

Wyższa Szkoła Informatyki Stosowanej i Zarządzania

pod auspicjami Polskiej Akademii Nauk

WYDZIAŁ INFORMATYKI

Kierunek INFORMATYKA

Studia I stopnia (dyplom inżyniera)



Język Java – wykład 7

dr inż. Łukasz Sosnowski
lukasz.sosnowski@wit.edu.pl
sosnowsl@ibspan.waw.pl
l.sosnowski@dituel.pl

www.lsosnowski.pl



Część 1 – wprowadzenie do Swing



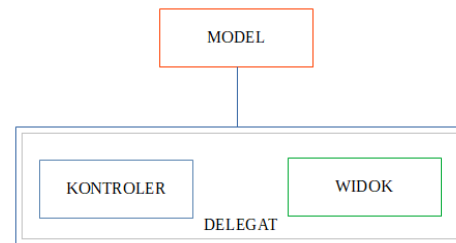
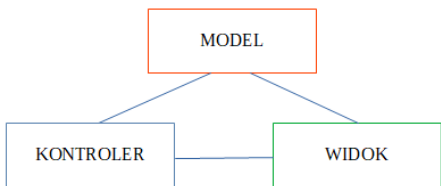
Informacje podstawowe, historia

- Swing jest drugim i historycznie nowszym podejściem do tworzenia interfejsu graficznego w JAVA. Pierwszym był pakiet AWT (Abstract Window Toolkit), który jednak zawierał liczne braki i ograniczenia głównie wynikające z wykorzystania komponentów wbudowanych w konkretne platformy. Podejście to okazało się jednak mocno zawodne, ze względu na różnorodność komponentów ich działania oraz wygląd w poszczególnych platformach.
- Swing został stworzony w celu poprawy zaistniałej sytuacji. Korzysta częściowo z AWT (klasa Component i Container), lecz w pozostałej części został stworzony na nowo.



Architektura model-delegat

- Klasyczna architektura model, widok i kontroler w której:
 - model – odpowiada stanowi komponentu;
 - widok – określa sposób wyświetlania komponentu na ekranie;
 - kontroler – określa sposób reakcji komponentu na wykon. akcje;została zmodyfikowana w taki sposób, że widok i kontroler połączono w jeden byt, zwany *delegatem interfejsu użytkownika*.
- Komponenty SWING bazują zatem na klasycznej architekturze MVC lecz nie używają jej klasycznej implementacji.





Komponenty i kontenery

- Komponent stanowi niezależną kontrolkę, posiadającą swój wygląd (np.. pole tekstowe, pole wyboru, etc)
- Kontener jest również komponentem, lecz przeznaczony do przechowywania innych komponentów (kontrolerek) w tym również innych kontenerów.
- Komponent aby mógł być wyświetlony w programie, musi należeć do jakiegoś kontenera.
- Każdy interfejs napisany w SWING musi zawierać przynajmniej jeden kontener.
- W Swing budowana jest hierarchia zawierania na bazie zagnieżdżonych kontenerów, zakończona kontenerem szczytowym.



Komponenty - szczegóły

- Komponenty Swing są klasami pochodnymi klasy JComponent (poza kontenerami szczytowymi).
- Klasa JComponent dostarcza wspólnej funkcjonalności dla wszystkich kontrolek.
- Klasa JComponent jest klasą pochodną dla klasy Container i Component z AWT. Dzięki temu komponenty Swing nadbudowują komponenty AWT i są z nimi zgodne.
- Wszystkie komponenty Swing zrealizowane są poprzez klasy znajdujące się w pakiecie javax.swing.



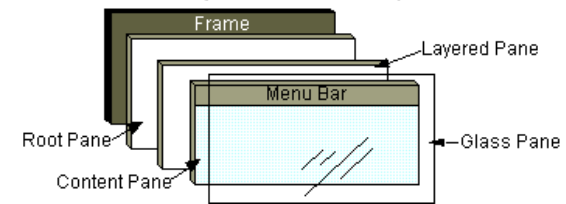
Kontenery - szczegóły

- W SWING wyróżniamy 2 typy kontenerów: szczytowe i pozostałe.
- Kontenery szczytowe to: JFrame, JApplet, JWindow, JDialog.
- Kontener JApplet od JAVA w wersji 9 jest już uznane za przestarzałe (ang. deprecated).
- Kontenery szczytowe nie pochodzą od klasy JComponent, ale pochodzą od klas AWT: Component i Container. Kontenery te nie są wyłącznie napisane w języku JAVA.
- Wszystkie pozostałe kontenery dziedziczą z JComponent i są w całości napisane w JAVA.
- Przykłady pozostałych kontenerów to: JPanel, JScrollPane, JRootPane.
- Ten typ kontenerów jest często używany do zbiorczego zarządzania grupą komponentów.



Panele kontenerów szczytowych

- Każdy kontener szczytowy definiuje zbiór paneli (ang. panes).
- Szczytem hierarchii paneli jest instancja JRootPane.
- Panel ten przeznaczony jest do zarządzania innymi panelami:
 - panelem warstw (ang. layered pane) – określa między innymi położenie w osi Z komponentu, decydujące o ew. zakrywaniu. Panel ten zawiera panel zawartości i opcjonalnie panel menu.
 - panelem zawartości (ang. content pane) – zawiera komponenty wizualne z którymi pracuje programista.
 - panelem szklanym (ang. glass pane) - domyślnie ukryty, pozwalana na zarządzanie zdarzeniami dot. całego kontenera.
 - opcjonalnym paskiem menu (ang. menu bar).

