Wykład 7

Swing wprowadzenie

kontenery i komponenty

LayoutManager

komponenty tekstowe

inne przydatne komponenty

Czym jest Swing?

Swing w Javie to narzędzie do tworzenia graficznego interfejsu użytkownika. Jest **to biblioteka, która umożliwia tworzenie wysoce interaktywnych i estetycznie atrakcyjnych aplikacji** czy okienkowe interfejsy użytkownika.

Obecnie rzadko używa się tej funkcjonalności, ze względu na to, że teraz pisze się częściej aplikacje przeglądarkowe.

tworzenie nowego okna:

```java
import javax.swing.*;
public class HelloWorldSwing {
    private static void createAndShowGUI() {
        // nowe okno o tytule HelloWorldSwing
        JFrame frame = new JFrame("HelloWorldSwing");
        // zamkniecie okna spowoduje zakonczenie programu
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        // nowy napis
        JLabel label = new JLabel("Hello World");
        // napis jest dodawany do zawartosci okna
        frame.getContentPane().add(label);
        // dopasowanie rozmiarow okna do umieszczonych w nim komponentow
        frame.pack();
        // wyswietlenie okna
        frame.setVisible(true);
    }
    public static void main(String[] args) {

        // stworzenie nowego watku, w ktorym zostanie 'uruchomione' okno
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                createAndShowGUI();
            }
        });
    }
}
```

> tworzenie nowego okienka graficznego odbywa się za pomocą konstruktora na bazie instancji np. „frame” klasy *JFrame*

`setDefaultCloseOperation()` — ustawienie domyślnego zachowania okienka, wtedy, gdy jest ono zamykane (np. guzikiem x)

`JFrame.EXIT_ON_CLOSE` — zakończenie działania programu, zniknięcie okienka

Domyślna opcja:

`JFrame.HIDE_ON_CLOSE` – zniknięcie okienka, ale nie zniszczenie go - ***cały czas program będzie w pamięci!***

`JFrame.DISPOSE_ON_CLOSE` — okienko zniknie z ekranu, zostanie zniszczone i pamięć się zwolni, może zakończyć program, ale ***nie musi*** jeśli istnieją inne okna

Różnica między `JFrame.EXIT_ON_CLOSE` & `JFrame.DISPOSE_ON_CLOSE`

```
Jeśli masz otwarte kilka JFrames i zamkniesz ten, który jest ustawiony na
EXIT_ON_CLOSE, wszystkie ramki zostaną zamknięte. Przeciwnie jest
z DISPOSE_ON_CLOSE, tzn. zamknie się tylko to jedno okno.
```

Najprostszy komponent to stały napis w okienku, który tworzymy poprzez konstruktor na bazie instancji np. *„label”* klasy `JLabel` .

Zapis

```
JLabel label = new JLabel(“Hello World”);
```

tworzy komponent, ale jeszcze nie dodaje go do okienka! Aby to zrobić napisać:

```
frame.getContentPane().add(label);
```

`frame.getContentPane()` — pobieramy obiekt reprezentujący zawartość tego okienka „frame”

`add(label)` — dodaje do zawartości okienka "*frame*" utworzony wcześniej "*label*", czyli napis "*Hello World*"

`pack()` – powinno się ją wywołać jeszcze przed wyświetleniem okienka – wszystkie komponenty okienka są ustawiane w odpowiedni sposób i następnie rozmiar okienka jest dostosowywany do tego, aby wszystkie komponenty się tam zmieściły.

`setVisible(true)` – wyświetla okienko na ekranie

> **Uruchamianie okienek zaleca się robić w osobnym wątku**, bo niektóre operacje wywoływane na okienkach są czasochłonne, więc jeśli będzie to wykonywane w bieżącym wątku, w którym dzieje się coś jeszcze innego, no to ten bieżący wątek zostanie przerwany. Swing ma do tworzenia osobnych swoich wątków specjalne narzędzia, nie musimy więc stosować *thread*.

Jednym z tych narzędzi jest:
`SwingUtilities.invokeLater(new Runnable() … )` – tworzy nowy wątek i uruchamia go, gdy nadejdzie odpowiedni moment.

