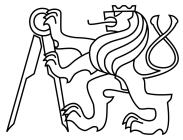
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Knihovna generátoru zadání písemek
Student:	Tomáš Drbota
Vedoucí:	Ing. Jiří Hunka
Studijní program:	Informatika
Studijní obor:	Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2016/17

Pokyny pro vypracování

Vytvořte softwarovou knihovnu pro generování zadání písemek a ukázkovou aplikaci, na které bude použití této knihovny demonstrováno. Knihovna bude pracovat s databází otázek uloženou v souboru v ručně editovatelném formátu.
Otázky obsahují text zadání se základním formátováním a podporou obrázků, ohodnocení složitosti, klíčová slova a mohou obsahovat i spustitelný kód pro program Wolfram Mathematica. Součástí otázky je také specifikace proměnných, do kterých se při generování zadání dosadí náhodné hodnoty na základě podmínek určených v otázce.
Knihovna bude umožňovat na základě uživatelem definovaných pravidel vybrat určený počet otázek a sestavit z nich zadání testu. Zadání je možno generovat podle výběru uživatele pro tisk a ve formátu pro program Wolfram Mathematica.
Knihovna s ukázkovou aplikací musí být spustitelná na operačních systémech Windows a Linux. Součástí práce je výběr a zhodnocení vhodného jazyka, případně i dalších knihoven použitých pro implementaci.

Seznam odborné literatury

Dodá vedoucí práce.

L.S.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 2. ledna 2016

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra Softwarového inženýrství

Bakalářská práce

Knihovna generátoru zadání písemek

Tomáš Drbota

Vedoucí práce: Ing. Jiří Hunka

15. května 2016

Poděkování

Děkuji především svému vedoucímu práce, Ing. Jiřímu Hunkovi. Dále bych chtěl poděkovat své rodině za podporu po celou dobu mého vzdělávání.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 15. května 2016 .....

České vysoké učení technické v Praze
Fakulta informačních technologií

Odkaz na tuto práci

Drbota, Tomáš. *Knihovna generátoru zadání písemek*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2016.

Abstrakt

Cílem této prace je vytvořit knihovnu a ukázkovou aplikaci, které budou sloužit k generování zadání písemných prací. Aplikaci se předá sada otázek a pravidel pro složení testu a knihovna jej potom vytvoří. Výsledný test je možné exportovat do formátu HTML či formátu podporovaného softwarem Wolfram Mathematica. Knihovna je napsaná v Javě a pro ukládání dat využivá formát XML.

Klíčová slova knihovna, generátor zadání, písemná práce, wolfram mathematica, java, xml

Abstract

The goal of this thesis is to create a library and an example application to generate exams. Given a set of questions and a set of rules, the library will generate the corresponding exam. The result may then be exported to either HTML or the format used by Wolfram Mathematica. The library is written using Java and uses XML to persist data.

Keywords library, exam generator, exam, wolfram mathematica, java, xml

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Školství je jedno z nejdůležitějších odvětví každého státu. Vzdělávat děti a mladé lidi je nutnost pro jakýkoliv pokrok. Neoddělitelnou součástí vzdělávání je i průběžné testování žáků či studentů, kde se zjišťuje, do jaké míry danou látku pochopili. Dříve byly tradičně testy papírové, dnes se již začínají rozmáhat digitální testy, z důvodu ušetření času i peněz.

Nicméně ve škole při psaní písemných prací může nastat problém s opisováním. Aby se tomu předešlo, učitelé často vymýšlí více variant zadání. Ovšem u některých předmětů, například u matematických je to obzvlášť zjevné, stačí ke zmatení souseda jen přeformulovat zadání či obměnit čísla.

Pro tuto činnost je vhodné mít software, kterému se předá předem připravená sada otázek a ten potom zvládne podle nějakých omezujících pravidel rychle vygenerovat velké množství variant toho samého zadání, s objektivně stejnou obtížností, ale přecijen jiným složením otázek. Toto zmírní či uplně zamezí jakékoliv opisování.

Toto téma jsem si zvolil především proto, že mi tato problematika přijde zajímavá a protože věřím, že moje řešení povede k vyšší kvalitě testů obecně, ať už jen kvůli tomu, že kantor nebude muset trávit spoustu času vytvářením několika variant.

V práci se zabývám vytvořením aplikace, ve které si uživatel může připravit úlohy s různými obtížnostmi, vlastní zadání se skládají z různých elementů, např. náhodných hodnot z určitého rozmezí apod. Z těchto úloh si uživatel potom může jednoduše vygenerovat zadání v jednom ze dvou (nebo obou) podporovaných formátů, tj. HTML a Wolfram Mathematica Notebook.

Nápad automatizovaného generování testů určitě není revoluční, nicméně dle mého výzkumu neexistuje žádný program nebo systém, který splňuje po-

žadavky konkrétně kladené na tuto práci.

Kapitola 1

Cíl práce

Cílem této práce obecně je vytvořit knihovnu a zároveň s knihovnou ukázkovou aplikaci, jež ji využívá. Knihovně se pomocí aplikace předá sada či více sad otázek spolu s příslušnými pravidly pro sestavení testu. Knihovna potom tento vstup vyhodnotí a vrátí výslednou písemnou práci.

Cílem rešeršní části je především zmapování požadavků uživatele a zmapování existujících (kusů) řešení. Dále je potřeba vymyslet, jakým způsobem se budou tato zadání a pravidla uchovávat a zpracovávat. Kromě vstupů je také nutností zanalyzovat výstupní formáty, kterými v tomto případě jsou formáty HTML a Wolfram Mathematica Notebook. Pro rozbor druhého formátu bude vhodné manuálně rozebrat existující notebooky softwaru Wolfram Mathematica, který je využívá.

Cílem praktické části je tedy logicky navrhnout a naimplementovat funkční knihovnu pro generování písemných prací a ukázkovou aplikaci, která na ní bude napojená. Jedním z největším problémů bude efektivně vymyslet, jak podle pravidel vybrat vhodnou skladbu otázek.

Kapitola 2

Struktura práce

Vzhledem k tomu, že práce se fakticky skládá ze dvou částí - knihovny a aplikace, jež ji využívá, práci jsem se rozhodl rozdělit takto.

- Analýza uživatelských a technických požadavků
- Analýza formátu softwaru Wolfram Mathematica
- Návrh knihovny pro generování písemných prací
- Návrh ukázkové aplikace
- Implementace knihovny a aplikace
- Testování správné funkcionality knihovny

Kapitola 3

Analýza požadavků

Tato kapitola se zabývá především rešeršní částí bakalářské práce. Zde se bude pojednávat o všech požadavcích této práce a možných řešeních.

3.1 Aktuální stav problému

V současné době se věc má tak, že pro většinu písemných prací se volí manuální přístup, kdy vyučující před každým testem dle svého uvážení vymýšlí nové otázky a rozdělí je mezi různé varianty testu. I když často mají připravené otázky či recyklují otázky z minulých let, písemné práce musí stejně nějakým způsobem obměňovat.

