Otimização de Código na Criação de Animação Gráfica para Dispositivos Móveis, utilizando J2ME

Fábio Ottoni Júlio[1], Lívia Márcia Silva[1]

[1]Departamento de Ciência da Computação – DCC
Universidade Presidente Antônio Carlos – UNIPAC – Barbacena/MG - Brasil

fojulio2004@yahoo.com.br, livimarcia@yahoo.com.br

Resumo. *Este artigo tem como finalidade demonstrar técnicas para otimização de um código utilizado na criação de computação gráfica voltado para a utilização em dispositivos móveis, mais especificamente, celulares. Será proposto um código em J2ME devidamente otimizado com base em um código previamente testado. O código testado utiliza-se de uma imagem onde a mesma simula um cronômetro. Será verificado os pontos onde deverá ser otimizado através de testes.*

Palavras-chave*: CDC, CLDC, Computação Gráfica, J2ME, Java, MIDP, Otimização.*

1. Introdução

Com o crescimento do uso de dispositivos móveis cada vez mais modernos e com capacidade de processamento bem mais veloz, faz-se necessário o desenvolvimento de aplicações que possam acompanhar tais evoluções.

Assim, a computação gráfica está literalmente ligada a tal evolução tecnológica, bem como os dispositivos móveis que estão deixando de ser meramente um celular ou um palm qualquer, e viraram um centro de divertimento e de tecnologia.

A plataforma escolhida para se realizar os testes foi o J2ME, que é uma versão reduzida para desenvolvimento na linguagem JAVA que permite que aplicativos sejam criados para dispositivos móveis com diversas vantagens, como por exemplo, a manipulação de imagens com a intenção de se utilizar melhor os recursos disponíveis e poder atingir um número maior de dispositivos, não ficando limitado somente a dispositivos mais modernos.

O J2ME foi projetado para dispositivos com limitações tanto na quantidade de memória, tamanho e dimensão de tela e poder de processamento. Assim, o trabalho tende a demonstrar técnicas para otimização de código, a fim de se ter um

melhor aproveitamento do dispositivo e até mesmo um melhor desempenho do aplicativo e/ou código a ser utilizado.

A fim de se padronizar os desenvolvimentos para tais dispositivos a SUN introduziu o conceito de Configurações[1] a fim de se ter uma ampla variedade de produtos que se encaixam dentro do escopo do J2ME. Existem dois tipos de Configurações recentemente definidas: o CDC[2] e CLDC[3]. (MUCHOW, 2006)

A diferença entre as configurações CDC e CLDC está presente na hora do desenvolvimento para tais dispositivos que utilizem estas configurações, ou seja, uma dispõe de recursos avançados do aparelho, que é o caso da CDC, e a outra dispõe apenas de recursos básicos (tabela 1):

Tabela 1 – Comparação entre os tipos de Configurações recentemente criadas

CDC – Configuração de Dispositivo Conectado	CLDC – Configuração de Dispositivo Conectado Limitado
- 512 kilobytes (no mínimo) de memória para executar o Java; - 256 kilobytes (no mínimo) de memória para alocação de memória em tempo de execução; - Conectividade de rede, largura de banda possivelmente persistente e alta.	- 128 kilobytes de memória para executar o Java; - 32 kilobytes para alocação de memória em tempo de execução; - Interface restrita com o usuário; - Baixo poder, normalmente alimentado por bateria; - Conectividade de rede, normalmente dispositivos sem fio com largura de banda baixa e acesso intermitente.

Esta divisão criada pelos grupos de trabalho do setor do código-fonte aberto utilizando o *Java Community Process Program* da Sun, foi no intuito de se padronizar técnicas de desenvolvimento e também a fim de se efetuar uma separação para qual tipo de dispositivo móvel deve-se focar o desenvolvimento. (MUCHOW, 2006)

Contudo, esta separação às vezes não é fácil, visto que a tecnologia está em amplo crescimento e com o avanço tecnológico haverá uma sobreposição entre tais categorias, o que seria inevitável. Assim, veio a idéia da criação de Perfis[4].

