



**UNIWERSYTET TECHNOLOGICZNO-PRZYRODNICZY
IM. J. I J. ŚNIADECKICH W BYDGOSZCZY**

**WYDZIAŁ TELEKOMUNIKACJI, INFORMATYKI I ELEKTROTECHNIKI
ZAKŁAD TECHNIKI CYFROWEJ**

PROGRAMOWANIE OBIEKTOWE

LAB 1 - KALKULATOR

AUTOR:

MATEUSZ BIRKHOLZ

DATA WYKONANIA

30.11.2020

DATA ODDANIA

30.11.2020

KIERUNEK:

INFORMATYKA STOSOWANA

GRUPA 1

SEMESTR 3

ROK AKADEMICKI 2019/2020

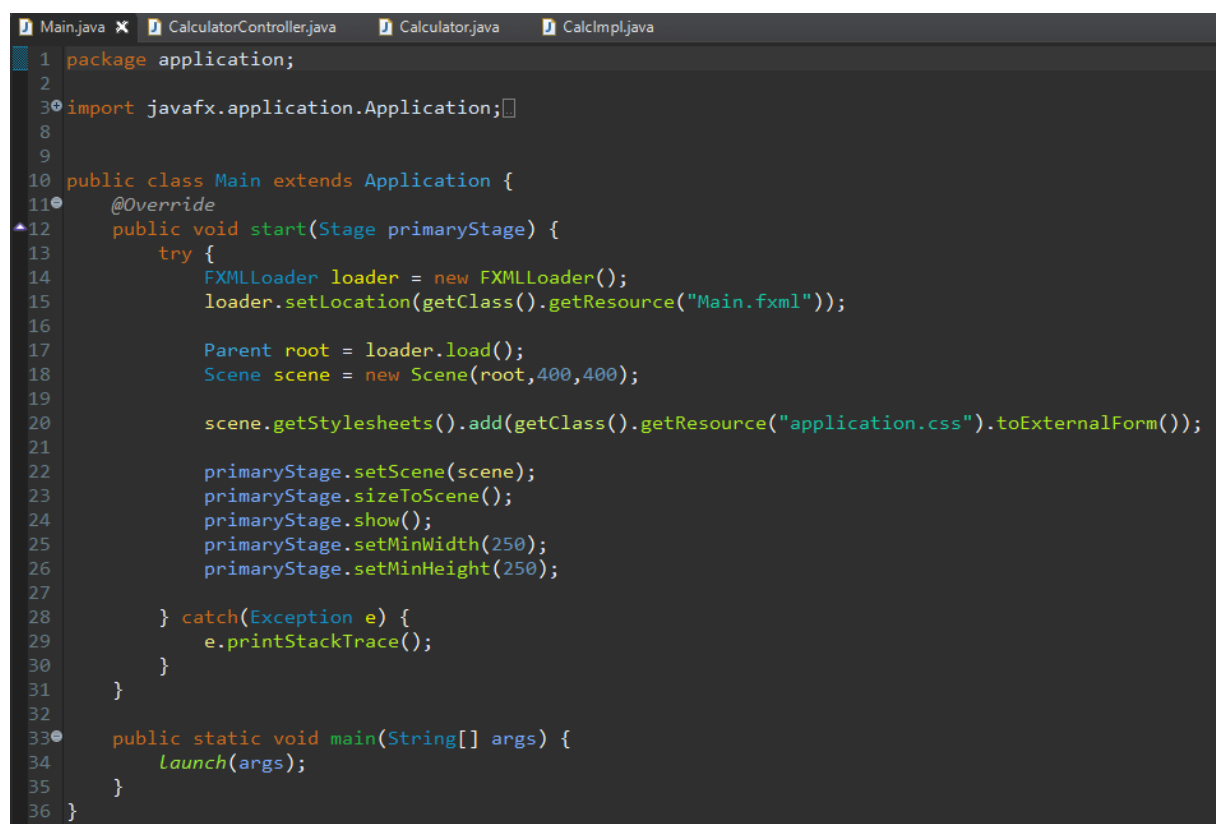
TRYB STUDIÓW: STACJONARNE

Cel:

Celem zadania była nauka wykorzystania programowania zorientowanego obiektowo do tworzenia aplikacji okienkowych z wykorzystaniem JavyFX oraz SceneBuildera. Zadanie polegało na przygotowaniu interfejsu użytkownika kalkulatora oraz utworzeniu klasy odpowiedzialnej za jego obsługę. Dodatkowo klasę odpowiedzialną za obsługę kalkulatora należało ukryć za interfejsem.

