

Elementy programowania GUI

© Krzysztof Barteczko, PJWSTK 2009-2017

Po co i co?

Aby móc łatwo tworzyć interfejsy graficzne, choćby dla swoich skryptów.

Na platformie Javy programowanie GUI jest nieco "rozmyte" i niepewne.

Mamy utrwalone, dojrzałe, ale nierozwijane narzędzie – Swing oraz promowaną jako bardziej nowoczesna – JavęFX.

JavaFX nie jest jeszcze całkowicie dojrzałym produktem (2017) i nie cieszy się zbyt dużą popularnością. Swing wydaje się łatwiejszy i bardziej nadający się do w/w prostych zastosowań dla skryptów ogólnego przeznaczenia.

Groovy dostarcza tzw. builderów dla różnych środowisk programowania GUI (dla JavaFX też).

Tutaj parę słów będzie poświęcone SwingBuilderowi.

Ogólne zasady

Standardowe pakiety `java.awt` (AWT) oraz `javax.swing` (Swing) zawierają klasy definiujące wiele różnorodnych komponentów wizualnej interakcji programu z użytkownikiem (okna, przyciski, listy, menu, tablice itp.). Są gotowe do wykorzystania w naszych programach.

Groovy dostarcza `SwingBuilder` do tworzenia GUI w sposób deklaratywny.

Komponenty mają właściwości

Właściwości

mogą być ustalane przy użyciu `SwingBuilder`
ustalane i pobierane dynamicznie (`get...` , `set...`)

Kontener - komponent mogący zawierać inne komponenty

hierarchia zawierania się - w deklaracjach `SwingBuilder`,
może być modyfikowana metodami `add(...)`, `remove(...)`

Rozkład określa rozmiar i położenie komponentów w kontenerze

Okna (kontenery najwyższego poziomu)- zapewniają interakcję z użytkownikiem

Interakcja odbywa się za pomocą obsługi zdarzeń.

Trochę historii: AWT i Swing

AWT (Abstract Windowing Toolkit) – obecny w Javie od samego początku zestaw prostych komponentów wizualnej interakcji.

Problemy AWT:

- * ubogie możliwości graficzne i interakcyjne komponentów,
- * brak bardziej złożonych komponentów (np. tabel),
- * zależny od platformy systemowej wygląd komponentów

Rozwiązaniem był projekt Swing

Pakiet Swing (javax.swing i podpakiety) zawiera nowsze i wzbogacone w możliwości komponenty.

Wszystkie komponenty Swingu oprócz kontenerów kontenerów najwyższego poziomu są komponentami lekkimi. W przeciwieństwie komponenty AWT są komponentami ciężkimi.

Komponenty ciężkie są realizowane poprzez użycie graficznych bibliotek GUI systemu operacyjnego.

Komponenty lekkie są rysowane za pomocą kodu Javy

Nazwy komponentów w SwingBuilder

Windows

- * dialog
- * frame
- * window

Embeddable Windows

- * optionPane
- * fileChooser
- * colorChooser

Containers

- * box
- * desktopPane
- * internalFrame
- * panel
- * scrollPane
- * splitPane
- * tabbedPane
- * toolBar

Borders

- * emptyBorder
- * etchedBorder
- * lineBorder
- * loweredBevelBorder
- * matteBorder
- * raisedBevelBorder
- * titledBorder

Layouts

- * BorderLayout
- * boxLayout
- * cardLayout
- * flowLayout
- * gridLayout

Menus

- * menuBar
- * popupMenu
- * menu
- * menuItem
- * checkBoxMenuItem
- * radioButtonMenuItem

Widgets

- * button
- * checkBox
- * comboBox
- * editorPane
- * label
- * list
- * passwordField
- * progressBar
- * radioButton
- * scrollbar
- * separator
- * slider
- * spinner
- * table
- * textArea
- * textPane
- * textField
- * toggleButton
- * tree

Swing **SwingBuilder**

JComp = comp

np. JButton = button

Składnia

```
swing = new SwingBuilder()
swing.edt { ←
    frame(properties_map) {
        container1(properties_map) {
            component11(properties_map)
            component12(properties_map)
        }
        container2(properties_map) {
            component21(properties_map)
            component22(properties_map)
        }
    }
}
```

Tworzenie GUI w EDT
(event dispatching thread) – dla GUI ważne jest, by wszystkie operacje na komponentach wizualnych były wykonywane w jednym wątku – EDT to naturalny wybór.

Gdzie znaleźć SwingBuilder docs?
Trzeba szukać fragmentów w Sieci!

