Programowanie w języku JAVA

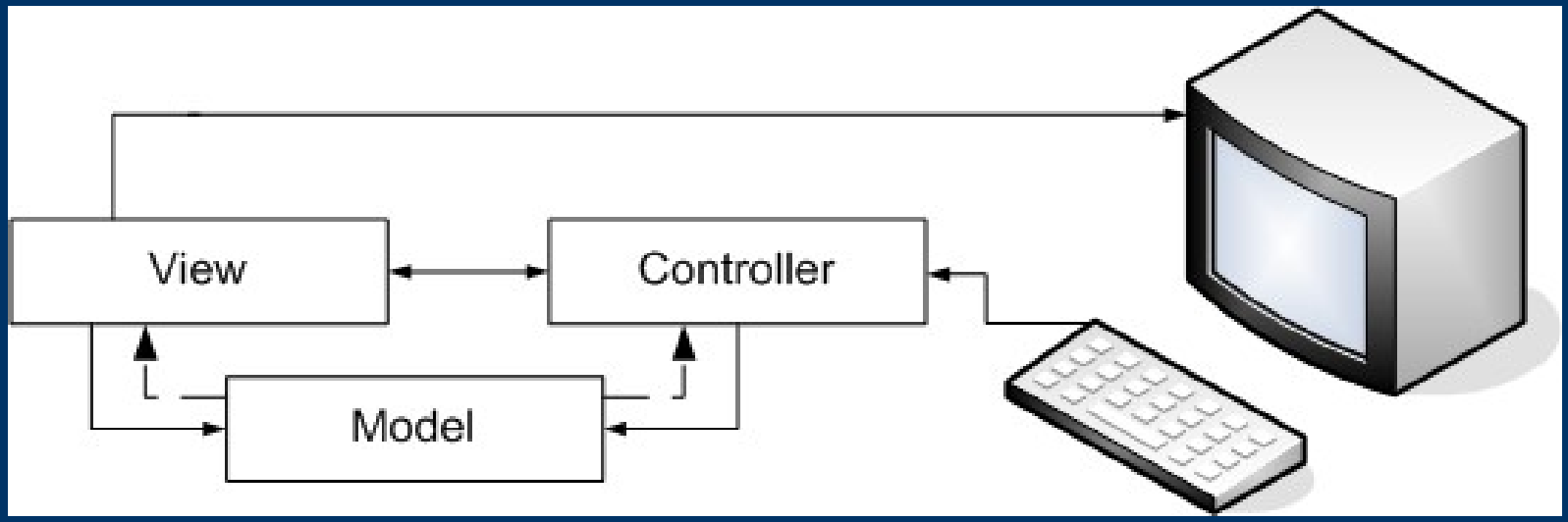
Wykład IV
Swing - GUI

Architektura

- JFC (Java Foundation Classes) – zbiór klas do budowy interfejsu graficznego użytkownika i interaktywności aplikacji Javy
  - Komponenty Swing – etykiety, przyciski, listy, panele, okna, itd.,
  - Przełączany wygląd Look-and-Feel – dostosowanie GUI do platformy systemowej, wiele różnych wyglądów,
  - API Dostępności (Accessibility API) – przesyłanie informacji z GUI do urządzeń typu ekrany Braila,
  - Java 2D – biblioteka do tworzenia wysokiej jakości grafiki 2D,
  - Internacjonalizacja – aplikacje mogą wykorzystywać różne języki i zlokalizowane dane

Architektura

- Architektura oparta o zorientowaną obiektowo dekompozycję projektową interfejsu użytkownika MVC (Model-View-Controller)



Architektura

- ***Model*** – opisuje stan komponentu, np. czy przycisk jest wciśnięty, jaki jest tekst w polu tekstowym. Może być odpowiedzialny za pośrednią komunikację pomiędzy kontrolerem i widokiem. Nie zna referencji do widoku i kontrolera. Wysyła zdarzenia, rozgłasza. Linie --
- ***Widok*** – określa wizualną reprezentację modelu komponentu. Wyświetla odpowiedni kolor, czcionkę, itd. Odpowiada za aktualizację widoku. Odbiera bezpośrednie komunikaty od kontrolera i pośrednie od modelu.
- ***Kontroler*** – jest odpowiedzialny za reakcje komponentu na oddziaływanie za pomocą urządzeń wejściowych: klawiatury i myszy. Jest „czuciem” komponentu. Decyduje o akcjach jakie są podejmowane kiedy komponent jest używany. Może odbierać bezpośrednie komunikaty od widoku i pośrednie od modelu.

Architektura c.d.

- Zalety MVC
  - Tworzenie własnego widoku (look and feel) bez modyfikacji modelu
  - Możliwość stworzenia własnego modelu, np: książki telefonicznej o określonej strukturze
  - Możliwość podłączania jednego modelu do wielu różnych widoków
- Kontenery JApplet, JFrame, JDesktopPane, itd. nie posiadają modelu
- Niektóre komponenty posiadają więcej modeli: JList (składniki listy i składniki zaznaczenia)

Architektura - model-delegat

- Widok i kontroler tworzą delegata interfejsu graficznego: architektura *model-delegat*
- Każdy delegat jest dziedziczony po klasie ComponentUI
- W celu wymuszenia użycia określonego delegata wykorzystuje się metodę setUI() komponentu, np:

```
JButton b = new JButton(„OK”);
b.setUI(javax.swing.plaf.metal.MetalButtonUI.
  createUI(b));
```

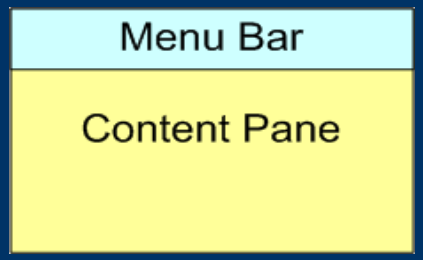
- Zmiana wyglądu całego interfejsu

```
UIManager.setLookAndFeel("com.sun.java.swing.plaf
  .motif.MotifLookAndFeel");
```

Więcej: http://java.sun.com/products/jfc/tsc/articles/architecture/

Kontenery najwyższego poziomu

- Wszystkie komponenty Swing muszą być częścią drzewa, którego korzeniem jest kontener: JFrame, JInternalFrame , JDialog lub JApplet,
- Budowa okna – JFrame



- Do panelu zawartości dodajemy komponenty:

```
okno.getContentPane().add(przycisk, BorderLayout.NORTH);
okno.add(przycisk, BorderLayout.NORTH);
```

- Możemy dodać cały panel:

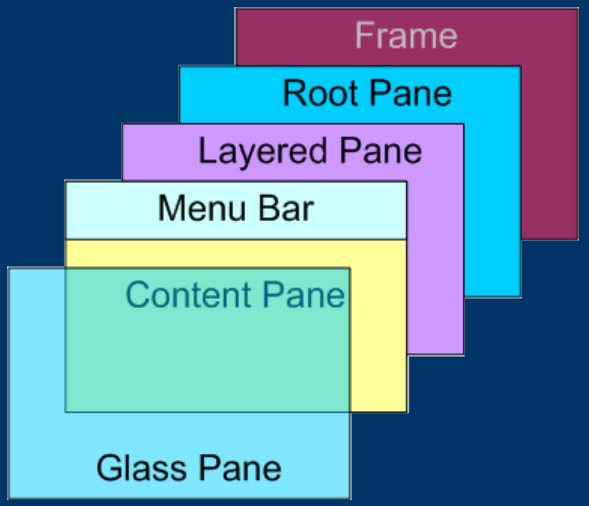
```
JPanel cp = new JPanel(new GridLayout(3,3));
cp.add(....);
okno.setContentPane(cp);
```

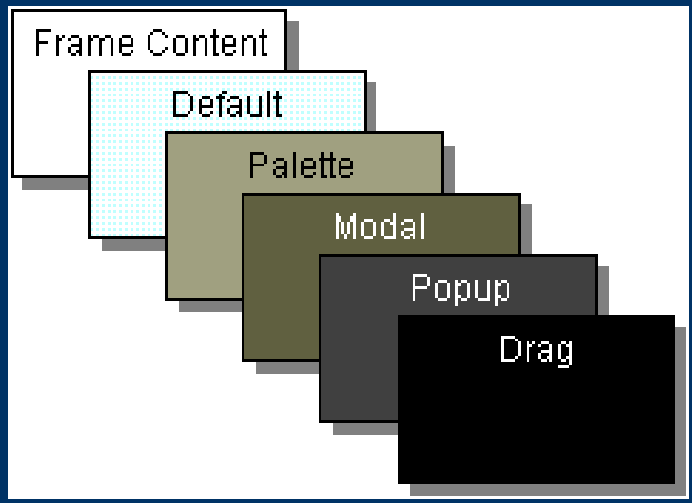
- Dodanie menu: `okno.setJMenuBar(menu);`

Kontenery najwyższego poziomu

- Wszystkie kontenery najwyższego poziomu posiadają pośredni kontener zwany Root Pane:
  - Glass Pane – przykrywa okno jak szyba
  - Layered Pane – może zawierać komponenty w porządku względem osi Z

    Frame Content – poziom Content Pane (-30000)
    Default – poziom domyślny dla dodawanych komponentów (0)
    Palette – poziom pływających pasków narzędziowych i palet (100)
    Modal – poziom modalnych okien wewnętrznych (200)
    Popup – wyskakujące menu (300)
    Drag – poziom przeciągania elementów (400)





Klasa JComponent

- Wszystkie komponenty Swing których nazwy zaczynają się od „J” są pochodnymi klasy **`JComponent`**
- Właściwości:
  - **`setToolTipText()`** - ustawienie dymku podpowiedzi,
  - **`setBorder()`** - ustawienie obramowania,
  - **`paintComponent()`** - rysowanie na komponencie,
  - Każdy komponent ma odpowiadający obiekt **`ComponentUI`**, który przeprowadza rysowanie, przechwytywanie zdarzeń, ustalanie rozmiaru, itd. jest on zależny od bieżącego wyglądu interfejsu, który ustawiamy poleceniem **`UIManager.setLookAndFeel(...)`**
  - **`putClientProperty()`** - można dołączyć pary klucz wartość do każdego komponentu (get...)

