Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Analýza možností použití nástrojů pro testování JavaFX GUI

Plzeň 2016 Marek Rojík

PROHLÁŠENÍ

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 4. května 2016 . . . . . .

Marek Rojík

ABSTRACT

The goal of this thesis is to analyze usability of tools for testing of JavaFX GUI. The purpose of these tools is to test user interface which is written in JavaFX. According to the results of the analysis the TestFX tool is chosen for testing because the tool is usable and still actively developed. This tool is analyzed in detail and explained. Then an application is created and tested with this tool. On the base of the application tests a tutorial is created that explains how to use the testing tool. The tutorial is tested on the students of bachelor's degree. According to the feedback from the students the TestFX tool can be easily used thanks to the tutorial.

KEYWORDS

testing tools, JavaFX, TestFX, test, tutorial

ABSTRAKT

Cílem práce je analyzovat možnosti použití nástrojů pro testování JavaFX. Jedná se o nástroje, které slouží k testování uživatelského rozhraní, které je psané v jazyce JavaFX. Podle výsledků vyplývající z analýzy je vybrán testovací nástroj TestFX, protože se nachází v použitelné verzi a je stále vyvíjen. Tento nástroj je dále podrobně rozebrán a zhodnocen. Následně je vytvořena aplikace a pomocí tohoto nástroje otestována. Na základě testů aplikace je vytvořen tutoriál, který popisuje použití testovacího nástroje. Tutoriál je otestován na studentech bakalářského studia. Podle zpětné vazby od studentů tutoriál naučí, jak používat testovací nástroj TestFX.

KLÍČOVÁ SLOVA

testovací nástroje, JavaFX, TestFX, test, tutoriál

ROJÍK, Marek *Analýza možností použití nástrojů pro testování JavaFX GUI*: bakalářská práce. Plzeň: Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Katedra informatiky a výpočetní techniky, 2016. 53 s. Vedoucí práce byl Ing. Richard Lipka, Ph.D.

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu bakalářské práce panu Ing. Richardu Lipkovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

OBSAH

1 ÚVOD

Cílem práce je analyzovat možnosti použití nástrojů pro testování JavaFX. Jedná se o nástroje, které slouží k testování uživatelského rozhraní napsané v jazyce JavaFX. Tyto nástroje simulují akce uživatele, který pracuje s aplikací.

V dnešní době se testování podceňuje, i když je to důležitá fáze při vývoji aplikace. Základem je testovat jádro programu, což je relativně jednoduché, protože se nemusí simulovat akce od uživatele. Testování uživatelského rozhraní je v tomto ohledu obtížnější, protože vyžaduje hodně ruční práce. Naštěstí to jde zautomatizovat pomocí unit testů. Proto je potřeba najít nástroje, které pomocí unit testů dokáží otestovat uživatelské rozhraní.

Úvodem je potřeba vysvětlit, jaké typy testování se v dnešní době používají. V první části práce je nutno rozhodnout, který nástroj pro testování uživatelského rozhraní bude nejlepší. Pro toto zjištění je vybráno několik nástrojů, které jsou porovnávány podle různých kritérií. Nejdůležitějším kritériem je, zda se nástroj nachází v použitelné verzi a je stále vyvíjen a podporován. Dalším kritériem je, zda má nástroj pro testování vytvořenou dokumentaci, podle které je možné se naučit používat daný nástroj. Posledním kritériem je, jestli je nástroj dostupný zdarma, nebo se musí pro používání zakoupit licence. Pro srovnání jsou přidány do porovnávání i další nástroje, které neslouží pro testování uživatelských rozhraní platformy JavaFX, ale například Swing, WinForm, WPF a internetových stránek.

V druhé části práce je na základě výše zmíněných kritérií vybrán jeden nástroj pro testování platformy JavaFX a ten je dále podrobně rozebrán a zhodnocen. Tady se již převážně z jeho programové dokumentace. Jsou rozebrány jeho důležité metody pro testování.

Další část práce je zaměřená přímo na testování uživatelského rozhraní pomocí vybraného nástroje. Cílem je vytvoření aplikace, která bude obsahovat komponentu typu tabulka, seznam nebo strom. Dále rozhraní této aplikace je potřeba kompletně pokrýt testy pomocí vybraného testovacího nástroje a přehledně ho zdokumentovat.

V poslední části práce je vytvořen výukový tutoriál pro tvorbu testů pro studenty bakalářského studia, kteří mají o testování softwarové platformy JavaFX zájem. Tutoriál obsahuje popis, jak se nástroj používá. Následně je otestován přímo na studentech bakalářského studia, kteří na základě použitelnosti tutoriálu zhotoví hodnocení. V případě zjištění nedostatků bude tutoriál upraven, aby vyhovoval výuce testování v daném testovacím nástroji.

2 METODY TESTOVÁNÍ

2.1 Manuální a automatické testování

Manuální testování se používá především v případě, kdy není potřeba jednotlivé testy opakovat, což ale obvykle je. Dále se využívá, pokud test požaduje lidské ohodnocení a úsudek. To je potřeba hlavně u testování uživatelského rozhraní, kdy počítač neví, kam přesně kliknout a ověřit, jestli se provedlo to, co mělo. Výhodou manuálního testování je, že tester využívá svého myšlení a dokáže odhalit při testování určité akce i chybu, která se objevila zcela náhodně v jiné části aplikace. Nevýhodou je nutná přítomnost testera. Zde je výběr patnácti nejlepších testovacích nástrojů pro manuální testování: qTest, PactiTest, Zephyr, Test Collab, JIRA, XQual, TestRail, TestLodge, HP ALM/Quality center, QMetry, Testuff, Gemini, TestLink, QAComplete a Silk Central [13].

Naopak automatické testování se provádí pravidelně během vývoje aplikace opakovaně, protože stačí test napsat pouze jednou a poté test spustit, kolikrát je potřeba. Tímto se zvyšuje úspora času a kvalita výsledné aplikace, což je výhoda oproti manuálnímu testování. Nevýhodou je nutná aktualizace všech testů, pokud dojde i k malé změně jádra aplikace [5].

2.2 Statické a dynamické testy

Statické testy nevyžadují spuštění softwaru. Je možno s nimi začít již v raných fázích vývoje. Tyto testy provádí z velké části překladač, který ověřuje dodržení doporučených tzv. Best Practices. *Dynamické* testy probíhají nad spuštěným softwarem. Používají se v dalších fázích vývoje, kdy již existuje alespoň spustitelný prototyp software [6].

2.3 Progresní a regresní testy

Progresní testy probíhají při kontrole nových implementovaných funkcí aplikace. Jednotlivé testy jsou prováděny podle dokumentace. *Regresní* testy se využívají při testování funkcí, které již byly v minulosti otestovány. Smyslem těchto testů je ověřit, že nebyly kvůli implementaci nových funkcí porušeny funkce, které již byly v minulosti úspěšně otestovány [7].

2.4 Základní úrovně testování

Podle toho, v jaké fázi se nachází vývoj produktu, je prováděno určité testování.

Testování programátorem

Ihned po vytvoření zdrojového kódu je tento kód otestován programátorem. Bývá zvykem, že svůj program netestuje sám autor, ale jiný programátor. V praxi se tento stupeň bohužel příliš nepoužívá a je podceňován.

Testování jednotek

Používá se pro testování malých částí kódu, konkrétněji tříd a metod. Testy mají být co nejjednodušší, aby jeden test pokrýval nejlépe jen jednu metodu. Existuje metrika pokrytí kódu testy, která určuje kolik příkazů, hran, podmínek nebo jiných cest je otestováno. Například v případě, že metoda obsahuje podmínky, je potřeba otestovat každou větev podmínky, která může nastat. Tyto testy slouží pro základní testování a ověření, že metody dělají to, co mají. Základ pro testování je knihovna xUnit, od které jsou odvozeny další knihovny pro dané programovací jazyky.

FAT - Funkční testy

Factory acceptance tests (FAT) je fáze testování, kdy testy provádí sám zadavatel. Aplikace nemusí být na plně integrovaném prostředí.

Integrační testování

Integrační testy již neprovádí sám programátor, ale je to úkol pro testovací tým. Pomocí testů se zkoumá, zda je komunikace mezi jednotlivými komponentami uvnitř aplikace bezchybná. Dále může být testována i komunikace mezi komponentou a operačním systémem, hardwarem či rozhraním různých systémů. Integrační testy mohou být manuální nebo automatizované.

Systémové testování

Jedná se o testování aplikace jako funkční celek. Tyto testy jsou používány v pozdějších fázích vývoje. Testování probíhá tak, jako by aplikaci používal zákazník. Provádí se testy různých kroků a stavů aplikace, které mohou při používání nastat. Systémové testování probíhá do té doby, než jsou odstraněny všechny chyby aplikace. Tento typ testů slouží jako výstupní kontrola aplikace.

UAT - Akceptační testování

User acceptance test (UAT) je typ testování, který probíhá již u zákazníka. Testy jsou prováděny podle předem připravených scénářů, které byly připraveny mezi zákazníkem a dodavatelem. Pokud je nalezena nějaká chyba, aplikace je vrácena zpět dodavateli k opravě [8].

2.5 GUI testy

Obecně testy uživatelského rozhraní (Graphical User Interface) se dělí na několik typů. Hlavní typ je otestování, že aplikace funguje podle očekávání. Tedy že všechny algoritmy, práce s daty a různé nastavování funguje jak má. Jedná se spíše o otestování aplikační logiky, kde GUI je používáno spíše jako nástroj pro zadávání vstupních dat a kontrola očekávaných výstupů.

Další typ je zaměřen na samotné GUI. To obecně zobrazuje různé ovládací prvky. Testuje se, zda dané okno obsahuje všechny očekávané formuláře, ovládací prvky a výstupy. U formulářů se testuje, zda obsahují všechna očekávaná pole a že u těchto polí funguje validace (například povinná pole nebo formát).

Součástí testování GUI jsou i testy použitelnosti. Otestování, zda je aplikace uživatelsky přehledná, aby její používání bylo uživatelsky přívětivé. Oproti předchozím typům testů, tyto není možné zautomatizovat. S tím souvisí i otestování kompatibility. Jde o testy, které ověří správné zobrazení a funkčnost i na jiných operačních systémech než pouze na tom, na kterém je aplikace vyvíjená [14].

3 NÁSTROJE PRO TESTOVÁNÍ UŽIVATELSKÝCH ROZHRANÍ

3.1 Uživatelské rozhraní

Uživatelské rozhraní slouží pro ovládání např. strojů, počítačových programů a dalších systémů. GUI umožňuje ovládat počítačové programy pomocí grafických prvků, které reagují na dané události od uživatele. GUI nahrazuje práci s příkazovou řádkou, která může být pro méně zdatného uživatele komplikovanější.

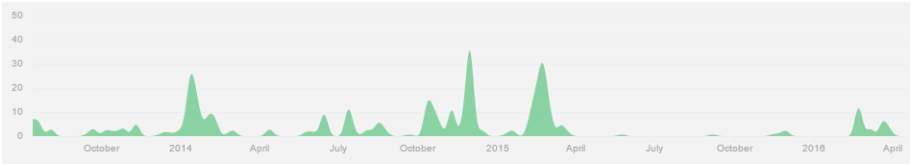
3.2 Nástroje pro testování

Cílem práce je vybrat nejpoužitelnější nástroj pro testování uživatelského rozhraní. Pro porovnání jsou vybrány nástroje TestFX, Automaton, MarvinFX a TestComplete, které dokážou testovat JavaFX. Společně s nimi jsou porovnány frameworky Selenium WebDriver, White Framework a Sikuli. Tyto nástroje slouží pro otestování GUI, které není napsáno v JavaFX nebo testuje nezávisle na typu.

3.2.1 TestFX

Jedná se o jeden z mála frameworků pro JavaFX, který je stále podporován a vyvíjen. Byl vytvořen převážně z důvodu složitého a zbytečně dlouhého kódu při testování frameworkem JemmyFX. Pro tento framework neexistuje žádný nástroj, který by umožňoval testování aplikace i bez znalostí programování (např. nahráváním akcí, které se provádějí na obrazovce a následným spouštěním). Pro používání je nutné mít jazyk Java verze 8, neboť jsou používány právě novinky z této verze. Současně v této verzi Java se objevila i JavaFX. Aplikace se testuje pomocí JUnit testů. Mezi hlavní výhody oproti ostatním nástrojům patří možnost vytvoření automatického screenshotu při neúspěšném testu.

