VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

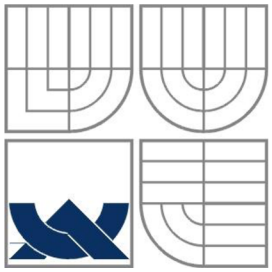
JEDNOTKOVÉ TESTOVÁNÍ ZOBRAZOVACÍHO JÁDRA PROHLÍŽEČE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

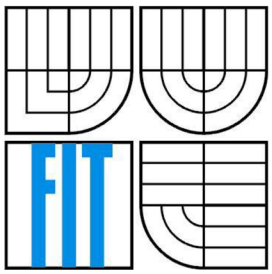
AUTOR PRÁCE
AUTHOR

MICHAL ŠŤASTNÝ

BRNO 2013



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

JEDNOTKOVÉ TESTOVÁNÍ ZOBRAZOVACÍHO JÁDRA PROHLÍŽEČE

UNIT TESTING OF A RENDERING ENGINE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MICHAL ŠŤASTNÝ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. RADEK BURGET, Ph.D.

BRNO 2013

Abstrakt

Tato bakalářská práce je zaměřena na jednotkové testování zobrazovacího jádra prohlížeče pomocí nástroje JUnit. V práci je popsána architektura zobrazovacího jádra CSSBox a seznámení se s nástrojem pro jednotkové testování JUnit. Dále je v práci uveden návrh rámce umožňující tvorbu testů včetně jeho implementace a implementace testů zvolených vlastností CSS. Na závěr jsou shrnuty výsledky těchto testů a návrh možných rozšíření.

Abstract

This bachelor thesis is focused on unit testing of a rendering engine using JUnit. The thesis decribes an architecture of a rendering engine CSSBox and familiarization with the tool for unit testing of JUnit. In the thesis is also suggested the framework allowing the creation of tests, including its implementation and implementation of the tests of selected properties of CSS. Finally, there are the evaluation of results of these tests and the suggestion of possible extensions.

Klíčová slova

CSS, HTML, DOM, Java, zobrazovací jádro, jednotkové testování, JUnit

Keywords

CSS, HTML, DOM, Java, rendering engine, unit testing, JUnit

Citace

Šťastný Michal: Jednotkové testování zobrazovacího jádra prohlížeče, bakalářská práce, Brno, FIT VUT v Brně, 2013

Jednotkové testování zobrazovacího jádra prohlížeče

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Radka Burgeta, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Michal Šťastný
15. května 2013

Poděkování

Dovoluji si tímto poděkovat vedoucímu této práce panu Ing. Radku Burgetovi, Ph.D. za odborné vedení, pomoc, rady, připomínky při zpracování bakalářské práce a čas věnovaný konzultacím.

Obsah

1 Úvod

S vytvářením projektů také vzniká potřeba tvorby testů, které zjistí správnost běhu aplikací. U tohoto projektu tomu není jinak. Na Fakultě informačních technologií, Vysokém učení technickém v Brně, vzniká od roku 2007 projekt s názvem CSSBox. CSSBox využívá dále čtyři další podprojekty. Všechny jeho části jsou implementovány v jazyku Java. S implementací tohoto projektu bylo tedy nutné vytvořit sadu testů ověřujících správnou funkci zobrazovacího jádra CSSBox. Všechny testy pro tuto aplikaci byly vytvořeny pomocí nástroje JUnit.

1.1 Cíle práce

Hlavním cílem této práce je otestování zobrazovacího jádra CSSBox. Prvním úkolem je seznámení se s nástroji pro jednotkové testování v jazyku Java, zejména se zaměřením na nástroj JUnit, prostudovat si dokumentaci a zjistit možnosti použití tohoto nástroje.

V další části je nutné se seznámit s architekturou samotného zobrazovacího jádra CSSBox včetně podprojektů. A to převážně pomocí manuálu a dokumentace, které jsou přístupné na stránkách projektu. Pro tento úkol je nutné stáhnout CSSBox ze stránek projektu [5] a zprovoznit je v IDE[1].

Dalším úkolem je navrhnout rámec umožňující tvorbu testů prověřujících jednotlivé části zobrazovacího jádra. Pro tento úkol je třeba znát jakou verzi CSS podporuje zobrazovací jádro CSSBox. Pro návrh je dobré využít seznam testů, které byly využity při testování CSS1.

Na základě předchozích úkolů můžeme implementovat navržený rámec.

Výstupem práce jsou implementované testy ověřující podporu vybraných vlastností CSS zobrazovacím jádrem. K dispozici je i jednoduchý web (testovací rámec), na kterém byly testy prováděny. Po vyhodnocení testů jsou navržena možná rozšíření testů.

1.2 Struktura dokumentu

Práce je členěna do sedmi kapitol. Po úvodní kapitole přichází na řadu kapitola 2, která se zabývá CSS a zobrazovacím jádrem CSSBox. V rámci této kapitoly je popsána historie CSS, využití, výhody a nevýhody. Dále jsou zde popsány základní informace o CSSBox a jeho struktura.

V kapitole 3 je vysvětleno jednotkové testování a nástroj JUnit včetně vytváření testů pomocí tohoto nástroje. Kapitola 4 se zabývá návrhem rámce pro tvorbu testů. Kapitola 5 pak obsahuje implementaci řešení, výčet postupů při řešení projektu a dále zajímavá místa implementace.

V posledních kapitolách 6 a 7 jsou rozebrány výsledky práce, které byly zjištěny testováním a možnosti rozšíření testů.

[1] IDE-vývojové prostředí je software usnadňující práci programátorů obsahující editor, kompilátor, interpret a debbuger

2 CSS a zobrazovací jádro CSSBox

Jak již název této práce napovídá, jejím hlavním cílem je seznámit se a otestovat zobrazovací jádro CSSBox. V této kapitole se nejdříve seznámíme s CSS, jeho historií, jednotlivými verzemi a celkovým vývojem. Dále jaké jsou výhody a nevýhody použití kaskádových stylů. V další části se seznámíme se samotným renderovacím nástrojem CSSBox, jeho strukturou a knihovnami, které využívá. Je zde zmíněno i pár příkladů použití CSSBox, které jsou součástí balíčku, jenž lze stáhnout z webu.

2.1 CSS – kaskádové styly

CSS vzniklo v roce 1996. Zkratka CSS znamená Cascading Style Sheets, česky „kaskádové styly". Kaskádové styly je jazyk popisující způsob zobrazení dokumentů napsaných v jazycích (X)HTML nebo XML. Jazyk byl navržen a standardizován konsorciem W3C. Styly je možné seskupit do účelových šablon. Šablony jsou jako pracovní ohrádky sdružující styly, které slouží k určitému záměru. Díky šablonám je možné aplikovat na dokument hned několik stylů najednou a změnit tak veškeré formátování v dokumentu jedním krokem. Tímto způsobem je možné dokumenty formátovat pro nejrůznější účely. Kaskádovost stylů je dána hierarchickým uspořádáním pravidel pro styly. Každý podřízený styl předefinuje stejnojmenná nadřazená pravidla pro jistý prvek.[1]

Existují tři úrovně kaskádových stylů, které jsou popsány níže. Verze CSS2.1 byla dokončena v roce 2011 a nyní se pracuje na verzi CSS3.

Úroveň CSS, kterou používáme, není třeba nijak definovat. Většina moderních prohlížečů sice kaskádové styly podporuje, ale úroveň jejich podpory je mezi nimi dramaticky odlišná.

Hlavní výhodou stylů je to, že stačí na jednom místě změnit definici a výsledná změna se projeví u všech prvků, které daný styl používají. Při nastavování prvků zvlášť by však bylo třeba nastavit jednotlivě všechny příslušné prvky. Styly jsou tedy snadným prostředkem k aktualizacím formátování dokumentu a udržení konzistentnosti ve všech dokumentech. Další výhodou je „Cachování stylů" neboli prohlížeč si může soubor se styly uložit do Cache paměti, čímž může být dosaženo rychlejšího načtení stránky. Z jiného pohledu při použití externího CSS souboru dochází k dalšímu http požadavku, navíc oproti tomu, když bychom použili buď online CSS nebo přímo formátování HTML.[1][4][9]

Závažnou a hlavní nevýhodou CSS je ne vždy dostatečná podpora v majoritních prohlížečích. Prohlížeče obsahují v implementaci CSS chyby, a proto je občas nesnadné napsat kód tak, aby se ve všech prohlížečích výsledek zobrazil stejně. Situace se poslední dobou zlepšuje opravami chyb v prohlížečích.[1]

2.1.1 CSS1

První verze CSS , která byla oficiálně vydána konsorciem W3C v roce 1996 zavedla syntax, který je používaný ve všech následujících verzích. Dále definuje základní funkce stylů s omezenou podporou písma a nastavování pozic prvků. V této verzi byly podporovány: vlastnosti písma, barvy textu a pozadí, vlastnosti textu, vlastnosti blokových elementů, způsoby zobrazení prvků a řízení dispozice.

