

# Sprawozdanie – laboratoria 2, kalkulator oraz zadanie z wielomianem

Marta Piotrowska, gr. 2

---

## Laboratorium 2a – powtórka z zajęć poprzednich

---

### Zadanie A - Hello i pętla.

#### Opis działania programu:

Program wyświetla napis „Witaj!” oraz liczby od 1 do X, gdzie X to liczba przekazana jako argument podczas uruchamiania programu.

W metodzie main sprawdzamy, czy podano argument, a jeśli nie – wyświetlamy komunikat o błędzie. Następnie konwertujemy wartość argumentu na typ int i za pomocą pętli for wypisujemy liczby od 1 do X.

#### Kod:

```
1 package pojava.lab2a.zadA;
2
3 public class Hello {
4     public static void main(String[] args) {
5         System.out.println("Witaj!");
6
7         // sprawdzamy czy podano argument
8         if (args.length == 0) {
9             System.out.println("Error: Podaj argument!");
10            return;
11        }
12
13        // wypisujemy liczby od 1 do 20
14        // int x = 20;
15
16        // zmiana argumentu x na liczbę
17        int x = Integer.parseInt(args[0]);
18
19        for (int i = 1; i <= x; i++) {
20            System.out.println(i);
21        }
22    }
23 }
```

#### Output:

```
C:\Users\marta\.jdk\openjdk-21.0.2\bin\java.exe
Witaj!
1
2
3
4
5

Process finished with exit code 0
```

---

## Zadanie B - Tablica String

### Opis działania programu:

Program tworzy tablicę na 4 ciągi znaków (String), wypełnia ją pierwszymi czterema argumentami przekazanymi przy uruchomieniu i wypisuje zawartość.

Najpierw sprawdzamy, czy użytkownik wprowadził co najmniej 4 argumenty – w przeciwnym razie zwracamy błąd. Następnie zapisujemy wartości do tablicy w pętli for i wyświetlamy jej zawartość w postaci tekstowej.

### Kod:

```
1 package pojava.lab2a.zadB;
2
3 import java.util.Arrays;
4
5 public class TablicaZeStringami {
6     @
7     public static void main(String[] args) {
8
9         if (args.length < 4) {
10             System.out.println("Error: Podaj 4 argumenty!");
11             return;
12         }
13
14         String[] array = new String[4];
15         for (int i = 0; i < 4; i++) {
16             array[i] = args[i];
17         }
18
19         System.out.println(Arrays.toString(array));
20     }
```

### Output:

```
C:\Users\marta\.jdk\openjdk-21.0.2\bin\java.exe
[ala, ma, rudego, kotka]

Process finished with exit code 0
```

---

## Zadanie C – Dziedziczenie

### Opis działania programu:

Program demonstruje mechanizm dziedziczenia, modelując samochód i jego rozszerzoną wersję – taksówkę. Klasa Auto przechowuje informacje o miesięcznym przebiegu, a Taxi dodatkowo o zarobkach. Obie klasy generują losowe wartości dla 12 miesięcy i obliczają ich średnią. W metodzie main() tworzony jest obiekt Taxi, a następnie wyświetlane są jego średni przebieg i zarobki.

### Kod klasy Auto:

Klasa Auto definiuje tablicę przebieg, w której przechowywane są losowo generowane wartości miesięcznego przebiegu. Metoda srPrzebieg() oblicza średni przebieg pojazdu na podstawie tych danych.

```
1 package pojava.lab2a.zadC;
2
3 import java.util.Random;
4
5 1 usage 1 inheritor
6 public class Auto {
7     4 usages
8     protected float[] przebieg;
9     1 usage
10    public Auto() {
11        przebieg = new float[12];
12        Random rand = new Random();
13        for (int i = 0; i < 12; i++) {
14            przebieg[i] = 500 + rand.nextFloat() * 1500; // losowy przebieg 500-2000 km
15        }
16
17    1 usage
18    public float srPrzebieg() {
19        float suma = 0;
20        for (float km : przebieg) {
21            suma += km;
22        }
23        return suma / przebieg.length;
24    }
25 }
```

---

## Kod klasy Taxi:

Klasa Taxi rozszerza Auto, dodając tablicę zarobki, która przechowuje miesięczne dochody taksówki. Metoda srZarobki() oblicza ich średnią. W main() tworzony jest obiekt Taxi, a następnie wyświetlane są wartości średniego przebiegu i zarobków.

```
1 package pojava.lab2a.zadC;
2
3 import java.util.Random;
4
5 public class Taxi extends Auto {
6     private float[] zarobki;
7
8     public Taxi() {
9         super();
10        zarobki = new float[12];
11        Random rand = new Random();
12        for (int i = 0; i < 12; i++) {
13            zarobki[i] = 3000 + rand.nextFloat() * 5000; // losowe zarobki 3000-8000 zł
14        }
15    }
16
17    public float srZarobki() {
18        float suma = 0;
19        for (float zarobek : zarobki) {
20            suma += zarobek;
21        }
22        return suma / zarobki.length;
23    }
24 }
```

```
public static void main(String[] args) {
    Taxi taxi = new Taxi();
    System.out.println("Średni przebieg: " + taxi.srPrzebieg() + " km");
    System.out.println("Średnie zarobki: " + taxi.srZarobki() + " zł");
}
```

## Output:

```
C:\Users\martam\jdk\openjdk-21.0.2\bin\java.exe
Średni przebieg: 1263.3474 km
Średnie zarobki: 5666.638 zł

Process finished with exit code 0
```

---

## Laboratorium 2b

---

### Zadanie A - Dziedziczenie po JFrame

#### Opis działania programu:

Program ilustruje dziedziczenie, rozszerzając klasę JFrame o własną implementację CloseableFrame. Jego celem jest utworzenie prostego okna graficznego o wymiarach 640x480 pikseli, które zamyka się poprawnie, zwalniając zasoby. W metodzie main() tworzony jest obiekt okna i ustawiana jego widoczność.