Podle dokumentace lze vyhledávat grafické prvky podle `fx:id`. Jedná se o identifikátor, který se definuje jako atribut v komponentách FXML dokumentu (viz níže v kapitole 4.2.1). Lze vyhledávat prvky na základě textu, který obsahuje, nebo pomocí CSS identifikátoru. Dále je možné použít tzv. `Matcher` metody nebo rozhraní `Predicate`. Pokud je grafické rozhraní definováno v FXML, při hledání komponent v aplikaci se nejvíce používá identifikátor `fx:id`. Další výhodou je možnost čekání na určitou hodnotu prvku. Daný test lze pozastavit nebo nastavit, aby pokračoval, až daná komponenta bude mít určitou hodnotu [16].

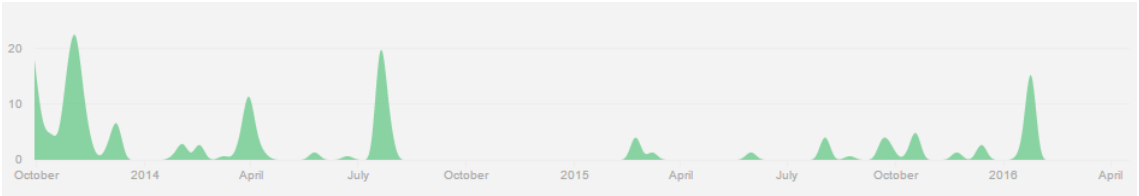


Obr. 3.1: Graf úprav TestFX (14. července 2013 – 17. duben 2016) [15]

3.2.2 Automaton

Je použitelný od verze Java 7 (aktualizace 65) nebo Java 8 (aktualizace 25). Používá se pro testování platforem Swing a JavaFX. Tento framework vyniká v tom, že lze psát testy pomocí Groovy scripts, které mají jednoduchou syntaxi a snadno se čtou (např. `clickOn 'text:Some Button'`). Testy se opět píšou jako klasické JUnit testy, které je možné psát pomocí Javy nebo Groovy frameworku. Pro začátečníky je nabízen demo program, ve kterém jsou demonstrovány možnosti tohoto nástroje. Je zajímavé, že byl vytvořen společností SmartBear, která vytvořila i TestFX.

Podle dokumentace lze vyhledávat komponenty v okně JavaFX na základě jejich ID, dále pomocí CSS identifikátoru jako u TestFX, podle textu, který obsahuje, a také podle typu (např. `TextArea`). V uživatelském rozhraní, vytvořeném ve Swing, se nalézají komponenty podle ID, obsahujícího textu a typu komponenty. Pokud by možnosti výběru komponenty ve Swing nestačily, Automaton nabízí vytvoření vlastního selektoru, pomocí kterého se můžou následně vybírat komponenty v okně. Pokud aplikace obsahuje část ve Swing a část v JavaFX, není problém takové uživatelské rozhraní otestovat. Automaton k tomu nabízí třídu `SwingerFxer`, která u komponent rozezná, zda patří do Swing nebo JavaFX. Testy se dají spouštět buď z příkazové řádky pomocí příkazu `java` nebo pomocí JUnit [2].



Obr. 3.2: Graf úprav Automaton (29. září 2013 – 17. duben 2016) [1]

3.2.3 MarvinFX

Pro tento již mrtvý testovací nástroj neexistuje podrobnější dokumentace, která by ukázala možnosti pro testování GUI. Pro provádění testů se používá opět JUnit. Byl vyvíjen jen jedním člověkem, který ale implementoval pouze pár možností pro testování a pak byl

vývoj ukončen. Tento nástroj je použitelný na testování vlastností komponent platformy JavaFX.

Při vytváření JUnit testu se vždy na prvním řádku vytvoří instance určité komponenty a následně je pomocí metody vložena do JavaFX scény. Po spuštění testu se vykreslí okno s danou komponentou. Dále se vytvoří kolekce `NodeFixture` obsahující typ komponenty, do které se přidá již vytvořená komponenta. Na rozdíl od výše zmíněných nástrojů umí tento jen kliknout levým či pravým tlačítkem na daný prvek. Dále obsahuje několik metod pro testování hodnoty (metody typu `assert`) [3].

3.2.4 TestComplete

TestComplete je z celého výběru testovacích nástrojů ten nejrozšířenější. Umí testovat internetové stránky, počítačové programy a mobilní aplikace nezávisle na platformách a zařízení. Jednotlivé testy je možno tvořit pomocí jazyků Python, VBScript, JScript, DelphiScript, C++Script, a C#. Vývojářům je doporučováno používat program Code Editor. Umožňuje využívat systémů pro verzování, jako jsou Git, Subversion nebo Mercurial. Integrace s nástrojem SoapUI NG Pro umožňuje provádět automatické testy na různé API nebo webové služby. Celý program tvoří několik testovacích nástrojů. Obsahuje nástroje pro nahrávání akcí, datově řízené testování, objektově řízené testování, klávesově řízené testování, testování databáze a mnoho dalších. Integrace s nástroji Selenium, SoapUI, AQtime, HP Quality Center, Visual Studio Team System a další.

Jak již bylo zmíněno, tento nástroj umožňuje nahrávat akce prováděné na obrazovce a následně pomocí toho testovat. V případě potřeby lze manuálně upravit výsledný kód testu, např. doplnit stisknutí tlačítka. Programovací jazyk Python má jednoduchou syntaxi, proto není těžké pomocí něho psát testy. Existuje nástroj QAComplete, který slouží k přehledné organizaci testů, jejich analýze, automatickému a manuálnímu spuštění. Umožňuje porovnávat mezi sebou obrázky (pixel po pixelu), soubory (byte po bytu) nebo vlastnosti objektů. Pro větší přehlednost odděluje testovací logiku od testovacích dat. Testy se dají spouštět vzdáleně z domova např. v laboratoři, virtuálním stroji nebo dokonce na cloudu.

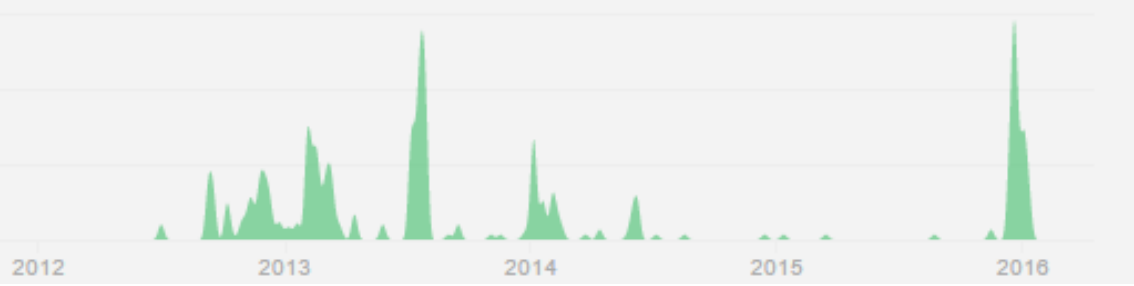
U uživatelských rozhraní určené pro desktopové aplikace je umožněno testovat ty, které jsou naprogramované například v platformě .NET, WPF, C++, Java, JavaFX, Qt nebo dokonce i konzolovou aplikaci. Pro GUI JavaFX je možné tvořit jednotkové, funkcionální nebo regresní testy. Kromě nahrávání testů se dá samotný test i napsat například v jazyce C#, což je doporučováno pokročilejším uživatelům. Z mobilních aplikací lze testovat uživatelské rozhraní Android nebo iOS. Pro testování internetových stránek podporuje prohlížeče Microsoft Edge, Internet Explorer 9 – 11, Google Chrome, Mozilla Firefox a Opera. U webových služeb je to například HTTP nebo HTTPS protokol, WS-I Basic profile 1.1, SOAL 1.1 a 1.2 nebo WSDL 1.1.

Jediný mezi zde porovnávanými nástroji je placený. Licence pro jednoho uživatele stojí €889 [12].

3.2.5 White Framework

Jedná se o testovací framework, který slouží pro testování uživatelských rozhraní platformy Win32, WinForms, WPF, Silverlight a SWT (Java). K psaní testů se používá programovací jazyk C#. K používání tohoto nástroje je třeba mít nainstalovaný .NET 3 framework. White testuje rozhraní odlišným způsobem než výše zmíněné nástroje. Vývojáři doporučují pro práci s tímto frameworkem využívat program VisuaUIAVerify, který slouží především pro zobrazení vlastností různých komponent daného programu (ID, text a další).

Pro psaní testovacích metod lze vybrat z více frameworků pro jednotkové testy (MSTest, NUnit a xUnit). K otestování je potřeba mít nejprve vytvořený spustitelný `exe` soubor, který se načte do programu pro další práci. White framework poté pomocí UI Automation najde veškeré komponenty okna včetně jejich vlastností (například ID, pozici, typ nebo text). K tomuto je ještě využito nástroje UISpy, který umí identifikovat komponenty v daném okně. Prvky lze filtrovat podle kritérií. Danou komponentu lze získat na základě ID nebo typu, který je definovaný pomocí UIAutomation. Dále pomocí textu uvnitř komponenty nebo podle indexu, který znamená pozici v kolekci všech prvků. Jelikož na začátku testování se musí najít všechny komponenty, je možné nastavit timeout, který znamená maximální dobu hledání [18].



Obr. 3.3: Graf úprav White Framework (1. leden 2012 – 17. duben 2016) [17]

3.2.6 Selenium WebDriver

Selenium je zaměřeno pouze na testování internetových stránek. Jedná se o nejznámější použitelný testovací nástroj pro testování internetových stránek. Ve verzi 1.0 bylo možné provádět testování jen pomocí nahrávání prováděných akcí v prohlížeči Mozilla Firefox. Následně byla vydána verze 2.0, která umožňuje provádět testy pomocí programovacích jazyků Java, C#, Python, Ruby a Perl. Po nahrání je lze daný kód lehce modifikovat. Tento

nástroj podporuje testování v různých prohlížečích. Základní prohlížeč je HTML Unit driver, který je nejrychlejší, protože se vytvoří na pozadí prohlížeč a spustí se daný test. Tester nevidí provádění jednotlivých akcí, vidí pouze výsledek testu. Dále podporuje prohlížeče Mozilla Firefox, Google Chrome, Internet Explorer, Opera, Android, iOS a Safari.

Internetové stránky se testují pomocí jednotkových testů napsaných v jazyce Java. Jednotlivé prvky se dají filtrovat pomocí ID, názvu třídy, tagu prvku, názvu atributu, hypertextového linku, CSS identifikátoru, XPath[1] nebo v JavaScriptu pomocí DOM. Mezi metody pro testování patří kliknutí na daný prvek, ze kterého se může přečíst jeho hodnota, nebo zasílání stisknutí na určité klávesy. Určité internetové stránky mají více rámců nebo oken, v testech je možno mezi okny přepínat. Novinkou ve verzi 2.0 je umožnění vytvoření vyskakovacího okna všech typů, které v HTML existují. Podporuje práci s historií, to znamená vracet se na již navštívené stránky. Zajímavé je, že tento testovací nástroj umí přidávat a odebírat cookies. Mezi další zajímavé vlastnosti patří možnost používat v testech akce typu Drag And Drop [10].