2.1.2 CSS2 a CSS2.1

CSS2 byla vyvinuta a zveřejněna konsorciem W3C v roce 1998. Je nadmnožinou CSS1. Obsahuje navíc sluchové vlastnosti, stránkovaná média a lepší podporu písem a nastavování pozic prvků. Rovněž byla vylepšena spousta dalších vlastností.

V roce 2004 vyšla verze CSS2.1, která řeší některé chyby CSS2. Tato verze je odvozena z verze CSS2, některé části mění a některé byly odstraněny. Nejdůležitější části byly nově definovány. Dále byly přidány nejvíce žádané funkce, které již byly široce implementovány. Ze všeho nejvíce CSS2.1 představuje obrázek použití CSS.

2.1.3 CSS3

Verze CSS3 se pojí se standardem HTML5, který je má plně využívat. Dokončení této verze je plánováno na rok 2015, ale většina vlastností je podporována již dnes. Obsahuje navíc vlastnosti určené pro prezentaci, díky čemuž je možné z webových dokumentů efektivně vytvářet prezentace. Dále přináší vlastnosti jako nové barevné modely RGBA, HSL a HSLA nebo průhlednost a další. To je jen nástin nových vlastností CSS3.

2.2 Základní informace o CSSBox

CSSBox je (X)HTML/CSS renderovací nástroj vyvinutý a testovaný v jazyku Java. Tento nástroj vzniká a vyvíjí se na Fakultě informačních technologií, Vysokém učení technickém v Brně, od roku 2007. Primárním účelem je poskytnout kompletní zpracovatelné informace o zobrazované stránce, jejím obsahu a dispozici. Může být však použit pro prohlížení zobrazovaných dokumentů v Java Swing aplikacích.

Vstupem renderovacího nástroje je dokument v DOM stromu a sada stylů vztahujících se k dokumentu. Výstupem je objektově orientovaný model uspořádání stránky. Tento model může být zobrazen přímo. Hlavně je však vhodný pro další zpracování algoritmů analýzy zobrazení, jako je například segmentace stránek nebo algoritmy extrakce informací. Jádro CSSBox může být také použito pro získání bitmapy nebo vektoru (SVG) obrázku vykresleného dokumentu. Pomocí Swing Box balíčku může být CSSBox použit jako interaktivní součást webového prohlížeče v aplikaci Java

Swing. Stroj je dále schopen automaticky načíst styly odkazované v dokumentu a vypočítá efektivní styl každého elementu. Později je vypočítáno rozložení dokumentu.[5]

Nástroj CSSBox je v současné době vyvíjen a testován v Javě 1.6. CSSBox využívá knihovnu jStyleParser pro parsování CSS souborů. Momentálně je knihovna testována s CSSBox vloženém v distribuovaném balíčku. Dále je v balíčku obsažena knihovna NekoHTML. Tento nástroj je využíván jako jednoduchý HTML skener a tag balancer, který umožňuje programátorům analyzovat HTML dokumenty a přístup k informacím pomocí standardního XML rozhraní. NekoHTML také dokáže přidat chybějící rodičovské elementy a automaticky přidat koncové elementy.[10]

CSSBox je dostupný pod licencí GNU Lesser General Public Licence verze 3.0.

2.2.1 Příklady použití CSSBox

Několik příkladů použití renderovacího nástroje je součástí distribučního balíčku.

ComputeStyles vypočítá efektivní styl každého elementu a dekóduje styly atributu uvnitř elementu. Modifikovaný HTML dokument je uložen do výstupního souboru.

TextBoxes renderuje dokument a tiskne seznam textových boxů společně s jejich pozicí na stránce.

SimpleBrowser je jednoduchý prohlížeč, ve kterém se staticky zadá URL adresa požadovaného webu, který je poté zobrazen.

BoxBrowser je prohlížeč, který zobrazuje nejen požadovanou stránku, ale také DOM strom a Box strom. Mimo jiné také styly jednotlivých elementů.

ImageRenderer renderuje dokument a ukládá výsledek do bitmapy nebo vektorového obrázku (v současné době jsou PNG a SVG obrazy podporovány).

2.3 Struktura zobrazovacího jádra

2.3.1 Načtení dokumentu

Nástroj CSSBox očekává na vstupu DOM reprezentovaný kořenovým elementem. V příkladech, které jsou součástí CSSBox, je NekoHTML parser používán pro budování DOM dokumentu. Je založen na XERCES 2, ale mohou být použity i jiné DOM analyzátory nebo může být použit jakýkoliv jiný

způsob získání DOM dokumentu. Pro dema, která jsou součástí balíčku, jsou vytvořeny jednoduché DOMSource třídy, které zahrnují základní operace s analyzátorem, jako je inicializace a získávání DOM. Když je jiný DOM parser požádán pro použití v CSSBox, tak je nutná pouze jediná změna v re-implementaci DOMSource třídy, aby bylo možné sledovat zvolený analyzátor.[5]

> **DOM (Document Object Model)** neboli objektový model dokumentu je objektově orientovaná reprezentace XML nebo HTML dokumentu, která je platformově a jazykově nezávislá. Toto rozhraní umožňuje programům a skriptům přistupovat k obsahu XML nebo HTML dokumentů. Taktéž umožňuje modifikovat obsah, strukturu a styl těchto dokumentů. DOM umožňuje přístup k dokumentu jako ke stromu, což je zároveň datová struktura používaná ve většině XML analyzátorů a XSL. Specifikace W3C DOM jsou rozděleny do několika úrovní. Každá z těchto úrovní obsahuje povinné a volitelné moduly. V současnosti existují tři úrovně.[4]

2.3.2 Získání rozložení

Grafickým objektem *BrowserCanvas* jsou reprezentovány nástroje rozložení. Nejjednodušší cesta pro vytvoření rozložení na požadované rozměry je pomocí konstruktoru *BrowserCanvas*. Pak je rozložení počítáno automaticky vytvořením instance tohoto objektu. Zbývající argumenty konstruktoru jsou kořenový DOM element, DOM analyzer (používá se pro získání prvku), styly a základní adresa dokumentu. Je-li nutné další nastavení prohlížeče, může být *BrowserCanvas* vytvořen bez uvedení rozměrů výřezu. Pak se rozložení počítá automaticky a musí být vytvořen na výzvu metody *createLayout()*. Před nastavením výzvy můžeme nastavení prohlížeče změnit.

2.3.3 Zobrazení dokumentu

Třída *BrowserCanvas* je přímo odvozena ze třídy Swing javax.swing.JPanel. Proto může být použita jako rozhraní Swing. Velikost komponenty je automaticky upravena podle výsledného rozložení dokumentu. Základní dokument zobrazení je uveden v demo příkladu *SimpleBrowser*. *BrowserCanvas* poskytuje jednoduché zobrazení renderované stránky bez interaktivních prvků. Pro získání interaktivního prohlížeče, což je komponenta vybraného textu s odkazy, bylo použito rozšíření SwingBox.

