1 Platforma JavaFX

Tato kapitola se věnuje platformě JavaFX, jejímu vzniku a vývoji. Definuje pojem JavaFX a následně popisuje, kde a kdy platforma vznikla, jaké verze byly již zveřejněny a jejich přínosy a nejzásadnější novinky. Definice pojmu JavaFX

JavaFX je soubor grafických a mediálních balíčků umožňujících vývojářům navrhovat, vytvářet, debugovat a spouštět RIA aplikace schopné jednotně fungovat na široké škále různých platforem. [4][5] RIA aplikace je typ aplikace, která má vlastnosti standardní desktopové aplikace a která se spouští ve webovém prohlížeči pomocí pluginů a JVM[1]. JVM je schopné přeložit bytekód vznikající při kompilaci do binárního kódu a následně předat procesoru, který tento kód začne provádět.

Hlavním záměrem vývoje této platformy je umožnit vývojářům snadnější vytváření grafických aplikací pomocí efektivnějšího oddělení grafické a logické části aplikace, efektivnější komunikací platformy s hardwarem zařízení, na kterém je aplikace spuštěna, a rozšíření palety dostupných ovládacích prvků společně s podporou 3D a videa. To vše postavené na platformě Java s možností „write once, run anywhere“. Aplikace je tedy možné spouštět kdekoliv, kde je k dispozici JVM.

Pojem JavaFX je často spojován s termíny JavaFX Script a FXML. JavaFX Script, je skriptovací jazyk vytvořený přímo pro platformu JavaFX. Tento jazyk je podobný jazyku JavaScript a byl používán pro zápis JavaFX aplikací až do roku 2010, kdy byl firmou Oracle oznámen konec dalšího vývoje jazyka JavaFX Script. FXML označuje značkovací jazyk na bázi XML, kterým lze definovat uživatelská rozhraní. Jazyku FXML se věnuje kapitola 2.2.

1.1 Historie platformy JavaFX

Myšlenka na jazyk poskytující podporu kvalitnějších uživatelských rozhraní se zrodila ve firmě SeeBeyond, konkrétně v hlavě Chrise Olivera. Firma potřebovala rozšířit funkčnost

[1] Java Virtual Machine

svých RIA aplikací, a právě pro tento účel vznikl předek JavaFX, jazyk F3[2] V roce 2005 byla firma SeeBeyond převzata firmou Sun, která tímto získala F3. Chris Oliver do firmy Sun přišel také a nadále zde vedl vývoj F3. Práce pokračovaly až do roku 2007, kdy byl F3, nyní již pod změněným názvem na JavaFX, představen na konferenci JavaOne.

JavaFX Script byl na počátku interpretovaný jazyk. Ihned po vydání vývojáři experimentovali s tímto novým jazykem a poskytovali důležitou zpětnou vazbu ve formě pozorovaných nedostatků či námětů na vylepšení. To velice pomohlo při vývoji a ladění následujících verzí platformy. Zároveň se ve firmě Sun horlivě pracovalo na kompilované verzi jazyka, což se na konci roku 2007 podařilo. Výsledkem byla verze jazyka, která stejně jako celá platforma Java je kompilovaná do Bytekódu. V roce 2008 pokračovaly práce na kompilované verzi, což znamenalo postupné přepsání většiny knihoven týkajících se především grafických komponent. Tým pracující na těchto vylepšeních se rozšířil a dokonce zahrnoval i několik vývojářů, kteří pracovali na grafických knihovnách Swing, jehož nástupcem je právě JavaFX. V tomto roce byl také publikován první JavaFX plug-in pro vývojové prostředí NetBeans. Oznámena byla také první verze platformy JavaFX, která vyšla v prosinci 2008. [6]

1.1.1 JavaFX 1

První verze platformy vedla k rychlejšímu osvojování mezi vývojáři hlavně díky tomu, že se platforma stala oficiální, stabilní a byla zajištěna její dlouhodobější podpora. První verze přinesla podporu pro desktopové aplikace a webové prohlížeče. Jako jediný jazyk pro vývoj se používal JavaFX Script. Následující verze platformy přinesla hlavně JavaFX Mobile, což je implementace JavaFX platformy určená pro mobilní zařízení a také vývoj RIA aplikací. JavaFX Mobile spolupracuje s Java ME platformou. Programovacím jazykem je také JavaFX Script. Data vydání jednotlivých verzí platformy a jejich hlavní přínosy jsou vidět v následujícím přehledu převzatém ze zdroje[4].

4. 12. 2008 – JavaFX 1.0 pro webové prohlížeče a desktopové aplikace,

12. 2. 2009 – JavaFX 1.1 pro mobilní vývoj,

2. 6. 2009 – JavaFX 1.2 pro Linux a Solaris a CSS rozhraní,

[2] Form Follows Function

22. 4. 2010 – JavaFX 1.3 se zvýšeným výkonem, podporou dalších platforem a zlepšení ovládacích prvků UI,

10. 10. 2011 – JavaFX 2.0.

1.1.2 JavaFX 2

Oproti předchozí verzi byla verze 2 v mnoha ohledech revoluční. Jelikož krátce po vydání verze 1.3 byla firma Sun odkoupena firmou Oracle, začal se vývoj nové verze platformy ubírat jiným směrem. JavaFX přestala být oddělenou kapitolou a přibližuje se klasickému Java prostředí. Nejzásadnější změnou je podpora nativního Java kódu. Vývojáři tak mohou psát přímo v jazyce Java a nemusí se učit nový jazyk. Zároveň se změnou zápisu kódu oznámil Oracle ukončení podpory JavaFX Script. V tomto případě ukončení podpory znamená, že od této verze počínaje není možné JavaFX Script používat, což je pro Javu neobvyklé, protože vše označené jako zastaralé lze většinou nadále používat. Dále byla také ukončena podpora JavaFX Mobile a oznámen záměr učinit platformu Open-source. Zároveň se navzdory heslu původní Java platformy stává JavaFX platformně závislou, kvůli nově oznámené podpoře videa a multimediálních souborů. Aby mohla JavaFX aplikace tyto soubory používat, je závislá na dostupných systémových kodecích pro přehrávání multimediálních souborů. Velkou novinkou je uvedení nového značkovacího jazyka FXML jako alternativu psaní uživatelského rozhraní. Verze 2 přichází s podporou pouze operačního systému Windows a příslibem podpory ostatních operačních systémů v následující verzi platformy.

Verze 2.1, vydaná 27.4.2012, přichází sice se slibovanou rozšířenou podporou dalších operačních systémů, avšak ne v takovém měřítku, jak se předpokládalo. Verze nově podporuje pouze Mac OS X pro desktopovou verzi. Mezi ostatní novinky se však řadí rozšířená podpora formátů audia a videa (Audio Video Codec, Advanced Audio Coding), drobná vylepšení grafických prvků, konkrétně Combo boxu a Menu panelu.

14.8.2012 je uvedena verze 2.2, která s sebou přináší hlavně podporu operačního systému Linux. Nová verze také obsahuje podporu HTTP Live Streaming, podporu dotykových obrazovek spolu s gesty, a nové grafické prvky, konkrétně Color Picker a Pagination. Dostupná je také nová možnost distribuce aplikace jako „Native Packaging“, která zajišťuje aplikaci nezávislost na nainstalovaném JRE nebo JavaFX SDK v cílovém zařízení, které

bude aplikaci spouštět. Možnosti Native Packaging se bude podrobněji věnovat kapitola Sestavení aplikace. Od této verze byla také oznámena integrace Java FX platformy se základní edicí Java, tedy od verze 7, update 6. [7]

1.1.3 JavaFX 8

Aktuální verze platformy nese označení 8 z důvodu její integrace se základní distribucí platformy Java. Platforma JavaFX tedy převzala číslování verzí platformy Java a oficiálně se stala součástí standardních knihoven. Zároveň se tímto stává oficiálně doporučenou grafickou knihovnou pro aplikace vyvíjené na platformě Java 8. JavaFX 8 obsahuje všechny možnosti předchozích verzí a podporu platforem Window, Linux, Solaris a Mac OS X pro desktop verzi, a přichází s řadou novinek.

Platforma JavaFX 8 umožňuje vývoj aplikací dvěma základními způsoby:

1. Aplikace je možné vytvářet stejně jako klasické Java aplikace s uživatelským rozhraním ve Swing. Uživatelské rozhraní je zapsáno v jedné nebo několika samostatných třídách v jazyce Java.

2. Druhým způsobem vytváření JavaFX aplikací je využití FXML. V tomto případě se aplikace vytvářejí tak, že uživatelské rozhraní zapíšeme v jazyce FXML, přičemž každá obrazovka je zapsána v samostatném FXML dokumentu. Dále aplikace musí obsahovat Controller soubory k FXML dokumentům, tedy Java třídy implementující rozhraní `Initializable`, které definuje jedinou metodu `initialize`. Popis Controller souborů je uveden v kapitole 3.2.2.

Platforma zároveň zajišťuje zabalení každé JavaFX aplikace tak, aby bylo možné ji spustit jako klasickou desktop aplikaci nebo webovou aplikaci. Existuje také možnost zabalení aplikace do balíčku nazývajícího se `self-contained application`. Tento balíček je nutné před spuštěním nainstalovat, jelikož obsahuje vlastní JRE, které využívá pouze zabalená aplikace. Tímto je tedy zajištěna i nezávislost na nainstalovaném JRE v cílovém systému. Detailní popis sestavení a spouštění aplikací je v kapitole 3.6.

Od roku 2011 bylo možné do Swingu zahrnout některé JavaFX prvky. Bohužel ale nebylo možné tuto strategii použít naopak. V platformě JavaFX 8 je nyní možné do vytvářené

JavaFX aplikace vkládat Swing komponenty, a dokonce sjednotit grafické vlákno EDT[3] ze Swingu s JavaFX Application Thread (dále jen FXT). Toto grafické vlákno je určeno výhradně pro práci s grafickými komponentami. Zároveň platí, že ke všem grafickým komponentám musí být přistupováno výhradně z tohoto vlákna. Vkládání Swing komponent do JavaFX aplikací je možné díky nové třídě `SwingNode`, do které se vkládají Swing komponenty, a následně se celá třída `SwingNode` přidá do existující scény JavaFX aplikace.

Dalším vylepšením je nový vzhled platformy nazvaný Modena. Tento vzhled definuje grafickou stránku všech JavaFX komponent a nahrazuje předchozí vzhled s názvem Caspian. Kompletní ukázka všech grafických komponent využívajících nový vzhled je dostupná na adrese zdroje[8]. Dále přibývá podpora Rich textu[4], která umožňuje definovat styly a přidávat efekty k textům, nebo definovat celý styl textu pomocí CSS souborů.

Mezi novinky této verze patří také dvě nové komponenty. Konkrétně se jedná o komponenty `DatePicker` a `TreeTableView`. `DatePicker` slouží pro pohodlný výběr data uživatelem pomocí zobrazení kalendáře při kliknutí na komponentu. Návratová hodnota při vybírání data je ve formátu `LocalDate`, vyjadřující datum bez časové zóny ve formátu ISO-8601. Zobrazení kalendáře je řízeno vlastností `Chronology`, která specifikuje formát kalendářního systému, který bude použit při zobrazení kalendářového vyskakovacího okna. Formátem zobrazení rozumíme uspořádání např. dnů v týdnu – týden může začínat pondělím nebo nedělí. Typ základního zobrazení se bere z operačního systému.

`TreeTableView` je nová komponenta umožňující zobrazování neomezeného množství řádků dat rozdělených do sloupců. Komponenta kombinuje výhody již existujících prvků `TreeView` a `TableView`. Zároveň umožňuje měnit šířku nebo pořadí sloupců za běhu aplikace a specifikovat pravidla, co se má stát, když uživatel změní šířky sloupců.

---

[3] Event Dispatch Thread – vlákno určené pro práci s grafickými komponentami ve Swingu

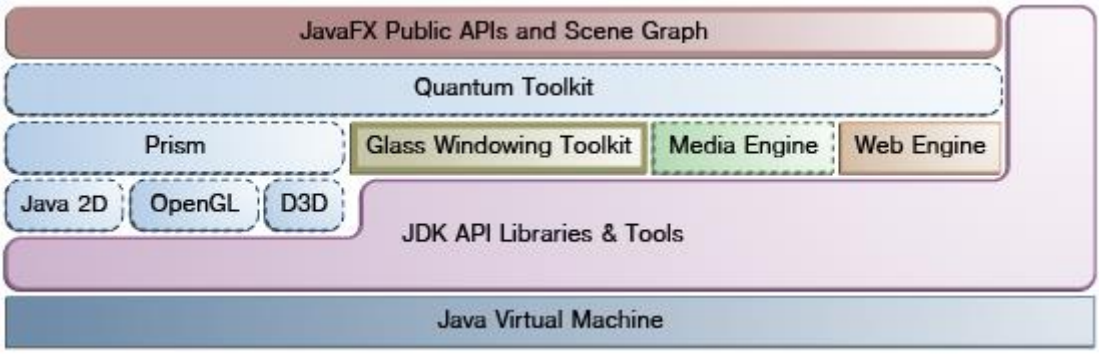
[4] Text podporující grafické efekty jako výplně, zrcadlení, průhlednost, atd

2 Architektura platformy JavaFX 8

Tato kapitola popisuje architekturu platformy, jednotlivé prvky a jejich vlastnosti. Nejprve je popsána architektura jako celek, dále pak je podrobněji popsána knihovna Prism, Glass a Media a Web Engine. Dále je popsána architektura aplikací postavených na JavaFX 8 a nakonec využití jazyka FXML a CSS při tvorbě aplikací.

2.1 Popis architektury platformy JavaFX

Architektura platformy je tvořena několika vrstvami, které jsou názorně vidět na obrázku 2.1.



Obrázek 2.1: Diagram architektury platformy JavaFX, [9]

Celá platforma JavaFX 8 stojí na JVM, tedy na virtuálním stroji Javy, který je zodpovědný za hardwarovou a systémovou nezávislost a na kterém jsou spouštěny všechny aplikace využívající platformu Java. Stejně tak jako následující vrstva JDK API[5] Libraries & Tools, je JVM základem pro vytváření a spouštění Java aplikací.

JDK API Libraries & Tools označuje soubor knihoven a nástrojů dostupných ve standardní edici platformy Java.

Nad vrstvou JDK API Libraries & Tools se nachází grafický systém platformy. Systém je tvořen vrstvou Quantum Toolkit, knihovnou Prism se svými součástmi Java 2D, OpenGL[6]

[5] Application Programming Interface

[6] Open Graphic Library

a D3D[7], knihovnou Glass Window Toolkit, Media Engine a Web Engine. Těmto komponentám se podrobněji věnuje podkapitola 2.1.1.

Vrchní vrstva představuje tu část platformy, která slouží jako rozhraní pro práci vývojáře.. Tato vrstva se skládá z JavaFX Public API a Scene Graph.

Scene Graph, neboli v českém překladu graf scény, je základním prvkem při tvorbě jakékoliv JavaFX 8 aplikace. Scene Graph slouží jako základní kořenový element, do kterého se dále vkládají všechny další grafické komponenty. Tyto grafické komponenty se v anglickém prostředí nazývají Node (v překladu uzel) a jsou to abstraktní třídy, jež zastupují všechny grafické komponenty dostupné v platformě JavaFX 8. Uzly (Node) mají vždy jediného rodiče (Parent) a zároveň mohou mít libovolný počet dětských uzlů (Children). Toto stromové uspořádání elementů a komponent je stejné jako vnořování elementů v jazyce XML. Jednotlivé uzly mají své vlastní identifikátory (ID), definovaný styl a vymezení funkčnosti. [9]

2.1.1 Grafický systém

Grafický systém platformy FavaFX 8 se skládá z komponenty Quantum Toolkit, obsahující Prism, Glass Windowing Toolkit a Media a Web engine. Tyto komponenty jsou vidět ve druhé a třetí vrstvě shora na obrázku 2.1.

Pojmem Prism se rozumí část grafického systému mající na starosti grafické zobrazování vrchnější vrstvy Scene Graph. Prism dokáže zobrazovat nejen jednoduchou 2D grafiku, ale i 3D grafiku, ke které využívá hardwarově nebo softwarově akcelerované renderování. Prism primárně využívá hardwarové renderování. Umí přitom využívat DirectX 9 na Windows XP a Vista, DirectX 11 na Windows 7, OpenGL na strojích Mac, Linux a embedded systems. Pokud v systému není dostupná hardwarová akcelerace renderovacích procesů, využívá Prism softwarovou akceleraci, která je dostupná ve všech JRE. [9]

Quantum Toolkit, vyskytující se ve druhé vrstvě shora, je součástí grafického systému platformy, a je zodpovědný za integraci Prism a Glass Windowing Toolkitu dohromady za účelem přístupnosti pro JavaFX vrstvu nad ní. Dále je Quantum Toolkit zodpovědný za řízení pravidel pro práci s vlákny v aplikaci, především vzhledem k řízení přednosti zpra-

[7] Direct3D

covávání jednotlivých akcí. Jedná se především o zpracovávání událostí a vykreslování grafiky. [9]

V části platformy řešící grafické zobrazování aplikací je Glass Windowing Toolkit (dále jen Glass) tou nejnižší vrstvou, která pracuje s prostředím, ve kterém je aplikace spuštěna. Její hlavní odpovědností je podporovat nativní služby operačního systému, například správu času a aktivních oken. Tato vrstva slouží jako platformově závislá součást platformy JavaFX a propojuje ji s operačním systémem.

Zároveň je Glass odpovědná za správu fronty událostí (anglicky Event queue). Na rozdíl od AWT, které si drželo vlastní frontu, využívá Glass frontu událostí operačního systému. Zároveň běží Glass ve stejném vlákně jako celá JavaFX aplikace. Důvodem jsou zkušenosti s předchozí grafickou knihovnou AWT, kde část systému běžela v odděleném vlákně, což způsobovalo problémy.

Jak již bylo naznačeno, platforma JavaFX má k dispozici vlastní vlákna určená výhradně pro práci s uživatelským rozhraním. Celý systém běží minimálně ve dvou ze tří základních vláken platformy.

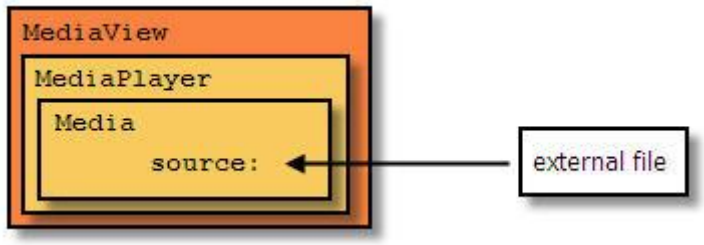
Prvním vláknem je dříve zmíněný `JavaFX application thread` (FXT). Toto vlákno je určeno pro využívání developery, a pro zobrazování „živé“ scény. Živá scéna je scéna, která je v programu využívána, a může nebo nemusí být zároveň zobrazena v aktivním okně. Platí, že k jakékoliv živé scéně, musí být přistupováno z tohoto vlákna. Je samozřejmě umožněno developerům využívat vlákna na pozadí provádějící časově náročné operace, zatímco uživatelské prostředí běží ve svém vlákně a reaguje a běží plynule.

