Faculty of Electronics and Information Technology
WARSAW UNIVERSITY OF TECHNOLOGY

Programowanie Aplikacyjne (PAP)

Java - Swing

Julian Myrcha
Institute of Computer Science
26.02.2025

AWT - Abstract Window Toolkit

- Najstarsza biblioteka
- Wykorzystuje natywne kontrolki każdego systemu ...
- Co jest problemem, bo nie wszystkie kontrolki są wszędzie dostępne
- Napisz raz, testuj wszędzie ;-) zamiast (napisz raz, uruchom wszędzie)

AWT - Abstract Window Toolkit

Swing

- System operacyjny dostarcza gołe, puste okno
- Wszystko jest rysowane przez bibliotekę, zatem sposób działania i wygląd elementów Swinga jest taki sam na różnych platformach
- Swing w niewielkim stopniu zależy od platformy, dzięki czemu jest mniej podatny na błędy związane z danym systemem.
- Wygląd aplikacji Java może się znacznie różnić od wyglądu innych aplikacji w systemie
- Mamy możliwość wybrania jednego z motywów (`GTK`, `Windows`, `Metal`) który może naśladować aplikacjie w aktualnym systemie
- Swing wykorzystuje AWT - zatem nie zastąpił go całkowicie

AWT - Abstract Window Toolkit

Swing

JavaFx
- Najnowsza biblioteka, powoli zastępuje Swinga
- Spójne podejście do kontrolek i obiektów rysowanych (np. linie, cienie, kształty)
- Wykorzystanie karty Graficznej
- Wykorzystanie arkuszy styli (CSS) na wzór stron internetowych
- Niestety priorytety przesuwają się do aplikacji webowych

AWT - Abstract Window Toolkit

Swing

JavaFx

Swing jest prostszy i bardziej popularny, JavaFx jest nowocześniejszy

- okno projektanta to pojedynczy ekran aplikacji

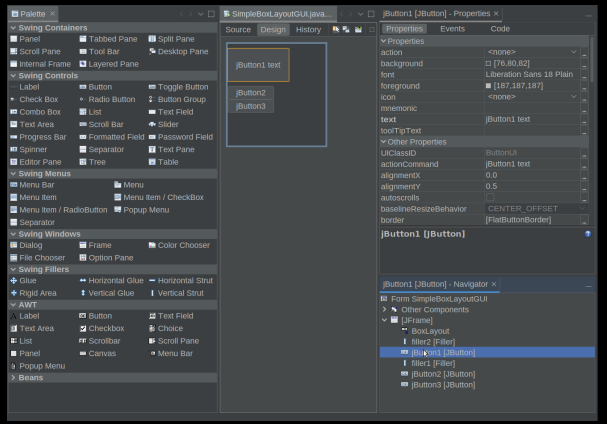
- okno projektanta to pojedynczy ekran aplikacji
- przeciągamy komponenty z palety na ekran

- okno projektanta to pojedynczy ekran aplikacji
- przeciągamy komponenty z palety na ekran
- dla każdego komponentu ustawiamy właściwości

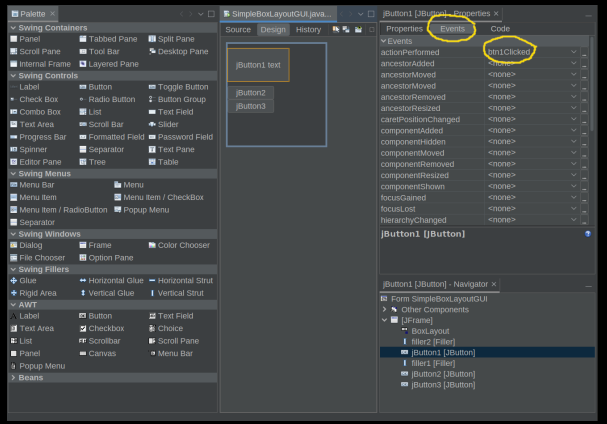
- okno projektanta to pojedynczy ekran aplikacji
- przeciągamy komponenty z palety na ekran
- dla każdego komponentu ustawiamy właściwości
- kod związany z reakcją na zdarzenia to także właściwość

- okno projektanta to pojedynczy ekran aplikacji
- przeciągamy komponenty z palety na ekran
- dla każdego komponentu ustawiamy właściwości
- kod związany z reakcją na zdarzenia to także właściwość
  - w Javie nie ma pointerów do metod - możemy użyć referencji do obiektu który ma metodę

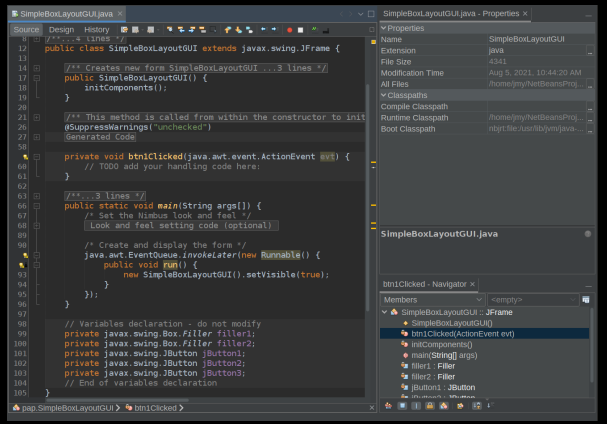
- okno projektanta
  - właściwości



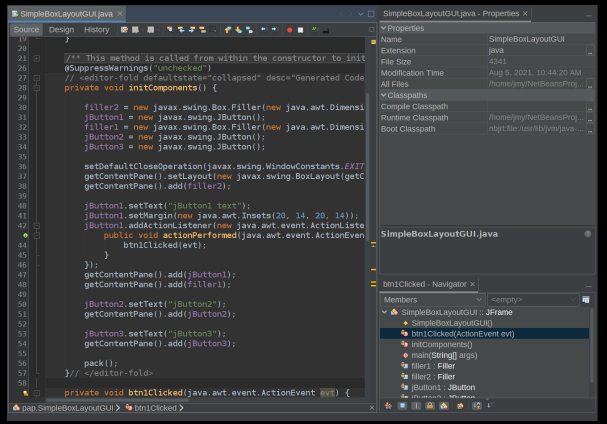
- okno projektanta
  - właściwości
  - zdarzenia



- okno projektanta
  - właściwości
  - zdarzenia
- kod
  - programisty

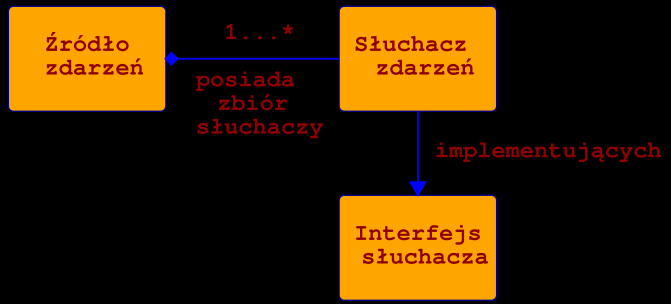


- okno projektanta
  - właściwości
  - zdarzenia
- kod
  - programisty
  - generowany



- interfejs słuchacza:

```
1 public interface ActionListener extends EventListener {
2    public void actionPerformed(ActionEvent e);
3 }
```

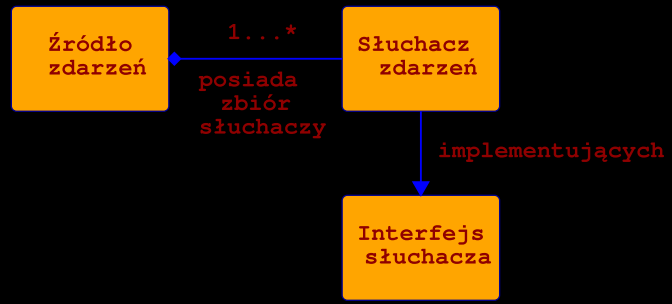


- interfejs słuchacza:

```
1 public interface ActionListener extends EventListener {
2    public void actionPerformed(ActionEvent e);
3 }
```

- słuchacz zdarzeń - implementacja interfejsu słuchacza:

```
1 class ButtonListener implements ActionListener {
2 public void actionPerformed(ActionEvent event) {
3     . . .
4     }
5 }
```



- interfejs słuchacza:

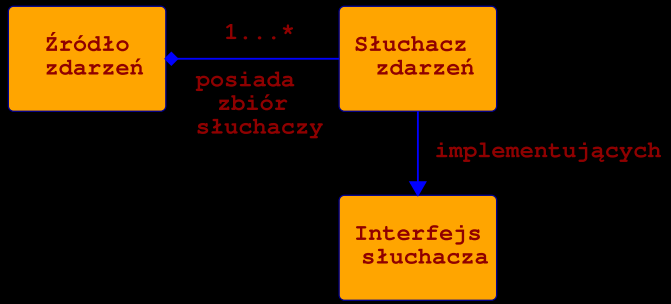
```
public interface ActionListener extends EventListener {
   public void actionPerformed(ActionEvent e);
}
```

- słuchacz zdarzeń - implementacja interfejsu słuchacza:

```
class ButtonListener implements ActionListener {
public void actionPerformed(ActionEvent event) {
    . . .
    }
}
```

- źródła zdarzeń - (przyciski, paski przewijania) - rejestrują słuchaczy i wysyłają im obiekty zdarzeń

```
// kazde nacisniecie przycisku powoduje zawolanie metody
// listener.actionPerformed(event
ActionListener listener=new ButtonListener(); // sluchacz zdarzen
JButton button = new JButton("OK");           // zrodlo zdarzen
button.addActionListener(listener);      // rejestracja sluchacza
```



- interfejs słuchacza:

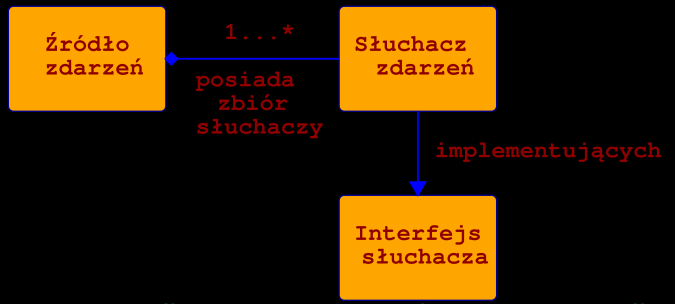
```
public interface ActionListener extends EventListener {
   public void actionPerformed(ActionEvent e);
}
```

- słuchacz zdarzeń - implementacja interfejsu słuchacza:

```
class ButtonListener implements ActionListener {
public void actionPerformed(ActionEvent event) {
    . . .
    }
}
```

- źródła zdarzeń - (przyciski, paski przewijania) - rejestrują słuchaczy i wysyłają im obiekty zdarzeń

```
// kazde nacisniecie przycisku powoduje zawolanie metody
// listener.actionPerformed(event
ActionListener listener=new ButtonListener(); // sluchacz zdarzen
JButton button = new JButton("OK");           // zrodlo zdarzen
button.addActionListener(listener);      // rejestracja sluchacza
```



W przypadku wystąpienia zdarzenia w źródle wszyscy zarejestrowani do tego źródła słuchacze zostaną poinformowani
Każdy słuchacz podejmuje decyzję co ma w takim przypadku zrobić

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

class ButtonListener implements ActionListener {
  @Override
  public void actionPerformed(ActionEvent e) {
    ((JButton) e.getSource()).setText("nacisnieto");
  }
}

public class SimpleSwingSample {
  public static void main(String[] args) {
    SwingUtilities.invokeLater(() -> {
      JFrame frame = new JFrame("Pierwszy Przycisk");
      frame.setBounds(100, 100, 450, 300);
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      JButton closeButton = new JButton("jeszcze nie nacisnieto");
      closeButton.addActionListener(new ButtonListener());
      frame.getContentPane().add(closeButton);
      frame.setVisible(true);
    });
  }
}
```

wersja prosta ze zwykłą klasą implementującą słuchacza

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

public class SimpleSwingSample2 {
  public static void main(String[] args) {
    SwingUtilities.invokeLater(() -> {
      JFrame frame = new JFrame("Pierwszy Przycisk");
      frame.setBounds(100, 100, 450, 300);
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      JButton closeButton = new JButton("jeszcze nie nacisnieto 2");
      closeButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
          ((JButton) e.getSource()).setText("nacisnieto");
        }
      });
      frame.getContentPane().add(closeButton);
      frame.setVisible(true);
      // pokaz onkno
    });
  }
}

```

wersja prosta z klasą anonimową implementującą słuchacza

```
1  package pap;
2  import java.awt.event.ActionListener;
3  import java.awt.event.ActionEvent;
4  import javax.swing.*;
5
6  public class SimpleSwingSample3 {
7    public static void main(String[] args) {
8      SwingUtilities.invokeLater(() -> {
9        JFrame frame = new JFrame("Pierwszy Przycisk");
10       frame.setBounds(100, 100, 450, 300);
11       frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
12       JButton closeButton = new JButton("jeszcze nie nacisnieto 3");
13       closeButton.addActionListener(new ActionListener() {
14         @Override
15         public void actionPerformed(ActionEvent e) {
16           closeButton.setText("nacisnieto");
17         }
18       });
19       frame.getContentPane().add(closeButton);
20       frame.setVisible(true);
21     });
22   }
23 }
24
25
```

wersja prosta z klasą anonimową implementującą słuchacza

```
1  package pap;
2
3  import javax.swing.JButton;
4  import javax.swing.JFrame;
5  import javax.swing.SwingUtilities;
6
7  public class SimpleSwingSample4 {
8    public static void main(String[] args) {
9      SwingUtilities.invokeLater(() -> {
10       JFrame frame = new JFrame("Pierwszy Przycisk");
11       frame.setBounds(100, 100, 450, 300);
12       frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
13       JButton closeButton = new JButton("jeszcze nie nacisnieto 4");
14       closeButton.addActionListener(e -> {
15         closeButton.setText("nacisnieto"); // a jak inne kontrolki?
16       });
17       frame.getContentPane().add(closeButton);
18       frame.setVisible(true);
19     });
20   }
21 }
22
23
24
25
```

Closure daje dostęp do wartości zmiennych w miejscu deklaracji

```java
package pap;
import javax.swing.*;

public class SimpleSwingSample5 extends javax.swing.JFrame {
  public SimpleSwingSample5() {
    initComponents();
  }
  private void initComponents() {
    this.setTitle("Pierwszy Przycisk");
    this.setBounds(100, 100, 450, 300);
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    JButton closeButton = new JButton("jeszcze nie nacisnieto 5");
    closeButton.addActionListener(e -> {
      closeButton.setText("nacisnieto"); // a jak inne kontrolki?
    });
    this.getContentPane().add(closeButton);
  }
  public static void main(String[] args) {
    SwingUtilities.invokeLater(() -> {
      new SimpleSwingSample5().setVisible(true);
    });
  }
}


```

Refaktoring - zamiast tworzyć frame dziedziczymy z JFrame

```
1  package pap;
2  import javax.swing.*;
3  public class SimpleSwingSample6 extends javax.swing.JFrame {
4    public SimpleSwingSample6() {
5      initComponents();
6    }
7    private void initComponents() {  // tworzenie interfejsu nie zawiera logiki aplikacji
8      this.setTitle("Pierwszy Przycisk");
9      this.setBounds(100, 100, 450, 300);
10     this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
11     this.closeButton = new JButton("jeszcze nie nacisnieto 6");
12     closeButton.addActionListener(e -> closeButtonClicked(e));
13     this.getContentPane().add(closeButton);
14   }
15   public static void main(String[] args) {
16     SwingUtilities.invokeLater(() -> {
17       new SimpleSwingSample6().setVisible(true);
18     });
19   }
20   private void closeButtonClicked(java.awt.event.ActionEvent evt) {
21     this.closeButton.setText("nacisnieto"); // dostep do pol i metod okna
22   }
23   JButton closeButton;  // kontrolki widoczne we wszystkich metodach jako pola okna
24 }
```

Refaktoring - wydzielenie procedur obsługi zdarzeń do metod okna (netbeans)

- Należy do komponentu (np. przycisku) podpiąć obsługę zdarzenia danego interfejsem `MouseListener`:

```
1 public interface MouseListener extends EventListener {
2     public void mouseClicked(MouseEvent e);  // wcisniecie (calosc - po parze zdarzen)
3     public void mousePressed(MouseEvent e);  // wcisniecie
4     public void mouseReleased(MouseEvent e); // zwolnienie
5     public void mouseEntered(MouseEvent e);  // najechanie
6     public void mouseExited(MouseEvent e);   // opuszczenie
7 }
```

- problem: trzeba zaimplemenować wszystkie metody, nawet jeżeli są puste.
- Boli, bo powstaje dużo niepotrzebnego kodu (niepotrzebny kod zawsze boli)

