

Arquitecturas Paralelas I
Computação Paralela em Larga Escala

LESI - 4º Ano

Conceitos de Programação com Componentes

João Luís Ferreira Sobral
Departamento de Informática
Universidade do Minho



Outubro 2003

Introdução à Programação com Componentes

● Programação com componentes

- A tecnologia de programação com componentes é especialmente importante no desenvolvimento de aplicações que são distribuídas por várias máquinas, uma vez que a distribuição dos componentes pelas máquinas é mais fácil do que se forem utilizadas outras tecnologias
- A programação com componentes pretende aumentar o grau de reutilização do software, no entanto, o seu sucesso apenas foi significativo no desenvolvimento das interfaces gráficas das aplicações.
- A ideia base de programação com componentes é possibilitar o desenvolvimento de aplicações apenas através da colagem de peças de software (componentes) já existentes.

● O que são componentes?

- Um componente pode ser um bloco de código, uma função, um objecto ou uma aplicação, ao qual foi adicionada lógica para implementar uma interface.
- Um componente é uma peça de software (i.é., executável) que implementa uma determinada funcionalidade (interface) e que pode ser configurada (alterando as propriedades do componente) em função do contexto onde é utilizada.
- Cada componente implementa um conjunto de serviços (i.é., interfaces que implementa) e pode basear-se em serviços fornecidos por outros componentes (i.é., pode utilizar interfaces implementados outros componentes)
- Os componentes são frequentemente utilizados em ambientes de programação visual, os quais devem ser capazes de determinar os interfaces implementados por cada componente e os interfaces que cada componente utiliza

● Como são construídas as aplicações com componentes

- A programação por componentes permite desenvolver aplicações simplesmente através da colagem de vários componentes, possivelmente de diferentes fabricantes. A “colagem” consiste em identificar as interligações entre componentes as propriedades de cada componente
- Uma forma de desenvolver aplicações baseadas em componentes é utilizando um ambiente visual de programação que permite, através de uma interface gráfica, fazer a configuração da aplicação.

Introdução aos Componentes do GUI em Java

● Historial do GUI (*Graphical User Interface*) de Java

- Java 1.0 incluía a versão inicial do *Abstract Window Toolkit* (AWT), que continha várias limitações, nomeadamente uma GUI medíocre em todos os sistemas e não adoptava uma filosofia orientada por objectos
- Java 1.1 melhorou o AWT original, com uma aproximação mais “limpa” e orientada ao objecto, juntamente com a adição de JavaBeans, um modelo de programação por componentes orientado para a criação de ambientes de programação visuais
- Java 2 terminou a transformação com a substituição de todos os elementos da GUI por um novo conjunto, designado por *Swing* (classes `javax.swing`), presente nas *Java Foundation Classes* (JFC)

● Introdução aos *Applets*

- São pequenos programas em Java que executam num *browser* Web
- Por serem executados no *browser* dos clientes os *applets* devem ser seguros, razão que leva a que sejam limitadas as operações realizadas, ex: não podem aceder ao disco local, todas as classes necessárias ao *applet* devem ser empacotadas num ficheiro JAR (*Java Archive*)
- Não é necessário instalar os *applets* e estes são verdadeiramente portáteis e seguros.
- Os *applets* devem estender a classe `JApplet`, devendo redefinir os seguintes métodos:

```
void init();           // invocado quando é iniciado
void destroy();       // invocado quando é terminado
void start();         // invocado quando recebe a atenção
void stop();          // invocado quando perde a atenção
```

os métodos `start()` e `stop()` também são invocados, em `init()` e `destroy()`.

- Exemplo:

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

public class Applet1 extends JApplet {
    public void init() {
        getContentPane().add(new JLabel("App1"));
    }
}
```

Componentes do GUI em Java

● Introdução aos *Applets* (continuação)

- O applet anterior pode ser incorporado numa página www através de seguinte código:

```
<applet code=Applet1.class width=100 height=50>
</applet>
```

- O utilitário *appletviewer* do JDK permite visualizar um applet
- É possível construir uma classe que é um applet e que também pode ser executada como uma aplicação normal, a partir da linha de comando. Para tal basta incluir a applet numa JFrame, adicionando o seguinte main() à definição do applet:

```
public static void main(String[] str) {
    JApplet ap = new Applet1();
    JFrame frame = new JFrame("Appx");
    frame.getContentPane().add(ap);
    frame.setSize(100,50);
    ap.init();    // simula o focus do browser
    ap.start();
    frame.setVisible(true); // torna visível
}
```

● Componentes da interface *Swing*

- O método *getContentPane()* permite obter o *Container* actual, o qual utiliza o padrão composto, onde *add(Component c)* adiciona um componente ao composto e o método *setLayout(LayoutManager mgr)* indica a estratégia de posicionamento dos elementos (é um padrão do tipo estratégia). *JButton(String)* é um componente que implementa um botão.

```
public class Applet1 extends JApplet {
    public void init() {
        Container ct = getContentPane();
        // altera a estratégia de colocação dos elem
        ct.setLayout(new FlowLayout());
        ct.add(new JButton("Botão1"));
        ct.add(new JButton("Botão2"));
    }
}
```

Componentes do GUI em Java

● Componentes da interface *Swing* (continuação)

- As respostas aos vários eventos gerados pelos componentes são implementadas por padrões do tipo observador. Por exemplo, para responder aos eventos gerados pelos *JButton* o observador deve implementar a interface *ActionListener*, que contém apenas o método *void actionPerformed(ActionEvent e)*. O observador deve registrar-se junto do *JButton* (o assunto, ou o observado), através do método *addActionListener(ActionListener a)*. O seguinte exemplo mostra o nome do botão premido numa caixa de texto *JTextField*. O método *setText(String)* é utilizado para alterar o texto da caixa.

```
public class Applet1 extends JApplet {
    JButton b1 = new JButton("Botão1");
    JButton b2 = new JButton("Botão2");
    JTextField tx = new JTextField(10);
    class BListener implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            String s = ((JButton) e.getSource()).getText();
            tx.setText(s);
        }
    }
    BListener bt = new BListener();
    public void init() {
        b1.addActionListener(bt);
        b2.addActionListener(bt);
        Container ct = getContentPane();
        ct.setLayout(new FlowLayout());
        ct.add(b1);
        ct.add(b2);
        ct.add(tx);
    }
}
```

- A classe *BListener* só é utilizada só neste *applet*, pelo que pode ser transformada numa classe anónima interior:

```
ActionListener bt = new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        String s = ( (JButton) e.getSource()).getText();
        tx.setText(s);
    }
};
```

- Como alternativa à classe *JTextField*, pode ser utilizada uma área de texto (*JTextArea*), utilizando *append(String)* para adicionar texto. *JScrollPane()* adiciona uma barra de deslocamento e é um exemplo do padrão decorador:

```
...
JTextArea tx = new JTextArea(10,20);
...
tx.append(s);
...
ct.add(new JScrollPane(tx));
```

Componentes do GUI em Java

● Componentes da interface *Swing* (continuação)

- O posicionamento e a dimensão dos componentes dentro de um contentor (classes *JApplet*, *JFrame*, *JWindow*, *JDialog*) é determinada por uma estratégia definida por um *LayoutManager*. Algumas estratégias pré-definidas são:
 - **BorderLayout** – o contentor é dividido em 5 regiões, NORTH, SOUTH, EAST, WEST, CENTER. Na operação de *add* especifica-se a posição do elemento (o elemento central, utilizado por defeito, estende-se até às margens):
`add(BorderLayout.NORTH, component);`
 - **FlowLayout** – os elementos são dispostos sucessivamente numa linha, da esquerda para a direita, até se encher essa linha. Então, é preenchida a linha seguinte.
 - **GridLayout(colunas,linhas)** – os elementos são dispostos numa grelha de colunas x linhas, da esquerda para a direita e de cima para baixo.
- A interacção entre os vários elementos do GUI é efectuada através de eventos, utilizando o padrão observador. Cada elemento GUI funciona como um Assunto, no qual observadores se podem registar através de um método do tipo *addXXXListener()*, onde XXX representa o tipo de evento. Cada observador deve também implementar uma interface específica a cada evento. A tabela seguinte apresenta alguns dos eventos e das interfaces que devem ser implementadas pelos observadores:

Evento	Interface do Observador	Métodos na Interface
ActionEvent	ActionListener	actionPerformed()
FocusEvent	FocusListener	focusGained() focusLost()
KeyEvent	KeyListener	keyPressed() keyReleased() keyTyped()
MouseEvent	MouseListener	mouseClicked() mouseEntered() mouseExited() mousePressed() mouseReleased()
WindowEvent	WindowListener	windowOpened() windowClosed() ...

- A generalidade das classes anteriores fornece um adaptador (classe *xxxAdapter*) que implementa a interface, ignorando todos os eventos. As classes da aplicação podem herdar dessa classe para tratar apenas alguns eventos.

Introdução aos Componentes JavaBeans

● Programação com componentes em Java (JavaBeans)

- JavaBeans é uma arquitectura de componentes de software em Java, surgindo após o sucesso do Microsoft Visual Basic e do Borland Delphi. Esta arquitectura proporciona a interoperabilidade entre componentes e ferramentas de vários fabricantes, para além da independência da plataforma, assegurada para própria linguagem Java
- Em JavaBeans, um componente é constituído por uma classe Java, sendo apenas necessárias pequenas alterações para tornar uma classe Java num *Bean*, o que torna o desenvolvimento de componentes extremamente simples em Java.
- Na programação por componentes é necessária uma ferramenta que permita a configuração e a ligação de um conjunto de *beans*, formando uma aplicação. Em Java existe uma ferramenta visual de construção de aplicações através de componentes, muito simples, o *Bean Developer Kit (BDK)*, fornecida pela SUN e que é de utilização livre.
- A arquitectura JavaBeans baseia-se em propriedades que permitem controlar o comportamento do objecto, podendo ser variáveis de estado; em métodos que são invocados para executar código no componente e em eventos que são notificações geradas quando o objecto altera o estado
- As ferramentas visuais de programação com JavaBeans permitem editar graficamente as propriedades dos componentes e visualmente associar os eventos gerados por um componente a invocações de métodos noutros componentes. Esta última operação é implementada através da utilização do padrão Observador e de Adaptadores para tornar as interfaces compatíveis.
- As ferramentas de desenvolvimento de aplicações por componentes baseiam-se num mecanismo designado por introspecção, que permite determinar as propriedades, eventos e métodos de cada componente.

● Componentes empresariais em Java (Enterprise JavaBeans)

- Extensão aos componentes Java para funcionar em aplicações empresariais distribuídas, mais vocacionados para o desenvolvimento de aplicações com várias camadas
- Suportam a distribuição de componentes, transações distribuídas, replicação de componentes, autorização e autenticação de acessos.
- Baseado no conceito de contentor, onde o componente é executado e que fornece os vários serviços. A especificação dos componentes implica a compatibilidade entre componentes e contentores de vários fabricantes.

Componentes JavaBeans

• Como desenvolver uma classe *Bean* em Java:

- Para cada propriedade com o nome *xxx*, criar dois métodos: *xxx getXxx()* e *setXxx(xxx)*; (para uma propriedade booleana pode ser utilizado *is* em vez de *get*);
- Os métodos normais da classe, que são declarados como públicos, serão automaticamente os métodos do *bean*; cada componente deve ter um construtor sem parâmetros, utilizado pelas ferramentas para instanciar o componente;
- os eventos são tratados (i.e. gerados) pelo *bean* utilizando o padrão observador, para tal o *bean* deve fornecer métodos para gerir os observadores. Assim, para o evento *xxxEvent* devem ser fornecidos os métodos *addXxxListener(XxxListener)* e *removeXxxListener(XxxListener)*.

• Exemplo 1 (sem interface gráfico):

- Um *Frog bean* que escreve “croak” sempre que recebe um evento e possui uma propriedade que permite silenciar o *Frog*:

```
package frogbean;

import java.awt.*;
import java.awt.event.*;

public class Frog {
    private int croaks;
    private boolean quiet;
    public boolean isQuiet() { return quiet; }
    public void setQuiet(boolean q) { quiet = q; }
    public int getCroaks() { return croaks; }
    public void setCroaks(int c) { croaks = c; }
    public void croak() {
        if (quiet!=true) {
            croaks++;
            System.out.println("Croak!");
        }
    }
}
```

- Para compilar o *bean* é necessário criar uma directoria com o nome da *package* (frogbean, neste caso), onde se coloca a classe e um ficheiro *Manifest Temple*, (frogbean.mft) que descreve o *bean*:

```
Name: frogbean/Frog.class
Java-Bean: True
```

- finalmente, o *bean* deve ser colocado num ficheiro JAR (Java Archive) e colocado na directoria jar do bdk:

```
jar cfm c:\jdk1.1\jars\frogbean.jar frogbean\*.mft frogbean\*.class
```

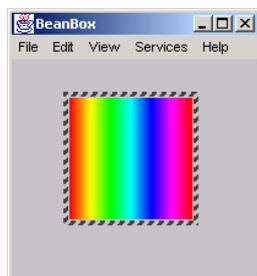
Componentes JavaBeans

● Exemplo 2 (com interface gráfico):

- Um *bean* que desenha as cores de espectro:

```
import java.awt.*;

public class Spectrum extends Canvas {
    private boolean vertical;
    public Spectrum() {
        vertical = true;
        setSize(100, 100);
    }
    public boolean getVertical() {
        return vertical;
    }
    public void setVertical(boolean vertical) {
        this.vertical = vertical;
        repaint();
    }
    public void paint(Graphics g) {
        float saturation = 1.0f;
        float brightness = 1.0f;
        Dimension d = getSize();
        if(vertical) {
            for(int y = 0; y < d.height; y++) {
                float hue = (float)y/(d.height - 1);
                g.setColor(Color.getHSBColor(hue, saturation, brightness));
                g.drawLine(0, y, d.width - 1, y);
            }
        } else {
            for(int x = 0; x < d.width; x++) {
                float hue = (float)x/(d.width - 1);
                g.setColor(Color.getHSBColor(hue, saturation, brightness));
                g.drawLine(x, 0, x, d.height - 1);
            }
        }
    }
}
```



- O método *paint* é invocado sempre que é necessário desenhar o componente.
- Sempre que a aplicação necessita de redesenhar deve invocar a função *repaint()* que automaticamente invoca o método *paint()*

Eventos em JavaBeans

● Eventos

- Um *bean* pode herdar de um componente do *swing* já existente, herdando também a capacidade de gerar eventos. Assim poderá gerar os eventos implementados nesse componente:

```
package actionevents;
import java.awt.*;

public class ActionSource1 extends Button {

    public ActionSource1() {
        super("ActionSource1");
    }
}
```

```
package actionevents;
import java.awt.*;
import javax.swing.*;

public class ActionSource2 extends List {

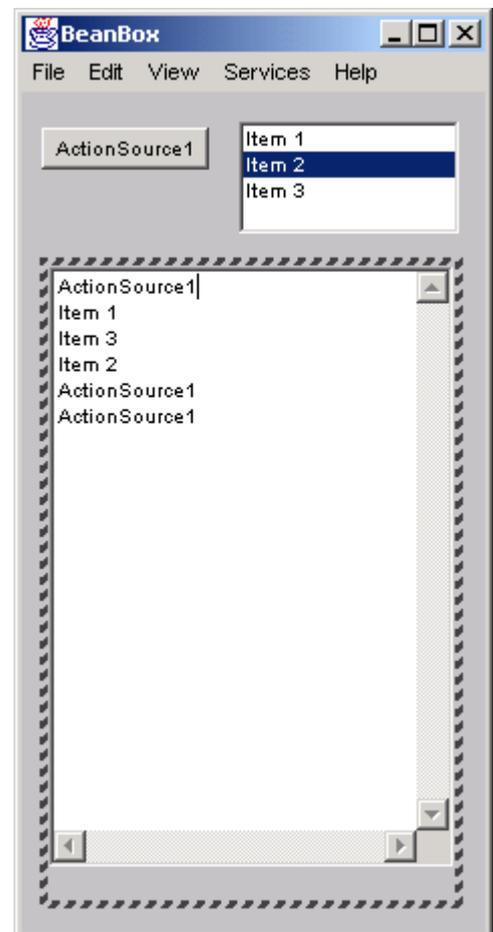
    public ActionSource2() {
        add("Item 1");
        add("Item 2");
        add("Item 3");
    }
}
```

```
package actionevents;
import java.awt.*;
import java.awt.event.*;

public class ActionReceiver extends Panel
    implements ActionListener {
    private TextArea ta;

    public ActionReceiver() {
        setLayout(null);
        ta = new TextArea();
        ta.setBounds(0, 0, 200, 300);
        add(ta);
        setSize(200, 300);
    }

    public void actionPerformed(ActionEvent ae) {
        String ac = ae.getActionCommand();
        ta.append(ac + "\n");
    }
}
```



Eventos em JavaBeans

● Eventos personalizados

- É possível definir novos eventos em Java. Para tal é necessário criar um objecto que define o novo evento, estendendo a classe *EventObject*, criar um novo interface para os observadores e fornecer as primitivas para gerir os observadores no *bean* que gera esse eventos.
- O exemplo seguinte mostra dois *beans*, o primeiro permite seleccionar uma cor, gerando um *ColorEvent*, o segundo utiliza a cor definida pelo primeiro para preencher uma região.

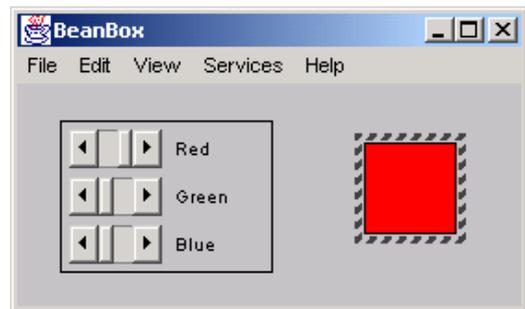
```
// definição do evento
public class ColorEvent
    extends EventObject {
    private Color color;

    public ColorEvent(Object source, Color color) {
        super(source);
        this.color = color;
    }
    public Color getColor() { return color; }
}

// definição da interface dos observadores
public interface ColorListener extends EventListener {
    public void changeColor(ColorEvent ce);
}

// definição do printor
public class Painter extends Canvas implements ColorListener {
    private Color color;

    public Painter() {
        color = Color.white;
        setSize(50, 50);
    }
    public void paint(Graphics g) {
        Dimension d = getSize();
        int w = d.width;    int h = d.height;
        g.setColor(color);
        g.fillRect(0, 0, w - 1, h - 1);
        g.setColor(Color.black);
        g.drawRect(0, 0, w - 1, h - 1);
    }
    public void changeColor(ColorEvent ce) {
        this.color = ce.getColor();
        repaint();
    }
}
}
```



Eventos em JavaBeans

● Eventos personalizados (continuação)

```
public class Selector extends Panel implements AdjustmentListener {
    private Color color;
    private Vector listeners = new Vector();
    private Scrollbar rScrollbar, gScrollbar, bScrollbar;

    public Selector() {
        setLayout(new GridLayout(3, 2, 5, 5)); // Initialize GUI elements
        rScrollbar = new Scrollbar(Scrollbar.HORIZONTAL, 255, 10, 0, 265);
        add(rScrollbar);
        rScrollbar.addAdjustmentListener(this);
        Label rLabel = new Label("Red", Label.LEFT);
        add(rLabel);
        gScrollbar = new Scrollbar(Scrollbar.HORIZONTAL, 255, 10, 0, 265);
        add(gScrollbar);
        gScrollbar.addAdjustmentListener(this);
        Label gLabel = new Label("Green", Label.LEFT);
        add(gLabel);
        bScrollbar = new Scrollbar(Scrollbar.HORIZONTAL, 255, 10, 0, 265);
        add(bScrollbar);
        bScrollbar.addAdjustmentListener(this);
        Label bLabel = new Label("Blue", Label.LEFT);
        add(bLabel);
    }

    public Insets getInsets() { return new Insets(5, 5, 5, 5); }
    public Color getColor() { return color; }
    public void setColor(Color color) { this.color = color; }
    public void paint(Graphics g) {
        Dimension d = getSize();
        g.drawRect(0, 0, d.width - 1, d.height - 1);
    }

    public void adjustmentValueChanged(AdjustmentEvent ae) {
        Scrollbar source = (Scrollbar)ae.getSource();
        int value = ae.getValue();
        source.setValue(value);
        int r = rScrollbar.getValue();
        int g = gScrollbar.getValue();
        int b = bScrollbar.getValue();
        color = new Color(r, g, b);
        fireColorEvent(new ColorEvent(this, color));
    }

    public void addColorListener(ColorListener cl) {
        listeners.addElement(cl);
    }

    public void removeColorListener(ColorListener cl) {
        listeners.removeElement(cl);
    }

    public void fireColorEvent(ColorEvent ce) {
        Vector v; // suporte a vários fios de execução
        synchronized(this) { v = (Vector)listeners.clone(); }
        for(int i = 0; i < v.size(); i++) {
            ColorListener cl = (ColorListener)v.elementAt(i);
            cl.changeColor(ce);
        }
    }
}
```

Serialização em JavaBeans

● Serialização e de-serialização

- Quando um *bean* é gravado em disco, o seu estado é guardado com base no mecanismo de serialização. Por esta razão os *beans* devem poder ser serializados, o que implica que todas as classes utilizadas pelo *bean* devem implementar o interface *Serializable*.
- Existem algumas classes que não são serializáveis, nomeadamente a classe *Thread*. Nestes casos é necessário marcar as variáveis de instância como *transient* e reimplementar as funções de escrita e leitura do *bean* em *streams*, através dos métodos *readObject()* e *writeObject()*.
- Exemplo de uma classe que implementa um relógio:

```
public class Clock extends Panel implements Runnable {
    private TextField tf;
    private transient Thread thread;

    public Clock() {
        tf = new TextField("", 6);
        add(tf);
        startThread();
    }

    private void startThread() {
        thread = new Thread(this);
        thread.start();
    }

    public void run() {
        try {
            SimpleDateFormat sdf;
            sdf = new SimpleDateFormat("HH:mm:ss");
            while(true) {
                Thread.sleep(1000);
                tf.setText(sdf.format(new Date()));
            }
        } catch(Exception ex) {    ex.printStackTrace();    }
    }

    private void readObject(ObjectInputStream ois)
    throws IOException, ClassNotFoundException {
        try {
            ois.defaultReadObject();
            startThread();
        }
        catch(Exception ex) {    ex.printStackTrace();    }
    }
}
```

Reflexão e Introspecção em JavaBeans

● Reflexão e introspecção

- Em Java é possível, através dos mecanismos de reflexão, determinar as características de uma classe durante a execução da aplicação. Tal é possível através de uma instância da classe *Class* que representa uma classe. Esta classe contém vários métodos que permitem obter as informações da classe:

Método	Descrição
static Class.forName(String className)	Devolve um objecto <i>Class</i> da classe <i>className</i>
Constructor[] getConstructors()	devolve uma lista dos construtores da classe
Field[] getFields()	devolve uma lista dos campos declarados
Method[] getMethods()	“ “ dos métodos declarados
String getName()	devolve o nome da classe

- O seguinte bloco de código mostra os campos de uma classe (*args[0]*)

```
try { Class c = Class.forName(args[0]);
    Field f[] = c.getFields();
    for(int i=0; i<f.length; i++)
        System.out.println(f[i].getName());
} catch(Exception e) { e.printStackTrace(); }
```

- O mecanismo de introspecção permite obter informação sobre as propriedades, eventos e métodos de um *bean*. Utilizando este mecanismo é possível a um *bean* controlar a informação que é visível nas ferramentas de construção. Para controlar as características visíveis de um componente deve-se desenvolver uma classe que estende *SimpleBeanInfo* e designada por *<<Classe>>BeanInfo*:

```
public class Frog2BeanInfo extends SimpleBeanInfo {
    public PropertyDescriptor[] getPropertyDescriptors() {
        try { PropertyDescriptor pd1, pd2;
            pd1 = new PropertyDescriptor("quiet",Frog2.class);
            pd2 = new PropertyDescriptor("croaks",Frog2.class);
            PropertyDescriptor pds[] = { pd1, pd2 };
            return pds;
        } catch(Exception ex) { }
        return null;
    }
    public EventSetDescriptor[] getEventSetDescriptors() {
        EventSetDescriptor esds[] = { };
        return esds;
    }
    public MethodDescriptor[] getMethodDescriptors() {
        try { // utiliza para construir o método croak
            Method[] m = (Frog2.class).getMethods(); // reflexão
            for(int i=0; i<m.length; i++) // procura na lista
                if ((m[i].getName()).equals("croak")) m1=m[i];
            ParameterDescriptor pds[] = new ParameterDescriptor[1];
            pds[0] = new ParameterDescriptor();
            MethodDescriptor mds[] = { new MethodDescriptor(m1, pds); };
            return mds;
        } catch(Exception ex) { ex.printStackTrace(); }
        return null;
    }
}
```

Aplicações JavaBeans

● Como distribuir uma aplicação desenvolvida com *JavaBeans*?

- As ferramentas de desenvolvimento por componentes geram código Java que cria as instâncias dos vários componentes, atribui os valores definidos às propriedades de cada componente e efectua a ligação entre componentes.
- Por exemplo, na aplicação do *Painter* e *Selector* poderia ser gerada a seguinte classe (a função *Beans.instantiate* permite criar uma instância do *Bean*):

```
package cselector;

import java.awt.*;
import java.awt.event.*;
import java.beans.*;

public class MainClass {

    public static void main(String[] arg) {
        new MainClass();
    }
    public MainClass() {
        Frame f = new Frame();
        f.setSize(200,130);
        f.addWindowListener(new MyWindowAdapter());
        try {
            Painter pp =
                (Painter) Beans.instantiate(null,"cselector.Painter");
            Selector ss =
                (Selector) Beans.instantiate(null,"cselector.Selector");
            ss.setColor(Color.black);
            ss.addColorListener(pp);
            f.setLayout(new FlowLayout());
            f.add(ss);
            f.add(pp);
        } catch(Exception e) { e.printStackTrace(); }
        f.setVisible(true);
    }
    class MyWindowAdapter extends WindowAdapter {
        public void windowClosing(WindowEvent e) {
            System.exit(0);
        }
    }
}
```

- Uma forma alternativa consiste em colocar as várias classes num JAR e utilizar um *class loader* para carregar as classes do JAR.

Introdução ao desenvolvimento de aplicações com GUI

● Exercícios (Introdução aos applets e swing)

1.1 Codifique e execute os programas exemplo apresentados.

1.2 Desenvolva um programa para experimentar as várias estratégias de posicionamento dos elementos num contentor.

1.3 Construa uma aplicação que mostre os seguintes eventos numa *JTextArea*:

Evento	Interface do Observador	Métodos na Interface
ActionEvent	ActionListener	actionPerformed()
FocusEvent	FocusListener	focusGained() focusLost()
KeyEvent	KeyListener	keyPressed() keyReleased() keyTyped()
MouseEvent	MouseListener	mouseClicked() mouseEntered() mouseExited() mousePressed() mouseReleased()

1.4 Cada elemento capaz de conter um menu, incluindo os *JApplet* e *JFrame*, possui o método *setJMenuBar()*, que aceita um *JMenuBar* que representa uma barra de menu. Podem ser adicionados menus à barra (instâncias de *JMenu(String)*). Por sua vez, cada menu é constituído por itens (instâncias de *JMenuItem(String)*). Cada item do menu gera o evento do tipo *ActionEvent* quando é seleccionado. Desenvolva um applet com um pequeno conjunto de menus.

Introdução a JavaBeans

● Exercícios (Introdução a JavaBeans)

2.1 Inicie o BDK (c:\jdk1.1\beanbox\run.bat). Aparecem três janelas: a barra de ferramentas (*ToolBox*) com os beans existentes, a *BeanBox* onde se colocam e ligam os componentes e a *Properties* que mostra as propriedades de cada componente. Seleccionar a *Juggler Bean*, colocá-la na *BeanBox* e alterar o *animationRate* na janela de propriedades para 300. Adicionar também o *OurButton bean* ao ambiente e alterar o *label* para stop. Experimente alterar outras propriedades.

Para mapear um evento gerado pelo botão ao *Juggler*, seleccione Edit | Events | action | actionPerformed e clique no *Juggler*. Da lista de métodos disponíveis seleccione *stopJuggling*. É então gerada uma classe adaptadora para que o *Juggler* possa ser observador do evento do botão, executando o método *stopJuggling* em resposta ao evento. Crie um novo botão de start.

2.2 Repita o exercício 2.1 agora com a *Molecule Bean* e *TickTock Bean*.

2.3 Codifique o *Bean* exemplo apresentado (*Frog*). Utilize os botões definidos para testar este componente. Altere o componente por forma a que seja possível “silenciar” o *Frog* através de um botão e por forma a que o texto surja numa *TextArea* (para ter acesso à interface gráfica, pode desenvolver uma *Applet*, derivando de *JApplet*)

2.4 Desenvolva um *Bean* que implemente um contador. Deve ser possível reiniciar o contador, parar o contador e continuar a contagem. Teste o contador com o *TickTock Bean*. Utilize o método *drawString(str,x,y)* para desenhar uma *string* no ecrã.

Conceitos de Programação com Componentes

● Exercícios (Eventos, serialização e reflexão/introspecção)

- 3.1 Desenvolva um *bean* que gere um evento quando um contador chega ao fim. Utilize uma propriedade para indicar o valor inicial do contador e um fio de execução para apresentar graficamente o valor actual da contagem (em ms). Nota: o contador deve ser implementado com um fio de execução.
- 3.2 Desenvolva uma versão do *bean* anterior que permita que este seja guardado em disco.
- 3.3 Desenvolva um programa que permite determinar as variáveis de estado, métodos e construtores de uma classe de objectos.
- 3.4 Utilize o exemplo *Frog*, da aula anterior para testar as propriedades de introspecção. Verifique as propriedades, métodos e eventos suportados por este componente, sem utilizar a informação do *bean*. Adicione ao *bean* a classe *FrogBeanInfo* para controlar a informação, apenas tornando visível o método *croak*. Verifique agora as propriedades visíveis.