Pierwsza przykładowa aplikacja

```
import groovy.swing.SwingBuilder
import java.awt.*
import static javax.swing.WindowConstants.*
import static javax.swing.SwingConstants.*

swing = new SwingBuilder()

swing.edt {
    lookAndFeel('nimbus')
    frame(title: 'Groovy Swing', pack: true,
        visible: true,
        defaultCloseOperation: EXIT_ON_CLOSE) {

        button( text: 'Welcome Swing!',
            icon:  ImageIcon('java_logo.png'),
            foreground: Color.BLUE,
            font:   new Font('Dialog', Font.BOLD, 24)
            verticalTextPosition: BOTTOM,
            horizontalTextPosition: CENTER

        )
    }
}
```

LookAndFeel
co to i po co?



Użycie konstrukcji języka w SwingBuilder

```
import groovy.swing.SwingBuilder
import java.awt.*;
```

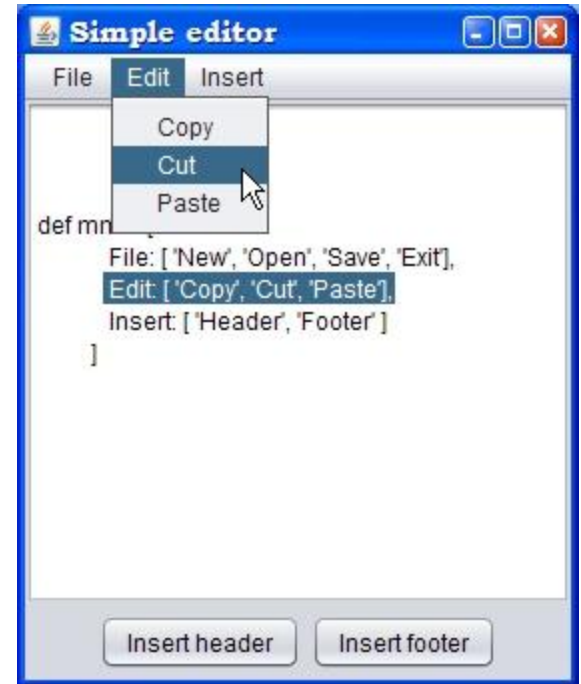
```
def mm = [
    File: [ 'New', 'Open', 'Save', 'Exit'],
    Edit: [ 'Copy', 'Cut', 'Paste'],
    Insert: [ 'Header', 'Footer' ]
]
```

```
new SwingBuilder().edt {
    lookAndFeel('nimbus')
    frame(title: 'Simple editor', pack: true,
        visible: true) {
        menuBar() {
            mm.each { key, val ->
                menu(key) {
                    val.each { menuItem(it) }
                }
            }
        }
    }
}
```

```
textArea(rows: 15, columns: 40)
```

```
def type = 'Insert'
```

```
panel(constraints: BorderLayout.SOUTH) {
    mm[type].each { button( type + ' ' + it[0].toLowerCase() + it[1..-1]) }
}
}
```



Layout constraints

Rozkłady (zarządzacy rozkładów)

Z każdym kontenerem jest skojarzony tzw. zarządca rozkładu, który określa rozmiary i położenie komponentów przy wykreślaniu kontenera.

FlowLayout - komponenty w wierszu (domyślny dla Panel)

BorderLayout - komponenty rozłożone geograficznie:
NORTH, EAST, WEST, SOUTH, CENTER (domyślny dla okien)

GridLayout - siatka komponentów

BoxLayout - wiersz lub kolumna (zalety Flow i Grid)
proste tworzenie konterów z tym rozkładem:
hbox() lub vbox()

Zewnętrzne: bogate i łatwe w użyciu MigLayout i TableLayout

Zdefiniować rozkład: podać wartość dla klucza 'layout'.

Klucz 'constraints' w mapie właściwości komponentów określa parametry używane do układania komponentów.

Prosta demonstracja rozkładów

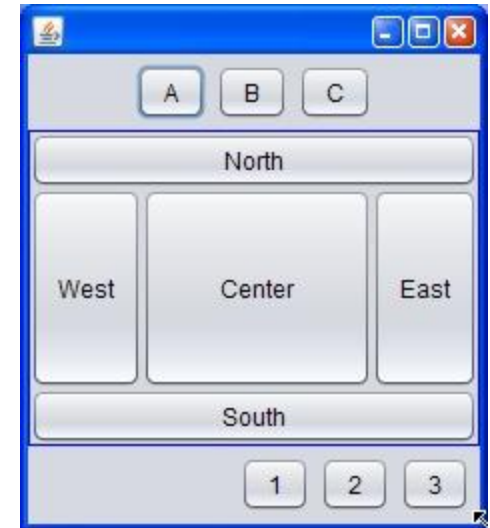
```
import groovy.swing.SwingBuilder
import static javax.swing.WindowConstants.*
import static java.awt.Color.*
import java.awt.*;

def dest = [ 'West', 'North', 'East', 'South', 'Center' ]
new SwingBuilder().edt {
    lookAndFeel('nimbus')
    frame(pack: true, visible: true,
        defaultCloseOperation: EXIT_ON_CLOSE) {
        vbox() {
            panel() { 'ABC'.each { button(it) } }

            panel(layout: new BorderLayout(),
                border: lineBorder(color:BLUE)) {
                dest.each {
                    button(it, constraints: it)
                }
            }

            panel(layout: new FlowLayout(FlowLayout.RIGHT)) {
                '123'.each { button(it) }
            }
        }
    }
}
```

