GRAFICZNY INTERFEJS UŻYTKOWNIKA

Programowanie obiektowe II

Dr inż. Wojciech Kozioł

Uniwersytet Rzeszowski

GUI – graphical user interface

W większości języków programowania dostępne są biblioteki graficznego interfejsu użytkownika. Pozwalają one użytkownikowi na wchodzenie w interakcje z aplikacją poprzez wykonywanie prostych czynności tj.: kliknięcie myszą na przycisk, wybór opcji z listy lub elementu combo, wprowadzenie danych do pola tekstowego, otwieranie, zamykanie, zamianę rozmiaru okna itp.

Kiedy użytkownik wykona jakąś czynność to generowane jest odpowiednie zdarzenie, które jest odpowiednio obsługiwane przez aplikację, na której pracuje użytkownik.

System jest w stanie zareagować na określone zdarzenia gdyż cały czas na nie oczekuje (nasłuchuje) i wie jakie zadania ma wykonać gdy ono nastąpi np. po wprowadzeniu tekstu do pola tekstowego sprawdź czy jest on liczbą a następnie wstaw go do listy, w przeciwnym razie wygeneruj komunikat o niepoprawnym formacie danych i poproś użytkownika o ponowne wprowadzenie poprawnych danych.

GUI w JAVIE

Początkowo celem GUI w Javie (Java 1.0) było dostaczenie programistom interfejsu, który miał cechować się dobrym wyglądem na każdej platformie. Nie udało się jednak osiągnąć tego celu.

Zamiast tego wykonano AWT (Abstract Window Toolkit), które wyglądało równie źle na wszystkich systemach operacyjnych. Narzucało ono różne ograniczenia np. brak dostępu do bardziej wyszukanych elementów GUI dostarczanych przez systemy operacyjne czy możliwości obsługi jedynie czterech typów czcionek.

AWT przeszło kilka transformacji, które zakończyły się powstaniem biblioteki Swing, która jest częścią JFC (Java Fundation Classes). Swing jest bogatym zbiorem komponentów, które są zrozumiałe i łatwe w użyciu.

Biblioteka SWING – tworzenie formatki

Większość aplikacji SWING bazować będzie na klasie JFrame. Klasa ta tworzy okno aplikacji w danym systemie operacyjnym, które nazywane jest „formatką”.

Zacznijmy od prostego przykładu:

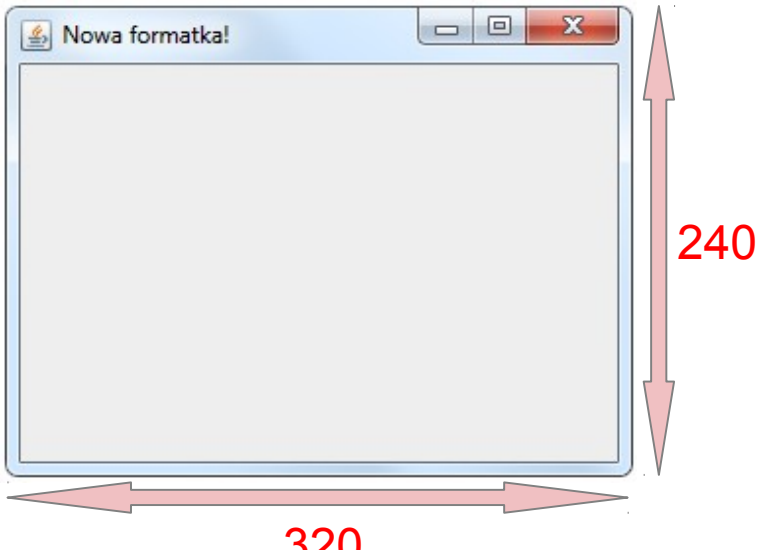
```
import javax.swing.*;

public class Dydaktyka {

    public static void main(String[] args) {

        JFrame frame = new JFrame("Nowa formatka!");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setBounds(100, 100, 320, 240);
        frame.setVisible(true);

    }
}
```

Utworzenie instancji obiektu formatki z konstruktorem, który w parametrze ustawia tekst tytułu formatki.

Określenie działania, które jest wykonywane podczas zamykania formatki.

Określenie położenia (1 i 2 argument) oraz rozmiaru (3 i 4 argument) formatki

Uwidocznienie formatki



Zamykanie formatki

Możemy zdefiniować cztery typy zachowań dla JFrame, które określają co się stanie podczas zamknięcia formatki. Sposób zamknięcia określamy poprzez wywołanie metody setDefaultCloseOperation z parametrem wyznaczonym przez jedną z powyższych opcji.

- DO_NOTHING_ON_CLOSE - nie podejmuj żadnego działania kiedy użytkownik zamyka formatkę.
- HIDE_ON_CLOSE – ukryj formatkę kiedy użytkownik ją zamyka. To jest domyślny rodzaj zachowania. Formatka jest niewidoczna ale program wciąż jest uruchomiony.
- DISPOSE_ON_CLOSE – zamknij formatkę i zniszcz ją podczas jej zamknięcia. Niszczenie polega na usunięciu obiektu formatki z pamięci.
- EXIT_ON_CLOSE – wyjdź z aplikacji. Ta opcja opuszcza aplikację.

Zmiana położenia i rozmiaru formatki i kontrolek

Podczas pracy z formatką niezbędne jest określenie jej rozmiaru i położenia. Dotyczy to również położenia innych elementów w formatce jeśli nie chcemy używać specjalnych menadżerów ułożeń komponentów (kontrolek).

Położenie elementów GUI definiujemy przy użyciu metody **setLocation**, która występuje w dwóch przeciążeniach:

setLoaction(int x, int y) – określenie położenia poprzez podanie współrzędnych na ekranie,

przykład użycia: `frame.setLocation(100,100);`

setLoaction(Point p) – określenie położenia poprzez wykorzystanie obiektu typu Point,

przykład użycia: `button1.setLocation(new Point(100,100));`

Rozmiar elementów GUI definiujemy określając ich szerokość (ang. width) i wysokość (ang. height) definiowane liczbą pikseli. Wykorzystywana jest do tego metoda **setSize**. Podobnie jak w przypadku ustawiania położenia istnieją dwa przeciążenia tej metody:

setSize(int width, int height) – określenie rozmiaru poprzez podanie długości i szerokości elementu,

przykład użycia: `button1.setSize(100,100);`

setSize(Dimension d) – określenie położenia poprzez wykorzystanie obiektu typu Dimension,

przykład użycia: `frame.setSize(new Dimension(100,100));`

Zmiana położenia i rozmiaru formatki i kontrolek

Możliwa jest również jednoczesna zmiana położenia i rozmiaru elementów GIU. Służy do tego metoda **setBounds**. Również i w tym przypadku istnieją dwa przeciążenia tej metody:

setBounds(int x, int y, int width, int height) – określenie położenia poprzez podanie współrzędnych (x, y) oraz rozmiaru (width, height),

przykład użycia: `frame.setBounds(100,100, 200, 320);`

setBounds(int x, int y, int width, int height) – określenie położenia i rozmiaru poprzez wykorzystanie obiektu typu Rectangle,

przykład użycia: `button1.setBounds(new Rectangle(100, 100, 320, 240));`

Klasy Point, Dimension i Rectangle należą do biblioteki AWT, należy zatem pamiętać aby przed ich użyciem dołączyć odpowiednią bibliotekę: `import java.awt.*;`

Kontenery na przykładzie JPanel

Panel jest kontenerem dla innych komponentów. Można dla niego ustawić menadżer ułożenia, obramowanie, oraz kolor tła. JPanel służy do grupowania komponentów.