W *Runnable()* implementujemy własną metodę *run()*, która w tym wypadku wywołuje metodę `createAndShowGUI()` , która tworzy i wyświetla okienko.

Komponenty / Kontenery

Kontener — to konstrukcja, do której można dodawać komponenty. Do kontenerów najwyższego poziomu można dodawać inne kontenery niższego poziomu.

Każda aplikacja wykorzystująca Swing używa co najmniej jeden
kontener najwyższego poziomu.

- **JFrame**
- **JDialog**
- **JApplet**

`JFrame` — okienko

`JDialog` — okienko dialogowe, służy do wprowadzania danych

`JApplet` — umieszczało się je na stronach www, służyły do wstawiania kodu javowego i przeglądania w formie przeglądarki (okienko w ramach przeglądarki)

> Zasadniczo *JFrame* od *JDialog* różni się tym, że okienko *JDialog* może być modalne. Co to znaczy? Że kompilacją klawiszy (zależy od systemu) nie możemy przełączać się między okienkami. **Służy to do tego, aby najpierw wpisać potrzebne dane, a dopiero po tym program może pójść dalej.** Po ludzku to jest to natrętne okienko w którym musisz coś zatwierdzić żebby ci z oczu zniknęło i pozwoliło iść dalej.

W takim kontenerze umieszcza się kolejne kontenery lub komponenty:

```
frame.getContentPane().add(yellowLabel, BorderLayout.CENTER);
```

A tutaj tworzymy *panel (kontener)* dodajemy do niego komponenty
a nastepnie umieszczamy go w kontenerze najwyzszego poziomu *(frame)*

```
JPanel contentPane = new JPanel(new BorderLayout());
contentPane.setBorder(someBorder);
contentPane.add(someComponent, BorderLayout.CENTER);
contentPane.add(anotherComponent, BorderLayout.PAGE_END);
frame.getContentPane().add(contentPane);
```

`new BorderLayout()` – oznacza, że będzie można pozycjonować komponenty okienka

`setBorder()` – dodaj obramowanie

Za rozmieszczenie komponentów w kontenerze odpowiedzialny jest obiekt typu `LayoutManager`. Możemy go określić za pomocą metody `setLayout()`.

Domyślnym *LayoutManager'em* dla każdego kontenera jest `FlowLayout()` (instancja tej klasy). Jeśli nie chcemy żadnej domyślnej metody z klasy *LayoutManager*, tylko **samemu określać położenie komponentów**, to używamy instrukcji `container.setLayout(null)`.

Do kontenerów najwyższego poziomu można także dodać menu:

```
frame.setJMenuBar() // ma nie wymagać szczegółów na egzaminie
```

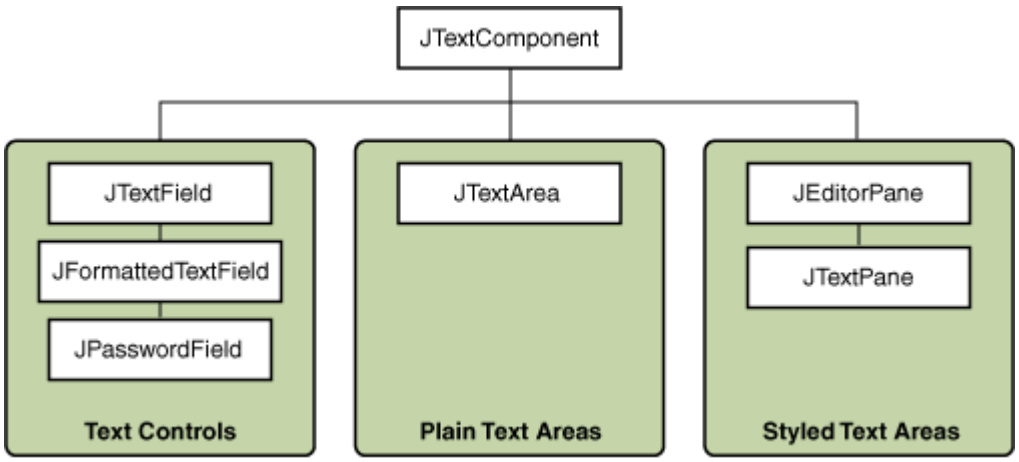
Komponent — to podstawowy element interfejsu użytkownika, które można dodawać do kontenerów. Wszystkie komponenty rozszerzają klasę **JComponent**. Klasa *JComponent* implementuje następujące funkcjonalności:

