
POLITECHNIKA GDAŃSKA

# Język Java podstawy

Jacek Rumiński

# Język Java podstawy

Jacek Rumiński

Katedra Inżynierii Biomedycznej,
Wydział Elektroniki, Telekomunikacji i Informatyki
Politechnika Gdańska

1. Obsługa zdarzeń

2. Rysowanie w Javie

## Co to jest zdarzenie i jak jest modelowane?

Zdarzenie to sytuacja, która wystąpiła (i ewentualnie trwała) w określonym czasie wywołana działaniem użytkownika lub działaniem (innego) programu.

Przykładowe zdarzenia:

- wciśnięto klawisz klawiatury
- wciśnięto przycisk myszki
- poruszono myszką
- wywołano zamknięcie okna
- wciśnięto przycisk interfejsu graficznego

## Co to jest zdarzenie i jak jest modelowane?

Zdarzenie można modelować za pomocą klas.

Klasą bazową zdarzeń w Javie będzie klasa Event. Twórcy biblioteki Javy zaprojektowali szereg klas dziedziczących po klasie Event, które modelują konkretne zdarzenia:

- wciśnięto klawisz klawiatury -> KeyEvent
- wciśnięto przycisk myszki -> MouseEvent
- poruszono myszką -> MouseMotionEvent
- wywołano zamknięcie okna -> WindowEvent
- wciśnięto przycisk interfejsu graficznego ->ActionEvent

## Obsługa zdarzeń

Zdarzenia związane są również z określonym źródłem (ang. source), którym najczęściej jest określony komponent graficzny. Przykładowo przycisk (obiekt klasy Button) jest źródłem możliwych zdarzeń związanych z wciśnięciem tego przycisku.
Ale to, że zdarzenia powstają to na razie nic nowego - w naszych wcześniejszych aplikacjach wciskaliśmy przyciski (powstawały zdarzenia), ale nic się nie działo!

Co zrobić, żeby coś się działo? Trzeba obsłużyć zdarzenie. Oznacza to, że należy napisać taki zbiór instrukcji (kod obsługi zdarzenia), który będzie wykonany w odpowiedzi na powstające zdarzenie określonego typu.

Jak powiązać kod obsługi zdarzenia z określonymi zdarzeniami? W tym celu należy kod obsługi zdarzenia zapisać w ciele określonej metody, którą zaprojektowali twórcy bibliotek Javy. Metody te są zadeklarowane w odpowiednio nazwanych interfejsach (w pełni abstrakcyjne klasy). Interfejsy te nazywane są zwykle **NNNListener**, gdzie **NNN** to słowo (słowa) określające zwykle rodzaj (źródła) zdarzenia, np. MouseListener, KeyListener, WindowListener, itd. Potem pozostaje już tylko powiązać źródła zdarzeń z ich obsługą.

# Budowa GUI: obsługa zdarzeń

Źródło zdarzenia
(np. Button b)

Rejestruj obiekt nasłuchujący
(np. b.addActionListener(SLUCHACZ))

Wzorzec obsługi zdarzenia (np. jeśli wskaźnik myszki był na komponencie i wówczas wciśnięto przycisk myszki to stwórz nowy obiekt zdarzenia (np. ActionEvent) i umieść go w kolejce zdarzeń.

Obiekt SLUCHACZ musi implementować określony interfejs – czyli podać kod obsługi zdarzenia określonego typu, np.

```
implements ActionListener {
   actionPerformed(){ TU KOD OBSŁUGI};
}
```

Z3 (WindowEvent) → Z2 (MouseEvent) → Z3 (ActionEvent)

```
import java.awt.*; import javax.swing.*;
import java.awt.event.*; //Koniecznie należy pamiętać o importowaniu pakietu event
public class ZdarzeniaJedi extends JFrame {
  //JTextField jako pole obiektu, bo tf musi być widoczne w metodzie obsługi zdarzeń
  private JTextField tf;
  public void init() {
                JButton bJEDI, bSITH;
                tf = new JTextField(50);
                tf.setFont(new Font(Font.DIALOG, Font.BOLD, 20));
                add(tf, BorderLayout.NORTH);
                JPanel jp=new JPanel();
                bJEDI = new JButton("JEDI");
                bSITH = new JButton("SITH");
                jp.add(bJEDI);
                jp.add(Box.createRigidArea(new Dimension(50,0))); //wstaw przerwę
                jp.add(bSITH);
                add(jp,BorderLayout.CENTER);
                //dodajemy obiekty nasłuchujące zdarzenia
                bJEDI.addActionListener(new Postac("Jedi"));
                bSITH.addActionListener(new Postac("Sith"));
  } // koniec public void init()
```