Existem seis tipos de Perfis em utilização (tabela 2):

Tabela 2 – Tipos de Perfis em utilização do mercado

[1] Configurações - Define uma plataforma Java para uma ampla variedade de dispositivos, ou seja, define os recursos da linguagem Java e as bibliotecas Java básicas da JVM (Java Virtual Machine) para essa configuração em particular.

[2] CDC – Configuração de Dispositivo Conectado

[3] CLDC – Configuração de Dispositivo Conectado Limitado

[4] Perfis – Pode ser chamada de uma extensão de uma configuração. Ele fornece as bibliotecas para um desenvolvedor escrever aplicativos para um tipo em particular de dispositivo, p. ex. MIDP

Mobile Information Device Profile 1.0 – MIDP 1.0
Mobile Information Device Profile 2.0 – MIDP 2.0
PDA Profile 1.0
Foundation Profile
Personal Basis Profile
Personal Profile

Neste artigo será utilizado o MIDP 1.0 que é voltado para dispositivos móveis com menos recursos do que os que utilizam o MIDP 2.0. Foi escolhido este perfil tendo em vista a escassez de recurso disponíveis nestes dispositivos, bem como a proposta do trabalho que é a melhor utilização de recursos na manipulação de imagens.

Será tratado nas próximas sessões sobre por que e quando otimizar, tipos de otimização, ferramentas de otimização, bem como as técnicas utilizadas neste artigo.

2. Por que e quando otimizar

De acordo com E. JÚNIOR (2002), otimizar significa "estabelecer o valor ótimo de algo ou alguma coisa, através da criação das condições mais favoráveis possíveis para chegar a esse objetivo".

Assim, com base neste conceito:

> *"Em qualquer ambiente computacional, a aplicação de técnicas de otimização é uma tarefa de extrema relevância. Na área de computação móvel, a importância dessa tarefa chega a ser fundamental e às vezes decisiva no processo de desenvolvimento de software". (JÚNIOR, 2002)*

Martin J. Wells em J2ME Game Programming Citado por E. JÚNIOR (2002) é enfático ao afirmar: "*Não tente otimizar código que você pensa que é lento, otimize aquilo que você sabe que é lento!*".

Enquanto desenvolvedores, deve-se tentar ao máximo fazer com que o código que esteja sendo desenvolvido apresente uma forma próxima ao ideal. Se a preocupação maior for em otimizar durante o desenvolvimento, não se consegue chegar ao fim do objetivo que é de se fazer um código teoricamente correto, ou então, o tempo de desenvolvimento se tornará inviável, visto que perde-se tempo em tentar otimizar algo que não se sabe se precisa ser otimizado.

Então, deve-se ter em mente que a otimização não é somente verificar a eficiência do algoritmo desenvolvido, deve-se preocupar, também, com o tamanho da aplicação, pois pode ocorrer que a aplicação esteja em um patamar de eficiência extremamente alto, porém pode ocorrer que não se consiga implantar em um determinado dispositivo.

2.1 Tipos de Otimização

Os tipos de otimização surgiram a partir de pequenas dificuldades ao se desenvolver aplicações que funcionem tanto em dispositivos móveis de pequeno porte como de grande porte. As dificuldades que surgiram foram no sentido de que não bastava apenas ter uma aplicação ideal, mas sim ter uma aplicação que pudesse ser implantada nos mais diversos tipos de dispositivos. Assim as otimizações foram classificados da seguinte forma: (JÚNIOR, 2002)

2.1.1 Otimização de Alto Nível

Trata-se de escolher um algoritmo correto que possa resolver um determinado problema. Sabe-se que um problema em si, pode ser resolvido de várias maneiras, mas, sabe-se também, que existem diferenças que podem ser razoáveis em se tratando de eficiência e eficácia, principalmente se levar em consideração a pequena disponibilidade de recursos de hardwares oferecidos por alguns dispositivos móveis.

