

# GUI v Javě a událostmi řízené programování

Jan Faigl

Katedra počítačů

Fakulta elektrotechnická

České vysoké učení technické v Praze

Přednáška 4

**A0B36PR2 – Programování 2**

# Obsah přednášky

GUI v Javě (připomínka)

Návrhář GUI

Příklad aplikace

MVC – Model-View-Controller

Události

Vnitřní třídy

## Základní prvky grafického rozhraní

- **Komponenty** – tlačítka, textová pole, menu, posuvníky, ...
- **Kontejnery** – komponenty, do kterých lze vkládat komponenty  
*Například pro rozdělení plochy a volbu rozmístění*
- **Správce rozvržení** (*Layout manager*) – rozmísťuje komponenty v ploše kontejneru
- Interakce s uživatelem dále zpravidla vyžaduje mechanismus událostí a jejich zachytávání

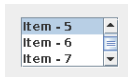
## Swing Toolkit

<http://docs.oracle.com/javase/tutorial/uiswing>

# Základní komponenty

## ■ **Komponenty** a dialogové prvky

- Tlačítka, text, textová pole, seznamy, přepínače



javax.swing

## ■ **Kontejnery** (v oknech, která zpravidla řeší prostředí OS)

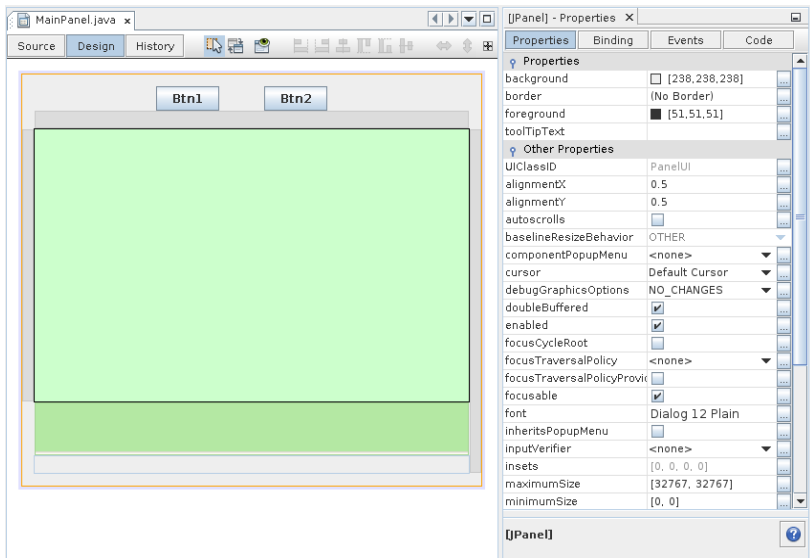
javax.swing

- Komponenty obsahují komponenty

*Komponenty musí být umístěny v kontejneru*

- Kontejnery se vkládají do oken
- **JFrame** – obecný kontejner
- **JPanel** – kontejner pro jednoduché komponenty

## „Návrhář formulářů“



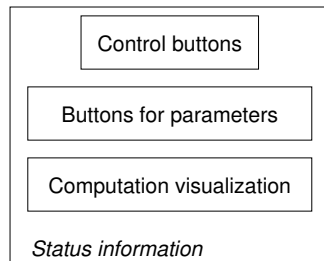
The screenshot displays the NetBeans IDE's GUI Designer. The main window, titled "MainPanel.java", is in Design mode. It features a light gray title bar with two buttons labeled "Btn1" and "Btn2". Below the buttons is a large green rectangular area, and at the bottom is a thin green horizontal bar. The Properties window on the right, titled "[JPanel] - Properties", shows the configuration for the selected component. The Properties tab is active, displaying a list of properties and their values.

