechoAtual().fTempo );

edTempo.setString(MostraTempoFormatado(xTempoFinalCalculadoEmSegundos));
                }
      }
    }
```

No QUADRO 2 pode-se visualizar o código do processo B. A especificação deste código-fonte pode ser vista na Figura 5.

**QUADRO 2: IMPLEMENTAÇÃO DO PROCESSO B (VERIFICA VELOCIDADE)**

```
  public void itemStateChanged(Item item)
    {
    if (item == edVelocidade)
      {
```

```
      if (edOdometro.getString().equals(""))
        display.setCurrent(alert03);
      else
        // Se odômetro <> 0, velocidade DEVE ser igual a ÚltimaVelocidade.
        if (edOdometro.getString().equals("0") == false)
          if (Integer.parseInt(edVelocidade.getString()) !=
fPlanilha.PrimeiraLinhaDoTrechoAtual().fVelocidade)
            display.setCurrent(alert01);
          else
            // Se odômetro=0, então é a 1ª linha de um trecho de Desloc. ou Neutro,
            // e o tempo = último tempo do trecho anterior, então não deve zerar.
            if ((edVelocidade.getString().equals("0")) ||
(edVelocidade.getString().equals("1")))
                edTempo.setString("00:00:00");
      }
    }
```

No QUADRO 3 pode-se visualizar o código do processo C. A especificação deste código-fonte pode ser vista na Figura 6.

**QUADRO 3: IMPLEMENTAÇÃO DO PROCESSO C (EXIBE RESULTADO)**

```
  public RallySubForm(String aTitle, RallyTimeCalculator aMidlet)
    {
    // Chama o constructor da super-classe Form.
    super(aTitle);

    // Salva referência para o MIDlet
    this.fMidlet = aMidlet;

    cmdAnterior = new Command("Anterior", Command.OK, 2);
    cmdProximo  = new Command("Próximo", Command.OK, 1);
    cmdSair = new Command("Sair", Command.OK, 3);

    siTrecho     = new StringItem("Trec.....:", "0");
    siOdometro   = new StringItem("Odm....:", "0");
    siVelocidade = new StringItem("Vlde....:", "0");
    siTempo      = new StringItem("Tmp.....:", "00:00:00");

    append(siTrecho);
    append(siOdometro);
    append(siVelocidade);
    append(siTempo);

    addCommand(cmdAnterior);
    addCommand(cmdProximo);
    addCommand(cmdSair);
    setCommandListener(this);

    if (aMidlet.fPlanilha.QuantidadeLinhas() > 0)
      atualizaTela();
    else
      setTitle("Exibir- [vazio]");

    }

  public void atualizaTela()
    {
    TLinha xLinha = fMidlet.fPlanilha.PegaLinha(fIdRegistroAtual);

    siTrecho.setText(Integer.toString(xLinha.fTrecho));
    siOdometro.setText(xLinha.fOdometro);
    siVelocidade.setText(Integer.toString(xLinha.fVelocidade));
```

```
    siTempo.setText(fMidlet.MostraTempoFormatado(xLinha.fTempo));

    setTitle("Exibir- " + Integer.toString(fIdRegistroAtual+1) + " de " +
fMidlet.fPlanilha.QuantidadeLinhas());
    }
```

No QUADRO 4 pode-se visualizar o código do processo D. A especificação deste código-fonte pode ser vista na Figura 7.

**QUADRO 4: IMPLEMENTAÇÃO DO PROCESSO D (CALCULA TEMPO TOTAL)**

```
  public int CalculaTempoTotal(String aOdometro, String aVelocidade, int
aTempoInicialDoTrecho)
    {
    Float xFloat = new Float(Integer.parseInt(aOdometro),-2);
    xFloat = xFloat.Div(new Float(Integer.parseInt(aVelocidade),0));
    xFloat = xFloat.Mul(new Float(3600));

    String vlrArredondado = xFloat.RoundFloatToString(xFloat);
    int xTempoDaLinhaAtual = Integer.parseInt(vlrArredondado);

    return (xTempoDaLinhaAtual + aTempoInicialDoTrecho);
    }
```

No QUADRO 5 pode-se visualizar o código do processo E. A especificação deste código-fonte pode ser vista na Figura 8.

**QUADRO 5: IMPLEMENTAÇÃO DO PROCESSO E (MOSTRA TEMPO FORMATADO)**

```
  public String MostraTempoFormatado(int aTempo)
    {
    String xH, xM, xS, retorno="";

    int xTmp;

    if (aTempo <= 59) // Somente segundos
      {
      if (aTempo < 9)
        retorno = "00:00:0" + Integer.toString(aTempo);
      else
        retorno = "00:00:" + Integer.toString(aTempo);
      }

    if (aTempo <= 3599) // Somente minutos e segundos (máximo 00:59:59)
      {
      xTmp = aTempo / 60;
      if (xTmp < 9)
        xM = "0" + Integer.toString(xTmp);
      else
        xM = Integer.toString(xTmp);

      xTmp = aTempo % 60;
      if (xTmp < 9)
        xS = "0" + Integer.toString(xTmp);
      else
        xS = Integer.toString(xTmp);

      retorno = "00:" + xM + ":" + xS;
      }

    if (aTempo >= 3600) // Tem horas
      {
```

```
      xTmp = aTempo / 3600;
      if (xTmp < 9)
        xH = "0" + Integer.toString(xTmp);
      else
        xH = Integer.toString(xTmp);

      xTmp = ((aTempo / 60) % 60);
      if (xTmp < 9)
        xM = "0" + Integer.toString(xTmp);
      else
        xM = Integer.toString(xTmp);

      xTmp = (aTempo % 60);
      if (xTmp < 9)
        xS = "0" + Integer.toString(xTmp);
      else
        xS = Integer.toString(xTmp);

      retorno = xH + ":" + xM + ":" + xS;
      }

    return retorno;
    }
```

A implementação completa do protótipo pode ser encontrada no Apêndice A.

### 4.3.3 PONTO-FLUTUANTE

Para o desenvolvimento do protótipo de software, foi necessário a criação de uma classe em Java que emulasse cálculos matemáticos com números de ponto-flutuante, utilizando somente aritmética inteira, pois o J2ME não suporta números de ponto-flutuante.

A classe *Float* possui operações de adição, subtração, divisão e multiplicação, além de funções de igualdade, maior que, menor que, negação e arredondamento. A implementação da operação de adição é apresentada no

QUADRO 6.

**QUADRO 6: IMPLEMENTAÇÃO DA OPERAÇÃO DE ADIÇÃO**