Klasa JComponent II

- Właściwości c.d.:
  - Umożliwiają ustawianie układu (layout) – klasa Component wprowadza metody getPreferredSize, getAligmentX, getMinimumSize, getMaximumSize, set.... (implementacja w klasach pochodnych)
  - Wspomagają dostępność (accessibility) – JComponent udostępnia API dla urządzeń takich jak syntezatory mowy odczytujące interfejs użytkownika,
  - Wspomagają technikę „przeciągnij i upuść”,
  - Wspomagają podwójne buforowanie – płynne rysowanie na ekranie,
  - Umożliwiają definicje skrótów klawiszowych

```
komponent.getInputMap().put(KeyStroke.getKeyStroke
  („F2”), „pressed”);
komponent.getActionMap().put(„pressed”, akcja(typu
  javax.swing.Action));
```

lub `komponent.setMnemonic(KeyEvent.VK_A);`

Tworzenie GUI

- Dodawanie komponentu do kontenera
  - utworzenie komponentu (obiektu pewnej klasy)
  - określenie kontenera pośredniego – panelu z zawartością (ang. *content pane*) - niekonieczne
  - wywołanie metody *add()* kontenera lub kontenera pośredniego
    `add(komponent)`
- Kontener – także rodzaj komponentu

Aplikacja Przycisk

```
import javax.swing.*;
import java.awt.*;
public class Przycisk extends JFrame {
  public Przycisk() {
    JButton p = new JButton("Naciśnij");
    Container kont = getContentPane();
    kont.add(p); //lub poprostu add(p);
  }
  public static void main(String args[]){
     Przycisk b = new Przycisk(„Aplikacja 1”);
     b.setVisible(true);
  }
}
```

Rozmieszczanie komponentów

- Sposób rozmieszczenia komponentów
  - zależny od zarządcy układu (Layout Manager)
- Zastosowanie zarządcy układu – metoda *setLayout()* kontenera
- Predefiniowani zarządcy układu
  - ciągły – *FlowLayout*
  - siatkowy - *GridLayout*
  - brzegowy – *BorderLayout*
  - kartowy – *CardLayout*
  - torebkowy - *GridBagLayout*
  - pudełkowy – *BoxLayout*
  - wiosenny – *SpringLayout*
  - grupowy - *GroupLayout*

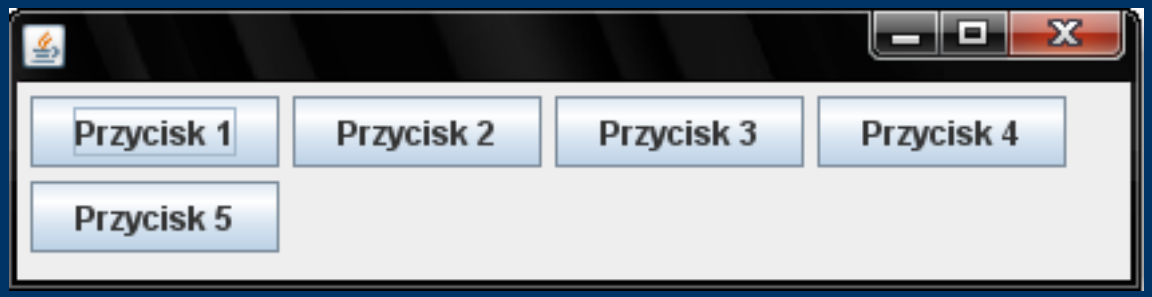
Rozkład ciągły (FlowLayout)

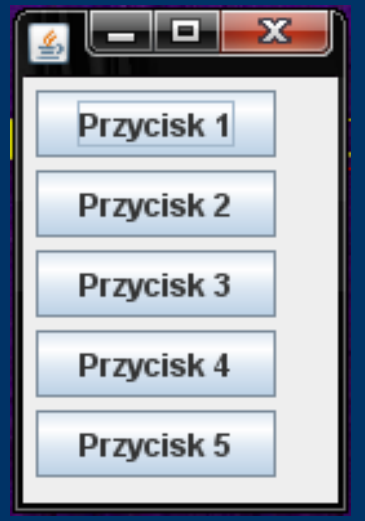
- Rozmieszcza komponenty w kolejnych wierszach od lewej do prawej (rozmiar komponentu – minimalny)
- Konstruktory
  - **`FlowLayout()`** **// wyrównanie do środka**
  - **`FlowLayout(int wyrównanie)`**
  - **`FlowLayout(int wyrównanie, int dX, int dY)`**
- Określenie wyrównania:

  `FlowLayout.LEFT, FlowLayout.RIGHT, FlowLayout.CENTER`
- Odstęp między komponentami w poziomie (*dX*) i pionie (*dY*) – domyślnie 5 pikseli

Rozkład ciągły - aplikacja

```
import javax.swing.*;
import java.awt.*;
public class PrzyciskFL extends JFrame {
     PrzyciskFL() {
        setLayout(new FlowLayout(FlowLayout.LEFT));
        setSize(300,300);
        for(int i=1;i<6;i++)
           add(new JButton("Przycisk " + i));
        setVisible(true);
     }
      public static void main(String[] args){
         PrzyciskFL fl = new PrzyciskFL();
      }
}
```





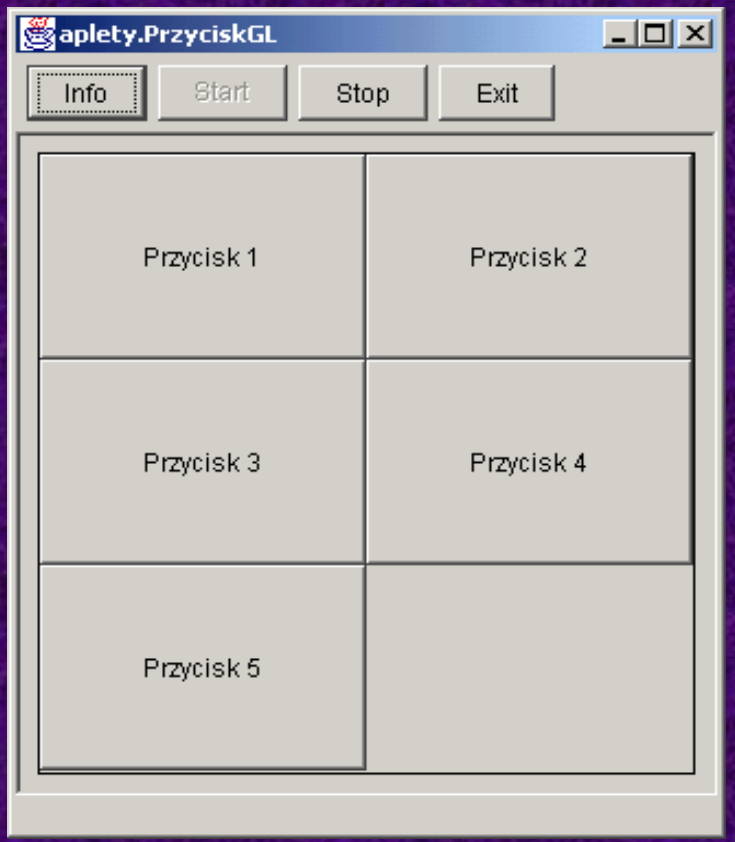
Rozkład siatkowy (*GridLayout*)

- **Podział kontenera na wiersze i kolumny – rozmieszcza komponenty od lewej do prawej i od góry do dołu (rozmiar komponentów – aby wypełnić obszar)**
- **Konstruktory (0 – dowolna ilość wierszy lub kolumn)**
  - `GridLayout(int wiersze, int kolumny)`
  - `GridLayout(int wiersze, int kolumny, int dX, int dY)`

Rozkład siatkowy (*GridLayout*)

- **Przykład**

```
kont.setLayout(new  GridLayout(3,2));
```



Rozkład brzegowy (*BorderLayout*)

- Dodawanie komponentu
  `add(int stała, komponent);`
- Dopuszczalne wartości stałej
  `BorderLayout.NORTH`
  `BorderLayout.SOUTH BorderLayout.EAST`
  `BorderLayout.WEST BorderLayout.CENTER`
- Mogą być też stałe relatywne:

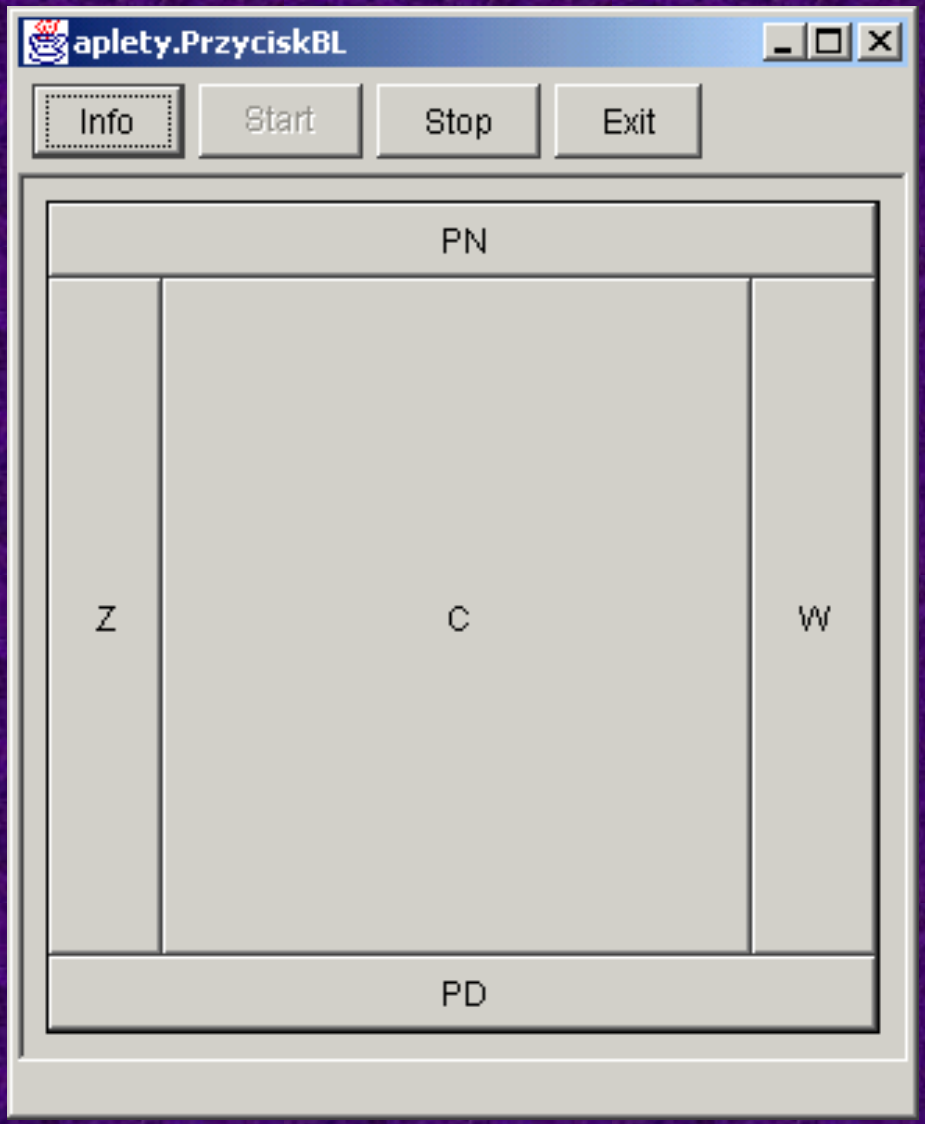
`PAGE_START, PAGE_END, LINE_START i LINE_END`

ponieważ dodawanie komponentów może mieć różne orientacje w kontenerze, np.
`ComponentOrientation.LEFT_TO_RIGHT`

Rozkład brzegowy – aplet *PrzyciskBL*

```
import javax.swing.*; import java.awt.*;
public class PrzyciskBL extends JApplet {
  public void init() {
    Container k = getContentPane();
    k.add(BorderLayout.NORTH,
          new JButton("PN"));
    k.add(BorderLayout.SOUTH,
          new JButton("PD"));
    k.add(BorderLayout.WEST,
          new JButton("Z"));
    k.add(BorderLayout.EAST,
          new JButton("W"));
    k.add(BorderLayout.CENTER,
          new JButton("C"));  } }

//new Borderlayout(int odsPion, int
  odsPoziom)
```

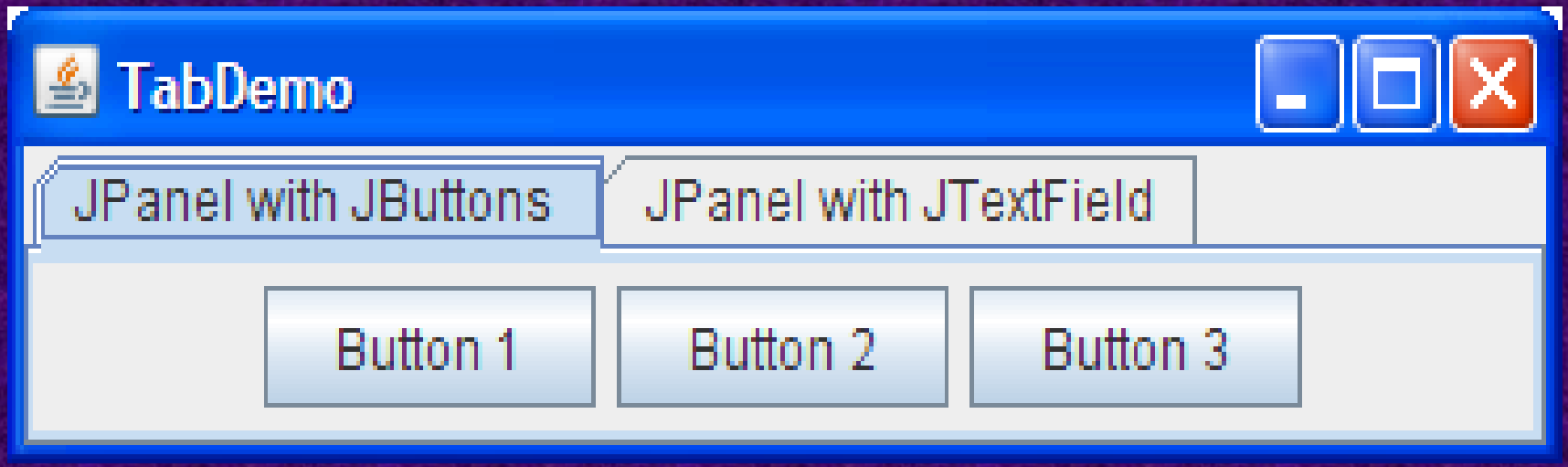
Rozkład brzegowy – aplet *PrzyciskBL*



Rozkład kartowy (*CardLayout*)

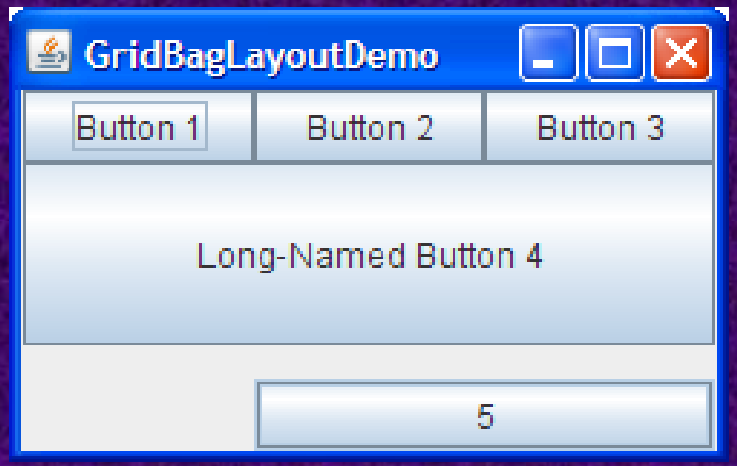
```
JPanel zakladka1 = new JPanel();
JPanel zakladka2 = new JPanel();
..
JPanel zakladki = new JPanel(new CardLayout());
zakladki.add(zakladka1, „JPanel with Buttons”);
zakladki.add(zakladka2, „JPanel with JtextField”);
```

źródło:http://java.sun.com/docs/books/tutorial/uiswing/layout



Rozkład torebkowy (*GridBagLayout*)

```
JPanel pane = new JPanel(new GridBagLayout());
GridBagConstraints c = new GridBagConstraints();
button = new JButton("Button 1");
c.weightx = 0.5;
c.gridx = 0;
c.gridy = 0;
pane.add(button, c);
...
button = new JButton("Long-Named Button 4");
c.ipady = 40;      //make this component tall
c.weightx = 0.0;
c.gridwidth = 3;
c.gridx = 0;
c.gridy = 1;
pane.add(button, c);
```