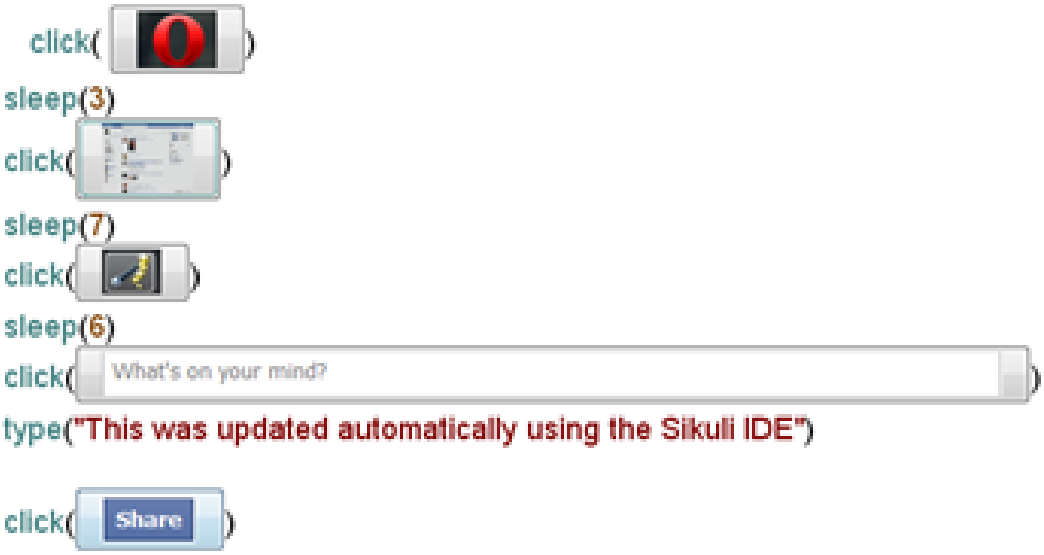
3.2.7 Sikuli

Sikuli se od již zmíněných testovacích nástrojů liší v tom, že může testovat cokoliv, co je zobrazeno na obrazovce. Na rozdíl od jiných nástrojů, které k identifikaci prvků používaly například ID, tento nástroj používá obrázek toho prvku. K identifikaci prvku hledá pozici na obrazovce, která bude obsahovat prvek totožný s prvkem na obrázku, a následně nad nalezenou pozicí provede akci. Je možné testovat internetové stránky, desktopové programy, mobilní aplikace a další pomocí jednoho nástroje. K vytvoření testů se používají skriptovací jazyky Python (verze 2.7), Ruby (verze 1.9 / 2.0) nebo JavaScript.

Vývojáři doporučují používat k testování jejich nástroj SikuliX IDE, který podporuje několik výchozích příkazů pro testování a hlavně vkládání obrázků, které jsou vidět v těle testu. Díky nim může testovat i člověk, který nemá žádné znalosti s programováním, protože jejich použití je jednoduché. Ale pro vytvoření rozsáhlejších skriptů jsou doporučeny výše zmíněné skriptovací jazyky. SikuliX je WYSIWYS-Tool[2]. Tento testovací nástroj lze plně využít i v programovacích jazycích Java nebo C#, ve kterých stačí pouze naimportovat knihovnu a poté použít metody pro testování například v JUnit v případě jazyku Java [9].

[1] XPath je počítačový jazyk, pomocí kterého lze vybírat elementy v XML dokumentu a pracovat s jejich hodnotami a atributy.

[2] WYSIWYS-Tool - What You See Is What You Script (Co vidíte, to skriptujete)



Obr. 3.4: Ukázka použití SikuliX [11]

3.3 Porovnání nástrojů

	Vývojáři	**Rok vydání**	**Licence**	**Poslední úprava**	**Podpora**
TestFX	Dain Nilsson, Benjamin Gudehus, Henrik Stråth	2012	EUPL	03/2016	JavaFX
Automaton	Renato Athaydes	2014	Apache 2.0 License	01/2016	Swing, JavaFX
MarvinFX	Hendrik Ebbers	2013	-	04/2013	JavaFX
TestComplete	SmartBear	-	Node-Locked Floating User	11/2015	JavaFX, Java, WPF, atd.
Selenium WebDriver	SeleniumHQ	2004	Apache 2.0 License	10/2015	Web
White Framework	Mehdi Khalili, Michael Whelan	2012	Apache 2.0 License, MIT	01/2016	Win32, WinForms, WPF, atd.
Sikuli	User Interface Design Group (MIT)	2014	MIT License	11/2015	Windows, Linux, Mac

Tab. 3.1: Porovnání nástrojů

4 PODROBNĚJŠÍ ANALÝZA NÁSTROJE TESTFX

Na základě porovnání různých testovacích nástrojů pro platformu JavaFX podle dat z tabulky 3.1 a popisu podle dokumentace je vybrán nástroj TestFX, protože je stále vyvíjen, podporován oproti ostatním nástrojům a je zdarma.

4.1 JUnit

Jedná se o testovací framework pro jednotkové testy. Díky svému úspěchu byly dále rozšířeny do dalších programovacích jazyků, jako jsou například C#, C++, Python a mnoho dalších. JUnit testy fungují na principu provedení akce samostatně testovatelné části programu a následné porovnání výstupu aplikace s očekávaným výstupem.

4.2 JavaFX

Softwarová platforma JavaFX byla vyvinuta pro vývoj RIA[1] aplikací. Oproti starší platformě Swing je výhodou oddělení grafické a logické části aplikace. JavaFX byla nejdříve vydána ve verzích 1.0 - 1.3. V těchto verzích byl používán skriptovací jazyk JavaFX Script. Následovala verze 2.0, ve které bylo možné použít jazyk FXML pro tvorbu uživatelského rozhraní. Tato verze ukončila vývoj jazyka JavaFX Script, který byl nahrazen jazykem Java. Verze 8 již nahradila stávající platformu Swing. Stále je ale možné v JavaFX aplikaci používat Swing komponenty. Nejnovější verze JavaFX je 9 [19].

4.2.1 FXML

Tento jazyk je využíván v JavaFX pro tvorbu uživatelského rozhraní. Je založen na XML formátu. FXML dokument popisuje rozmístění jednotlivých komponent v okně aplikace, slouží pro oddělení grafické části aplikace od ostatních částí. Pomocí jednoznačného identifikátoru `fx:id` lze s komponentou pracovat v controlleru. K FXML dokumentu se často připojuje controller, který definuje události k jednotlivým prvkům. Jazyk FXML nemá vlastní validační schéma, má ale základní předdefinovanou strukturu popisující řazení elementů a atributů. Dále je možné používat CSS pro nastavení vzhledu komponenty.

4.2.2 JavaFX aplikace

Aplikace může být tvořena pouze jednou Java třídou, která obsahuje základní metodu `main(String[] args)`, příkazy pro vytvoření okna, scény a příslušných komponent. Podobně funguje platforma Swing, která je předchůdcem pro JavaFX. Další možností, jak

[1] RIA jsou webové aplikace, které se podobají grafickým desktopovým aplikacím.

vytvořit aplikaci, je vytvořit FXML soubor k hlavní třídě (viz 4.2.1) a k tomu příslušnou Java třídu označovanou jako controller. Tato třída obsahuje definice použitých komponent v FXML dokumentu a dále jejich akce.

Základní třída aplikace

Tato třída musí dědit od třídy `javafx.application.Application`, proto je potřeba implementovat metodu `start(Stage stage)`. Každému oknu aplikace je potřeba nastavit tzv. scénu, která obsahuje veškerý obsah k vykreslení z FXML dokumentu [19].

4.2.3 GUI testování

Pro automatické testování GUI je nutno simulovat akce uživatele při práci s aplikací. Je potřeba hledat ovládací prvky v okně, psát na klávesnici, klikat myší nebo provádět jiné činnosti podobně jako uživatel. Následně se musí ověřit, jestli výsledek dané akce je správný. Například jestli po kliknutí na tlačítko došlo k přidání položky do seznamu.

4.3 TestFX

V kapitole 2 jsou popsány různé metody testování, které se v testování software používají. Testovací nástroj TestFX se používá k automatickému testování. Tento nástroj při testování spustí aplikaci a automaticky provádí předdefinované akce. Spadá do kategorie pro dynamické testy. TestFX slouží k testování jednotlivých metod tříd, provádí se testování jednotek (kapitola 4.1). Zde konkrétně pomocí knihovny JUnit.

4.3.1 Integrace TestFX do testů

Na začátek je potřeba do projektu JavaFX vložit knihovny pro JUnit testy. Dále je potřeba pro provádění testů vložit knihovny nástroje TestFX, které se nacházejí v JAR souborech.

Pro testy se vytváří třída, která má název typicky ukončený klíčovým slovem `Test`, ve které je potřeba naimportovat knihovny JUnit a TestFX. Tato třída musí dědit od třídy `ApplicationTest`, která je součástí knihoven TestFX.

Stejně jako musí hlavní třída aplikace v JavaFX implementovat metodu `start`, musí i tady třída implementovat stejnou metodu, ve které se načte scéna okna programu, který se bude testovat. Následně se tvoří již jen testovací metody JUnit testů, které jsou označeny anotací `@Test`, která označuje JUnit test.

4.3.2 Určení pozice prvku

Níže jsou vyjmenovány třídy objektů, které slouží k identifikaci ve vykresleném okně aplikace. Mohou být obsaženy v parametru metod, které potřebují znát k vykonání pozici nebo přímo objekt daného prvku. Taková metoda může provést přesunutí kurzoru myši na určenou pozici a provede akci, ke které je určena.

- `double positionX`, `double positionY`
- `Point2D`
- `Bound`
- `PointQuery`

Všechny tyto typy označují místo na obrazovce, které je určeno přesnými X a Y souřadnicemi. Většina metod, které pracují s kurzorem myši, přesune kurzor na X a Y souřadnice vzhledem k levému hornímu rohu obrazovky. K přesunutí dojde i pokud na daném místě obrazovky není okno testované aplikace. Pouze metody končící příponou `By` přesunou kurzor relativně k jeho aktuální pozici.

Identifikátory

Pro nalezení komponenty se používá několik identifikátorů. Nejpoužívanější je identifikátor `fx:id`, který lze definovat pouze v FXML komponentách. Jedná se o textový identifikátor. V metodách pro nalezení komponenty pomocí tohoto identifikátoru je potřeba vložit před tento identifikátor znak `#`. Jestliže komponenta má nastavený styl pomocí CSS identifikátoru, je možné tento CSS identifikátor použít k nalezení. Pozici lze dále určit pomocí identifikátorů `Scene`, `Node` a `Window`. `Scene` a `Window` označují objekty, které jsou základem pro vykreslení okna aplikace. Objekt typu `Node` značí každou komponentu, která je zobrazena v okně. Další možnost určení pozice prvku je pomocí řetězce `String`. Ten určí pozici podle porovnávání nalezeného textu v aplikaci, nezávisle na typu objektu daného prvku.

Matcher a Predicate

Jako další se využívají `Matcher` metody, které se nachází v knihovně `org.testfx.matcher.base.NodeMatchers`. Jedná se o knihovnu, která obsahuje metody reprezentující určité podmínky. První metoda se jmenuje `isNull()`, která otestuje, jestli má prvek hodnotu `null`. Opakem této metody je `isNotNull()`. Další metodou je `isVisible()`, která ověří, jestli je nalezená komponenta viditelná, popřípadě opačnou funkci má metoda `isInvisible()`. Další metody jsou `isEnabled()` a `isDisabled()`, které zjišťují, zda je nebo není možné komponentu používat. Pro testování na základě obsaženého textu slouží metoda `hasText()`. Pro ověření, jestli má prvek konkrétního potomka, se používá metoda `hasChild()`. Poslední metoda, která má podobnou funkčnost jako předchozí, se jmenuje

`hasChildren()`, která v prvním parametru očekává počet potomků a v druhém parametru název prvku. Testuje, zda má komponenta zadaný počet potomků s určitým názvem.

Jako poslední identifikátor se používá rozhraní `Predicate`. Jedná se o rozhraní, které má definované vlastnosti pomocí vnitřní třídy nebo lambda výrazu. Každé rozhraní `Predicate` musí mít hodnotu ve výsledku pravda nebo nepravda. V lambda výrazu se pracuje s konkrétním objektem, který je dále zpracováván. Objekt může být zpracováván a v závislosti na podmínce je vrácena hodnota logického datového typu, aby měl `Predicate` vždy hodnotu pravda nebo nepravda.