- podpowiedzi (`setToolTipText()`),
- ramki (`setBorder()`),
- styl (`UIManager.setLookAndFeel()`),
- dodatkowe właściwości (`setClientProperties()`),
- rozmiary (layout)
- przystępność (accessibility)
- przeciągnij i upuść
- podwójne buforowanie
- wiązanie klawiszy

Komponenty tekstowe

Komponenty tekstowe(`JTextComponent`) — służą do wpisywania tekstu.

dzielą się na trzy kategorie:



Text controls:

`JTextField` — jednolinijkowe pole tekstu

`JFormattedTextField` — tekst po wpisaniu w pole tekstowe zostaje sformatowany (np. wpisywanie złotych i groszy)

`JPasswordField` — wpisywany tekst jest ukryty, zagwiazdkowany

Plain Text Areas:

`JTextArea` — pole tekstowe, ale **nie jednolinijkowe**

Styled Text Areas:

`JEditorPane` — umożliwiają wpisywanie dłuższego tekstu, tak jak *JTextArea* oraz obsługuje wyświetlanie i edycję *HTML*(obsługuje html wersja 3)

`JTextPane` — Jest to podklasa *JEditorPane* i oferuje dodatkowe funkcje przetwarzania tekstu, takie jak czcionki, style tekstu, kolory itp.

Przykład programu:

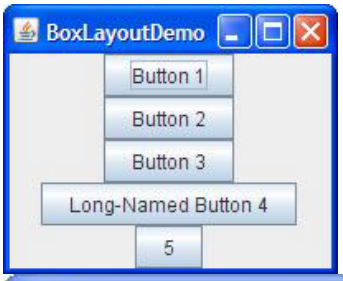
```
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.net.URL;
import javax.swing.*;
public class Browser extends JFrame implements ActionListener {
    private static final String COMMAND_GO = "go";

    private JEditorPane webpage;
    private JTextField url;
    private JTextArea htmlPage;

    private JPanel createMainPanel() {
        JPanel mp = new JPanel();
        // panel z kolumnowym (Y_AXIS) ukladem elementow
        mp.setLayout(new BoxLayout(mp, BoxLayout.Y_AXIS));
        // panel z domyślnym FlowLayout
        JPanel p = new JPanel();
        this.url = new JTextField();

        // sugerowane rozmiary – mogą zostać zmienione przez layout
        // managera
        this.url.setPreferredSize(new Dimension(500, 20));
        this.url.setText("http://www.simongrant.org/web/guide.html");
        JLabel l = new JLabel("adres");
        // label opisujacy url
        l.setLabelFor(this.url);

        // dodajemy do panelu JLabel I JTextField
        p.add(l);
        p.add(this.url);
        // tworzymy przycisk
        JButton b = new JButton("Go");
        b.setActionCommand(COMMAND_GO);
        // ktorego “akcje” beda obslugiwane przez biezacy obiekt
        // (implementujacy interfejs ActionListener. Akcja “produkowana”
        // przez przycisk będzie identyfikowane Stringiem COMMAND_GO
        b.addActionListener(this);
        b.setPreferredSize(new Dimension(100, 40));
        // dodanie przycisku do panelu
        p.add(b);
```

