7 Formulářové aplikace v JavaFX

Technologie JavaFX původně vznikla pro tvorbu tzv. Rich Internet Applications (RIA)[1], určitého hybridu mezi desktopovou a grafickou aplikací, jako konkurence technologie Adobe Flash, což do určité míry určuje její vlastnosti. Jedná se o technologii, která poměrně pohodlně umožňuje pracovat s grafikou, animacemi, multimédii a zároveň vytvářet aplikace s tradičními vlastnostmi jako jsou formuláře a dialogová okna.

RIA se s nástupem HTML5 ukázaly jako slepá větev vývoje, ale technologie JavaFX byla vybrána jako náhrada za zastarávající knihovnu Swing. Bohužel v rámci změn, které proběhly u verze Java 11, se Oracle rozhodl vyčlenit JavaFX z JDK. To je nyní distribuováno odděleně jako OpenJFX[2]. To sebou přináší tu nepříjemnou vlastnost, že tento balík je potřeba dodávat s aplikacemi napsanými pro JavaFX. Navíc každá platforma[3] potřebuje svou verzi JavaFX a ztrácíme tak, na Javě tolik oceňovanou, vlastnost *Write Once Run Anywhere*. Je to daň za to, že JavaFX umožňuje akcelerovanou práci s grafikou, práci s multimédii nebo třeba použít plnohodnotnou komponentu webového prohlížece (QtWebKit), které v čisté Javě nelze efektivně realizovat a je potřeba k tomu využívat nativní kód.

V tomto a následujícím semináři si ukážeme tvorbu jednoduchých aplikací postavených na JavaFX. Vývojová prostředí sice umožňují tvorbu aplikací pomocí grafických nástrojů, my se to mu vyhneme, abychom si mohli představit základní principy práce s touto platformou. Samotnému zprovoznění OpenJFX ve vývojových prostředích je věnován oddělený dokument.

1 Jednoduchá aplikace

1.1 HelloWorld I

Začneme jednoduchou aplikací, která vypíše obligátní statický text *Hello World*.

```
public class HelloWorldFX extends Application {
    public static void main(String[] args) {
        launch(args);
    }
    @Override
    public void start(Stage stage) throws Exception {
        Text lbHello = new Text("Hello World");
        FlowPane rootPane = new FlowPane(lbHello);
```

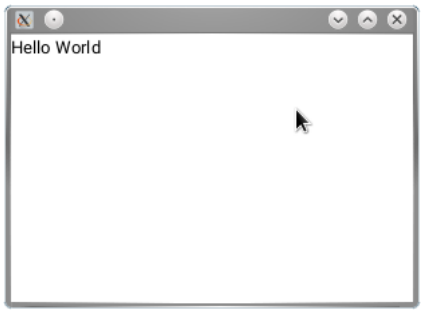
[1] https://en.wikipedia.org/wiki/Rich_web_application
[2] https://openjfx.io/
[3] Windows, Linux, macOS

```
        Scene scene = new Scene(rootPane, 300, 200);
        stage.setScene(scene);
        stage.show();
    }
}
```

Aplikace v JavaFX je potomkem třídy `javafx.application.Application`, která se stará o inicializaci aplikace, její spuštění a běh. Proto v metodě `main(String[])` předáváme argumenty příkazové řádky metodě `Application.launch(String[])`, která obstará spuštění aplikace.

Třída `Application` má abstraktní metodu `void start(Stage)`, která je vyvolána po základní inicializaci. Této metodě je přádán objekt `Stage`, který byl vytvořen platformou a který odpovídá (hlavnímu) oknu. V této metodě by měl být definován vzhled a chování okna.

To v našem případě znamená, že vytvoříme popisek (třída `Text`) a panel (třída `FlowPane`), do nějž popisek vložíme. Dále vytvoříme scénu (třída `Scene`), která definuje rozměry okna a kořen stromu jednotlivých ovládacích prvků (komponent). Tuto scénu nastavíme objektu třídy `Stage` a necháme jej zobrazit zavoláním metody `stage.show()`. Výslednou aplikaci ukazuje Obrázek 1.