Property	Value
background	[238, 238, 238]
border	(No Border)
foreground	[51, 51, 51]
toolTipText	
<b>Other Properties</b>	
UIClassID	PanelUI
alignmentX	0.5
alignmentY	0.5
autoscrolls	<input type="checkbox"/>
baselineResizeBehavior	OTHER
componentPopupMenu	<none>
cursor	Default Cursor
debugGraphicsOptions	NO_CHANGES
doubleBuffered	<input checked="" type="checkbox"/>
enabled	<input checked="" type="checkbox"/>
focusCycleRoot	<input type="checkbox"/>
focusTraversalPolicy	<none>
focusTraversalPolicyProvided	<input type="checkbox"/>
focusable	<input checked="" type="checkbox"/>
font	Dialog 12 Plain
inheritsPopupMenu	<input type="checkbox"/>
inputVerifier	<none>
insets	[0, 0, 0, 0]
maximumSize	[32767, 32767]
minimumSize	[0, 0]

## Příklad návrhu aplikace – BarComp

*Naším cílem je vytvořit jednoduchou aplikaci s dvěma sadami tlačítek pro ovládání výpočtu s vizualizací postupu výpočtu a stavu aplikace.*

- Aplikace má 4 základní komponenty
  1. Hlavní ovládací tlačítka
  2. Tlačítka pro nastavení
  3. „progress bar”
  4. Stavový řádek



*Aplikaci použijeme pro demonstraci zpracování událostí a ukázkou dílčích konceptů.*

## Struktura aplikace – BarComp

- Aplikace se skládá z výpočetního modelu `Model`, grafických komponent `MyBarPanel` a spouštěcí třídy `DemoBarComp`

```
public class DemoBarComp {  
  
    void start() { ... }  
  
    public static void main(String[] args) {  
        DemoBarComp demo = new DemoBarComp();  
        demo.start();  
  
    }  
}
```

lec04/DemoBarComp

## Struktura aplikace – DemoBarComp – start

```
void start() {  
    JFrame frame = new JFrame("PR2 - lec04 - Demo  
    Progress Bar of the Computation");  
    frame.setDefaultCloseOperation(JFrame.  
    EXIT_ON_CLOSE);  
    frame.setMinimumSize(new Dimension(480, 240));  
  
    MyBarPanel myBarPanel = new MyBarPanel();  
  
    frame.getContentPane().add(myBarPanel);  
    frame.pack();  
    frame.setVisible(true);  
  
    myBarPanel.setComputation(new Model());  
}
```

lec04/DemoBarComp



## MyBarPanel – start

```
public class MyBarPanel extends JPanel {
    JTextField status;
    JProgressBar bar;

    Model computation;

    public MyBarPanel() {
        computation = null;
        createComponents();
    }
    public void setComputation(Model computation) {
        this.computation = computation;
    }
    private void createComponents() { ... }
}
```

lec04/MyBarPanel

## MyBarPanel – createComponents

```
private void createComponents() {  
    // 1st row of the control buttons  
    JPanel controlButtonsPanel = new JPanel();  
    createControlButtons(controlButtonsPanel);  
  
    // 2nd row of the buttons  
    JPanel buttonsPanel = new JPanel();  
    createButtons(buttonsPanel);  
  
    // 3rd row with the progress bar  
    bar = new JProgressBar(0, 100); // 0-100%  
    JPanel progressPanel = new JPanel();  
    createProgress(progressPanel, bar);  
  
    // 4th row with the status bar  
    status = createStatusBar("Waiting for your commands");  
  
    // Set layout and add the rows  
    setLayout(new BorderLayout(this, BorderLayout.Y_AXIS));  
    add(controlButtonsPanel);  
    add(buttonsPanel);  
    add(progressPanel);  
    add(status);  
}
```

## MyBarPanel – createControlButtons

```
private JPanel createControlButtons(JPanel panel) {  
    JButton btnCompute = new JButton("Compute");  
    JButton btnStop = new JButton("Stop");  
    JButton btnQuit = new JButton("Quit");  
  
    panel.add(btnCompute);  
    panel.add(btnStop);  
    panel.add(btnQuit);  
    return panel;  
}
```

