VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

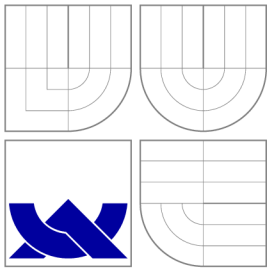
APLIKACE PRO VÝUKU PRÁCE S REGULÁRNÍMI VÝRAZY

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

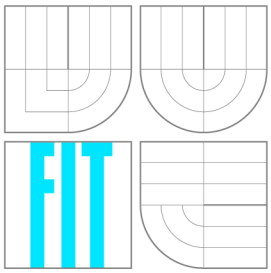
AUTOR PRÁCE
AUTHOR

TEREZA HEROUTOVÁ

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

APLIKACE PRO VÝUKU PRÁCE S REGULÁRNÍMI VÝRAZY

APPLICATIONS FOR TEACHING WORK WITH REGULAR EXPRESSIONS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

TEREZA HEROUTOVÁ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. JIŘÍ KOUTNÝ

BRNO 2009

Abstrakt

Dokument se zabývá vznikem aplikace, která umožňuje interaktivní výuku práce s regulárními výrazy. Velký důraz je kladen na názorné podání látky a efektivní uživatelské rozhraní tak, aby ovládání bylo co nejvíce intuitivní. V dokumentu je popsána teorie regulárních výrazů, pedagogická stránka věci, ale také vlastní implementace a možnosti budoucího rozšíření.

Abstract

The document describes the formation of an application, which allows interactiv instruction how to work with regular expressions. We stress clear interpretation of material and effective user interface to make handling intuitive as much as possible. The document describes theory of regular expressions, pedagogic side of the things and implementation with feature extensions.

Klíčová slova

regulární výrazy, grafické uživatelské rozhraní, didaktika, XML, jazyk Java, knihovna Swing

Keywords

regular expressions, graphical user interface, didactics, XML, Java language, Swing toolkit

Citace

Tereza Heroutová: Aplikace pro výuku práce s regulárními výrazy, bakalářská práce, Brno, FIT VUT v Brně, 2009

Aplikace pro výuku práce s regulárními výrazy

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracovala samostatně pod vedením pana Ing. Jiřího Koutného.
Uvedla jsem všechny literární prameny a publikace, ze kterých jsem čerpala.

.....
Tereza Heroutová
18. května 2009

Poděkování

Tímto chci poděkovat vedoucímu své bakalářské práce za průběžné konzultace a trpělivé zodpovídání dotazů, dále své rodině za diskuse o tématech spojených s prací a také svému příteli nejen za psychickou podporu.

Obsah

Kapitola 1

Úvod

Motto [17]:

> Zeptejte se jakéhokoliv relativně zkušeného *NIX uživatele na 10 nejlepších věcí okolo operačního systému, a téměř vždy v jeho mumlání, někde mezi „99% spolehlivostí“ a „vzdáleným rebootováním systému“, uslyšíte „REGULÁRNÍ VÝRAZY“.
>
> Zeptejte se jakéhokoliv relativně zkušeného *NIX uživatele na 10 nejprotivnějších věcí okolo operačního systému, a někde mezi „zombie procesy“ a „instalací“ uslyšíte „REGULÁRNÍ VÝRAZY“.

Přestože regulární výrazy nemusí být v oboru informatiky předmětem každodenní práce, mohou v určitých situacích zpříjemnit a hlavně urychlit práci s textem. Dají se používat vědomě a cíleně v různých unixových aplikacích (například pro zpřehlednění zdrojového kódu odstaněním více prázdných řádků za sebou), ale spousta uživatelů je používá aniž by o nich věděla. Nejznámější jsou nejspíš textové editory jako je MS Word, kde jsou dostupné funkce pro vyhledávání a nahrazování textu. Jako příklad lze uvést situaci, kdy je v textovém dokumentu na několika místech špatně uvedené příjmení. Uživatel může dokument projít ručně a chybu na všech místech opravit, anebo může použít automatické vyhledávání a nahrazování, ve kterém hrají hlavní roli právě regulární výrazy.

Jsou také využívány v několika programovacích jazycích a už tyto dva fakty mohou být důvodem pro seznámení se s nimi a naučení se alespoň základů, na které se dá relativně lehce navázat. Naučení může probíhat formou samostudia, kdy si zájemce najde literaturu, obvykle z více zdrojů, a nejlépe metodou pokus-omyl regulární výrazy rovnou aplikuje. Nabízí se ale cesta elegantnější, pohodlnější a snad i efektivnější, a to přes výukový program. Právě tato cesta je tématem mé bakalářské práce.