Druhým vláknem je `Prism render thread`, které zpracovává renderování. Umožňuje renderovat obrázek, zatímco se další obrázek načítá a zpracovává. Toto poskytuje obrovskou výhodu především na strojích disponujících více procesory.

Třetím vláknem je `Media Thread`, které je používané v případě, že aplikace obsahuje multimediální obsah. Toto vlákno běží v pozadí a synchronizuje obraz skrze Scene Graph pomocí FXT. [9]

Media engine slouží pro přehrávání audia a videa v JavaFX aplikacích. Mezi podporované audio formáty patří MP3, AIFF a WAV, a mezi video formáty patří FLV. Funkcionalita je poskytována kombinací tří komponent. Jedná se o `Media` objekt, reprezentující multimedi-

ální soubor, `MediaPlayer`, tento soubor přehrává a `MediaView` dokáže soubor zobrazit. Obrázek 2.2 zobrazuje vrstvy obalující externí multimediální soubor.



Obrázek 2.2: Vrstvy obalující multimediální soubor. [10]

Web engine poskytuje platformě schopnost spolupracovat s obsahem v podobě HTML5[8], CSS, JavaScript, DOM[9] a SVG[10]. Tento engine je založen na populárním open-source API nazývaném WebKit. Toto API se používá například v prohlížeči Safari, Amazon Kindle zařízeních a bylo v upravené verzi pod názvem Blink používáno i pro populární prohlížeč Google Chrome. [2][9]

2.2 Architektura JavaFX aplikací

JavaFX aplikace je možné vytvářet dvěma základními způsoby. První způsob je vývojářům dobře známý, jelikož se jedná o klasický způsob vývoje, jako tomu bylo při vývoji aplikací využívající Swing. Tedy zápisem grafické stránky aplikace v kódu pomocí jazyka Java a bez oddělení grafické a logické části aplikace. Je tak čistě na vývojáři jak svou aplikaci sestaví a rozdělí do balíčků či tříd. Ukázka vytvoření aplikace pomocí tohoto způsobu se nachází v kapitole 5.

Druhým způsobem vytváření aplikací je využití jazyka FXML pro definici grafického uživatelského rozhraní. Aplikace je konstruována principiálně tak, že obsahuje minimálně jednu třídu definující základní spouštěcí metodu `start()`, dále soubor definující uživatelské rozhraní – FXML dokument, a soubor připojený k FXML dokumentu definující kód

[8] HyperText Markup Language – značkovací jazyk definující obsah webových stránek

[9] Document Object Model – model definující xml dokumenty jako strom elementů

[10] Scalable Vector Graphics – určeno pro vytváření vektorové grafiky a ukládané jako XML

obsluhující události jednotlivých komponent FXML dokumentu. FXML dokumenty se dají tvořit jak ve vývojovém prostředí NetBeans, tak i v jakémkoliv XML editoru. Další možností vytváření FXML dokumentů je využití specializovaného nástroje na vytváření JavaFX uživatelských rozhraní. Tento nástroj je vytvářen přímo firmou Oracle a dokáže usnadnit vytváření uživatelských rozhraní tím způsobem, že uživatel nemusí znát FXML elementy a pouze přetahuje vybrané komponenty na pracovní plochu, zatímco se v pozadí generuje FXML dokument popisující právě vytvářené uživatelské rozhraní. Tento nástroj nese název Scene Builder, a je mu věnována celá kapitola 4.

Vzhledem k tomu, že je příručka koncipována jako studijní materiál pro kurz 4IT115, který neobsahuje výklad jazyka XML, doporučuje autor v tomto kurzu vyvíjet programy klasickým způsobem, tedy abstrahovat od zápisů uživatelských rozhraní pomocí FXML dokumentů. Zároveň s tímto doporučením musí autor podotknout, že mimo kurz je lepší v aplikacích využívat FXML dokumenty a s nimi spojené Controller soubory pro definici grafického uživatelského rozhraní.

2.3 FXML

FXML jazyk je založen na XML a byl vytvořen primárně pro účely vytváření uživatelského rozhraní jinak než psaním nativního Java kódu. FXML soubory poskytují předem definovanou strukturu pro vytváření uživatelského rozhraní odděleně od aplikační logiky. FXML dokument popisuje pouze vzhled aplikace, tedy rozmístění jednotlivých instancí komponent – prvků na obrazovce, a jejich velikost. K souboru je doporučeno připojit handlery událostí k jednotlivým ovládacím prvkům, umístěné v odděleném souboru. Soubory s handlery mohou být popsány v jazyce Java, Javascript nebo PHP. Handler události je kód spouštějící se při výskytu nějaké události vyvolané externím vstupem, například stisknutí tlačítka.

Jazyk FXML nemá vlastní validační schéma, má ale základní předdefinovanou strukturu popisující řazení elementů a atributů. Díky tomu, že předdefinovaná struktura FXML je přímo součástí Javy, může vývojář využívat oficiální dokumentaci pro kontrolu, jaké elementy a atributy jsou v daných situacích povoleny. Jelikož je jazyk založen na XML, je samozřejmě možné kód otevřít a upravovat v jakémkoliv XML editoru. V zásadě každá JavaFX třída je element a vlastnosti těchto tříd, jako název, text nebo velikost, mohou být

definovány jako atributy elementu. FXML je primárně určen pro návrh složitějšího uživatelského rozhraní,. Je vhodné jak pro vytváření statických formulářů nebo tabulek, tak pro vytváření dynamických návrhů za podpory doplňujících souborů obsahujících handlery událostí. [11]

Praktická ukázka vytváření uživatelských rozhraní pomocí jazyku FXML je obsažena v kapitole 4.6.1.

2.4 CSS

Platforma JavaFX také umožňuje měnit vzhled komponent pomocí CSS souborů. Základní vzhled všech JavaFX aplikací je také určen CSS souborem. Do verze platformy 2.0 byl základní styl aplikací nazývaný Caspian a od verze 8 přichází již zmíněný základní vzhled Modena, přičemž vzhled Caspian je stále součástí platformy. Uživatel má tedy možnost přepínat mezi starším a novým vzhledem, ale hlavně má možnost také tvořit své vlastní.

Detailnějšímu vysvětlení fungování a používání CSS souborů při určování vzhledu aplikací se věnuje samostatná kapitola 3.5.

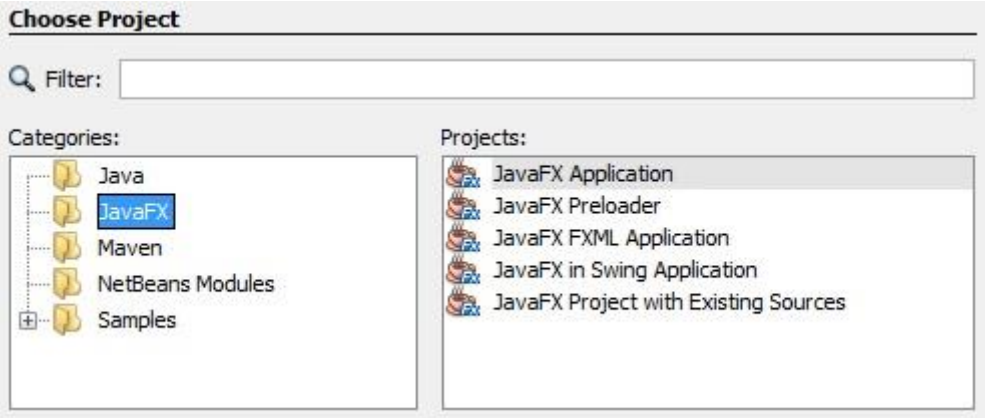
3 Příručka vytvoření aplikace na platformě JavaFX

Tato kapitola vysvětluje postup při vytvoření a náležitosti pro správné spuštění aplikace. Dále kapitola popisuje rozdílné struktury celé aplikace při vývoji klasické aplikace, nebo aplikace využívající FXML dokumenty. Následně jsou vysvětleny principy, které usnadňují vývoj a popsány vybrané základní kontejnery a práce s nimi. Na závěr je vysvětlena integrace CSS souborů s aplikací, možnosti změny vzhledu za běhu aplikace a sestavování hotových aplikací.

3.1 Vytvoření projektu

Vytvoření projektu pomocí vývojového nástroje NetBeans probíhá standardně ve dvou krocích.

1. V horní nabídce menu klikneme na File, a následně ze seznamu vybereme první volbu - New Project. Toto otevře okno pro výběr konkrétního projektu
2. Zvolení požadovaného typu aplikace z otevřené nabídky provedeme v levé části seznamu v této nabídce tak, že vybereme složku JavaFX, jak je znázorněno na obrázku 3.1.



Obrázek 3.1, Výběr projektu, [autor]

V pravé části se objeví celkem šest typů projektu, které můžeme vytvořit. Pro naše účely jsou zajímavé pouze projekty `JavaFX Application` a `JavaFX FXML Application`, kterým se věnují následující podkapitoly.

3. Po vybrání typu projektu stiskneme v dolní části obrazovky tlačítko Next a v zobrazeném formuláři vyplníme název projektu, jeho umístění na disku. Zároveň je doporučeno zaškrtnout možnosti vytvoření složky lib pro externí knihovny a vytvoření spouštěcí třídy celé aplikace.

Ze zbylých projektů stojí za zmínku JavaFX Preloader, označující speciální typ aplikace využívající se primárně jako doplněk ke standardním JavaFX aplikacím. Preloader se spouští před svou přiřazenou klasickou aplikací, a zobrazuje její postup načítání a přípravy.

3.1.1 Struktura klasické aplikace

Zvolením možnosti `JavaFX Application` se vytvoří klasická aplikace, která se skládá z jedné třídy obsahující základní spouštěcí metody. Tyto metody jsou konkrétně popsány v následující kapitole zabývající se vývojem aplikací. Třída má ihned po vytvoření implementován jednoduchý kód, který umí spustit jednoduchou aplikaci. Tato aplikace obsahuje jedno tlačítko, které po stisknutí vypíše text na konzoli.

3.1.2 Struktura FXML aplikace

Zvolením možnosti `JavaFX FXML Application` se vytvoří komplikovanější struktura aplikace tvořená celkem třemi soubory. Prvním souborem je FXML dokument. Druhým souborem je spouštěcí Java třída celé aplikace a třetím souborem je `Controller Class`, což je Java třída připojená k FXML dokumentu, obsahující definici všech použitých komponent v FXML dokumentu a jejich listenery. Základní třídě a třídě Controller jsou věnovány další kapitoly.

V FXML aplikaci je z důvodu přehlednosti a logické uspořádanosti kódu doporučeno pro každý FXML dokument, který ideálně popisuje pouze jednu obrazovku, vytvořit k němu přiřazený `Controller class.`. Jmenné konvence pro pojmenovávání FXML a Controller souborů zatím pevně stanoveny nejsou, avšak ve zdrojových kódech, se kterými se autor doposud setkal, a v samotném vývojovém prostředí NetBeans je konvence pojmenování FXML dokumentů následující: Před název souboru se typicky přidává předpona `FXML` a za název slovo `View`. Soubor může být tedy například pojmenován takto: `FXMLPřihlášeníView.fxml`. Controller soubory se pojmenovávají stejným stylem, s tím rozdílem, že místo koncovky `View` se v názvu vyskytuje slovo `Controller`. Vý-

sledné pojmenování Controller souboru může vypadat například takto: `FXMLPřihlášeníController.java`.

3.2 Programování JavaFX aplikace

Po vytvoření projektu lze přistoupit k samotnému programování aplikace. Z principu potřebuje každá aplikace vědět, kde má při spuštění začít, tedy který řádek kódu má provést jako první. Pro tento účel je definována podoba spouštěcí metody aplikace, která slouží jako startovní pozice, ze které se celá aplikace rozebíhá. Kapitola vysvětluje programování na JavaFX FXML Application.

3.2.1 Metoda pro spuštění aplikace

Tato metoda slouží jako startovní čára, ze které se každá aplikace rozebíhá. V klasických aplikacích postavených na platformě Java byla spouštěcí třída rozeznatelná podle toho, že implementovala metodu `main(String [] args)`. Aplikace postavená na platformě JavaFX taktéž implementuje tuto metodu, ale samotná metoda `main` nezaručí start JavaFX aplikace. JavaFX aplikace vyžaduje, aby se v projektu vyskytovala třída, která je potomkem třídy `javafx.application.Application`, a tedy i implementovala zděděnou abstraktní metodu `start()`. Zobrazení metod `main()` a `start()` v JavaFX aplikaci se vyskytuje v následujícím kódu 3.1. Metoda `main()` je při vytvoření nového projektu přítomna také, ale i v případě jejího vymazání se aplikace dokáže v pořádku spustit.

```
public class ZakladniMetody extends Application {

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

    @Override
    public void start(Stage primaryStage) throws Exception {
           // start aplikace
    }
}
```

Kód 3.1, Základní metody aplikace, [autor]

Třída `ZakladniMetody` v kódu 3.1 obsahuje v metodě `main(String[] args)` pouze jeden řádek, který volá veřejnou statickou metodu `launch()` z třídy `Application`, která zaručí následné spuštění celé aplikace. Spuštění celé aplikaceznamená vytvoření aplikace, zavolání metody `init()`, pokud je implementována, a následné zavolání povinně implementované metody `start(Stage primaryStage)`. Metoda `init()` je spouštěna po načtení základních tříd aplikace a je určena pro vývojáře, kteří potřebují vykonat inicializační procedury předem pro správný start aplikace. Třída Application nedefinuje v metodě `init()` žádný kód. Vývojář tedy může ve svém programu v metodě `init()` uvést vlastní kód, aniž by narušil fungování platformy. Aby se mohla aplikace řádně spustit, nestačí pouze zavolat prázdnou metodu `start()`, ale v této metodě je třeba se postarat minimálně o vytvoření a zobrazení okna aplikace. Vytvoření okna je jednoduché, a to hlavně díky tomu, že instance okna je vytvořena při vytváření celé aplikace a je dostupná přímo z parametru metody `start()`. Okno je od verze platformy 2.0 reprezentováno instancí třídy `Window`, jejímž potomkem je právě používaná instance třídy `Stage`, nebo `PopupWindow`. Základní tělo metody `start()` nezbytné pro řádný start aplikace je zobrazeno v kódu 3.2. Toto tělo metody se objevuje výhradně v JavaFX FXML Application využívající FXML dokumenty a Controller soubory.

```
@Override
    public void start(Stage stage) throws Exception {
        Parent root = FXMLLoader.load(getClass()
                  .getResource("FXMLDokument.fxml"));
        Scene scena = new Scene(root);

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

Kód 3.2, Tělo metody start(), [autor]

Prvním řádkem těla metody je načtení hierarchické struktury elementů FXML dokumentu a její uložení do instance třídy `Parent`, která umožňuje uchovávání objektů typu `Node` uspořádaných podle struktury daného FXML dokumentu. Vedle třídy `Parent` lze také pro tyto účely použít její potomky, tedy třídy `Group`, `Region` nebo `WebView`. Následuje řádek vytvářející instanci třídy `Scene`. Tato třída je kontejnerem pro celý obsah objektů k vykreslení na vrstvě Scene Graph. Poslední dva řádky kódu upravují vlastnosti instance třídy `Stage`, která, jak již bylo zmíněno výše, představuje okno aplikace. Předposlední

řádek určuje, jaká scéna se má do okna vykreslit, a poslední řádek zajišťuje zviditelnění celého okna.

V případě klasické aplikace tělo metody `start()` obsahuje ty samé řádky s výjimkou prvního řádku načítajícího FXML dokument. Jelikož jsou sestavované prvky uživatelského rozhraní uspořádány do stromové struktury, která obsahuje pouze jediný kořenový prvek (Node), předá se právě tento prvek jako argument při vytváření Scény.

3.2.2 Controller soubor

Třída Controller, zmíněná v předchozích kapitolách, je přímo přiřazena FXML dokumentu a obsahuje definici všech použitých komponent a jejich listenerů, tedy metod specifikujících akci při výskytu definované události. Tato třída musí implementovat rozhraní `Initializable`, které definuje jedinou metodu `initialize(URL url, ResourceBundle rb)`. Tato metoda umožňuje přidat základní data do seznamů, načíst obrázky a přiřadit je do příslušné komponenty. Vše probíhá při načítání FXML dokumentu.

Přiřazený Controller zároveň definuje komponenty vyskytující se v FXML dokumentu. Není však nezbytně nutné, aby tento soubor definoval všechny použité komponenty. Je tedy možné v FXML dokumentu popsat obrazovku s několika tlačítky, popisky a obrázky, a v Controller souboru definovat pouze jedno tlačítko. Controller souborem rozumíme třídu, která je potomkem třídy Controller a je přiřazena právě jednomu FXML dokumentu. Způsob, jak aplikace pozná, že Controller soubor definuje například právě třetí tlačítko zleva na obrazovce, je jednoduchý. Propojení probíhá pomocí atributu `fx:id`. Daná komponenta musí mít v FXML dokumentu definovaný `fx:id`, který by měl nejlépe vystihovat, k čemu komponenta slouží. Tedy tlačítko (`Button`), které slouží jako potvrzení přihlašovacích údajů s názvem „Přihlásit“, může mít `fx:id="prihlasit"`. Abychom mohli s touto komponentou pracovat v Controller souboru, musíme zavést proměnnou typu `Button` s názvem `prihlasit` s anotací `@FXML`. Anotací sdělujeme překladači, že následující řádek má přímou souvislost s FXML dokumentem. Tímto zaručíme, že můžeme s tlačítkem pracovat v kódu. Kód 3.3 zobrazuje část FXML dokumentu obsahující tlačítko a část Controller souboru.