Pro ukázku tento lambda výraz zpracovává komponentu `TableView`, která obsahuje jako řádky objekty typu `Entry`. Pokud alespoň jeden objekt obsahuje v atributu `title` text `Seznam`, je vrácena logická hodnota pravda.

```
1 (TableView<Entry> table) -> {
2     for(Entry row : table.getItems())
3     {
4         if(row.getTitle().compareTo("Seznam") == 0)
5             return true;
6     }
7     return false;
8 }
```

Ukázka 4.1: Ukázka použití Predicate

4.3.3 Vyhledávání komponenty

K vyhledání komponenty se používá metoda `org.testfx.api.FxRobot.lookup()`, která obsahuje v parametru jeden z výše uvedených identifikátorů. Nalezne komponentu a vrátí ji jako typ `org.testfx.service.query.NodeQuery`. Jedná se o objekt třídy z TestFX, který obsahuje výsledek hledání. Pro získání první nalezené instance dané komponenty tohoto typu objektu slouží metoda `org.testfx.service.query.NodeQuery.query()`, která vrací hledanou komponentu, která je ale typu `Node`. Tento typ již patří mezi typy komponent JavaFX. Je potřeba ještě objekt přetypovat na původní typ komponenty. Pokud je nalezeno více objektů, používá se metoda `queryAll()`, která vrací kolekci `Set` obsahující objekty typu `Node`. Pomocí iterátoru se získají jednotlivé komponenty a následně se otestují.

Další metoda pro nalezení komponenty je `org.loadui.testfx.GuiTest.find()`. Obsahuje stejný parametr jako výše zmíněná metoda `lookup()`. Liší se jen v tom, že tato metoda vrací již rovnou objekt typu `Node`. Pokud se v aplikaci nachází více komponent se stejným identifikátorem, je nalezena pouze první komponenta.

4.3.4 Metody pro práci v okně aplikace

Při testování je potřeba simulovat v okně různé akce, které mohou nastat. Například nastavit pozici kurzoru myši, stisknout určitou klávesu nebo kombinaci kláves, nasimulovat stisknutí tlačítka, kliknutí na komponentu nebo napsání textu do textového pole. Metody, které vyžadují v parametru určit pozici v aplikaci, používají jeden z různých typů identifikátorů pro určení pozice z kapitoly 4.3.2. Tyto metody se nacházejí v knihovně `org.testfx.api.FxRobot`.

Akce klávesnice

Akce klávesnice lze simulovat různými metodami. Metoda `press()` očekává v parametru objekt typu `KeyCode` klávesy, která se má stisknout, popřípadě kombinaci kláves, které se stisknou naráz, nebo objekt tlačítka myši `MouseButton`. Je možné i opačné akce, uvolnění klávesy pomocí `release()`, která obsahuje v parametru klávesy nebo tlačítka myši, které se mají uvolnit. Pokud se pomocí metody `press()` stiskne klávesa, je stisknutá do té doby, než je zavolána metoda `release()`.

Metoda `push()` funguje podobně jako metoda `press()`. Jediný rozdíl je, že se klávesa stiskne a hned uvolní. V parametru může obsahuvat typ `KeyCode` nebo `KeyCodeCombination`. Metoda `type()` funguje obdobně jako předchozí metoda. Jediný rozdíl je ten, že nemůže obsahovat v parametru typ `KeyCodeCombination`, ale může obsahovat jako druhý parametr číslo, které znamená, kolikrát se má daná klávesa stisknout.

Pomocí metody `write()` se zapíše daný text do prvku, který je vybrán. Jako parametr se zadává typ `String`. V implementaci této metody se projdou všechny znaky z parametru. Každý znak je stisknut pomocí metody `press()` a uvolněn metodou `release()`.

Pozice kurzoru

Pro určení přesné pozice v aplikaci se používá objekt třídy `PointQuery`, což je třída implementována v TestFX. Pozici lze samozřejmě zadat i jinak, ale vždy je zavolána metoda `point()`, která vrátí právě objekt třídy `PointQuery`. Tato metoda vrátí pozici vzhledem k levému hornímu rohu obrazovky. Další metoda pro získání tohoto objektu se jmenuje `offset()`. Na rozdíl od metody `point()`, vyžaduje jako první parametr prvek určující pozici (kapitola 4.3.2) v aplikaci a jako další parametry X a Y souřadnice. Výsledné souřadnice mají hodnotu, která vznikla relativním posunutím hodnot X a Y od středu prvku, který byl zadán v prvním parametru.

Akce pro přesunutí kurzoru myši

Pro přesunutí kurzoru myši na určitou pozici slouží metoda `moveTo()`, která může obsahovat parametry určující pozici. Tato metoda přesune kurzor přesně na pozici, která je

zadána. Existuje ještě metoda `moveBy()`, která očekává v parametru x a y souřadnice pozice, které se ale přičtou k současné pozici, tedy přesune kurzor relativně k aktuální pozici.

Akce pro myš

Další jsou metody pro práci s myší. Kliknutím myši lze snadno nasimulovat pomocí metody `click()`, která provede klik levého tlačítka na myši na dané pozici kurzoru. Případně se v parametru může nastavit tlačítko myši, které se má stisknout. Aby se nemuselo vždy před kliknutím volat `moveTo()` pro specifikaci pozice a dále klik, je implementována metoda `clickOn()`, která obsahuje v parametru souřadnice pozice prvku nebo komponentu, na které se má provést klik levého tlačítka na myši.

Stejné typy parametrů platí i pro metodu `rightClick()`, která má stejnou funkčnost jako `click()`, jen je nasimulováno stisknutí pravého tlačítka na myši. Dále existuje metoda `rightClickOn()`, která funguje a má stejné parametry jako `clickOn()`, akorát je opět provedeno stisknutí pravého tlačítka na myši.

Jako poslední metoda pro práci s myší je `doubleClick()`¸ která provede dvojité kliknutí objektem `MouseButton` uvedeným v parametru. Metoda `doubleClickOn()` provede dvojité kliknutí levého tlačítka myši na zadanou pozici v parametru.

Akce pro drag and drop

Některé komponenty v JavaFX umožňují přesunutí pomocí drag and drop. Pro simulaci této akce je vytvořena metoda `drag()`. Tato metoda vrátí objekt `FxRobot`, na kterém se zavolá metoda `to()`, která obsahuje v parametru pozici nebo objekt, kam má být komponenta přesunuta. Na výstupním objektu této metody se volá metoda `drop()`, která uvolní tlačítko myši a ukončí drag and drop operaci. Při využití metody `dropTo()` se v parametru určí pozice, na které se uvolní tlačítko myši a ukončí drag and drop operace. Jedná se o spojení metod `to()` a `drop()`.

Akce pro posouvání v okně

V případě, že je okno nebo oblast komponenty, která podporuje skrolování, větší než současná zobrazená velikost, je zobrazen posuvník pro skrolování. Pro testování je možné skrolovat pomocí metody `scroll()`, která očekává jako první parametr směr posunu (výčtový typ třídy `VerticalDirection`) a druhý nepovinný parametr počet kroků pro skrolování.

Uspání okna

Metoda `sleep()` slouží k uspání okna aplikace. V parametru obsahuje čas, po který má být okno uspáno v milisekundách. Případně jako další parametr může být objekt typu

`TimeUnit`, který udává časový formát (milisekundy, sekundy, hodiny).

4.3.5 Testování složité komponenty

Testování komponent, jako jsou například `Button` nebo `TextField`, je relativně jednoduché. Většinou se testuje, zda obsahuje určitý text nebo jestli má dané vlastnosti. Komponenty `ListView`, `TreeView` a `TableView` jsou již obtížnější na otestování, protože každá z nich tvoří strom, který má své další potomky a teprve ty se dají testovat.

Pro ověření, jestli komponenta obsahuje určitý text, se používá `hasText()`. Pokud se testují složitější komponenty, využívá se metoda `hasChild()`, která slouží pro testování potomků prvku. Ta ověří, zda má prvek potomka s názvem zadaným v parametru. Testování uzlů, které mají jako rodiče hlavní uzel stromu, je pomocí metody `hasChild()` jednoduché. K otestování komponent se používají především `Predicate` rozhraní nebo jiné `Matcher` metody z kapitoly 4.3.2.

TreeView

Problém nastává, pokud se má otestovat prvek, který má vyšší úroveň zanoření než 1, tedy jejich rodič není hlavním prvkem stromu. K tomu se využívá rozhraní `Predicate`, které se vkládá opět jako druhý parametr metody pro testování. Jedná se o dekladaci proměnné například typu `TreeView<String>`, která je rozšířena lambda výrazem. V tomto výrazu je již možné zpracovávat komponentu `TreeView<String>`. Například pokud se hledá uzel s úrovní zanoření 2, projdou se v cyklech postupně všichni potomci a pokud se najde shoda, je vrácena logická hodnota pravda a test je úspěšný.

TableView

Pro komponentu `TableView` jsou vytvořeny speciální metody, které se nachází ve třídě `org.testfx.matcher.control.TableViewMatchers`. Ta obsahuje celkem tři metody. K ověření, jestli tabulka obsahuje v libovolné buňce určitou hodnotu, se používá metoda s názvem `hasTableCell()`. K otestování celkového počtu řádků v tabulce je implementována metoda `hasItems()`. Jako poslední je vytvořena `containsRow()`, která slouží k otestování hodnoty první buňky v zadaném řádku.

Bohužel chybí metoda na otestování hodnoty buňky v zadaném řádku a sloupci. K otestování buňky na přesně zadané pozici, se musí využít opět `Predicate`, kde se v lambda výrazu zjistí hodnota konkrétní buňky. Poté v závislosti na podmínce je vrácena hodnota logického datového typu podle toho, zda došlo nebo nedošlo ke shodě. Bylo by možné se vyhnout rozhraní `Predicate`, získat instanci tabulky metodou `find()` a použít stejné

metody pro nalezení konkrétní buňky jako v `Predicate`, ale už by nebylo možné vyhodnotit test metodou `verifyThat()` (viz kapitola 4.3.6), ale musela by se použít například metoda `assertEquals()` z knihovny JUnit.

ListView

Pro tuto komponentu opět existuje třída se speciálními metodami, které jsou v knihovně `org.testfx.matcher.control.ListViewMatchers`. První metoda se jmenuje `hasListCell()`, která má podobnou funkčnost jako metoda zmíněná výše pro tabulku nebo `hasChild()`. Třetí se jmenuje `isEmpty()`, která otestuje, jestli komponenta obsahuje nějaké prvky.

Další metody testují výchozí text v seznamu. Metoda `hasPlaceholder()` testuje, zda seznam obsahuje daný výchozí text. V této komponentě se výchozí text vypíše nejjednodušeji pomocí komponenty `Label`. To znamená, že při testování se do parametru metody `hasPlaceholder()` musí uvést komponenta `Label` s textem. Poslední je metoda `hasVisiblePlaceholder()`. Ta testuje, jestli je zobrazen výchozí text. To znamená jestli `Label`, pomocí kterého je text vypsán, má nastavený atribut `visible` na `true`. Neznamená to, že je výchozí text skrytý kvůli prvkům, které seznam obsahuje. I zde parametr obsahuje komponentu `Label` s textem.

4.3.6 Vyhodnocení testu

Pro vyhodnocení testu se používá nejčastěji testovací metoda `verifyThat()`, která je součástí třídy `org.loadui.testfx.Assertions`. Je to metoda implementovaná v testovacím nástroji TestFX, proto je schopna využívat tzv. `Matcher` metody a `Predicate` rozhraní.

Tato metoda je přetížená, může obsahovat až 3 parametry. První je zpráva, která se uživateli zobrazí, dopadne-li test neúspěšně. Tento parametr je volitelný. Dalším parametrem je přímo testovaný objekt, například tlačítko, textové pole nebo jiná komponenta. Poslední parametr značí očekávanou hodnotu komponenty, která je testována. Může být použit `Predicate` nebo `Matcher` metoda z kapitoly 4.3.2. Pokud se použije `Predicate` pro otestování očekávané hodnoty, nemůže být jako první parametr použitá chybová zpráva. Taková metoda `verifyThat()` není implementována. Naopak při použití `Predicate`, nemusí být testovaným objektem přímo objekt komponenty, ale může se využít i identifikátoru typu `String`, podle kterého je následně automaticky komponenta vyhledána.

