Faculty of Electronics
and Information
Technology
WARSAW UNIVERSITY OF TECHNOLOGY

Graphical User Interfaces (EGUI)

01-introduction

Julian Myrcha
Institute of Computer Science

October 6, 2024

General trends

Many programs communicate with the user

- Data processing
  - Data recording, management, reporting
- User utilities
  - text processors
  - graphical processors
  - browsers
  - other thousands types

General trends

We are going to Web based solutions

- Server-based - (almost) all interactions involve round-trip to the server
- SPA - Single-Page Application
  - Angular
  - React
  - Vue
  - Svelte

User Interface generations

- bach mode (ODRA 1305)
- Terminals for multi-user information systems
- Graphical user interface (WIMP)
- Post-WIMP

Batch mode

- it is a history ...
  - punched cards
  - magnetic tapes
  - line printers
- lack of user interface
- lack of interactive users

Terminals for multi-user information systems

- interaction with the user
  - question and answer
  - question is a command with parameters
- Examples of the systems:
  - DOS
  - Linux

Graphical User Interface

- WIMP - standard created by XEROX
  - W - Windows
  - I - Icons
  - M - Menus
  - P - Pointer (cursor)
- Synthesis of the popular graphics and character forms of human-computer interaction
- Widely used in following systems:
  - Macintosh
  - Windows
  - Unix (Linux)

Pros and Cons od WIMP (1)

- complicated
- number of functions exceeds user needs:
  - users spend to much time on manipulating the user interfaca not productively creating the results
- only human vision used

Pros and Cons od WIMP (2)

- its is victim of its success - little progress
- but the world is changing:
  - browsers
  - mobile devices
    - smartphones
    - tablets
    - car navigation
- virtual reality

Pros and Cons od WIMP (3)

- touch interface
- gesture recognition
- virtual reality headsets
- speach recognition
- speach commands

Generations

- character based - second generation
- graphical - third generation

Character based interfaces

- communication based on character sequences
- we have following options:
  - question-answer
  - command language
  - natural language

Graphical user interfaces

- quality of the interface is a function of compatibility between user expectations and system behaviour
- interface should strengthen natural flow of work

Layers

Layers of GUI

metaphor layer - imitation of the real situation
method layer - way of communication with the user
device layer - devices used to communicate
physical layer - what is done

Methaphor Layer

- It is similarity between system behaviour and real life situations
- examples:

  document - screen like sheet of the paper
  desktop - screen like desktop (a table to work), with documents and accessories
  dashboard - screen like dashboard in the factory
  path and folders - data organised in a hierarchical way like accountant stores

Elements of the GUI

- Menu
- Form
- Windows
- Buttons
- Input fields
- others ...

Standards

- IBM CUA - Common User Access
- Set of the rules describing possible ways of communication:
  - how keyboard and mouse is used
  - how menu should be organised
  - how window should be organised
  - how input data verification should be done
  - how help and tips should be provided
  - how user could move between forms and applications
- User interface based on graphical elements
  - icons
  - visible elements

Rules of the graphical design

concrete methaphores - system should be based on concrete and widelly undersood metaphores from the real world

direct influence - user has a wheel

- user is providing commands
- system responds with information about sucsess/failure and the reasons of failure

indication - see and point

- screen should be working environment
- direct interaction with the displayed elements
- mouse, keyboard, other devices

Rules of the graphical design

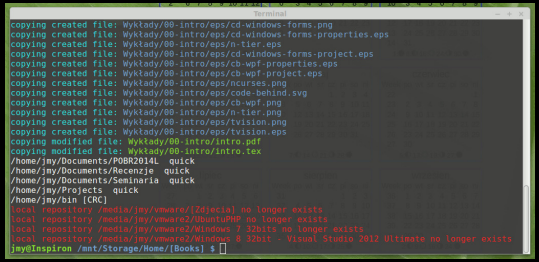
cohesion - consistence of the way in which user is doing their work
WYSIWYG - what you see is what you get, printer result of the work should be the same as displayed

There are technical limitations of WYSIWYG:

- resolution of the printers and screens
- character sets
- colors

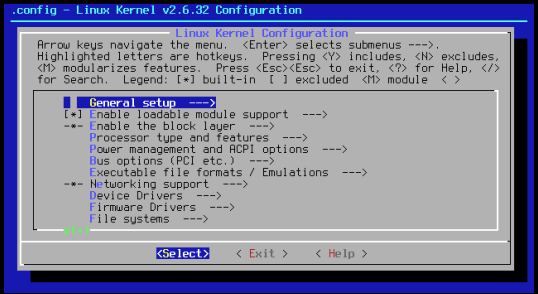
console programs

- **stdin/stdout**
- ncurses
- turbo vision



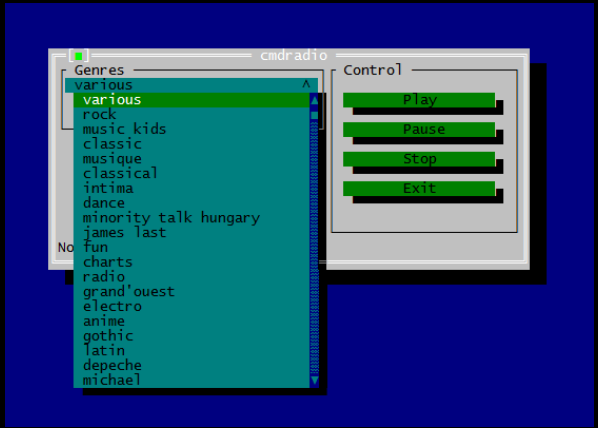
console programs

- stdin/stdout
- **ncurses**
- turbo vision



console programs

- stdin/stdout
- ncurses
- turbo vision



Windowing programs

- original Windows programming -> Charles Petzold
- MFC - Microsoft Foundation Classes

Charles Petzold


```
1  #include <windows.h>
2  LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
3  int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
4          PSTR szCmdLine, int iCmdShow) {
5       static TCHAR szAppName[] = TEXT ("HelloWin") ;
6       HWND   hwnd ;
7       MSG    msg ;
8       WNDCLASS     wndclass ;
9       wndclass.style   = CS_HREDRAW | CS_VREDRAW ;
10      wndclass.lpfnWndProc   = WndProc ;
11      wndclass.cbClsExtra    = 0 ;
12      wndclass.cbWndExtra    = 0 ;
13      wndclass.hInstance     = hInstance ;
14      wndclass.hIcon   = LoadIcon (NULL, IDI_APPLICATION) ;
15      wndclass.hCursor       = LoadCursor (NULL, IDC_ARROW) ;
16      wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
17      wndclass.lpszMenuName  = NULL ;
18      wndclass.lpszClassName = szAppName ;
19      if (!RegisterClass (&wndclass)) {
20     MessageBox (NULL, TEXT ("This program requires Windows NT!"),
21           szAppName, MB_ICONERROR) ;
22     return 0 ;
23      }
24      hwnd = CreateWindow (szAppName,      // window class name
25         TEXT ("The Hello Program"), // window caption
26         WS_OVERLAPPEDWINDOW,  // window style
27         CW_USEDEFAULT,        // initial x position
28         CW_USEDEFAULT,        // initial y position
29         CW_USEDEFAULT,        // initial x size
30         CW_USEDEFAULT,        // initial y size
31         NULL,           // parent window handle
32         NULL,           // window menu handle
33         hInstance,      // program instance handle
34         NULL) ;         // creation parameters
```

Charles Petzold


```
35
36        ShowWindow (hwnd, iCmdShow) ;
37        UpdateWindow (hwnd) ;
38
39        while (GetMessage (&msg, NULL, 0, 0))     {
40       TranslateMessage (&msg) ;
41       DispatchMessage (&msg) ;
42        }
43        return msg.wParam ;
44  }
45  LRESULT CALLBACK WndProc (HWND hwnd,
46          UINT message, WPARAM wParam, LPARAM lParam){
47        HDC   hdc ;
48        PAINTSTRUCT ps ;
49        RECT  rect ;
50
51        switch (message)   {
52        case WM_CREATE:
53       PlaySound (TEXT("hellowin.wav"),NULL,SND_FILENAME|SND_ASYNC);
54       return 0 ;
55        case WM_PAINT:
56       hdc = BeginPaint (hwnd, &ps) ;
57       GetClientRect (hwnd, &rect) ;
58           DrawText (hdc, TEXT ("Hello, Windows 98!"),-1,&rect,
59           DT_SINGLELINE | DT_CENTER | DT_VCENTER) ;
60       EndPaint (hwnd, &ps) ;
61       return 0 ;
62        case WM_DESTROY:
63       PostQuitMessage (0) ;
64       return 0 ;
65        }
66        return DefWindowProc (hwnd, message, wParam, lParam) ;
67  }
```