#### Klasa CloseableFrame:

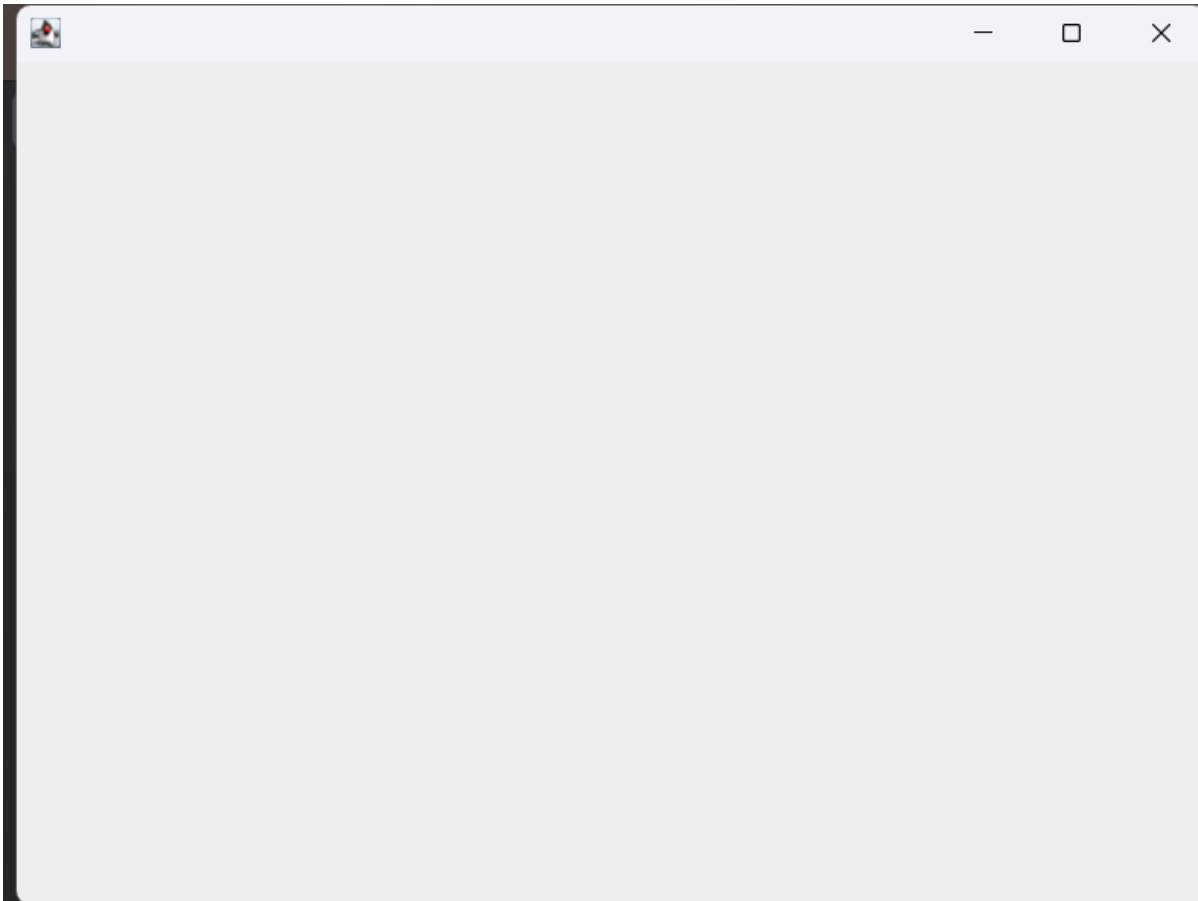
Klasa CloseableFrame rozszerza JFrame, definiując cztery konstruktory. Każdy konstruktor określa domyślny rozmiar oraz sposób zamykania (DISPOSE\_ON\_CLOSE). W main() tworzony jest obiekt klasy, który zostaje wyświetlony na ekranie.

#### Kod:

```
1 package pojava.lab2b.zadA;
2
3 import javax.swing.*;
4 import java.awt.*;
5
6 public class CloseableFrame extends JFrame {
7     2 usages
8     public CloseableFrame() throws HeadlessException {
9         this.setSize( width: 640, height: 480);
10        setDefaultCloseOperation(DISPOSE_ON_CLOSE);
11    }
12    no usages
13    public CloseableFrame(GraphicsConfiguration gc) {
14        super(gc);
15        this.setSize( width: 640, height: 480);
16        setDefaultCloseOperation(DISPOSE_ON_CLOSE);
17    }
18    no usages
19    public CloseableFrame(String title) throws HeadlessException {
20        super(title);
21        this.setSize( width: 640, height: 480);
22        setDefaultCloseOperation(DISPOSE_ON_CLOSE);
23    }
```

```
23    }
24    no usages
25    public CloseableFrame(String title, GraphicsConfiguration gc) {
26        super(title, gc);
27        this.setSize( width: 640, height: 480);
28        setDefaultCloseOperation(DISPOSE_ON_CLOSE);
29    }
30    public static void main(String[] args) {
31        CloseableFrame window = new CloseableFrame();
32        window.setVisible(true);
33    }
34 }
35 }
36 }
```

## Output:



---

## Zadanie B - Figury w losowych kolorach

### Opis działania programu:

Program tworzy okno podzielone na dwa panele – jeden z losowo kolorowanymi figurami, drugi z interaktywnymi komponentami. Klasa `ThreeShapesPanel` dziedziczy `JPanel` i rysuje trzy figury: koło, kwadrat i trójkąt w losowych kolorach. W metodzie `main()` tworzony jest obiekt `CloseableFrame`, w którym umieszczane są dwa panele – `ThreeShapesPanel` oraz klasyczny `JPanel` z przyciskami, etykietą i polem tekstowym.

