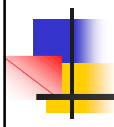


Programowanie w języku Java

Wykład 7: JavaBeans

Programowanie komponentowe

- Klasy – uniwersalne komponenty
- Wymagania:
  - Standardowy dostęp do pól
  - Standardowy mechanizm wywoływania metod
- Komponent:
  - Zestaw własności
  - Zestaw metod operujących na własnościach
  - Obsługa zdarzeń
  - Trwały

Komponenty w Javie

- Java.beans
  - Pakiet zawierający API zgodny z architekturą JavaBeans
- NetBean BUI Builder
  - Środowisko graficzne
  - Wspomaga definiowanie komponentów

Komponenty JavaBean

- Dowolna klasa Javy zawierająca metody:
  - getXxxx i setXxxx - dla każdej własności Xxxx, lub
  - isXxxx i setXxxx - dla własności Xxxx typu boolean
  - addYyyy i removeYyyy – dla obsługi zdarzeń Yyyy
  - brak publicznych pól
  - tylko konstruktor domyślny (bezparametrowy)
- Zalety:
  - programowanie komponentowe
  - łatwo wydobywać informacje z dowolnego Beana

Przykład (1)

```
public class BangBean extends JPanel
    implements Serializable {
  protected int xm, ym;
  protected int cSize = 20;
  protected String text = "Bang!";
  protected int fontSize = 48;
  protected Color tColor = Color.red;
  protected ActionListener actionListener;
  public BangBean() {
    addMouseListener(new ML());
    addMouseMotionListener(new MML());
  }
  public int getCircleSize() { return cSize; }
  public void setCircleSize(int newSize) {
    cSize = newSize;
  }
```

```
public String getBangText() { return text; }
  public void setBangText(String newText) {
    text = newText;
  }
  public int getFontSize() { return fontSize; }
  public void setFontSize(int newSize) {
    fontSize = newSize;
  }
  public Color getTextColor() { return tColor; }
  public void setTextColor(Color newColor) {
    tColor = newColor;
  }
  public void paintComponent(Graphics g) {
    super.paintComponent(g);
    g.setColor(Color.black);
    g.drawOval(xm - cSize/2, ym - cSize/2,
      cSize, cSize);
  }
```


Przykład (2)

```
public void addActionListener ( ActionListener l)
      throws TooManyListenersException {
   if(actionListener != null)
     throw new TooManyListenersException();
   actionListener = l;
  }
  public void removeActionListener(ActionListener l) {
    actionListener = null;
  }
class MML extends MouseMotionAdapter {
    public void mouseMoved(MouseEvent e) {
      xm = e.getX();   ym = e.getY();
      repaint();
    }
  }
```

```
class ML extends MouseAdapter {
    public void mousePressed(MouseEvent e) {
      Graphics g = getGraphics();
      g.setColor(tColor);
      g.setFont(new Font(
         "TimesRoman", Font.BOLD, fontSize));
      int width =  g.getFontMetrics().stringWidth(text);
      g.drawString(text, (getSize().width - width) /2,
        getSize().height/2);
      g.dispose();
      if(actionListener != null)
        actionListener.actionPerformed(
          new ActionEvent(BangBean.this,
            ActionEvent.ACTION_PERFORMED, null));
    }
  }
public Dimension getPreferredSize() {
    return new Dimension(200, 200);
  }
}
```

Własności

- Simple
  - typu prostego, wartość niezależna od innych własności
- Indexed
  - Zakres wartości
- Bound
  - Zmiana wartości powoduje powiadomienie innego komponentu
- Constrained
  - Zmiana dozwolona za zezwoleniem innego komponentu

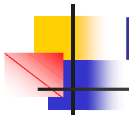
Proste komponenty

```
public class MyBean {
   public MyBean() {
   }
   /**
    * Holds value of property title.
    */
   private String title;
   /**
    * Getter for property title.
    * @return Value of property title.
    */
   public String getTitle() {
      return this.title;
   }
   /**
    * Setter for property title.
    * @param title New value of property title.
    */
   public void setTitle(String title) {
      this.title = title;
   }
}
```