Konstruktory klasy JPanel:

JPanel() – tworzy JPanel z domyślnym menedżerem układu FlowLayout i z domyślnie ustawioną flagą podwójnego buforowania (double buffeting).

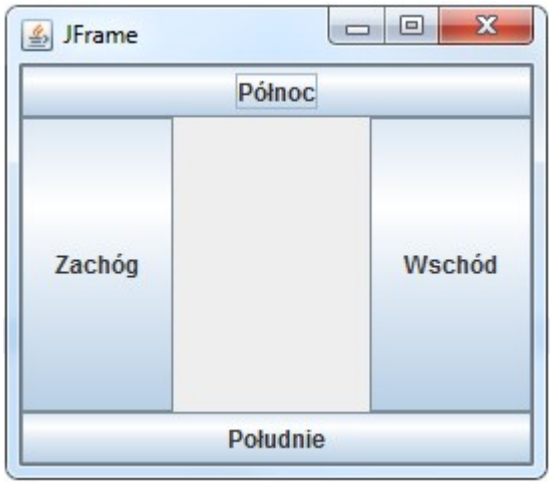
JPanel(boolean isDoubleBuffered) – tworzy JPanel z domyślnym menedżerem układu FlowLayout i określonym w argumencie stanem flagi podwójnego buforowania.

JPanel(LayoutManager layout) – tworzy JPanel z określonym w parametrze menadżerem ułożeń i ustawioną domyślnie flagą podwójnego buforowania.

JPanel(LayoutManager layout, boolean isDoubleBuffered) – tworzy JPanel z określonymi w parametrach menadżerem ułożenia i flagą podwójnego buforowania.

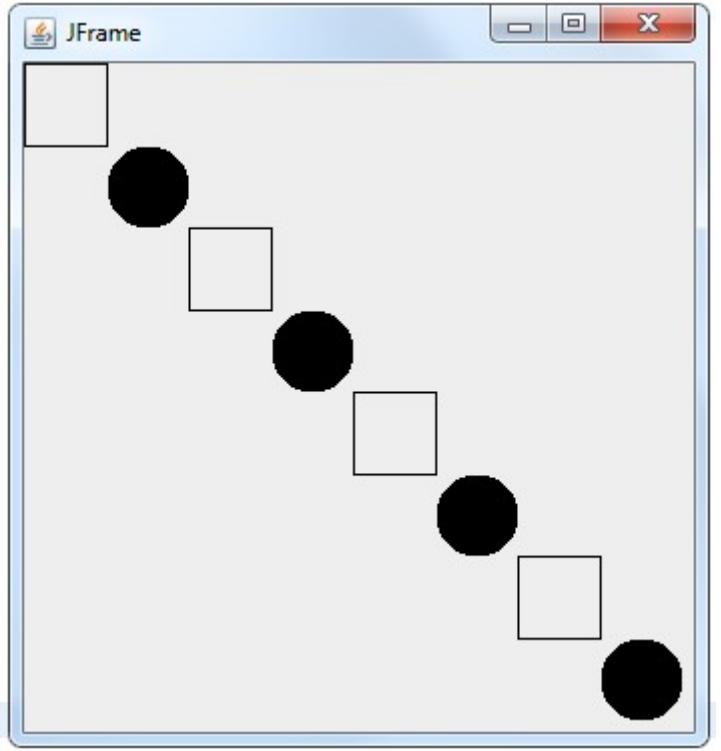
JPanel - przykład

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

public class Dydaktyka {

    public static void main(String[] args) {
        JFrame frame = new JFrame("JFrame");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JPanel buttonPanel = new JPanel(new BorderLayout());
        JButton northButton = new JButton("Północ");
        JButton southButton = new JButton("Południe");
        JButton eastButton  = new JButton("Wschód");
        JButton  westButton = new JButton("Zachóg");

        buttonPanel.add(northButton, BorderLayout.NORTH);
        buttonPanel.add(southButton, BorderLayout.SOUTH);
        buttonPanel.add(eastButton,  BorderLayout.EAST);
        buttonPanel.add(westButton,  BorderLayout.WEST);

        frame.add(buttonPanel,  BorderLayout.CENTER);
        frame.pack();
        frame.setVisible(true);
    }
}
```



Rysowanie w JPanel - przykład

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

public class Dydaktyka extends JPanel {

    public void paintComponent(Graphics g) {
        int width = (int) getWidth() / 8;
        int height = (int) getHeight() / 8;
        int x = 0;
        int y = 0;

        g.setColor(Color.black);

        for (int i = 0; i < 4; i++) {

            g.drawRect(x, y, width, height);
            x += width; y += height;
            g.fillOval(x, y, width, height);
            x += width; y += height;
        }
    }

    public static void main(String[] args) {
        JFrame frame = new JFrame("JFrame");

        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.add(new Dydaktyka());
        frame.setSize(300, 200);
        frame.setVisible(true);

        frame.pack();
        frame.setVisible(true);
    }
}
```



Menadżery ułożenia

Menadżer ułożenia oblicza cztery właściwości dla położenia (x,y) i rozmiaru (width, height) wszystkich komponentów w danym kontenerze.

Lista najczęściej używanych menadżerów ułożeń:

- FlowLayout
- BorderLayout
- CardLayout
- BoxLayout
- GridLayout
- GridBagLayout
- GroupLayout
- SpringLayout

Menadżer ułożenia BorderLayout

Menadżer **BorderLayout** jest domyślnym menadżerem ułożeń, używają go domyślnie formatki JFrame i kontenery. Użyty bez żadnych dodatkowych instrukcji powoduje, że wszystko, co zostanie dodane przez metodę **add**, będzie umieszczone w środku obszaru i rozciągnięte we wszystkich kierunkach, aż do krawędzi formatki.

