Curso de Java - Semana Acadêmica

Augusto Mecking Caringi
caringi.sul@terra.com.br

Outubro de 2002

Conteúdo

Capítulo 1

Introdução

1.1 História

No início da década de 90, a Sun Microsystems trabalhava no projeto Green, que tinha como objetivo desenvolver novos e revolucionários dispositivos, como por exemplo o Star7, uma espécie de controle remoto capaz de interagir com os mais diversos equipamentos, desde a televisão até o cortador de grama.

O sistema operacional do Star7 estava sendo programado em C++, entretando os engenheiros do projeto a consideravam uma linguagem muito complexa e propensa a erros, foi então que resolveram desenvolver uma nova linguagem, aproveitando as características positivas do C++ e elimando os aspectos que a tornavam uma linguagem díficil.

Para demonstrar o potencial da linguagem Java e evitar que o projeto de pesquisa fosse engavetado, foi criado, em 1994, um navegador da Web que podia executar Applets Java. Embora a linguagem tenha recebido muita atenção da comunidade da Web, ela só decolou realmente depois que a Netscape se tornou a primeira empresa a licenciá-la, em agosto de 1995. Logo após o primeiro lançamento público da linguagem, a Sun ampliou os esforços para o desenvolvimento da Java com uma nova subsidiária, chamada JavaSoft, e contratou centenas de funcionários para continuar a expandir a linguagem.

1.2 Características da Linguagem

1.2.1 Simples

Ainda que Java tenha por base o C++, ela omite muitas das características tidas como confusas do C++, incluindo aquelas que causam mais problemas aos iniciantes: ponteiros, sobrecarga de operadores, herança múltipla e templates. Ao mesmo tempo, Java adiciona uma característica importante que simplifica a programação: coleta de lixo automática.

1.2.2 Orientada a Objetos

A portabilidade é uma das características que se inclui nos objetivos almejados por uma linguagem orientada a objetos. Em Java ela foi obtida de maneira inovadora com relação ao grupo atual de linguagens orientadas a objetos. Java suporta herança, mas não herança múltipla. A ausência de herança múltipla pode ser compensada pelo uso de herança e interfaces, onde uma classe herda o comportamento de sua

superclasse além de oferecer uma implementação para uma ou mais interfaces. Java permite a criação de classes abstratas.

1.2.3 Distribuída

A linguagem Java tem uma biblioteca extensa de rotinas para lidar com protocolos TCP/IP, como HTTP e FTP. Os aplicativos Java podem abrir e acessar objetos na Internet através de URLs com a mesma facilidade que se acessa um arquivo no sistema local.

1.2.4 Robusta

Java tem a intenção de escrever programas que precisam ser confiáveis de várias formas. O Java coloca ênfase na verificação antecipada de possíveis problemas, na verificação dinâmica posterior (em tempo de execução), e na eliminação de situações sujeitas a erros... A única e grande diferença entre o Java e o C++ está em que o java usa um modelo de ponteiros que elimina a possibilidade de sobrescrita de memória e conseqüente destruição de dados.

1.2.5 Segura

A presença de coleta automática de lixo, evita erros comuns que os programadores cometem quando são obrigados a gerenciar diretamente a memória (C , C++, Pascal). A eliminação do uso de ponteiros, em favor do uso de vetores, objetos e outras estruturas substitutivas traz benefícios em termos de segurança. O programador é proibido de obter acesso a memória que não pertence ao seu programa, além de não ter chances de cometer erros comuns tais como "reference aliasing"e uso indevido de aritmética de ponteiros. Estas medidas são particularmente úteis quando pensarmos em aplicações comerciais desenvolvidas para a internet. Ser fortemente tipada também é uma vantagem em termos de segurança, que está aliada a eliminação de conversões implícitas de tipos de C++. A presença de mecanismos de tratamento de exceções torna as aplicações mais robustas, não permitindo que elas abortem, mesmo quando rodando sob condições anormais. O tratamento de exceções será útil na segunda parte para modelar situações tais como falhas de transmissão e formatos incompatíveis de arquivos.

1.2.6 Neutra em Relação à Arquitetura

O compilador gera um formato de arquivo de objeto neutro em relação a arquitetura - o código compilado é executável em muitos processadores, desde que exista uma máquina virtual. O compilador Java faz isso gerando instruções em bytecodes que nada têm a ver com uma arquitetura particular de um computador específico. Em vez disso, elas são projetadas para serem fáceis de interpretar em qualquer máquina e convertidas facilmente para o código nativo da máquina em tempo real.

1.2.7 Portável

Java foi criada para ser portável. O "byte-code"gerado pelo compilador para a sua aplicação específica pode ser transportado entre plataformas distintas que suportam Java (Solaris, Windows, Mac/Os etc). Não é necessário recompilar um programa para que ele rode numa máquina e sistema diferente, ao contrário do que acontece por exemplo com programas escritos em C e outras linguagens. Esta portabilidade é importante para a criação de aplicações para a internet. O próprio compilador Java é escrito em Java, de modo que ele é portável para qualquer sistema que possua o interpretador de "byte-codes".