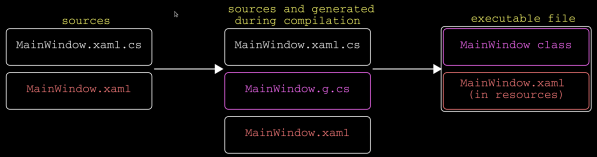
Rapid Application Development = Code Behind

- simple idea - drag and drop a control from the controll palette
- provide a code for the events
- other components of the form are form attributes


```
1  private void btnEgui_Click(object sender, RoutedEventArgs e) {
2      btnEgui.Content = "Clicked";
3  }
```

WPF - how source files are used during compilation

- Developer create `*.xaml` file to design view of the application
- Developer declare in `*.xaml` events raised in response to user interaction
- Developer define in `*.xaml.cs` partial class source file methods to be connected with those events
- Compiler based on `*.xaml` create second part of partial class responsible for:
  - define class members for controls declared in `*.xaml`
  - populating class members properties using values in `*.xaml` retrieved from application resources
  - binding events with methods declared by the programmer



Rapid Application Development = Code Behind

- MainWindow.xaml file created by a programmer (in designer or in text editor)

```
1  <Window x:Class="SampleApp.MainWindow"
2    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
5    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
6    xmlns:local="clr-namespace:SampleApp"
7    mc:Ignorable="d"
8    Title="MainWindow" Height="350" Width="525">
9    <Grid>
10     <Button x:Name="btnEgui" Content="EGUI" HorizontalAlignment="Left"
11           Margin="120,70,0,0" VerticalAlignment="Top"
12           Width="70" Click="btnEgui_Click" />
13   </Grid>
14 </Window>
```

(content was simplified)

Rapid Application Development = Code Behind

- Control properties are declared `*.xaml` file
- If some property is not set there then default value is used
- `XML` file is stored in text form in application resources (is inside `*.exe` file)
- When window is created its controls are initialised using values from `*.xaml`

WPF - MainWindow.g.cs

- file MainWindow.g.cs generated automatically during compilation (not visible in Visual Studio)

```
1  namespace SampleApp {
2    public partial class MainWindow: System.Windows.Window,
3            System.Windows.Markup.IComponentConnector {
4      internal System.Windows.Controls.Button btnEgui;
5      public void InitializeComponent() {
6        System.Uri resLocater = new System.Uri("/SampleApp;component/mainwindow.xaml", System.UriKind.Relative);
7        System.Windows.Application.LoadComponent(this,resLocater);// <--- set member properties
8      }                 // using *.xaml from resoutces
9      void System.Windows.Markup.IComponentConnector.Connect(int connectionId, object target) {
10       switch (connectionId) {
11       case 1:  // connect this.btnEgui reference to control created in LoadComponent
12   this.btnEgui = ((System.Windows.Controls.Button)(target));
13   this.btnEgui.Click+=new System.Windows.RoutedEventHandler(this.btnEgui_Click);
14   return;
15       }}}}
```

(content was simplified)

WPF - MainWindow.xaml.cs

- file MainWindow.xaml.cs modified by a programmer

```
1  namespace SampleApp {
2      public partial class MainWindow : Window { // <--- class is partial
3    public MainWindow() {
4        InitializeComponent();
5    }
6
7    private void btnEgui_Click(object sender, RoutedEventArgs e) {
8        btnEgui.Content = "Clicked";      // <--- reference to member from second part
9    }
10     }
11 }
```

- By using partial classes content generated during compilation handled separately
- Event handling code declared in separate private methods of the window
- We have access to members located in second part of partial class

