Programowanie w Ruby

Wykład 9

Marcin Młotkowski

10 grudnia 2018

Plan wykładu

Środowisko okienkowe Tcl/Tk

- Tcl: Tool Command Language
- Tk: Tk Toolkit, przenośna biblioteka graficzna

Przykład

Kalkulator graficzny

- Okienko do wprowadzania danych
- Przycisk <Oblicz>
- Miejsce na wynik
- Przycisk <Koniec>

Implementacja

```
require 'tk'

class Okienko
    def run
        @win = TkRoot.new { title 'Kalkulator' }
```

Implementacja

```
require 'tk'

class Okienko
    def run
        @win = TkRoot.new { title 'Kalkulator' }
```

```
        @entry = TkEntry.new(@win) { pack }
        @butt = TkButton.new(@win) { text 'Oblicz'; pack }
        @butt.command { self.oblicz }
        @label = TkLabel.new(@win) { text '0'; pack }
        TkButton.new(@win) { text 'KONIEC';
                             command { exit }; pack }
```

Implementacja

```
require 'tk'

class Okienko
    def run
        @win = TkRoot.new { title 'Kalkulator' }

        @entry = TkEntry.new(@win) { pack }
        @butt = TkButton.new(@win) { text 'Oblicz'; pack }
        @butt.command { self.oblicz }
        @label = TkLabel.new(@win) { text '0'; pack }
        TkButton.new(@win) { text 'KONIEC';
                             command { exit }; pack }

        Tk.mainloop
```

Implementacja obsługi zdarzeń

butt.command { oblicz }

```
def oblicz
    @label.text = eval(@entry.value)
    @entry.value = ''
end
```

Wynik

Opis obrazka

Układanie kontrolek

Model grid

- Okno jest dzielone na komórki ułożone w wiersze i kolumny
- Kontrolki są wstawiane w komórki
- kontrolki mogą zajmowć więcej niż jedną komórkę

Układanie kontrolek w komórkach siatki

```
@entry = TkEntry.new(@win)
          { grid(:row => 1, :column => 1) }
@butt = TkButton.new(@root)
          { grid(:row => 1, :column => 2) }
@label = TkLabel.new(@root)
          { grid('row' => 1, 'column' => 3) }
TkButton.new(@root) { grid(:row => 2, :column => 2) }
```

Rezultat

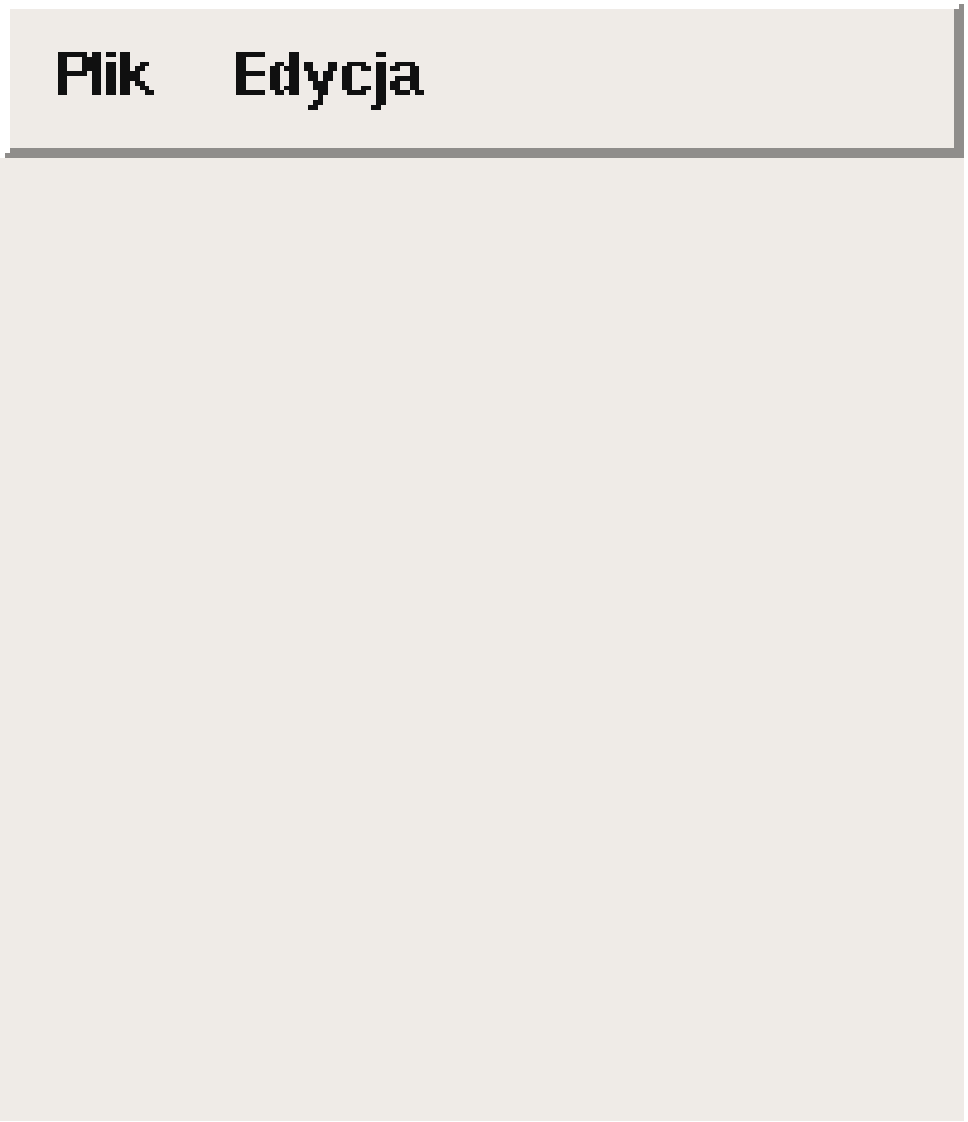
Pierwsze menu

```
menu = TkMenu.new()
one = TkMenu.new(menu)
menu.add('cascade', :menu => one, :label => 'Plik')
one.add('command', :label => 'Zapisz',
            :command => proc{ puts 'Zapisane'})
    one.add('command', :label => 'KONIEC',
            :command => proc{ root.destroy })
```