Ten menadżer ułożenia działa w oparciu o cztery rejony brzegowe i obszar środkowy. Dodając coś do kontenera, można użyć przeciążonej metody **add**, która jako drugi parametr przyjmuje jedną ze stałych:

BorderLayout.NORTH – północ (góra),

BorderLayout.SOUTH – południe (dół),

BorderLayout.EAST – wschód (prawo),

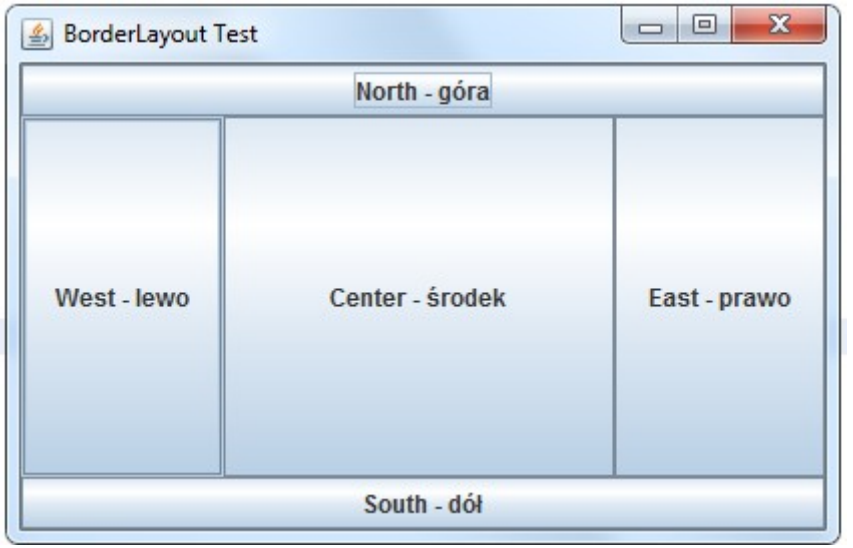
BorderLayout.WEST – zachód (lewo),

BorderLayout.CENTER – wypełni środek rozciągając daną kontrolkę do innych komponentów lub krawędzi formatki.

BorderLayout – przykład użycia

```
package menadzerulozenia;

import java.awt.BorderLayout;
import java.awt.Container;
import javax.swing.JButton;
import javax.swing.JFrame;

public class MenadzerUlozenia {

    public static void main(String[] args) {
        JFrame frame = new JFrame("BorderLayout Test");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        Container container = frame.getContentPane();

        // Dodananie przycisków do każdego z pięciu obszarów BorderLayout
        container.add(new JButton("North - góra"), BorderLayout.NORTH);
        container.add(new JButton("South - dół"), BorderLayout.SOUTH);
        container.add(new JButton("East - prawo"), BorderLayout.EAST);
        container.add(new JButton("West - lewo"), BorderLayout.WEST);
        container.add(new JButton("Center - środek"), BorderLayout.CENTER);

        frame.pack();
        frame.setVisible(true);
    }
}
```

Dołączenie menadżera BorderLayout z biblioteki AWT



Menadżer ułożenia FlowLayout

Ten układ po prostu „wypełnia” formatkę komponentami od prawej do lewej, dopóki przestrzeń na górze nie zostanie zapełniona. Następnie przechodzi do kolejnego wiersza i kontynuuje zapełnianie. Manager ten nie zmienia „naturalnego” rozmiaru kontrolek.

```
import java.awt.Container;
import java.awt.FlowLayout;
import javax.swing.JButton;
import javax.swing.JFrame;

public class MenadzerUlozenia {

    public static void main(String[] args) {

        JFrame frame = new JFrame("Layout");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        Container contentPane = frame.getContentPane();
        contentPane.setLayout(new FlowLayout(FlowLayout.CENTER, 10, 10));

        for (int i = 1; i <= 10; i++) {
          contentPane.add(new JButton("Button  " + i));
        }

        frame.pack();
        frame.setVisible(true);
    }
}
```

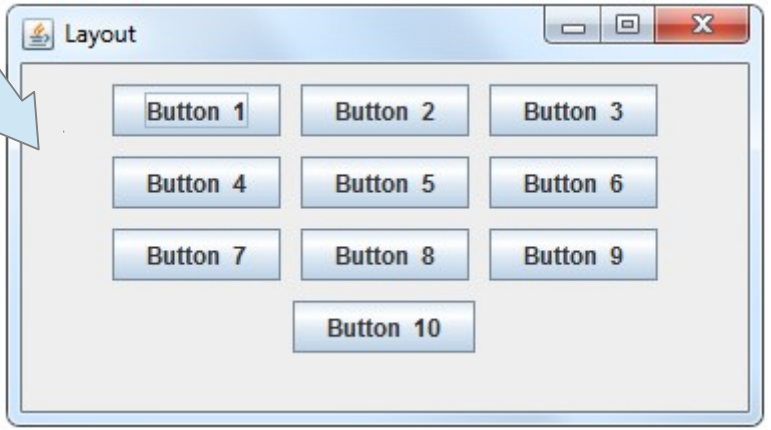
FlowLayout.LEFT – wyrównywanie elementów do lewej

FlowLayout.RIGHT – wyrównywanie elementów do prawej

FlowLayout.CENTER – wyrównywanie elementów do środka

FlowLayout.LEADING – wyrównywanie elementów do lewej lub prawej w zależności od orientacji elementu.
Jeśli jest ona ustawiona na **RIGHT_TO_LEFT**, nastąpi wyrównanie prawo.
Jeśli na **LEFT_TO_RIGHT**, nastąpi wyrównanie na lewo.

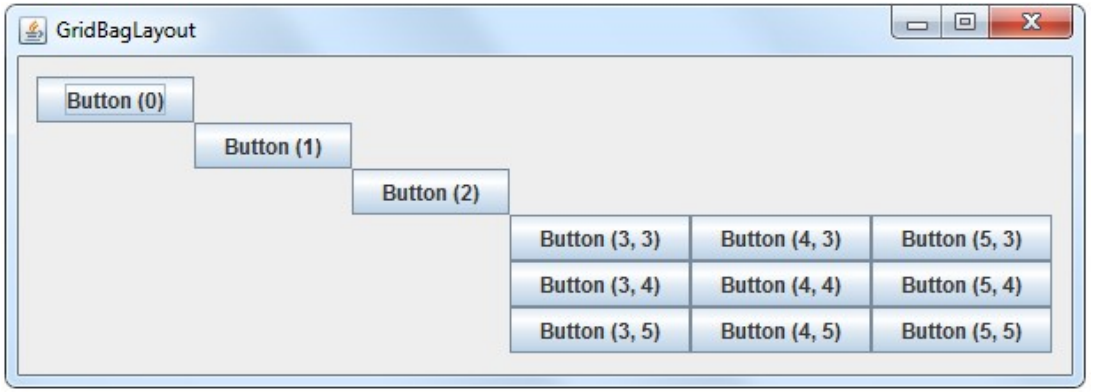
FlowLayout.TRAILING – wyrównywanie elementów do lewej lub prawej w zależności od orientacji elementu.
RIGHT_TO_LEFT → wyrównanie w lewo,
LEFT_TO_RIGHT → wyrównanie w prawo.