E. JÚNIOR (2002) inclusive cita um exemplo onde se precisa ordenar um vetor de 20 posições, e que poderia usar dois algoritmos diferentes: o método da Bolha ou o Quicksort. Foi escolhido o método da Bolha, pois a ordenação se tratava de um número pequeno de posições de um vetor, onde o método escolhido era mais eficaz. Já o método do Quicksorte é mais eficiente para ordenação de um número grande de posições. (JÚNIOR, 2002)

Assim, o simples fato de se escolher um método que seja eficiente para atender aquele problema já estaria sendo feito uma otimização de alto nível.

2.1.2 Otimização de Baixo Nível

Já a otimização de Baixo Nível trata da mudança de pequenos trechos de código, onde uma simples mudança pode ocasionar um ganho significativo de desempenho e de tamanho do arquivo.

Ao se deparar com um código, o simples fato de olhar e identificar que em uma variável do tipo string pode ser alterada para inteiro ou diminuir o número de parâmetros em um método, pode significar um ganho de desempenho. Assim, tal ganho pode ocorrer em uma simples linha de código.

2.2 Ferramentas e Técnicas de Otimização

Algumas ferramentas estão disponíveis nas próprias IDE's de desenvolvimento, como é o caso da ferramenta WTK[5] que possui uma ferramenta de *profiling*[6]. Além dessa ferramenta pode-se fazer chamadas ao método *System.currentTimeMillis()*, a fim de se obter manualmente o tempo total decorrido. Ocorre que tal utilização é trabalhosa e pode inserir código inútil ao funcionamento da aplicação, podendo, inclusive, inserir erros. (JÚNIOR, 2002)

[5] WKT - Wireless Toolkit 2.5-windows – uma ferramenta para desenvolvimento

[6] *Profiling* – Ferramentas normalmente incorporadas às IDE's de desenvolvimento que auxiliam na verificação de execução de código e a verificação de quanto de recurso cada método está utilizando do dispositivo

3. O problema

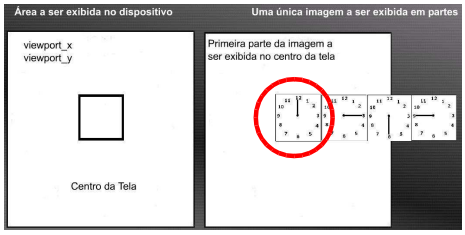
O problema proposto neste artigo é a animação gráfica de uma série de imagens que se encontram agrupadas, ou seja, trata-se da imagem de um relógio onde a mesma representa quatro posições diferentes do ponteiro que, ao ser trabalhada, irá simular um cronômetro (figura 1, figura 2, figura 3 e figura 4). O objetivo é otimizar o código proposto por J.W. MUCHOW (2006), onde o mesmo sugere um código para realizar uma animação gráfica utilizando tal imagem, fazendo com que a mesma pareça animada.

Foi escolhido este algoritmo, tendo em vista que o mesmo trata da manipulação de imagens em dispositivos com recursos reduzidos.

As figuras a seguir mostram como a figura será manipulada, demonstrando, por exemplo, o centro da tela e como a imagem será exibida na tela, de tal forma que ao longo em que o código seja executado a imagem se desloca para a esquerda simulando um cronômetro. As figuras a seguir não foram retiradas dos dispositivos, é a titulo de exemplo.

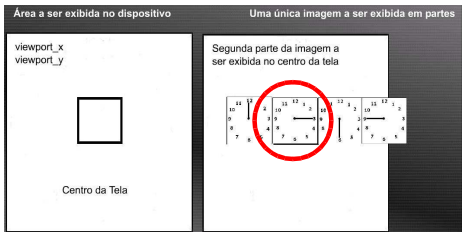
A seguir (figura 1) demonstra a imagem agrupada em sua primeira posição, isto é, demonstra o cronômetro zerado, ou seja, o ponteiro estará posicionado no número 12 do relógio, que é a posição inicial.