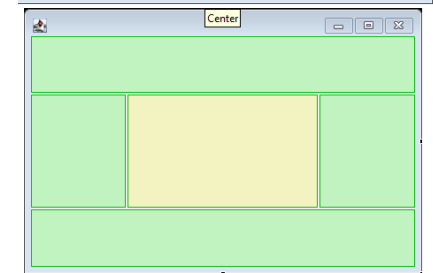
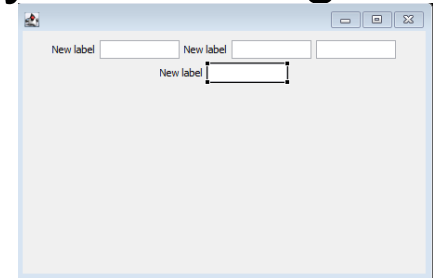


* Obrazek z dokumentacji ORACLE JAVA: <https://docs.oracle.com/javase/tutorial/uiswing/components/rootpane.html>
Język Java – dr inż. Łukasz Sosnowski



Menedżery układu

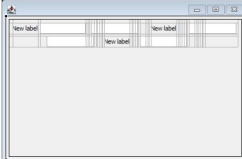
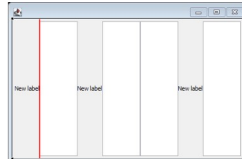
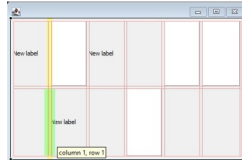
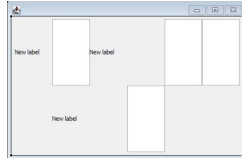
- W SWING rozmieszczenie komponentów w kontenerze kontrolowane jest poprzez menedżera układu.
- Dostępnych jest wiele menedżerów, w większości pochodzących z AWT, lecz SWING ma też kilka własnych.
- Wszystkie menedżery układu są instancjami klas implementujących interfejs `LayoutManager` lub `LayoutManager2`.
- `FlowLayout` – układ komponentów od lewej do prawej i z góry na dół, tzw. układ pływający. Po zmianie wielkości okna rozmieszczenie ulega zmianie.
- `BorderLayout` – układ komponentów: centralnie, wschód, zachód, północ, południe.





Menedżery układu c.d.

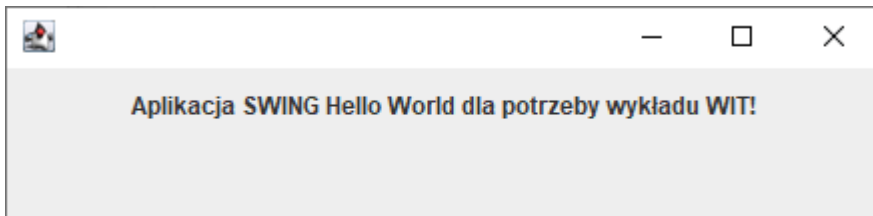
- GridLayout – rozmieszcza komponenty na siatce. Wszystkie komponenty mają identyczne rozmiary.
- GridBagLayout – rozmieszcza komponenty o różnych rozmiarach na elastycznej siatce.
- BoxLayout – rozmieszcza komponenty pionowo lub poziomo wewnątrz boksu.
- MigLayout - używa siatki (wierszy i kolumn) z automatyczną obsługą luk między komponentami.
- Inne menedżery: TableLayout, GroupLayout, SpringLayout, CardLayout
- W dokumentacji można znaleźć szczegółowy opis każdego z układów oraz wizualny szkic:
<https://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html>





Przykład programu w SWING

```
public class WITFrameExample extends JFrame {  
    private JPanel contentPane;  
  
    public WITFrameExample() {  
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //Zamknięcie okna kończy działanie programu  
        setBounds(100, 100, 452, 113); //Określa pozycję i rozmiar okna  
        contentPane = new JPanel(); //Tworzy panel zawartości  
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5)); //Określa pozycję i rozmiar panelu  
        setContentPane(contentPane); //Ustawia panel  
  
        contentPane.setLayout(new FlowLayout(FlowLayout.CENTER, 5, 5)); //Ustawia menedżera układu  
  
        JLabel lblNewLabel = new JLabel("Aplikacja SWING Hello World dla potrzeby wykładu WIT!");  
        contentPane.add(lblNewLabel); //Dodaje do panelu komponent etykiety  
        setVisible(true); //Uwidocznienie okna (domyślnie jest ukryte)  
    }  
}
```





Obsługa zdarzeń

- Aplikacje SWING sterowane są zdarzeniami wynikającymi z interakcji użytkownika z programem, poprzez czas lub inne elementy.
- W momencie przesłania zdarzenia do programu, zostaje ono przekazane do procedury obsługi zdarzeń.
- SWING wykorzystuje *delegacyjny model zdarzeń*, tzn. taki w którym *źródło* generuje *zdarzenie* i przesyła je do *obiektów nasłuchujących* (jednego lub wielu).
- Obiekty nasłuchujące oczekują na odebranie zdarzenia.
- Po odbiorze obiekt nasłuchujący dokonuje obsługi zdarzenia i zwraca sterowanie.
- Delegacja obsługi zdarzenia następuje od interfejsu użytkownika do innego fragmentu kodu.