Menadżer ułożenia GridBagLayout

Ten menadżer dostarcza ogromnych możliwości kontroli sposobu ułożenia komponentów w oknie oraz sposobu przekształcania elementów podczas zmiany rozmiaru okna. Jest to najbardziej skomplikowany i trudny do zrozumienia menadżer układu. Przeznaczony jest głównie do automatycznego generowania kodu przez graficzne narzędzia tworzenia interfejsu użytkownika.

```
import java.awt.Container;

import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import javax.swing.JButton;
import javax.swing.JFrame;

public class MenadzerUlozenia {

    public static void main(String[] args) {

    JFrame frame = new JFrame("GridBagLayout");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    Container contentPane = frame.getContentPane();
    contentPane.setLayout(new GridBagLayout());

    GridBagConstraints gbc = new GridBagConstraints();

    for (int i = 0;i<3; i++){
        gbc.gridx = i;
        gbc.gridy = i;
        contentPane.add(new JButton("Button (" + i + ")"), gbc);
    }

    for (int y = 3; y < 6; y++) {
      for (int x = 3; x < 6; x++) {
        gbc.gridx = x;
        gbc.gridy = y;
            contentPane.add(new JButton("Button (" + x + ", " + y + ")"), gbc);
      }
    }

    frame.pack();
    frame.setVisible(true);
   }
}
```



Klasa **GridBagConstraints** określa ograniczenia dla elementów, które są określone za pomocą menadżera ułożeń GridBagLayout.

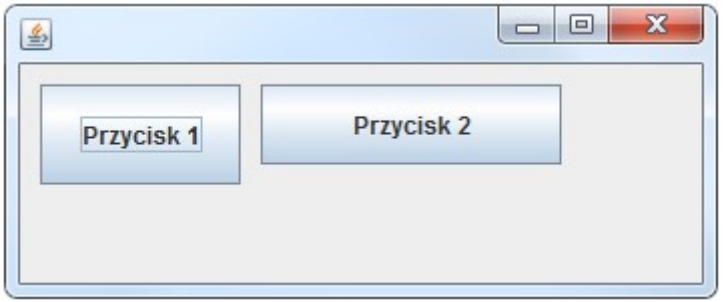
Wyłączanie menadżera ułożeń

Wyłączenie domyślnego menadżera ułożeń, wymusza określenie dla każdego komponentu jego położenia oraz rozmiaru poprzez użycie metod: **setLocation**, **setSize**, **setBounds**.

Wyłączenie menadżera ustawień realizowane jest poprzez wywołanie na wskazanym kontenerze metody **setLayout** z parametrem **null** np:

container.setLayout(null);

```
public static void main(String[] args) {

    JFrame frame = new JFrame();
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    Container contentPane = frame.getContentPane();
    contentPane.setLayout(null);

    JButton b1 = new JButton("Przycisk 1");
    JButton b2 = new JButton("Przycisk 2");
    contentPane.add(b1);
    contentPane.add(b2);

    b1.setBounds(10, 10, 100, 50);
    b2.setBounds(120, 10, 150, 40);

    frame.setBounds(0, 0, 350, 100);
    frame.setVisible(true);

}
```



Przyciski - JButton

```
public static void main(String[] args) {

    JFrame frame = new JFrame("Button test");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    Container contentPane = frame.getContentPane();
    frame.setLayout(new FlowLayout(FlowLayout.LEFT, 20, 40));

    JButton przycisk1 = new JButton();
    contentPane.add(przycisk1);

    JButton przycisk2 = new JButton("Naciśnij mnie!");
    przycisk2.addActionListener( new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            JOptionPane.showMessageDialog(null,"Nacisnąłem cię!");
        }
    });
    contentPane.add(przycisk2);

    Icon audioIcon = new ImageIcon("audio.png");
    JButton przycisk3 = new JButton(audioIcon);
            przycisk3.addActionListener( new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            JOptionPane.showMessageDialog(null,"Gra muzyka!!!");
        }
    });
    contentPane.add(przycisk3);

    Icon mikrofon = new ImageIcon("mikrofon.png");
    JButton przycisk4 = new JButton("Record audio",mikrofon );
    przycisk4.addActionListener( new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            JOptionPane.showMessageDialog(null,"Nagrywam cię!");
        }
    });
    contentPane.add(przycisk4);

    frame.pack();
    frame.setVisible(true);
}
```

Utworzenie formatki JFrame

Utworzenie kontenera na kontrolki z menadżerem widoku FlowLayout

Utworzenie czterech różnych przycisków poprzez użycie 4 różnych przeciążeń konstruktorów:

przycisk1 – konstruktor bezparametrowy

przycisk2 – konstruktor ustawia napis na przycisku

przycisk3 – konstruktor ustawia obrazek na przycisku

przycisk4 – konstruktor ustawia napis wraz z obrazkiem na przycisku

Obsługa zdarzeń wymaga zaimportowania odpowiednich bibliotek:

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
```

Polimorfizm zdarzeń

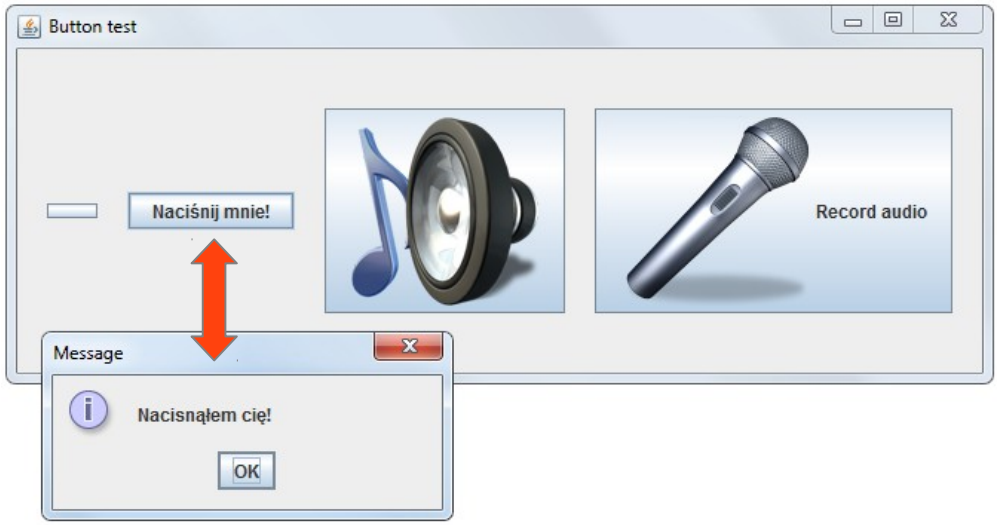
```
JButton przycisk2 = new JButton("Naciśnij mnie!");
przycisk2.addActionListener( new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        JOptionPane.showMessageDialog(null,"Nacisnąłem cię!");
    }
});
contentPane.add(przycisk2);
```

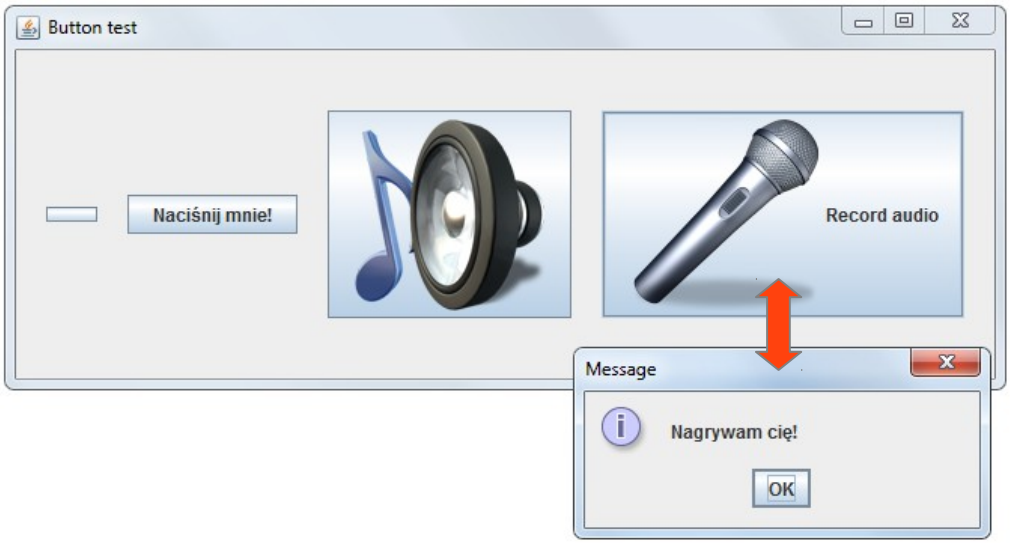
```
Icon audioIcon = new ImageIcon("audio.png");
JButton przycisk3 = new JButton(audioIcon);
        przycisk3.addActionListener( new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        JOptionPane.showMessageDialog(null,"Gra muzyka!!!");
    }
});
contentPane.add(przycisk3);
```

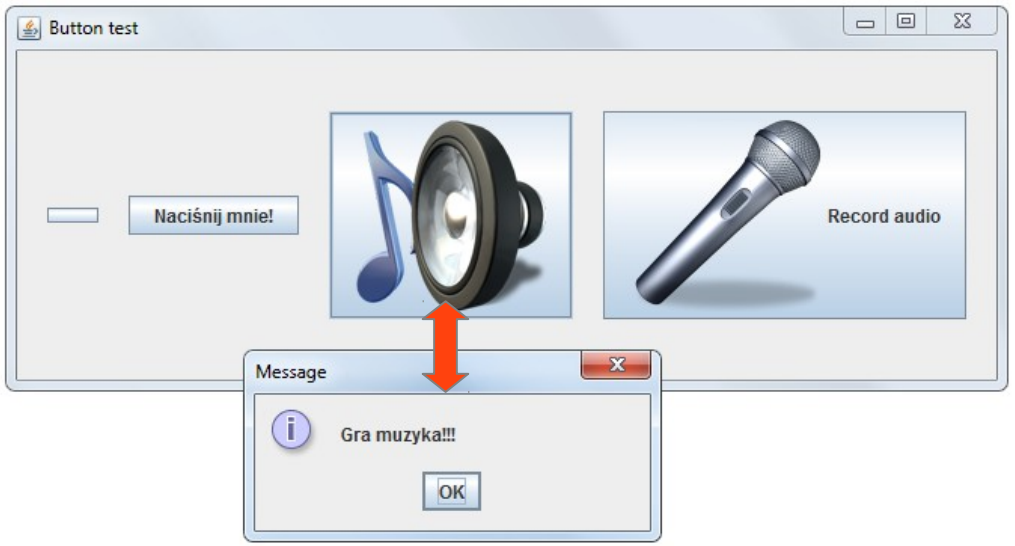
```
Icon mikrofon = new ImageIcon("mikrofon.png");
JButton przycisk4 = new JButton("Record audio",mikrofon );
przycisk4.addActionListener( new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        JOptionPane.showMessageDialog(null,"Nagrywam cię!");
    }
});
contentPane.add(przycisk4);
```

- Trzy przyciski mają zdefiniowane różne metody obsługi zdarzenia naciśnięcia przycisku.
- Metody te są różnią się od siebie jedynie implementacją ciała tzn. że są one przesłaniane.
- Występuje tutaj zatem zjawisko polimorfizmu. Przyciski choć produkowane są przez tę samą klasę **JButton**, to jednak każdy z nich reaguje na naciśnięcie inaczej.

Polimorfizm zdarzeń – wyniki działania







Etykiety JLabel

Kontrolki JLabel reprezentują etykiety, które wyświetlają tekst bez możliwości jego edycji. W JLabel wyświetlać można teksty obrazy a nawet renderować tagi języka HTML.

Poniżej przedstawiono warianty konstruktorów JLabel:

```
public JLabel ();

public JLabel (java.lang.String text);

public JLabel (java.lang.String text, int horizontalAlignment);

public JLabel (Icon image);

public JLabel (Icon image, int horizontalAlignment);

public JLabel (Java.lang.String text, Icon icon, int horizontalAlignment);
```

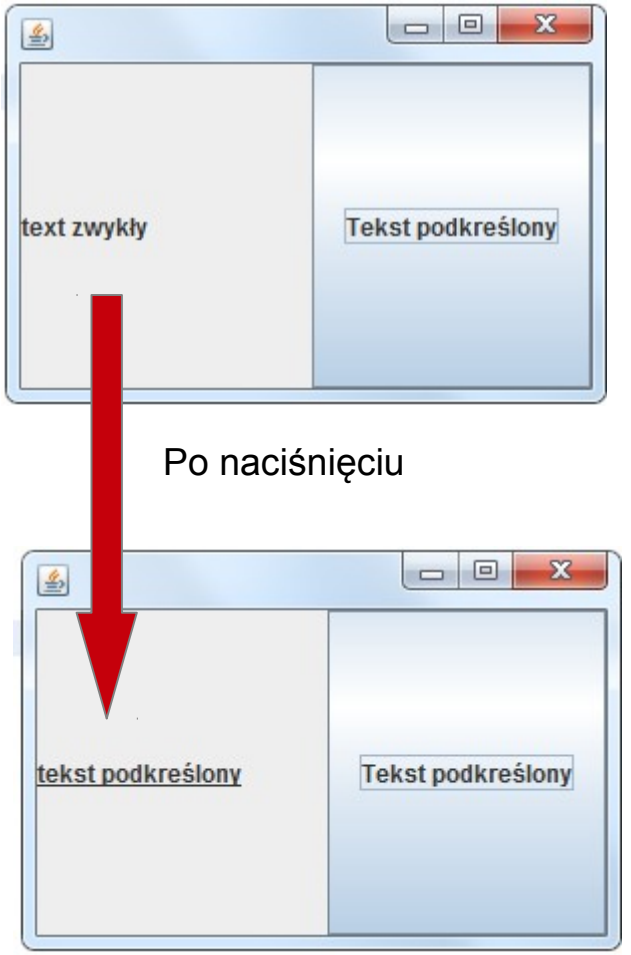
Parametr horizontalAlignment przyjmuje następujące właściwości:

```
SwingConstants.LEFT

SwingConstants.CENTER

SwingConstants.RIGHT

SwingConstants.LEADING

SwingConstants.TRAILING
```

Metoda **setText** służy do zmiany zawartości tekstu w etykiecie.

JLabel - przykład

```
import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.*;

public class Dydaktyka {

    public static void main(String[] args) {
        JFrame frame = new JFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JLabel label = new JLabel("text zwykły");
        frame.add(label, BorderLayout.WEST);
        JButton b= new JButton("Tekst podkreślony");
        b.addActionListener(new ActionListener(){
            @Override
            public void actionPerformed(ActionEvent e) {
                label.setText("<html><u>tekst podkreślony</u></html>");
            }
        });
        frame.add(b, BorderLayout.EAST);

        frame.setSize(300, 200);
        frame.setVisible(true);
    }
}
```



Komponenty tekstowe

W bibliotece Swing zdefiniowano następujące komponenty tekstowe pozwalające na wprowadzanie tekstu w jednej lub wielu liniach.

Do komponentów tekstowych jedno-liniowych zalicza się:

- JTextField,
- JPasswordField,
- JFormattedTextField.

Do komponentów tekstowych wielo-liniowych zalicza się:

- JTextArea,
- JEditorPane,
- JTextPane.

Komponenty tekstowe cd..

Komponenty te można podzielić również w zależności od rodzaju przechowywanego tekstu.

Wyróżnić tutaj można komponenty przechowujące zwykłe teksty oraz komponenty przechowujące teksty wraz ze stylem (formatowaniem). Ta druga grupa przechowuje tekst z określonym formatowaniem tj. np. pogrubienie (bold), pochylenie (italic), podkreślenie (underlined), rodzaj i wielkość czcionki (font), kolor (color) i inne.

Do grupy przechowującej teksty bez formatowania należą następujące komponenty:

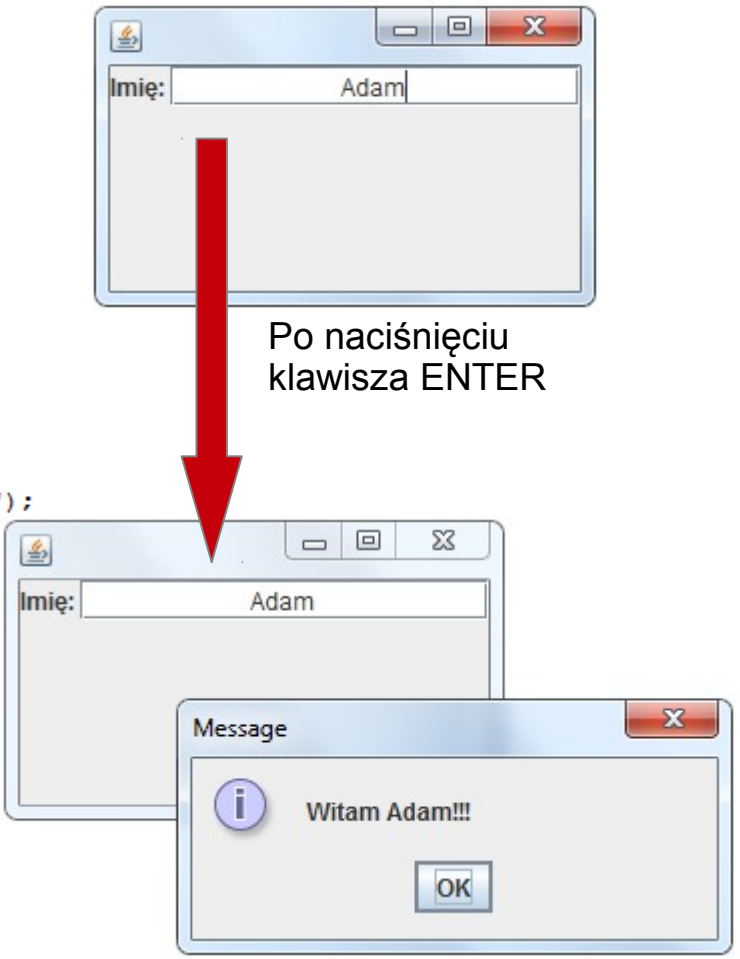
- JTextField,
- JPasswordField,
- JFormattedTextField,
- JTextArea.

Do grupy przechowującej teksty wraz ze stylem należą:

- JEditorPane,
- JTextPane.

JTextField - przykład

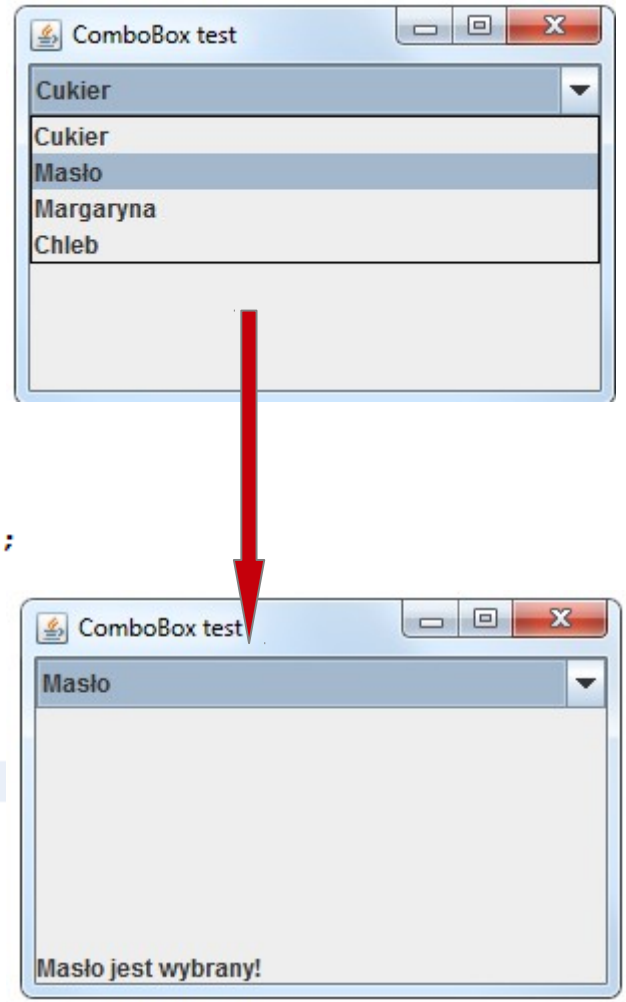
```
import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.KeyEvent;
import javax.swing.*;

public class Dydaktyka {

    public static void main(String[] args) {
        JFrame frame = new JFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JPanel panel = new JPanel(new BorderLayout());
        JLabel label = new JLabel("Imię: ");
        label.setDisplayedMnemonic(KeyEvent.VK_N);

