VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

KOMPONENTA JAVA SWING ŘÍZENÁ POMOCÍ CSS

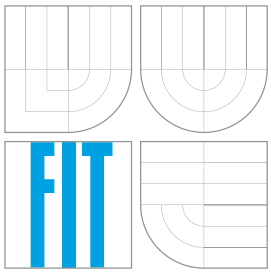
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MATĚJ HVĚZDA

BRNO 2014

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

KOMPONENTA JAVA SWING ŘÍZENÁ POMOCÍ CSS

CSS-DRIVEN JAVA SWING COMPONENT

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — MATĚJ HVĚZDA
AUTHOR

VEDOUCÍ PRÁCE — Ing. RADEK BURGET, Ph.D.
SUPERVISOR

BRNO 2014

Zadání bakalářské práce/15863/2013/xhvezd03

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2013/2014

Zadání bakalářské práce

Řešitel: **Hvězda Matěj**
Obor: Informační technologie
Téma: **Komponenta Java Swing řízená pomocí CSS**
CSS-Driven Java Swing Component
Kategorie: Uživatelská rozhraní

Pokyny:

1. Seznamte se s platformou Java a aplikačním rozhraním Swing pro tvorbu uživatelského rozhraní.
2. Seznamte se s jazyky HTML, CSS a architekturou zobrazovacího stroje CSSBox.
3. Navrhněte komponentu Swing umožňující zobrazit obsah s vizuálními vlastnostmi definovanými pomocí jazyka CSS.
4. Po konzultaci s vedoucím implementujte navržené řešení s využitím zobrazovacího jádra CSSbox.
5. Implementujte jednoduchou aplikaci demonstrující funkčnost komponenty.
6. Zhodnoťte dosažené výsledky a navrhněte další možná rozšíření.

Literatura:

- Bos, B. et al.: Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification, W3C Candidate Recommendation 19 July 2007
- Staníček, P.: CSS Kaskádové styly, Computer Press, 2003, ISBN 80-7226-872-4
- Dokumentace projektu CSSBox, http://cssbox.sourceforge.net

Při obhajobě semestrální části projektu je požadováno:

- Body 1 až 3

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Burget Radek, Ing., Ph.D.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2013
Datum odevzdání: 21. května 2014

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Tato práce je zaměřená na vytvoření komponenty Java Swing, která zobrazí obsah HTML dokumentu na základě kaskádových stylů (CSS). Důvod k vytvoření takové komponenty je umožnění vývojářům tvořit uživatelské rozhraní pomocí CSS v Javě bez nutnosti toho, aby znali rozhraní Java Swing. K syntaktickému rozboru HTML a CSS je použit zobrazovací stroj *CSSBox*, který umožní získat veškeré potřebné informace ke správnému zobrazení dokumentu. Komponenta je implementována pomocí komponent Swingu kupř. element HTML je implementován na základě komponenty `JPanel`, a i všechny editovatelné prvky ve formulářích jsou implementovány vhodnými komponentami Java Swing. U komponenty řízené pomocí CSS je možnost měnění její velikosti, přístup k prvkům formuláře, nastavení odkazů a jejich vzhledu, aj.

Abstract

This bachelor's thesis is focused on creating Java Swing component, that can display content of HTML document based on cascade style sheets (CSS). Purpose for creating this component is enabling developers to create graphic user interface with help of CSS in Java without knowledge of Java Swing. For parsing HTML document and CSS is used rendering engine *CSSBox*. This engine enables obtaining all the right informations about HTML document and its styles for displaying this document. Java Swing API was used for implementing this component, for example HTML element is implementented based on `JPanel` and all editable elements of form are implemented by fitting Swing component. With component Java Swing driven by CSS is possible to change her size, access to form's editable elements, set up links and their look, etc.

Klíčová slova

Java, Java Swing, CSS, HTML, grafické uživatelské rozhraní, kaskádové styly, JSON, komponenta

Keywords

Java, Java Swing, CSS, HTML, graphic user interface, cascade style sheets, JSON, component

Citace

Matěj Hvězda: Komponenta Java Swing řízená pomocí CSS, bakalářská práce, Brno, FIT VUT v Brně, 2014

Komponenta Java Swing řízená pomocí CSS

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Radka Burgeta, Ph.D.

.....
Matěj Hvězda
19. května 2014

Poděkování

Chtěl bych především poděkovat vedoucímu práce Ing. Radku Burgetovi, Ph.D. za jeho odbornou pomoc a rady při psaní této práce.

© Matěj Hvězda, 2014.
Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

Kapitola 1

Úvod

Tvorba uživatelských rozhraní se pouvažuje za jednu z nejnáročnějších prací k vytvoření, kde mj. věci se musí brát do úvahu i platforma, kde toto rozhraní má být představeno (implementováno). Každá platforma může mít jiný implementační jazyk, kdežto programátor tento jazyk znát nemusí, ale zná jazyk HTML a CSS, protože tyto technologie jsou dobře známy.

Dnes se většina uživatelských rozhraní už definuje pomocí CSS (kaskádových stylů), a tento postup definice vzhledu se v dnešní době stává standardem. A tedy snaha této práce je umožnit tvorbu uživatelských rozhraní (GUI) v platformě Java pomocí rozhraní Java Swing, bez toho aniž by uživatel musel znát samotné rozhraní Java Swing, ale stačí mu umět definovat HTML dokument, kde jeho vzhled je popsán kaskádovými styly (CSS). Tudíž si jenom pomocí HTML stačí popsat strukturu dokumentu a vzhled tohoto dokumentu definovat kaskádovými styly, tento způsob je dnes už běžně zažitý. Dále jen vytvořený dokument HTML vložit na vstup stroje (technologie), který na výstupu vytvoří definovanou reprezentaci za pomocí rozhraní Java Swing.

Stroj k tvoření HTML dokumentu by měl být další komponenta Swingu, která bude vytvářet z elementů HTML a jejich CSS stylů její vzhled. Aby komponenta měla přistup k elementům HTML a jejich kaskádovým stylů (CSS), aby na základě nich mohla tvořit vzhled, tak k tomu bude sloužit zobrazovací stroj *CSSBox*. Pomocí stroje *CSSBox* je možno získat velikost, barvu pozadí, textu elementu a všechny další potřebné informace ke správnému zobrazení HTML dokumentu.

Jak bylo výše zmíněno kaskádové styly (CSS) definují vzhled hypertextových dokumentů (HTML), čehož lze využít ke tvorbě různých vizualizací informací. Těchto vlastností využívá i komponenta. Komponenta by měla být obecná, univerzální (odpovídat CSS stylu). Tím chci říct, že by měla vizualizovat jakýkoli element HTML, u kterého je vzhled definován CSS stylem.

Jak by se dala komponenta využít samozřejmě záleží na uživatelově představivosti, ale komponenta je vytvořena na základě idei. Idea je taková, že by mohla být komponenta použitelná k vytvoření grafického uživatelského rozhraní pomocí HTML a CSS v Javě za pomocí Swingu, ale bez znalostí rozhraní Java Swing. Grafické uživatelské rozhraní, které lze ovládat podle grafických prvků (např. ovládacích prvků formuláře), které jsou v tomhle případě tlačítka, odkazy, textové pole, atd. U těchto grafických prvků je vzhled definován pomocí CSS a tím je myšleno to, že komponenta je řízena CSS.

Zpráva začíná teorií nad použitými technologiemi ve druhé kapitole, kde je popsáno z jakých teoretický znalostí se dospělo k vytvoření komponenty. Jako například, v kterém jazyku se komponenta implementovala a jaké různé vlastnosti jazyka se využili. V této

kapitole je také zmíněno něco o značkovacím jazyku HTML a kaskádových stylech (CSS). V podkapitole CSS je více řečeno o zobrazovacím stroji *CSSBox* a také o tom jak se definuje takový CSS styl. V podkapitole konceptuální návrh popisuji jak jsem komponentu navrhl a jak budu postupovat k její implementaci. Potom navazuji kapitolou implementace návrhu, kde detailněji popisuji implementaci konceptuálního návrhu a taky popisuji demonstrační aplikaci, která má ukázat funkčnost komponenty. V Kapitole návrhy k rozšíření ukazuji, jak by se dalo pokračovat v implementaci komponenty, neboli jak ji dále rozšířit. A v poslední řadě je poslední kapitola (závěr), kde zhodnocuji práci nad vytvořením komponenty.

Kapitola 2

Aplikované technologie

V této kapitole stručně popíši teoretické znalosti a technologie, které budou tvořit základ k následujícím kapitolám. Záčatek teorie je jazyk, ve kterém se komponenta bude implementovat. Bude i zmíněno, které rozhraní se využilo při tvorbě komponenty. A taky, které časti jazyka se využilo při tvorbě. Dále je zmíněno něco o historii platformy java, ale velmi stručně pro zajímavost. Po jazyku bude něco zmíněno o značkovacích jazycích a hlavně o HTML z rodiny značkovacích jazyků. HTML totiž tvoří kostru, jak bude komponenta vypadat. A CSS, které jedna s posledních podkapitol, a které určují vzhled HTML dokumentu a bude řídit komponentu. A úplně na konci je něco málo o zobrazovacím stroji *CSSBox*, který je použit pro rozbor HTML a CSS a taky je zmíněn JSON řetězec.

2.1 Platforma Java

Většina textu v této sekci je pořízená z literatur [1, 2]. Komponenta je implementována v jazyce Java za použití rozhraní Java Swing. Vlastně je vytvořená další komponenta rozhraní Java Swing. O rozhraní Java Swing se dozvíte více v podsekci 2.1.2.

Java je univerzální programovací jazyk a lze ji zařadit mezi tzv. třetí generaci programovacích jazyků (3GL) [1]. Java je programovací jazyk, který je považován za objektový nebo objektově orientovaný *(object-oriented)*. Jazyku Java vložil základní formu objektově orientovaný jazyk C++, ale dalo by se říct, že předchůdcem Javy je jazyk C. Avšak je patrné z konceptuálního hlediska, že při vývoji Javy dokázali návrháři odstranit z originálního C++ řadu chyb a vznikl tak jazyk „čistší“ bez většiny triků typických pro C++ [1].

U jazyku Java je postačujcící napsat, přeložit a odladit program pouze jednou a poté je na kterékoliv platforem spustitelný (jsou to např. 32 bitová MS Windows, Linux, MAC OS, Solaris a další). Tato vlastnost Javy je velice oceňovatelná, a tak je jednou z největší předností tohoto jazyka, protože Java má plnou přenositelnost programů na jinou platformu bez nutného překladu na dané platfromě. Této přenositelnosti se dosahuje pomocí bajtkódu, jehož interpretace je potom úkolem speciálních programů, nazývaných souhrně ***Java platforma***, které jsou ovšem pro tuto platformu předem připraveny [1].

Prvky platformy Java:

- Jedním z hlavních prvků implementace platformy je virtuální počítač - *Java Virtual Machine* (JVM), který zajišťuje běh přeložených javovských aplikací. Aplikace tedy neběží přimo ve strojovém kódu počítače, na němž jsou spouštěny, ale jen díky JVM [1].

- Překladač, debugger a stroj pro generování dokumentace, všechny tyto nástroje jsou dalšími důležitými prvky platfromy.
- *Java Core API*, prvek k implementaci tříd.

Java Core API API (*Application programming iterface*) je zkratka, s kterou se při využívání jazyku Java potkávate často a když program využívá metody z API, tak kód programu není součástí programu, ale je součástí API. Pod API se skrývá značné množství knihovních tříd až několik tisíc, které jsou považovány za standartní, čímž se myslí, že se musí vyskytovat v každém prostředí, kde se Java používá [1].

JIT kompilátor Kompilované jazyky mají výhodu nad interpretovanými, protože jsou rychlejší. Java tuto nevýhodu částečně řeší pomocí tzv. *JIT kompilátorů* (*Just in time*), které v době zavádění programu z disku do paměti počítače jej přeloží *on-line* do strojového jazyka konkrétního počítače, čímž z něj prakticky vyrobí v paměti spustitelný program. Ten pak běží stejnou rychlostí, jako kterýkoliv jiný program kompilovaný třeba v jazyku C [1].

Rozhraní

V jazyku C++ je k dispozici vícenásobná dědičnost, která nám umožňuje dědit více tříd najednou, ale tato vlastnost nám přinese mezi výhodami i různé problémy. Z toho plyne rozhodnutí autorů jazyku Java, že vícenásobnou dědičnost podporovat nebudou, protože byli přesvědčeni, že to přináši více problémů než výhod. V Javě je částečně nahrazena tato vlastnost používáním rozhraní.

Rozhraní definuje soubor metod, které ale v něm nejsou implementovány, tj. v deklaraci rozhraní je pouze hlavička metody, stejně jako je to u abstrakní metody. Třída, která toto rozhraní implementuje (tj. jakoby dědí), musí implementovat všechny jeho metody [2].

Vlastnosti rozhraní:

- rozhraní nemůže deklarovat žádné proměnné,
- třída může implementovat více než jedno rozhraní,
- rozhraní je nezávislé na dědičné hierarchii tříd.
- Chceme třídě pomocí implementace rozhraní vnutit zcela konkrétní metody.
- Vidíme jednoznačně podobnosti v různých třídách, ale začlenit tyto podobnosti do vlastností tříd pomocí dědění by vyžadovalo vytvořit jejich předka.