- nie da się zastosować `wyrażeń lambda`
- interfejs musi być zaimplementowany w całości

```
closeButton.addMouseListener(new MouseListener() {
    @Override public void mouseClicked(MouseEvent e) {
        System.out.println("mouseClicked");
    }
    @Override public void mousePressed(MouseEvent e) {
        System.out.println("mousePressed");
    }
    @Override public void mouseReleased(MouseEvent e) {
        System.out.println("mouseReleased");
    }
    @Override public void mouseEntered(MouseEvent e) {
        System.out.println("mouseEntered");
    }
    @Override public void mouseExited(MouseEvent e) {
        System.out.println("mouseExited");
    }
});
```

- Rozwiązanie: do każdego interfejsu o nazwie `XxxListener` zawierającego więcej niż jedną metodę utworzono klasę `XxxAdapter` zawierającą puste implementacje

```
public interface XxxListener {
    public void m1();
    public void m2();
}
public abstract class XxxAdapter implements XxxListener {
    @Override public void m1() {}
    @Override public void m2() {}
}
```

- teraz możemy przeciążyć tylko jedną metodę

```
closeButton.addMouseListener(new MouseAdapter() {
    @Override public void mouseClicked(MouseEvent e) {
        System.out.println("mouseClicked");
    }
});
```

- jest to generalnie dobre rozwiązanie tak długo, ja nie chcemy aby obsługą zdarzeń zajmowało się okno

metoda	opis
getX()	pozycja X w ramach komponentu
getXOnScreen()	pozycja X na ekranie
getClickCount()	liczba szybkich wciśnięć (ale zawsze najpierw 1)
getSource()	komponent źródłowy

```
JButton first = new JButton("First");
JButton second = new JButton("Second");
MouseAdapter adapter = new MouseAdapter() {
    @Override
    public void mouseClicked(MouseEvent e) {
        ((JButton)e.getSource()).setText("wcisnieto");
    }
};
first.addMouseListener(adapter);
second.addMouseListener(adapter);
```

```
1 public interface WindowListener {
2     void windowOpened(WindowEvent e);  // po otwarciu okna
3     void windowClosing(WindowEvent e);  // zamykamy (trzeba zawolac hide lub dispose)
4     void windowClosed(WindowEvent e);   // po zamknieciu
5     void windowIconified(WindowEvent e); // po zikonifikowaniu
6     void windowDeiconified(WindowEvent e);
7     void windowActivated(WindowEvent e); // po aktywacji
8     void windowDeactivated(WindowEvent e);
9 }
```

- Komponent, odpowiadający oknu, nie ma rodzica
- Dziedziczy z `Frame` (AWT) - nie należy zapomnieć o 'J'

```
public class SimpleFrameTest {
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable(){
            public void run(){
                SimpleFrame frame = new SimpleFrame();
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                frame.setVisible(true);
            }
        });
    }
}
-------------------------- SimpleFrame.java
class SimpleFrame extends JFrame {
    private static final int DEFAULT_WIDTH = 300;
    private static final int DEFAULT_HEIGHT = 200;
    public SimpleFrame() {
        setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
    }
}
```

- można to też zrobić w jednej klasie ...

Polecenie (1)

Proszę o wyświetlenie pustego okienka

Rozwiązanie (11)

```java
package pap;
import javax.swing.JFrame;
import javax.swing.SwingUtilities;

public class Simple {
  public static void main(String[] args) {
    SwingUtilities.invokeLater(  () -> {
        JFrame frame = new JFrame("Test");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setBounds(50, 50, 200, 200);
        frame.setVisible(true);
      }
    );
  }
}
```

```java
package pap;
import java.awt.EventQueue;
import javax.swing.JFrame;
public class AppWnd {
  private JFrame frmSwing;
  public static void main(String[] args) {
    EventQueue.invokeLater(new Runnable() {
      public void run() {
        try {
          AppWnd window = new AppWnd();
          window.frmSwing.setVisible(true);
        } catch (Exception e)
          e.printStackTrace();
      }
    });
  }
  public AppWnd() {
    frmSwing = new JFrame();
    frmSwing.setBounds(100, 100, 450, 300);
    frmSwing.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  }
}
```

```java
package pap;
import java.awt.EventQueue;
import javax.swing.JFrame;
public class AppWnd extends JFrame {
  public static void main(String[] args) {
    EventQueue.invokeLater(new Runnable() {
      public void run() {
        try {
          AppWnd window = new AppWnd();
          window.setVisible(true);
        } catch (Exception e) {
          e.printStackTrace();
        }
      }
    });
  }
  public AppWnd() {
    setBounds(100, 100, 450, 300);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  }
}
```

```java
package pap;
import java.awt.EventQueue;
import javax.swing.JFrame;
public class AppWnd extends JFrame {
  public static void main(String[] args) {
    new AppWnd().runApp();
  }
  void runApp() {
    EventQueue.invokeLater(new Runnable() {
      public void run() {
        try {
          setBounds(100, 100, 450, 300);
          setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
          setVisible(true);
        } catch (Exception e) {
          e.printStackTrace();
        }
      }
    });
  }
}
```

```java
package pap;
import java.awt.EventQueue;
import javax.swing.JFrame;

public class AppWnd extends JFrame {
  public static void main(String[] args) {
    new AppWnd().runApp();
  }
  void runApp() {
    EventQueue.invokeLater(()-> {
      try {
        setBounds(100, 100, 450, 300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
      } catch (Exception e) {
        e.printStackTrace();
      }
    });
  }
}
```

- Konfiguracja każdego komponentu Swing musi się odbywać w wątku dystrybucji zdarzeń

```
EventQueue.invokeLater(new Runnable() {
    public void run() {
        instrukcje ...
    }
});
```

- Co się stanie, gdy użytkownik zamknie ramkę:

```
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

- Jeżeli tego nie określimy, to ramka będzie ukryta, ale aplikacja nadal działa ;-(
- Ponieważ wątek obsługi zdarzeń będzie nadal czekał na zdarzenia
- Główny wątek aplikacji (funkcja `main()`) skończył się po utworzeniu okna

Położenie

```
setLocation(x, y)
setBounds(x, y, width, height)
setLocationByPlatform(true)
setTitle(String title)
getContentPane()
```

Rozmiar aplikacji:

```
Toolkit kit = Toolkit.getDefaultToolkit();   // worek z ciekawymi metodami systemowymi
Dimension screenSize = kit.getScreenSize();  // rozmiar ekranu
int screenWidth = screenSize.width;
setSize(screenWidth / 2, screenHeight / 2);  // rozmiar polowy ekranu
setLocationByPlatform(true);                 // niech system sam znajdzie gdzie okno
Image img = new ImageIcon("icon.gif").getImage();
setIconImage(img);
```

- aby dodać coś do ramki musimy pobrać jej panel zawartości:

```
Container contentPane = frame.getContentPane();
Component c = . . .;
contentPane.add(c);
```

Naszym komponentem może być własna klasa:

```
public class MyComponent extends JComponent {
    public static final int MESSAGE_X = 75;
    public static final int MESSAGE_Y = 100;
    private static final int DEFAULT_WIDTH = 300;
    private static final int DEFAULT_HEIGHT = 200;
    public void paintComponent(Graphics g)  {
        g.drawString("Witamy na okienku", MESSAGE_X, MESSAGE_Y);
    }
    public Dimension getPreferredSize() {
        return new Dimension(DEFAULT_WIDTH,    DEFAULT_HEIGHT);
    }
}
```

- `Point` - położenie

```
Point p = new Point(20, 40);
int x = p.getX();
int y = p.getY();
p.setLocation(10, 60);
closeButton.setLocation(10, 15);
closeButton.setLocation(new Point(10, 15));
Point p = closeButton.getLocation();
```

- `Dimension` - rozmiar

```
Dimension d  = new Dimension(100, 50);
closeButton.setSize(100, 50);
closeButton.setSize(d);
Dimension d2 = closeButton.getSize();
int width = d2.width;
int height = d2.height;
```

- `Insets` - miejsce wokół kontenera

```
Insets ins = new Insets(20, 5, 5, 5);
Insets ins = frame.getInsets();
int top = ins.top;
int left = ins.left;
int bottom = ins.bottom;
int right = ins.right;
```

- `Rectangle` - prostokąt

```
Rectangle r1 = new Rectangle();                    // same zera
Rectangle r2 = new Rectangle(new Point(10, 10));    // ale rozmiar nadal zero
Rectangle r3 = new Rectangle(new Point(10, 10), new Dimension(200, 100));
Rectangle r4 = new Rectangle(10, 10, 200, 100);
```

Prostokąta używamy do określenia położenia i rozmiaru komponentów

- W celu zbudowania GUI trzeba zbudować drzewo komponentów

- Na przykład JPane ma następującą budowę:

```
JFrame
        root pane
                glass pane
                layered pane
                        menu bar
                        content pane
```

- Należy pobrać contentPane i dodać do niego przycisk

```
JButton closeButton = new JButton("Close");
Container contentPane = frame.getContentPane();
contentPane.add(closeButton);
```

- zajmuje cały ekran ;-(
- nic nie robi

Aby ustalić odpowiedni rozmiar trzeba użyć pack - pyta wszystkie kontrolki ile potrzebują

```
JFrame frame = new JFrame("Test");

JButton closeButton = new JButton("Close");
Container contentPane = frame.getContentPane();
contentPane.add(closeButton);

frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setLocation(50, 50);
frame.pack();
frame.setVisible(true);
```

- Dziedziczymy z klasy `JDialog` lub `JFrame`

```
class SecondWindow extends JDialog  {
  SecondWindow(TwoWindows parent) {
    var btn2 = new JButton("drugie"); // tu ustawiamy
    btn2.addActionListener(e->{

    });
    this.getContentPane().add(btn2);
  }
}
...
public class TwoWindows {
  public JButton btn;
  ....
      btn.addActionListener(e->{
        SecondWindow wnd = new SecondWindow(this);
        wnd.setVisible(true);         // niemodalne - pojdzie dalej natychmiast!
      });
```

- Dziedziczymy z klasy `JDialog` lub `JFrame`
- pokazanie drugiego okna nie zatrzymuje interakcji z pierwszym

```
class SecondWindow extends JDialog  {
  SecondWindow(TwoWindows parent) {
    var btn2 = new JButton("drugie"); // tu ustawiamy
    btn2.addActionListener(e->{

    });
    this.getContentPane().add(btn2);
  }
}
...
public class TwoWindows {
  public JButton btn;
  ....
      btn.addActionListener(e->{
        SecondWindow wnd = new SecondWindow(this);
        wnd.setVisible(true);         // niemodalne - pojdzie dalej natychmiast!
      });
```

- Dziedziczymy z klasy `JDialog` lub `JFrame`
- pokazanie drugiego okna nie zatrzymuje interakcji z pierwszym
- to drugie okno musi przekazać wynik

```
class SecondWindow extends JDialog  {
  SecondWindow(TwoWindows parent) {
    var btn2 = new JButton("drugie"); // tu ustawiamy
    btn2.addActionListener(e->{
      parent.btn.setText("nacisnieto");
    });
    this.getContentPane().add(btn2);
  }
}
...
public class TwoWindows {
  public JButton btn;
  ....
      btn.addActionListener(e->{
        SecondWindow wnd = new SecondWindow(this);
        wnd.setVisible(true);         // niemodalne - pojdzie dalej natychmiast!
      });
```

```
1  package proz;
2  import javax.swing.JButton;
3  import javax.swing.JFrame;
4  import javax.swing.JDialog;
5  import javax.swing.SwingUtilities;
6
7  class SecondWindow extends JDialog  {
8    static final long serialVersionUID = 1L;
9    SecondWindow(TwoWindows parent) {
10     this.setBounds(300, 300, 450, 300);          // nadaje rozmiar oknu
11     var btn2 = new JButton("drugie"); // tu ustawiamy
12     btn2.addActionListener(e->{
13       parent.btn.setText("nacisnieto");
14     });
15     this.getContentPane().add(btn2);
16   }
17 }
18 public class TwoWindows {
19   public JButton btn;
20   void run() {
21     SwingUtilities.invokeLater(() -> {
22       JFrame frame = new JFrame("Pierwsze okno");
23       frame.setBounds(100, 100, 450, 300);          // nadaje rozmiar oknu
24       frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
25       btn = new JButton("pokaz okno"); // tu ustawiamy
26       btn.addActionListener(e->{
27         SecondWindow wnd = new SecondWindow(this);
28         wnd.setVisible(true);                  // teraz sie nie zatrzyma
29       });
30       frame.getContentPane().add(btn);
31       frame.setVisible(true);                  // pokaz onkno
32     });
33   }
34   public static void main(String[] args) {
35     new TwoWindows().run();              // tworzymy obiekt
36   }
37 }
```

- Dziedziczymy z klasy `JDialog`

```
class SecondWindow extends JDialog  {

  SecondWindow() {

    var btn2 = new JButton("drugie"); // tu ustawiamy
    btn2.addActionListener(e->{
      result="nacisnieto";
    });
    this.getContentPane().add(btn2);
  }
}
...
      btn.addActionListener(e->{
        SecondWindow wnd = new SecondWindow();
        wnd.setVisible(true);         // modalne - dalej pojdzie po zamknieciu okna!

      });
```

- Dziedziczymy z klasy `JDialog`
- ustawiamy typ Modal za pomocą: `setModalityType`

```
class SecondWindow extends JDialog  {

  SecondWindow() {
    this.setModalityType(Dialog.ModalityType.APPLICATION_MODAL);
    var btn2 = new JButton("drugie"); // tu ustawiamy
    btn2.addActionListener(e->{
      result="nacisnieto";
    });
    this.getContentPane().add(btn2);
  }
}
...
      btn.addActionListener(e->{
        SecondWindow wnd = new SecondWindow();
        wnd.setVisible(true);         // modalne - dalej pojdzie po zamknieciu okna!

      });
```

- Dziedziczymy z klasy `JDialog`
- ustawiamy typ Modal za pomocą: `setModalityType`
- wynik działania możemy zapisać w polu klasy

```
class SecondWindow extends JDialog  {
  public String result = "";
  SecondWindow() {
    this.setModalityType(Dialog.ModalityType.APPLICATION_MODAL);
    var btn2 = new JButton("drugie"); // tu ustawiamy
    btn2.addActionListener(e->{
      result="nacisnieto";
    });
    this.getContentPane().add(btn2);
  }
}
...
      btn.addActionListener(e->{
        SecondWindow wnd = new SecondWindow();
        wnd.setVisible(true);         // modalne - dalej pojdzie po zamknieciu okna!
        btn.setText(wnd.result);
      });
```

```java
package proz;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JDialog;
import java.awt.Dialog;
import javax.swing.SwingUtilities;

class SecondWindow extends JDialog  {
  static final long serialVersionUID = 1L;
  public String result = "";
  SecondWindow() {
    this.setModalityType(Dialog.ModalityType.APPLICATION_MODAL);
    this.setBounds(300, 300, 450, 300);          // nadaje rozmiar oknu
    var btn2 = new JButton("drugie"); // tu ustawiamy
    btn2.addActionListener(e->{
      result="nacisnieto";
    });
    this.getContentPane().add(btn2);
  }
}
public class TwoWindows {
  void run() {
    SwingUtilities.invokeLater(() -> {
      JFrame frame = new JFrame("Pierwsze okno");
      frame.setBounds(100, 100, 450, 300);          // nadaje rozmiar oknu
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      JButton btn = new JButton("pokaz okno"); // tu ustawiamy
      btn.addActionListener(e->{
        SecondWindow wnd = new SecondWindow();
        wnd.setVisible(true);
        btn.setText(wnd.result);
      });
      frame.getContentPane().add(btn);
      frame.setVisible(true);                  // pokaz onkno
    });
  }
  public static void main(String[] args) {
    new TwoWindows().run();              // tworzymy obiekt
  }
}
```

- A co jak dodamy drugi przycisk?

