Programowanie obiektowe

Wykład 12: Interfejs użytkownika

dr inż. Marcin Luckner
mluckner@mini.pw.edu.pl

Wydział Matematyki i Nauk Informacyjnych

Wersja 1.3
4 marca 2021

Fundusze Europejskie Wiedza Edukacja Rozwój
Rzeczpospolita Polska
Politechnika Warszawska
Unia Europejska Europejski Fundusz Społeczny

Projekt „NERW 2 PW. Nauka – Edukacja – Rozwój – Współpraca” współfinansowany jest ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

Zadanie 10 pn. „Modyfikacja programów studiów na kierunkach prowadzonych przez Wydział Matematyki i Nauk Informacyjnych”, realizowane w ramach projektu „NERW 2 PW. Nauka – Edukacja – Rozwój – Współpraca”, współfinansowanego jest ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

Organizacja Interfejsu użytkownika

- Interfejs użytkownika jest budowany z kontrolek.
- Kontrolki nadpisują klasę `JComponent`, dzięki czemu zapewniają współdzieloną funkcjonalność.
- Rozmieszczenie kontrolek w interfejsie jest określana przez menadżera układu.
- Chociaż środowiska programistyczne wspomagają tworzenie IU warto wiedzieć na jakich zasadach jest on budowany.

Klasa `JComponent`

- Oparcie kontrolek o klasę `JComponent` zapewnia między innymi mechanizmy do
  - Generowania podpowiedzi,
  - Rysowania i wymiarowania,
  - Generowania przenośnego i natywnego wyglądu,
  - Wsparcia dla menadżerów układów,
  - Wsparcia dla przeciągnij i upuść,
  - Podwójnego buforowania,
  - Przypisywania klawiszy.

Wzorzec Model-Widok-Kontroler

- Interfejs Swing realizuje wzorzec MVC Model-Widok-Kontroler.

  Model - stan obiektu (treść),
  Widok - sposób prezentacji obiektu (wygląd),
  Kontroler - sposób obsługi zdarzeń (zachowanie).

- Każdy z tych elementów określamy niezależnie, co daje nam wiele możliwości
  - Możemy wywoływać tę samą akcję z menu i z przycisku.
  - Możemy użyć tych samych kontrolek do wyświetlania różnych danych.
  - Możemy zmienić wygląd całego interfejsu bez zmieniania jego funkcjonalności.

Komponent JList

- Komponent JList wyświetla teksty i umożliwia ich wybór.
- Zbiór tekstów stanowi modyfikowalny Model powiązany z listą

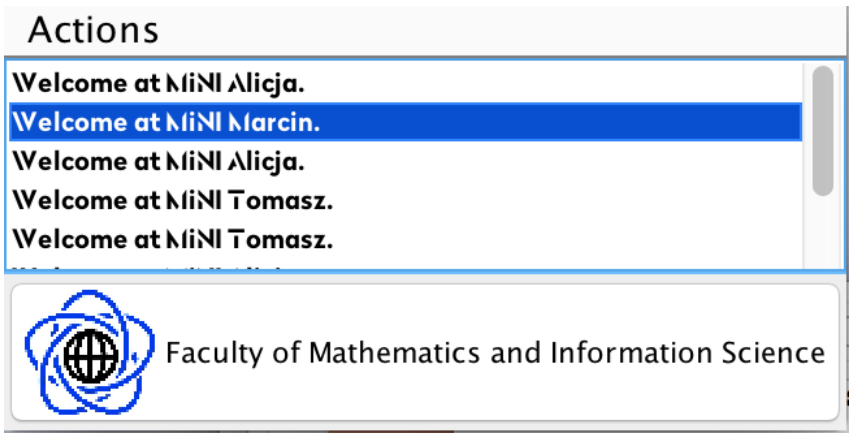
```
1   JList miniList= new JList(new DefaultListModel());
2    ((DefaultListModel)miniList.getModel()).addElement(item);
```

- Model prezentowany jest przez Widok określany w parametrach listy

```
1   this.setFont(new Font("Radikal WUT", Font.BOLD, 10));
```