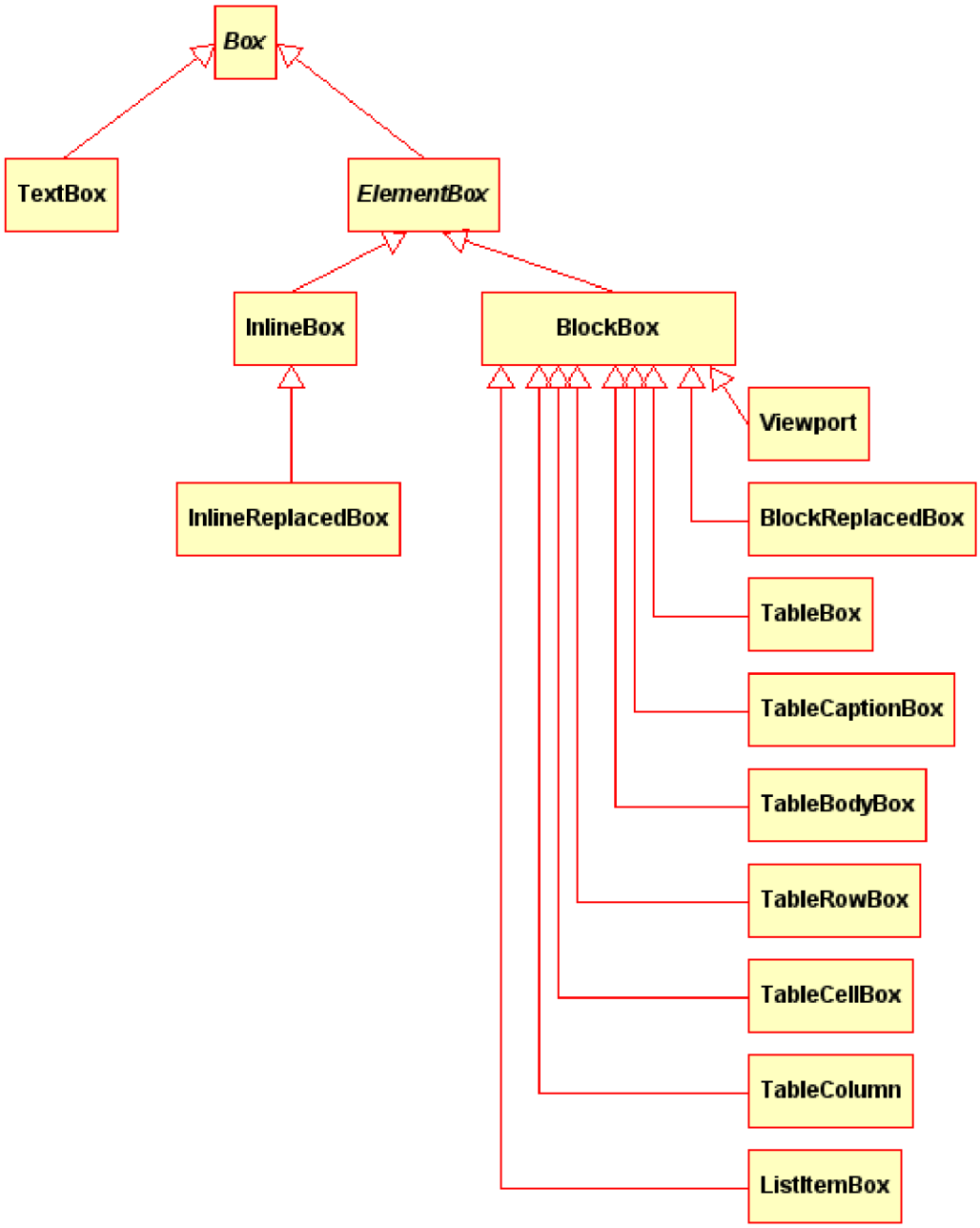
2.3.4 Možnosti konfigurace

Běžné nastavení prohlížeče je reprezentováno použitím objektu *BrowserConfig*, který může být přístupný pomocí metody *getConfig()*. Lze nakonfigurovat, zda bude obsah odkazovaných snímků načten automaticky. Ve výchozím stavu je toto nastaveno. Dále lze nastavit automatické načtení

pozadí z CSS. Toto je opět nastaveno ve výchozím stavu. Umožňuje konfiguraci, zda může nástroj používat HTML rozšíření nebo ne.

2.3.5 Renderovaný dokument model

Výsledný vzhled dokumentu je reprezentován jako strom boxů. Každý box vytváří obdélníkovou oblast ve výsledné stránce a odpovídá určitým HTML elementům. Může zde být více boxů odpovídajících jednomu prvku. Každý box je reprezentován objektem, který rozšiřuje abstraktní třídu. Existuje několik typů boxů, které přibližně odpovídají CSS hodnotě zobrazené pro daný prvek.



Obrázek 2.1: Hierarchie typu Box. Převzato z [5]

TextBox obvykle koresponduje s DOM uzlem typu Text. Je reprezentován jako objekt TextBox. Pokud je text rozdělen na několik řádků, tak násobné boxy korespondují s jednotlivými DOM uzly.

ElementBox koresponduje s DOM uzlem typu Element a je reprezentován objektem a abstraktní třídou ElementBox. Je implementován jako InlineBox nebo BlockBox. Každý element může obsahovat n potomků, kteří jsou indexováni od 0 do n. Pokud existují násobné ElementBoxy, které odpovídají jednomu DOM uzlu, tak všechny sdílejí potomky.

InlineBoxy jsou elementy, které mají vlastnost CSS display: inline. Tyto boxy jsou reprezentovány objekty InlineBox. Nemají žádné zvláštní vlastnosti, kromě vlastností definovaných ve třídě ElementBox.

BlockBoxy mají oproti InlineBoxům nastavenu vlastnost display: block. Jsou reprezentovány objektem BlockBox. Kromě toho jsou některé třídy odvozeny od této. Jsou to např. ListItemBox, TableBox aj. Tyto objekty se liší především ve způsobu, jakým jsou boxy a jejich obsah vytyčen na stránce. Z výsledného rozložení modelu pohledu nemá žádné zvláštní vlastnosti kromě vlastností definovaných ve třídě ElementBox.

Fonty a barvy - Pro některé boxy objekt *VisualContext* definuje a shromažďuje informace o druzích písma. Tento objekt je využíván metodou *getVisualContext()*.

2.4 Podprojekty

2.4.1 jStyleParser

jStyleParser je nástroj určený pro analýzu CSS vyjádření, který je vytvořený v jazyku Java. Má své vlastní aplikační rozhraní, které je navrženo pro efektivní zpracování CSS v Javě a mapování hodnot do datových typů užívaných v jazyku Java. Analyzuje styly verze CSS2.1 do struktur dříve než mohou být efektivně přiřazeny DOM elementům. Je primárně určen jako CSS analyzátor, který je použit jako knihovna pro CSSBox renderovací nástroj. Chyby jsou zjišťovány podle specifikace CSS.

2.4.2 SwingBox

SwingBox je komponenta Java Swing, která umožňuje zobrazení (X)HTML dokumentů včetně podpory CSS stylů. Je vytvořena jako JEditorPane komponenta s výrazně lepším vykreslováním výsledků. SwingBox je implementován čistě v Javě a užívá jej CSSBox renderovací nástroj pro vykreslování dokumentů.

2.4.3 Pdf2Dom

Pdf2Dom je PDF analyzátor, který převádí dokumenty do HTML DOM reprezentace. Získaný DOM strom může být serializován do HTML souboru nebo dále zpracován. Vložené CSS definice obsažené ve výsledném dokumentu se používají pro vytváření HTML stránky jako možný PDF vstup. Nástroj pro převedení PDF dokumentu do HTML je součástí distribučního balíčku. Pdf2Dom může být také použit jako nezávislá knihovna v jazyku Java se standardním DOM rozhraním pro DOM aplikaci nebo jako alternativní analyzátor pro CSSBox renderovací nástroj za účelem zpracování PDF v CSSBox.

2.4.4 WebVector

WebVector slouží pro převod dokumentu z HTML do SVG nebo PNG formátu. Převádí HTML dokument do vektorového obrázku ve formátu SVG nebo do bitmapového obrázku ve formátu PNG. Standard Vector Graphics (SVG) soubory mohou být upraveny různými vektorovými editory jako je Inkscape.

3 Jednotkové testování a nástroj JUnit

Cílem této kapitoly je seznámit se s jednotkovým testováním a nástrojem pro jednotkové testování JUnit. V první části této kapitoly je popsáno samotné jednotkové testování. Co to vlastně je, jeho podstata, význam, výhody a nevýhody. Ve druhé části je vysvětlen nástroj pro jednotkové testování JUnit jeho výhody i nevýhody. Dále jsou zde popsány základní prostředky pro vytváření testů v tomto nástroji.

3.1 Jednotkové testování

Jednotkové testování je část programu vytvořená vývojáři, pro otestování určité oblasti funkcionality testovaného programu. Obvykle jednotkové testy testují metody v určitém kontextu. Jednotkové testy provádí prokázání toho, že část kódu provádí to, co si vývojář myslí, že provádět má. V ideálním případě je každý testovaný případ nezávislý na ostatních, a proto se při testování snažíme testovanou část izolovat od ostatních částí programu. V některých případech jsou za tímto účelem vytvářeny pomocné objekty, které simulují předpokládaný kontext, ve kterém testovaná část pracuje.[11]

3.1.1 Výhody

Výhoda jednotkového testování je v tom, že lze izolovat jednotlivé části programu a zjistit zda pracují korektně. Jednotkové testy poskytují přesný výsledek, který musí určitá část kódu splňovat. V důsledku toho poskytuje několik výhod.[12]

Včasné nalezení chyb

Jednotkové testování nalezne problémy velmi brzo ve vývojovém cyklu. V programování řízeném testy (test-driven development – TDD[2]), které je často používáno v souvislosti s Extrémním programováním[3] a vývojem SCRUM, jsou jednotkové testy vytvářeny předtím, než je samotný kód napsán. Pokud kód testy úspěšně projde, tak je považován za splněný. Stejné jednotkové testy jsou opět často používány, když se vytvoří větší část kódu, a poté se pomocí automatizovaného procesu sestavení pozmění. Pokud nejsou testy splněny, víme kde je chyba a její nalezení a napravení je jednodušší. V případě nalezení chyby jednotkovými testy je stále brzy v procesu vývoje.

[2] TDD je přístup k vývoji software, který je založen na malých, stále se opakujících krocích vedoucí k zefektivnění celého vývoje.

[3] Extrémní programování je agilní metodologie vývoje software. Jedná se o tradiční činnosti, které jsou však dovedeny do extrému.

Zjednodušuje integrace

Jednotkové testování může snížit nejistotu v samotných jednotkách a může být použito v přístupu zdola-nahoru. Při testování se testují nejdříve části programu, poté větší části a na závěr celek. Tímto se stává integrace mnohem jednodušší. Propracovaná hierarchie jednotkových testů není totožná s testováním integrace. Integrace s periferními jednotkami by měla být zahrnuta do integračních testů, ale ne k jednotkovým testům. Integrační testování se stále spoléhá na testování člověkem, tedy „ručně“. Na vysoké úrovni globálního testování může nastat problém testy automatizovat. Proto se manuální testování často jeví jako rychlejší a levnější varianta.

