Módulo 5

Desenvolvimento de Aplicações Móveis

Lição 8

Otimizações

Versão 1.0 - Set/2007

Autor
xxx

Equipe
Rommel Feria
John Paul Petines

Necessidades para os Exercícios

Sistemas Operacionais Suportados

NetBeans IDE 5.5 para os seguintes sistemas operacionais:

- Microsoft Windows XP Profissional SP2 ou superior
- Mac OS X 10.4.5 ou superior
- Red Hat Fedora Core 3
- Solaris™ 10 Operating System (SPARC® e x86/x64 Platform Edition)

NetBeans Enterprise Pack, poderá ser executado nas seguintes plataformas:

- Microsoft Windows 2000 Profissional SP4
- Solaris™ 8 OS (SPARC e x86/x64 Platform Edition) e Solaris 9 OS (SPARC e x86/x64 Platform Edition)
- Várias outras distribuições Linux

Configuração Mínima de Hardware

Nota: IDE NetBeans com resolução de tela em 1024x768 pixel

Sistema Operacional	Processador	Memória	HD Livre
Microsoft Windows	500 MHz Intel Pentium III workstation ou equivalente	512 MB	850 MB
Linux	500 MHz Intel Pentium III workstation ou equivalente	512 MB	450 MB
Solaris OS (SPARC)	UltraSPARC II 450 MHz	512 MB	450 MB
Solaris OS (x86/x64 Platform Edition)	AMD Opteron 100 Série 1.8 GHz	512 MB	450 MB
Mac OS X	PowerPC G4	512 MB	450 MB

Configuração Recomendada de Hardware

Sistema Operacional	Processador	Memória	HD Livre
Microsoft Windows	1.4 GHz Intel Pentium III workstation ou equivalente	1 GB	1 GB
Linux	1.4 GHz Intel Pentium III workstation ou equivalente	1 GB	850 MB
Solaris OS (SPARC)	UltraSPARC IIIi 1 GHz	1 GB	850 MB
Solaris OS (x86/x64 Platform Edition)	AMD Opteron 100 Series 1.8 GHz	1 GB	850 MB
Mac OS X	PowerPC G5	1 GB	850 MB

Requerimentos de Software

NetBeans Enterprise Pack 5.5 executando sobre Java 2 Platform Standard Edition Development Kit 5.0 ou superior (JDK 5.0, versão 1.5.0_01 ou superior), contemplando a Java Runtime Environment, ferramentas de desenvolvimento para compilar, depurar, e executar aplicações escritas em linguagem Java. Sun Java System Application Server Platform Edition 9.

- Para **Solaris**, **Windows**, e **Linux**, os arquivos da JDK podem ser obtidos para sua plataforma em http://java.sun.com/j2se/1.5.0/download.html
- Para **Mac OS X**, Java 2 Plataform Standard Edition (J2SE) 5.0 Release 4, pode ser obtida diretamente da Apple's Developer Connection, no endereço: http://developer.apple.com/java (é necessário registrar o download da JDK).

Para mais informações: http://www.netbeans.org/community/releases/55/relnotes.html

Colaboradores que auxiliaram no processo de tradução e revisão

Aécio Júnior
Alexandre Mori
Alexis da Rocha Silva
Allan Souza Nunes
Allan Wojcik da Silva
Anderson Moreira Paiva
Andre Neves de Amorim
Angelo de Oliveira
Antonio Jose R. Alves Ramos
Aurélio Soares Neto
Bruno da Silva Bonfim
Carlos Fernando Gonçalves
Denis Mitsuo Nakasaki
Fábio Bombonato
Fabrício Ribeiro Brigagão
Francisco das Chagas
Frederico Dubiel
Herivelto Gabriel dos Santos
Jacqueline Susann Barbosa
João Vianney Barrozo Costa
Kefreen Ryenz Batista Lacerda
Kleberth Bezerra G. dos Santos
Leandro Silva de Morais
Leonardo Ribas Segala
Lucas Vinícius Bibiano Thomé
Luciana Rocha de Oliveira
Luiz Fernandes de Oliveira Junior
Marco Aurélio Martins Bessa
Maria Carolina Ferreira da Silva
Massimiliano Giroldi
Mauro Cardoso Mortoni
Paulo Afonso Corrêa
Paulo Oliveira Sampaio Reis
Pedro Henrique Pereira de Andrade
Ronie Dotzlaw
Seire Pareja
Sergio Terzella
Vanessa dos Santos Almeida
Robson Alves Macêdo