- Interakcję z listą określa Kontroler będący obsługą zdarzeń

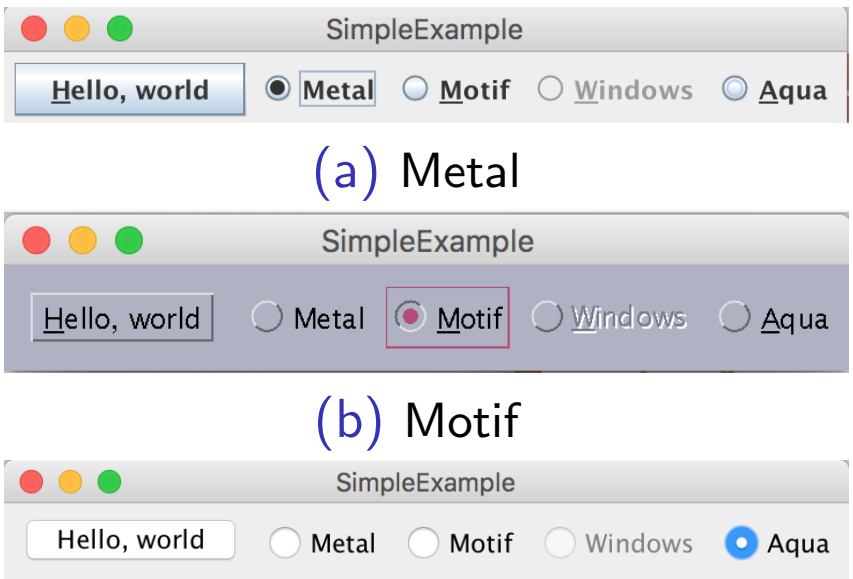
Implementacja MVC



Rysunek 1: Lista jako implementacja MVC

- Model został zaprezentowany w Widoku przy użyciu określonej czcionki.
- Kontroler obsługuje pasek przewijania i możliwość wyboru elementów z listy.
- Widok określa jak pokazać zaznaczenie elementów.

Style wyświetlania



(a) Metal

(b) Motif

(c) Aqua

Rysunek 2: Zastosowanie styli

- Możemy zmieniać wygląd komponentów w zależności od potrzeb korzystając ze stylów (Feel and Look).
- Mamy do dyspozycji style natywne
  - Aqua
  - Windows
- Oraz style tworzone przez Javę
  - Metal
  - Motif
- Można także definiować własne style

Tworzenie menu

- Menu dodajemy do obiektu JFrame przez utworzenie paska menu.

```
1  JMenuBar menuBar = new JMenuBar();
```

- Do paska dodajemy poszczególne sekcje menu JMenu

```
1  JMenu actions = new JMenu("Actions");
```

- Do sekcji możemy dodawać podsekcje (podmenu), separatory, lub elementy menu

```
1  actions.add(new JMenuItem(miniAction));
2  actions.addSeparator();
3  actions.add(new JMenu("Additional actions"));
```

Elementy menu

- Elementy menu odpowiadają dostępnym typom przycisków

JButton zwyczajny przycisk,

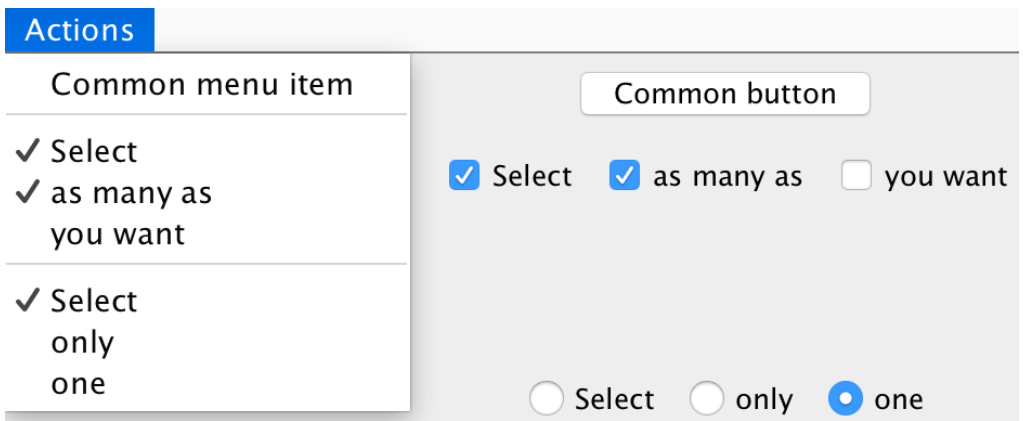
JCheckBox przycisk z zaznaczeniem,

JRadioButton przycisk z ekskluzywnym zaznaczeniem.