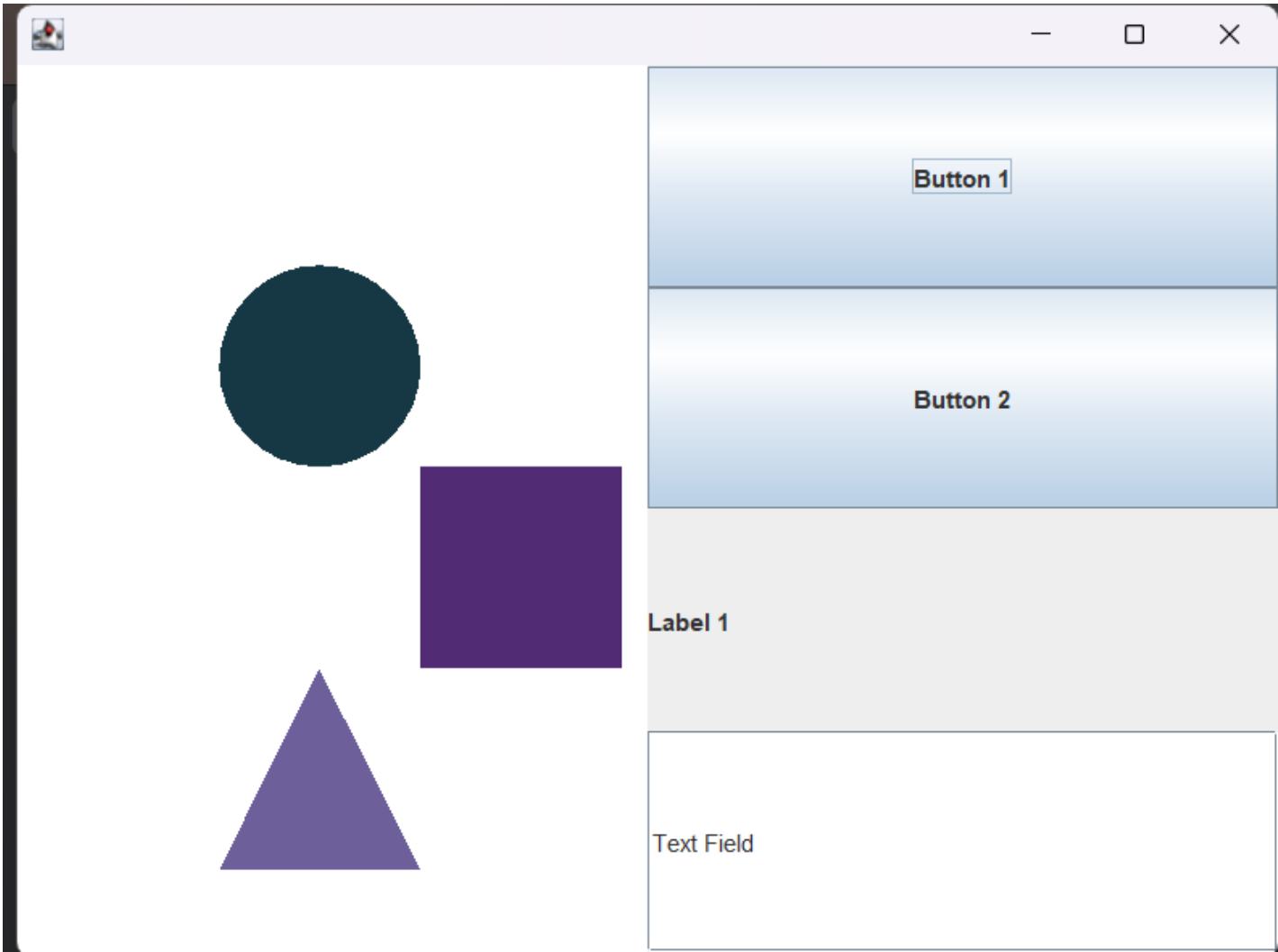
## Kod:

```
1 package pojava.lab2b.zadB;
2
3 import pojava.lab2b.zadA.CloseableFrame;
4
5 import javax.swing.*;
6 import java.awt.*;
7 import java.util.Random;
8
9
10 public class ThreeShapesPanel extends JPanel {
11     5 usages
12     private final Color[] colors = new Color[3];
13     1 usage
14     public ThreeShapesPanel() {
15         Random random = new Random();
16         for (int i = 0; i < colors.length; i++) {
17             colors[i] = new Color(random.nextInt( bound: 256), random.nextInt( bound: 256), random.nextInt( bound: 256));
18         }
19     }
20
21     @Override
22     protected void paintComponent(Graphics g) {
23         super.paintComponent(g);
24
25         // koło
26         g.setColor(colors[0]);
27         g.fillOval( x: 100, y: 100, width: 100, height: 100);
28     }
29 }
```

```
23         // koło
24         g.setColor(colors[0]);
25         g.fillOval( x: 100, y: 100, width: 100, height: 100);
26
27         // kwadrat
28         g.setColor(colors[1]);
29         g.fillRect( x: 200, y: 200, width: 100, height: 100);
30
31         // trójkąt
32         g.setColor(colors[2]);
33         int[] x = {100, 150, 200};
34         int[] y = {400, 300, 400};
35         g.fillPolygon(x, y, nPoints: 3);
36     }
37
38
39 public static void main(String[] args) {
40     CloseableFrame frame = new CloseableFrame();
41     frame.setLayout(new GridLayout( rows: 1, cols: 2));
42
43     ThreeShapesPanel shapesPanel = new ThreeShapesPanel();
44     shapesPanel.setBackground(Color.white);
45     frame.add(shapesPanel);
46
47     JPanel controlPanel = new JPanel();
48     controlPanel.setLayout(new GridLayout( rows: 4, cols: 1));
49     controlPanel.add(new JButton( text: "Button 1"));
50     controlPanel.add(new JButton( text: "Button 2"));
51     controlPanel.add(new JLabel( text: "Label 1"));
52     controlPanel.add(new JTextField("Text Field"));
53 }
```

```
53
54
55     frame.add(controlPanel);
56
57     frame.setVisible(true);
58 }
59 }
```

Output:





## Zadanie C - Okno z trzema przyciskami

### Opis działania programu:

Program tworzy okno z trzema przyciskami, z których każdy wykonuje inną akcję. Klasa ThreeButtonFrame dziedziczy JFrame, a w jej konstruktorze inicjalizowane są przyciski umieszczone na panelu w układzie siatki. Pierwszy przycisk zamyka program, drugi zmienia tytuł okna na losową liczbę całkowitą, a trzeci losowo zmienia kolor tła panelu. W metodzie main() tworzony jest obiekt klasy ThreeButtonFrame i ustawiany jako widoczny.