```
1  public class Simple {
2    public static void main(String[] args) {
3      SwingUtilities.invokeLater(  () -> {
4        JFrame frame = new JFrame("Dwa przyciski");
5        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
6
7        JButton closeButton = new JButton("Close");
8        JButton helpButton = new JButton("Help");
9        Container contentPane = frame.getContentPane();
10       contentPane.add(closeButton);
11       contentPane.add(helpButton);
12       frame.pack();
13       System.out.println("Mamy "+contentPane.getComponents().length+" przyciski");
14       frame.setVisible(true);
15     });
16   }
17 }
```

- co się stało z przyciskiem **Close**?
- dlaczego mamy 2 przyciski?

- `Content pane` domyślnie używa układu `BorderLayout`
- Zadaniem zarządcy układów jest dla każdego komponentu, który został do niego dodany, określenie `Rectangle`
- Można to wprawdzie zrobić bez zarządcy ...
- Ale wtedy mamy problem gdy rozmiar okna/fontu inny niż na początku

Polecenie (2)

Proszę napisać program do gry w kółko i krzyżyk

Wskazówki:

- Aby zmienić napis na przycisku, wywołujemy metodę `setText`

```
closeButton.addActionListener(e->{
              closeButton.setText("O");
            });
```

Rozwiązanie (11)

```
1  package pap;
2
3  import java.awt.Container;
4  import java.awt.Dimension;
5  import java.awt.Point;
6
7  import javax.swing.JButton;
8  import javax.swing.JFrame;
9  import javax.swing.SwingUtilities;
10
11 class tictac {
12   int count;
13   public int getTic() {
14     return count;
15   }
16   public void inc() {
17     count++;
18   }
19
20 }
21
22 public class Kolko {
23   public static void main(String[] args) {
24     SwingUtilities.invokeLater(  () -> {
25       final tictac tic = new tictac();
26       JFrame frame = new JFrame("Test");
27       frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
28       frame.setBounds(50, 50, 400, 200);
29       Container pane  = frame.getContentPane();
30       pane.setLayout(null);
31       for(int i =0; i<9;i++) {
32         JButton btn = new JButton(" ");
33         btn.setSize(new Dimension(100,20));
34         btn.setLocation(new Point(10+(i%3)*110,10+(i/3)*30));
35         btn.addActionListener(e->{
36           if(btn.getText() == " ") {
37             btn.setText(tic.getTic() % 2 == 0? "X":"O");
38             tic.inc();
39           }
40         });
```

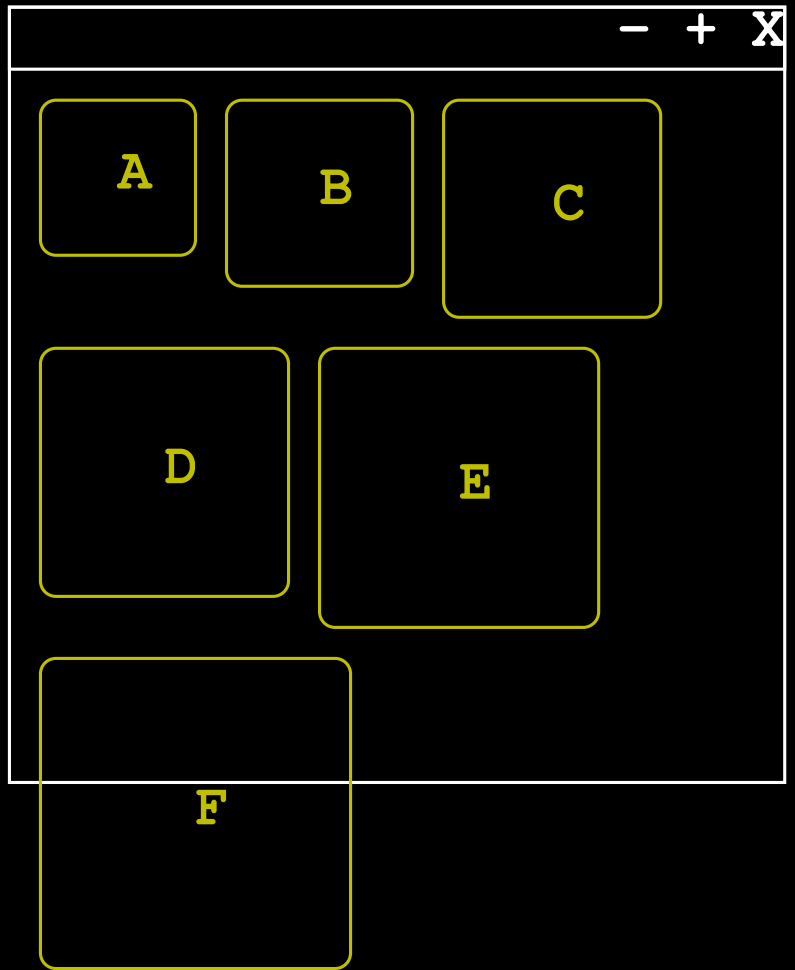
```
          pane.add(btn);
        }
        frame.setVisible(true);
      }
    );
    }
}

```

- układ w którym kolejne komponenty są dodawane od lewej do prawej (domyślnie)
- jak osiągniemy prawy brzeg okna, zaczynamy dodawanie w kolejnym wierszu
- jak dojdziemy do końca okna to wypadamy od dołu

```
JFrame frame = new JFrame("Flow Layout Test");
frame.setBounds(100, 100, 450, 300);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
Container pane = frame.getContentPane();
pane.setLayout(new FlowLayout(FlowLayout.LEFT));
for(int i=0; i< 10; i++) {
  JButton btn = new JButton(String.valueOf(i));
  btn.setPreferredSize(new Dimension(70+i*10,20+i*10));
  pane.add(btn);
}
frame.setVisible(true);
```

```java
package pap;
import java.awt.Container;
import java.awt.Dimension;
import java.awt.FlowLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.SwingUtilities;
public class FlowLayoutTest {
  public static void main(String[] args) {
    SwingUtilities.invokeLater(  () -> {
      JFrame frame = new JFrame("Flow Layout Test");
      frame.setBounds(100, 100, 450, 300);
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      Container pane = frame.getContentPane();
      pane.setLayout(new FlowLayout(FlowLayout.LEFT));
      for(int i=0; i< 10; i++) {
        JButton btn = new JButton(String.valueOf(i));
        btn.setPreferredSize(new Dimension(70+i*10,20+i*10));
        pane.add(btn);
      }
      frame.setVisible(true);
    });
  }
}
```

LEFT - do lewej
RIGHT - do prawej
CENTER - centruj (domyślnie)
LEADING - do xxx w `ComponentOrientation.xxx_TO_yyy`
TRAILING - do yyy w `ComponentOrientation.xxx_TO_yyy`



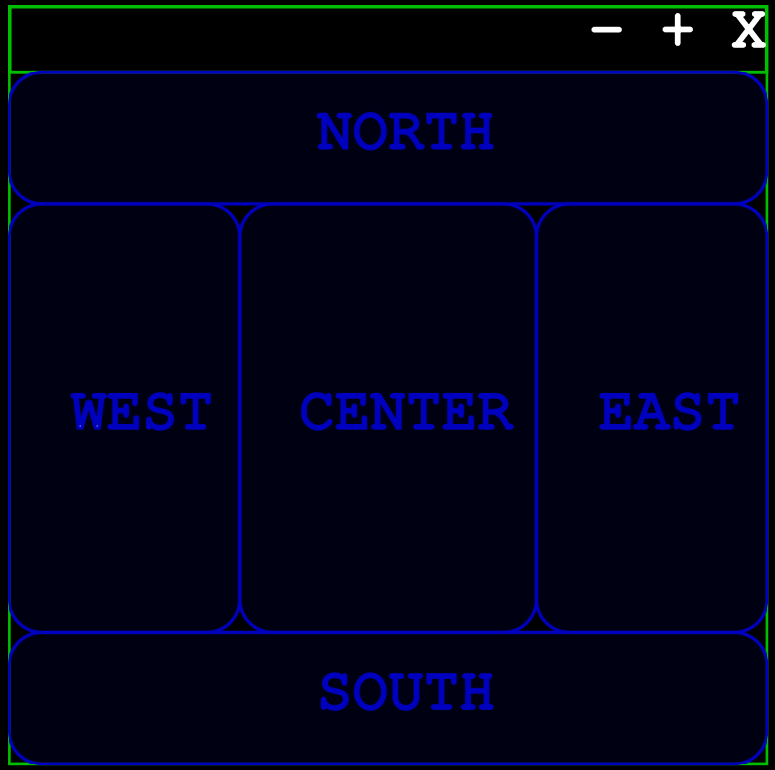
Proszę napisać program zawierający grupę przycisków zmieniających kierunki wypełnienia układu FlowLayout

proszę napisać program odkrywający przyciski

```
1  package proz;
2  import javax.swing.JButton;
3  import javax.swing.JFrame;
4  import java.awt.FlowLayout;
5  import javax.swing.SwingUtilities;
6  import java.awt.Container;
7  public class FlowLayoutSample {
8    void run() {
9      SwingUtilities.invokeLater(() -> {
10       JFrame frame = new JFrame("Flow Layout Test");
11       frame.setBounds(100, 100, 450, 300);
12       frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
13       Container pane = frame.getContentPane();
14       pane.setLayout(new FlowLayout(FlowLayout.LEFT));
15       int[] digits = {1,2,3,5,8,1,0,2,7,9,8};
16       for(int i=0; i< digits.length; i++) {
17         final int index = i;
18         JButton btn = new JButton("?");
19         btn.addActionListener(e->
20           btn.setText(String.format("%d",digits[index]))
21         );
22         pane.add(btn);
23       }
24       frame.setVisible(true);
25     });
26   }
27   public static void main(String[] args) {
28     new FlowLayoutSample().run();              // tworzymy obiekt
29   }
30 }
```

```
JButton northButton = new JButton("North");
container.add(northButton, BorderLayout.NORTH);
```

NORTH - góra
SOUTH - dół
EAST - prawo
WEST - lewo
CENTER - środek



W tym układzie poszczególne komponenty leżą jeden nad drugim

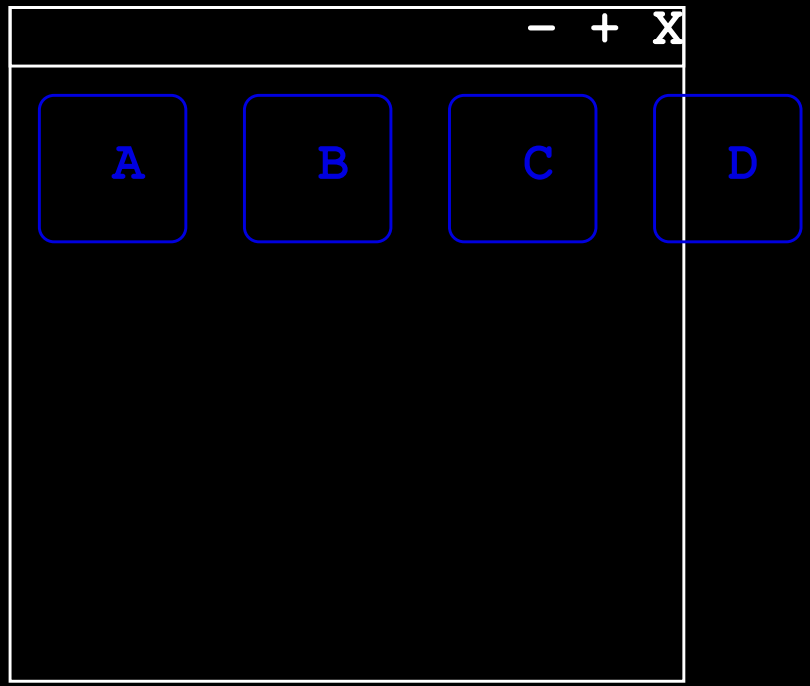
```
SwingUtilities.invokeLater(  () -> {
    JFrame frame = new JFrame("Cards");
    frame.setBounds(100, 100, 450, 300);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    Container pane = frame.getContentPane();
    pane.setLayout(new CardLayout());
    for(int i=0; i< 10; i++) {
        JButton btn = new JButton(String.valueOf(i));
        btn.setSize(new Dimension(100,20));
        btn.addActionListener(e->{
            ((CardLayout)pane.getLayout()).next(pane);
        });
        pane.add(btn);

    }
    pane.applyComponentOrientation(ComponentOrientation.RIGHT_TO_LEFT);
    frame.setVisible(true);
  }
);
```

Polecenie (4)

Proszę napisać program testujący CardLayout

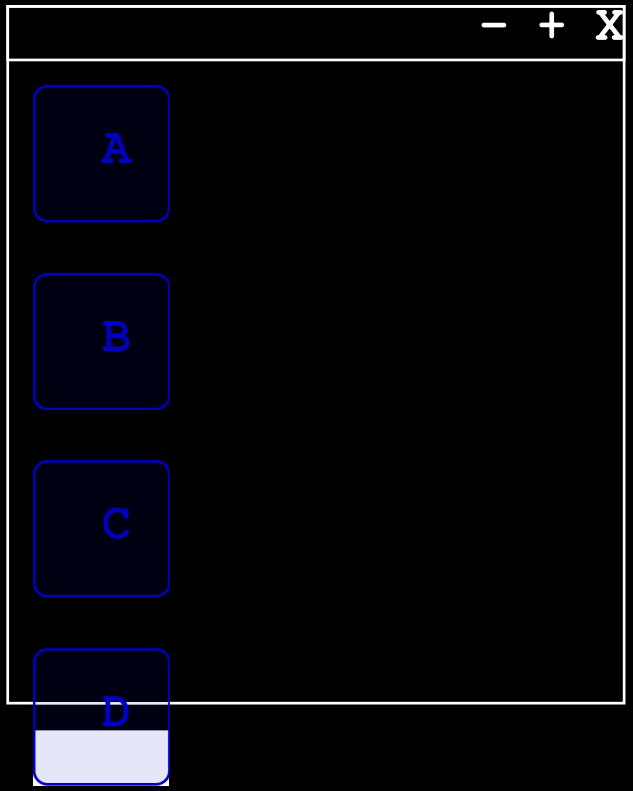
```
JPanel hPanel = new JPanel();
BoxLayout boxLayout = new BoxLayout(hPanel, BoxLayout.X_AXIS);
hPanel.setLayout(boxLayout);
hPanel.add(new JButton("Button 1"));
hPanel.add(new JButton("Button 2"));
```

X_AXIS - poziomo
Y_AXIS - pionowo
LINE_AXIS - dziedziczy
PAGE-AXIS - dziedziczy



```
JPanel hPanel = new JPanel();
BoxLayout boxLayout = new BoxLayout(hPanel, BoxLayout.X_AXIS);
hPanel.setLayout(boxLayout);
hPanel.add(new JButton("Button 1"));
hPanel.add(new JButton("Button 2"));
```

X_AXIS - poziomo
Y_AXIS - pionowo
LINE_AXIS - dziedziczy
PAGE -AXIS - dziedziczy



Klasa pomocnicza tworząca kontener z Layoutem:

```
// poziomo
Box hBox = Box.createHorizontalBox();
// pionowo
Box vBox = Box.createVerticalBox();
hBox.add(new JButton("Button 1");
hBox.add(new JButton("Button 2");
```

Glue - niewidoczny, rozszerzalny, tylko szerokość

Strut - niewidoczny, stały, tylko szerokość

Rigid Area - niewidoczny stały rozmiar o szerokości i wysokości

Filler - niewidoczny, szerokość, wartości min, max i preferowana

- niewidoczny, rozrzerzalny

```
Box hBox = Box.createHorizontalBox();
hBox.add(new JButton("First"));
hBox.add(Box.createHorizontalGlue());
hBox.add(new JButton("Last"));
```

- niewidoczny, stały rozmiar

```
Box hBox = Box.createHorizontalBox();
hBox.add(new JButton("First"));
hBox.add(Box.createHorizontalStrut(100));
hBox.add(new JButton("Last"));
```

- niewidoczny, stały rozmiar

```
Box hBox = Box.createHorizontalBox();
hBox.add(new JButton("First"));
hBox.add(Box.createRigidArea(new Dimesnion(10, 5)));
hBox.add(new JButton("Last"));
```

- niewidoczny, rozmiar o ustwianej zmiennosci

```
Dimension minSize = new Dimension(0, 0);
Dimension prefSize = new Dimension(0, 0);
Dimension maxSize = new Dimension(Short.MAX_VALUE, Short.MAX_VALUE);
Box.Filler filler = new Box.Filler(minSize, prefSize, maxSize);
```

Proszę napisać program testujący BoxLayout

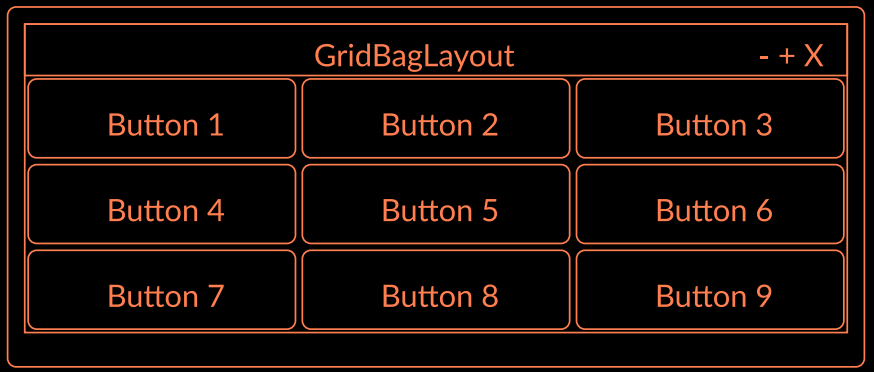
- prostokątna siatka o elementach równych rozmiarów
- nie bierze pod uwagę preferencji
- konstruktor bezparametrowy tworzy jeden wiersz, równo podzielony
- konstruktor dwuparametrowy bierze liczbę wierszy i kolumn, jedna z nich może być 0 (wylicz)
- konstruktor 4 parametrowy pozwala ustalić przerwy
- koleność dodawania kontrolek określa położenie