Dokumentace

Vývojáři, kteří chtějí zjistit, jaké funkce jsou poskytovány, jak lze jednotkové testy používat, mohou základním znalostem porozumět díky API[4]. V daném případě jednotkový test ztělesňuje vlastnosti, které rozhodují o úspěchu testu jednotky. Tyto jednotky mohou indikovat vhodné i nevhodné použití jednotky stejně jako negativní chování, které má být zachyceno. Jednotkový test sám osobě dokumentuje tyto důležité vlastnosti. Naopak obyčejná dokumentace je náchylnější na to, aby byla postupem času zastaralá.

Usnadňuje změny

Jednotkové testování umožňuje programátorovi pozdější změnu kódu a ujišťuje se, zda modul stále správně funguje. Postupem času lze napsat testy pro všechny funkce a metody tak, že kdykoliv změna způsobí poruchu, mohou být chyby rychle identifikovány a opraveny. Dále umožňuje použití dostupných jednotkových testů tak, aby bylo snadné zjistit, zda kus kódu pracuje stále správně. Za stálého prostředí jednotkové testy prostřednictvím vlastní práce trvalé údržby budou i nadále přesně odrážet předpokládané použití spustitelného kódu i přes jakékoliv změny, v závislosti na stanovených rozvojích postupů a pokrytí testy.

Návrh

Pokud je software ve vývoji, tak je používán přístup nazvaný programování řízené testy. Každý jednotkový test může vypadat jako navržený prvek s uvedením třídy, metody a pozorovaného chování. Dále postrádá některé příslušnosti diagramu, ale

[4] API – application programming interface označuje v informatice rozhraní pro programování aplikací. Jde o sbírku procedur, funkcí, tříd či protokolů nějaké knihovny.

UML diagramy jsou snadno generovány pro většinu moderních jazyků pomocí volně dostupných nástrojů, které jsou obvykle k dispozici jako rozšíření IDE.

Parametrizace jednotkových testů

Parametrizované jednotkové testy (PUTs – Parametrized Unit Tests) jsou testy, které pracují s parametry. Na rozdíl od tradičních jednotkových testů, které jsou obvykle uzavřené metody, PUTs přijímají veškeré parametry. PUTs jsou podporovány JUnit 4 a dalšími .NET testovacími rámci. Vhodné parametry pro jednotkové testy mohou být dodány ručně nebo v některých případech automaticky generovány testovacími rámci. Existují také nástroje pro generování testovacích vstupů pro výstup jako např. QuickCheck.

Separace rozhraní od implementace

Některé třídy mohou mít reference i na jiné třídy, díky čemuž mohou tyto třídy zahrnout do testování. Běžným příkladem jsou třídy, které jsou závislé na databázi. Za účelem testování třídy často píše tester kód, který spolupracuje s databází. To je chyba, protože test jednotky zpravidla nesmí jít ze své vlastní třídní hranice a zejména by neměl překročit tuto proces/síťovou hranici. Toto může představovat nepřijatelné problémy s výkonem na jednotku testu. Překračování hranice změní jednotkové testy do integračních testů a testovacích případů, kdy selžou. Z tohoto důvodu je méně jasné, která složka je příčinou selhání. Místo toho by vývojář softwaru mohl vytvořit abstraktní rozhraní kolem databázových dotazů, a pak implementovat toto rozhraní s vlastním objektem. Tím, že odstraní potřebnou přílohu z kódu (dočasně snižuje efektivitu spojení), může být samostatná jednotka důkladně testována. To má za následek vyšší kvalitu jednotky, která je více udržovatelná.

3.1.2 Nevýhody

Mezi nevýhody jednotkového testování patří to, že už podle definice testují pouze funkčnost samotných jednotek. Proto nedokáže zachytit integrace chyby nebo chyby na širší úrovni systému. Jednotkové testování by mělo být prováděno ve spojení s ostatními činnostmi testování softwaru, protože mohou zobrazit pouze přítomnost nebo nepřítomnost určitých chyb. Nemohou prokázat absenci chyb s cílem zajistit správné chování pro každou cestu, provedení všech možných vstupů a

zajistit absenci chyb, kde jsou vyžadovány ostatní techniky. Například uplatnění formálních metod pro prokázání, že softwarová komponenta nemá neočekávané chování.

Dalším problém souvisí se samotným psaním jednotkových testů. Je obtížné nastavení reálného a účelného testu. Je třeba vytvořit příslušné počáteční podmínky tak, aby se část zkoušené aplikace chovala jako součást celého systému. Pokud nejsou tyto podmínky nastaveny správně, test nemůže být proveden v realistickém kontextu a snižuje hodnotu a přesnost výsledků jednotkových testů. [12]

3.2 JUnit

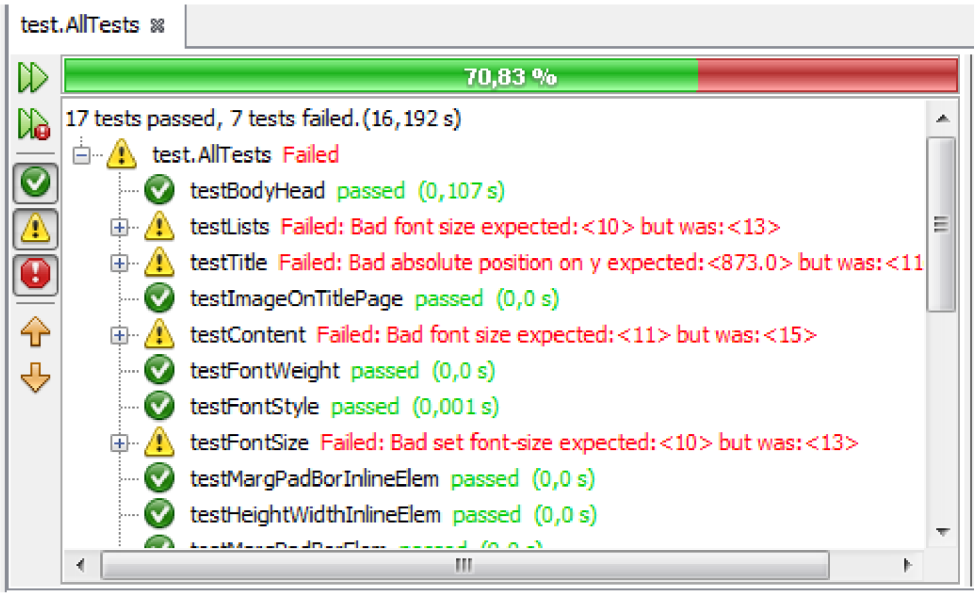
JUnit je jednoduchý rámec pro tvorbu testů v jazyku Java. JUnit je důležitý pro vývoj typu programování řízené testy a patří do rodiny rámců jednotkového testování, který je kolektivně známý jako xUnit, jenž vznikl z SUnit. Pomocí JUnit můžeme levně a postupně vybudovat testovací sadu, která pomůže změřit pokrok při vývoji. Na vývoji JUnit se podíleli vývojáři Kent Beck a Erich Gamma. Aktuálně používanou verzí je verze 4.11. [2]

JUnit poskytuje rámce pro provádění zkoušek.
Poskytuje:

- Poskytuje šablony pro psaní testů s nastavením, realizací a tearDown
- Umožňuje organizovat zkoušky v hierarchii
- Umožňuje automaticky a snadno provádět testy
- Odděluje zkušební zprávy z provedení, což umožňuje používat různé TestRunners se stejným TestSuite

Ačkoliv je JUnit silný ve své jednoduchosti, jiné nástroje mají spoustu příležitostí k vyplnění mezer. JUnit nepodporuje následující vlastnosti:

- Automaticky generovat testy pro testované jednotky
- Zajistit pokrytí metrik
- Uvést kdy byl špatný test vytvořen



Obrázek 3.1: Příklad zobrazení výsledků testů JUnit v NetBeans IDE

3.3 Vytváření testů pomocí JUnit

3.3.1 Anotace

Následující anotace jsou dostupné v JUnit 4.x[8]

@Test

public void nazevMetody() - Tato anotace identifikuje, zda je daná metoda testovací. Je to metoda typu public

@Before

public void nazevMetody() - Tato metoda je prováděna před každým prováděným testem. Může připravit testovací prostředí.

@After

public void nazevMetody() - Je to metoda prováděná po provedení každého testu. Tato metoda čistí testovací prostředí. Může být také použita pro vyklizení paměťových struktur.

@BeforeClass

public static void nazevMetody() – Tato metoda je prováděna pouze jednou a to před zahájením provádění testů. Může být použita k provedení časově náročné aktivity jako je například připojení k databázi. Tato metoda musí být definována jako *static*, aby mohla pracovat s JUnit.

@AfterClass

public static void nazevMetody() – Tato metoda může být použita pouze jednou a to po provedení všech testů. Dále může být použita pro úklid jako je například odpojení od databáze. Metoda musí být definována jako *static*, aby mohla pracovat s JUnit.