1.2.8 Interpretada

O interpretador Java pode executar Java diretamente em qualquer máquina na qual o interpretador tenha sido escrito.

1.2.9 Eficiente

Como Java foi criada para ser usada em computadores pequenos, ela exige pouco espaço, pouca memória. Java é muito mais eficiente que grande parte das linguagens de "scripting"existentes, embora seja cerca de 20 vezes mais lenta que C, o que não é um marco definitivo. Com a evolução da linguagem, estão sendo criados compiladores "just-in-time"cada vez mais otimizados que trarão as marcas de desempenho da linguagem cada vez mais próximas das de C++ e C.

1.2.10 Suporta Concorrência

A linguagem permite a criação de maneira fácil, de várias "threads"de execução. Threads são muito importantes no desenvolvimento de aplicações modernas que fazem comunicação em rede, aplicações com interface gráfica, ou qualquer outra aplicação que seja necessário executar várias tarefas ao mesmo tempo.

1.2.11 Dinâmica

De várias formas, o Java é uma linguagem mais dinâmica que o C ou C++, tanto pela alocação de objetos na memória, que é feita de forma dinâmica , como pelos recursos de instrospecção de classes da linguagem.

1.3 Diferenças entre C++ e Java

Java baseia-se em C++, entretanto conta com uma série de aprimoramentos que tem como objetivo tornar a linguagem mais simples, menos confusa e segura. A seguir discutiremos detalhadamente os principais destes aprimoramentos.

- Java não permite o uso de ponteiros: C/C++ são muitas vezes classificadas como linguagens de médio nível, mais próximas das linguagens de máquina do que as outras linguagens de alto nível como Pascal, por permitirem manipulação direta de endereços de memória através de ponteiros, que não é nada mais que um tipo de variável capaz de armazenar endereços de memória. Embora os ponteiros sejam um elemento chave em C/C++, muitos programadores consideram a utilização dos ponteiros difícil e sujeita a erros.

- Java não utiliza pré-processador: Antes de se compilador um programa em C/C++, ele é submetido a um pré-processador. O pré-processador tem vários objetivos como substituição de macros e constantes, compilação condicional, etc. Java por sua vez não tem necessidade de um pré-processador pois provê alternativas que possibilitam resultados similares.

- Java passa parâmetros de tipo primitivo sempre por valor e parâmetros de tipo Array e Objeto sempre por referência: em C/C++ se utiliza ponteiros para fazer passagem de parâmetro por refêrencia (chamada por referência), Java fazendo isto automaticamente torna desnecessário o uso de ponteiros para esta tarefa.

- Os programas Java liberam automaticamente a memória alocada: Java utiliza uma técnica chamada coleta de lixo (garbage collection) para detectar quando o seu programa não utiliza mais um objeto. Diferentemente de C/C++, você não precisa fazer chamadas explícitas às funções *delete* ou *free*.

- As condições de laço em Java deverão ser do tipo boolean: Em C/C++ é possível ter uma condição contendo uma expressão que retorne qualquer tipo de valor (inteiro, ponto flutuante, etc.), isso se deve ao fato de C/C++ serem linguagens fracamente tipadas e isso pode ser a fonte de muitos bugs. Java elimina esses possíveis bugs obrigando que as condições sejam do tipo boolean (verdadeiro ou falso apenas).

1.4 Java API

A linguagem Java é pequena e simples, bem menor do que C++ e pode-se até dizer que seja menor do que C. Entretanto, ela vem com uma grande e sempre em expansão biblioteca de classes. Felizmente, você somente precisa conhecer aquelas partes da biblioteca realmente necessárias, pode-se aprender um pouco dela de cada vez, além disso, existe uma excelente documentação online em http://java.sun.com/j2se/1.4/docs/api/. As classes são agrupadas em pacotes (packages). A Java 2 API contém por volta de 60 pacotes. Para o desenvolvimento de aplicações simples, você provavelmente só precisará conhecer classes destes três pacotes:

- *java.lang* contém classes para manipular strings, e outros recursos "built-in"da linguagem;

- *java.io* contém classes que dão suporte a entrada e saída;

- *java.util* contém classes úteis como estruturas de dados, classes para se trabalhar com números aleatórios, etc.

Alguns dos principais pacotes da Java API:

Pacote	*Descrição*
`java.applet`	The Java Applet Package
`java.awt`	The Java Abstract Windowing Toolkit Package
`java.io`	The Java Input/Output Package
`java.lang`	The Java Language Package
`java.math`	The Java Arbitrary Precision Math Package
`java.net`	The Java Networking Package
`java.sql`	The Java Database Connectivity Package
`java.text`	The Java Text Package
`java.util`	The Java Utilities Package
`javax.swing`	The Java Swing GUI Components Package

1.5 Java SDK

O ambiente de desenvolvimento de software Java, Java SDK (antigamente, JDK), É formado essencialmente por um conjunto de aplicativos que permite, entre outras tarefas, realizar a compilação e a execução

de programas escritos na linguagem Java. Ele pode ser baixado gratuitamente a partir do site oficial do Java http://java.sun.com As ferramentas básicas do kit de desenvolvimento Java são:

- o compilador Java, javac,
- o interpretador de aplicações Java (máquina virtual), java e
- o interpretador de applets Java, appletviewer.