Auxiliadores especiais

Revisão Geral do texto para os seguintes Países:

- **Brasil** – Tiago Flach
- **Guiné Bissau** – Alfredo Cá, Bunene Sisse e Buon Olossato Quebi – ONG Asas de Socorro

Coordenação do DFJUG

- **Daniel deOliveira** – JUGLeader responsável pelos acordos de parcerias
- **Luci Campos -** Idealizadora do DFJUG responsável pelo apoio social
- **Fernando Anselmo -** Coordenador responsável pelo processo de tradução e revisão, disponibilização dos materiais e inserção de novos módulos
- **Rodrigo Nunes -** Coordenador responsável pela parte multimídia
- **Sérgio Gomes Veloso -** Coordenador responsável pelo ambiente JEDI™ (Moodle)

Agradecimento Especial

John Paul Petines – Criador da Iniciativa JEDI™
Rommel Feria – Criador da Iniciativa JEDI™

1. Objetivos

Antes de realmente fazer qualquer otimização em seus programas, deve-se certificar de que o pacote de software é de boa qualidade. A execução das otimizações nas classes do projeto deve ser deixado por último. Algumas técnicas discutidas nesta lição são úteis para evitar alguns erros de programação.

Ao final desta lição, o estudante será capaz de:

- Utilizar as diferentes técnicas de otimização em aplicações móveis.

2. Execução de Classes

2.1. Utilizar StringBuffer ao invés de String

Deve-se recordar que em Java *Strings* são objetos imutáveis. Usando-se os métodos de *String* criamos objetos individuais da *String.* A simples concatenação de *Strings* cria múltiplos objetos de *Strings* (a menos que as Strings sejam constantes e o compilador seja aguçado o suficiente para concatená-las durante sua compilação). O uso da *StringBuffer* não apenas otimiza o tempo de execução de suas classes (menos tempo de execução na criação de objetos), como também irá otimizar o uso da memória (menos *Strings* para alocar).

String	***StringBuffer***

`String a, b, c;`
`...`
`String message =`
`"a=" + a + "\n"`
`+ "b=" + b + "\n"`
`+ "c=" + c + "\n";`
	`String a, b, c;`
	`...`
	`StringBuffer message = new StringBuffer(255);`
	`message.append("a=");`
	`message.append(a);`
	`message.append("\n");`
	`message.append("b=");`
	`message.append(b);`
	`message.append("\n");`
	`message.append("c=");`
	`message.append(c);`
	`message.append("\n");`

2.2. Utilizar uma área de clipping ao desenhar

Usar *Graphics.setClip()* reduz o tempo de execução porque se está desenhando, somente, o número correto de *pixels* na tela. Lembre-se que gráficos desenhados na tela consomem muito tempo de execução. Reduzir o número de *pixels* a serem desenhados melhora o desempenho de seu programa durante a execução.

```
Graphics g;
int x1, y1, x2, y2;
...
g.setClip(x1, y1, x2, y2);
g.drawString("JEDI", x, y, Graphics.TOP | Graphics.HCENTER);
// mais operações com desenhos...
```

2.3. Evitar o modificador sincronizado

Usar um modificador do tipo sincronizado aumentará a velocidade de execução de sua classe, pois ao mesmo tempo que ele terá de executar algumas medidas extras e não poderá ser acessado simultaneamente.

2.4. Passar o menor número de parâmetros possível

Ao chamar um método, o interpretador empurra todos os parâmetros para a pilha da execução. Passar muitos parâmetros afeta a velocidade de execução e utiliza muita memória.

2.5. Reduzir as chamadas de métodos

As chamadas de métodos ocupam muita memória e tempo de execução. Veja o item anterior.