Obsługa zdarzeń c.d.

- **Zdarzenie** – to obiekt opisujący zmianę stanu źródła zdarzenia. Mogą być generowane przez interakcję użytkownika lub programowo przez aplikację. Klasa bazowa wszystkich zdarzeń to `java.util.EventObject`.
- **Źródło zdarzenia** – to obiekt generujący zdarzenie. Zdarzenia są rozsyłane do obiektów nasłuchujących, które zarejestrowały się w danym źródle. Rejestracja w źródle następuje poprzez wywołanie dedykowanej metody źródła dla poszczególnych typów. Metody te zazwyczaj mają nazwy zgodne ze schematem: `add[Typ]Listener`, gdzie `Typ` oznacza konkretny typ dla którego metoda jest dedykowana, np.. akcja, etc.



Obsługa zdarzeń c.d.

- **Obiekty nasłuchujące** – to obiekt, który jest informowany o zajściu określonego zdarzenia i który implementuje metodę do odbioru i obsługi zdarzenia. Metody odbierające i obsługujące zostały zdefiniowane w interfejsach znajdujących się w pakietach: *java.awt.event* oraz *javax.swing.event*.
- **Klasy zdarzeń** – klasą bazową wszystkich zdarzeń jest *java.awt.EventObject*. Swing używa zdarzeń AWT, ale definiuje też kilka własnych umieszczonych w pakiecie *javax.swing.event*.
- Wybrane klasy zdarzeń: *ActionEvent* – zdarzenie generowane po akcji w kontrolce, np.. kliknięcie przycisku., *ItemEvent* – generowane po zaznaczeniu elementu, np. pola wyboru, *ListSelectionEvent* – generowane po zmianie elementu na liście/
- **Wybrane interfejsy ob. nasł.** - *ActionListener*, *ItemListener*, *ListSelectionListener*.



Komponent JLabel

- Klasa dostarcza kilku konstruktorów w zależności od funkcjonalności, która do której obiekt ma być użyty: wyświetlenie tekstu, ikony lub tekstu i ikony.
- Komponent nie generuje zdarzeń.
- Komponent zazwyczaj towarzyszący innemu komponentowi obsługującemu zdarzenia.
- Dostarcza dwie podstawowe metody *void setText(String text)* i *String getText()*, które odpowiednio ustawiają tekst do wyświetlenia i pobierają ustawiony tekst.
- Stanowi najprostszy komponent wizualny SWING.



Komponent JButton

- JButton jest klasą pochodną klasy *AbstractButton* definiującej wspólne funkcjonalności dla wszystkich przycisków w SWING.
- Przyciski mogą zawierać tekst, obrazek lub tekst i obrazek.
- Klasa udostępnia kilka konstruktorów, jednym z nich jest *JButton(String msg)*.
- Naciśnięcie przycisku generuje zdarzenie *ActionEvent*, dlatego klasa dostarcza metody *void addActionListener(ActionListener al)* do rejestracji obiektów nasłuchujących oraz *void removeActionListener(ActionListener al)* do usuwania obiektów.
- Parametrem metod jest obiekt klasy implementującej interfejs *ActionListener*. Interfejs ten definiuje metodę *void actionPerformed(ActionEvent ae)*, która musi być zaimplementowana w klasie obiektu nasłuchującego, w celu obsługi zdarzenia.