V některých disciplínách je tento přístup v pořádku, ale ve většině případů to může vést k nevyváženým variantám a studenti si potom stěžují, že někdo jiný měl test „lehčí“ a není žádné objektivní měřítko, dle kterého by se obtížnost testu dala srovnat.

V případě konkrétního problému, který řeší tato práce, se písemné práce pro předměty BI-CAO a BI-SAP vytvářejí ručně či se recyklují starší.

3.2 Současné řešení problematiky

V současné době je k dispozici velké množství různých aplikací pro tvorbu písemných prací či testování studentů obecně. Neexistuje ale žádný nastroj, který splňuje naše konkrétní požadavky, konkrétně vkládání spustitelného kódu do otázky a následný export do programu Wolframu Mathematica. Následuje výběr některých zajímavějších nástrojů, které bohužel nevyhovují požadavkům.

3.2.1 Moodle

Moodle je jeden z nejpopulárnějších frameworků pro správu vzdělávání studentů a jejich testování. Nabízí širokou škálu možností zadávání otázek a jejich automatické vyhodnocování [1]. Učitel si potom ze systému dokáže vytáhnout vše, co potřebuje.

Moodle je velice populární na velkém množství univerzit, některé ho používají ke kompletní správě studentů, ne jen k testování. Níže uvedené jsou důvody pro to, v čem je Moodle dobrý:

- Umožňuje spravovat otázky a jejich hodnocení
- Lze si pohodlně navolit skladbu potenciálního testu
- Automatické vyhodnocení testu a ohodnocení studenta
- Přímé napojení na výsledky studentů, lze sledovat statistiky
- Ve vývoji již dlouhou dobu
- Možnost jednodušše dopisovat vlastní pluginy

I když by se mohlo zdát, že po několika uprávách by se Moodle dal přizpůsobit našim potřebám, bohužel nesplňuje požadavky této aplikace hned z několika důvodů:

- Je nutné jej mít běžící na nějakém serveru
- Není určen k off-line využití
- Používá se spíše k vlastnímu testování než generování testů
- Neexistuje nativní podpora pro tisk či export zadání
- Nemožnost zadání komplexnějších zadání s např. náhodnými elementy

Hlavní problém je v podstatě fakt, že Moodle je celý obrovský LMS[1], a ač se do něj dají dopisovat pluginy, bude výhodnější napsat vlastní aplikaci, ať už jen kvůli tomu, že nebude žádné omezení co se týče platformy.

[1] Learning Management System

3.2.2 Blackboard

Blackboard LMS také slouží jako systém pro vzdělávání studentů. Mimo jiné také nabízí možnosti zadávání testů [2], ale jejich komplexita je velice podobná Moodlu, a pro naše využití je nástroj moc rozsáhlý a drahý. Také nepodporuje tisk a ani export do programu Wolfram Mathematica.

V důsledku se klady i zápory tohoto systému dají srovnat s Moodlem a není nutno je více rozepisovat.

3.2.3 DoTest

Tento nástroj se od předchozích dvou liší tím, že se nejedná o celý framework se vším všudy, ale jde pouze o nástroj ke generování testů. DoTest [3] umožňuje spoustu věcí, které se po tomto projektu požadují.

Umožňuje uživatelům sestavit si sadu testů a připravit k nim odpovědi. Lze si vybrat ze šesti typů otázek. Existuje také možnost si otázky a možné odpovědi kategorizovat do různých skupin a po vytvoření nějaké varianty ji vytisknout. Tato aplikace je asi nejblíže tomu, co je potřeba, ale stejně trpí několika neduhy.

Mezi klady tedy patří:

- Možnost snadného přidání otázek
- Rozdělení testů do kategorií
- Použitelný výběr typů otázek
- Automatické sestavení a vyhodnocení testů
- Existuje již dlouho, rozsáhlá podpora v ČR
- Podporuje tisk výsledného testu

Nevyhovuje ale z několika důvodů:

- Typy otázek se nedají jednodušše rozšířit
- Slouží spíše pro otázky, kde se odpověď vybírá ze seznamu
- K otázkám se nedají přiřadit elementy s náhodnou hodnotou
- Nepodporuje export do formátu NB, ani nejde snadno dodělat

3.3 Sběr požadavků

Před tím, než se vytvoří funkční a nefunkční požadavky, je důležité nějakým způsobem zkontaktovat cílovou skupinu uživatelů projektu a zjistit, co od programu vůbec čekají. Z výstupu těchto výzkumů se potom vytvoří finální požadavky.

Pro tento projekt byly stěžejní především rozhovory, a to s učiteli, kteří tuto knihovnu budou nějakým způsobem využívat. Následuje seznam nejdůležitějších otázek, které mi později umožnily sestavit požadavky:

- Jaký obsah se může objevit v otázkách?
- Jak se budou modifikovat či přidávat otázky?
- Podle jakých pravidel se budou testy skládat?
- Jaké možnosti výstupu má program mít?
- Na čem musí program běžet?

3.4 Funkční požadavky

V této sekci se vytyčí všechny požadavky na projekt co se týče funkcionality. Každý z těchto požadavků musí být správně naimplementován a otestován.

3.4.1 Definování vlastních otázek

Uživatel má možnost nadefinovat si otázky skládající se z různých elementů.

Každé otázce se dá nastavit množství parametrů, konkrétně jaké elementy obsahuje, obtížnost, tagy, apod.

3.4.2 Formátování otázek a vkládání elementů

Každá otázka se skládá z částí, které se nazývají „elementy“. Element může být například formátovaný text, seznam či náhodná hodnota z nějakého rozmezí.

Systém musí být dostatečně flexibilní na to, aby se tyto elementy daly volně kombinovat a vytvářely tak složitější zadání s náhodnými elementy.

3.4.3 Uspořádání otázek do sad

Nakonfigurované otázky je možné uspořádat do sad otázek. Z těchto sad se potom vybírají otázky, které budou ve výsledném testu.

Jednou z dalších vlastností programu bude i možnost slučovat různé sady dohromady pro lepší přehlednost a organizaci.

3.4.4 Definování pravidel pro skladbu testu

Aby se mohly do výsledného testu vybrat ty správné otázky dle potřeby, je nutné mít možnost nastavit určitá pravidla.

Konkrétně se v pravidlech nadefinuje konečný počet otázek a potom se každé potenciální otázce přiřadí pravidla, která vybraná otázka musí splňovat. Pravidla se potom dají skládat dohromady pomocí logických operátorů a vytvářet tím komplexnější pravidla.

3.4.5 Generování testů

Programu se jako vstup předá sada otázek, ze které má vybírat a seznam pravidel, podle kterých vybere vhodné otázky. Program potom vygeneruje vyhovující zadání (pokud takové existuje) a výstup uloží do souboru.

3.4.6 Výstup do formátu vhodného pro tisk

Program musí umět výsledný test uložit do formátu vhodného pro tisk.

To znamená, že musí umět i chytře napozicovat otázky tak, aby vycházely dobře na stránku, a to včetně možného volného místa na odpověď.