V kapitole 2 je popsána teorie regulárních výrazů, včetně demonstrace na příkladech a rozebrání některých konstrukcí, jež se dají uplatnit v praxi. Kapitola 3 se zabývá přípravami na implementaci. Hlavními body jsou zde anketa a na jejím základě vytvořené didaktické pojetí výuky, dále uživatelský návrh a popis programového návrhu (i ve formě diagramu). Kapitola 4 obsahuje detailní popis implementace rozdělený na větší logické celky a také vývoj prací na projektu. V kapitole 5 je celkové shrnutí, tipy na možná rozšíření a popis použitých nástrojů.

Kapitola 2

Regulární výrazy

2.1 Historie

Základy regulárních výrazů [13] vychází z teorie automatů a teorie formálních jazyků. Tyto jsou obě součástí teoretické informatiky a obě zkoumají způsoby, jak popsat a klasifikovat formální jazyky. V 50. letech 20. století popsal matematik Stephen Cole Kleene tyto modely pomocí matematického zápisu zvaného regulární množiny (regular sets). Jazyk SNOBOL byl první, ovšem ne identickou, implementací porovnávání řetězců. Ken Thompson zakomponoval Kleensův zápis do interaktivního textového editoru QED na vyhledávání řetězců v textových souborech. Tuto schopnost později přidal do unixového editoru ed a ten nakonec vedl k vytvoření vyhledávacího nástroje grep, který používá regulární výrazy. Od této doby se regulární výrazy využívají ve velkém počtu v unixových aplikacích, jako například *expr, AWK, Emacs, vi* a *lex*.

2.2 Využití regulárních výrazů

Regulární výrazy se používají pro práci s textem – zejména pro vyhledávání určitých pasáží v souboru (grep), vyhledání a nahrazování řetězců (sed), analýza a výpočet údajů (awk) [11], ověřování správnosti zadaných dat (telefonní číslo, e-mailová adresa, datum narození) nebo parsování kódu (HTML, CSV, XML). Prostřednictvím regulárních výrazů definujeme vzory, které se aplikují na daný text.

Regulární výrazy jsou jednoduchým, ale nesmírně mocným nástrojem. Lze je najít v mnoha unixových programech a poměrně velkém počtu programovacích jazyků – C#, Java, Visual Basic, .NET, Perl, PHP, Javascript.

2.3 Teorie regulárních výrazů

Nejjednoduššími vzory jsou znaky – číslice, písmena a různé symboly (*metaznaky*). Lze zadat počet opakování vzoru (*kvantifikátory*) nebo přesně určit místo hledání (*hranice*), pro snadnější práci existují též *předdefinované skupiny znaků*.

V následujícím textu budou všechny tyto i další části podrobně rozebrány. Pro lepší pochopení jsou poznatky shrnuty do tabulek, stejně tak jako příklady, u kterých je navíc pro názornost **zvýrazněn** výsledek aplikace regulárního výrazu na prohledávaný řetězec. U sekce „Příklady náročnějších konstrukcí“ jsou uváděny zdrojové kódy a pseudokódy, ty jsou odlišeny `stylem písma`.

Regulární výrazy mají při použití v různých jazycích mírně odlišnou podobu, zde jsou probrány základy, které umožňují získat obecný přehled, a mutace pro jednotlivé použití si lze dostudovat samostatně. Výklad, tabulkové přehledy a ukázky na příkladech byly inspirovány těmito zdroji: [16], [2], [14] a [15].

2.3.1 Tečka

Tečka je zástupný symbol pro libovolný znak. Lze ji použít v situaci, kdy na hodnotě některého znaku hledaného řetězce nezáleží. Jediný znak, který nemůže tečka zastoupit, je znak konce řádku. Je také důležité, že za tečku musí být nějaký znak přiřazen, nelze ji zcela ignorovat.

testovaný řetězec	Regulární výrazy jsou prima.
regulární výraz	.
první výskyt	**R**egulární výrazy jsou prima.
druhý výskyt	R**e**gulární výrazy jsou prima.
všechny výskyty	**Regulární výrazy jsou prima.**