Przebieg ćwiczenia:

Najpierw wygenerowałem plik main w którym zapisałem instrukcje konfiguracji oraz wyświetlenia okna programu. Początkowe wymiary okna ustawiam na 400x400 oraz ustawiam jego najmniejsze rozmiary na 250x250. Importuję tutaj również plik Main.fxml który zawiera ułożenie elementów interfejsu oraz plik application.css który zawiera arkusz stylu interfejsu. W razie problemów wyświetlam błąd.



```
1 package application;
2
3 import javafx.application.Application;
4
5
6
7
8
9
10 public class Main extends Application {
11     @Override
12     public void start(Stage primaryStage) {
13         try {
14             FXMLLoader loader = new FXMLLoader();
15             loader.setLocation(getClass().getResource("Main.fxml"));
16
17             Parent root = loader.load();
18             Scene scene = new Scene(root, 400, 400);
19
20             scene.getStylesheets().add(getClass().getResource("application.css").toExternalForm());
21
22             primaryStage.setScene(scene);
23             primaryStage.sizeToScene();
24             primaryStage.show();
25             primaryStage.setMinWidth(250);
26             primaryStage.setMinHeight(250);
27
28         } catch (Exception e) {
29             e.printStackTrace();
30         }
31     }
32
33     public static void main(String[] args) {
34         launch(args);
35     }
36 }
```

W pliku CalculatorController definiuje wszystkie przyciski jakie będą mi potrzebne do obsługi kalkulatora:

- btnC – czyszczenie kalkulatora
- btnS – zmiana znaku
- btnAdd – dodawanie
- btnDif – odejmowanie
- btnMul – mnożenie
- btnDiv – dzielenie
- btnSqr – pierwiastek
- btnFra – ułamek
- itd

Definiuję również pole tekstowe, zmienną kontrolną kropki oraz obiekt kalkulatora. Aby wszystko działało importuję `ActionEvent` z biblioteki `JavyFX` i przy jej obiektach pisze `@FXML`.

```
import javafx.event.ActionEvent;

public class CalculatorController{
    @FXML
    private Button btnC, btnS, btnAdd, btnDif, btnMul, btnDiv, btnSqr, btnFra, btnLog, btnPer, btnEq, Btn;
    @FXML
    private TextField txtFld;

    private boolean Dot; // Czy w polu jest kropka

    private Calculator calculator;
```

Tworzę dwa konstruktory różniące się przyjmowanymi parametrami. Daje mi to możliwość wykorzystania istniejącego obiektu kalkulatora lub utworzenie nowego.

```
public CalculatorController() {
    //Tutaj nigdy nie edytować zmiennych
    calculator = new CalcImpl();
}

public CalculatorController(CalcImpl calculator) {
    this.calculator = calculator;
}
```

Następnie funkcja inicjalizująca który ustawia kropkę na false.

```
@FXML
public void initialize() {
    Dot = false; //Kropki nie ma poniewaz pole puste
}
```

Do pomocy przygotowałem również funkcję czyszczenie pola dzięki której nigdy nie zapomnę przy czyszczeniu zmienić wartości kropki.

```
public void clearField() //Czyszczenie pola i resetowanie kropki
{
    txtFld.setText("");
    Dot = false;
}
```

Funkcja odpowiedzialna za obsługę przycisku kropki sprawdza wartość kontrolki u w zależności od niej dodaje do pola albo '.' albo '0'.

```
@FXML
public void onActionBtndot(ActionEvent event) { //Kontrola kropki
    if(!Dot)
    {
        if(txtFld.getText().equals(""))
        {
            txtFld.appendText("0.");
            Dot = true;
        }
        else
        {
            txtFld.appendText(".");
            Dot = true;
        }
    }
}
```