```
Container pane = frame.getContentPane();
pane.setLayout(new GridLayout(3,3,10,30));
for(int i=0; i< 10; i++) {
    JButton btn = new JButton(String.valueOf(i));
    btn.setSize(new Dimension(100,20));
    pane.add(btn);
}
```

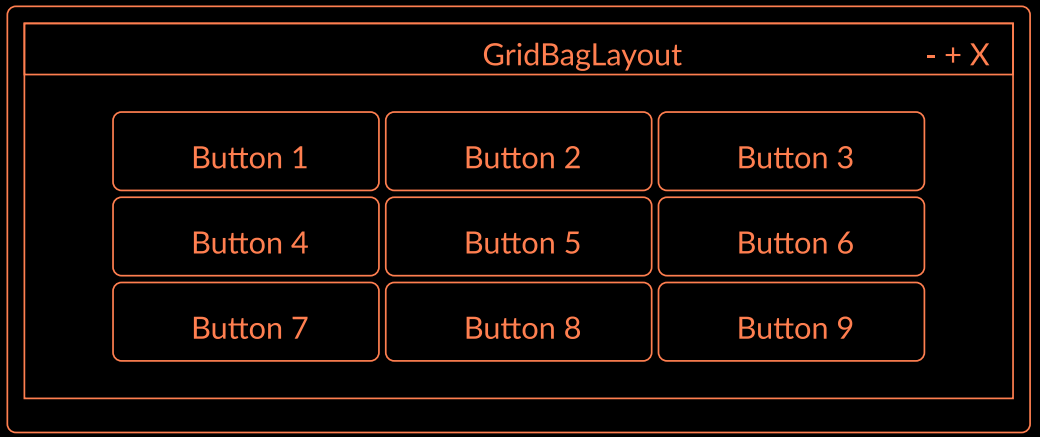
- prostokątna siatka o elementach różnych rozmiarów
- element może zajmować więcej niż jedną celę
- wiersze i kolumny mogą mieć własne zasady zmiany rozmiarów
- położenie dodawanej kontrolki możemy określić

```
GridBagConstraints gbc = new GridBagConstraints();
gbc.gridx = x;  // RELATIVE, liczba calkowita
gbc.gridy = y;
gbc.gridwidth = 1; // 1, integer, relative, remainder
if(x == 1 && y == 2)
    gbc.gridwidth = GridBagConstraints.REMAINDER;
gbc.weightx = 0.5*x;
gbc.weighty = 0.5*y;
gbc.fill = GridBagConstraints.BOTH; // NONE, HORIZONTAL, VERTICAL
gbc.ipadx = 100;  // dodatkowy bufor wewnatrz kontrolki
gbc.ipady = 100;  // zwiekszajacy jej rozmiar
gbc.insets = new Insets(10,10,10,10);
JButton btn = new JButton(String.valueOf(i));
pane.add(btn,gbc);
```

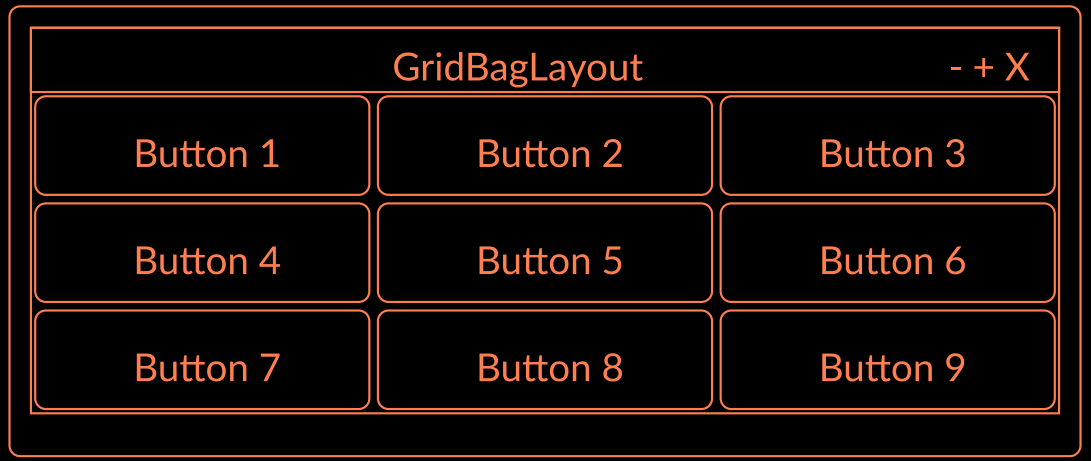
- prostokątna siatka o elementach różnych rozmiarów
- element może zajmować więcej niż jedną celę
- wiersze i kolumny mogą mieć własne zasady zmiany rozmiarów
- położenie dodawanej kontrolki możemy określić
- rozszerzaniem można sterować tak wygląda okno na początku



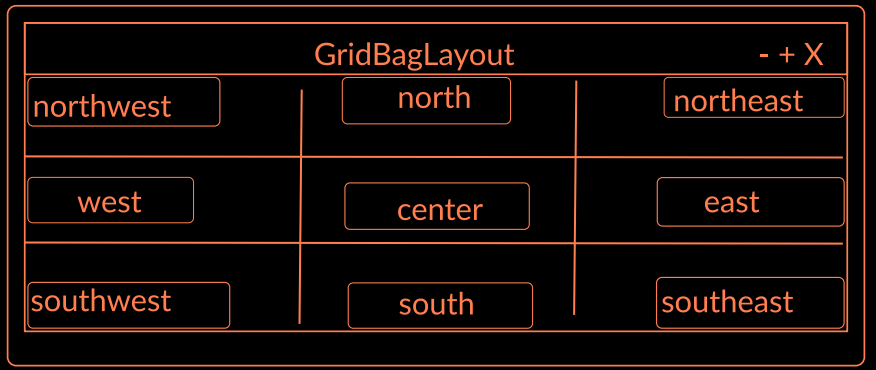
- prostokątna siatka o elementach różnych rozmiarów
- element może zajmować więcej niż jedną celę
- wiersze i kolumny mogą mieć własne zasady zmiany rozmiarów
- położenie dodawanej kontrolki możemy określić
- rozszerzaniem można sterować
  - tak wygląda rozciągniemy gdy nie ustawimy `weightx` oraz `weighty`



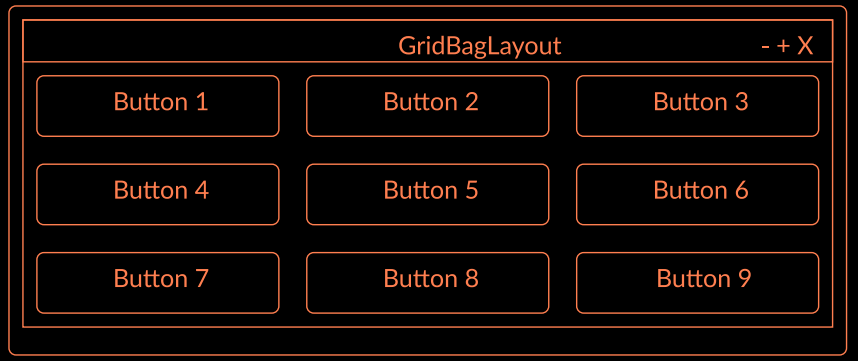
- prostokątna siatka o elementach różnych rozmiarów
- element może zajmować więcej niż jedną celę
- wiersze i kolumny mogą mieć własne zasady zmiany rozmiarów
- położenie dodawanej kontrolki możemy określić
- rozszerzaniem można sterować
  - tak wygląda rozciągniemy gdy ustawimy ustawimy `weightx` oraz `weighty` na 1.0



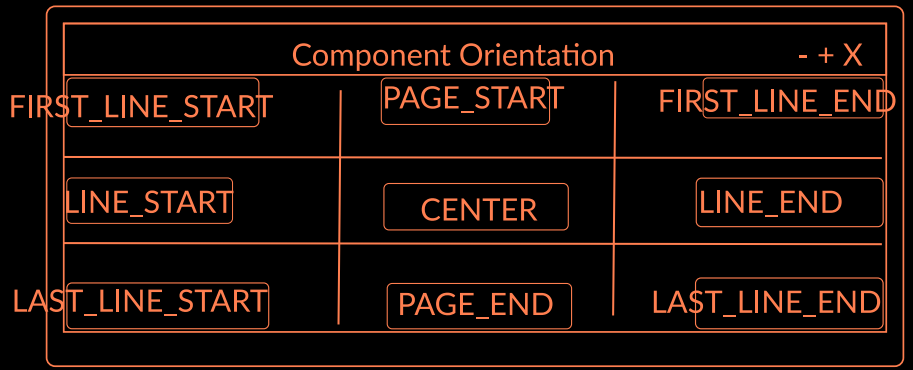
- prostokątna siatka o elementach różnych rozmiarów
- element może zajmować więcej niż jedną celę
- wiersze i kolumny mogą mieć własne zasady zmiany rozmiarów
- położenie dodawanej kontrolki możemy określić
- rozszerzaniem można sterować
- określamy przylepianie kontrolek do krawędzi (anchor)



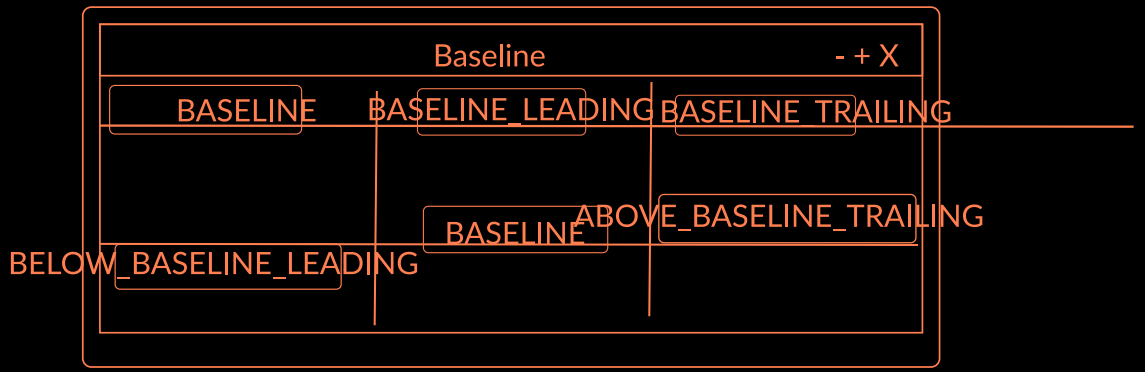
- prostokątna siatka o elementach różnych rozmiarów
- element może zajmować więcej niż jedną celę
- wiersze i kolumny mogą mieć własne zasady zmiany rozmiarów
- położenie dodawanej kontrolki możemy określić
- rozszerzaniem można sterować
- określamy przylepianie kontrolek do krawędzi (anchor)
- określamy odstępy między kontrolkami (insets)



- prostokątna siatka o elementach różnych rozmiarów
- element może zajmować więcej niż jedną celę
- wiersze i kolumny mogą mieć własne zasady zmiany rozmiarów
- położenie dodawanej kontrolki możemy określić
- rozszerzaniem można sterować
- określamy przylepianie kontrolek do krawędzi (anchor)
- określamy odstępy między kontrolkami (insets)
- ComponentOrientation



- prostokątna siatka o elementach różnych rozmiarów
- element może zajmować więcej niż jedną celę
- wiersze i kolumny mogą mieć własne zasady zmiany rozmiarów
- położenie dodawanej kontrolki możemy określić
- rozszerzaniem można sterować
- określamy przylepianie kontrolek do krawędzi (anchor)
- określamy odstępy między kontrolkami (insets)
- ComponentOrientation
- BaseLine



- Tak jak wszystkie layouty ustawia prostokąt
- Mamy pełną kontrolę nad wyrażeniami
- Bardzo skomplikowane, ale można wiele ustawić

przykład ustawienia absolutnego:

- ustawiamy obie kontrolki w zadanej odległości od lewej i górnej krawędzi
- nie ma zależności od innych krawędzi

```
frame.setSize(300,200);
Container contentPane = frame.getContentPane();
SpringLayout springLayout = new SpringLayout();
contentPane.setLayout(springLayout);
JButton b1 = new JButton("Przycisk 1");
JButton b2 = new JButton("Troche wiekszy przycisk 2");
SpringLayout.Constraints b1c = new SpringLayout.Constraints();
SpringLayout.Constraints b2c = new SpringLayout.Constraints();
Spring yPadding = Spring.constant(20);  // odleglosc od gory
b1c.setX(Spring.constant(10));
b1c.setY(yPadding);
b2c.setX(Spring.constant(100));
b2c.setY(yPadding);
contentPane.add(b1, b1c);
contentPane.add(b2, b2c);
```

przykład wyliczanego ustawienia absolutnego:

- pierwszy przycisk pozycjonujemy tak samo
- drugi przycisk pozycjonujemy względem prawego (EAST) brzegu przycisku pierwszego

```
frame.setSize(300,200);
Container contentPane = frame.getContentPane();

SpringLayout springLayout = new SpringLayout();
contentPane.setLayout(springLayout);

JButton b1 = new JButton("Przycisk 1");
JButton b2 = new JButton("Troche wiekszy przycisk 2");

SpringLayout.Constraints b1c = new SpringLayout.Constraints();
SpringLayout.Constraints b2c = new SpringLayout.Constraints();

Spring yPadding = Spring.constant(20);  // odleglosc od gory
b1c.setX(Spring.constant(10));
b1c.setY(yPadding);
contentPane.add(b1, b1c);
Spring b1Right = springLayout.getConstraint(SpringLayout.EAST, b1);
Spring b2Left = Spring.sum(b1Right, Spring.constant(5));
b2c.setX(b2Left);
b2c.setY(yPadding);
contentPane.add(b2, b2c);
```

Proszę napisać program do testujący względne ustawienie 2 przycisków

Wskazówki:

- Zmiana wielkości pierwszego przycisku ma przesunąć drugi przycisk

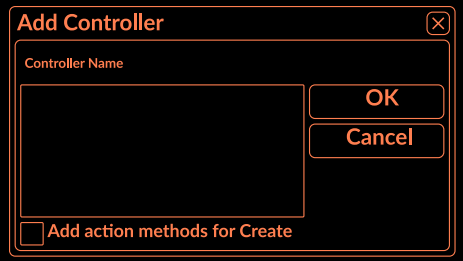
```java
package pap;
import java.awt.Container;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.Spring;
import javax.swing.SpringLayout;
import javax.swing.SwingUtilities;

public class Springz {

  public static void main(String[] args) {
    SwingUtilities.invokeLater(  () -> {
            JFrame frame = new JFrame("SpringLayout2");
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.setSize(300,200);
            Container contentPane = frame.getContentPane();

            SpringLayout springLayout = new SpringLayout();
            contentPane.setLayout(springLayout);
            JButton b1 = new JButton("Przycisk 1");
            JButton b2 = new JButton("Troche wiekszy przycisk 2");

            SpringLayout.Constraints b1c = new SpringLayout.Constraints();
            SpringLayout.Constraints b2c = new SpringLayout.Constraints();

            Spring yPadding = Spring.constant(20);  // odleglosc od gory
            b1c.setX(Spring.constant(10));
            b1c.setY(yPadding);
            contentPane.add(b1, b1c);
            Spring b1Right = springLayout.getConstraint(SpringLayout.EAST, b1);
            Spring b2Left = Spring.sum(b1Right, Spring.constant(5));
            b2c.setX(b2Left);
            b2c.setY(yPadding);
            contentPane.add(b2, b2c);
            frame.setVisible(true);
          }
    );
  }
}
```

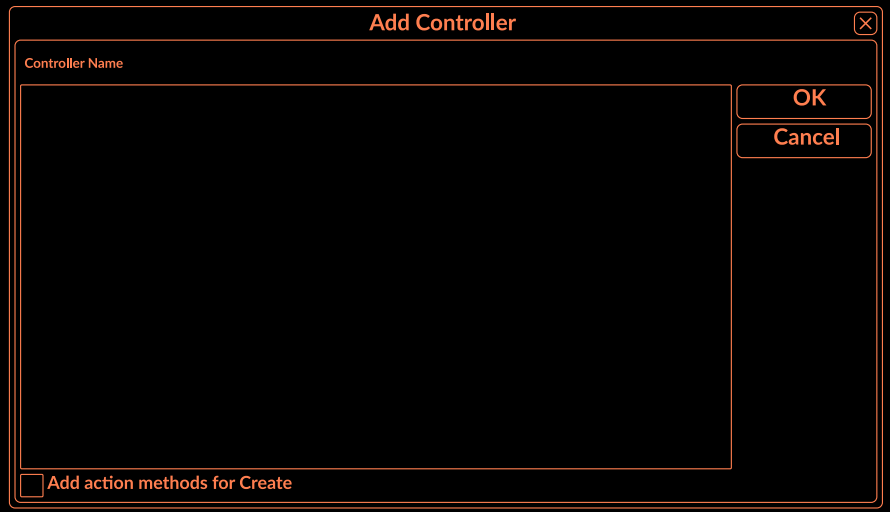
ograniczenia można dodawać na końcu:

```
SpringLayout sl = new SpringLayout();
contentPane.setLayout(sl);
JButton b1 = new JButton("Przycisk 1");
JButton b2 = new JButton("Troche wiekszy przycisk 2");
// dodajemy bez ograniczen
contentPane.add(b1);
contentPane.add(b2);
// najpierw ustawiamy warunki
sl.putConstraint(SpringLayout.WEST, b1, 10, SpringLayout.WEST, contentPane);
sl.putConstraint(SpringLayout.NORTH, b1, 20, SpringLayout.NORTH, contentPane);
sl.putConstraint(SpringLayout.WEST, b2, 5, SpringLayout.EAST, b1);
sl.putConstraint(SpringLayout.SOUTH, b2, 0, SpringLayout.SOUTH, b1);
sl.putConstraint(SpringLayout.SOUTH, contentPane, 10, SpringLayout.SOUTH, b1);
sl.putConstraint(SpringLayout.EAST, contentPane, 10, SpringLayout.EAST, b2);
// a teraz narzucimy taki sam rozmiar przyciskow poprzez powiekszenie pierwszego
SpringLayout.Constraints b1c = sl.getConstraints(b1);
SpringLayout.Constraints b2c = sl.getConstraints(b2);
Spring maxWidth = Spring.max(b1c.getWidth(), b2c.getWidth());
b1c.setWidth(maxWidth);
b2c.setWidth(maxWidth);
frame.pack();
frame.setVisible(true);
```

Polecenie (7)

Za pomocą SpringLayout zrealizować w kodzie okno z rozszerzalnym środkiem:





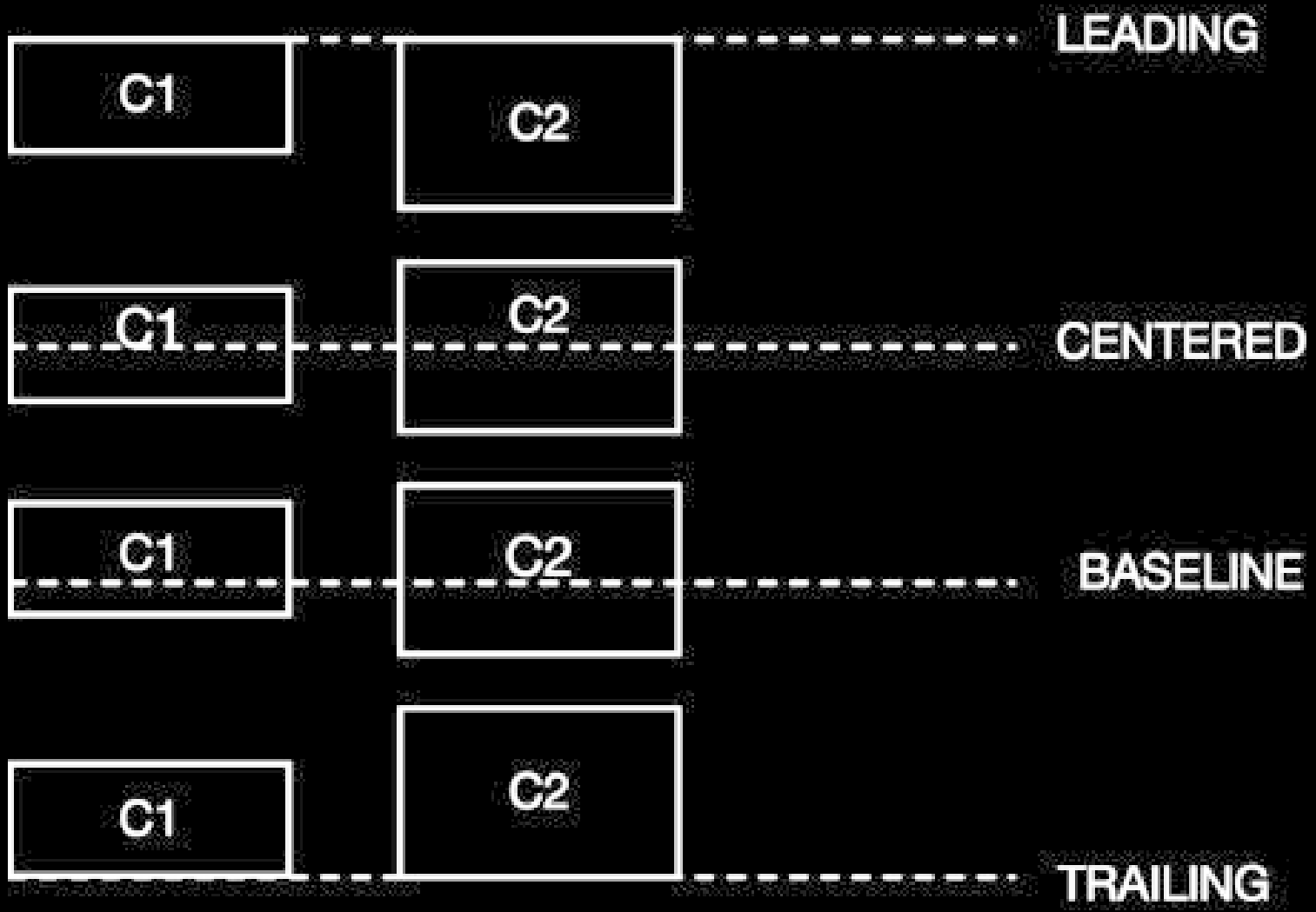
Wskazówki:

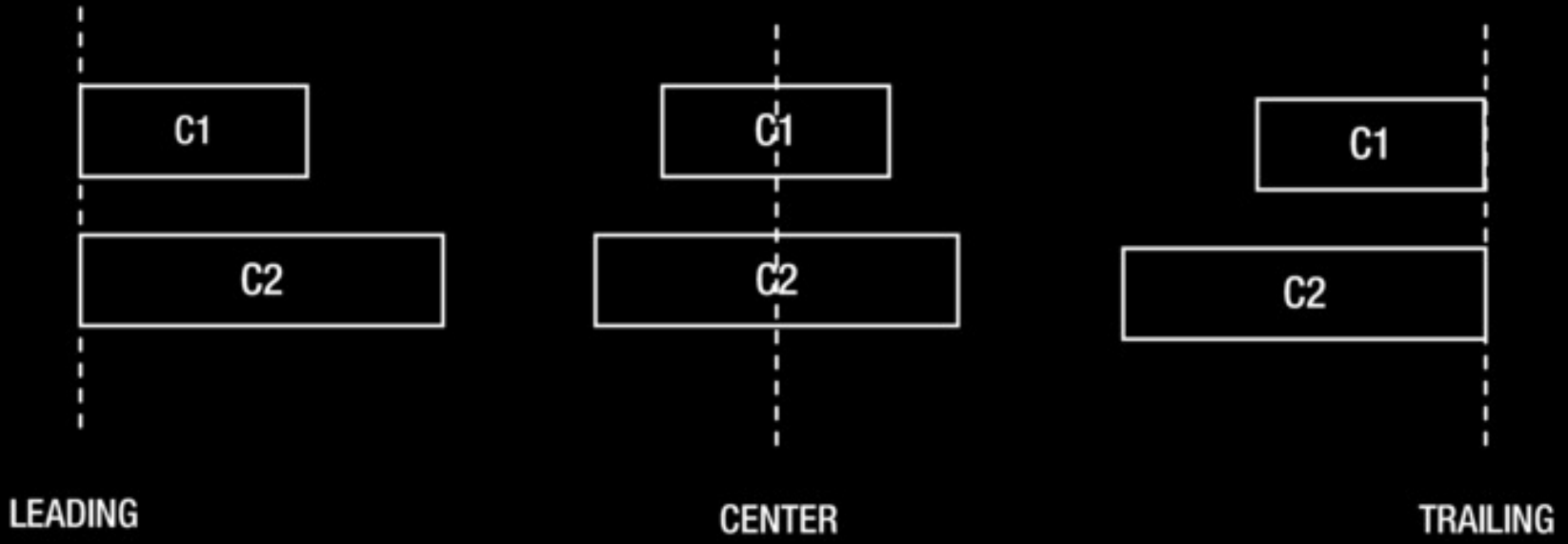
- Kontrolka, która ma się rozszerzać to `JTextArea`
- Aby miała paski przewijania, trzeba ją umieścić w kontrolce JScrollPane:

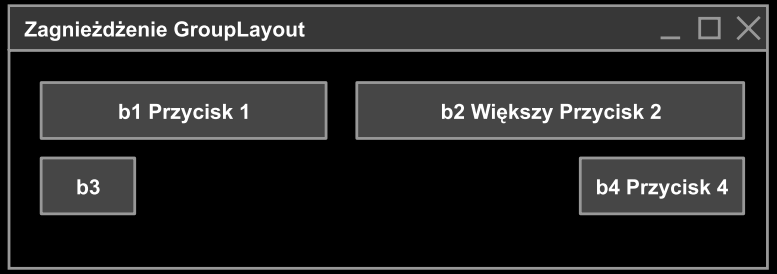
```
1 textArea = new JTextArea(8, 40);
2 JScrollPane scrollPane = new JScrollPane(textArea);
```

- grupa składa się z elementów:
  - komponent
  - odstęp
  - inna grupa
- mamy dwa rodzaje grup:
  - sekwencyjne - elementy kładzionoe jeden za drugim
  - równoległe - elementy kładzione równolegle: `baseline centered leading trailing`

te same komponenty są równoległe w jedym kierunku i sekwencyjne w drugim


C1
C2
LEADING
C1
C2
CENTERED
C1
C2
BASELINE
C1
C2
TRAILING


C1
C2
LEADING
C1
C2
CENTER
C1
C2
TRAILING




```
GroupLayout groupLayout = new GroupLayout(contentPane);
groupLayout.setAutoCreateGaps(true);
groupLayout.setAutoCreateContainerGaps(true);
contentPane.setLayout(groupLayout);

// Add four JButtons to the content pane
JButton b1 = new JButton("b1 Przycisk 1");
JButton b2 = new JButton("b2 Wiekszy Przycisk 2");
JButton b3 = new JButton("b3");
JButton b4 = new JButton("b4 Przycisk 4");
groupLayout.setHorizontalGroup(
        groupLayout.createSequentialGroup()
        .addGroup(groupLayout.createParallelGroup(LEADING, true)
                .addComponent(b1)
                .addComponent(b3))
        .addGroup(groupLayout.createParallelGroup(TRAILING, true)
                .addComponent(b2)
                .addComponent(b4))
);
groupLayout.setVerticalGroup(
        groupLayout.createSequentialGroup()
        .addGroup(groupLayout.createParallelGroup(BASELINE,true)
                .addComponent(b1)
                .addComponent(b2, GroupLayout.DEFAULT_SIZE,
                        GroupLayout.DEFAULT_SIZE, Integer.MAX_VALUE))
        .addGroup(groupLayout.createParallelGroup(BASELINE,true)
                .addComponent(b3)
                .addComponent(b4))
```

```
);
//groupLayout.linkSize(b1, b2, b4);
frame.pack();
```

- mamy klasę JOptionPane implementującą większość typowych sytuacji

```
int selection = JOptionPane.showConfirmDialog(parent, "Message", "Tytul",
      JOptionPane.OK_CANCEL_OPTION, JOptionPane.QUESTION_MESSAGE);
if (selection == JOptionPane.OK_OPTION) . . .
```

Funkcje zwracają:

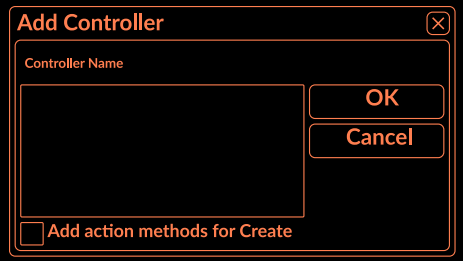
showMessageDialog - Brak

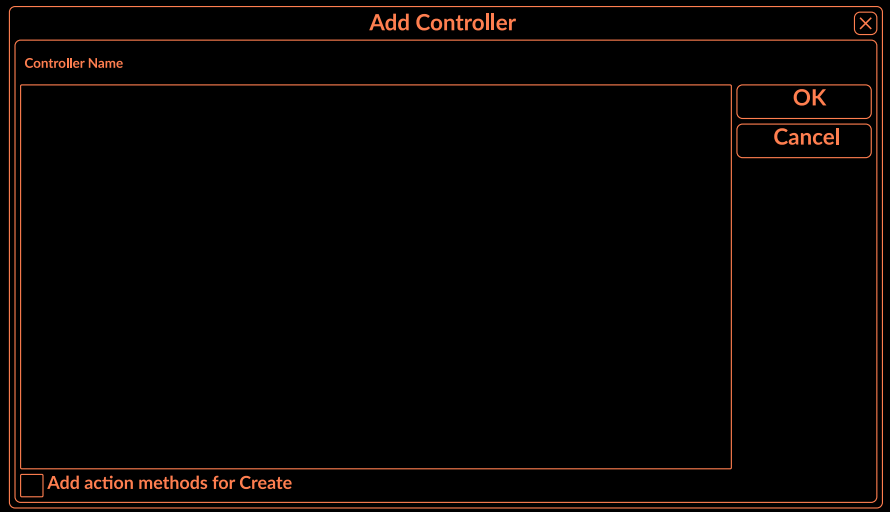
showConfirmDialog - Liczba całkowita reprezentująca wybraną opcję

showOptionDialog - Liczba całkowita reprezentująca wybraną opcję

showInputDialog - Łańcuch wprowadzony lub wybrany przez użytkownika

uzupełnić aplikację:





- Wciśnięcie klawisza OK powinno wyświetlić zawartość pola tekstowego
- Wciśnięcie Cancel powinno zapytać się o zgodę i po jej uzyskaniu skasować zawartość pola tekstowego

- Dziedziczymy po `JDialog`
- Nasz konstruktor musi dostać właściciela
- Tworzymy ciało okna
- Tworzymy obsługę zdarzeń
- Ustawiamy rozmiar okna

```
public AboutDialog extends JDialog {
    public AboutDialog(JFrame owner) {
        super(owner, "Test okna O programie", true);
        add(new JLabel("pap"),BorderLayout.CENTER);
        JPanel panel = new JPanel();
        JButton ok = new JButton("OK");
        ok.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent event) {
                setVisible(false);
            }
        });
        panel.add(ok);
        add(panel, BorderLayout.SOUTH);
        setSize(250, 150);
    }
}
----------------------
JDialog dialog = new AboutDialog(this);
dialog.setVisible(true);
```

Kontrolka JTextComponent - wprowadzanie tekstu:

- `getText()` - pobiera tekst
- `setText(String text)` - ustawia tekst
- `boolean isEditable()` - czy można edytować
- `void setEditable(boolean b)` - ustawia czy można edytować
- `void setColumns(int n)` - zgrubne określenie wielkości, ale użytkownik może wpisać więcej

ponieważ pola tekstowe trzeba względem sebie ułożyć, umieszczamy je na panelu:

```
1 JPanel panel = new JPanel();
2 JTextField textField = new JTextField("Lancuch testowy", 20);
3 panel.add(textField);
```

Kontrolka JLabel - umożliwia opis pola wprowadzania

- konstruktor: JLabel([String text,] [Icon icon,] [int horizntalAlignment])

Kontrolka JLabel - umożliwia opis pola wprowadzania

- konstruktor: JLabel([String text,] [Icon icon,] [int horizntalAlignment])
- gdzie horizntalAlignment to:
  - SwingConstants.LEFT - (domyślne)
  - SwingConstants.CENTER -
  - SwingConstants.RIGHT -
  - SwingConstants.LEADING -
  - SwingConstants.TRAILING -

Kontrolka JLabel - umożliwia opis pola wprowadzania

- konstruktor: JLabel([String text,] [Icon icon,] [int horizntalAlignment])
- gdzie horizntalAlignment to:
- możemy zdefiniować skróty klawiszowe i podpiąć kontrolkę która otrzyma fokus

```
JTextField nameTextField = new JTextField("Podaj nazwisko...");
JLabel nameLabel = new JLabel("Nazwisko:");
nameLabel.setDisplayedMnemonic('N');
nameLabel.setLabelFor(nameTextField);
contentPane.add(nameLabel);
contentPane.add(nameTextField);
```

Polecenie (9)

zrealizować własny Dialog Box implementujący wprowadzanie adresu:

konstruktor	opis
JButton()	Tworzy przycisk bez napisu i ikony
JButton(String text)	Tworzy przycisk z napisem
JButton(Icon icon)	Tworzy przycisk z ikoną
JButton(String text, Icon icon)	Tworzy przycisk z ikoną i napisem
JButton(Action action)	Tworzy przycisk który bierze ustawienia z obiektu action