```
//***************************************************************************
// Adição
//***************************************************************************

  public Float Add(Float value)
    {
    if (value.Equal(ZERO))
      return new Float(this);

    long e1 = m_E;
    long e2 = value.m_E;
    long v1 = m_Val;
    long v2 = value.m_Val;

    // "e" precisa ser igual em ambos operadores
```

```
    while (e1 != e2)
      {
      if (e1 > e2)
        {
        if (Math.abs(v1) < Long.MAX_VALUE/100)
          {
          v1 *= 10;
          e1--;
          }
        else
          {
          v2 /= 10;
          e2++;
          }
        }
      else
        if (e1 < e2)
          {
          if (Math.abs(v2) < Long.MAX_VALUE/100)
            {
            v2 *= 10;
            e2--;
            }
          else
            {
            v1 /= 10;
            e1++;
            }
          }
      }

    if ( ((v1 > 0) && (v2 > Long.MAX_VALUE-v1)) || ((v1 < 0) && (v2 <
Long.MIN_VALUE-v1)) )
      {
      v1 /= 10;
      e1++;
      v2 /= 10;
      e2++;
      }

    if ((v1 > 0) && (v2 > Long.MAX_VALUE-v1))
      return new Float(ERROR);
    else
      if((v1 < 0) && (v2 < Long.MIN_VALUE-v1))
        return new Float(ERROR);

    return new Float(v1 + v2, e1);
    }
```

A implementação da operação de subtração é apresentada no QUADRO 7.

**QUADRO 7: IMPLEMENTAÇÃO DA OPERAÇÃO DE SUBTRAÇÃO**

```
//*****************************************************************************
// Subtração
//*****************************************************************************

  public Float Sub(Float value)
    {
    if (value.Equal(ZERO))
      return new Float(m_Val, m_E);

    return Add(new Float(-value.m_Val, value.m_E));
```

```
    }
```

A implementação da operação de multiplicação é apresentada no QUADRO 8.

**QUADRO 8: IMPLEMENTAÇÃO DA OPERAÇÃO DE MULTIPLICAÇÃO**

```
//*************************************************************************
// Multiplicação
//*************************************************************************

  public Float Mul(long value)
    {
    return Mul(new Float(value, 0));
    }

  public Float Mul(Float value)
    {
    if (value.Equal(ONE))
      return new Float(this);
    if (value.Equal(ZERO) || this.Equal(ZERO))
      return new Float(ZERO);

    // Verifica overflow e underflow
    do
      {
      if (Math.abs(value.m_Val) > Math.abs(m_Val))
        {
        if (Long.MAX_VALUE / Math.abs(m_Val) < Math.abs(value.m_Val))
          {
          value.m_Val /= 10;
          value.m_E++;
          }
        else
          break;
        }
      else
        {
        if (Long.MAX_VALUE / Math.abs(value.m_Val) < Math.abs(m_Val))
          {
          m_Val /= 10;
          m_E++;
          }
        else
          break;
        }
      } while(true);

    long e = m_E + value.m_E;
    long v = m_Val * value.m_Val;
    return new Float(v, e);
    }
```

A implementação da operação de divisão é apresentada no QUADRO 9.

**QUADRO 9: IMPLEMENTAÇÃO DA OPERAÇÃO DE DIVISÃO**

```
//*************************************************************************
// Divisão
//*************************************************************************

  public Float Div(long value)
    {
    return Div(new Float(value, 0));
    }
```

```
public Float Div(Float value)
  {
  if (value.Equal(ONE))
    return new Float(this);

  long e1 = m_E;
  long e2 = value.m_E;
  long v1 = m_Val;
  if (v1 == 0L)
    return new Float(ZERO);
  long v2 = value.m_Val;
  if (v2 == 0L)
    return new Float(ERROR);
  long val = 0L;

  while(true)
    {
    val += (v1 / v2);
    v1 %= v2;
    if (v1 == 0L || Math.abs(val) > (Long.MAX_VALUE / 10L))
      break;
    if (Math.abs(v1) > (Long.MAX_VALUE / 10L))
      {
      v2 /= 10L;
      e2++;
      }
    else
      {
      v1 *= 10L;
      e1--;
      }
    val *= 10L;
    }