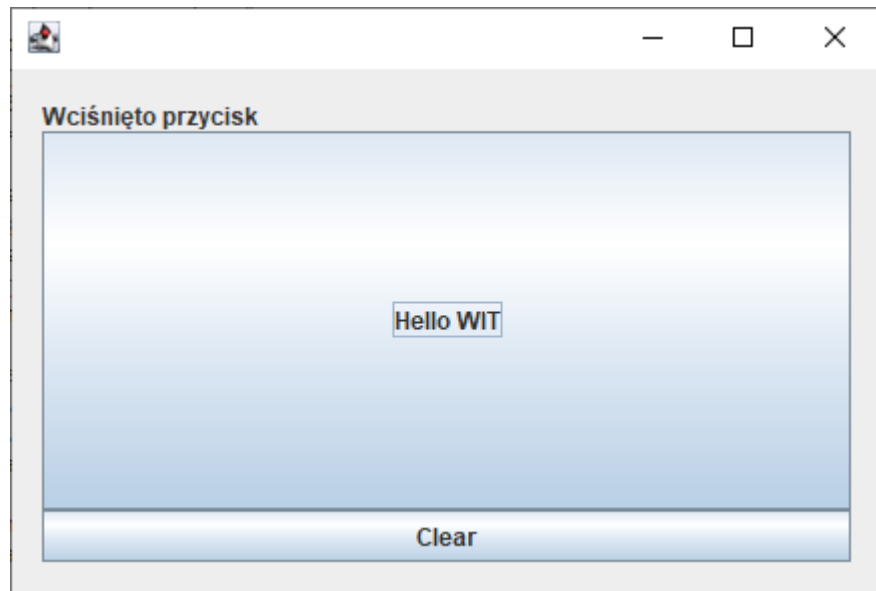


Przykład

```
public class MyButtonExample extends JFrame implements ActionListener
{
    private JPanel contentPane;
    private JLabel lblResult;
    private JButton btnClear;

    public MyButtonExample() {
        //Zamknięcie okna kończy działanie programu
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        //Pozycja i rozmiar okna
        setBounds(100, 100, 450, 300);
        contentPane = new JPanel();
        //Ramka
        contentPane.setBorder(new EmptyBorder(15, 15, 15, 15));
        //Menedżer układu
        contentPane.setLayout(new BorderLayout(0, 0));
        //Panel zawartości
        setContentPane(contentPane);
        //Przycisk
        JButton btnNewButton = new JButton("Hello WIT");
        btnNewButton.setActionCommand("btnWIT");
        btnNewButton.addActionListener(this);
        contentPane.add(btnNewButton, BorderLayout.CENTER);
        //Etykieta
        lblResult = new JLabel("");
        contentPane.add(lblResult, BorderLayout.NORTH);
        //Przycisk
        btnClear = new JButton("Clear");
        btnClear.addActionListener(this);
        btnClear.setActionCommand("btnCLR");
        contentPane.add(btnClear, BorderLayout.SOUTH);
    }
}
```

```
public void actionPerformed(ActionEvent e) {
    //Obsługa wciśnięcia przycisku HelloWIT
    if(e.getActionCommand().equals("btnWIT")) {
        lblResult.setText("Wciśnięto przycisk");
        //Obsługa wciśnięcia przycisku Clear
    }else if(e.getActionCommand().equals("btnCLR")){
        lblResult.setText("");
    }
}
```





Komponent `JTextField`

- Klasa pochodna klasy *JTextComponent* będącej klasą bazową dla wszystkich komponentów tekstowych.
- Komponent umożliwiający wprowadzenie wiersza tekstu.
- Dostarcza kilka konstruktorów, jednym z nich jest *JTextField(int cols)*, gdzie *cols* określa szerokość pola tekstowego w kolumnach.
- Komponent generuje zdarzenie *ActionEvent* po wciśnięciu ENTER przez użytkownika podczas wprowadzania tekstu w polu.
- Klasa dostarcza metodę do rejestracji i usuwania obiektu nasłuchującego w postaci: *void addActionListener()* i *void removeActionListener()*.
- Komponent udostępnia metodę *setActionCommand(String)* gdzie można ustalić wartość związaną z komponentem przekazywaną do zdarzenia.



Komponent JTextArea

- Klasa JTextArea udostępnia komponent, który wyświetla wiele wierszy tekstu i opcjonalnie umożliwia użytkownikowi edycję tekstu.
- Udostępnia konstruktor *JTextArea(int rows, int cols)* w którym można zdefiniować rozmiar. Udostępnia też inne wersje konstruktorów.
- Klasa udostępnia metody `void setEditable(boolean)` `boolean isEditable()`, za pomocą których można włączyć/wyłączyć edytowalność pola oraz sprawdzić stan tego ustawienia.
- Komponent może generować zdarzenie *DocumentEvent* za pomocą których można monitorować zmiany tekstu w polu. Wymaga obiektu nasłuchu którego klasa implementuje interfejs *DocumentListener*.



Przykład

```
public class MyTextFieldExample extends JFrame implements ActionListener{
    private JPanel contentPane;
    private JPasswordField passPassword1;
    private JPasswordField passPassword2;
    private JTextField textFirstName;
    private JTextField textSurname;
    private JTextField textLogin;
    private JTextArea textDescription;

    public MyTextFieldExample() {
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setBounds(100, 100, 450, 262);
        contentPane = new JPanel();
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
        setContentPane(contentPane);
        contentPane.setLayout(new MigLayout("", "[29px][86px,grow]", "[20px][][][][][grow][][][][]"));
        JLabel lblNewLabel = new JLabel("Login:");
        lblNewLabel.setToolTipText("Podaj swój preferowany login");
        contentPane.add(lblNewLabel, "cell 0 0,alignx trailing,aligny center");
        textLogin = new JTextField();
        contentPane.add(textLogin, "cell 1 0,growx");
        textLogin.setColumns(10);
        JLabel lblNewLabel_1 = new JLabel("Hasło:");
        contentPane.add(lblNewLabel_1, "cell 0 1,alignx trailing");
        passPassword1 = new JPasswordField();
        contentPane.add(passPassword1, "cell 1 1,growx");
        JLabel lblNewLabel_2 = new JLabel("Powtórz hasło:");
        contentPane.add(lblNewLabel_2, "cell 0 2,alignx trailing");
        passPassword2 = new JPasswordField();
        contentPane.add(passPassword2, "cell 1 2,growx");
        JLabel lblNewLabel_3 = new JLabel("Imię:");
        contentPane.add(lblNewLabel_3, "cell 0 3,alignx trailing,aligny center");
        textFirstName = new JTextField();
        contentPane.add(textFirstName, "cell 1 3,growx");
        textFirstName.setColumns(10);
    }
}
```

login=lukasz.sosnowski
hasło=hasło1
hasło2=hasło2
imię=Łukasz
nazwisko=Sosnowski
opis=Administrator aplikacji



Komponent JCheckBox

- Pole wyboru dające możliwość zaznaczenia danej opcji.
- Dostarcza kilka konstruktorów, jednym z nich jest `JCheckBox(String str)`, który ustawia tekst wyświetlany w interfejsie w danym komponencie.
- Zmiana stanu komponentu (zaznaczenie lub odznaczenie) generuje zdarzenie klasy `ItemEvent`.
- Obiekt nasłuchujący musi być klasy implementującej interfejs `ItemListener`.
- Klasa musi dostarczać implementację metody `void itemStateChanged(ItemEvent ie)`. W celu uzyskania referencji do komponentu który wygenerował zdarzenie należy użyć metody `Object getItem()` i rzutować ją na odpowiednią klasę komponentu.