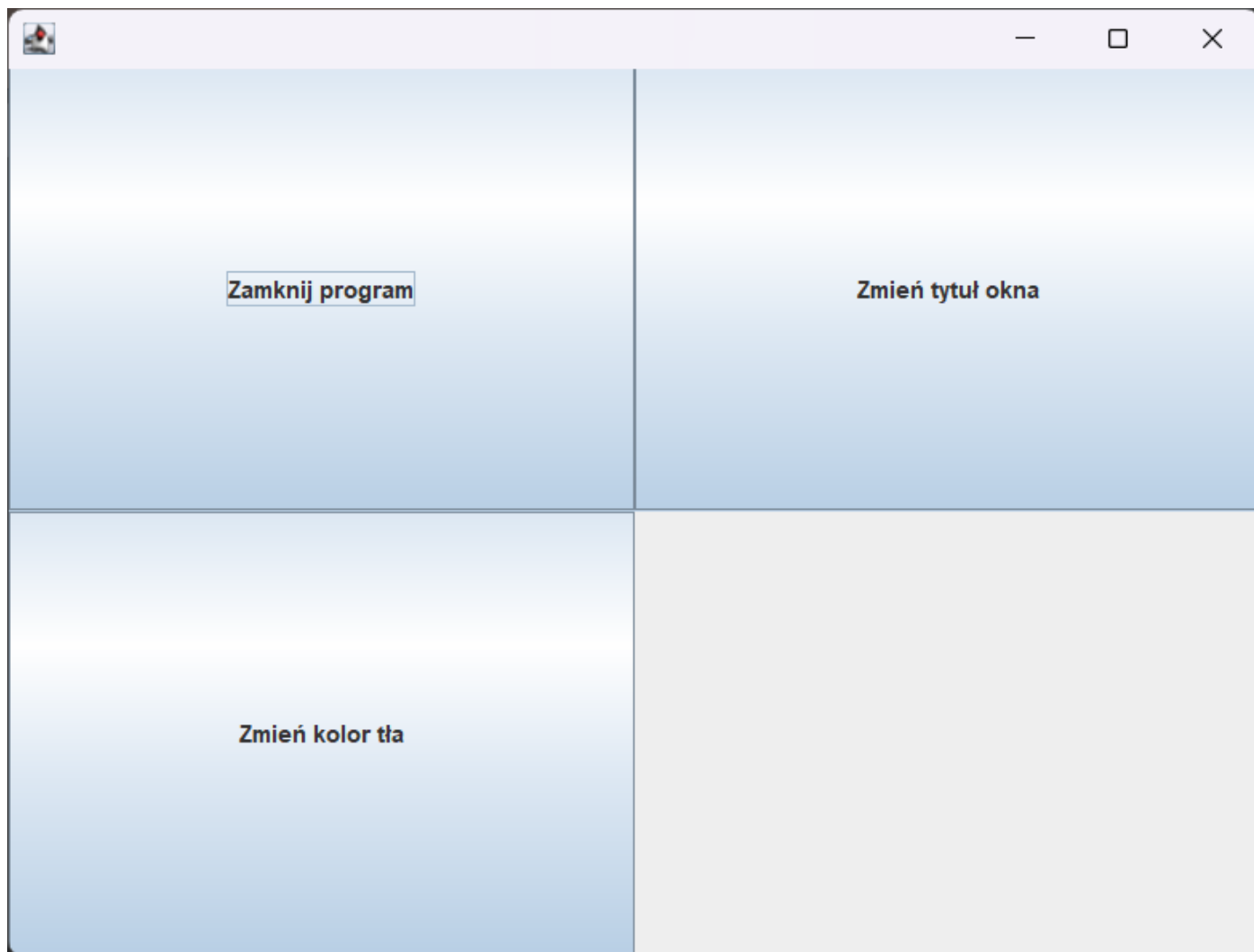
### Kod:

```
1 package pojawa.lab2b.zadC;
2
3 import javax.swing.*;
4 import java.awt.*;
5 import java.util.Random;
6
7 public class ThreeButtonFrame extends JFrame {
8
9     1 usage
10     public ThreeButtonFrame() throws HeadlessException {
11         this.setSize( width: 640, height: 480);
12         setDefaultCloseOperation(DISPOSE_ON_CLOSE);
13
14         JPanel panel = new JPanel();
15         this.add(panel);
16         panel.setLayout(new GridLayout( rows: 2, cols: 3));
17
18         JButton exitButton = new JButton( text: "Zamknij program");
19         exitButton.addActionListener(e -> {System.exit( status: 0);});
20         panel.add(exitButton);
21
22         JButton changeTitle = new JButton( text: "Zmień tytuł okna");
23         changeTitle.addActionListener(e -> {
24             Random random = new Random(); this.setTitle(String.valueOf(random.nextInt( bound: 100)));});
25         panel.add(changeTitle);
26
27         JButton changeBackgroundColor = new JButton( text: "Zmień kolor tła");
28         changeBackgroundColor.addActionListener(e -> {
29             Random color = new Random();
30             panel.setBackground(new Color(color.nextInt( bound: 256), color.nextInt( bound: 256), color.nextInt( bound: 256)));
31         });
32
33     }
34
35     public static void main(String[] args) {
36         ThreeButtonFrame frame = new ThreeButtonFrame();
37         frame.setVisible(true);
38     }
39 }
40 }
```