3.4.7 Výstup do formátu NB pro Wolfram Mathematica

Program musí umět výsledný test uložit do formátu NB (notebook), který je podporován programem Wolfram Mathematica.

Účelem je, aby student přímo pod zadání mohl psát řešení právě pomocí výše uvedeného programu. Tento formát má specifické složení, o kterém se pojednává v pozdější kapitole.

3.5 Nefunkční požadavky

Tyto požadavky jsou kladeny na faktory, které přímo nesouvisí s funkcionalitou, ale stejně musí být naplněny.

3.5.1 Program musí být spustitelný na běžných distribucích OS Linux a OS Windows

Je nutno zvolit jazyk či platformu, která je běžně podporována na většině známých operačních systémů.

3.5.2 Rychlé generování testů

Generování testu by nemělo zabrat více než několik minut. Je potřeba vybrat vhodný algoritmus pro vybírání otázek podle pravidel.

3.5.3 Systém se musí dát snadno rozšířit

Program musí být navržen tak, aby se do něj dalo snadno zasahovat.

Konkrétně pak je potřeba mít možnost jednodušše přidávat či upravovat elementy, popřípadě možnost měnit formát vstupu a výstupu.

3.5.4 Otázky musí být v ručně editovatelném formátu

Formát použitý pro uložení otázek a sad otázek musi být pohodlně ručně editovatelný, tzn. žádné binární formáty.

3.5.5 Pravidla musí být v ručně editovatelném formátu

Zde platí to samé jako pro předchozí požadavek, ale pro pravidla.

3.5.6 Program musí fungovat bez sítě

Tento nástroj by měl být spustitelný na libovolném počítači i bez připojení k internetu.

3.6 Srovnání ostatních řešení s požadavky

V následující tabulce lze vidět srovnání tohoto programu vůči ostatním, podobným.

Jako hlavní kritéria jsem využil funkční a nefunkční požadavky uvedené výše, kde je také napsáno podrobnější vysvětlení, co znamenají.

Tabulka 3.1: Srovnání funkcionality podobných programů

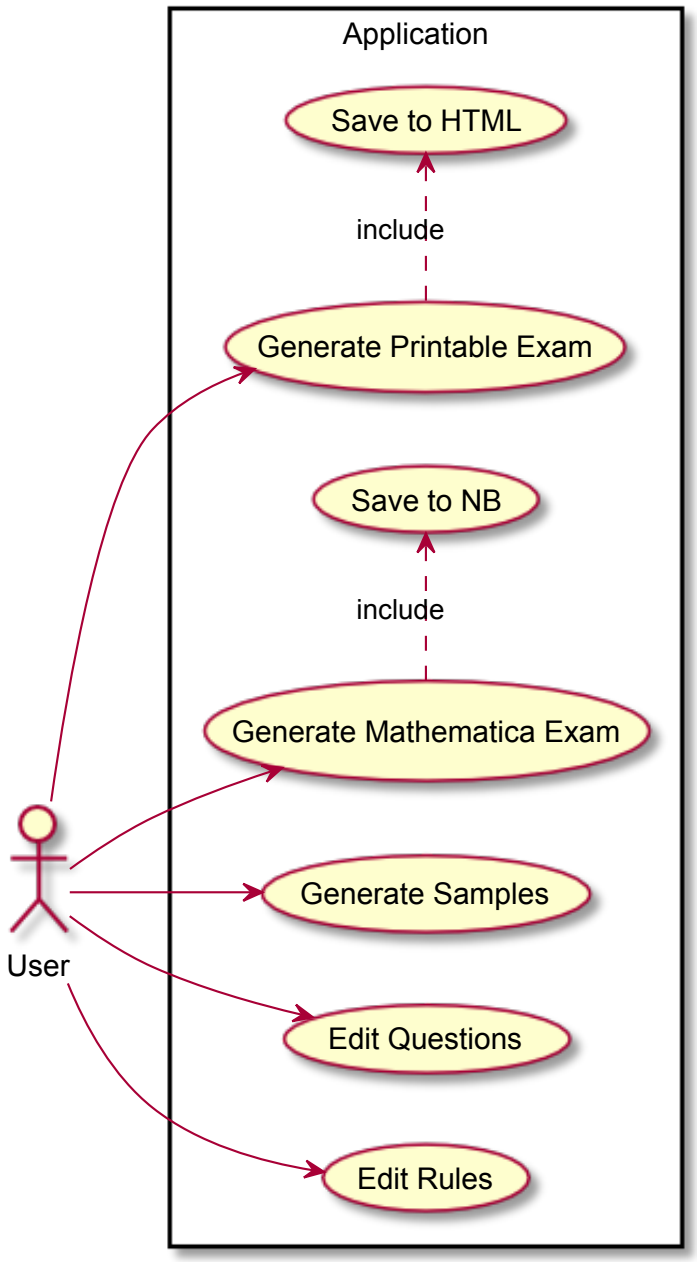
	Tato práce	DoTest	Moodle	Blackboard
Definování vlastních otázek	Ano	Ano	Ano	Ano
Formátování otázek a vkládání elementů	Ano	Ne	Ne	Ne
Uspořádání otázek do sad	Ano	Ano	Ano	Ano
Definování pravidel pro skladbu testu	Ano	Ano	Ano	Ano
Generování testu	Ano	Ano	Ano	Ano
Výstup do tisknutelného formátu	Ano	Ano	Ne	Ne
Výstup do formátu NB	Ano	Ne	Ne	Ne
Snadná rozšiřitelnost	Ano	Ne	Ano	Ne
Funguje bez sítě	Ano	Ano	Ne	Ne
Licencován zdarma	Ano	Ne	Ano	Ne

3.7 Případy užití

V této sekci je výčet případů užití, které uživatel může provést.

3.7.1 Diagram případů užití

Obrázek 3.1: Případy užití



3.7.2 Scénáře

Pro používání této aplikace budou nejdůležitější dva konkrétní scénáře.

3.7.2.1 Příprava otázek a pravidel

Tento scénář popisuje, jak připravit sadu otázek a pravidel pro pozdější generování testu.

1. Tento případ užití začíná ve chvíli, kdy máme připravenou spustitelnou formu této aplikace.
2. Uživatel spustí aplikaci a předá ji parametr pro tvorbu ukázkových souborů.
3. Program interně vytvoří ukázkové soubory pomocí zabudované knihovny a uloží je na disk.
4. Uživatel může tyto soubory otevřít v libovolném textovém editoru a upravit si je podle potřeby. V praxi půjde hlavně o přidávání dalších otázek a pravidel, která na ně budou napojena.

3.7.2.2 Generování testu

V tomto scénáři se předpokládá, že uživatel již má připravenou sadu otázek (či více sad) a zároveň k nim má připravená pravidla, podle kterých se test má složit.