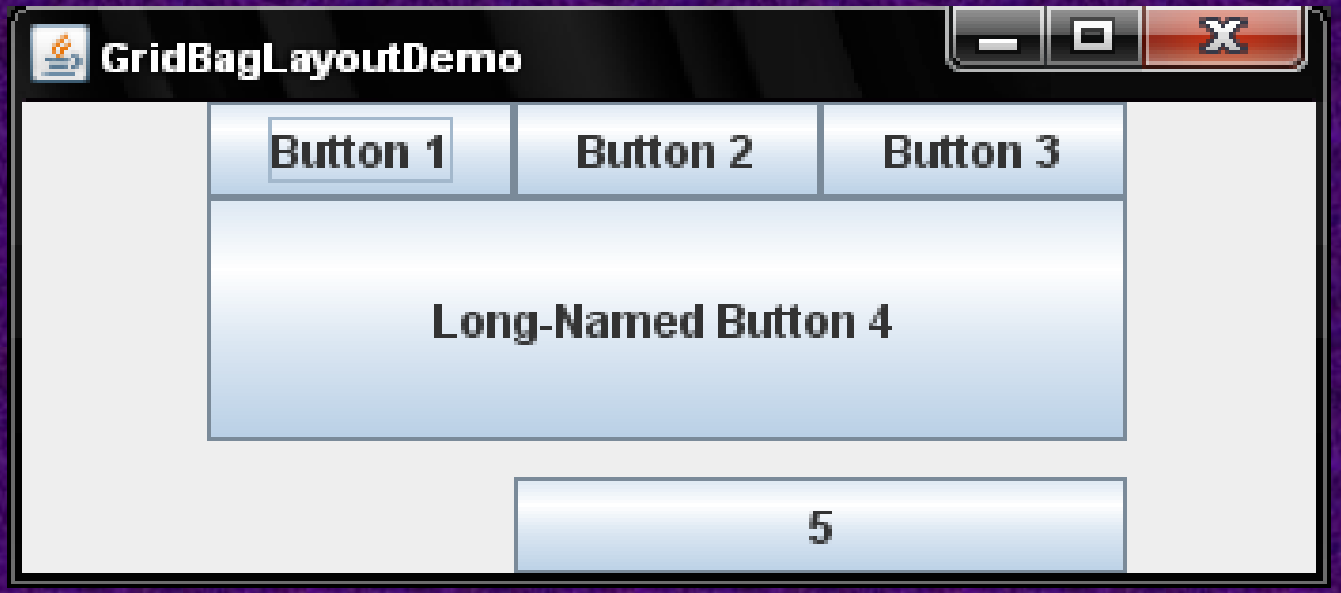


źródło:http://java.sun.com/docs/books/tutorial/uiswing/layout

Rozkład torebkowy (*GridBagLayout*)

```
button = new JButton("Button 1");
c.weightx = 0.0;

button = new JButton("Button 2");
c.weightx = 0.0;

button = new JButton("Button 3");
c.weightx = 0.0;
```

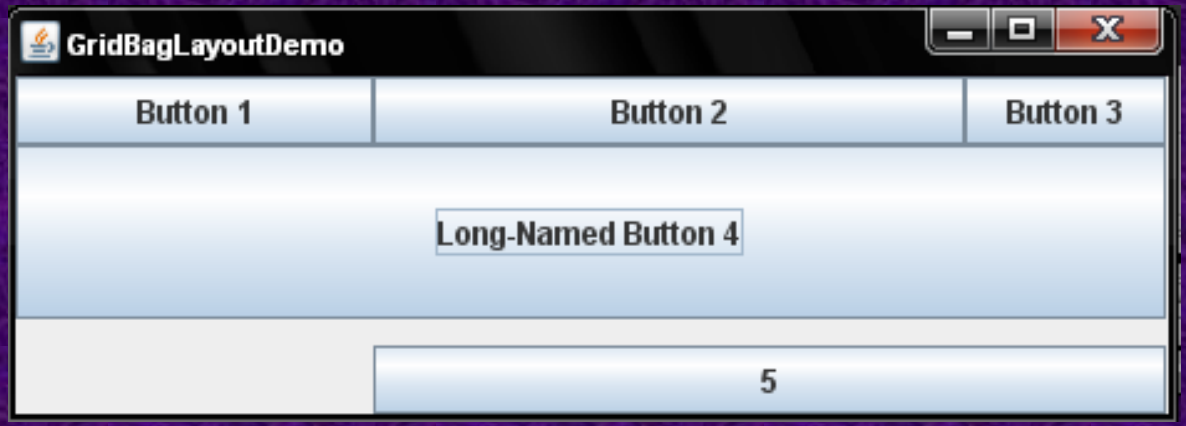
Komponenty na środku rozszerzają się do szerokości najszerszego.



Rozkład torebkowy (*GridBagLayout*)

```
button = new JButton("Button 1");
c.weightx = 0.2;

button = new JButton("Button 2");
c.weightx = 0.5;

button = new JButton("Button 3");
c.weightx = 0.0;
```

Komponenty wypełniają obszar proporcjonalnie. Ten z największym weightx zmienia rozmiar najbardziej.



Rozkład torebkowy (*GridBagLayout*)

Constraints

gridx, gridy – która komórka

gridwidth, gridheight – ile komórek w poziomie i pionie

fill (HORIZONTAL, VERTICAL, BOTH) – jak rozciągać komponent

ipadx, ipady – rozszerzenie komponentu w stosunku do wymiaru minimalnego

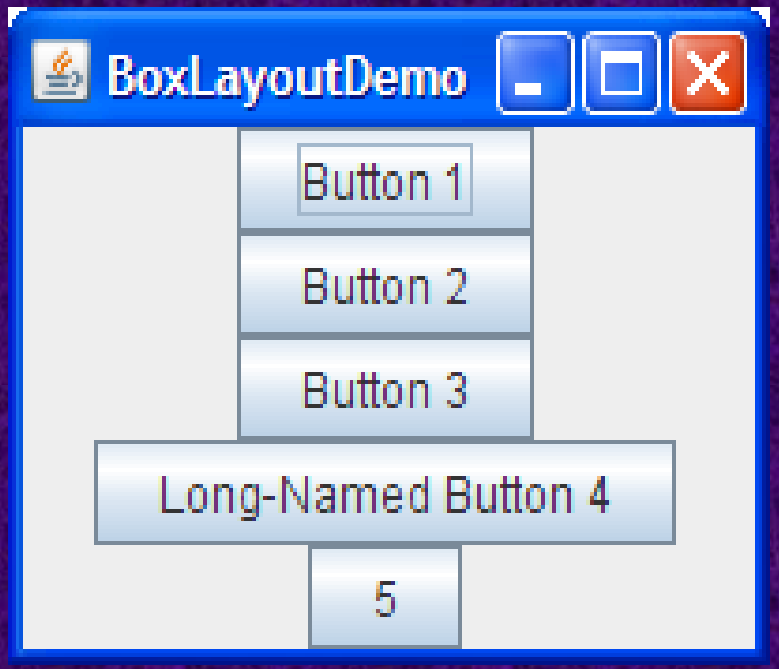
insets – odległości między komponentami w pionie i poziomie

anchor (PAGE_START, PAGE_END, LINE_START, LINE_END, FIRST_LINE_START, FIRST_LINE_END, LAST_LINE_END, and LAST_LINE_START)- położenie w komórce

weightx, weighty – sposób wypełnienia kolumn i wierszy

Rozkład pudełkowy (*BoxLayout*)

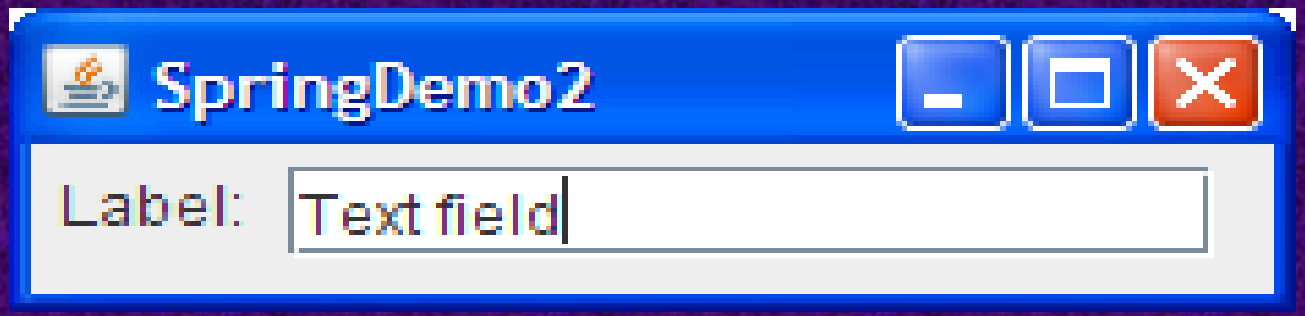
```
JPanel listPane = new JPanel();
listPane.setLayout(new BoxLayout(listPane,
   BoxLayout.PAGE_AXIS)); //BoxLayout.Y_AXIS
//PAGE i LINE dostosowuje do języków o różnych
   kierunkach
JLabel label = new Jlabel(labelText);
//skaluje komponenty: PreferredSize jeśli nie to inna
   wartość pomiędzy
    min i max, aby wypełnić
    przestrzeń w zalezności
    od kierunku układania -
    pion. Lub poziom
//Component.CENTER_ALIGNMENT ->
mogą być ustawiane w linii także
prawe lub lewe krawędzie
źródło:http:
//java.sun.com/docs/books/tutorial
    /uiswing/layout
```



Rozkład wiosenny (*SpringLayout*)

```
Container contentPane = frame.getContentPane();
SpringLayout layout = new SpringLayout();
contentPane.setLayout(layout);
layout.putConstraint(SpringLayout.WEST, label,
   5,SpringLayout.WEST, contentPane);
layout.putConstraint(SpringLayout.WEST, textField,
   5,SpringLayout.EAST, label);
```

