

Stefan Sokołowski

PROGRAMOWANIE OBIEKTOWE II

wykład 10

Inst. Informatyki Stosowanej, ANS
Elbląg, 2024/2025

Wykład 10, str. 1

Podsumowanie reakcji na zdarzenia

Biblioteczny interfejs dla nasłuchiaczy:

```
interface zdarzenieListener {  
    public void reakcja(zdarzenieEvent e);  
}
```

Konstrukcja nasłuchiacza:

```
new zdarzenieListener() {  
    public void reakcja(zdarzenieEvent e) {  
        ...co zrobić w reakcji na zdarzenie...  
    }  
}
```

Reakcja na zdarzenia różnego rodzaju

Do obsługi różnych zdarzeń:

nazwa interfejsu	co obsługuje
<code>ActionListener</code>	kliknięcia przycisków w oknie
<code>WindowListener</code>	otwieranie/zamykanie okien
<code>KeyListener</code>	klawisze na klawiaturze
<code>MouseListener</code>	przyciski na myszy
<code>MouseMotionListener</code>	ruchy myszy
<code>MouseWheelListener</code>	kółeczko na myszy
	i jeszcze inne...

W każdym z tych interfejsów są metody *callback* wymagające zaimplementowania.

Operacje okienne

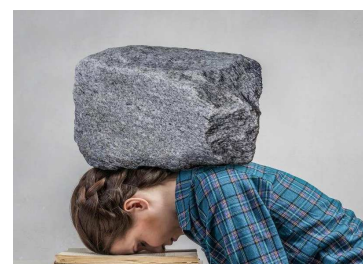
Np. interfejs `WindowListener` ma 7 metod:

```
class MojaObsluga implements WindowListener {
    public void windowOpened(WindowEvent e) { ... }
    public void windowClosing(WindowEvent e) { ... } ← patrz dalej
    public void windowClosed(WindowEvent e) { ... }
    public void windowIconified(WindowEvent e) { ... }
    public void windowDeiconified(WindowEvent e) { ... }
    public void windowActivated(WindowEvent e) { ... }
    public void windowDeactivated(WindowEvent e) { ... }
}
```

Trzeba zaimplementować je *wszystkie*.

Nie można użyć lambdy, bo ona działa tylko dla interfejsów z *jedną* metodą.

OBŁĘD, nie?



Operacje okienne

Próba zamknięcia ramki `ramka` (krzyżykiem)

Klasa obsługująca:

```
class Zamykacz implements WindowListener {
    private JFrame ramka;
    public Zamykacz(JFrame rm) { ramka = rm; } ← konstruktor
    public void windowClosing(WindowEvent e) {
        System.out.println("zamykam"); ← reakcja na zamknięcie krzyżykiem
    }
    public void windowOpened(WindowEvent e) { }
    public void windowClosed(WindowEvent e) { }
    public void windowIconified(WindowEvent e) { }
    public void windowDeiconified(WindowEvent e) { }
    public void windowActivated(WindowEvent e) { }
    public void windowDeactivated(WindowEvent e) { }
}
```

← zbędny balast

iis.ans-elblag.pl/~stefan/Dydaktyka/ProgObiII/Prog10/P1.Zamykanie.java

Klasy adaptacyjne do nasłuchiaczy

Interfejsom nasłuchiaczy towarzyszą klasy **adaptacyjne**, zawierające **puste** definicje metod interfejsu:

Interface do obsługi zdarzeń okiennych

```
interface WindowListener {
    void windowOpened(WindowEvent e);
    void windowClosing(WindowEvent e);
    void windowClosed(WindowEvent e);
    void windowIconified(WindowEvent e);
    void windowDeiconified(WindowEvent e);
    void windowActivated(WindowEvent e);
    void windowDeactivated(WindowEvent e);
}
```

Adapter do tego interfejsu

```
class WindowAdapter implements WindowListener {
    public void windowOpened(WindowEvent e) { }
    public void windowClosing(WindowEvent e) { }
    public void windowClosed(WindowEvent e) { }
    public void windowIconified(WindowEvent e) { }
    public void windowDeiconified(WindowEvent e) { }
    public void windowActivated(WindowEvent e) { }
    public void windowDeactivated(WindowEvent e) { }
}
```