```
FXML dokument:
<Button layoutX="126" layoutY="90" text="Přihlásit"
          onAction="#handleButtonAction" fx:id="prihlasit" />
```

```
Controller soubor:
@FXML
Button prihlasit;

@Override
public void initialize(URL url, ResourceBundle rb) {
    assert prihlasit != null : "fx:id=\"prihlasit\" was not injected: check your
           FXML file 'FXMLLogIn.fxml'.";
    prihlasit.setCursor(Cursor.HAND);
}

@FXML
private void handleButtonAction(ActionEvent event) {
    System.out.println("Tlačítko stisknuto");
}
```

Kód 3.3, FXML Controller injection, [autor]

Pokud má tlačítko definované `fx:id`, které je použité i v Controller souboru, můžeme s tímto tlačítkem v tomto souboru pracovat, aniž bychom ho zde znovu definovali. Do vytvořené proměnné se vloží objekt tlačítka existujícího v FXML dokumentu a tedy nehrozí výjimka `NullPointerException`. V kódu je také zobrazeno přidělení listeneru k tlačítku. V FXML dokumentu se vyskytuje atribut `onAction="#handleButtonAction"`, který specifikuje, která metoda v Controller souboru slouží tomuto tlačítku jako listener. Znak `#` před názvem metody v atributu `onAction` je povinný, ačkoliv není součástí názvu metody v Controller souboru. Zároveň metoda určená jako listener pro FXML komponenty také musí mít anotaci `@FXML`.

3.3 Properties a Bindings

Při vývoji uživatelských rozhraní se každý setká s problémem, jak synchronizovat jednotlivé prvky mezi sebou, a to logické s grafickými, nebo pouze logické a grafické mezi sebou. Tento problém řeší návrhové vzory jako model view controller (MVC), model view presenter (MVP), nebo model view view-model (MVVM). Předtím, než Properties vznikly, se vývojáři uchylovali ke známému návrhu tříd JavaBeans. JavaBeans jsou v podstatě klasické třídy, které se řídí určitými konvencemi pro psaní a nazývání přístupových metod. Zároveň JavaBean specification poskytovala JavaBeans API, které umožňovalo podporu

pozorování změn atributů. K jednotlivým proměnným třídy tedy bylo možné přiřadit listener. Jediné co JavaBeans API chybělo, bylo provázání jednotlivých vlastností mezi sebou. JavaBeans API postrádalo jakoukoliv robustnější cestu pro synchronizaci jednotlivých vlastností. Z toho důvodu vznikly v platformě JavaFX Properties, které elegantně a robustně slučují původní JavaBeans API a zároveň přidávají podporu provázání vlastností – Bindings. [2]

3.3.1 Properties

Properties (vlastnosti) jsou ve své podstatě obalové objekty pro atributy JavaFX objektů, mezi které patří například String nebo Integer. Properties obsahují přístupové metody hodnotě atributů, poskytují možnost přiřadit listener spouštějící se při každé změně hodnoty, a podporu provázání vlastností mezi sebou. Listener je objekt, který poslouchá a něco vykonává při nějaké definované události. Přiřadit k vlastnostem můžeme dva typy listenerů – `changeListener` a `invalidationListener`. Hlavní rozdíl mezi těmito listenery je v tom, co vyjadřují. `ChangeListener` vyjadřuje, že se hodnota vlastnosti změnila, a poskytuje jak starou, tak novou hodnotu vlastnosti. `InvalidationListener` vyjadřuje, že hodnota vlastnosti přestala být aktuální. [2]

Properties mohou být pouze dvou typů. `Read/Writeabe` a `Read-Only`. `Read/Writeable Properties` jsou vlastnosti, které je možné měnit i číst. JavaFX tyto vlastnosti nazývá `SimpleProperty`. Třídy těchto vlastností se různí podle hodnoty datového typu, kterou uchovávají. Pro datový typ `String` existuje třída `SipmleStringProperty`, pro `integer` `SimpleIntegerProperty` atd. Tyto vlastnosti mají definované jednoduché API pro získávání a nastavování hodnot, jelikož instanci třídy `Property` nelze používat stejným způsobem jako hodnotu datového typu. Mezi základní metody pro získání hodnoty drženého datového typu patří `.get()`, případně `.getValue()`. Pro modifikaci držené hodnoty slouží metody `set()` a `setValue()`. Tyto páry metod ve své podstatě provádějí tu samou operaci. Důvod existence dvou metod provádějících to samé je kvůli primitivním datovým typům. Metody `get()`/`set()` pracují s hodnotou, kdežto `getvalue()`/`setValue()` pracují s obalovými třídami primitivních datových typů. Pro datový typ `int` jsou metody definovány následovně: `set(int hodnota)` a `setValue(java.lang.Number hodnota)`. U referenčních datových typů

fungují obě metody stejným způsobem. Obecně je doporučeno používat základní metody `set()`/`get()`.

`Read-Only Properties` jsou vlastnosti, které se dají pouze číst. Změna hodnoty datového typu není přípustná. Vytvoření `Read-Only Property` je rozděleno do dvou kroků. Nejprve je nutné vytvořit instanci `Wrapper` třídy, která obsahuje zadanou hodnotu, a následně pomocí metody `.getReadOnlyProperty()` získáme `ReadOnlyProperty`, což je kopie hodnoty vyskytující se ve vytvořené `Wrapper` instanci, která je samozřejmě se zdrojovou hodnotou ve `Wrapper` instanci synchronizována. Tudíž při změně hodnoty ve `Wrapper` instanci se adekvátně změní i hodnota v Read-Only vlastnosti. Důvod, proč je vytváření Read-Only vlastností rozděleno do dvou kroků, je tedy následující: `Wrapper` instance uchovává hodnotu, která se ale může postupem času měnit. Instance `třídy Wrapper` proto při své konstrukci vytvoří dva objekty. První objekt je `Read-Writeable`, který má implementovanou metodu `set()`. Druhým objektem je Read-Only vlastnost, která je určena pouze pro čtení a je tedy bezpečně distribuovatelná. Tímto se při jakémkoliv požadavku na změnu obalované hodnoty jednoduše hodnota nastaví v instanci Wrapper třídy, která se propíše i do Read-Only vlastnosti. Tím pádem není nutné při změně obalované hodnoty konstruovat novou vlastnost s aktuální hodnotou. [2] Následující kód 3.4 ukazuje konstrukci jednoduché třídy s jedinou Read-Only vlastností.

```
public class Uzivatel{
    private ReadOnlyStringWrapper jmeno;

    Uzivatel{
    jmeno = new ReadOnlyStringWrapper("Jméno");
    jmeno.set("Martin");
    }

    public final String getJmeno(){
    return jmeno.get();
    }
    public ReadOnlyStringProperty getJmenoProperty(){
           return jmeno.getReadOnlyProperty();
    }
}
```

Kód 3.4, Read-Only Property, [autor]

Třída obsahuje jedinou vlastnost - jméno, kterou má možnost si sama interně měnit, což je předvedeno řádkem `jmeno.set("Martin")`, ale do vnějšího světa poskytuje pouze aktuální hodnotu řetězce a Read-Only vlastnost jména.

3.3.2 Bindings

`Properties` mohou být mezi sebou „provázány“, což umožňuje synchronizování hodnot mezi těmito provázanými vlastnostmi na základě změny jedné, nebo všech hodnot. Možnosti, jak provázat požadované vlastnosti, jsou celkem tři:

- `Bidirectional binding`,
- `High-level binding`
- `Low-level binding`.

`Bidirectional binding`, neboli oboustranné provázání, umožňuje provázat vlastnosti stejného datového typu, přičemž lze hodnotu měnit v kterékoliv z provázaných vlastností. Změna hodnoty je okamžitě vidět v ostatních provázaných vlastnostech. Kromě stejného datového typu musejí být také obě vlastnosti Read-Writeable. K tomu účelu slouží metoda `bindBidirectional()`. [2]

`High-level binding` využívá Fluent API a třídu `Bindings`. Fluent API jsou metody, které umožňují provádět operace s vlastnostmi. Každá vlastnost implementuje sadu funkcí Fluent API, které poskytují schopnost operace s ostatními vlastnostmi. `IntegerProperty` například definuje funkce `add()` a `multiply()`, kdežto `StringProperty` tyto metody nemá, ale definuje například metodu `isEqualTo()`. Třída `Bindings` je pomocná třída poskytující vlastní tovární metody, které fungují stejně jako u Fluent API. [2] [16]

```
package bindingdemo;

import javafx.beans.property.IntegerProperty;
import javafx.beans.property.SimpleIntegerProperty;
import javafx.beans.binding.NumberBinding;

public class Main {

    public static void main(String[] args) {
       IntegerProperty num1 = new SimpleIntegerProperty(1);
```

```
        IntegerProperty num2 = new SimpleIntegerProperty(2);
        IntegerProperty num3 = new SimpleIntegerProperty(3);
        IntegerProperty num4 = new SimpleIntegerProperty(4);
        NumberBinding total =
            Bindings.add(num1.multiply(num2),num3.multiply(num4));
        System.out.println(total.getValue());
        num1.setValue(2);
        System.err.println(total.getValue());
    }
}
```

Kód 3.5, Použití Fluent API společně s Bindings třídou, [16]

V kódu 3.5 je zobrazena deklarace čtyř vlastností držících číselnou hodnotu. Následně je vytvořena proměnná `total`, která obsahuje výslednou hodnotu vzorce. Vzorec kombinuje metody Fluent API a tovární metody třídy `Bindings`. `Bindings.add()` je jedna z továrních metod třídy `Bindings`, a metody `.multiply()` jsou součástí Fluent API. Výsledek proměnné total závisí na hodnotách jednotlivých proměnných vstupujících do vzorce. Z toho důvodu následuje výpis hodnoty proměnné `total`, změna proměnné num1 a znovu výpis hodnoty proměnné `total`. Ve výpisu se objeví 2 různé výsledky, aniž bychom museli explicitně přepočítávat vzorec a ukládat výsledek.

`Low-level binding` je používáno v případě, že vývojář potřebuje implementovat složitější vzorce, nebo potřebuje větší flexibilitu. Abychom využili Low-level binding, musíme vytvořit novou instanci jedné z `Binding` tříd, ve které v konstruktoru definujeme, které vlastnosti chceme používat, pomocí řádku `super.bind(v1, v2)`, a překryjeme funkci `computeValue()`. Následující kód 3.6 ukazuje použití Low-level binding pro výpočet obsahu koule. [2] [16]

```
DoubleProperty radius = new SimpleDoubleProperty(2);
DoubleBinding volumeOfSphere = new DoubleBinding() {
    {
    super.bind(radius); // initial bind
    }

    @Override protected double computeValue() {
    // Math.pow() (power) cubes the radius
    return (4 / 3 * Math.PI * Math.pow(radius.get(), 3));
    }
};
```

Kód 3.6, A Property Binding (volumnOfSphere) Using the Low-Level Binding Strategy, [2]

3.4 Kontejnery

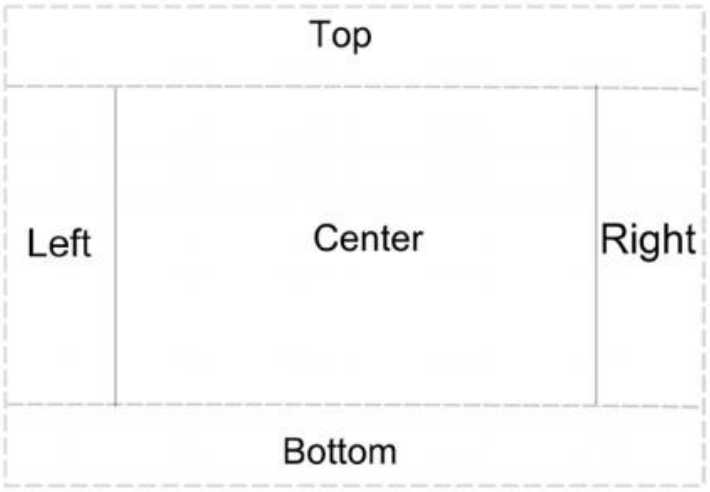
Kontejnery definují způsob rozložení a skládání grafických komponent v nich obsažených na obrazovku. Platforma JavaFX obsahuje množství použitelných kontejnerů, přičemž každý má své jedinečné vlastnosti a styl skládání komponent. V této podkapitole práce popisuje základní kontejnery používané při návrhu uživatelských rozhraní. Seznam použitelných kontejnerů je zobrazen i v nástroji Scene Builder v panelu komponent, který je popsán v kapitole 4.4.2.

3.4.1 AnchorPane

`AnchorPane` je jednoduchý kontejner umožňující pevné ukotvení komponent na určité místo na obrazovce, což je využitelné při vytváření statických prostředí. Kontejner podporuje překrývání, přičemž lze určit, která komponenta je umístěná nahoře a která pod ní. Každá umístěná komponenta má nastavené přesné odsazení od okrajů, přičemž tuto hodnotu můžeme ukotvit. Ukotvení způsobí, že se hodnota odsazení nebude měnit při změně velikosti okna.

3.4.2 BorderPane

`BorderPane` je známým kontejnerem, vyskytujícím se převážně na internetových stránkách. Tento kontejner rozděluje obrazovku do celkem pěti částí – Top, Bottom, Left, Right a Center. Obrázek 3.2 názorně zobrazuje toto rozdělení obrazovky.



Obrázek 3.2, BorderPane layout, [2]

Pokud využijeme pouze některé části a ostatní zanecháme nevyužité, budou v důsledku mít tyto ostatní části nulovou výšku a šířku a jejich původní prostor připadne centrální části. Zároveň podporuje každá část pouze jednoho potomka, což je jednoduše řešitelné pomocí vložení dalšího kontejneru, do kterého se následně vkládají další komponenty.

3.4.3 FlowPane

`FlowPane` označuje kontejner, který každou novou komponentu přidá za komponentu předchozí. Komponenty jsou tedy řazeny za sebe, přičemž si vývojář může zvolit, jestli se budou komponenty řadit do řádku, nebo sloupce. Kontejner se snaží využít maximum místa v řádku, případně sloupci v závislosti na velikosti okna. Pokud se mu další komponenta na zbylé místo nevejde, umísí jí do dalšího řádku/sloupce. Komponenty mají tedy jiné umístění na velké a jiné na malé obrazovce, případně mění svou pozici při změně velikosti okna. Tomuto kontejneru je možné nastavit hodnotu preferované šířky/výšky panelu. Pro tento účel je definována metoda `setPrafWrapLength()`.

3.4.4 GridPane

`GridPane` layout je kontejner fungující jako tabulka tvořená z definovaného počtu řádků a sloupců. V každé buňce se sice může vyskytovat více komponent, budou se však navzájem překrývat. Výhodou je, že můžeme definovat omezení pro jednotlivé buňky, konkrétně jejich výšku, šířku a chování při změně velikosti okna. Buňky jsou číslovány od nuly, a formát zápisu je (sloupec, řádek). První buňka má tedy index (0,0) a například buňka (1,2) označuje buňku ve druhém sloupci a třetím řádku.

3.4.5 HBox, VBox

`HBox` a `VBox` jsou kontejnery, které uspořádávají komponenty do řádku nebo do sloupce. `HBox`, neboli HorizontalBox, řadí své komponenty do jednoho řádku a VerticalBox do jednoho sloupce. Rozílem oproti kontejneru `FlowPane` je ten, že komponenty se nikdy nedají umístit do dalšího řádku/sloupce. Na to bychom potřebovali další kontejner `HBox`/`VBox`. Tyto kontejnery se nejlépe hodí pro malé skupiny komponent, které je potřeba držet pohromadě ve stejné struktuře. Například položky bočního menu chceme, aby byly

vždy pouze v jednom sloupci, a nezalamovaly se do druhého sloupce při malé velikosti okna.

3.5 Nastavení vzhledu pomocí CSS souborů

CSS soubory umožňují nastavit vzhled aplikace bez změny jejího chování. JavaFX poskytuje možnost vytvářet, modifikovat nebo použít již existující vzhledy pro vyvíjenou aplikaci. Pro definování vzhledu aplikace se používají CSS soubory. Tento typ souboru je primárně určen pro definování vzhledu HTML stránek odděleně od samotného obsahu dokumentu. Postupem doby se CSS začalo využívat pro definování vzhledu XHTML[11] a XML souborů. [17] Protože platforma JavaFX 8 definuje uživatelské rozhraní také pomocí FXML dokumentů založených na jazyku XML, čemuž je také přizpůsobena vrstva Scene graph, je možné zde použít CSS soubory také. Z důvodu podpory specifických JavaFX vlastností využívá platforma JavaFX CSS v rozšířené verzi, která se nazývá JavaFX CSS. Cílem JavaFX CSS je umožnit vývojářům obeznámeným s CSS soubory používat tuto technologii i pro JavaFX prostředí bez potřeby učit se zcela nový typ zápisu. [18]

Vytvoření stylu pro komponentu musí začínat definováním komponenty, na kterou se tento styl vztahuje. Určení stylované komponenty probíhá pomocí selektorů, kterým se věnuje následující kapitola. Po určení komponenty, na kterou se styl aplikuje, přichází na řadu definování stylu atributů komponenty. Definování stylu atributů JavaFX komponent musí začínat znaky `-fx-`, názvem atributu, dvojtečkou a požadovanou hodnotou atributu. Příklad definice stylu atributu je také zobrazen v kódu 3.7.

3.5.1 Selektory JavaFX CSS

Selektory jsou znaky pro usnadnění vyhledávání JavaFX komponent ve vrstvě Scene graph, ke kterým se vztahuje daný styl. Platforma JavaFX definuje dva druhy selektorů, `id` a `class`. Id selektor označuje instanci komponenty s jedinečným názvem. Každá komponenta definuje metodu `setId(String id)`, kterou lze nastavit jedinečný identifikátor této komponenty. Konvencí pro pojmenování komponent, které se skládají z více než jed-

[11] Extensible Hypertext Markup Language

noho slova, je vložit mezi slova pomlčku. Pokud má nějaká komponenta přidělen vlastní identifikátor, můžeme definovat styl aplikovatelný pouze pro ni. Abychom mohli přiřadit styl pouze komponentě, která má definovaný atribut `id`, musíme v CSS souboru přidat před `id` komponenty znak "#". [2] Kód 3.7 zobrazuje definici stylu tlačítka s přiděleným `id moje-tlacitko`.

```
#moje-tlacitko {
    -fx-padding: 5 5 5 5;
    -fx-font: 18pt "Verdana";
    -fx-text-fill: rgba(54, 153, 0);
}
```

Kód 3.7, využití id selektoru pro definici stylu konkrétní komponenty, [autor]

`Class selektor` je typ selektoru, který využívá pojmenované skupiny a uplatňuje styl na množinu komponent patřících do této skupiny. Jakékoliv komponentě lze přiřadit Class selektor pomocí metody `getStyleClass().add("Název supiny")`. Pomocí této metody je možné přiřadit komponentu do více skupin. V CSS souboru se před jméno skupiny komponent přidává znak tečka ".". Zároveň lze pomocí Class selektoru vybrat komponenty umístěné v určité části uživatelského rozhraní pomocí vzorce, který může vypadat následovně: `.vbox > .button {//kód}`. Tento vzorec specifikuje styl pro všechna tlačítka, která mají předka kontejner `VBox`. Zároveň lze specifikovat stejný styl pro více skupin najednou. Definování stejného stylu pro tlačítka a popisky by vypadalo následovně: `.button, .label {//kód}`.

3.5.2 Aplikování stylu v aplikaci

Aplikovat styly můžeme pomocí samostatných CSS souborů připojených ke scéně, nebo přímo v kódu. Přiřazení CSS souboru ke scéně je možné pomocí metody `.getStylesheets().add()`. V kódu 3.8 je znázorněn konkrétní příklad použití této metody.