Além dessas, o Java SDK oferece os aplicativos de desenvolvimento javadoc, um gerador de documentação para programas Java; jar, um manipulador de arquivos comprimidos no formato Java Archive, jdb, um depurador de programas Java; javap, um disassembler de classes Java; e javah, um gerador de arquivos header para integração com código nativo em C. Java oferece também aplicativos para o desenvolvimento e execução de aplicações Java distribuídas.

1.6 Ambientes de Desenvolvimento Integrado

Um ambiente de desenvolvimento Integrado (IDE - Integrated Development Environment) é um conjunto de programas que combinam um editor de código-fonte, um compilador, um depurador e outras ferramentas, tais como profiler e utilitário de impressão etc., capazes de proporcionar um maior conforto no desenvolvimento de programas. Existe um grande número de IDE Java, desde iniciativas acadêmicas até ferramentas comerciais de uso profissional. Entre os mais utilizados podemos citar:

- JBuilder (Borland)
- Forte (Sun)
- Visual Café (Symantec)
- Visual J++ (Microsoft)

Capítulo 2

Fundamentos

2.1 Primeiro Exemplo: Hello World!

```
// HelloWorld.java

public class HelloWorld
{
    public static void main(String arguments[])
    {
        System.out.println("Hello World.");
    }
}
```

2.2 Tipos Primitivos

Ao declarar variáveis, deve-se informar ao compilador Java o nome e o tipo da variável. Java é uma linguagem fortemente tipada, o que significa que todas as variáveis deverão ter um tipo declarado. Um tipo define o conjunto de valores que a variável poderá armazenar. Java conta com 8 tipos primitivos:

Nome do Tipo	*Tamanho*	*Variação*
long	8 bytes	-9.223.372.036.854.775.808L até -9.223.372.036.854.775.807L
int	4 bytes	-2.147.483.648 até 2.147.483.647
short	2 bytes	-32.768 até 32.767
byte	1 byte	-128 até 127
double	8 bytes	+/- 1,79759313486231570E+308 (15 dígitos significativos)
float	4 bytes	+/- 3,40282347E+38 (7 dígitos significativos)
char	2 bytes	65.536 caracteres possíveis
boolean	1 bit	verdadeiro ou falso

2.3 Estruturas de Controle e Seleção

2.3.1 Estruturas de Seleção

A instrução condicional mais simples em Java tem a forma:

`if (condição) instrução;`

Mas em Java, como na maioria das linguagens de programação, freqüentemente se quer executar várias instruções quando uma condição apenas é verdadeira. Neste caso, a instrução condicional toma a forma:

`if (condição) {bloco}`

A construção if/else pode ser incômoda quando se tem de lidar com várias seleções com muitas alternativas. A linguagem Java, seguindo os passos do C/C++, usa a instrução switch. Pode-se testar somente o tipo char e todos os tipos inteiros, menos long; e não se pode usar conjunto de valores.

```
switch(opcao)
{
    case 1:
        . . .
        break;
    case 2:
        . . .
        break;
    case 3:
        . . .
        break;
    case 4:
        . . .
        break;
    default:
        . . .
        break;
}
```

2.3.2 Estruturas de Controle

As estruturas de repetição da linguagem Java são muito semelhantes as do C/C++. São elas: laço while, laço for e laço do.

```
while (condição)
{
    // Instruções aqui
}

for (valor_inicial; condição_de_encerramento; valor_de_incremento)
{
    // Instruções para executar
}
```

A instrução `while` permite repetir uma ou mais instruções até que uma condição específica seja verdadeira. Da mesma maneira, a instrução `for` permite repetir uma ou mais instruções por um número específico de vezes. Por sua vez, a instrução `do` permite executar uma ou mais instruções *pelo menos uma vez*, e depois, se for necessário, repetir as instruções.

```
do {
  // Bloco de instruções
} while (condição)
```

2.4 Manipulação de Strings

Quando você trabalha com strings de caractetes em Java, você utiliza instâncias da classe java.lang.String. Diferentemente das strings caractere em C/C++, os objetos String em Java não são terminados por null (terminados com um zero em ASCII). Além disso, a classe String provê uma série de métodos úteis para se trabalhar com cadeias de caracteres.

```
String vazio = ""; // string vazio
String cumprimento = "oi";
```

Operação de Concatenação: Operação binária indicada por + onde os dois operandos strings são justapostos, criando um novo string.

```
String A = "bana";
String B = "nada";
String C = A + B;
```

A string C é igual a "bananada".