JMenuItem zwyczajny wpis,

JCheckBoxMenuItem wpis z zaznaczeniem,

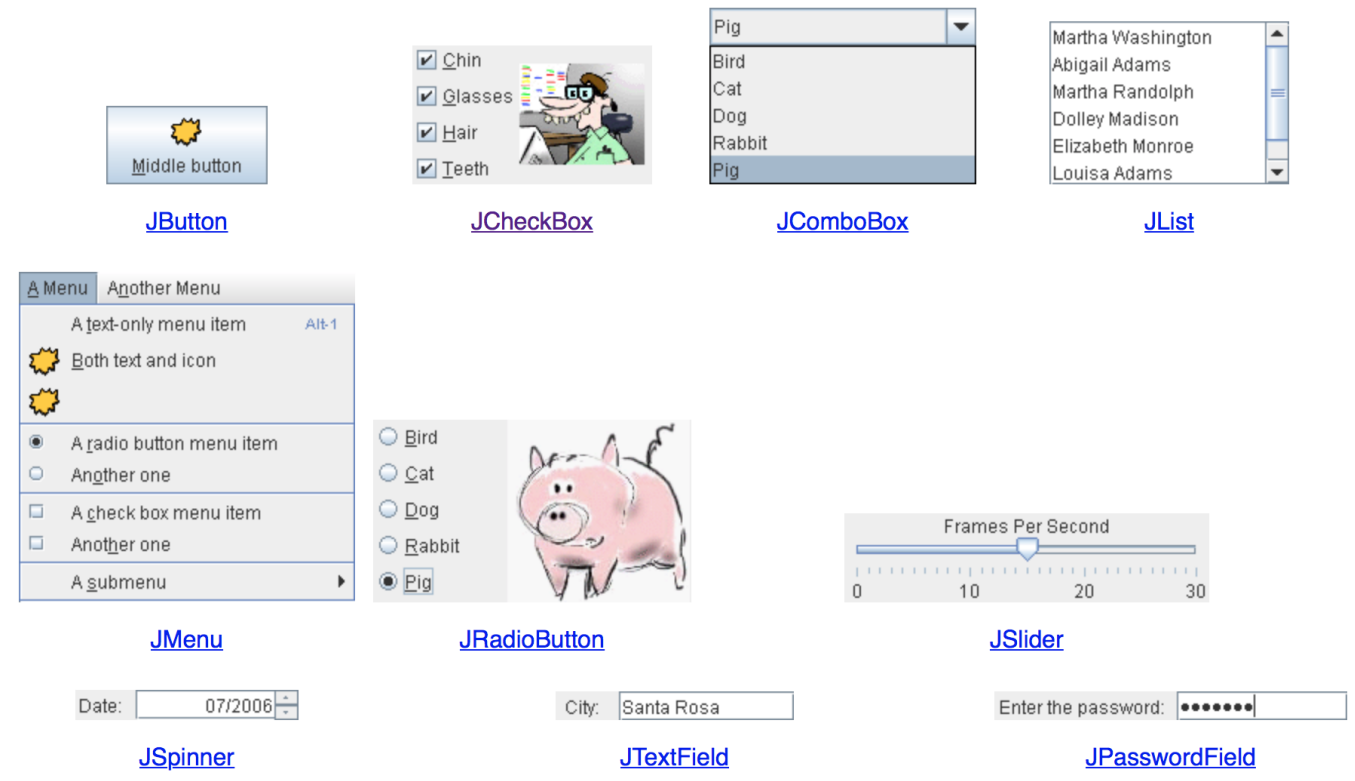
JRadioButtonMenuItem wpis z ekskluzywnym zaznaczeniem.



Rysunek 3: Przyciski i pozycje menu

Inne komponenty

- Z innymi komponentami Swing można się zaznajomić poprzez *A Visual Guide to Swing Components*.



Rysunek 4: Przewodnik po kontrolkach Swing [MIT, 2017]