c.d.n.

```
 public static void main(String []a){
            ZdarzeniaJedi zj=new ZdarzeniaJedi();
            zj.init();
            zj.setSize(500,150);
            zj.setVisible(true);
 } //koniec main()
/* Klasa wewnętrzna - ma dostęp do pól i metod klasy zewnętrznej;
  * implementuje interfejs ActionListener. Twórcy klasy JButton
  * przewidzieli obsługę zdarzenia przyciśnięcia przycisku poprzez
  * przygotowanie metody actionPerformed() w interfejsie ActionListener  */
 class Postac implements ActionListener {
            String rodzaj;
            Postac(String s){
                        rodzaj=s;
            } //koniec Postac()
   public void actionPerformed(ActionEvent e) {
     tf.setText(rodzaj);
   } //koniec actionPerformed()
 } // koniec class
}//koniec class ZdarzeniaJedi
```

## Budowa GUI: obsługa zdarzeń

Realizując obsługę zdarzeń musimy rozważyć:

1. Jaki rodzaj zdarzenia może wystąpić – czyli obiekt jakiej klasy zostanie utworzony.
2. Jaki interfejs zdarzeń chcemy implementować oraz jakie konkretne zdarzenia związane są z metodami w danym interfejsie. W dokumentacji Javy (klas Java API) możemy zobaczyć jaką metodę **addNNNListener** możemy wykorzystać dla danego komponentu. Znając nazwę interfejsu "nasłuchiwacza" (**Listener**) można uzyskać szczegółowy opis metod, np.:

| | |
|---|---|
| **KeyListener** | **keyPressed(KeyEvent)** |
| **KeyAdapter** | **keyReleased(KeyEvent)** |
| | **keyTyped(KeyEvent)** |
| **MouseListener** | **mouseClicked(MouseEvent)** |
| **MouseAdapter** | **mouseEntered(MouseEvent)** |
| | **mouseExited(MouseEvent)** |
| | **mousePressed(MouseEvent)** |
| | **mouseReleased(MouseEvent)** |

```
import java.awt.*;
import java.awt.event.*;
class Ekran extends Canvas{ //Canvas - pole graficzne
  public String s="Witam"; //początkowa wartość pola
            private Font f;
            Ekran (){
            super();    f = new Font("Times New Roman",Font.BOLD,16);//ustaw czcionkę
            setFont(f);
            addKeyListener(new KeyAdapter(){
                        public void keyPressed(KeyEvent ke){//wciśnięto przycisk klawiatury
                          s=new String(ke.paramString());//parametry zdarzenia jako String
                          repaint();//wywołaj metodę paint() - czyli wyświetl nową wartość s
                        }            });
            addMouseListener(new MouseAdapter(){
                        public void mousePressed(MouseEvent me){//wciśnięto przycisk myszki
                          s=new String(me.paramString());//parametry zdarzenia jako String
                          repaint();//wywołaj metodę paint() - czyli wyświetl nową wartość s
                        }            });
  }//koniec Ekran()
  public void paint(Graphics g){          g.drawString(s,10,180);          }//koniec paint()
}// koniec class Ekran
```

c.d.n.

```
public class KomunikatorJedi extends Frame  {
        KomunikatorJedi (String nazwa){
                super(nazwa);
        }//koniec KomunikatorJedi()
        public static void main(String args[]){
                KomunikatorJedi okno = new KomunikatorJedi("Komunikator");
                okno.setSize(1000,400);
                Ekran e = new Ekran();
                okno.add(e);
                okno.addWindowListener(new WindowAdapter(){
                        public void windowClosing(WindowEvent e){
                                System.out.println("Dziękujemy za pracę...");
                                System.exit(0);
                        }
                 });
                okno.setVisible(true);
        }//koniec main()
}// koniec public class KomunikatorJedi
```

# DEMO

1. Obsługa zdarzeń

2. Rysowanie w Javie

Pakiet AWT zarówno w wersjach wcześniejszych jak i w wersji 2 wyposażony jest w klasę Graphics, a od wersji 2 dodatkowo w klasę Graphics2D. Klasy te zawierają liczne metody umożliwiające tworzenie i zarządzanie grafiką w Javie.