  Float f = new Float(val, e1-e2);
  f.RemoveZero();
  return f;
  }
```

A implementação completa da classe *Float* pode ser encontrada no Apêndice B.

### 4.3.4 OPERACIONALIDADE DO PROTÓTIPO

Nesta seção é apresentado o funcionamento do protótipo de software.

O protótipo é formado por duas telas:

a) cadastrar planilha: esta tela permite ao usuário informar o odômetro, velocidade e tempo referentes a cada linha da planilha;

b) exibir: tem o objetivo de exibir as informações de trecho, odômetro, velocidade e tempo de toda a planilha. Esta tela é usada pelo usuário durante a prova de rally para consultar as informações de cada referência.

A FIGURA 11 apresenta a tela Cadastrar Planilha. As opções disponíveis nos botões são:

a) incluir: inclui as informações de trecho, odômetro, velocidade e tempo da linha atual na planilha;
b) fim: finaliza o processo de inclusão e abre a tela Exibir.

O botão menu é utilizado nessa tela para ativar a inserção de dados nos campos.

**FIGURA 11: TELA CADASTRAR PLANILHA**


A FIGURA 12 apresenta a tela Exibir. As opções disponíveis nos botões são:

a) anterior: exibe a linha anterior da planilha;
b) próximo: exibe a próxima linha da planilha.

Na tela Exibir, o botão menu possui a opção sair, que permite encerrar a execução do protótipo de software.

**FIGURA 12: TELA EXIBIR**


A FIGURA 13 apresenta uma tela com um exemplo de mensagem de erro. As mensagens de erro permanecem na tela durante 5 segundos.

**FIGURA 13: MENSAGEM DE ERRO**


## 4.4 TESTE E VALIDAÇÃO

No teste aplicado ao protótipo foi feito a inclusão de várias linhas de uma planilha, simulando uma situação real de rally de regularidade. O protótipo foi submetido a vários testes, com o objetivo de verificar se os tempos ideais estão sendo corretamente calculados, e se o protótipo respeita a lógica da planilha.

Para a realização dos testes, foi utilizado como exemplo a página de uma planilha contendo quatro trechos, em um total de 8 linhas. Essa página pode ser visualizada na Figura 2.

Na FIGURA 14 observa-se o cadastro da primeira linha da planilha de exemplo.

**FIGURA 14: CADASTRO DA PRIMEIRA LINHA**


Na FIGURA 15 observa-se o cadastro da segunda linha da planilha. Visualiza-se também que o tempo ideal já foi calculado pelo protótipo.

**FIGURA 15: CADASTRO DA SEGUNDA LINHA**


Após todos os cadastros concluídos, observa-se na FIGURA 16 a tela Exibir, aonde as três primeiras linhas, referentes ao primeiro trecho, estão sendo mostradas.

**FIGURA 16: VISUALIZAÇÃO DO PRIMEIRO TRECHO**


Na FIGURA 17 observa-se a visualização das linhas 7, 8 e 9, referentes ao terceiro trecho, que é o trecho de deslocamento.

**FIGURA 17: VISUALIZAÇÃO DO TRECHO DE DESLOCAMENTO**


Na FIGURA 18 observa-se a visualização das linhas 10 e 11, correspondentes ao último trecho da planilha de exemplo.

**FIGURA 18: VISUALIZAÇÃO DO ÚLTIMO TRECHO**


Para validar os resultados apresentados pelo protótipo, comparamos os mesmos com a planilha de exemplo calculada e mostrada na FIGURA 19.

**FIGURA 19: PLANILHA DE EXEMPLO CALCULADA.**

| Odômetro | Referência | Velocidade | Tempo |
|---|---|---|---|
| 0,00 | Placa | 40 | 00:15:00 |
| 2,02 | Cruzamento | | 00:18:03 |
| 3,52 / 0,00 | Pegue à direita | 30 | 00:20:17 |
| 0,21 | Ponto de ônibus | | 00:20:42 |
| 0,24 / 0,00 | Deslocamento | D | 00:20:46 |
| 0,78 | Pegue à esquerda | | 00:00:00 |
| 1,48 / 0,00 | Placa à direita | 45 | 00:23:12 |
| 1,02 | Cuidado: preferencial | | 00:24:34 |

O teste apresentado nesta seção, valida a implementação do protótipo especificado e comprova sua eficácia. Sendo assim, os objetivos específicos do presente trabalho, descritos na seção 1.1, foram atingidos.

# 5 CONCLUSÕES

O presente trabalho permitiu a realização de um estudo sobre a tecnologia J2ME, verificando suas características, conceitos, pontos positivos e negativos.

Durante o decorrer dos estudos observou-se que desenvolver aplicações em J2ME para dispositivos sem fio apresenta muitos desafios aos programadores, visto que o mesmo é uma versão compacta da linguagem padronizada da Sun, por ser justamente destinado ao desenvolvimento de aplicações para dispositivos móveis, e portanto possuindo algumas limitações computacionais.

Uma limitação crítica é que o J2ME não suporta números com ponto-flutuante, o que pode comprometer o desenvolvimento de aplicações específicas. Essa limitação foi introduzida no J2ME pois as operações com ponto flutuante requerem um maior processamento por parte do *hardware*, processamento este quase sempre não disponível nos dispositivos portáteis.

Observou-se que o problema do ponto-flutuante pode ser resolvido através de emulação por *software*. Para realizar o desenvolvimento do trabalho foi necessário a criação de uma classe em Java que emulasse cálculos matemáticos com números de ponto-flutuante, utilizando somente aritmética inteira.

O protótipo desenvolvido neste trabalho comprovou, através de testes, ser um *software* capaz de calcular e armazenar um Livro de Bordo completo, com informações referentes a trecho, odômetro, velocidade média, e tempo ideal.

O protótipo é útil para calcular planilhas de rally de regularidade, que normalmente teria que ser feito manualmente com o auxílio de uma calculadora. Como as planilhas são freqüentemente extensas, esses cálculos manuais consomem muito tempo, e o competidor nem sempre dispõe de tempo hábil para isso, pois em alguns rallys a planilha é entregue ao competidor minutos antes do início da prova. Nesse contexto, o protótipo se mostra uma forma rápida, segura e eficiente de obter todos os cálculos referentes à planilha, prevenindo também falhas matemáticas que o competidor possa cometer.

## 5.1 DIFICULDADES ENCONTRADAS

Foram encontradas algumas dificuldades no decorrer do trabalho, dentre as quais pode-se destacar:

a) falta de documentação sobre J2ME na biblioteca da Universidade Regional de Blumenau;
b) dificuldade de depuração do protótipo, devido a falta de ferramentas gratuitas;
c) dificuldade com a falta de suporte a números com ponto-flutuante no J2ME;
d) falta de um ambiente de desenvolvimento gratuito;
e) impossibilidade da aquisição do celular i85s da Motorola, devido à restrições impostas pela revendedora no Brasil, que obriga ao comprador adquirir um número mínimo de dois aparelhos, devidamente habilitados por um período mínimo de um ano;
f) impossibilidade da aquisição do cabo de dados para transferência do protótipo para o celular i85s da Motorola, ocasionado pela falta de tempo disponível, pois o produto deve ser importado.

## 5.2 LIMITAÇÕES

O protótipo apresenta algumas limitações, dentre as quais pode-se destacar:

a) o protótipo roda apenas sobre o ambiente J2ME com a configuração CLDC e o perfil MIDP;
b) o protótipo não permite armazenar as informações cadastradas fisicamente;
c) após o cadastro de todas as linhas, o protótipo não permite mais que nenhuma informação seja alterada;
d) o protótipo não permite exibir mais de uma linha de cada vez, devido a limitação do visor.

## 5.3 EXTENSÕES

Durante o desenvolvimento do trabalho surgiram algumas idéias para a extensão do mesmo:

a) implementar a possibilidade de alteração das informações após o cadastro das linhas ter sido concluído;

b) implementar o armazenamento físico das informações cadastradas, permitindo uma posterior consulta;
c) implementar comunicação de dados usando J2ME, como por exemplo um protótipo cliente-servidor.

# APÊNDICE A

A seguir está listado o código fonte completo do protótipo. Esta listagem contém os arquivos RallyTimeCalculator.java e RallySubForm.java.