Komponent zwykły

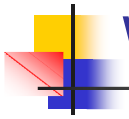
```
import java.awt.Graphics;
import java.io.Serializable;
import javax.swing.JComponent;
public class MyBean   extends JComponent
      implements Serializable  {
   private String title;
   public String getTitle()    {
      return this.title;
   }
   public void setTitle( String title )   {
      this.title = title;
   }
   protected void paintComponent( Graphics g ){
      g.setColor( getForeground() );

      int height = g.getFontMetrics().getHeight();
      if ( this.title != null )
         g.drawString(this.title, 0, height );
      }
  }
```

Komponent graficzny



Dostęp do własności

- Read/Write
  - Metody setWłasność() i getWłasność()
- Read only
  - Tylko metoda getWłasność()
- Write only
  - Tylko metoda setWłasność()



Własności powiązane

- Gdy wartość własności zostaje zmieniona wysyłane jest powiadomienie PropertyChangeEvent do „nasłuchujących” komponentów
- Metoda propertyChange()
  - wysyła w/w powiadomienie
- PropertyChangeSupport
  - klasa zwiera metody umożliwiające rejestrację nasłuchujących komponentów
- PropertyChangeListener
  - Komponent „nasłuchujący” zmiany własności

```
import java.awt.Graphics;
import java.beans.PropertyChangeListener;
import java.beans.PropertyChangeSupport;
import java.io.Serializable;
import javax.swing.JComponent;

public class MyBean  extends JComponent
        implements Serializable{
    private String title;
    private String[ ] lines = new String[10];

    private final PropertyChangeSupport pcs =
            new   PropertyChangeSupport( this );

    public String getTitle()    {
        return this.title;
    }

    public void setTitle( String title )    {
        String old = this.title;
        this.title = title;
        this.pcs.firePropertyChange( "title", old, title );
    }

    public String[ ] getLines()    {
        return this.lines.clone();
    }

    public String getLines( int index )    {
        return this.lines[index];
    }

    public void setLines( String[ ] lines )    {
        String[ ] old = this.lines;
        this.lines = lines;
        this.pcs.firePropertyChange( "lines", old, lines );
    }

    public void setLines( int index, String line )    {
        String old = this.lines[index];
        this.lines[index] = line;
        this.pcs.fireIndexedPropertyChange( "lines", index,
          old, lines );
    }

    public void addPropertyChangeListener(
         PropertyChangeListener listener )    {
          this.pcs.addPropertyChangeListener( listener );
    }

    public void removePropertyChangeListener(
      PropertyChangeListener listener )    {
        this.pcs.removePropertyChangeListener( listener );
    }

    protected void paintComponent( Graphics g )    {
        g.setColor( getForeground() );
        int height = g.getFontMetrics().getHeight();
        paintString( g, this.title, height );
        if ( this.lines != null )
        {
            int step = height;
            for ( String line : this.lines )
                paintString( g, line, height += step );
        }
    }

    private void paintString( Graphics g, String str, int height )
    {
        if ( str != null )
            g.drawString( str, 0, height );
    }
}
```