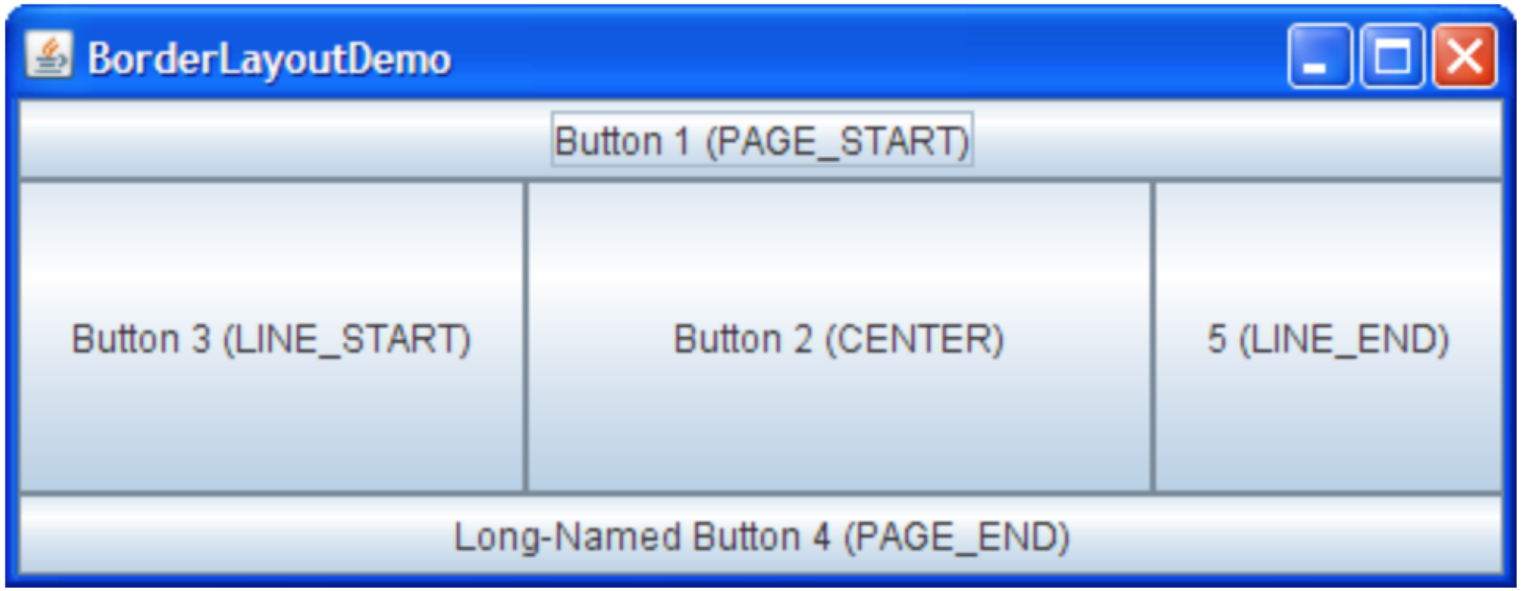
Menadżer układu

- Menadżer układu (*LayoutManager*) wspomaga rozłożenie kontrolek w interfejsie.
- Swing oferuje wiele menadżerów układu, my przyjrzymy się kilku wybranym pod kątem ich użyteczności.
- Dokładniejszy opis prezentuje *A Visual Guide to Layout Managers* [Oracle, 2017]

Dodawanie menadżera układu

- Każdy pojemnik (`Container`) ma przypisany menadżer wyglądu.
- Menadżer wyglądu jest klasą implementującą interfejs `LayoutManager`.
- Przypisuje się go przy pomocy metody `setLayout`.
- Brak przypisania oznacza stosowanie domyślnego menadżera, jeżeli chcemy pracować bez menadżera należy przypisać `null`.
  - nie jest to zalecane, bo wtedy należy ręcznie ustalać położenie i rozmiar komponentów.

Rozmieszczenie komponentów w oknie



Rysunek 5: `BorderLayout`

- Domyślnym menadżerem dla okna jest `BorderLayout`.
- Tworzy on pięć obszarów, w których możemy umieszczać komponenty.
- Zazwyczaj w obszarach umieszcza się kontenery `JPanel` grupujące kontrolki.

Opis klasy BorderLayout

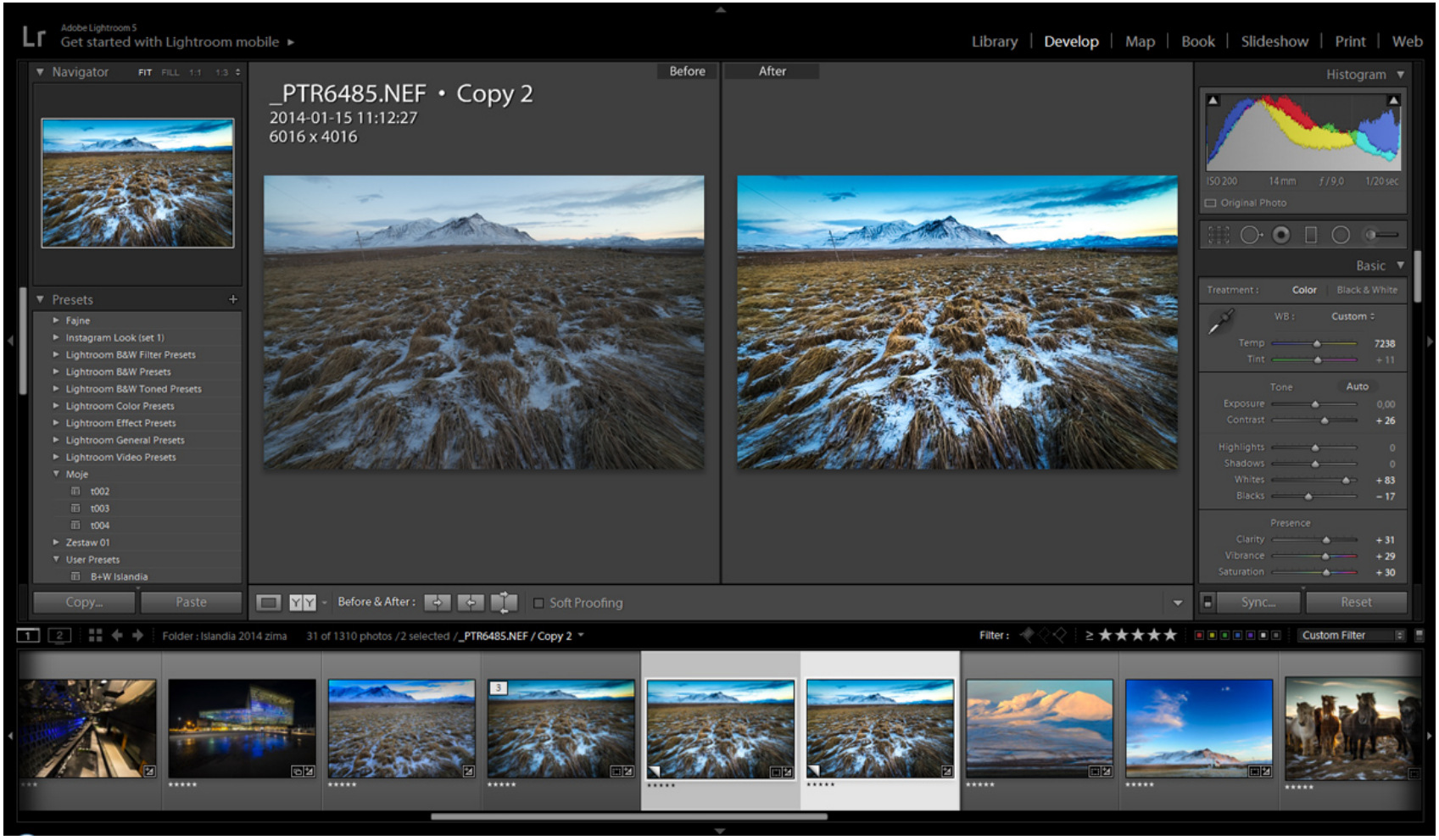
- Konstruktory

```
1  BorderLayout();
2  BorderLayout(int horz, int vert); //przestrzeń między regionami
```