źródło:http://java.sun.com/docs/books/tutorial/uiswing/layout



Rozkład grupowy (*GroupLayout*)

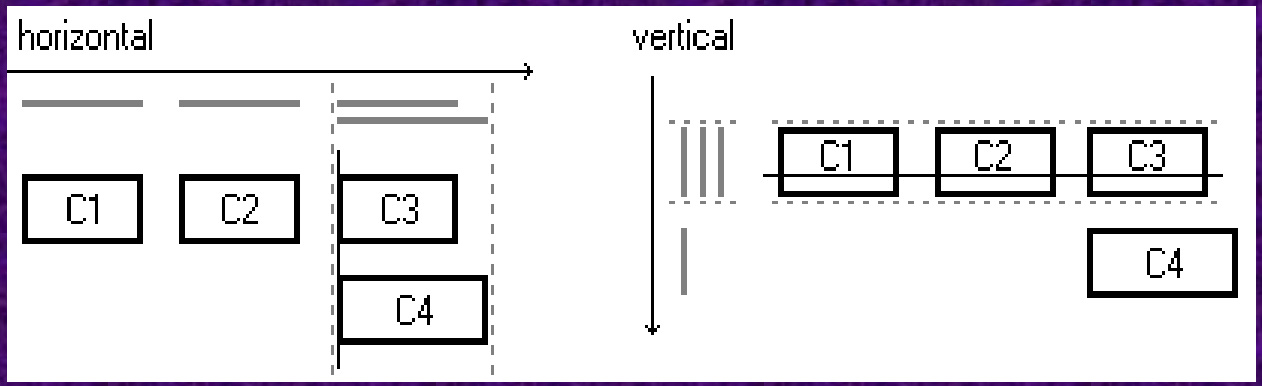
- Stworzony dla narzędzi do automatycznego tworzenia GUI (np. Matisse w NetBeans)
- Oddzielne zarządzanie układem w pionie i poziomie
- Dwa typy ułożenia sekwencyjny (jak FlowLayout) i równoległy (do linii) . Zwykle różne dla obu osi – wtedy komponenty się nie nakładają

Rozkład grupowy (*GroupLayout*)

```
GroupLayout layout = new GroupLayout(panel);
panel.setLayout(layout);
layout.setAutoCreateGaps(true); //między
                                //komponentami
layout.setAutoCreateContainerGaps(true);
          //między komponentami i kontenerem
layout.setHorizontalGroup(
   layout.createSequentialGroup()
      .addComponent(c1)
      .addComponent(c2)
      .addGroup(layout.createParallelGroup(
            GroupLayout.Alignment.LEADING)
         .addComponent(c3)
         .addComponent(c4)
      )
);
```

Rozkład grupowy (*GroupLayout*)

```
layout.setVerticalGroup(
    layout.createSequentialGroup()
   .addGroup(layout.createParallelGroup(
       GroupLayout.Alignment.BASELINE)
             .addComponent(c1)
             .addComponent(c2)
             .addComponent(c3))
       .addComponent(c4)
);
layout.linkSize(SwingConstants.HORIZONTAL, c3, c4);
//taki sam rozmiar
```



http://java.sun.com/docs/books/tutorial/uiswing/layout/group.html

Rezygnacja z zarządcy układu

```
JButton b1 = new JButton("Zatwierdź");
JButton b2 = new JButton("Anuluj");

panel.add(b1);
panel.add(b2);

Insets insets = panel.getInsets(); //wstawki -
                                   //marginesy
  //odległości od krawędzi komponentu, np.
  //grubość ramki, tytuł
Dimension size = b1.getPreferredSize();
b1.setBounds(20 + insets.left, 0 + insets.top,
             size.width, size.height);

size = b2.getPreferredSize();
b2.setBounds(50 + insets.left, 35 + insets.top,
             size.width, size.height);
```

Specjalny obiekt – ikona

- Ikona - mały obiekt graficzny (zwykle GIF)
- Wczytanie ikony z pliku – stworzenie obiektu klasy *ImageIcon*
- Przykład

```
ImageIcon rys = new ImageIcon("open.gif");
JButton p = new JButton(rys);
Container kont = getContentPane();
kont.setLayout(new
FlowLayout(FlowLayout.CENTER));
kont.add(p1);
```

Etykieta (*JLabel*)

- **Komponent wyświetlający łańcuch znaków lub ikonę (bez możliwości edycji)**
- **Konstruktory**
  - `JLabel()`
  - `JLabel(String napis)`
  - `JLabel(String napis, int wyrównanie)`
  - `JLabel(Icon obrazek)`
  - `JLabel(Icon obrazek, int wyrównanie)`
  - `JLabel(String napis, Icon obrazek, int wyrównanie)`
- `Sposoby wyrównania etykiet np.`
  - `Label.LEFT, Label.RIGHT, Label.CENTER`

Aplikacja *Etykiety*


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

public class TestEtykiet extends JFrame{

    TestEtykiet(){
        JLabel e1, e2, e3;
    ImageIcon left = new ImageIcon("d:/lewy.gif");
    ImageIcon right = new ImageIcon("d:/prawy.gif");
    setLayout(new GridLayout(3,1));
    e1 = new JLabel("Lewy", left, JLabel.LEFT);
    e2 = new JLabel("Prawy", right, JLabel.RIGHT);
    e3 = new JLabel("Środek", JLabel.CENTER);
    add(e1); add(e2); add(e3);
    }
    public static void main(String[] args) {
        TestEtykiet te = new TestEtykiet();
        te.pack();
        te.setVisible(true);
    }
}
```

Przycisk (*JButton*)

- **Komponent, którego naciśnięcie może spowodować wykonanie pewnych czynności**
- **Niektóre konstruktory**
  - `JButton()`
  - `JButton(String napis)`
  - `JButton(Icon obrazek)`
  - `JButton(String napis, Icon obrazek)`

Inne przyciski

- **JRadioButton – przycisk radiowy (grupa przycisków z których tylko jeden może być wciśnięty)**

```
JRadioButton b1 = new JRadioButton(„Opcja 1”);
b1.setActionCommand(nazwa przycisku 1);
//można nazwać akcję „nazwa przycisku 1” i zprawdzać
  tą nazwę po zajściu zdarzenia (nie na podstawie
  tekstu przycisku) i stosować wersje wielojęzyczne
  o różnym tekście wyświetlanym
b1.setSelected(true);
JRadioButton b2 = new JRadioButton(„Opcja 2”);
b2.setActionCommand(nazwa przycisku 2);
ButtonGroup grupa = new ButtonGroup();
group.add(b1);
group.add(b2);
```

- **JToggleButton – przycisk który pozostaje wciśnięty lub nie**

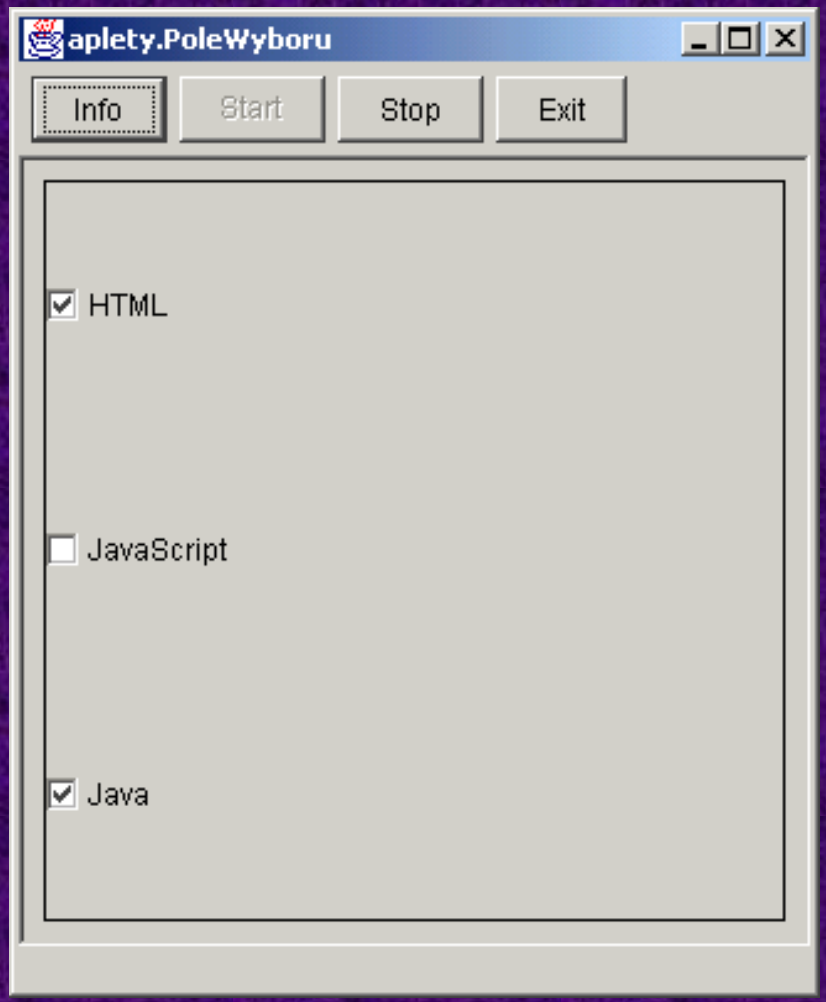
Pole wyboru (*JCheckBox*)

- **Wybór z możliwością zaznaczenia kilku pól**
- **Klasa pochodna po *JAbstractButton***
- **Konstruktory – jak w klasie *JButton*, dodatkowy argument *boolean* zaznacza/odznacza pole**
- **Metoda *setSelected(boolean)* – zaznacza/odznacza pole**

Pole wyboru - przykład