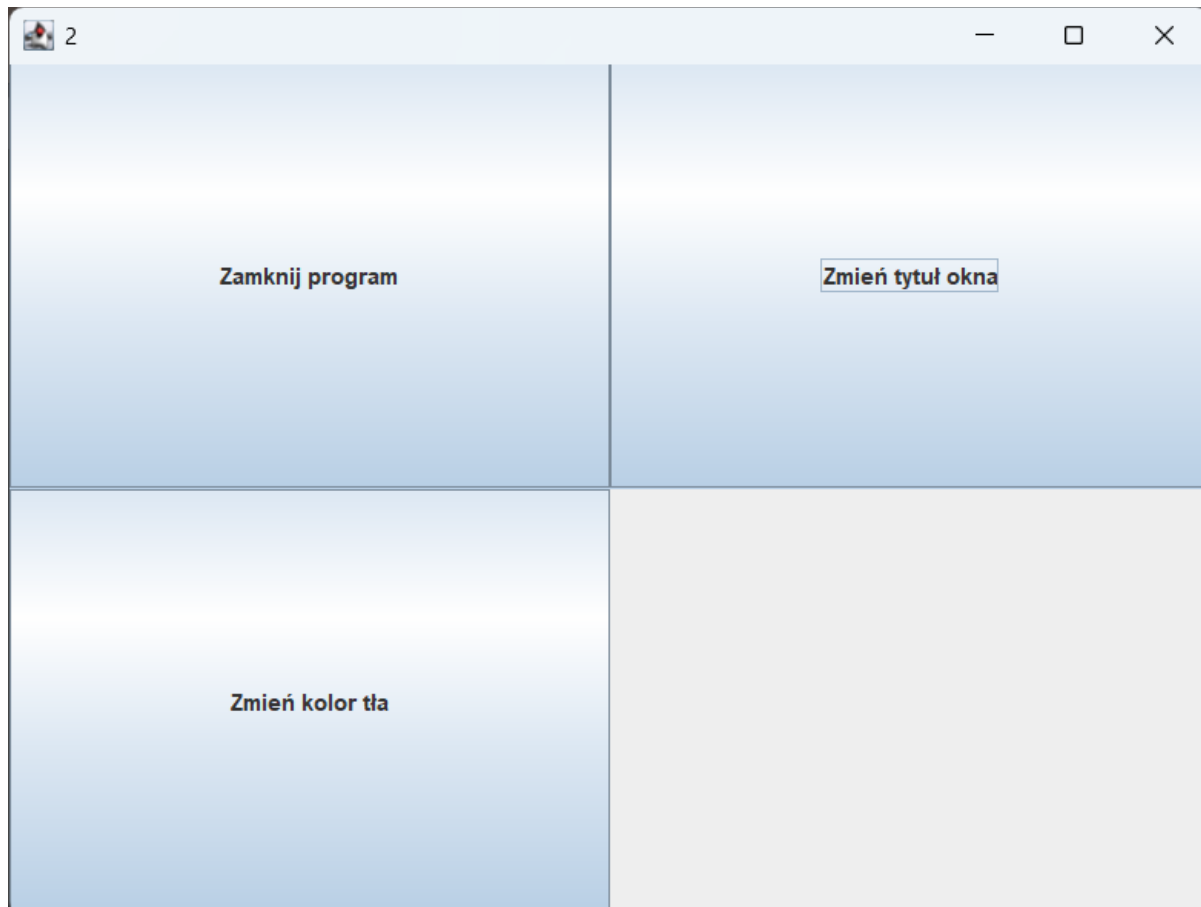
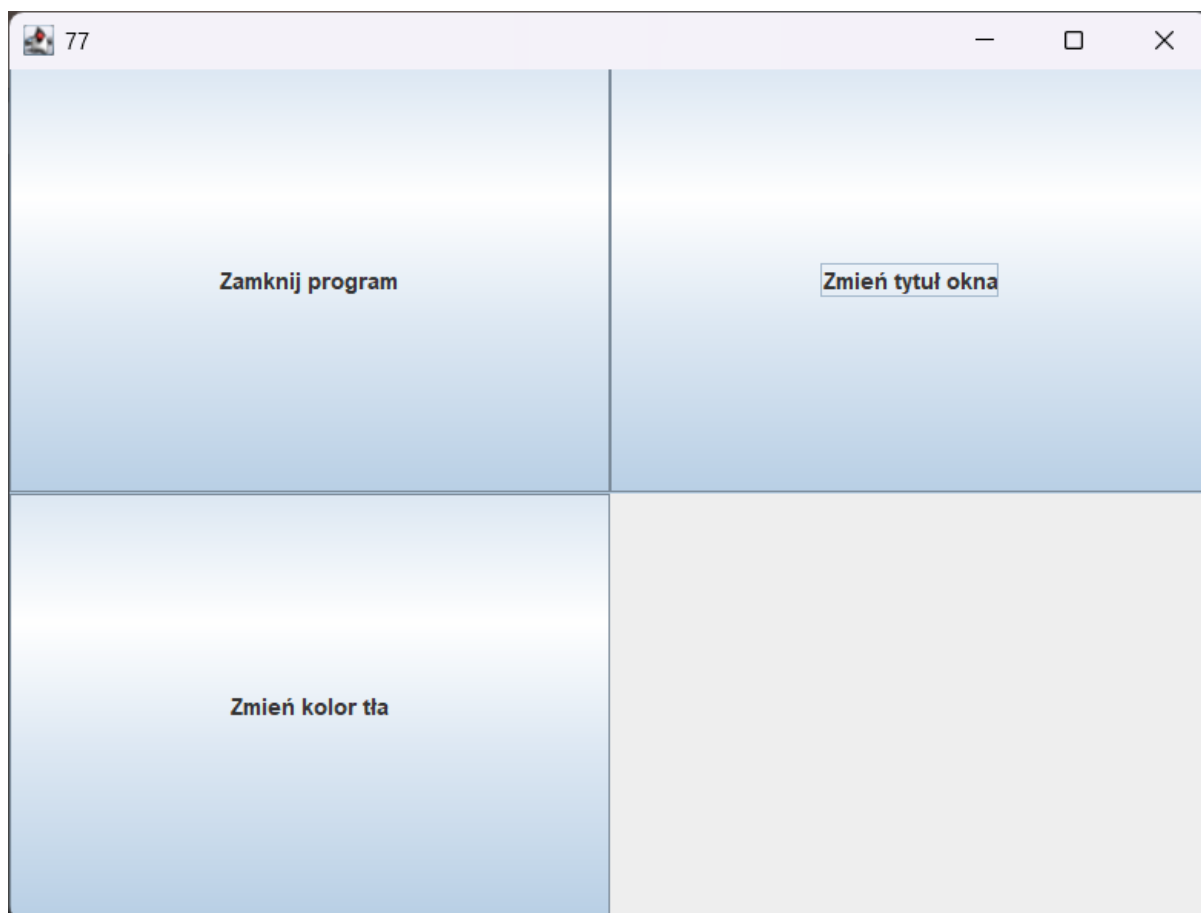
### Output:

Po włączeniu programu wyświetla się nam na ekranie okno:

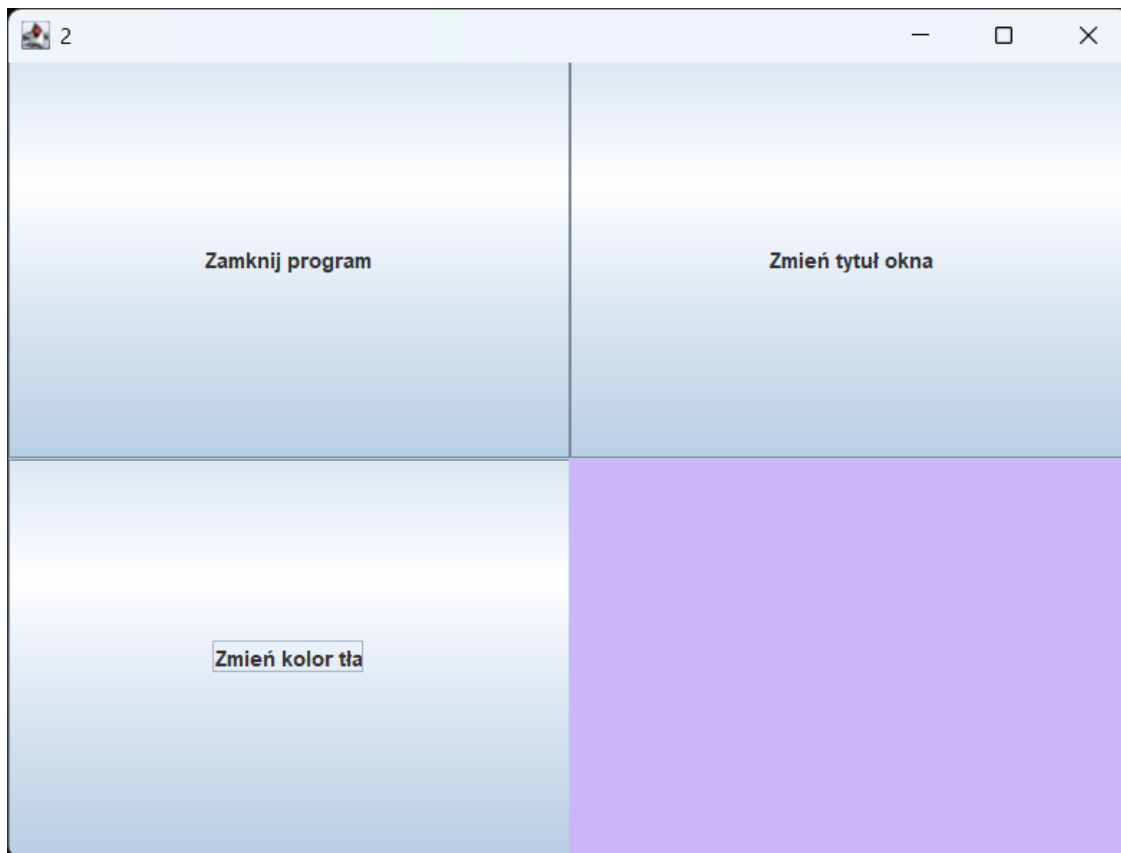
- Przycisk „Zamknij program”: domyślnie zamyka program



- Przycisk „Zmień tytuł okna”: z każdym kolejnym kliknięciem tego przycisku, tytuł okna zmieni się na losową liczbę całkowitą z zakresu od 0 do 99, co można zauważyć na przykładach poniżej:



- Przycisk „Zmień kolor tła”: z każdym kolejnym kliknięciem tego przycisku, kolor tła okna zmieni się na inny, co przedstawiono na przykładach poniżej:



---

## Kalkulator

---

### Opis działania programu:

Program implementuje kalkulator wykonujący podstawowe operacje arytmetyczne oraz operacje na pamięci.

