Oprogramowanie Systemów Medycznych

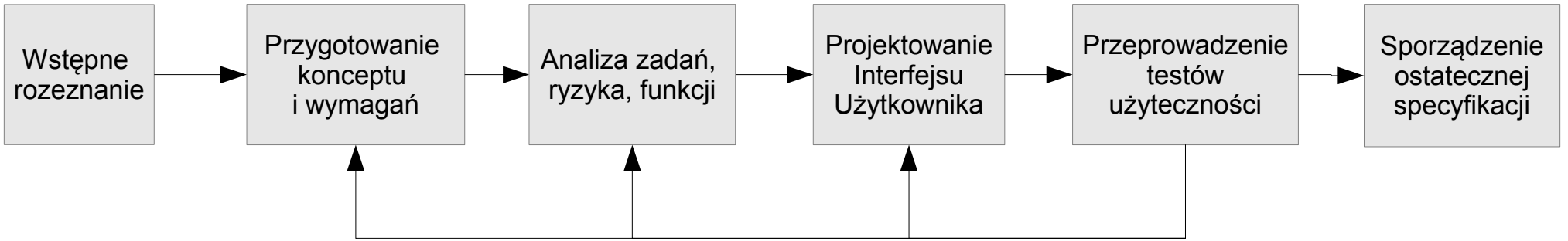
Wykład 3

Programowanie interfejsów użytkownika

- Wszystkie aspekty projektu powinny być spójne i zgodne z oczekiwaniami użytkownika
- Należy wziąć pod uwagę poprzednie doświadczenia użytkownika z podobnymi urządzeniami i uwzględnić przyjęte konwencje
- Projekt powinien być możliwie prosty i dostosowany do możliwości psychofizycznych użytkownika (percepcja, pamięć, itd.)
- Projekt powinien być dobrze zorganizowany i przejrzysty (unikanie przeładowania)
- Etykiety i komunikaty powinny być zrozumiałe i czytelne (poprzez dobór kolorów i czcionki jak i przekazywaną treść)
- Używane skróty i akronimy powinny być zgodne zobowiązującymi konwencjami oraz zgodne z opisem w instrukcji obsługi
- Rozmieszczenie kontrolek graficznych powinno być adekwatne do zastosowania interfejsu, tak aby zminimalizować ryzyko nieumyślnej aktywacji
- Użycie kolorów i symboli graficznych do szybkiej identyfikacji wizualnej
- Interfejs powinien odpowiadać na interakcję

Inżynieria Ergonomii

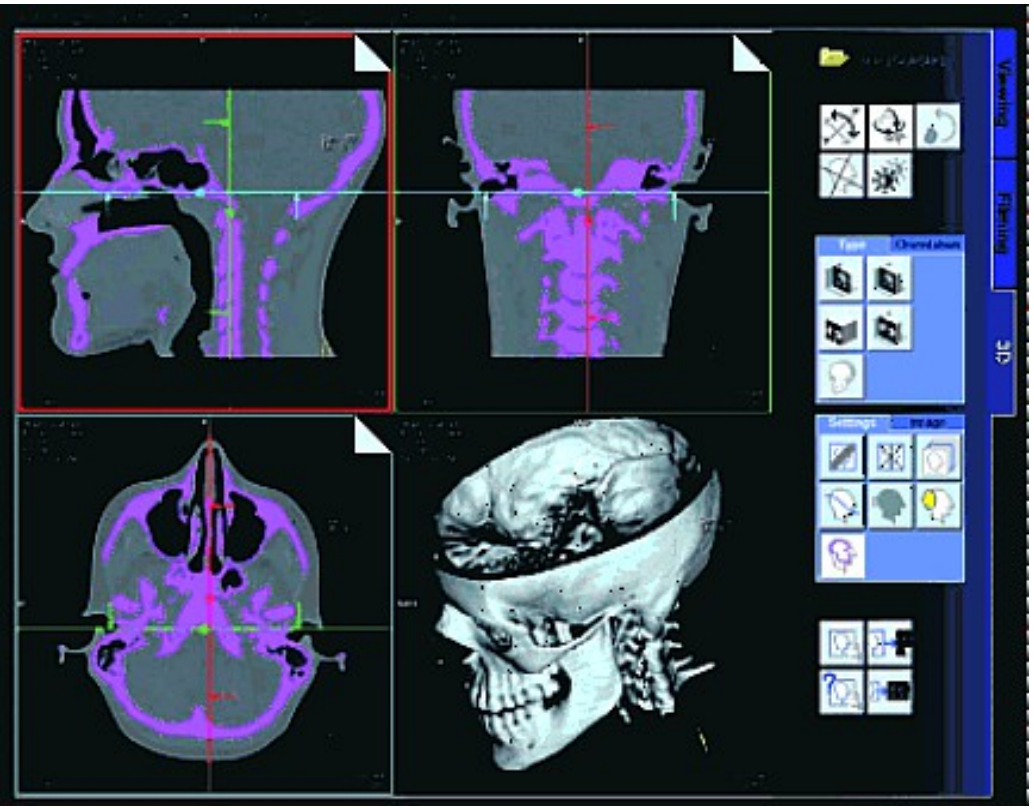
- Ergonomia - „dostosowanie interfejsu urządzenia medycznego do możliwości psychofizycznych człowieka”