```
/*
 FURB - CCEN - DSC - BCC Noturno
 Trabalho de Conclusão de Curso
 Blumenau, Novembro de 2002.
 Fábio Marcelo Depiné -
depine@senior.com.br

 - Arquivo: RallyTimeCalculator.java.
 - Tela principal do protótipo.
Cadastro da planilha e cálculos de
tempo.
 */


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

//**********************************
// RallyTimeCalculator
//**********************************

public class RallyTimeCalculator
extends MIDlet implements
CommandListener, ItemStateListener
  {
  private Command cmdMais;
  private Command cmdFim;
  private Display display;
  StringItem siTrecho;
  TextField edOdometro;
  TextField edVelocidade;
  TextField edTempo;
  Form frmMain;
  RallySubForm frmSub;

  TPlanilha fPlanilha;
  Alert alert01 = new Alert("Erro:");
  Alert alert02 = new Alert("Erro:");
  Alert alert03 = new Alert("Erro:");
  Alert alert04 = new Alert("Erro:");
  Alert alert05 = new Alert("Erro:");

  public RallyTimeCalculator()
    {
    alert01.setType(AlertType.ERROR);
    alert02.setType(AlertType.ERROR);
    alert03.setType(AlertType.ERROR);
    alert05.setType(AlertType.ERROR);
    alert01.setTimeout(5000);
    alert02.setTimeout(5000);
    alert03.setTimeout(5000);
    alert04.setTimeout(5000);
    alert05.setTimeout(5000);
```

```
    alert01.setString("Velocidade
média só pode ser alterada qdo muda de
trecho.");
    alert02.setString("O tempo inicial
não pode ser zero.");
    alert03.setString("Odômetro deve
ser informado.");
    alert04.setString("O trecho
inicial não pode ser Neutro e nem
nulo.");
    alert05.setString("Odômetro
inicial deve ser zero.");

    display =
Display.getDisplay(this);

    cmdMais = new Command("Incluir",
Command.OK, 1);
    cmdFim  = new Command("Fim",
Command.OK, 0);

    siTrecho     = new
StringItem("Trec:", "1");
    edOdometro   = new
TextField("Odm:", "0", 5,
TextField.NUMERIC);
    edVelocidade = new
TextField("Vlde:", "30", 3,
TextField.NUMERIC);
    edTempo      = new TextField("T:",
"02:22:31", 8, TextField.ANY);

    frmMain = new Form("Cadastrar
Planilha");
    frmMain.append(siTrecho);
    frmMain.append(edOdometro);
    frmMain.append(edVelocidade);
    frmMain.append(edTempo);

    frmMain.addCommand(cmdMais);
    frmMain.addCommand(cmdFim);
    frmMain.setCommandListener(this);
   frmMain.setItemStateListener(this);

    fPlanilha = new TPlanilha();
    }

  public void startApp()
    {
    displayMainMIDlet();
    }

  public void displayMainMIDlet()
    {
    display.setCurrent(frmMain);
```

```
    }

  public void itemStateChanged(Item
item)
    {
    if (item == edVelocidade)
      {
      if
(edOdometro.getString().equals(""))
        display.setCurrent(alert03);
      else
        // Se odômetro <> 0,
velocidade DEVE ser igual a
ÚltimaVelocidade.
        if
(edOdometro.getString().equals("0") ==
false)
          if
(Integer.parseInt(edVelocidade.getStri
ng()) !=
fPlanilha.PrimeiraLinhaDoTrechoAtual()
.fVelocidade)

display.setCurrent(alert01);
          else
            // Se odômetro=0, então é
a 1ª linha de um trecho de Desloc. ou
Neutro,
            // e o tempo = último
tempo do trecho anterior, então não
deve zerar.
            if
((edVelocidade.getString().equals("0")
) ||
(edVelocidade.getString().equals("1"))
)

edTempo.setString("00:00:00");
      }

    if (item == edOdometro)
      {
      if
(edOdometro.getString().equals(""))
        display.setCurrent(alert03);
      else
        if ((fPlanilha.fLinhas.size()
== 0) &&
(edOdometro.getString().equals("0") ==
false))
          display.setCurrent(alert05);
        else
          if
(edOdometro.getString().equals("0"))
            {

siTrecho.setText(Integer.toString(
fPlanilha.PrimeiraLinhaDoTrechoAtual()
.fTrecho + 1 ));
            if
(fPlanilha.UltimaLinha() != null)

edTempo.setString(MostraTempoFormatado
( fPlanilha.UltimaLinha().fTempo ));
// Tempo do novo trecho = último tempo
            }
          else
            // Se odômetro=0, então é
a 1ª linha de um trecho de Desloc. ou
Neutro,
            // e o tempo = último
tempo do trecho anterior, então não
deve zerar.
            if
((edVelocidade.getString().equals("0")
) ||
(edVelocidade.getString().equals("1"))
)
              {
              if
(edOdometro.getString().equals("0") ==
false)

edTempo.setString("00:00:00");
              }
            else
              if
(fPlanilha.PrimeiraLinhaDoTrechoAtual(
) != null)
                {
                int
xTempoFinalCalculadoEmSegundos =
CalculaTempoTotal(
edOdometro.getString(),
edVelocidade.getString(),
fPlanilha.PrimeiraLinhaDoTrechoAtual()
.fTempo );

edTempo.setString(MostraTempoFormatado
( xTempoFinalCalculadoEmSegundos ));
                }
      }
    }

  public void commandAction(Command c,
Displayable s)
    {
    if (c == cmdMais)
      {
      TLinha xLinha;