JUnit testy je možné vyhodnocovat pomocí `assert` metod z třídy `org.junit.Assert`, která je implementována přímo v knihovně JUnit. Testování je ale obtížnější, protože není možné využívat metody a objekty nástroje TestFX.

5 TUTORIÁL PRO POUŽITÍ TEXTFX

5.1 Představení aplikace

Tutoriál popisuje otestování aplikace, která slouží k uložení přihlašovacích údajů k různým účtům. Aplikaci si můžete představit jako jednoduchý program KeePass. Aplikace umí základní funkce, jako je vytvořit, upravit nebo smazat položku.

5.1.1 Struktura

Aplikace je vytvořena pomocí FXML. Díky tomu je možné se při testování odkazovat na jednotlivé komponenty pomocí jejich identifikátoru `fx:id`. Hlavní třída, která obsahuje metodu `main()`, se jmenuje `App`. V metodě `main()` se načte FXML soubor, ze kterého se vytvoří scéna a ta se následně vykreslí v okně aplikace. Další třída se nazývá `Entry`. Tato třída reprezentuje účet. Jako atributy obsahuje název účtu, přihlašovací jméno a heslo, popis účtu, datum vypršení platnosti hesla a logickou hodnotu, zda může heslo vypršet. Controller se jmenuje `FXMLAppController`. Obsahuje metody, které definují různé akce, například pro zobrazení detailu účtu, přidání nebo odebrání účtu. Dalším souborem je `FXMLApp.fxml`. Jedná se o grafickou část aplikace, která obsahuje všechny komponenty aplikace.

Posledním souborem je třída `FXMLAppControllerTest`, která obsahuje všechny testy této aplikace. Metoda `start()` zde funguje úplně stejně jako ve třídě `App`, slouží pro vykreslení okna. Dále obsahuje jen testovací metody nástroje JUnit.

5.1.2 Použité komponenty

- `TableView` - slouží pro zobrazení seznamu uložených přihlašovacích údajů, kde 1. sloupec označuje název účtu a 2. je přihlašovací jméno
- `TextField` - tato komponenta je použita pro zadání a editaci názvu účtu, přihlašovacího jména a hesla
  - textové pole pro heslo je validováno - musí obsahovat minimálně 6 znaků
    - pokud zadané heslo má méně než 6 znaků, text a rámeček mají červenou barvu
- `Button` - v aplikaci jsou celkem 3 tlačítka - přidání účtu do tabulky, odebrání a uložení editovaných údajů
- `Label` - zobrazuje notifikace aplikace
- `TextArea` - slouží pro zadání popisu účtu
- `DatePicker` - označuje datum, kdy vyprší platnost hesla

- `CheckBox` - povoluje nebo zakazuje možnost zadat datum pro vypršení platnosti hesla

5.2 Testování

5.2.1 Knihovny

Testovací nástroj JUnit je obsažen v jednom `.jar` souboru. Nejnovější verze JUnit je 4.12, po stažení má soubor název `junit-4.12.jar`. Od verze 4.11 je potřeba stáhnout i knihovnu `hamcrest-all-1.3.jar`. Do verze 4.10 byla tato knihovna obsažena v `junit.jar`. Knihovna Hamcrest obsahuje `Matchers` a `assertThat` metody [4].

Nástroj TestFX potřebuje ke svému použití celkem 4 `jar` soubory. Nejnovější verze TestFX je 4.0.4. Všechny tyto knihovny musí být vloženy v *ClassPath* projektu (viz níže). Názvy jednotlivých knihoven jsou následující:

- `testfx-core-4.0.4-alpha.jar`
- `testfx-junit-4.0.4-alpha.jar`
- `testfx-legacy-4.0.4-alpha.jar`
- `guava-19.0.jar`[1]
  - knihovna potřebná pro plné využití `Predicate`

5.2.2 Stažení knihoven

Jednotlivé nejnovější verze knihoven jsou ke stažení na následujících odkazech:

- `junit-4.12.jar` - `http://goo.gl/HB8rJF`
- `hamcrest-all-1.3.jar` - `http://goo.gl/6wXypa`
- `testfx-core-4.0.4-alpha.jar` - `http://goo.gl/CBkSOG`
- `testfx-junit-4.0.4-alpha.jar` - `http://goo.gl/gHyvKo`
- `testfx-legacy-4.0.4-alpha.jar` - `http://goo.gl/jHsvSS`
- `guava-19.0.jar` - `http://goo.gl/dnsgCg`

5.2.3 Import knihoven do projektu

Eclipse

Aby bylo možné pracovat s externími knihovnami, musí být vloženy v *ClassPath* projektu.

1. v Eclipse klikneme v `Package Explorer` pravým tlačítkem myši na náš vytvořený projekt, který obsahuje aplikaci pro otestování
2. v zobrazených možnostech klikneme na `Properties`

[1] Google Guava - knihovna pro Javu od Google

3. vlevo klikneme položku `Java Build Path`
4. nyní se zobrazily všechny externí knihovny, které jsou doposud importované v projektu
5. klikneme na tlačítko `Add External JARs`
6. ze složky vybereme všechny `JAR` stažené knihovny
   (a) `junit-4.12.jar`
   (b) `hamcrest-all-1.3.jar`
   (c) `testfx-core-4.0.4-alpha.jar`
   (d) `testfx-junit-4.0.4-alpha.jar`
   (e) `testfx-legacy-4.0.4-alpha.jar`
   (f) `guava-19.0.jar`
7. nyní bychom měli vidět přidané `JAR` soubory v seznamu

NetBeans

Zde je velice podobný postup jako v Eclipse.

1. v `Package Explorer` klikneme pravým tlačítkem myši na náš vytvořený projekt, který obsahuje aplikaci pro otestování
2. v zobrazených možnostech klikneme na `Properties`
3. vlevo klikneme položku `Libraries`
4. dále klikneme na tlačítko `Add JAR/Folder`
5. ze složky vybereme všechny `JAR` stažené knihovny
   (a) `junit-4.12.jar`
   (b) `hamcrest-all-1.3.jar`
   (c) `testfx-core-4.0.4-alpha.jar`
   (d) `testfx-junit-4.0.4-alpha.jar`
   (e) `testfx-legacy-4.0.4-alpha.jar`
   (f) `guava-19.0.jar`
6. nyní bychom měli vidět přidané `JAR` soubory v seznamu

5.2.4 Testy

Níže je uveden podrobný postup, jak tvořit testovací metody. Jednotlivé kroky jsou označeny číselnými odrážkami.

1. aplikace je testována pomocí JUnit testů
2. to znamená, že jednotlivé testovací metody musí mít anotaci `@Test`, deklarace hlavičky začíná klíčovými slovy `public void` a neobsahují žádný parametr
3. následuje začátek názvu metody, který je typicky `test`
4. jednotlivé komponenty budeme hledat pomocí jejich `fx:id` identifikátoru

(a) v metodách nástroje TestFX je potřeba před tento identifikátor vložit znak `#`, aby bylo jasné, že se jedná o `fx:id` identifikátor (viz kapitola 4.3.2)
    i. občas se používá `String` identifikátor, pokud je potřeba například kliknout na položku v seznamu

Přidání účtu

1. první testovací metoda ověří, pokud po vyplnění textového pole pro název účtu a přihlašovací jméno a následnému kliknutí na tlačítko `Add entry`, jestli se objeví daný účet v tabulce
   (a) pomocí metody `clickOn()` klikneme na textové pole pro název účtu a následně metodou `write()` zapíšeme název `Centrum` do textového pole (řádek č. 4 v ukázce 5.1)
   (b) stejnými metodami zapíšeme do textového pole pro login přihlašovací jméno `NovyPavel.2` (řádek č. 5)
   (c) nakonec klikneme metodou `clickOn()` na tlačítko `Add entry` (řádek č. 6)
   (d) všechny tyto metody jsou popsány v kapitole 4.3.4
2. nyní už jen ověříme pomocí metody `verifyThat()`, jestli se skutečně přidal účet do tabulky
   (a) tato ověřovací metoda je popsána v kapitole 4.3.6
   (b) v prvním parametru napíšeme chybovou hlášku, která se vypíše v případě, že test skončí chybou
   (c) v druhém parametru této metody pomocí metody `find()` z kapitoly 4.3.3 najdeme komponentu tabulky
   (d) v třetím parametru využijeme metodu `containsRow()`, která je popsána v kapitole 4.3.5, k ověření, zda na první buňce poslední řádky tabulky je hodnota `Centrum` (řádek č. 8)
   (e) pro ověření, zda obsahuje druhá buňka správnou hodnotu, musíme zavolat znovu metodu `verifyThat()`, tentokrát využijeme jako druhý parametr rozhraní `Predicate` (řádek č. 9)
   (f) v lambda výrazu zjistíme, jaká hodnota je na posledním řádku a druhé buňce, kterou následně porovnáme s textem `NovyPavel.2` a vrátíme logickou hodnotu `Predicate` (řádek č. 11 a 12)
3. kompletní testovací metoda vypadá následovně

```
1 @Test
2 public void testAddEntry()
3 {
4   clickOn("#titleAddTF").write("Centrum");
5   clickOn("#loginAddTF").write("NovyPavel.2");
6   clickOn("#addBT");
7   TableView<Entry> entryTable = find("#entryTable");
8   verifyThat("Invalid title at last row in table", entryTable,
        containsRow(entryTable.getItems().size() - 1, "Centrum"));
9   verifyThat(entryTable, (TableView<Entry> table) ->
10  {
11    String login = table.getColumns().get(1).getCellData(table.
        getItems().size() - 1).toString();
12    return login.compareTo("NovyPavel.2") == 0;
13  });
14 }
```

Ukázka 5.1: Přidání účtu

Přidání účtu s neúplnými informacemi

1. pokud uživatel zapomene vyplnit jedno z textových polí pro přidání, účet se nemůže přidat do tabulky
2. k otestování jsou vytvořeny dvě testovací metody
   (a) v první metodě nejprve klikneme na textové pole pro název účtu, zapíšeme text `Centrum` a následně klikneme na tlačítko `Add entry`
   (b) v druhé metodě vyplníme pouze přihlašovací jméno `NovyPavel.2` a klikneme na tlačítko `Add entry`
   (c) v obou případech otestujeme v ověřovací metodě `verifyThat()` pomocí `Predicate`, zda existuje v tabulce řádek s hodnotou, která byla vyplněna v textovém poli
      i. metodou `find()` najdeme komponentu tabulky (řádek č. 6 v ukázce 5.2)
      ii. v lambda výrazu v cyklu `for` projdeme všechny instance třídy `Entry`, které tabulka obsahuje (řádek č. 8)
      iii. u každé instance testujeme shodu s názvem účtu nebo přihlašovacím jménem (řádek č. 10)
      iv. pokud nenalezneme ani jednu shodu, vrátíme logickou hodnotu pravda, jinak vrátíme logickou hodnotu nepravda a test je neúspěšný
   (d) v případě jakékoliv chyby je vypsána informační hláška
      i. po nalezení příslušné komponenty typu `Label` ověříme, jestli má očekávaný text pomocí `Matcher` metody `hasText()` z kapitoly 4.3.2 (řádek č. 15)

Tyto testovací metody jsou téměř stejné.

```
1 @Test
2 public void testIncompleteAddEntry()
3 {
4   clickOn("#titleAddTF").write("Centrum");
5   clickOn("#addBT");
6   verifyThat(find("#entryTable"), (TableView<Entry> entryTable) ->
7   {
8     for(Entry temp : entryTable.getItems())
9     {
10      if(temp.getTitle().compareTo("Centrum") == 0)
11        return false;
12    }
13    return true;
14  });
15  verifyThat("Invalid notification", find("#notification"), hasText
      ("Please fill in the title and login text field."));
16 }
17
18 @Test
19 public void testIncompleteAddEntry2()
20 {
21  clickOn("#loginAddTF").write("NovyPavel.2");
22  clickOn("#addBT");
23  verifyThat(find("#entryTable"), (TableView<Entry> entryTable) ->
24  {
25    for(Entry temp : entryTable.getItems())
26    {
27      if(temp.getLogin().compareTo("NovyPavel.2") == 0)
28        return false;
29    }
30    return true;
31  });
32  verifyThat("Invalid notification", find("#notification"), hasText
      ("Please fill in the title and login text field."));
33 }
```

Ukázka 5.2: Přidání účtu s neúplnými informacemi

Smazání účtu

Smazání účtu z tabulky je jednoduchá operace.