2.1.1 Zajímavosti z historie

Od roku 1991 vyvíjela firma Sun Mircrosystems programovací jazyk na principech C a C++ pro „vestavěné systémy“. Jazyk měl původně název Oak (dub), podle dubu, který stál před oknem pana Goslinga, vedoucího týmu. Posléze se zjistilo, že již programovací jazyk s tímto názvem existuje, takže vývojová skupina hledala název nový a po návštěvě firemního bufetu padl návrh Java, což znamená v americkém slangu „kafe“ [1].

Java na rozdíl od řady programovacíh jazyků nebyla ani dílem akademického výzkumu a vývoje, ani živelného „nabalování“ se standardizací ex-post. Vznikla jako profesionální dílo poměrně přísně řízeného komerčního vývoje [2]. Pokud jde o standardizaci, je třeba rozlišovat standard definující danou javovou platformu od konkrétní implementace dané platformy. Standard sám je dílem vývoje řízeného společností. Předepisuje závazně syntaxi

a sémantiku jazyka, chování běhového prostředí a strukturu *Java Core API*. Ke standardu bývá společností a dalšími výrobci poskytována implementace dané platformy.

2.1.2 Rozhraní Java Swing

Většina textu z této podsekce je získana z dokumentace rozhraní [3], dále je text inspirován z článku [4] a manuálu [5]. Swing je knihovna (rozhraní) uživatelských prvků na platformě Java. Swing patří do JFC (*Java Foundation Classes*). JFC je API, které poskytuje uživatelské grafické rozhraní (GUI) pro javovské programy. Hierarchie tříd grafických komponent Swingu je založena na rodičovských třídách z AWT, jak je vidět na obrázku 2.1. Základní nadtřídou všech grafických komponent je třída `Component`. Instance třídy `Component` představují objekty, které mají svoji grafickou reprezentaci a mohou být zobrazeny na obrazovce a integrovat s uživatelem. Z toho plyne, že všechny prvky, které Java vykreslí na obrazovku, je instancí třídy `Component`. Od třídy Component je odvozena třída `Container` (kontejner). Kontejner je důležitá grafická komponenta, neboť dokáže v sobě udržet a kreslit ostatní grafické Swingovské komponenty. Jakým způsobem jsou komponenty vykresleny, uspořádany, pozicovány v kontejneru, záleží na tzv. správce rozložení (*Layout Manager*), ale je možnost i absolutního pozicování bez správce rozložení.

U rozhraní Swing je to právě třída `JComponent`, která dědí od třídy `Component`. Tato třída je nadtřídou každé swingovské komponenty. `JComponent` je podtřídou `Container`, jak vidět na obrázku 2.1. Takže je logické, že všechny swingovské komponenty v sobě mohou obsahovat další komponenty[1].

Architektura Swingu:

Ve Swingu má každá komponenta svůj model v podobě rozhraní, který je na dané komponentě nezávislý. Swing poskytuje základní implementaci modelu jako standard, ale uživatel si může implementovat komponentu s jiným modelem. Toto umožňuje uživateli změnit chování grafické komponenty, ale využívat její vzhled.

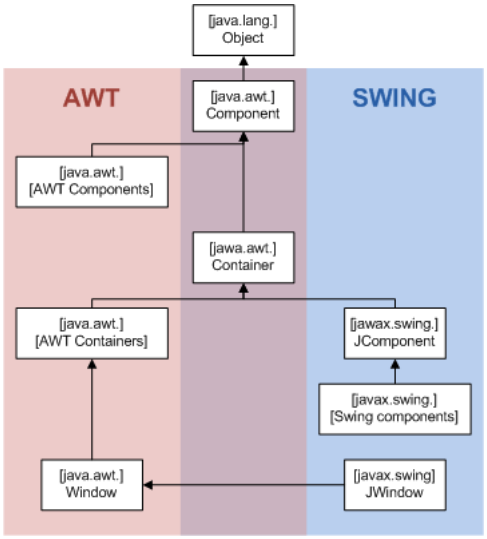
- **Základ:** Rozhraní Swing je platformě nezávislý, protože je celé napsané v Javě.
- **Rozšířitelnost:** Uživatelé můžou přepsat základní implementaci komponent vlastní implementací, kde přitom využívají javovské dědičnosti.
- **Pružnost/flexibilita:** Vzhled swingovských komponent je složení stadardním souhrnem prvků jako jsou okraje, pozice, vykreslování a dalších. A uživatelé si všechny tyto prvky můžou upravovat podle sebe.

Vykreslování v rozhraní Swing

Rozhraní Swing rozšiřuje dál základní vykreslovací model rozhraní AWT a maximilizuje jeho výkon a zlepšuje spolehlivost [4]. Jako AWT, rozhraní Swing podporuje opakované vykreslování a použití metody `repaint` (překreslení) ke spouštění aktualizací. Navíc Swing poskytuje vestavěnou podporu pro dvojitou vyrovnávací pamět a taky změnu k původní architektuře jako jsou např. okraje, atd. Rozhraní Swing taky poskytuje správce vykreslování (*RepaintManager API*) pro aplikace, které chtějí malovací mechanismus více upravit.

Podpora dvojité vyrovnávací paměti je jedna z nejcennějších vlastností rozhraní Swing, která je přímo vestavěná v technologii. Při dvojité ukládání do vyrovnávací paměti jsou

[1] Swingovské komponenty mají před svým názvem písmeno J, aby se daly odlišit od svých protějšků z AWT.



Obrázek 2.1: Hierarchie tříd AWT a Swingu[2].

všechny operace kreslení nejprve vykresleny do vyrovnávací paměti namísto plochy kresby na obrazovce. Po dokončení všech operací kreslení je obsah vyrovnávací paměti zkopírován přímo na povrch výkresu s ní spojené.

Swing poskytuje vlastnost `doubleBuffered` v balíku `javax.swing.JComponent` a metody s ní spojené `isDoubleBuffered` a `setDoubleBuffered`. Implicitně je tato vlastnost nastavena. Swing má však další nastavitelné vlastnosti ke zlepšení výkonnosti vykreslovacího algoritmu. Jsou to průhlednost (*Transparency*) k umožnění kreslení cokoliv je pod komponentou prvně. Ve swingu k tomu slouží vlastnost `opaque` ve stejném balíku a k její měnitelnosti slouží metody `isOpaque` a `setOpaque`. Další vlastnosti je překrývání komponent (*Overlapping components*), pomocí které lze nějakou komponentou překrývat druhou.

Operace táhnutí objektu myší (*drag and drop*)

V počítačovém uživatelském rozhraní je operace *drag and drop* něco jako stlačení na nějaký objekt a táhnutí tohoto objektu na jinou pozici [5]. Obecně tuto operaci může vyvolat různé operace nebo to může vytvořit vztah mezi dvěma objekty. *Drag and drop* operace je jedna z nejviditelnějších operací uživatelských operací. Umožní uživatelům dělat komplexní věci intuitivně.

Naprostá většina tříd zahrnujících operaci *drag and drop* v swingovském nástroji je ohromující. Komponenta, u které začíná operace *drag and drop* musí mít registrovaný objekt `DragSource`. A `DropTarget` je objekt, který je zodpovědný za akceptování této operace. Data, které můžou být takhle přeneseny můžou být různé typy. Objekt `DataFlavor` poskytuje informace o datech, které můžou být takhle použity. Několik swingovských komponent má tuto operaci už vestavěnou. A pokud ne, tak programátor musí celou operaci vytvořit .

[2]Obrázek převzat od Závěrka J. z internetové stránky http://commons.wikimedia.org/wiki/File:AWTSwingClassHierarchy.png

Java Swing události

Události jsou důležitou části každé GUI aplikace. Všechna GUI aplikací je řízena událostmi [5]. Aplikace reaguje na různé typy události během jejího běhu. Většina událostí je generována uživatelem této aplikace, ale můžou být generovány i jinak. V událostním modelu jsou tři složky a to událostní zdroj, událostní objekt a událostní posluchač. Událostní zdroj je objekt, kterému se změní stav. Událostní objekt zapouzdřuje změny, které nastanou u událostního zdroje. Událostní posluchač je objekt, který chce vědět co se stalo. Objekt, který reprezentuje událostní zdroj, sdělí vykonání události událostnímu posluchači.

Zpracování událostí v rozhraní Java Swing je velmi mocné a flexibilní. Java používá model oznámení (*Event delegation model*), kde specifikujeme objekty, které chceme, aby byly oznámeny, když jistá událost nastane. Když něco nastane v aplikaci, tak událostní objekt je vytvořen např., když stiskneme tlačítko nebo označíme položku v listu. Existuje několik druhů událostí kupř. *ActionEvent*, *TextEvent*, *FocusEvent*, *ComponentEvent*, atd.

Java Swing *layout* plánování (správce rozložení)

Rozhraní Swing má dva druhy komponent. Jsou to kontejnery a jejich potomci [5]. Kontejnery sdružují jejich potomky do patřičného rozložení (*layout*). K vytváření rozložení se používají správce rozložení (*layout manager*). Správce rozložení jsou jedny z nejnáročnější částí GUI programování. V Javě existují mnoho různých správců rozložení kupř. *BoxLayout*, *FlowLayout*, *GridBagLayout*, *GridLayout*, atd.

Souběžnost (*Concurrency*) v Java Swing

Uživatelé počítačů berou jako samozřejmost, že jejich operační systém, může dělat více věcí najednou. Předpokládají, že můžou pracovat s konsolí, zatímco ostatní aplikace stahují soubory, tisknou nebo přehrávají audio. Např. aplikace pro přehrávání audio musí zároveň číst audio soubor ze sítě, rozbalit ho, umožnit přehrání a zobrazit všechno na monitor. Software, který dokáže tyto věci splnit je *concurrent* software.

Opatrné použití *concurrency*[3] ve Swingu je zejména důležité. Dobře napsaná swingovská aplikace používá *concurrency* k tvoření uživatelských rozhraní, aby aplikace nikdy „nezamrzla“[4].

Swingovský programátor pracuje s následujícími vlákny (threads):

- *Initial threads*, vlákna, která spouští inicializaci aplikace. Ve Swingu tyto vlákna toho nemají moc na práci. Jejich podstatná práce je vytvořit `Runnable` objekt, který inicializuje GUI a naplánuje tento objekt ke spuštění v *event dispatch thread*. Po vytvoření GUI, je GUI řízeno pomocí událostí. Plánovaní tvoření GUI je realizováno voláním následujících metod:
  - `javax.swing.SwingUtilities.invokeLater`, jednoduše naplánuje úkol a vrací se,
  - `javax.swing.SwingUtilities.invokeAndWait`, naplánuje úkol a čeká, až se dokončí před vrácením se.

  Obě metody mají jediný parametr, objekt `Runnable`, který definuje nový úkol.

[3] *Concurrency* - souběžný běh.
[4] Aplikace je vždy responsivní k práci bez ohledu na to co dělá.

- *Event dispatch thread*, vlákno, kde všechny události jsou spouštěny. Většina kódu, který spolupracuje se Swingem se spouštějí na tomto vlákně, je to nutné, protože většina swingovských objektů nejsou „*thread safe*“.

Programátor nemusí tyto vlákna vytvořit, jsou mu poskytnuty rozhraním Java Swing. Programátor má za úkol využít tato vlákna k vytvoření responsivního uživatelského rozhraní.

Worker Threads **a** *SwingWorker* Když swingovská aplikace potřebuje spustit dlouho trvající úkol, tak většinou použije jeden z vláken *worker threads* (*background threads*). Každý úkol běžící ve *worker* vlákně je reprezentován objektem `javax.swing.SwingWorker`.

`StringWorker` poskytuje několik komunikací a funkcí:

- `StringWorker` podtřída může definovat metodu `done`, která se automaticky spustí v hlavní vlákně (*event dispatch thread*) po dokončení úkolu, vykonávající ho se na pozadí.
- `StringWorker` implementuje rozhraní `java.util.concurrent.Future`, které umožní úkolům na pozadí, vrácení hodnoty do jiného vlákna. Umožňuje také rušení ukolů na pozadí a zjištění, jestli byly dokončeny.
- Úkol na pozadí (background task) může poskytnout nynější výsledky pomocí volání metody `SwingWorker.publish`[5], více o úkolech na pozadí v dokumentaci [3].

Těžké, lehké komponenty (*Lightweight UI*)

AWT bylo založeno jako rozhraní mezi Javou a grafickým API platformy. Tzn., že systémem byly přímo kresleny komponenty a měly tedy nativní vzezření. Tyto nativně vykreslené prvky se nazývají „těžké“ (*heavyweight*), protože měly své vlastní nativní neprůhledné okno v systému. Na systém se spoléhaly i v dalších věcech, např. zajišťování pozice na ose Z[6].