- Komponenty dodajemy do regionów

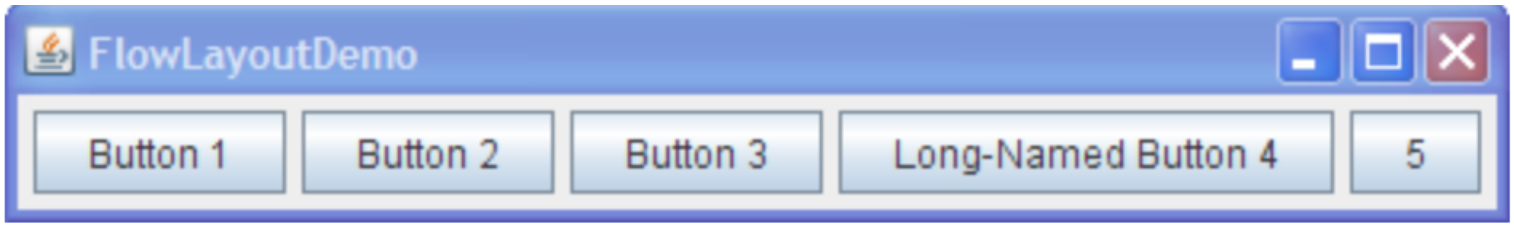
```
1  void add(Component component, Object region);
2  BorderLayout.PAGE_START || BorderLayout. NORTH
3  BorderLayout.PAGE_END || BorderLayout.SOUTH
4  BorderLayout.LINE_START || BorderLayout.WEST
5  BorderLayout.LINE_END || BorderLayout.EAST
6  BorderLayout.CENTER
```

Przykład interfejsu



Rysunek 6: Interfejs programu Lightroom stosuje podział `BorderLayout`

Rozmieszczenie kontrolek

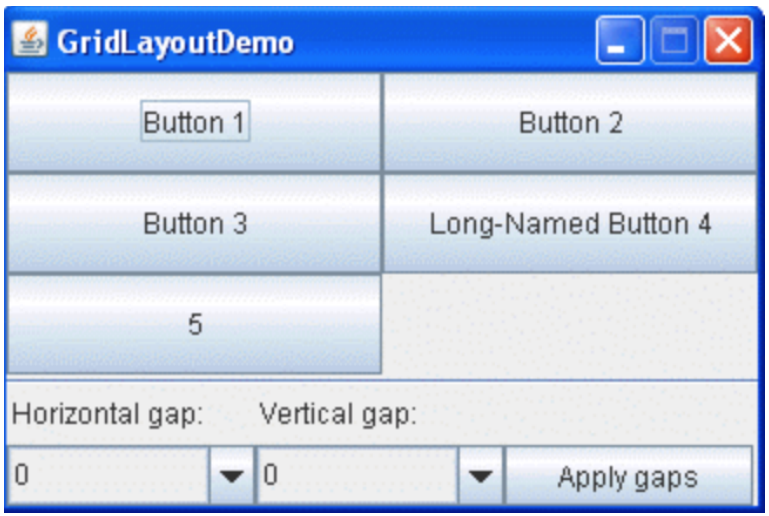


Rysunek 7: `FlowLayout`

- Domyślnym menadżerem dla kontenera `JPanel` jest `FlowLayout`.
- Służy on do rozmieszczania kontrolek jedna po drugiej.
- Kontrolki przepływają do kolejnych linii jak tekst w książce.
- Można określić kierunek tekstu

```
1  FlowLayout (int align, int hgap, int vgap)
```