        JTextField textField = new JTextField();
        textField.setHorizontalAlignment(JTextField.CENTER) ;
        textField.addActionListener(new java.awt.event.ActionListener() {
          public void actionPerformed(ActionEvent e) {
            JOptionPane.showMessageDialog(null, "Witam "+textField.getText() +"!!!");
          }
        });

        label.setLabelFor(textField);
        panel.add(label, BorderLayout.WEST);
        panel.add(textField, BorderLayout.CENTER);
        frame.add(panel, BorderLayout.NORTH);
        frame.setSize(250, 150);
        frame.setVisible(true);
    }
}
```



TextArea wielowierszowePoleTekstowe = new TextArea("", 5, 30) - wielowierszowe pole tekstowe

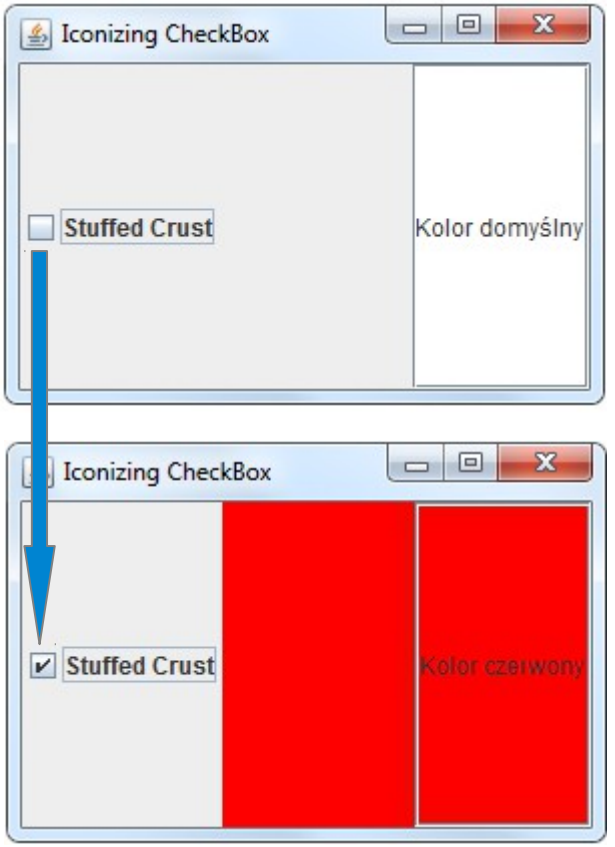
JComboBox - przykład

```
import java.awt.BorderLayout;
import java.awt.Container;
import java.awt.event.ItemEvent;
import javax.swing.*;

public class Dydaktyka {

    public static void main(String[] args) {
        JFrame frame = new JFrame("ComboBox test");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JLabel label1 = new JLabel();
        frame.add(label1, BorderLayout.SOUTH);

        String[] sList = new String[]{"Cukier", "Masło", "Margaryna", "Chleb"};
        JComboBox<String> seasons = new JComboBox<>(sList);

        seasons.addItemListener((ItemEvent e) -> {
            Object item = e.getItem();
            if (e.getStateChange() == ItemEvent.SELECTED) {
                label1.setText(item+" jest wybrany!");
            }
        });

        Container contentPane = frame.getContentPane();
        contentPane.add(seasons, BorderLayout.NORTH);

        frame.setSize(300, 200);
        frame.setVisible(true);
    }
}
```



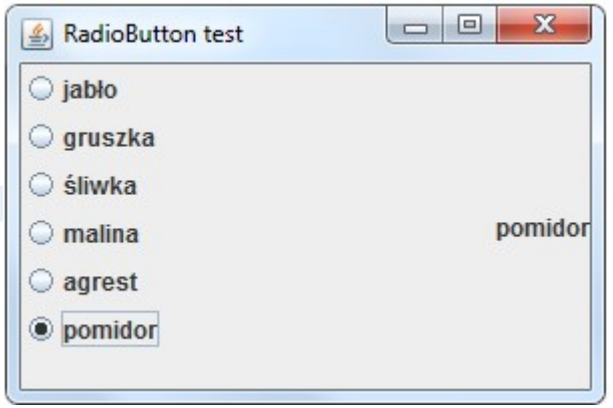
JCheckBox - przykład

```
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.*;

public class Dydaktyka {

    public static void main(String[] args) {
    JFrame frame = new JFrame("Iconizing CheckBox");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    Color defaultColor = frame.getBackground();

    JTextField jTextField = new JTextField("Kolor domyślny");

    JCheckBox jCheckBox = new JCheckBox("Stuffed Crust");
    ActionListener actionListener = new ActionListener() {
      public void actionPerformed(ActionEvent actionEvent) {
          System.out.println(jCheckBox.isSelected());
        if (jCheckBox.isSelected()){
            frame.getContentPane().setBackground(Color.RED);
            jTextField.setBackground(Color.red);
            jTextField.setText("Kolor czerwony");
        }
        else {
            frame.getContentPane().setBackground(defaultColor);
            jTextField.setBackground(Color.WHITE);
            jTextField.setText("Kolor domyślny");
        }
      }
    };

    jCheckBox.addActionListener(actionListener);
    frame.add(jCheckBox,BorderLayout.WEST);
    frame.add(jTextField, BorderLayout.EAST);

    frame.setSize(300, 200);
    frame.setVisible(true);
    }
}
```



JRadioButton - przykład

```
import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.*;

public class Dydaktyka {

    public static void main(String[] args) {
    JFrame frame = new JFrame("RadioButton test");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    JLabel label = new JLabel();

    ButtonGroup grupaOwocow = new ButtonGroup();
    JRadioButton[] owoce = new JRadioButton[6];
    owoce[0]=new JRadioButton("jabło");
    owoce[1]=new JRadioButton("gruszka");
    owoce[2]=new JRadioButton("śliwka");
    owoce[3]=new JRadioButton("malina");
    owoce[4]=new JRadioButton("agrest");
    owoce[5]=new JRadioButton("pomidor");

    for(int i=0; i<owoce.length;i++) grupaOwocow.add((AbstractButton) owoce[i]);

    Box b1 = Box.createVerticalBox();
    for(int i=0; i<owoce.length;i++) b1.add(owoce[i]);

     ActionListener sliceActionListener = new ActionListener() {
      public void actionPerformed(ActionEvent actionEvent) {
        AbstractButton aButton = (AbstractButton) actionEvent.getSource();
        label.setText(aButton.getText());
      }
    };

    for(int i=0; i<owoce.length;i++) owoce[i].addActionListener(sliceActionListener);

    frame.add(b1, BorderLayout.CENTER);
    frame.add(label, BorderLayout.EAST);

    frame.setSize(300, 200);
    frame.setVisible(true);
    }
}
```



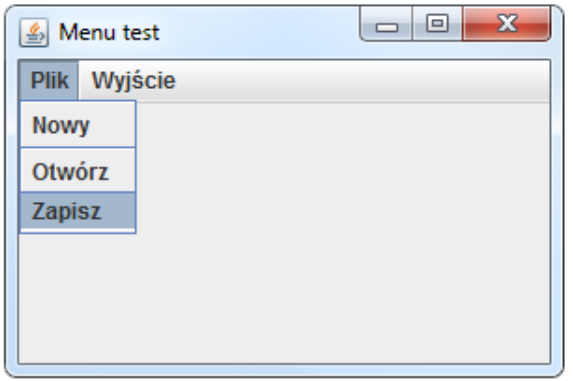
JMenuBar - przykład