Własności ograniczone

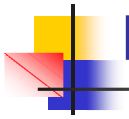
- Podobnie jak własności powiązane ale komponent nasłuchująca jest typu VetoableChangeListener
- Komponent może zawetować zmianę zgłaszając wyjątek PropertyVetoException
- Implementacja metody setXxx:
  - Zapamiętanie starej wartości
  - Powiadomienie komponentu nasłuchującego
  - Jeśli komponent zawetował zmianę to przywrócenie starej wartości

```
import java.io.Serializable;
import java.beans.PropertyChangeListener;
import java.beans.PropertyChangeSupport;
import java.beans.PropertyVetoException;
import java.beans.VetoableChangeListener;
import java.beans.VetoableChangeSupport;
import java.awt.Graphics;
import javax.swing.JComponent;
public class MyBean        extends JComponent
                implements Serializable{
private String title;
private String[] lines = new String[10];
private final PropertyChangeSupport pcs = new
                PropertyChangeSupport( this );
private final VetoableChangeSupport vcs = new
                VetoableChangeSupport( this );
public String getTitle()    {        return this.title;    }
public void setTitle( String title )  throws
                PropertyVetoException {
   String old = this.title;
   this.vcs.fireVetoableChange( "title", old, title );
   this.title = title;
   this.pcs.firePropertyChange( "title", old, title );
}
public String[] getLines()    {  return this.lines.clone();  }
public String getLines( int index ) {
 return this.lines[index];
 }
public void setLines( String[ ] lines )    throws
        PropertyVetoException    {
        String[] old = this.lines;
        this.vcs.fireVetoableChange( "lines", old, lines );
        this.lines = lines;
        this.pcs.firePropertyChange( "lines", old, lines );
  }
```

```
public void setLines( int index, String line )  throws
                PropertyVetoException    {
  String old = this.lines[index];
  this.vcs.fireVetoableChange( "lines", old, line );
    this.lines[index] = line;
  this.pcs.fireIndexedPropertyChange( "lines", index, old, line );
 }
public void addPropertyChangeListener( PropertyChangeListener
listener )    {
   this.pcs.addPropertyChangeListener( listener );
 }
public void removePropertyChangeListener( PropertyChangeListener
listener )    {
   this.pcs.removePropertyChangeListener( listener );
}
 public void addVetoableChangeListener(
            VetoableChangeListener listener )    {
    this.vcs.addVetoableChangeListener( listener );
}
public void removeVetoableChangeListener(
                VetoableChangeListener listener )    {
  this.vcs.removeVetoableChangeListener( listener );
}
protected void paintComponent( Graphics g )    {
   g.setColor( getForeground() );
   int height = g.getFontMetrics().getHeight();
   paintString( g, this.title, height );
   if ( this.lines != null )        {
         int step = height;
         for ( String line : this.lines )
           paintString( g, line, height += step );        }
   }
 private void paintString( Graphics g, String str, int height )    {
    if ( str != null )
    g.drawString( str, 0, height );
  }
}
```

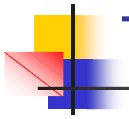
Własności indeksowane

- Dostęp do elementów:
  - public PropertyElement getPropertyName(int index)
  - public void setPropertyName(int index, PropertyElement element)
- Dostęp do całej tablicy:
  - public PropertyElement[] getPropertyName()
  - public void setPropertyName(PropertyElement element[])



Komunikacja pomiędzy komponentami

- Nadajnik:
  - wysyła zdarzenie
  - public void add<EventListenerType>(<EventListenerType> a)
  - public void remove<EventListenerType>(<EventListenerType> a)
- Zdarzenie:
  - zawiera informację
  - ActionEvent
- Komponent nasłuchujący:
  - odbiera zdarzenie
  - <EventListenerType>



Trwałość komponentów

- Komponenty są serializowalne
- Można kontrolować serializację
- Trwałość „długoterminowa”
  - Serializacja do formatu XML
  - XMLEncoder, XMLDecoder

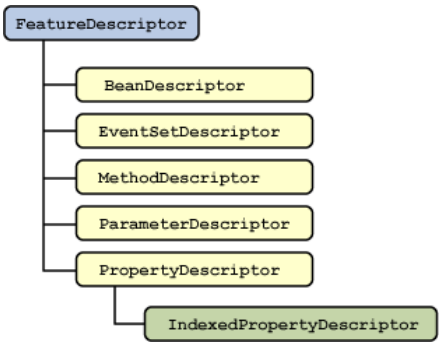
Przykład komponentu w XMLu

```
<?xml version="1.0" encoding="UTF-8" ?>
<java>
 <object class="javax.swing.JFrame">
  <void method="add">
   <object class="java.awt.BorderLayout" field="CENTER"/>
   <object class="SimpleBean"/>
  </void>
  <void property="defaultCloseOperation">
   <object class="javax.swing.WindowConstants" field="DISPOSE_ON_CLOSE"/>
  </void>
  <void method="pack"/>
  <void property="visible">
   <boolean>true</boolean>
  </void>
 </object>
</java>
```


Introspekcja komponentów

- BeanInfo
  - Intefejs do implementacji klas zawierających informacje o komponencie (deskryptory)
- Introspector
  - Klasa do introspekcji getBeanInfo(beanName)

Deskryptory komponentów



Koniec