lec04/MyBarPanel

## MyBarPanel – createButtons

```
private JPanel createControlButtons(JPanel panel) {  
    JButton btnCompute = new JButton("Compute");  
    JButton btnStop = new JButton("Stop");  
    JButton btnQuit = new JButton("Quit");  
  
    btnQuit.addActionListener(new ActionListener() {  
        @Override  
        public void actionPerformed(ActionEvent e) {  
            System.out.println("Force quit");  
            System.exit(0);  
        }  
    });  
    panel.add(btnCompute);  
    panel.add(btnStop);  
    panel.add(btnQuit);  
    return panel;  
}
```

lec04/MyBarPanel

## MyBarPanel – createProgress

```
private JPanel createProgress(JPanel panel,
    JProgressBar progress) {
    TitledBorder border = BorderFactory.
        createTitledBorder("Computations");
    panel.setBorder(border);
    panel.add(progress);
    return panel;
}
```

lec04/MyBarPanel

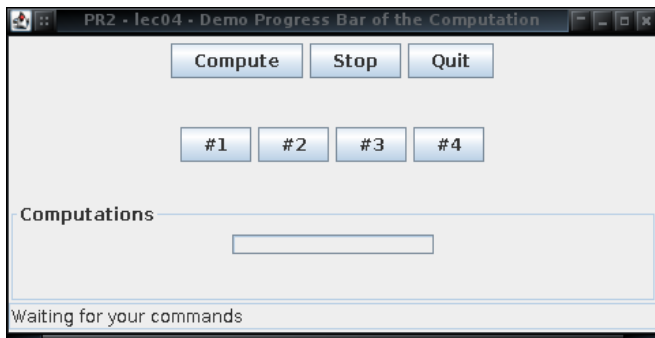
## MyBarPanel – createStatusBar

```
private JTextField createStatusBar(String initMessage
    ) {
    JTextField statusBar = new JTextField();
    statusBar.setEditable(false);
    statusBar.setText(initMessage);
    statusBar.setHorizontalAlignment(JTextField.LEFT);
    statusBar.setMaximumSize(
        new Dimension(
            Integer.MAX_VALUE,
            statusBar.getPreferredSize() .height
        ));

    return statusBar;
}
```

lec04/MyBarPanel

## MyBarPanel – grafické rozhraní

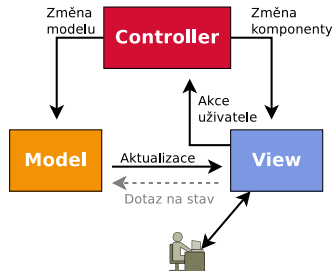


Pro „oživení“ tlačítek musíme vytvořit reakce na události a propojit grafické rozhraní s modelem výpočetní části.

# Model-View-Controller (MVC)

- Architektura pro návrh aplikace s uživatelským rozhraním

- Rozděluje aplikaci na tři základní **nezávislé** komponenty
  - Datový model aplikace
  - Uživatelské rozhraní
  - Řídící logika



- **Nezávislé** – ve smyslu, že změna některé komponenty má minimální vliv na komponenty ostatní

<http://www.oracle.com/technetwork/articles/javase/index-142890.html>

youtube – Elementary Model View Controller (MVC) by Example

[https://www.youtube.com/watch?v=LiBdzE\\_DJn4](https://www.youtube.com/watch?v=LiBdzE_DJn4)



## MVC – Obecný princip

- **Model** – datová reprezentace, se kterou aplikace pracuje
- **View** (pohled)– zajišťuje vizualizaci dat aplikace do podoby vhodné k prezentaci
- **Controller** (řadič) – zajišťuje změny dat nebo vizualizace na základě událostí (typicky od uživatele)

## Oddělení modelu od vizualizace je klíčové

*Umožňuje sdílení kódu a jeho snadnou údržbu*

## MVC – Příklad průběhu

1. Uživatel stiskne tlačítko v GUI
2. Řadič (controller) je informován o události
3. Řadič provede příslušnou akci a přistoupí k modelu, který modifikuje
4. Model zpracuje požadavek od řadiče
5. Pohled (view) provede zobrazení aktualizovaného modelu

