

Java v rytme Swingu

(vývoj okienkových aplikácií)

Róbert Novotný
robert.novotny@upjs.sk



- grafické rozhranie (**GUI**) je v súčasnosti už štandardom
- bežný používateľ sa stretáva takmer výhradne len s nimi
 - textový procesor, prehliadač, kalkulačka...
- spoločným úsilím bez ohľadu na jazyk je
 - **uľahčiť vývoj** aplikácií (najlepšie vizuálne)
 - poskytnúť **multiplatformnosť**

- úspešné pokusy o rýchly návrh už v minulosti
- **Borland Delphi** (Pascal, od 1995)
 - prvý RAD (rapid app. development)
 - grafický návrh. Komponenty vkladáme do formulára. Výzor pri návrhu je rovnaký ako počas behu. Dopíšeme lepiaci kód a ideme.
- légie nasledovníkov
 - **Visual Basic** [Pascal], **C++ Builder** [C++], **Visual Studio.NET** [C#]

- Java nemohla nepokryť túto oblasť už od počiatku
 - pokryla ju však veľmi povážlivým spôsobom
- v súčasnosti hneď tri knižnice pre vývoj
 - AWT (prapôvodná knižnica)
 - zobrazovanie ovládacích prvkov ponecháva na OS: natívny vzhľad, ale množstvo problémov s prenositeľnosťou
 - Swing (druhý pokus) *NetBeans*
 - značne vylepšený návrh, v súčasnosti štandard.
 - vykreslenie si rieši samo. Aplikácie vyzerajú jednotne, možnosť používať Look & Feel (alias *skiny*)
 - SWT (IBM) *Eclipse*
 - zobrazenie prvkov rieši OS: natívny vzhľad, ale potreba medziknižnice – limitovaný počet platform

- aplikáciu možno navrhovať **vizuálne** alebo ručne (písaním kódu)
- **vizuálne** – vývojové prostredie poskytuje možnosť tvorby formulárov, vkladania komponentov, dodávanie kódu pre obsluhu udalostí...
 - *NetBeans*
 - *IntelliJ IDEA*
- **ručne** – písaním kódu



Swing – základné prvky

- **komponenty**
 - všetko s čím sa dá interagovať
ovládacie prvky, teda okná, gombíky, tlačidlá, rozbaľovacie boxy, ...
- komponenty sú v **kontajneroch**
 - *okno obsahuje panel, ktorý obsahuje panel, ktorý obsahuje, ..., ktorý obsahuje gombík*
- komponenty reagujú na **udalosti**
 - interakciou s komponentom generujeme udalosti
klik na gombík, pohyb myšou, výber položky v zozname...
- komponent môže mať **model**
 - model poskytuje dáta, komponent ich zobrazuje
 - dáta: zoznam, model: prostredník, komponent: rozbaľovací box

- Prázdny formulár vytvoríme jednoducho:

```
public class MainForm extends JFrame {  
  
}
```

```
public class Runner {  
    public static void main(String[] args) {  
        MainForm mainForm = new MainForm();  
        mainForm.setBounds(0, 0, 640, 480);  
        mainForm.setDefaultCloseOperation(  
            WindowConstants.EXIT_ON_CLOSE);  
        mainForm.setVisible(true);  
    }  
}
```

- **bounds** -
rozmary okna v
pixeloch
- **default close
operation** –
štandardná
operácia
vykonaná pri
zavretí okna, tu:
ukončenie
aplikácie
- **setVisible()** –
zobrazí/skryje
okno

- Veci môžeme presunúť do konštruktora

```
public class MainForm extends JFrame {
    public MainForm() {
        setBounds(0, 0, 640, 480);
        setDefaultCloseOperation(
            windowConstants.EXIT_ON_CLOSE);
    }
}
```

```
public class Runner {
    public static void main(String[] args) {
        MainForm mainForm = new MainForm();
        mainForm.setVisible(true);
    }
}
```


- okno (**JFrame**) je ako okno na dome. Má rámec, kľučky, atď.
- analógia: Vianoce v škôlke
 - lepíme na okno Mikulášov
 - my budeme lepíť na okno ovládacie prvky
- v skutočnosti ich ale lepíme na *sklo*
- sklo = **ContentPane**
- postup
 - **vytvor inštanciu** komponentu
 - **nastav** jej rozmery, vlastnosti, atď.
 - **prilep** ju na sklo

- gombík s textom OK umiestnený vľavo hore
- zodpovedá mu trieda **JButton**

```
public class MainForm extends JFrame {
    public MainForm() {
        ...
        //toto zatiaľ necháme, okomentujeme neskôr
        setLayout(null);

        JButton button = new JButton("OK");
        button.setBounds(10, 10, 80, 20);

        getContentPane().add(button);
    }
}
```

- môžeme veselo pridávať ďalšie komponenty

```
public class MainForm extends JFrame {
    public MainForm() {
        ...
        JTextField textField = new JTextField("Hello world");
        textField.setBounds(10, 50, 80, 20);
        getContentPane().add(textField);
        ...
    }
}
```

- **JTextField** – textové políčko



Udalost'ami riadené programovanie kedysi

- kedysi: údaje zadávané do programu postupne

```
Enter username: novotnyr  
New UNIX password: *****  
Retype new UNIX password: *****  
passwd: all authentication tokens updated successfully.
```