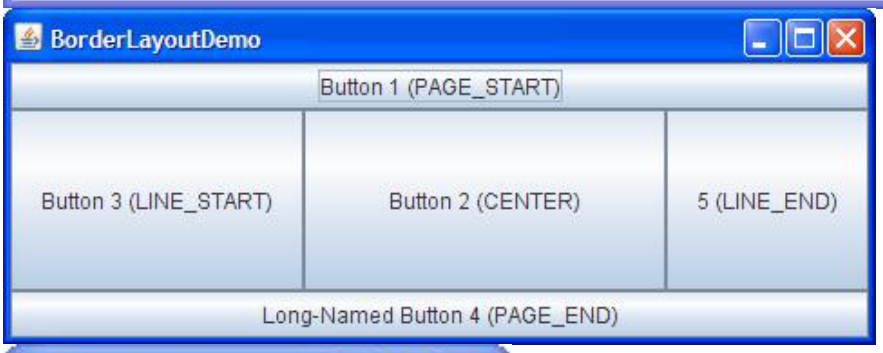
```
    // dodanie panelu p (z komponentami rozmieszczonymi przez
    // FlowLayout) do panelu, w ktorym obowiazuje BoxLayout
    mp.add(p);
    this.webpage = new JEditorPane();
    this.htmlPage = new JTextArea();
    try {
        // wczytujemy zawartosc strony “startowej”
        this.setPage(
                new URL("http://www.simongrant.org/web/guide.html"));
    } catch (IOException e) { }
    // Tworzymy panel z zakladkami
    JTabbedPane tp = new JTabbedPane();
    tp.setPreferredSize(new Dimension(600, 400));
    // pole tekstowe webpage umieszczamy wewnatrz panela
    // scrollowanego. Dzieki temu zawartosc okienka bedzie mogla
    // zajmowac wiecej miejsca niz widok
    JScrollPane sp = new JScrollPane(this.webpage);
    // zakladka “page” bedzie zawierac webpage (wewnatrz JScrollPane)
    tp.add("page", sp);
    // zakladka “html” bedzie zawierac htmlPage (wewnatrz JscrollPane)
    sp = new JScrollPane(this.htmlPage);
    tp.add("html", sp);

    // przygotowany JTabbedPane zostaje dodany do panelu mp
    mp.add(tp);
    return mp;
}
private void setPage(URL page) throws IOException {
    String s;
    this.webpage.setPage(page);
    BufferedReader br = new BufferedReader(new InputStreamReader(
                                                    page.openStream()));
    while ((s = br.readLine()) != null)
        this.htmlPage.append(s + "\n");
}

public Browser() {
    // zawsze na początku powinniśmy wywołać konstruktor nadklasy
    super();
    // zawartoscia okna Browser bedzie panel mp
    this.getContentPane().add(this.createMainPanel());
}
```

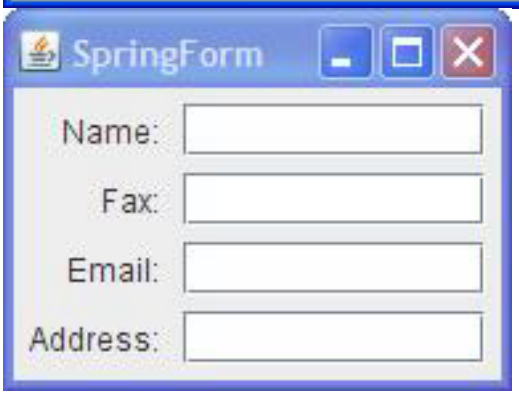
```
    public static void createAndShow() {
        Browser b = new Browser();
        b.setDefaultCloseOperation(EXIT_ON_CLOSE);
        b.pack();
        // okno zostanie umieszczone na srodku ekranu
        b.setLocationRelativeTo(null);
        b.setVisible(true);
    }

    // Interfejs ActionListener implementuje jedna metode, ktora jest
    // uruchamiana gdy nastapi zdarzenie na komponencie nasluchiwanym
    // przez ten obiekt. Informacje o zrodle akcji sa przekazywane przez
    // argument ActionEvent
    @Override
    public void actionPerformed(ActionEvent e) {
        if (COMMAND_GO.equals(e.getActionCommand())) {
            try {
                // przeladowujemy strone
                this.setPage(new URL(this.url.getText()));
            } catch (IOException e2) {
                this.webpage.setText(
                                "Problem z adresem " + this.url.getText());
                this.htmlPage.setText(
                                "Problem z adresem " + this.url.getText());
            }
        }
    }
    // uruchomienie programu
    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                createAndShow();
            }
        });
    }
}
```

`BoxLayout(mp, BoxLayout.Y_AXIS)` — zastępuje domyślny layout. Taki LayoutManager, że ustawia elementy kontenera *mp* wzdłuż osi Y, czyli **jeden pod drugim**

Przykłady działania różnych Layout'ów:







BoxLayout ogarnia w widocznym okienku poniżej zaznaczone elementy, bezpośredniej zawartości wyświetlanej strony już nie. Czyli wyświetla np. guziki *"Go"* , *"page"* , *"html"* , pole na wpisanie adresu url.

