1. díl - Úvod do JavaFX

JavaFX a Swing

V Javě je spolu s JavaFX přítomný ještě starší okenní framework Swing. Oracle oznámil, že ho má JavaFX v budoucnu nahradit. Proto má nové aplikace smysl programovat již jen v JavaFX. U existujících projektů se budete ještě nějakou dobu setkávat se starším Swingem a je dobré mít v něm alespoň základní přehled.

JavaFX

JavaFX je moderní framework pro tvorbu bohatých okenních aplikací. Bohatých je zde myšleno vizuálně. JavaFX přináší podporu obrázků, videa, hudby, grafů, CSS stylů a dalších technologií, které zajistí, že výsledná aplikace je opravdu líbivá. Zároveň je kladen důraz na jednoduchost tvorby, všechny zmiňované věci jsou v JavěFX v základu. JavaFX se hodí jak pro desktopové aplikace, tak pro webové applety nebo mobilní aplikace.

FXML

V JavaFX je možné vyvíjet podobně jako ve starším Swingu, tedy tak, že tvoříte instance jednotlivých formulářových prvků (tlačítko, textové pole). Ty poté vkládáte do tzv. layoutů, což jsou vlastně kontejnery na formulářové komponenty.

Druhým způsobem, který budeme my v seriálu používat, je FXML. FXML je jazyk pro návrh formulářů. Asi vás podle názvu nepřekvapí, že je to další jazyk odvozený z XML. Použití XML pro návrh prezentační části aplikace (to je ta část, se kterou komunikuje uživatel) není nic nového, naopak se jedná o osvědčený princip z webových aplikací. Java se zde stejně jako C# inspiruje a přenáší principy HTML a CSS do desktopových aplikací.

Java Scene Builder

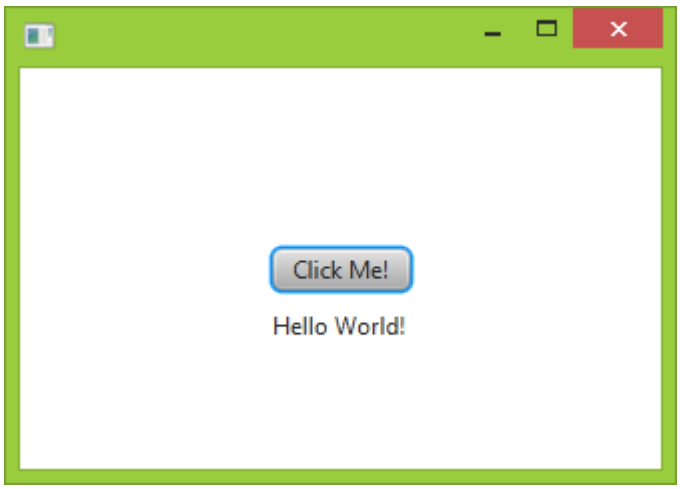
Dobrou zprávou je, že FXML nemusíme psát ručně, máme totiž k dispozici grafický designer. Ten není přímo v NetBeans, ale jedná se o samostatnou aplikaci, kterou pod názvem JavaFX Scene Builder stáhnete zde:http://www.oracle.com/…2157684.html. Aplikaci si rovnou nainstalujte.

První formulářová aplikace

Vytvořme si první okenní JavaFX aplikaci. Nezačneme ničím jiným, než Hello World. To je program, který nedělá nic jiného, než že vypíše nějaký text. V našem případě tento text zobrazíme ve formuláři.

Jako IDE budeme používat NetBeans. Vytvoříme nový projekt z kategorie JavaFX, u typu projektu zvolíme JavaFX FXML Application. Právě to je projekt, ve kterém můžeme používat grafický návrhář. Aplikaci pojmenujeme HelloFX a potvrdíme.

NetBeans nám vygeneruje projekt, který již obsahuje malou ukázkovou aplikaci. Když projekt spustíme, uvidíme formulář s jedním tlačítkem. Po kliknutí na něj se zobrazí nápis Hello World. Jsme tedy v podstatě bez práce

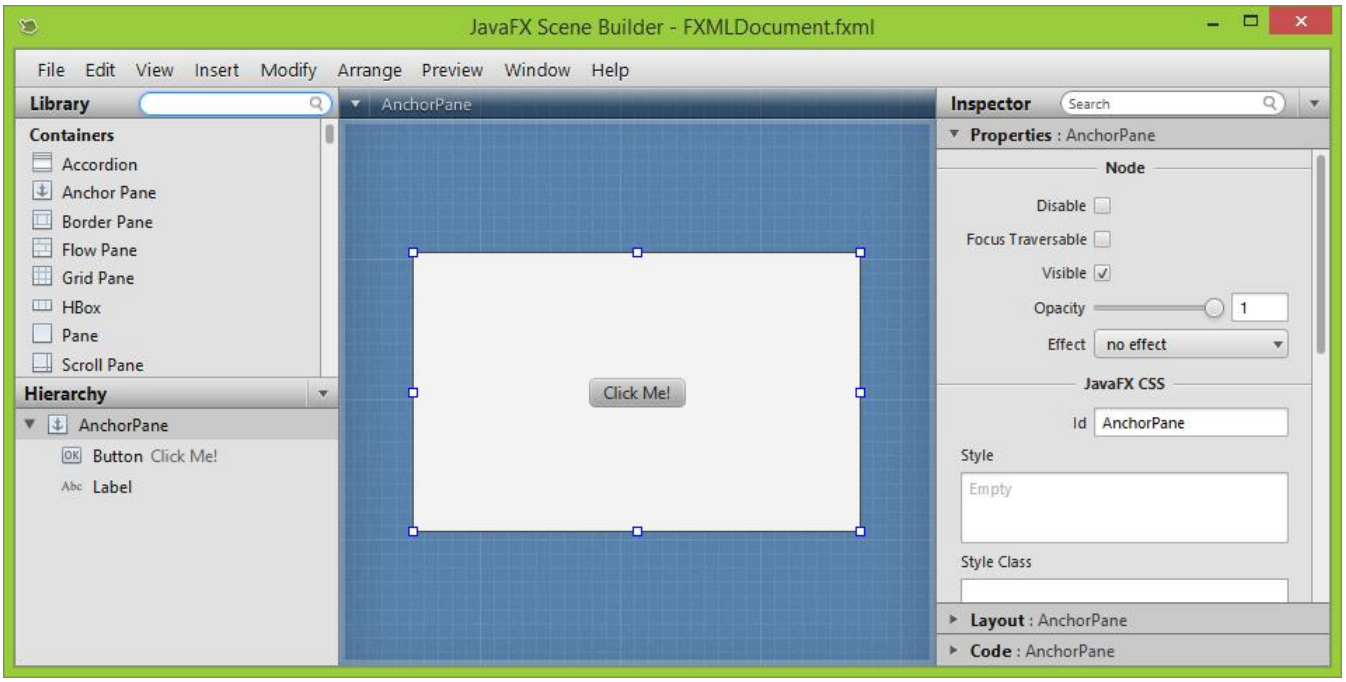


Struktura JavaFX projektu

Projekt obsahuje 3 soubory, popišme si je.

FXMLDocument.fxml

Soubor obsahuje FXML kód, který popisuje, jak formulář vypadá. Pokud na soubor v oknu Projects 2x klikneme a máme nainstalovaný JavaFX Scene Builder, otevře se formulář právě v tomto nástroji:

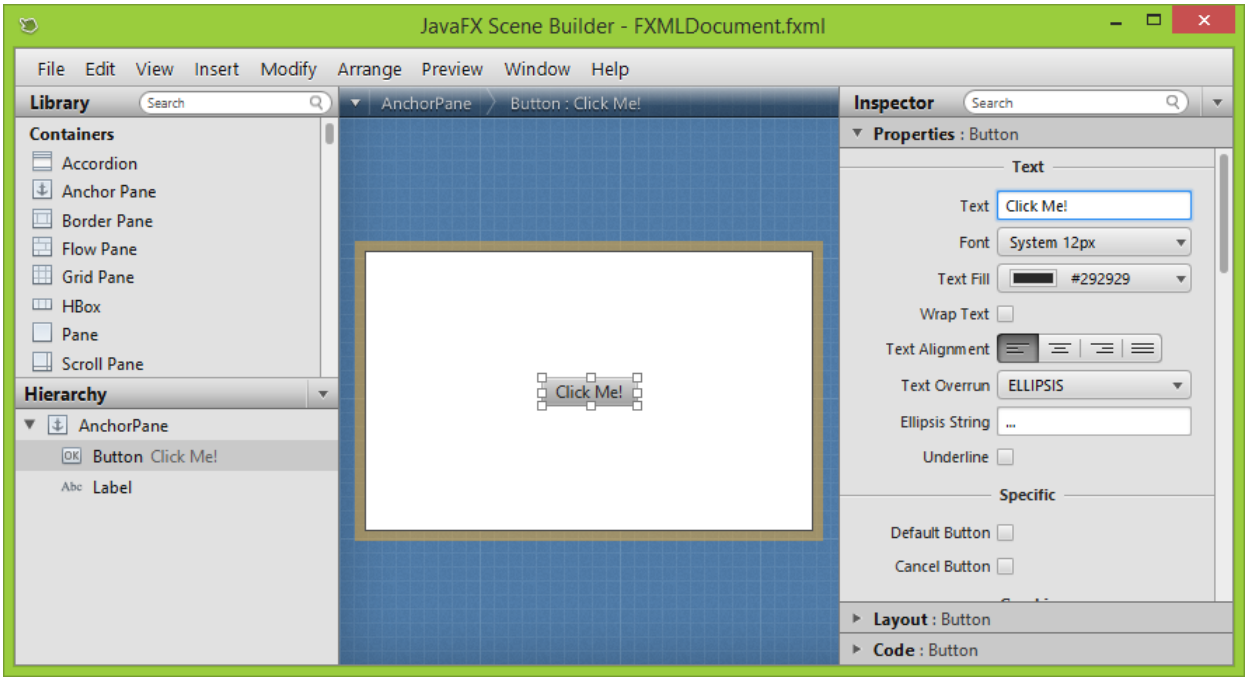


Pozn. Pokud jste JavaFX Scene Builder instalovali, když byl NetBeans spuštěný, je ho nejprve nutné restartovat.

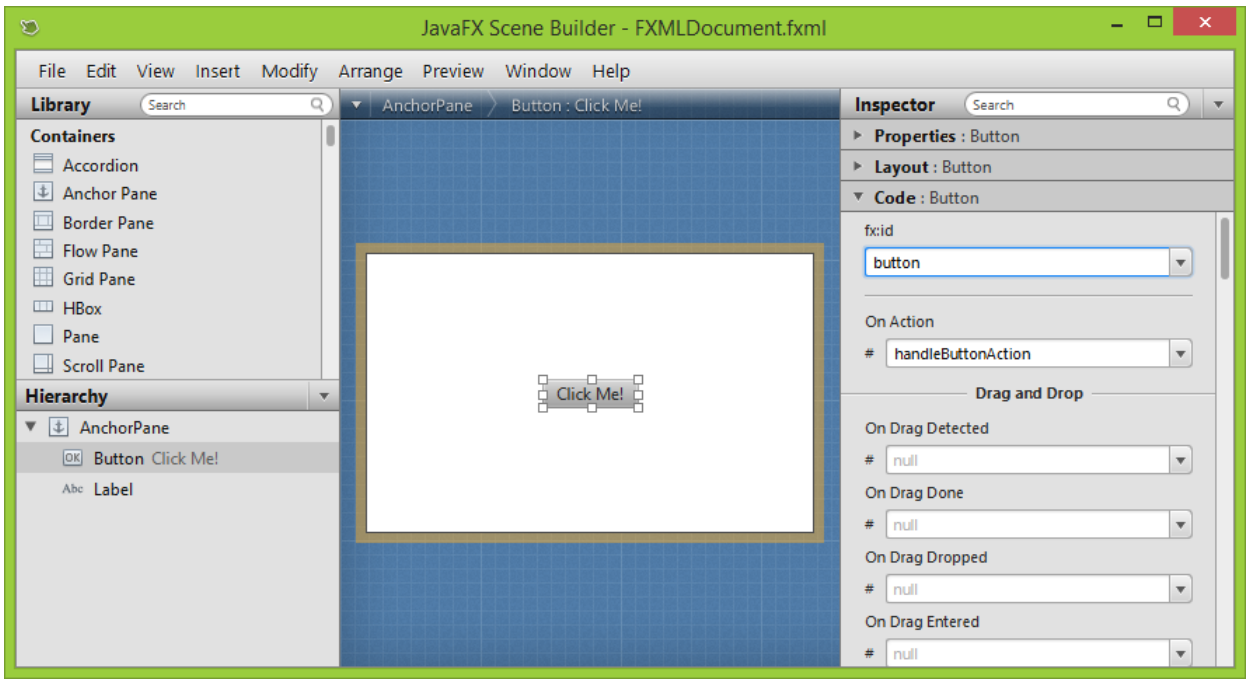
V prostředním sloupci okna vidíme, jak formulář vypadá.

V levé části okna se nachází Library. To je knihovna komponent, které na formulář můžeme přesouvat. Pod ní, v oknu Hierarchy, vidíme stromovou strukturu komponent, které jsou na formuláři vložené. V našem případě se jedná o Button (tlačítko) a Label (textový popisek).

V pravém sloupci vidíme Inspector. V něm vidíme vlastnosti označené komponenty. Zkuste si v prostředním sloupci kliknout na Button a potom na Label pod ním. Vidíme, že v záložce properties je vidět např. text tlačítka.



Záložky Layout si zatím nebudeme všímat a přesuneme se do záložky Code. Ta je velmi důležitá. Pokud nějakou komponentu chceme používat v Java kódu, musíme jí přidělit fx:id. To je název proměnné, přes kterou ke komponentě budeme přistupovat. Vidíme, že tlačítko má fx:id button. Níže vidíme také přiřazení událostí, konkrétně události On Action metodě handleButtonAction.



Vrátíme se do NetBeans a místo dvojkliku na soubor FXMLDocument.fxml na něj klikneme pravým tlačítkem a zvolíme Edit. NetBeans nám otevře přímo obsah dokumentu, který vypadá takto:

```
<?xml version="1.0" encoding="UTF-8"?>

<?import java.lang.*?>
<?import java.util.*?>
<?import javafx.scene.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>

<AnchorPane id="AnchorPane" prefHeight="200" prefWidth="320"
xmlns:fx="http://javafx.com/fxml"
fx:controller="hellofx.FXMLDocumentController">
    <children>
        <Button layoutX="126" layoutY="90" text="Click Me!"
onAction="#handleButtonAction" fx:id="button" />
        <Label layoutX="126" layoutY="120" minHeight="16" minWidth="69"
fx:id="label" />
    </children>
</AnchorPane>
```

Tento kód nám JavaFX Scene Builder generuje podle toho, co v něm naklikáme a ručně do něj nebudeme zasahovat. Přesto si ho popíšeme, abychom chápali, jak technologie funguje.

Vidíme zde XML hlavičku, dále nějaké importy balíčků s komponentami a poté AnchorPane, což je vlastně okno aplikace. K tomu je přiřazený javovský kontroler, ke kterému se hned vrátíme. AnchorPane má v sobě element children a v něm komponenty, které jsou na něm vložené. V našem případě je to Button a Label. Vidíme, že atributy opravdu souhlasí s tím, co jsme viděli v JavaFX Scene Builderu.