Figura 1 – Primeira parte da imagem encontra-se no centro da tela



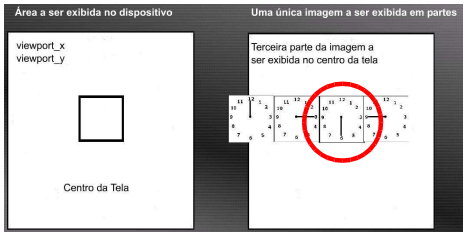
Já nesta parte (figura 2) demonstra a mesma imagem, porém na posição do número 3, significando que decorreu algum tempo entre o estado inicial e o tempo 3.

Figura 2 – Segunda parte da imagem encontra-se no centro da tela



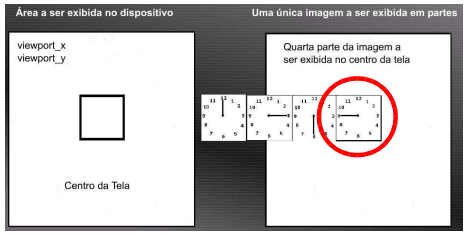
Na próxima imagem (figura 3) o cronômetro já teria avançado, passando para o estado 3, ou seja, no tempo 6 da figura.

Figura 3 – Terceira parte da imagem encontra-se no centro da tela



E por fim (figura 4), o cronômetro está na sua última posição, qual seja no tempo 9, passando para o estado 4 da imagem.

Figura 4 – Quarta parte da imagem encontra-se no centro da tela



Passando por todos os estados da imagem, o algoritmo em execução simulará uma cronômetro partido da posição 12, ou seja, da posição inicial, seguindo pelos estados 3, 6 e 9, chegando novamente na posição 12, o que iria ser repetido um número de vezes pré-definido, simulando, assim, um cronômetro de três em três tempos.

4. Estrutura Geral do Sistema

O sistema proposto por J.W. MUCHOW (2006) é composto das seguintes classes (tabela 3):

Tabela 3 – Classes disponíveis na aplicação

TimerCanvas.class	Classe responsável pela animação das imagens
AnimatedTimer.class	MIDLEt principal, mostra a tela com o cronômetro desenhado na tela
SleepForm.class	Ajusta o intervalo de pausa do cronômetro
OptionList.class	Lista as opções de configuração do cronômetro
DisplayManager.class	Gerencia os objetos Displayable

4.1 Algoritmo original

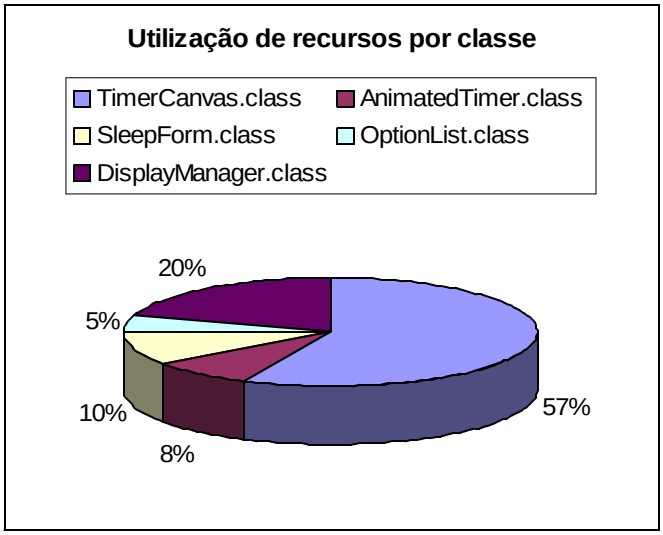
Os resultados nos testes foram obtidos através do emulador da ferramenta Wireless Toolkit 2.5 da SUN, tendo como base o algoritmo proposto por J.W. MUCHOW (2006).

Foi constatado a utilização dos recursos por classe, bem como pelos principais métodos de cada classe. Por exemplo, a classe *TimerCanvas* consumia 57% dos recursos disponíveis no momento. Tal fato se dá por, justamente, ser a classe

responsável pelo carregamento da imagem e a manipulação da mesma durante a execução da aplicação.