```
JCheckBox
   cb1 = new JCheckBox("HTML"),
   cb2 = new JCheckBox("JavaScript"),
   cb3 = new JCheckBox("Java", true);
Container k = getContentPane();
k.setLayout(new GridLayout(0,1));
   //proporcjonalnie
k.add(cb1); k.add(cb2);
k.add(cb3);
cb1.setSelected(true);
```

Pole wyboru - przykład

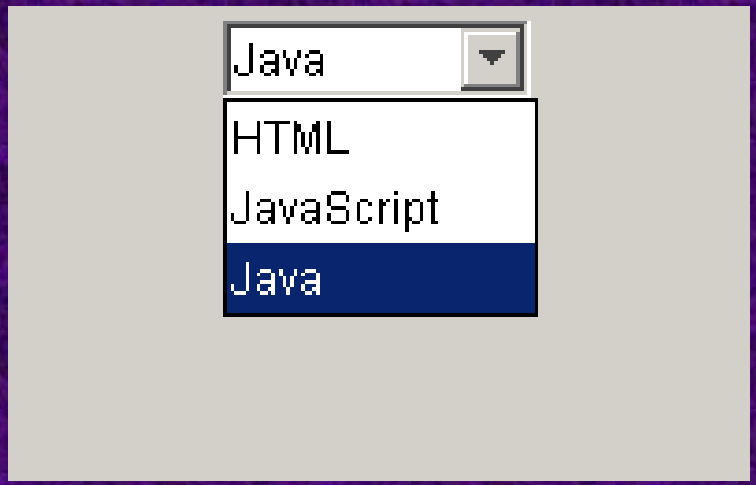


Lista rozwijana (*JComboBox*)

- **Lista rozwijana przy aktywacji**
- **Można wybrać tylko jeden element z listy**
- **Elementy wyświetlane – np. teksty, ikony**
- **Zaznaczanie elementu – metoda *setSelectedIndex(int)***

Lista rozwijana - przykład

```
String[] tab = {"HTML", "JavaScript",
  "Java"};
JComboBox cmb = new JComboBox(tab);
Container k = getContentPane();
k.setLayout(new FlowLayout());
k.add(cmb);
cmb.setSelectedIndex(2);
```

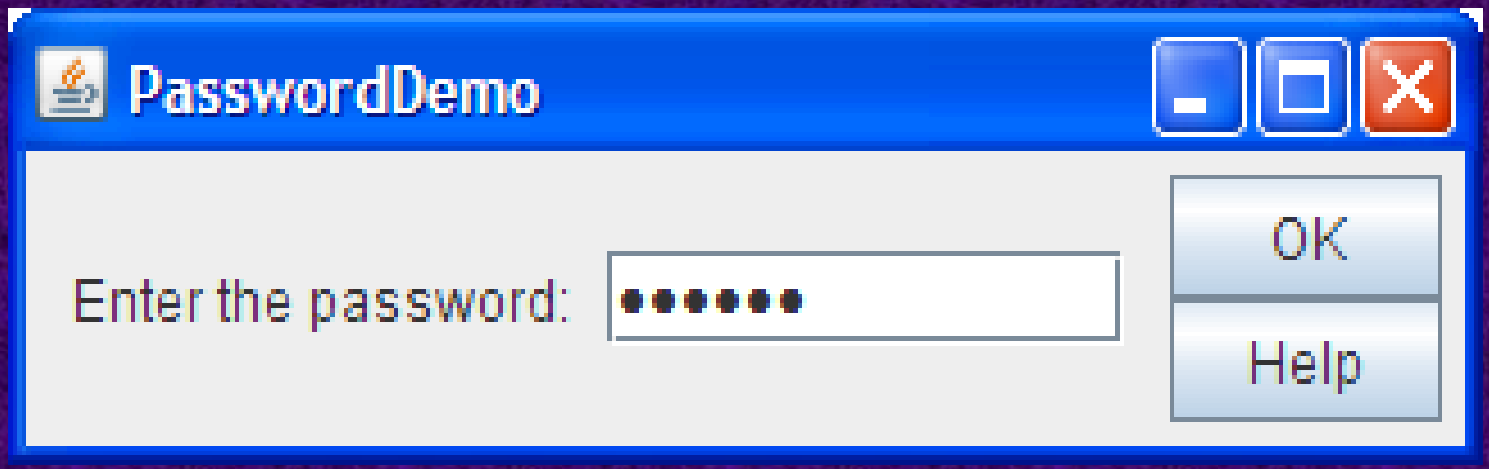


Pole tekstowe (*JTextField*)

- Jednowierszowe pole tekstowe z możliwością edycji
- Niektóre konstruktory
  - `JTextField(String napis)`
  - `JTextField(String napis, int il_kolumn)`
  - `JTextField(il_kolumn)`

Pole *JPasswordField*

- **Do tworzenia pól tekstowych z zamaskowanym napisem**
- **Konstruktory - jak w klasie *JTextField***
- ***public void setEchoChar(char c)***



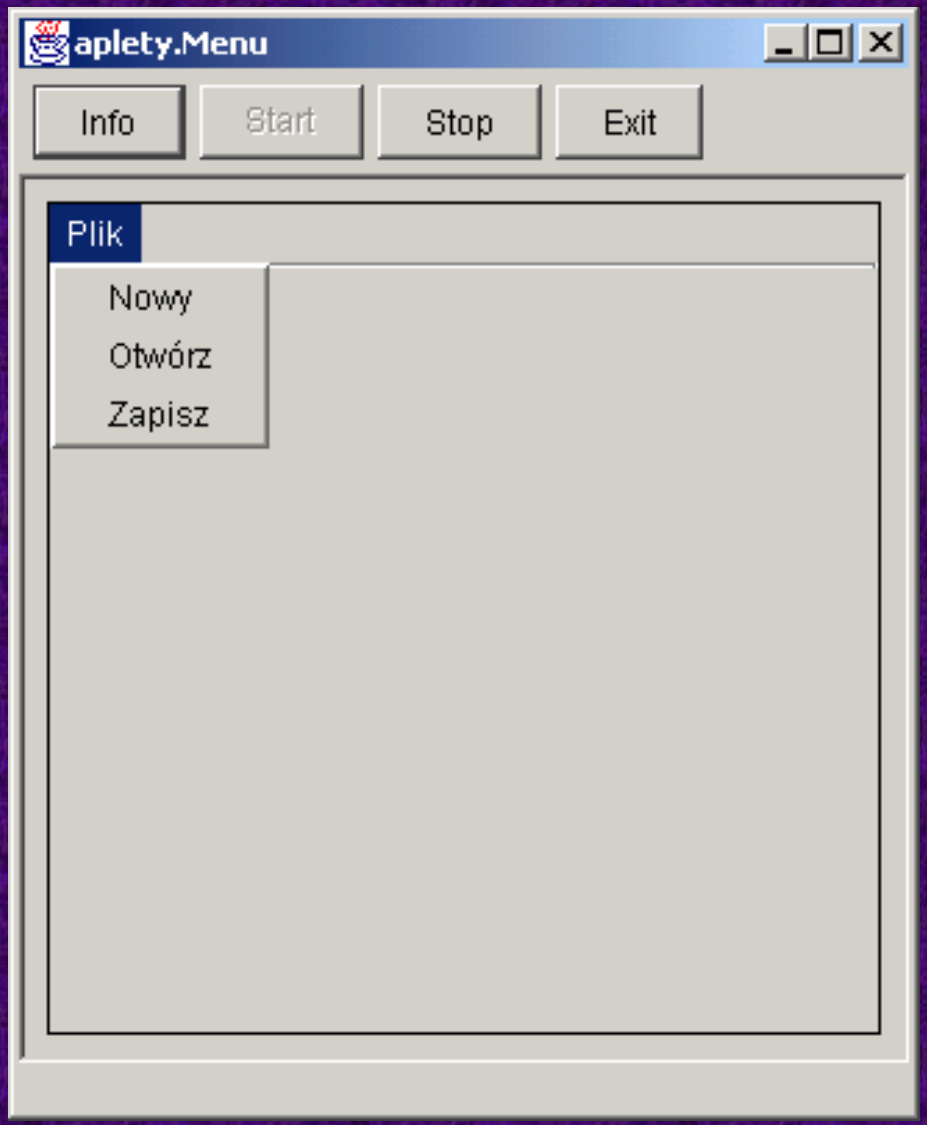
Tworzenie menu

- **Pasek menu – *JMenuBar***
- **Komponenty zdolne do wyświetlania menu - zawierają metodę *setJMenuBar()***
- **Menu rozwijane, umieszczane na pasku - *JMenu***
- **Element menu rozwijanego – *JMenuItem***

Tworzenie menu - przykład

```
JMenuBar pasek = new JMenuBar();

JMenu menu1 = new JMenu("Plik");