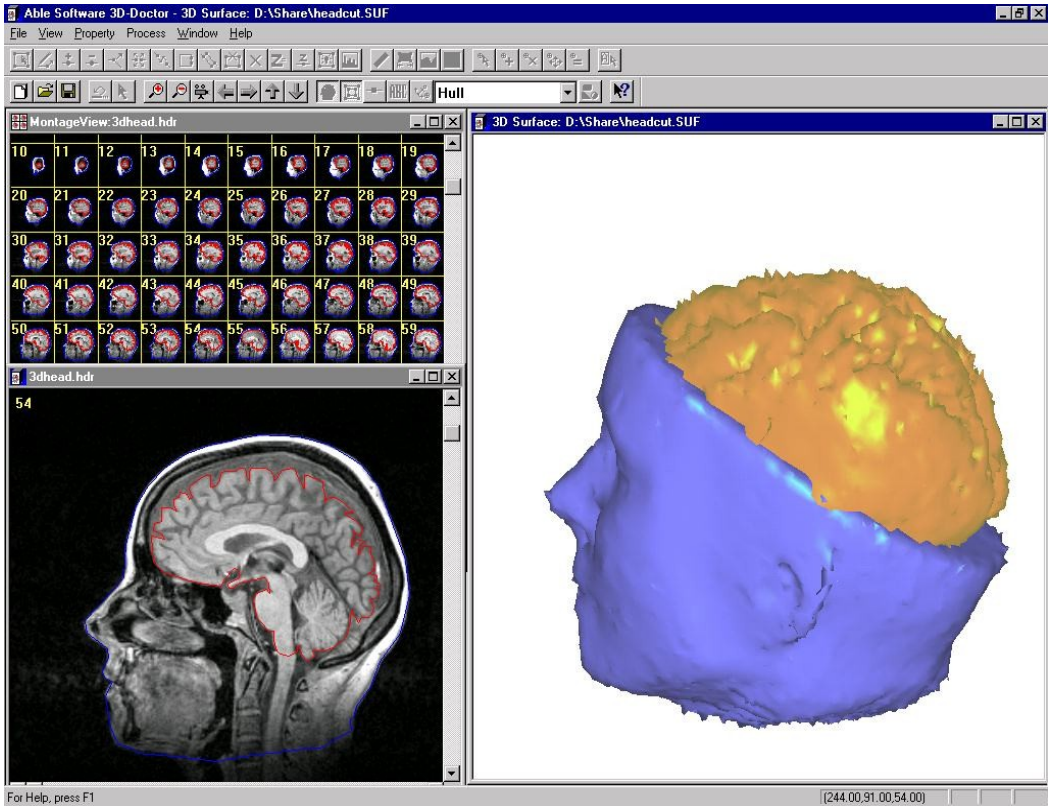


Literatura
Wytyczne
Formaty danych

Użytkownicy oprogramowania
Eksperci danej dziedziny
Konsultanci

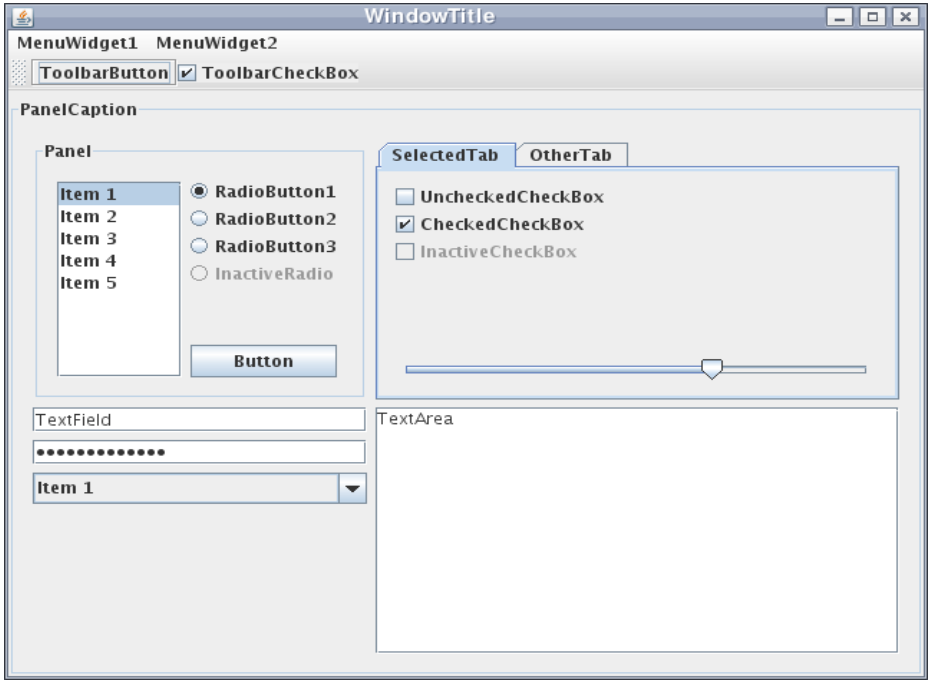
Rysunki
Atrapy
Prototypy
Storyboard's




Able Software 3D-Doctor - 3D Surface: D:\Share\headcut.SUF
File View Property Process Window Help
Hull
MontageView:3dhead.hdr
3dhead.hdr
54
3D Surface: D:\Share\headcut.SUF
For Help, press F1
(244.00,91.00,54.00)

Projektowanie Graficznych Interfejsów Użytkonika w Javie

- Dostępne biblioteki graficzne
  - AWT (Abstract Window Toolkit)
  - **JFC/Swing**
  - SWT (Standard Widget Toolkit)
  - JavaFX
  - QT Jambi