1. Případ užití začíná ve chvíli, kdy máme připravenou spustitelnou formu aplikace a nějaká data.
2. Uživatel aplikaci předá soubor s pravidly a jednu či více sad otázek.
3. Program si interně spojí sady otázek, pokud jich je více, a tyto otázky a pravidla předá knihovně. Knihovna potom vygeneruje test podle požadavků a vrátí aplikaci výsledek. Aplikace potom uloží výsledný test do souboru, formát je určen dle zadání pravidel.
4. Uživatel tento výstup může otevřít buď v prohlížeči (tisknutelná forma HTML) či v programu Wolfram Mathematica (formát NB).

3.8 Popis formátu NB

Jedná se o formát, který využívá software Wolfram Mathematica. Jeho kompletní popis a reference lze najít na oficiálních stránkách Wolframu [4].

V této práci ale bude stačit seznámit se s některými základními prvky, které tato knihovna, potažmo aplikace, využívá.

3.8.1 Ukázka kódu

Toto je příklad výstupu po generování písemné práce. Jedná se o jednu otázku s jednoduchým textem.

Ukázka kódu 3.1: Výstup aplikace pro Wolfram Mathematica

```
Notebook[{

Cell[CellGroupData[{

Cell[TextData[StyleBox["CAO Test 1", FontWeight->"Bold
    "]], "Title", TextAlignment->Center],

Cell[TextData[{
"Toto je ukazka textu."
}], "Text", Background->RGBColor[0.87, 0.94, 1]],

Cell[BoxData[RowBox[{"(* sem napiste reseni *)", "\[
    IndentingNewLine]", "\[IndentingNewLine]"}]], "Input
    "]

}, Open  ]]},
WindowSize->{1272, 698},
WindowMargins->{{0, Automatic}, {Automatic, 0}},
PrivateNotebookOptions->{"VersionedStylesheet"->{"
    Default.nb"[8.] -> False}},
Magnification:>FEPrivate`If[
FEPrivate`Equal[FEPrivate`$VersionNumber, 6.], 1.5, 1.5
    Inherited],
FrontEndVersion->"10.0 for Microsoft Windows (64-bit) (
    September 9, 2014)",
StyleDefinitions->"Default.nb"
]
```

Obrázek 3.2: Výsledný výstup testu ve formátu NB

3.8.2 Rozbor kódu

Tento formát je ve své podstatě podobný ostatním formátům pro uchovávání dat jako například XML. Naštěstí je také v plaintextu, což zlepšuje čitelnost i upravitelnost; není třeba zkoumat hlavičku souboru či dekódovat data.

Notebooky se skládají dohromady z různých tagů, které v sobě můžou mít další tagy. Každý tag může mít svoje vlastní parametry. Každý soubor v tomto formátu obsahuje kořenový tag *Notebook*, do kterého se potom vkládá veškerý obsah včetně meta-parametrů jako velikost okna apod.

Základním stavebním kamenem obsahu jsou buňky. Buňky se v kódu vyznačují tagem *Cell*. Tyto buňky se dají slučovat do hierarchických skupin a poté se v programu může s těmito celými skupinami pracovat. Skupina buněk se vyznačí jako *CellGroupData*.

Do buněk se potom dají vkládat další objekty, například formátovaný text, obrázky či další Wolframovský kód. Za zmínku ještě stojí tag *RowBox*, který slouží pro uložení souvislého seznamu dalších libovolných objektů.

Největší problém vytváření těchto souborů externě způsobují obrázky. Mathematica má svoji vlastní kompresi obrázků, která není nikde veřejně popsána a to způsobí potíže při implementaci.

Na konci každého Notebooku potom následují metadata, jako např. velikost okna či verze operačního systému. Tato metadata umožňují programu

rozpoznat verzi formátu, aby věděl, jak daný soubor otevřít.

Kapitola 4

Návrh knihovny a aplikace

Tato kapitola se věnuje návrhu především knihovní části této práce.

4.1 Popis entit systému

4.1.1 Element

Elementy jsou základními stavebními kameny každé otázky. Knihovna je navržena tak, aby se snadno daly upravovat či přidávat další. Jak lze vidět z diagramu, knihovna obsahuje následující typy elementů:

Formátovaný text Obyčejný text, kterému se dá nastavit, zda má být tučný, kurzívou apod.

Odstavec Blokový element, který do sebe umožňuje vkládat další pod-elementy, které jsou potom uzavřeny v tomto kontejneru.

Náhodná hodnota Element, kterému se zadá počáteční hodnota, koncová hodnota a krok, a on poté vygeneruje náhodné celé číslo z tohoto rozsahu.

Neřazený seznam Obyčejný seznam jiných elementů, které se potom zobrazí zarovnány pod sebou.

Obrázek Obrázek vložitelný na libovolné místo v textu.

Wolframovský kód Čistý kód, který bude vložen do výstupu neupravený.

4.1.2 Otázka

Každá písemná práce musí obsahovat otázky. Otázky se dají ručně upravovat v souboru. Otázka bude obsahovat následující atributy:

Název otázky Slouží pro lepší identifikaci při upravování otázek.

Elementy Seznam elementů, které otázka obsahuje.

Tagy Každá otázka může mít libovolné množství tagů, která se poté dají filtrovat v pravidlech.

Obtížnost Jedním z nejdůležitějších atributů každé otázky je její obtížnost. Knihovna rozlišuje celkem pět stupňů obtížnosti. Obtížnost se dá využít jako podmínka při skládání výsledných písemných prací.

Volné místo U každé otázky lze vyhradit, kolik volného místa potřebuje pro odpověď.

Typ otázky Lze specifikovat pro každou otázku, zda slouží pro papírové testy, testy do Mathematicy či do obou.

4.1.3 Sada otázek

Otázky se dají shlukovat do množin. Sady otázek se potom dají načítat a ukládat do ručně upravovatelného formátu. Pokud má uživatel více sad otázek vedle sebe, je možné je za běhu sloučit dohromady do jedné sady.

4.1.4 Pravidlo

Pravidla se využívají při generování písemných prací ze sady otázek. Slouží pro vyfiltrování otázek, aby uživatel měl možnost ovlivnit, které otázky a v jakém pořadí se v testu objeví.

Bude existovat několik druhů pravidel, s možností velmi snadno přidat další. Každé pravidlo bude umět říct, zda splňuje danou otázku. V základu budou pravidla schopna vymezovat tyto vlastnosti:

- Obtížnost
- Tag
- Název otázky

Budou také existovat speciální „logická“ pravidla, která uživateli umožní skládat více pravidel dohromady pomocí logických spojek a tvořit tím komplexní pravidla.

4.1.5 Sada pravidel

Stejně jako otázky, i pravidla se dají shlukovat do sad. Ty se potom dají také načítat a ukládat do ručně upravitelného formátu. Každá sada pravidel bude obsahovat následující atributy:

Počet otázek Určuje, kolik otázek se má objevit ve výsledném testu.

Název testu Název, který by výsledný test vygenerovaný podle těchto pravidel měl mít.

Druh testu Uživatel si vybírá, zda má výstupem být test na papír či test do programu Wolfram Mathematica.

Seznam pravidel Pro každou otázku jsou specifikována pravidla, která daná otázka musí splňovat.