Następna jest funkcja obsługująca klawisze przycisków. Pobiera ona wartość przycisku i dodaje go do pola tekstowego. Jeśli wartość pola jest wynikiem działania to przed wprowadzeniem wartości pole jest czyszczone. Parametr ten sprawdzany jest za pomocą zmiennej obiektu kalkulator eq.

```
@FXML
public void onActionBtn(ActionEvent event) //Przyciski cyfr
{
    if(calculator.getEq())
    {
        clearField();
        calculator.setEq(false);
    }
    Button button = (Button)event.getSource();
    txtFld.appendText(button.getText());
}
```

Przycisk od czyszczenia kalkulator czyści ekran oraz resetuje wartości zmiennych obiektu kalkulatora.

```
@FXML
public void onActionBtnC(ActionEvent event) { //Kasowanie
    clearField();
    calculator.setVar1(null);
    calculator.setVar2(null);
    calculator.setEq(false);
}
```

Przycisk zmiany znaku pobiera zawartość pola i mnoży ją przez -1.

```
@FXML
public void onActionBtnS(ActionEvent event) { //Zmiana znaku
    if(!txtFld.getText().isEmpty())
    {
        txtFld.setText(String.valueOf(Double.parseDouble(txtFld.getText())*-1));
    }
}
```

Funkcja przycisku '=' wywołuje metodę kalkulatora odpowiedzialną za wykonanie działania, podając jako parametr zawartość pola, oraz wypisuje zwróconą przez nią wartość do pola.

```
@FXML
public void onActionBtnEq(ActionEvent event) { //Rowna sie
    txtFld.setText(calculator.eqAction(txtFld.getText()));
}
```

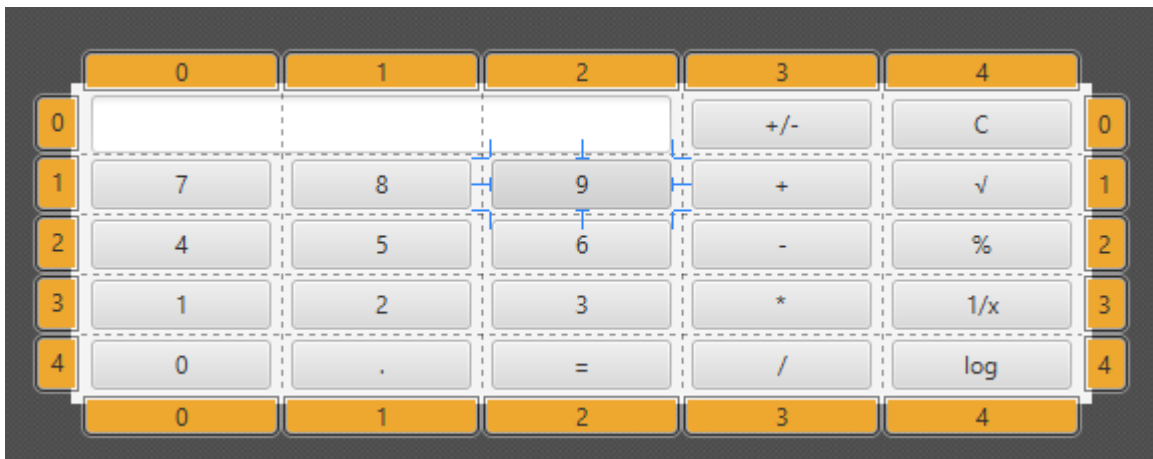
Przyciski działań na wielu zmiennych wywołują metodę odpowiedzialną za obsługę takich działań, podając jako parametr znak określający działanie, oraz wartość pola tekstowego. Więcej o niej niżej.

```
@FXML
public void onActionBtnAdd(ActionEvent event) { //Dodawanie
    txtFld.setText(calculator.doubleAction(txtFld.getText(), '+'));
}
@FXML
public void onActionBtnDif(ActionEvent event) { //Odejmowanie
    txtFld.setText(calculator.doubleAction(txtFld.getText(), '-'));
}
@FXML
public void onActionBtnDiv(ActionEvent event) { //Dzielenie
    txtFld.setText(calculator.doubleAction(txtFld.getText(), '/'));
}
@FXML
public void onActionBtnMul(ActionEvent event) { //Mnożenie
    txtFld.setText(calculator.doubleAction(txtFld.getText(), '*'));
}
@FXML
public void onActionBtnPer(ActionEvent event) { //Modulo
    txtFld.setText(calculator.doubleAction(txtFld.getText(), '%'));
}
```