Quando uma string é concatenada com um valor que não é uma string, este valor é convertido em uma string. Por exemplo:

```
String disciplina = "CI0" + 55;
```

Coloca na string disciplina o valor "CI055".

Isto é feito frequentemente em comandos de impressão, como no exemplo a seguir:

```
System.out.println("A resposta e " + resposta);
```

2.5 Funções Matemáticas

A classe `java.lang.Math` de Java oferece uma série de métodos estáticos para exponenciação, trinometria, funções de máximo e mínimo, etc. Como todos esses métodos são estáticos, é possível referenciá-los pelo nome da classe sem ter de criar uma instância da mesma.

Principais métodos da classe `Math`:

Método	*Descrição*
`abs(x)`	valor absoluto de x (esse método também tem versões para valores `float`, `int` e `long`).
`ceil(x)`	arredonda x para o menor inteiro não menor que x
`cos(x)`	cosseno trigonométrico de x (x em radianos)
`exp(x)`	método exponencial
`floor(x)`	arredonda x para o maior inteiro não maior que x
`log(x)`	logaritmo natural de x (base e)
`max(x, y)`	maior valor entre x e y (esse método também tem versões para valores `float`, `int` e `long`).
`min(x, y)`	menor valor entre x e y (esse método também tem versões para valores `float`, `int` e `long`).
`pow(x, y)`	x elevado à potência y
`sin(x)`	seno trigonométrico de x (x em radianos)
`sqrt(x)`	raiz quadrada de x
`tan(x)`	tangente trigonométrica de x (x em radianos)

2.6 Vetores e Matrizes

Vetores são objetos, eles possuem papel importante no estilo de programação desta linguagem que exclui ponteiros. Por serem objetos, vetores são obrigatoriamente alocados de maneira dinâmica. O exemplo a seguir aloca um vetor de inteiros com três posições, seguindo uma sintaxe semelhante a de alocação de objetos:

```
// VetorTest.java

public class VetorTest
{
    public static void main (String args[])
    {
        int vetor[] = new int[3];
        vetor[0]=0;       //indexacao semelhante a C , C++
        vetor[1]=10;
        vetor[2]=20;
        System.out.println(vetor[0] + " " + vetor[1] + " " + vetor[2] + " ");
    }

}
```

Perceba que a faixa útil do vetor vai de 0 até (n - 1) onde n é o valor dado como tamanho do vetor no momento de sua criação, no nosso caso 3. O mesmo ocorre com matrizes. Esta convenção pode confundir programadores Pascal onde a indexação vai de 1 até n. Java checa se você usa corretamente os índices

do vetor. Se ocorrer um acesso ao vetor[i] onde i é um índice inválido (fora da faixa de valores permitidos), você receberá uma mensagem parecida com: `java.lang.ArrayIndexOutOfBoundsException`. Esta mensagem indica que houve uma tentativa do programa de acessar um elemento fora dos limites do vetor/matriz.

Um atributo muito útil quando se trabalha em um vetor de dados: `vetor.length` com isto podemos obter o número de elementos do vetor.

```
// InitArray.java

import java.util.*;
import javax.swing.*;

public class InitArray
{
    public static void main(String args[])
    {
        String output = "Índice\tValor\n";
        int n[] = new int[10];
        Random r = new Random();

        for (int i = 0; i < n.length; i++)
        {
            n[i] = r.nextInt(100);
            output += i + "\t" + n[i] + "\n";
        }

        JTextArea outputArea = new JTextArea(11, 10);
        outputArea.setText(output);
        JOptionPane.showMessageDialog(null, outputArea,
            "Vetor com valores do tipo inteiro",
            JOptionPane.INFORMATION_MESSAGE);
        System.exit(0);
    }
}
```

Capítulo 3

Programação Gráfica: Swing

Quando a linguagem Java chegou ao mercado em 1995, ela contava com um conjunto classes para desenvolvimento de interfaces gráficas (GUI - Graphical User Interface) que a Sun chamou de AWT (Abstract Windowing Toolkit, em português: kit de ferramentas de janelas abstratas), entretanto ela era limitada e deixava a desejar em relação ao número de componentes, flexibilidade, etc. Dessa forma, quando a versão 1.2 do Java foi lançada, a linguagem incorporou um conjunto de classes denominado Swing, oferecendo uma gama de novos recursos: maior número de componentes (widgets), aparência e comportamento plugáveis, etc.