Swing má jiný přístup, a to takový, že každá komponenta si sama zodpovídá za svůj vlastní vzhled a neodkazuje se na systém. Každá komponenta sama definuje, jak vypadá, a na požádání se vykreslí na obrazovku. Samotné kreslení se odehrává přímo v Javě, a operačnímu systému se pouze předá hotový vzhled k vykreslení. Proto se swingovské komponenty nazývají „lehké“ (*lightweight*).

Nastavitelnost vzhledu

Operační systém nemá kontrolu v rozhraní Java Swing nad tím, jak bude komponenta vykreslena, a tak je možné ve Swingu změnit vzhled, chování a některé další charakteristiky/vlastnosti všech komponent pomocí tzv. Look and Feel („Vzhled a Pocit“). „Vzhled a Pocit“ je soubor delegátů, které určují způsob vykreslení komponenty např. pro tlačítko `JButton` je to třída ButtonUI. Když se má komponenta vykreslit, tak se zjistí způsob vykreslení od vhodného delegáta, a vykreslí se podle toho. „Vzhled a Pocit“ je dokonce měnitelný za běhu aplikace, tudíž je možné během běhu aplikace úplně změnit vhled aplikace, protože se pouze změní třída zodpovědná za poskytování informací o vzhledu.

[5] Volání této metody způsobí volání `SwingWorker.process` v *event dispatch thread*.
[6] Hodnota, která rozhoduje, která komponenta je v popředí, a která v pozadí.

2.1.3 Reflexe v platformě Java

Podklady pro tuto podsekci byly zjištěny z dokumentace Java [6]. Termínem reflexe označujeme schopnost získat za běhu informace o typu objektu, s nímž program pracuje.

Všechny potřebné informace o každém typu získáme z jeho class-objektu, instance třídy `java.lang.Class`. Class-objekt je vytvářen instancí třídy *ClassLoader* při zavádění daného typu do paměti. Java je schopná získat za běhu programu od objektu kompletní informaci o jeho typu, jako např. název typu, druh typu, konstruktory, metody, atd. S těmito informacemi zjištěnými za běhu se dá pracovat, voláním konstruktorů, metod a další.

Využití:

- Vzdálené zpracování - metoda musí umět zjistit, zda obdržela po síti parametr požadovaného typu.
- Na této schopnosti je založená řada aplikací, které za běhu přizpůsobují své chování objektům s nimiž pracují.
- Dynamické chování programu.

2.2 Značkovací jazyky a HTML

Základní informace i text z této sekce je použito z opory předmětu IIS [7]. Značkovací jazyky (*markup laguages*) jsou speciální typ programovací jazyků, sloužící zejména pro popis dokumentů. Ve značkovacích jazycích převládá nestrukturovaný text, do kterého jsou vloženy strukturované úseky pomocí značek.

SGML (*Standard Generalized Markup Language*) je univerzální značkovací metajazyk, který umožňuje definovat značkovací jazyky, které se stanou jeho podmnožiny. Do SGML rodiny patří právě mj. i jazyk HTML.

V rodině SGML vznikl nejdříve metajazyk SGML, pro deklaraci různých typů dokumentů, a právě jeho nejznámější aplikací, je jazyk pro hypertextových dokumentů HTML [7]. Existuje několik základních vlastností, které odlišují SGML od ostatních značkovacích jazyků:

- preferuje univerzální značkování,
- umožňuje deklarovat vlastní typy dokumentů,
- je nezávislý na platformě, kde je program v tomto jazyce používán.

Dalším důležitým pojmem je **popisné značkování**, popisné značkovací systémy používají značky, které poskytují jména pro označení části dokumentu. Značky jako `<div>` nebo `</ul>` jednoduše identifikují části dokumentu a prohlašují, že následující značka je odstavec nebo konec seznamu v dokumentu. U procedurálních značkovacích systémů je to trochu jinak, ony definují, jaká procedura se má provést na daném místě v dokumentu.

Dále SGML zavádí pojem typu dokumentu a s ním definici typu dokumentu (*DTD - Document Type Definition*). Dokumenty jsou uvažovány jako typované, podobně jako jiné objekty zpracované informačními technologiemi. Dokumenty mají svůj typ definovaný pomocí DTD. Vzniká pojem metadat v SGML, metadata jsou popsána jejich částmi a strukturou těchto částí [7]. SGML poskytuje datovou nezávislost a jeho obsah má textový tvar. Textový tvar byl zvolen kvůli tomu, aby dokumenty mohly být přemisťovány z jednoho hardwarového nebo i softwarového prostředí do jiného bez ztráty dat.

2.2.1 HTML

Zkratka HTML znamená *HyperText Markup Language*, a je to značkovací jazyk pro hypertext. Jeden z hlavních jazyků pro vytváření stránek v systému *World Wide Web*, který umožňuje umístit dokumenty na Internet. HTML je typem dokumentu SGML, kde je značkám přiřazena sémantika. HTML obsahuje mimo vlastní definici značkovacího jazyka mnoho dalších formálních jazyků, a jsou to např.:

- skriptovací jazyky a převážně JavaScript,
- kaskádové styly (CSS),
- případně jiné méně rozsáhlé jako jsou různá kódování.
- Direktivy, tj. začínají znaky <!, jsou určeny pro prohlížeč, označují komentáře v dokumentu

Jazyk HTML je definován velkou množinou značek (*tagů*) a jejich vlastností (*atributů*) definovaných pro danou verzi HTML. Mezi značky se uzavírají části textu dokumentu a tím se určuje sémantika obsaženého textu [7]. Tvar názvů jednotlivých značek a jejich vlastností jsou úhlové závorky `<a>`. Část dokumentu tvořená otevírací značkou a nějakým obsahem a odpovídající ukončovací značkou tvoří element nebo taky prvek dokumentu. Značky jsou většinou párové, přičemž koncová značka je téměř shodná se značkou otevírací, s tím rozdílem, že má před názvem znak lomítko, např. `<a> ahoj </a>`. Některé značky jsou nepárové, tj. nemají žádný obsah a nepoužívají koncovou značku a je to např. `<br>` pro zalomení řádku.

Struktura HTML dokumentu:

- *Doctype*, úvod do dokumentu sdělující prohlížeči, že otevřel dokument HTML a jakou verzi HTML. Je povinný a zapisuje se `<!DOCTYPE html verze>`.
- Kořenový element, prvek html (značky `<html>` a `</html>`), který reprezentuje celý dokument.
- Hlavička dokumentu, prvek head (značky `<head>` a `</head>`), který obsahuje metadata, která se vztahují k celému dokumentu. Definuje např. kódování, název dokumentu, autora, popis, klíčová slova, titulek dokumentu nebo kaskádové styly, aj.
- Tělo dokumentu, prvek body (značky `<body>` a `</body>`), který zahrnuje vlastní obsah dokumentu.

V HTML jsou různé **druhy značek** a jsou to:

- Strukturální značky rovrhují strukturu dokumentu.
- Popisné (sémantické) značky popisují povahu obsahu prvku.
- Stylistické značky popisují vzhled elementů, dokumentu.
- Značka pro skriptovací jazyk.

Příklad dokumentu HTML:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<HTML>
  <HEAD>
    <TITLE>dokument</TITLE>
  </HEAD>
  <BODY>
    <P>...text...
    <P> text </p>
    <P> dlouhy
       text </EM>opět začíná text..
  </BODY>
</HTML>
```

Formulář

Formulář je párová značka `FORM`, který uzavírá skupinu ovládacích polí do jednoho formuláře [7].

Ovládací pole:

- **INPUT**, definuje libovolný z prvků formuláře. Bez atributů vytváří implicitní textové pole.
  - typ **text** je typ, který vytváří textové pole.
  - typ **password** vytváří textové pole, ale jako heslo, tj. znaky jsou skryty.
  - typ **radio** vytváří vybírací tlačítko (radio button).
  - typ **checkbox** vytváří zaškrtávací tlačítko.
  - typ **submit reset**, **button**, tyto typy vytvářejí tlačítka, ale na web stránce mají jinačí funkčnost.
- **SELECT** prvek vytváří dropdown menu. Bez vnitřních prvků a atributů bude toto menu prázdné.
- **TEXTAREA** prvek vytváří pole pro zadávání víceřádkového textu. Bez atributů vytváří standartní velikost okénka a jeho počátečním obsahem je obsah těla prvku.
- **LABEL** prvek bez vnitřních prvků a atributů nemá speciální sémantiku, slouží jako text ve formuláři.
- **BUTTON** prvek vytváří tlačítko, které vizuálně reaguje na stlačení. Bez atributů vytváří tlačítko s popisem v těle prvku.

2.3 CSS (Kaskádové styly)

Většina informací a textu z této sekce byli inspirovány z literatury [8] a normy [9]. Jak využít kaskádové styly je nastudované z normy [9]. Kaskádové styly (*Cascading Style Sheets*) jsou nadstavbou jazyka HTML určenou k definování vzhledu např. uživatelského rozhraní, WWW stránek (elementů v dokumentu HTML), atd. V této technické zprávě budeme uvažovat verzi kaskádových stylů *CSS2.1*. Každý z elementů HTML má základní sémantiku a je dána jeho základními vlastnostmi a jeho atributy. Tyto základní atributy se nezaměřují

na vzhled, ale na shromáždění informací o elementu v dokumentu, přitom se předpokládá, že o vzhled se stará prohlížeč nebo jiná technologie.

V roce 1998 vznikl návrh normy CSS2. Jazyk CSS2 je poměrně rozsáhlý, a jeho úplnou definici je třeba hledat v normě [9].

CSS styly umožňují přiřadit každému elementu v HTML určité vlastnosti (vzhled, způsob zobrazení, umístění atd.), a to dokonce i v závislosti na druhu zařízení, na němž je HTML dokument prezentován (obrazovka PC, tiskárna, mobilní telefon, zvukový výstup atd.). V CSS existují mj. i pravidla stylů. Styl definuje pravidla jak prvek reprodukovat. Reprodukováním je myšleno, zobrazení, ale i jiné způsoby interpretace prvku. Takových pravidel může být spousty. Nejvíce používané jsou např. barva, font, pozice, aj. Kaskádovost stylů je dána hierarchickým uspořádáním pravidel pro styly. Každý podřízený styl předefinuje stejnojmenná nadřazená pravidla pro jistý prvek. Pravidla jsou interpretována v této kaskádě:

- Za prvé externí šablony prvků.
- Za druhé lokální styly prvků.
- A za třetí s největší prioritou styl u konkrétního prvku.

Způsoby použití stylů: Definice CSS mohou být součástí HTML dokumentu nebo mohou být v samostatném souboru. Možnosti definování stylů jsou:

- Přímo v HTML dokumentu, v části označené pá rovou značkou `<STYLE>` v sekci `<HEAD>`.
- V konkrétní značce v HTML dokumentu.
- V externím souboru. Soubor se načítá pomocí `<LINK>` v sekci `<HEAD>`.
- V externím souboru. Soubor se načítá CSS direktivou `@import` v sekci `<STYLE>`, avšak pouze v *CSS2*.

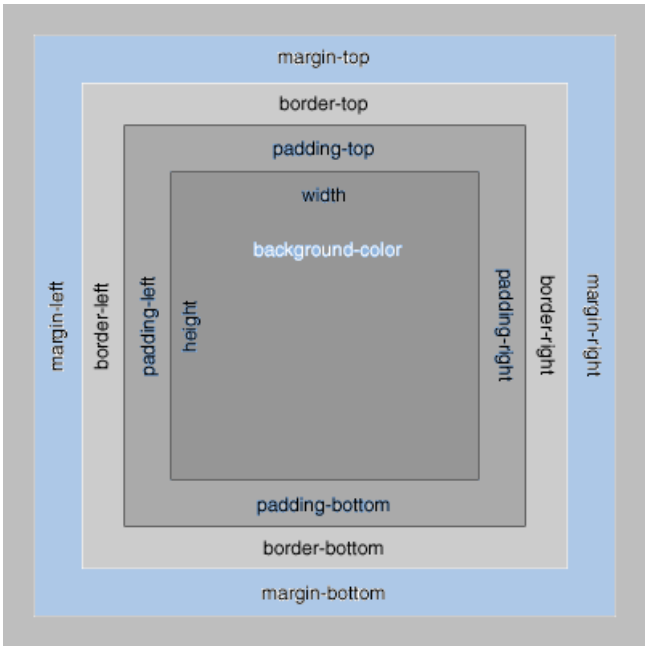
Pravidla stylů tvoří seznam položek. Položkou může být buď to pravidlo ve tvaru `selektor definice`. Selektor popisuje, na který prvek/prvky se příslušný styl vztahuje. V případě inline stylů (přímo ve značkách HTML) se už selektor neuvádí, selektorem je značka sama. Definice potom je pak jedním z atributů značky, ve tvaru `<ZNAČKA...style="definice»`. Definice již popisuje konkrétní vlastnosti zvoleného selektoru (kupř. HTML značky) a má tvar `vlastnost:hodnota`, případně seznam těchto vlastností oddělených středníkem, jako např. `vlastnost:hodnota; ... ; vlastnost:hodnota`.

Příklad definice stylu (nadpis značka H1):