```
Scene scena = new Scene(Parent root);
scena.getStylesheets().add(getClass()
    .getResource("cesta/nazevSouboru.css")
    .toExternalForm());
```

Kód 3.8, Připojení CSS souboru ke scéně, [autor]

Definování stylu komponent přímo v kódu probíhá pomocí metody `setStyle(String cssKod)`. Do parametru stačí zapsat pouze atributy, které chceme upravovat. Další možností je definovat styl pro celou aplikaci. Pro tyto účely slouží statická metoda `.setUserAgentStylesheet(String url);` z třídy Application. Tato metoda nastaví styl, který definuje vzhled všech prvků. Jako parametr této metody můžeme kromě adresy CSS souboru vložit i předdefinované vzhledy `Modena` a starší `Caspian`. Tyto vzhledy můžeme nastavit jednoduše pomocí konstant `STYLESHEET_CASPIAN` a `STYLESHEET_MODENA`. Nastavení základního vzhledu aplikace lze zajistit předáním hodnoty `null` v parametru této metody, což zaručí nastavení stylu Modena v JavaFX 8 aplikacích a stylu Caspian ve starších JavaFX aplikacích.

3.6 Sestavení a spuštění aplikace

Pokud vývoj aplikace probíhá bez problémů a chceme se podívat, jak aplikace funguje „naživo“, potřebujeme nejdříve aplikaci sestavit a následně ji spustit. Tyto operace je možné jednoduše provést pomocí NetBeans. JavaFX aplikace mají velkou výhodu v tom, že samotné sestavení aplikace je velice jednoduché a sestavené aplikace jsou připraveny na spuštění hned několika různými způsoby, které jsou popsány v podkapitole 3.6.2.

3.6.1 Sestavení aplikace

Sestavení aplikace pomocí vývojového prostředí NetBeans probíhá automaticky s každým spuštěním nebo explicitním sestavením aplikace pomocí příkazu `Clean and Build`. Každá JavaFX aplikace je automaticky sestavena a zabalena do několika souborů s příponami `JAR`[12], `JNLP`[13], `HTML` a `web-files` složky.

Soubor s příponou `JAR` obsahuje zkompilované třídy a ostatní soubory potřebné pro řádný běh aplikace. Těmito soubory mohou být v nejčastějším případě obrázky a audio soubory. Soubor je založen na komprimačním formátu ZIP a nedílnou součástí tohoto souboru je

[12] Zkratka z Java Archive

[13] Java Network Launch Protocol

dokument `Manifest.mf`. Tento dokument je vygenerován automaticky se vznikem JAR souboru a v souboru se smí vyskytnout pouze jednou. [19]

`JNLP` soubor je primárně určen pro spouštění a správu Java programů přes lokální síť nebo web. Soubor se též označuje jako `deployment descriptor` a slouží jako spouštěcí příkaz pro dva webové moduly – Web Start a spuštění v prohlížeči. `Deployment descriptor` je XML dokument popisující všechny prvky aplikace, požadavky na typ a verzi platformy, adresu JAR souboru a pravidla pro spouštění aplikace. [20]

HTML soubor obsahuje JavaScript kód pro spouštění aplikací z webových stránek a využívá Deployment Toolkit. Tento kód je možné přidat na jakoukoliv webovou stránku, čímž je zaručeno, že kdykoliv je tento kód zavolán, spustí se celá aplikace ve webovém prostředí. [19]

Kromě toho lze aplikaci sestavit do balíčku, který se nazývá `Self-Contained Application`. Tato varianta obsahuje vlastní JRE a instalátor. To znamená, že před samotným spuštěním je nutné pomocí instalátoru aplikaci nainstalovat. Výhodou této varianty je, že vývojář má kontrolu nad používanou verzí JRE tím, že aplikace používá vlastní prostředí, které si nainstaluje. Aplikace tedy běží i na hardwaru, který nemá žádný JRE nainstalován. Instalace může také probíhat bez povolení administrátora při použití ZIP archivu. Mezi hlavní nevýhody této varianty patří právě nutnost instalace aplikace. S tímto je spojená také větší velikost zabalené aplikace z důvodu obsažení celého JRE. Největší nevýhodou je ztráta platformové nezávislosti aplikace. Balíček se sestavuje na míru určitého operačního systému, což znamená, že pro každý systém musíme generovat další balíček. Poslední nevýhodou je update Java platformy. Aplikace využívající vlastní JRE by se měly sami starat o její aktualizaci. [21]

3.6.2 Spuštění aplikace

Jak již bylo zmíněno výše, sestavená aplikace je připravená na několik módů nasazení a spuštění. Mód spuštění označuje, jakým způsobem lze aplikaci spouštět. JavaFX aplikace lze spouštět v celkem čtyřech módech, přičemž každý mód ke svému úspěšnému spuštění potřebuje různé soubory vytvořené při sestavení aplikace.

Následující tabulka zobrazuje možné módy nasazení aplikace společně s potřebnými soubory pro jejich spuštění.

Tabulka 3.1, Files Required for Each Deployment Mode, [19]

Deployment Mode	**Files Required**
Standalone	JAR
Run in Browser	JAR, JNLP, HTML
Web Start	JAR, JNLP, HTML
Self-Contained Application	Folder that contains the application and supporting files

Standalone aplikaci lze jednoduše spustit dvojklikem na JAR soubor, nebo pomocí příkazové řádky zadáním příkazu `java –jar nazevSouboru.jar`. Podmínkou je, že příkaz voláte ve složce, kde se JAR soubor vyskytuje. Dále je možné v příkazu specifikovat hlavní třídu aplikace obsahující `main()`. Tento příkaz vypadá následovně: `java -cp`[14] `nazevSouboru.jar balíček.Třída`.

Run in Browser a Web Start varianty spuštění aplikace jsou možné přímo z prostředí NetBeans. Klasicky se každá aplikace vyvíjená v tomto prostředí spouští jako Standalone aplikace, což lze ale jednoduše změnit kliknutím pravého tlačítka na projekt, zvolením Properties, kategorii Run, a zhruba v polovině obrazovky se vyskytuje RadioButton označující preferované spouštění aplikace.

Při troše štěstí by měla aplikace běžet v prohlížeči bez problémů, avšak je pravděpodobné, že budeme muset prostředí prohlížeče a aplikaci upravit, abychom zajistili bezchybný běh. První úpravou je kontrola plug-inu prohlížeče. Java plug-in v prohlížeči je závislý na multiplatformní plug-inové architektuře nazývané NPAPI[15]. V prohlížeči Google Chrome od verze 42 je nutné tento plug-in ručně povolit, a od verze 45 je plánováno přestat podporovat tento plug-in úplně. Ruční povolení plug-inu v prohlížeči Google Chrome probíhá v následujících krocích:

1. Do URL řádku vložíme příkaz: `chrome://flags/#enable-npapi`
2. V zobrazeném seznamu klikneme na `enable NPAPI`
3. Restartujeme prohlížeč

[14] Lze zde místo –cp napsat –classpath. Jedná se o ten samý příkaz.

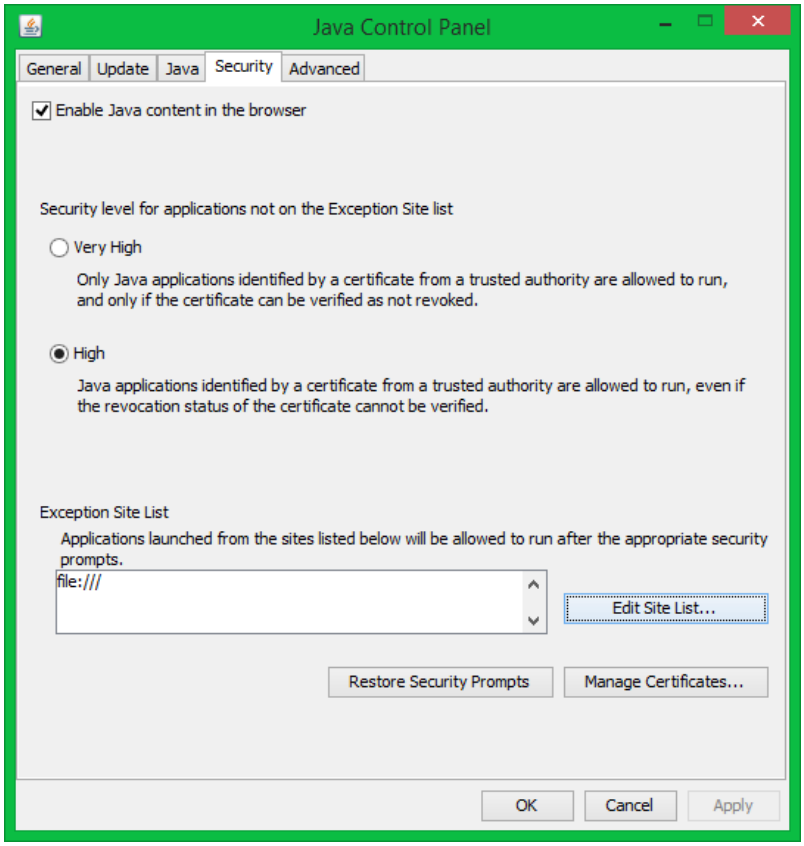
[15] Netscape Plugin Application Programming Interface

Tímto je prohlížeč připraven ke spouštění Java obsahu. [23]

3.6.3 Podepsání webové aplikace

Další možnou překážkou při spouštění vyvíjené aplikace na webu je bezpečnost. Od verze 8u20 platforma Java vyžaduje, aby jakákoliv aplikace spouštěná přes web a obsahující alespoň jednu `@FXML` anotaci byla digitálně podepsána důvěryhodnou certifikační autoritou. V případě pokusu o spuštění nepodepsané aplikace ve webovém prostředí se objeví hláška oznamující, že se aplikaci nepovedlo spustit z důvodu nesplnění bezpečnostních požadavků.

Nastavení bezpečnostních požadavků je možné v nastavení platformy Java. Toto nastavení se jmenuje „Configure Java“. Zde na kartě „Security“ se kromě úplného zakázání Java obsahu na webu dá nastavit požadovaná úroveň zabezpečení aplikací. Způsob jak lze aplikaci spustit, aniž bychom splnili požadavek na podepsání aplikace, je přidat výjimku pro tento soubor zde v konfiguraci platformy Java, jak je zobrazeno na obrázku 3.3.



Obrázek 3.3, Konfigurace platformy Java, [autor]

Jediný správný způsob, jak tuto situaci řešit, je si svou aplikaci podepsat platným certifikátem vydaným důvěryhodnou certifikační autoritou. Podepsání aplikace lze provést pomocí NetBeans, nebo využitím programů keytool pro generování klíčů a certifikátů a jarsigner pro podepisování JAR souborů. Tyto programy jsou dostupné ve složce nainstalovaného JDK, ve složce bin, a pracuje se s nimi pomocí příkazové řádky.

Podepisování JAR souborů pomocí NetBeans se dá jednoduše nastavit skrze vlastnosti projektu. Nastavení automatického podepisování aplikace lze nastavit ve třech krocích:

1. Na vybraný projekt klikneme pravým tlačítkem myši a zvolíme `Properties`
2. V otevřeném okně vybereme v seznamu položku `Deployment` a zaškrtneme možnost `Request unrestricted access (enable signing)` a možnost `Enable BLOB signing`
3. Stiskneme tlačítko Edit u těchto možností a zvolíme vlastní certifikát, kterým se od této chvíle bude aplikace podepisovat

4 Scene Builder 2.0

Tato kapitola je zaměřena na představení vývojového nástroje SceneBuilder verze 2.0 určeného pro vytváření uživatelských rozhraní pro platformu JavaFX. Kapitola nejdříve popisuje nástroj z hlediska obecných vlastností a předpokladů, následně popisuje historii vývoje tohoto nástroje. Dále je popsána instalace, spuštění, základní ovládání a práce s nástrojem.

4.1 Popis nástroje Scene Builder obecně

JavaFX Scene Builder je vizuální nástroj podporující technologii drag&drop pro vytváření grafického uživatelského rozhraní v JavaFX aplikacích bez potřeby psaní kódu. Uživatelé pouze „přetahují“ komponenty na pracovní plochu, upravují jejich vlastnosti a aplikují různé předem připravené styly, zatímco se v pozadí generuje výsledný FXML kód. (Práce se věnuje FXML v samostatné kapitole 2.3 a následně v kapitole 4.6.1 věnující se ukázkám kódu.) Tento výstup lze volně upravovat. Manuální změny v FXML kódu jsou ihned po uložení viditelné v návrhovém zobrazení nástroje. Scene Builder je napsán jako JavaFX aplikace a ukazuje tak sílu a eleganci této platformy. Nástroj je určen pouze pro vytváření aplikací postavených na platformě JavaFX, není tedy možné vyvíjet grafické rozhraní pro jiné jazyky postavené na jiných platformách. Nástroj umožňuje vytvářet grafickou část aplikace, avšak ostatní části (logika aplikace, či CSS soubory) nástroj vytvářet neumí. Doporučuje se tedy Scene Builder používat v kombinaci s dostupnými IDE. Výstupy z jiných vývojových nástrojů v podobě Java nebo CSS souborů lze napojit na projekt vytvářený ve Scene Builderu, stejně tak lze výstupní FXML dokument ze Scene Builderu využít v jiných vývojových nástrojích. Nejvíce je však Scene Builder integrován s NetBeans IDE, který, stejně jako samotnou platformu JavaFX, vyvíjí firma Oracle. [12]

4.2 Novinky Scene Builder verze 2.0

Otevřená beta verze tohoto nástroje byla uvedena v dubnu roku 2012 a od té doby prošla dalším vývojem. Momentálně je uvedena verze 2.0, která se objevila v dubnu roku 2014.

Nová verze 2.0 především podporuje všechny JavaFX API knihovny, které jsou součástí platformy Java, verze 8 (Java SE 8), a přináší řadu nových vlastností a vylepšení. Zároveň byla celá aplikace přebudována tak, aby její jednotlivé komponenty mohly být součástí větší aplikace, jako například samotného IDE. Tato vlastnost se nazývá JavaFX Scene Builder Kit. Na oficiálních stránkách dokumentace jsou dostupné návody jak používat SceneBuilder s různými IDE[16]. Dostupné jsou návody pro NetBeans, Eclipse a IntelliJ. Tato práce poskytuje návod na integraci nástroje pouze s NetBeans IDE.

Grafické komponenty vytvořené třetími stranami v JAR archivu mohou být nyní přidány do knihovny již existujících komponent. Byla přidána podpora nových již zmíněných javaFX 8 UI komponent (`TreeTableView`, `DatePicker` a `SwingNode`) přicházejících s novou Java SE 8. SceneBuilder nyní dokáže otevírat a upravovat FXML dokumenty obsahující 3D objekty a upravovat vlastnosti těchto objektů přes Inspector panel. Možnost vytváření 3D objektů zde ale stále chybí. V nejnovější verzi byl implementován nový kontejner TextFlow do knihovny grafických komponet. Upraveny byly také prvky Library panel, Hierarchy panel, Content panel, Inspector panel a Preview Window. [13][13]

4.3 Instalace nástroje

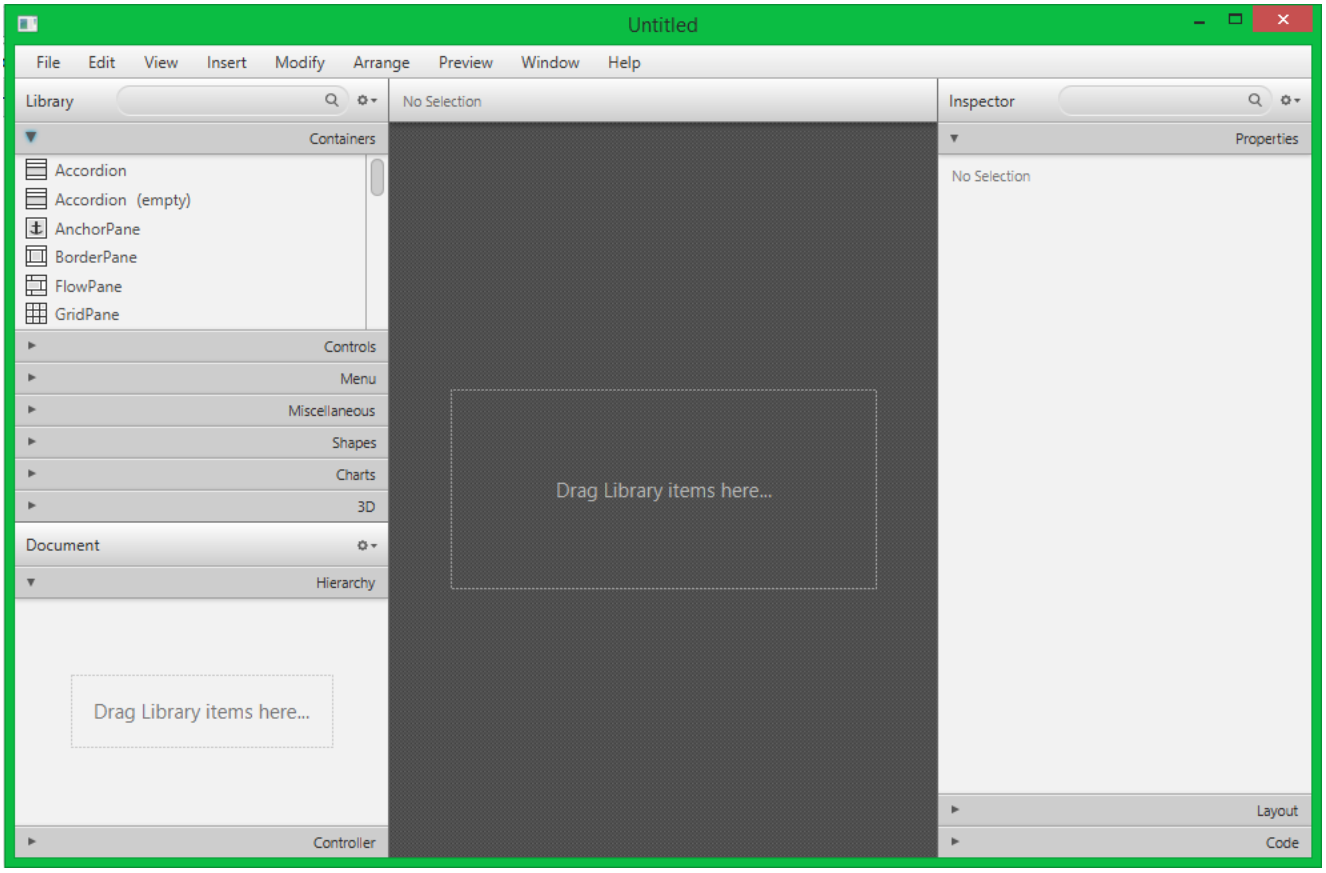
Před samotným stažením nástroje je potřeba ověřit, zda počítač, na který hodláme nástroj nainstalovat, splňuje systémové požadavky pro instalaci platformy Java. Pokud ano, je zapotřebí mít nainstalovanou minimálně verzi platformy Java SE 8. Adresa pro stažení nejnovější verze platformy Java a popis jednotlivých dostupných částí je zde: http://www.oracle.com/technetwork/java/javase/downloads/index.html . Autor doporučuje si stáhnout JDK, které v sobě obsahuje všechny potřebné součásti pro vytváření i spouštění Java aplikací. Následně budete přesměrováni na stránku, kde musíte přijmout licenční podmínky a následně si vybrat, pro jaký operační systém platformu stahujete.

Zároveň pro vytváření uživatelských rozhraní v nástroji se doporučuje nainstalovat si i některé z dostupných IDE. Jak již bylo zmíněno výše, v kombinaci se Scene Builderem se doporučuje používání NetBeans IDE. Vývojové prostředí NetBeans je možné si stáhnout a

[16] Integrated development environment

nainstalovat z této adresy: https://netbeans.org/downloads/. Pro účely této práce stačí verze prostředí určená pro Java SE.

Po úspěšné instalaci platformy a vývojového prostředí lze přejít k samotné instalaci nástroje. Nástroj je možné stáhnout z následující adresy: http://www.oracle.com/technetwork/java/javase/downloads/javafxscenebuilder-1x-archive-2199384.html. Jedná se o archiv dostupných verzí nástroje. Nás zajímá verze 2, tedy musíme vybírat pouze z tabulky s názvem „JavaFX Scene Builder 2.0 Related Downloads“, kde přijmeme licenční podmínky, a vybereme příslušný instalační soubor. Následně soubor otevřeme, postupně projdeme jednotlivými kroky instalačního procesu, a nástroj spustíme. Pokud vše proběhlo v pořádku, měl by se nástroj bez problémů spustit a být připraven na okamžitý vývoj uživatelských rozhraní. Následující obrázek 4.1 zobrazuje základní obrazovku nástroje ihned po spuštění.



Obrázek 4.1: Úvodní obrazovka nástroje SceneBuilder 2.0. [Autor]

Na úvodní obrazovce vidíme na každé straně jeden panel, obsahující vše potřebné pro výběr komponent umístitelných na scénu a správu jejich vlastností. Popisu prostředí nástroje se detailně věnuje následující kapitola.

4.4 Popis prostředí nástroje

Obrazovka nástroje je rozdělena na celkem tři části a menu. Z těchto tří částí jsou dvě zobrazeny jako panely po stranách nástroje, jak je vidět na obrázku 5.1. Nazvěme tedy prostřední šedou část nástroje návrhovou plochou, panel na levé straně nástroje panelem komponent a panel na pravé straně panelem vlastností. (Konkrétní oficiální názvy těchto panelů jsou zmíněny v příslušných podkapitolách.) Těmto panelům a liště menu se podrobněji věnují následující samostatné podkapitoly. V době vytváření této práce je základním jazykem nástroje angličtina, bez možnosti jiných jazykových variant. Z tohoto důvodu práce používá vedle oficiálních anglických názvů také autorem přeložené ekvivalenty těchto názvů.

Ve střední části nástroje je zobrazena tmavě šedá plocha s ohraničeným nápisem „Drag Library items here“. Jak již nápis naznačuje, jedná se o plochu, kam se umisťují jednotlivé komponenty. Plocha slouží i jako vizuální zpětná vazba toho, jak bude výsledné uživatelské rozhraní vypadat. Jakmile začne uživatel vkládat komponenty na plochu, začínají se v horní liště s původním nápisem „No Selection“, která je na obrázku 5.1 vidět nad horním koncem šedé plochy, vypisovat jednotlivé vrstvy vyskytující se nad danou vloženou či vybranou komponentou. Toto umožňuje rychlý přehled, kde přesně se komponenta nachází ve stromové struktuře dokumentu. Detailní pohled na tuto strukturu je také dostupný v dedikované části v panelu komponent, která je podrobněji popsána v následujících podkapitolách.

4.4.1 Menu

Lišta menu, nacházející se typicky v horní části obrazovky, obsahuje klasickou škálu akcí, která je dostupná v drtivé většině aplikací. Avšak lišta menu v tomto nástroji má pár schopností, které stojí za vyzdvižení.

V menu View stojí za zmínku možnost poslední akce, nesoucí název `Show Sample Controller Skeleton`. Tato akce dokáže zobrazit okno s naznačenou kostrou souboru starajícího se o logickou část uživatelského rozhraní. Vlastnosti a důležitost tohoto typu souboru budou popsány v následujících kapitolách. Tuto kostru je možné automaticky zobrazovat v několika verzích. Základní zobrazení vypisuje pouze ty řádky kódu, které by se měly v každém `Controller`u připojeném k tomuto uživatelskému rozhraní objevit. Zob-

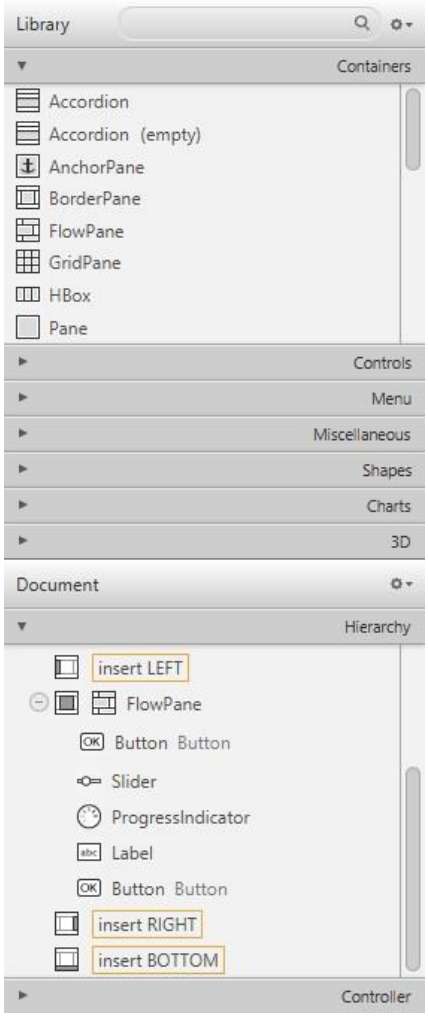
razení kostry lze pomocí zaškrtávacích polí v pravém dolním rohu rozšířit. Kostru můžeme rozšířit buď o komentáře jednotlivých řádků, popisující činnost řádku, pomocí zaškrtávacího pole s názvem „Comments“. Zároveň lze úvodní kostru rozšířit pomocí zaškrtávacího pole „Full“ o základní překrytou metodu `initialize()`, testující správné načtení instancí komponent vyskytujících se v uživatelském rozhraní. Samozřejmě vedle doplněné metody jsou příslušně doplněny i importy potřebných knihoven. Kostru lze pomocí tlačítka „Copy“ zkopírovat do schránky a případně jí použít při vytváření `Controller` souboru.

V tomto menu je také volba `Show Sample Data`, která umožňuje do vytvářeného uživatelského rozhraní vložit ukázková data pro názornou simulaci reálného zobrazení. Toto je samozřejmě demonstrovatelné pouze u komponent, které jsou schopné zobrazovat data. Mezi tyto komponenty patří například rozbalovací seznamy, textová pole, či tabulky. Tato možnost je nejvíce přínosná při zobrazení náhledu rozhraní, dostupného v části lišty menu nesoucí název `Preview`.

Menu `Preview` je zaměřeno na zobrazení náhledu na vyvíjené uživatelské rozhraní. V tomto menu se nachází volba `Show Preview in Window`, která způsobí spuštění vytvářeného uživatelského rozhraní. Tato ukázka slouží k praktickému vyzkoušení vzhledu a chování grafických prvků. Naprogramovaná logika aplikace ale v této ukázce funkční není. Zobrazení aplikace může být ve formě samotných grafických komponent nedržící v sobě žádná data, nebo s aktivovanou výše zmíněnou možností Show Sample Data je aplikace zobrazena se základními ukázkovými daty v sobě. Lze tedy prakticky vyzkoušet zalamování textů v textových polích, či správně zvolenou velikost buněk tabulky.

4.4.2 Panel komponent

Tento panel se nachází na levé straně nástroje, jak je vidět na obrázku 5.1. Panel se skládá ze dvou částí: samotného panelu komponent, v anglických dokumentacích nazývaný `Library Panel`, a dokumentového panelu, v angličtině `Document Panel`. V horní části celého panelu je také umístěno vyhledávací pole. Pole umožňuje rychlé vyhledávání komponent bez nutnosti se proklikávat všemi kategoriemi a příslušnou komponentu hledat. Detail panelu je zobrazen na obrázku 4.2



Obrázek 4.2: Panel komponent. [Autor]

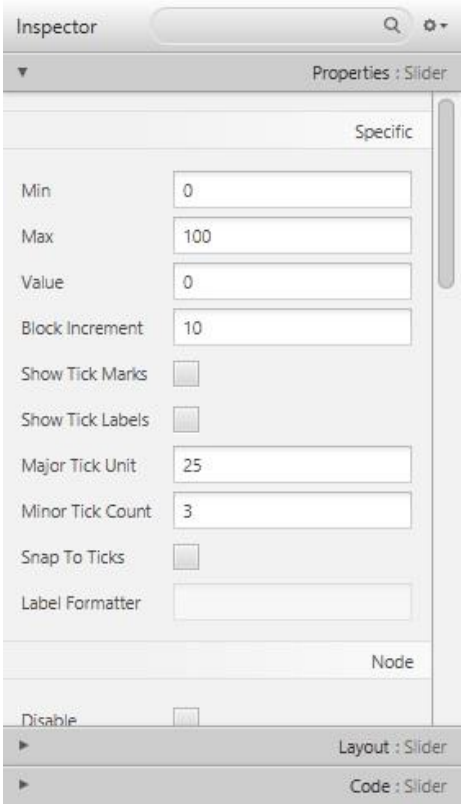
`Library panel` obsahuje všechny dostupné grafické komponenty použitelné při tvorbě uživatelského rozhraní. Panel je rozdělen do celkem sedmi kategorií. Zároveň všechny komponenty, které jsou novinkou verze JavaFX 8 jsou označeny závorkou (FX8). V nové verzi platformy i nástroje byla přidána kategorie 3D, která obsahuje pouze prvky osvětlení a kamera, které přišly s novou verzí platformy.

`Document panel` obsahuje pouze dvě kategorie. První kategorie nese název `Hierarchy`, a zobrazuje vizuální podobu stromu prvků vytvářeného FXML dokumentu. Tento strom lze po jednotlivých vrstvách rozbalovat a přinutit tak nástroj k zobrazení kompletní struktury dokumentu. Ovšem při zobrazování kompletní struktury se stává strom nepřehledným, a jelikož `Document panel` sdílí levou část nástroje s `Library panel`em, nelze zobrazit celý strom najednou. Zároveň je možné komponenty z `Library panel`u vkládat přímo do tohoto stromu. Změny ve stromu dokumentu jsou okamžitě viditelné na návrhové ploše.

Druhá kategorie nese název `Controller`, a umožňuje uživateli zvolit `Controller` soubor výchozí pro vyvíjené uživatelské rozhraní pomocí rozbalovacího seznamu v horní části tohoto panelu. Rozbalovací seznam obsahuje všechny dostupné třídy v balíčku, ve kterém je umístěn samotný FXML dokument. Pod rozbalovacím seznamem se nachází tabulka všech komponent s jejich přiřazeným `fx:id`. Soubor `Controller`, jeho důležitost, vlastnosti a struktura je vysvětlena v následujících kapitolách.

4.4.3 Panel vlastností

Jak již bylo zmíněno výše, panel vlastností se nachází na pravé straně základní obrazovky nástroje (obrázek 5.1). V horní části je umístěno vyhledávací pole, které, stejně jako u panelu komponent, umožňuje rychlé vyhledávání. Toto pole však nevyhledává jednotlivé komponenty, ale vlastnosti jedné vybrané komponenty umístěné na návrhové ploše. Celý panel se dělí na tři kategorie: `Properties`, `Layout` a `Code`. Panel vlastností je zobrazen na obrázku 4.3.



Obrázek 4.3: Panel Vlastností. [Autor]

Kategorie `Properties` se zabývá vlastnostmi dané komponenty. Tato kategorie je interně rozdělena do dalších podkategorií podle typu vybrané komponenty. Každá komponenta má

sadu základních, a specifických vlastností. Mezi základní vlastnosti většiny komponent patří kategorie „Node“, obsahující vlastnosti `Disable`, `Opacity`, `Node Orientation`, `Visible`, `Focus Traversable`, `Cache Shape`, `Center Shape`, `Scale Shape`, `Opaque Insets`, `Cursor` a `Effect`. Mezi nejzajímavější vlastnosti patří například vlastnost `Opacity`, která určuje průhlednost komponenty. Tato vlastnost nabývá hodnot od 0 do 1 včetně, přičemž 0 znamená úplnou průhlednost a 1 plnou sytost. Dále vlastnost `Visible`, určující viditelnost a nabývající hodnot true pro viditelnost a false pro neviditelnost komponenty. Za zmínku stojí také vlastnosti `Cursor` a `Effect`.

Vlastnost `Cursor` nám dovoluje nastavit zobrazení kurzoru při jeho přítomnosti na komponentě. Obrázek kurzoru můžeme vybrat z 20 různých možností, obsahující klasický kurzor, obrázek zavřené či otevřené dlaně, nebo kurzory zobrazující se při změně velikosti oken v operačním systému.

Vlastnost `Effect` dovoluje přiřadit komponentě jeden z široké nabídky dostupných efektů. Každý efekt má přitom svou vlastní škálu vlastností a funkcí. Mezi nejpoužívanější přiřazované efekty patří `Shadow`, `Inner Shadow`, `Motion Blur` a `Reflection`. Efekt `Shadow` a `Inner Shadow` označují stín vytvoření stínu k dané komponentě, a to buď vnější (`Shadow`), nebo vnitřní (`Inner Shadow`). U obou těchto stínů lze upravovat výška, šířka, nebo posun stínu po ose X a Y pro vytvoření simulace nasvícení komponenty z nějaké strany.

Druhou kategorií tohoto panelu je kategorie `Layout`, která umožňuje nastavení velikosti, umístění a zobrazení komponenty. U kontejnerů lze navíc oproti obyčejným komponentám nastavovat horizontální a vertikální mezeru mezi vnitřními komponentami. Všechny komponenty mají společné nastavitelné vlastnosti velikosti, otočení a měřítka. Velikost komponenty je určena výčtem minimální, preferované a maximální hodnoty pro výšku i šířku, přičemž je možné tyto hodnoty zadat jako `USE_COMPUTED_SIZE`, což nám zaručí použití automaticky vypočítané hodnoty. Tuto volbu lze nastavit pro všechna pole. Další možností, kromě natvrdo napsaných rozměrů, je vyplnit do polí pro maximální a minimální hodnotu `USE_PREF_SIZE`, která se postará o to, aby se komponenta zobrazila v preferované velikosti, kterou nelze měnit. Další možností u pole pro maximální hodnotu je volba `MAX_VALUE`, zajišťující prakticky neomezenou velikost.

Dále je zde volba otočení komponenty. Tato volba má několik nastavitelných parametrů. Jedná se o určení úhlu natočení, určení osy otáčení a navíc hodnota posunu po ose. Hodno-

ty posunu, v anglickém prostředí nazývány `Translate` slouží k určení hodnot kam se má komponenta při animaci posunout. Výhodou je, že při zadání těchto parametrů aplikace počítá s tím, že se komponenta bude pravděpodobně pohybovat po scéně a adekvátně k tomu při animaci nastaví i novou pozici této komponenty. Při běhu aplikace a spouštění animací tak nehrozí změna původních nastavení pozice.

Poslední kategorií v tomto panelu je kategorie `Code`. Tato kategorie je pro fungování celé aplikace nejdůležitější, jelikož zajišťuje propojení s `Controller` souborem, o jehož identifikaci se stará `Document panel`, který je popsán v předchozích kapitolách. Tato kategorie obsahuje nastavení jedinečného identifikátoru komponenty a výčet všech událostí, které komponenta podporuje. Těmto událostem lze jednotlivě přiřadit handlery událostí definované v `Controller` souboru. Podrobněji se práce věnuje propojení výstupního FXML dokumentu a `Controller` souboru v následujících kapitolách.

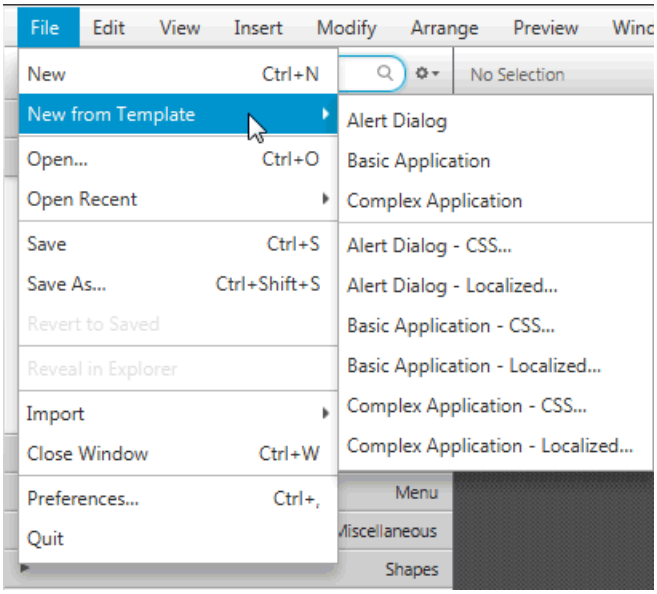
4.5 Integrace nástroje s NetBeans IDE

Scene Builder je schopný fungovat jako samostatná aplikace, avšak, jak již bylo zmíněno, je dobré jej integrovat s některým z dostupných vývojových prostředí, nejlépe s NetBeans IDE. Integrace probíhá jednoduše a z velké části automaticky. Samotná integrace závisí zcela na prostředí NetBeans. Při vytvoření nového projektu obsahující FXML dokument se při pokusu o otevření tohoto souboru NetBeans rovnou snaží o spuštění nástroje Scene Builder s parametrem otevíraného FXML dokument. Pokud je nástroj Scene Builder nainstalován na přednastavenou adresu, neměl by nastat žádný problém a nástroj by měl otevřít požadovaný soubor.

Pokud uživatel nainstaloval nástroj Scene Builder do jiné než přednastavené složky, je potřeba cestu k této složce definovat v NetBeans. Předefinování adresy probíhá následovně: V NetBeans v anglickém prostředí klikneme v hlavním menu na Tools, otevřeme Options, zde vybereme kategorii Java, dále záložku JavaFX a objeví se rozbalovací textové pole s popiskem „Scene Builder Home“, které rozbalíme a zvolíme možnost „Browse“. Dále se pomocí průzkumníku dostaneme do složky obsahující platnou instalaci nástroje Scene Builder. Toto by mělo zaručit bezchybnou integraci těchto dvou programů. [15]

4.6 Vytváření uživatelských rozhraní

Používání programu je velice intuitivní a jednoduché. Po spuštění můžeme ihned začít pracovat na novém projektu, nebo můžeme načítat již existující projekty. Pokud již máme nějaké projekty rozpracované, dostaneme se k nim z horního menu, kde klikneme na `File`, `Open Recent` a vybereme z nabídky existujících projektů. Dále je zde možnost načíst předpřipravené základy uživatelských rozhraní. Pro načtení předpřipravených projektů klikneme znovu v horním menu na `File`, dále na `New from Template`, a vybereme jeden ze tří předpřipravených projektů, které se nacházejí v horní části zobrazené nabídky na obrázku 4.4.



Obrázek 4.4: Výběr předpřipravených projektů, [14]

Jakmile je načtený template, spustí se další instance nástroje Scene Builder obsahující vybraný template. Následně je možné začít s vytvářením uživatelského rozhraní jednoduše tím, že z panelu komponent vkládáme požadované komponenty na pracovní plochu. Nelze však jednotlivé komponenty jen tak začít „házet na hromadu“. Jako první je potřeba umístit na plochu kontejner z kategorie `Containers` v levém panelu komponent, do kterého teprve můžeme začít umisťovat ostatní komponenty, případně i další kontejnery. Popis základních kontejnerů a práce s nimi je popsána v dalších kapitolách. Aplikace se sestavuje v principu tak, že jednotlivé komponenty vkládáme do jiných, čímž tvoříme zmiňovanou stromovou strukturu výsledného dokumentu. Toto umožňuje lepší kontrolu nad celým designem a jednodušší generování FXML kódu.

4.6.1 Struktura a náležitosti souboru FXML

Generovaný FXML dokument obsahuje kompletní uživatelské rozhraní v textové formě. FXML dokument se v mnoha ohledech neliší od klasického XML dokumentu, jelikož to ve své podstatě XML dokument je. Lehce rozšířená je u těchto souborů hlavička, která obsahuje kromě klasické definice verze jazyka XML a použitého kódování i importy komponent vyskytujících se níže v dokumentu. Ukázka hlavičky souboru je zobrazena v kódu 1.