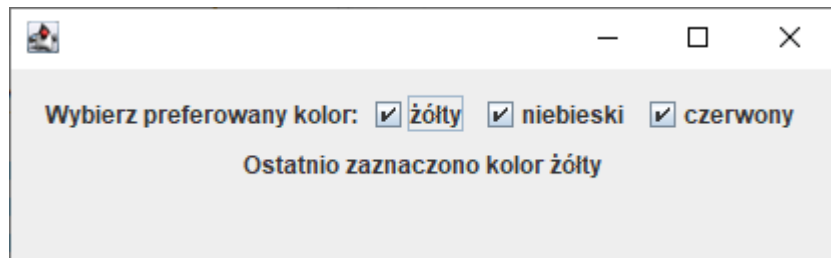


Przykład

```
public class MyCheckboxExample extends JFrame implements ItemListener {
    private JPanel contentPane;
    JLabel lblResult;

    public MyCheckboxExample() {
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setBounds(100, 100, 428, 136);
        contentPane = new JPanel();
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
        setContentPane(contentPane);
        contentPane.setLayout(new FlowLayout(FlowLayout.CENTER, 5, 5));
        JLabel lblNewLabel = new JLabel("Wybierz preferowany kolor:");
        contentPane.add(lblNewLabel);
        JCheckBox chckbxYellow = new JCheckBox("żółty");
        chckbxYellow.addItemListener(this);
        chckbxYellow.setActionCommand("CMD_YELLOW");
        contentPane.add(chckbxYellow);
        JCheckBox chckbxBlue = new JCheckBox("niebieski");
        chckbxBlue.addItemListener(this);
        chckbxYellow.setActionCommand("CMD_BLUE");
        contentPane.add(chckbxBlue);
        JCheckBox chckbxRed = new JCheckBox("czerwony");
        chckbxRed.addItemListener(this);
        chckbxYellow.setActionCommand("CMD_RED");
        contentPane.add(chckbxRed);
        lblResult = new JLabel("");
        contentPane.add(lblResult);
    }

    public void itemStateChanged(ItemEvent ie) {
        JCheckBox cb = (JCheckBox) ie.getItem();
        lblResult.setText("Ostatnio " + (cb.isSelected() ? "zaznaczono " : "odznaczono ") + "kolor " + cb.getText());
        cb.getText();
    }
}
```





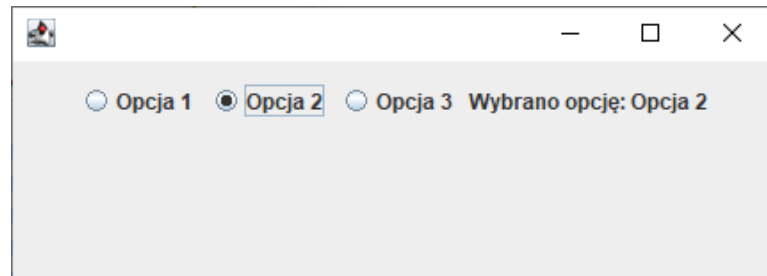
Komponent JRadioButton

- Klasa dostarcza funkcjonalności przycisków opcji, które wzajemnie się wykluczają.
- Dostarcza kilka konstruktorów w tym JRadioButton(String str), gdzie parametr str ustawia etykietę opcji.
- Dodatkowo obiekty tej klasy trzeba dodać do grupy przycisków, tzn. obiektu klasy ButtonGroup za pomocą metody add(AbstractButton ab).
- Klasa komponentu JRadioButton generuje zdarzenia akcji (ActionEvent), zdarzenia elementów (ItemEvent) i zdarzenia zmian (ChangeEvent)
- Do identyfikacji klikniętego elementu można użyć kilku różnych technik, m.in. z użyciem metody getActionCommand.