```
H1 {
  color: blue;
  font-style: italic;
}
```

2.3.1 Box v CSS

Každý box je tvořen svým obsahem. Okolo něj je několik oblastí, které mohou mít libovolnou velikost. Vnitřní oblasti je samotný obsah boxu. Jeho rozměry určují vlastnosti *width* a *height* a kolem obsahu je oblast *padding*. Dále vně oblasti *padding* je oblast *border* (rámec) a kolem rámce je pak oblast *margin*. Více pochopitelné je to z obrázku 2.2.



Obrázek 2.2: Box v CSS.

2.3.2 Zobrazovací stroj CSSBox

Informace pro tuto podsekci jsou nastudovány z dokumentace stroje [10]. *CSSBox* je renderovací nástroj pro parsování *(X)HTML/CSS*) napsaný v čisté Javě. Jejím hlavním účelem je poskytnout další zpracovatelné informace o renderované stránce a jejím obsahu a dispozici. Běžným použitím nástroje je získání potřebných informací o pozicích a stylech elementů dokumentu. Vstupem nástroje je dokument *DOM tree*. Potom nástroj automaticky načte odkazované kaskádové styly v dokumentu a získá informaci o stylu každého elementu a dále vypočítá pozici tohoto elementu na stránce. Výstupem stroje je „strom boxů“, který tvoří dokument, aby se element objevil ve stromu boxů je dobré vždy definovat jeho vzhled CSS styly. Více o stromu boxů v podsekci 4.2.1. Jak je využit tento stroj pro vytvoření komponenty řízené pomocí CSS je v podsekci 4.2.1 v sekci 4.2. Pro více informací, jak tento nástroj funguje, odkazuji na dokumentaci [10].

2.4 JSON

Text z této podkapitoly je inspirován z webové stránky [11]. JSON (*JavaScript Object Notation*) je způsob zápisu dat nezávislý na počítačové platformě a používá se pro přenos dat, která mohou být organizována v polích nebo agregována v objektech [11]. JSON řetězec je pro uživatele dobře čitelný a jednoduše se píše, dále je snadno analyzovatelný a tvořitelný počítačově (strojově). Je založen na podmnožině Programovacího jazyka JavaScript, Standard ECMA-262 3rd Edition - December 1999 [11]. JSON řetězec má podobu textu a má obecný formát,tudíž není závislý na jazyce, avšak JSON řetězec využívá konvece známe programátorům[7]. Díky těmto vlastnostem je JSON řetězec velice vhodným až ideálním jazykem pro přenos dat.

[7] Programtoři jazyků rodiny C (C, C++, C#, Java, JavaScript, Perl, Python a dalších).

Struktury JSON:

- Kolekce párů název a hodnota. Ta bývá v různých jazycích realizována jako objekt, záznam, struktura, slovník, hash tabulka, klíčový seznam nebo asociativní pole.
- Seřazený seznam nebo posloupnost hodnot. Ten je ve většině jazycích realizován jako pole, vektor, seznam nebo posloupnost.

Všechny moderní jazyky v podstatě JSON podporují, protože se jedná o univerzální datové struktury [11].

V JSON jsou tyto struktury realizovány pomocí následujících konstrukcí (formát JSON řetězce): Objekt je neuspořádaná množina párů název a hodnota. Objekt je uvozen znakem levou složenou závorkou a zakončen znakem pravou složenou závorkou. Každý název je následován dvojtečkou a páry název a hodnota jsou pak odděleny znakem čárkou. Pole je seřazenou kolekcí hodnot. Začíná levou hranatou závorkou and končí pravou hranatou závorkou. Hodnoty jsou odděleny čárkou. Ilustračně je to vidět na následujícm obrázku.

Kapitola 3

Technologie využívající HTML a CSS

V této kapitole je řečeno něco o nynějších technologií využívající HTML jazyk a technologii CSS. Pro mnoho webových vývojářů, kteří znají HTML a CSS, ale nejsou dostatečně obeznámeni s ostatními technologiemi [12] na jiných platformách jako např. Android, iOS, atd. Proto existují určité frameworky, které mají ulehčit práci těmto webovým vývojářům. V dalších podkapitolách se zmínim o dvou technologií, které slouží k tvoření mobilních aplikací.

3.1 jQuery Mobile

JQuery Mobile je dotykově optimalizovaný HTML5 UI framework navržený k tvoření webových stránek a aplikací, které jsou dostupné na všech smartphone, tabletů a stolních zařízení [13]. Tzn. není potřeba vymýšlet design, ani optimalizovat ovládání pro malé mobilní displeje, celá mobilní verze lze snadno navrhnout a zprovoznit pomocí *JQuery Mobile* [14]. Všechno co musíte dělat je, dodržovat strukturu HTML5 kódu a zásad platformy *JQuery Mobile*, kód je čistý, snadno udržovatelný, veškeré vlastnosti *JQuery Mobile* se nastavují v atributech jejich prvků, jednoduše tak lze změnit chování i vzhled mobilní aplikační verze.

3.2 PhoneGap

PhoneGap je webově založený mobilní vývojový framework [15], založený na projektu *Cordova* [16]. Technologie umožňuje využít standardní webové technologie jako HTML5, CSS3 a JavaScript pro multi platformní vývoj aplikací. Bez ohledu na to jaký vývojový jazyk používá mobilní platforma např. Java pro Android, .NET pro Windows Phone, atd. Aplikace jsou obaleny s ohledem na každou platformu, kde jsou spouštěny a spoléhají na standardní API pro přístup k přistrojů na každém mobilním zařízení jako např. senzory, data, atd.

PhoneGap je určen pro mobilní vývojáře, kteří chtějí rozšířit svoji aplikaci na více platform, bez toho, aniž by museli implentovat znovu aplikaci v každém vývojovém prostředí. Nebo pro webové vývojáře, kteří chtějí rozmístit svoji aplikaci, která je určena pro distribuci v různých obchodních portálech. Sama aplikace je implementována jako webová stránka implicitně pojmenovaná *index.html*, která referencuje na CSS, JavaScript, obrázky, audio, nebo další zdroje potřebné k běhu aplikace.

Kapitola 4

Konceptuální návrh

Kapitola se věnuje základními vlastnosti a funkčnosti komponenty. Návrh je založen na teoretických znalostí z kapitoly 2. V první sekci bude vystvětleno něco o swingovských komponentách a také jak se tvoří swingovská aplikace. Dále v další sekci se podrobněji probere koncept komponenty řízené pomocí CSS, kde se ukáže, jak byl využit zobrazovací stroj *CSSBox*, rozhraní Swing a jakým způsobem se tvoří nebo skládá komponenta z boxů CSS. A jak se bude postupovat při implementaci komponenty.

4.1 Rozbor komponent v rozhraní Java Swing

Všechny komponenty Java Swing dědí od třídy `JComponent`, která dědí od třídy `Container`, která sama dědí od třídy `Component` [3]. Třída `Container` zajišťuje, aby se komponenty daly vkládat do sebe a uvnitř je rozvrhovat. Třída `JComponent` je hlavní třídou rozhraní Java Swing, která zajišťuje pozicování komponent, vykreslování, události, atd. Třída poskytuje další funkcionalitu svým potomkům jako nápovědu při najetí myši, vykreslování okrajů, kopírování a tažení textu, vlastní nastavení vzhledu komponenty atd.

Swing definuje oddělené modelové rozhraní pro každou komponentu, které má data (*logical data*) nebo *GUI*. Tyto rozhraní poskytuje aplikacím vlastní použití implementovaného svého modelu pro Java Swing komponentu. Modely, které poskytuje Swing, spadají do dvou kategorií, a to buď GUI model (*GUI-state model*) nebo programové-data model (*application-data model*). GUI modely jsou rozhraní, které definují vizualizaci komponenty jako např. tlačítko je stisknuto nebo jsou vybrané položky v listu. Programové-data modely jsou rozhraní, které reprezentují nějaké použitelné data, které mají význam v kontextu programu jako např. hodnota buňky v tabulce nebo položky v listu [4].

Abyste vytvořili swingovskou aplikaci musí vždy obsahovat jeden z tzv. kontejnerů nejvyšší úrovně (*top-level container*), do které se vkládají komponenty [3]. Nejčastěji se používají `JFrame`, `JApplet` a `JDialog`. V demonstrační aplikaci této komponenty je právě použit `JFrame`. Aby se komponenty objevily na obrazovce, musí tvořit uzavřenou hierarchii (*containment hierarchy*) [3]. Uzavřená hierarchie je strom komponent, kde jeho kořen je jeden z kontejnerů nejvyšší úrovně. Každá komponenta může být vložená jenom jednou do kontejneru. Když je komponenta vložena do nějakého kontejneru a pokusíte si ji vložit do jiného, tak se z toho prvního smaže a vloží do druhého [3]. Každý kontejner nejvyšší úrovně má svůj `ContentPane`, který obsahuje viditelné komponenty. Ke kontejneru nejvyšší úrovně se může také přidat panel menu. Panel menu je konvence, která je uvnitř kontejnerů nejvyšší úrovně, ale mimo `ContentPane`.

K vytvoření komponenty řízené CSS byly využity už definované komponenty Java Swing. Jsou to komponenty `JButton` pro tlačítka, `JTextField` pro textová pole, `JTextArea` pro textové oblasti, `JRadioButton`, `JCheckBox` a další, které budou zmíněné v další podkapitole 4.2.

4.2 Koncepce komponenty řízené CSS

Jak bylo řečeno v minulé podkapitole, tak komponenta využívá Java Swing komponenty, je vlastně složena z několika komponent Java Swing, které jsou patřičně upravené. Tyto komponety představují většinou nějaké elementy z HTML dokumentu, přičemž jejich vzhled je upraven podle kaskádových stylů (CSS).

Důležitou částí je renderovací stroj *CSSBox*, který mj. zpracuje HTML a CSS a vytvoří informace, které se použijí k implementaci. Po ukončení syntaktického rozboru stroje *CSSBox* vznikne „strom boxů“, který tvoří elementy HTML. Každý z těchto boxů má v sobě definované všechny kaskádové styly. Tento box tvoří obdélníkovou oblast ve zpracovávaném HTML dokumentu a reprezentuje příslušný HTML element, ale může nastat i případ, kdy více boxů definuje jeden element [10]. Box reprezentuje třída `Box`[1], ale existují další třídy boxů, které dědí od této třídy. Třídy, které přímo dědí od třídy `Box` jsou `TextBox`, která vizualizuje jakýkoli text, vyskytující se v HTML kódu, a objekt `ElementBox`, který implementuje elementy HTML s jejich CSS styly. Od třídy `ElementBox` dále dědí `BlockBox` pro blokové elementy a `InlineBox` pro inlinové elementy, od těchto tříd dědí další třídy, které blíženi přísluší k nějakému elementu. Více o hierarchii boxů ve stroji *CSSBox* se dozvíte v jeho dokumentaci [10].

4.2.1 Procházení stromu boxů

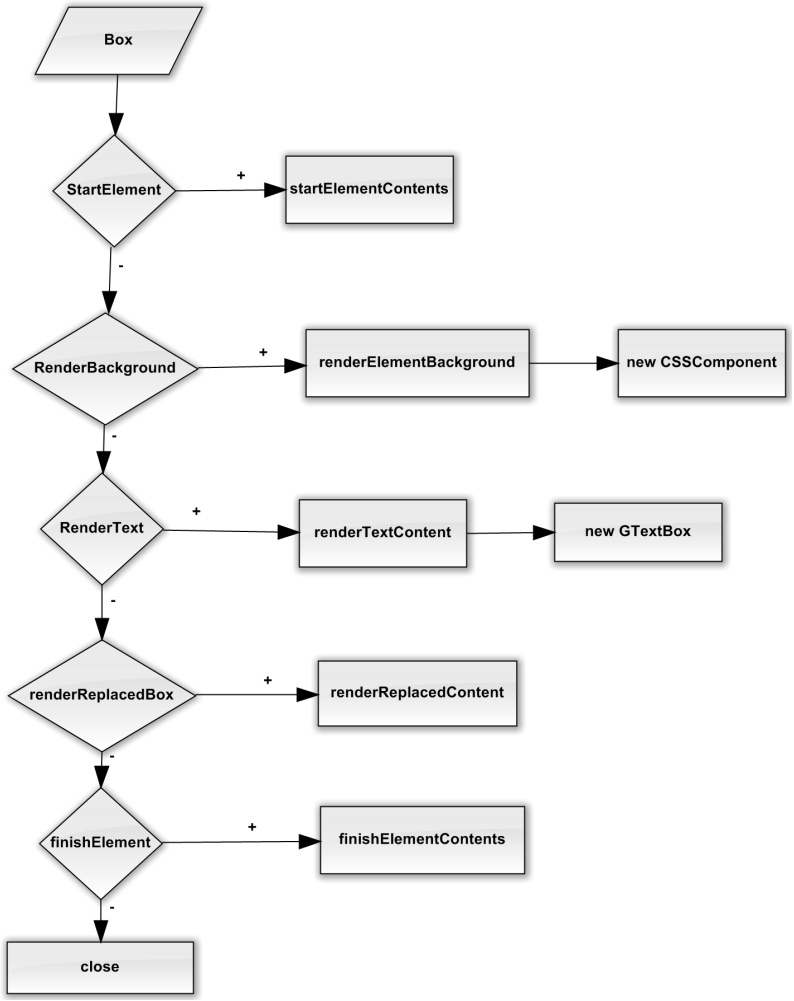
Vznik stromu boxů

Celý výkres stránky je reprezentován objektem `BrowserCanvas`, který patří do stroje *CSSBox*. Nejjednodušší způsob vytvoření výkresu stránky je předání velikosti dimenze vzhledu dokumentu konstruktoru třídy `BrowserCanvas`, který automaticky vypočítá/vytvoří výkres skrz vytvoření instance této třídy. Lze i vytvořit nákres zavoláním metody `createLayout` tohoto objektu, jejíž parametry jsou také velikost dimenze stránky. Výsledný výkres dokumentu je prezentován jako „strom boxů“, kde každý box je reprezentován objektem, který dědí od abstraktní třídy `Box` [10].

Na renderování „stromu boxů“ je použit `CSSRenderer`, který implementuje rozhraní obecného rendereru `BoxRenderer`[2]. Tento renderer umožňuje zobrazit obsah boxů jako např. pozadí, text, okraje, atd. `CSSRenderer` zobrazuje obsah pomocí třídy `Viewport` (třída patří do renderovacího modelu stroje *CSSBox*), která implementuje metodu `draw`. Tato metoda nastaví `CSSRenderer` jako nynější renderer a vykreslí obsah HTML s definovanými styly pomocí tohoto rendereru. `CSSRenderer` má metody, kterými zobrazuje obsah a to jsou pro renderování pozadí elementu, textu v HTML dokumentu a metody pro začátek, inicializování elementu ještě před renderování potomků elementů, práce s elementem a koncem, ukončení renderování elementu po té, co jsou všechny potomci vykresleny.

[1] Abstraktní třída `Box` je součástí implementace stroje *CSSBox* a kořen renderovacího modelu.

[2] Rozhraní `BoxRenderer` je součástí implementace stroje *CSSBox* a slouží k obecné implementaci rendereru.



Obrázek 4.1: Základní struktura, podle rozhraní `BoxRenderer`, třídy `CSSRenderer`.

Nejdůležitějšími metodami jsou renderování pozadí elementu a renderovaní textového obsahu. Tyto metody jsou hlavními pro zobrazování. Z těchto metod, které musí `CSSRenderer` implementovat, protože implementuje rozhraní `BoxRenderer`, používá jenom dvě metody a to `renderElementBackground` k vytvoření elementu, kde element implementuje třída `CSSComponent`, která se postará o pozicování a vzhled elementu a `renderTextContent`, která zobrazí text, kde tuto vlastnost implementuje třída `GTextBox`. Jak postupuje renderer při zobrazování je stručně ukázáno na obrázku 4.1. Ve třídě `CSSRenderer` jsou vytvořeny ještě další metody potřebné ke správnému zobrazení dokumentu, ale o těch bude víc dále.

4.2.2 Text, pozadí a okraje boxů

V této komponentě má text na starosti třída `GTextBox`. Tato třída zjistí pomocí stroje *CSSBox* font a barvu textu, a pozadí vykreslí patřičně podle toho. Všechny tyto informace zjistí z proměnné typu třídy `TextBox`, která dědí od třídy `Box`, kde tato třída má metodu

`getVisualContext`, pomocí které lze zjistit veškeré vlastnosti textu definované pomocí CSS. Postará se i o pozici textu v komponentě a měnění velikosti textu. Protože se vykresluje jenom text, tak se nikdy nezobrazují okraje a pokud není definované žádné pozadí textu, tak je automaticky průhledné pozadí.

U pozadí boxů je to zajímavější. Informace o pozadí a okrajů boxu opět získame pomocí stroje *CSSBox*, který nám poskytne podrobné znalosti buď o barvě pozadí nebo jestli je obrázek jako pozadí, to záleží na definovaném stylu. Pokud je nastaveno pozadí jako obrázek, tak dále se rozlišují další možnosti zobrazení pozadí a to „ repeat-x“, „ repeat-y“ a „ no-repeat“ opět podle definovaného CSS stylu. Jak se vykreslují obrázky, když jsou nastaveny jako pozadí, je podrobněji popsáno v další kapitole 5.

U okrajů to probíhá podobně, získáme informace o tloušce a barvy čáry, stylu okraje strojem *CSSBox* a poté se nastaví, přitom se tato informace nastavuje pro každou stranu okraje zvlášť. Pro komponentu je implementovaná vlastní třída pro vykreslovaní okrajů a je to `CSSBorder`, u které se nejdříve nastaví všechny strany okrajů a potom se teprve vykreslí okraje. Tato třída implementuje čtyři základní styly okrajů a to „solid“, „dotted“, „dashed“ a „ double“.

4.2.3 Použití rozhraní Java Swing

Jak vlastně vypadá struktura komponenty popíši v této podsekci. Jako např. která konkrétní komponenta Java Swing implementuje nějaký element HTML a v čem jsou jednotlivé komponenty vloženy atd.

V předchozí podsekci 4.2.1 bylo řečeno, jak se taková komponenta skládá z HTML a CSS, procházením „stromu boxů“. Pro obecný blokový nebo i inlinový element je použita implementováná třída `CSSComponent`, která dědí od swingovské komponenty `JPanel`. Při vytvoření instance této třídy se sama pozicuje v komponentě, sestaví se okraje, vyplní se pozadí, určí se velikost a ostatní detaily záleží na konkrétním elementu. Jak tyto vlastnosti nastavit zjistí objekt `CSSComponent` ze své instanční proměnné, kterou získá (přes parametr konstruktoru) při vytvoření instance této třídy, tato proměnná je typ třídy `ElementBox`, která dědí od třídy `Box`. Pokud má element definován obrázek jako pozadí, tak lze pro tuto možnost využit komponentu `JLabel`, do které se vloží ikona založená na tomto obrázku. A objekt `JLabel` je vložen do komponenty a zároveň pozicován v komponentě, ale o tom víc v podkapitole Měnění velikosti a pozicování 4.2.4.

Text obsažený v HTML dokumentu je v komponentě zobrazen pomocí objektu `GTextBox`, který dědí od třídy `JTextField`. Tento objekt sestaví pozadí, font a všechny další vlastnosti definované pomocí CSS tohoto textu. Tento text není obsahem žádného elementu zobrazeného pomocí objektu `CSSComponent`, ale je umístěn do úplného popředí celé komponenty.

Tyto dvě hlavní komponenty Java Swing `JTextField` a `JPanel`, slouží jako základní kameny komponenty řízené CSS, ale dále je využito rozhraní swing pro HTML element `FORM` (formuláře) a jeho potomky (podelementy).

Element input: U elementu `INPUT` záleží na jeho atributu „type“, podle tohoto atributu se rozhoduje, jak se dále bude postupovat k zobrazení tohoto elementu. Takže to vezmu popořadě a začnu u typu „text“, jenž je běžné textové pole, které je v komponentě reprezentováno objektem `JTextField`. U tohoto objektu se nastaví vlastnosti jako font textu nebo obsah textového pole, barva pozadí a textu, a další vlastnosti nastavené pomocí CSS. Další typ v pořadí je „password“, který je podobně zobrazen v komponentě, ale pro tento typ je v rozhraní swing už připravená swing komponenta a to `JPasswordField`, u kterého opět

nastavíme vlastnosti. Typ „radio“ je zobrazen swingovskou komponentou `JRadioButton` a typ „checkbox“ je prezentován pomocí swing komponenty `JCheckBox`, u kterých jsou upraveny vlastnosti podle CSS stylu. Dalšími podobnými typy, které reprezentují tlačítko v komponentě, ale liší se jen nepatrně jsou „submit“, „reset“, „button“. Tyto typy jsou reprezentovány komponentou swing `JButton`, která je opět patřičně upravena podle CSS stylu. Posledními typy jsou „file“ a „image“, ten první je složen ze dvou swingovkých komponent `JTextField` pro cestu souboru a `JButton` pro výběr souboru pomocí objektu `JFileChooser` a ten druhý je vytvořen skrze objekt `JButton`, který má nastavený obrázek jako pozadí.

Element `INPUT`, ale není jediný element obsažený ve formuláři, který je implementován v komponentě. Jsou tu ještě další elementy. Jeden z nich je element `TEXTAREA`, což je textová oblast, která se liší od textového pole, že obsahuje víc než jeden řádek. Tento element je implementován `JTextArea` Swing komponentou. Druhý z elementů je `Button`, který je téměř stejný jako typ elementu `INPUT`, který reprezentuje tlačítko tzn. opět tento element reprezentuje komponenta `JButton`. Poslední element je `SELECT`, kde tento element je zobrazen pomocí objektu `JComboBox` nebo `JList`, s tímto elementem souvisí element `OPTION`, který je vlastně položkou elementu `SELECT` a tedy i `JComboBox` (`JList`. Celý vzhled těchto swingovských komponent je řízen CSS styly, a tak jsou vhodně upravovány podle toho. Tvorba těchto elementů je součástí implementace třídy `CSSComponent`, a tak jsou vloženy do tohoto objektu.

Všechny objekty `CSSComponent` a `GTextBox` jsou pozicovány a vkládány do swingovské komponenty `JLayeredPane` více v další podsekci 4.2.4. Blíže k implementaci těchto elementů v další kapitole 5.

4.2.4 Měnění velikosti a pozicování

Úprava velikosti celé komponenty řízené CSS je možná, a v této podsekci popíšu, jak je provedena. Všechno to začíná u třídy `CSSRenderer`, která obsahuje metodu `componentsResize`, která změnu velikosti odstartuje. Tato metoda má dva klíčové parametry a jsou to `percWidth`, který obsahuje procento o kolik se má zvětšit/zmenšit šířka všech komponent oproti své původní šířky (šířky při vytvoření komponenty) stejně je tomu tak i u parametru `percHeight` s tím rozdílem, že jde o výšku. Tyto procenta jsou buď kladná (zvětšení velikosti) nebo záporná (zmenšení velikosti). Jak je vidět v pseudokódu metody `componentsResize` 1 začíná to změnou velikosti objektu `JLayeredPane`, který obsahuje všechny zobrazené elementy (komponenty vytvořené z elementů). A po té se prochází všechny komponenty obsažené v objektu `JLayeredPane`, které jsou vlastně objekty `CSSComponent` a `GTextBox`, které obsahují taky metodu pro změnu velikosti toho objektu a je to metoda `componentResize` 2, tato metoda je u obou objektů velice podobná, ale je tu pár rozdílů. U objektu `GTextBox`, který reprezentuje text v komponentě se musí starat i o velikost zobrazeného textu, ale u objektu `CSSComponent` je pouze navíc u nějakého formulářového elementu komponenta, reprezentující editovatelný element např. `INPUT`, atd., který se musí také upravit. Poslední velice důležitá změna u těchto dvou objektů, je změna jejich souřadnic v komponentě, která se počítá obdobně (stejným způsobem) jako u velikosti, a to podle procent.

Pozicování

Už bylo zmíněno, že všechny komponenty jsou vkladány do objektu `JLayeredPane` a je k tomu důvod. Tento objekt má totiž vlastnost vkládat komponenty do vrstev. Tato vlastnost je využita při tvorbě komponenty řízené CSS. A to tak, že každý objekt `CSSComponent`

Algorithm 1 Pseudokód metody `componentsResize`