4.1.6 Tag

Krátký textový popisek, který se dá přiřadit libovolnému množství otázek. Slouží pro kategorizaci otázek do uživatelem určených skupin.

4.1.7 Písemná práce

Výsledný test složený z programem vybraných otázek. Knihovna takto uloženou písemnou práce dokáže uložit jako výstup do formátu, který byl specifikován v sadě pravidel. Obsahuje tyto atributy:

Název testu Název, který se ve výstupu objeví jako nadpis celého testu.

Druh testu Buď tisknutelný formát či Notebook, podle tohoto knihovna pozná, který výstup má použít.

Seznam otázek Nyní již seřazený seznam výsledných otázek, které byly vybrány pro test.

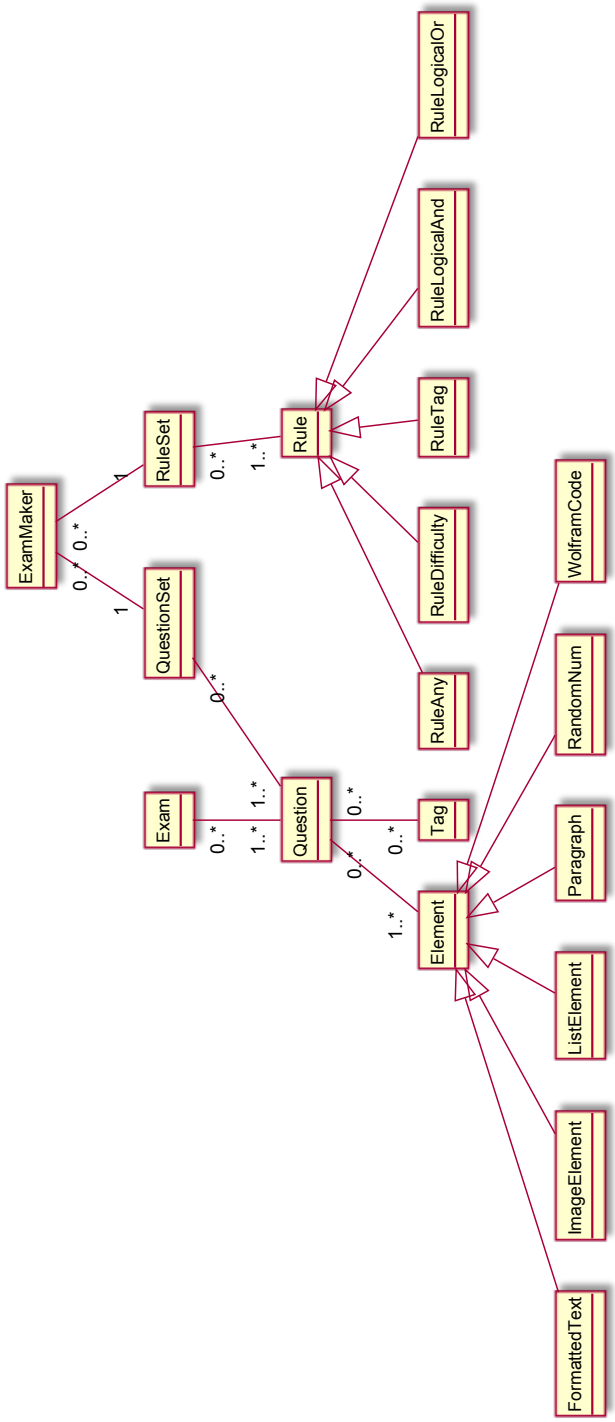
4.1.8 Generátor písemných prací

Stěžejní entita, která zodpovídá za hlavní činnost - skládání písemných prací. Generátoru se předá sada (či více sad) otázek, sada pravidel a on vrátí písemnou práci, která splňuje stanovené podmínky.

4.2 Doménový model

Účelem doménového modelu je vytyčit základní entity, které aplikace využívá a graficky popsat vztahy mezi nimi. Diagram doménového modelu níže byl vytvořen pomocí programu PlantUML [5].

Obrázek 4.1: Doménový model knihovny



4.3 Výběr technologií

Vzhledem k tomu, že práce je fakticky rozdělena do dvou částí - knihovny a ukázkové aplikace, je nutno toto zohlednit při výběru technologií, které se při realizaci projektu budou využívat.

Jedním z hlavních požadavků bylo, aby program byl multiplatformní, a dalším důležitým požadavkem bylo, aby k používání nebyl potřeba přístup k internetu. Je potřeba vybrat si takové technologie, které splňují obojí.

4.3.1 Jazyk

Jako první věc, než se vůbec může začít programovat, je nutné si vybrat vhodný programovací jazyk. S ohledem na požadavky a na svoje vlastní zkušenosti jsem se nakonec rozmýšlel mezi celkem třemi jazyky.

4.3.1.1 C++

C++ je poměrně moderní a vyspělý jazyk, se kterým se dá dělat v podstatě cokoliv, navíc podporuje objektově orientované programování. Tento jazyk splňuje požadavek multiplatformnosti, dá se zkompilovat v podstatě na čemkoliv.

Mezi hlavní klady patří fakt, že je možné knihovnu více optimalizovat a mít větší kontrolu nad tím, co se děje. Toto je ale zároveň zápor, protože pro účely této práce žádné sofistikované optimalizace potřebné nejsou. Současně by bylo potřeba starat se o více věcí jako např. správu paměti apod. a programovací cyklus by se zbytečně prodloužil.

Zároveň je v této situaci nevýhodný fakt, že je projekt potřeba kompilovat pro každou platformu zvlášť a dělat menší úpravy v kódu může být nepohodlné.

4.3.1.2 Python

Mou další volbou byl skriptovací jazyk Python. V Pythonu člověk také zvládne udělat v podstatě cokoliv, ale podstatně pohodlněji a rychleji. Tento jazyk také podporuje objektově orientované programování a samozřejmě je multiplatformní.

Mezi klady patří to, že jazyk je interpretovaný, veškeré úpravy je možné vyzkoušet ihned. Python s sebou také má správce balíčků, čili doinstalovávat další knihovny je triviální. Jako zápor bych uvedl to, že objektové programování v Pythonu není moc striktní, umožňuje spoustu nepříjemných věcí a v při velkém množství souborů se člověk může začít ztrácet.

4.3.1.3 Java

Posledním jazykem, o kterém jsem uvažoval, a zároveň jazykem, který jsem si nakonec vybral, je Java. Tento jazyk je čistě objektově orientovaný, čili veškeré koncepty a návrhové vzory se zde skvěle aplikují.

Jedním z velkých kladů je přenosnost - aplikaci stačí zkompilovat jen jednou a výsledná binární podoba se dá interpretovat v podstatě na všech známých operačních systémech. Další výhodou je rozsáhlá standardní knihovna a obrovské množství dalších knihoven. Navíc s tímto jazykem mám zkušenosti. Co se týče nevýhod, hlavní nevýhodou je, že Java vnucuje určitý styl programování, ale pro tento projekt to není problém.