```
<?xml version="1.0" encoding="UTF-8"?>
<?import javafx.scene.control.*?>
<?import java.lang.*?>
<?import javafx.scene.layout.*?>
```

Kód 5.1, Hlavička FXML dokumentu, [autor]

Na prvním řádku Kódu 1 se vyskytuje klasická definice všech XML souborů, označující použitou verzi jazyka a použité kódování. Na dalších řádcích jsou zobrazeny importy knihoven z platformy Java a JavaFX. Seznam knihoven se mění v závislosti na použitých komponentách. Za hlavičkou souboru se nachází tělo, které obsahuje popis vytvořeného uživatelského rozhraní. Následující kód zobrazuje tělo souboru popisující jednoduché uživatelské rozhraní. Parametry kontejneru nejsou zobrazeny z důvodu přehlednosti kódu.

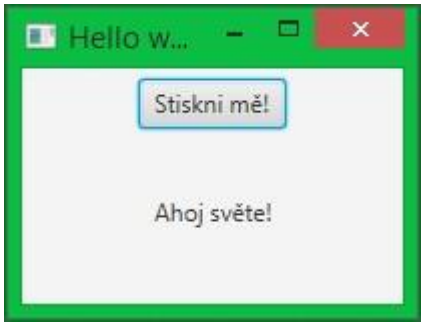
```
<BorderPane>
   <center>
      <Button text="Stiskni mě!" BorderPane.alignment="CENTER" />
   </center>
   <top>
      <Label text="Ahoj světe!" BorderPane.alignment="CENTER" />
   </top>
</BorderPane>
```

Kód 5.2, Syntaxe jazyka FXML, [autor]

Tělo programu začíná tagem `<BorderPane>` označující běžně používaný kontejner pro uchovávání ostatních komponent v předem definované struktuře. Detailněji se kontejneru BorderPane věnuje jedna z následujících kapitol. Prozatím stačí zmínit, že kontejner BorderPane má svou pracovní plochu rozdělenou do celkem čtyř částí, přičemž při vkládání komponent do každé části máme možnost rozhodnout, kam se bude komponenta zarovnávat. V kódu výše je použito zarovnání komponent na střed v rámci části kontejneru, do které spadají.

Za tagem `<BorderPane>` se nachází tag `<Center>`, označující jedu ze čtyř částí kontejneru. V tomto tagu se vyskytuje tag `<Button>`, označující tlačítko s vypsanými parametry `Text`, definující zobrazovaný název tlačítka, a `alignment`, určující zarovnání v příslušné části kontejneru. Následuje zavírací tag `</Center>` a nový tag `<Top>`, ve kterém se vyskytuje `<Label>` s parametry `Text` a `alignment`.

Tento jednoduchý příklad FXML zápisu definuje uživatelské rozhraní zobrazené na obrázku 4.5.



Obrázek 4.5, Výstup ukázkového FXML dokumentu, [autor]

5 Ukázková JavaFX aplikace Hledání Měst

Kapitola popisuje vytvoření jednoduché aplikace postavené na platformě JavaFX. Aplikace je vytvářena jako JavaFX Application, tedy bez FXML dokumentů a Controller souborů. Kapitola popisuje požadavky na vytvářenou aplikaci, následně její architekturu, třídy a jejich obsah. Následuje kapitola věnující se popsání implementace grafické části aplikace. Kompletní kód aplikace se nachází v příloze 1.

5.1 Zadání aplikace

Základní požadavky na vytvářenou aplikaci jsou především demonstrace vytváření uživatelského rozhraní pomocí jazyka Java, dále programové rozdělení aplikační logiky od uživatelského rozhraní a práce s událostmi. Aplikace je koncipována jako jednoduchá hra s mapou, kde má uživatel za úkol najít zadané hlavní město na mapě. Aplikace počítá body za každé uhádnuté město a odečítá za každé přeskočené město. Aplikace umožňuje ukončení hry a vypsání výsledku do samostatného okna.

5.2 Architektura aplikace

Aplikace se skládá celkem ze tří tříd a obrázku mapy. Spouštěcí třída nese název `HledaniMest` a slouží pouze pro vytvoření instance aplikační logiky a instance třídy vytvářející uživatelské rozhraní, které předá jako parametr konstruktoru právě vytvořenou instanci logiky. Druhou třídou v aplikaci je třída `Logika`, zajišťující seznam dostupných hlavních měst a poskytující veřejné metody potřebné pro správné fungování aplikace. Poslední třídou je třída `GUI`, vytvářející kompletní uživatelské rozhraní, což zahrnuje správu zobrazeného okna, definování kódu spouštěného při definované události a dotazy do vlastní instance logiky, která byla předána jako parametr konstruktoru třídy GUI. Společně s těmito třídami se v aplikaci vyskytuje obrázek politické mapy Evropy. Mapa v aplikaci reprezentuje hrací plochu.

5.3 Implementace aplikace

Výsledný vzhled vytvářené aplikace Hledání měst je zobrazen na obrázku 5.1, a obrazovka vyskakující při ukončení hry je zobrazena na následujícím obrázku 5.2.

Obrázek 5.1, Vzhled ukázkové aplikace Hledání měst, [autor]

Obrázek 5.2, Okno oznamující výsledné skóre hry, [autor]

Vše začíná vytvořením nového projektu ve vývojovém prostředí NetBeans. Typ aplikace zvolíme JavaFX Application, pojmenujeme `HledaniMest` a zaškrtneme volbu vytvoření aplikační třídy. Projekt nyní obsahuje jedinou třídu `HledaniMest.java`. Nyní vytvoříme

prozatím prázdné třídy `Logika` a `GUI`, které následně přesuneme do samostatných balíčků z důvodu přehlednosti a fyzickému oddělení grafiky od logiky.

Nyní doplníme do spouštěcí třídy do metody `start()` kód, který vytvoří instanci logiky, kterou předá jako parametr nové instanci grafiky. Kód metody start je zobrazen v kódu 5.1.