```
1: function componentsResize(percWidth, percHeight)
2:     int width = vypocetNoveSirky
3:     int height = vypocetNoveVysky
4:     layeredPane.nastaveniNoveVelikosti(width, height)
5:     for Componentcomp : layeredPane.getComponents() do
6:         if comp instanceof CSSComponent then
7:             comp.componentResize(percWidth, percHeight)
8:         end if
9:         if comp instanceof GTextBox then
10:            comp.componentResize(percWidth, percHeight)
11:        end if
12:    end for
13:    layeredPane.prekresli()
14: end function
```

Algorithm 2 Pseudokód metody `componentResize`

```
    function componentResize(percWidth, percHeight)
2:      int width = vypocetNoveSirky
        int height = vypocetNoveVysky
4:      int x = vypocetNoveXSouradnice
        int y = vypocetNoveYSouradnice
6:      this.nastaveniNoveVelikosti(width, height)
        this.nastaveniNovePozice(x, y)
8:      if comp instanceof CSSComponent then
            this.upravObrazkyPodleProcent(percWidth, percHeight)
10:         this.upravOkrajePodleProcent(percWidth, percHeight)
            if this.component ≠ null then
12:             formComponentResize(percWidth, percHeight)
            end if
14:     end if
        if comp instanceof GTextBox then
16:         int fonsize = vypocetNoveVelikostTextu
            this.nastaveniNoveVelikostiTextu(fonsize)
18:     end if
        this.prekresli()
20: end function
```

nebo `GTextBox` je vložen do vrstvy podle hloubky zanoření jejich instanční proměnné `box` ve „stromu boxů“ vytvořený strojem *CSSBox*. Pro zjistění hloubky je u obou objektů implementovaná metoda `getPlunge`, která vrátí tuto informaci. Dále jsou tyto objekty ve vrstvách absolutně pozicovány podle vypočítaných souřadnic. Tyto souřadnice jsou vypočítány strojem `CSSBox` podle definovaného stylu CSS. Důvod pro vkládání objektů do vrstev podle hloubky zanoření je za účelem nezastiňování/nepřekrývání se mezi sebou. U objektu `JLayeredPane` je vrstva s největším číslem nejvíc v popředí.

4.2.5 Odkazy

Odkazy v této komponentě nejsou jako odkazy v HTML dokumentu zobrazeném v prohlížeči, ale mají jinou funkcionalitu. Fungují na základě obsahu JSON řetězce obsaženém v atributu „href“ elementu `A`. Tento řetězec obsahuje úplný název třídy, která se bude instanciovat podle obsaženého konstruktoru v řetězci a případně volání metod s jejich parametry, která obsahuje tato třída. Tzn. že odkaz v komponentě řízené CSS je něco jako volání nějaké akce (události) definováné pomocí řetězce JSON. Avšak je nutné počítat s jistými detaily při předávání parametrů u konstruktorů a u parametrů metod, o kterých bude řečeno víc v další kapitole 5. **Odkaz může obsahovat jenom jednu hodnotu třídy a jenom jedno pole volání metod.** Při nesprávnem zadání řetězce JSON se neprovede nic (při reakci na odkaz), ale vyvolá se vyjímka, která se vypíše do konsole.

Příklad řetězce JSON v elementu `A`:

```
<A id="item1" href="{
  'class':
          { 'name' : 'ibp.csscomponent.DemoCSSComponent',
            'constructor' : { 'param1' : '1' } },
  'methods':
      [{ 'name' : 'setSize', 'param1' : '143', 'param2' : '34'},
       { 'name' : 'setColor', 'param1' : '143'}]
}"> Home </A>
```

4.2.6 Přístup k formulářům a k jeho ovládacím prvkům

K editovatelným elementům umístěných ve formulářích je dobré mít nějaký přístup, aby jsme s nimi mohli dále pracovat po vytvoření komponenty. Pro přistup k elementů umístěných ve formuláři musíme nejdříve přistoupit k formuláři a k němu se dostaneme přes objekt `CSSRenderer` komponenty. Objekt `CSSRenderer` obsahuje asociativní pole pro formuláře, kde klíč je hodnota atributu „id“ a hodnota je formulář, reprezentovaný jako objekt třídy `CSSComponent`. Třída `CSSComponent` obsahuje asociativní pole pro všechny editovatelné elementy umístěné ve formuláři, kde klíč je opět atribut „id“ těchto elementů. Aby se nějaký element objevil v tomto asociativním poli, tak musí obsahovat jedinečný atribut „id“, jinak tento element asociativní pole nebude obsahovat. V další kapitole 5 budou zmíněny názvy metod, které vrací ona asociativní pole. Více o přistupování k těmto prvků v další kapitole.

Kapitola 5

Implementace návrhu

V kapitole se popisuje základní vlastnosti implementované podle návrhu z předešlé kapitoly. Za pomocí vlastností rozhraní Java Swing je implementovaná většina komponenty a je detailněji ukázano, co všechno je použito při implementaci z rozhraní Swing. Využití zobrazovacího stroje *CSSBox* bude popsáno ve větším detailu, kupř. jaké funkce byly použity nebo, které konkrétní vlastnosti byly použity. V první sekci se popisuje podrobněji o implementaci návrhu komponenty, kde je vystvětleno, jak se detailněji postupovalo při implementaci. Ve druhé sekci je představena a popsána aplikace, která slouží k demonstraci komponenty, tj. její vlastnosti a funkce.

5.1 Implementace návrhu komponenty řízené CSS

Podle kapitoly 2 je zřejmé, že implementační jazyk této komponenty je Java. V této sekci a v jeho podsekcích je řečeno o implementaci tříd, které dohromady implementují komponentu řízenou pomocí CSS. V následující podsekci 5.1.3 je i řečeno jak se využila reflexe v Javě, o tomto tématu už padla zmínka v minulé kapitole, ale v této podsekci se přiblížím blíže k implementaci.

Vytvoření komponenty: Pro vytvoření komponety jsou k dispozici dva způsoby. První způsob, ten jednodušší, je pomocí konstruktoru `CSSComponent`, který potřebuje jeden parametr a to je `Viewport`, který obsahuje informace o stránce, „stromu boxů“, atd. Jak tento `Viewport` získat bude řečeno v další podsekci. Dalším způsobem je vytvořit komponentu přímo pomocí rendereru (`CSSRenderer`), předchozí způsob také využívá tento renderer, jenom je součástí konstruktoru. I k tomuto způsobu je potřeba `Viewport`, pomocí kterého využijete renderer k vykreslení stránky, ve třídě `Viewport` je metoda `draw`, která tento problém implementuje, a právě potřebuje parametr typu `BoxRenderer`.

5.1.1 Získání Viewport (CSSRendererFactory)

Už bylo několikrát zmíněno, že `Viewport` je důležitou částí k renderování komponenty (stránky), a tak v této podsekci popíši, jakým způsobem je implementováno/zajištěno, získání tohoto objektu. K zajištění objektu `Viewport` je implementována třída `CSSRendererFactory`, která obsahuje statické metody k získání oněho objektu.

Statické metody třídy `CSSRendererFactory`: Tyto metody mají společné dva poslední parametry. Je to šířka a výška stránky (`Viewport`).

- `getViewportOfUrl` - jeho první parametr je url adresa dokumentu, ze kterého se vytvoří `Viewport`
- `getViewportOfFilePath` - jeho první parametr je cesta k dokumentu, ze kterého se vytvoří `Viewport`
- `getViewportOfResources` - mimo parametry výšky a šířky má další dva, a to první třídu, druhý cestu k dokumentu, který je *resource* této třídy, ze kterého se vytvoří `Viewport`
- `getViewport` - má jako první parametr objekt `DocumentSource`, což je objekt vytvořený z dokumentu

5.1.2 Skládání komponenty (CSSRenderer)

V této podsekci se podrobněji popíšou důležité metody pro renderování komponenty mimo jiných, které byly zmíněny v minulé kapitole o koncepci. Začnu těmi, které vytváří komponentu. Předtím než začnu s popisem, zmíním dvě důležité asociativní pole. A to instanční proměnná `forms` typu `HashMap`, která obsahuje veškeré formuláře obsažené v dokumentu a instanční proměnná `links` stejného typu, která obsahuje veškeré objekty `MouseListener` všech odkazů obsažených v dokumentu.

Metoda `renderElementBackground`: Tato metoda má parametr typu `ElementBox`, který nese informace pro renderování. Co má tato metoda za úkol už bylo popsané v minulé kapitole 4, ale tady se popíše blíže, co všechno metoda vlastně provádí.

V této metodě se využívá pomocná metoda `isChildOfForm`, která vrací `boolean` hodnotu, o tom jestli je element potomkem elementu `FORM`. Důvod k tomuto je ten, že pokud je potomkem, tak se nepokračuje, protože element `FORM` byl celý už zpracován. Formulář se zpracovává najednou celý i s jeho potomky, ale o tom více v další podsekci. Pokud není potomkem, pokračuje se dál a renderuje se element. Při elementu `FORM` se element (`CSSComponent`) vloží do asociativního pole `forms` s klíčem obsahu atributu „id“ elementu, jestli element neobsahuje tento atribut, nebude vložen do asociativního pole. Poslední důležitou funkcí této metody je vložení objektu `MouseListener` do potomků (přidání objektu `MouseListener` do jejich „MouseListenerů“ z asociativního pole `links`) elementu `A` (odkazu). K zjištění potomků odkazu slouží metoda `isChildOfTagA`.

Metoda `renderTextContent`: Tato metoda má parametr typu `TextBox`, který nese informace pro renderování. Účel metody byl popsán v kapitole 4, tady jen popíši detaily ohledně provedení metody. K renderovaní objektu `TextBox` nedojde ve třech případech a to v případě, že element je typu `TEXTAREA`, `BUTTON` nebo `LEGEND`, neboť text obsažený v těchto elementech má jinou funkcionalitu. Dále se ošetří odkazy jako v metodě `renderElementBackground` (přidání „MouseListenerů“).

Změna barvy pozadí a textu při přejetí myši u odkazech: K nastavení barvy pozadí a textu u odkazů probíhá už při sestavování objektu `MouseListener`, kde se nastaví metody `mouseEntered`, při přejetí myši, a `mouseExited`, při odjetí myši. Barva, která se nastaví jako pozadí, je definována instační proměnnou `hoverBackground` třídy `CSSRenderer` a barva textu instanční proměnnou `hoverForeground`. Tyto dvě proměnné lze nastavovat a tedy

[0]Třída `DocumentSource` je součástí implementace stroje *CSSBox*.
[0]Objekt `MouseListener` v sobě obsahuje provedení akce na stisknutí odkazu.

je lze měnit. K tomu slouží metody `setHoverForeground` a `setHoverBackground`, které potřebují parametr typu `Color`.

5.1.3 Obecný blokový a inline element (CSSComponent)

Mimo konstruktoru pro vytvoření celé komponenty má tato třída `CSSComponent` i konstruktor pro tvoření blokového nebo inlinového elementu. Během realizování objektu pomocí tohoto konstruktoru se splní několik vlastností, které teď popíši. Nejdříve je potřeba pozicovat objekt v komponentě (v panelu `JLayeredPane`), k tomu se zavolá metoda `setAbsolutePosition`, která se postará o nastavení pozice objektu nastavením souřadnic metodou `setBounds`, která je zděděna od třídy `JPanel`. Pozice je spočítána strojem *CSSBox* a získá se voláním metody `getAbsoluteBounds` třídy `ElementBox`, ovšem výsledná souřadnice se dopočítá přičtením hodnot „margin“ (metoda `getEMargin` třídy `ElementBox`). Pokračuje se nastavením velikosti objektu a vykreslením pozadí. Informace o pozadí vrací metoda `getBgColor` třídy `ElementBox`, pokud je navrácená hodnota rovna `null` bude pozadí průhledné (*transparent*). Existuje i možnost, že jako pozadí je nastaven obrázek, k tomu slouží metoda `displayBackgroundImages`, která obrázek patřičně zpracuje, ale o tomto tématu více informací dále v textu (Obrázky).

Následuje sestavení okrajů zavoláním metody `setCSSBorder`, která už zařídí nastavení okrajů (nastavení okrajů pomocí objektu `CSSBorder` je popsáno v podsekci 5.1.5).

Další vlastnosti, které jsou nastavovány při vytváření objektu už jsou specifičtější a závisí na typu elementu. Element `IMG`, který reprezentuje objekt `BlockReplacedBox` nebo `InlineReplacedBox` ve „stromu boxů“ 4.2.1. K sestavení tohoto objektu je implementována metoda `displayReplacedBox`, kde tato metoda vykreslí obrázek obsažený v elementu (více o obrázcích dále v textu). Metodou `setMouseListener` se nastaví odkaz elementu `A` spolu se stylem označování odkazů při přejíždění myši, více níže v textu (Sestavení odkazu). Další vlastnosti se převážně týká formuláře a jeho potomků. V minulé kapitole 4 bylo poznamenáno, že celý element `FORM` (formulář) se realizuje najednou a je to implementované metodou `processForm`, více níže v textu (Kompletování formuláře). Pro element `INPUT` slouží metoda `setInput`, element `TEXTAREA` je sestaven pomocí metody `setTextArea`, pro elementy `SELECT` a `BUTTON` posluhují metody `setSelect` (nebo `setSelectList`) a `setButton`.

V případě elementu seznamu (`UL`, aj.) a jeho položek `LI`, kde zobrazení ikon u těchto položek záleží na vlastnosti CSS „list-style-type“. V případě typu „circle“ se vykreslí kružnice, u typu „square“ se vykreslí čtverec a u ostatních typů se zobrazí text získaný metodou `getMarkerText` objektu `ListItemBox`, který ve stroji *CSSBox* implementuje element `LI`.

Kompletování formuláře

Při vytvoření elementu `FORM` se inicializují několik speciálních instančních proměnných (asociativních polí), které slouží k přístupu k editovatelným prvkům formuláře. A jsou to `textFields` pro typ „text“ (a typ „file“) elementu `INPUT` a další jeho typy. Pole `passwordFs` pro typ „password“, pole `checkBoxes` pro typ „checkbox“, pole `radioButtons` pro typ „radiobutton“, kde pro tento typ je každé tlačítko ve skupině tlačítek `ButtonGroup`, která je přístupná přes asociativní pole `buttonGroups`, kde klíč je atribut „name“ elementu `INPUT`. Asociativní pole `buttons` pro typ „button“, „reset“, „submit“, „image“ a element `BUTTON`. Dále pole `textAreas` pro element `TEXTAREA` a pole `selects` (nebo `lists`) pro element `SELECT`. **Přes tyto asociativní pole má uživatel této komponenty přístup k těmto elementům, které může upravovat.** Asociativní pole jsou implementovány typem `HashMap`. Výše bylo zmíněno, které metody mají na starost tvoření těchto elementů.

V těchto metodách se využijí swingovské komponenty (využití swingovských komponent v podsekci 4.2.3), které se upraví podle definovaných kaskádových stylů a jejich vlastností, podle atributů elementu definovaného HTML kódem jako např. „readonly“, „disabled“, „checked“, atd. Více o elementech HTML v podsekci 2.2.1.

Kompletování formuláře má za úkol metoda `processForm`, která vlastně prochází „strom boxů“ s tím rozdílem, že kořenem stromu je element `FORM`. Tato metoda má parametr typu `Box`, který prochází. V metodě se „strom boxů“ zpracovává rekurzivně. V každém objektu `Box` vždy prochazí jeho potomky, a když je potomek jeden z editovatelných elementů, jako např. `INPUT`, `TEXTAREA`, atd., tak se element vytvoří a vloží do patřičného asociativního pole, ale pokud je to obecný blokový nebo inlinový element, tak se znova zanořuje a opět se prochází jejich potomky. Toto zpracovávání stromu by také mohlo být alternativním řešením oproti rendereru (`CSSRenderer`).

Sestavení odkazu

V kapitole 4 (koncepce) je poznamenáno, že význam, co se má provést při reakci na odkaz, je obsaženo v atributu „href“ a obsah tohoto atributu je ve formátu JSON 2.4. Pro parsování JSON řetězce slouží knihovna *org.json* [17]. Kontkrétně třída `JSONObject`, která patří do této knihovny. Pokud JSON řetězec není ve správném formátu, tak dojde k výjimce `JSONException`, která je implementovaná taky v této knihovně.

Jak to probíhá popíši zde. Nejdříve se získá jméno třídy, která se bude instanciovat. Ještě se musí získat parametry konstruktoru, tedy pokud byly definovány nějaké. Poté se třída instanciuje buď s parametry konstruktoru nebo bez. Poté probíhá syntaktický rozbor metod v JSON řetězci, pokud byly zadány. Získají se jména metod a jejich parametry, které se hned po rozboru volají. Je zde využita reflexe, která je popsána v podsekci 2.1.3.

Předávání parametrů:

- U konstruktoru jsou předávány jako při běžném volání konstruktorů, parametry mají typ `Object`
- U metod jsou předávány jako pole typu `Object[]` (jediný parametr), kde parametry jsou obsaženy v tomto poli a opět jsou typu `Object`

Obrázky

V této sekci se zmíním o obrázcích jako pozadí a o vykreslování elementu `IMG`. Jak už bylo zmíněno výše, slouží k tomu metody `displayBackgroundImages` a `displayReplacedBox`. Ačkoliv u té první je to více komplexnější, protože existují tři možnosti zobrazení obrázku jako pozadí. Začnu tedy u té první metody. První možnost zobrazení je definování pomocí CSS stylů pozadí „repeat-x“ a druhé „repeat-y“. Jediný rozdíl je v tom, že u první možnosti se obrázky skladají podle osy X a u druhé podle osy Y. Obrázek, který se má zobrazit, získám od objektu `ElementBox` (třída je součást renderovacího modelu stroje *CSSBox*) konkrétně metodou `getBackgroundImages`. Tento obrázek si obstarám a vykreslím ho do nově vytvořeného obrázku (nově vytvořeného objektu `BufferedImage`) podle osy tolikrát, aby to odpovídalo šířce/výšce elementu. Kreslení probíhá pomocí objektu `Graphics`, který jsem získal od objektu `BufferedImage` (nově vytvořeného obrázku), který jsem nově vytvořil. V objektu `Graphics` ke kreslení obrázku slouží metoda `drawImage`. Při třetí možnosti „no-repeat“ se pouze vykreslí obrázek, jak je definován.

Po upravování obrázku je na základě právě tohoto obrázku vytvořen objekt `ImageIcon`, který je vložen do swingovské komponenty `JLabel`. Objekt `JLabel` je potom vložen do objekt `CSSComponent`, kde se právě tvoří pozadí na základě obrázku.

U metody `displayReplacedBox` je to obdobné s několika rozdíly. Obrázek se získá z objektu `BlockReplacedBox` a konkrétně metodou `getContentObj`, kde tyto metody patří do stroje *CSSBox*. Zbytek probíhá podobně, jako předtím, vytvoří se objekt `ImageIcon` na základě obrázku, který se nijak neupravuje a vloží do `JLabel`.

Tyto dvě metody mají za úkol i měnění velikosti obrázku při změně velikosti komponenty řízené pomocí CSS. Opět se počítá nová šířka a výška (podle procent více v podsekci 4.2.4), a potom se upraví obrázek podle nové šířky a výšky metodou `getScaledInstance`, kterou implementuje třída `Image`. A tyto komponenty `JLabel` jsou i vkládány do instanční proměnné `labelImages` k dalšímu použití.

U elementů typu `BUTTON`, `TEXTAREA` a `INPUT` to neprobíhá, tak jak bylo popsáno výše. U elementů, které využívají swingovskou komponentu `JButton` se obrázek vytvoří metodou `displayBackgroundImages` a uloží do proměnné `labelImages`, ale nezobrazí se. U objektu `JButton` se obrázek nastaví jako ikona tohoto tláčítka (metoda `setIcon` objektu `JButton`), kde obrázek se právě získá z instanční proměnné `labelImages`. U zbylých elementů se využívá metoda `paintComponent`, kde se spolu s vykreslením objektu `JPanel` vykreslí i obrázek opět získaný z proměnné `labelImages`.

5.1.4 Text v komponentě (GTextBox)

Účel této třídy byl zmíněn v kapitole 4, ale za to v této podsekci vylíčím, co které metody mají za úkol. Tato třída dědí od swingovské komponenty `JTextField`, kterou upravím tak, že zneviditelním pozadí a okraje, protože chceme pouze zobrazit text. Dále co se upraví při vytvoření objektu `GTextBox`, je barva a font textu (podle informací získaných metodou `getVisualContext` patřící stroji *CSSBox*). A v poslední řadě nastavíme pozici textu metodou `setAbsolutePosition`.