Przykład

```
public class MyRadiosExample extends JFrame implements ActionListener {  
    private JPanel contentPane;  
    JLabel lblResult;  
    public MyRadiosExample() {  
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        setBounds(100, 100, 443, 160);  
        contentPane = new JPanel();  
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));  
        setContentPane(contentPane);  
        contentPane.setLayout(new FlowLayout(FlowLayout.CENTER, 5, 5));  
        JRadioButton rdbtnOption1 = new JRadioButton("Opcja 1");  
        rdbtnOption1.addActionListener(this);  
        contentPane.add(rdbtnOption1);  
        JRadioButton rdbtnOption2 = new JRadioButton("Opcja 2");  
        rdbtnOption2.addActionListener(this);  
        contentPane.add(rdbtnOption2);  
        JRadioButton rdbtnOption3 = new JRadioButton("Opcja 3");  
        rdbtnOption3.addActionListener(this);  
        contentPane.add(rdbtnOption3);  
        ButtonGroup bg = new ButtonGroup();  
        bg.add(rdbtnOption1);  
        bg.add(rdbtnOption2);  
        bg.add(rdbtnOption3);  
        lblResult = new JLabel("");  
        contentPane.add(lblResult);  
    }  
  
    @Override  
    public void actionPerformed(ActionEvent e) {  
        lblResult.setText("Wybrano opcję: "+e.getActionCommand());  
    }  
}
```





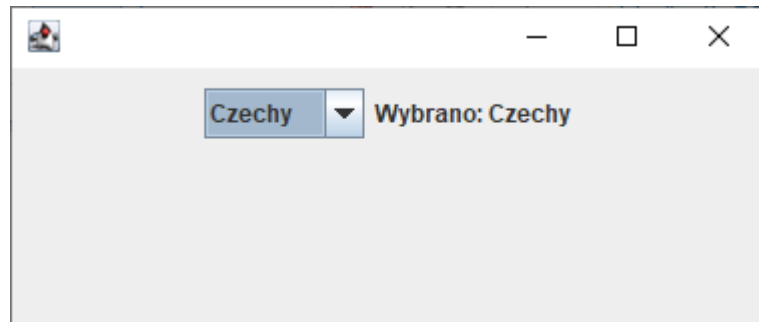
Komponent JComboBox

- Dostarcza funkcjonalności tzw. listy kombinowanej, tzn. połączenia pola tekstowego z listą rozwijaną.
- Klasa ta jest parametryzowana typem obiektu przetwarzanego przez komponent.
- Jednym z konstruktorów jest JComboBox(T[] items) przyjmujący w argumencie tablice obiektów to wyświetlenia na liście.
- Komponent umożliwia dynamiczne dodawanie elementów z użyciem metody addItem(T obj).
- Generuje zdarzenie akcji w momencie wybrania elementu listy. Generuje również zdarzenie elementu w momencie zaznaczenia elementu na liście lub usunięcia zaznaczenia.
- Uzyskanie informacji o zaznaczonym elemencie może nastąpić poprzez metodę Object.getSelectedItem().



Przykład

```
public class MyComboBoxExample extends JFrame {  
  
    private JPanel contentPane;  
    private String countries[] = new String[] {"----", "Polska", "Czechy", "Słowacja", "Węgry"};  
    public MyComboBoxExample() {  
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        setBounds(100, 100, 450, 300);  
        contentPane = new JPanel();  
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));  
        setContentPane(contentPane);  
        contentPane.setLayout(new FlowLayout(FlowLayout.CENTER, 5, 5));  
        JLabel lblNewLabel = new JLabel("");  
        JComboBox<String> comboCountries = new JComboBox<String>(countries);  
        comboCountries.addActionListener(new ActionListener() {  
  
            @Override  
            public void actionPerformed(ActionEvent e) {  
                if (comboCountries.getSelectedItem().toString().equals("----"))  
                    lblNewLabel.setText("Nic nie wybrano");  
                else  
                    lblNewLabel.setText("Wybrano: " + comboCountries.getSelectedItem().toString());  
            }  
        });  
        contentPane.add(comboCountries);  
        contentPane.add(lblNewLabel);  
    }  
}
```





Komponent JList

- Klasa reprezentująca komponent listy umożliwiającej wybór pojedynczego elementu lub wielu.
- Klasa od wersji JDK7 jest sparametryzowana, dzięki czemu lepiej kontroluje otypowanie elementów do wyświetlenia.
- Dostarcza kilku konstruktorów, m.in. `JList(T[] items)`, gdzie `T` jest typem sparametryzowanym obiektów listy, a `items` tablicą elementów typu `T`.
- Listę najczęściej opakowuje się kontenerem `JScrollPane` regulującym przewijanie zawartości.
- Komponent `JList` generuje zdarzenie `ListSelectionEvent` przy wyborze elementu listy lub usunięciu wyboru.
- Interfejs wymagany dla klasy obiektu nasłuchującego to `ListSelectionListener` oraz implementacja metody `valueChanged()`.



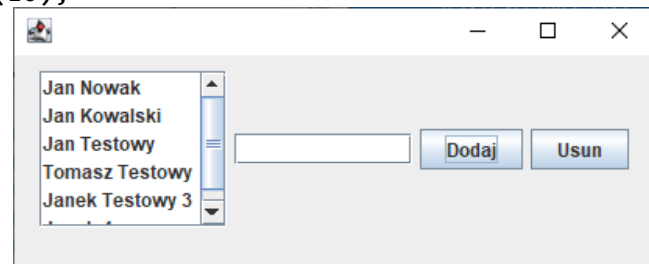
Komponent JList c.d.