FXMLDocumentController.java

Kontroler je javovský kód, který formulář obsluhuje. V našem případě vypadá takto:

```
public class FXMLDocumentController implements Initializable {

    @FXML
    private Label label;

    @FXML
    private void handleButtonAction(ActionEvent event) {
        System.out.println("You clicked me!");
        label.setText("Hello World!");
    }

    @Override
    public void initialize(URL url, ResourceBundle rb) {
        // TODO
    }

}
```

Kontroler obsluhuje formulář, umožňuje měnit vlastnosti komponent a reagovat na jejich události.

Všimněte si proměnné label. Proměnná je typu Label a je označená anotací @FXML. Atributy s touto anotací reprezentují komponenty na formuláři. Musí se samozřejmě jmenovat stejně, jako fx:id komponenty. Java tuto proměnnou poté sama "propojí" s formulářem a my se nemusíme o nic starat.

Co se týče metod, máme zde initialize(), která slouží k inicializaci formuláře, zde v ní není žádný kód.

Metoda handleButtonAction() se zavolá ve chvíli, kdy se klikne na tlačítko. Je tomu tak samozřejmě proto, že je to u tlačítka v FXML souboru nastavené. Metoda vypíše text do konzole a také změní text labelu na formuláři.

HelloFX.java

Soubor HelloFX.java obsahuje spustitelné metody.

```
public class HelloFX extends Application {

    @Override
    public void start(Stage stage) throws Exception {
        Parent root =
FXMLLoader.load(getClass().getResource("FXMLDocument.fxml"));

        Scene scene = new Scene(root);
```

```
        stage.setScene(scene);
        stage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }

}
```

Důležitá je pro nás metoda start(). Vidíme, že pomocí FXMLLoaderu načte FXML soubor a na jeho základě vytvoří tzv. scénu. Scéna je v podstatě formulář. Následně objektu stage (jeviště) nastavíme tuto scénu a zobrazíme ji.

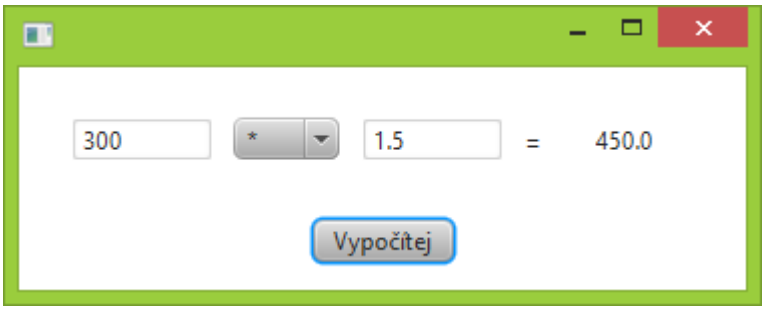
V našich aplikacích budeme editovat v podstatě jen kontroler a fxml přes JavaFX scene builder, stejně je však důležité, že již víme, jak aplikace funguje uvnitř.

Příště si vytvoříme svou první jednoduchou aplikaci v JavaFX, bude se jednat o kalkulačku. Zdrojové kódy dnešního projektu jsou v příloze ke stažení. Bude tomu tak u všech dílů.

2. díl - Jednoduchá kalkulačka v JavaFX

V minulém dílu našeho seriálu tutoriálů o JavaFX jsme si vysvětlili standardní HelloWorld projekt v NetBeans. Dnes si vytvoříme svou první vlastní aplikaci v JavaFX, bude to jednoduchá kalkulačka.

Založte si nový projekt s názvem KalkulackaFX. Rovnou si ukažme, jak bude naše výsledná aplikace vypadat:



Návrh formuláře

Aplikaci započneme návrhem formuláře. Otevřeme FXMLDocument.fxml v JavaFX Scene Builderu a z formuláře odstraníme Button a Label.

Na prázdný formulář natahneme následující komponenty z Library.

TextField

Jak asi tušíte, TextField slouží k zadávání textu. My ho využijeme i k zadání čísel. Na formulář ho vložíme 2x.

ComboBox

ComboBox je rozbalovací nabídka, která nám umožňuje vybrat z několika předdefinovaných hodnot. Nám poslouží k výběru početní operace (sčítání, odčítání, násobení, dělení).

Label

Textové popisky budeme potřebovat dva. Jeden pro znak "rovná se" a druhý pro výsledek, který kalkulačka vypočítala. Labelům v Properties nastavte text na "=" a "0".

Button

Na formulář vložíme poslední komponentu, kterou bude tlačítko. Nastavíme mu text na "Vypočítej".

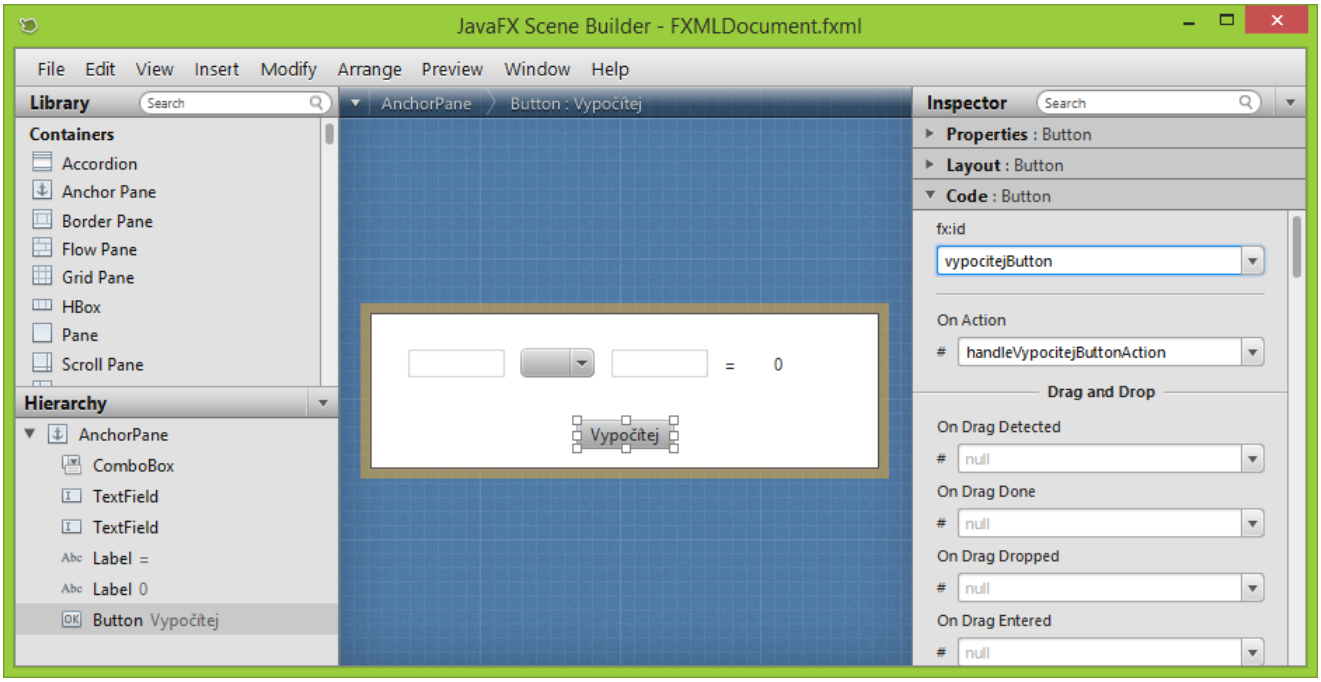
Provázání kontroleru a FXML

Přesuneme se do java kontroleru. V kódu budeme potřebovat načíst hodnoty z textových polí a vybranou položku z ComboBoxu. Také budeme potřebovat vypsat výsledek do labelu a reagovat na kliknutí na tlačítko. Z toho vyplývá, že v kontroleru potřebujeme s těmito komponentami pracovat. Založíme si pro ně ve třídě tedy atributy s anotací @FXML. Anotace musí být opravdu nad každým atributem. Zároveň si ve třídě připravíme obslužnou metodu pro naše tlačítko. Třída kontroleru bude vypadat asi takto:

```
public class FXMLDocumentController implements Initializable {

    @FXML
    private TextField cislo1TextField;
    @FXML
    private TextField cislo2TextField;
    @FXML
    private ComboBox operaceComboBox;
    @FXML
    private Button vypocitejButton;
    @FXML
    private Label vysledekLabel;

        @FXML
    private void handleVypocitejButtonAction(ActionEvent event) {
        }

        @Override
    public void initialize(URL url, ResourceBundle rb) {
    }
```

```
}
```

Původní label a tlačítko s obslužnou metodou jsme ze třídy odstranili. Kód si v NetBeans uložte a přesuňte se do JavaFX Scene Builderu. Jednotlivým komponentám na formuláři přiřadíme jejich fx:id podle toho, jak jsme si pojmenovali atributy v kontroleru. Celkem tedy přiřadíme 5 fx:id. U tlačítka nesmíme zapomenout přiřadit i obslužnou metodu.



JavaFX Scene Builder pracuje s java kódem kontroleru. Pokud zadáme název neexistujícího atributu, upozorní nás na to chybovou hláškou. Javovský kód samozřejmě musí být uložený, jinak to nemá šanci poznat. To je bohužel nevýhoda toho, že se aplikace tvoří ve dvou různých nástrojích.

Obsluha formuláře

Konečně máme vše připraveno k tomu, abychom se mohli věnovat samotné obsluze formuláře.

Naplnění ComboBoxu

Jako první budeme potřebovat nějaké položky do ComboBoxu. Proto upravíme inicializační metodu kontroleru do následující podoby:

```
@Override
public void initialize(URL url, ResourceBundle rb) {
        ObservableList<String> operace =
FXCollections.observableArrayList("+", "-", "*", "/");
        operaceComboBox.setItems(operace);
        operaceComboBox.getSelectionModel().selectFirst();
}
```

Nejprve vytvoříme kolekci položek. V JavaFX se pracuje s tzv. observable collections, což by se dalo přeložit jako pozorovatelné kolekce. Jakmile použijeme ObservableArrayList a nastavíme ho jako zdroj dat nějaké komponentě formuláře, tato komponenta si svůj obsah sama aktualizuje podle toho, jak se kolekce mění. Tomuto chování se někdy říká binding a ušetří nám spoustu starostí a kódu, který bychom jinak museli pro obnovování formulářů napsat. V našem případě sice položky měnit nebudeme, nicméně komponenty v JavaFX jsou implementované tak, že pracují s observable kolekcemi. Ještě dodám, že se kolekce vytvářejí pomocí továrních (statických) metod na třídě FXCollections.

Nastavení položek pomocí metody setItems() je triviální. Aby byla po spuštění aplikace hned vybraná první položka (sčítání), musíme si získat tzv. selectionModel(), který se stará o vybrané položky a na tom vybranou položku nastavit.

Obsluha tlačítka

Zbývá již jen naprogramovat obslužnou metodu tlačítka. Její kód bude následující:

```
@FXML
private void handleVypocitejButtonAction(ActionEvent event) {
        double cislo1 = Double.parseDouble(cislo1TextField.getText());
        double cislo2 = Double.parseDouble(cislo2TextField.getText());
        String operace =
(String)operaceComboBox.getSelectionModel().getSelectedItem();
        double vysledek = 0;
        switch (operace)
        {
                case "+":
                        vysledek = cislo1 + cislo2;
                        break;
                case "-":
                        vysledek = cislo1 - cislo2;
                        break;
                case "*":
                        vysledek = cislo1 * cislo2;
                        break;
                case "/":
                        if (cislo2 != 0)
                        vysledek = cislo1 / cislo2;
                        break;
        }
        vysledekLabel.setText(String.valueOf(vysledek));
}
```

Jako první si naparsujeme čísla z obou TextFieldů. Text získáme metodou getText(), vyjímky při parsování pro zjednodušení nebudeme ošetřovat.

Získání vybrané operace provedeme opět přes SelectionModel, přes metodu getSelectedItem(), která vrací vybranou položku v ComboBoxu. Položku musíme přetypovat na String. Podobně bychom mohli využít metodu getSelectedIndex(), která vrátí číselný index vybrané položky.

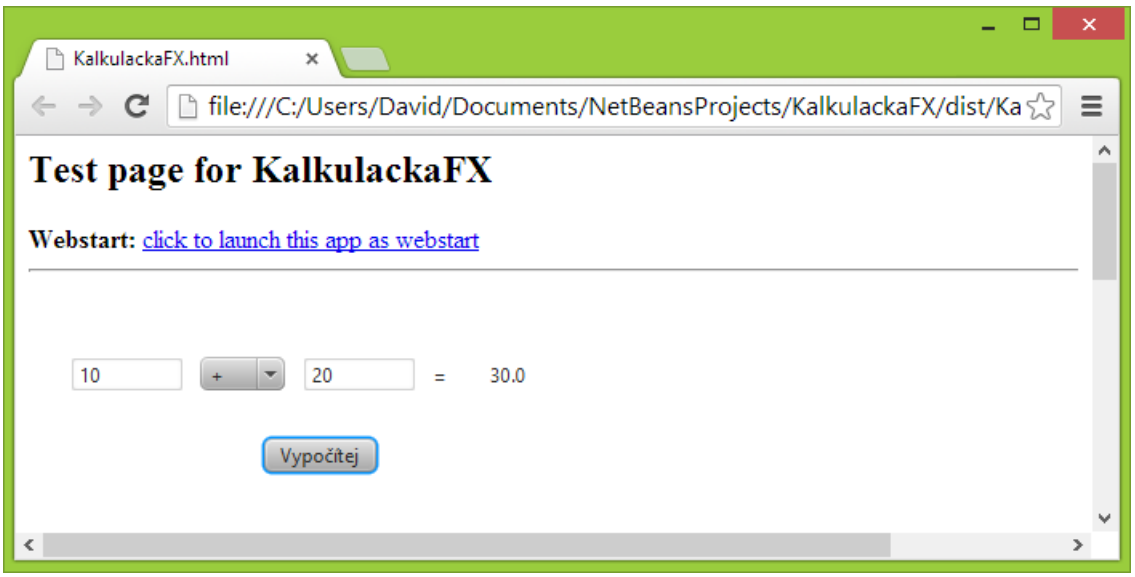
Získávání přímo objektů je však velmi praktické, v ComboBoxu totiž můžeme mít v pokročilejších aplikacích např. instance uživatelů. Vše si během seriálu ukážeme.

Následuje switch, který podle operace vypočítá výsledek. Všimněte si ošetření dělení nulou. Ideálně bychom při dělení nulou měli zobrazit nějakou chybovou hlášku. JavaFX bohužel zatím tuto funkcionalitu stále neobsahuje a musí se obcházet, což si ukážeme v dalších dílech seriálu.

Již nezbývá nic jiného, než nastavit výsledek příslušnému labelu. Samozřejmě ho musíme nejprve převést na String. Můžeme se těšit z hotové aplikace.

Distribuce

Na závěr si řekněme, jak aplikaci šířit. V NetBeans je vedle tlačítka Play tlačítko Clean and Build Project (shift + F11). Po jeho stisknutí se ve složce s vaším projektem vytvoří složka dist a v ní spustitelný JAR s vaší aplikací. Můžete ji poslat známým a pochlubit se. Dokonce je tam i vygenerovaná HTML stránka, kde je vaše kalkulačka jako applet. Můžete si ji dát i na svůj web.



Příště začneme programovat upomínač narozenin přátel. Bude se jednat o praktickou a středně pokročilou aplikaci, na které se naučíme spoustu zajímavých technik. Zdrojový kód dnešního projektu je ke stažení níže pro případ, že by vám něco nefungovalo.