```
Icon okIcon = new ImageIcon("button_ok.png");
Icon cancelIcon = new ImageIcon("button_cancel.png");
JButton first = new JButton("First", okIcon);
JButton second = new JButton("Second",cancelIcon);
```

metoda	opis
addActionListener(ActionListener l)	dodaje procedurę obsługi wciśnięcia przycisku
setMnemonic(int mnemonic)	klawisz, który z altem uruchamia przycisk
int getMnemonic()	zwraca kod skrótu
Icon getIcon()	Zwraca obiekt ikonki
String getText()	Zwraca napis na przycisku
void setEnabled(Boolean enabled)	ustawianie wyszarzenia

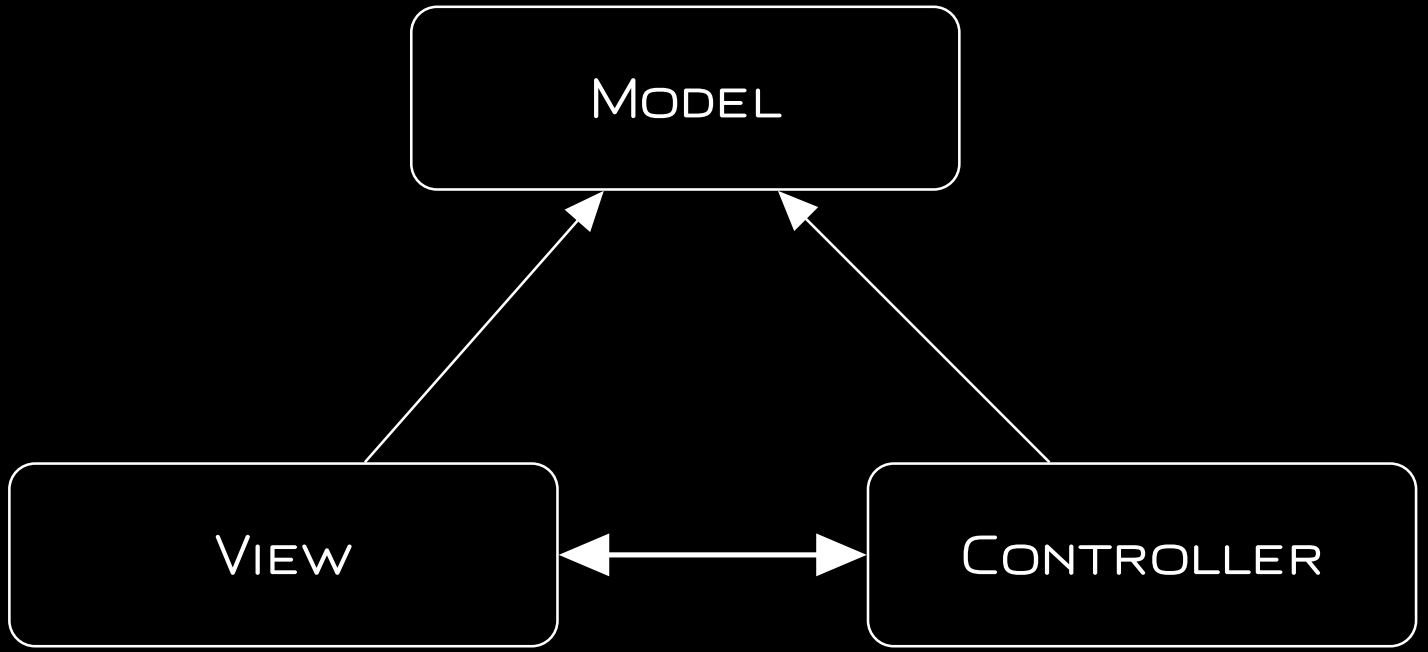
```
first.addActionListener(e->System.out.println("first"));
first.addActionListener(e->System.out.println("drugi raz"));
first.setMnemonic('I');
first.setMnemonic(KeyEvent.VK_I);
```

- Jest to Interfejs do przechowywania danych przycisku. Można przypiąć wiele przycisków i będą się one zachowywały identycznie
- `AbtractAction` to abstrakcyjna klasa implementująca ten interfejs

metoda	**opis**
Action getAction()	Zwraca akcję związaną z obiektem
void setAction(Action a)	Ustawia obiekt akcji. jeżeli był to nadpisuje. Zdarzeń nie nadpisuje

```
1  Icon okIcon = new ImageIcon("button_ok.png");
2  public class FirstAction extends AbstractAction {
3      public FirstAction() {
4          super("First",okIcon);
5      }
6      @Override
7      public void actionPerformed(ActionEvent event) {
8          System.out.println(getValue(NAME));
9      }
10 }
11 Action firstAction = new FirstAction();
```

- Wszystkie komponenty Swinga zbudowane są zgodnie ze wzorcem MVC:



- Widok nie musi wszystkiego wyświetlać

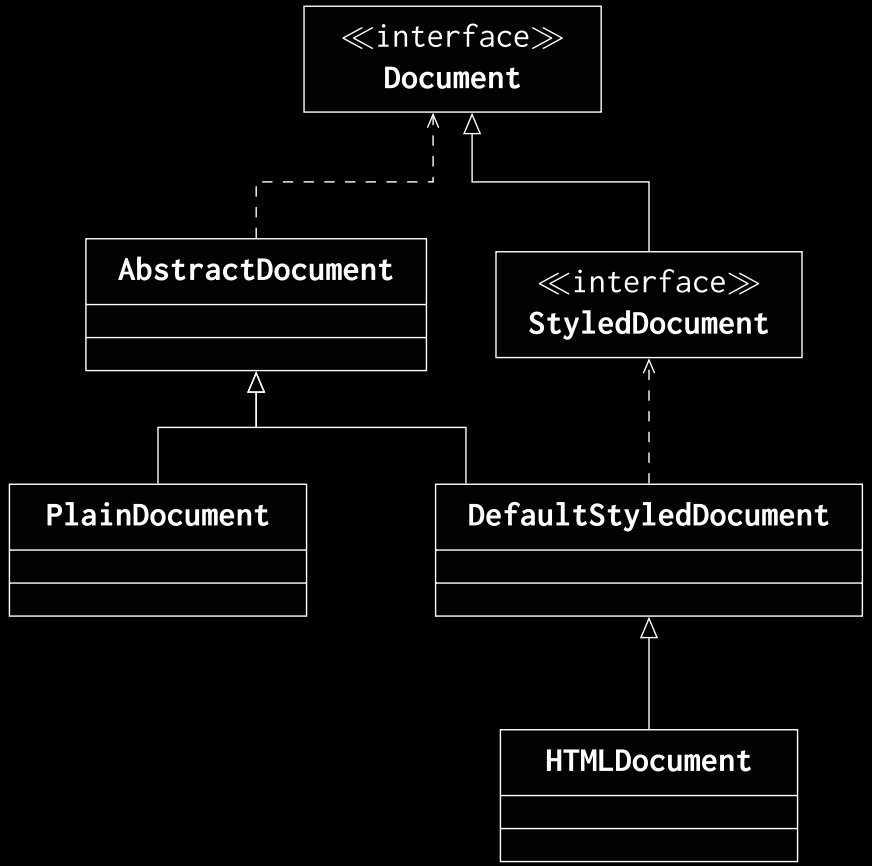
```
frame.setLayout(new GridLayout(2, 0));
JLabel namesLabel = new JLabel("Name:") ;
JLabel mirroredNameLabel = new JLabel("Mirrored Name:") ;
JTextField names = new JTextField(20);
JTextField mirroredName = new JTextField(20);
contentPane.add(namesLabel);
contentPane.add(names);
contentPane.add(mirroredNameLabel);
contentPane.add(mirroredName);
Document nameModel = names.getDocument();
mirroredName.setDocument(names.getDocument());
frame.setVisible(true);
```

```
public class LimitedCharDocument extends PlainDocument {
    private int limit = -1; // < 0 means an unlimited characters

    public LimitedCharDocument() {
    }

    public LimitedCharDocument(int limit) {
        this.limit = limit;
    }

    @Override
    public void insertString(int offset, String str, AttributeSet a)
                            throws BadLocationException {
        String newString = str;
        if (limit >=0 && str != null) {
                // Check for the limit
                int currentLength = this.getLength() ;
                int newTextLength = str.length();
                if (currentLength + newTextLength > limit) {
                        newString = str.substring(0, limit - currentLength);
                }
        }

        super.insertString(offset, newString, a);
    }
}
```

Polecenie (10)

zrealizować klasę JLimitedText przyjmującą limit liter

- Hierarchia


≪interface≫
Document
AbstractDocument
≪interface≫
StyledDocument
PlainDocument
DefaultStyledDocument
HTMLDocument

za pomocą klasy JEditorPane zrealizować edytor

Uwagi

- klasa abstrakcyjna JTextComponent wystawia tablicę akcji oraz metozy zaznaczania (selekcji)

- często ta sama funkcjonalność aplikacji wywoływana na wiele sposobów(np. menu lub pasek narzędzi)
- w javie możemy podpiąć tego samego słuchacza do wielu źródeł zdarzeń
- nadal mamy jednak problem, jak zapewnić spójność w interfejsie:
  - jednoczesne wyszarzanie akcji
  - ustawianie wspólnych ikon
  - ustawianie takich samych nazw

- Zamiast tego mamy interfejs `Action`

```
public interface Action extends ActionListener {
    public static final String NAME = "Name";
    public static final String SHORT_DESCRIPTION = "ShortDescription";
    public static final String LONG_DESCRIPTION = "LongDescription";
    public static final String SMALL_ICON = "SmallIcon";
    public static final String ACTION_COMMAND_KEY = "ActionCommandKey";
    public static final String ACCELERATOR_KEY="AcceleratorKey";
    public static final String MNEMONIC_KEY="MnemonicKey";
    public static final String SELECTED_KEY = "SwingSelectedKey";
    public static final String DISPLAYED_MNEMONIC_INDEX_KEY = "SwingDisplayedMnemonicIndexKey";
    public static final String LARGE_ICON_KEY = "SwingLargeIconKey";
    public Object getValue(String key);
    public void putValue(String key, Object value);
    public void setEnabled(boolean b);
    public boolean isEnabled();
    public void addPropertyChangeListener(PropertyChangeListener listener);
    public void removePropertyChangeListener(PropertyChangeListener listener);
}
```

- łączy opis (nazwa, ikona) komendy z parametrami jej wykonania

- Tworzymy obiekt `MenuBar` (1) i dodajemy go do okna (2)

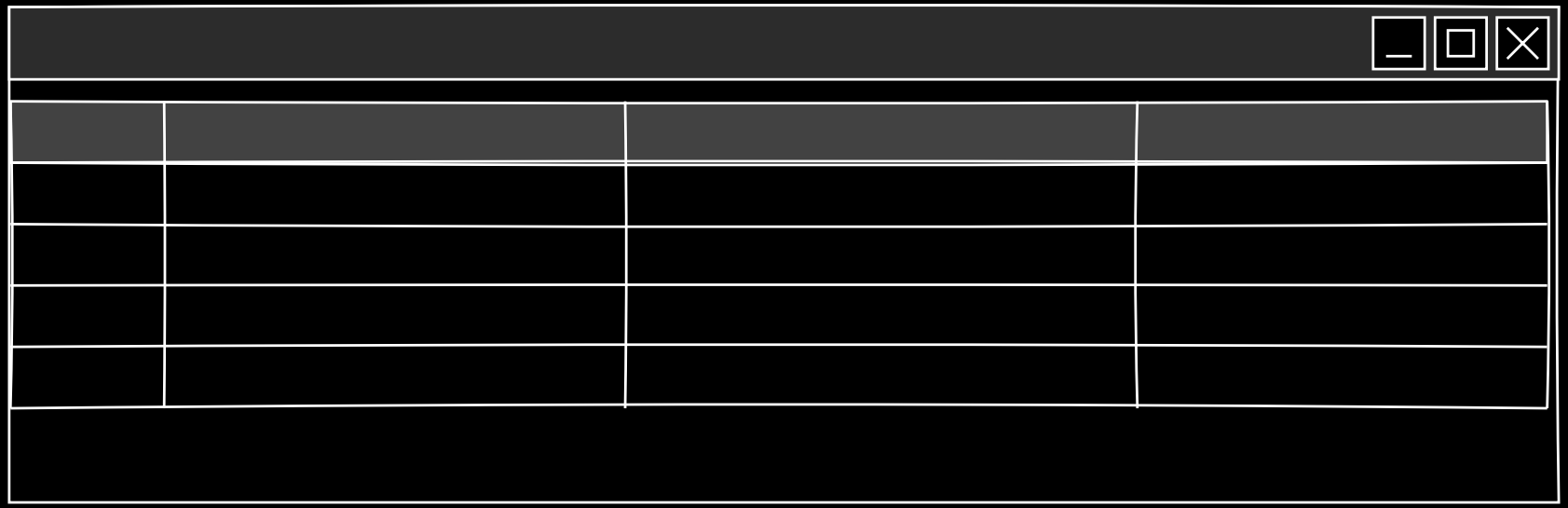
```
void run() {
  SwingUtilities.invokeLater(() -> {
    JFrame frame = new JFrame("Menu example");
    frame.setSize(400, 600);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    JMenuBar menuBar = new JMenuBar();
    frame.setJMenuBar(menuBar);
    JMenu editMenu = new JMenu("Edit");
    menuBar.add(editMenu);
    JMenuItem changeItem = new JMenuItem("Change");
    editMenu.add(changeItem);
    changeItem.addActionListener(e->changeItem.setEnabled(false));
    editMenu.addSeparator();
    JMenu optionsMenu = new JMenu("Options");; // a submenu
    JMenuItem helpItem = new JMenuItem("Help");
    optionsMenu.add(helpItem);
    editMenu.add(optionsMenu);
    frame.setVisible(true);
  });
}
```

- Najprostsza tablica

```
String[] columnNames = {"Imie","Nazwisko","Rok", "Studia" };
Object[][] data = {
    {"Jan", "Kowalski",new Integer(1984), new Boolean(true)},
    {"Adam", "Nowak",new Integer(1980), new Boolean(false)},
    {"Zosia", "Samosia",new Integer(1970), new Boolean(true)}
};
final JTable table = new JTable(data, columnNames);
table.setPreferredScrollableViewportSize(new Dimension(500, 70));
table.setFillsViewportHeight(true);
JScrollPane scrollPane = new JScrollPane(table);
frame.getContentPane().add(scrollPane);
frame.pack();
```

- Jest edycja - ale gdzie zapisują się wyniki?

```java
package pap;
import java.awt.Dimension;
import java.awt.EventQueue;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTable;

public class SimpleFrame {
  protected JFrame frame;
  public static void main(String[] args) {
    EventQueue.invokeLater(new Runnable() {
      public void run() {
        try {
          SimpleFrame window = new SimpleFrame();
          window.frame.setVisible(true);
        } catch (Exception e) {
          e.printStackTrace();
        }
      }
    });
  }
  public SimpleFrame() {
    String[] columnNames = {"Imie","Nazwisko","Rok", "Studia" };
    Object[][] data = {
        {"Jan", "Kowalski",new Integer(1984), new Boolean(true)},
        {"Adam", "Nowak",new Integer(1980), new Boolean(false)},
        {"Zosia", "Samosia",new Integer(1970), new Boolean(true)}
    };
    final JTable table = new JTable(data, columnNames);
    table.setPreferredScrollableViewportSize(new Dimension(500, 70));
    table.setFillsViewportHeight(true);
    JScrollPane scrollPane = new JScrollPane(table);
    frame.getContentPane().add(scrollPane);
    frame.pack();
  }
}
```



- wszystkie pola edytowalne, modyfikacja danych źródłowych
- pola przedstawiane jako stringi
- równa szerokość kolumn(można użyć `setPreferredWidth`

```
TableColumn column = null;
for (int i = 0; i < columnNames.length; i++) {
    column = table.getColumnModel().getColumn(i);
    if (i == 2)
        column.setPreferredWidth(200); // druga kolumna
    else
        column.setPreferredWidth(50);
}
```

- możemy używać klasy `DefaultTableModel`
- możemy używać jej potomnych
- możemy używać potomnych klasy `AbstractTableModel`

  fireTableCellUpdated -zmieniono niektóre cele

  fireTableRowsUpdated -zmieniono niektóre wiersze

  fireTableDataChanged -zmieniono całe dane w tablicy

  fireTableRowsInserted -dodano wiersze

  fireTableRowsDeleted -skasowano wiersze

  fireTableStructureChanged -cała struktura z danymi nieaktualna

```
1  package pap;
2  package proz;
3  import java.awt.Dimension;
4  import java.awt.EventQueue;
5  import javax.swing.JFrame;
6  import javax.swing.JScrollPane;
7  import javax.swing.JTable;
8  import javax.swing.event.TableModelEvent;
9  import javax.swing.event.TableModelListener;
10 import javax.swing.table.AbstractTableModel;
11
12 public class TableModelTest {
13   protected JFrame frame;
14   public static void main(String[] args) {
15     EventQueue.invokeLater(new Runnable() {
16       public void run() {
17         try {
18           TableModelTest window = new TableModelTest();
19           window.frame.setVisible(true);
20         } catch (Exception e) {
21           e.printStackTrace();
22         }
23       }
24     });
25   }
26   public TableModelTest() {
27     frame = new JFrame();
28     frame.setBounds(100, 100, 450, 300);
29     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
30
31     final JTable table = new JTable(new AbstractTableModel() {
32       static final long serialVersionUID = 1L;
33       private String[] columnNames = { "Imie", "Nazwisko", "Rok", "Studia" };
34       private Object[][] data = { { "Jan", "Kowalski", Integer.valueOf(1984), Boolean.valueOf(true) },
35           { "Adam", "Nowak", Integer.valueOf(1980), Boolean.valueOf(false) },
36           { "Zosia", "Samosia", Integer.valueOf(1970), Boolean.valueOf(true) } };
37
38       @Override
39       public int getRowCount() {
40         return data.length;
```