- Za pomocą metody `void setSelectionMode(int mode)` można ustawić tryb wyboru elementów z listy. Dostępne są trzy: `SINGLE_SELECTION` – umożliwiający zaznaczenie tylko jednego elementu listy, `SINGLE_INTERVAL_SELECTION` – umożliwia wybór jednego zakresu elementów listy, `MULTIPLE_INTERVAL_SELECTION` – umożliwia wybór wielu zakresów elementów. Ten ostatni tryb jest domyślny.
- Metoda `getSelectedIndex` zwraca indeks pierwszego elementu zaznaczonego na liście. Jeśli brak zaznaczenia to -1.
- Metoda `int[] getSelectedIndices()` zwraca wszystkie zaznaczone pozycje elementów na liście. W przypadku braku zaznaczenia zostanie zwrócona tablica o rozmiarze 0.
- Lista może wykorzystywać dowolne obiekty, należy zapewnić metodę `toString()`.



Przykład

```
public class MyListExample extends JFrame {  
    private List<String> items = new ArrayList<String>();  
    private JPanel contentPane;  
    private JTextField textField;  
    public MyListExample() {  
        items.add("Jan Nowak"); items.add("Jan Kowalski"); items.add("Jan Testowy"); items.add("Tomasz Testowy");  
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        setBounds(100, 100, 450, 300);  
        contentPane = new JPanel();  
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));  
        setContentPane(contentPane);  
        contentPane.setLayout(new FlowLayout(FlowLayout.CENTER, 5, 5));  
        JList<String> list = new JList<String>(items.toArray(new String[0]));  
        list.setSelectionMode(ListSelectionModel.SINGLE_INTERVAL_SELECTION);  
        JScrollPane scroll = new JScrollPane(list);  
        scroll.setPreferredSize(new Dimension(120, 100));  
        contentPane.add(scroll);  
        textField = new JTextField(); contentPane.add(textField); textField.setColumns(10);  
        JButton btnAdd = new JButton("Dodaj");  
        btnAdd.addActionListener(new ActionListener() {  
            public void actionPerformed(ActionEvent e) {  
                if(!textField.getText().equals(""))  
                    items.add(textField.getText());  
                textField.setText("");  
                list.setListData(items.toArray(new String[0])); } });  
        contentPane.add(btnAdd);  
        JButton btnDel = new JButton("Usun");  
        btnDel.addActionListener(new ActionListener() {  
            public void actionPerformed(ActionEvent e) {  
                Integer[] selection = Arrays.stream(list.getSelectedIndices()).boxed().toArray(Integer[]::new);  
                if(selection.length > 0) {  
                    Arrays.sort(selection, Collections.reverseOrder());  
                    for(int i: selection)  
                        items.remove(i);  
                }  
                list.setListData(items.toArray(new String[0])); } });  
        contentPane.add(btnDel);  
    }  
}
```





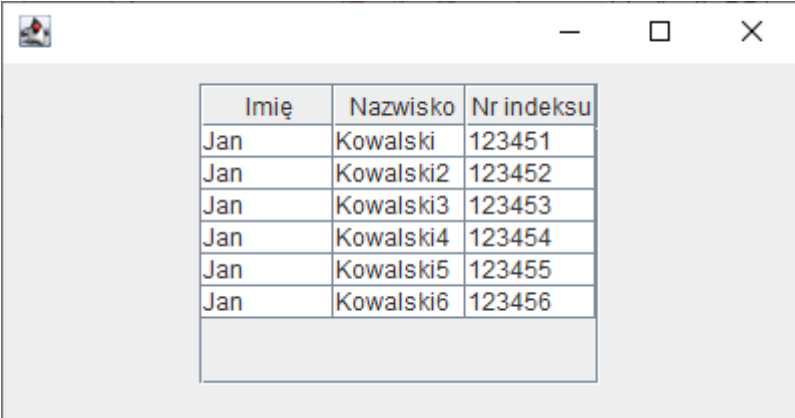
Komponent JTable

- Komponent o bardzo dużych możliwościach funkcjonalnych do wyświetlania danych w postaci tabeli. Umożliwia przestawianie kolumn zarówno programowo jak i przez użytkownika. Kolumny mogą być jako tekst, obrazki. Daje możliwość sortowania wierszy w kolumnach, etc.
- Podobnie jak lista, w większości przypadków ten komponent również opakowany jest poprzez JScrollPane zapewniający funkcjonalność przewijania.
- Dostarcza kilka konstruktorów. Jednym z prostszych sposobów utworzenia jest podanie tablicy nagłówek kolumn oraz tablicę dwuwymiarową danych do wyświetlenia.
- Podobnie jak lista daje 3 tryby pracy z zaznaczaniem



Przykład

```
public class MyTableExample extends JFrame {  
  
    private JPanel contentPane;  
    private JTable table;  
    private String[] cols = new String[] {"Imię", "Nazwisko", "Nr indeksu"};  
    private Object[][] data= {  
        {"Jan", "Kowalski", "123451"},  
        {"Jan", "Kowalski2", "123452"},  
        {"Jan", "Kowalski3", "123453"},  
        {"Jan", "Kowalski4", "123454"},  
        {"Jan", "Kowalski5", "123455"},  
        {"Jan", "Kowalski6", "123456"},  
    };  
  
    public MyTableExample() {  
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        setBounds(100, 100, 450, 300);  
        contentPane = new JPanel();  
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));  
        setContentPane(contentPane);  
        contentPane.setLayout(new FlowLayout(FlowLayout.CENTER, 5, 5));  
  
        table = new JTable(data, cols);  
        JScrollPane scroll = new JScrollPane(table);  
        scroll.setPreferredSize(new Dimension(200, 150));  
        contentPane.add(scroll);  
    }  
}
```