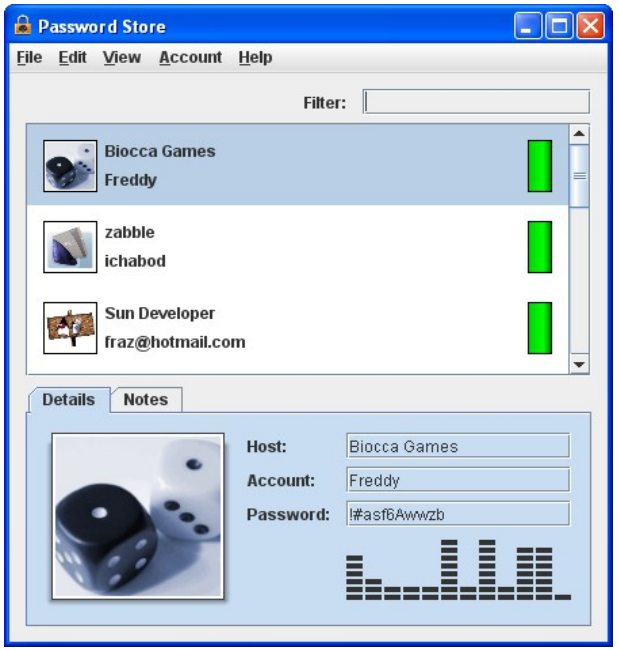
Swing?

- Kontrolki GUI
- Rysowanie 2D (java2d)
- Definiowalny schemat wyglądu i zachowania aplikacji
- Transfer danych – kopiuj, wklej, przesuń-i-puść
- Regionalizacja językowa – zmiana języka aplikacji
- Integracja z narzędziami systemu operacyjnego (np. uruchamianie zewnętrznych aplikacji powiązanych z danym typem plików)

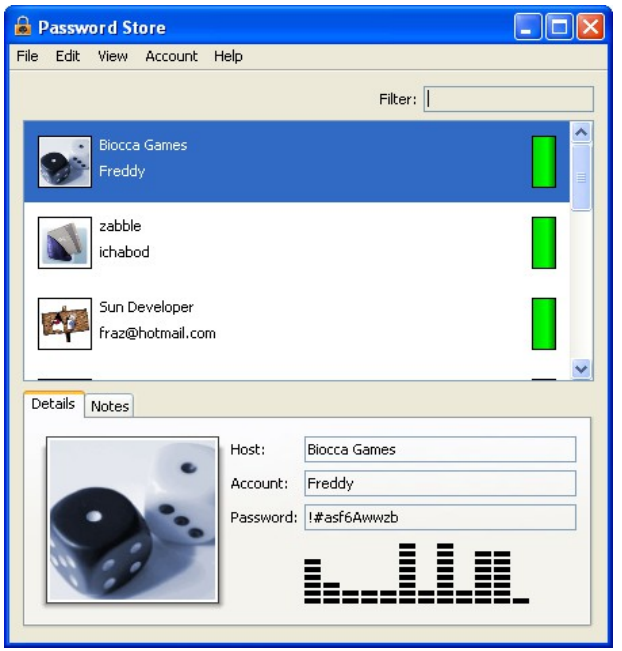
Look and Feel (L&F)

- Look - zapewnienie **wyglądu** kontrolek zgodnego z systemem operacyjnym (lub projektem)
- Feel – zapewnienie **zachowania** kontrolek zgodnego z systemem operacyjnym

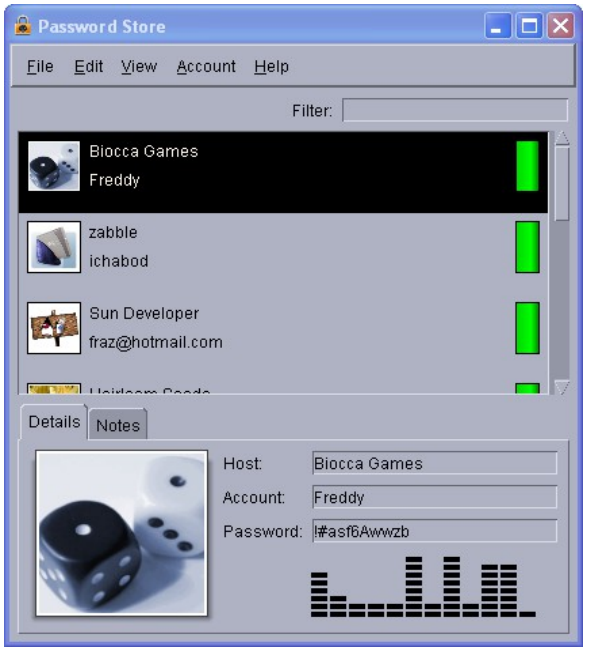
Domyślny



Windows



Motif

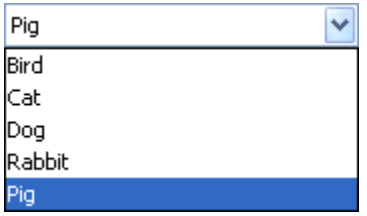


Komponenty

- Swing zapewnia standardowe kontrolki interfejsu użytkownika, tzw. **komponenty**
- Wszystkie komponenty biblioteki Swing (oprócz nadrzędnych) wywodzą się z klasy `javax.swing.JComponent`
- Rodzaje komponentów
  - **Elementy GUI**: kontrolki z których stworzony jest interfejs użytkownika takie jak guziki, listy, menu, pola tekstowe, i inne .
  - **Kontenery -** kontrolki, które są pojemnikami na komponenty, np. okna, panele, paski narzędzi itd.