Klasy adaptacyjne do nasłuchiaczy

Zamiast implementowania interfejsu w ten sposób:

```
class Zamykacz implements WindowListener {
    public void windowClosing(WindowEvent e) { TREŚĆ }
    public void windowOpened(WindowEvent e) { }
    public void windowClosed(WindowEvent e) { }
    public void windowIconified(WindowEvent e) { }
    public void windowDeiconified(WindowEvent e) { }
    public void windowActivated(WindowEvent e) { }
    public void windowDeactivated(WindowEvent e) { }
}
```

wystarczy rozszerzenie jego adaptera:

```
class Zamykacz extends WindowAdapter {
    public void windowClosing(WindowEvent e) { TREŚĆ }
}
```

iis.ans-elblag.pl/~stefan/Dydaktyka/ProgObiII/Prog10/P2.Zamykanie.java

Klasy adaptacyjne do nasłuchiaczy

Zamiast implementowania interfejsu w ten sposób:

```
class Zamykacz implements WindowListener {
    public void windowClosing(WindowEvent e) { TREŚĆ }
    public void windowOpened(WindowEvent e) { }
    public void windowClosed(WindowEvent e) { }
    public void windowIconified(WindowEvent e) { }
    public void windowDeiconified(WindowEvent e) { }
    public void windowActivated(WindowEvent e) { }
    public void windowDeactivated(WindowEvent e) { }
}
```

nie trzeba definiować nowej klasy,

wystarczy klasa anonimowa w programie głównym:

```
ramka.addWindowListener (new WindowAdapter() {
    public void windowClosing(WindowEvent e) { TREŚĆ }
} );
```

iis.ans-elblag.pl/~stefan/Dydaktyka/ProgObiII/Prog10/P3.Zamykanie.java

Zdarzenia klawiaturowe

`KeyEvent` — klasa zdarzeń klawiaturowych

Zdarzenia klawiaturowe posiadają identyfikujące je **kody**; jeśli `ev` jest w klasie `KeyEvent` to można pytać np.:

```
ev.getKeyCode() == KeyEvent.VK_CONTROL
```

— prawda, jeśli `ev` jest zdarzeniem związanym z klawiszem `Ctrl`

```
ev.getKeyCode() == KeyEvent.VK_ALT
```

— prawda, jeśli `ev` jest zdarzeniem związanym z klawiszem `Alt`

Ze zdarzeń związanych z klawiszem znaku można bezpośrednio wyciągnąć ten znak:

```
ev.getKeyChar()
```

— znak naciśnięty na klawiaturze; np. `'s'`, jeśli naciśnięto znak `S`

```
int getKeyCode()
char getKeyChar()
```

} — metody klasy `KeyEvent`

Nasłuchiwanie klawiatury

Interface `KeyListener` — nasłuchiwanie zdarzeń klawiaturowych

```
void keyPressed(KeyEvent ev)
```

— został *naciśnięty* jakiś klawisz na klawiaturze;

```
void keyReleased(KeyEvent ev)
```

— został *zwolniony* jakiś klawisz na klawiaturze;

```
void keyTyped(KeyEvent ev)
```

— został *naciśnięty* klawisz generujący znak *Unicode*.

Naciśnięcie/zwolnienie klawisza generuje `KeyEvent ev`.

Implementujemy którąś metodę interface'u tak, żeby sprawdziła właściwości `ev` i odpowiednio zareagowała.

```
iis.ans-elblag.pl/~stefan/Dydaktyka/Prog0biII/Prog10/P4_KoloryOkna.java
```

```
iis.ans-elblag.pl/~stefan/Dydaktyka/Prog0biII/Prog10/P5_Klawiatura.java
```

Nasłuchiwanie myszy

`MouseEvent` — klasa zdarzeń myszowatych

Jeśli `ev` jest w klasie `MouseEvent` to można pytać np.:

- `ev.getButton()`
— o który przycisk myszy chodzi;
- `ev.getX(), ev.getY()`
— współrzędne myszy w oknie (ew. w składowej okna);



Zdarzenia myszowe są obsługiwane przez dwa interfejsy:

- `MouseListener` — zdarzenia związane z
 - wprowadzeniem myszy do okna (ew. do składowej okna),
 - wyprowadzeniem myszy z okna (ew. ze składowej okna),
 - kliknięciem;
- `MouseMotionListener` — zdarzenia związane z
 - ruchem myszy wewnątrz okna (ew. składowej okna).

Nasłuchiwanie myszy

Interfejs, który trzeba zaimplementować:

```
public interface MouseListener extends EventListener {
    void mouseEntered(MouseEvent ev); ←wejście do okna
    void mouseExited(MouseEvent ev); ←wyjście z okna
    void mousePressed(MouseEvent ev); ←wciśnięcie przycisku
    void mouseReleased(MouseEvent ev); ←zwolnienie przycisku
    void mouseClicked(MouseEvent ev); ←pełne kliknięcie
}
```