```
41        }
42
43        @Override
44        public int getColumnCount() {
45          return columnNames.length;
46        }
47
48        @Override
49        public Object getValueAt(int rowIndex, int columnIndex) {
50          return data[rowIndex][columnIndex];
51        }
52        // gdy chcemy aby tablica pokazywala edytowala dane a nie stringi
53        public Class<?> getColumnClass(int c) {
54          return getValueAt(0, c).getClass();
55        }
56
57        // tylko dla tablic edytowanych
58        public boolean isCellEditable(int row, int col) {
59          return col == 1; // edytowalne tylko nazwisko
60        }
61
62        // tylko dla tablic edytowanych
63        public void setValueAt(Object value, int row, int col) {
64          data[row][col] = value;
65          fireTableCellUpdated(row, col);
66        }
67
68        // gdy nie chcemy nazw A B C
69        public String getColumnName(int col) {
70          return columnNames[col];
71        }
72      });
73      table.setPreferredScrollableViewportSize(new Dimension(500, 70));
74      table.setFillsViewportHeight(true);
75      table.getModel().addTableModelListener(new TableModelListener() {
76        @Override
77        public void tableChanged(TableModelEvent e) {
78          int row = e.getFirstRow();
79          int column = e.getColumn();
80          AbstractTableModel model = (AbstractTableModel) e.getSource();
81          String columnName = model.getColumnName(column);  // not used
82          Object data = model.getValueAt(row, column);      // not used
83        }
84      });
85
```

```
    // Utworz okienko przewijania i dodaj do niego tabele.
    JScrollPane scrollPane = new JScrollPane(table);

    // Dodaj okienko przewijania do tego panelu.
    frame.getContentPane().add(scrollPane);
    frame.pack();
  }
}
```

- Wyzerowanie układu wymusza ręczne ustawienie pozycji i położenia (setBounds)

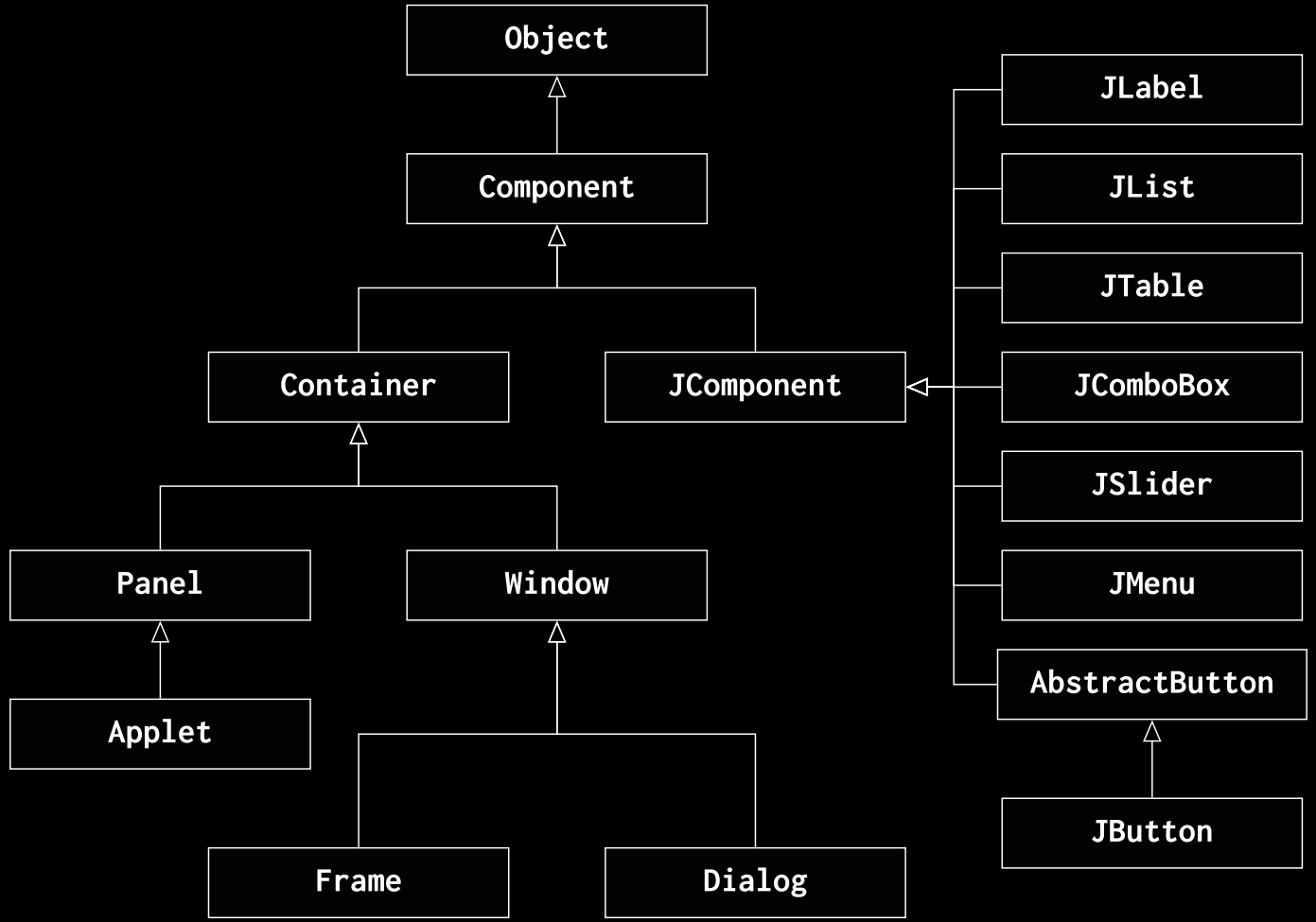
```
1  package proz;
2  import javax.swing.JButton;
3  import javax.swing.JTextField;
4  import javax.swing.JLabel;
5  import javax.swing.JFrame;
6  import javax.swing.SwingUtilities;
7  public class TextFieldExample {
8    void run() {
9      SwingUtilities.invokeLater(() -> {
10       JFrame frame = new JFrame("JTextField");
11       frame.setSize(400, 400);
12       frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
13       JTextField ed=new JTextField("jeszcze niczego nie ma");   // final ...
14       ed.setBounds(50,100,200,30);
15       JLabel lbl=new JLabel("czekam na napis ...");             // final ...
16       lbl.setBounds(80,150,200,30);
17       JButton btn = new JButton("nacisnij");                    // final ...
18       btn.addActionListener(e->{lbl.setText(ed.getText());});
19       btn.setBounds(100,200,200,30);
20       frame.add(btn);frame.add(lbl);frame.add(ed);
21       frame.setLayout(null);
22       frame.setVisible(true);
23     });
24   }
25   public static void main(String[] args) {
26     new TextFieldExample().run();              // tworzymy obiekt
27   }
28 }
```

- w kodzie zdarzenia mamy dostęp do zmiennych metody (poprzez ich zamknięcie)

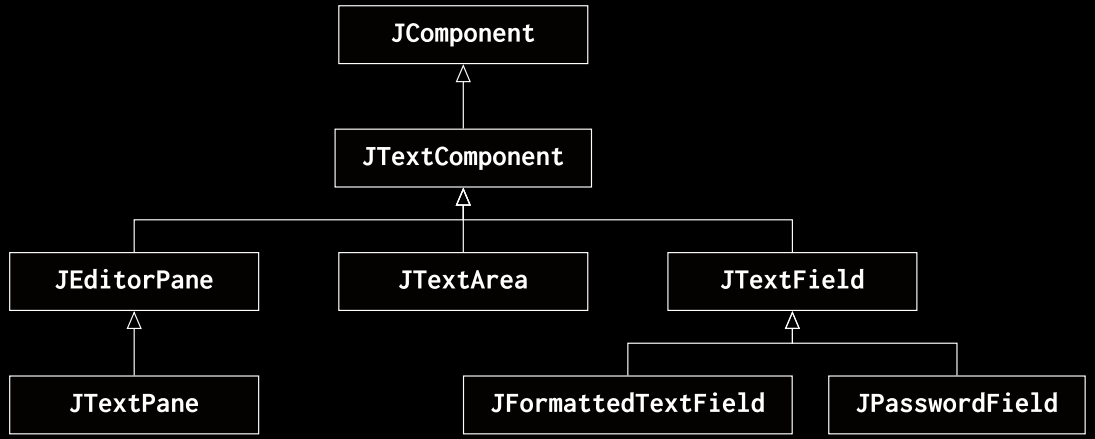
```
1  package proz;
2  import javax.swing.*;
3  public class TextFieldExample {
4    void run() {
5      SwingUtilities.invokeLater(() -> {
6        JFrame frame = new JFrame("JTextField");
7        frame.setSize(400, 400);
8        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
9        JTextField ed=new JTextField("jeszcze niczego nie ma");   // final ...
10       ed.setBounds(50,100,200,30);
11       JLabel lbl=new JLabel("czekam na napis ...");             // final ...
12       lbl.setBounds(80,150,200,30);
13       JButton btn = new JButton("nacisnij");                    // final ...
14       btn.addActionListener(e->{lbl.setText(ed.getText());});
15       btn.setBounds(100,200,200,30);
16       frame.add(btn);frame.add(lbl);frame.add(ed);
17       frame.setLayout(null);
18       frame.setVisible(true);
19     });
20   }
21   public static void main(String[] args) {
22     new TextFieldExample().run();              // tworzymy obiekt
23   }
24 }
```

```java
package proz;
import javax.swing.JButton;
import javax.swing.JFrame;
import java.awt.GridLayout;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import javax.swing.SwingUtilities;
public class GridLayoutExample {
  private static int count = 0;
  public static void main(String[] args) {
    SwingUtilities.invokeLater(() -> {
      JFrame frame = new JFrame("GridLayoutExample");
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.setSize(350, 350);
      frame.setLayout(new GridLayout(3, 3, 10, 10));
      for(int i=0;i<5;i++) {
        var idx = i;
        JButton b1 = new JButton("B"+i);
        b1.addActionListener(new ButtonListener(i));
        frame.add(b1);
      }
      frame.setVisible(true);
    });
  }
}
class ButtonListener implements ActionListener {
  private int index;
  public ButtonListener(int index) { this.index=index;}
  @Override
  public void actionPerformed(ActionEvent e) {
    ((JButton)e.getSource()).setText("A"+this.index);
    ((JButton)e.getSource()).setEnabled(false);
  }
}
```

```java
package proz;
import javax.swing.JButton;
import javax.swing.JFrame;
import java.awt.GridLayout;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import javax.swing.SwingUtilities;
public class GridLayoutExample {
  private static int count = 0;
  public static void main(String[] args) {
    SwingUtilities.invokeLater(() -> {
      JFrame frame = new JFrame("GridLayoutExample");
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.setSize(350, 350);
      frame.setLayout(new GridLayout(3, 3, 10, 10));
      for(int i=0;i<5;i++) {
        var idx = i;
        JButton b1 = new JButton("B"+i);
        b1.addActionListener(e-> {
          ((JButton)e.getSource()).setText("A"+idx+count++);
          b1.setEnabled(false);
        });
        frame.add(b1);
      }
      frame.setVisible(true);
    });
  }
}
```

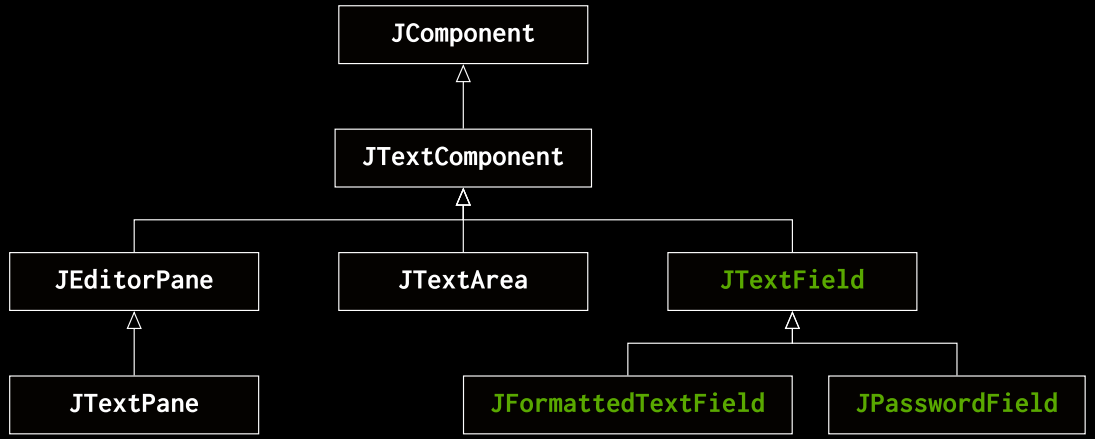

Object
Component
Container
JComponent
Panel
Window
Applet
Frame
Dialog
JLabel
JList
JTable
JComboBox
JSlider
JMenu
AbstractButton
JButton

Pola tekstowe dzielimy ze względu na:



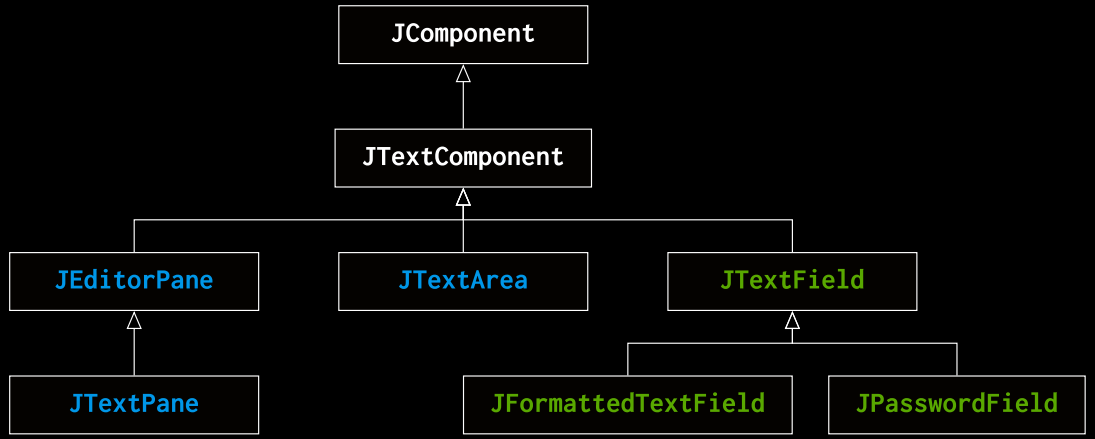
Pola tekstowe dzielimy ze względu na:

- ilość linii:
  - jednoliniowe (JTextField, JPasswordField, oraz JFormattedTextField)



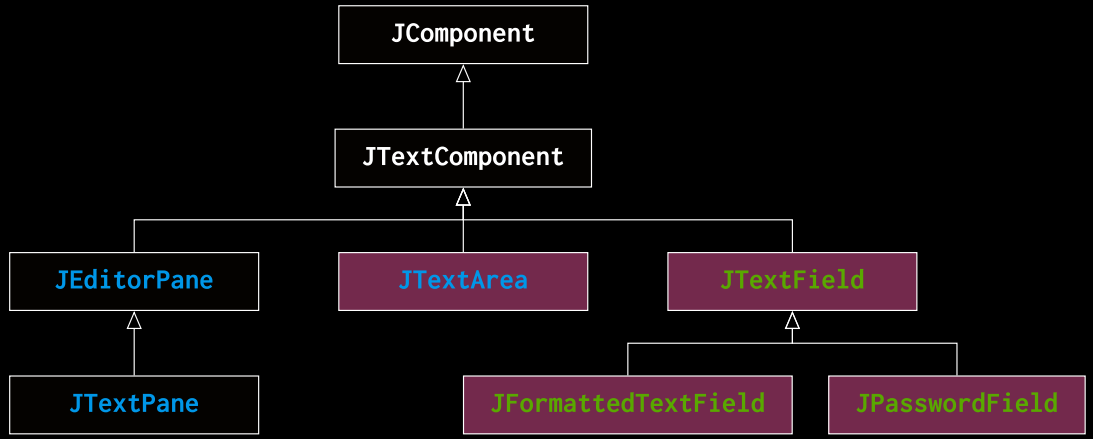
Pola tekstowe dzielimy ze względu na:

- ilość linii:
  - jednoliniowe (JTextField, JPasswordField, oraz JFormattedTextField)
  - wieloliniowe (JTextArea, JEditorPane, oraz JTextPane)



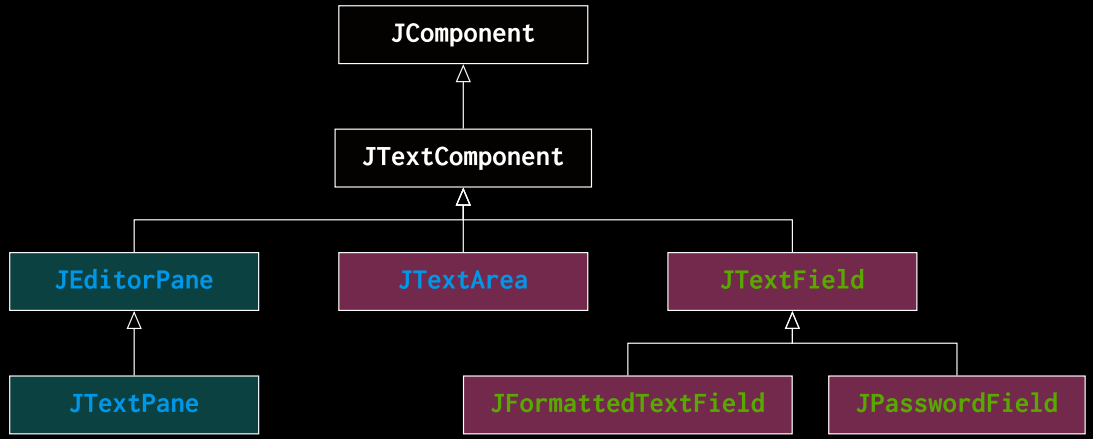
Pola tekstowe dzielimy ze względu na:

- ilość linii:
  - jednoliniowe (JTextField, JPasswordField, oraz JFormattedTextField)
  - wieloliniowe (JTextArea, JEditorPane, oraz JTextPane)
- formatowanie:
  - proste (JTextField, JPasswordField, JFormattedTextField, oraz JTextArea)



Pola tekstowe dzielimy ze względu na:

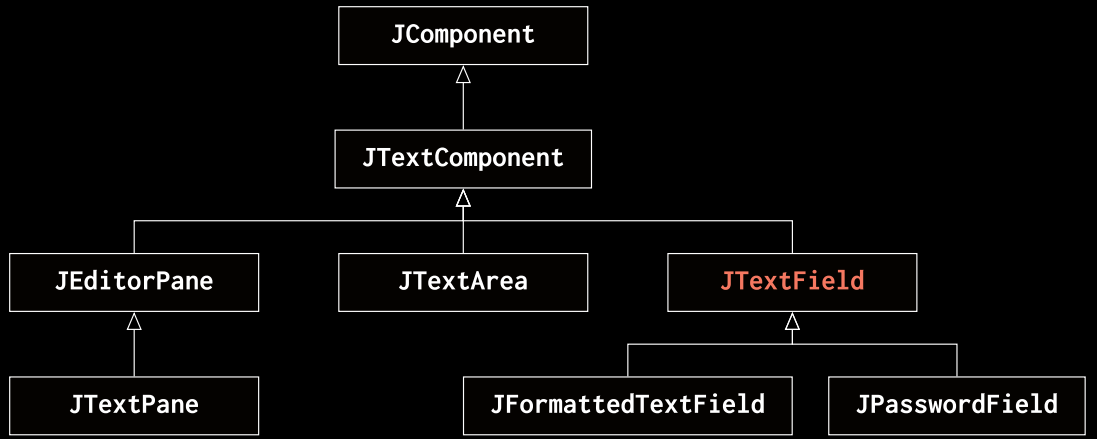
- ilość linii:
  - jednoliniowe (JTextField, JPasswordField, oraz JFormattedTextField)
  - wieloliniowe (JTextArea, JEditorPane, oraz JTextPane)
- formatowanie:
  - proste (JTextField, JPasswordField, JFormattedTextField, oraz JTextArea)
  - formatowane (JEditorPane oraz JTextPane)



- możliwość wprowadzenia linii tekstu

  getText -odczyt do stringu

  setText -zapis do stringu



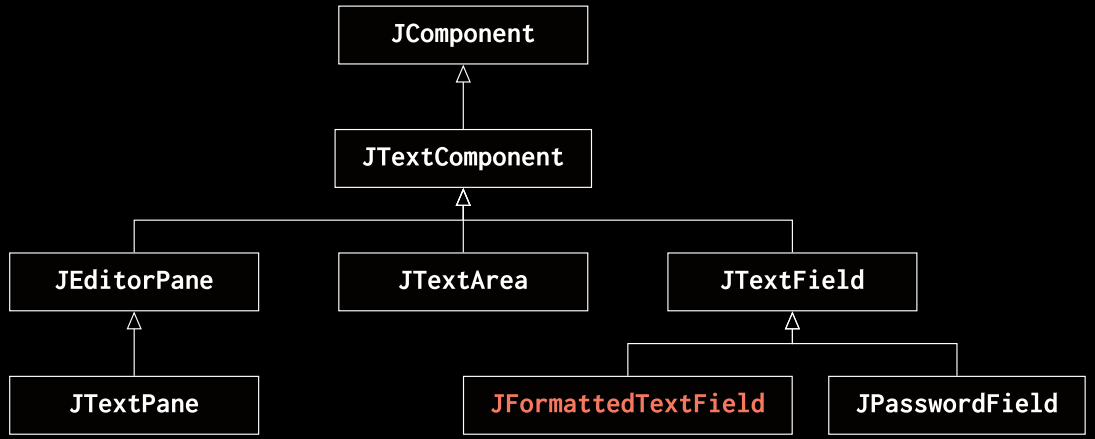
- możliwość wprowadzenia linii tekstu

  getText -odczyt do stringu

  setText -zapis do stringu