@Ignore – Tyto testovací metody jsou ignorovány. Toto je užitečné v případě, že testovací metoda není hotova nebo daná část projektu není hotova a nemůže být testována.

@Test (expected = Exception.class) – Test není úspěšný pokud metoda nevyvolá výjimku.

@Test (timeout = 10) – Test není úspěšný pokud je jeho délka trvání delší než 10 milisekund.

3.3.2 Asserty

JUnite poskytuje statické metody ve třídě Assert pro testování určitých podmínek. Sada metod typu assert je užitečná pro psaní testů. Metody umožní v případě neúspěchu tisk chybového hlášení a oznámí očekávaný a chybný výsledek. Z těchto metod je nejčastěji používanou metoda *assertEqual()*. Dále jsou popsány některé metody z této třídy.[8]

assertsEquals(message, expected, actual) – Porovnává dvě hodnoty, zda jsou shodné. V případě neúspěchu tiskne chybovou zprávu a zobrazí aktuální a očekávanou hodnotu.

assertsEquals(message, expected, actual, tolerance) - Porovnává dvě hodnoty, zda jsou shodné. Tolerance je desetinná hodnota, o kterou se může lišit očekávaná a aktuální hodnota. V případě neúspěchu tiskne chybovou zprávu a zobrazí aktuální a očekávanou hodnotu.

assertTrue(message, condition) – Ověřuje, zda vrácená hodnota typu boolean je true.
assertFalse(message, condition) – Ověřuje, zda vrácená hodnota typu boolean je false.

assertSame(message, expected, actual) – Ověřuje, zda obě proměnné ukazují na stejný objekt.

assertNotSame(message, expected, actual) – Ověřuje, zda proměnné expected a actual neukazují na stejný objekt.

assertNull(message, object) – Ověřuje, zda hodnota object je *null*.

assertNotNull(message, object) - Ověřuje, zda hodnota object není *null*.

fail(message) – Test skončí neúspěchem a vrací chybové hlášení. Může být použita pro kontrolu, že určité části kódu není dosaženo, nebo že je test vadný před provedením testu implementovaného kódu.

3.3.3 Parametrizovaný test

JUnit umožňuje použití parametrů v testovací třídě. Tato třída může obsahovat jednu testovací metodu, která je prováděna s různými parametry. Testovací třídu označíme jako parametrizovaný test s *@RunWith(ParametrizovanaTrida.class)* anotací. Testovací třída tohoto typu musí obsahovat statickou metodu s anotací *@Parameters*, která generuje a vrátí kolekci polí. Každá položka v této kolekci se používá jako parametr pro zkušební metodu. Musí být také vytvořen konstruktor, ve kterém se budou ukládat hodnoty pro každý test. Počet prvků v každém poli poskytovaném metodou s anotací *@Parameters* musí odpovídat počtu parametrů v konstruktoru třídy. Třída je vytvořena pro každý parametr a testovací hodnoty jsou předány prostřednictvím konstruktoru. [2]

3.3.4 Pravidla

Prostřednictvím anotace *@Rule* můžete vytvářet objekty, které lze použít pro nastavování testovacích metod. Pravidla umožňují velice flexibilní přidávání nebo předefinování chování každé testovací metody. Testeři mohou znovu využít nebo rozšířit jedno z uvedených pravidel.[8]

4 Návrh a implementace rámce pro tvorbu testů

V této kapitole se budeme zabývat vlastnostmi, které byly vybrány pro testování zobrazovacího jádra. Dále je zde rozebrán návrh samotného rámce a rozvržení jednotlivých vlastností do HTML dokumentů. Na závěr této kapitoly je zmíněna implementace rámce včetně prostředků, které byly použity.

4.1 Testované vlastnosti

Při návrhu rámce bylo nutné nejdříve vybrat ty vlastnosti CSS, které jsou nejdůležitější při zobrazování stránek. Proto jsem se při výběru testovaných vlastností inspiroval vlastnostmi, které byly testovány u CSS1. Dále jsem vyhledal seznam vlastností, které se testují při ověřování podpory CSS3. Poté bylo nutné z těchto seznamů vlastností vybrat ty vlastnosti, které lze pomocí JUnit otestovat.

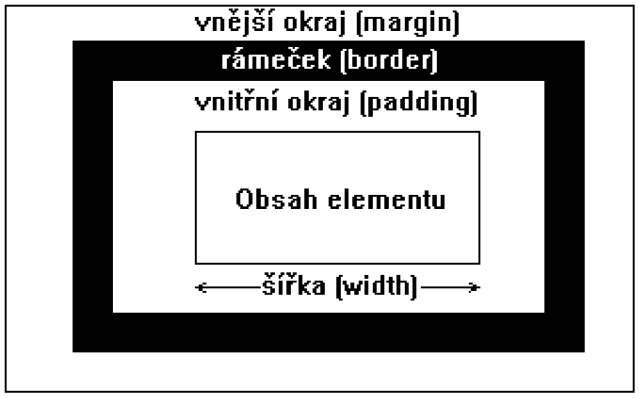
Vlastnosti písma

Z CSS vlastností zastupujících tuto skupinu jsem vybral čtyři vlastnosti a to *font-style*, *font-weight*, *font-size* a *font-variant*. Pomocí vlastnosti *font-style* můžeme nastavit písmo na kurzivu, skloněné písmo nebo nechat v základním nastavení, tedy, že se zobrazí normálně. CSS vlastnost *font-weight* nastavuje tučnost písma neboli duktus. Písmo lze nastavit na normal, kdy nebude tučné. Dále na bold, kdy je písmo tučné anebo nastavit pomocí hodnot od 100 do 900, kde 100 je co nejslabší (nejméně tučné) a 900 bude písmo co nejtučnější. Jsou zde ještě dvě možnosti nastavení a to bolder, kdy je písmo o něco tučnější než při normal a lighter nastavuje písmo o něco slabší než při normal. Bohužel ne všechny prohlížeče zvládnou zobrazit všechny stupně tučnosti, a proto se zde testují pouze bold a normal. Další vybranou vlastností je *font-size*. Tato vlastnost určuje velikost písma. V testovacím rámci je určována velikost písma pomocí tří jednotek a to v pixelech (px), typografických bodech (pt) a velikosti písmena M (em). Poslední testovanou vlastností je *font-variant*, která nastavuje písmo na kapitálky.

Vlastnosti obdélníku (Box model)

Z CSS vlastností, které pro toto lze využít to jsou vlastnosti: *margin*, *padding a border*. U obou těchto vlastností jsou hodnoty v testovacím rámci zadávány v pixelech (px) a typografických bodech (pt). První z těchto vlastností *margin*, určuje šířku vnějšího okraje prvku. Lze ji zadávat jako čtveřici horní, pravý, dolní a spodní okraj, ale také lze nastavit přímo vlastnost daného

okraje jak např. *margin-top* (tímto je nastavena vlastnost horního okraje). Další vlastností je *padding*, která určuje šířku vnitřního okraje prvku. Nastavení této vlastnosti je stejné jako u předchozí. Poslední vlastností patřící do této skupiny je vlastnost *border*. Je to CSS vlastnost určující všechny vlastnosti rámečku najednou. Můžeme ji také rozepsat do vlastností *border-width*, *border-style* a *border-color*. Vlastnost *border-width* může jako předchozí vlastnosti padding a margin obsahovat až čtyři hodnoty. Navíc kromě toho lze zadat jedno z klíčových slov určující šířku. Vlastnost *border-style* udává styl rámečku. *Border-color* může opět obsahovat až čtyři hodnoty typu barva.[7]



Obrázek 4.1: Formátování prvků. Převzato z [9]

Formátování modelu

Formátování modelu lze rozdělit na vertikální a horizontální formátování. Obě formátování k tomuto využívají vlastností *margin* a *padding*, které byly zmíněny ve vlastnostech obdélníku. Dále do této skupiny patří plovoucí elementy, řádkové elementy a nahrazované elementy. Poslední testovanou vlastností z této skupiny je šířka linky. Tato vlastnost je v testovacím rámci nastavována pouze v pixelech.

Barvy a vlastnosti pozadí

V této skupině jsou pouze dvě vlastnosti, které byly vybrány a je možné je testovat pomocí JUnit. Jsou to vlastnosti *color* a *background-color*. Vlastnost *color* určuje barvu textu a rámečku. Barva je pro účely testování zadávána RGB notací. Jinak barvu lze zadat hexadecimálně nebo slovně u základních barev. Vlastnost *background-color* určuje barvu pozadí. Barva se zde nastavuje stejně jako u vlastnosti *color*. Navíc zde lze nastavit vlastnost na transparent, což je průhledné pozadí.