3. díl - Upomínač narozenin v JavaFX - Návrh formuláře

V minulém dílu našeho seriálu tutoriálů o JavaFX jsme si naprogramovali jednoduchou kalkulačku. Dnes započneme tvorbu složitější aplikace, která bude sloužit k upomínání narozenin přátel.

Návrh formuláře

Začneme tradičně s návrhem formuláře. Pořádně si u toho zaklikáme a proto jsem se snažil do článku vložit co nejvíce obrázků, abyste si mohli kontrolovat, že postupujete správně. Vysvětleme si však nejprve absolutní a relativní pozicování.

Absolutní pozicování

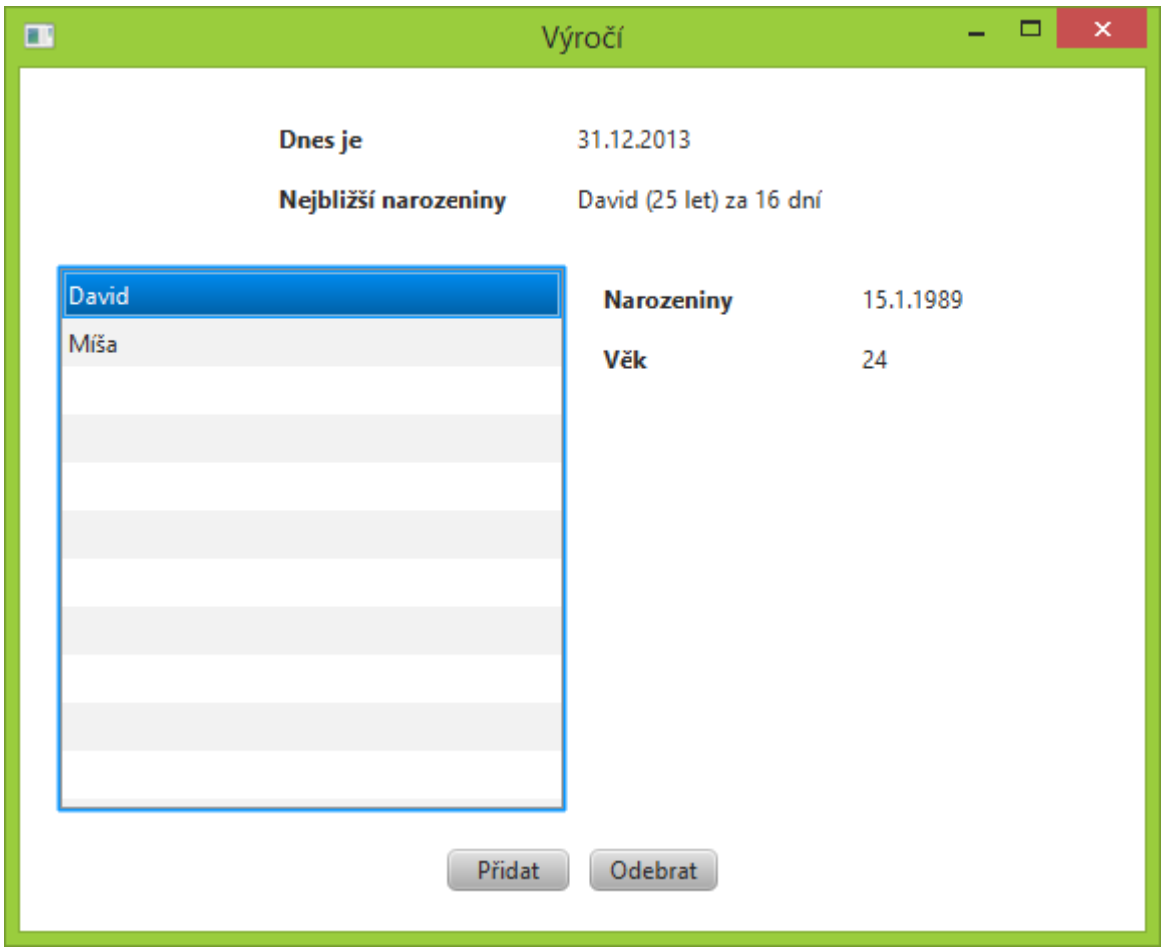
V minulé aplikaci jsme si formulář naklikali tak, jak nás to napadlo. Jednotlivé komponenty jsme umístili přesně na určitou pozici na formuláři. Tomuto přístupu se říká absolutní pozicování. Jeho nevýhodou je, že aplikace poté nereaguje na rozšíření okna. Drtivá většina aplikací má proto komponenty na formuláři pozicované relativně.

Relativní pozicování

Pokud má komponenta relativní pozici, znamená to, že se tato pozice vztahuje k nějaké jiné komponentě, ve které je vložena. Nejedná se tedy přímo o pozici v hlavním okně, ale o pozici v nadřazené komponentě. V Library v JavaFX Scene Builderu nalezneme širokou nabídku tzv. kontejnerů. To jsou komponenty, do kterých se vkládají další komponenty. Právě z nich formulář poskládáme a ve výsledku docílíme kvalitního uzpůsobení velikosti okna. To je v dnešní době velmi důležité, jelikož aplikaci můžeme cílit na telefony, tablety nebo stolní PC. Díky přenositelnosti Javy bude fungovat ta samá aplikace všude a díky relativnímu pozicování použijeme na každém zařízení ten samý formulář, i když velikosti displejů jsou různé.

Příprava layoutu

V relativním pozicování se často o rozložení prvků na formuláři hovoří jako o layoutu. Pojďme si ho vytvořit. Vytvořte si novou JavaFX FXML Application a pojmenujte ji UpominacNarozenin. Rovnou si na začátku ukažme, jak bude výsledný formulář aplikace vypadat:



Nový FXML dokument si rovnou otevřeme v JavaFX Scene Builderu a odstraníme z něj vygenerovaný Button a Label.

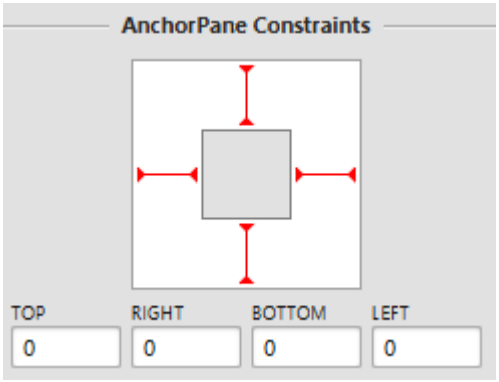
VBox

Jako první do formuláře vložíme VBox. To je kontejner, do kterého můžeme vkládat další komponenty. Tyto komponenty se řadí pod sebe (vertikálně).

VBox roztahněte po celém formuláři a v Inspectoru v záložce Properties nastavte Alignment na CENTER. Tím se budou komponenty vložené ve VBoxu centrovat. Přejděte do záložky Layout. Komponentě zde nastastavíme kotvy a okraje.

Kotvy

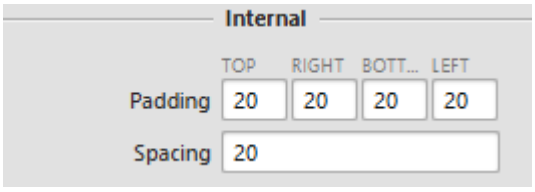
Kotvy (Anchors) určují jak se komponenta přichytává k okrajům rodičovské komponenty. Rodičovská komponenta je ta komponenta, ve které je vložená, v našem případě je to formulář (přesněji AnchorPane). My budeme chtít, aby se VBox přichytil ke všem čtyřem okrajům formuláře. Když budeme formulář roztahovat, bude se roztahovat i VBox. Jednotlivé kotvy aktivujeme kliknutím na příslušnou přerušovanou čáru na obrázku dvou čtverců. Do políček TOP, RIGHT, BOTTOM, LEFT vložte samé nuly. VBox tak bude těsně přilepený k okraji formuláře.



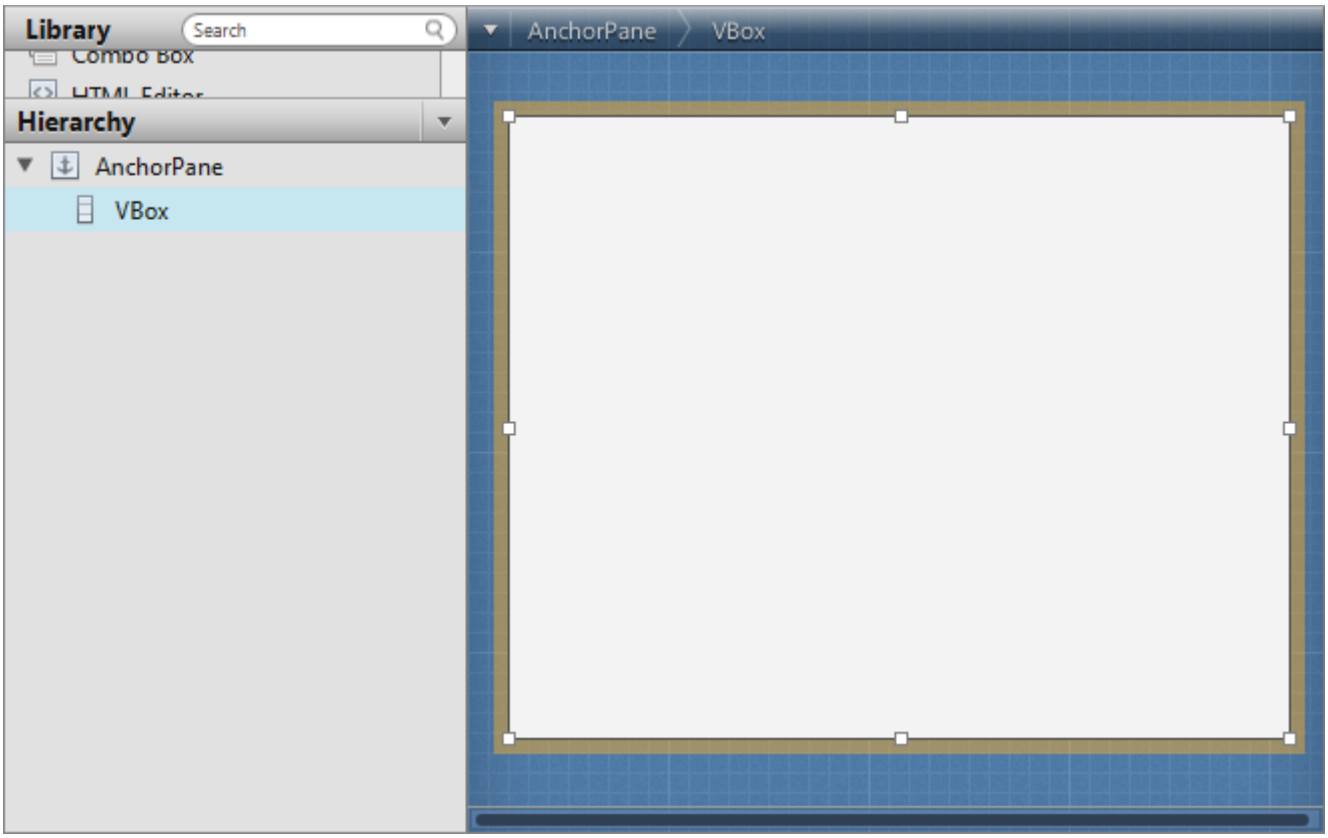
Okraje

Okraje určují velikost volného místa. Můžeme nastavit Padding, což je volné místo okolo obsahu komponenty. Dále také Spacing, což je volné místo mezi komponentami, které jsou ve VBoxu vložené.

Všechny okraje nastavíme na hodnotu 20. Výsledkem bude, že na sobě nebudou komponenty nalepené, což by nevypadalo dobře.



Náš formulář prozatím vypadá takto:



GridPane

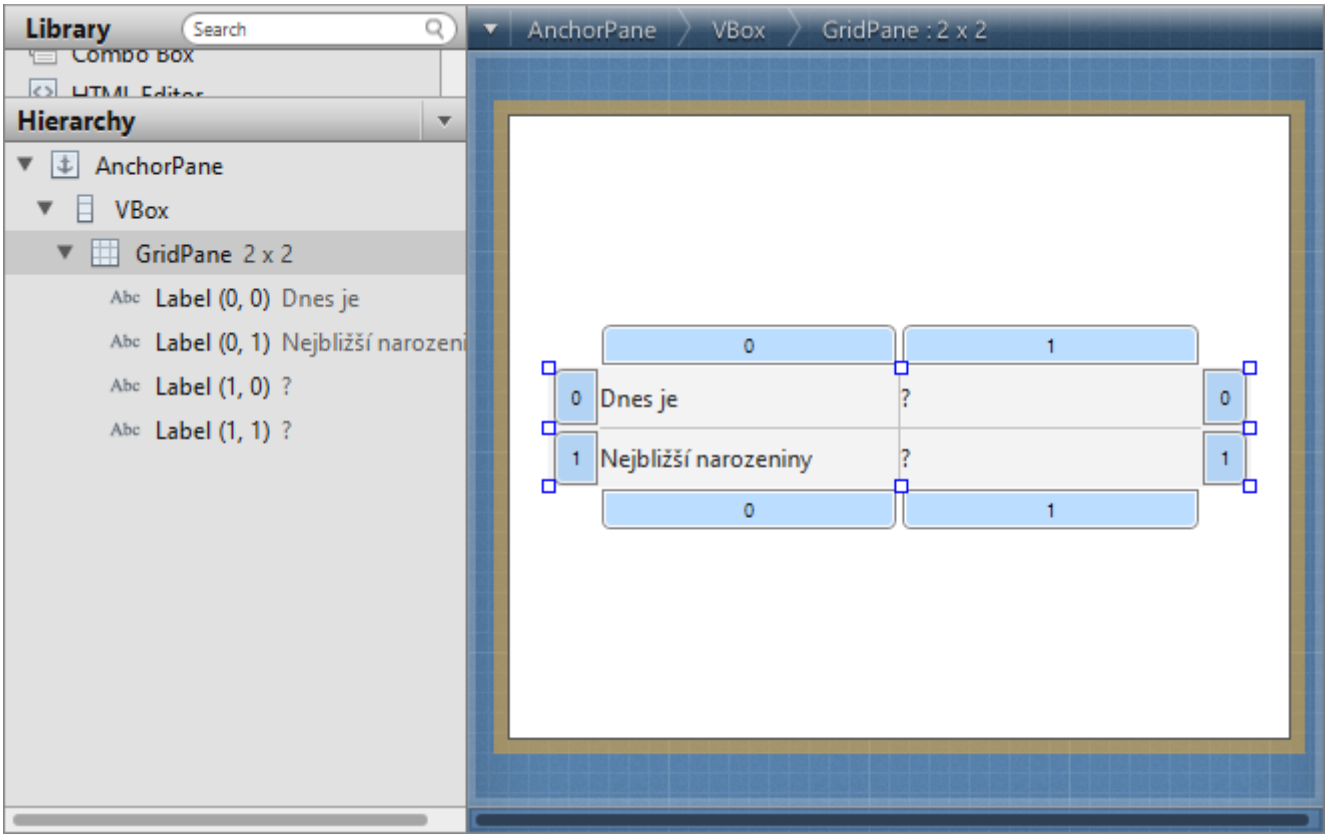
K zobrazení dnešního data a nejbližších narozenin použijeme 4 Labely. Aby byly hezky uspořádané, vložíme je do GridPane. To je další kontejner, který komponenty řadí do tabulky.

GridPane vložíme do VBoxu. Jako výchozí má vždy 2 sloupce a 3 řádky. Nám stačí tabulka 2x2 a proto poslední řádek označíme kliknutím na modré číslo 2 a klávesou delete ho odstraníme. Pokud bychom v nějaké aplikaci naopak chtěli řádek nebo sloupec přidat, uděláme to pomocí kontextového menu (klikneme na GridPane pravým tlačítkem).

Alignment nastavíme na CENTER, v záložce Layout nastavíme Min Height (minimální výšku) na 60. Následně oběma sloupcům tabulky nastavíme Max Width na 150. Sloupec označíme vždy kliknutím na jeho modré číslo.

Náš Grid Pane tedy bude široký maximálně 300 a vysoký minimálně 60.

Do každé buňky GridPane přetáhněte jeden Label a nastavte jim texty podle obrázku:



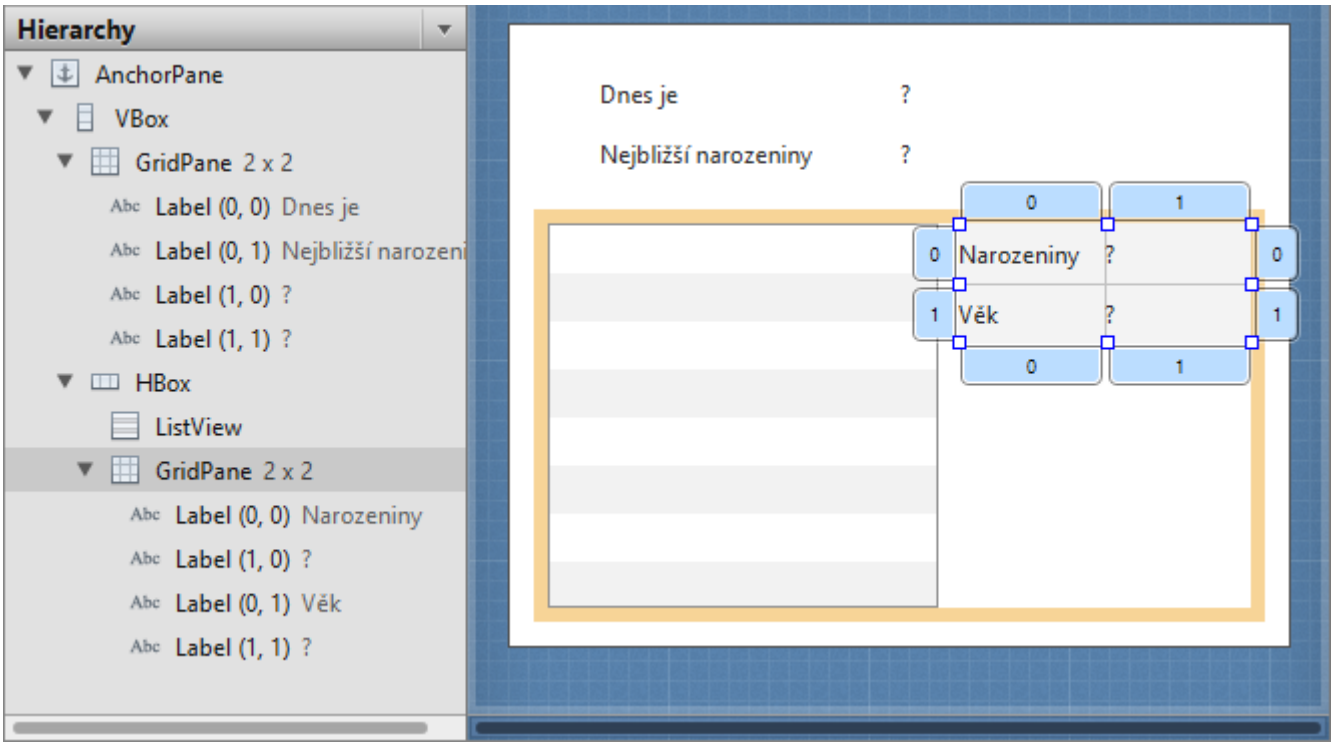
HBox

Jako druhou komponentu do VBoxu vložíme HBox. Ten se používá pro řazení komponent vedle sebe (horizontálně). V záložce Layout mu nastavíme Spacing na 10 a Vgrow na SOMETIMES. Pomocí Vgrow tak nastavíme, aby se HBox roztahoval na výšku s formulářem.

ListView

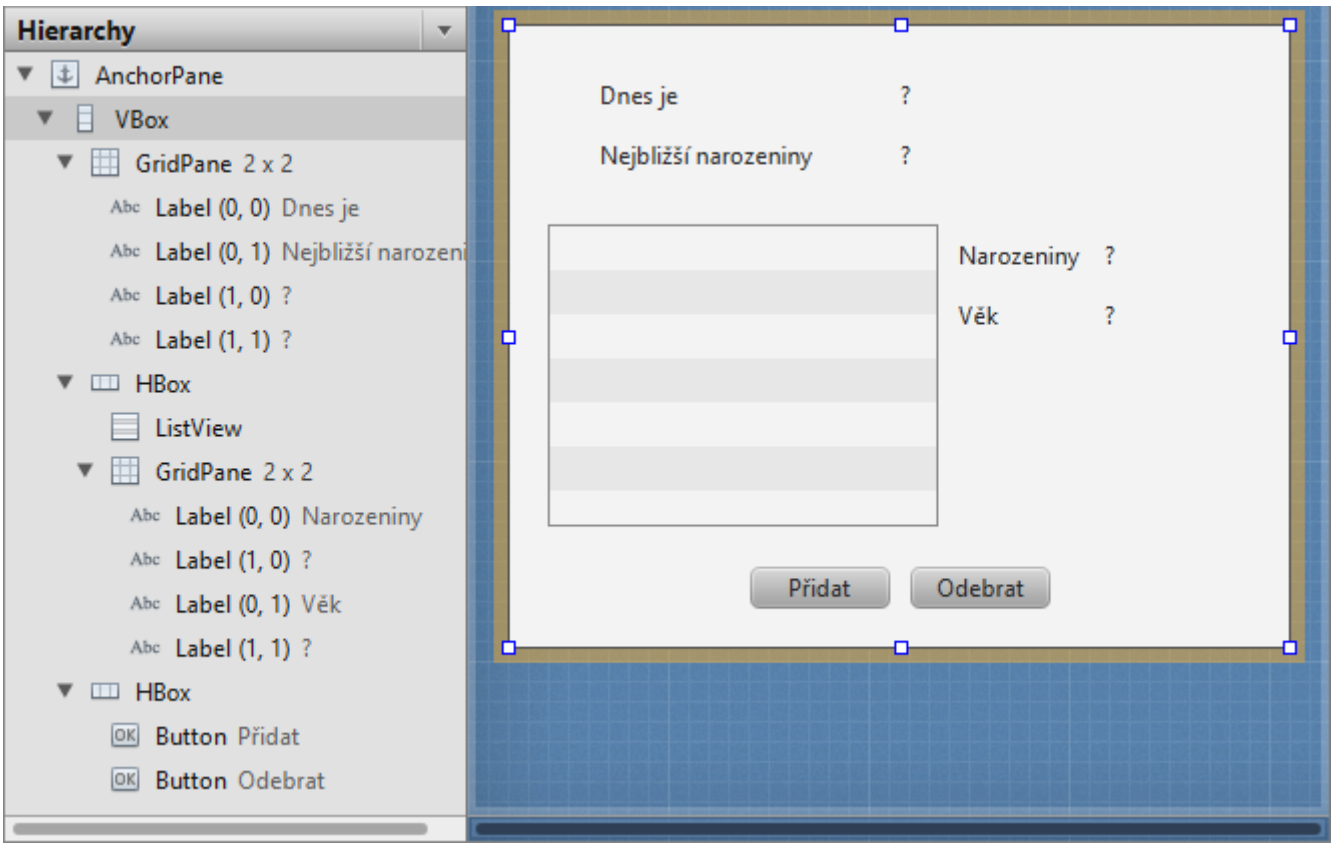
Jako první komponentu do HBoxu vložíme ListView. To je jednoduchý seznam položek, my ho budeme používat pro zobrazení seznamu přátel. ListView nastavíme Pref Height i Pref Width na hodnotu USE_COMPUTED_SIZE a Hgrow na SOMETIMES. Bude se tak roztahovat co nejvíce to půjde. Rodičovskému HBoxu nastavíme totéž.

Jako druhou komponentu do HBoxu vložíme další GridPane. Ten bude opět 2x2 (2 sloupce a 2 řádky). Protože nechceme, aby se roztahoval po celé výšce, nastavíme mu v záložce Layout Max Height na 60. Do každé buňky vložíme Label s textem podle obrázku:

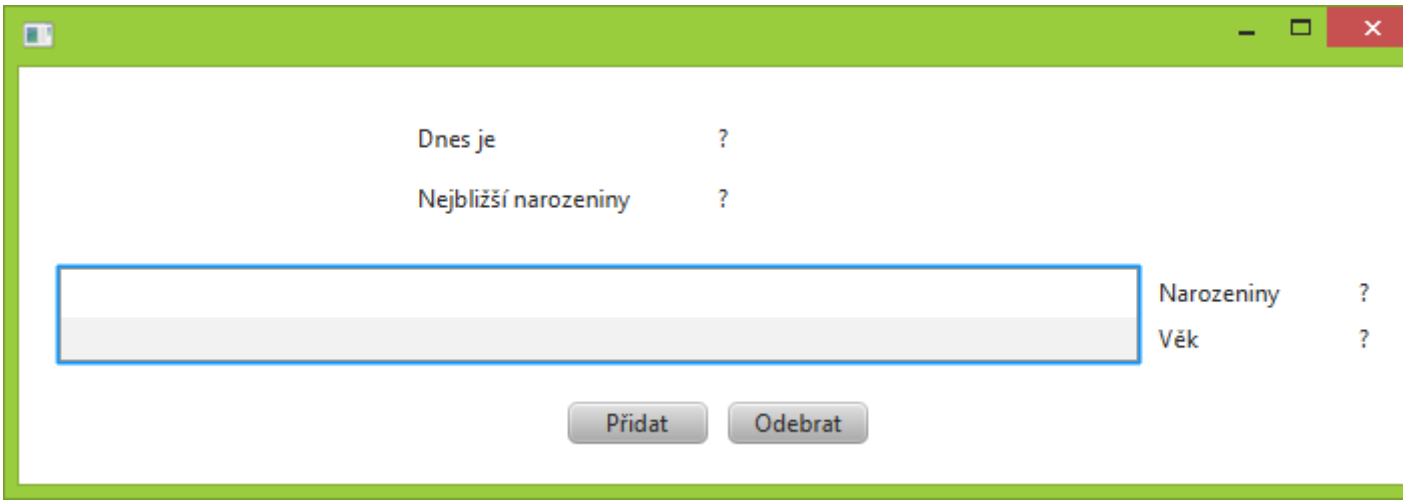


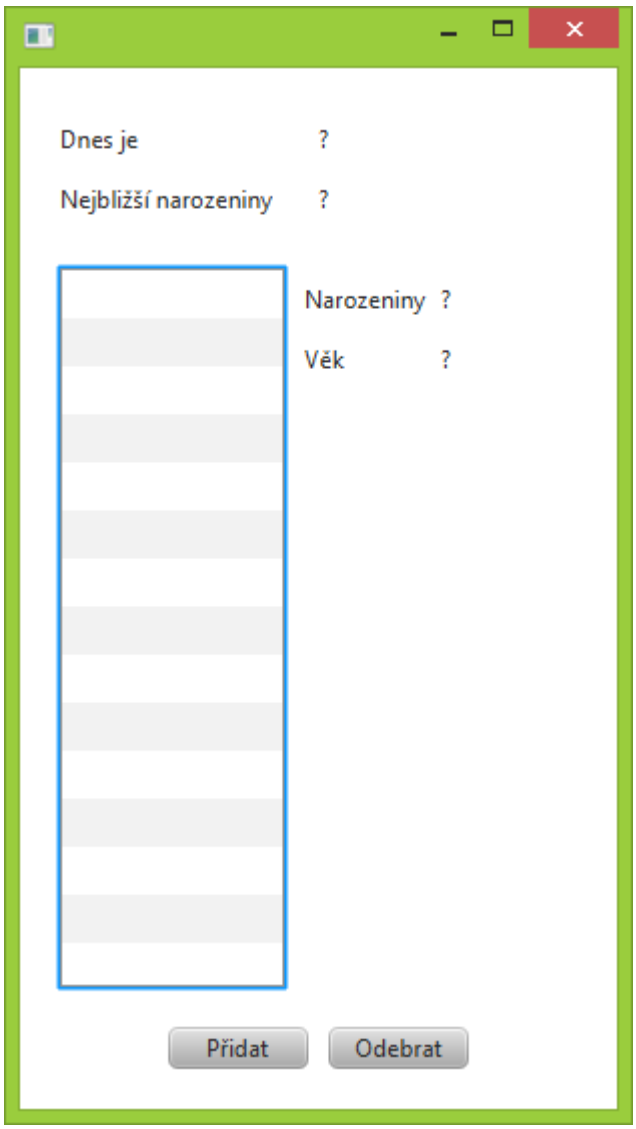
Jako poslední komponentu do VBoxu vložíme další HBox se Spacing 10. V záložce Properties mu nastavíme Alignment na CENTER. Tím se budou položky v něm vložené centrovat. V něm budou obsažena 2 tlačítka (Buttony). Tlačítkům nastavíme texty podle obrázku a Pref Width (v záložce Layout) na 70. Budou tak stejně široká.

Slušelo by se dodat, že Pref Width je synonymum pro šířku. Ta je označena jako Preferred, protože se od skutečné šířky komponenty může lišit. Např. když nastavíme preferovanou šířku na 70 a formulář zmenšíme na 60, výška se nutně sníží s ním (tlačítko nemůže být širší, než formulář, pokud ovšem nemá nastavenou minimální šířku, to by se zmenšení formuláře na takovou velikost nepovolilo).



Náš formulář máme hotový. Zkuste si aplikaci spustit a formulář roztahnout, jeho obsah se velikosti okna přizpůsobuje. (Před spuštěním musíte uložit scénu v JavaFX Scene Builderu a z kontroleru vymazat atributy a metody s @FXML anotací, které tam zbyly po výchozím projektu)





Pro případ, že se vám něco nepovedlo, si ho můžete stáhnout níže a najít si chybu. Příště naprogramujeme základ logické vrstvy aplikace.

4. díl - Upomínač narozenin v JavaFX - Formuláře podruhé

V minulém dílu našeho seriálu tutoriálů o JavaFX jsme si v nástroji JavaFX Scene Builder navrhli hlavní formulář aplikace. Dnes budeme s formuláři pokračovat.

V projektu bude figurovat třída Osoba, proto si ji k němu přidáme a zatím ji ponecháme prázdnou.

```
public class Osoba
{
}
```

Příprava kontroleru

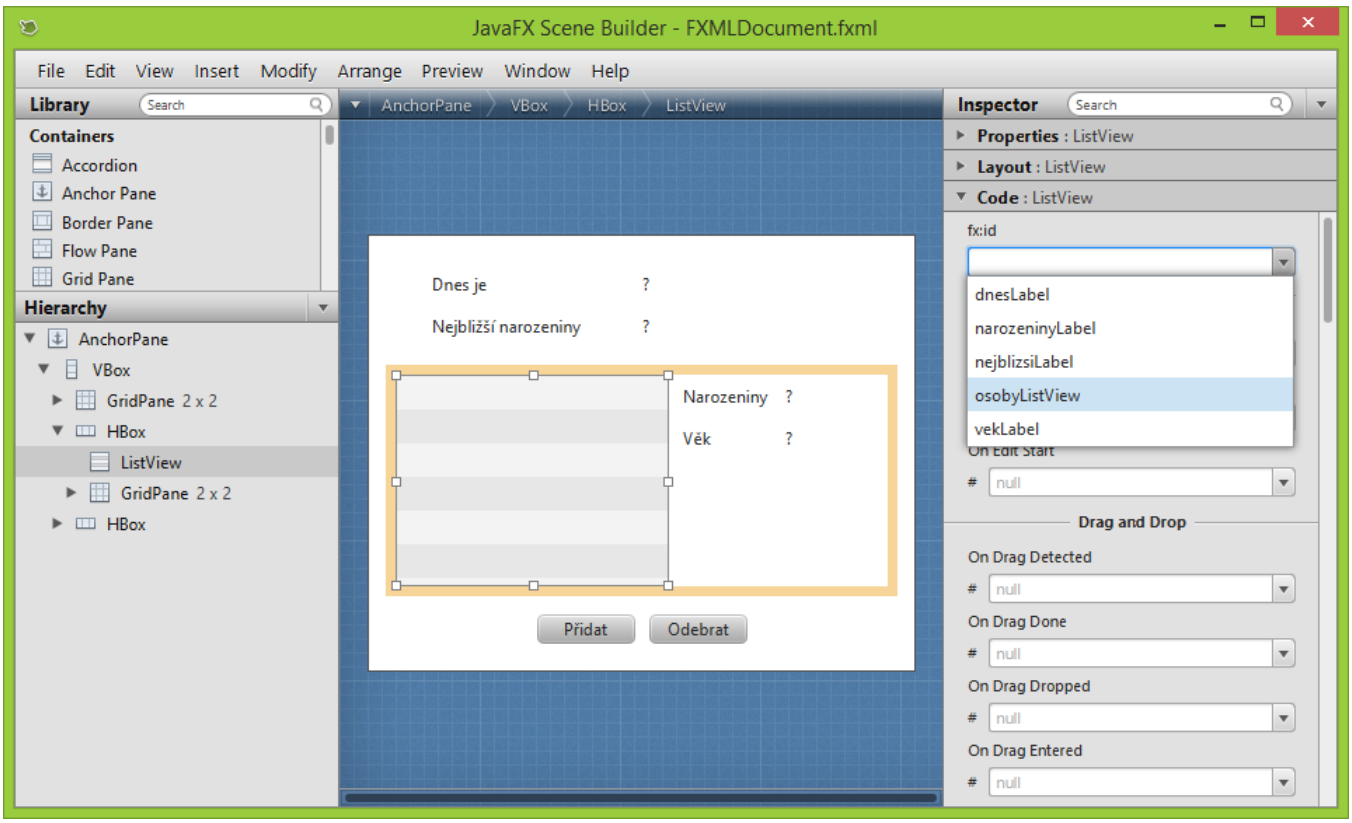
Přejděme do kontroleru formuláře a vytvořme atributy s @FXML anotací pro komponenty formuláře, které budeme z kódu používat. Rovnou si připravme i obslužné metody. Náš kontroler bude vypadat asi takto:

```
public class FXMLDocumentController implements Initializable {

    @FXML
    private ListView<Osoba> osobyListView;
    @FXML
    private Label dnesLabel;
    @FXML
    private Label nejblizsiLabel;
    @FXML
    private Label narozeninyLabel;
    @FXML
    private Label vekLabel;

    @FXML
    public void handlePridatButtonAction(ActionEvent event) {

    }

    @FXML
    public void handleOdebratButtonAction(ActionEvent event) {

    }

    @Override
    public void initialize(URL url, ResourceBundle rb) {
        // TODO
    }

}
```

Pracovat budeme s ListView a potom se čtyřmi Labely (to jsou ty, které mají na formuláři v textu otazník). Zbytek komponent nebudeme v kontroleru potřebovat a je proto zbytečné si je ukládat. Dále zde máme připravené obslužné metody pro obě tlačítka.

Propojení kontroleru a FXML

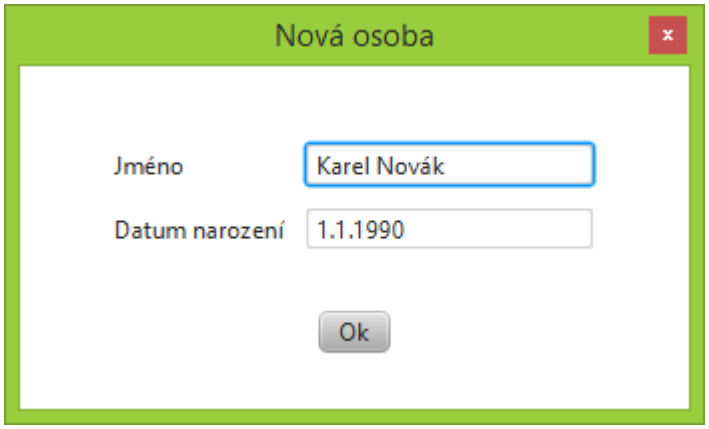
Změny v souboru uložte a přejděte do JavaFX Scene Builderu, kde ListView a čtyřem Labelům nastavte příslušná fx:id. Tlačítkům nastavte obslužné metody, jako jsme to dělali u kalkulačky. Nastavovat fx:id jim nemusíme.



Pozn.: U obslužných metod tlačítek můžeme parametr ActionEvent vynechat. Standardně se tam však uvádí, i když ho zrovna nevyužijeme.

Dialog

V aplikaci budeme dále kromě hlavního formuláře potřebovat dialog pro vytvoření nové osoby. Dialog bude vypadat takto:



Formulář je opravdu jednoduchý. Obsahuje pouze dva TextFieldy a Button. Při úvodu do JavaFX jsme si říkali, že kromě FXML a JavaFX Scene Builderu můžeme formulář vytvořit i jako instance objektů. U tohoto formuláře si to vyzkoušíme.

K projektu si přidáme novou třídu se jménem OsobaDialog, kterou oddědíme od Stage.

```
public class OsobaDialog extends Stage {

}
```

Vytvoření scény

Třídě nejprve dodáme metodu k vytvoření scény. Bude vypadat následovně:

```
private Scene vytvorScenu() {
        VBox box = new VBox();
        box.setAlignment(Pos.CENTER);
        box.setSpacing(20);

        // Mřížka s TextFieldy a Labely
        GridPane grid = new GridPane();
        grid.setAlignment(Pos.CENTER);
        grid.setPadding(new Insets(10));
        grid.setHgap(10);
        grid.setVgap(10);

        // Komponenty
        TextField jmenoTextField = new TextField();
        TextField datumTextField = new TextField();
        Label jmenoLabel = new Label("Jméno");
        Label datumLabel = new Label("Datum narození");

        grid.add(jmenoLabel, 0, 0);
        grid.add(jmenoTextField, 1, 0);
        grid.add(datumLabel, 0, 1);
        grid.add(datumTextField, 1, 1);

        // Tlačítko
        Button tlacitko = new Button("OK");
        tlacitko.setOnAction(new EventHandler<ActionEvent>() {
                @Override public void handle(ActionEvent e) {
                        // Obsluha tlačítka
                }
        });

        box.getChildren().addAll(grid, tlacitko);
        return new Scene(box);
}
```

Metoda dělá v podstatě to, co za nás jinak dělá JavaFX Scene Builder. Tento přístup se u JavaFX také používá, čili když si ho teď popíšeme, budete schopni porozumět dalším JavaFX aplikacím, které FXML z nějakého důvodu nevyužívají.

Na začátku si vytvoříme VBox, kterému nastavíme zarovnání a okraje. Následně vytvoříme GridPane, který opět nastavíme podobným způsobem.

Vytvoříme si komponenty, které naskládáme no buněk GridPane. Uvedeme vždy komponentu, sloupec a řádek, do kterého ji chceme vložit.

Další řádky vytvoří tlačítko a přiřadí mu rovnou i obslužnou metodu, která je zatím prázdná. Pro události v Javě se používá tzv. anonymní třída, pokud jste na ni ještě nenarazili. Anonymní třída zde implementuje rozhraní EventHandler a obsahuje jednu metodu handle(), která se spustí ve chvíli, kdy událost nastane.

Naplněný GridPane a tlačítko přidáme do VBoxu pomocí metody addAll(). Musíme si nejprve získat kolekci komponent ve VBoxu, do které je přidáme. Těmto komponentám se říká child (děti), VBox je jejich rodič (parrent), jelikož jsou v něm obsažené.

Nakonec vytvoříme novou scénu z VBoxu, kterou vrátíme.

Třídě OsobaDialog přidáme následující konstruktor:

```
public OsobaDialog(Window okno) {
        setTitle("Nová osoba");
        setWidth(350);
        setHeight(250);

        initStyle(StageStyle.UTILITY);
        initModality(Modality.WINDOW_MODAL);
        initOwner(okno);
        setScene(vytvorScenu());
}
```

Konstruktor bere v parametru instanci Window. Tu později získáme z libovolné komponenty hlavního formuláře. Jevišti (Stage) nastavíme titulek a rozměry. Následujících několik řádek nastaví scénu jako tzv. modální okno. To je charakterem pomocný formulář, který se zobrazí přes hlavní okno. Nakonec jevišti nastavíme scénu na tu, kterou jsme před chvílí vytvořili.

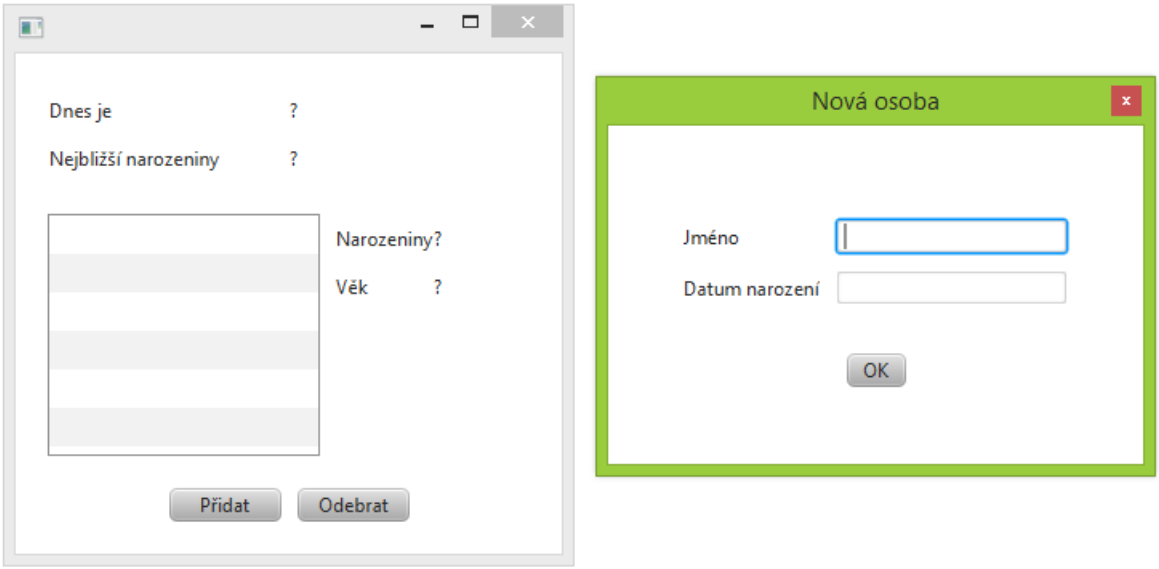
Dialog, který jsme vytvořili, by se mohl používat i pro editaci, ale tu v naší aplikaci zanedbáme, už tak ji budeme programovat ještě několik dílů.

Abychom z té práce vůbec něco viděli, přesuneme se do kontroleru a upravíme obslužnou metodu přidávacího tlačítka do následující podoby:

```
@FXML
public void handlePridatButtonAction(ActionEvent event) {
        OsobaDialog dialog = new
OsobaDialog(dnesLabel.getScene().getWindow());
        dialog.showAndWait();
}
```

Vytváříme zde instanci našeho dialogu. Instanci Window získáme např. z nějakého labelu pomocí metod .getScene().getWindow(). Dialog nakonec zobrazíme metodou showAndWait(). Tím zajistíme, že hlavní okno nebude reagovat do té doby, než se dialog uzavře. U dialogu se toto většinou dělá, již jen proto, aby si uživatel nemohl ten samý dialog vyvolat vícekrát. I když nám by v podstatě nevadilo, kdyby uživatel během zadávání nové osoby aplikaci používal a otevřel třeba další dialog k zadávání nové osoby.

Aplikaci můžeme spustit a kliknout na tlačítko Přidat:



Tím máme tedy formuláře hotové, příště se pustíme do samotné logiky aplikace. Dnešní projekt je se zdrojovými kódy jako vždy ke stažení níže.

5. díl - Upomínač narozenin v JavaFX - Logická vrstva

V minulém dílu seriálu tutoriálů o tvorbě formulářových aplikací v JavaFX jsme navrhli formulář pro naši aplikaci. V tomto dílu se budeme zabývat návrhem logické vrstvy, tedy tříd, které obsahují logiku aplikace.

Datum

Jelikož práce s datem a časem je v Javě poměrně nepohodlná, přidejme si k projektu kratičkou statickou třídu, která nám ulehčí práci.

```
public class Datum {

    private static DateFormat formatData = new SimpleDateFormat("d.M.yyyy");

    public static void nastavPulnoc(Calendar datum)
```

```
    {
        datum.set(Calendar.HOUR_OF_DAY, 0);
        datum.set(Calendar.MINUTE, 0);
        datum.set(Calendar.SECOND, 0);
        datum.set(Calendar.MILLISECOND, 0);
    }

    public static String zformatuj(Calendar datum)
    {
        String datumText = formatData.format(datum.getTime());
        return datumText;
    }

    public static Calendar naparsuj(String datumText) throws ParseException
    {
        Calendar datum = Calendar.getInstance();
        datum.setTime(formatData.parse(datumText));
        return datum;
    }

}
```

Třída obsahuje definici formátu data. Dále umí určité datum nastavit na půlnoc (tedy vynulovat čas a datum ponechat). To se nám velmi hodí při výpočtech, jelikož často budeme chtít počítat s celými dny a čas by nám mohl výpočty rohodit).

Další dvě metody převádí mezi datem a jeho textovou podobou. Datum můžeme načíst z textu (jak ho uživatel zadal) a nebo ho jako text např. vypsat.

Osoba

V naší aplikaci budou zcela jistě figurovat osoby, třídu jsme jim již vytvořili. Pojďme jí dodat implementaci.

Vlastnosti

Osoba bude mít 2 vlastnosti: jméno a narozeniny. Jméno bude String, narozeniny budou typu Calendar. Obě vlastnosti implementujeme jako privátní atributy a vygenerujeme k nim gettery:

```
public class Osoba
{
        private String jmeno;
        private Calendar narozeniny;

        public String getJmeno() {
        return jmeno;
    }
```

```
        public Calendar getNarozeniny() {
        return narozeniny;
    }

}
```

Metody

Osoba bude mít několik metod, nyní se však omezme pouze na její konstruktor, abychom naši aplikaci co nejdříve dostali do spustitelné podoby. Později ji doplníme. Do třídy tedy přidáme parametrický konstruktor.

Konstruktor

Kromě nastavení vlastností instance bude mít konstruktor také za úkol tyto vlastnosti zvalidovat. Ukažme si jeho kód:

```
public Osoba(String jmeno, Calendar narozeniny) throws
IllegalArgumentException
{
        Datum.nastavPulnoc(narozeniny);
        if (jmeno.length() < 3)
                throw new IllegalArgumentException("Jméno je příliš krátké");
        if (narozeniny.after(Calendar.getInstance()))
                throw new IllegalArgumentException("Datum narození nesmí být v
budoucnosti");

        this.jmeno = jmeno;
        this.narozeniny = narozeniny;
}
```

Nejprve vynulujeme čas narozenin, aby nám nedělal při výpočtech problémy, pracovat budeme pouze s celými dny. Následně ověříme, zda není jméno příliš krátké nebo zda nejsou zadané narozeniny v budoucnosti. Pokud některá ze situací nastane, vyhodíme výjimku IllegalArgumentException a do jejího konstruktoru vložíme zprávu pro uživatele.

Pokud jste s výjimkami ještě nepracovali, vůbec to nevadí. Stačí vám vědět, že je to způsob, jakým se v objektových aplikacích pracuje s chybami, zejména těmi, které jsou způsobené zadáním špatné hodnoty od uživatele nebo které vznikly při práci se soubory. Vyvolání výjimky metodu okamžitě ukončí. Jak na výjimku reagovat si ukážeme dále v seriálu. Výjimky budeme vždy vyhazovat v logických třídách.

toString

Jelikož budeme osoby vypisovat, přepíšeme metodu toString() tak, aby vracela jméno osoby:

```
@Override
```

```
public String toString()
{
        return jmeno;
}
```

Právě tato metoda je později použita ListView k výpisu jeho položek.

Správce osob

Poslední logickou komponentou aplikace bude správce osob. Třída se bude starat o osoby, bude je umět přidávat, odebírat a jejich seznam ukládat do souboru a opětovně načíst. Konečně bude umět mezi osobami vyhledat tu, která má nejbližší narozeniny.

Přidejte si k projektu tedy třídu SpravceOsob.

Atributy

Jedinou vlastností třídy je seznam osob. Seznam je typu ObservableList. S touto kolekcí jsme se v seriálu již setkali. Díky tomu, že budeme mít osoby právě v této kolekci, docílíme automatického obnovení všech komponent na formuláři, které s touto kolekcí pracují. Jakmile tedy např. přidáme novou osobu do ObservableListu, objeví se i v ListView. Této technice se říká binding a již víme, že nám ušetří spoustu práce s ručním obnovováním formuláře. K ObservableListu vygenerujeme getter:

Třída zatím vypadá takto:

```
public class SpravceOsob {

    private ObservableList<Osoba> osoby = FXCollections.observableArrayList();

    public ObservableList<Osoba> getOsoby() {
        return osoby;
    }
}
```

Metody

Opět do třídy vložme zatím jen ty nejdůležitější metody.

pridej a odeber

Metody pro přidání a odebrání osoby jsou naprosto triviální:

```
public void pridej(Osoba osoba)
{
        osoby.add(osoba);
}
```

```
public void odeber(Osoba osoba)
{
        osoby.remove(osoba);
}
```

Základ logické vrstvy máme hotový. Příště si ukážeme, jak propojit logiku s formulářem a celou aplikaci zprovozníme.

6. díl - Upomínač narozenin v JavaFX - Propojení vrstev

V minulém dílu seriálu tutoriálů o tvorbě formulářových aplikací v JavaFX jsme dokončili základ logické vrstvy aplikace. Dnes ji propojíme s formulářem a aplikaci tak zprovozníme.

Propojení prezentační a logické vrstvy

Nyní máme dokončenou tzv. prezentační část aplikace (formulář) a logickou část (třídy). Tyto 2 vrstvy se v aplikaci striktně oddělují, jelikož jinak je kód velmi nepřehledný. **Nikdy byste neměli provádět výpočty, zápisy do souborů, databáze a podobné věci přímo v kontroleru formuláře!** Vždy si vytvoříme třídu, která poskytuje příslušné metody a tuto třídu z kontroleru pouze používáme. Logika zůstane ve třídě. Třída by naopak vůbec neměla vědět o formuláři. Neměla by tedy např. zobrazovat chybové hlášky, ale pouze vyhazovat v případě chyby výjimky. Je potom na formuláři, aby uživateli chybu zobrazil. Právě formulář je ta část aplikace, která s uživatelem komunikuje. Žádná jiná to nedělá.

Pokud vás napadlo, že naše jednoduchá kalkulačka, kterou jsme vytvořili v prvních dílech seriálu, byla návrhově špatně, máte pravdu. Z důvodu jednoduchosti jsme napsali výpočty rovnou do obslužné metody tlačítka. Správně bychom měli mít nějakou třídu, která by výsledky počítala a tu bychom z formuláře pouze volali.

Ukážeme si tedy, jak se to dělá správně.

Propojení prezentace a logiky

Přejdeme do kontroleru formuláře a třídě přidáme privátní atribut typu SpravceOsob, ve kterém rovnou vytvoříme instanci správce:

```
private SpravceOsob spravceOsob = new SpravceOsob();
```

Instance správce se vytvoří při vytvoření formuláře a formulář s ní dále bude komunikovat a tak provádět úkony, které si přeje uživatel.

V metodě initialize() nastavíme dnesLabel na aktuální datum a ListView nastavíme položky na ObservableList ze správce osob. Odteď bude ListView zobrazovat obsah ObservableListu a

pokud se do listu něco přidá, projeví se to i v ListView. Pokud jsou v listu nějaké osoby, nastavíme vybranou položku ListView na první index.

```
@Override
public void initialize(URL url, ResourceBundle rb) {
        dnesLabel.setText(Datum.zformatuj(Calendar.getInstance()));
        osobyListView.setItems(spravceOsob.getOsoby());
        if (!spravceOsob.getOsoby().isEmpty())
                osobyListView.getSelectionModel().select(0);
}
```

Přidávání a mazání osob

Abychom něco konečně také viděli, přejdeme k přidávání osob. Nejprve přejdeme do kódu dialogu OsobaDialog, kde si přidáme privátní atribut typu osoba, ke kterému vygenerujeme getter. Až budeme dialog později zobrazovat, právě odtud si načteme osobu, jejíž údaje uživatel zadal.

```
private Osoba osoba = null;

public Osoba getOsoba() {
        return osoba;
}
```

Nyní přejdeme do připravené obslužné metody tlačítka OK, ta se nachází v metodě vytvorScenu().

Abychom mohli ve vnitřní třídě přistupovat k proměnným jmenoTextField a datumTextField, musíme je nejprve označit jako final, případně bychom je také mohli umístit mezi atributy třídy OsobaDialog. Proveďme tuto úpravu:

```
final TextField jmenoTextField = new TextField();
final TextField datumTextField = new TextField();
```

Vraťme se do obslužné metody tlačítka OK. Zde se pokusíme vytvořit osobu na základě údajů, které uživatel zadal. Osobu uložíme do privátního atributu. Pokud se něco nepovede, vypíšeme chybu. Jistě jste se již setkali s konstrukcí try-catch, která umožňuje odchytit vyvolané výjimky a nějak na tyto chyby reagovat místo toho, abychom nechali aplikaci spadnout.

```
tlacitko.setOnAction(new EventHandler<ActionEvent>() {
        @Override public void handle(ActionEvent e) {
                try {
                        Calendar narozeniny =
Datum.naparsuj(datumTextField.getText());
                        osoba = new Osoba(jmenoTextField.getText(),
narozeniny);
                        hide();
                } catch (ParseException | IllegalArgumentException ex) {
```

```
                        System.out.println("Chyba: " + ex.getMessage());
                }
        }
});
```

Právě řádek:

```
osoba = new Osoba(jmenoJTextField.getText(), narozeniny);
```

Může vyvolat výjimku, jelikož výjimku přímo vyhazujeme v konstruktoru osoby v případě špatných údajů, můžete se tam podívat. Další výjimku může vyvolat i parsování datumu, protože do TextFieldu může uživatel napsat prakticky cokoli a komponenta k výběru data není v JavaFX zatím obsažená. Kód, který výjimku vyvolává, umístíme do bloku try. V bloku catch poté výjimku odchytíme a zobrazíme její zprávu uživateli. Pokud vše proběhne hladce, program se do bloku catch ani nedostane. Výjimky většinou vyvoláváme v logických třídách a potom je chytáme ve formulářích. Výjimky jsou podrobněji popsané v článku Výjimky v Javě. Pomocí hide() zavíráme aktuální formulář.

Jistě jste si všimli, že chybu vypusujeme pouze do konzole. JavaFX zatím neobsahuje podporu chybových hlášek. Nic nám ovšem nebrání hlášku zobrazit jako další formulář. Vy byste již dokázali velmi jednoduše vytvořit formulář s jedním labelem a tlačítkem, který by chybu zobrazil. V tutoriálu jsem toto zanedbal, můžete si projekt později vylepšit.

V tomto formuláři jsme skončili. Přesuneme se zpět do kontroleru a doplníme kód do obslužných metod tlačítek Přidat a Odebrat.

```
@FXML
public void handlePridatButtonAction(ActionEvent event) {
        OsobaDialog dialog = new
OsobaDialog(dnesLabel.getScene().getWindow());
        dialog.showAndWait();

        Osoba nova = dialog.getOsoba();
        if (nova != null)
        {
                spravceOsob.pridej(nova);
        }
}
```

V obslužné metodě tlačítka pridatButton již vytváříme novou instanci dialogu OsobaDialog. Osobu z dialogu si nyní i načteme a pokud tam nějaká je (uživatel dialog potvrdil tlačítkem OK a neodkřížkoval ho), přidáme osobu do správce. Protože používáme ObservableList, zobrazí se osoba ihned i v ListView na formuláři.

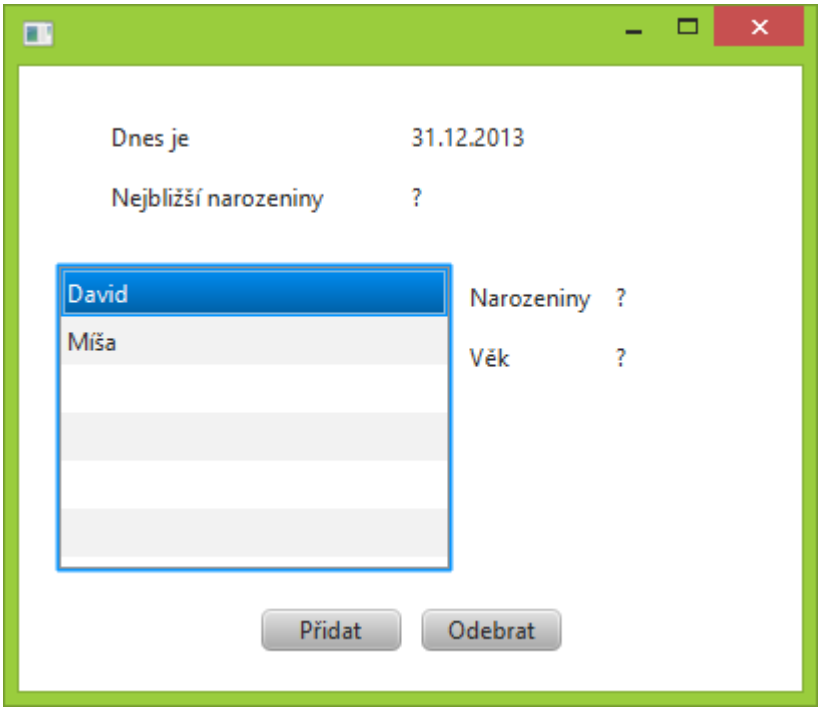
Obslužná metoda tlačítka odebratButton bude vypadat takto:

```
@FXML
```

```
public void handleOdebratButtonAction(ActionEvent event) {
        Osoba vybrana =
(Osoba)osobyListView.getSelectionModel().getSelectedItem();
        if (vybrana != null)
        {
                spravceOsob.odeber(vybrana);
        }
}
```

Důležitá je podmínka, která zjišťuje, zda je v ListView vybraná nějaká položka. Jak vidíte, k vybrané položce se dostaneme přes vlastnost getSelectedItem na SelectionModelu. Položku následně přetypujeme na Osobu, jelikož je typu object (to aby byl ListView univerzální). Tuto osobu předáme metodě odeber na správci, která dále vykoná fyzické odebrání z ObservableListu.

Aplikaci si nyní můžete vyzkoušet, již půjde přidávat a odebírat osoby. Přidané osoby se ihned objeví v ListView díky tomu, že používáme ObservableList. ListView vždy zobrazuje to, co vrací metoda toString() objektu. U osob tedy zobrazuje jejich jméno.



Příště doplníme logickou vrstvu aplikace o další metody a tím ji dokončíme.

JavaFX 2 quickstart: layouty a uzly

JavaFX je knihovna, která umožňuje vývoj grafických aplikací v Javě.

Zdrojové kódy budu z důvodu úspory místa uvádět bez importů. Všechny zde uvedené kódy si můžete stáhnout v přiloženém zipu.

Netbeans

Vytvoření JavaFX projektu: File → New Project → JavaFX → JavaFX Application → název projektu plus si můžete zvolit, zda zda chcete předvytvořit JavaFX třídu (Create Application Class).

V případě, že při zakládání nového projektu nemáte na výběr možnost JavaFX, zkuste si doinstalovat plugin: Tools → Plugins → JavaFX 2 Support, popřípadě stáhnout nejnovější plnou verzi Netbeans IDE.

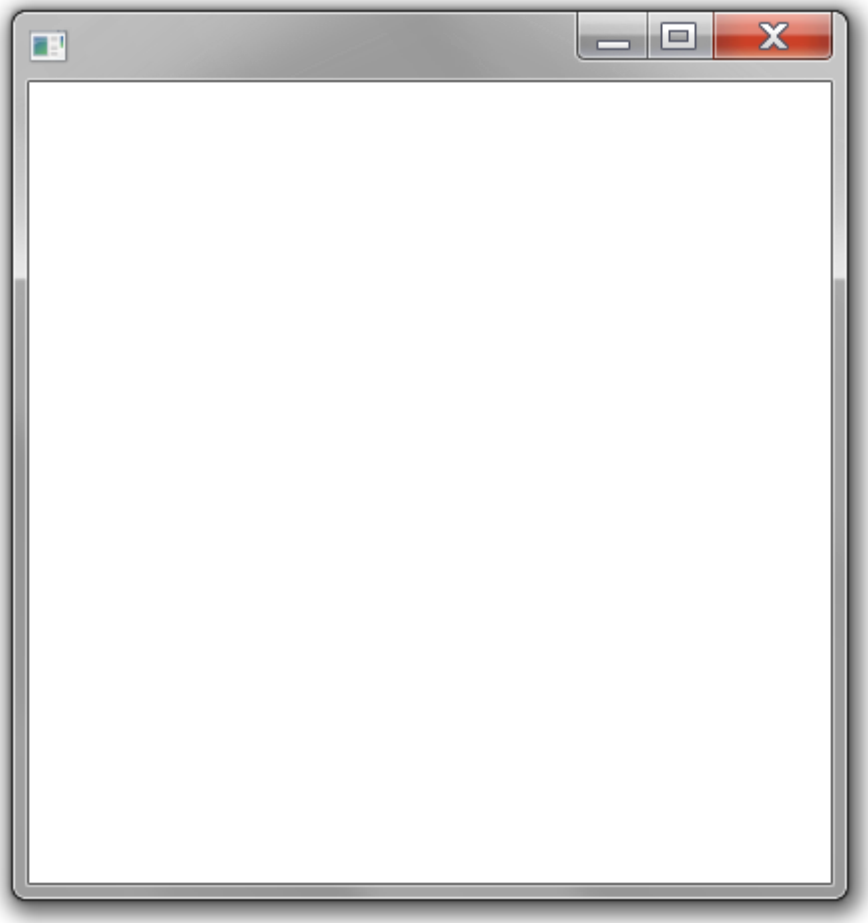
Eclipse

Pro Eclipse je třeba doinstalovat plugin e(fx)clipse: Help → Install New Software → Add → Name: e(fx)clipse | Location: http://download.eclipse.org/…nightly/site → OK → rozevřít nabídku e(fx)clipse a zašktnout volbu dle verze vašeho Eclipse → Next → Next → musíte souhlasit s licenčním ujednáním → Finish (postup instalace s printscreen naleznete na: http://www.eclipse.org/…install.html)

Vytvoření JavaFX projektu: File → New Project → JavaFX → JavaFX Project → název projektu.

Struktura JavaFX aplikace

```
public class MyBasicGUI extends Application {

        @Override
        public void start(Stage stage) {
                try {
                        BorderPane root = new BorderPane();

                        // Zde přijde náš kód.

                        Scene scene = new Scene(root, 400, 400);
                        stage.setScene(scene);
                        stage.show();
                } catch(Exception e) {
                        e.printStackTrace();
                }
        }

        public static void main(String[] args) {
                launch(args);
        }
}
```



Hlavní třída JavaFX aplikace dědí (extends) z javafx.application.Application class. Metoda start() je hlavním vstupním bodem JavaFX aplikace. Pokud vyvíjíte v Netbeans, tak metoda main(String[] args) není vyžadována, ale například v Eclipse bez ní projekt nespustíte.

JavaFX Stage je kontejner nejvyšší úrovně, který dostáváme předpřipravený jako parametr metody start().

JavaFX Scene je kontejner pro veškerý obsah scene grafu.

JavaFX Scene Graph je stromová struktura naší grafické aplikace. Graf se skládá z uzlů (Node) a každý uzel je buď typu kořen (root), větev (branch node), nebo list (leaf node). Root node (kořen) je první uzel. Tento uzel nemá žádného rodiče (parent). Branch node (větev) může mít děti (může mít pod sebou další uzly) a má rodiče (parent). Leaf node (list) je konečný uzel, který nemůže obsahovat další uzly. Jako root uzel (node) se většinou používá nějaký layout pane (plocha, panel pro prostorové rozmístění uzlů), do kterého se pak umisťují další uzly (tlačítka, štítky, další layouty, …) a vytváří se tak strom grafické aplikace.

Vytvoření aplikace

Vytvoříme třídu, která dědí (extends) z javafx.application.Application třídy. Implementujeme metodu start(Stage stage), která dostává jako parametr objekt typu Stage (jeviště). Tento objekt dostáváme již hotový. Dále si vytvoříme kořenový uzel (root node). Většinou použijeme nějaké předdefinované rozvržení (BorderPane, GridPane, Hbox, Vbox, FlowPane, …). Následně

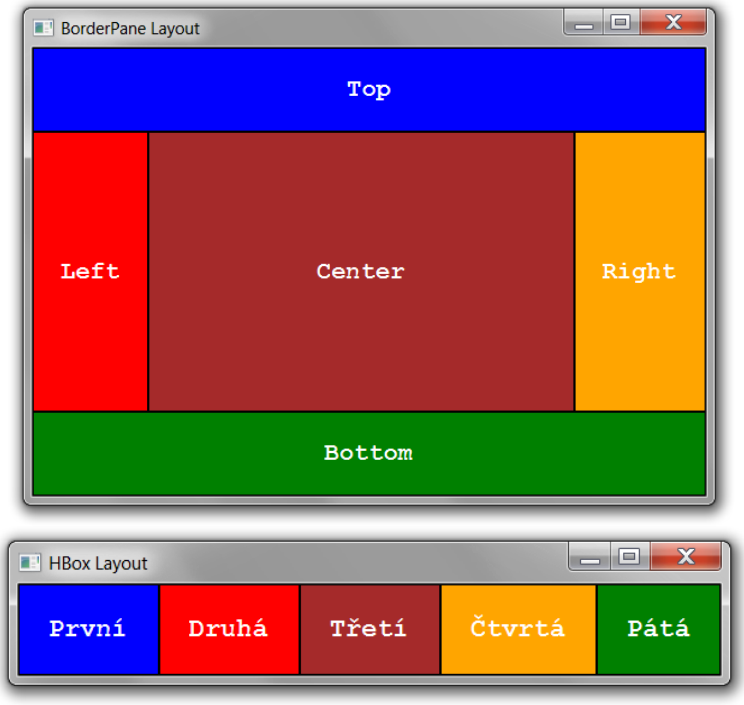
do rozvržení umisťujeme grafické komponenty - uzly (tlačítko, štítek, tabulku, posuvník, textové pole, …). Pak vytvoříme objekt typu Scene (scéna), vložíme do něj kořenový uzel a nastavíme velikost. Scene je kontejner pro veškerý námi vytvořený grafický obsah. Scene vložíme do Stage, nastavíme titulek a zavoláme metodu show(). Ještě nezapomeňme na metodu main(String[] args).

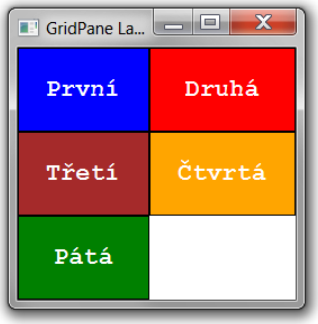
Layouty

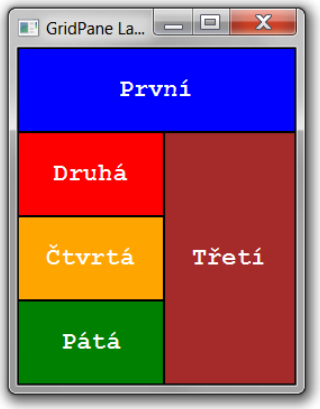
Při vytváření grafické aplikace potřebujeme umístit jednotlivé komponenty (uzly) na určitá místa, abychom dostali požadovaný vzhled. K tomu můžeme použít předpřipravené layouty (kontejnery, které nám umožňují pozicovat jednotlivé uzly grafického uživatelského rozhraní). Layout je uzel typu větev (branch node). To znamená, že v sobě může obsahovat další uzly a těmto uzlům je navíc možné určit, kam se v rámci layout kontejneru mají umístit.

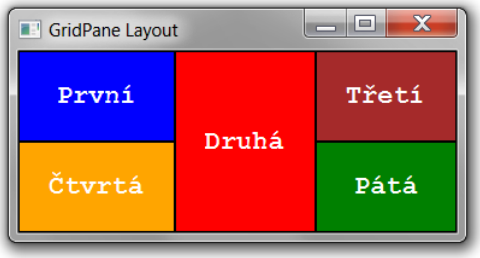
Příklady některých layoutů:

- BorderPane – poskytuje pět oblastí pro umístění uzlů (nahoru, doleva, doprostřed, doprava a dolů)
- HBox – uzly umisťuje jeden vedle druhého
- Vbox – uzly umisťuje pod sebe
- GridPane – vytváří mřížku do jejichž buněk je možné uzly umisťovat









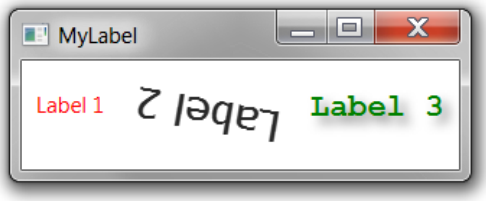
Label (štítek)

```
public class MyLabel extends Application {

        @Override
        public void start(Stage stage) {
```

```
                try {
                        HBox hBoxPane = new HBox();
// vytvoření HBox rozvržení
                        hBoxPane.setSpacing(20);
// nastaví velikost mezery mezi každým potomkem (child) v HBox
                        hBoxPane.setPadding(new Insets(20, 10, 20, 10));
// horní, pravý, dolní a levý prostor okolo obsahu HBox

                        Label label1 = new Label();                    //
vytvoření labelu bez textu
                        label1.setText("Label 1");                     //
nastavení textu
                        label1.setTextFill(Color.RED);                 //
nastaví barvu textu
                        hBoxPane.getChildren().add(label1);            //
vložení label1 do HBox rozvržení

                        Label label2 = new Label("Label 2");           //
vytvoření labelu s textem
                        label2.setFont(new Font("Verdana", 28));       //
nastavení typu písma a velikosti písma
                        label2.setRotate(190);                         //
nastaví úhel otočení
                        hBoxPane.getChildren().add(label2);            //
vložení label2 do HBox rozvržení

                        DropShadow ds = new DropShadow();       // vytvoření
efektu DropShadow
                        ds.setOffsetX(5);                       // posun ve
směru x v pixelech
                        ds.setOffsetY(5);                       // posun ve
směru y v pixelech
                        ds.setColor(Color.GRAY);                // nastavení
barvy efektu

                        Label label3 = new Label("Label 3");
                        label3.setFont(Font.font("Courier New",
FontWeight.BOLD, 22));  // jiný způsob nastavení písma
                        label3.setTextFill(Color.GREEN);
// nastavení barvy textu
                        label3.setEffect(ds);
// nastavení efektu
                        hBoxPane.getChildren().add(label3);
// vložení label3 do HBox rozvržení

                        Scene scene = new Scene(hBoxPane);      // vložení
HBox rozvržení do Scene (HBox je root node scene grafu)
                        stage.setScene(scene);                  // přidání
scene do stage
```

```
                        stage.setTitle("MyLabel");              // nastavení
titulku
                        stage.show();                           // zobrazení
okna

                } catch(Exception e) {
                        e.printStackTrace();
                }
        }

        public static void main(String[] args) {
                launch(args);
        }
}
```

Výsledek:

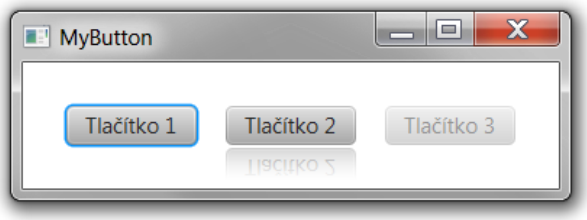


Button (tlačítko)

```
public class MyButton extends Application {

        @Override
        public void start(Stage stage) {
                try {
                        HBox hBoxPane = new HBox();
                        hBoxPane.setSpacing(20);
                        hBoxPane.setPadding(new Insets(30));    // nastavení
prostoru okolo obsahu HBox rozvržení

                        Button button1 = new Button();          // vytvoření
nového tlačítka bez popisku
                        button1.setText("Tlačítko 1");                  //
nastavení popisku (textu)
                        hBoxPane.getChildren().add(button1);    // vložení
tlačítka do HBox rozvržení

                        Reflection reflection = new Reflection();       //
vytvoření efektu odrazu
                        reflection.setFraction(0.8);                    //
nastavení viditelné části odrazu
```

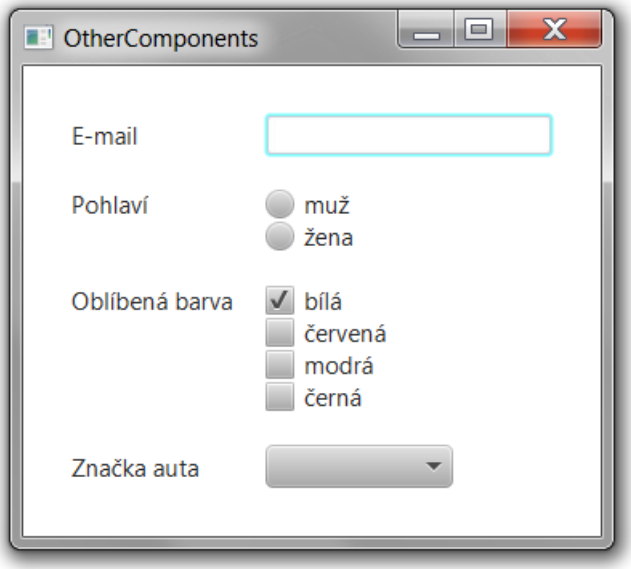
```
                        Button button2 = new Button("Tlačítko 2");     //
vytvoření tlačítka s popiskem
                        button2.setCursor(Cursor.CLOSED_HAND);           //
nastavení typu kurzoru
                        button2.setEffect(reflection);                   //
přiřazení efektru tlačítku
                        hBoxPane.getChildren().add(button2);

                        Button button3 = new Button("Tlačítko 3");
                        button3.setOpacity(0.5);                         //
nastavení neprůhlednosti tlačítka
                        hBoxPane.getChildren().add(button3);

                        Scene scene = new Scene(hBoxPane);
                        stage.setScene(scene);
                        stage.setTitle("MyButton");
                        stage.show();

                } catch(Exception e) {
                        e.printStackTrace();
                }
        }

        public static void main(String[] args) {
                launch(args);
        }
}
```

Výsledek:



Další ...

JavaFX nabízí mnoho ovládacích prvků uživatelského rozhraní. Je mimo rozsah tohoto krátkého tutoriálu všechny je zmiňovat, navíc jejich použití je podobné tomu, co jsme si již ukázali. V přiložením zipu naleznete bonusový příklad, který používá vstupní textové pole (TextField), přepínač (RadioButton), zaškrtávací políčka (CheckBox) a výběr (ChoiceBox).



Ve druhé části tohoto tutoriálu si probereme události (event) a stylování pomocí CSS.

Zdroje:

- http://docs.oracle.com/…i/index.html
- http://docs.oracle.com/… started.htm
- http://docs.oracle.com/…controls.htm
- http://docs.oracle.com/…b-layout.htm
- http://docs.oracle.com/…enegraph.htm
- http://www.eclipse.org/…install.html

JavaFX 2 quickstart: Události a CSS

Zdrojové kódy budu, tak jako v předchozím díle, z důvodu úspory místa uvádět bez importů. Všechny zde uvedené kódy si můžete stáhnout v přiloženém zipu.

Event (událost)

Událost vzniká určitou akcí (činností) jako například pohybem myši, stiskem tlačítka či klávesy (KeyEvent – stiskem klávesy na klávesnici, MouseEvent – pohybem myši nebo stiskem tlačítka na myši,...). Událost (event) je objekt, který má v sobě všechny informace, týkající se dané události (jaký je to typ události, kde událost vznikla, kdo ji vyvolal, …).