```
1 import javax.swing.JTextField;
2 ...
3 JTextField ed = new JTextField("JTextField");
4 System.out.println(ed.getText()); // zwraca string
```

- możliwość wprowadzenia linii tekstu w wybranym formacie (procenty, kod pocztowy, etc)



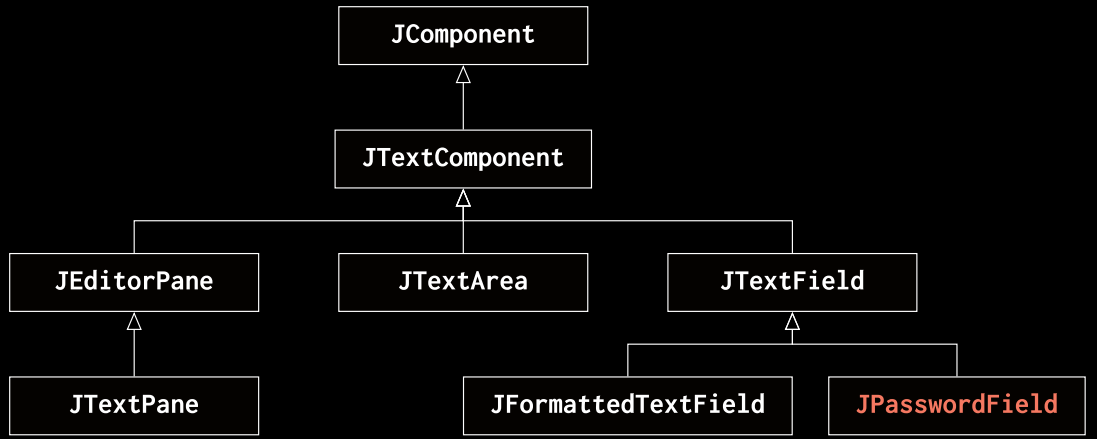
- możliwość wprowadzenia linii tekstu w wybranym formacie (procenty, kod pocztowy, etc)

```
import javax.swing.JFormattedTextField;
import java.text.NumberFormat;
...
double rate = 12.5;
var percentFormat = NumberFormat.getNumberInstance();
percentFormat.setMinimumFractionDigits(2);
var fed = new JFormattedTextField(percentFormat);
fed.setValue(Double.valueOf(rate));
fed.setColumns(10);
System.out.println(fed.getValue()); // zwraca Object
```

- możliwość wprowadzenia linii tekstu bez pokazywania wpisywanego tekstu (gwiazdki)

  getPassword -odczyt do stringu

  setPassword -zapis do stringu



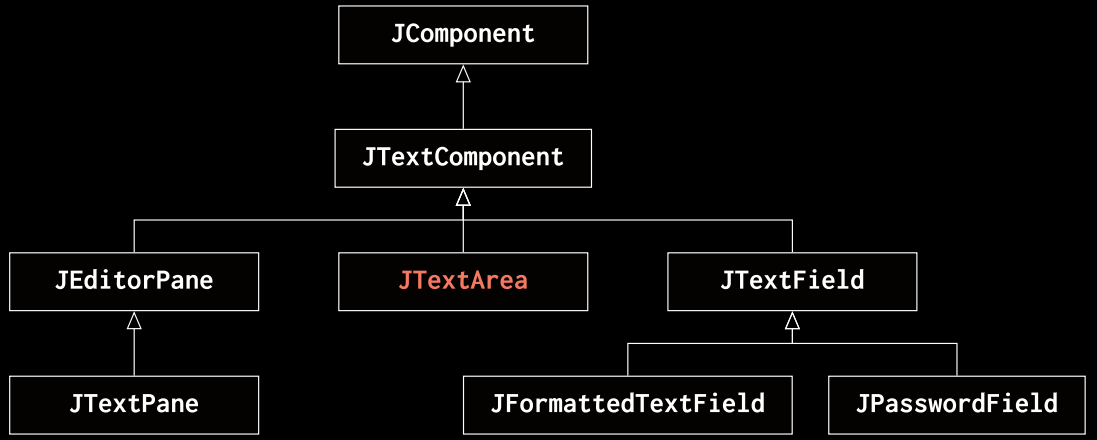
- możliwość wprowadzenia linii tekstu bez pokazywania wpisywanego tekstu (gwiazdki)

  getPassword -odczyt do stringu

  setPassword -zapis do stringu

```
1 import javax.swing.JPasswordField;
2 ...
3 var ped = new JPasswordField("ala ma kota");
4 System.out.println(ped.getPassword());
5 frame.add(ped);
```

- Obszar tekstowy to „zwykły” element tekstowy, co oznacza, że chociaż może wyświetlać tekst dowolną czcionką, cały tekst jest zapisany tą samą czcionką.



- Obszar tekstowy to „zwykły” element tekstowy, co oznacza, że chociaż może wyświetlać tekst dowolną czcionką, cały tekst jest zapisany tą samą czcionką.

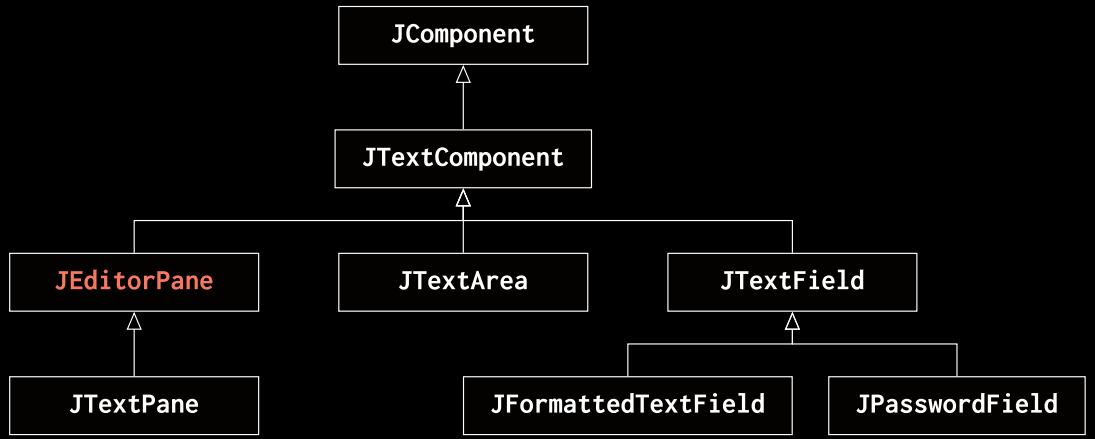
```
import javax.swing.JTextArea;
...
var tea = new JTextArea(
       "This is an editable JTextArea. \n" +
       "A text area is a plain text component" );
tea.setFont(new Font("Serif", Font.ITALIC, 16));
tea.setLineWrap(true);
tea.setWrapStyleWord(true);

var esp_tea = new JScrollPane(tea);
esp_tea.setVerticalScrollBarPolicy(
         JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
esp_tea.setPreferredSize(new Dimension(250, 145));
esp_tea.setMinimumSize(new Dimension(10, 10));
esp_tea.setBounds(50, 150, 300, 150);
frame.add(esp_tea);
```

- edycja tekstu w formatach: `text/plain`, `text/html`, `text/rtf` (domyślny EditorKit to `text/plain`)

  setText -ustawienie ze stringu, z domyślnym EditorKit

  read - czytanie z obiektu reader

  setPage - czytanie z url, wybieramy EditorKit na podstawie url



- edycja tekstu w formatach: `text/plain`, `text/html`, `text/rtf` (domyślny EditorKit to `text/plain`)

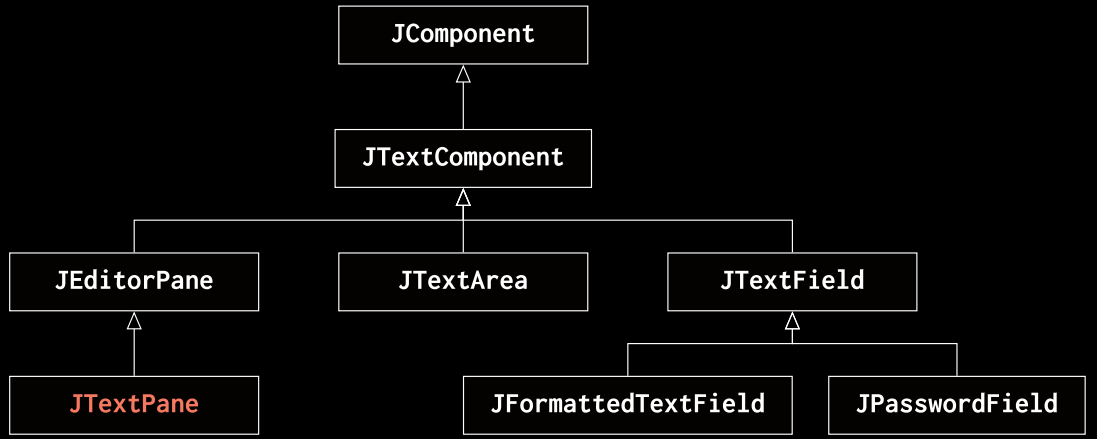
  setText -ustawienie ze stringu, z domyślnym EditorKit

  read - czytanie z obiektu reader

  setPage - czytanie z url, wybieramy EditorKit na podstawie url

```
1  import javax.swing.JEditorPane;
2  import javax.swing.JScrollPane;
3  import java.awt.Dimension;
4  ...
5  var ep = new JEditorPane("text/html", String.join("\n",
6  "<html>",       // mozna tekst nadac w konstruktorze
7    "<body>",
8      "to <span style=\"color:#f57860;\">jest</span> plik",
9    "</body>",
10 "</html>"));
11 JScrollPane esp = new JScrollPane(ep);
12 esp.setVerticalScrollBarPolicy(
13                  JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
14 esp.setPreferredSize(new Dimension(250, 145));
15 esp.setMinimumSize(new Dimension(10, 10));
```

- stylizowany komponent tekstowy, który obsługuje osadzone komponenty i osadzone ikony



- stylizowany komponent tekstowy, który obsługuje osadzone komponenty i osadzone ikony

```
import javax.swing.JTextPane;
import javax.swing.text.BadLocationException;
import javax.swing.text.Style;
import javax.swing.text.StyleConstants;
import javax.swing.text.StyleContext;
import javax.swing.text.StyledDocument;
...
protected void addStyles(StyledDocument doc) {
  Style def=StyleContext.getDefaultStyleContext()
            .getStyle(StyleContext.DEFAULT_STYLE);
  Style regular = doc.addStyle("regular", def);
  StyleConstants.setFontFamily(def, "SansSerif");
  Style s = doc.addStyle("italic", regular);
  StyleConstants.setItalic(s, true); // ustawianie
}
...
JTextPane tpane = new JTextPane();
StyledDocument doc = tpane.getStyledDocument();
addStyles(doc);
try {
  doc.insertString(doc.getLength(),
     "to jest regular ", doc.getStyle("regular"));
  doc.insertString(doc.getLength(),
     "to jest italic ",  doc.getStyle("italic"));
} catch (BadLocationException ble) {
  System.err.println("Couldn't insert text.");
}
JScrollPane psp = new JScrollPane(tpane);
psp.setVerticalScrollBarPolicy(
           JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
psp.setPreferredSize(new Dimension(250, 155));
psp.setMinimumSize(new Dimension(10, 10));
frame.add(psp);
```