JMenuItem[] elem = {
     new JMenuItem("Nowy"),
     new JMenuItem("Otwórz"),
     new JMenuItem("Zapisz")};

setJMenuBar(pasek); pasek.add(menu1);

for(int i=0; i<elem.length; i++)
    menu1.add(elem[i]);
```

Tworzenie menu - przykład



Klasa *JMenuItem*

- **Pochodna po *JAbstractButton*; tworzenie elementów menu – jak przycisków, np.**

```
ImageIcon rys = new ImageIcon("open.gif");

JMenuItem[] elem = {
  new JMenuItem("Nowy"),
  new JMenuItem("Otwórz", rys),
  new JMenuItem("Zapisz")
  };
```

Inne klasy menu

- Klasy dziedziczące po *JMenuItem*
  - *JMenu* – tworzy menu kaskadowe
  - *JCheckBoxMenuItem* – dodaje pola wyboru
  - *JRadioButtonMenuItem* – dodaje przyciski wyboru

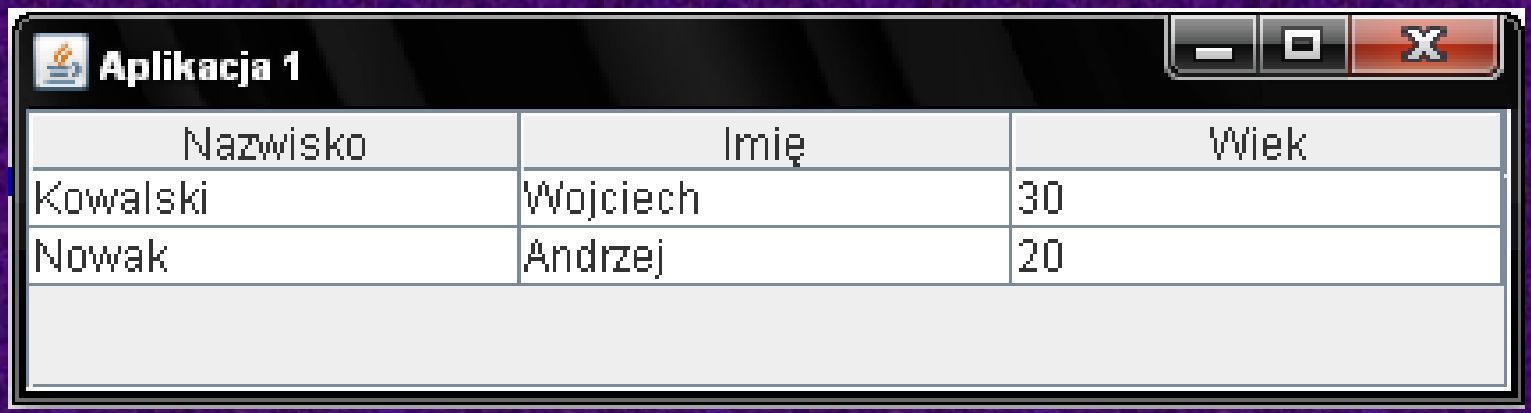
```
menuItem = new JMenuItem("Tekst i ikonka",
                         new ImageIcon("ikonka.gif"));
```

- Specjalne menu – tzw. kontekstowe (*JPopupMenu*), niewidzialne dopóki użytkownik nie wykona pewnej czynności (tworzenie – jak *JMenuBar*)

```
popup = new JPopupMenu();
menuItem = new JMenuItem("A popup menu
  item");
menuItem.addActionListener(this);
popup.add(menuItem);
```

Komponent JTable

```
String[] nazwyKolumn = {"Nazwisko","Imię","Wiek"};
Object[][] dane = {{"Kowalski", "Wojciech",new
                                    Integer(30)},
                  {"Nowak", "Andrzej", new
                                    Integer(20)}};
JTable tabelka = new JTable(dane, nazwyKolumn);
JScrollPane scrollPane = new JScrollPane(tabelka);
this.add(scrollPane);
```



Model komponentu JTable

```
class ModelDanych extends AbstractTableModel {
    private String[] nazwyKolumn = ...
    private Object[][] dane = ...

    public int getColumnCount() {
        return nazwyKolum.length;
    }
    public int getRowCount() {
        return dane.length;
    }
    public String getColumnName(int kolumna) {
        return nazwyKolumn[kolumna];
    }
```

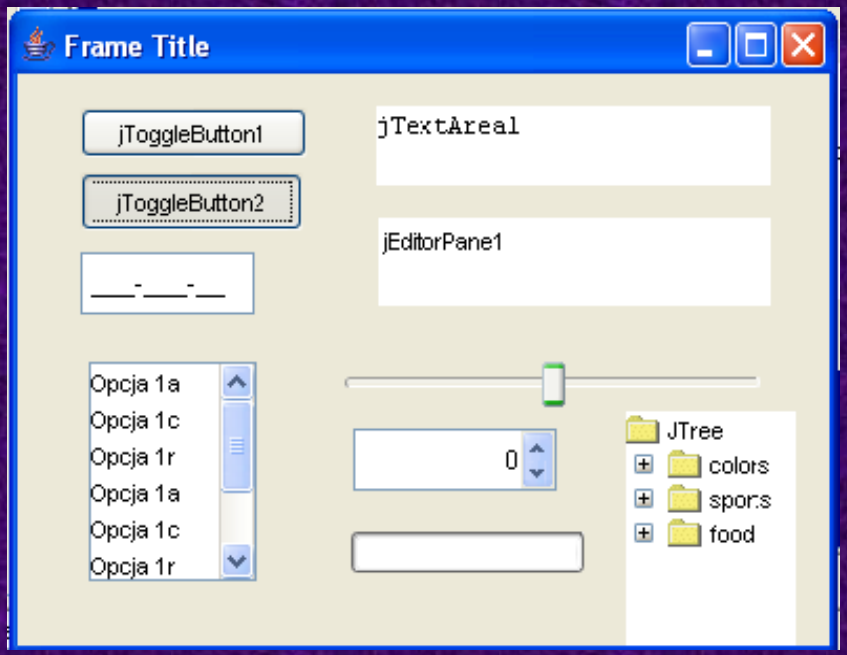
Model komponentu JTable c.d.

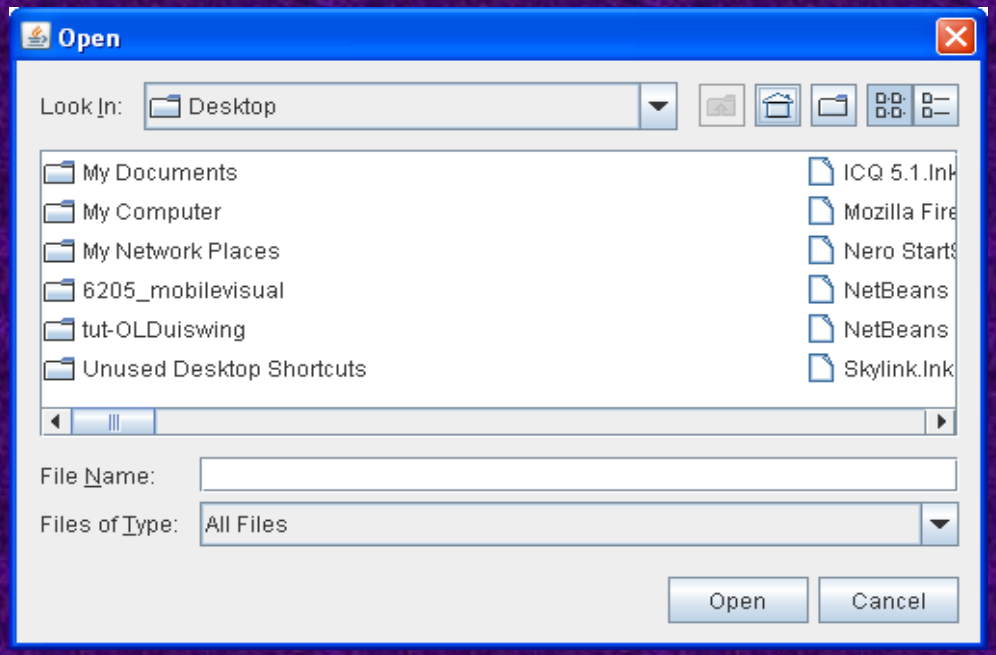
```
public Object getValueAt(int wiersz, int kolumna) {
    return dane[wiersz][kolumna];
}