Pozicování

V CSS existují dva naprosto odlišné způsoby pozicování. Absolutní pozice umístí objekt do stránky na udané souřadnice bez ohledu na okolní text. Oproti tomuto relativní pozice určuje pouze to, o kolik se má objekt posunout oproti své normální poloze. V CSS se toto nastavuje pomocí klíčového slova *position.* Absolutní polohu nastavíme pomocí slova *absolute* a relativní pomocí *relative.*

Klasifikační vlastnosti

První vlastností z této skupiny je vlastnost *display*, která určuje způsob zobrazení prvku. Tuto vlastnost lze nastavit na *block*, což zobrazí prvek jako blok a znamená to, že na konci řádku je zlom. Dále na *inline,* díky čemuž je prvek zobrazen jako kus řádku, tedy bez řádkového zlomu. A poslední *inline-block*, kde lze nastavit rozměry a prvek je zobrazen jako kus řádku beze zlomu. Další vlastností je *list-style-position*. Tato vlastnost určuje, zda bude odrážka v textu nebo vedle něj. Toto lze nastavit pomocí hodnot *outside* a *inside*. Hodnota *outside* nastaví, že je odrážka umístěna vlevo od bloku textu a řádky začínají pod sebou. *Inside* nastaví, že je odrážka umístěna hned vedle textu, takže druhý řádek je pod odrážkou.

Meze

V této skupině jsou kromě již zmíněných vlastností *padding* a *border* také vlastnosti *width* a *height*. Vlastnost *width* určuje šířku prvku. Pokud není zadána, tak je přirozená šířka blokových prvků 100%. Druhá vlastnost *height* udává výšku prvku. Výšku si prvky obvykle počítají podle obsahu, ale dá se i zadat touto vlastností.

4.2 Implementace rámce

Pro účely testování jsem vytvořil sadu HTML dokumentů a k nim přiložený CSS soubor, který popisuje grafické vyjádření těchto dokumentů. Obsahem těchto dokumentů jsou jednoduché stránky smyšleného festivalu. Jak CSS soubor, tak HTML dokumenty byly otestovány pomocí W3C validátoru. Jsou zde implementovány základní vlastnosti, ale také vlastnosti, které zobrazovací jádro CSSBox nepodporuje. Při možném rozšíření podpory i samotných testů JUnit lze tyto vlastnosti otestovat. Každý test využívá určitý HTML dokument (viz. Příloha 1.).

5 Implementace testů

V této kapitole se budeme zabývat implementací samotných testů, které ověřují podporu vlastností CSS zobrazovacím jádrem.

V první části této kapitoly si nejdříve uvedeme nástroje, které byly při tvorbě testů použity.

Další část kapitoly je věnována zajímavějším částem z implementace sady testů. Zde se zaměříme na práci se vstupním HTML dokumentem a jeho CSS vyjádřením. Dále jsou rozebrány jednotlivé testovací třídy, jaké provádějí testy, co k tomu používají. Závěr kapitoly je věnován třídě implementující testy použité v testovacích třídách.

5.1 Nástroje použité při implementaci testů

První technologií, použitou při implementaci testů, je jazyk Java a nástroj JUnit. Tento jazyk i nástroj pro jednotkové testování byl přidělen v zadání práce. Dále jsem zvolil vývojové prostředí NetBeans IDE pro své zkušenosti s tímto prostředím a pro jeho přednosti.

5.1.1 Java

Jazyk Java je objektově orientovaný jazyk vyvinutý v roce 1995 firmou Sun Microsystems. Může být používán na různých systémech díky své přenositelnosti. Java je nejen jazyk, ale i platforma, kterou realizuje virtuální stroj. Pro práci s jazykem Java je nutné mít nainstalován JDK (Java Development Kit). Syntaxe jazyka je zjednodušenou verzí syntaxe jazyků C a C++. Konstrukce, které dělaly programátorům problémy, byly odstraněny. Navíc byla přidána užitečná rozšíření.[3]

V jazyku Java je implementován renderovací nástroj CSSBox včetně testů. Pro testování byl využit nástroj JUnit (viz. Kapitola 3.2).

5.1.2 NetBeans IDE

Pro implementaci jsem zvolil vývojové prostředí NetBeans IDE. Je to open-source nástroj, pomocí nějž lze psát, překládat a ladit programy. Tento nástroj je primárně určen pro vývoj aplikací. Všechny funkce tohoto prostředí jsou poskytovány v modulech. Každý modul poskytuje dobře definované funkce, které jsou podporovány jazykem Java. Dále podporuje verzovací systém CVS a SVN. NetBeans obsahuje všechny moduly potřebné pro vývoj aplikací v jazyku Java. Podporu dalších funkcí lze rozšířit stažením a instalací dalších modulů. [6]

5.2 Implementace

Testy vytvářené pro ověření podpory CSS vlastností jsou založené na třídě *Assert*. Jednoduché testy jsou prováděny přímo pomocí metod této třídy. Pro složitější a opakující se testy byla vytvořena samostatná třída *AssertTests*. Každý test, který byl vytvořen, má přidělen HTML dokument s CSS vyjádřením.

Pro spuštění všech testů současně byla vytvořena třída *AllTests*. K tomuto účelu využívá anotace nástroje JUnit *@RunWith* a *@Suite*. První anotace je určená k rozšíření třídy o odkazované testy. Pomocí druhé anotace lze vytvořit sadu testů složenou z více tříd.

5.2.1 Práce s testovaným dokumentem

Každý test, který byl vytvořen, je založen na práci se třídou *Browser*. Úkolem této třídy je načíst HTML dokument včetně jeho CSS grafického vyjádření a vyhledat element/y určené k testování. Pro případ testování například seznamů používá tato třída třídu *ArrayList*. Všechny metody této třídy jsou důležité při testování zobrazovacího jádra, a proto jsou popsány níže.

- **createBrowser** – je metoda určená k načtení dokumentu a získání rozložení. Na vstupu očekává URL adresu, která je v testech zadávána jako absolutní cesta k testovanému HTML dokumentu. Nejdříve je navázáno spojení na danou URL adresu a vytvoří se abstrakce zdrojového dokumentu. Z tohoto dokumentu je dále získán DOM strom. Pomocí třídy *DOMAnalyzer* je DOM strom dále analyzován. Jsou k tomu použity metody:
  - *attributeToStyles()*, která převádí HTML prezentaci atributů v dokumentu do „inline“ stylů.
  - *addStyleSheet()* je metoda pro použití standardních nebo rozšířených CSS stylů.
  - *getStyleSheets()*, která vrací objekt CSSStyleSheet odkazující se z dokumentu.

  Na závěr je vytvořen objekt *BrowserCanvas*, který reprezentuje rozložení. Při vytváření objektu jsou také zadány rozměry zobrazované stránky. Po té metoda *createBrowser* vrací tento objekt.

- **rtrnBox** – tato metoda je používána pro vyhledávání elementů v DOM stromu. Na vstupu očekává kořenový element DOM stromu, který je většinou <html> a hledaný element. Metoda tedy prochází DOM strom a při nalezení hledaného element jej vrátí. Je používána v případech, kdy je element jednoznačně určen podle vlastností id,

protože ihned po nalezení prvku je ukončena, což by v případě vlastnosti class bylo nevhodné.

- **rtrnBoxList** – tato metoda má podobnou funkci jako předchozí. Jejím úkolem je nalézt elementy, které odpovídají hledanému. Hlavní rozdíl mezi těmito metodami je v tom, že hledáme elementy podle vlastnosti class a nebo elementy bez vlastností. Tedy elementy, které se mohou v DOM stromu vyskytovat na více místech. Na závěr je vrácen seznam těchto elementů a jejich vlastností.

- **getBoxArrayListItem** – je metoda určená pro práci se seznamem obsahujícím hledané elementy. Vrací položku seznamu tedy objekt typu *Box* podle zadané pozice v seznamu.

- **initBoxList** – je metoda pro inicializaci seznamu s hledanými elementy. Je používána v případě, že v jednom testu provádíme ověření správného zobrazení u více druhů elementů.

5.2.2 Struktura testů

Implementované testy byly rozděleny do kategorií uvedených v kapitole 4.1. Každá kategorie testů je implementována ve vlastní třídě. Navíc je zde třída provádějící ověření podpory základních a jednoduchých vlastností CSS. Všechny tyto třídy využívají třídu *Browser*, jak je uvedeno v předchozí kapitole (5.2.1). Použití této třídy je provedeno před zahájením všech testů pomocí JUnit anotace *@BeforeClass*, což znamená, že je daný úsek kódu proveden pouze jednou a to na samotném začátku. V této části kódu tedy získáme základní rozložení zahrnuté v DOM stromu a kořenový element tohoto stromu. S tímto dále pracují všechny testy. Před každou metodou reprezentující samotný test je anotace *@Test*, která každou metodu identifikuje jako testující.

Základní přehled tříd, ve kterých jsou implementovány testy zobrazovacího jádra:

Bounds (Meze) – tato třída zahrnuje testy *testBounds(),testContentBounds(), testClippedBounds()* a *testClippedContentBounds()*. Jsou zde testovány tři vlastnosti (meze, meze obsahu a absolutní meze obsahu), pro které jsou vytvořeny testovací metody ve třídě *AssertTests*. Všechny tyto metody využívají metodu *assertEquals()*. Správné zobrazení daných vlastností je testováno na dvou elementech.