Podstawowe komponenty

JButton

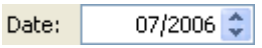


JComboBox

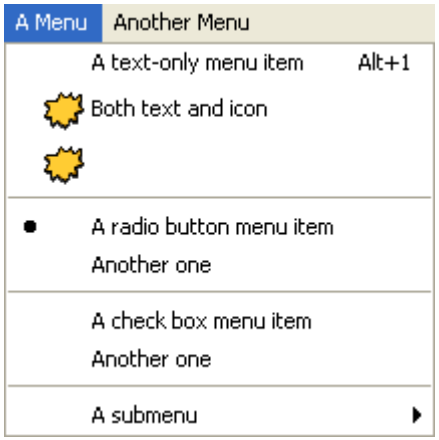
JRadioButton

JList

JCheckBox

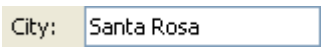


JSpinner



JMenu

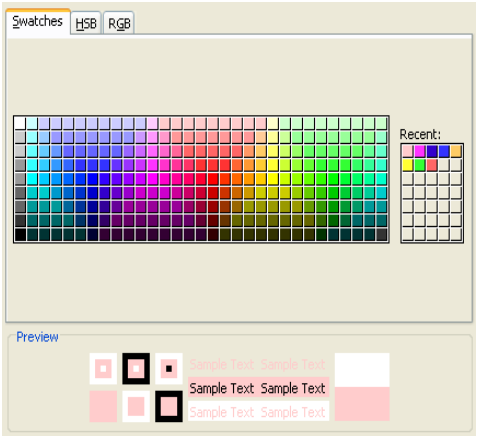
JSlider



JTextField

JPasswordField

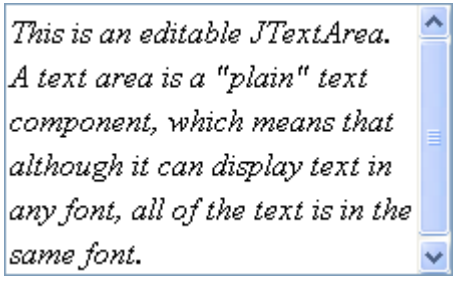
Komponenty formatowane i interaktywne



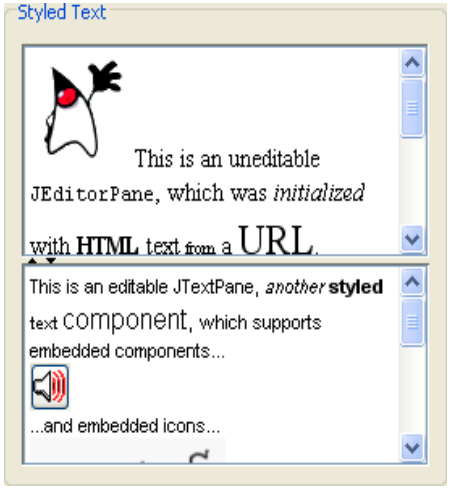
JColorChooser

Host	User	Password	Last Modified
Biocca Games	Freddy	!#asf6Awwzb	Mar 16, 2006
zabble	ichabod	Tazb!34$fZ	Mar 6, 2006
Sun Developer	fraz@hotmail.com	AasW541!fbZ	Feb 22, 2006
Heirloom Seeds	shams@gmail.com	bkz[ADF78!	Jul 29, 2005
Pacific Zoo Shop	seal@hotmail.com	vbAf124%z	Feb 22, 2006

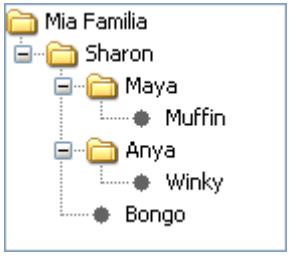
JTable



JTextArea



JEditorPane

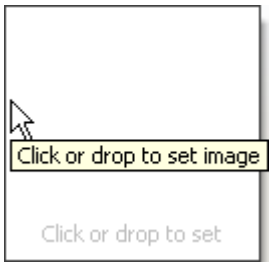


JTree

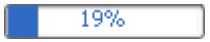
JFileChooser

Komponenty informacyjne

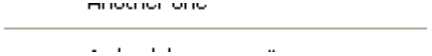
JLabel



JToolTip



JPorgressBar



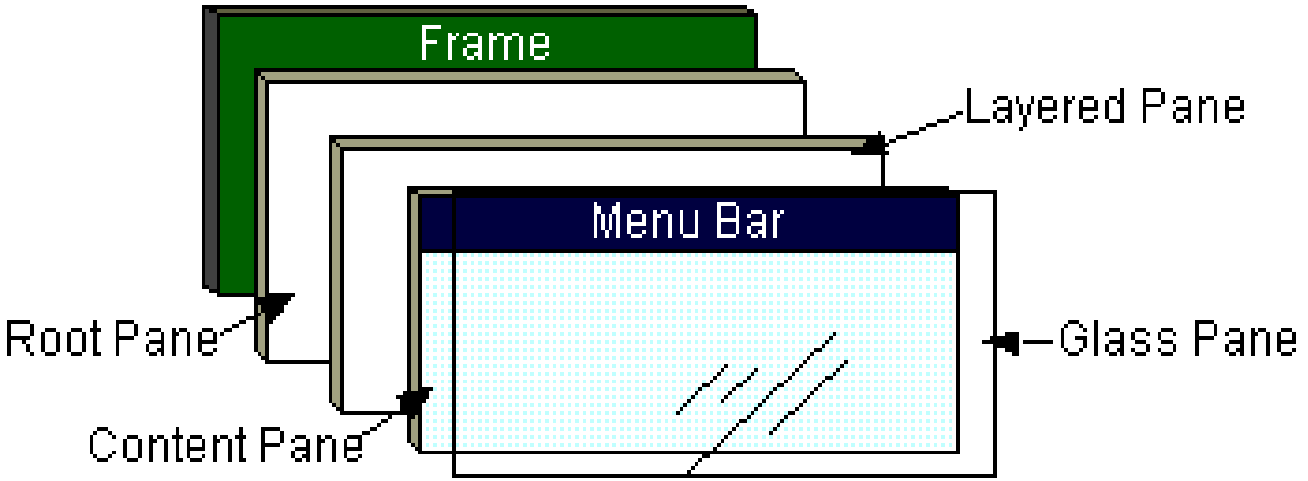
JSeparator

Kontenery

- Wywodzą się z klasy `java.awt.Container` (dot. wszystkich klas pochodnych od JComponent)
- Mogą zawierać w sobie inne komponenty lub kontenery
- Zazwyczaj używają zarządcy rozmieszczenia (`java.awt.LayoutManager`) do określenia pozycji i rozmiaru komponentów przetrzymywanych przez dany kontener
- Dodawanie komponentów zazwyczaj odbywa się za pomocą jednego z wariantów metody **add**