      // A planilha nunca pode começar
com Neutro (0), somente com Desloc.
(1).
      if ((fPlanilha.fLinhas.size() ==
0) &&
(edVelocidade.getString().equals("")
||
edVelocidade.getString().equals("0")))
        display.setCurrent(alert04);
      else
        if ((fPlanilha.fLinhas.size()
== 0) &&
(edOdometro.getString().equals("0") ==
false))
          display.setCurrent(alert05);
        else
          if
(edOdometro.getString().equals(""))
```

```
display.setCurrent(alert03);
          else
            if
((TempoEmSegundos(edTempo.getString())
== 0) &&
(Integer.parseInt(siTrecho.getText())
== 1))

display.setCurrent(alert02);
            else
              {
              xLinha = new TLinha();
              xLinha.fTrecho     =
Integer.parseInt(siTrecho.getText());
              xLinha.fOdometro   =
edOdometro.getString();
              xLinha.fVelocidade =
Integer.parseInt(edVelocidade.getStrin
g());
              xLinha.fTempo      =
TempoEmSegundos(edTempo.getString());

fPlanilha.AdicionaLinha(xLinha);

              LimpaTela();

siTrecho.setText(Integer.toString(fPla
nilha.UltimaLinha().fTrecho));

edVelocidade.setString(Integer.toStrin
g(fPlanilha.UltimaLinha().fVelocidade)
);
              }
      }

    if (c == cmdFim)
      {
      frmSub  = new
RallySubForm("Exibir", this);
      display.setCurrent(frmSub);
      }

    }

  public void pauseApp()
    {
    //
    }

  public void destroyApp(boolean
unconditional)
    {
    //
    }

  public void LimpaTela()
    {
    siTrecho.setText("");
    edOdometro.setString("");
    edVelocidade.setString("");
    edTempo.setString("");
    }
```

```
  public String
MostraTempoFormatado(int aTempo)
    {
    String xH, xM, xS, retorno="";
    int xTmp;

    if (aTempo <= 59) // Somente
segundos
      {
      if (aTempo < 9)
        retorno = "00:00:0" +
Integer.toString(aTempo);
      else
        retorno = "00:00:" +
Integer.toString(aTempo);
      }

    if (aTempo <= 3599) // Somente
minutos e segundos (máximo 00:59:59)
      {
      xTmp = aTempo / 60;
      if (xTmp < 9)
        xM = "0" +
Integer.toString(xTmp);
      else
        xM = Integer.toString(xTmp);

      xTmp = aTempo % 60;
      if (xTmp < 9)
        xS = "0" +
Integer.toString(xTmp);
      else
        xS = Integer.toString(xTmp);

      retorno = "00:" + xM + ":" + xS;
      }

    if (aTempo >= 3600) // Tem horas
      {
      xTmp = aTempo / 3600;
      if (xTmp < 9)
        xH = "0" +
Integer.toString(xTmp);
      else
        xH = Integer.toString(xTmp);

      xTmp = ((aTempo / 60) % 60);
      if (xTmp < 9)
        xM = "0" +
Integer.toString(xTmp);
      else
        xM = Integer.toString(xTmp);

      xTmp = (aTempo % 60);
      if (xTmp < 9)
        xS = "0" +
Integer.toString(xTmp);
      else
        xS = Integer.toString(xTmp);

      retorno = xH + ":" + xM + ":" +
xS;
      }
    return retorno;
    }
```

```
  public int CalculaTempoTotal(String
aOdometro, String aVelocidade, int
aTempoInicialDoTrecho)
    {
    Float xFloat = new
Float(Integer.parseInt(aOdometro),-2);
    xFloat = xFloat.Div(new
Float(Integer.parseInt(aVelocidade),0)
);
    xFloat = xFloat.Mul(new
Float(3600));

    String vlrArredondado =
xFloat.RoundFloatToString(xFloat);
    int xTempoDaLinhaAtual =
Integer.parseInt(vlrArredondado);

    return (xTempoDaLinhaAtual +
aTempoInicialDoTrecho);
    }

  public boolean VerificaTempo(String
aTempo)
    {
    if ((aTempo.length() == 6) &&
(aTempo.indexOf(':') == -1)) // HHMMSS
      if ((aTempo.charAt(4) <= '5') &&
(aTempo.charAt(5) <= '9')) // SS ok
        if ((aTempo.charAt(2) <= '5')
&& (aTempo.charAt(3) <= '9')) // MM ok
          if (((aTempo.charAt(0) ==
'2') && (aTempo.charAt(1) <= '3')) ||
((aTempo.charAt(0) <= '1') &&
(aTempo.charAt(1) <= '9'))) // HH ok
            return true;

    if ((aTempo.length() == 8) &&
(aTempo.charAt(2) == ':') &&
(aTempo.charAt(5) == ':')) // HH:MM:SS
      if ((aTempo.charAt(6) <= '5') &&
(aTempo.charAt(7) <= '9')) // SS ok
        if ((aTempo.charAt(3) <= '5')
&& (aTempo.charAt(4) <= '9')) // MM ok
          if (((aTempo.charAt(0) ==
'2') && (aTempo.charAt(1) <= '3')) ||
((aTempo.charAt(0) <= '1') &&
(aTempo.charAt(1) <= '9'))) // HH ok
            return true;

    return false;
    }

  public int TempoEmSegundos(String
aTempo)
    {
    int tmpValue = 0;

    if (VerificaTempo(aTempo))
      {
      if ((aTempo.length() == 6) &&
(aTempo.indexOf(':') == -1)) // HHMMSS
        {
        tmpValue =
(Integer.parseInt(aTempo.substring(0,2
)) * 3600);
        tmpValue +=
(Integer.parseInt(aTempo.substring(2,4
)) * 60);
        tmpValue +=
Integer.parseInt(aTempo.substring(4,6)
);
        }
      if ((aTempo.length() == 8) &&
(aTempo.charAt(2) == ':') &&
(aTempo.charAt(5) == ':')) // HH:MM:SS
        {
        tmpValue =
(Integer.parseInt(aTempo.substring(0,2
)) * 3600);
        tmpValue +=
(Integer.parseInt(aTempo.substring(3,5
)) * 60);
        tmpValue +=
Integer.parseInt(aTempo.substring(6,8)
);
        }
      }
    return tmpValue;
    }