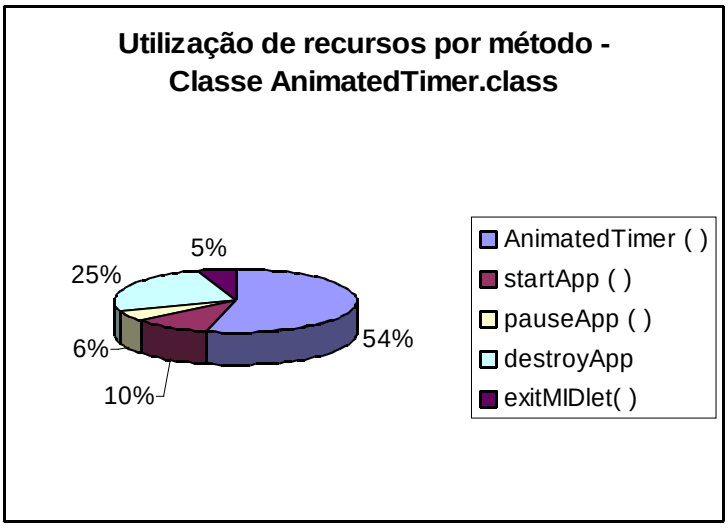
A seguir será demonstrado a utilização de recursos durante a execução da aplicação, por classe. O gráfico abaixo (figura 5) demonstra justamente a utilização desses recursos por classe.

Figura 5 – Porcentagem da utilização de recursos por classe



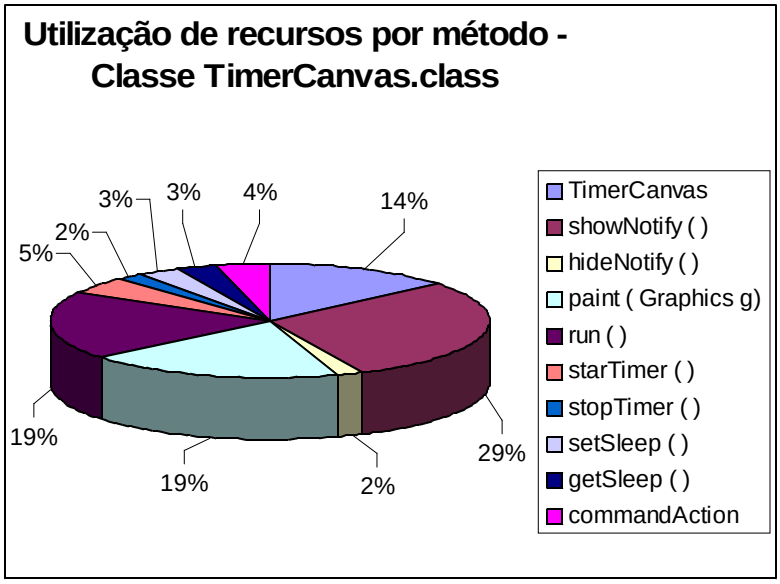
Já o gráfico a seguir (figura 6) demonstra a utilização dos recursos disponibilizados à classe *AnimatedTimer.class*.

Figura 6 – Porcentagem da utilização de recursos pelos métodos na classe AnimatedTimer.class



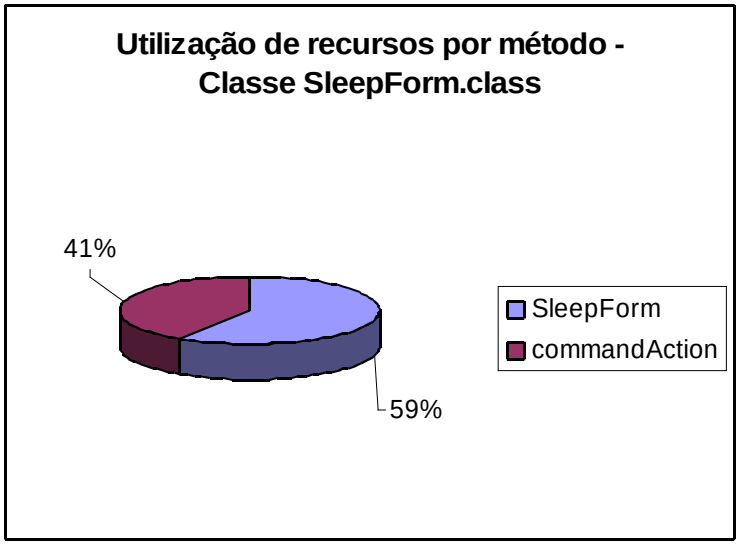
No que diz respeito a classe TimerCanvas.class, é a classe que mais utiliza recursos durante a execução. A utilização dos recursos é mostrada através de seus métodos, conforme gráfico (figura 7).