Console interaction

```
1  package pap;
2  import java.util.Scanner;
3
4  public class Main {
5      static String selectItem(String items[]) {
6          Scanner in = new Scanner(System.in);
7          System.out.println("enter color number or 0 to cancel");
8          var number = 1;
9          for(var c:items) {
10             System.out.println(String.valueOf(number++)+" "+c);
11         }
12         int a = in.nextInt();
13         if(a==0) return "";
14         return items[a-1];
15     }
16
17     public static void main(String[] args) {
18         Scanner in = new Scanner(System.in);
19         var color = "";
20         var model = "";
21         var choice = 0;
22         do {
23             System.out.println("selected color: "+color+", selected model: "+model);
24             System.out.println("1 - change color");
25             System.out.println("2 - change model");
26             System.out.println("enter operation number or 0 to cancel");
27             choice = in.nextInt();
28             switch(choice) {
29                 case 1:
30                     color = selectItem(new String[]{"red","green","blue"});
31                     break;
32                 case 2:
33                     model = selectItem(new String[]{"ford","citroen","fiat"});
34                     break;
```

Console interaction

```
35              }
36          } while(choice != 0);
37          System.out.println("selected color: "+color+", selected model: "+model);
38      }
39  }
```

Console interaction

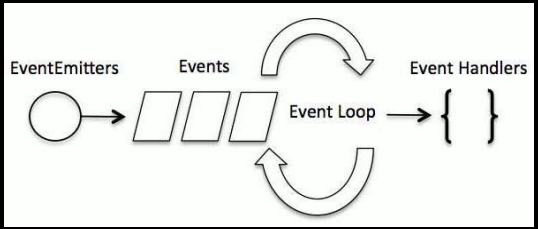
```
selected color: , selected model:
1 - change color
2 - change model
enter operation number or 0 to cancel
1
enter color number or 0 to cancel
1 red
2 green
3 blue
3
selected color: blue, selected model:
1 - change color
2 - change model
enter operation number or 0 to cancel
2
enter color number or 0 to cancel
1 ford
2 citroen
3 fiat
2
selected color: blue, selected model: citroen
1 - change color
2 - change model
enter operation number or 0 to cancel
1
enter color number or 0 to cancel
1 red
2 green
3 blue
1
selected color: red, selected model: citroen
1 - change color
2 - change model
enter operation number or 0 to cancel
```

Console interaction

```
35  0
36  selected color: red, selected model: citroen
37
38  Process finished with exit code 0
```

Event driven programming

- An event can be anything (key press/release, mouse movement, ...)
- Events are placed in a queue
- The appearance of a new event does not interrupt the previous one - we get free synchronization
- Very easy to handle different event sources (keyboard, mouse, timekeeping, interaction with another system or hardware)
- Events in the queue can be modified (e.g. duplicates removed)



Event driven programming Windows

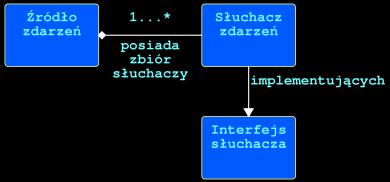
```
1  #include <windows.h>
2  LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
3  int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
4                      PSTR szCmdLine, int iCmdShow) {
5      ...
6      WNDCLASS     wndclass ;                          // fill the structure!!!
7      if (!RegisterClass (&wndclass)) {
8           MessageBox (NULL, TEXT ("Problem!"),  szAppName, MB_ICONERROR) ;
9           return 0 ;
10     }
11     hwnd = CreateWindow (...) ;                      // creation parameters
12     ShowWindow (hwnd, iCmdShow) ;
13     UpdateWindow (hwnd) ;
14     while (GetMessage (&msg, NULL, 0, 0))     {
15          TranslateMessage (&msg) ;
16          DispatchMessage (&msg) ;
17     }
18     return msg.wParam ;
19 }
```

Event driven programming Windows

```
1  LRESULT CALLBACK WndProc (HWND hwnd,  UINT message, WPARAM wParam, LPARAM lParam){
2       HDC         hdc ;    PAINTSTRUCT ps ;    RECT        rect ;
3       switch (message)   {
4       case WM_CREATE:
5            PlaySound (TEXT("hellowin.wav"),NULL,SND_FILENAME|SND_ASYNC);
6            return 0 ;
7       case WM_PAINT:
8            hdc = BeginPaint (hwnd, &ps) ;
9            GetClientRect (hwnd, &rect) ;
10                    DrawText (hdc, TEXT ("Hello, Windows 98!"),-1,&rect,
11                    DT_SINGLELINE | DT_CENTER | DT_VCENTER) ;
12           EndPaint (hwnd, &ps) ;
13           return 0 ;
14      case WM_DESTROY:
15           PostQuitMessage (0) ;
16           return 0 ;
17      }
18      return DefWindowProc (hwnd, message, wParam, lParam) ;
19 }
```

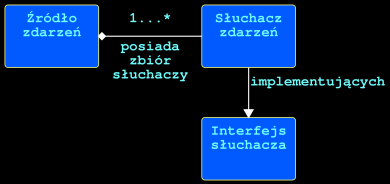
Events

- event listeners objects - instances of classes that implement a special interface



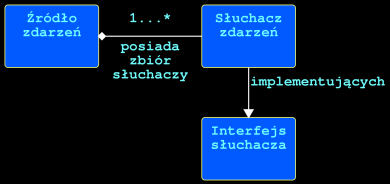
Events

- event listeners objects - instances of classes that implement a special interface
- event sources (buttons, scroll bars) - record listeners and send them event objects



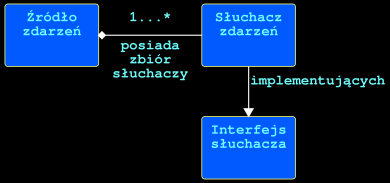
Events

- event listeners objects - instances of classes that implement a special interface
- event sources (buttons, scroll bars) - record listeners and send them event objects
- In the event of an event, all registered listeners are informed



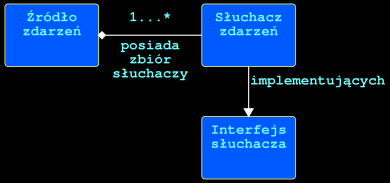
Events

- event listeners objects - instances of classes that implement a special interface
- event sources (buttons, scroll bars) - record listeners and send them event objects
- In the event of an event, all registered listeners are informed
- Each listener decides what to do in this case



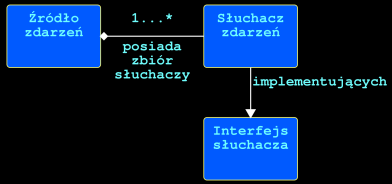
Events

- The students are implemented as:
  - Methods of the object (most often the instance of the window containing the controls -> Delphi, C#



Events

- The students are implemented as:
  - Methods of the object (most often the instance of the window containing the controls -> Delphi, C#
  - listener objects implementing interface -> Java



Event handling - Delphi, C

- Event is a piece of code to be run after well defined condition
- Current window is a default parameter of the event
- Window is represented by an object
- So to define an event we need pair: an object and a method
- All event handlers are the methods of the object

Event handling - Delphi, C

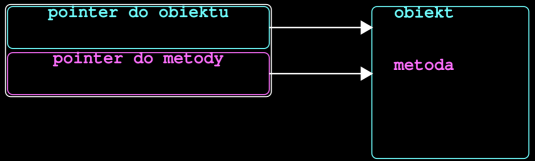
- Event is a piece of code to be run after well defined condition
- Current window is a default parameter of the event
- Window is represented by an object
- So to define an event we need pair: an object and a method
- All event handlers are the methods of the object

Event handling - Delphi, C

- Event is a piece of code to be run after well defined condition
- Current window is a default parameter of the event
- Window is represented by an object
- So to define an event we need pair: an object and a method
- All event handlers are the methods of the object

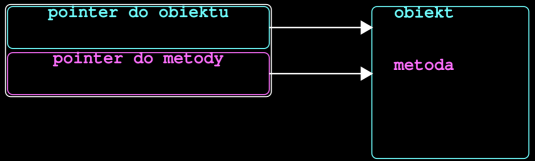
Event handling - Delphi, C

- Event is a piece of code to be run after well defined condition
- Current window is a default parameter of the event
- Window is represented by an object
- So to define an event we need pair: an object and a method
- All event handlers are the methods of the object



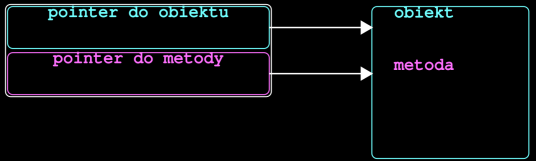
Event handling - Delphi, C

- Event is a piece of code to be run after well defined condition
- Current window is a default parameter of the event
- Window is represented by an object
- So to define an event we need pair: an object and a method
- All event handlers are the methods of the object
  - preferable a window containing controls