- program si v cykle pýta údaje
- možnosť zmeny predošlých údajov je často nemožná



Udalosťami riadené programovanie dnes

- údaje zobrazované sú v oknách a ovládacích prvkoch (komponentoch)
- tie **pasívne** čakajú na užívateľovu interakciu s nimi
- užívateľ interakciou s komponentami generuje udalosti (events)
- komponent vie zareagovať na vhodnú udalosť a vykonať príslušnú akciu
- **event driven programming**:
 - naprogramujeme metódy, ktoré sa budú volať z komponentov pri spracovávaní udalostí
 - rýchle programovanie, nenáročné, ľahko pochopiteľné

- komponent má nadefinovanú sadu udalostí, na ktoré vieme zareagovať
- Príklad: gombík JButton podporuje udalost(i):
 - vyvolať akciu
- Príklad 2: textové políčko JTextField:
 - pohyb kurzora; vyvolanie akcie; zmena obsahu
- zoznam podporovaných udalostí zistíme:
 - otvoríme dokumentáciu
 - pohľadáme metódy `addXXXListener()`
 - popozeráme komentáre k interfejsu `XXXListener`
- gombík:
 - `addActionListener()`
 - interfejs `ActionListener`

- ovládací prvok je ako **rozhlasový vysielateľ**
- po vyvolaní udalosti **oznami okoliu** "Vážení poslucháči, niekto na mne vyvolal udalosť U"
- každý **poslucháč** vie na to zareagovať.

- niekto klikne na gombík (udalosť *action*)
- gombík oznami okoliu "Vážení poslucháči, niekto na mne vyvolal udalosť **ActionEvent**")
- každý poslucháč (**ActionListener**) vie na to zareagovať

- ako vytvorím poslucháča pre udalosť *action*?
 - vytvorím inštanciu triedy, ktorá implementuje interfejs `ActionListener`

```
public class SysoutListener implements ActionListener {  
    public void actionPerformed(ActionEvent e) {  
        System.out.println("Klik!");  
    }  
}
```

- ako docielim, aby poslucháč dostával informácie od gombíka?
 - inštanciu poslucháča zaregistrujem na gombíku

- ako docielim, aby poslucháč dostával informácie od gombíka?
 - inštanciu poslucháča zaregistrujem na gombíku

```
public class MainForm extends JFrame {
    public MainForm() {
        ...
        gombík.addActionListener(new SysoutListener());
        ...
    }
}
```

- anonymné vnútorné triedy – umožňujú na jednom riadku
 - vytvoriť triedu implementujúcu rozhranie
 - vytvoriť jej inštanciu

```
public class SysoutListener implements ActionListener {  
    public void actionPerformed(ActionEvent e) {  
        System.out.println("Klik!");  
    }  
}
```

```
gombik.addActionListener(new SysoutListener());
```



```
gombik.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent e) {  
        System.out.println("Klik!");  
    }  
});
```



Udalosti – kód generovaný NetBeans-om

```
public class MainForm extends JFrame {
    public MainForm() {
        ...
        gombík.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                gombíkActionPerformed(e);
            }
        });
        ...
    }
    public void gombíkActionPerformed(ActionEvent e) {
        System.out.println("klik!");
    }
}
```

- **model** je objekt, ktorý poskytuje komponentu dáta na zobrazenie
- viacero komponentov môže zobrazovať dáta z modelu odlišným spôsobom (v zozname, tabuľke, ...)
- **ListBox** v starom Delphi:
 - jednotlivé položky listboxu sa museli pridať ručne
 - nevýhoda: dáta máme aj v listboxe aj v nejakom poli
 - nevýhoda: musíme synchronizovať dáta v komponente s dátami v poli
- **JList** v Swingu
 - pole obalíme ListModelom, ktorý pridáme ku JListu



Zoznam súborov ako model pre JList

- vyrobíme si vlastnú triedu pre model zoznamu (dedíme od `AbstractListModel`)

```
public class DirectoryFilesListModel
                                extends AbstractListModel{
    private File file;

    public DirectoryFilesListModel(File file) {
        this.file = file;
    }

    public int getSize() {
        return file.listFiles().length;
    }

    public Object getElementAt(int index) {
        return file.listFiles()[index];
    }
}
```

- model priradíme nasledovne

```
ListModel fileListModel  
    = new DirectoryFilesListModel(new File("D:"))  
JList listOfFiles = new JList(fileListModel);  
getContentPane().add(listOfFiles);
```

- jeden model môže zdieľať aj viacero komponentov

```
JComboBox fileCombo = new JComboBox(fileListModel);  
getContentPane().add(fileCombo);
```

- zmena v modeli sa prejaví vo všetkých komponentoch, ktoré s ním pracujú

- model je niekedy zamaskovaný, predošlý príklad je ekvivalentný s

```
File[] files = new File("D:").listFiles();  
JList listOfFiles = new JList(files);
```

- v útrobach JListu sa vytvoril model
- v niektorých komponentoch model nie je, niekde sa zaobídeme aj bez neho, inde je zase nutnosťou (stromy)