2.6. Atrasar as inicializações

Para ganhar tempo no início das aplicações, atrase todas as inicializações pesadas até que sejam

necessárias. Não ponha inicializações no construtor de *MIDlet's* ou no método *startApp*. Apressar a inicialização fará com que a aplicação demore mais para ficar plenamente utilizável. A maioria dos usuários recusaria aplicações que exigem um longo tempo para inicializar. Lembre-se que o tempo de carga da aplicação afeta diretamente a primeira impressão que o usuário tem de seu programa.

2.7. Utilizar arrays (matrizes) ao invés de Collection

Acessar matrizes é mais rápido do que usar um objeto do tipo Vector.

2.8. Utilizar atributos locais

É mais rápido acessar variáveis locais do que variáveis globais.

3. Tamanho do Arquivo JAR

3.1. Utilizar um ofuscador (obfuscator)

A intenção original do ofuscador é complicar o máximo possível os arquivos da classe compilada para que seja muito complicado reverter essa situação. O *Netbeans* e o pacote *Mobility* vêm com um ofuscador. Não está ativo por padrão. Selecione a aba propriedade da aplicação e clique no ícone "*Obfuscating*":

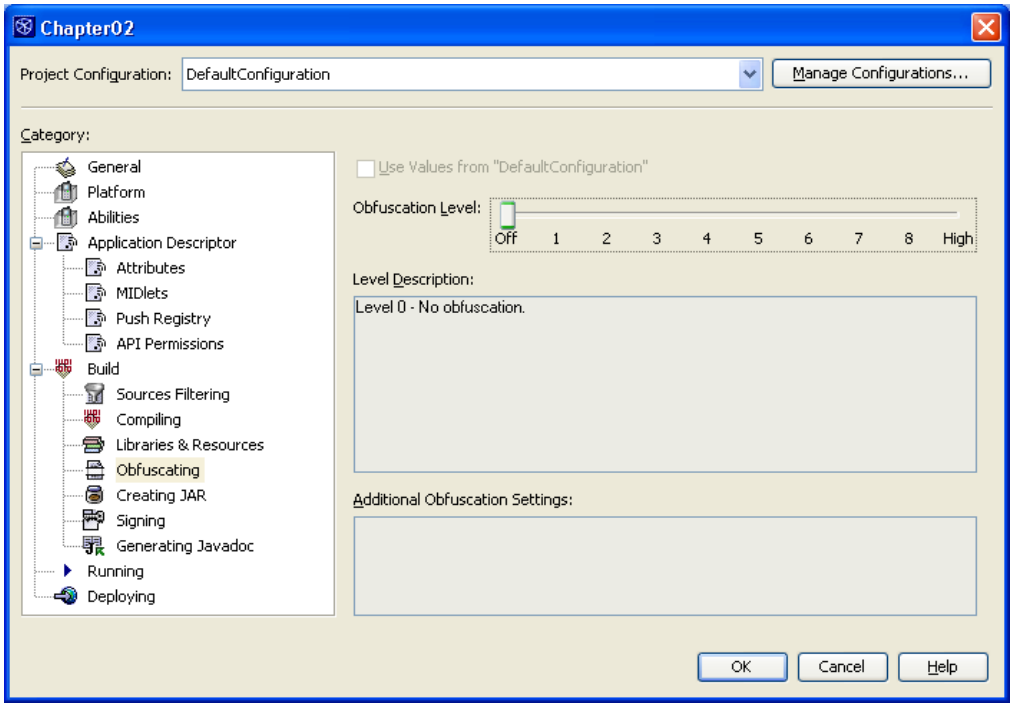

Figura 1: Ativando o ofuscador

São dez níveis de ofuscação e deve-se ser o mais agressivo possível em se tratando de ofuscamento:

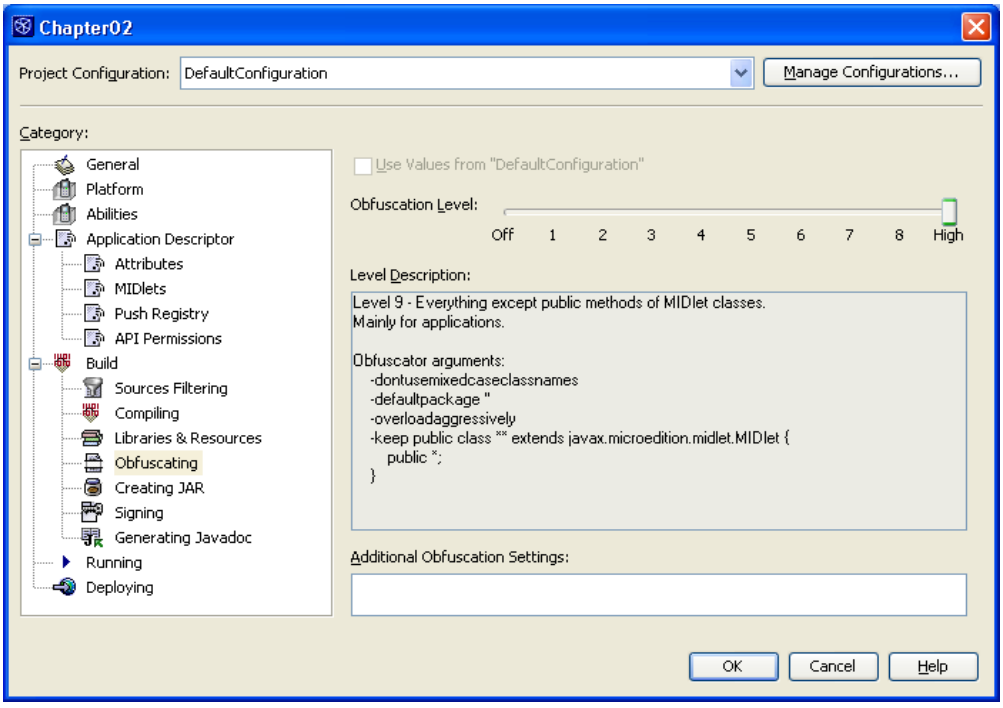

Figura 2: Maximizando o ofuscador

Entretanto, o processo de ofuscar também reduz o tamanho da aplicação. Um dos métodos empregados pelo ofuscador é renomear as classes utilizando letras simples (por exemplo, classe A). O ofuscador consegue fazer isto por possuir um transformador de métodos. Se o método tiver um modificador *private* ou *protected*, então podemos, com segurança, supor que este método não será usado por outros pacotes e poderá, conseqüentemente, ser renomeado sem problemas.

3.2. Compressão dos arquivos JAR

Antes de distribuir seu aplicativo, deve-se comprimir o arquivo JAR final para distribuí-lo. Um arquivo tipo JAR é um arquivo tipo ZIP, e um arquivo tipo ZIP possui diversos níveis de compressão (incluindo a não compressão). O NetBeans não implementa os níveis de compressão.