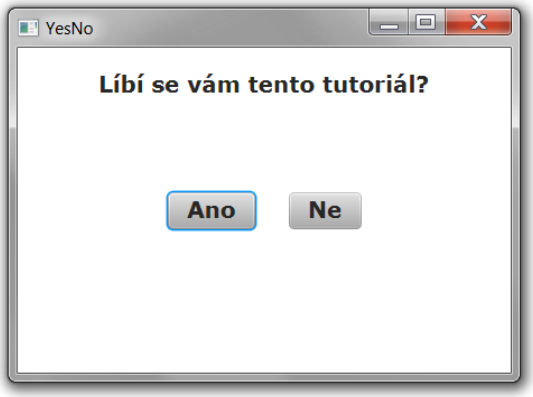
Uzel (node) si může zaregistrovat ovladač (handler), který bude zavolán v případě, že vznikne určitá událost. Handler pak v metodě handle() definuje, co se má stát.

```
public class YesNo extends Application {
        private Button yesBT, noBT;
        private final int PANE_WIDTH = 400;
```

```
        private final int PANE_HEIGHT = 200;

        @Override
        public void start(Stage stage) {
                try {
                        BorderPane rootPane = new BorderPane();         //
vytvoření BorderPane rozvržení, tento uzel bude root node scene grafu
                        rootPane.setPadding(new Insets(20));
                        Font font = Font.font("Verdana", FontWeight.BOLD, 20);

                        VBox topPane = new VBox();
// vytvoření VBox rozvržení (do něj vložíme Label)
                        topPane.setAlignment(Pos.CENTER);
// nastavení zarovnání obsahu VBox na střed
                        Label questionLB = new Label("Líbí se vám tento
tutoriál?");        // vytvoření nového štítku
                        questionLB.setFont(font);
// nastavení písma textu na štítku
                        topPane.getChildren().add(questionLB);

                        VBox bottomPane = new VBox();
                        bottomPane.setAlignment(Pos.CENTER);
                        final Label responseLB = new Label();
                        responseLB.setFont(font);
                        bottomPane.getChildren().add(responseLB);

                        rootPane.setTop(topPane);               // do horní
části BorderPane rozvržení vloží VBox rozvržení se štítkem
                        rootPane.setBottom(bottomPane);         // totéž pro
dolní část BorderPane rozvržení


                        yesBT = new Button("Ano");              // vytvoří
nové tlačítko s popiskem
                        yesBT.setFont(font);                    // nastaví
písmo pro popisek tlačítka
                        noBT = new Button("Ne");
                        noBT.setFont(font);

                        GridPane centerPane = new GridPane();
// vytvoří GridPane rozvržení
                        centerPane.setPrefSize(PANE_WIDTH, PANE_HEIGHT);
// nastaví požadovanou velikost pro rozvržení GridPane
                        centerPane.add(yesBT, 0, 0);
// umístí tlačítko yesBT na pozici sloupec (x) = 0, řada (y) = 0 v rozvržení
GridPane
                        centerPane.add(noBT, 1, 0);
// umístí tlačítko noBT na pozici x = 1, y = 0
```

```
                        centerPane.setHgap(30);
// nastaví vodorovnou mezeru mezi komponentami
                        centerPane.setAlignment(Pos.CENTER);
// nastaví zarovnání obsahu GridPane rozvržení na střed
                        rootPane.setCenter(centerPane);
// vloží GridPane rozvržení do centrální části BorderPane rozvržení


                        // tlačítko yesBT si zaregistruje ovladač (handler),
který bude reagovat na událost kliknutí na tlačítko
                        yesBT.setOnAction(new EventHandler<ActionEvent>() {

                                @Override
                                // metoda, ve které je určeno, co se v případě
zavolání handleru má provést
                                public void handle(ActionEvent event) {
                                        responseLB.setText("Díky, pochvala
potěší.");               // nastaví text štítku
                                }

                        });

                        // tlačítko si registruje ovladač, který bude reagovat
na událost, kdy kurzor najede na tlačítko
                        yesBT.setOnMouseEntered(new EventHandler<MouseEvent>()
{

                                @Override
                                public void handle(MouseEvent event) {
                                        yesBT.setScaleX(1.5);           //
nastavuje měřítko, dle kterého se bude měnit velikost komponenty podél osy x
                                        yesBT.setScaleY(1.5);           //
totéž pro osu y
                                }

                        });

                        // tlačítko si registruje ovladač, který bude reagovat
na událost, kdy kurzor opustí tlačítko
                        yesBT.setOnMouseExited(new EventHandler<MouseEvent>()
{

                                @Override
                                public void handle(MouseEvent event) {
                                        yesBT.setScaleX(1);
// velikost dle osy x se vrátí k původnímu rozměru
                                        yesBT.setScaleY(1);
// totéž pro osu y
                                }
```

```
                        });

                        noBT.setOnMouseEntered(new EventHandler<MouseEvent>()
{

                                @Override
                                public void handle(MouseEvent event) {
                                        Random gen = new Random();
                                        int x = gen.nextInt((int)(PANE_WIDTH -
noBT.getWidth()));                 // generuje náhodné číslo pro x (v rozsahu
šířky GridPane)
                                        int y = gen.nextInt((int)(PANE_HEIGHT
- noBT.getHeight()));               // totéž pro y (v rozsahu výšky GridPane)
                                        noBT.setLayoutX(x);
// nové umístění tlačítka (pozice na ose x)
                                        noBT.setLayoutY(y);
// totéž pro osu y
                                }
                        });

                        noBT.setOnAction(new EventHandler<ActionEvent>() {

                                @Override
                                public void handle(ActionEvent event) {
                                        noBT.setVisible(false);
                                }

                        });

                        Scene scene = new Scene(rootPane);
                        stage.setScene(scene);
                        stage.setTitle("YesNo");
                        stage.show();
                } catch(Exception e) {
                        e.printStackTrace();
                }
        }

        public static void main(String[] args) {
                launch(args);
        }
}
```



Kaskádové styly (CSS)

Kód, který slouží jen pro stylování vzhledu aplikace a pro její funkčnost není nezbytný, lze vypustit a celé stylování lze provést pomocí kaskádových stylů. Pokud máte nějaké zkušenosti s tvorbou HTML stránek, určitě vám to zní povědomě.

Styl aplikace lze nadefinovat pomocí kaskádových stylů v samostatném souboru a tento soubor umístit do adresáře, kde se nachází náš .java soubor. V .java souboru pak stačí říci, v jakém souboru se styly nacházejí.

Uzly jako například Label, Button, … již automaticky patří do tříd (.label, .button). Všem uzlům je možné přidat třídu i id a ty následně pomocí CSS stylovat. Stylování je hierarchické. To znamená, že styl rodičovského uzlu se vztahuje i na jeho potomky a potomky potomků. V potomkovi se styl dá předefinovat a má vyšší prioritu (přebije styl rodiče).

Jako příklad si vezmeme předchozí aplikaci, jejíž vzhled ale nyní budeme definovat pomocí CSS.

.java soubor

```
public class YesNoCSS extends Application {
        private Button yesBT, noBT;
        private final int PANE_WIDTH = 400;
        private final int PANE_HEIGHT = 200;

        @Override
        public void start(Stage stage) {
                try {
                        BorderPane rootPane = new BorderPane();
                        rootPane.setId("rootPane");
// nastaví id tomuto uzlu

                        VBox topPane = new VBox();
                        topPane.getStyleClass().add("vBox");
// nastaví třídu (class) tomuto uzlu
                        Label questionLB = new Label("Líbí se vám tento
tutoriál?");
```

```
                        topPane.getChildren().add(questionLB);

                        VBox bottomPane = new VBox();
                        bottomPane.getStyleClass().add("vBox");
                        final Label responseLB = new Label();
                        bottomPane.getChildren().add(responseLB);

                        rootPane.setTop(topPane);
                        rootPane.setBottom(bottomPane);

                        yesBT = new Button("Ano");
                        noBT = new Button("Ne");

                        GridPane centerPane = new GridPane();
                        centerPane.setId("centerPane");
                        centerPane.setPrefSize(PANE_WIDTH, PANE_HEIGHT);
                        centerPane.add(yesBT, 0, 0);
                        centerPane.add(noBT, 1, 0);
                        rootPane.setCenter(centerPane);


                        /*
                         * Kód metod zůstává stejný.
                         */

                        Scene scene = new Scene(rootPane);

scene.getStylesheets().add(getClass().getResource("styles.css").toExternalForm
());      // nastaví scene styl
                        stage.setScene(scene);
                        stage.setTitle("YesNoCSS");
                        stage.show();
                } catch(Exception e) {
                        e.printStackTrace();
                }
        }

        public static void main(String[] args) {
                launch(args);
        }
}
```

.css soubor

```
/* námi definované id rootPane */
#rootPane {
        -fx-font-size: 20pt;
        -fx-font-family: "Verdana";
        -fx-font-weight: bold;
```

```
        -fx-padding: 20;
}

/* námi definované id centerPane */
#centerPane {
        -fx-hgap: 30;
        -fx-alignment: center;
}

/* class (třída), kterou mají všechny štítky */
 .label {
        -fx-text-fill: black;
        -fx-alignment: center;
        -fx-text-alignment: center;
}

/* námi definovaná třída */
.vBox {
        -fx-alignment: center;
}
```

Vzhledově se nic nezměnilo a náš kód je nyní lépe čitelný. CSS umožňují použití i tzv. pseudo-tříd (např. hover, která se aplikuje, pokud je nad daným uzlem kurzor myši). Pomocí pseudo-třídy hover, můžeme nahradit i kód metod yesBT.setOnMouseEntered() a yesBT.setOnMouseExited(). Příklad naleznete v přiloženém zipu v balíčku applicationCSS02.

A to je vše. Doufám, že se vám tento krátký tutoriál líbil, a že jste se něco nového naučili. Pokud vás JavaFX zaujala, doporučuji se podívat na zdroje uvedené ke konci každého dílu tohoto tutoriálu.

Zdroje:

- http://docs.oracle.com/…tutorial.htm
- http://docs.oracle.com/…/cssref.html

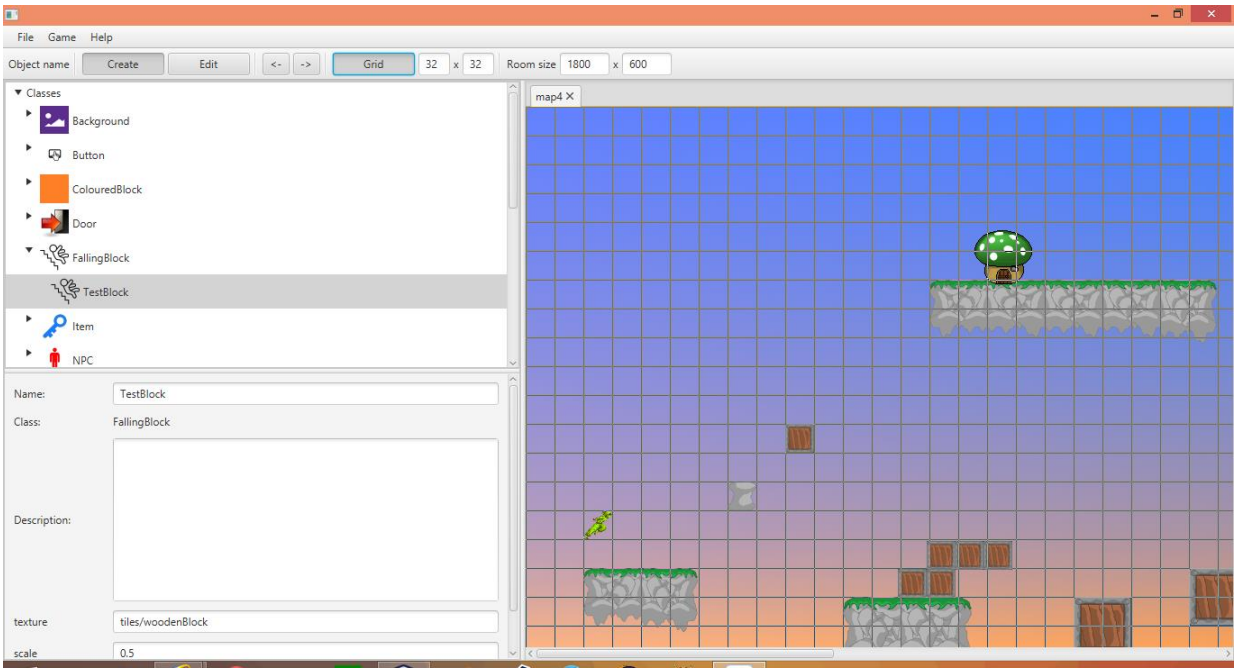
GeasyEngine pro Javu

http://www.itnetwork.cz/ - screenshots

GeasyEngine je velmi mocný nástroj na tvorbu her v Java FX, který je pro Windows, Mac a Linux. GeasyEngine je rychlý a jednoduchý k použití. Dá se v něm udělat spoustu 2d platformovek (2d rpg v pozdější verzi).

GeasyEngine je vyroben v JavěFX a používá pár novinek v Javě 8, proto je za potřebí mít nainstalovanou **Javu 8** nebo výše! Je postaven na BunnyHuntru který jsem dělal do soutěže "Machr na javu", jenom se k tomu přidalo editor, zpřehlednil kód a odstranili deprecated funkce, kterých jsem si nevšiml když jsem psal BunnyHuntra.

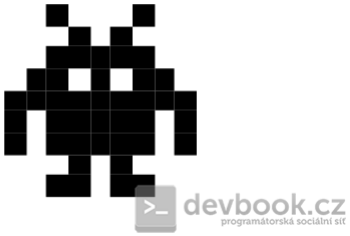
S GeasyEnginem můžete vytvořit svůj vlastní libovolný svět, nemusíte být programátoři aby jste vyrobili bezva akční skákačků! Engine obsahuje předpřipravené základní objekty, ze kterých stačí hru poskládat. Samozřejmě můžete jednoduše přidávat další třídy a využívat mechanismy enginu při programování herní logiky.



Hry vytvořené na GeasyEnginu

- Stario - http://monarezio.net/Games/Stario nebo http://www.itnetwork.cz/hra-zwbm
- Ninja - http://monarezio.net/Games/Ninja

Screenshoty



Program byl vytvořen roku: 2014

Zdrojákoviště Java - JavaFX

Počítač řádků II v JavaFX v 1.1

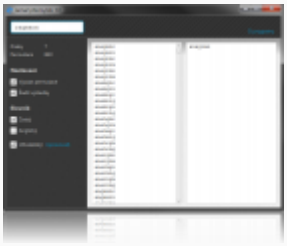
Počítač řádků je jednoduchý program, který mimo jiné spočítá řádky vašeho projektu. Výsledky přehledně zobrazí v tabulce. Včetně zdrojových kódů v Javě.

106x Napsal Hartrik | Vydáno: 2014

RSS čtečka v JavaFX v 1.0

Jednoduchá RSS čtečka na platformě JavaFX. Umožňuje otevřít internetovou stránku přímo v programu. Včetně zdrojových kódů.

200x Napsal Hartrik | Vydáno: 2013



Lamač přesmyček v 3.1

Program za vás vyřeší přesmyčky. Obsahuje rozsáhlý český a anglický slovník. Včetně zdrojových kódů v Javě 8.

2 986x Napsal Hartrik | Vydáno: 2015

Bunny Hunter v JavaFX v 1.0

Bunny Hunter je velikonoční 2D hra v JavaFX na jednoduchém enginu se zdrojovým kódem. S králíkem musíte musí chytat vajíčka aby se nerozbila.

246x Napsal Monarezio | Vydáno: 2014

ASCII Artist II v 1.0

Aplikace vytvoří ASCII art z libovolného obrázku. Výstup je možné uložit do obrázku nebo HTML. K dispozici je celá řada nastavení a zdrojáky v Javě FX.

201x Napsal Hartrik | Vydáno: 2014

JavaFX - přehrátí souboru *.mp3 v 1.0

Ukázka jak přehrát soubor mp3 v JavaFX

286x Nehodnoceno Napsal Fugiczek | Vydáno: 2012