Figura 7 – Porcentagem da utilização de recursos pelos métodos na classe TimerCanvas.class



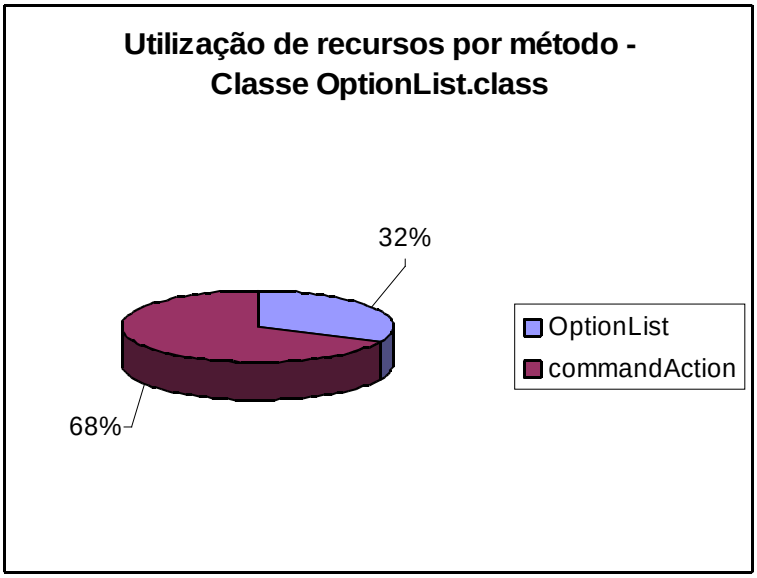
No gráfico a seguir (figura 8), é demonstrado a utilização de recursos na classe SleepForm.class.

Figura 8 – Porcentagem da utilização de recursos pelos métodos na classe SleepForm.class



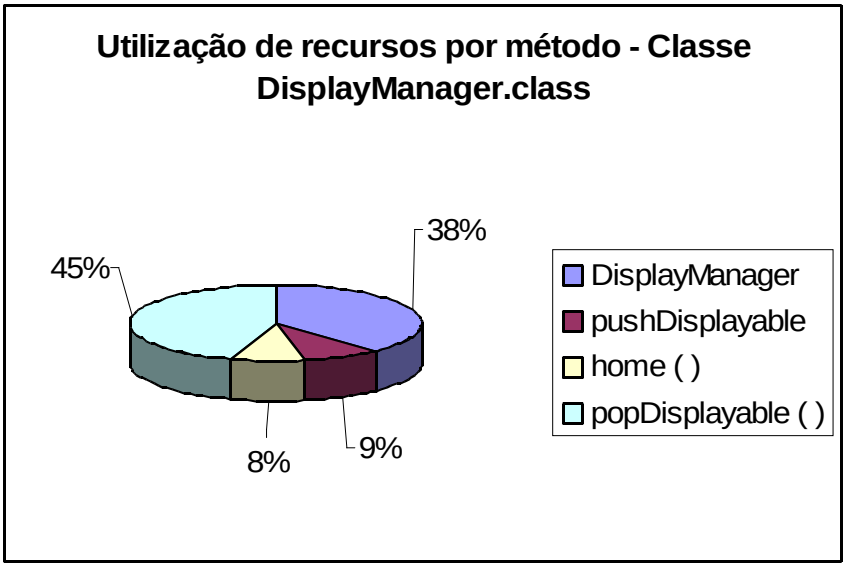
Já a classe *OptinList.class* também é demonstrada de acordo com a utilização de cada método, conforme gráfico (figura 9).