Równomierne rozmieszczenie elementów

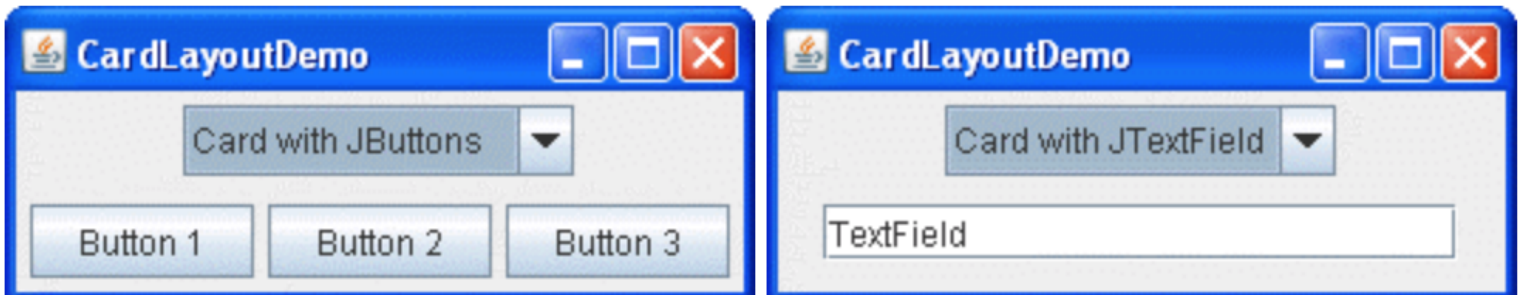


Rysunek 8: `GridLayout`

- Komponenty można porządkować w listy lub tabele korzystając z menadżera `GridLayout`.
- Po określeniu liczby kolumn i wierszy tworzy on kontrolki o tym samym rozmiarze, które umieszcza w komórkach siatki

```
1  GridLayout(int numRows, int
       numColumns, int horz,
        intvert)
```

Tworzenie zakładek



Rysunek 9: `CardLayout`

- W przypadku konieczności dodania dużej liczby kontrolek można zastosować `CardLayout`.
- Tworzy on karty, na których umieszczamy kontrolki.
- Świetnie się sprawdza w panelach konfiguracyjnych.

Opis klasy CardLayout

- Konstruktor

```
1  CardLayout( inthorz,  intvert)
```

- Dodawanie karty

```
1  void add( ComponentpanelObj,  Objectname)
```

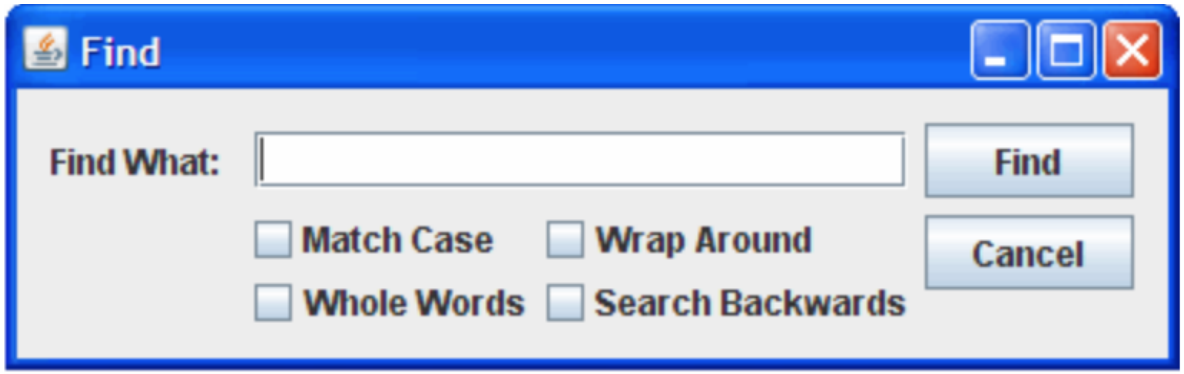
- Nawigacja

```
1  void first(Container deck)
2  void last(Container deck)
3  void next(Container deck)
4  void previous(Container deck)
5  void show(Container deck,String cardName)
```