*Např. použitím návrhového vzoru **Observer** nebo notifikací od řadiče.*

6. Uživatelské rozhraní čeká na další akci uživatele

## Zpracování událostí

- Událost je **objekt**, který vznikne změnou stavu zdroje
  - Důsledek interakce uživatele s řídicími elementy GUI
- Událost vznikne
  - kliknutím na tlačítko
  - stiskem klávesy
  - posunem kurzoru (myši)
- Události jsou produkovány tzv. **producenty** což jsou
  - tlačítka, rámy, grafické prvky
- Na události reagují posluchači událostí což jsou metody schopné zpracovat událost

*Posluchači se registrují u producentů pro odběr zpráv*

**Java obsahuje promyšlený a konzistentní koncept vzniku a zpracování událostí**

# Sekvenční vs událostmi řízené programování

- **Sekvenční** programování – kód je vykonáván postupně dle zadaného pořadí
  - Program začíná voláním `main` a pokračuje sekvenčně podle větvení v řídicích strukturách (`if`, `while`, ...)
  - Uživatelský vstup blokuje aplikaci dokud není zadán
  - Neumožňuje čekat na vstup z více zdrojů (např. klávesnice a myši)
- **Událostmi řízené programování** (`Event-driven programming`) – kód je vykonáván na základě aktivace nějakou událostí
  - Systém čeká na akci uživatele
  - Událost spouští odpovídající akci
  - Událostmi řízené programování řeší
    - Jak současně čekat na události z více zdrojů
    - Co dělat pro konkrétní událost

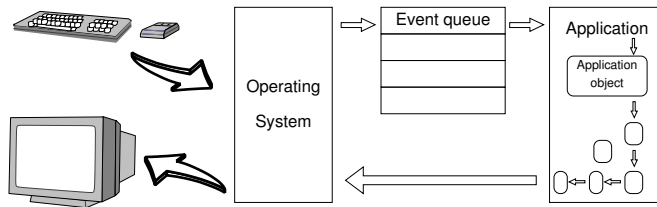
## Event-driven programming – základní koncept

- Základní koncept je postaven na frontě zpráv
- Operační systém spolu se správcem oken zpracovává vstupní události z připojených zařízení

*Pohyb myši, stisk klávesy*

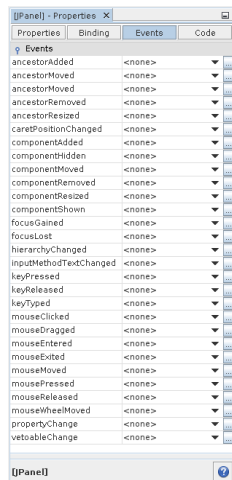
- Správce oken identifikuje příslušné okno a aplikace, které patří událost a přešle ji do aplikace

*Aplikace (Swing) používá podobný mechanismus pro identifikaci, která komponenta obdrží příslušnou zprávu*



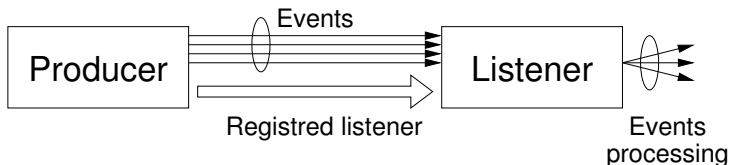
## Zpracování událostí – koncepce

- Informace o události (zdroj události, poloha kurzoru, atd.) jsou shromážděny v objektu jehož třída určuje charakter události:
  - `ActionEvent` – událost generovaná tlačítkem
  - `WindowEvent` – událost generovaná oknem
  - `MouseEvent` – událost generovaná myší
- Všechny třídy událostí jsou následovníky třídy `ActionEvent` a jsou umístěny v balíku `java.awt.event`.