```
@Override
    public void start(Stage primaryStage) {
        Logika logic = new Logika();
        GUI gui = new GUI(logic, primaryStage);
    }
```

Kód 5.1, Metoda start ukázkové aplikace, [autor]

Nyní je čas vrhnout se na vytvoření uživatelského rozhraní. Celá aplikace se bude zobrazovat v jednom okně. Okno by mělo obsahovat lištu nabídky Menu, jako většina současných aplikací, dále řádek, kde se vypisuje zadání hry, ukazatel skóre a tlačítko pro získání dalšího zadání. Většinu aplikace by měla vyplňovat samotná mapa, na které se celá hra odehrává. Nakonec je možné přidat pod mapu řádek, do kterého se vypisuje nápověda ke hře.

Pro začátek vytvoříme lištu nabídky Menu, která se bude skládat z jednoho rozbalovacího seznamu obsahujícího tři položky. Vytvoření celé lišty je zobrazeno v kódu 5.2

```
private void initMenuBar() {
        newGame = new MenuItem("Nová Hra");
        enableHints = new MenuItem("Nápověda");
        endGame = new MenuItem("Konec Hry");

        // vložení položek menu do rozbalovacího seznamu
        menu = new Menu("Menu");
        menu.getItems().addAll(newGame, new SeparatorMenuItem(),
                enableHints, endGame);

        // vytvoření lišty menu a přidání rozbalovacího seznamu
        menuBar = new MenuBar();
        menuBar.getMenus().add(menu);
    }
```

Kód 5.2, Vytvoření lišty nabídky Menu, [autor]

Na prvních řádcích kódu 6.2 je vytvoření třech instancí třídy `MenuItem`, tedy položek menu, které se následně vloží do nově vzniklé instance třídy `Menu` reprezentující rozbalovací seznam, který je možná vložit rovnou na lištu, nebo je vkládat do jiných Menu instancí. Do

této proměnné menu vložíme pomocí metod `getitems().addAll()` vytvořené položky menu společně s instancí třídy `SeparatorMenuItem`, která se zobrazí jako horizontální čára vizuálně oddělující položky menu.

Následně můžeme vytvořit horní panel aplikace obsahující zmíněné zadání úkolu, ukazatel skóre a tlačítko pro získání nového zadání. Tyto komponenty vložíme do kontejneru GridPane mající strukturu klasické tabulky. Kontejner je použitý hlavně z důvodu možnosti jednoduchého zarovnávání sloupců v prostoru aplikace.

```
task = new Text("Hledejte na mapě, začněte tlačítkem Nová hra v Menu");
score = new Text(SCORE + "0");
nextCity = new Button("Další město");

topPanel = new GridPane();

ColumnConstraints taskColumn = new ColumnConstraints();
taskColumn.setHalignment(HPos.LEFT);
taskColumn.setHgrow(Priority.ALWAYS);
topPanel.getColumnConstraints().add(taskColumn);

ColumnConstraints scoreColumn = new ColumnConstraints();
scoreColumn.setHalignment(HPos.RIGHT);
topPanel.getColumnConstraints().add(scoreColumn);

ColumnConstraints buttonColumn = new ColumnConstraints();
buttonColumn.setHalignment(HPos.RIGHT);
topPanel.getColumnConstraints().add(buttonColumn);

topPanel.addRow(0, task, score, nextCity);
```

Kód 5.3, Vytvoření horního panelu aplikace, [autor]

V kódu 5.3 jsou nejprve vytvořeny proměnné, které jsou obsahem panelu. Na dalších řádcích probíhá vytváření kontejneru a jeho sloupců. První sloupec nesoucí název taskColumn má nastavené zarovnání na levou stranu prostoru a na dalším řádku má nastavený růst, který v tomto případě označuje, že daný sloupec bude zabírat tolik místa, kolik bude aktuálně k dispozici. Další dva sloupce jsou identické s tím rozdílem, že nemají nastavený růst. Budou se tedy chovat tak, že zaberou jen tolik místa, kolik potřebují. Poslední řádek do vytvořeného kontejneru přidá řádek na pozici 0, který bude ve sloupcích obsahovat jednotlivé komponenty vytvořené na začátku tohoto kódu.

Poslední částí je samotná hrací plocha aplikace. Způsob vytvoření hrací plochy zachycuje kód 5.4.

```
picture = new ImageView("/gui/Evropa.JPEG");
guessedPosition = new Circle(3, Paint.valueOf(Color.GREEN.toString()));
hintText = new Text(HINT_TITLE);

picturePanel = new Pane(picture, guessedPosition);

logic.getCityPoints().stream().forEach(p -> { // vyznačení bodů měst do mapy
    Circle circle = new Circle(p.x, p.y, 2);
    picturePanel.getChildren().add(circle);
});

playSurface = new VBox(picturePanel, hintText);
```

Kód 5.4, Vytvoření hrací plochy, [autor]

První řádek vytvoří instanci třídy `ImageView`, která je určena pro uchovávání obrázků, a které je v konstruktoru předán řetězec označující cestu k obrázku. Následující řádek vytváří kolečko, které slouží pro označení pozice na obrázku, kam uživatel kliknul. Následuje vytvoření textu nápovědy a vytvoření panelu obrázku, do kterého jsou vloženy první dvě komponenty `picture` a `guessedPosition`. V dalším řádku se začínají do panelu obrázku vkládat malé body označující všechna města, které logika spravuje. Vkládání bodů je provedeno pomocí novinky Javy 8 – proudů. Z instance logiky se pomocí metody `getCityPoints()` získá kolekce bodů. Pro každý bod v této kolekci se vytvoří kolečko se stejnými souřadnicemi, a to se na dalším řádku přidá do kontejneru `picturePanel`. Tímto je vytvořen panel obrázku obsahující mapku, bod označující kliknutí uživatele a spoustu bodů označujících hlavní města jednotlivých zemí. Posledním krokem je vložit panel obrázku a text nápovědy do kontejneru Hrací plocha.

Na závěr je nutné všechny vytvořené části uživatelského rozhraní vložit do posledního kontejneru, který předáme jako parametr zobrazované scéně. Scénu poté stačí pouze vložit do již existující instance třídy `Stage` reprezentující v platformě JavaFX okno. Instance třídy `Stage` byla obdržena při konstrukci instance `GUI`. Kód 5.5 znázorňuje konkrétní řešení řečeného postupu.

```
VBox content;
content = new VBox(COMPONENT_SPACING, menuBar, topPanel, playSurface);

Scene scene = new Scene(content);
window.setScene(scene);
window.setTitle("Hledání měst");
window.show();
```

Kód 5.5, Vytvoření okna aplikace, [autor]

Finální kód 5.5 vytváří kontejner VBox, kterému předá jako parametr velikost mezer mezi komponentami, a vytvořené části uživatelského rozhraní. Dále se vytvoří nová scéna scene obsahující právě vytvořený kontejner jako kořenový prvek uživatelského rozhraní. Zbývá pouze oknu přiřadit zobrazovanou scénu, nastavit její titulek a nakonec pomocí metody `show()` okno zobrazit.

Po dokončení základní obrazovky je potřeba dodělat slíbené samostatné okno zobrazující výsledek hry při zmáčknutí tlačítka „Konec hry“ v menu. Kód zajišťující vytvoření nového okna je analogický k předchozímu kódu 5.5, a je zobrazen v kódu 5.6.

```
endGame.setOnAction((ActionEvent e) -> {
    // Vytvoření oznámení o ukončení hry
    Text message = new Text("Hra ukončena. Výsledné skóre: " + playerScore);
    message.setCache(true);
    message.setFill(Color.RED);
    message.setFont(Font.font(null, FontWeight.BOLD, 30));
    Reflection r = new Reflection();
    r.setFraction(0.5f);
    message.setEffect(r);

    // Vytvoření nového okna
    FlowPane pane = new FlowPane(message);
    pane.setPadding(new Insets(COMPONENT_SPACING));
    Scene endGameScene = new Scene(pane);
    Stage endGameStage = new Stage();
    endGameStage.setTitle("Konec hry");
    endGameStage.setScene(endGameScene);
    endGameStage.show();