4.3.2 Vstupní formáty

Vstupní formát otázek i pravidel musí být ručně editovatelný v libovolném textovém editoru. Je možné vybrat si mezi spoustou možností, např. JSON, XML, YAML apod. Všechny tyto formáty v jádru fungují víceméně stejně, ale liší se syntaxí. Pro potřeby této práce jsem si vybral XML, protože je to známý formát a existuje velké množství knihoven, které XML podporují.

Pro serializaci a deserializaci dat používám knihovnu XStream [6], která umožňuje převádět jednodušše celé objekty do XML a potom je znovu načítat. Knihovna je nakonfigurována tak, aby výsledné XML bylo intuitivní a snadno čitelné a upravitelné.

4.3.3 Výstupní formáty

Knihovna má za úkol vytvářet jak písemné práce tisknutelné na papír, tak testy pro software Wolfram Mathematica. Je nutné tedy zvolit vhodný tisknutelný formát.

V úvahu jsem bral celkem tři formáty: PDF, Office Open XML a HTML. Problém s PDF je ten, že po vygenerování testu by již nebylo možné doupravit formátování apod. U formátu Office Open XML (koncovka .docx) je zase poměrně složitá implementace a je potřeba mít program, který takový formát dokáže otevřít.

Nakonec jsem se tedy rozhodl pro formát HTML, který lze běžně otevřít v libovolném prohlížeči. Nativně podporuje odstavce, obrázky, seznamy a podobné elementy, které se přímo v této knihovně využívají. To povede k pohodlné implementaci. Navíc si výsledný test může uživatel pomocí externích stylovacích souborů přizpůsobit podle sebe a zajistit, aby např. otázky nepřecházely přes stránku, a další podobné detaily.

4.4 Diagram tříd

Z doménového modelu po vyhodnocení použitých technologií vyplývá následující model tříd.

4.5 Ukázková aplikace

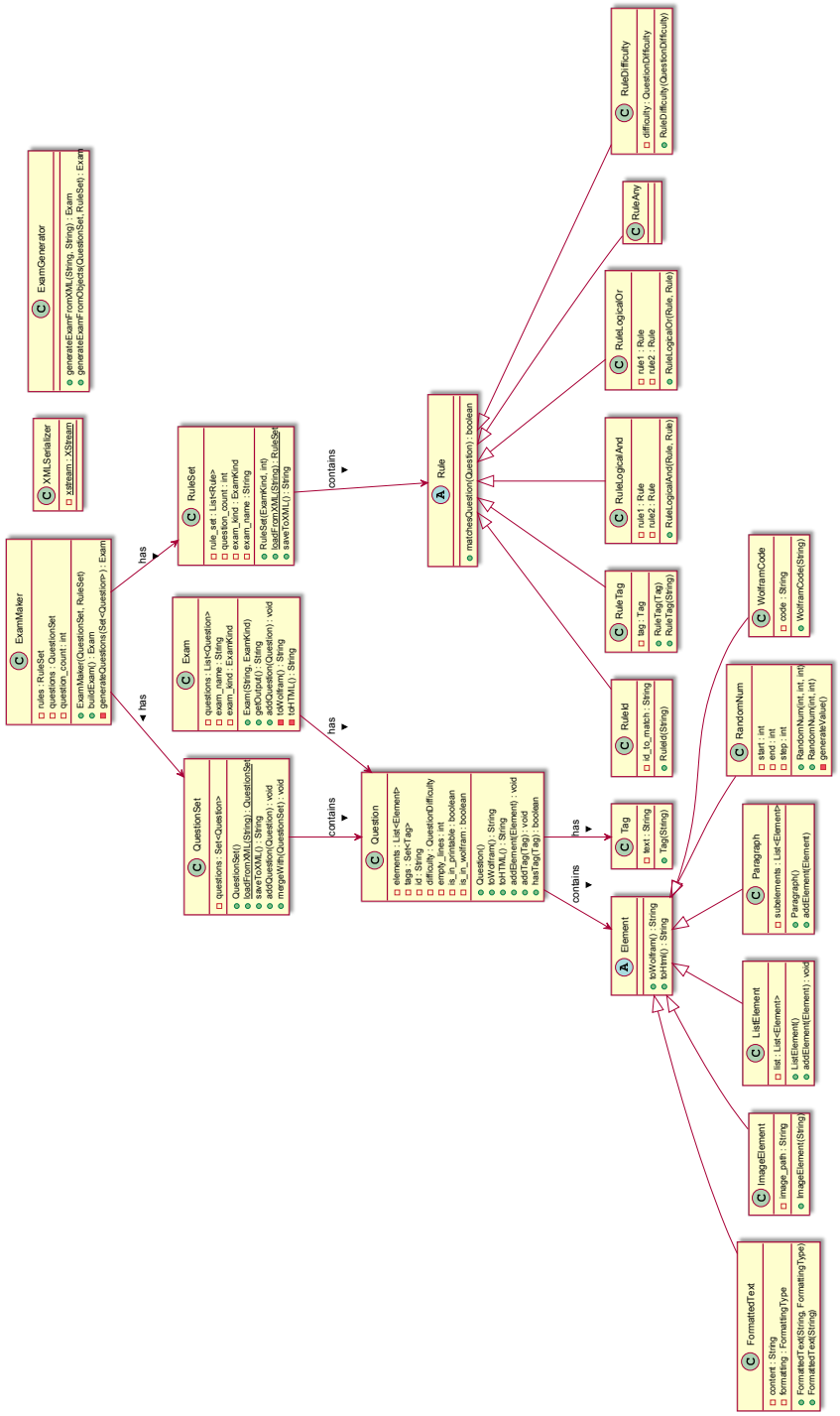
Vzhledem k povaze práce by pořádná aplikace s podporou WYSIWYG[2] editování byla velmi složitá, skoro v rozsahu jiné bakalářské práce. S ohledem na náročnost tedy bylo rozhodnuto, že pro teď se udělá konzolová aplikace.

Aplikace je navržena tak, že se jí jako parametr předá sada pravidel a potom jedna či více sad otázek. Tyto sady otázek si program potom umí interně spojit v jednu a použít pro generování písemných prací.

Program také umí podle aktuální specifikace automaticky vygenerovat ukázkovou sadu otázek a pravidel ve formátu XML, ze kterých se dá ihned postavit ukázkový test demonstrující všechny elementy.

[2]Editor s náhledem výstupu

Obrázek 4.2: Třídní diagram knihovny



Kapitola 5

Implementace

V této kapitole jsou obsaženy veškeré informace týkající se použitých technologií a bude zde detailně popsána realizace některých částí knihovny.

5.1 Použité nástroje

Knihovna i ukázková aplikace budou napsány v Javě, což ovlivní konkrétní nástroje, které budou použity.

5.1.1 Vývojové prostředí

Pro vývoj Javy je zpravidla doporučeno používat nějaké IDE[3]. Na tento projekt jsem se rozhodl použít IntelliJ IDEA [7], konkrétně verzi 15. Má velké množství funkcí jako např. chytré doplňování kódu, spouštění testů, refaktorizaci, zabudované verzování apod.