3.1 Usando JOptionPane

Caixas de diálogo são janelas que em geral são utilizadas para exibir mensagens importantes para o usuário de um aplicativo. O pacote Swing conta com uma classe chamada JOptionPane que permite que você exiba facilmente uma caixa de diálogo contendo informações ou para entrada de dados.

```
// JOptionPaneTeste.java

import javax.swing.*;

public class JOptionPaneTeste
{
    public static void main(String args[])
    {
        String primeiroNumero, segundoNumero;
        int primeiro, segundo, soma;

        primeiroNumero = JOptionPane.showInputDialog("Entre com o primeiro número");
        segundoNumero = JOptionPane.showInputDialog("Entre com o segundo número");

        primeiro = Integer.parseInt(primeiroNumero);
        segundo = Integer.parseInt(segundoNumero);

        soma = primeiro + segundo;
```

```
        JOptionPane.showMessageDialog(null, "A soma é " + soma, "Resultado",
            JOptionPane.PLAIN_MESSAGE);

        System.exit(0);
    }
}
```

Tipo de diálogo de mensagem	*Descrição*
JOptionPane.ERROR_MESSAGE	Exibe uma caixa de diálogo que indica um erro para o usuário do aplicativo.
JOptionPane.INFORMATION_MESSAGE	Exibe uma caixa de diálogo com uma mensagem informacional para o usuário do aplicativo.
JOptionPane.WARNING_MESSAGE	Exibe um diálogo que adverte o usuário do aplicativo de um problema potencial.
JOptionPane.QUESTION_MESSAGE	Exibe um diálogo que impõe uma pergunta para o usuário do aplicativo. Isso normalmente requer uma resposta como clicar em botão Yes ou No.
JOptionPane.PLAIN_MESSAGE	Exibe um diálogo que simplesmente contém uma mensagem sem ícone.

3.2 Criando uma Janela

```
// PrimeiraJanela.java

import javax.swing.*;

class Janela extends JFrame
{
    public Janela()
    {
        setTitle("Primeira Janela");
        setSize(400, 200);
    }
}

public class PrimeiraJanela
{
    public static void main(String arguments[])
    {
        Janela j = new Janela();
        j.show();
```

```
    }
}
```

3.3 Adicionando Componentes e Manipulando Eventos

```
// UsandoComponentes.java

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

class Janela extends JFrame
{
    JTextField tfCampo1, tfCampo2, tfResultado;

    public Janela()
    {
        setTitle("Usando Componentes e Manipulando Eventos");

        JLabel lValor1 = new JLabel("Primeiro Valor");
        JLabel lValor2 = new JLabel("Segundo valor");
        JButton bSomar = new JButton("Calcular");
        tfCampo1 = new JTextField("0");
        tfCampo2 = new JTextField("0");
        tfResultado = new JTextField();

        tfResultado.setEditable(false);
        bSomar.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e)
            {
                int v1 = Integer.parseInt(tfCampo1.getText());
                int v2 = Integer.parseInt(tfCampo2.getText());
                tfResultado.setText(String.valueOf(v1 + v2));
            }
        });

        Container c = getContentPane();
        c.setLayout(new GridLayout(3, 2));
        c.add(lValor1);
        c.add(tfCampo1);
        c.add(lValor2);
        c.add(tfCampo2);
        c.add(bSomar);
        c.add(tfResultado);
        pack();
```

```
    }
}

public class UsandoComponentes
{
    public static void main(String arguments[])
    {
        Janela j = new Janela();
        j.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e)
            {
                System.exit(0);
            }
        });
        j.show();
    }
}
```

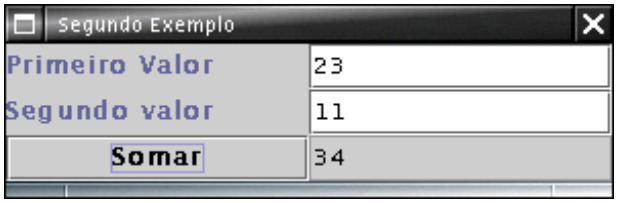

Figura 3.1: Screenshot do segundo exemplo

3.4 Trabalhando com Menus

```
// MenuDemo.java

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

class Janela extends JFrame
{
    JTextArea texto;

    public Janela()
    {
        setTitle("Visualizador de Arquivos");
        JMenuBar bar = new JMenuBar();
        setJMenuBar(bar);
```

```
        JMenu menuArquivo = new JMenu("Arquivo");
        menuArquivo.setMnemonic('A');
        JMenu menuEstilo = new JMenu("Estilo");
        menuEstilo.setMnemonic('E');
        JMenu menuAjuda = new JMenu("Ajuda");
        menuAjuda.setMnemonic('U');
        bar.add(menuArquivo);
        bar.add(menuEstilo);
        bar.add(menuAjuda);
        JMenuItem itemAbrir = new JMenuItem("Abrir");
        itemAbrir.setMnemonic('A');
        itemAbrir.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e)
            {
                JFileChooser fileChooser = new JFileChooser();
                int result = fileChooser.showSaveDialog(Janela.this);
            }
        });
        menuArquivo.add(itemAbrir);
        menuArquivo.addSeparator();
        menuArquivo.add(new JMenuItem("Sair"));
        menuEstilo.add(new JMenuItem("Copiar"));
        menuEstilo.add(new JMenuItem("Colar"));
        menuEstilo.add(new JMenuItem("Recortar"));
        JMenuItem itemSobre = new JMenuItem("Sobre...");
        itemSobre.setMnemonic('S');
        itemSobre.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e)
            {
                JOptionPane.showMessageDialog(null, "Mini Visualizador de Arquivos");
            }
        });
        menuAjuda.add(itemSobre);
        setSize(400, 200);
        texto = new JTextArea();
        getContentPane().add(texto);
    }
}

public class MenuDemo
{
    public static void main(String args[])
    {
        Janela j = new Janela();
        j.show();
```

```
    }
}
```