Drugie menu

```
two = TkMenu.new(menu)
menu.add('cascade', :menu => two, :label => 'Edycja')
two.add('command', :label => ''Skopiuj", :command => proc{ })
root.menu(menu)
```

Wynik



Kontrolka TkCanvas

```
cv = TkCanvas.new(root)
```

Kontrolka TkCanvas

```
cv = TkCanvas.new(root)
```

Ustalenie rozmiarów

```
cv.place(:height => 200, :width => 200)
```

Rysowanie na płótnie

Na płótnie umieszcza się elementy Tkc*, na przykład TkcArc, TkcLine czy TkcBitmap.

Przykład

```
cv.create(TkcOval, 50, 50, 150, 150, 'width' => 1)
cv.create(TkcLine, 100, 80, 100, 110)
cv.create(TkcArc, 80, 110, 120, 130,
          :start => 180, :extent => 180)
cv.create(TkcOval, 70, 70, 90, 100, :fill => 'black')
cv.create(TkcOval, 110, 70, 130, 100, :fill => 'black')
TkcText.new(cv, 100, 170, :font => 'Arial 12 bold',
          :text => 'Hello, world!', :anchor => 'center')
```

Wynik

Wynik

Pliki graficzne

Kontrolki wyświetlające plik

```
img = TkPhotoImage.new(:file => 'plik.gif')
```

Uwagi

- Obrazków nie można umieszczać jako samodzielnych kontrolek
- Obrazy mogą być elementem przycisków lub etykiet

Uwagi

- Obrazków nie można umieszczać jako samodzielnych kontrolek
- Obrazy mogą być elementem przycisków lub etykiet

Przykłady

```
TkLabel.new(win) { image img; pack }

TkButton.new(win) { image img; pack }

TkcImage.new(cv, 100, 100, 'image' => img)
```

Plan wykładu

Wątki

- Zadania permanentnie wykonywane podczas działania programu
- Zadania wykonywane na żądanie w niezależnym wątku

Przykład

Zegar

```
@clock = TkLabel.new(@win) { text '  '; pack }
@timer = TkAfter.new(1, -1,
         proc{ @clock.text = Time.now.to_s })
@timer.start
```

Zadania w wątkach

```
menu.add('command',
        :label => 'Wykonaj',
        :command => proc { Thread.new { self.run } })
```

Plan wykładu

Standardowe argumenty

```
def foo(arg1, arg2, arg3)
    ...
end
```

Standardowe argumenty

```
def foo(arg1, arg2, arg3)
    ...
end
foo("jeden", 2, trzy)
```

Argumenty opcjonalne

```
def foo(arg1, arg2, arg3=3)
    ...
end
```

Argumenty opcjonalne

```
def foo(arg1, arg2, arg3=3)
    ...
end
foo("jeden", 2, trzy)
foo("jeden", 2)
```

Argumenty *keyword*

```
def write(file:, data:, mode: 'ascii')
    puts mode
end
```

Argumenty *keyword*

```
def write(file:, data:, mode: 'ascii')
    puts mode
end
write(data: 123, file: "test.txt")
```

Zmienna liczba argumentów

```
def foo(*args)
    args.each { | a | puts s }
end
```

Zmienna liczba argumentów

```
def foo(*args)
    args.each { | a | puts s }
end
foo("jeden", 2, trzy)
foo(2)
```

Słownik argumentów

```
def foo(**args)
    ...
end
```

Słownik argumentów

```
def foo(**args)
    ...
end
foo(pierwszy: 1, drugi: "dwa", trzeci: 'trzy')
```

Kolejność argumentów

wymagane
opcjonalne
ze zmienną liczbą argumentów
keyword

Plan wykładu

Zasady

Składowe projektu

- przynajmniej trzy modele;
- przynajmniej jedna relacja jeden–do–wielu;
- udokumentowana zgodnie z jakimś systemem generowania dokumentacji;
- testy;
- uwierzytelnienie użytkowników;
- przygotowana do dystrybucji w formie GEM'ów

Termin

28 stycznia 2019