## Základní princip zpracování události

- Události jsou **generovány zdroji** událostí
  - Jsou to **objekty** nesoucí informaci o události
- Události jsou **přijímány** ke zpracování **posluchači** událostí
  - **Objekty** tříd s metodami schopnými událost zpracovat
- Zdroj události rozhoduje o tom, který posluchač má reagovat
  - Registruje si svého posluchače



## Model šíření událostí

- Události jsou předávány posluchačům, které **nejprve musí producent zaregistrovat**
  - Například `addActionListener()`, `addWindowListener()`, `addMouseListener()`
  - Producent vysílá událost **jen těm posluchačům, které si sám zaregistroval**
- Posluchač musí implementovat některé z **rozhraní** posluchačů (tj. schopnost naslouchat)
  - `ActionListener`, `WindowListener`, `MouseListener`
- Zatímco událost **producenta** je typicky objekt některé knihovní třídy (např. tlačítka), **posluchač** je objekt, jehož třída je deklarována v aplikaci
  - Registrace metodou `add*?Listener()`
  - Registrovaná třída musí implementovat rozhraní `*?Listener`



## Příklad posluchače

- Registrujeme obsluhu události tlačítka #3
- Využijeme k tomu anonymní vnitřní třídu (odvozenou od `ActionListener`)
- Třidu (objekt) posluchače registrujeme metodou `addActionListener`

```
btn3.addActionListener(new ActionListener() {  
    @Override  
    public void actionPerformed(ActionEvent e) {  
        btn3.setText("clicked");  
    }  
});
```

*Musíme implementovat všechny metody abstraktní třídy. S výhodou můžeme využít automatického generování vývojového prostředí.*

## Příklad posluchače jako vnitřní třídy

```
public class MyBarPanel extends JPanel {
    ...
    private class SimpleButtonListener implements
        ActionListener {
        final String msg;

        public SimpleButtonListener(String msg) {
            this.msg = msg;
        }
        @Override
        public void actionPerformed(ActionEvent e) {
            status.setText(msg);
        }
    }
    ...
}
```

## Příklad – Třidu posluchače můžeme instancovat vícekrát

```
private JPanel createButtons(JPanel panel) {  
    ...  
    btn1.addActionListener(new SimpleButtonListener(  
        "Button #1 pressed"));  
    btn2.addActionListener(new SimpleButtonListener(  
        "Button #2"));  
    ...  
}
```

## Implementace modelu událostí

- Posluchač události musí implementovat příslušné rozhraní
  - Implementovat příslušné **abstraktní metody** rozhraní
- Pro **každý druh události je definována abstraktní metoda handler**, která událost ošetřuje
  - `actionPerformed`, `mouseClicked`, `windowClosing`, ...
- *Handlery* jsou deklarovány v rozhraní – posluchači
  - `ActionListener`, `MouseListener`, `WindowListener`, ...
- Předání události posluchači ve skutečnosti znamená vyvolání činnosti handleru,
  - Objekt události je předán jako skutečný parametr handleru

## Registrace posluchače

- Producent registruje posluchače zavoláním registrační metody:
  - `addActionListener`, `addMouseListener`,  
`addWindowListener`, ...
- Vazba mezi producentem a posluchačem je vztah N:M
  - Jeden posluchač může být registrován u více producentů
  - U jednoho producenta může být registrováno více posluchačů
- Událost se předá všem posluchačům, avšak pořadí zpracování není zaručeno

## Příklad – Posluchač může mít svůj vlastní stav

```
private class ToggleButtonListener implements
    ActionListener {
    final String msg;
    boolean state;
    public ToggleButtonListener(String msg) {
        this.msg = msg;
        state = false;
    }
    @Override
    public void actionPerformed(ActionEvent e) {
        state = !state;
        status.setText(
            msg + " " + (state ? "On" : "Off")
        );
    }
}
```

*MVC?*

## Příklad – Zdroj může mít více posluchačů

```
...  
btn1.addActionListener(new SimpleButtonListener(  
    "Button #1 pressed"));  
btn2.addActionListener(new ToggleButtonListener(  
    "Button #2"));  
  
ButtonListener buttonListener = new ButtonListener();  
  
btn1.addActionListener(buttonListener);  
btn4.addActionListener(buttonListener);  
...
```