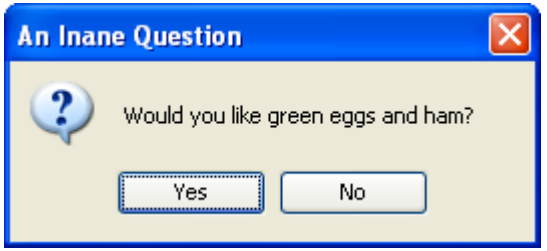
Kontenery nadrzędne (top level)

- Każdy interfejs graficzny musi być opakowany w co najmniej jeden tego typu kontener
  - Są 3 takie kontenery: JFrame, JDialog, JApplet
  - Aby pojawić się na ekranie, każdy komponent GUI musi być częścią hierarchii komponentów (kompozytu), która ma komponent typu top-level w swoim korzeniu
- Kontenery typu top-level zapewniają aplikacji malowanie interfejsu użytkownika oraz obsługę zdarzeń



Kontenery nadrzędne

- `javax.swing.JFrame` - „ramka” - okno aplikacji zawierające dekoracje takie jak obramowanie, przyciski do zamykania, minimalizowania, itd.
- `javax.swing.JDialog` - „dialog” - okno powiadomienia, potwierdzenia lub innej interakcji (np. wybór pliku), które jest zależne od JFrame
  - Zamknięcie głównego okna zamyka wszystkie powiązane z nim okna dialogowe
  - Kiedy ramka jest minimalizowana/maksymalizowana, powiązane z nią okna dialogowe również
  - Okna dialogowe mogą być modalne, tzn. blokować interakcję z powiązaną ramką dopóki nie zostanie zamknięte (np. użytkownik nie wciśnie przycisk „OK”)

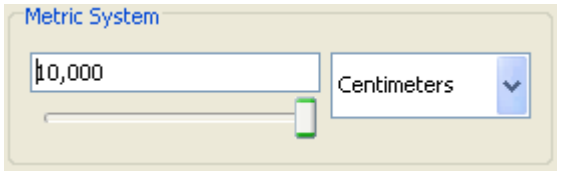


Hello Swing

```
import javax.swing.*;

public class SwingApplication extends JFrame {

    public SwingApplication() {
        // initialize GUI
        setTitle("This is a frame title");
        add(new JLabel("Hello Swing"));
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        pack();
    }

    public static void main(String[] args) {

        // crate GUI
        SwingApplication app = new SwingApplication();
        // show GUI
        app.setVisible(true);
    }
}
```

Kontenery podrzędne

- Panele (**panel**, lub **pane** – jak w *window pane*) pozwalające na grupowanie komponentów wewnątrz głównego okna aplikacji