W bibliotece AWT obiekt klasy Graphics jest dostarczany do komponentu poprzez następujące metody:

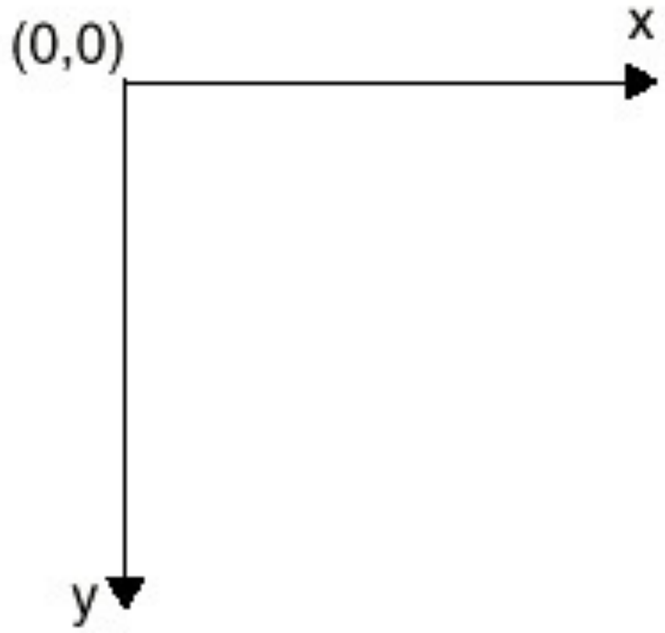
- paint,
- paintAll,
- update,
- print,
- printAll,
- getGraphics.

Obiekt graficzny (kontekst) zawiera informacje o stanie grafiki potrzebne dla podstawowych operacji wykonywanych przez metody Javy.

Zaliczyć tu należy następujące informacje:
- obiekt komponentu, który będzie obsługiwany,
- współrzędne obszaru rysowania oraz obcinania,
- aktualny kolor,
- aktualne czcionki,
- aktualna funkcja operacji na pikselach logicznych (XOR lub Paint),
-aktualny kolor dla operacji XOR.

Posiadając obiekt graficzny można wykonać szereg operacji rysowania np.: Graphics g;
g.drawLine(int x1, int y1, int x2, int y2) - rysuje linię,
g.drawRect(int x, int y, int width, int height) - rysuje prostokąt,
g.drawString(String str, int x, int y) - rysuje tekst,
g.drawImage(Image img, int x, int y, Color bgcolor, ImageObserver observer) - wyświetla obraz

W celu rysowania elementów grafiki konieczna jest znajomość układu współrzędnych w ramach, którego wyznacza się współrzędne rysowania. Podstawowym układem współrzędnych w Javie jest układ użytkownika, będący pewną abstrakcją układów współrzędnych dla wszystkich możliwych urządzeń. Układ użytkownika definiowany jest w sposób następujący.



Pierwotne wersje AWT definiują kilka obiektów geometrii jak np. Point, Rectangle. Elementy te są bardzo przydatne dlatego, że, co jest właściwe dla języków obiektowych, nie definiujemy za każdym razem prostokąta za pomocą atrybutów opisowych (współrzędnych) lecz przez gotowy obiekt - prostokąt, dla którego znane są (różne metody) jego liczne właściwości:

```
Point p = new Point(x,y);
Rectangle r = new Rectangle(x,y,width,height);
```

Przykładowo metoda translate():

```
    Insets insets = getInsets();
    g.translate (insets.left, insets.top);
```

zmienia początek układu współrzędnych przesuwając go do aktywnego pola graficznego (bez ramek).

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

class PoleGraficzne extends Canvas{
    public void paint (Graphics g) {
        g.drawLine (5, 5, 195, 5);
        g.drawLine (5, 75, 5, 75);
        g.drawRect (25, 10, 50, 75);
        g.setColor(Color.blue);
        g.fillRect (25, 110, 50, 75);
        g.setColor(Color.black);
        g.drawRoundRect (100, 10, 50, 75, 60, 50);
        g.fillRoundRect (100, 110, 50, 75, 60, 50);
        g.setColor(Color.red);
        g.setFont(new Font("Dialog",Font.BOLD,20));
        g.drawString ("Test grafiki",30, 105);
        g.setColor(Color.black);
   }//koniec paint()
}//koniec class PoleGraficzne
```

```
public class RysunkiJedi extends JFrame {
   RysunkiJedi () {
      super ("Rysunki");
      add(new PoleGraficzne());
      setSize(200, 220);
   }//koniec RysunkiJedi()
   public static void main (String [] args) {
            RysunkiJedi r = new RysunkiJedi ();
            r.setVisible(true);
            r.addWindowListener(new WindowAdapter(){
                              public void windowClosing(WindowEvent e){
                              System.out.println("Dziekujemy za prace z programem...");
                              System.exit(0);
                              }
            });

   }//koniec main()
}// koniec public class Rysunki extends Frame
```


Jacek Rumiński

# Zapraszamy na kolejne zajęcia