1. při testování nejprve vybereme metodou `clickOn()` konkrétní účet v tabulce, který chceme smazat
   (a) tentokrát nebudeme uvádět na začátek znak `#`, protože se nejedná o `fx:id`, ale o klasický `String` identifikátor (řádek č. 4 v ukázce 5.3)
2. následně klikneme na tlačítko `Remove entry`
3. objeví se dialogové okno, ve kterém je potřeba stisknout tlačítko `OK`, díky čemuž dojde ke smazání účtu
4. jelikož neexistuje `Matcher` metoda, která by ověřila, že prvek neexistuje v tabulce, musíme zde použít opět `Predicate`
   (a) tento testovací postup je podobný výše zmíněnému postupu, kde jsme testovali, jestli se nepřidal neúplný účet do tabulky

(b) metoda `verifyThat()` obsahuje jako druhý parametr lambda výraz, ve kterém projdeme všechny instance třídy `Entry` v tabulce (řádek č. 9)

(c) u každé instance porovnáme, zda nemá název účtu stejný, jako měl účet, který jsme právě smazali

(d) pokud nenalezneme ani jednu shodu, vrátíme logickou hodnotu pravda, jinak vrátíme logickou hodnotu nepravda a test je neúspěšný

```
1 @Test
2 public void testRemoveEntryOk()
3 {
4   clickOn("Twitter");
5   clickOn("#removeBT");
6   clickOn("OK");
7   verifyThat(find("#entryTable"), (TableView<Entry> entryTable) ->
8   {
9     for(Entry temp : entryTable.getItems())
10    {
11      if(temp.getTitle().compareTo("Twitter") == 0)
12        return false;
13    }
14    return true;
15  });
16 }
```

Ukázka 5.3: Smazání účtu

Dialogové okno pro potvrzení smazání

1. aby nedocházelo k nechtěnému odstranění účtu z tabulky, je potřeba smazání potvrdit v dialogovém okně
2. bohužel nástroj TestFX neumožňuje otestovat, že se vytvořilo dialogové okno
   (a) musíme to obejít otestováním jiné činnosti, která by se neprovedla, pokud by bylo zobrazeno dialogové okno
3. podobně jako v minulém testu vybereme účet, který chceme smazat, a klikneme na tlačítko `Remove entry`
4. momentálně by se mělo zobrazit dialogové okno, které obsahuje tlačítko pro potvrzení smazání účtu nebo zrušení této akce
   (a) pomocí kliknutí na položku mimo dialogové okno můžeme otestovat, že se v hlavním okně aplikace neprovede žádná změna, protože je kvůli dialogovému oknu hlavní okno zablokované
   (b) pomocí metody `clickOn()` klikneme na položku `Seznam` v tabulce
      i. pokud by neexistovalo žádné dialogové okno, textová pole pro detail účtu by se vyplněny údaji účtu `Seznam` (řádek č. 7 v ukázce 5.4)
   (c) ověřovací metodou `verifyThat()` otestujeme, jestli textové pole pro změnu názvu účtu obsahuje hodnotu účtu, který jsme chtěli smazat
      i. v druhém parametru metodou `find()` najdeme textové pole pro název

účtu a pokud by obsahovalo hodnotu `Seznam`, dialogové okno by neexistovalo

(d) pro jistotu ještě otestujeme, jestli se hodnota, kterou jsme chtěli smazat, nesmazala

i. v dialogovém okně jsme nepotvrdili smazání účtu, proto by měl být účet stále v tabulce

ii. najdeme tabulku a pomocí `Matcher` metody `hasTableCell()` z kapitoly 4.3.5 zjistíme, jestli v nějaké buňce existuje hodnota, kterou jsme chtěli smazat (řádek č. 8)

```
1 @Test
2 public void testRemoveDialogBox()
3 {
4   clickOn("Twitter");
5   clickOn("#removeBT");
6   clickOn("Seznam");
7   verifyThat("Invalid title", find("#titleTF"), hasText("Twitter"))
        ;
8   verifyThat("Wrong delele entry", find("#entryTable"),
        hasTableCell("Twitter"));
9 }
```

Ukázka 5.4: Dialogové okno pro potvrzení smazání

Zobrazení detailu účtu

1. po kliknutí na libovolný účet v tabulce je potřeba, aby se zobrazily jednotlivé informace o účtu do textových polí
2. klikneme metodou `clickOn()` na požadovaný účet v tabulce
3. v pravé části aplikace se vyplní textová pole údaji konkrétního účtu
4. v této testovací metodě použijeme více testovacích metod `verifyThat()`
   (a) v každé této metodě vždy najdeme příslušnou komponentu metodou `find()`
   (b) pokud se jedná o komponentu typu `TextField`, pomocí metody `hasText()` ověříme, zda komponenta obsahuje očekávaný text
   (c) pole pro zobrazení data je typu `DatePicker`
      i. zde by metoda `hasText()` nefungovala, proto se musí použít `Predicate`
      ii. v lambda výrazu získáme aktuální text této komponenty, porovnáme s očekávaným textem a vrátíme logickou hodnotu (řádek č. 10 v ukázce 5.5)
   (d) pole typu `DatePicker` může ještě být povolené nebo zakázané (v závislosti na příslušném tlačítku)
      i. to otestujeme pomocí `Matcher` metody `isEnabled()`, která ověří, jestli je daná komponenta povolena (řádek č. 12)

```
1 @Test
2 public void testShowDetails()
3 {
4   clickOn("Facebook");
5   verifyThat("Invalid title", find("#titleTF"), hasText("Facebook")
        );
6   verifyThat("Invalid login", find("#loginTF"), hasText("pavelek"))
        ;
7   verifyThat("Invalid password", find("#passTF"), hasText("987654
        asd"));
8   verifyThat(find("#dateTF"), (DatePicker dateTF) ->
9   {
10    return dateTF.getValue().toString().compareTo("2016-09-23") ==
          0;
11  });
12  verifyThat("Date should be disabled", find("#dateTF"), isEnabled
        ());
13  verifyThat("Invalid description", find("#descTA"), hasText("Ucet
        na facebooku"));
14 }
```

Ukázka 5.5: Zobrazení detailu účtu

Editace přihlašovacího jména účtu

1. metodou `clickOn()` si vybereme požadovaný účet, u kterého chceme změnit přihlašovací jméno
2. abychom jednoduše vybrali veškerý text v textovém poli pro jméno, klikneme na textové pole a pomocí metody `push()` zmáčkneme kombinaci `Control + A` (řádek č. 6 v ukázce 5.6)
3. nyní můžete napsat nové přihlašovací jméno účtu metodou `write()`
4. kliknutím na tlačítko `Save entry` účet uložíme
5. abychom mohli zkontrolovat, jestli se správně uložilo nové jméno, musíme kliknout na jiný účet v seznamu a zpět na náš editovaný, aby se aktualizovaly informace na ty, které se uložily
6. zde použijeme metodu `verifyThat()` dvakrát
   (a) nejprve zkontrolujeme, jestli textové pole pro přihlašovací jméno obsahuje nové očekávané jméno, které jsme zadali
      i. zde použijeme klasickou `Matcher` metodu pro kontrolu textu `hasText()`
   (b) druhá ověřovací metoda testuje, jestli se změnilo přihlašovací jméno účtu i v tabulce v druhém sloupci
   (c) zde použijeme opět `Predicate`
      i. v lambda výrazu si zjistíme, který řádek v tabulce máme vybraný (řádek č. 14)
      ii. dále podle indexu tohoto řádku zjistíme, jaká je hodnota buňky, která je ve druhém sloupci (řádek č. 15)

iii. nakonec ji porovnáme s naším změněným přihlašovacím jménem

(d) místo `Predicate` rozhraní bychom mohli použít metodu `hasTableCell()`, která projde všechny buňky v tabulce

(e) to by ale nebylo tak přesné, protože v tabulce může existovat i jiná buňka, která má stejnou hodnotu

```
1  @Test
2  public void testEditLogin()
3  {
4    clickOn("Google");
5    clickOn("#loginTF");
6    push(KeyCode.CONTROL, KeyCode.A);
7    write("novypavel2");
8    clickOn("#saveBT");
9    clickOn("Seznam");
10   clickOn("Google");
11   verifyThat("Invalid login in text field", find("#loginTF"),
         hasText("novypavel2"));
12   verifyThat(find("#entryTable"), (TableView<Entry> entryTable) ->
         {
13     int index = entryTable.getSelectionModel().getSelectedIndex();
14     String nick = entryTable.getColumns().get(1).getCellData(index)
           .toString();
15     return nick.compareTo("novypavel2") == 0;
16   });
17 }
```

Ukázka 5.6: Editace přihlašovacího jména účtu

Validace hesla

1. u editace hesla je možné testovat jeho validaci
   (a) pokud je počet znaků menší než 6, text a rámeček je obarvený červeně
2. klikneme na účet v tabulce, u kterého chceme změnit heslo
3. pomocí metody `doubleClickOn()` si označíme text v textovém poli a metodou `write()` do pole zapíšeme heslo s počtem znaků menší než 6, aby se otestovala validace (řádek č. 5 v ukázce 5.7)
4. v metodě `verifyThat()` v prvním parametru najdeme komponentu textového pole pro heslo a v druhém parametru použijeme `Predicate` rozhraní
   (a) v lambda výrazu otestujeme, jestli textové pole pro heslo má nastavené určité styly (řádek č. 8)
      i. červený text a rámeček

```
1 @Test
2 public void testChangePasswordInvalid()
3 {
4   clickOn("Google");
5   doubleClickOn("#passTF").write("heslo");
6   verifyThat(find("#passTF"), (TextField passTF) ->
7   {
8     return passTF.getStyle().contains("-fx-text-inner-color: red; -
          fx-border-color: red");
9   });
10 }
```

Ukázka 5.7: Validace hesla

Počet účtů v tabulce

1. počet jednotlivých účtů v tabulce se otestuje velice jednoduše
2. v metodě `verifyThat()` se najde komponenta tabulky
3. následně se použije `Matcher` metoda `hasItems()` (řádek č. 4 v ukázce 5.8)
   (a) ta obsahuje v parametru očekávaný počet položek tabulky
   (b) metoda je popsána v kapitole 4.3.5

```
1 @Test
2 public void testCountEntry()
3 {
4   verifyThat("Invalid count of items in table", find("#entryTable")
        , hasItems(4));
5 }
```

Ukázka 5.8: Počet účtů v tabulce

Povolení pole pro datum

1. textové pole pro zadání data může být povolené nebo zakázané, záleží jestli může vypršet platnost hesla
2. po kliknutí na zaškrtávací tlačítko by se měl změnit stav textového pole pro datum
3. vybereme účet v tabulce metodou `clickOn()`
4. dále klikneme na zaškrtávací tlačítko
5. metodou `verifyThat()` otestujeme, jestli se změnil stav textového pole pomocí `Matcher` metody
   (a) najdeme textové pole pro datum a v dalším parametru ověříme stav tohoto pole
   (b) použijeme metodu `isEnabled()` nebo `isDisabled()` (řádek č. 6 v ukázce 5.9)
      i. podle toho, jaký stav očekáváme, že bude textové pole mít

```
1 @Test
2 public void testEnableDate()
3 {
4   clickOn("Seznam");
5   clickOn("#dateCh");
6   verifyThat("Date should be enabled", find("#dateTF"), isEnabled()
        );
7 }
```

Ukázka 5.9: Povolení pole pro datum

6 SHRNUTÍ HODNOCENÍ TUTORIÁLU

6.1 Zadání pro studenty

V testované aplikaci bylo záměrně vytvořeno 5 chyb. Tato upravená verze aplikace byla předána studentům z různých ročníků studující obor Informatika. Přesně na tuto aplikaci bylo vytvořeno zadání a návod k řešení. Na základě zadání, řešení, tutoriálu, komentářů a podrobného popisu testovacího nástroje TestFX měli studenti doplnit chybějících 5 JUnit testů. Tyto testy měly skončit chybou, protože testují část aplikace, ve které byla udělána chyba.

