Grafické uživatelské rozhraní a knihovna Swing

3

V tomto a několika navazujících seminářích se budeme zabývat problematikou tvorby grafických uživatelských aplikací. Tento úvodní seminář se bude věnovat zejména základním principům práce s knihovnou Swing.

1 Možnosti pro tvorbu grafických uživatelských aplikací

Platforma Java nabízí několik knihoven, které umožňují vytvářet aplikace s grafickým uživatelským rozhraním (GUI[1]). Žádná z nich nepředstavuje univerzální řešení a při její volbě je potřeba brát v potaz její vlastnosti a případně potřeby jednotlivých aplikací. Popíšeme si teď čtyři možnosti, které lze prakticky použít.

1.1 Abstract Window Toolkit

Nejstarší knihovnou pro tvorbu GUI, která v Javě existuje, je *Abstract Window Toolkit (AWT)*, která poskytuje abstrakci nad nativními komponentami, které poskytuje daná platforma (např. Windows, macOS), takže vizuálně odpovídají dané platformě, zároveň z toho plyne celá řada omezení a sada komponent, které lze použít, je omezená. Dnes se AWT přímo pro tvorbu grafických aplikací nepoužívá, ale využívají se její třídy například pro reprezentaci grafických objektů a je nad ní postavena knihovna Swing.

1.2 Swing

Swing je knihovna pro tvorbu grafických aplikací, která je kompletně napsaná v Javě, z toho plynou některé její důležité vlastnosti. Aplikace napsaná s touto knihovnou vypadá stejně bez ohledu na použitou platformu, tj. vypadá stejně ve Windows jako v Linuxu nebo macOS. To může být výhoda i nevýhoda, záleží na požadavcích, které na aplikaci klademe.[2] Fakt je, že po vizuální stránce aplikace napsané s využitím knihovny Swing nezapadají ani do jednoho operačního systému, i když je možné zvolením správného Look & Feel tento nedostatek minimalizovat. Skutečnost, že je celá knihovna napsaná v Javě se odráží na její rychlosti. Připomeňme, že Java typicky pro svůj běh využívá JIT překladač, který části kódu, které jsou prováděny často, převede do strojového kódu a nechává provést přímo procesorem. Ty méně často prováděné části kódu interpretuje. Z povahy grafických aplikací může být takto intepretovaných částí kódu nezanedbatelné množství, a proto se aplikace napsané s knihovnou Swing mohou zdát subjektivně pomalejší. Velkou výhodu knihovny Swing je, že je součástí standardního běhového prostředí a nevyžaduje

[1] Graphical User Interface

[2] Například z důvodů zaškolení můžeme chtít, aby aplikace vypadala vždy stejně, bez ohledu na jakém operačním systému běží.

žádné dodatečné závislosti.

1.3 Standard Window Toolkit

Určitou reakci na nedostatky AWT a Swing představuje *Standard Windows Toolkit (SWT)*, knihovna, která nabízí mapování standardních komponent nativních grafických knihoven (Win32, GTK+, Cocao) na třídy Javy. Díky tomu může grafické rozhraní vypadat jako svižnější a zapadne dobře do dané platformy. Nevýhodou ve srovnání s knihovnou Swing je, že je narušen princip Write-Once-Run-Anywhere, protože je nutné vždy s aplikací distribuovat i odpovídající binární soubory používané SWT na dané platformě.

1.4 JavaFX

JavaFX představuje moderní platformu, která umožňuje jak tvorbu grafických aplikací, tak obecně práci s grafikou a práci s multimédii. JavaFX používá vlastní sadu grafických komponent, proto, podobně jako Swing, nezapadá do grafického stylu žádného operačního systému. Avšak protože nabízí relativně moderní vzhled, není to tak nepříjemné jako v případě knihovny Swing. Práce s touto knihovnou je komfortní a nabízí celou škálu funkcí pro tvorbu aplikací, které jdou nad rámec toho, co nabízí AWT, Swing nebo SWT. Jako příklad můžeme zmínit deklarativní přístup k animacím nebo plnohodnotný HTML prohlížeč[3] jako jednu z komponent. Dříve se uvažovalo, že JavaFX nahradí Swing v roli hlavní knihovny pro tvorbu GUI. Avšak od verze Java 11 byla platforma JavaFX vyčleněna do samostatného projektu (OpenJFX) a je vyvíjena odděleně. Z toho plyne, že je potřeba s aplikací distribuovat i odpovídající knihovny a binární soubory podobně jako v případě SWT.