Natomiast przyciski obsługujące działania na jednej wartości wywołują metodę obsługującą działanie na jednej zmiennej, podając te same parametry.

```
@FXML
public void onActionBtnSqr(ActionEvent event) { //Pierwiastek
    txtFld.setText(calculator.singleAction(txtFld.getText(),'s'));
}
@FXML
public void onActionBtnFra(ActionEvent event) { //Ułamek
    txtFld.setText(calculator.singleAction(txtFld.getText(),'f'));
}
@FXML
public void onActionBtnLog(ActionEvent event) { //Logarytm
    txtFld.setText(calculator.singleAction(txtFld.getText(),'l'));
}
```

Sam projekt interfejsu prezentuje się następująco:



Jest to siatka 5x5 umieszczona w środku okna. Każdy z elementów interfejsu ma ustawiony elastyczny rozmiar oraz ma przypisaną odpowiednią funkcję która go obsługuje. Każdy z elementów jest oddzielony od innych 10px marginesu.

Wszystkie obliczenia kalkulatora wykonywane są za pośrednictwem obiektu klasy CalcImpl która jest ukryta za interfejsem Calculator. Interfejs zawiera metody odpowiedzialne za obliczenia oraz metody odpowiedzialne za dostęp do jego odpowiednich zmiennych.

```
Main.java CalculatorController.java Calculator.java X CalcImpl.java
1 package application;
2
3 public interface Calculator {
4
5     public void setVar1(Double newVar1);
6     public void setVar2(Double newVar2);
7     public void setEq(boolean newEq);
8     public boolean getEq();
9
10    public String eqAction(String txt);
11    public String doubleAction(String txt, char cmd);
12    public String singleAction(String txt, char cmd);
13 }
```

Klasa kalkulatora implementuje interfejs. Obiekt tego interfejsu posiada 2 zmienne typu Double które przechowują wartości na których pracuje kalkulator, zmienną typu char która przechowuje znak działania oraz zmienną eq o której wspominałem wcześniej. Określa ona czy zawartość pola jest wynikiem działania.

```
public class CalcImpl implements Calculator {  
    private char action; // Polecenie  
    private Double var1;  
    private Double var2;  
    private boolean eq = false; // Czy to co wyświetlane jest wynikiem "="
```

Funkcje odpowiedzialne za dostęp do zmiennych pozwalają na zmianę ich wartości, lub jak w przypadku zmiennej eq, zwracają jej wartość.

```
public boolean getEq()  
{  
    return eq;  
}  
public void setVar1(Double newVar1)  
{  
    var1 = newVar1;  
}  
public void setVar2(Double newVar2)  
{  
    var2 = newVar2;  
}  
public void setEq(boolean newEq)  
{  
    eq = newEq;  
}
```