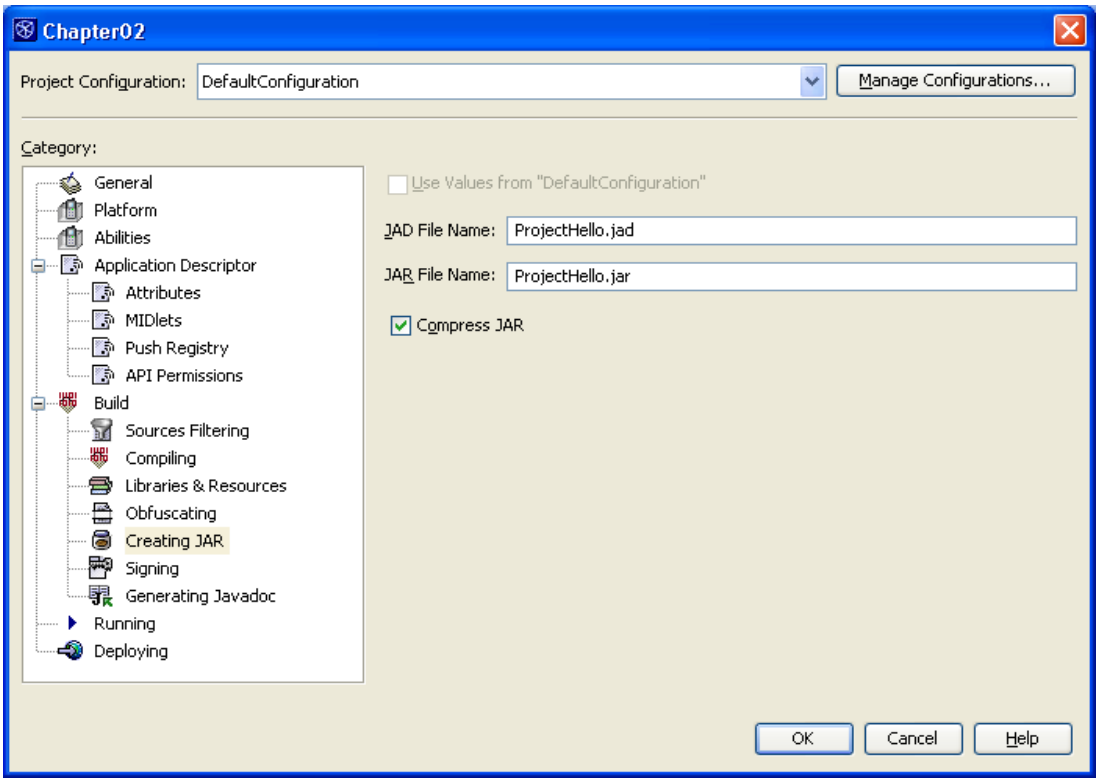

Figura 3: Comprimindo o arquivo JAR

Para definir a compressão do JAR, abra a janela de propriedades do projeto e selecione a opção "Creating JAR". Marque a opção "Compress JAR" para comprimir o arquivo final. Não esqueça de gerar (*rebuild*) o projeto novamente.

O *Netbeans* armazena o arquivo JAR final na pasta denominada *dist* abaixo da pasta de projeto. Pode-se renomear a extensão do arquivo JAR para um arquivo com extensão ZIP e abrí-lo com qualquer programa de compressão (exemplo, *WinZip*) para ver os tamanhos de seus arquivos de classe compilados.

3.3. Evitar a criação de classes desnecessárias

Isto pode parecer contraditório aos princípios da orientação a objetos, entretanto uma classe vazia e simples como:

```
public class EmptyClass {
    public EmptyClass(){}
}
```

será compilada em um arquivo do tipo class com com um tamanho de arquivo de no mínimo 250kb (não comprimíveis). Compile esta classe vazia e observe o resultado.

3.4. Evitar a criação de interfaces

Esta técnica está relacionada com a anteriormente vista. Quanto mais classes e interfaces, mais (kilo)bytes teremos na aplicação final.

3.5. Evitar a criação de classes internas e anônimas

Classes internas (*inner class*) são classificadas do mesmo modo. Classes anônimas (*anonymous class*) podem não ter um nome, entretanto ocupam o mesmo espaço nas definições de classe.

3.6. Utilizar um objeto único (padrão Singleton) para múltiplos objetos

Isto reduz o número de classes em sua aplicação. Faça com que seu *MIDlet* implemente a interface *CommandListener* e lhe ajudaria a reduzir seu pacote através de uma classe (isso é 250, ou mais, bytes menos).

3.7. Utilizar um pacote "padrão" (não significa não usar package)

Utilize um tamanho de pacote pequeno, encurtando (e não usando) nomes de pacote, o que contribui para a redução dos bytes.

3.8. Utilizar o limite dos inicializadores estáticos

Usando inicializações estáticas, tipo:

```
int[] tones = { 64, 63, 65, 76, 45, 56, 44, 88 };
```

seria traduzido pelo compilador de Java nas seguintes declarações:

```
tones[0] = 64;
tones[1] = 63;
tones[2] = 65;
tones[3] = 76;
tones[4] = 45;
tones[5] = 56;
tones[6] = 44;
tones[7] = 88;
```

Este exemplo ilustra apenas oito elementos. Pense na possibilidade de inicializar centenas de valores que utilizam declarações separadas. Tentar realizar isso, estaria muito acima do tamanho das possíveis aplicações.

Como uma alternativa, é possível utilizar o método *getResourceAsStream*() para obter valores de um arquivo ou utilizar uma única String para armazenar os valores do *array*.