5.1.2 Organizace projektu

Pro lepší organizaci projektu a kompilaci se bude používat systém Maven [8]. Maven má specifickou adresářovou strukturu, podle které se řídí zdrojový kód i testy. Konfigurace probíhá pomocí souboru s názvem `pom.xml`. Zde se také dají nastavit další knihovny, které projekt využívá, a Maven si je potom při kompilaci stáhne ve verzi, která je požadována.

5.1.3 Správa verzí

Pro správu verzí se využívá verzovací systém Git [9]. Repozitář je potom vzdáleně synchronizován s půjčeným serverem, na kterém běží Bitbucket [10].

[3] vývojové prostředí

5.2 Implementace knihovny

Knihovna se skládá z několika klíčových částí, které se zde pokusím nějakým způsobem popsat z pohledu implementačního.

5.2.1 Elementy

Jak již bylo řečeno výše, elementy jsou stavebními kameny každé otázky. Každá otázka obsahuje seřazený seznam elementů jak jdou za sebou, a z toho se potom vygeneruje správný výstup.

Element je abstraktní třída, která obsahuje dvě další abstraktní funkce, jednu pro výstup do HTML a druhou pro výstup do programu Wolfram Mathematica. To v praxi znamená, že každý element se umí sám konvertovat do obou výstupních formátů, což výrazně zpříjemňuje konečný výstup programu.

Výhodou tohoto řešení je, že je triviální přidávat další elementy, stačí jen zdědit rodičovskou třídu a nadefinovat výstupní funkce. Ergo je jednoduché i přidávat další typy výstupů, v takovém případě je potřeba jen přidat další funkci.

5.2.2 Pravidla

K tomu, aby byla vygenerovaná písemná práce, jsou potřeba nějaká pravidla. Jako rodič slouží opět abstraktní funkce Rule, která obsahuje jen jednu abstraktní funkci. Této funkci se jako parametr předá nějaká otázka a pravidlo vrátí, zda jej daná otázka splňuje či nikoliv.

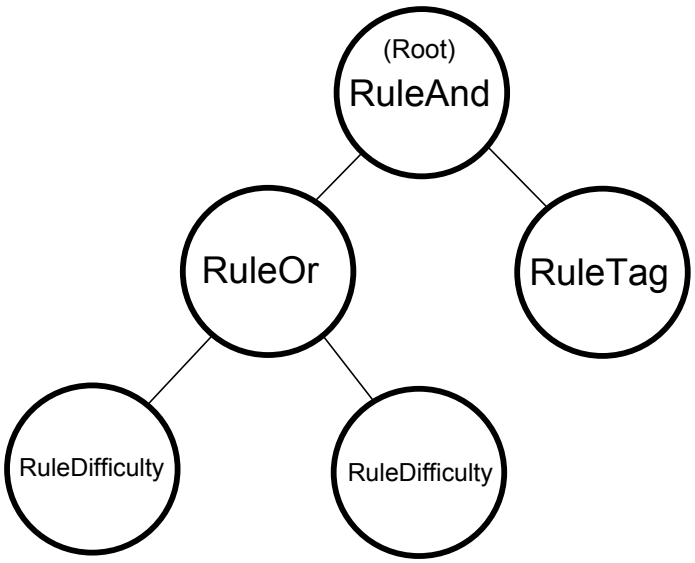
Ve svém jádru jsou pravidla jednoduchá. Zpravidla omezují jen základní parametry otázek, jako tagy, které obsahuje, či obtížnost. Pravidla jsou udělána tak, že každý test připraven ke generování má vyhrazenou sadu pravidel pro každou otázku. Např. chci test co má 3 otázky, první otázka má nějaká pravidla, druhá jiná a třetí třeba žádná. Největší problém zde bylo vyřešit, jak pravidla kombinovat.

Nejdříve mně napadlo prostě každé potenciální otázce dát seznam pravidel, která má aplikovat. Toto by fungovalo bez problémů, ale jednou z nutností bylo, aby se pravidla dala kombinovat i logicky, např. „Tato otázka musí obsahovat tag SAP1 a zároveň nesmí obsahovat tag SAP2“. Když bych měl pouhý seznam pravidel, toto by nefungovalo.

Rozhodl jsem se tedy pro jiný postup. Každá otázka má dovoleno jen jedno pravidlo. Pro skládání pravidel dohromady jsem zavedl tzv. „logická“ pravidla, kterým se při konstrukci předají 2 či více pravidel a ony je vyhodnotí. Potom

když mám nějaké komplikovanější pravidlo, stačí se zeptat tohoto kořenového pravidla, které pak rekurzivně vyhodnotí celý tento strom. Ukázku nějakého složeného pravidla jsem se rozhodl demonstrovat na obrázku. Toto pravidlo může sloužit k něčemu jako „Chci otázku, která obsahuje tag SAP1 a zároveň je buď těžká či velmi těžká“.

Obrázek 5.1: Ukázka složeného pravidla



Toto řešení bohatě stačí na momentální způsob skládání testů, ale do budoucna by se dalo ještě vylepšit. Konkrétně dle momentálního návrhu nejde dělat otázky závislé na ostatních. V praxi to znamená, že v tuto chvíli knihovně nejde říct „V testu chci mít aspoň 2 otázky s tagem CAO1a“. Toto je určitě věc, kterou by stálo za to v budoucnu napravit, protože by mohla být užitečná.

5.2.3 Generování písemných prací

Je také samozřejmě nutné vysvětlit, jak probíhá samotný proces generování testů.

Za vlastní tvorbu písemných prací zodpovídá třída ExamMaker. Konstruktoru této třídy se předá sada otázek a sada pravidel. Tato třída pak v několika krocích vytvoří test dle zadaných parametrů. Konkrétně v těchto krocích:

1. Generátoru se předá sada otázek a sada pravidel.
2. Z otázek se vyfiltrují ty, které nejsou kompatibilní s druhem výstupu.
3. V písemné práci se vytvoří dostatečný počet míst podle množství cílových otázek uvedeného v pravidlech.

4. Pro každý tento „slot“ se vyberou všechny otázky, které splňují přidružené pravidlo.

5. Ze všech možných otázek se náhodně vybere jedna.

6. Vybraná otázka se přidá do písemné práce a proces pokračuje pro všechny potenciální otázky.

S tímto procesem může nastat jeden specifický problém. Vzhledem k tomu, že pro každou potenciální otázku se nejprve vyberou všechny možné otázky ze sady a pak z nich se náhodně zvolí jedna, může se stát, že při určitých kombinacích test již nepůjde složit.

Pro lepší představu uvedu příklad. Budu mít jednu těžkou otázku s tagem SAP1 a jednu velmi těžkou otázku s tagem SAP2. Uživatel si do pravidel zadá, že test má obsahovat dvě otázky. První otázka má být buď těžká či velmi těžká a druhá otázka musí obsahovat tag SAP2. V tomto případě se může stát, že program pro test vybere jako první otázku již zmiňovanou velmi těžkou, ale potom už nebude v sadě existovat otázka, která by splňovala i podmínky druhé otázky.