Metoda eqAction to metoda wywoływana przez przycisk '='. Odpowiedzialna jest obliczania wartości działań które wymagają 2 zmiennych. Na początku sprawdza czy pierwsza wartość jest już pobrana. Jeśli nie to zwraca pobraną wartość pola. Następnie sprawdza czy 2 wartość jest pusta. Jeśli tak to, o ile to możliwe, przypisuje 2 wartości pobraną wartość. Jednak jeśli 2 wartość jest już pobrana to sprawdzany jest czy pobrana wartość nie jest wartością działania. Jeśli tak to wartość 2 jest nadpisywana. Następnie rozpoczyna się instrukcja switch która na podstawie znaku działania określi jaką instrukcję wykonać, a jej wynik zapisuje do 1 zmiennej. Następnie ustawia wartość eq na true oraz zwraca string z wartością działania. W przypadku dzielenia istnieje dodatkowy warunek który sprawdza czy 2 wartość nie jest zerem.

```

public String eqAction(String txt)
{
    if(var1 != null)
    {
        if(var2 == null)
        {
            if(txt.isEmpty())
            {
                eq = true;
                return "Bład!";
            }
            else
            {
                var2 = Double.parseDouble(txt);
            }
        }
        if(!eq)
        {
            var2 = Double.parseDouble(txt);
        }
        switch(action)
        {
            case '+':
                var1 = var1 + var2;
                eq = true;
                return String.valueOf(var1);
            case '-':
                var1 = var1 - var2;
                eq = true;
                return String.valueOf(var1);
            case '*':
                var1 = var1 * var2;
                eq = true;
                return String.valueOf(var1);
            case '%':
                var1 = var1%var2;
                eq = true;
                return String.valueOf(var1);
            case '/':
                if(var2 != 0.0)
                {
                    var1 = var1 / var2;
                    eq = true;
                    return String.valueOf(var1);
                }
                else
                {
                    eq = true;
                    return "Bład! Dzielenie przez 0!";
                }
            }
        }
        return txt;
    }
}

```

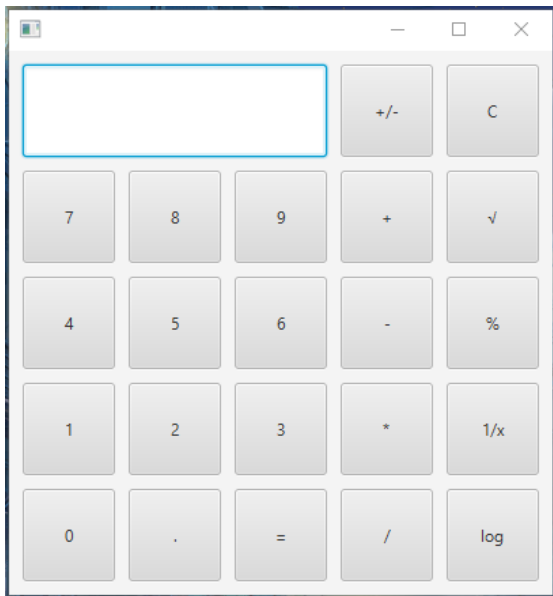
Instrukcja do obsługi działań na 2 zmiennych skonstruowana jest tak aby w przypadku pustego pola tekstowego zwrócić pustego stringa oraz ustawić pobrany znak działania. Jednak w przypadku pobranej wartości sprawdza czy 1 wartość jest równa null oraz czy nie jest wynikiem działania. W takim wypadku wywołuję funkcję eqAction która zajmuje się liczeniem, następnie ustawia znak, a na końcu zwraca wynik. Jednak w innym wypadku przypisze tylko pobraną wartość do pierwszej zmiennej, ustawi znak i zwróci pusty string.

```
public String doubleAction(String txt, char cmd) // Funkcja do obsługi 2 wartości
{
    if(txt.isEmpty())
    {
        action=cmd;
        return "";
    }
    else
    {
        if(var1!=null && !eq)
        {
            String result = eqAction(txt);
            action=cmd;
            return result;
        }
        else
        {
            var1 = Double.parseDouble(txt);
            action=cmd;
            return "";
        }
    }
}
```