Event handling - Delphi, C

- Event is a piece of code to be run after well defined condition
- Current window is a default parameter of the event
- Window is represented by an object
- So to define an event we need pair: an object and a method
- All event handlers are the methods of the object
  - preferable a window containing controls
  - methods of the class will have access to that contols



listener interface (Java)

☐ listener interface

```
1  public interface ActionListener extends EventListener {
2      public void actionPerformed(ActionEvent e);
3  }
```

listener interface (Java)

☐ listener interface

```
1  public interface ActionListener extends EventListener {
2     public void actionPerformed(ActionEvent e);
3  }
```

☐ listener interface implementation

```
1  class ButtonListener implements ActionListener {
2  public void actionPerformed(ActionEvent event) {
3        . . .
4     }
5  }
```

listener interface (Java)

☐ listener interface

```
1  public interface ActionListener extends EventListener {
2     public void actionPerformed(ActionEvent e);
3  }
```

☐ registering a listener to the source

```
1  ActionListener listener = new ButtonListener();
2  JButton button = new JButton("OK");
3  button.addActionListener(listener);
```

☐ listener interface implementation

```
1  class ButtonListener implements ActionListener {
2  public void actionPerformed(ActionEvent event) {
3        . . .
4     }
5  }
```

listener interface (Java)

☐ listener interface

```
1  public interface ActionListener extends EventListener {
2     public void actionPerformed(ActionEvent e);
3  }
```

☐ listener interface implementation

```
1  class ButtonListener implements ActionListener {
2  public void actionPerformed(ActionEvent event) {
3        . . .
4     }
5  }
```

☐ registering a listener to the source

```
1  ActionListener listener = new ButtonListener();
2  JButton button = new JButton("OK");
3  button.addActionListener(listener);
```

☐ each press of the button calls the method `listener.actionPerformed(event)`

code of the first example in Java (Swing)

```
1  package SimpleSwing;
2
3  import java.awt.event.ActionEvent;
4  import java.awt.event.ActionListener;
5
6  import javax.swing.JButton;
7  import javax.swing.JFrame;
8  import javax.swing.SwingUtilities;
9
```

code of the first example in Java (Swing)

```
10  class ButtonListener implements ActionListener {
11    @Override
12    public void actionPerformed(ActionEvent e) {
13      ((JButton)e.getSource()).setText("nacisnieto"); // a jak inne kontrolki?
14    }
15  }
16
17  public class SimpleSwingSample {
18    public static void main(String[] args) {
19      SwingUtilities.invokeLater(() -> {
20        JFrame frame = new JFrame("Pierwszy Przycisk");
21        frame.setBounds(100, 100, 450, 300);          // nadaje rozmiar oknu
22        // zakoncz aplikacje po zamknieciu okna
23        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
24        JButton closeButton = new JButton("jeszcze nie nacisnieto");
25        closeButton.addActionListener(new ButtonListener());
26        frame.getContentPane().add(closeButton);
27        frame.setVisible(true);                       // pokaz onkno
28      });
29    }
30  }
```

code of the first example in Java (Swing+Lambda)


```
1  package SimpleSwing;
2  import java.awt.event.ActionEvent;
3  import java.awt.event.ActionListener;
4
5  import javax.swing.JButton;
6  import javax.swing.JFrame;
7  import javax.swing.SwingUtilities;
8
9  public class SimpleSwingSample {
10   public static void main(String[] args) {
11     SwingUtilities.invokeLater(() -> {
12       JFrame frame = new JFrame("Pierwszy Przycisk");
13       frame.setBounds(100, 100, 450, 300);          // nadaje rozmiar oknu
14       // zakoncz aplikacje po zamknieciu okna
15       frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
16       JButton closeButton = new JButton("jeszcze nie nacisnieto");
17       closeButton.addActionListener(e->((JButton)e.getSource()).setText("nacisnieto"));
18       frame.getContentPane().add(closeButton);
19       frame.setVisible(true);                  // pokaz okno
20     });
21   }
22 }
```

event handling - Java

- But in Java we do not have pointers to methods (we have only references to objects)
- It is not so nice in Java ...

- So we could reference object, which implies that this object should implement given interface (because method should be known)

```
1  button.addActionListener(new ActionListener() {
2      public void actionPerformed(ActionEvent event) {
3          buttonPanel.setBackground(backgroundColor);
4      }
5  });
```