Obrázek 1: Ukázka HelloWorld

1.2 HelloWorld II

Ukažme si složitější příklad, kde pozměníme metodu `start`.

```
public void start(Stage stage) throws Exception {
    Text      lbInput  = new Text("Name:");
    TextField txtInput = new TextField("");
    Button    btnOk    = new Button("Ok");

    btnOk.setOnAction(e -> {
        System.out.println("Hello: " + txtInput.getText());
    });

    FlowPane root = new FlowPane(10, 5);
    root.setAlignment(Pos.CENTER);
    root.getChildren().addAll(lbInput, txtInput, btnOk);
```

```
        stage.setTitle("Hello World");
        stage.setScene(new Scene(root, 300, 200));
        stage.show();
    }
```

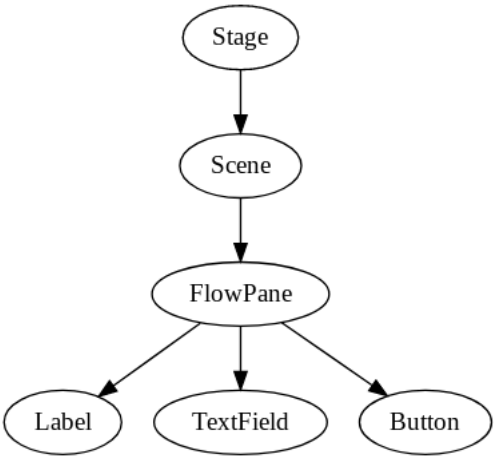
Nyní součástí okna není jen textový popisek, ale i textový vstup a tlačíko. Na stisk tlačítka jsme pomocí lambda výrazu navázali vypsání obsahu textového vstupu na terminál. Toto nijak nevybočuje z toho, co známe z knihovny Swing, avšak pro jednotlivé události lze nastavit pouze jednu akci.

Stejně jako v úvodním příkladu jsme použili panel `FlowPane`, ten skládá komponenty vedle sebe, avšak dojde-li místo, udělá zalomení a pokračuje na dalším „řádku". De facto to odpovídá použití `JPanelu` společně s `FlowLayout`. Pokud bychom chtěli komponenty uspořádat vertikálně nebo horizontálně, použijeme třídy `VBox` nebo `HBox`.

```
HBox root = new HBox(10); // sklada komponenty do vodorovne rady
VBox root = new VBox(10); // sklada komponenty do svisle rady
```

Dá se tedy říct, že JavaFX nerozlišuje mezi panelem a layout managerem (jako knihovna Swing), ale tato funkcionalita je realizována specializovanými panely pro každý typ uspořádání. U jednotlivých panelů můžeme dále nastavit vlastnosti, které upravují, jak má rozložení komponent vypadat, např. jaké mají být rozestupy mezi komponentami.

Za povšimnutí stojí také, že v tomto příkladu nejsou komponenty vloženy do panelu v konstruktoru, ale pomocí metody `getChildren()`, která vrací seznam potomků, a se kterým můžeme dál pracovat. Platforma JavaFX otevřeně přiznává, že komponenty tvoří strom (viz Obrázek 2) a umožňuje tak s ním i pracovat.

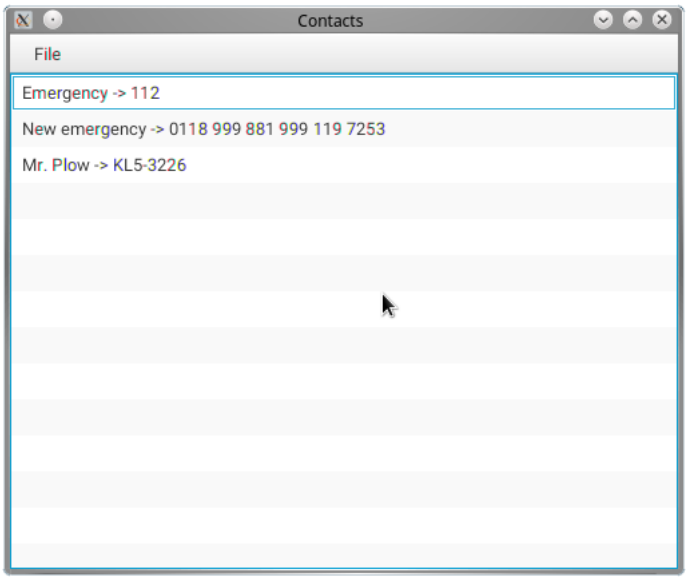


Obrázek 2: Graf objektů

2 Složitější aplikace

Pro představení dalších vlastností si uděláme jednoduchou aplikaci na evidenci telefonních kontaktů. Tato aplikace se bude skládat ze seznamu telefonních čísel a menu, které umožní zadat nové telefonní číslo a