Rozhodl jsem se tento problém vyřešit jednodušše. Vzhledem k tomu, že test zpravidla nebude mít více než 10 otázek a generování probíhá rychle, při selhání skládání testu se celý proces opakuje, a to až tolikrát, kolik má program nastaveno pokusů. V tuto chvíli je celkový počet pokusů nastaven na 100, což se zdá přiměřené.

5.2.4 API

V této knihovně probíhá veškerá logika a je dobré dát jiným aplikacím, co ji využívají, lepší přístup k hlavní funkcionalitě.

Z tohoto důvodu v knihovně existuje třída ExamGenerator, která obsahuje jen dvě funkce. Buď se ji předá validní XML s otázkami a pravidly, nebo již vykonstruované objekty těchto sad. Při psaní nějakých jiných aplikací, např. webových, by se měla využívat právě tato třída.

5.3 Implementace ukázkové aplikace

Jak již bylo řečeno dříve, rozhodl jsem jako ukázkový program vytvořit jednoduchou konzolovou aplikaci.

5.3.1 Vstup a výstup

Aplikace z příkazové řádky přijímá jako parametry soubor se sadou pravidel a jeden či více souborů s otázkami. Sady otázek se potom interně spojí dohromady v jednu a předají knihovně. Pro načítání parametrů z příkazové řádky využívám knihovnu JCommander [11], ve které se snadno dají nastavit druhy argumentů a další vlastnosti, jako např. zda je daný argument povinný či nikoliv.

5.3.2 Generování ukázek

Konzolová aplikace také po předaní správného parametru dokáže podle aktuální verze knihovny vygenerovat funkční XML soubory s otázkami a pravidly připravené k použití. Toto slouží jako ukázka pro uživatele, kteří chtějí vidět momentální formát otázek a pravidel, a mohli si podle něj vytvořit vlastní.

Kapitola 6

Testování

Jednou z nejdůležitějších částí softwarového vývoje je testování. Cílem testů je odhalit problémy v implementaci, především základní funkcionalitu a mezní případy. Některé testy se musí provádět manuálně, ale většinou stačí automatické. Testy také pomáhají objevit problémy, které již dříve byly opraveny, ale z nějakého důvodu se znovu objevily.

6.1 Použité nástroje

Pro testování této knihovny využívám testovací framework JUnit [12] verze 4. Tento framework umožňuje snadnou integraci s Javou a pohodlné psaní testů. Pro ještě větší produktivitu obsahuje mnou používané IDE přímou podporu pro JUnit.

6.2 Jednotkové testy

Jednotkové testy, nebo Unit testy, jak již název napovídá, testují individuální jednotky, v našem případě třídy. Jejich cílem je zajistit, že každá třída funguje tak, jak se od ni očekává.

Vzhledem k tomu, že velkou část funkcionality programu zajišťuje standardní knihovna, či nějaká jiná knihovna, která má vlastní testy, není nutné provádět mnoho jednotkových testů. Správnost většiny programu je zajištěna automaticky.

V mém případě jsem testoval převážně mnou implementované složitější funkce, tzn. žádné gettery a settery. Za zmínku stojí např. test slučovaní sad otázek dohromady, včetně mezních případů.

6.3 Integrační testy

Integrační testy potom jsou testy většího rozsahu, kde se testuje více jednotek společně či dokonce celý systém najednou.

V tomto projektu existuje test, který zadá otázky a pravidla tak, aby existoval jen jeden způsob, jak je složit. Test potom ověřuje, že výsledný test jde složit a zároveň že podává očekávaný výstup (jak HTML, tak Wolfram kód).

Závěr

Cílem této práce bylo vytvořit knihovnu a ukázkovou aplikaci pro tvorbu zadání písemných prací. Aplikace by ze začátku měla sloužit především pro předměty technicky či matematicky založené, ale dá se použít v podstatě pro cokoliv.

Osobně bych návrh a implementaci zhodnotil jako úspěšné. Podařilo se splnit zadání i všechny funkční a nefunkční požadavky. Návrh je poměrně čistý a využívá principů objektově orientovaného programování. Implementace se dle návrhu dělala pohodlně a co bylo potřeba otestovat, se otestovalo. Přidávání nových elementů pro jiný druh testů je v tuto chvíli triviální.

Samozřejmě jsou zde věci, které by se do budoucna daly přidat či zlepšit. Kdybych práci dělal znovu, vytvořil bych pro potřeby ukládání otázek a pravidel místo XML nějaký vlastní formát, který by nebyl tak „upovídaný“ a lépe reprezentoval elementy, které se používají. Další věcí, kterou bych mohl zlepšit, by byl systém pravidel. Momentálně má každá otázka svoje vlastní pravidla, a neexistuje způsob, jak udělat jednou otázku závislou na jiné (navazující otázky apod.).

Literatura

[1] Moodle Quiz Module. [online], [cit. 2016-04-24]. Dostupné z: https://docs.moodle.org/25/en/Quiz_module

[2] Blackboard Tests, Surveys and Pools. [online], [cit. 2016-04-24]. Dostupné z: https://en-us.help.blackboard.com/Learn/9.1_2014_04/Instructor/110_Tests_Surveys_Pools

[3] DoTest. [online], [cit. 2016-04-24]. Dostupné z: http://www.dotest.cz/

[4] Wolfram Language & System Documentation Center. [online], [cit. 2016-04-27]. Dostupné z: http://reference.wolfram.com/language/

[5] PlantUML : Open-source tool that uses simple textual descriptions to draw UML diagrams. [online], [cit. 2016-04-29]. Dostupné z: http://plantuml.com/

[6] XStream. [online], [cit. 2016-05-04]. Dostupné z: http://x-stream.github.io/

[7] IntelliJ IDEA the Java IDE. [online], [cit. 2016-05-05]. Dostupné z: https://www.jetbrains.com/idea/

[8] Apache Maven. [online], [cit. 2016-05-05]. Dostupné z: https://maven.apache.org/

[9] Git. [online], [cit. 2016-05-05]. Dostupné z: https://git-scm.com/

[10] Bitbucket. [online], [cit. 2016-05-05]. Dostupné z: https://bitbucket.org/

[11] JCommander. [online], [cit. 2016-05-08]. Dostupné z: http://jcommander.org/

[12] JUnit. [online], [cit. 2016-05-12]. Dostupné z: `http://junit.org/junit4/`

[13] Gamma, E.; Helm, R.; Johnson, R.; aj.: *Design Patterns: Elements of Reusable Object-oriented Software*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995, ISBN 0-201-63361-2.

Příloha **A**

Seznam použitých zkratek

GUI Graphical user interface

XML Extensible markup language

LMS Learning Management System

NB Notebook, název formátu, který využívá software Wolfram Mathematica

WYSIWYG What You See Is What You Get - editor s vizualizací výstupu

IDE Integrated Development Environment - vývojové prostředí

API Application programming interface

Příloha B

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `bin` ..... adresář se spustitelnou formou implementace
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF