

JavaBeans

Java Deployment Course: Aula 8



por Jorge H C Fernandes
(jhcf@di.ufpe.br)
CESAR-DI-UFPE
Recife, 1999

Referências



⌘ Sun Microsystems

- ☒ Documentos, Especificações, Tutoriais, Software, etc
- ☒ <http://java.sun.com/products/beans>
- ☒ <http://java.sun.com/beans/spec.html>
- ☒ <http://java.sun.com/docs/books/tutorial/javabeans/index.html>

⌘ Flashline

- ☒ Comercialização de Beans
- ☒ <http://www.flashline.com/components/javabeans.jsp>

Objetivos desta Aula



- ⌘ Apresentar os conceitos que suportam a construção de Beans
- ⌘ Reforçar conceitos do modelo de tratamento de eventos do AWT
- ⌘ Criar e instalar no JBuilder um componente de software simples no formato de um Bean

Conteúdo



- ⌘ O que é JavaBeans?
- ⌘ Para quê Servem?
- ⌘ Principais Conceitos
- ⌘ Tratamento de Eventos no AWT
- ⌘ Propriedades
- ⌘ Introspecção
- ⌘ Exemplos

O que é JavaBeans?

⌘ Um modelo de componente de software para Java

- ☑ Descrição auto-contida

- ☑ Reutilizável

- ☑ Facilita programação visual

- ☑ Pode ser inserido em um palete de componentes

 - ☒ Consultas e configurações

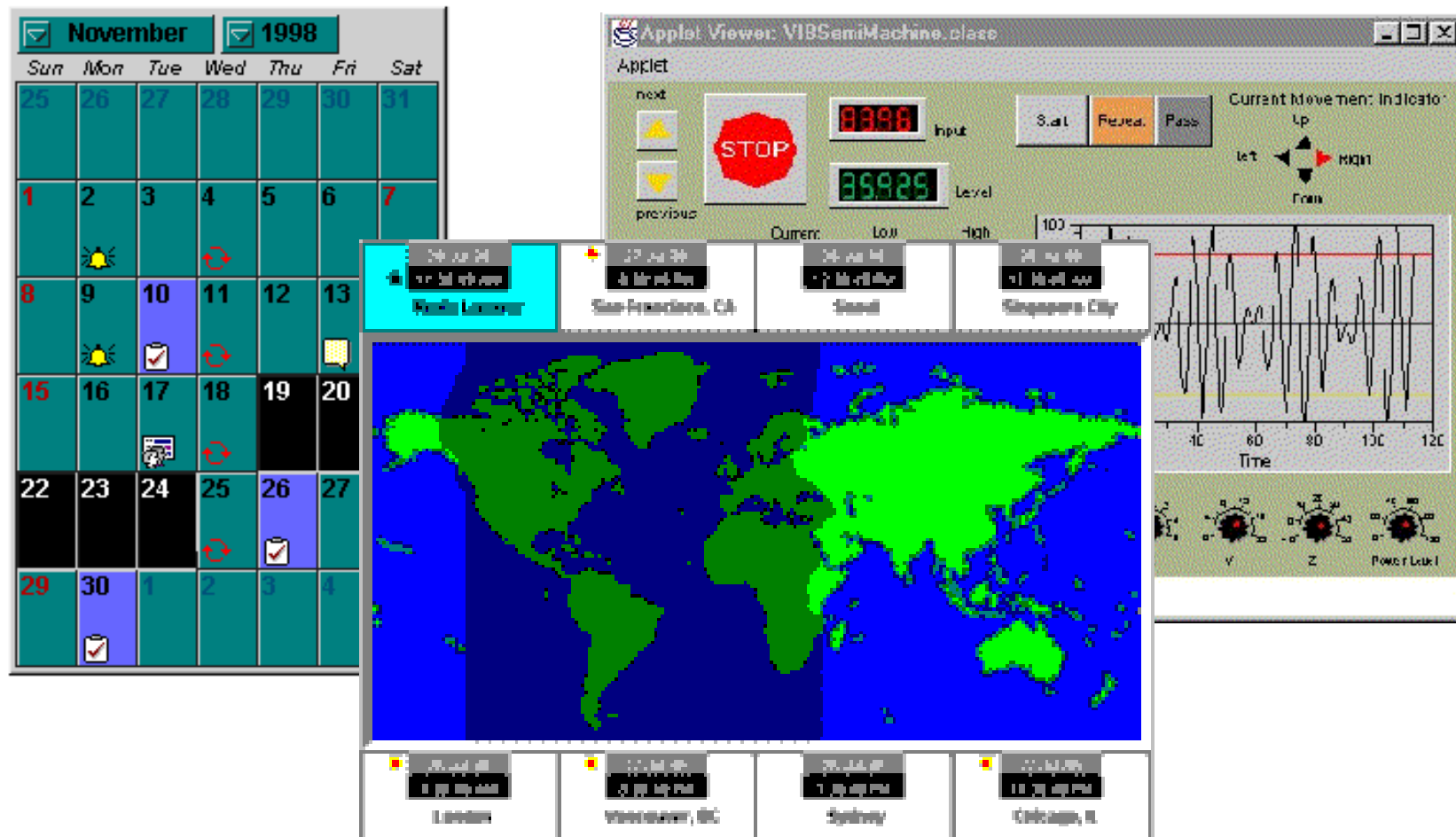
- ☑ Rumo à construção e comercialização de software plug-and-play