3.5 Selecionando a Aparência e Comportamento

```
// MudaAparencia.java

import javax.swing.*;
import java.awt.*;

class Janela extends JFrame
{
    public Janela()
    {
        Container c = getContentPane();
        c.setLayout(new GridLayout(2, 2));
        c.add(new JButton("Teste"));
        c.add(new JTextField());
        c.add(new JComboBox());
        c.add(new JRadioButton("Teste"));
        pack();
    }
}

public class MudaAparencia
{
    public static void main(String args[])
    {
        try
        {
            UIManager.setLookAndFeel("com.sun.java.swing.plaf.motif.MotifLookAndFeel");
        } catch (Exception e) { }

        Janela j = new Janela();
        j.show();
    }
}
```

Argumento para setLookAndFeel	*Descrição*
`UIManager.getCrossPlatformLookAndFeelClassName()`	Retorna a string do look-and-feel que é garantido funcionar: o Java Look-and-Feel.
`UIManager.getSystemLookAndFeelClassName()`	Especifica o look-and-feel para a plataforma corrente. Em plataformas Win32, seta o look-and-feel do Windows, em plataformas Mac OS, o Mac OS Look-and-feel.
"`javax.swing.plaf.metal.MetalLookAndFeel`"	Especifica o Java look-and-feel (nome de código era Metal).
"`javax.swing.plaf.windows.WindowsLookAndFeel`"	Somente pode ser usado em plataformas Win32.
"`com.sun.java.swing.plaf.motif.MotifLookAndFeel`"	Especifica o look-and-feel CDE/Motif. Pode ser usado em qualquer plataforma.
"`javax.swing.plaf.mac.MacLookAndFeel`"	Especifica o look-and-feel do Mac OS, somente pode ser usado em plataformas rodando Mac OS.

Capítulo 4

Desenvolvimento de Applets

Applets são pequenos programas que podem ser embutidos em uma página web. Quando Java foi lançada, a possibilidade do desenvolvimento de applets foi o carro chefe do marketing da linguagem. As applets trouxeram um grau de interatividade que até então não era possível de ser encontrada na web.

```
// HelloWorldApplet.java

import java.applet.Applet;
import java.awt.Graphics;

public class HelloWorldApplet extends Applet
{
    public void paint(Graphics g)
    {
        g.drawString("Hello world!", 50, 25);
    }
}
```

Para que o programa acima possa ser executado, ele precisa ser embutido em uma página html, como a mostrada a seguir:

```
<HTML>
  <HEAD>
    <TITLE> Hello World </TITLE>
  </HEAD>

  <BODY>
    <applet code="HelloWorldApplet.class" width="150" height="50">
    </applet>
  </BODY>
</HTML>
```

4.1 A Tag HTML Applet

Applets são embutidas em páginas web através das tags ⟨APPLET⟩ e ⟨/APPLET⟩. A tag ⟨APPLET⟩ é similar à tag ⟨IMG⟩. Assim como a tag ⟨IMG⟩, ⟨APPLET⟩ referencia um arquivo que não faz parte da página HTML no qual ele é embutido. A tag ⟨IMG⟩ faz isso através do atributo SRC, ⟨APPLET⟩ usa para isso o atribudo CODE. O atributo CODE diz ao browser aonde encontrar um arquivo compilado .class.

4.2 Segurança

A possibilidade de surfar na Internet, e executar applets em sites não confiáveis poderia ser um seríssimo problema de segurança se não fossem os mecanismos de proteção e segurança da linguagem Java. Na verdade, tanto applets como aplicativos Java são muito mais seguros do que código escrito em linguagens tradicionais. Programas em Java são muito menos sucetíveis a erros comuns de programação envolvendo acesso a memória como em C. Applets, além disso, implementam restrições de segurança adicionais para proteger usuários de código malicioso.

O que uma applet pode fazer:

- Desenhar imagens;
- Criar uma nova janela;
- Efeitos sonoros;
- Receber entrada do usuário a partir do teclado ou mouse;
- Fazer uma conexão de rede com o servidor a partir do qual a applet foi baixada.

Por outro lado, uma applet não tem permissão para:

- Escrever dados em qualquer unidade local;
- Ler dados de unidades locais sem permissão do usuário;
- Apagar arquivos;
- Ler ou escrever blocos arbitrários de memória. Todo acesso a memória é estritamente controlado.
- Fazer uma conexão de rede com exceção da máquina que a applet foi baixada;
- Introduzir um vírus ou um cavalo de tróia no sistema;

Capítulo 5

Manipulação de Arquivos

5.1 Lendo Arquivos

```
// LeArquivo.java

import java.io.*;

public class LeArquivo
{
    public static void main(String args[])
    {
        try
        {
            FileInputStream file = new FileInputStream(args[0]);
            int cont = 0;
            int input;
            while (true)
            {
                input = file.read();
                if (input == -1)
                    break;
                System.out.print((char)input);
                cont++;
            }
            file.close();
            System.out.println("\nBytes lidos: " + cont);
        }
        catch (IOException e)
        {
            System.out.println("Erro" + e.toString());
        }
    }
}
```

5.2 Gravando Arquivos

```
// GravaArquivo.java

import java.io.*;

public class GravaArquivo
{
    public static void main(String args[])
    {
        int[] data = { 71, 73, 70, 56, 57, 97, 15, 0, 15, 0,
                       128, 0, 0, 255, 255, 255, 0, 0, 0, 44, 0, 0, 0,
                       0, 15, 0, 15, 0, 0, 2, 33, 132, 127, 161, 200,
                       185, 205, 84, 128, 241, 81, 35, 175, 155, 26,
                       228, 254, 105, 33, 102, 121, 165, 201, 145, 169,
                       154, 142, 172, 116, 162, 240, 90, 197, 5, 0, 59 };
        try
        {
            FileOutputStream file = new FileOutputStream("pix.gif");
            for (int i = 0; i < data.length; i++)
            {
                file.write(data[i]);
            }
            file.close();
        }
        catch (IOException e)
        {
            System.out.println("Erro:" + e.toString());
        }
    }
}
```

5.3 Listando o Conteúdo de um Diretório

```
// Dir.java

import java.io.*;

public class Dir
{
    public static void main(String args[])
    {
        File dirAtual = new File(".");
        String listaArquivos[] = dirAtual.list();
```

```
        for (int i = 0; i < listaArquivos.length; i++)
        {
            File arquivoAtual = new File(listaArquivos[i]);
            System.out.println(listaArquivos[i] + " [" +
                               arquivoAtual.length() + " bytes]");
        }
    }
}
```

5.4 Obtendo Entrada do Console

```
// LendoConsole.java

import java.io.*;

public class LendoConsole
{
    public static void main(String args[])
    {
        BufferedReader br = new BufferedReader (new InputStreamReader(System.in));
        int v1, v2, soma;
        try
        {
            System.out.print("Digite o primeiro valor: ");
            v1 = Integer.parseInt(br.readLine());
            System.out.print("Digite o segundo valor: ");
            v2 = Integer.parseInt(br.readLine());
            soma = v1 + v2;
            System.out.println("Soma dos valores: " + soma);
        }
        catch (IOException e)
        {
            System.out.println("Erro de entrada: " + e);
        }
    }
}
```

Capítulo 6

Programação Concorrente: Multithreading

Java é a única entre as linguagens de programação de uso geral e popular que tem recursos nativos de programação concorrente. Isso significa que um único programa pode ter múltiplas linhas de execução rodando de forma simultânea.

6.1 Criando uma Linha de Execução

```
// ThreadDemo1.java

class Linha extends Thread
{
    public void run()
    {
        for (int i = 0; i < 6; i++)
        {
            System.out.println("Teste" + i);
            try { sleep(10); }
            catch (InterruptedException e)
            { System.err.println(e.toString()); }
        }
    }
}

public class ThreadDemo1
{
    public static void main(String args[])
    {
        Linha l1 = new Linha();
        Linha l2 = new Linha();
```

```
        l1.start();
        l2.start();
    }
}
```

6.2 Sincronizando Dados Compartilhados entre Linhas

Compartilhar dados é uma armadilha do sistema de linhas de execução. Se duas linhas têm acesso compartilhado à mesma variável, isto é, se ambas têm acesso à leitura e gravação, podem acontecer resultados estranhos caso o seu programa não tome cuidados especiais para coordenar o acesso das linhas aos dados. Para sincronizar dados compartilhados, cria-se um método para acessar os dados compartilhados, e utiliza-se a palavra-chave `synchronized` na declaração do método. Então Java restringe o acesso ao método para que somente uma linha de cada vez possa acessá-lo.

```
// DadosCompartilhados.java

class Conta
{
    static int balanco = 10000000;
    static int despesa = 0;

    static public synchronized void transacao(int montante)
    {
        if (montante <= balanco)
        {
            balanco = balanco - montante;
            for (float a = 0; a < 100000; a++);
            despesa = despesa + montante;
            System.out.print(balanco + despesa + " ");
        }
        else
        {
            System.out.println("Saldo inferior ao necessário");
        }
    }
}

class minhaThread extends Thread
{
    public void run()
    {
        for (int i = 0; i < 1000; i++)
            Conta.transacao((int)(Math.random() * 100));
    }
}
```

```
public class DadosCompartilhados
{
    public static void main(String args[])
    {
        new minhaThread().start();
        new minhaThread().start();
        new minhaThread().start();
        new minhaThread().start();
        new minhaThread().start();
        new minhaThread().start();
    }
}
```

Capítulo 7

Programação em Rede

Uma das metas do projeto Java era oferecer suporte ao desenvolvimento de programas complexos que fossem pequenos, livros de erros e fáceis de entender. Em outras linguagens de programação, executar a tarefa de rede mais simples pode exigir páginas de código. Em Java muitas destas tarefas podem ser reduzidas a uma única e simples intrução.

7.1 Lendo um Arquivo através da Internet

```
// URLTeste.java

import java.io.*;
import java.net.*;

public class URLTeste
{
    public static void main(String args[])
    {
        try
        {
            URL url = new URL(args[0]);
            DataInputStream dis = new DataInputStream(url.openStream());
            String line = dis.readLine();
            while (line != null)
            {
                System.out.println(line);
                line = dis.readLine();
            }
        }
        catch (IOException e)
        {
            System.out.println("Erro: " + e.getMessage());
        }
```

```
    }
}
```

7.2 Trabalhando com Sockets

```
// CriandoSocket.java

import java.net.*;
import java.io.*;

public class CriandoSocket
{
    public static void main(String args[])
    {
        String host;
        int porta;
        host = args[0];
        porta = Integer.parseInt(args[1]);
        try
        {
            Socket s = new Socket(host, porta);
            DataInputStream dis = new DataInputStream(s.getInputStream());
            PrintStream ps = new PrintStream(s.getOutputStream());
            ps.println("teste\n");
            while (true)
            {
                String msg = dis.readLine();
                if (msg == null)
                    break;
                System.out.println(msg);
            }
        }
        catch (IOException e)
        {
            System.out.println("Erro: " + e );
        }
    }
}
```

7.3 Criando um ServerSocket

7.4 Cliente Multithreaded

```
// ClienteMultiThread.java

import java.net.*;
import java.io.*;

public class ClienteMultiThread
{
    public static void main(String args[])
    {
        String host;
        int porta;
        InputStream istream = System.in;
        DataInputStream distream = new DataInputStream(istream);
        host = args[0];
        porta = Integer.parseInt(args[1]);
        try
        {
            Socket s = new Socket(host, porta);
            PrintStream ps = new PrintStream(s.getOutputStream());
            new MostraDados(s).start();
            while (true)
            {
                ps.println(distream.readLine());
            }
        }
        catch (IOException e)
        {
            System.out.println("Erro: " + e );
        }
    }
}

class MostraDados extends Thread
{
    Socket s;

    public MostraDados(Socket s)
    {
        this.s = s;
    }

    public void run()
    {
```

```
        try
        {
            DataInputStream dis = new DataInputStream(s.getInputStream());
            while (true)
            {
                String msg = dis.readLine();
                if (msg == null)
                    break;
                System.out.println(msg);
            }
        }
        catch (IOException e)
        {
            System.out.println("Erro: " + e);
        }
    }
}
```

Capítulo 8

Conectividade com Banco de Dados: JDBC

JDBC significa Java Database Connectivity (Conectividade de Banco de Dados em Java), e é uma interface de programação de aplicações (API) desenvolvida pela Sun que permite a um programa Java acessar um SGBD (Sistema Gerenciador de Banco de Dados). A JDBC facilita a conexão com um banco de dados, o envio de instruções SQL e a obtenção de resultados.

8.1 Fazendo uma Consulta

```
// DatabaseTeste.java

import java.net.*;
import java.sql.*;

public class DatabaseTeste
{
    public static void main(String args[])
    {
        java.sql.Connection conn = null;
        String dburl = "jdbc:mysql://localhost/pds?user=root&password=";
        String sql_str = "SELECT * FROM usuarios";
        try
        {
            Class.forName("org.gjt.mm.mysql.Driver").newInstance();
            conn = DriverManager.getConnection(dburl);
            Statement sq_stmt = conn.createStatement();
            ResultSet rs = sq_stmt.executeQuery(sql_str);

            while (rs.next())
            {
                int id = rs.getInt("id");
```

```
                String nome = rs.getString("nome");
                String endereco = rs.getString("endereco");

                System.out.println("ID:" + id);
                System.out.println("Nome: " + nome);
                System.out.println("Endereco: " + endereco + "\n");
            }
        }
        catch (Exception e)
        {
            System.out.println("Erro: " + e);
        }

    }
}
```

Bibliografia

[1] Java 1001 Dicas de Programação. Mark C. Chan, Steven W. Griffith, Anthony F. Iasi. Makron Books. 1999.

[2] Aprenda Java 1.2 em 21 Dias. Laura Lemay, Rogers Cadenhead. Ed. Campus. 1999.

[3] Java 2: Como Programar, H. M. Deitel, P. J. Deitel. Bookman. 3ª edição. 2001.

[4] Core Java Volume I - Fundamentos. Cay S. Horstmann, Gary Cornell. Makron Books. 2001.