`setText()` — domyślny tekst umieszczony w pasku url
`setPage()` — pomaga w załadowaniu wezwanej strony

`setLocationRelativeTo()` – pozwala ustalić miejsce, w którym zostanie wyświetlone nasze okienko w relacji do jakiegoś innego komponentu (wpisując „null” okienko wyświetla się na środku ekranu, a gdyby tej instrukcji w ogóle nie było, okienko wyświetli się w lewym górnym rogu)

`e.getActionCommand()` – sprawdza, czy komenda związana z akcją jest równa komendzie przypisanej do jakiegoś przycisku (tutaj: czy podjęta akcja zwraca komendę zawartą w COMMAND_GO)

`url.getText()` – pobiera stringa z komponentu tekstowego *"url"*

`new URL (…)` – tworzy obiekt typu url, więcej na późniejszym wykładzie

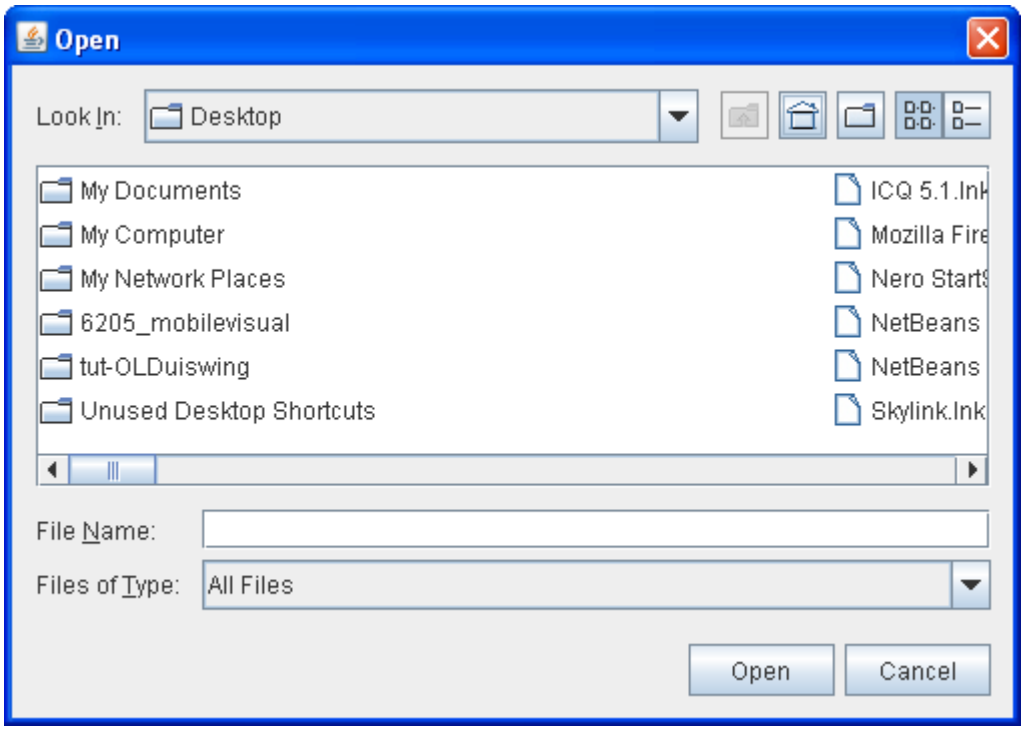
> `ActionListener` — **jest interfejs używanym do nasłuchiwania zdarzeń akcji, takich jak kliknięcie przycisku w interfejsie użytkownika**. Głównym celem ActionListenera jest reagowanie na te zdarzenia poprzez wykonanie określonych operacji. Kiedy dany obiekt implementujący ActionListener jest zarejestrowany do komponentu interfejsu użytkownika, jak na przykład przycisk, będzie on oczekiwał na zdarzenie akcji, takie jak kliknięcie, i wywoła odpowiednią metodę, np. `actionPerformed(ActionEvent e)` , w celu obsługi tego zdarzenia.