Na przykład tak zdefiniowana metoda

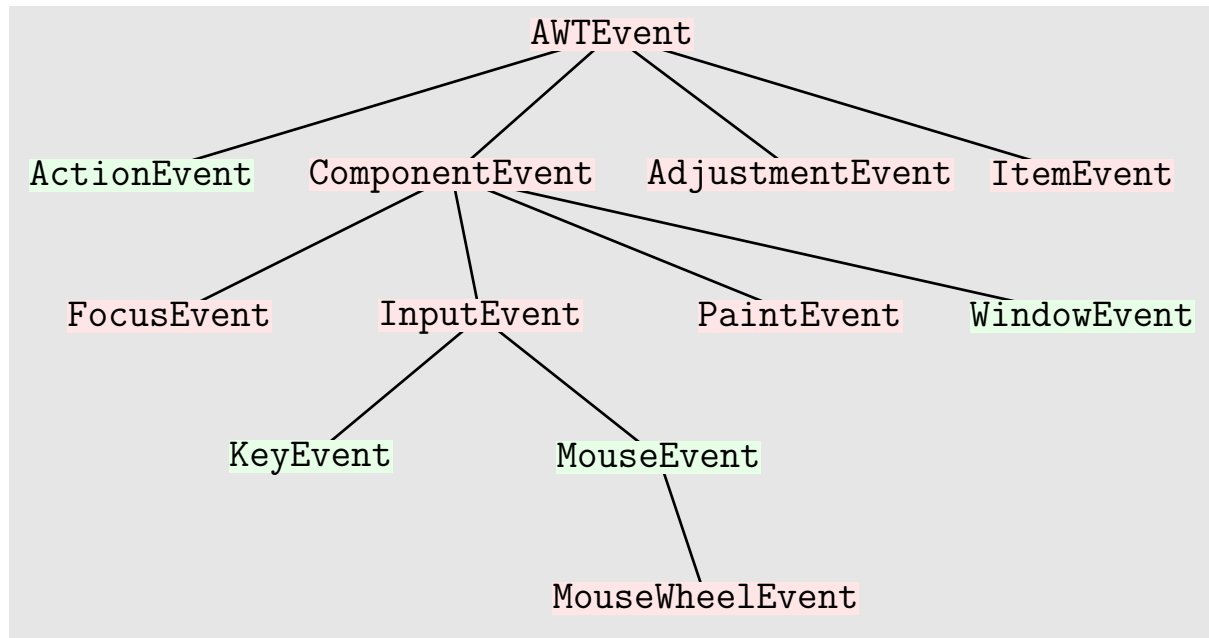
```
public void mousePressed(MouseEvent ev) {
    System.out.print("wcisnieto przycisk " + ev.getButton());
    System.out.println(" w punkcie (" + ev.getX() + "," + ev.getY() + ")");
}
```

po kliknięciu prawym przyciskiem na lewy górny róg okna wydrukuje
wcisnieto przycisk 3 w punkcie (0,0)

Drzewo rodzajów zdarzeń

Pakiet `java.awt.event.*`:

omówione
nieomówione



Przyciski w oknie — rozmieścić i połączyć z akcjami



```
JButton ziel = new JButton("zielony");
```

— utworzyć przycisk z etykietką „zielony”

```
ziel.addActionListener(new Akcja(panel, Color.GREEN));
```

— z przyciskiem związać akcję zmiany tła na zielone

```
panel.add(ziel);
```

— dodać przycisk do panelu

akcję trzeba
zdefiniować

Przyciski w oknie — rozmieścić i połączyć z akcjami

Do klawisza z nazwą koloru przymocujemy taki nasłuchiвач:

```
class Akcja implements ActionListener {
    private JPanel panel;
    private Color cl;

    public Akcja(JPanel pn, Color c) { panel = pn; cl = c; }

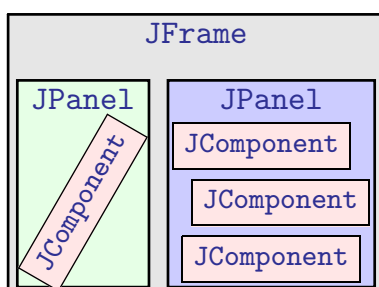
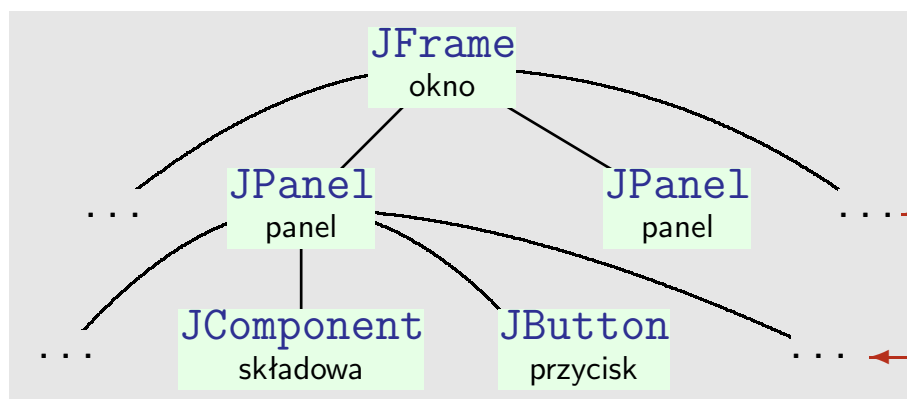
    public void actionPerformed(ActionEvent e) {
        panel.setBackground(cl);
    }
}
```