JPanel

JSplitPane

JScrollPane

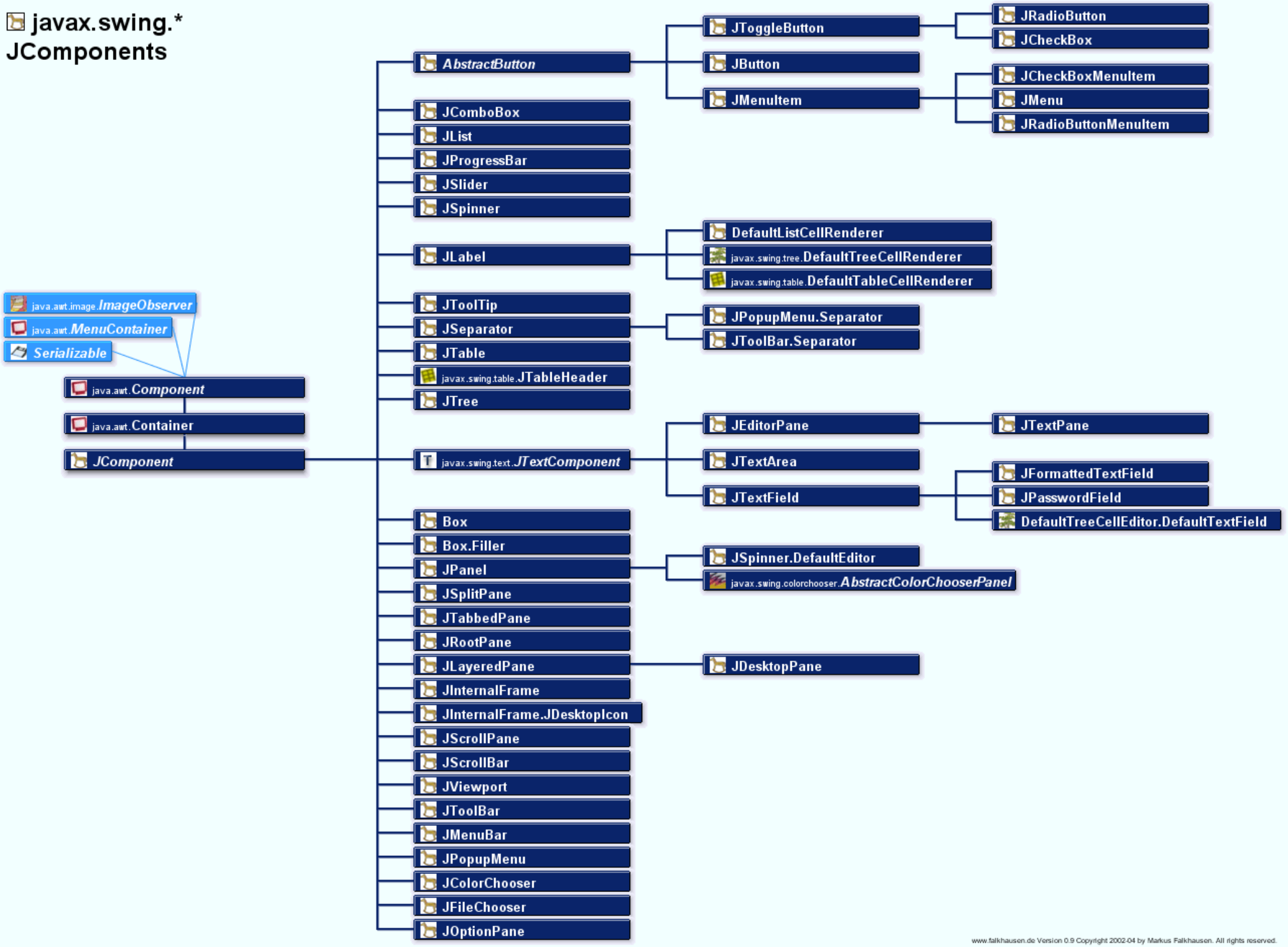
JTabbedPane

JToolBar

Kontenery podrzędne

- Domyślnie kontenery (panele), nie malują nic poza swoim tłem
- Domyślnie panele są nieprzezroczyste (opaque)
  - nieprzezroczysty panel może być ustawiony w komponencie nadrzędnym (top-level) jako jego *content pane*
  - Panele mogą być również przezroczyste – nie malować swojego tła

javax.swing.*
JComponents

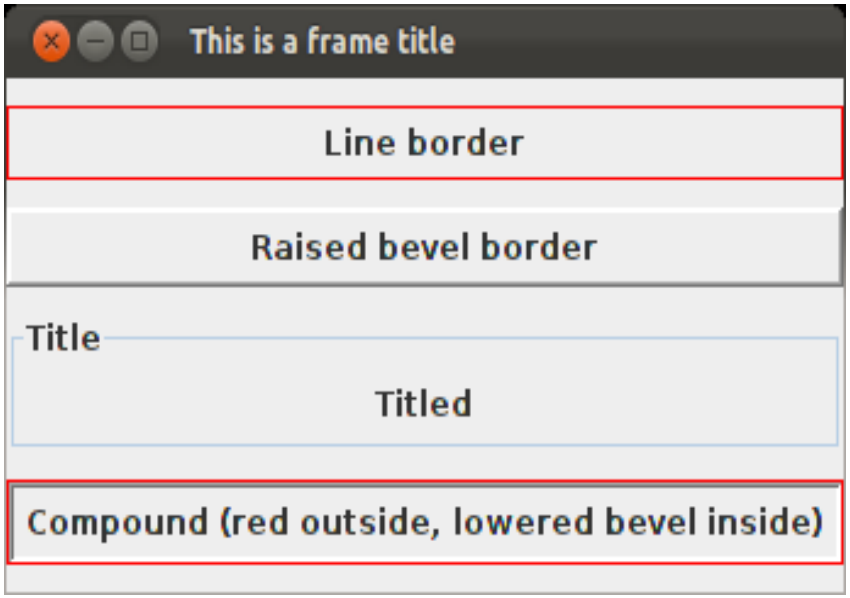


Obramowanie

- Każdy komponent może posiadać obramowanie
- Do tworzenia obramowania służy klasa `javax.swing.BorderFactory`
- Obramowanie może być złożeniem każdych dwóch innych ramek (tzw. compound border)

```
panel = new JPanel();
panel.setBorder( BorderFactory.createLineBorder( Color.red) );
panel.add( new JLabel("Line border") );
```

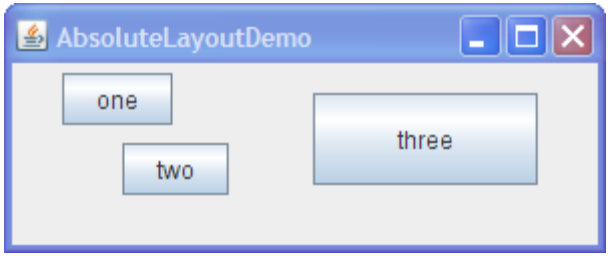
```
panel.setBorder( BorderFactory.createCompoundBorder(
            BorderFactory.createLineBorder(Color.red),
            BorderFactory.createLoweredBevelBorder() ) );
panel.add( new JLabel("Compound (red outside, lowered bevel
inside)" ) );
```



Zarządzanie rozmieszczeniem

- Rozmieszczenie komponentów w kontenerze może być wykonane z wykorzystaniem **absolutnej pozycji** poszczególnych komponentów
  - Wymaga podania rozmiaru komponentu
  - Wzajemne położenie komponentów nie jest aktualizowane kiedy rozmiar kontenera jest aktualizowany (np. przez zmianę wielkości okna)

```
pane.setLayout(null);

JButton b1 = new JButton("one");
JButton b2 = new JButton("two");
JButton b3 = new JButton("three");

pane.add(b1);
pane.add(b2);
pane.add(b3);

Insets insets = pane.getInsets();
Dimension size = b1.getPreferredSize();
b1.setBounds(25 + insets.left, 5 + insets.top,
             size.width, size.height);
size = b2.getPreferredSize();
b2.setBounds(55 + insets.left, 40 + insets.top,
             size.width, size.height);
size = b3.getPreferredSize();
b3.setBounds(150 + insets.left, 15 + insets.top,
             size.width + 50, size.height + 20);
```