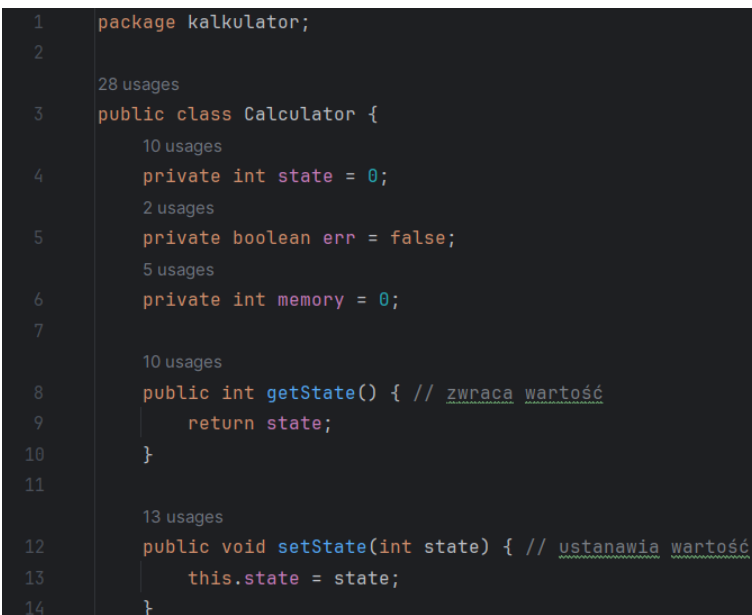
Klasa Calculator przechowuje pola takie jak:

- aktualną wartość (int state),
- flagę błędu (boolean err),
- pamięć (memory).

### Metody dostępu do wartości kalkulatora:

- getState() – zwraca aktualną wartość state
- setState() – ustawia state na podaną wartość

### Opisany fragment kodu:

A screenshot of a code editor with a dark background. The code is in Java and defines a Calculator class. Line numbers 1 through 14 are visible on the left. The code includes package, class, private fields (state, err, memory), and public methods (getState, setState).

```
1 package kalkulator;
2
3 public class Calculator {
4     private int state = 0;
5     private boolean err = false;
6     private int memory = 0;
7
8     public int getState() { // zwraca wartość
9         return state;
10    }
11
12    public void setState(int state) { // ustanawia wartość
13        this.state = state;
14    }
```

### Zadanie 2:

Kod obsługuje także podstawowe operacje arytmetyczne, takie jak:

- add(int value) - dodawanie,
- mult(int value) – mnożenie,
- sub(int value) – odejmowanie,
- oraz div(int value) – dzielenie. Ustanawiamy tutaj warunek if, który zapobiega dzieleniu przez 0 i powoduje ustawienie błędu.

### Opisany fragment kodu:

```
16      // podstawowe operacje
17      2 usages
18      public void add(int value){ // dodawanie
19          state += value;
20      }
21
22      3 usages
23      public void mult(int value){ // mnożenie
24          state *= value;
25      }
26
27      2 usages
28      public void sub(int value){ // odejmowanie
29          state -= value;
30      }
31
32      3 usages
33      public void div(int value){ // dzielenie
34          if (value == 0) { // error dla dzielenia przez 0
35              err = true;
36          } else {
37              state /= value;
38          }
39      }
40  }
```

---

### Zadanie 3:

Dodatkowo kalkulator pozwala na:

- zachowywanie aktualnej wartości wyniku w pamięci (saveMem),
- dodawanie zawartości wyniku zapisanego w pamięci jako aktualny wynik (useMem),
- a także modyfikację wyniku poprzez dodawanie (addMem) wartości z pamięci,
- lub odejmowanie od wyniku (subMem) wartości z pamięci,
- getMem() natomiast pozwala na zwrócenie aktualnej wartości memory.

### Opisany fragment kodu:

```
37      // przyciski MS, MR, M+, M-
38      2 usages
39      public void saveMem(){ // ustawia pamięć na aktualną wartość
40          memory = state;
41      }
42
43      1 usage
44      public void useMem(){ // dodaje zawartość pamięci do obecnej wartości
45          state += memory;
46      }
47
48      1 usage
49      public void addMem(){ // dodaje wartość do pamięci
50          memory += state;
51      }
52
53      1 usage
54      public void subMem(){ // odejmuje wartość od pamięci
55          memory -= state;
56      }
57  }
```

```

53      3 usages
54      public int getMem(){ // zwraca pamięć
55      |      return memory;
56      }

```

#### Zadanie 4:

Oczywiście kalkulator obsługuje także błędy obliczeń za pomocą metody `getError()`. Zwraca ona ustawione pole `err`, które jest wartością typu `boolean` i informuje, czy dany błąd występuje, czy też nie.

#### Opisany fragment kodu:

```

5      2 usages
      private boolean err = false;
      5 usages

```

```

58      1 usage
      public boolean getError(){
59      |      return err;
60      }

```

Przykład użycia tego stanu w jednej z istniejących operacji:

```

29      3 usages
      public void div(int value){ // dzielenie
30      |      if (value == 0) { // error dla dzielenia przez 0
31      |          err = true;
32      |      } else {
33      |          state /= value;
34      |      }
35      }

```

---

## Testy do zadań:

---

Działanie kalkulatora zostało zweryfikowane testami sprawdzającymi poprawność obliczeń oraz obsługę błędów. Testy znajdują się poniżej:

### Zadanie 1 i 2 – testy:

```
4
5  import static org.junit.Assert.assertEquals;
6  import static org.junit.Assert.assertTrue;
7
8  public class CalculatorTest {
9      @Test
10     public void testAddOne(){
11         // Arrange
12         // sut = System Under Test
13         Calculator sut = new Calculator();
14         // Act
15         sut.add(1);
16         // Assert
17         assertEquals("0+1 = 1", expected: 1, sut.getState());
18     }
19
20     @Test
21     public void testMultOneByTwo(){
22         Calculator sut = new Calculator();
23         sut.setState(1);
24         sut.mult(value: 2);
25         assertEquals("1*2 = 2", expected: 2, sut.getState());
26     }
27
```

```
28     @Test
29     public void testAddNegative() {
30         Calculator sut = new Calculator();
31         sut.setState(1);
32         sut.add(-1);
33         assertEquals("1+(-1) = 0", expected: 0, sut.getState());
34     }
35
36     @Test
37     public void testMultiplyOneByNegative() {
38         Calculator sut = new Calculator();
39         sut.setState(1);
40         sut.mult(value: -1);
41         assertEquals("1*(-1) = -1", expected: -1, sut.getState());
42     }
43
44     @Test
45     public void testMultiplyOneByZero() {
46         Calculator sut = new Calculator();
47         sut.setState(1);
48         sut.mult(value: 0);
49         assertEquals("1*0 = 0", expected: 0, sut.getState());
50     }
51
```



```

52     @Test
53     public void testSubtractOne() {
54         Calculator sut = new Calculator();
55         sut.setState(1);
56         sut.sub( value: 1);
57         assertEquals( message: "1-1 = 0", expected: 0, sut.getState());
58     }
59
60     @Test
61     public void testSubtractNegative() {
62         Calculator sut = new Calculator();
63         sut.setState(1);
64         sut.sub( value: -1);
65         assertEquals( message: "1-(-1) = 2", expected: 2, sut.getState());
66     }
67
68     @Test
69     public void testDivideTenByTwo() {
70         Calculator sut = new Calculator();
71         sut.setState(10);
72         sut.div( value: 2);
73         assertEquals( message: "10 / 2 = 5", expected: 5, sut.getState());
74     }
75
76     @Test
77     public void testDivideTenByNegative() {
78         Calculator sut = new Calculator();
79         sut.setState(10);
80         sut.div( value: -2);
81         assertEquals( message: "10 / (-2) = -5", expected: -5, sut.getState());
82     }

```

### Zadanie 3 – testy:

```

84     @Test
85     public void testSaveMemory() {
86         Calculator sut = new Calculator();
87         sut.setState(1);
88         sut.saveMem(); // zawiera 1
89         assertEquals( message: "Memory: 1", expected: 1, sut.getMem());
90     }
91
92     @Test
93     public void testUseMemory() {
94         Calculator sut = new Calculator();
95         sut.setState(1);
96         sut.saveMem(); // zawiera 1
97         sut.useMem(); // dodaje 1
98         assertEquals( message: "1 +(M 1) = 2", expected: 2, sut.getState());
99     }
100
101     @Test
102     public void testMPlus(){
103         Calculator sut = new Calculator();
104         sut.setState(1);
105         sut.addMem();
106         assertEquals( message: "(M 0)+1 = (M 1)", expected: 1, sut.getMem());
107     }

```

```

109     @Test
110     public void testMMinus(){
111         Calculator sut = new Calculator();
112         sut.setState(1);
113         sut.subMem();
114         assertEquals("message: "(M 0)-1 = (M -1)", expected: -1, sut.getMem());
115     }
116

```

#### Zadanie 4 – testy:

```

@Test
public void testDivideByZero() {
    Calculator sut = new Calculator();
    sut.setState(1);
    sut.div(value: 0);
    assertTrue(message: "1 / 0 = err", sut.getError());
}

```

#### Output dla wszystkich testów:

Run CalculatorTest x

✓ CalculatorTest (kalkulator) 5 ms

✓ Tests passed: 14 of 14 tests – 5 ms

C:\Users\marta\.jdk\openjdk-21.0.2\bin\java.exe ...

Process finished with exit code 0

Test Name	Duration
testDivideByZero	4 ms
testMPlus	0 ms
testAddNegative	0 ms
testUseMemory	0 ms
testDivideTenByTwo	0 ms
testDivideTenByNegative	0 ms
testSaveMemory	0 ms
testAddOne	1 ms
testMultOneByTwo	0 ms
testMMinus	0 ms
testMultiplyOneByZero	0 ms
testSubtractNegative	0 ms
testSubtractOne	0 ms
testMultiplyOneByNegative	0 ms

## Zadanie z wielomianem

### Zadanie 5:

W zadaniu tym należało obliczyć wartość wielomianu:

$$y = (x^5 + 4*(x^4) + 3*(x^3) - 2*(x^2) + 17) / (x^2 - 7*x + 1)$$

dla danych wartości:

$$x = \{-3, -2, -1, 1, 2, 3\}$$

oraz przedstawić wynik danych obliczeń w tabeli.

Klasa Wielomian posiada dwie metody:

- **obliczWielomian(int x)**, która oblicza wartość wielomianu dla podanego x i zwraca wynik jako int. Funkcja Math.pow() zwraca domyślnie wartości typu double, dlatego konieczna była konwersja do int.
- **main**, która tworzy dwie tablice (obie długości 6):
  - int[] x - gdzie przechowywane są wartości kolejnych x-ów, dla których obliczano wielomian
  - int[] wyniki - gdzie przechowywane są wyniki obliczeń.

Następnie przy pomocy pętli for dla każdej wartości x obliczana jest wartość wielomianu i zapisywana w tablicy wyniki. Na końcu tablica wyniki jest drukowana.

### Opisany fragment kodu:

```
1 package kalkulator;
2
3 import java.util.Arrays;
4
5 public class Wielomian {
6
7     public static int obliczWielomian(int x) { 1 usage
8         return (int) ((Math.pow(x, 5) + 4 * Math.pow(x, 4) + 3 * Math.pow(x, 3) - 2 * Math.pow(x, 2) + 17)
9             / (Math.pow(x, 2) - 7 * x + 1));
10    }
11
12    public static void main(String[] args) {
13        int[] x = new int[]{-3, -2, -1, 1, 2, 3};
14        int[] wyniki = new int[x.length];
15        for (int i = 0; i < x.length; i++) {
16            wyniki[i] = obliczWielomian(x[i]);
17        }
18        System.out.println(Arrays.toString(wyniki));
19    }
20 }
21
```

### Output:

```
C:\Users\martas\jdk\openjdk-21.0.2\
[0, 0, 1, -4, -14, -58]

Process finished with exit code 0
```