ukončit aplikaci. Jak bude hlavní okno vypadat, ukazuje Obrázek 3



Obrázek 3: Hlavní okno aplikace

Nejdříve si vytvoříme třídu, která bude reprezentovat položky telefonního seznamu.

```
public class PhoneContact {
    private final String name;
    private final String phoneNum;

    public PhoneContact(String name, String phoneNum) {
        this.name     = name;
        this.phoneNum = phoneNum;
    }
    public String getName() {
        return name;
    }
    public String getPhoneNum() {
        return phoneNum;
    }
    @Override
    public String toString() {
        return name + " -> " + phoneNum;
    }
}
```

2.1 Hlavní okno

Základní struktura aplikace je stejná jako u předchozích příkladů:

```
public class PhoneFX extends Application {
    public static void main(String[] args) {
        launch(args);
    }
    public void start(Stage primaryStage) throws Exception {
        // FIXME: zde chybi vytvoreni obsahu okna
        primaryStage.setTitle("Contacts");
        primaryStage.setScene(new Scene(root, 500, 400));
        primaryStage.show();
    }
}
```

Výše uvedený kód není kompletní (a ani se nedá přeložit), protože potřebujeme definovat, co bude náplní okna.

Vytvoříme proto komponentu obsahující seznam telefonních kontaktů:

```
ListView<PhoneContact> listPhoneNums = new ListView<PhoneContact>();
```

Dále přímočarým způsobem vytvoříme hlavní menu (třída `MenuBar`), které se skládá z rozbalovacích nabídek (`Menu`) a ty se skládají z jednotlivých položek (třída `MenuItem`), jak ukazuje následující kód.

```
MenuItem mnuNew = new MenuItem("_New");
mnuNew.setAccelerator(new KeyCodeCombination(KeyCode.N, KeyCombination.CONTROL_DOWN));

MenuItem mnuExit = new MenuItem("E_xit");
mnuExit.setAccelerator(new KeyCodeCombination(KeyCode.X, KeyCombination.CONTROL_DOWN));
mnuExit.setOnAction(e -> primaryStage.close());

Menu menuFile = new Menu("_File");
menuFile.getItems().addAll(mnuNew, new SeparatorMenuItem(), mnuExit);

MenuBar menuBar = new MenuBar(menuFile);
```

Na rozdíl od jiných platforem nemá v JavaFX hlavní menu nijak významné postavení a jedná se o grafický uživatelský prvek, který je na stejné úrovni jako třeba tlačítko nebo textový vstup. Proto, pokud jej chceme umístit do horní části okna nad hlavní obsah, nabízí se použití třídy `BorderPane`, který uspořádá komponenty podobně jako `BorderLayout`:

```
BorderPane root = new BorderPane();
root.setTop(menuBar);
root.setCenter(listPhoneNums);
```

Úkol: Vyzkoušejte si umístit menu do spodní části okna.

Nyní již lze aplikaci přeložit, ale chybí jí jedna podstatná vlastnost – seznam telefonních čísel.

Na rozdíl od knihovny Swing, kde se u složitějších komponent v návrhu dbá na oddělení prezentace, modelu a dat, platforma JavaFX velmi intenzivně využívá návrhový vzor *Observer*[4] a je tak tomu i při práci se seznamem.

Pro tyto potřeby má JavaFX vlastní sadu kolekcí typu `ObservableList<T>`, `ObservableSet<T>` a `ObservableMap<K, V>`, které se liší od tradičních kolekcí v tom, že je možné navázat reakce na změny jejich obsahu. To znamená, že podobně jako můžeme definovat akci při stisku tlačítka, můžeme definovat akci při vložení nebo odebrání prvku do/z seznamu. K vytvoření těchto kolekcí slouží statické metody z třídy `FXCollections`.

Seznam telefonních čísel bychom mohli vytvořit následovně:

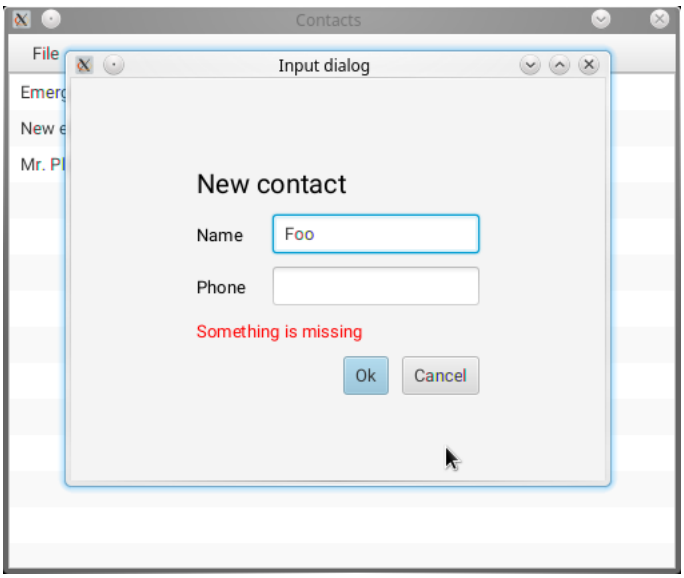
```
private ObservableList<PhoneContact> phoneNums = FXCollections.observableArrayList(
    new PhoneContact("Emergency", "112"),
    new PhoneContact("New emergency", "0118 999 881 999 119 7253"),
    new PhoneContact("Mr. Plow", "KL5-3226"));
```

A nyní jej můžeme navázat na odpovídající komponentu:

```
listPhoneNums.setItems(phoneNums);
```

2.2 Dialogové okno

Dialogová okna hrají významnou roli u řady aplikací, proto si nyní ukážeme, jak ukázkovou aplikaci rozšířit o dialog sloužící k zadání nového telefonního čísla do seznamu. Tento dialog by měl mít vzhled, jaký ukazuje Obrázek 4.



Obrázek 4: Výsledné dialogové okno

[4] `https://en.wikipedia.org/wiki/Observer_pattern`

Jelikož jsme se s celou řadou věcí již setkali při tvorbě jednoduchých formulářů, popíšeme si nyní jen to, v čem se práce s dialogem odlišuje.

Pro vyvolání dialogu použijeme samostatnou metodu `void newItem(Stage)`.

```
private void newItem(Stage primaryStage) {
    Stage stage = new Stage();
    stage.initOwner(primaryStage);
    stage.initModality(Modality.WINDOW_MODAL);
    stage.setTitle("Input dialog");
```

Vytvořili jsem nový objekt `Stage` a oknu nastavili rodiče (`primaryStage`) a příznak modality, aby nebylo možné přepnout z dialogu do rodičovského okna.

K rozložení komponent ve formuláři použijeme velice praktický `GridPane`, který umožňuje definovat rozložení pomocí mřížky (tabulky)[5], která nemusí být pravidelná a sousedící buňky mohou být spojeny a komponenty tak mohou obsadit více buněk. Vytvoření struktury formuláře ukazuje následující kód.

```
GridPane root = new GridPane();

Text lbCaption = new Text("New contact");
root.add(lbCaption, 0, 0, 2, 1); // roztazeni pres dva sloupce

TextField txtName = new TextField();
TextField txtPhone = new TextField();

root.addRow(1, new Text("Name"), txtName);
root.addRow(2, new Text("Phone"), txtPhone);

// text, ktery se zobrazi v pripade chyby
Text lbNotification = new Text();
root.add(lbNotification, 0, 3, 2, 1);

// tlacitka
Button btnOk = new Button("Ok");
Button btnCancel = new Button("Cancel");

// vytvori blok tlacitek
HBox buttons = new HBox(10);
buttons.getChildren().addAll(btnOk, btnCancel);
root.add(buttons, 0, 4, 2, 1);
```

Nejdříve jsem si vytvořili textový popisek pro nadpis okna a umístili jej na pozici 0, 0 a zároveň jsme uvedli, že má pokrývat dva sloupce a jeden řádek.

[5] Podobně jako se v minulosti dělaly formuláře v HTML.