6.1.1 Jednotlivé testy

Studenti měli napsat celkem 5 testů, které se svou implementací podobaly testům, které již v testovací třídě byly napsány. U každého testu byl podrobný popis, jak test tvořit a jaké metody použít. Jedná se o následující testy:

1. Omezení přidání účtu, pokud nebylo vyplněno pole pro název účtu
2. Editace názvu účtu
3. Zobrazení notifikace po odstranění účtu z tabulky
4. Validace hesla pomocí barvy textu a rámečku
5. Povolení textového pole pro datum po kliknutí na tlačítko

6.2 Dotazník

Dále byl vytvořen dotazník, který každý student vyplnil po kompletním vypracování všech testů. Dotazník obsahuje jednoduché otázky týkající se jednotlivých testů. Každý student si měřil čas, jak dlouho mu trvalo test vytvořit. Na základě tohoto dotazníku je možné vytvořit ohodnocení tutoriálu. V případě negativních odpovědí by se tutoriál upravil, aby lépe popisoval části, které dělaly studentům problémy.

První otázkou dotazníku je, ve kterém ročníku student studuje. Jako další následuje otázka týkající se přehlednosti testovací třídy a porozumění všem napsaným komentářům. Dotazník pokračuje otázkami, jak dlouho student konkrétní test psal a zda narazil na nějaký problém nebo nejasnost. Jako další otázky jsou, jaký test se studentovi psal nejsnadněji a nejobtížněji. Závěrečná otázka je, jestli se studentovi líbí testovací nástroj TestFX.

6.2.1 Vyhodnocení dotazníku

Níže je dotazník včetně odpovědí od studentů. Názvy testů a jejich číselné označení jsou napsány v kapitole 6.1.1.

	Ročník	Pochopil jsi všechny napsané testy a komentáře?	Doba vytváření testu č. 1 [min]	Doba vytváření testu č. 2 [min]	Doba vytváření testu č. 3 [min]	Doba vytváření testu č. 4 [min]	Doba vytváření testu č. 5 [min]	Číslo nejlehčího testu	Číslo nejobtížnějšího testu	Líbí se ti TestFX?
student 1	3.	ano	3	3,5	2	2	1	5	2	ano
student 2	4.	ano	5	10	5	5	5	5	2	ano
student 3	3.	ano	4	3	2	2	2	5	1	ano
student 4	4.	ano	3	4	3	2	2	5	2	ano
student 5	1.	ano	3	6	7	3	9	1	5	ano

Tab. 6.1: Dotazník včetně odpovědí

Dotazník dále obsahuje odpovědi týkající se problémů při psaní testů. Jeden student uvedl, že měl ze začátku problém u testu č. 1 s pojmenováním textových polí pro zadání názvu účtu. Druhý student měl problém v tom samém testu. Ten porovnával v lambda výrazu výsledný řetězec se špatnou metodou třídy `Entry`. Jeden student uvedl u testu č. 5, že se mu z počátku nedařilo použít `Matcher` metodu pro ověření, zda je textové pole povolené.

Celkem 5 studentů si vyzkoušelo dopsat chybějící testy v aplikaci. Všichni úspěšně zvládni vytvořit požadované testy. Dále jsou rozebrány jednotlivé odpovědi z dotazníku od studentů.

Ročník

Z celkového počtu studentů, kteří dotazník vyplnili, 1 studuje v 1. ročníku, 2 studují ve 3. ročníku a 2 ve 4. ročníku. Časy studenta z 1. ročníku se u nějakého testu mírně liší od starších

studentů. Pravděpodobně to bude kvůli předmětu Ověřování kvality software, který starší studenti již vystudovali.

Komentáře

Všichni studenti uvedli, že pochopili všechny již napsané testy. Dále skoro všichni studenti pochopili všechny testovací metody nástroje TestFX, které jsou podrobně popsány komentáři. Jen 2 studenti uvedli, že pochopili téměř všechny testovací metody.

Testy

Test č. 1 vytvořila většina studentů v rozmezí 3 až 5 minut. U tohoto testu nenastaly žádné větší komplikace. Jednoho studenta zmátlo pojmenování textových polí. Druhý měl problém s porovnáním výsledné hodnoty buňky, protože používal špatnou metodu objektu `Entry`. Spletl si metodu `getLogin()` s metodou `getTitle()`, protože tuto metodu viděl v testu výše, který testoval přidání účtu do tabulky, pokud je vyplněno pouze pole s názvem účtu.

Vytvořit druhý test trvalo v rozmezí 3 až 10 minut. Jen jeden student uvedl, že chvilku přemýšlel, jak kliknout na jiné tlačítko.

V rozmezí 2 až 7 minut vytvářeli studenti třetí test. Nenastal žádný velký problém. Jen jeden student uvedl, že chvilku přemýšlel, jak vyhledávat komponentu podle obsaženého textu.

Předposlední test vytvořili studenti v rozmezí 2 až 5 minut. Zde vše proběhlo bez komplikací.

Poslední test měl jeden student vytvořený za 1 minutu. Nejdéle tento test psal student 9 minut. U tohoto testu nastal pouze jeden problém. Student uvedl, že se mu nedařilo použít `Matcher` metodu. Tato metoda je uvedena v popisu testu a téměř stejný test se nachází hned nad tímto testem. Ostatní studenti tento test napsali bez problémů v krátkých časech.

Vyhodnocení

Na základě odpovědí z tabulky 6.1 se každému zúčastněnému studentovi se použití testovacího nástroje TestFX líbí. Podle zaznamenaných časů a komentářů k jednotlivým testům to vypadá, že pomocí tutoriálu nemají studenti problém s otestováním JavaFX aplikace.

Za nejlehčí test studenti označili povolení textového pole pro datum po kliknutí na tlačítko. Naopak nejobtížnějším byl zvolen test editace názvu účtu. V dotazníku ani jeden student neuvedl žádný vážnější problém s implementací tohoto testu. Po kontaktování studentů mi bylo řečeno, že měli problém s `Matcher` metodou `containsRow()`, protože si pozorně nepřečetli komentáře u testovací metody `testAddEntry()`, kde je použití této metody vysvětleno. Proto nevěděli, co přesně má obsahovat první parametr.

Na základě nejasností, které mi byly sděleny studenty, jsem více podrobněji popsal problémové akce.

7 ZÁVĚR

V rámci tohoto projektu byly prozkoumány různé testovací nástroje pro uživatelské rozhraní. Každý nástroj byl popsán podle dostupných informací. Některé projekty jsou již mrtvé. Velice se mi líbí nástroj Selenium WebDriver, ale ten podporuje pouze testování internetových stránek. Dobrým kandidátem by byl TestComplete. Obsahuje mnoho různých testů pro různé platformy. Jen je bohužel placený. Další vhodný nástroj by byl TestFX, který splňuje všechny kritéria. Má sice méně funkcí než TestComplete, ale pro otestování JavaFX aplikace jednotkovými testy je dostačující. Nakonec jsem vybral nástroj TestFX, protože je zdarma a zcela dostačující pro ukázku, jak testovat uživatelské rozhraní JavaFX.

Bylo provedeno prozkoumání jeho dokumentace pro přesné pochopení, jak s nástrojem pracovat. Dokumentace ještě není zcela kompletní, proto bylo potřeba často nahlížet přímo do zdrojových kódů nástroje. Stále je vidět, že je nástroj ve vývoji. Některé testovací metody jsou implementované například pouze pro testování komponenty `ListView`, ale u komponenty `TableView` chybí. Nicméně i když se nástroj stále vyvíjí, je možné s ním kompletně bezproblémově otestovat JavaFX aplikaci. Podporuje řadu metod pro simulování určité akce stejně, jak by ji provedl uživatel.

Vytvořil jsem ukázkovou JavaFX aplikaci, kde jsem do jisté míry pokryl uživatelské rozhraní jednotkovými testy. V této aplikaci jsem následně vytvořil 5 chyb, které měli studenti otestovat. Vytvořená testovací třída je v tutoriálu detailně popsána, aby čtenář po prvním přečtení věděl, jak daná metoda pracuje. V této třídě je vynecháno celkem 5 testů, které měli studenti za úkol doplnit. Na základě dotazníku, který studenti vyplnili po napsání všech testů, skoro všichni pochopili všechny testovací metody, které obsahovala testovací třída aplikace. Díky pozitivní odezvě od studentů nebylo potřeba tutoriál téměř upravovat. Tutoriál bude v budoucnu sloužit studentům předmětu Úvod do uživatelského rozhraní pro ukázku, jak je možné testovat JavaFX aplikace.

S testováním jsem měl již zkušenosti. Po pochopení nástroje TestFX nebylo pro mě obtížné tvořit jednotkové testy.

LITERATURA

[1] Automaton. GitHub [obrázek]. Dostupné z: https://github.com/renatoathaydes/Automaton/graphs/contributors. [cit. 2016-04-17].

[2] Automaton. GitHub [online]. Dostupné z: https://github.com/renatoathaydes/Automaton, 2016. [cit. 2016-02-28].

[3] H. Ebbers. Introducing MarvinFx [online]. Dostupné z: http://www.guigarage.com/2013/03/introducing-marvinfx/, 2013. [cit. 2016-02-28].

[4] P. Herout. Přednášky z OKS. Dostupné z: http://www.kiv.zcu.cz/~herout/vyuka/oks/prednasky/oks-1a4.pdf, 2015. [cit. 2016-05-01].

[5] T. Hlava. Automatizované testování [online]. Dostupné z: http://testovanisoftwaru.cz/automatizovane-testovani/, 2011. [cit. 2016-05-01].

[6] T. Hlava. Statické a dynamické testy [online]. Dostupné z: http://testovanisoftwaru.cz/metodika-testovani/druhy-typy-a-kategorie-testu/staticke-a-dynamicke-testy/, 2011-08-12. [cit. 2016-05-01].

[7] T. Hlava. Progresní a regresní testy [online]. Dostupné z: http://testovanisoftwaru.cz/tag/progresni-testy/, 2011-08-20. [cit. 2016-05-01].

[8] T. Hlava. Fáze a úrovně provádění testů [online]. Dostupné z: http://testovanisoftwaru.cz/category/metodika-testovani/druhy-typy-a-kategorie-testu/, 2011-08-21. [cit. 2016-02-28].

[9] R. Hocke. Sikuli [online]. Dostupné z: http://sikulix-2014.readthedocs.org/en/latest/index.html, 2014. [cit. 2016-02-28].

[10] SeleniumHQ. Selenium WebDriver [online]. Dostupné z: http://www.seleniumhq.org/docs/03_webdriver.jsp, 2015-12-11. [cit. 2016-02-28].

[11] Sikuli. Facebook status update [obrázek]. Dostupné z: http://www.sikuli.org/uploads/1/3/6/8/13689586/_6384480_orig.png. [cit. 2016-02-28].

[12] SmartBear. Software Testing and Test Automation Tool TestComplete [online]. Dostupné z: http://smartbear.com/product/testcomplete/overview/, 2016. [cit. 2016-02-28].

[13] Software testing help. 15 Best Test Management Tools for Software Testers [online]. Dostupné z: http://www.softwaretestinghelp.com/15-best-test-management-tools-for-software-testers/, 2016-01-29. [cit. 2016-02-28].

[14] Sw testování. Testování GUI - malé nakousnutí velkého tématu [online]. Dostupné z: http://www.swtestovani.cz/index.php?option=com_content&view=article&id=68:testovani-gui-male-nakousnuti-velkeho-tematu&catid=3:zaklady&Itemid=11, 2014. [cit. 2016-03-18].

[15] TestFX. GitHub [obrázek]. Dostupné z: https://github.com/TestFX/TestFX/graphs/contributors. [cit. 2016-04-17].