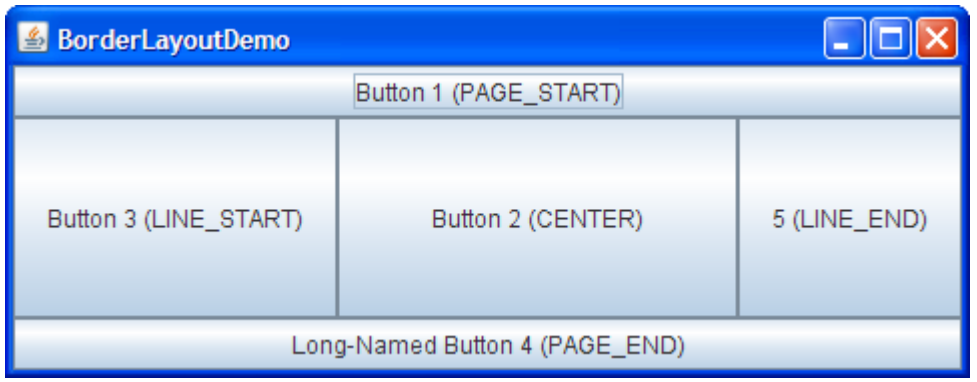


Zarządzanie rozmieszczeniem

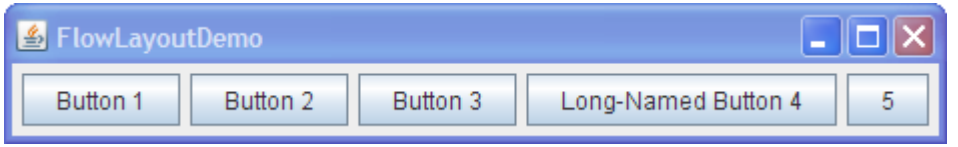
- Ułożenie komponentów może być realizowane przez klasy implementujące interfejs javax.swing.LayoutManager, który jest strategią rozmieszczenia, aktualizacji rozmiaru i wzajemnego ułożenia komponentów w zależności od rozmiaru kontenera otaczającego
- Użycie:
  - Wybór LayoutManager'a przypisanego do danego kontenera, w zależności od tego jak ma wyglądać rozmieszczenie komponentów w projektowanym GUI
  - Określenie *wskazówek* (*hints*) rozmieszczenia przy dodawaniu kolejnych komponentów do kontenera. Wskazówki mogą, choć nie muszą mieć wpływ na to jak komponenty są ustawione, jakie posiadają rozmiary itd.

```
pane.setLayout( new BorderLayout() );

JButton b1 = new JButton("one");
pane.add(b1, BorderLayout.CENTER );
```