Para quê Servem JavaBeans?



- ⌘ Blocos de componentes de GUI
- ⌘ Geradores de gráficos e relatórios
- ⌘ Planilhas
- ⌘ Instrumentos de aquisição e display de dados
- ⌘ Calendários, Relógios, Agendas
- ⌘ Editores

GUI de Alguns Beans Comercializados por Flashline - 1999



Principais Conceitos usados no Modelo Java Beans



⌘ Eventos, Propriedades e Métodos

- ☒ JavaBeans divulgam um conjunto bem definido de eventos que produzem, e permitem que outros objetos registrem interesse na ocorrência destes eventos
- ☒ JavaBeans divulgam um conjunto bem definido de propriedades e métodos, permitindo que as propriedades sejam alteradas e os métodos sejam invocados

⌘ Introspecção e Reflexividade

- ☒ Um JavaBean usa um padrão de codificação que permite que uma ferramenta de edição visual interaja com o componente e deduza/altere suas características (eventos, propriedades e métodos) em build-time ou run-time

⌘ Persistência e Empacotamento

- ☒ Capacidade de armazenar, recuperar ou transmitir um componente através de uma mídia digital (disco, conexão de rede, etc)

Código de um Bean Minimalista

```
import java.awt.*;
import java.io.Serializable;

public class SimpleBean
    extends Canvas
    implements Serializable {

    private Color cor = Color.green;

    public SimpleBean() {
        setSize(60,40);
        setBackground(Color.red);
    }
```

```
    public void paint(Graphics g) {
        g.setColor(cor);
        g.fillRect(20, 5, 20, 30);
    }

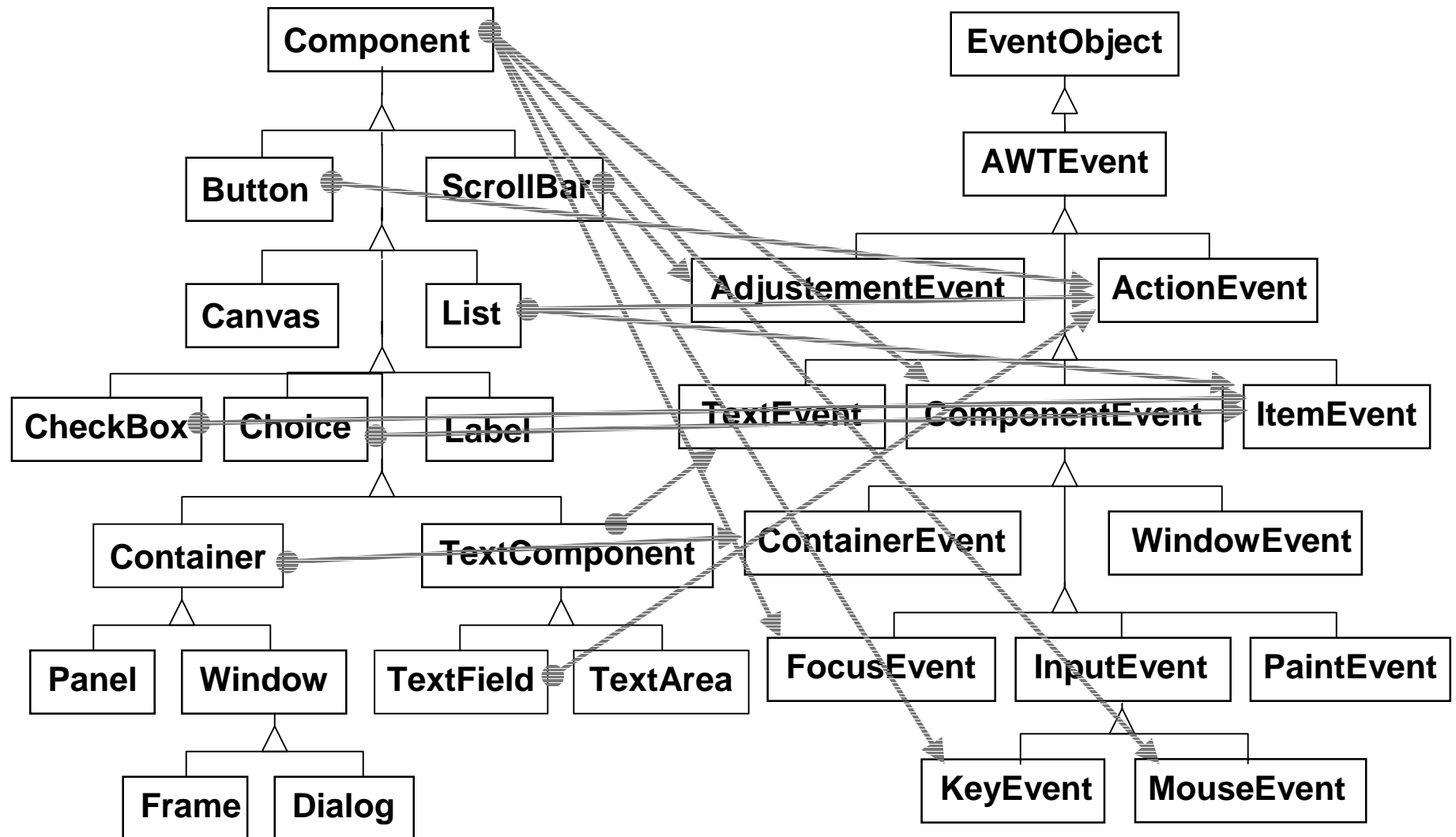
    public void setCor(Color newCor) {
        cor = newCor;
    }

    public Color getCor() {
        return cor;
    }
}
```