  } // end RallyTimeCalculator


//************************************
// TPlanilha
//************************************

class TPlanilha
  {
  Vector fLinhas;

  public TPlanilha()
    {
    fLinhas = new Vector();
    }

  public void AdicionaLinha(TLinha
aLinha)
    {
       fLinhas.addElement(aLinha);
    }

  public int QuantidadeLinhas()
    {
    return fLinhas.size();
    }

  public TLinha PegaLinha(int i)
    {
    TLinha tmpLinha = null;

    tmpLinha = (TLinha)
fLinhas.elementAt(i);
    return tmpLinha;
    }

  public TLinha UltimaLinha()
```

```
    {
    TLinha tmpLinha = null;

    if (QuantidadeLinhas() > 0)
      tmpLinha = (TLinha)
fLinhas.elementAt(QuantidadeLinhas() -
1);

    return tmpLinha;
    }

  public TLinha
PrimeiraLinhaDoTrechoAtual()
    {
    int i;
    TLinha tmpLinha = null;
    if (fLinhas.isEmpty())
      return tmpLinha;

    // Vai voltando até que odômetro=0
    for (i=fLinhas.size(); i>=0 ; --i)
      {
      tmpLinha = (TLinha)
fLinhas.elementAt(i-1);
      if (tmpLinha != null)
```

```
        if
(tmpLinha.fOdometro.equals("0"))
          return tmpLinha;
      }
     return tmpLinha;
    }

  } // end TPlanilha


//***********************************
// TLinha
//***********************************

class TLinha
  {
  int fTrecho = 0;
  String fOdometro = "";
  int fVelocidade = 0;
  int fTempo = 0;
  } // end TLinha
```

```
/*
 FURB - CCEN - DSC - BCC Noturno
 Trabalho de Conclusão de Curso
 Blumenau, Novembro de 2002.
 Fábio Marcelo Depiné -
depine@senior.com.br

 - Arquivo: RallySubForm.java.
 - Classe da tela de visualização da
planilha calculada.
 */

import javax.microedition.lcdui.*;

//***********************************
// RallySubForm
//***********************************

public class RallySubForm extends Form
implements CommandListener
  {
  private Command cmdAnterior;
  private Command cmdProximo;
  private Command cmdSair;
  private RallyTimeCalculator fMidlet;
  StringItem siTrecho;
  StringItem siOdometro;
  StringItem siVelocidade;
  StringItem siTempo;
  int fIdRegistroAtual = 0;

  public RallySubForm(String aTitle,
RallyTimeCalculator aMidlet)
    {
    // Call the Form constructor
(chama o constructor da super-classe
Form, por isso o "extends Form").
    super(aTitle);

    // Save reference to MIDlet
```

```
    this.fMidlet = aMidlet;

    cmdAnterior = new
Command("Anterior", Command.OK, 2);
    cmdProximo  = new
Command("Próximo", Command.OK, 1);
    cmdSair = new Command("Sair",
Command.OK, 3);

    siTrecho     = new
StringItem("Trec.....:", "0");
    siOdometro   = new
StringItem("Odm....:", "0");
    siVelocidade = new
StringItem("Vlde....:", "0");
    siTempo      = new
StringItem("Tmp.....:", "00:00:00");

    append(siTrecho);
    append(siOdometro);
    append(siVelocidade);
    append(siTempo);

    addCommand(cmdAnterior);
    addCommand(cmdProximo);
    addCommand(cmdSair);
    setCommandListener(this);

    if
(aMidlet.fPlanilha.QuantidadeLinhas()
> 0)
      atualizaTela();
    else
      setTitle("Exibir- [vazio]");
    }

  public void atualizaTela()
    {
```

```
TLinha xLinha =
fMidlet.fPlanilha.PegaLinha(fIdRegistr
oAtual); //toda vez q exibe um
registro, cria um cara desses...


siTrecho.setText(Integer.toString(xLin
ha.fTrecho));

siOdometro.setText(xLinha.fOdometro);

siVelocidade.setText(Integer.toString(
xLinha.fVelocidade));

siTempo.setText(fMidlet.MostraTempoFor
matado(xLinha.fTempo));

    setTitle("Exibir- " +
Integer.toString(fIdRegistroAtual+1) +
" de " +
fMidlet.fPlanilha.QuantidadeLinhas());
    }

  public void commandAction(Command c,
Displayable s)
    {
    if (c == cmdAnterior)
      {
      if (fIdRegistroAtual > 0)
        {
        --fIdRegistroAtual;
        atualizaTela();
        }
      }

    if (c == cmdProximo)
      {
      if (fIdRegistroAtual <
(fMidlet.fPlanilha.QuantidadeLinhas()-
1))
        {
        ++fIdRegistroAtual;
        atualizaTela();
        }
      }

    if (c == cmdSair)
      {
      fMidlet.destroyApp(false);
      fMidlet.notifyDestroyed();
      }
    }

 } // end RallySubForm
```

# APÊNDICE B

A seguir está listado o código fonte da classe Float, criada com a finalidade de emular cálculos matemáticos com números de ponto-flutuante, utilizando somente aritmética inteira, visto que o J2ME não suporta números com ponto-flutuante. Esta listagem contém o arquivo Float.java.