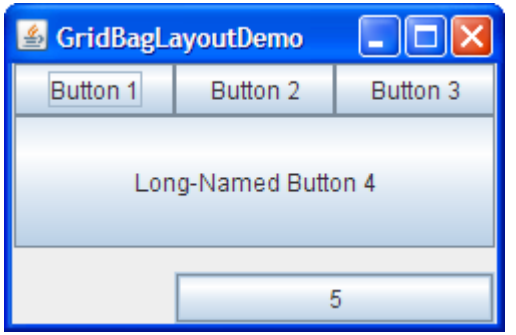
LayoutManager



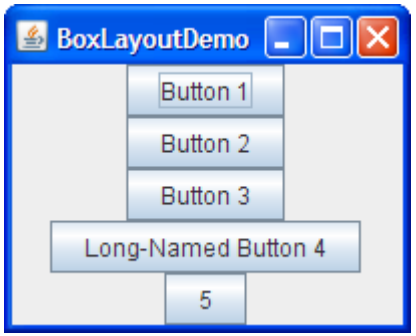
BorderLayout



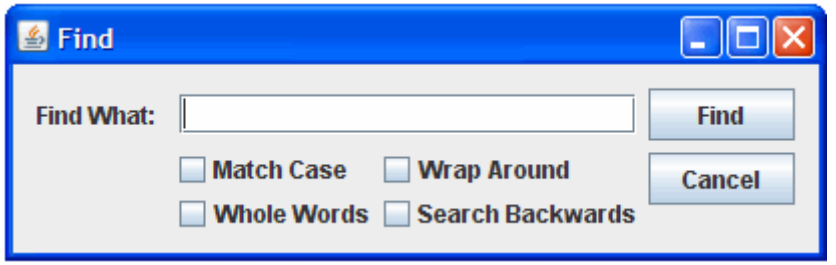
FlowLayout



GridBagLayout



BoxLayout

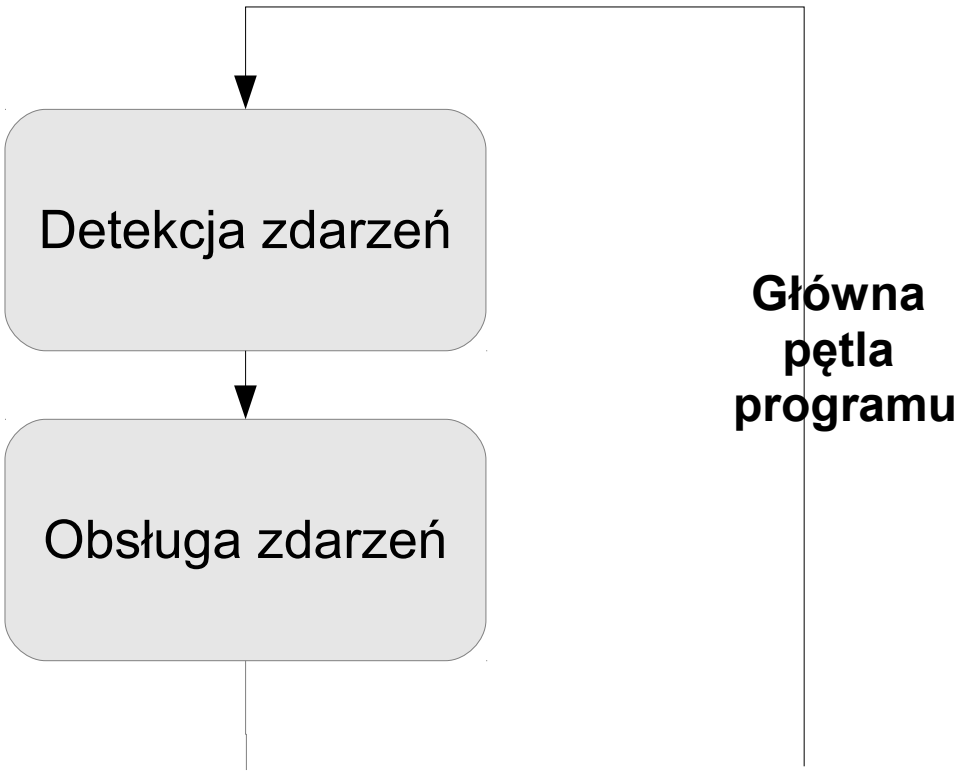


GroupLayout

CardLayout

Programowanie zdarzeniowe

- Przebieg wykonywania programu nie jest z góry ustalony, lecz jest modyfikowany przez wystąpienie pewnych **zdarzeń** (np. interakcja użytkownika z interfejsem)

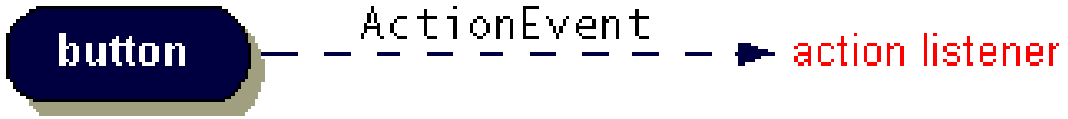


Problemy:

- Wielowątkowość
- Czas obsługi zdarzeń
- Pułapki rekurencyjne

Programowanie zdarzeniowe w Swing'u

- Każde zdarzenie polegające na ruchu myszą, naciśnięciu przycisku itd. generuje zdarzenie GUI (e*vent*)
- Każdy obiekt może zostać powiadomiony o takim zdarzeniu poprzez zarejestrowanie się jako „obserwator” (*event listener*) w danym obiekcie generującym zdarzenia (*event source*) – zazwyczaj jest to komponent
- Jeden komponent może obsługiwać wielu obserwatorów różnego typu, jednak kolejność przekazywania zdarzeń do obserwatorów jest arbitralna



Typy EventListener'ów

Typ	Kiedy generorwane?
ActionListener	Użytkownik nacisnął przycisk, lub nacisnął ENTER podczas edycji pola tekstowego lub wybrał element menu
WindowListener	Użytkownik zamyka, modyfikuje okno aplikacji
MouseListener	Użytkownik naciska przycisk w czasie kiedy kursor myszy znajduje się nad komponentem
MouseMotionListener	Użytkownik przesuwa mysz ponad komponentem
ComponentListener	Komponent staje się widoczny
FocusListener	Komponent staje się aktywny do edycji za pomocą klawiatury
ListSelectionListener	Użytkownik wybiera elementy listy lub tabeli

Implementacja obsługi zdarzeń

- Implementacja interfejsu obserwatora (*listener*) lub rozszerzenie klasy implementującej interfejs
- Zarejestrowanie egzemplarza klasy obsługującej zdarzenie jako *listener*, w jednym lub wielu interesujących komponentach
- Implementacja metody interfejsu obserwatora w celu obsługi konkretnego zdarzenia wygenerowanego przez obiekt obserwowany

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

public class SwingApplication extends JFrame {

    private int numclicks;
    private JLabel label;
    private JButton button;

    public SwingApplication() {

        // initialize components
        label  = new JLabel(" ");
        button = new JButton( "Click me" );
        // add listeners
        button.addActionListener( new ActionListener() {
            public void actionPerformed( ActionEvent e ) {
                numclicks++;
                label.setText( "Num clicks " + numclicks );
            }
        });

        // initialize GUI
        setTitle("This is a frame title");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        getContentPane().setLayout( new BorderLayout() );

        // lay out components
        JPanel panel = new JPanel();
        panel.setLayout( new BoxLayout(panel, BoxLayout.Y_AXIS) );
        label.setAlignmentX( Component.CENTER_ALIGNMENT );
        button.setAlignmentX( Component.CENTER_ALIGNMENT );
        panel.add( label );
        panel.add( button );
        panel.setBorder( BorderFactory.createTitledBorder("Count clicks") );
        getContentPane().add( panel, BorderLayout.CENTER );

        pack();
    }

    public static void main(String[] args) {

        // crate GUI
        SwingApplication app = new SwingApplication();
        // show GUI
        app.setVisible(true);
    }
}
```

Więcej informacji

http://docs.oracle.com/javase/tutorial/uiswing