V dalším kroce jsme vytvořili dva textové vstupy a pomocí metody `GridPane.addRow` jsme je vložili na odpovídající řádky s jejich textovými popisky.

Dále jsme vytvořili prázdný textový popisek, kterým budeme moci zobrazit text chyby, pokud bude formulář špatně vyplněn, a blok tlačítek *Ok* a *Cancel* pro zavření dialogu.

Pokud bychom chtěli vidět, jak je mřížka použita, můžeme nastavit vlastnost `gridLinesVisible` na `true`.

```
root.setGridLinesVisible(true);
```

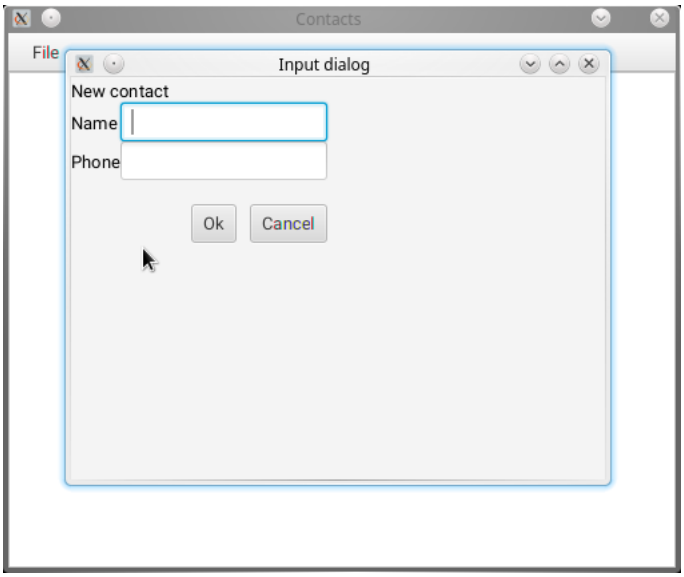
Zbývá jen metodu `newItem` dokončit tím, že nastavíme scénu a zobrazíme okno. Pro lepší dojem okno ještě vystředíme na ploše.

```
stage.setScene(new Scene(root, 400, 300));
stage.centerOnScreen();
stage.show();
```

Můžeme vyzkoušet, jak nově vytvořený dialog funguje a vypadá. K tomu potřebujeme na volbu menu navázat otevření formuláře:

```
mnuNew.setOnAction(e -> newItem(primaryStage));
```

Výsledek ukazuje Obrázek 5 a zatím není zcela uspokojivý. Vzhled pokulhává a nejsou implementovány akce spojené s tlačítky.



Obrázek 5: Rozpracované dialogové okno

Akce, které budou vyvolány po stisku tlačítek, vypadají následovně.

```
btnOk.setOnAction(e -> {
    String name = txtName.getText().trim();
    String phone = txtPhone.getText().trim();
```

```
    if (name.equals("") || phone.equals("")) {
        lbNotification.setText("Something is missing");
    } else {
        phoneNums.add(new PhoneContact(name, phone));
        stage.close();
    }
});
btnCancel.setOnAction(e -> stage.close());
```

Kód obou událostí je přímočarý. Při stisku *Ok* ověříme platnost vstupu, a pokud něco chybí, nastavíme textové upozornění. Pokud je vše v pořádku, přidáme číslo do seznamu a okno zavřeme. U tlačítka *Cancel* zavřeme formulář bez ohledu na vstup.

Jelikož se jedná o standardní tlačítka formulářů, chceme, aby reagovaly dle zvyklostí na stisk *Enter* a *Escape*. Toho docílíme nastavením odpovídajících atributů.

```
btnOk.setDefaultButton(true); // stisknuti enter odpovida stisknuti tlacitka
btnCancel.setCancelButton(true); // stisknuti escape odpovida stisknuti tlacitka
```

Závěrem doladíme vzhled. Zarovnáme komponenty na střed a vložíme mezi ně rozestupy.

```
root.setAlignment(Pos.CENTER);
root.setHgap(20);
root.setVgap(10);
```

Tlačítka zarovnáme doprava:

```
buttons.setAlignment(Pos.CENTER_RIGHT);
```

A popiskům upravíme vzhled na základě jejich funkce:

```
lbCaption.setFont(Font.font(20));
lbNotification.setFill(Color.RED);
```