```
/*
 FURB - CCEN - DSC - BCC Noturno
 Trabalho de Conclusão de Curso
 Blumenau, Novembro de 2002.
 Fábio Marcelo Depiné -
depine@senior.com.br

 - Arquivo: Float.java.
 - Classe que emula números com ponto-
flutuante, que não existem
   em J2ME (MIDP 1.0 e CLDC 1.0).
 - Aqui estão implementados vários
métodos utilizados para cálculos
   com ponto-flutuante, usando somente
aritmética inteira.
*/

//***********************************
// Float
//***********************************

public class Float
  {
  final static Float ERROR = new
Float(-93242L);
  final static public Float ZERO = new
Float();
  final static public Float ONE = new
Float(1L);

  public long m_Val;
  public long m_E;

  public Float()
    {
    m_Val = m_E = 0;
    }

  public Float(long value)
    {
    m_Val = value;
    m_E = 0;
    }
  public Float(long value, long e)
    {
    m_Val = value;

    if (m_Val == 0)
      m_E = 0;
    else
      m_E = e;
    }

  public Float(Float value)
    {
    m_Val = value.m_Val;

    if (m_Val == 0)
      m_E = 0;
    else
      m_E = value.m_E;
    }

//***********************************
// toLong
//***********************************

  public long toLong()
    {
    long tmpE = m_E;
    long tmpVal = m_Val;

    while (tmpE != 0)
      {
      if(tmpE < 0)
        {
        tmpVal /= 10;
        tmpE++;
        }
      else
        {
        tmpVal *= 10;
        tmpE--;
        }
      }
    return (int) tmpVal;
    }

//***********************************
// toShortString
//***********************************

  public String toShortString()
    {
    Long l = new Long(m_Val);
    String str = l.toString();
    int len = str.length() + (int)m_E;

    if (len > 0)
      return str.substring(0, len);

    return "0";
```

```
    }

//***********************************
// toString
//***********************************

  public String toString()
    {
    if (this.Equal(ERROR))
      return "NaN";

    Long l = new Long(m_Val);
    String str = l.toString();
    int len = str.length();

    if (m_E < 0L)
      {
      int absE = (int) Math.abs(m_E);
      if (absE < len)
        {
        str = str.substring(0, len-
absE) + "." + str.substring(len-absE);
        }
      else
        {
        for (int i=0; i < (absE-len);
i++)
          str = "0" + str;
        str = "0." + str;
        }
      }
    else
      {
      for (int i=0; i < m_E; i++)
        str = str + "0";
      }
    return str;
    }

//***********************************
// Adição
//***********************************

  public Float Add(Float value)
    {
    if (value.Equal(ZERO))
      return new Float(this);

    long e1 = m_E;
    long e2 = value.m_E;
    long v1 = m_Val;
    long v2 = value.m_Val;

    // "e" precisa ser igual em ambos
operadores
    while (e1 != e2)
      {
      if (e1 > e2)
        {
        if (Math.abs(v1) <
Long.MAX_VALUE/100)
          {
          v1 *= 10;
          e1--;
          }
        else
          {
          v2 /= 10;
          e2++;
          }
        }
      else
        if (e1 < e2)
          {
          if (Math.abs(v2) <
Long.MAX_VALUE/100)
            {
            v2 *= 10;
            e2--;
            }
          else
            {
            v1 /= 10;
            e1++;
            }
          }
      }

    if ( ((v1 > 0) && (v2 >
Long.MAX_VALUE-v1)) || ((v1 < 0) &&
(v2 < Long.MIN_VALUE-v1)) )
      {
      v1 /= 10;
      e1++;
      v2 /= 10;
      e2++;
      }

    if ((v1 > 0) && (v2 >
Long.MAX_VALUE-v1))
      return new Float(ERROR);
    else
      if((v1 < 0) && (v2 <
Long.MIN_VALUE-v1))
        return new Float(ERROR);

    return new Float(v1 + v2, e1);
    }

//***********************************
// Subtração
//***********************************

  public Float Sub(Float value)
    {
    if (value.Equal(ZERO))
      return new Float(m_Val, m_E);
    return Add(new Float(-value.m_Val,
value.m_E));
    }

//***********************************
// Multiplicação
//***********************************

  public Float Mul(long value)
    {
    return Mul(new Float(value, 0));
    }
```

```
  public Float Mul(Float value)
    {
    if (value.Equal(ONE))
      return new Float(this);
    if (value.Equal(ZERO) ||
this.Equal(ZERO))
      return new Float(ZERO);

    // Verifica overflow e underflow
    do
      {
      if (Math.abs(value.m_Val) >
Math.abs(m_Val))
        {
        if (Long.MAX_VALUE /
Math.abs(m_Val) <
Math.abs(value.m_Val))
          {
          value.m_Val /= 10;
          value.m_E++;
          }
        else
          break;
        }
      else
        {
        if (Long.MAX_VALUE /
Math.abs(value.m_Val) <
Math.abs(m_Val))
          {
          m_Val /= 10;
          m_E++;
          }
        else
          break;
        }
      } while(true);

    long e = m_E + value.m_E;
    long v = m_Val * value.m_Val;
    return new Float(v, e);
    }

//***********************************
// Divisão
//***********************************

  public Float Div(long value)
    {
    return Div(new Float(value, 0));
    }

  public Float Div(Float value)
    {
    if (value.Equal(ONE))
      return new Float(this);

    long e1 = m_E;
    long e2 = value.m_E;
    long v1 = m_Val;
    if (v1 == 0L)
      return new Float(ZERO);
    long v2 = value.m_Val;
    if (v2 == 0L)
      return new Float(ERROR);
    long val = 0L;

    while(true)
      {
      val += (v1 / v2);
      v1 %= v2;
      if (v1 == 0L || Math.abs(val) >
(Long.MAX_VALUE / 10L))
        break;
      if (Math.abs(v1) >
(Long.MAX_VALUE / 10L))
        {
        v2 /= 10L;
        e2++;
        }
      else
        {
        v1 *= 10L;
        e1--;
        }
      val *= 10L;
      }

    Float f = new Float(val, e1-e2);
    f.RemoveZero();
    return f;
    }

//***********************************
// RemoveZero
//***********************************

  public void RemoveZero()
    {
    if (m_Val == 0)
      return;
    while (m_Val%10 == 0)
      {
       m_Val /= 10;
       m_E++;
      }
    }