Przykład zastosowania klasy CardLayout

```
1  public class CardLayoutDemo implements ItemListener {
2      JPanel cards; //a panel that uses CardLayout
3      final static String BUTTONPANEL = "Card with JButtons";
4      final static String TEXTPANEL = "Card with JTextField";
5
6      public void addComponentToPane(Container pane) {
7      JPanel comboBoxPane = new JPanel(); //use FlowLayout
8          String comboBoxItems[] = { BUTTONPANEL, TEXTPANEL };
9          JComboBox cb = new JComboBox(comboBoxItems);
10         cb.setEditable(false);
11         cb.addItemListener(this);
12         comboBoxPane.add(cb);
13         JPanel card1 = new JPanel();
14         card1.add(new JButton("Button 1"));
15         card1.add(new JButton("Button 2"));
16         card1.add(new JButton("Button 3"));
17         JPanel card2 = new JPanel();
18         card2.add(new JTextField("TextField", 20));
19         cards = new JPanel(new CardLayout());
20         cards.add(card1, BUTTONPANEL);
21         cards.add(card2, TEXTPANEL);
22         pane.add(comboBoxPane, BorderLayout.PAGE_START);
23         pane.add(cards, BorderLayout.CENTER);
24     }
25
26     public void itemStateChanged(ItemEvent evt) {
27         CardLayout cl = (CardLayout)(cards.getLayout());
28         cl.show(cards, (String)evt.getItem());
29     }
30 }
```

Tworzenie złożonych interfejsów

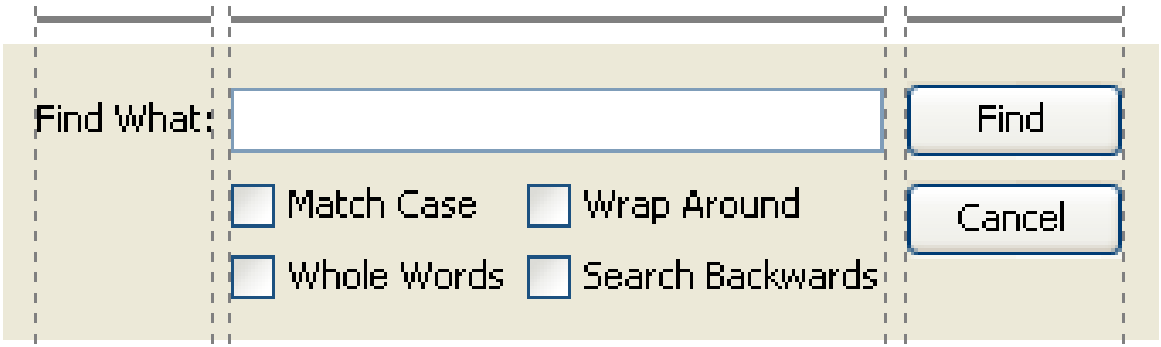


Rysunek 10: `GroupLayout`

- Jedną z możliwości tworzenia złożonego interfejsu jest zastosowanie `GroupLayout`.
- Menadżer określa niezależnie horyzontalne i wertykalne relację między komponentami.
- Robi to łącząc komponenty w sekwencyjne i równoległe grupy

```
1  GroupLayout.SequentialGroup createSequentialGroup()
2  GroupLayout.ParallelGroup createParallelGroup();
```

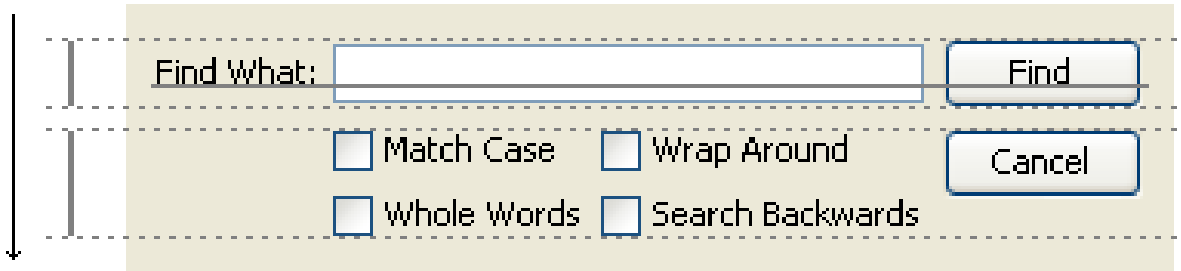
Grupowanie horyzontalne



Rysunek 11: GroupLayout grupowanie horyzontalne

```
1   layout.setHorizontalGroup(layout.createSequentialGroup()
2       .addComponent(label)
3       .addGroup(layout.createParallelGroup(GroupLayout.Alignment.LEADING)
4           .addComponent(textField)
5           .addGroup(layout.createSequentialGroup()
6               .addGroup(layout.createParallelGroup(GroupLayout.Alignment.LEADING)
7                   .addComponent(caseCheckBox)
8                   .addComponent(wholeCheckBox))
9               .addGroup(layout.createParallelGroup(GroupLayout.Alignment.LEADING)
10                  .addComponent(wrapCheckBox)
11                  .addComponent(backCheckBox))))
12      .addGroup(layout.createParallelGroup(GroupLayout.Alignment.LEADING)
13          .addComponent(findButton)
14          .addComponent(cancelButton))
15  );
```