```java
import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.*;

class SluchaczMenu implements ActionListener {

    public void actionPerformed(ActionEvent e) {
        String wybrano = e.getActionCommand();
        if (wybrano.equals("Nowy")) {
            System.out.println("Tworzę nowy plik");
        } else if (wybrano.equals("Otwórz")) {
            System.out.println("Otwieram plik");
        } else if (wybrano.equals("Zapisz")) {
            System.out.println("Zapisuję plik");
        } else {
            System.exit(0);
        }
    }
}
```

```java
public class Dydaktyka {

    public static void main(String[] args) {
        JFrame frame = new JFrame("Menu test");
        frame.setTitle("Menu test");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JMenuBar pasekMenu = new JMenuBar();
        JMenu menuPlik = new JMenu("Plik");
        pasekMenu.add(menuPlik);

        JMenuItem opcjaNowy = new JMenuItem("Nowy");
        opcjaNowy.addActionListener(new SluchaczMenu());
        menuPlik.add(opcjaNowy);

        menuPlik.addSeparator();

        JMenuItem opcjaOtworz = new JMenuItem("Otwórz");
        opcjaOtworz.addActionListener(new SluchaczMenu());
        menuPlik.add(opcjaOtworz);

        JMenuItem opcjaZapisz = new JMenuItem("Zapisz");
        opcjaZapisz.addActionListener(new SluchaczMenu());
        menuPlik.add(opcjaZapisz);

        JMenu menuWyjscie = new JMenu("Wyjście");
        JMenuItem opcjaZamknij = new JMenuItem("Zamknij program");
        opcjaZamknij.addActionListener(new SluchaczMenu());
        menuWyjscie.add(opcjaZamknij);
        pasekMenu.add(menuWyjscie);

        frame.add(pasekMenu, BorderLayout.NORTH);
        frame.setSize(300, 200);
        frame.setVisible(true);
    }
}
```

JMenuBar – wynik działania

```
public class Dydaktyka {

    public static void main(String[] args) {
        JFrame frame = new JFrame("Menu test");
        frame.setTitle("Menu test");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JMenuBar pasekMenu = new JMenuBar();
        JMenu menuPlik = new JMenu("Plik");
        pasekMenu.add(menuPlik);

        JMenuItem opcjaNowy = new JMenuItem("Nowy");
        opcjaNowy.addActionListener(new SluchaczMenu());
        menuPlik.add(opcjaNowy);

        menuPlik.addSeparator();

        JMenuItem opcjaOtworz = new JMenuItem("Otwórz");
        opcjaOtworz.addActionListener(new SluchaczMenu());
        menuPlik.add(opcjaOtworz);

        JMenuItem opcjaZapisz = new JMenuItem("Zapisz");
```

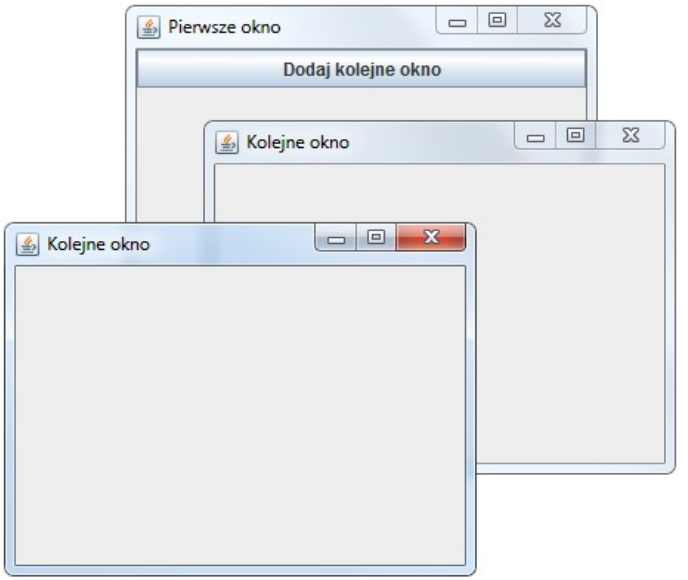


Search Results | Output - Dydaktyka (run) | Notifications

```
run:
Tworzę nowy plik
Otwieram plik
```

Kilka okien w programie

```
import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.*;

public class Dydaktyka {

    public static void main(String[] args) {
        JFrame frame = new JFrame();
        frame.setTitle("Pierwsze okno");

        JButton przycisk = new JButton("Dodaj kolejne okno");
        przycisk.addActionListener(new ActionListener() {

            @Override
            public void actionPerformed(ActionEvent e) {
                JFrame drugieOkno = new JFrame("Kolejne okno");
                drugieOkno.setLocation(300, 300);
                drugieOkno.setSize(320, 240);
                drugieOkno.setVisible(true);
                frame.repaint();
            }
        });

        frame.getContentPane().add(przycisk, BorderLayout.NORTH);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLocation(200, 200);
        frame.setSize(320, 240);
        frame.setVisible(true);
    }
}
```



Okna MDI

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

public class Dydaktyka {

    public static void main(String[] args) {
        JDesktopPane desktopPane = new JDesktopPane();
        JFrame mainWindow = new JFrame("Okno główne");
        mainWindow.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JInternalFrame okno1 = new JInternalFrame("Okno 1", true, true, true,true);
        //okno1.setLocation(100,100);
        okno1.getContentPane().add(new JLabel("Zwartość okna 1 !"));
        okno1.pack();
        okno1.setBounds(100,100,100,100);
        okno1.setVisible(true);

        JInternalFrame okno2 = new JInternalFrame("Okno 2", false, false, false, false);
        okno2.getContentPane().add(new JLabel("Zawartość okna 2 !"));
        okno2.pack();
        okno2.setVisible(true);

        int x2 = okno1.getX() + okno1.getWidth() + 10;
        int y2 = okno1.getY();
        okno2.setLocation(x2, y2);
        okno2.setSize(200,200);

        desktopPane.add(okno1);
        desktopPane.add(okno2);

        mainWindow.add(desktopPane, BorderLayout.CENTER);

        mainWindow.setMinimumSize(new Dimension(300, 300));

        mainWindow.pack();
        mainWindow.setVisible(true);
        mainWindow.setExtendedState(mainWindow.MAXIMIZED_BOTH);
    }
}
```

możliwość zmiany rozmiaru okna

przycisk zamknięcia okna

przycisk maksymalizacji okna

przycisk minimalizacji okna