BoxProperties (Vlastnosti obdélníku) – v této testovací třídě ověřujeme vlastnosti, jako jsou border, margin a padding jejichž význam je popsán v kapitole 4.1. Pro tyto vlastnosti byly

vytvořeny v testovací třídě *AssertTests* testovací metody, které využívají metody *assertEquals()*. Jsou tedy pouze porovnávány hodnoty vrácené zobrazovacím jádrem s hodnotami, které očekáváme podle hodnot uvedených v CSS vyjádření HTML dokumentu. Dalšími CSS vlastnostmi, které jsou zde ověřovány, jsou *width* a *height* určující šířku a výšku. Pro ověření těchto vlastností je využito testovací metody *assertWidthHeight()* třídy *AssertTests*. Tyto testy jsou prováděny na elementech i synovských elementech.

ClassificationProperties (Klasifikační vlastnosti) – je třída, která ověřuje CSS vlastnost *display* určující způsob zobrazení prvků, což je testováno na seznamu. Dále ověřuje správné zobrazení odsazení seznamu. CSS vlastnost *display* je testována pomocí tří metod. Každá z nich ověřuje jednu ze tří možností zobrazení. K ověření korektnosti je využita metoda *assertEquals()*. Správnost odsazení seznamu je ověřována pomocí vytvořené metody *assertListPosition()* ve třídě *AssertTest*, která je popsána níže (kapitola 5.2.3). Na závěr v této třídě ověřujeme CSS vlastnost *white-space* pro níž byla opět vytvořena testovací metoda *assertWhiteSpace()*.

ClrBackgrndProp (Vlastnosti barev a pozadí) – tato testovací třída zahrnuje testy *testColor()* a *testBackgroundColor()*. Třída ověřuje, zda barvy pozadí, písma aj. odpovídají barvám zadaným v CSS vyjádření HTML dokumentů. V CSS vyjádření jsou barvy zadávány různě, ale v těchto testech jsou barvy ověřovány pouze pomocí RGB vyjádření. Pro otestování tohoto vyjádření je použita testovací metoda *assertRGBColor()* třídy *AssertTests*, která má na vstupu objekt typu *Color* a předpokládané číselné hodnoty R, G, B.

FontProperties (Vlastnosti písma) – je testovací třída, která zahrnuje ověřování podpory téměř všech vlastností písma. Tato třída zahrnuje celkem čtyři testy a to testy pro ověření velikosti, varianty, hmotnosti a stylu písma (*testFontSize(), testFontVariant(), testFontWeight(), testFontStyle()*). Pomocí třídy JUnit, nelze důkladně ověřit vlastnost „font-family“, a proto zde byl testován pouze správný název písma. Při testování těchto vlastností jsem se nejvíce zaměřil na testování velikosti písma, která je testována na čtyřech elementech. Bylo to dáno i výsledky testů, které jsou rozebírány v následující kapitole 6. Ostatní vlastnosti byly testovány pouze na jednom elementu.

FormatingModel (Formátování modelu) – je třída, která ověřuje formátování v různých variantách a šířku vodorovné čáry. Pro ověření CSS vlastností margin, padding, border (vysvětleny v kapitole 4.1), byly vytvořeny testovací metody ve třídě *AssertTests*. Nejdříve tato třída testuje horizontální a vertikální formátování elementů. Pro důležitost tohoto formátování je každý z těchto testů proveden na několika elementech včetně seznamů. Proto

byla vytvořena i zvláštní testovací metoda *testFormatingMenu()* ověřující správné zobrazení horizontálního menu. Všechny tyto vlastnosti jsou testovány v různých kombinacích i jednotkách, pro co nejspolehlivější ověření správného zobrazení.

V další části je ověřována CSS vlastnost *float*, která je ověřena opět pomocí metody *assertEquals()*. Zde je nejen testováno nastavení vlastnosti, ale i absolutní poloha elementu.

Poslední vlastností, která je ověřována v této třídě, je šířka vodorovné čáry. Zde je testována pouze hodnota vrácená zobrazovacím jádrem, s předpokládanou hodnotou.

PositionElements (Pozicování elementů) – v této třídě jsou testovány plovoucí elementy a pozice elementů (absolutní nebo relativní). Pro testování plovoucích elementů je zde testovací metoda *testFloatPosition()*. V případě, který je testován touto metodou, jsou umístěny elementy do úzkého prostoru. Metoda ověřuje, zda se element, který se nevejde na stejný řádek jako předchozí element, zobrazí na novém řádku. Pro toto ověření byla vytvořena metoda *assertAbsoluteContentBounds()* ve třídě *AssertTests*, která je popsána níže včetně dané metody. Pro ověření pozicování elementů jsou zde metody *testPositionAbsolute()* ověřující absolutní pozici elementu vůči referenčnímu elementu a metoda *testPositionRelative()*, která opět ověřuje pozici elementu vůči referenčnímu elementu.

STest (Test základních vlastností) – je třída, která ověřuje správné zobrazení různorodých CSS vlastností zobrazovacím jádrem. Poskytuje jednoduché a základní testování vlastností. Skládá se z pěti testovacích metod, které postupně testují daný HTML dokument. V první části jsou testovány vlastnosti písma, odsazování a rámečky. V následujícím testu je ověřována správná absolutní pozice nadpisu, která je testována metodou *assertAbsolutePosition()* testovací třídy *AssertTests*. U tohoto nadpisu také testujeme styl písma, kterým je zobrazen.

Následující test ověřuje správnou velikost elementu a nastavení CSS vlastnosti *float*, u které je dále testována absolutní pozice.

Závěrečný test této třídy testuje zbylé vlastnosti jako velikost a dekorace písma, odsazování a umístění elementů.

5.2.3 Třída pro testování vlastností

Pro testování CSS vlastností, které jsou náročnější na testování anebo se několikrát opakují, jsem vytvořil speciální třídu *AssertTests*. Tato třída implementuje sadu vlastních testů využívajících třídu *Assert*. Jelikož nástroj JUnit neumožňuje vytvoření vlastních testů např. typu *assertEquals(...)*, bylo nejvhodnějším řešením pro vytvoření vlastních testů vytvoření statických metod, které využívají metody ze třídy *Assert*.

Při porovnávání hodnot typu *double* bylo nutné u testů určit odchylku. Tato odchylka je v implementaci určena proměnnou *deviation* a nastavena na hodnotu 0.0. Tato proměnná je statická.

Téměř všechny testovací metody mají jako parametry zadány objekty vyjadřující CSS vlastnost, kterou chceme testovat a očekávané hodnoty. Pouze u ověřování pozice elementu vůči jinému elementu, předáváme dva objekty reprezentující dané CSS vlastnosti. Při testování jsou využívány zejména metody *assertEquals()* a *assertTrue()*, které jsou rozebrány v kapitole 3.3.2.

Nejzajímavější částí z implementace této třídy jsou statické metody *assertAbsoluteContentBounds()* a *assertListPosition()*. První z těchto metod je metoda přetížená a ověřuje pozici plovoucích elementů např. v situaci, kdy dojde k přesunutí elementu na nový řádek z důvodu nedostatku místa. Tato metoda tedy porovnává absolutní pozice dvou elementů. Z důvodu nedostatku místa dojde k tomu, že druhý element je zobrazen na novém řádku. Absolutní hodnoty na x-ové ose jsou tedy shodné a na y-ové musí být odlišné o výšku elementu a odsazení. V případě, že tato podmínka není splněna, je test neúspěšný. Obdobným způsobem pracuje metoda *assertListPosition()*. Metoda ověřuje CSS vlastnost „*list-style-postiton: inside/outside*“. Odsazování v seznamu je zde opět porovnáváno pomocí absolutní pozice, ale tentokrát pouze na x-ové ose.

6 Vyhodnocení testů

V této kapitole se budeme zabývat vyhodnocením testů prováděných pomocí testovacího rámce a nástroje JUnit. Byly testovány základní a dohodnuté vlastnosti CSS. V první části kapitoly si rozebereme testy vlastností, které proběhly úspěšně. V další části uvedeme testy, které proběhly neúspěšně. U každého neúspěšného testu si zmíníme, v čem selhal. Dále se zaměříme na CSS vlastnosti, u kterých se při testování podařilo odhalit, že nejsou podporovány zobrazovacím jádrem CSSBox. Na závěr této kapitoly si zmíníme možná rozšíření testů, vzhledem k vývoji CSS a CSSBox.

Správnost testů, u kterých bylo nejednoznačné ověřit správnost zobrazení, byla ověřena pomocí prohlížeče Firefox a jeho rozšíření Firebug, které slouží k ladění a editaci webových stránek

6.1 Úspěšné testy

Úspěšnost správného zobrazení dohodnutých vlastností CSS zobrazovacím jádrem prohlížeče po provedení těchto testů pomocí násroje JUnit je **64,71%**.