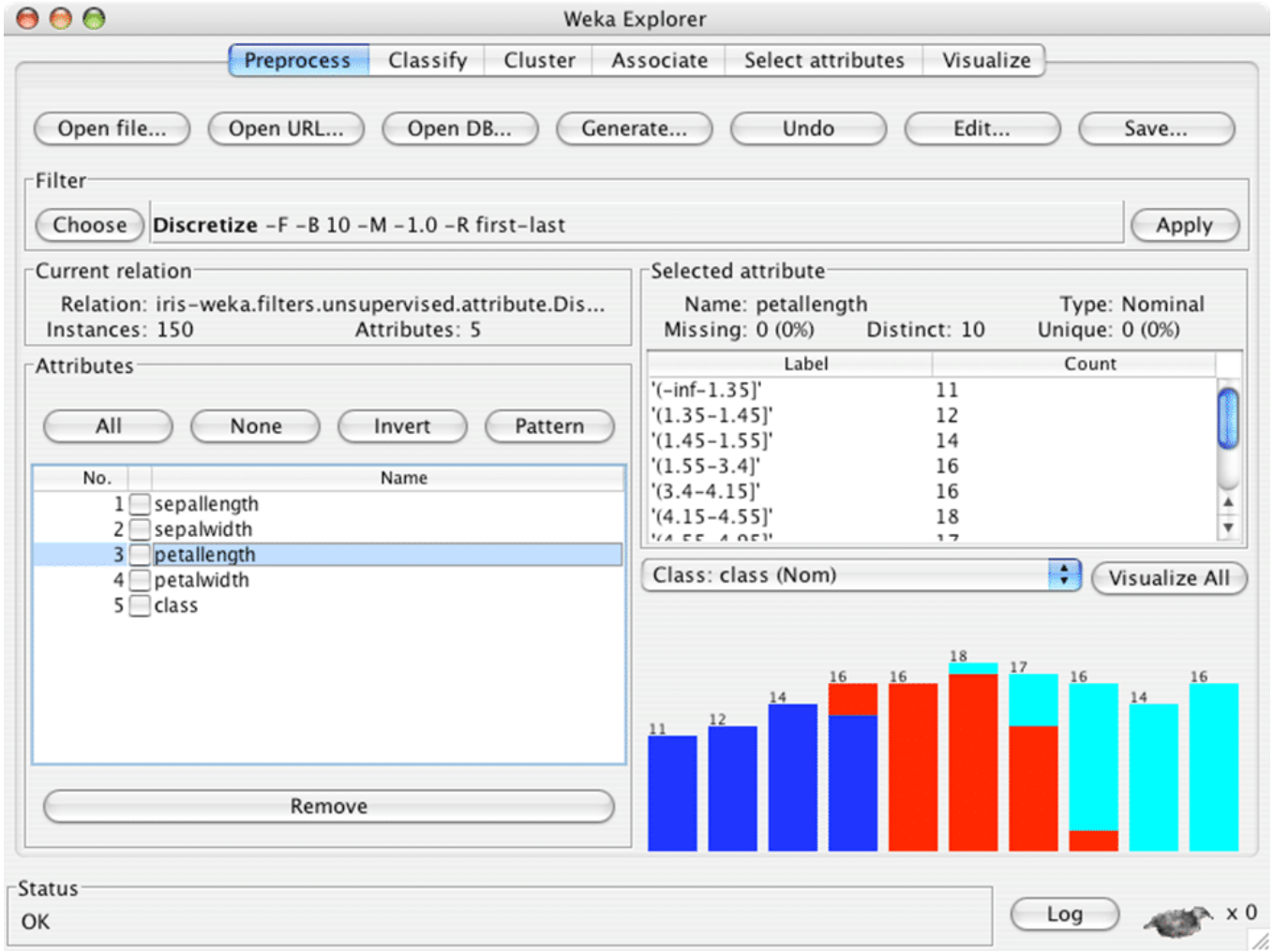
Grupowanie wertykalne



Rysunek 12: `GroupLayout` grupowanie wertykalne

```
1   layout.setVerticalGroup(layout.createSequentialGroup()
2       .addGroup(layout.createParallelGroup(GroupLayout.Alignment.BASELINE)
3           .addComponent(label)
4           .addComponent(textField)
5           .addComponent(findButton))
6       .addGroup(layout.createParallelGroup(GroupLayout.Alignment.LEADING)
7           .addGroup(layout.createSequentialGroup()
8               .addGroup(layout.createParallelGroup(GroupLayout.Alignment.BASELINE)
9                   .addComponent(caseCheckBox)
10                  .addComponent(wrapCheckBox))
11              .addGroup(layout.createParallelGroup(GroupLayout.Alignment.BASELINE)
12                  .addComponent(wholeCheckBox)
13                  .addComponent(backCheckBox)))
14          .addComponent(cancelButton))
15  );
```

Przykład zastosowania biblioteki Swing



Rysunek 13: Interfejs programu do analizy danych Weka wykorzystuje bibliotekę Swing

Bibliografia

[MIT, 2017] MIT (2017).
A visual guide to swing components.

[Oracle, 2017] Oracle (2017).
A visual guide to layout managers.