Proszę uruchomić aplikację i obserwować zachowanie komponentów



Obsługa zdarzeń - ogólne zasady

Koncepcja programowania zdarzeniowego i ***callback***

Zdarzenia są obiektami odpowiednich klas, określających rodzaj zdarzeń.

Słuchacze są obiektami klas implementujących interfejsy nasłuchu. **Interfejsy nasłuchu** określają zestaw metod obsługi danego rodzaju zdarzeń.

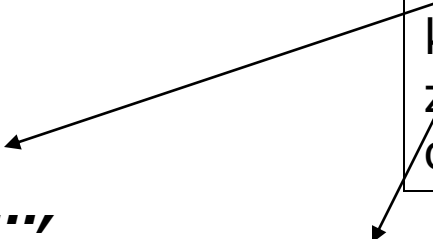
Zdarzenie (obiekt odpowiedniej klasy zdarzeniowej) jest przekazywane do obsługi obiektowi-słuchaczowi tylko wtedy gdy Słuchacz ten jest przyłączony do Źródła zdarzenia. (przyłączenie za pomocą odwołania `z.addNNNListener(h)`, gdzie: `z` – Źródło zdarzenia, `NNN` - rodzaj zdarzenia, `h` – Słuchacz danego rodzaju zdarzenia)

Przekazanie zdarzenia do obsługi polega na wywołaniu odpowiedniej dla danego zdarzenia metody obsługi zdarzenia (zdefiniowanej w klasie Słuchacza) z argumentem obiekt-zdarzenie.

Argument (obiekt klasy zdarzeniowej) zawiera informacje o okolicznościach zajścia zdarzenia. Jako parametr w metodzie obsługi może być odpytany o te informacje.

Obsługa zdarzeń w SwingBuilder

W mapie właściwości
komponentów
zdefiniuj domknięcie do
obsługi zdarzenia



```
component(..., ...,  
  handlerMethodName: { event ->  
    handlingCode  
  })
```

Rodzaje zdarzeń i metody obsługi

Zdarzenie	Metoda
ACTION_PERFORMED	actionPerformed
MOUSE_ENTERED MOUSE_EXITED MOUSE_PRESSED MOUSE_RELEASED MOUSE_CLICKED	mouseEntered mouseExited mousePressed mouseReleased mouseClicked
MOUSE_MOVED MOUSE_DRAGGED	mouseMoved mouseDragged
KEY_PRESSED KEY_RELEASED KEY_TYPED	keyPressed keyReleased keyTyped
FOCUS_GAINED FOCUS_LOST	focusGained focusLost

**Więcej
w dok.
JDK**

Przykład obsługi zdarzeń

```
import groovy.swing.SwingBuilder
import java.awt.*;

def btxt = "<html><center><b>Click me<br>" +
    "and I'll say 'Hello'</b><center></html>"

new SwingBuilder().edt {
    lookAndFeel('nimbus')
    f = frame(title: 'Events', pack: true, visible: true) {
        button(btxt,
            actionPerformed: { e->
                f.title = 'Hello'
                e.source.background = Color.YELLOW
                e.source.text = 'Done!'
            })
    }
}
```



Proszę zaobserwować:
uzyskiwanie info o źródle zdarzenia,
dostęp do innych komponentów