public Class getColumnClass(int c) {
    return getValueAt(0, c).getClass();
}

public void setValueAt(Object obiekt, int wiersz,
                                      int kolumna) {
        data[wiersz][kolumna] = obiekt;
        fireTableCellUpdated(wiersz, kolumna);
}
Table tabelka = new JTable(new ModelDanych());
```

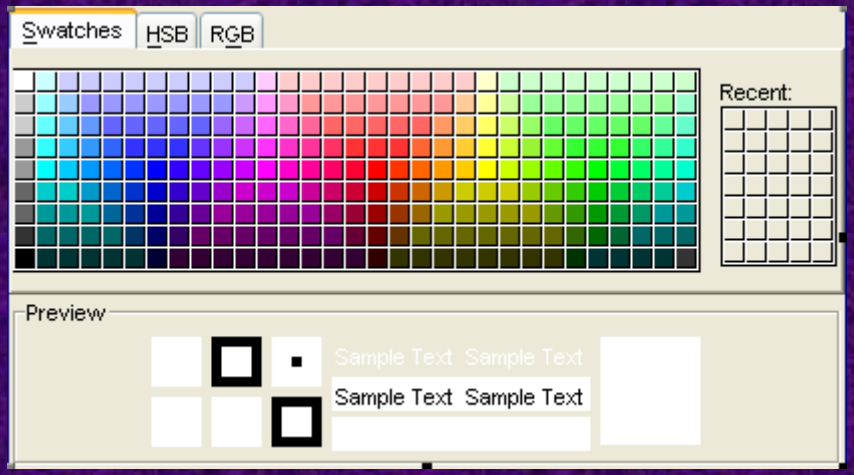
Inne komponenty Swingu

- *JToggleButton* – przycisk z dwoma stanami (wciśnięty/zwolniony)
- *JList* – lista pojedynczego/wielokrotnego wyboru
- *JSpinner* - pole umożliwiające zmianę wartości w górę lub w dół
- *JTextArea* – wielowierszowe pole tekstowe
- *JTextEditor* – umożliwia formatowanie tekstu
- *JSlider* - suwak
- *JProgressBar* – pasek postępu
- *JScrollBar* – pasek przewijania
- *JTable* - tablica
- *JTree* – drzewo (np. katalogów)
- *JColorChooser* – paleta kolorów
- *JFileChooser* – lista plików

Inne komponenty Swingu



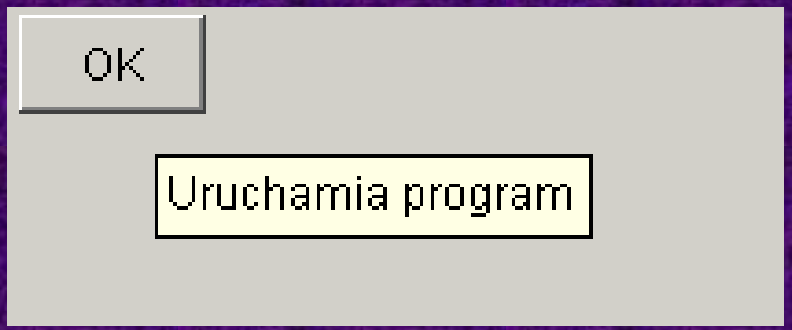




Podpowiedzi

- **Dodawanie podpowiedzi do komponentu GUI – metoda *setToolTipText(String)* klasy *JComponent***
- **Przykład**

```
JButton p = new JButton("OK");
p.setToolTipText("Uruchamia program");
```



Kontenery Swingu

- **Kontenery podstawowe**
  - **aplet (*JApplet*)**
  - **ramka (*JFrame*)**
  - **okno komunikatów (*JOptionPane*)**

Kontenery Swingu

- **Kontenery ogólnego przeznaczenia**
  - **panel (*JPanel*)**
  - **panele specjalne (*JScrollPane*, *JSplitPane*, *JTabbedPane*)**
  - **okno dialogowe (*JDialog*)**
  - **pasek narzędziowy (*JToolbar*)**

Kontenery Swingu

- **Kontenery specjalnego przeznaczenia, np.**
  - ***JDesktopPane***
  - ***JInternalFrame***
  - ***JRootPane***
  - ***JLayeredPane***

Ramka (*JFrame*)

- **Osobne okno specyficzne dla platformy, z wyraźnym obramowaniem, tytułem, przyciskiem zamykania itp.**
- **Domyślny zarządca rozkładu - *BorderLayout***
- **Konstruktory**
  - `JFrame()`
  - `JFrame(String napis)`

Ramka (*JFrame*)

- **Domyślnie – ramka niewidzialna**
- **Uczynienie ramki widzialną – metoda *setVisible(true)* lub *show()***
- **Ustalenie rozmiaru ramki**
  - **metoda *pack()* –najmniejszy możliwy rozmiar**
  - **metoda *setSize(int szer, int wys)* – nie zalecana**

Aplet *Ramka*

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

public class Ramka extends JApplet {

  public void init() {

    JFrame win = new JFrame("Moje okno");
    JLabel etykieta = new JLabel("Java");
    win.getContentPane().add(etykieta);
    win.pack();
    win.setVisible(true);

  }

}
```

Okno dialogowe (*JOptionPane*)

- **Okno wyświetlające pewne informacje, umieszczane w ramce (blokuje używanie innych okien)**
- **Rodzaje okien dialogowych**
  - ***ConfirmDialog* – okno potwierdzenia**
  - ***InputDialog* – okno wprowadzania tekstu**
  - ***MessageDialog* – komunikat**
  - ***OptionDialog* – okno z wyborem opcji**

Okno dialogowe (*JOptionPane*)

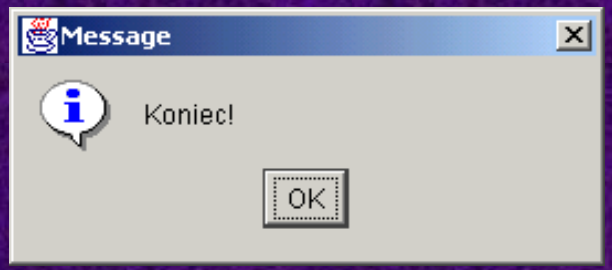
- **Tworzenie okna dialogowego – odpowiednia metoda**

  ```
  JOptionPane.showXxDialog(Component,
  String);
  ```

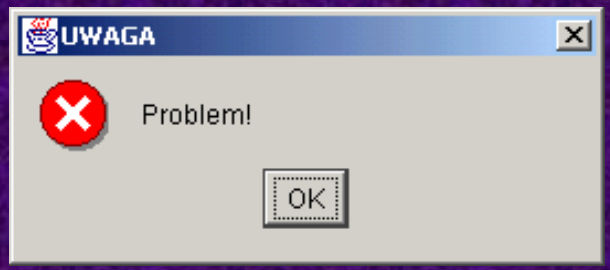
- **Metody klasy *JOptionPane* – przeciążone, możliwość zmiany tytułu, napisu na przycisku, ikony itp.**

Przykłady okien komunikatów

```
JOptionPane.showMessageDialog(win,
  "Koniec!");
```

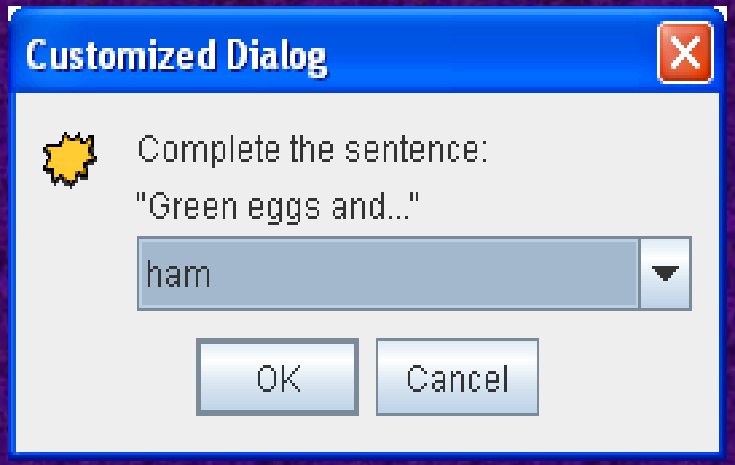



```
JOptionPane.showMessageDialog(win,
  "Problem!", "UWAGA",
  JOptionPane.ERROR_MESSAGE);
```



Przykłady okien komunikatów

```
Object[] possibilities = {"ham", "spam", "yam"};
String s = (String)JOptionPane.showInputDialog(
                    frame,
                    "Complete the sentence:\n"
                    + "\"Green eggs and...\"",
                    "Customized Dialog",
                    JOptionPane.PLAIN_MESSAGE,
                    icon,
                    possibilities,
                    "ham");
```



http://java.sun.com/docs/books/tutorial/uiswing/components/dialog.html#input

Panel (*JPanel*)

- **Najprostszy kontener, domyślnie nieprzezroczysty**
- **Umożliwia rysowanie**
- **Domyślny zarządca rozkładu - *FlowLayout***

Panel (*JPanel*)

- **Ustalenie, że panel ma być kontenerem pośrednim – metoda *setContentPane()* kontenera podstawowego, np.**

```
JPanel panel = new JPanel();
contentPane.setLayout(new BorderLayout());
contentPane.setBorder(obramowanie);
contentPane.add(etykieta);
JApplet.setContentPane(panel);
```

Aplikacja *AplikSwing*

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

public class AplikSwing {

 public static void main(String[] args){
   JFrame ramka = new JFrame("Swing!!!");
   ramka.setLayout(new GridLayout(0,1));
   ramka.add(new JLabel("Program w Swingu"));
   ramka.add(new JButton("OK"));
   ramka.pack();
   ramka.setVisible(true);
 }

}
```

Zmodyfikowana aplikacja *AplikSwing*

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

public class AplikSwing extends JFrame {

 AplikSwing(String tytul) {
   super(tytul);
   Container k = getContentPane();
   k.setLayout(new GridLayout(0,1));
   k.add(new JLabel("Program w Swingu"));
   k.add(new JButton("OK"));
 }
 public static void main(String[] args) {
   AplikSwing ramka = new
              AplikSwing("Swing!!!");
   ramka.pack();
   ramka.setVisible(true);
  }
}
```