2 Knihovna Swing

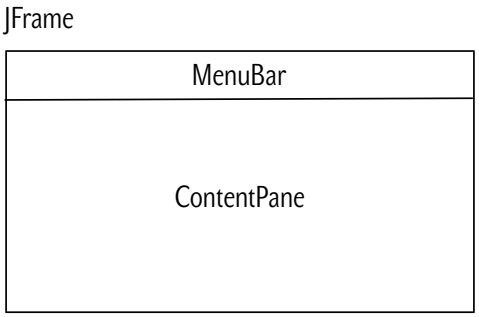
V rámci tohoto semináře se zaměříme na knihovnu Swing a platformu JavaFX. Knihovna Swing, ač letitá, je dostupná všude[4] bez nutnosti řešit další závislosti. JavaFX zase reprezentuje moderní knihovnu, která umožňuje pohodlně vytvářet vizuálně pěkné aplikace splňující požadavky dnešní doby.

I když vývojová prostředí nabízí docela slušné nástroje, které umožňují graficky rozvrhnout obsah okna a nastavit jednotlivým komponentám vlastnosti. My v tomto semináři tyto nástroje používat nebudeme a zaměříme se na tvorbu uživatelského rozhraní čistě programově. Motivací pro tento přístup je skutečnost, že nástroje, které jsou v IDE, počítají s určitou množinou scénařů, která ale nepokrývá všechny možnosti, se kterými se v praxi můžeme setkat, a pak je nutné vytvářet části grafického rozhraní programově. Dalším důvodem je zkušenost, že se někdy tyto nástroje „rozbijí", a pak je potřeba se s tím nějak vypořádat, k čemuž je dobré mít znalosti toho, jak grafické uživatelské rozhraní vypadá z pohledu programu.

Jak již bylo zmíněno v úvodní kapitole, knihovna Swing je zcela implementovaná v jazyce Java a jednotlivé prvky uživatelského rozhraní jsou přirozeně implementovány jako standardní třídy a objekty v Javě. Tvorba grafických aplikací se proto příliš neodlišuje od tvorby jiných aplikací.

[3] Jedná se integrované jádro WebKit.

[4] Alespoň zatím.



Obrázek 1: Rozdělení obsahu okna

2.1 Základní okno

Základem každé grafické aplikace je minimálně jedno okno, s kterým uživatel interaguje. Okna v knihovně Swing jsou reprezentována třídou `JFrame`. Chceme-li vytvořit vlastní okno, nabízí se vytvořit potomka této třídy a nastavit mu obsah a chování. Minimální příklad by mohl vypadat následovně.

```
public class FirstWindow extends JFrame {
    public FirstWindow() {
        this.setDefaultCloseOperation(EXIT_ON_CLOSE);
        this.setPreferredSize(new Dimension(300, 100));
        this.pack();
    }
}
```

První řádek konstruktoru nám definuje, co se má stát, když dojde k zavření okna, např. křížkem v horním rohu. Zvolili jsme `EXIT_ON_CLOSE`, že se má ukončit celá aplikace, další možnosti jsou: zrušit okno a s ním uvolnit navázané prostředky, skrýt okno nebo nedělat nic, viz JavaDoc dokumentace metody `setDefaultCloseOperation`.

Na druhém řádku jsme definovali, že preferovaná velikost okna má být 300×100 pixelů. Třetí řádek se postará o přizpůsobení okna obsahu a preferovaným rozměrům.