Tratamento de Eventos

A thick, horizontal yellow brushstroke underline that spans the width of the slide, positioned directly beneath the title text.

Hierarquia de Componentes e Eventos do AWT

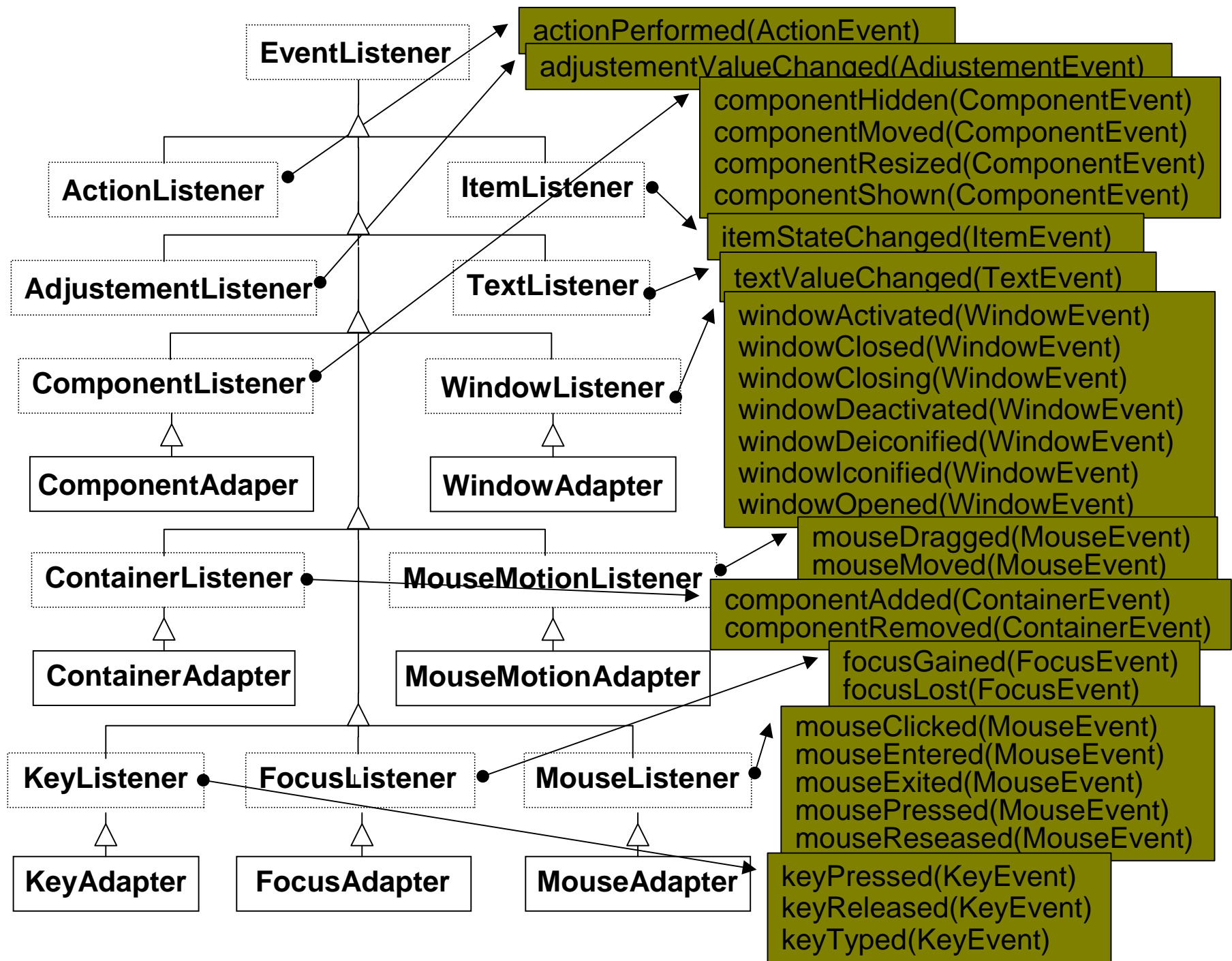


Padrão de Codificação para Eventos

- ⌘ Categorias de evento devem ser criadas (ou reutilizadas)
`class <EVENTNAME>Event extends EventObject`
- ⌘ Interfaces de consumidores evento devem ser criadas (ou reutilizadas)
`interface <EVENTNAME>EventListener extends EventListener {
 public void <MÉTODONOTIFICADOR-1>(<EVENTNAME>Event event);
 public void <MÉTODONOTIFICADOR-2>(<EVENTNAME>Event event);
}`
- ⌘ Produtores de evento (JavaBeans) devem conter métodos para cadastrar consumidores de eventos
`public void add<EVENTNAME>Listener(<EVENTNAME>Listener);`
- ⌘ Consumidores de evento (usários do JavaBean) devem implementar a interface adequada
`class <CONSUMIDORDEEVENTOS> implements <EVENTNAME>Listener`

Interfaces e Classes para Observadores de Eventos do AWT





Propriedades em JavaBeans



Propriedades



- ⌘ Atributos ou características que são publicamente expostas pelo componente
- ⌘ Padrões de Design para propriedades permitem
 - ☑ Descobrir que propriedades um Bean expõe
 - ☑ Determinar atributos de leitura e gravação da propriedade
 - ☑ Determinar o tipo da propriedade