    playerScore = 0;
});
```

Kód 5.6, Vytvoření druhého okna aplikace, [autor]

6 Ukázková aplikace nových JavaFX 8 komponent

Cílem této kapitoly je vytvořit aplikaci schopnou běžet ve všech dostupných módech spuštění a obsahující nové grafické prvky platformy. Kapitola popisuje postup vytvoření aplikace, použité techniky a podrobně vysvětluje použití nových grafických prvků platformy. Zdrojové kódy celé aplikace jsou uvedeny v příloze. Aplikace je vytvářena jako JavaFX FXML Application. Kompletní kód aplikace se nachází v příloze 2.

6.1 Zadání aplikace

Mezi základní požadavky na vytvářenou ukázkovou aplikaci patří především implementace nových grafických prvků vyskytujících se ve verzi 8 platformy JavaFX. Dále by měla aplikace splňovat základní požadavky pro spuštění ve všech třech dostupných módech.

6.2 Architektura aplikace

Aplikace je vytvářena ve vývojovém prostředí NetBeans a pomocí nástroje Scene Builder. Projekt je založen jako nová JavaFX FXML Application a pojmenován UkazkovaObrazovka. Projekt se skládá z balíčku `soucasti` a `ukazkovaobrazovka`. V balíčku `ukazkovaobrazovka` je umístěna základní třída aplikace, FXML dokument a jeho Controller soubor, a pomocná třída `Osoba`, určená pouze pro demonstraci grafické komponenty `TreeTableView`. V balíčku `soucasti` je obsažen obrázek plochy a CSS soubor popisující vzhledy tlačítek.

6.3 Implementace aplikace

Nejdříve byl vytvořen vzhled aplikace pomocí nástroje Scene Builder. Výsledný FXML dokument má jako kořenový element kontejner `AnchorPane`. Tento kontejner obsahuje pouze jediného potomka, kontejner `TabPane`, který umožňuje rozdělit obrazovku na něko-

lik záložek, z nichž kompletně viditelná pouze jedna. Kontejner TabPane obsahuje celkem tři záložky s názvy Bindings, CSS Styly a JavaFX8. V záložce Bindings je umístěn obrázek představující pozadí záložky a posuvník, určený pro ovládání průhlednosti obrázku. V záložce CSS Styly je umístěno celkem 6 tlačítek s různými atributy `id`, která jsou využita pro demonstraci aplikování různých stylů zobrazení pomocí CSS ouboru. V poslední záložce nesoucí název JavaFX8 jsou umístěny nové grafické komponenty 8. verze platformy.

Následujícím krokem je zajistit funkčnost uživatelského prostředí doplněním Controller souboru. V metodě initialize se v prvních řádcích nastavuje rozsah a hodnota posuvníku které následuje řádek kódu zajišťující propojení posuvníku s vlastností obrázku definující jeho průhlednost. Následuje vytvoření Swingového tlačítka, který se vloží do vytvořeného Swingového panelu. Panel se následně vloží do komponenty SwingNode pomocí řádku `node.setContent(panel);`. Posledním řádkem této metody je řádek volající privátní pomocnou metodu `naplnitTabulku()` provádějící vytvoření struktury a doplnění dat do `TreeTableView` komponenty.

Metoda `naplnitTabulku()` využívá pomocnou třídu `Osoba`, jejíž instance vkládá do komponenty. Metoda nejprve definuje textové řetězce označující skupiny osob, a následně definuje sloupce tabulky. Každému sloupci je zároveň nastaveno jaké atributy třídy `Osoba` se do něj budou ukládat. Nastavení probíhá pomocí metody `setCellValueFactory`, která přebírá jako parametr instanci statické třídy `CellDataFeatures`. Instance `CellDataFeatures` je po vytvoření neměnná a vrací definovanou Read-only Vlastnost atributu Osoby. Dále přichází na řadu vytvoření položek tabulky. Jelikož se v této komponentě dají vkládat položky do sebe, vytvoříme si 3 skupiny položek, do kterých pomocí filtrů vložíme instance `Osoby`. Jediným omezením této komponenty je, že může mít pouze jedinou kořenovou položku. Vytvoříme tedy prázdnou položku, do které vložíme již vytvořené a naplněné skupiny Osob a tuto prázdnou položku vložíme do komponenty. Následně máme možnost pomocí metody `setShowRoot()` skrýt prázdnou kořenovou položku, a pomocí metody `setTableMenuButtonVisible()` zobrazit tlačítko s výběrem viditelných sloupců.

Hlavní třída aplikace má na starosti spuštění aplikace, což v tomto případě znamená načtení FXML dokumentu, který je vložen do základní scény. Této scéně se také následně přiřadí CSS soubor, který definuje vzhledy tlačítek s určitým `id`. Díky tomuto řádku se tlačítka

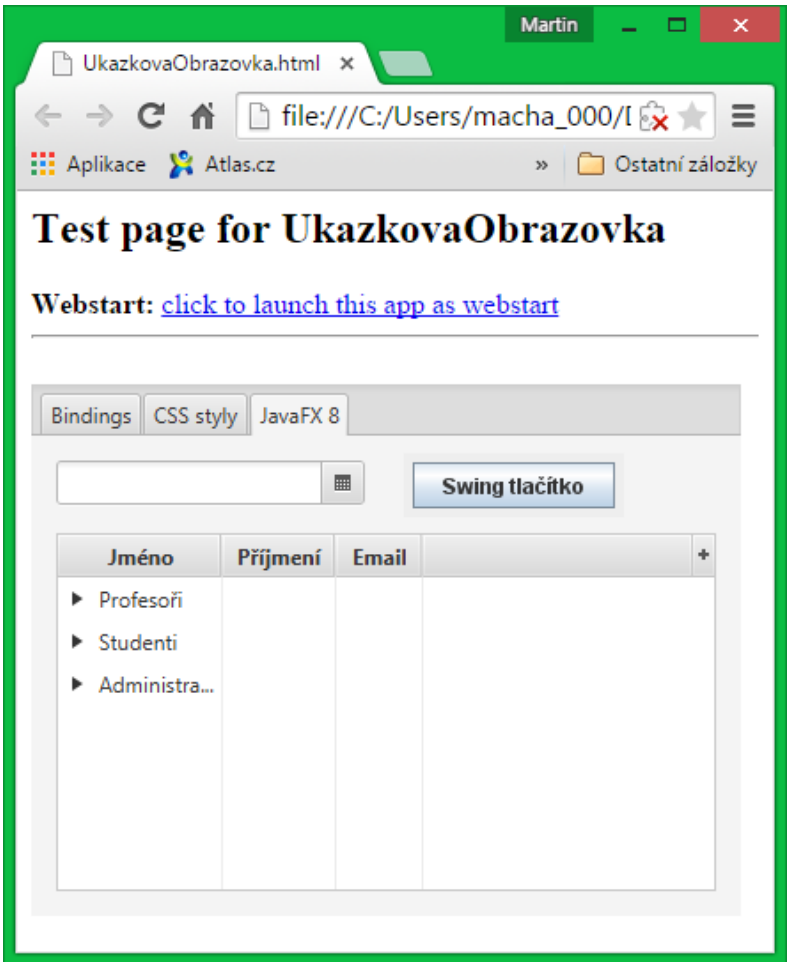
s definovaným `id` v FXML dokumentu budou automaticky zobrazovat podle definice v přiřazeném CSS souboru. Následně se scéna přidá do základního okna aplikace, kterému se nastaví titulek a zobrazí se.

6.4 Spuštění aplikace

Aplikace je nastavená tak, aby byla spustitelná ve všech módech spuštění, které platforma poskytuje. Provedené změny ve vlastnostech projektu jsou následující:

1. V kategorii `Build` v položce `Packaging` byl přidán atribut Application-name do dokumentu Manifest
2. Tamtéž v položce Deployment byla zaškrtnuta možnost Request unrestricted access a Enable BLOB signing, a jako výchozí certifikát byla vybrána možnost Self-signed

Jelikož není aplikace podepsána platným certifikátem důvěryhodné certifikační autority, objeví se při spuštění varování o možné závadnosti softwaru. Pokud povolíme spuštění, bude bez problémů běžet i v prohlížeči, jak je názorně předvedeno na obrázku 6.1.



Obrázek 6.1, Spuštění aplikace ve webovém prohlížeči, [autor]

Terminologický slovník

Termín	Zkratka	Význam[zdroj]
Rich Internet Application	RIA	Webová aplikace, obsahující znaky desktopové aplikace, kterou je možné spouštět formou plug-inů, přes JavaScript nebo pomocí JVM. [vlastní definice autora]
Form Follows Function	F3	Původní jazyk pro vytváření uživatelských rozhraní, který byl následně přejmenován na JavaFX Script [6]
JavaFX Markup Language	FXML	Jazyk na bázi XML, který poskytuje strukturu pro vytváření uživatelských rozhraní odděleně od aplikační logiky [11]
Cascade Style Sheet	CSS	Jazyk umožňující definování designu webových stránek odděleně od jejich obsahu [17]
Extensible Markup Language	XML	Značkovací jazyk určený pro popis dat v libovolné, uživatelem definované struktuře [vlastní definice autora]
plugin		Softwarová komponenta přidávající funkcionalitu k již existujícím aplikacím [vlastní definice autora]
NetBeans		Oficiální vývojové prostředí pro platformu Java [vlastní definice autora]
SceneBuilder		Vizuální nástroj umožňující rychlé a efektivní vytváření uživatelských rozhraní pro JavaFX aplikace. [12]
Java Virtual Machine	JVM	Virtuální stroj, sloužící pro běh aplikací využívajících platformu Java [vlastní definice autora]
JavaFX Application Thread	FXT	Hlavní vlákno platformy výhradně určené pro práci s grafickými prvky aplikace [vlastní definice autora]
Application Programming Interface	API	Soubor knihoven, nástrojů a postupů pro vytváření softwarových aplikací [vlastní definice autora]

Open Graphic Library	OpenGL	API pro renderování 2D a 3D vektorové grafiky [vlastní definice autora]
Direct3D	D3D	API pro renderování 3D grafiky pro Microsoft Windows [vlastní definice autora]
HyperText Markup Language	HTML	Značkovací jazyk pro vytváření webových stránek [vlastní definice autora]
Document Object Model	DOM	Konvence pro znázorňování a práci s elementy značkovacích jazyků [vlastní definice autora]
Scalable Vector Graphics	SVG	Formát vektorového 2D obrázku s podporou interaktivity a animace [vlastní definice autora]
Integrated Development Environment	IDE	Aplikace umožňující vývoj softwarových aplikací, obsahující minimálně editor zdrojového kódu [vlastní definice autora]
Netscape Plugin Application Programming Interface	NPAPI	Multiplatformní plug-inová architektura [23]

Seznam literatury

[1] WEAVER, James L. *Pro JavaFX 2*: *a definitive guide to rich clients with java technology*. Lexington: Apress, 2012, xx, 620 s. The expert´s voice in Java. ISBN 978-1-4302-6872-7.

[2] *Javafx 8: introduction by example. 2ND ED*. Berkeley: Apress, 2014. ISBN 978-143-0264-606.

[3] *Pro javafx: a definitive guide to building rich java clients*. Berkeley: Apress, 2014. ISBN 978-1-4302-6460-6.

[4] ČAPEK, Ondřej. *JavaFX2 a Swing*. Praha, 2012. Bakalářská práce. Vysoká škola ekonomická. Vedoucí práce Ing. Luboš Pavlíček.

[5] JavaFX Overview. *Docs.oracle.com* [online]. 2014 [cit. 2014-12-10]. Dostupné z: http://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-overview.htm#JFXST784

[6] A Brief History of JavaFX. In: *What-when-how.com* [online]. 2012 [cit. 2015-01-10]. Dostupné z: http://what-when-how.com/javafx-2/a-brief-history-of-javafx-getting-a-jump-start-in-javafx/

[7] JavaFX. In: *Wikipedia.org* [online]. 2013, 8.1.2015 [cit. 2015-01-15]. Dostupné z: http://en.wikipedia.org/wiki/JavaFX

[8] Modena theme (update). In: *Fxexperience.com* [online]. 2013 [cit. 2015-03-06]. Dostupné z: http://fxexperience.com/2013/03/modena-theme-update/

[9] Understanding the JavaFX Architecture. In: *Docs.oracle.com* [online]. 2014 [cit. 2015-03-08]. Dostupné z: http://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-architecture.htm#JFXST788

[10] Incorporating Media Assets Into JavaFX Applications. *Docs.oracle.com* [online]. 2012, 2013 [cit. 2015-03-19]. Dostupné z: http://docs.oracle.com/javafx/2/media/simpleplayer.htm

[11] Mastering FXML. *Docs.oracle.com* [online]. 2014 [cit. 2015-03-19]. Dostupné z: http://docs.oracle.com/javafx/2/fxml_get_started/why_use_fxml.htm#CHDCHIBE

[12] JavaFX Scene Builder. *Oracle.com* [online]. 2014 [cit. 2015-03-19]. Dostupné z: http://www.oracle.com/technetwork/java/javase/downloads/javafxscenebuilder-info-2157684.html

[13] JavaFX Scene Builder 2.0 Release Notes. *Docs.oracle.com* [online]. 2014 [cit. 2015-03-19]. Dostupné z: http://docs.oracle.com/javase/8/scene-builder-2/release-notes/jfxsb-release_notes_2-0.htm#CHDFCHEJ

[14] Starting Up Scene Builder. *Docs.oracle.com* [online]. 2014 [cit. 2015-03-19]. Dostupné z: https://docs.oracle.com/javase/8/scene-builder-2/user-guide/startup-window.htm#JSBRG102

[15] Using Scene Builder with NetBeans IDE. In: *Docs.oracle.com* [online]. 2014 [cit. 2015-03-21]. Dostupné z: http://docs.oracle.com/javase/8/scene-builder-2/work-with-java-ides/sb-with-nb.htm#JSBID108

[16] Using JavaFX Properties and Binding. In: *Docs.oracle.com* [online]. 2014 [cit. 2015-04-04]. Dostupné z: http://docs.oracle.com/javase/8/javafx/properties-binding-tutorial/binding.htm

[17] What is CSS?. In: *Webdesign.about.com* [online]. 2015 [cit. 2015-04-06]. Dostupné z: http://webdesign.about.com/od/beginningcss/a/aa021607.htm

[18] JavaFX CSS Reference Guide. In: *Docs.oracle.com* [online]. 2014 [cit. 2015-04-06]. Dostupné z: http://docs.oracle.com/javase/8/javafx/api/javafx/scene/doc-files/cssref.html

[19] Deploying Your First JavaFX Application. In: *Docs.oracle.com* [online]. 2013 [cit. 2015-04-06]. Dostupné z: http://docs.oracle.com/javafx/2/get_started/basic_deployment.htm

[20] .JNLP File Extension. In: *Fileinfo.com* [online]. 2013 [cit. 2015-04-12]. Dostupné z: http://fileinfo.com/extension/jnlp

[21] Self-Contained Application Packaging. In: *Docs.oracle.com* [online]. 2015 [cit. 2015-04-19]. Dostupné z: http://docs.oracle.com/javase/8/docs/technotes/guides/deploy/self-contained-packaging.html

[22] Styling FX Buttons with CSS. In: *Fxexperience.com* [online]. 2011 [cit. 2015-04-19]. Dostupné z: http://fxexperience.com/2011/12/styling-fx-buttons-with-css/

[23] How do I use Java with the Google Chrome browser?. In: *Java.com* [online]. 2014 [cit. 2015-04-20]. Dostupné z: https://java.com/en/download/faq/chrome.xml

Seznam obrázků a tabulek

Základní text

Seznam obrázků

Seznam kódů

Seznam tabulek

Přílohy

Seznam Kódů

Kód přílohy 1.1, HledaniMest, [autor]

```
package hledanimest;

import gui.GUI;
import javafx.application.Application;
import javafx.stage.Stage;
```

```
import logika.Logika;

/**
 * Třída HledaniMest zajišťuje start aplikace
 *
 * @author Martin Macháček
 */
public class HledaniMest extends Application {


    @Override
    public void start(Stage primaryStage) {
        Logika logic = new Logika();
        GUI gui = new GUI(logic, primaryStage);
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        launch(args);
    }
}
```

Kód přílohy 1.2, Logika, [autor]

```
package logika;

import java.awt.Point;
import java.util.ArrayList;
import java.util.Collection;
import java.util.HashMap;
import java.util.Map;

/**
 * Třída Logika zajišťuje logickou a datovou stránku aplikace.
 * Dodává uživatelskému rozhraní hádaná města a vyhodnocuje pokusy uživatele
 *
 * @author Martin Macháček
 */
public class Logika {
    private final Map<String, Point> cities;
    private String findCity = "";
    private final int TOLERANCE = 7;

    /**
     * Konstruktor třídy
```

```
     */
    public Logika() {
        cities = new HashMap<>();
        createCityDatabase();
    }

    /**
     * Metoda vybere náhodné město z mapy a vrátí String řetězec označující
     * název tohoto města.
     *
     * @return String název města
     */
    public String getRandomCity(){
        ArrayList<String> cityNames = new ArrayList<>();
        cities.keySet().stream().forEach((city) -> {
            cityNames.add(city);
        });
        int randomNumber = (int)(Math.random()*cityNames.size());
        while (cityNames.get(randomNumber).equals(findCity)){ // pokud je stejné
město 2x
            randomNumber = (int)(Math.random()*cityNames.size()); // hledá se
nové město
        }
        findCity = cityNames.get(randomNumber);
        return findCity;
    }

    /**
     * Metoda vyhodnocuje zda-li předané parametry pozice X a Y jsou v
akceptovatelné
     * blízkosti hlednaného bodu města.
     *
     * @param x hádaná pozice x
     * @param y hádaná pozice y
     * @return boolean True pokud bylo město uhádnuto
     */
    public boolean isGuessCorrect(double x, double y){
        if (findCity.equals("")){return false;} // kvůli nullpointerexc před
novou hrou
        double posX = cities.get(findCity).getX();
        double posY = cities.get(findCity).getY();
        if ((posX-TOLERANCE) < x && x < (posX+TOLERANCE) && // klik v toleranci
5 bodů
                (posY-TOLERANCE) < y && y < (posY+TOLERANCE)){
            return true;
        }
        return false;
    }
```

```
    /**
     * Metoda vyhodnocuje předané parametry pozice X a Y a oproti pozici
hledaného
     * bodu a vrací String řetězec napovídající pozici hledaného bodu
     *
     * Příklad: Hledaný bod na pozici (10,10), zadané souřadnice (8,15).
     *          Jelikož je X souřadnice bodu = 10 a zadaná souřadnice X je 8,
     *          musíme se na ose x posunout o 2 body napravo.
     *          Metoda v tomto případě vrátí výsledný String řetězec "napravo
nahoru".
     *
     * @param x hádaná pozice x
     * @param y hádaná pozice y
     * @return String řetězec napovídající pozici hledaného bodu
     */
    public String getHint(double x, double y){
        String hint = "";
        if (x < cities.get(findCity).x){ // kliknul nalevo od hledaného města
            hint += "napravo";
        }
        else{
            hint += "nalevo";
        }

        if (y < cities.get(findCity).y){ // kliknul nad hledané město
            hint += " dolů";
        }
        else{
            hint += " nahoru";
        }
        return hint;
    }

    /**
     * Metoda vrací seznam všech bodů, která označují města
     * @return kolekce instancí třídy Point
     */
    public Collection<Point> getCityPoints(){
        return cities.values();
    }

    /**
     * Soukromá pomocná metoda.
     * Metoda vkládá do mapy jednotlivá města a souřadnice jejich bodů
     */
    private void createCityDatabase(){ // 23 hlavních měst Evropy
        cities.put("Praha", new Point(228, 230));
```

```
        cities.put("Bratislava", new Point(253, 246));
        cities.put("Řím", new Point(212, 335));
        cities.put("Berlín", new Point(219, 195));
        cities.put("Waršava", new Point(277, 197));
        cities.put("Vídeň", new Point(245, 250));
        cities.put("Minsk", new Point(326, 170));
        cities.put("Moskva", new Point(389, 136));
        cities.put("Helsinki", new Point(288, 101));
        cities.put("Stockholm", new Point(245, 119));
        cities.put("Kiev", new Point(352, 207));
        cities.put("Bukurešť", new Point(336, 288));
        cities.put("Ankara", new Point(402, 328));
        cities.put("Sofia", new Point(314, 318));
        cities.put("Zagreb", new Point(244, 281));
        cities.put("Athény", new Point(319, 373));
        cities.put("Bern", new Point(168, 266));
        cities.put("Londýn", new Point(114, 204));
        cities.put("Paříž", new Point(125, 241));
        cities.put("Madrid", new Point(63, 336));
        cities.put("Lisabon", new Point(11, 341));
        cities.put("Dublin", new Point(77, 176));
        cities.put("Oslo", new Point(205, 109));
    }
}
```

Kód přílohy 1.3, GUI, [autor]

```
package gui;

import javafx.event.ActionEvent;
import javafx.geometry.HPos;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Menu;
import javafx.scene.control.MenuBar;
import javafx.scene.control.MenuItem;
import javafx.scene.control.SeparatorMenuItem;
import javafx.scene.effect.Reflection;
import javafx.scene.image.ImageView;
import javafx.scene.input.MouseEvent;
import javafx.scene.layout.ColumnConstraints;
import javafx.scene.layout.FlowPane;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.Pane;
import javafx.scene.layout.Priority;
import javafx.scene.layout.VBox;
```

```
import javafx.scene.paint.Color;
import javafx.scene.paint.Paint;
import javafx.scene.shape.Circle;
import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;
import javafx.scene.text.Text;
import javafx.stage.Stage;
import logika.Logika;

/**
 * Třída GUI se stará o vytvoření a udržování grafického uživatelského rozhraní
 *
 * @author Martin Macháček
 */
public class GUI {
    // Deklarace proměnných
    private final String CORRECT = "Správně, uhodl jsi!";
    private final String FIND = "Hledejte na mapě: ";
    private final String HINT_TITLE = "Můžete si zapnout nápovědu v Menu";
    private final String HINT = "Vaše hledáné město je kousek ";
    private final String SCORE = "Skóre: ";
    private int playerScore;
    private boolean hintEnabled;
    private MenuBar menuBar;
    private final int COMPONENT_SPACING = 3;
    private Circle guessedPosition;

    private final VBox content;
    private final GridPane topPanel;
    private final VBox playSurface;
    private Text task;
    private Text score;
    private final Button nextCity;
    private Pane picturePanel;
    private final ImageView picture;
    private Text hintText;
    private Menu menu;
    private MenuItem newGame;
    private MenuItem enableHints;
    private MenuItem endGame;

    private final Logika logic;
    private final Stage window;

    /**
     * Konstruktor třídy
     *
     * Konstruktor vytváří celé uživatelské rozhraní
```

```
     *
     * @param logic
     * @param window
     */
    public GUI(Logika logic, Stage window) {
        // Definice proměnných
        this.logic = logic;
        this.window = window;
        task = new Text("Hledejte na mapě, začněte tlačítkem Nová hra v Menu");
        score = new Text(SCORE + "0");
        nextCity = new Button("Další město");
        picture = new ImageView("/gui/Evropa.JPEG");
        guessedPosition = new Circle(3, Paint.valueOf(Color.GREEN.toString()));
        hintText = new Text(HINT_TITLE);

        // Nastavení proměnných
        task.setFont(new Font("Arial", 15));
        score.setFont(new Font("Arial", 15));

        nextCity.setDefaultButton(true);
        nextCity.setDisable(true);
        nextCity.setOnAction((ActionEvent e) -> {
            if (task.getText().equals(CORRECT)){
                playerScore++;
            }
            else{playerScore--;}
            score.setText("Skóre: " + playerScore + " ");
            task.setText(FIND + logic.getRandomCity());
        });

        picture.setOnMouseClicked((MouseEvent e) -> {
            guessedPosition.setCenterX(e.getX());
            guessedPosition.setCenterY(e.getY());

            if (logic.isGuessCorrect(e.getX(), e.getY())){
                task.setText(CORRECT);
            }
            else{ // zobrazení nápovědy
                if(hintEnabled && !task.getText().equals(CORRECT)){
                    hintText.setText(HINT + logic.getHint(e.getX(), e.getY()));
                }
                else{
                    if (!task.getText().equals(CORRECT)){
                        hintText.setText(HINT_TITLE);
                    }
                }
            }
        });
```

```
        // vytvoření lišty menu
        initMenuBar();

        // Vytvoření horní části obrazovky
        topPanel = new GridPane();

        ColumnConstraints taskColumn = new ColumnConstraints();
        taskColumn.setHalignment(HPos.LEFT);
        taskColumn.setHgrow(Priority.ALWAYS);
        topPanel.getColumnConstraints().add(taskColumn);

        ColumnConstraints scoreColumn = new ColumnConstraints();
        scoreColumn.setHalignment(HPos.RIGHT);
        topPanel.getColumnConstraints().add(scoreColumn);

        ColumnConstraints buttonColumn = new ColumnConstraints();
        buttonColumn.setHalignment(HPos.RIGHT);
        topPanel.getColumnConstraints().add(buttonColumn);

        topPanel.addRow(0, task, score, nextCity);

        // Vytvoření hrací plochy
        picturePanel = new Pane(picture, guessedPosition);

        logic.getCityPoints().stream().forEach(p -> { // vyznačení bodů měst do
mapy
            Circle circle = new Circle(p.x, p.y, 2);
            picturePanel.getChildren().add(circle);
        });

        playSurface = new VBox(picturePanel, hintText);

        // Vytvoření hlavního okna aplikace
        content = new VBox(COMPONENT_SPACING, menuBar, topPanel, playSurface);

        Scene scene = new Scene(content);
        window.setScene(scene);
        window.setTitle("Hledání měst");
        window.show();
    }

    /**
     * Metoda kompletně vytvoří lištu menu a dodá její funkcionalitu
     */
    private void initMenuBar(){
        //Vytvoření a zpřístupnění položek menu
        newGame = new MenuItem("Nová Hra");
```

```
        enableHints = new MenuItem("Nápověda");
        endGame = new MenuItem("Konec Hry");
        setEnableGameButtons(false);

        // Definice handlerů událostí
        newGame.setOnAction((ActionEvent e) -> {
            task.setText(FIND + logic.getRandomCity());
            setEnableGameButtons(true);
            playerScore = 0;
            score.setText("Skóre: " + playerScore + " ");
        });

        enableHints.setOnAction((ActionEvent e) -> {
            if (hintEnabled){
                hintEnabled = false;
                hintText.setText(HINT_TITLE);
            }
            else{
                hintEnabled = true;
                hintText.setText("Nápověda zapnuta");
            }
        });

        endGame.setOnAction((ActionEvent e) -> {
            task.setText("Hra ukončena");
            nextCity.setDisable(true);
            setEnableGameButtons(false);

            // Vytvoření oznámení o ukončení hry
            Text message = new Text("Hra ukončena. Výsledné skóre: " +
playerScore);
            message.setCache(true);
            message.setFill(Color.RED);
            message.setFont(Font.font(null, FontWeight.BOLD, 30));
            Reflection r = new Reflection();
            r.setFraction(0.5f);
            message.setEffect(r);

            // Vytvoření nového okna
            FlowPane pane = new FlowPane(message);
            pane.setPadding(new Insets(COMPONENT_SPACING));
            Scene endGameScene = new Scene(pane);
            Stage endGameStage = new Stage();
            endGameStage.setTitle("Konec hry");
            endGameStage.setScene(endGameScene);
            endGameStage.show();

            playerScore = 0;
```

```
        });

        // vložení položek menu do rozbalovacího seznamu
        menu = new Menu("Menu");
        menu.getItems().addAll(newGame, new SeparatorMenuItem(),
                enableHints, endGame);

        // vytvoření lišty menu a přidání rozbalovacího seznamu
        menuBar = new MenuBar();
        menuBar.getMenus().add(menu);
    }

    /**
     * Metoda zpřístupní tlačítka "Další město" a tlačítka menu dle předaného
prarametru
     * označující stav hry.
     *
     * Předaná hodnota True způsobí znepřístpnění tlačítka "Nová hra" a
zpřístupní
     * všechna ostatní tlačítka.
     *
     * @param isNewGame True, pokud hrajeme hru; False pokud je hra ukončena
     */
    private void setEnableGameButtons(boolean isNewGame){
        endGame.setDisable(!isNewGame);
        enableHints.setDisable(!isNewGame);
        newGame.setDisable(isNewGame);
        nextCity.setDisable(!isNewGame);
    }
}
```

Kód přílohy 2.1, FXML Dokument, [autor]

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

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

<AnchorPane id="AnchorPane" prefHeight="300.0" prefWidth="400.0"
xmlns="http://javafx.com/javafx/8" xmlns:fx="http://javafx.com/fxml/1"
fx:controller="ukazkovaobrazovka.FXMLDocumentController">
```

```
   <children>
      <TabPane prefHeight="200.0" prefWidth="200.0"
tabClosingPolicy="UNAVAILABLE" AnchorPane.bottomAnchor="0.0"
AnchorPane.leftAnchor="0.0" AnchorPane.rightAnchor="0.0"
AnchorPane.topAnchor="0.0">
        <tabs>
          <Tab text="Bindings">
            <content>
              <AnchorPane minHeight="0.0" minWidth="0.0" prefHeight="180.0"
prefWidth="200.0">
                     <children>
                        <ImageView fx:id="pozadi" fitHeight="271.0"
fitWidth="400.0" AnchorPane.bottomAnchor="0.0" AnchorPane.leftAnchor="0.0"
AnchorPane.rightAnchor="0.0" AnchorPane.topAnchor="0.0">
                           <image>
                              <Image url="/soucasti/ls_background.jpg" />
                           </image>
                        </ImageView>
                        <Slider fx:id="posuvnikOpacity" layoutX="131.0"
layoutY="55.0" AnchorPane.leftAnchor="131.0" AnchorPane.topAnchor="55.0" />
                        <Label layoutX="152.0" layoutY="22.0" text="Image
Opacity" textFill="#9a9898">
                           <font>
                              <Font size="15.0" />
                           </font>
                        </Label>
                     </children>
                  </AnchorPane>
            </content>
          </Tab>
            <Tab text="CSS styly">
               <content>
                  <VBox alignment="CENTER" prefHeight="200.0" prefWidth="100.0"
spacing="10.0">
                     <children>
                        <Button mnemonicParsing="false" text="Základní" />
                        <Button id="green" mnemonicParsing="false" text="Green"
/>
                        <Button id="round-red" mnemonicParsing="false"
text="Round Red" />
                        <Button id="shiny-orange" mnemonicParsing="false"
text="Shiny Orange" />
                        <Button id="dark-blue" mnemonicParsing="false"
text="Dark Blue" />
                        <Button id="record-sales" mnemonicParsing="false"
text="Record Sales" />
                     </children>
                     <padding>
```

```
                        <Insets top="10.0" />
                     </padding>
                  </VBox>
               </content>
            </Tab>
            <Tab text="JavaFX 8">
              <content>
                <AnchorPane minHeight="0.0" minWidth="0.0" prefHeight="180.0"
prefWidth="200.0">
                     <children>
                        <DatePicker layoutX="14.0" layoutY="14.0"/>
                        <TreeTableView fx:id="tabulka" layoutX="14.0"
layoutY="57.0" prefHeight="200.0" prefWidth="375.0"
AnchorPane.bottomAnchor="14.0" AnchorPane.leftAnchor="14.0"
AnchorPane.rightAnchor="14.0" AnchorPane.topAnchor="55.0">
                           <columns>
                           </columns>
                        </TreeTableView>
                        <SwingNode fx:id="node" layoutX="277.0" layoutY="27.0"
AnchorPane.leftAnchor="210.0" AnchorPane.topAnchor="10.0" />
                     </children>
                  </AnchorPane>
              </content>
            </Tab>
        </tabs>
      </TabPane>
   </children>
</AnchorPane>
```

Kód přílohy 2.2, FXMLDocumentController, [autor]

```
package ukazkovaobrazovka;

import java.net.URL;
import java.util.Arrays;
import java.util.List;
import java.util.ResourceBundle;
import java.util.function.Predicate;
import javafx.beans.property.ReadOnlyStringWrapper;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.control.SelectionMode;
import javafx.scene.control.Slider;
import javafx.scene.control.TreeItem;
import javafx.scene.control.TreeTableColumn;
import javafx.scene.control.TreeTableView;
import javafx.scene.image.ImageView;

/**
 *
 * @author Martin Macháček
 */
public class FXMLDocumentController implements Initializable {

    @FXML
    private Slider posuvnikOpacity;
    @FXML
    private ImageView pozadi;

    @FXML
    private TreeTableView<Osoba> tabulka;

    @Override
    public void initialize(URL url, ResourceBundle rb) {
        // Záložka Bindings
        posuvnikOpacity.setMin(0);
        posuvnikOpacity.setMax(1);
        posuvnikOpacity.setValue(1);
        pozadi.opacityProperty().bind(posuvnikOpacity.valueProperty());

        // Záložka JavaFX8
        JPanel panel = new JPanel();
        JButton tlacitko = new JButton("Swing tlačítko");
        panel.add(tlacitko);

        node.setContent(panel);
        naplnitTabulku();
    }
```

```
    /**
     * Metoda vytvoří sloupce a doplní předpřipravená data do tabulky
     */
    private void naplnitTabulku() {
        final String PROFESORI = "Profesoři";
        final String STUDENTI = "Studenti";
        final String ADMINISTRATIVA = "Administrativa";

        // Vytvoření sloupců
        // sloupec Jméno
        TreeTableColumn<Osoba, String> sloupJmeno =
            new TreeTableColumn<>("Jméno");
        sloupJmeno.setCellValueFactory(
            (TreeTableColumn.CellDataFeatures<Osoba, String> param) ->
            new ReadOnlyStringWrapper(param.getValue().getValue().getJmeno())
        );
        // sloupec Příjmení
        TreeTableColumn<Osoba, String> sloupPrijmeni =
            new TreeTableColumn<>("Příjmení");
        sloupPrijmeni.setCellValueFactory(
            (TreeTableColumn.CellDataFeatures<Osoba, String> param) ->
            new ReadOnlyStringWrapper(param.getValue().getValue().getPrijmeni())
        );
        // sloupec Email
        TreeTableColumn<Osoba, String> sloupEmail =
            new TreeTableColumn<>("Email");
        sloupEmail.setCellValueFactory(
            (TreeTableColumn.CellDataFeatures<Osoba, String> param) ->
            new ReadOnlyStringWrapper(param.getValue().getValue().getEmail())
        );

        // vložíme sloupce do tabulky
        tabulka.getColumns().setAll(sloupJmeno, sloupPrijmeni, sloupEmail);

        // Vytvoření skupin osob
        final TreeItem<Osoba> skupProf = new TreeItem<>(new Osoba(PROFESORI,
null, null,null));
        final TreeItem<Osoba> skupStud = new TreeItem<>(new Osoba(STUDENTI,
null, null,null));
        final TreeItem<Osoba> skupAdm = new TreeItem<>(new Osoba(ADMINISTRATIVA,
null, null,null));

        // vytvoření osob
        List<Osoba> Osoby = Arrays.<Osoba>asList(
        new Osoba("Adam","Aligátor", "adam.aligator@mail.cz",PROFESORI),
                new Osoba("Bob", "Buben", "bob.buben@mail.cz",PROFESORI),
```

```
                new Osoba("Cecil", "Cyklista",
"cecil.cyklista@mail.cz",PROFESORI),
                new Osoba("David", "Dříč", "david.dric@mail.cz",STUDENTI),
                new Osoba("Emil", "Expert", "emil.expert@mail.cz",STUDENTI),
                new Osoba("Franta", "Fištrón",
"franta.fistron@mail.cz",STUDENTI),
                new Osoba("Gábina", "Galantní",
"gabina.galantni@mail.cz",STUDENTI),
                new Osoba("Honza", "Hbitý", "honza.hbity@mail.cz",STUDENTI),
                new Osoba("Iveta", "Inteligentní",
"iveta.inteligentni@mail.cz",ADMINISTRATIVA),
                new Osoba("Jakub", "Jablečný",
"jakub.jablecny@mail.cz",ADMINISTRATIVA),
                new Osoba("Karel", "Krychle",
"karel.krychle@mail.cz",ADMINISTRATIVA));

        // vytvoření filtrů pro výběr skupin osob
        Predicate<Osoba> profesor = (o) -> (PROFESORI.equals(o.getOddeleni()));
        Predicate<Osoba> student = (o) -> (STUDENTI.equals(o.getOddeleni()));
        Predicate<Osoba> administr = (o) ->
(ADMINISTRATIVA.equals(o.getOddeleni()));

        // Vložení vybraných osob do příslušné skupiny
        Osoby.stream().filter(profesor).forEach((osoba) -> {
            skupProf.getChildren().add(new TreeItem<>(osoba));
        });
        Osoby.stream().filter(student).forEach((osoba) -> {
            skupStud.getChildren().add(new TreeItem<>(osoba));
        });
        Osoby.stream().filter(administr).forEach((osoba) -> {
            skupAdm.getChildren().add(new TreeItem<>(osoba));
        });

        // vložení naplněných skupin do tabulky
        final TreeItem<Osoba> root = new TreeItem<>(new Osoba(null, null,
null,null));
        root.getChildren().addAll(skupProf,skupStud,skupAdm);
        tabulka.setRoot(root);
        tabulka.setShowRoot(false);
        tabulka.setTableMenuButtonVisible(true);
        tabulka.getSelectionModel().setSelectionMode(SelectionMode.SINGLE);
    }
}
```

Kód přílohy 2.3, UkazkovaObrazovka, [autor]

```
/**
 *
 * @author Martin Macháček
 */
public class UkazkovaObrazovka extends Application {

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

        Scene scene = new Scene(root);
        scene.getStylesheets().add("soucasti/CascadeStyleSheet.css");

        stage.setScene(scene);
        stage.setTitle("Ukázka použití komponent");
        stage.show();
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        launch(args);
    }
}
```

Kód přílohy 2.4, CascadeStyleSheet, [22]

```
#green {
    -fx-background-color:
        linear-gradient(#f0ff35, #a9ff00),
        radial-gradient(center 50% -40%, radius 200%, #b8ee36 45%, #80c800 50%);
    -fx-background-radius: 6, 5;
    -fx-background-insets: 0, 1;
    -fx-effect: dropshadow( three-pass-box , rgba(0,0,0,0.4) , 5, 0.0 , 0 , 1 );
    -fx-text-fill: #395306;
}
#round-red {
    -fx-background-color: linear-gradient(#ff5400, #be1d00);
    -fx-background-radius: 30;
    -fx-background-insets: 0;
    -fx-text-fill: white;
}

#shiny-orange {
    -fx-background-color:
        linear-gradient(#ffd65b, #e68400),
        linear-gradient(#ffef84, #f2ba44),
        linear-gradient(#ffea6a, #efaa22),
        linear-gradient(#ffe657 0%, #f8c202 50%, #eea10b 100%),
        linear-gradient(from 0% 0% to 15% 50%, rgba(255,255,255,0.9),
rgba(255,255,255,0));
    -fx-background-radius: 30;
    -fx-background-insets: 0,1,2,3,0;
    -fx-text-fill: #654b00;
    -fx-font-weight: bold;
    -fx-font-size: 14px;
    -fx-padding: 10 20 10 20;
}
#dark-blue {
    -fx-background-color:
        #090a0c,
        linear-gradient(#38424b 0%, #1f2429 20%, #191d22 100%),
        linear-gradient(#20262b, #191d22),
        radial-gradient(center 50% 0%, radius 100%, rgba(114,131,148,0.9),
rgba(255,255,255,0));
    -fx-background-radius: 5,4,3,5;
    -fx-background-insets: 0,1,2,0;
    -fx-text-fill: white;
    -fx-effect: dropshadow( three-pass-box , rgba(0,0,0,0.6) , 5, 0.0 , 0 , 1 );
    -fx-font-family: "Arial";
    -fx-text-fill: linear-gradient(white, #d0d0d0);
    -fx-font-size: 12px;
    -fx-padding: 10 20 10 20;
}
```

```
#dark-blue Text {
    -fx-effect: dropshadow( one-pass-box , rgba(0,0,0,0.9) , 1, 0.0 , 0 , 1 );
}
#record-sales {
    -fx-padding: 8 15 15 15;
    -fx-background-insets: 0,0 0 5 0, 0 0 6 0, 0 0 7 0;
    -fx-background-radius: 8;
    -fx-background-color:
        linear-gradient(from 0% 93% to 0% 100%, #a34313 0%, #903b12 100%),
        #9d4024,
        #d86e3a,
        radial-gradient(center 50% 50%, radius 100%, #d86e3a, #c54e2c);
    -fx-effect: dropshadow( gaussian , rgba(0,0,0,0.75) , 4,0,0,1 );
    -fx-font-weight: bold;
    -fx-font-size: 1.1em;
}
#record-sales:hover {
    -fx-background-color:
        linear-gradient(from 0% 93% to 0% 100%, #a34313 0%, #903b12 100%),
        #9d4024,
        #d86e3a,
        radial-gradient(center 50% 50%, radius 100%, #ea7f4b, #c54e2c);
}
#record-sales:pressed {
    -fx-padding: 10 15 13 15;
    -fx-background-insets: 2 0 0 0,2 0 3 0, 2 0 4 0, 2 0 5 0;
}
#record-sales Text {
    -fx-fill: white;
    -fx-effect: dropshadow( gaussian , #a30000 , 0,0,0,2 );
}
```

Kód přílohy 2.5, Třída Osoba, [autor]

```
/**
 * @author Martin Machacek
 */
public class Osoba {
    private final String jmeno;
    private final String prijmeni;
    private final String email;
    private final String oddeleni;

    public Osoba(String jmeno, String prijmeni, String email, String oddeleni) {
        this.jmeno = jmeno;
        this.prijmeni = prijmeni;
        this.email = email;
```

```
        this.oddeleni = oddeleni;
    }

    public String getJmeno() {
        return jmeno;
    }

    public String getPrijmeni() {
        return prijmeni;
    }

    public String getEmail() {
        return email;
    }

    public String getOddeleni() {
        return oddeleni;
    }
}
```