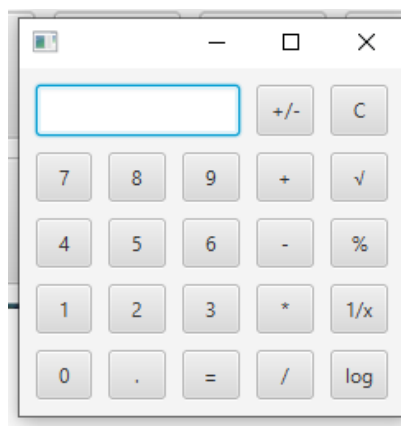
Ostatnią metodą jest metoda obsługująca działania na pojedynczych wartościach. Jej budowa jest bardzo prosta. Jeśli pobrana wartość nie jest pusta to wywoływany jest switch który wykona na niej odpowiednie działanie i zwróci jego wartość. Jeśli jednak jest pusta to zwróci pusty string.

```
public String singleAction(String txt, char cmd) //Funkcja do obsługi działań na 1 wartości
{
    if(!txt.isEmpty())
    {
        switch(cmd)
        {
            case 'l':
                return String.valueOf(Math.log(Double.parseDouble(txt)));
            case 'f':
                return String.valueOf(1/Double.parseDouble(txt));
            case 's':
                return String.valueOf(Math.sqrt(Double.parseDouble(txt)));
        }
        return "";
    }
    else
    {
        return "";
    }
}
```


Gotowy kalkulator prezentuje się następująco:



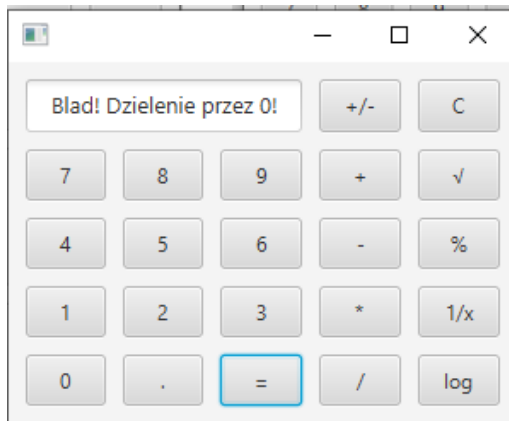
Skaluje się również poprawnie.



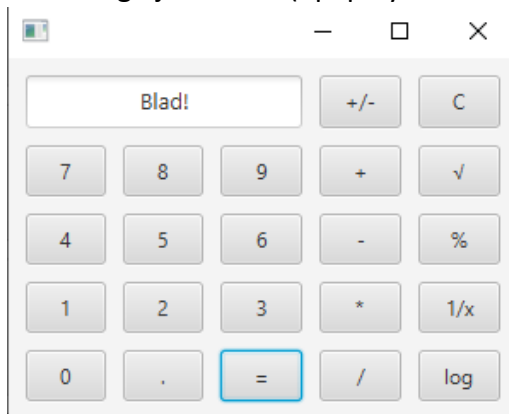
Podsumowanie i wnioski:

Zdaję sobie sprawę z tego że nie jest to najczytelniejszy kod, i że pewnie dałoby się to wszystko zapisać prościej jednak taka konstrukcja kalkulatora sprawia że jest odporny na kilka podstawowych błędów takich jak np.:

- Dzielenie przez 0



- Brak drugiej wartości (np. przy dodawaniu)



- Wcisnięcie przycisków działania bez wprowadzenia żadnej wartości (wynikiem będzie puste pole tekstowe)
- Wieloprzecinkowa wartość.
- Wprowadzanie kropki bez żadnej cyfry przed nią.

Dodatkowo wielokrotne wciskanie przycisku '=' powoduje powtórzenie poprzedniego działania wielowartościowego. Kalkulator obsługuje również następujące po sobie działania. Przykładowo wciskając kolejno $2+3/5-3=$ trzymamy kolejno wyniki 5,1,-2.

Wnioski:

- Wykorzystanie osobnej klasy na obliczenia kalkulatora pozwala na uszczuplenie kody w przypadku korzystania z wielu okienek kalkulatora na raz, ponieważ wystarczy tylko wywoływać jego metody z kontrolera zamiast pisać je w nim na nowo.
- Kalkulator teoretycznie jest prostym programem do nauki programowania obiektowego jednak jeśli chcemy zaimplementować dodatkowe funkcjonalności oraz kontrolę błędów nawet on może być wyzwaniem.
- Testerzy są bardzo przydatni w znajdowaniu błędów. Prosiłem znajomych o pomoc przy sprawdzaniu błędów i kilka z nich byłem w stanie zauważyć właśnie dzięki nim.