Przykład:

```java
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
public class MyFrame extends JFrame implements ActionListener {

    JButton myButton;

    public MyFrame() {
        // Inicjalizacja komponentów, w tym przycisku
        myButton = new JButton("Click me");
        myButton.addActionListener(this); // Dodanie ActionListener do
        add(myButton);

        // Konfiguracja ramki
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE) ;
        setSize(300, 200);
        setVisible(true);
    }

    // Implementacja metody actionPerformed
    @Override
    public void actionPerformed (ActionEvent e) {
        if (e.getSource() == myButton) {
            // Obsługa zdarzenia kliknięcia przycisku
            System.out.println("Button clicked!");
        }
    }
    public static void main(String[] args) {
        new MyFrame();
    }
}
```

JFileChooser

JFileChooser to komponent, który dostarcza **prosty mechanizm umożliwiający użytkownikowi wybór pliku**. Można go używać do nawigacji po systemie plików i wyboru pliku lub katalogu z listy, lub wprowadzenia nazwy pliku lub katalogu.



przykład:

```
JFileChooser chooser = new JFileChooser();
// filtr plików
FileNameExtensionFilter filter = new FileNameExtensionFilter(
                                        "Obrazy JPG i GIF", "jpg", "gif");
chooser.setFileFilter(filter);
// wyświetlenie okienka
int ret = chooser.showOpenDialog(parent);
// Jesli wcisnieto ok lub open
if(ret == JFileChooser.APPROVE_OPTION) {
    System.out.println("Wybrales plik: " +
                                        chooser.getSelectedFile().getName());
}
```

`FileNameExtensionFilter(„Obrazy JPG i GIF” , “jpg” , “gif”)` – wybieramy rozszerzenia plików, które pojawią się po wybraniu opcji *"Obrazy JPG i GIF"* w komponencie tekstowym Files of Type.

`setFileFilter()` – wstawia obiekt filter do komponentu *chooser*

`showOpenDialog(parent)` – wyświetla nasz komponent *chooser*. ***parent*** to okienko nadrzędne, jeśli go nie ma, wpisujemy ***null***. Do zmiennej *ret* zapisywana jest informacja o podjętej akcji, czy wcisnęliśmy **Ok** czy **Cancel**. (okienko jest modalne)

`showSaveDialog(parent)` – Zamiast open jest save. Konstrukcja taka sama. Służy do wskazania lokalzacji zapisu pliku.

`APPROVE_OPTION` – tutaj: przycisk **Ok**

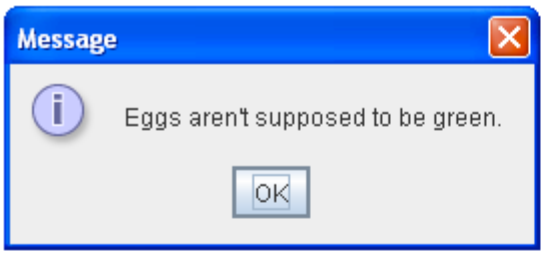
W `JFileChooser` może wybrać wiele plików, ale domyślnie wybiera jeden. **Można go ustawić tak, aby wybierał katalogi. Domyślnie otwiera się w katalogu domowym.**

`setMultiSelectionEnabled(true)` – Ustawia możliwość wybrania kilku plików.

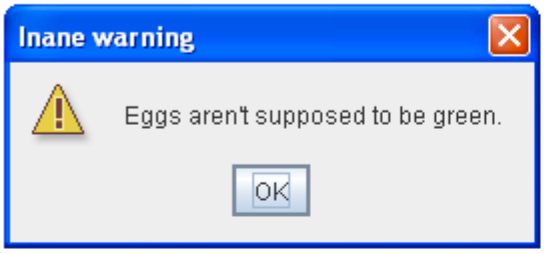
Klasa JOptionPane

JOptionPane to klasa, która umożliwia tworzenie różnych rodzajów okien dialogowych.

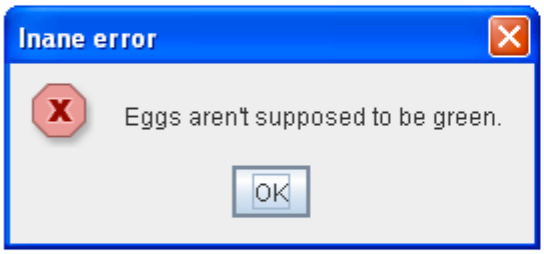
Posiada metody statyczne, które pozwalają na wyświetlanie okienek informacyjnych.
Używamy do tego metody `showMessageDialog()` .



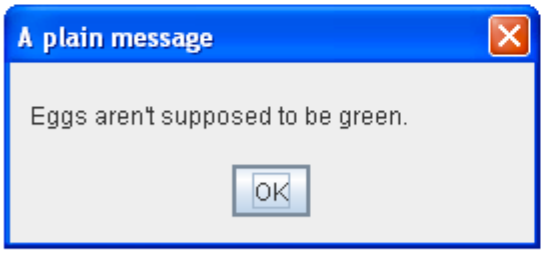

```
//default title and icon
JOptionPane.showMessageDialog(frame,
    "Eggs are not supposed to be green.");
```



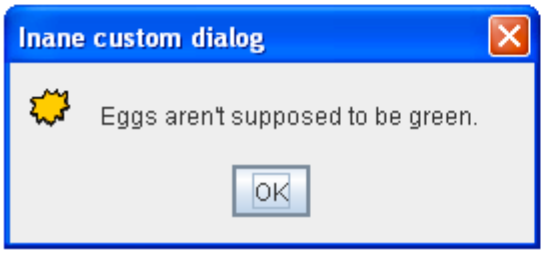

```
//custom title, warning icon
JOptionPane.showMessageDialog(frame,
    "Eggs are not supposed to be green.",
    "Inane warning",
    JOptionPane.WARNING_MESSAGE);
```




```
//custom title, error icon
JOptionPane.showMessageDialog(frame,
    "Eggs are not supposed to be green.",
    "Inane error",
    JOptionPane.ERROR_MESSAGE);
```




```
//custom title, no icon
JOptionPane.showMessageDialog(frame,
    "Eggs are not supposed to be green.",
    "A plain message",
    JOptionPane.PLAIN_MESSAGE);
```

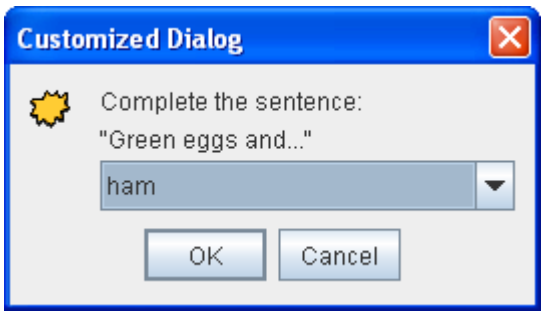



```
//custom title, custom icon
JOptionPane.showMessageDialog(frame,
    "Eggs are not supposed to be green.",
    "Inane custom dialog",
    JOptionPane.INFORMATION_MESSAGE,
    icon);
```

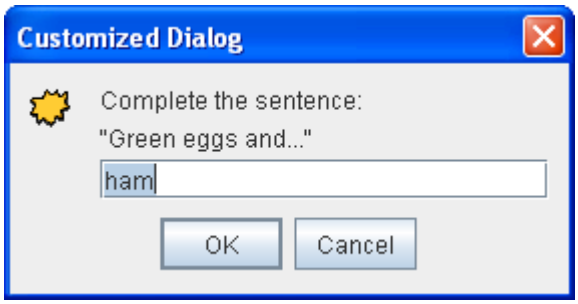
Pozwala także odebrać proste informacje od użytkownika.

Wówczas używamy metody `showInputDialog()`

```
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, // null
                    "ham");
```



Gdy ustawimy `possibilites` na **null**:



> **Podanie rodzica**, okienka nadrzędnego "frame" oznacza, ze **dane okienko jest modalne**, czyli trzeba wykonać wskazaną tam akcje, aby powrócić do okienka nadrzędnego.