Akcje

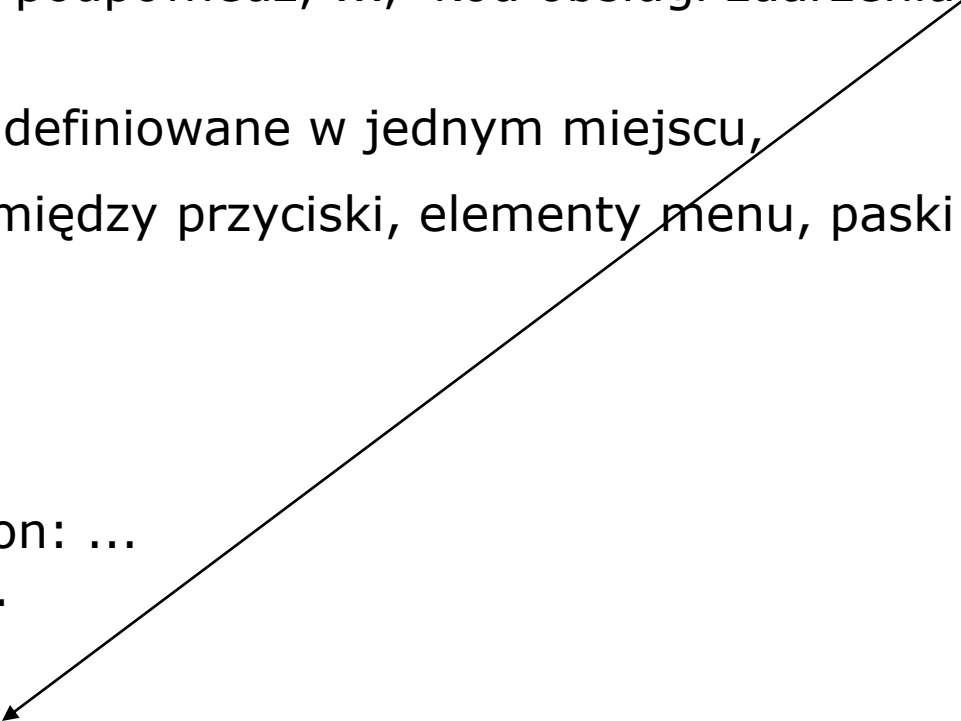
Obiekty typu Action:

określają tekst, ikonę, podpowiedź, ..., kod obsługi zdarzenia ACTION_PERFORMED

mogą (i powinny) być definiowane w jednym miejscu,

mogą być dzielone pomiędzy przyciski, elementy menu, paski narzędzi

```
action(  
    name: ...  
    icon: ...  
    shortDescription: ...  
    accelerator: ...  
    mnemonic: ...  
    ...  
    closure: { e-> ... }  
)
```



Przykład

```
def mm = [ File: [ 'New', 'Open', 'Save', 'Exit'], Edit: [ 'Copy', 'Cut', 'Paste'],
          Insert: [ 'Header', 'Footer' ] ]

new SwingBuilder().edt {
  lookAndFeel('nimbus')
  acts = actions() {
    action(id: 'ha', name: 'Header', closure: { e.insert 'Dir Sirs,\n', 0 })
    action(id: 'fa', name: 'Footer', closure: { e.append 'Sincerely yours, ' })
  }
  frame(title: 'Simple editor', pack: true, visible: true) {
    menuBar() {
      mm.each { key, val ->
        menu(key) {
          if (key == 'Insert')
            val.each { menuItem(it, action: it == 'Header' ? ha : fa) }
          else val.each { menuItem(it, actionPerformed: { e-> handler(e) } ) }
        }
      }
    }
    scrollPane() { textArea(id: 'e', rows: 15, columns: 40) }
    def type = 'Insert'
    panel(constraints: BorderLayout.SOUTH) {
      mm[type].each { mitxt ->
        what = mitxt[0].toLowerCase() + mitxt[1..-1]
        a = acts.find { it.getValue(Action.NAME) == mitxt }
        button( type + ' ' + what, action: a)
      }
    }
  }
}
```

Dostęp do komponentów przez id

```
def handler(e) {
  txt=e.source.text
  switch(txt) {
    case 'New' : ...
    case ... : ...
  }
}
```

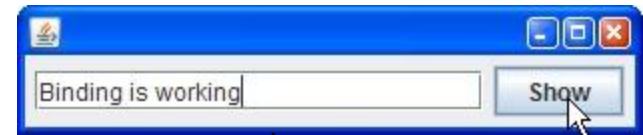

Wiązania (binding) w SwingBuilder

```
import groovy.beans.*;
import groovy.swing.SwingBuilder;

class TextModel {
    @Bindable cont
}

def tm = new TextModel()

new SwingBuilder().edt {
    frame(pack: true, visible: true) {
        panel() {
            textField id: 'tf', columns: 20
            bean tm, cont: bind{ tf.text }
            button 'Show', actionPerformed: {
                println tm.cont
            }
        }
    }
}
```



Wynik na konsoli:

Binding is working

Aby umieć więcej ...

dokumentacja javax.swing i SwingBuilder,
poznanie zaawansowanych rozkładów (np. MigLayout),
poznanie wzorca MVC i jego użycia dla list, tabel, komponentów tekstowych,
poznanie sposobów zmiany look & feel,
poznanie architektury okien,
zapoznanie się z grafiką: graphics 2D i Groovy GraphicsBuilder
Griffon (środowisko programowania GUI na platformie Groovy z różnymi pluginami i builderami)