Z osmi testovacích tříd, které byly vytvořeny, byly dvě plně úspěšně otestovány. První z těchto tříd je třída *Bounds* testující meze elementů, což znamená šířku, výšku elementů a pozici těchto elementů, tedy x-ové a y-ové souřadnice. Další třídou z této skupiny je třída *ClrBackgrndProp*, která ověřuje správné zobrazení barev. Při testování pomocí RGB vyjádření barev bylo ověřeno správné zobrazení barev, které bylo doplněno i o vizuální kontrolu. Z dalších testovacích tříd jsou v následujícím seznamu zmíněny ty testy vlastností CSS, které proběhly úspěšně, a které CSS vlastnosti testovaly.

- *testBodyHead* - border, padding
- *testImageOnTitlePage* - width, height, float
- *testFontWeight* - font-weight
- *testFontStyle* - font-style
- *testHeightWidthElem* - width, height
- *testHeightWidthInlineEleme* - width, height
- *testFloatingElements* - float
- *testHorizontalFormating* - margin, padding
- *testVerticalFormating* - margin, padding
- *testDisplayBlock* - display
- *testDisplayInline* - display
- *testDisplayInlineBlock* - display

- *testFloatPosition* - správné umístění elementu při nedostatku místa
- *testPositionRelative* - position
- *testListStylePositionInside* - list-style position
- *testWhiteSpace* - white-space

6.2 Neúspěšné testy

Pomocí testovacího nástroje JUnit a testů implementovaných pomocí tohoto nástroje, byly také nalezeny nedostatky zobrazovacího jádra CSSBox. Jelikož byly testy zaměřeny na odsazování, pozicování a jiné základní vlastnosti, byly chyby nalezeny převážně zde. Samozřejmě byly chyby objeveny i např. u práce s písmem.

Seznam testů a testovaných vlastností CSS, které proběhly neúspěšně. Níže je uveden seznam těchto testů. U těchto testů jsou vypsány pouze ty vlastnosti CSS, u kterých nastal problém.

- *testLists* - font-size
- *testTitle* - font-family, font-style, absolutní pozice
- *testContent* - font-size
- *testFontSize* - font-size
- *testFontVariant* - font-variant
- *testMargPadBorInlineElem* - margin
- *testMargPadBorElem* - margin
- *testInlineElements* - margin
- *testFormatingMenu* - margin
- *testHeightOfLines* - height
- *testPositionAbsolute* - position
- *testListStylePositionOutside* - list-style-position

Podle seznamu testů a vlastností, ve kterých tyto testy selhaly, lze zjistit, že se chyby týkají několika základních vlastností. Velikost písma *font-size* selhala ve všech testovacích případech. U testování vlastností písma nastal problém také u vlastností *font-style, font-family a font*-variant. Další problémovou vlastností je CSS vlastnost *margin.* Ta v určitých případech fungovala správně, ale problém nastával především u hodnoty *margin-top*.

U vlastnosti pro odsazování v seznamu *list-style-position* nastal problém při nastavení vlastnosti na hodnotu inside. Položky seznamu byly zobrazeny se špatným odsazením.

Hodnoty *width* a *height* byly získávány dvěma způsoby. V případě, že byla hodnota získána přímo z objektu typu *Box* pomocí metody *getWidth()*/*getHeight()*, nabývala špatných hodnot vždy. Pomocí metody *getContentBounds()*, byly získané hodnoty správné.

Poslední problém byl testovací třídou nalezen při práci s absolutní pozicí. Element byl při nastavení vlastnosti *position: absolute* bez určení pozice umístěn neprávně.

6.3 Nepodporované vlastnosti CSS

Při prostudování architektury, a také při samotném testování zobrazovacího jádra CSSBox, byly nalezeny nepodporované vlastnosti CSS. Některé vytvořené testy, které tyto vlastnosti odhalily, byly označeny JUnit anotací *@Ignore*. Znamená to, že je testovací metoda přeskočena.

Nepodporované vlastnosti CSS:

- *border-radius*
- *list-style-type* - tato vlastnost byla odhalena vizuální kontrolou, protože pomocí nástroje JUnit nelze tuto vlastnost ověřit graficky

6.4 Možná rozšíření

Možná rozšíření této práce mohou jít více směry. Tím hlavním je rozšíření testů o ověření vlastností CSS, které nebyly v této práci otestovány. Jako příklad bych uvedl, že jsem se při testování zaměřil na testování samotného písma, ale ne na vlastnosti textu. Lze tedy testy rozšířit o ověření správnosti zobrazení textu, tedy vlastností CSS jako je *text-align*, *text-indent*. U vlastnosti rámečku lze otestovat vlastnost *border-radius*.

S vývojem CSS, který neustále probíhá lze počítat i s rozšířením podpory vlastností v budoucnu zaváděného standardu CSS3 i u tohoto renderovacího nástroje. Jedním z hlavních směrů rozšíření jsou tedy testy ověřující podporu vlastností CSS3.

7 Závěr

Cíle vytyčené v úvodu práce se mi podařilo naplnit. Po dohodě s vedoucím práce byly vybrány konkrétní vlastnosti CSS a následně byl vytvořen testovací rámec zahrnující dané vlastnosti. Tímto testovacím rámcem byl jednoduchý web smyšleného festivalu. Pomocí vytvořené sady testů byla ověřena podpora těchto vlastností a správnost zobrazení zobrazovacím jádrem prohlížeče, které vzniklo na Fakultě Informačních Technologií. Po vyhodnocení těchto testů bylo zjištěno, že ve většině případů selhávali stejné vlastnosti CSS (podrobnosti viz. Kapitola 6), a proto byly dále dané vlastnosti důkladněji otestovány. Na závěr bylo navrženo rozšíření testů, o které může být sada v budoucnu rozšířena. Jak je již v předchozí kapitole zmíněno, tak se rozšíření těchto testů může ubírat více směry.

Při práci na tomto projektu jsem se seznámil s nástrojem JUnit, který se využívá pro jednotkové testování v jazyku Java. Nabyté znalosti jsou pro mne přínosem a v budoucnu je jistě využiji.

Literatura

[1] SCHAFER, Steven M. *HTML, XHTML a CSS*. 4. vydání. Praha: Grada Publishing, a.s., 2009, 637s. ISBN 978-80-247-2850-6.

[2] RAINSBERGER, J. B. SCOTT STRILING. *JUnit Recipes: Practical Methods for Programmer Testing*. Greenwich: Manning Publications Co., 2005, 721s. ISBN 1-932394-23-0.

[3] SPELL, Brett. *Java Programujeme profesionálně*. Praha: Computer Press, 2002, 1040s. ISBN 80-7226-667-5.

[4] HRUŠKA, Tomáš; BURGET Radek. *Internetové aplikace (WAP) II.: část SGML, HTML, CSS, DOM -Studijní opora*[online]. Brno, 2007[cit. 2013-04-15]. Dostupné z: https://www.fit.vutbr.cz/study/courses/WAP/private/opory/

[5] BURGET, Radek. *CSSBox* [online]. 2007, 2013-03-27 [cit. 2013-04-25]. Dostupné z: http://cssbox.sourceforge.net

[6] NetBeans. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2013-04-27]. Dostupné z: http://en.wikipedia.org/wiki/NetBeans

[7] WORLD WIDE WEB. *CSS1 Test Suite* [online]. 1996-2008 [cit. 2013-04-25]. Dostupné z: http://www.w3.org/Style/CSS/Test/CSS1/current/index.html

[8] *JUnit: A programmer-oriented testing framework for Java* [online]. 2013 [cit. 2013-04-25]. Dostupné z: http://junit.org

[9] JANOVSKÝ, Dušan. *Jak psát web* [online]. 1999, 2012-12-06 [cit. 2013-04-27]. Dostupné z: http://www.jakpsatweb.cz/

[10] CLARK, Andy a Marc GUILLEMOT. *CyberNeko HTML Parser* [online]. 2002 - 2009 [cit. 2013-04-27]. Dostupné z: http://nekohtml.sourceforge.net/

[11] HUNT, Andrew a David THOMAS. *Pragmatic Unit Testing*. 2. vydání. Dallas: The Pragmatic Bookshelf, 2007. ISBN 978-0-9776166-7-4.

[12] Unit testing. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2013-04-27]. Dostupné z: http://en.wikipedia.org/wiki/Unit_testing

Seznam příloh

Příloha 1.

Přidělené HTML dokumenty jednotlivým testům	
Název testu	**Název HTML dokumentu**
Bounds.java	kontakt.html
BoxProperties.java	program-sobota.html
ClassificationProperties.java	vstupenky.html
ClrBackgrndProp.java	index.html
FontProperties.java	program-patek.html
FormatingModel.java	index.html
PositionElements.java	program-sobota.html
STest.java	index.html

Příloha 2.

Obsah přiloženého CD

./lib/
Tento adresář obsahuje knihovny potřebné pro běh samotné aplikace i testů.

./src/org/
Adresář org obsahuje zobrazovací jádro CSSBox.

./src/test/test_frame/
Tento adresář obsahuje testovací rámec potřebný pro chod testů. Je to sada HTML dokumentů a jednoho CSS souboru obsahující grafické vyjádření.

./src/test/
Adresář obsahující sadu testů, ověřujících podporu zobrazovacího jádra.