//***********************************
// Maior que
//***********************************

  public boolean Great(Float x)
    {
    long e1 = m_E;
    long e2 = x.m_E;
    long v1 = m_Val;
    long v2 = x.m_Val;

    while (e1 != e2)
      {
      if (e1 > e2)
        {
        if (Math.abs(v1) <
Long.MAX_VALUE / 100)
          {
          v1 *= 10;
          e1--;
          }
        else
```

```
          {
          v2 /= 10;
          e2++;
          }
        }
        else
          if (e1 < e2)
            {
            if (Math.abs(v2) <
Long.MAX_VALUE / 100)
              {
              v2 *= 10;
              e2--;
              }
            else
              {
              v1 /= 10;
              e1++;
              }
            }
      }
    return v1 > v2;
    }

//***********************************
// Menor que
//***********************************

  public boolean Less(long x)
    {
    return Less(new Float(x, 0));
    }

  public boolean Less(Float x)
    {
    long e1 = m_E;
    long e2 = x.m_E;
    long v1 = m_Val;
    long v2 = x.m_Val;

    while (e1 != e2)
      {
      if (e1 > e2)
        {
        if (Math.abs(v1) <
Long.MAX_VALUE / 100)
          {
          v1 *= 10;
          e1--;
          }
        else
          {
          v2 /= 10;
          e2++;
          }
        }
        else
          if (e1 < e2)
            {
            if (Math.abs(v2) <
Long.MAX_VALUE / 100)
              {
              v2 *= 10;
              e2--;
              }
```

```
            else
              {
              v1 /= 10;
              e1++;
              }
            }
      }
    return v1 < v2;
    }

//***********************************
// Igualdade
//***********************************

  public boolean Equal(Float x)
    {
    long e1 = m_E;
    long e2 = x.m_E;
    long v1 = m_Val;
    long v2 = x.m_Val;

    while (e1 != e2)
      {
      if (e1 > e2)
        {
        if (Math.abs(v1) <
Long.MAX_VALUE / 100)
          {
          v1 *= 10;
          e1--;
          }
        else
          {
          v2 /= 10;
          e2++;
          }
        }
      else
        if (e1 < e2)
          {
          if (Math.abs(v2) <
Long.MAX_VALUE / 100)
            {
            v2 *= 10;
            e2--;
            }
          else
            {
            v1 /= 10;
            e1++;
            }
          }
      }
    return (v1 == v2);
    }

//***********************************
// Negação
//***********************************

  public Float Neg()
    {
    return new Float(-m_Val, m_E);
    }
```

```
//**********************************
// Arredondamento - retorna em string
//**********************************

  public String RoundFloatToString(
Float xFloat )
    {
    String strOrig =
xFloat.toString();
    if (strOrig.indexOf('.') != -1) //
Tem ponto decimal
      {
      int posPonto =
strOrig.indexOf('.');
      String strDecimal =
strOrig.substring(posPonto+1,posPonto+
3);
```

```
      int duasPrimeirasCasasDecimais =
Integer.parseInt(strDecimal);
      if (duasPrimeirasCasasDecimais
>= 55)
        {
        if (xFloat.Great(xFloat.ZERO))
// Se for maior que zero...
          xFloat =
xFloat.Add(xFloat.ONE); //   Soma 1
        else
// Senão...
          xFloat =
xFloat.Sub(xFloat.ONE); //   Subtrai 1
        return xFloat.toShortString();
// Retorna parte inteira
        }
      }
    return xFloat.toShortString();
    }

} // end Float
```

# REFERÊNCIAS BIBLIOGRÁFICAS

ANUFF, Ed. **Java sourcebook**. Nova York: Wiley, 1996.

FONSECA, Jorge Cavalcanti. **Portando a KVM**. 2002. 64 f. Trabalho de Conclusão de Curso (Graduação em Ciências da Computação) – Centro de Informática, Universidade Federal de Pernambuco, Recife.

HOFF, Arthur Van; SHAIO, Sami; STARBUCK, Orca. **Ligado em Java**. São Paulo: Makron Books, 1996.

HOPSON, K.C.; INGRAM, Stephen E. **Developing professional Java applets**. Indianapolis: Sams.Net, 1998.

INDRUSIAK, Leandro Soares. **Linguagem Java,** [S.l.], nov. 2002. Disponível em: <http://www.inf.ufrgs.br/tools/java/introjava.pdf>. Acesso em: 17 nov. 2002.

MICROJAVA Network. **The J2ME resource,** [S.l.], nov. 2002. Disponível em: <http://www.microjava.com/>. Acesso em: 17 nov. 2002.

MOTOROLA. **Developer community,** [S.l.], nov. 2002. Disponível em: <http://idenphones.motorola.com/iden/developer/developer_home.jsp>. Acesso em: 17 nov. 2002.

NEXTEL. **Programa de desenvolvedores,** [S.l.], nov. 2002. Disponível em: <http://www.nextel.com.br/w_prog_desen_br.htm>. Acesso em: 17 nov. 2002.

NUNES, Nuno. **A microarquitectura PicoJava,** [S.l.], nov. 2002. Disponível em: <http://www.fe.up.pt/~ee99043/picojava/picojava.pdf>. Acesso em: 17 nov. 2002.

SOUJAVA. **Sociedade de usuários Java,** [S.l.], nov. 2002. Disponível em: < http://www.soujava.org.br/>. Acesso em: 17 nov. 2002.

SUN. **Technology for creating mobile devices**, [S.l.], nov. 2002. Disponível em: <http://java.sun.com/products/cldc/wp/KVMwp.pdf>. Acesso em: 17 nov. 2002.

SUN J2ME. **Java 2 platform, micro edition**, [S.l.], nov. 2002. Disponível em: <http://java.sun.com/j2me/>. Acesso em: 17 nov. 2002.

SUN Brasil. **Soluções corporativas**, [S.l.], nov. 2002. Disponível em: <http://br.sun.com/>. Acesso em: 17 nov. 2002.

SUN Wireless. **Java wireless developer**, [S.l.], nov. 2002. Disponível em: <http://wireless.java.sun.com/>. Acesso em: 17 nov. 2002.