- Událost se předá všem posluchačům, pořadí však není zaručeno

## Příklad – Více zdrojů téže události a jeden posluchač

```
private class ButtonListener implements ActionListener {
    int count = 0;
    @Override
    public void actionPerformed(ActionEvent e) {
        count += 1;
        JButton btn = (JButton) e.getSource();
        System.out.println("BtnLst: event: " + e);
        System.out.println("BtnLst e.getSource: "
            + e.getSource());

        System.out.println("ActionCommand: " +
            e.getActionCommand());

        status.setText("BtnLst: received new event " +
            count + " from " + btn.getText());
    }
}
...
ButtonListener buttonListener = new ButtonListener();
btn1.addActionListener(buttonListener);
btn4.addActionListener(buttonListener);
...
```

*Zdroj události můžeme rozlišit podle textu nebo podle objektu (přetypování).  
Výhodnější je však vytvořit individuální posluchače.*



## Události myši

```
progress.addMouseListener(new MouseListener() {
    @Override
    public void mouseEntered(MouseEvent e) {
        border.setTitle("Mouse entered to the bar area");
        panel.repaint(); // force update titledborder
    }
    @Override
    public void mouseExited(MouseEvent e) {
        border.setTitle("Computations");
        panel.repaint(); // force update titledborder
    }
    @Override
    public void mouseClicked(MouseEvent e) { }
    @Override
    public void mousePressed(MouseEvent e) { }
    @Override
    public void mouseReleased(MouseEvent e) { }
});
```

- Události pohybu myši lze naslouchat prostřednictvím rozhraní `MouseMotionListener` s vlastnostmi
  - `mouseDragged` a `mouseMoved`

## Vnitřní třídy

- Logické seskupení tříd, které se používají jen v jednom konkrétním místě
  - Třídy posluchačů jsou využitelné pro producenty v GUI
  - Efektivita kódu
  - Princip „pomocné“ třídy
- Princip zapouzdření (třída B je vnitřní třídou vnější třídy A)
  - Třída B má přístup ke všem členům třídy A, které však mají být nepřístupné jiným třídám (jsou deklarovány jako `private`)

*Je-li B vnitřní třídou A, pak členy `private` třídy jsou přístupné i třídě B.*
  - Třída B je skryta mimo třídu A
  - Metody třídy A nemají přístup k proměnným a metodám třídy B
- Zvýšení čitelnosti kódu a zlepšení údržby kódu

<http://docs.oracle.com/javase/tutorial/java/java00/nested.html>

## Příklad vnitřní třídy

```
public class OutClass {  
    ...  
  
    private class InnerClass {  
  
        final String msg;  
  
        public InnerClass(String msg) {  
            this.msg = msg;  
        }  
  
    }  
  
    ...  
}
```

## Vnitřní třídy – přehled

- Prvkem třídy může být jiná třída – **vnořená/vnitřní třída**
  - Třída, která obsahuje vnořenou třídu – **vnější třída**

### Vnitřní třída

- **Statická vnořená třída – static**
  - Nemůže přímo přistupovat k instančním členům vnější třídy, musí vytvořit její instanci, přes ni má pak přístup
  - V podstatě se chová jako běžná statická třída, jen přístup je k ní přes jméno vnější třídy
- **Vnitřní třída (bez static)**
  - Má přístup ke všem členům vnější třídy včetně prvků **private**
    - Má své vlastní proměnné a metody
    - Nemá statické členy
- Vnější třída může do vnitřní jen přes její instanci
- Vnitřní třída není přístupná vně definice vnější třídy, jen v rámci vnější třídy

*Pokud nepotřebujeme jméno vnitřní třídy, můžeme použít anonymní vnitřní třídu.*

## Příklad anonymní vnitřní třídy

```
btn3.addActionListener(new ActionListener() {  
    @Override  
    public void actionPerformed(ActionEvent e) {  
        btn3.setText("clicked");  
    }  
});
```

# Shrnutí přednášky

## Diskutovaná témata

- GUI v Javě – „návrhář“ a programově definované grafické rozhraní
- Model-View-Controller
  - Model-Pohled-Řadič*
- Event-Driven Programming
  - Událostmi řízené programování*
- Události v Javě (Swing)
- Vnitřní třída a anonymní třída
  
- **Příště: Vlákna**