3.9. Combinar as imagens em um único arquivo

Imagens são comprimidas melhor quando estão agrupadas em um único arquivo de imagem. Isso é porque a compressão do formato de imagem (formato PNG) é mais específico para imagens do que o método de compressão do arquivo JAR. Há técnicas para se obter uma imagem específica de uma imagem maior, tal como recortá-la.

3.10. Experimentar métodos diferentes de compressão de imagens

Métodos de compressão não são criados de forma semelhante. Alguns podem comprimir melhor alguns tipos de imagem mas podem ter relação de compressão pobre em outros tipos. Escolha um formato de imagem que melhora a relação de compressão. Às vezes, a relação de compressão também é afetada pelo software de imagem que se está utilizando. Experimente com manipulação de imagem diferentes programas para conseguir tamanhos de imagem melhores.

3.11. Utilizar classes pré-instaladas

Não reinvente a roda. Utilize classes disponíveis na plataforma que está usando. Criar suas classes não só aumenta o tamanho da aplicação como também diminui a estabilidade.

4. Rede

4.1. Utilizar threads

Utilize uma *thread* separada para sua função de rede para evitar travamentos da tela.

4.2. Comprimir os dados da rede

Utilize dados comprimidos para diminuir o tráfego de rede da sua aplicação. Isso requerer que seu cliente e servidor estejam utilizando o mesmo protocolo de rede e método de compressão.

Comprimir XML resulta em melhor taxa de compressão porque o XML é representado em formato texto.

4.3. Reduzir o tráfego de rede

Já que as comunicações via rede são lentas e onerosas, tente o quanto mais possível colocar dentro de uma única requisição de rede vários comandos. Isso reduzirá a sobrecarga imposta pelos protocolos de rede.

5. Uso de Memória

5.1. Utilizar estruturas de dados mais compactas

Utilize estruturas de dados mais amigáveis para memória. *Arrays* espaçados podem ser representados de outra maneira sem consumir a mesma quantidade de memória.

Existe um equilíbrio quando se otimizando tamanho e velocidade. Utilizar estruturas complexas de dados pode afetar a velocidade de execução do programa.

5.2. Liberar objetos não usados para o Garbage Collector

Libere objetos que não serão mais utilizados para o *Garbage Collector* – tela, conexões de rede, Registros RMS, entre outros. Ao atribuir para estes objetos o valor nulo, informamos ao *Garbage Collector* que estes objetos podem ser seguramente descarregados da memória.

5.3. Criar as telas que são raramente usadas como objetos anônimos

Criar os objetos de Tela que são raramente utilizadas (como telas de "auxílio" e "sobre o sistema"*)* como objetos anônimos libera a necessidade de memória *heap*, embora tenhamos que pagar o preço por uma carga mais lenta destas telas em particular. A memória *heap* destas telas supostamente seria ocupada enquanto elas não estivessem sendo usadas e pode ajudar na conservação de memória.

```
public void commandAction(Command c, Displayable d) {
    if (c == helpCommand) {
        display.setCurrent(new HelpForm());
    }
}
```

6. Exercícios

6.1. Outras idéias de otimização

Discuta outras idéias de otimização que possui ou tem em mente ou, ainda, outras técnicas que desenvolveu. Compartilhe-as.

Parceiros que tornaram JEDI™ possível

Instituto CTS
Patrocinador do DFJUG.

Sun Microsystems
Fornecimento de servidor de dados para o armazenamento dos vídeo-aulas.

Java Research and Development Center da Universidade das Filipinas
Criador da Iniciativa JEDI™.

DFJUG
Detentor dos direitos do JEDI™ nos países de língua portuguesa.

Banco do Brasil
Disponibilização de seus *telecentros* para abrigar e difundir a Iniciativa JEDI™.

Politec
Suporte e apoio financeiro e logístico a todo o processo.

Borland
Apoio internacional para que possamos alcançar os outros países de língua portuguesa.

Instituto Gaudium/CNBB
Fornecimento da sua infra-estrutura de hardware de seus servidores para que os milhares de alunos possam acessar o material do curso simultaneamente.