Figura 9 – Porcentagem da utilização de recursos pelos métodos na classe OptionList.class



A classe DisplayManager.class também é parte do algoritmo proposto por (MUCHOW, 2006) e é representada por quatro métodos, conforme gráfico (figura 10).

Figura 10 – Porcentagem da utilização de recursos pelos métodos na classe Displau Manager.class

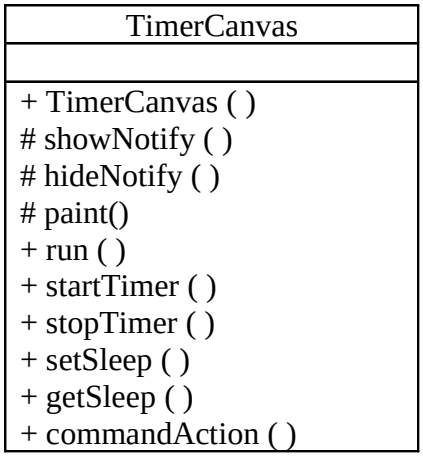


Foram realizados testes e propostas de mudanças na classe *TimerCanvas.class* e no método *static Image.createImagem(Image source)* que se encontra na classe acima citada, tendo em vista que são os pontos onde se utiliza a maior parte dos recursos disponíveis no momento.

4.2 Diagrama de Classe

Considerando que a classe que mais utiliza recursos dos dispositivo foi a classe *TimerCanvas.class*, foi realizado um estudo sobre a mesma (tabela 4).

Tabela 4 – Diagrama de classe TimerCanvas.class



5 Resultados Obtidos

Foram utilizadas as seguintes técnicas de otimização de alto e baixo nível, conforme descrito abaixo.

5.1 Otimização de Alto Nível

Foi constatado que a classe que mais utiliza recursos do emulador de dispositivo móvel foi a classe *TimerCanvas.class*, ou seja, 57%. Assim, foram realizados testes sem obtenção de êxito significativo nesta classe. Os métodos utilizados por J.W. MUCHOW (2006) em seu algoritmo utilizam-se recursos específicos e já pre-definidos no manuseio de imagens, tais como leitura de imagens (*Image.createImagem*), dimensionamento da imagem, etc.

Assim, nesta classe não foi constatado pontos onde podem ser otimizados utilizando-se da técnica de Alto Nível.

5.2 Otimização de Baixo Nível

Com relação a otimização de Baixo Nível, todos os tipos utilizados pelo algoritmo de J.W. MUCHOW (2006) também são tipos pré-definidos e corretos para a manipulação de imagens.

Como a limitação de recursos em dispositos é fortemente exigida, não foi encontrado pontos a serem otimizados.

Foi realizado testes por amostragem, ou seja, realizados testes onde a utilização de recursos é visivelmente exigida. Tal verificação é possível através da própria IDE de desenvolvimento da ferramente Wireless Toolkit.

Foi proposta a utilização do método *static Image.createImage (Image source)* no lugar do método *static Image.createImage(String name)*. O primeiro método é responsável por criar uma imagem estática a partir do objeto Imagem existente, já o segundo é responsável por criar uma imagem a partir do recurso. Com isso teve um aumento de utilização de recurso em 0,9%, ou seja, um aumento significativo. A intensão era diminuir a utilização de recursos e não aumentar.

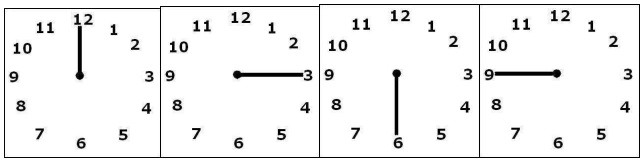
Com isso, chegou-se a conclusão que os métodos utilizados por J.W. MUCHOW (2006) não são possíveis de otimização, visto que se encontram em um estado próximo ao ideal ou perfeito na manipulação de imagens.