Okno zobrazíme tak, že vytvoříme jeho instanci a zavoláme metodu `setVisible`, jak ukazuje následující příklad.

```
public static void main(String[] args) {
    JFrame firstWindow = new FirstWindow();
    firstWindow.setVisible(true);
}
```

2.2 Obsah okna a jeho rozložení

Okno lze rozdělit na dvě pomyslné části – menu (MenuBar) a obsah okna (ContentPane), jak ukazuje Obrázek 1. Obsahem okna může být libovolná třída dědící z třídy `Container`, komponenta obsahující další grafické komponenty. Typicky se používá třída `JPanel`, která tuto podmínku splňuje a je navržena

pro tento účel. Třída `JPanel` představuje obecný panel, kam se dají vkládat další komponenty, včetně panelů. My si ukážeme vložení několika tlačítek. Níže popsaný kód by byl součástí konstruktoru.

```
// vytvori panel, do ktereho budou vkladany dalsi komponenty
mainPanel = new JPanel();

// vytvori ctyri komponenty predstavujici tlacitka
btn1 = new JButton("Foo");
btn2 = new JButton("Bar");
btn3 = new JButton("Baz");
btn4 = new JButton("Qux");

// vlozi tlacitka do panelu
mainPanel.add(btn1);
mainPanel.add(btn2);
mainPanel.add(btn3);
mainPanel.add(btn4);

// nastavi obsah okna
this.setContentPane(mainPanel);
```

Nejdříve jsme vytvořili prázdný panel, následně čtyři tlačítka (třída `JButton`), která jsme vložili do panelu a oknu jsme jako obsah nastavili námi vytvořený panel. Panel i tlačítka jsme deklarovali jako atributy dané třídy. Mohli bychom je deklarovat i jako lokální proměnné, ale takto budou jednotlivé komponenty přístupné i dalším metodám, např. těm, které reagují na události.

Zbývá zadat, jak jednotlivá tlačítka mají být uspořádána. Při návrhu knihovny Swing se počítalo, že výsledné aplikace poběží na zařízeních, které mohou mít značně odlišné zobrazovací schopnosti, zejména rozlišení. Jiné rozlišení má přenosné kapesní zařízení a jiné pracovní stanice, a proto bylo cílem, aby se grafické aplikace napsaly jednou a přizpůsobily se automaticky možnostem počítačů, na kterých běží. Proto knihovna Swing příliš nepodporuje pixel perfect grafiku, kde by programátor s přesností na jednotlivé pixely udával, kam mají být komponenty umístěny, ale raději umožňuje definovat vztahy mezi jednotlivými komponentami, jejich vzájemné uspořádání, a výsledné umístění již obstará knihovna Swing.[5]

Způsob uspořádání jednotlivých komponent definuje tzv. *layout manager*, je jich k dispozici celá řada a ukážeme si ty nejzákladnější.

Pro uspořádání tlačítek do jedné řady můžeme použít `BoxLayout`, kdy definujeme, zda se mají komponenty skládat vedle sebe horizontálně či vertikálně.

Pokud použijeme následující příklad, uvidíme, že tlačítka jsou uspořádána nad sebou, tj. podle osy Y.

```
mainPanel.setLayout(new BoxLayout(mainPanel, BoxLayout.Y_AXIS));
```

Určitou nevýhodou tohoto layout manageru je, že v případě, že máme menší okno, některé komponenty

[5]Tento přístup zažil svou renesanci na mobilních platformách, kde je nutné počítat s tím, že každé zařízení, telefon či tablet, může mít specifické a omezené rozlišení.

se do něj nemusí vměstnat. Alternativou může být `FlowLayout`, který umožní přetečení do dalšího řádku či sloupce. Vyzkoušejte si použít následující rozložení a různě měnit velikost okna.

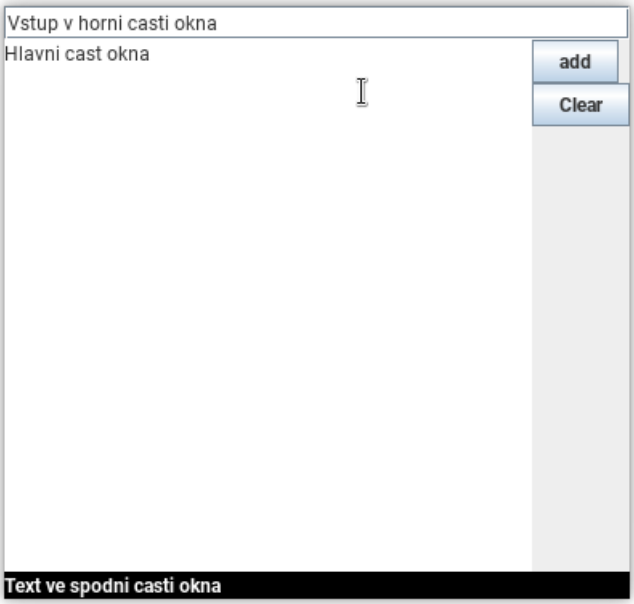
```
mainPanel.setLayout(new FlowLayout(FlowLayout.CENTER, 10, 10));
```

