Programowanie obiektowe

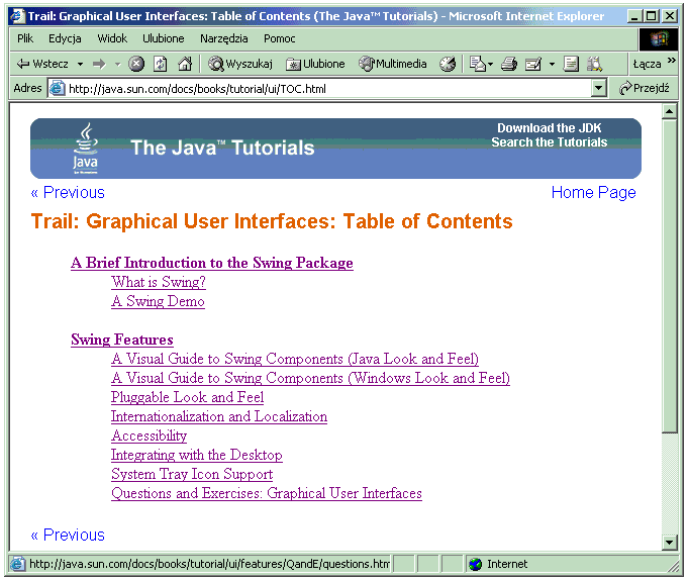
Programowanie graficznych interfejsów użytkownika

Paweł Rogaliński
Instytut Informatyki, Automatyki i Robotyki
Politechniki Wrocławskiej

pawel.rogalinski@pwr.wroc.pl

The Java Tutorials

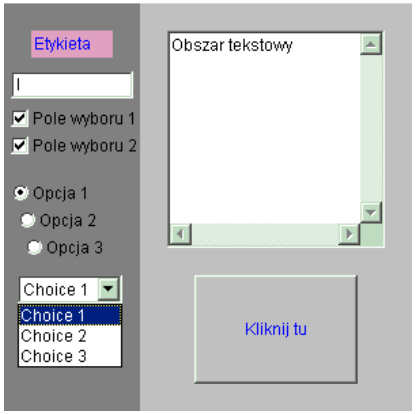
http://java.sun.com/docs/books/tutorial/ui/index.html



GUI – graficzny interfejs użytkownika

Standardowe pakiety ***java.awt*** (AWT) oraz ***javax.swing*** (Swing) zawierają klasy definiujące wiele różnorodnych komponentów wizualnej interakcji programu z użytkownikiem (okna, przyciski, listy itp.). Są one reprezentowane przez klasy wywodzące się z klasy ***java.awt.Component***.

Komponenty są umieszczane w kontenerach – specjalnych komponentach umożliwiających przechowywanie innych elementów GUI (komponentów oraz innych kontenerów)



Komponenty AWT a komponenty Swingu

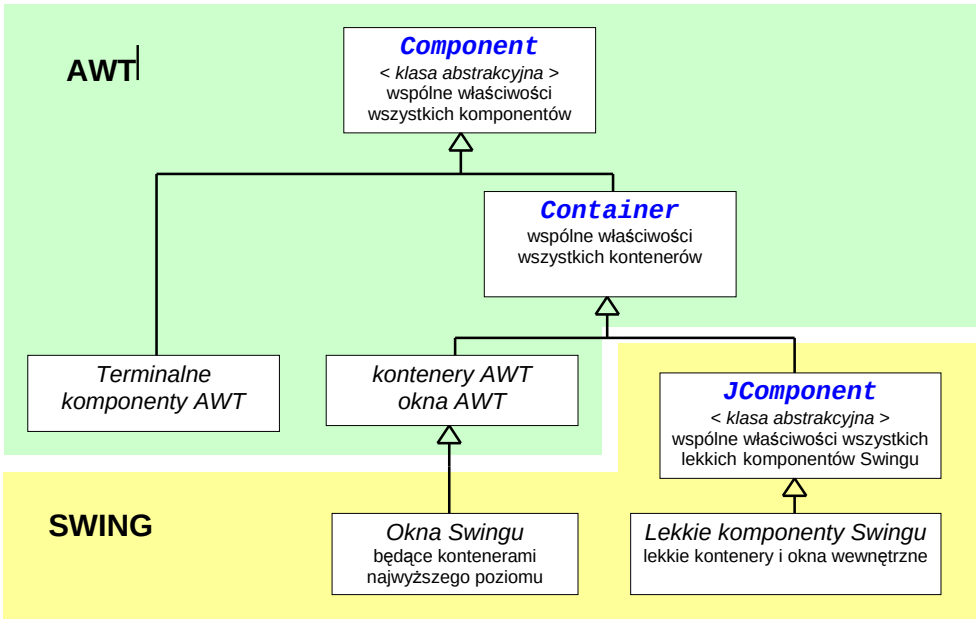
AWT (Abstract Windowing Toolkit) - zestaw klas definiujących proste komponenty interakcji wizualnej. Są to komponenty ciężkie – realizowane poprzez użycie graficznych bibliotek GUI systemu operacyjnego:

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

Pakiet Swing (JFC – Java Foundation Clases) - zestaw klas definiujących wiele komponentów i kontenerów interakcji wizualnej o zaawansowanych właściwościach. Są to w większości komponenty lekkie, które:

- są rysowane za pomocą kodu Javy w obszarze jakiegoś komponentu ciężkiego (zwykle kontenera najwyższego poziomu)
- mogą wygląd niezależny od platformy systemowej,
- mogą być przezroczyste, a zatem mogą przybierać wizualnie dowolne kształty,

Komponenty AWT a komponenty Swingu



Komponenty i kontenery

Elementy GUI dzielą się na **komponenty** (kontrolki, sterowniki) oraz **kontenery**, które są używane do przechowywania innych elementów.

komponenty:	kontenery:
➢ przyciski (`Button`, `JButton`) ➢ płótna (`Canvas`) ➢ pola wyboru (`Checkbox`, `JCheckbox`) ➢ etykiety (`Label`, `JLabel`) ➢ paski przewijania (`ScrollBar`, `JScrollBar`) ➢ listy (`List`, `JList`) ➢ listy rozwijane (`Choice`) ➢ polecenia menu (`MenuItem`, `JMenuItem`) ➢ komponenty tekstowe (`TextField` ,`TextArea`, `JTextField`, `JTextArea`)	➢ kontenery (`Container`) ➢ okna (`Window`, `JWindow`) ➢ ramki (`Frame`, `JFrame`) ➢ okna dialogowe (`Dialog`, `JDialog`) ➢ dialogi plikowe (`FileDialog`, `JFileChooser`) ➢ panele (`Panel`, `JPanel`) ➢ okna przewijalne (`ScrollPane`, `JScrollPane`)

Właściwości komponentów AWT i Swing

Wszystkie komponenty wywodzą się z abstrakcyjnej klasy `Component`, która definiuje m.in. metody ustalające właściwości komponentów:

- `getNNN()` – pobieranie właściwości komponentu
- `isNNN()` – sprawdzanie właściwości zerojedynkowych lub `boolean`
- `setNNN(...)` – ustalanie właściwości komponentu

gdzie `NNN` jest nazwą właściwości.

Właściwości komponentów AWT i Swing

Ważniejsze wspólne właściwości komponentów:

`is.../setVisible`	– sprawdza/ustawia widoczność komponentu,
`is.../setEnabled`	– sprawdza/ustawia dostępność komponentu,
`get.../setName`	– podaje/ustawia nazwę komponentu,
`get.../setFont`	– podaje/ustawia czcionkę,
`get.../setCursor`	– podaje/ustawia kursor dla komponentu,
`getParent`	– podaje kontener zawierający komponent,
`get.../setLocation`	– podaje/ustawia położenie górnego-lewego rogu,
`get.../setSize`	– podaje/ustawia rozmiar w pikselach,
`contains`	– sprawdza czy podany punkt zawiera się w polu komponentu.

Kontenery

Kontenery to komponenty, które mogą zawierać inne komponenty (w tym inne kontenery. Podstawowe metody (oprócz odziedziczonych z klasy **Component**) to:

add(<nazwa komponentu>**)** – dodawanie komponentu,

remove(<nazwa komponentu>**)** – usunięcie komponentu,

A ponadto:

getCopmonentCount() – zwraca liczbę komponentów,

getComponent(int n) – zwraca odniesienie na n-ty komponent,

getComponents() – zwraca tablicę wszystkich komponentów,

setLayout(...) – ustawia rozmieszczenie komponentów.