[16] TestFX. GitHub [online]. Dostupné z: https://github.com/TestFX/TestFX, 2016. [cit. 2016-04-17].

[17] TestStack. GitHub [obrázek]. Dostupné z: https://github.com/TestStack/White/graphs/contributors. [cit. 2016-04-17].

[18] TestStack. GitHub [online]. Dostupné z: https://github.com/TestStack/White, 2016. [cit. 2016-02-28].

[19] Wikipedia. JavaFX — Wikipedia, the free encyclopedia. Dostupné z: http://en.wikipedia.org/w/index.php?title=JavaFX&oldid=712279686, 2016. [cit. 2016-02-28].

SEZNAM OBRÁZKŮ

SEZNAM PŘÍLOH

SEZNAM ZKRATEK

CSS	Cascading Style Sheets
DOM	Document Object Model
FAT	Factory acceptance tests
GUI	Graphical User Interface
HTML	HyperText Markup Language
RIA	Rich Internet applications
UAT	User acceptance test
XPath	XML Path Language

A PŘÍKLAD NA TESTOVÁNÍ UŽIVATELSKÉHO ROZHRANÍ

Společně s tímto zadáním byl studentům dodán podrobný popis testovacího nástroje TestFX z kapitoly 4.

A.1 Zadání

V tomto příkladu bude vaším úkolem si vyzkoušet nástroj pro testování uživatelského rozhraní v jazyce JavaFX. Nástroj se jmenuje TestFX. Je vám dodán kompletní projekt, který obsahuje samotný program, nakonfigurované testovací prostředí (JUnit a TestFX knihovny) a třídu obsahující ukázkové JUnit testy. Ve třídě chybí u 5ti testovacích metod jejich těla. Vaším úkolem je tyto těla doplnit. U každého testu si měřte dobu, jak dlouho daný test píšete.

Standardně nástroj JUnit spouští všechny testovací metody ve třídě. Aby se vám stále nespouštěly všechny testy, ale vždy jen ten jeden, který píšete, klikněte na název metody testu, který chcete spustit. Poté stačí jen spustit testování stejným tlačítkem jako když spouštíte program (Run - `CTRL + F11`).

A.2 Výsledek

Pokud spustíte všechny testy ve třídě, měly by vaše testy skončit chybou a zbytek testů by měl skončit v pořádku. Pokud se vám to povedlo, vyplňte prosím následující dotazník, který bude sloužit jako hodnocení tohoto zadání a tutoriálu - `http://goo.gl/forms/WBXKTTE2aC`. Děkuji. Pokud budete chtít, můžete si po vyplnění dotazníku zažádat o zaslání verze aplikace, která neobsahuje žádné chyby a obsahuje správně vypracované testovací metody, které jste měli doplnit. Pokud nahradíte testovací metody v této aplikaci vašimi metodami a testy spustíte, měly by skončit úspěšně.

A.3 Řešení úlohy - návod

A.3.1 Představení aplikace

Tutoriál popisuje otestování aplikace, která slouží pro uložení přihlašovacích údajů k různým účtům. Aplikaci si můžete představit jako jednoduchý program KeePass, která umí základní funkce, jako je vytvořit, upravit nebo smazat položku.

Struktura

Aplikace je vytvořena pomocí FXML. Díky tomu je možné se při testování odkazovat na jednotlivé komponenty pomocí jejich identifikátoru `fx:id`. Hlavní třída, která obsahuje metodu `main()`, se jmenuje `App`. V metodě `main()` se načte FXML soubor, ze kterého se vytvoří scéna a ta se následně vykreslí v okně aplikace. Další třída se nazývá `Entry`. Tato třída reprezentuje účet. Jako atributy obsahuje název účtu, přihlašovací jméno a heslo, popis účtu, datum vypršení platnosti hesla a logickou hodnotu, zda může heslo vypršet. Controller se jmenuje `FXMLAppController`. Obsahuje metody, které definují různé akce například pro zobrazení detailu účtu, přidání nebo odebrání účtu. Dalším souborem je `FXMLApp.fxml`. Jedná se o grafickou část aplikace, která obsahuje všechny komponenty aplikace.

Posledním souborem je třída `FXMLAppControllerTest`, která obsahuje všechny testy této aplikace. Metoda `main()` zde funguje úplně stejně jako ve třídě `App`, slouží pro vykreslení okna. Dále obsahuje jen testovací metody nástroje JUnit.

Použité komponenty

- `TableView` - slouží pro zobrazení seznamu uložených přihlašovacích údajů, kde 1. sloupec označuje název účtu a 2. je přihlašovací jméno
- `TextField` - tato komponenta je použita pro zadání a editaci názvu účtu, přihlašovacího jména a hesla
  - textové pole pro heslo je validováno - musí obsahovat minimálně 6 znaků
    - pokud zadané heslo má méně než 6 znaků, text a rámeček mají červenou barvu
- `Button` - v aplikaci jsou celkem 3 tlačítka - přidání účtu do tabulky, odebrání a uložení editovaných údajů
- `Label` - zobrazuje notifikace aplikace
- `TextArea` - slouží pro zadání popisu účtu
- `DatePicker` - označuje datum, kdy vyprší platnost hesla
- `CheckBox` - povoluje nebo zakazuje možnost zadat datum pro vypršení platnosti hesla

Knihovny

Po naimportování projektu do programu Eclipse byste měli mít v `ClassPath` přidané knihovny JUnit a TestFX. Pokud je nemáte, tak si je naimportujte ze složky `extlib`.

1. v Eclipse klikneme v `Package Explorer` pravým tlačítkem myši na náš vytvořený projekt, který obsahuje aplikaci pro otestování
2. v zobrazených možnostech klikneme na `Properties`

3. vlevo klikneme položku `Java Build Path`
4. nyní se zobrazily všechny externí knihovny, které jsou doposud importované v projektu
5. klikneme na tlačítko `Add External JARs`
6. ze složky vybereme všechny `JAR` stažené knihovny
   (a) `junit-4.12.jar`
   (b) `hamcrest-all-1.3.jar`
   (c) `testfx-core-4.0.4-alpha.jar`
   (d) `testfx-junit-4.0.4-alpha.jar`
   (e) `testfx-legacy-4.0.4-alpha.jar`
   (f) `guava-19.0.jar`
7. nyní bychom měli vidět přidané `JAR` soubory v seznamu

Pokud používáte program NetBeans, postupujte podle následujícího návodu.

1. v `Package Explorer` klikneme pravým tlačítkem myši na náš vytvořený projekt, který obsahuje aplikaci pro otestování
2. v zobrazených možnostech klikneme na `Properties`
3. vlevo klikneme položku `Libraries`
4. dále klikneme na tlačítko `Add JAR/Folder`
5. ze složky vybereme všechny `JAR` stažené knihovny
   (a) `junit-4.12.jar`
   (b) `hamcrest-all-1.3.jar`
   (c) `testfx-core-4.0.4-alpha.jar`
   (d) `testfx-junit-4.0.4-alpha.jar`
   (e) `testfx-legacy-4.0.4-alpha.jar`
   (f) `guava-19.0.jar`
6. nyní bychom měli vidět přidané `JAR` soubory v seznamu

A.3.2 Testování

Třída `FXMLAppControllerTest` obsahuje spoustu testů. V testovacích metodách jsou okomentovány jednotlivé kroky, co která metoda dělá. Popis je vždy pouze u prvního výskytu dané metody. Pro začátek si nejprve jednotlivé testovací metody projděte a pročtěte komentáře, abyste porozuměli používání nástroje TestFX. Detailněji je nástroj TestFX popsán v souboru `TestFX.pdf`.

V této třídě je vynechaných celkem 5 testovacích metod, u kterých je napsán přesný popis, pomocí kterého byste měli doplnit tělo testu. Tyto metody testují funkčnost určitých akcí v aplikaci, které jsou schválně naimplementovány chybně. Výsledky těchto testů by měly skončit chybou, na rozdíl od ukázkových testů, které skončí úspěšně. Níže jsou

vyjmenovány testy, které je potřeba doplnit. Podrobnější popis, jak máte postupovat při tvoření jednotlivých testů, je popsán v komentáři přímo v testovací metodě.

`testIncompleteAddEntry2()`

Tento test testuje, zda po vyplnění pouze textového pole pro přihlašovací jméno uživatele a kliknutí na tlačítko pro přidání účtu, nedojde k přidání účtu do tabulky.

`testEditTitle()`

Zde se testuje, jestli se změní název účtu po jeho přepsání a kliknutí na tlačítko pro uložení. Tato změna se má projevit v tabulce v 1. sloupci a textovém poli, kde se název účtu edituje. Proto je potřeba kliknout na jinou položku v tabulce a pak na náš editovaný účet, aby se aktualizovaly data v textových polích.

`testShowNotificationRemove()`

Po smazání účtu z tabulky se má vypsat informační hláška, že došlo skutečně ke smazání účtu. Tento test netestuje, jestli se smazala položka z tabulky, to se provádí v jiném testu.

`testChangePasswordValid()`

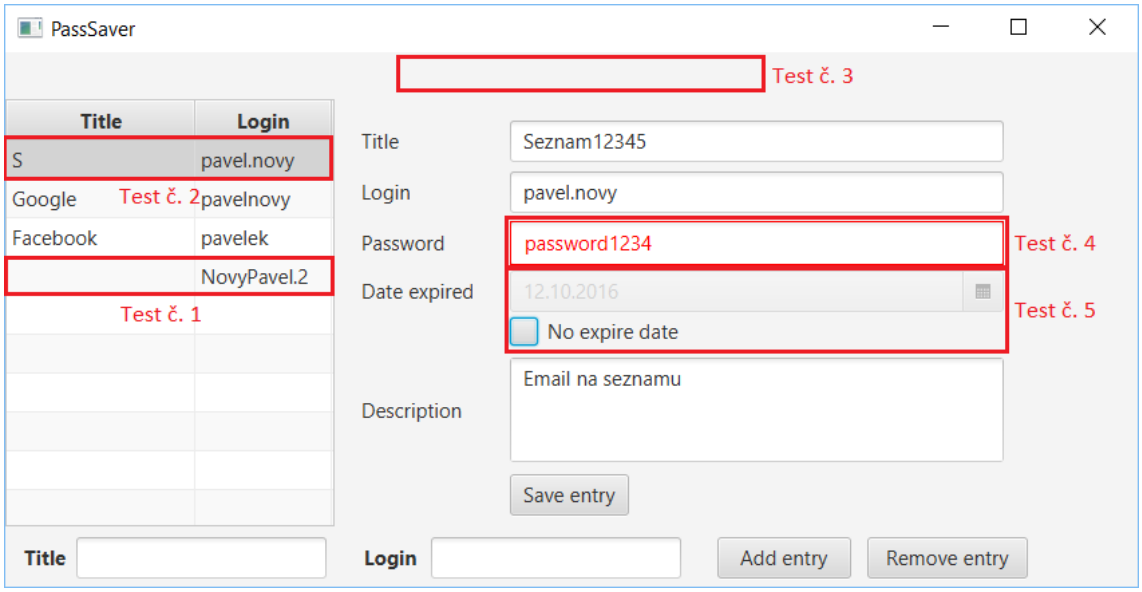
Při psaní hesla v textovém poli je heslo validováno, aby mělo minimálně 6 znaků. Pokud má méně, má zobrazený text a rámeček pole červenou barvu. Standardně má text černou barvu a rámeček je bez barvy. V testu se testuje, zda má textové pole správné styly, pokud zadáte heslo s minimální délkou 6 znaků.

`testEnableDate()`

Heslo účtu může po určité době vypršet. Aby bylo textové pole pro datum povolené (šlo do něho zapisovat), tlačítko `No expire date` nesmí být zaškrtnuté. Zde se testuje, jestli se po kliknutí na zaškrtnuté tlačítko povolí textové pole pro datum.

B APLIKACE S CHYBAMI

Screenshot testované aplikace s vyznačenými oblastmi, kde se vyskytly chyby po provedení konkrétní akce. U každé oblasti je popisek s číslem testu, který odpovídá testu ze zadání pro studenty.



Obr. B.1: Screenshot aplikace s vyznačenými chybami pro testy