5.1.5 Okraje (CSSBorder)

V této podsekci zmíním, jak se pracuje s objektem `CSSBorder`, a jak je tento objekt implementován. Třída `CSSBorder` dědí od abstraktní třídy `AbstractBorder`, která poskytuje základní třídu pro jednodušší implementaci ostatních tříd, sloužících k implementaci okrajů.

Když chceme vytvořit okraje, tak vytvoříme objekt pomocí konstruktoru, který nemá žádné parametry. Implicitní nastavení při vytvoření tohoto objektu je styl okrajů „solid“, tloušťka jedna a barva okrajů je černá. K vlastnímu nastavení okrajů jsou implementované metody `setTopBorder`, `setBottomBorder`, `setRightBorder` a `setLeftBorder`. Tyto metody mají tři parametry, první je barva okraje, druhý je styl okraje a třetí je tloušťka okraje. Tato třída poskytuje k nastavení čtyři styly okrajů a to „solid“, „dotted“, „dashed“ a „double“.

5.2 Aplikace demonstrující funkčnost komponenty (DemoCSSComponent)

Tato aplikace má demonstrovat použití komponenty řízené pomocí CSS. V aplikaci je tvorba několika komponent, které jsou obsahově různé a jsou zobrazeny v okně na patřičných místech s ohledem na jejich funkčnost. Také je předvedeno přistupování k editovatelným

Obrázek 5.1: Ukázka demostrující aplikace.

prvkům formuláře a jejich následným úpravám. Dále je uvedena ukázka měnění velikosti komponent, které se můžou měnit podle měnění velikosti rámce `JFrame`, pokud je tak nastaveno.

Po spuštění aplikace se zobrazí tři komponenty. Komponenta vlevo je považována za menu a skládá se ze čtyř tlačítek (elementů `BUTTON`), které mají funcionalitu takovou, že změní komponentu vpravo od komponenty menu. Komponenta zobrazená nahoře slouží jako neměnící se „nadpis“. Všechny tyto komponenty jsou vytvořeny zvlášť a seskládány dohromady. Tímto chci ukázat, že komponentou řízenou pomocí CSS, lze tvořit a pracovat s ní podobně, jako s kteroukoli jinou komponentou Java Swing. Na obrázku 5.1 je předvedena aplikace, v obrázku je ukázano rozpoložení komponent.

V demo aplikaci je i implementováno měnění velikosti okna spolu se všemi komponentami. Objektu `JFrame` je přidán objekt `componentListener`, u kterého je přepsána metoda `componentResize`, kde se spočítá procento, o kolik se zvětšilo/zmenšilo okno oproti původní velikosti a s touto hodnotou se potom volají metody `componentsResize` třídy `CSSRendererru` u všech zobrazených komponent v objektu `JFrame`.

Použití odkazů v komponentě je předvedeno několika způsoby. Např. otevření odkazu v prohlížeči, otevření obrázku v novém okně. Jeden odkaz je i vidět na obrázku 5.1 v komponentě, která slouží jako „nadpis“ aplikace (komponenta nahoře), je to obrázek s nadpisem a rukou.

Kapitola 6

Návrhy k rozšíření

V této kapitole je popsáno, jak by se mohlo pokračovat v implementaci komponenty řízené pomocí CS, co by se mohlo přidat ke komponentě. Přirozeně se nabízí nové možnosti CSS a HTML. Nová verze CSS *CSS3*, která nabízí spoustu nových vlastností, některé vlastnosti budou zde vypsány. A nová verze HTML *HTML5*, která rozšiřuje původní verzi o mnoho funkcí a nových značek. V následujících sekcí bude zmíněno několik vlastnostní k rozšíření komponenty s ohledem na HTML5 a CSS3.

6.1 Rozšíření s ohledem na HTML5

Text v této sekci byl inspirován a i použit z článků [18, 19]. HTML5 je nástupce HTML 4.01. HTML5 přináší mnoho nových elementů, a obohacuje i některé starší. Nové značky mohou být kratší, rychlejší a efektivnější. Autoři HTML5 při jeho vývoji kladou důraz na jednoduchost a efektivitu. V této sekci nebudu popisovat, jak velké změny proběhly v HTML5, ale popíši, které elementy by stálo za to implementovat do kompoenty. Než začnu s popisem, bylo by dobré zmínit změnu *Doctype*, která nám říká jakou verzi HTML používáme, a kde lze najít popis dokumentu (DTD). HTML5 nám nově zavedl zápis `<!DOCTYPE html>`, tento zápis nám říká, že jednoduše používáme HTML a nezáleží na tom jakou verzi používáme, a kde se nachází DTD. Tzn., že stačí změnit na záčatku HTML dokumentu tento zápis a můžeme začít tvořit s HTML5.

Elementy

Párový element `A` byl rozšířen, tak že HTML5 nijak nezasahuje do jeho původní struktury, ale pouze rozšiřuje jeho možnosti. Už se nemusí odkazovat jenom textem nebo obrázkem. HTML5 umožňuje do hypertextového odkazu zabalit jakékoli elementy dokonce i blokové. Párový element `CITE`, který se už objevil v minulých verzích HTML, ale HTML5 upřesňuje jeho využití, a to pro označení titulku díla, ze kterého je citováno. Párové elementy `FIGCAPTION` a `FIGURE`, kde `FIGCAPTION` je potomek elementu `FIGURE`, který umožňuje vložení doplňkového textu, např. k obrázku nebo ke zdrojovému kódu, který je právě spolu s prvkem `FIGCAPTION` potomkem elementu `FIGURE`.

Další zajímavý je element `TIME`, který označuje časové údaje a nabízí nastavení vlastního formátu data a času. Tento element má dva možné atributy a to „datetime“ pro datum, čas nebo datum i čas, které element představuje (časový formát) a „pubdate“, který označuje datum publikování např. článku. Dalším možným párovým elementem je `MENU`, který je v HTML5 přepracován a plní funkci jako seznam ovládacíh prvků pro formuláře.

Jedním z elementů, který se přímo nabízí k rozšíření implementace komponenty je element `INPUT`, který se dočkal v HTML5 velkého rozšíření. A to v podobě nových podporovaných typů obsahu. Nové typy, které element může obsahovat jsou následující:

- **datetime**, ovládací prvek pro nastavení data a času s ohledem na časové pásmo,
- **datetime-local**, ovládací prvek pro nastavení data,
- **date**, ovládací prvek pro nastavení data,
- **month**, ovládací prvek pro nastavení roku a měsíce,
- **time**, ovládací prvek pro nastavení času,
- **week**, ovládací prvek pro nastavení roku a týdne,
- **number**, jednořádkové pole pro číselnou hodnotu,
- **range**, jednořádkové pole pro určitý rozsah číselných hodnot,
- **email**, jednořádkové pole pro e-mailové adresy,
- **url**, jednořádkové pole pro URL adresy,
- **search**, jednořádkové pole pro zadávání jedno nebo více vyhledávacích termínů,
- **tel**, jednořádkové pole pro telefonní čísla,
- **color**, ovládací prvek pro vložení barvy.

6.2 Rozšíření s ohledem na CSS3

Text v této sekci byl inspirován a i citován z článku [20] a příručky [21]. CSS3 je nástupcem CSS2.1, a i když není stále plně dokončen konsorciem W3C vývoj této technologie, i tak nejnovější webové prohlížeče tuto technologii podporují. V této sekci popíši pár nových vlastností technologie CSS3, které by byli vhodné pro rozšíření komponenty řízené pomocí CSS.

CSS3 vlastnosti

CSS3 nabízí spoustu nových vlastností oproti CSS2.1, zde popíši jenom ty podle mě nejvhodnější (nejpoužívanější) k rozšíření komponenty. Jsou to následující vlastnosti:

- **border-radius**, vlastnost umožní nastavit zaoblení objektu, vlastnost má více možností, lze nastavit každému rohu radius (zaoblení),
- **box-shadow**, vlastnost umožní stín objektu, má čtyři vlastnosti: první má na statrosti posunutí stínu od objektu, druhá naopak vertikálně, třetí hodnota nastavuje okraj stínu, který přechází do ztracena a čtvrtá je barva stínu,
- **text-shadow**, vlastnost je podobná předchozí vlastnosti, ale jde tady o stín textu,
- **transform**, vlastnost nastavuje rotaci a pohyb objektu, několik možných hodnot: translate, rotate a scale,

- **opacity**, průhlednost, vlastnost má pouze jednu hodnotu v intervalu ¡0, 1¿,
- Model barev **RGBA**, vlastnost nastavuje barvu s průhledností, u stávajícího rgb se přídává průhlednost (opacity),
- Model barev **HSL**, možnost definování barev, a to barvy (hue), sytosti (saturation) a světlost (lightness), existuje i vlastnost HSLA tzn. s průhledností (opacity),
- **background-size**, vlastnost pro změnu velikosti obrázku na pozadí elementu, zajištění, aby obrázek na pozadí stránky pokryl celou její plochu, zmenšení obrázku na pozadí tak, aby byl vždy vidět celý,
- **border-image**, obrázkový rámeček, způsob, jak namísto nativních okrajů kolem elementu vykreslit pomocí obrázku naše vlastní,
- **font-face**, vlastnost pro nastavení vlastního fontu do stránky,
- **multiple backgrounds**, což není přímo vlastnost CSS3, ale je to nová možnost již existující vlastnosti, a je to vrstvení více obrázků nebo barev na pozadí jednoho elementu.

V minulé kapitole 5 bylo poznamenáno, že v komponentě jsou implementované čtyři stlyly okrajů, tak se nabízí rozšíření zbylých stylů definovaných v CSS jako např. „ridge“, „groove“, „ridge“, „inset“ a „outset“.

Kapitola 7

Závěr

Tato bakalářní práce je zaměřená na vytvoření stroje, který by dokázal zobrazit HTML dokument podle definovaných kaskádových stylů (CSS). Neboli tato práce je určena k vytvoření komponenty (Java Swing komponenty), která by se řídila CSS a podle CSS zobrazila obsah HTML dokumentu, přitom k implementaci je použit jazyk Java a jeho rozhraní Swing. Důraz k vytvoření této komponenty byl kladen na správné a efektivní zobrazení CSS stylů elementů HTML a tvoření editovatelných prvků formuláře.

Abych tuto práci vytvořil, začal jsem nastudováním si všech potřebných teoretických znalostí (kapitola 2). V prvé řadě bylo seznámení se s implementačním jazykem Java a jeho vlastnostmi, s tímto jazykem jsem měl už nějaké zkušenosti, takže v tomto případě slovo opakování než nastudování jazyka je vhodnější. Ale bylo potřeba dostudovat si pár vlastností jazyku jako je např. reflexe, atd. Na toto samozřejmě navazuje rozhraní Swing, které bylo nutné také nastudovat, konkrétně jeho vlastnosti, použití, způsoby zobrazení, vykreslování, atd. V práci se pracuje s HTML dokumentem, tzn. že dalším cílem k studování byly značkovací jazyky a samotný jazyk HTML. A po tomto tématu následují kaskádové styly (CSS), kterým spolu s rozhraním Java Swing jsem věnoval nejvíce času. V komponentě je využit řetězec JSON, takže nastudování této věci bylo potřeba. Jedním s posledních a velmi důležitým předmětem k nastudování byl zobrazovací stroj *CSSBox*, který patří k jádru komponenty.

Po studii souvisejících technologií jsem přešel k tvoření konceptuálního návrhu komponenty (kapitola 4). Při tvorbě jsem vycházel z komponent rozhraní Swing a ze získávání informací renderovacím strojem *CSSBox*. Musel jsem nejdříve zjistit, jak pracuje stroj *CSSBox*, abych věděl, jak ho využít, protože od tohoto stroje to vše začíná. Dále u stroje *CSSBox* jsem zjišťoval, co přesně vytvoří stroj, abych zjistil, jak vhodně bych mohl navázat, jaký box by bylo vhodné reprezentovat komponentou rozhraní Swing. Po zjištění veškerých potřebných věcí kolem stroje *CSSBox* přišlo na řadu zkompletovat strukturu komponenty, tzn. jak se bude tvořit. K procházení tohoto „stromu boxů“ (výsledek stroje *CSSBox*) jsem se inspiroval od rozhraní `BoxRenderer`, které je součástí renderovacího stroje *CSSBox*, ale tímto rozhraním se neprochází celý strom, pokud se vyskytuje formulář v HTML dokumentu, tak se prochází zvlášť a najednou, protože jsem chtěl, aby jeho všechny editovatelné prvky byly pohromadě a přístupné přes element formuláře. Dalšími věcmi, které byly vyřešeny při koncepci, je pozicování jednotlivých komponent, které jsou pozicovány absolutně, jiná funkcionalita odkazů, obrázky, pozadí, měnění velikosti, atd. Bylo potřeba vytvořit vlastní třídu pro vykreslování okrajů (rámečku), protože CSS styly jsou velice flexibilní, při definici okrajů a jazyk Java s jejími rozhraními nemá přesnou vlastnost, která by se dala použít. Komponentu jsem se rozhodl nazvat *CSSComponent* po CSS stylech a komponen-

tech Java Swing.

Implementace konceptuálního návrhu následovala, začal jsem nejdříve u rendereru, který se stal kostrou komponenty. Nakonec nebylo potřeba využít všechny možné metody rozhraní `BoxRenderer` k implementaci komponenty. Každý HTML element je instancí třídy `CSSComponent`, která obsahuje všechny vlastnosti k vytvoření elementu HTML. Následovalo definování vlastností pomocí CSS stylů. Renderovací stroj *CSSBox*, rendereruje jakýkoli text obsažený v HTML dokumentu mimo element, vytváří instanci třídy `TextBox`, tak jsem se rozhodl to oddělit od elementů a implementovat jinou třídu pro zobrazovaní textu v komponentě a to `GTextBox`. Možnosti k vytvoření komponenty jsou dva a ke každému je potřeba instance třídy `Viewport`, a tak byla implementována třída se statickými metodami k získání tohoto objektu a to třída `BoxRendererFactory`.

K ověřování funkčnosti komponenty se zobrazovaly dokumenty v komponentě a porovnávali se zobrazováním dokumentu ve webovém prohlížeči. U komponenty je hodně možností k rozšíření, které jsem zmínil v kapitole 6. Týká se hlavně stroje HTML5, ale hlavní věcí k rozšíření by bylo o vlastnosti technologie CSS3.

Literatura

[1] Herout, P. *Učebnice jazyka Java.* Nakladatelství Kopp, 2001. ISBN 80-7232-115-3.

[2] Pitner, T. *Java - začínáme programovat.* Grada Publishing, 2002. ISBN 80-247-0295-9.

[3] Oracle and/or its affiliates. *Trail: Creating a GUI With JFC/Swing.* [online], [cit. 2014-05-19]. URL docs.oracle.com/javase/tutorial/uiswing.

[4] Fowler, A. *Painting in AWT and Swing.* [online], [cit. 2014-05-19]. URL http://www.oracle.com/technetwork/java/painting-140037.html.

[5] Bodnar, J. *Java Swing tutorial.* [online], [cit. 2014-05-19]. URL http://zetcode.com/tutorials/javaswingtutorial/.

[6] Oracle and/or its affiliates. *The Reflection API.* [online], [cit. 2014-05-19]. URL http://docs.oracle.com/javase/tutorial/reflect.

[7] Hruška, T., Burget, R. *Internetové aplikace (WAP) II. část SGML, HTML, CSS, DOM.* [online], 2006 - [cit. 2014-05-19]. URL https://www.fit.vutbr.cz/study/courses/WAP/private/opory/OporaWAP2SGMLHTMLCSSDOM.pdf.

[8] Staníček, P. *CSS Kaskádové styly.* Computer Press, 2003. ISBN 80-7226-872-4.

[9] Bos, B. et al. *Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification.* W3C Candidate Recommendation, 2007.

[10] Burget, R. *CSSBox - Java HTML rendering engine.* [online], 2007 - [cit. 2014-05-19]. URL http://cssbox.sourceforge.net.

[11] WWW stránky. *Introducing JSON.* [online], [cit. 2014-05-19]. URL http://www.json.org/.

[12] Firdaus, T. *10 Frameworks to Build Mobile Application with HTML, CSS & JavaScript.* [online], [cit. 2014-05-19]. URL http://www.hongkiat.com/blog/mobile-frameworks/.

[13] The jQuery Foundation. *JQuery Mobile demosl.* [online], [cit. 2014-05-19]. URL http://demos.jquerymobile.com/1.4.2/.

[14] Večeřa, Z., Lehocký, Z. *jQuery Mobile: udělejte si mobilní verzi.* [online], [cit. 2014-05-19].
URL http://programujte.com/clanek/2011112100-jquery-mobile-udelejte-si-mobilni-verzi/.

[15] Adobe Systems Inc. *PhoneGap Documentation.* [online], [cit. 2014-05-19].
URL http://docs.phonegap.com/en/3.4.0/index.html.

[16] The Apache Software Foundation. *Apache cordova.* [online], [cit. 2014-05-19].
URL http://cordova.apache.org/.

[17] Crockford, D. *JSON in Java.* [online], 2011.
URL http://www.json.org/java/index.html.

[18] Sládek, H. *Webdesignérův průvodce po HTML5.* [online], [cit. 2014-05-19].
URL http://www.zdrojak.cz/serialy/webdesigneruv-pruvodce-po-html5/.

[19] Šťastný, J., Šimeček, M. *HTML5 - nové vlastnosti.* [online], [cit. 2014-05-19].
URL http://programujte.com/clanek/2010082200-html5-nove-vlastnosti/.

[20] Šťastný, J., Šimeček, M. *CSS3 - držte krok s dobou (nové vlastnosti).* [online], [cit. 2014-05-19].
URL http://programujte.com/clanek/2010070801-css3-drzte-krok-s-dobou-nove-vlastnosti/.

[21] WWW stránky. *CSS3 příručka.* [online], [cit. 2014-05-19].
URL http://www.vzhurudolu.cz/prirucka/css3.

Příloha A

Obsah CD

- technická zpráva ve formátu PDF
- zdrojový tvar technické zprávy, zdrojové kódy latexu
- vygenerovaná dokumentace komponenty řízené pomocí CSS
- aplikace pro demonstraci komponenty řízené pomocí CSS
- zdrojové kódy komponenty řízené pomocí CSS

Adresářová struktura:

```
root/
├── CSSComponent/ ..... komponenta řízená pomocí CSS
│   ├── demo.bat ..... spuštění demo aplikace ve Windows
│   ├── demo.sh ..... spuštění demo aplikace v Linux
│   ├── build.xml ..... spuštění pomocí programu ant – ant DemoCSSComponent
│   ├── cssomponent.jar
│   ├── doc/ ..... dokumentace komponenty
│   ├── lib/ ..... knihovny
│   ├── src/ ..... zdrojové kódy
│   └── ...
├── doc/ ..... zdrojové soubory Latexu
├── zprava.pdf
└── readme.txt
```

Příloha B

Manual

Příklady vytvoření komponenty:

1. způsob

```
// ziskani objektu Viewport
Viewport vp = CSSRendererFactory.getViewportOfResources(
DemoCSSComponent.class, "html/index.html", 250, 200);
// vytvoreni komponenty
CSSComponent panel6 = new CSSComponent(vp);
```

2. způsob

```
// ziskani objektu Viewport
Viewport vp = CSSRendererFactory.getViewportOfResources(
DemoCSSComponent.class, "html/index.html", 250, 200);
// vytvoreni komponenty
JPanel panel = new JPanel();
final CSSRenderer renderer = new CSSRenderer
                          (panel, vp.getWidth(),vp.getHeight());
vp.draw(renderer);
renderer.close();
```

Příklady přistoupení k ovládacím prvkům formuláře:

```
//pristoupeni k formulari s id=menu
render.getForms().get("menu")
// pristoupeni k tlacitku id=item3
// ve formulari id=menu a pridani action listeneru
render.getForms().get("menu")
.getButtons().get("item3").addActionListener(actionList);
// pristoupeni k checkboxu id=reply
// ve formulari id=panelinputs a zjisteni jestli je zaskrnut
render.getForms().get("panelinputs")
.getCheckBoxes().get("reply").isSelected()
```

Nastavení barvy odkazu při přejetí myši:

```
// nastaveni barvy pozadi
render.setHoverBackground(Color.YELLOW);
// nastaveni barvy textu
render.setHoverForeground(Color.GREY);
```