2.3 Border Layout a další běžné komponenty

Hlavní okno aplikace má velice často strukturu, která jej dělí do pěti oblastí (i) horní část pro nástrojové lišty, (ii) dolní část pro stavový řádek, (iii) boční části buď pro nástrojové lišty nebo specializované panely (iv) střední část obsahující hlavní obsah, se kterým aplikace pracuje, např. text v textovém editoru. Pro toto rozložení nabízí knihovna Swing třídu `BorderLayout`, jejíž použití si ukážeme na následujícím příkladu společně s dalšími běžně používanými komponentami.

```
// rozmisteni komponent po stranach okna
mainPanel = new JPanel(new BorderLayout());

// jednoradkovy textovy vstup
txtTop = new JTextField("Vstup v horni casti okna");

// viceradkove textove pole
txtaCenter = new JTextArea("Hlavni cast okna\n");

lbBottom = new JLabel("Text ve spodni casti okna");
lbBottom.setOpaque(true); // nastavi, aby stitek mel pozadi
lbBottom.setBackground(Color.BLACK);
lbBottom.setForeground(Color.WHITE);

// bocni panel obsahujici tlacitka
rightPanel = new JPanel();
btnAdd     = new JButton("add");
btnClear   = new JButton("Clear");

rightPanel.setLayout(new BoxLayout(rightPanel, BoxLayout.Y_AXIS));
rightPanel.add(btnAdd);
rightPanel.add(btnClear);

// vlozi komponenty do panelu
mainPanel.add(txtTop, BorderLayout.NORTH);
mainPanel.add(rightPanel, BorderLayout.EAST);
mainPanel.add(txtaCenter, BorderLayout.CENTER);
mainPanel.add(lbBottom, BorderLayout.SOUTH);
```

Nejdříve vytvoříme panel a nastavíme mu jako layout manager `BorderLayout`, dále vytvoříme jednořádkový vstup (třída `JTextField`) a textovou oblast s více řádky (třída `JTextArea`). Dále jsme si vytvořili



Obrázek 2: Rozmístění komponent s využitím `BorderLayout`

popisek (třída `JLabel`) a nastavili mu barevnost. Protože si budeme chtít ukázat reakce na události, vytvoříme ještě panel a do něj vložíme dvě tlačítka stejným způsobem, jako jsme to učinili v předchozí kapitole.

Na závěr jsme rozmístili komponenty po panelu. K určení pozice se zde, poněkud netradičně, používají světové strany. Jak vypadá takto vytvořené okno ukazuje Obrázek 2.

2.4 Reakce na události

Ve všech příkladech, co jsme si dosud uvedli, jsme pouze vytvářeli okna, která neumožňovala žádnou interakci s uživatelem. Pochopitelně u jednotlivých komponent je možné navázat akce, které se mají provést, když s nimi uživatel pracuje, např. když stiskne tlačítko, vloží text apod. K reakci na tyto události se používají tzv. *listenery*, třídy, které obsahují metody, které jsou vyvolány v momentě, že dojde k dané události, např. stisku tlačítka.