przemalowanie tła

KONSTRUKTOR:
ustalenie panelu i koloru

iis.ans-elblag.pl/~stefan/Dydaktyka/Prog0biII/Prog10/P7_KolPrzyciski.java

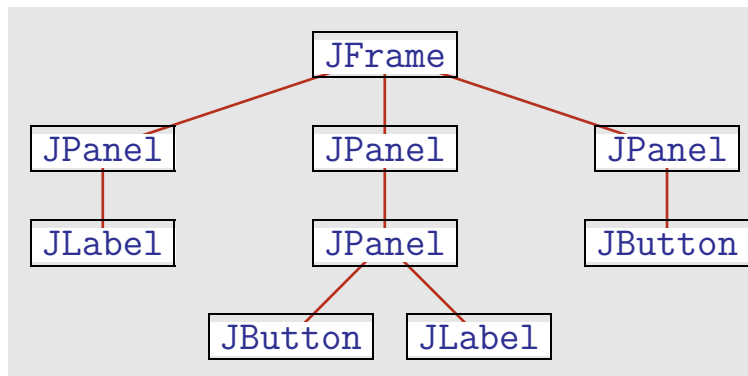
Organizacja okna



Składowa — przycisk, rysunek, itp.
Nie zaleca się umieszczania składowych bezpośrednio w oknie

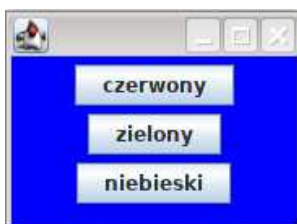
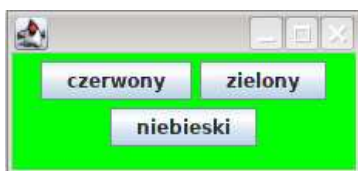
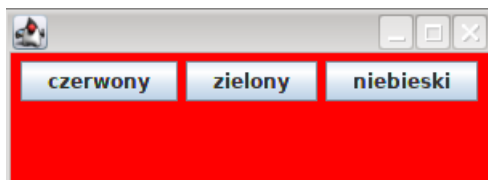
Panel — (ang.) płytki, wstawka
Składowe umieszczać w panelach,
a panele w oknach

Przykładowa struktura okna



- `JFrame` i `JPanel` to t.zw. *kontenery* — można do nich wkładać różne rzeczy
- do `JPanel` można wkładać również mniejsze `JPanel`e
- do kontenerów można wkładać na przykład `JButton` (przycisk) i `JLabel` (napis) oraz jeszcze inne

Organizacja okna



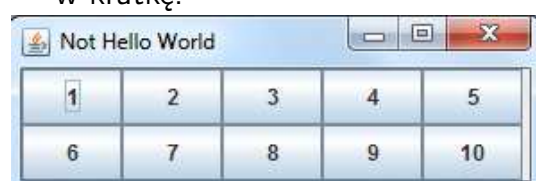
Rozmieszczenie przycisków w panelu:

- wierszami,
- centralnie.

Zmiana rozmiarów okna na nowo rozmieszcza przyciski.

Na to zachowanie można wpłynąć:

- `panel.setLayout(new FlowLayout());`
— domyślny
- `panel.setLayout(null);` — geometria podana przy każdym przycisku
- `panel.setLayout(new GridLayout());`
— w kratkę:



Organizacja okna

Rozmieszczenie brzegowe

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

```
okno.add(pn, BorderLayout.NORTH);
okno.add(za, BorderLayout.WEST);
okno.add(pd, BorderLayout.SOUTH);
okno.add(ws, BorderLayout.EAST);
okno.add(sr, BorderLayout.CENTER);
```



Rozmieszczenie pudełkowe

```
Box jeden = Box.createVerticalBox(),
    drugi = Box.createVerticalBox();
```

```
jeden.add(klaw0); jeden.add(klaw1);
jeden.add(klaw2); jeden.add(klaw3);
```

```
drugi.add(klaw4); drugi.add(klaw5);
drugi.add(klaw6);
```

```
JFrame okno = new JFrame();
okno.add(jeden); okno.add(drugi);
```