Imię	Nazwisko	Nr indeksu
Jan	Kowalski	123451
Jan	Kowalski2	123452
Jan	Kowalski3	123453
Jan	Kowalski4	123454
Jan	Kowalski5	123455
Jan	Kowalski6	123456



Zestawienie Nazwa klas komponentów Swing

JApplet	JButton	JCheckBox	JCheckBoxMenuItem
JColorChoser	JComboBox	JComponent	JDesktopPane
JDialog	JEditorPANE	JFileChooser	JFormattedTextField
JFrame	JInternalFrame	JLabel	JLayer
JLayeredPane	JList	JMenu	JMenuBar
JMenuItem	JOptionPane	JPanel	JPasswordField
JPopupMenu	JProgressBar	JRadioButton	JRadioButtonMenuItem
JRootPane	JScrollBar	JScrollPane	JSeparator
JSlider	JSpinner	JSplitPane	JTabbedPane
JTable	JTextArea	TextField	JTextPane
JToggleButton	JToolBar	JToolTip	JTree
JViewport	JWindow		



Część 2 – Komparatory w języku JAVA



Komparator

- Komparator to obiekt klasy implementującej interfejs `Comparator`.
- Interfejs `Comparator<T>` jest parametryzowany typem obiektu, który poddawany jest porównywaniu.
- Do JDK 8 interfejs określał jedynie 2 metody: `equals(Object obj)` oraz `compare(T obj1, T obj2)`. Pierwsza metoda określa wartość relacji większości między przekazanymi obiektami. Zwraca 0 jeśli obiekty są sobie równe, wartość dodatnią jeśli `obj1` jest większy niż `obj2` i wartość ujemną jeśli `obj2` jest większy niż `obj1`.
- Od JDK 8 pojawiło się znacznie więcej metod dostarczanych wraz z implementacjami domyślnymi:
 - `reversed()` - zwraca komparator przeciwny
 - `reverseOrder()` - zwraca komparator odwracający naturalną kolejność.



Komparator c.d.

- Metod dostarczane wraz z implementacjami domyślnymi c.d.:
 - `naturalOrder()`-zwraca komparator naturalnego porządku.
 - `nullsFirst`-zwraca komparator traktujące wartości `null` jako mniejsze od pozostałych.
 - `nullsLast()` - zwraca komparator traktujący wartości `null` jako większe od pozostałych.
 - `thenComparing` – zwraca komparator wykonujący kolejne porównanie jeśli wynik pierwszego zwróci informację że obiekty są sobie równe.
- Inne dedykowane dla typów prostych: `int`, `long` czy `double`.



Przykład

```
@Test
public void standardComparatorsTest() {
    Consumer<String> console = (n) -> System.out.println(n);
    List<String> list = new ArrayList<String>();
    list.add("C");
    list.add("F");
    list.add("A");
    list.add("B");
    list.add("D");
    list.add("E");

    for(String elem:list)
        console.accept(elem);
    console.accept("-----");
    List<String> list2 = new ArrayList<>(list);
    Collections.sort(list2,Comparator.reverseOrder());
    assertEquals("[F, E, D, C, B, A]",list2.toString());
    for(String elem:list2)
        console.accept(elem);

    console.accept("-----");
    Collections.sort(list2,Comparator.naturalOrder());
    for(String elem:list2)
        console.accept(elem);
    assertEquals("[A, B, C, D, E, F]",list2.toString());
}
```

Konsola

```
C
F
A
B
D
E
-----
F
E
D
C
B
A
-----
A
B
C
D
E
F
```



Przykład

```
@Test
public void ownComparatorsTest() {
    Consumer<String> console = (n) -> System.out.println(n);
    List<String> list = new ArrayList<String>();
    list.add("C A");
    list.add("F V");
    list.add("A W");
    list.add("B W");
    list.add("D Q");
    list.add("E E");
    Comparator<String> secondDesc = (a,b) ->{
        String[] arrA = a.split(" ");
        String[] arrB = b.split(" ");
        return arrB[arrB.length-1].compareTo(arrA[arrA.length-1]);
    };
    Comparator<String> finalComp =
secondDesc.thenComparing(Comparator.reverseOrder());
    for(String elem:list)
        console.accept(elem);

    assertEquals("[C A, F V, A W, B W, D Q, E E]",list.toString());
    Collections.sort(list,finalComp);
    console.accept("-----");
    for(String elem:list)
        console.accept(elem);
    assertEquals("[B W, A W, F V, D Q, E E, C A]",list.toString());
}
```

Konsola

```
C A
F V
A W
B W
D Q
E E
-----
B W
A W
F V
D Q
E E
C A
```



Podsumowanie

- Aplikacje okienkowe w SWING – podstawy
- Przegląd komponentów dostępnych w SWING:
 - JButton
 - JTextField
 - JTextArea
 - JList
 - JCheckbox
 - JRadioButton
 - JTable
 - I inne
- Komparatory w JAVA

Wyższa Szkoła Informatyki Stosowanej i Zarządzania

pod auspicjami Polskiej Akademii Nauk

WYDZIAŁ INFORMATYKI

Kierunek INFORMATYKA

Studia I stopnia (dyplom inżyniera)



Dziękuję za uwagę!