6. Considerações finais

Ao realizar os testes, foi constatado que o algoritmo proposto por J.W. MUCHOW (2006) encontra-se em seu estado ótimo, considerando os testes acima mencionados.

A idéia principal do algoritmo é a utilização de uma única imagem, onde a mesma encontra-se agrupada com outras 3 partes de uma mesma imagem, porém com o ponteiro do relógio em posições diferentes. Assim, esta imagem (figura 11) demonstra como esse imagem será composta.

Figura 11 – Imagem do cronômetro composta de 4 quadrantes



A imagem é composta de quatro quadrantes que, em dado momento, será movida na tela do dispositivo da direita para a esquerda, de quadrante a quadrante, a fim de se obter um deslocamento da mesma e, assim, simular uma animação gráfica.

Conforme descrito por J.W. MUCHOW (2006), a manipulação de uma única imagem ao invés da manipulação, neste caso, quatro imagens diferentes, demanda uma utilização maior de recursos do dispositivo, visto que seria necessário o carregamento de cada imagem individualmente, após o descarregamento ou limpeza da memória da imagem que estaria sendo utilizada no momento.

De fato, conforme os testes realizados, a utilização de uma única imagem, ao invés de quatro, demanda a utilização de menos recursos, visto que os dispositivos, por mais modernos que sejam, possui uma escassez de recursos, o que poderia ocasionar uma perda de performance, ou até mesmo, uma ineficiência da aplicação e do algoritmo utilizado.

Outra questão a ser analisada durante o desenvolvimento é o caso de se tratar exceções, visto que ao manipularmos várias imagens demandaria um número maior de exceções a serem tratadas, o que não é viável, considerando que o tratamento das mesmas pode comprometer a eficiência do algoritmo, porém são necessárias, mas devemos minimizar a utilização destas exceções.

Foram constatado através dos testes que as classes, bem como os métodos propostos, utilizam-se de técnicas ideais para a manipulação de imagens.

6.1 Trabalhos Futuros

Conforme dito, os avanços tecnológicos têm proporcionado o desenvolvimento de aplicações mais robustas e melhor elaboradas. Porém, nunca se deve esquecer sobre a

limitação dos dispositivos que ainda circulam no mercado, afim de se obter uma utilização maior destas aplicações.

Assim, propõem-se que a manipulação de imagens seja tratada de forma a garantir que um número maior de dispositivos possam melhor utilizar tais recursos.

Sugestões de trabalhos futuros:

- ✓ Computação gráfica na criação de animações gráficas;
- ✓ Criação de *profilings* para obtenção de resultados na utilização de recursos pelos dispositivos;
- ✓ Criação de métodos específicos para a manipulação de imagens pelos dispositivos.

7. Referências

DAIBERT, M. A. (V). Introdução ao Bluej. Aprenda visualmente programação OO e Java. *Java Magazine* (37), 62-67.

DOEDERIEIN, O. P. (V). 10 mais do Eclipse 3.2. As dez principais melhorias do mais novo release do Eclipse. *Java Magazine* (37), 20-29.

DOEDERIEIN, O. P. (V). Programação Java ME. Parte 2. Ferramentas e Primeiros Passos. *Java Magazine.* (45), 8-18.

FERNANDES, J. F. (02). Testes Unitários em JME. *Web Mobile* (12), 20-27.

HORSTMANN, C. S. (2003). *Core Java 2.* (Vols. I - Fundamentos). São Paulo: Makron Books.

JÚNIOR, E. (02). Otimização de aplicação Java ME. *Web Mobile* (ed 12), 28-36.

MATTOS, É. T. (2005). *Programação Java para Wireless - Aprenda a Desenvolver sistemas em J2ME!* São Paulo, SP, Brasil: Digerati Books.

MUCHOW, J. W. (2006). *Core J2ME, Tecnologia & MIDP*. (J. E. TORTELLO, Trans.) São Paulo, SP, Brasil: Pearson, Makron Books.