Uvažujme, že v předchozím příkladu po stisknutí tlačítka *Add* dojde k vložení textu, který je v horním vstupním poli, do textové oblasti uprostřed okna. Tradiční řešení by bylo použít anonymní třídu, například následovně.

```
btnAdd.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        String currentText = txtaCenter.getText();
        String inputText = txtTop.getText();
        txtaCenter.setText(currentText + "\n" + inputText);
    }
});
```

Toto řešení je ale zbytečně rozvleklé, a protože `ActionListener` je funkcionální rozhraní, můžeme použít místo anonymní třídy lambda výraz, tj.

```
btnAdd.addActionListener(e -> {
    String currentText = txtaCenter.getText();
    String inputText = txtTop.getText();
    txtaCenter.setText(currentText + "\n" + inputText);
});
```

Nevýhodou obou předchozích řešení je, že vyžadují aby jednotlivé akce byly definovány v rámci konstruktoru. Díky tomu, že `ActionListener` je funkcionální rozhraní, můžeme samotnou akci definovat ve standardní metodě.

```
private void onAddAction(ActionEvent e) {
    String currentText = txtaCenter.getText();
    String inputText = txtTop.getText();
    txtaCenter.setText(currentText + "\n" + inputText);
}
```

A následně navázat pomocí odkazu na metodu.

```
btnAdd.addActionListener(this::onAddAction);
```

Poslední varianta je sice o něco delší než použití lambda výrazu, ale vede ke srozumitelnějšímu kódu.

Při reakcích na události je často potřeba dát uživateli zpětnou vazbu, ať už formou krátké zprávy, např. že akce byla provedena nebo naopak, že akci nelze provést. Případně je často potřeba si vyžádat souhlas s provedením dané akce. K tomu se obvykle používají jednoduché dialogové okna s textem a několika tlačítky.

Pro tyto nejběžnější dialogy knihovna Swing nabízí třídu `JOptionPane`, která tyto dialogy umí realizovat formou volání metod. Pokud bychom se chtěli například ujistit, že uživatel chce provést vymázání obsahu okna, použili bychom potvrzovací dialog a ověřili bychom, že uživatel stiskl tlačítko OK následovně.

```
btnClear.addActionListener(e -> {
    int outcome = JOptionPane.showConfirmDialog(BorderLayoutForm.this,
                                                "Opravdu smazat obsah?",
                                                "Pozor",
                                                JOptionPane.OK_CANCEL_OPTION);
    if (outcome == JOptionPane.OK_OPTION)
          txtaCenter.setText("");
});
```

Všimněme si, že jako první argument metodě `showConfirmDialog` je předán odkaz na aktuální okno. To je nutné, aby takto vytvořený dialog nebylo možné překrýt hlavním oknem.

3 Menu

Závěrem tohoto semináře si ukážeme tvorbu menu. To se skládá ze tří základních komponent `JMenuBar` (hlavní menu v horní liště), `JMenu` (stahovací menu) a `JMenuItem` položky jednotlivých menu. Jejich vytvoření a provázání je přímočaré, jak ukazuje následující příklad.

```
mainMenu = new JMenuBar();

// vytvori objekt menu: ,,File'' (pristupny pres Alt+F)
menuFile = new JMenu("File");
menuFile.setMnemonic(KeyEvent.VK_F);

// vytvori polozku menu pristupnou pres Alt+X a CTRL+X
JMenuItem menuItemExit = new JMenuItem("Exit");
menuItemExit.setMnemonic(KeyEvent.VK_X);
menuItemExit.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_X, ActionEvent.CTRL_MASK));

// vlozi jednotlive casti menu na sva mista
menuFile.add(menuItemExit);
mainMenu.add(menuFile);
this.setJMenuBar(mainMenu);
```

U tohoto příkladu si zmiňme dvě nastavení. První je metoda `setMnemonic`, která pro jednotlivé části menu nastavuje klávesu, jež umožňuje jejich aktivaci pomocí kombinace kláves Alt+znak. Díky této vlastnosti je možné procházet menu bez použití myši, což může být užitečné pro rychlejší ovládání aplikace a zjednodušit práci lidem se zdravotním postižením. Dále můžeme pomocí metody `setAccelerator` definovat klávesovou zkratku, která přímo vyvolá akci spojenou s danou položkou bez toho, aniž by bylo nutné menu procházet.

Aby byl příklad úplný ukažme si ještě akci, která je navázana na volbu *Exit* a která ukončí aplikaci.

```
menuItemExit.addActionListener((ActionEvent e) -> {
    setVisible(false);
    dispose();
});
```

Uzavření okna a s tím spojené ukončení aplikace je realizováno ve dvou krocích. Kdy nejdříve okno skryjeme pomocí metody `setVisible` a následně uvolníme všechny prostředky s tímto oknem spojené.