Okna

Okna to kontenery najwyższego poziomu, za pomocą których aplikacja komunikuje się z użytkownikiem.

- Najważniejsze komponenty, które tworzą okna to:
  JFrame, **JDialog**, **JWindow**, **JApplet**, **JInternalFrame**.
- Główne okno aplikacji jest obiektem klasy **JFrame**, np:
  ```
  JFrame okno = JFrame ("Okno główne");
  ```
- Każde okno, mimo że samo jest kontenerem, zawiera kontenery wewnętrzne. Do manipulowania komponentami w oknie służy kontener **contentPane**. Dostęp do tego kontenera umożliwia metoda **getContentPane()**, np:
  ```
  Container cp = okno.getContentPane();
  ```
- Kolejne komponenty umieszcza się w kontenerze przy użyciu metody **add(...)**.

Obiekt klasy JFrame – przykład

```
import javax.swing.*;

public class ProstaAplikacja1 extends JFrame
{
  public ProstaAplikacja1()
  { super("Ramka 1");

    setSize(250,50);

    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    setVisible(true);
  }

  public static void main(String []args)
  { new ProstaAplikacja1();
  }

}
```

Wywołania konstruktora klasy bazowej i przekazanie tytułu okna

Ustawienia rozmiaru okna ramki

Obsługa zamykania okna

Wyświetlenie ramki na ekranie

Operacja zamknięcia okna

Ramka posiada przyciski *Minimalizuj*, *Maksymalizuj* oraz *Zamknij* zlokalizowane na pasku tytułu. W przypadku Javy domyślne zamkniecie ramki nie powoduje zamknięcia aplikacji. Aby to zmienić, należy wywołać metodę **setDefaultCloseOperation()** zawierającą jeden z czterech parametrów wywołania:

- **EXIT_ON_CLOSE** – zamyka aplikacje po zamknięciu ramki
- **DISPOSE_ON_CLOSE** – zamyka ramkę, usuwa obiekt reprezentujący ramkę, ale pozostawia pracującą aplikacje,
- **DO_NOTHING_ON_CLOSE** – pozostawia ramkę otwarta i kontynuuje prace aplikacji
- **HIDE_ON_CLOSE** – zamyka ramkę pozostawiając pracującą aplikacje.

Umieszczanie komponentów w oknie

```
import java.awt.*;
import javax.swing.*;

public class ProstaAplikacja2 extends JFrame
{
  public ProstaAplikacja2()
  { super("Ramka 2");
    setSize(250,70);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    JPanel p = new JPanel();
    p.add( new JLabel("Etykieta")  );
    p.add( new JButton("Przycisk") );
    p.add( new JTextField("Pole textowe"));

    Container cp = getContentPane();
    cp.add(p);

    setVisible(true);
  }

  public static void main(String []args)
  { new ProstaAplikacja2();
  }
}
```

Rozmieszczanie komponentów w kontenerze

Z każdym kontenerem jest skojarzony tzw. **zarządca rozkładu**, który określa rozmiary i położenie komponentów. Zarządca rozkładu jest obiektem klasy implementującej interfejs ***LayoutManager***. Ustalenie zarządcy rozkładu dla kontenera odbywa się za pomocą metody ***setLayout(...)*** np.:

```
FlowLayout flow = new FlowLayot();
Frame f = new Frme();
f.setLayout(flow);
```

Użycie zarządcy rozkładu pozwala unikać konieczności oprogramowania zmian rozmiarów i położenia komponentów przy zmianie rozmiarów kontenera.

Zarządca rozkładu – przykłady

Ustalenie rozkładu realizowane jest przez klasy implementujące interfejs ***LayoutManager*** (zarządca rozkładu):

- ***FlowLayout*** – komponenty ułożone są wierszami (jak litery tekstu); rozmiary komponentów nie zmieniają się.
- ***BorderLayout*** – komponenty ułożone są „geograficznie”:
  - o „***North***”, „South” (komponenty zmieniają wymiary w poziomie),
  - o „***East***”, „West” (komponenty zmieniają wymiary w pionie),
  - o „***Center***” (komponent zmienia oba wymiary),
- ***GridLayout*** – tablica n*m komponentów; wszystkie komponenty mają taki sam rozmiar zmieniający się wraz z kontenerem.

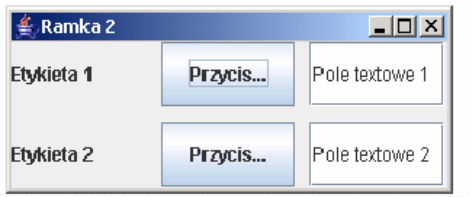
Zarządca rozkładu – przykłady c.d.

```
import java.awt.*;
import javax.swing.*;

public class ProstaAplikacja3 extends JFrame
{
  public ProstaAplikacja3()
  { super("Ramka 2");
    setSize(300,120);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    JPanel p = new JPanel();
    p.setLayout(new GridLayout(2, 3, 10, 10));
    p.add( new JLabel("Etykieta 1")  );
    p.add( new JButton("Przycisk 1") );
    p.add( new JTextField("Pole textowe 1"));

    p.add( new JLabel("Etykieta 2")  );
    p.add( new JButton("Przycisk 2") );
    p.add( new JTextField("Pole textowe 2"));

    Container cp = getContentPane();
    cp.add(p);

    setVisible(true);
  }

  public static void main(String []args)
  { new ProstaAplikacja3();
  }
}
```

Dodawanie zarządcy rozmieszczenia w postaci szachownicy 2 x 3.

Dodawanie pierwszego rzędu komponentów

Dodawanie drugiego rzędu komponentów



Zdarzenia

Graficzny interfejs użytkownika oprócz samego wyświetlania komponentów powinien reagować na zdarzenia pochodzące od użytkownika. Źródłami i słuchaczami zdarzeń są obiekty:

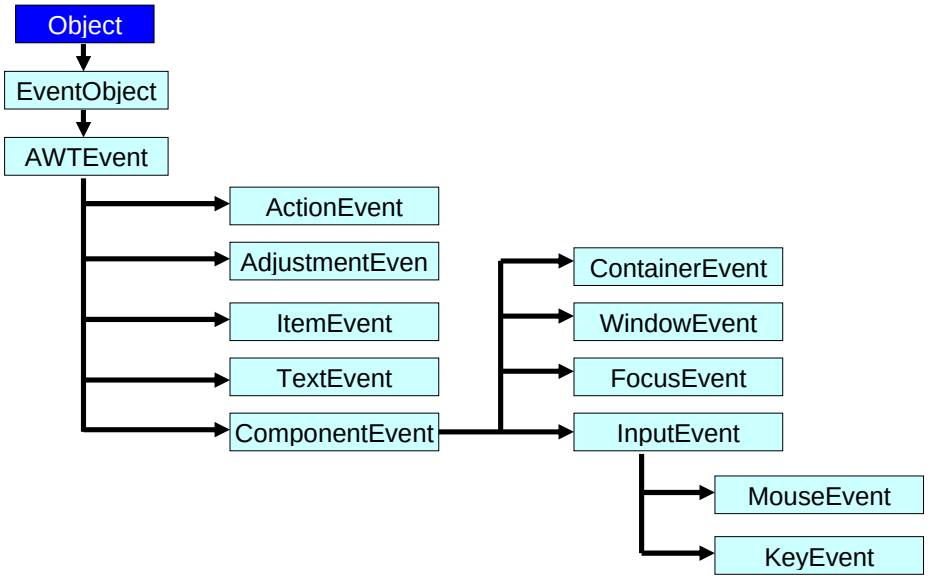
- **zdarzenie** (event) - obiekt "niosący" informację o stanie źródła,
- **źródło** (source) - obiekt, który generuje zdarzenia,
- **słuchacz** (listener) -obiekt powiadamiany o wystąpieniu zdarzenia.

Większość zdarzeń jest generowana przez:

- ● mysz,
- ● klawiaturę,
- ● elementy interfejsu graficznego.

Obsługa zdarzeń jest zawarta w pakiecie ***java.awt.event***.

Hierarcha zdarzeń



Tworzenie słuchacza

Słuchacz - to klasa, która może obsługiwać zdarzenie. Każda klasa, która implementuje interfejs nasłuchu staje się Słuchaczem, np.:

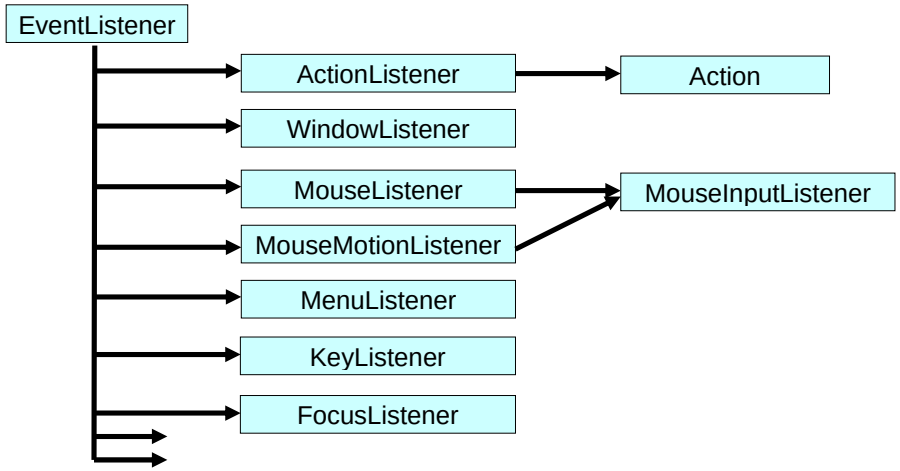
```
public class MojaKlasa implements ActionListener {
    // ...
}
```

Każda klasa implementująca interfejs musi zdefiniować metody interfejsu. Dla interfejsu ***ActionListener*** jest to:

```
public void actionPerformed(ActionEvent e) {
   // instrukcje obsługujące zdarzenie
}
```

Klasa-słuchacz może też implementować większą liczbę interfejsów nasłuchu (określamy w ten sposób zestaw obsługiwanych zdarzeń).

Interfejsy nasłuchu



Interfejsy nasłuchu

- **`ActionListener`** – obsługuje zdarzenia generowane przez użytkownika na rzecz danego składnika interfejsu (Np. klikniecie przycisku)
- **`AdjustmentListener`** – obsługuje zdarzenie jako zmianę stanu składnika (np. przesuwanie suwaka w polu tekstowym)
- **`FocusListener`** – obsługuje zdarzenie od przejścia składnika w stan nieaktywny
- **`ItemListener`** – obsługuje zdarzenie od np. zaznaczenia pola wyboru
- **`KeyListener`** – obsługuje zdarzenie np. od wpisywania tekstu z klawiatury
- **`MouseListener`** – obsługuje zdarzenie od naciśnięcia klawiszy myszy
- **`MouseMotionListener`** – obsługuje zdarzenie od przesuwania wskaźnika myszy nad danym składnikiem
- **`WindowListener`** – obsługuje zdarzenie od okna np. minimalizacja, maksymalizacja, przesuniecie, zamkniecie

Przyłączanie słuchacza

Wszystkie komponenty Swing umożliwiają przyłączanie/odłączanie określonych typów Słuchaczy. Służą do tego metody:

`addXXXListener()` oraz **`removeXXXListener()`**,

gdzie ***XXX*** jest typem słuchacza.

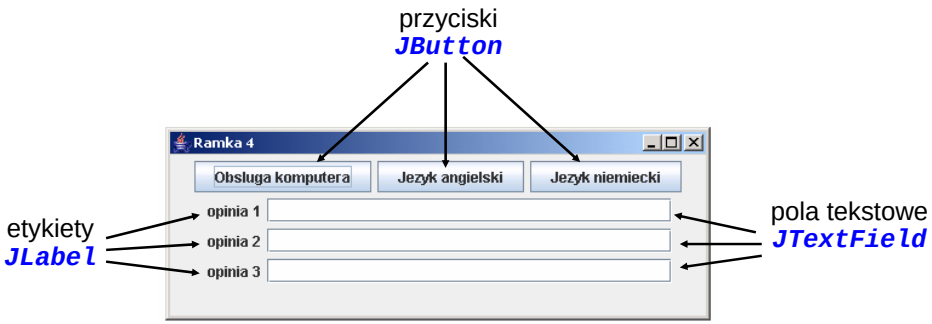
Np.: zdarzenie **`ActionEvent`** może być obsłużone przez Słuchacza implementującego interfejs **`ActionListener`**; Słuchacz taki może być przyłączony do komponentów, które mają dostęp do metody **`addActionListener()`**. Są to: **`Button`**, **`List`**, **`TextField`**, **`MenuItem`** oraz klasy pochodne.

Przykład:

```
Słuchacz słuchacz = new Słuchacz();
przyciskOK.addActionListener(słuchacz);
```


Obsługa zdarzeń od przycisków - przykład

Po kliknięciu w przycisk pole tekstowe ma zostać wypełnione tekstem.



Obsługa zdarzeń od przycisków – przykład cd.

```
import javax.swing.*;
import java.util.*;
import java.io.*;
import java.lang.*;
import java.awt.event.*;



class Dane
{ String opinia1,opinia2,opinia3;
  Dane()
   { opinia1=opinia2=opinia3=null;
   }
}

public class ProstaAplikacja4 extends JFrame implements ActionListener
{
  JButton weopinia1 = new JButton("Obsluga komputera");
  JButton weopinia2 = new JButton("Jezyk angielski");
  JButton weopinia3 = new JButton("Jezyk niemiecki");

  JTextField wyopinia1=new JTextField(30);
  JTextField wyopinia2=new JTextField(30);
  JTextField wyopinia3=new JTextField(30);

  Dane dana = new Dane();
```

Klasa do przechowywania tekstu wyrażanych opinii

klasa implementuje interfejs słuchacza zdarzeń `ActionListener`

przyciski `JButton` oraz pola tekstowe `JTextField` – aktywne elementy interfejsu użytkownika

obiekt pamiętający wprowadzone dane

Obsługa zdarzeń od przycisków – przykład cd.

```
public ProstaAplikacja4()
 { super("Ramka 4");
   setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   setSize(450,160);
   JPanel panel=new JPanel();

   weopinia1.addActionListener(this);
   weopinia2.addActionListener(this);
   weopinia3.addActionListener(this);

   panel.add(weopinia1);
   panel.add(weopinia2);
   panel.add(weopinia3);

   JLabel eopinia1= new JLabel(" opinia 1",SwingConstants.RIGHT);
   panel.add(eopinia1);
   panel.add(wyopinia1);
   JLabel eopinia2= new JLabel(" opinia 2",SwingConstants.RIGHT);
   panel.add(eopinia2);
   panel.add(wyopinia2);
   JLabel eopinia3= new JLabel(" opinia 3",SwingConstants.RIGHT);
   panel.add(eopinia3);
   panel.add(wyopinia3);

   setContentPane(panel);
   show();
 }
```

Przyłączenie słuchacza zdarzeń generowanych przez przyciski

Umieszczenie wszystkich komponentów w kontenerze ***panel***

Umieszczenie kontenera ***panel*** w głównym oknie aplikacji oraz wyświetlenie głównego okna na ekranie

Obsługa zdarzeń od przycisków – przykład cd.

```
public void actionPerformed (ActionEvent evt)
 {
   Object zrodlo = evt.getSource();

   if (zrodlo==weopinia1)
         dana.opinia1= new String("Znajomosc obslugi komputera");
   else if (zrodlo==weopinia2)
         dana.opinia2= new String("Znajomosc jezyka angielskiego");
   else if (zrodlo==weopinia3)
         dana.opinia3= new String("Znajomosc jezyka niemieckiego");

   wyopinia1.setText(dana.opinia1);
   wyopinia2.setText(dana.opinia2);
   wyopinia3.setText(dana.opinia3);

   repaint();
 }

public static void main(String[] arg) throws Exception
 { new ProstaAplikacja4();
 }
}
```

Metoda z interfejsu słuchacza ***ActionListener***, która obsługuje zdarzenia ***ActionEvent*** generowane przez przyciski

Odczytanie źródła wygenerowanego zdarzenia

utworzenie tekstu opinii po naciśnięciu przycisku

wpisanie tekstów opinii do pól tekstowych

odrysowanie wyglądu wszystkich komponentów po modyfikacji danych

utworzenie głównego okna aplikacji