 - ☑ Mostrar o nome e valor das propriedades (em um formulário);
 - ☑ Alterar o valor das propriedades (design-time)
- ⌘ Categorias de Propriedades
 - ☑ Simples, Indexadas, Ligadas, Restritas

Propriedades Simples

⌘ Padrão de codificação

```
public class MeuJavaBean {  
    private <PropertyType> <PropertyName> = <defaultValue>;  
    ...  
    public <PropertyType> get<PropertyName>();  
    public void set<PropertyName>(< PropertyType> value);  
    ...  
}
```

⌘ Propriedades também podem ser computadas (em vez de armazenadas em variáveis)

Propriedades Indexadas

⌘ Contem uma coleção de valores

⌘ Dois Padrões de Codificação

```
public class MeuJavaBean {  
    public < PropertyType>[] get<PropertyName>();  
    public void set<PropertyName>(< PropertyType>[] value);  
}
```

```
public class MeuJavaBean {  
    public < PropertyType> get<PropertyName>(int index);  
    public void set<PropertyName>(int index, < PropertyType> value);  
}
```

Propriedades Ligadas

- ⌘ Usadas quando vários objetos precisam ser automaticamente notificados de mudanças no valor de uma propriedade
- ⌘ Classes e interfaces envolvidas
 - ☐ `PropertyChangeListener`
 - ☐ `PropertyChangeEvent`
 - ☐ `PropertyChangeSupport`

Classes e Interfaces para Propriedades Ligadas

```
void propertyChange(PropertyChangeEvent)
```

⌘ PropertyChangeListener

- ☑ Interface implementada pelo objeto que quer ser notificado

⌘ PropertyChangeEvent

- ☑ Evento que contém informações sobre a propriedade que mudou e os valores antes e após a modificação

⌘ PropertyChangeSupport

- ☑ Implementação básica de cadastramento e descadastramento de listeners, notifica os listeners das modificações

- ☑ O JavaBean pode herdar de PropertyChangeSupport ou implementar internamente

```
Object getNewValue()  
Object getOldValue()  
Object getPropagationId() -- reservado  
String getPropertyName()  
void setPropagationId(Object)
```

```
addPropertyChangeListener(PropertyChangeListener)  
firePropertyChange(String, Object, Object)  
removePropertyChangeListener(PropertyChangeListener)
```

Padrão de Codificação para Beans com Propriedades Ligadas

```
import java.beans.*;

class MyButton extends Button {
    private PropertyChangeSupport changes = new PropertyChangeSupport(this);

    public void addPropertyChangeListener(PropertyChangeListener l) {
        changes.addPropertyChangeListener(l);
    }
    public void removePropertyChangeListener(PropertyChangeListener l) {
        changes.removePropertyChangeListener(l);
    }
    public void setLabel(String newLabel) {
        String oldLabel = label;
        label = newLabel;
        changes.firePropertyChange("label", oldLabel, newLabel);
    }
}
```

Listener de Propriedades Ligadas (Usando Adaptador)

```
public class MyClass {  
    MyButton button = new MyButton();  
    ...  
    PropertyChangeAdapter adapter = new PropertyChangeAdapter();  
    ...  
    button.addPropertyChangeListener(adapter);  
    ...  
    class PropertyChangeAdapter implements PropertyChangeListener {  
        public void propertyChange(PropertyChangeEvent e) {  
            reporter.reportChange(e);  
        }  
    }  
}
```

Propriedades Restritas



⌘ Uma mudança no valor da propriedade pode ser vetada por um ou mais `VetoableChangeListeners`

⌘ Classes envolvidas

- ☑ `VetoableChangeListener`

- ☑ `PropertyChangeEvent`

- ☑ `VetoableChangeSupport`

- ☑ `PropertyVetoException`

Classes e Interfaces para Propriedades Restritas

⌘ VetoableChangeListener

- ☑ Registra interesse em vetar propostas de mudanças de valores

⌘ VetoableChangeSupport

- ☑ Implementação básica de componente que aceita cadastramento e descadastramento de listeners que podem vetar modificações
- ☑ O JavaBean pode herdar de VetoableChangeSupport ou internalizar um objeto desta classe

⌘ PropertyVetoException

- ☑ Exceção levantada caso haja veto de algum listener

Padrão de Codificação para Propriedades Restritas

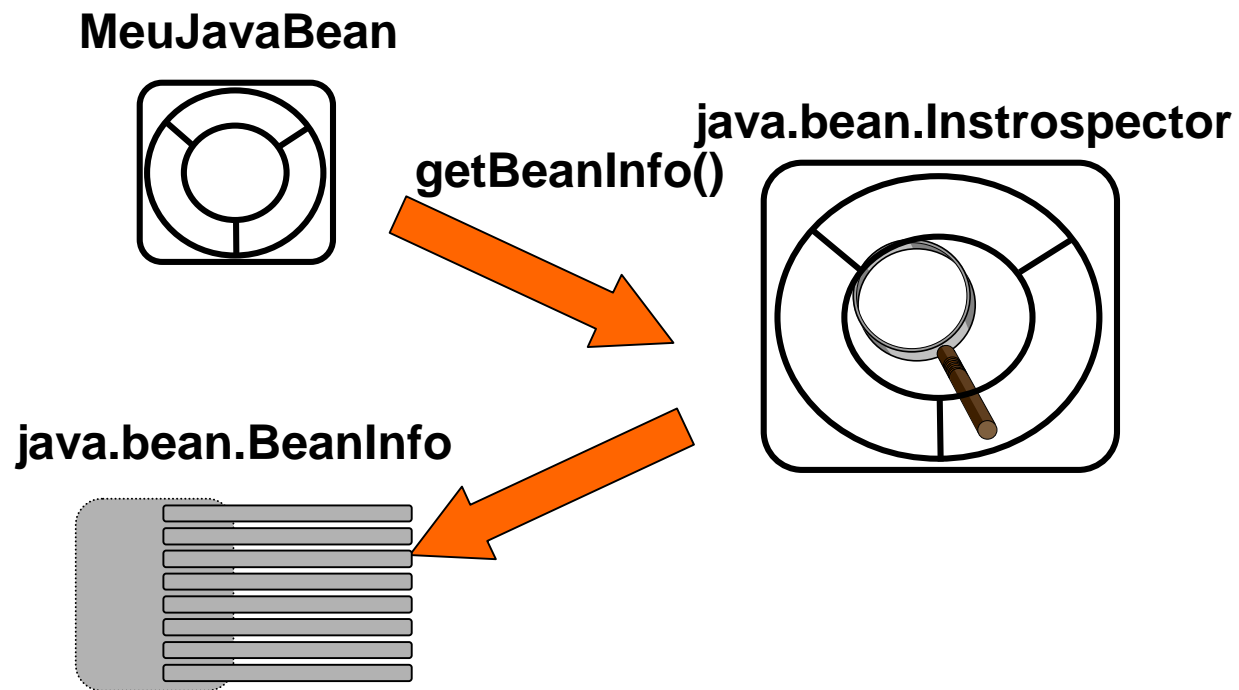
```
class <BeanClassName> {
    private VetoableChangeSupport vetos = new VetoableChangeSupport(this);

    public void addVetoableChangeListener(VetoableChangeListener l) {...}

    public void removeVetoableChangeListener(VetoableChangeListener l) {...}

    public void setPreco(int newPreco)
                                throws PropertyVetoException {
        int oldPreco = price;
        // Notifica os observadores sobre a proposta de mudança.
        vetos.fireVetoableChange("preco", ..oldPreco.., ..newPreco..);
        // Nenhum observador vetou a mudança. Prossegue com elas
        preco = newPreco;
        changes.firePropertyChange("preco", ..oldPreco.., ..newPreco..));
    }
}
```

Introspecção



Objeto java.bean.BeanInfo

BeanDescriptor getBeanDescriptor()

BeanInfo[] getAdditionalBeanInfo()

MethodDescriptor[] getMethodDescriptors()

PropertyDescriptor[] getPropertyDescriptors()


int getDefaultPropertyIndex()

EventSetDescriptor[] getEventSetDescriptors()

int getDefaultEventIndex()

Image getIcon(int iconKind)

Reflexividade (`java.lang.reflect`)



- ⌘ Permite manipular classes, interfaces e objetos contidos na máquina virtual
- ⌘ Usada na construção de depuradores, ferramentas de construção de GUI, browsers de classes
- ⌘ Executar em run-time, várias operações que normalmente são programadas

Classes de Suporte a Reflexividade



⌘ **java.lang.reflect.Array** - métodos para criar e acessar arrays de forma dinâmica

⌘ **java.lang.reflect.Class** - representa informação sobre classes e interfaces

⌘ **java.lang.reflect.Constructor** - provê informações e acesso aos construtores de objetos de uma classe. Permite instanciar uma classe dinamicamente

⌘ **java.lang.reflect.Field** - provê informações e acesso dinâmico a um atributo de uma classe ou interface

⌘ **java.lang.reflect.Method** - provê informação e acesso a um método de uma classe ou interface. Permite que se crie e execute invocação dinâmica do método.

⌘ **java.lang.reflect.Modifier** - provê métodos e constantes que fornecem informação sobre os modificadores de acesso de uma classe e de seus atributos e métodos.

⌘ **java.lang.Object** - provê o método getClass()

Possibilidades de Uso de Reflexão



- ⌘ Determinar a classe de um objeto
- ⌘ Obter informação sobre modificadores da classe, seus métodos, campos, construtores e superclasses
- ⌘ Observar quais constantes e métodos fazem parte de uma interface
- ⌘ Carregar uma classe na máquina virtual, cujo nome só é conhecido em tempo de execução
- ⌘ Ler e modificar o valor do campo de um objeto, mesmo que o nome do campo só seja conhecido em tempo de execução
- ⌘ Invocar um método de um objeto, mesmo que o método só tenha sido conhecido em tempo de execução
- ⌘ Criar um novo array cujo tamanho e tipo dos componentes so' sejam conhecidos em runtime, e modificar os componentes do array

Outras Características de JavaBeans



- ⌘ Editor de propriedades customizado
- ⌘ Java Activation Framework - JAF
- ⌘ Ajuste fino de serialização
 - ☑ interface Externalizable
- ⌘ Drag & Drop

Exercício



- ⌘ Construa um SimpleBean e o insira na paleta de componentes do JBuilder