Tabulka 2.1: Použití tečky jako zástupce kteréhokoli znaku

Například potřebujeme-li ze zdrojového textu HTML stránky získat začátky číslovaných a nečíslovaných seznamů, tedy tagy `<OL>` a `<UL>`. Oba mají stejné všechny znaky kromě druhého, čehož můžeme využít a vyhledávat je pomocí shodného regulárního výrazu. Jeho podoba a použití by bylo následující.

regulární výraz	`<.L>`
první výskyt	najde `<OL>` a `<UL>`, ale také `<BL>`, `<XL>`, `<EL>`, ...
poznámka	vyhovuje libovolný čtyřznakový řetězec začínající `<` a končící `L>`

Tabulka 2.2: Použití tečky v praxi

2.3.2 Předdefinované skupiny znaků

Tyto skupiny zkracují zápis některých často používaných sekvencí, např. pouze číslice, pouze alfanumerické znaky a podobně. Výrazně tím přispívají ke srozumitelnosti regulárních výrazů. Například chceme-li v textu najít pouze alfanumerické znaky (včetně znaku _), můžeme použít zápis `[a-zA-Z0-9_]` nebo zkrácený a přehlednější zápis `\w`.

Tabulka 2.3 ukazuje přehled těchto zkratek.

zkratka	význam	ekvivalentní zápisu[1]
`\d`	číslice 0-9	`[0-9]`
`\D`	jakýkoli znak kromě číslic	`[^0-9]`
`\w`	alfanumerické znaky a „_“	`[a-zA-Z0-9_]`
`\W`	jakýkoli znak kromě alfanumerického	`[^a-zA-Z0-9]`
`\s`	bílé znaky (mezera, tabulátor, ...)	`[\t\n\r\f]`
`\S`	jakykoli znak kromě bílého	`[^\t\n\r\f]`

Tabulka 2.3: Předdefinované skupiny znaků

testovaný řetězec	Regulární výrazy, kapitola 8.3.1
regulární výraz	`\D`
první výskyt	**R**egulární výrazy, kapitola 8.3.1
druhý výskyt	R**e**gulární výrazy, kapitola 8.3.1
všechny výskyty	**Regulární výrazy, kapitola** 4.1.2

Tabulka 2.4: Použití zkratek pro vyhledání určité skupiny znaků I

testovaný řetězec	Regulární výrazy, kapitola 8.3.1
regulární výraz	`\d`
první výskyt	Regulární výrazy, kapitola **8**.3.1
druhý výskyt	Regulární výrazy, kapitola 8.**3**.1
všechny výskyty	Regulární výrazy, kapitola **8**.**3**.**1**

Tabulka 2.5: Použití zkratek pro vyhledání určité skupiny znaků II

2.3.3 Kvantifikátory

Dosud byly vysvětleny konstrukce vzorů, které měly pevně danou délku. To však v praxi není příliš využitelné, existují proto znaky pro opakování vzorů nebo jejich částí. Tyto znaky se nazývají kvantifikátory a určují, kolikrát se smí (nebo musí) opakovat znak či skupina znaků předcházející tomuto kvantifikátoru.

Tabulka 2.6 ukazuje přehled identifikátorů opakovaných výskytů.

zkratka	význam	ekvivalentní popis
`*`	libovolný počet výskytů	minimálně 0x, maximálně neomezeno
`+`	alespoň jeden výskyt	minimálně 1x, maximálně neomezeno
`?`	nejvýše jeden výskyt	minimálně 0x, maximálně 1x
`{M,N}`	nejméně M a nejvýše N výskytů	minimálně Mx, maximálně Nx
`{,N}`	nejvýše N výskytů	minimálně 0x, maximálně Nx
`{M,}`	nejméně M výskytů	minimálně Mx, maximálně neomezeno
`{M}`	právě M výskytů	ani více ani méně než Mx

Tabulka 2.6: Kvantifikátory

testovaný řetězec	aabc abc bc
regulární výraz	`a*b`
první výskyt	**aab**c abc bc
druhý výskyt	aabc **ab**c bc
všechny výskyty	**aab**c **ab**c **b**c

Tabulka 2.7: Použití kvantifikátoru *

testovaný řetězec	aabc abc bc
regulární výraz	`a+b`
první výskyt	**aab**c abc bc
druhý výskyt	aabc **ab**c bc
všechny výskyty	**aab**c **ab**c bc

Tabulka 2.8: Použití kvantifikátoru +

testovaný řetězec	aabc abc bc
regulární výraz	`a?b`
první výskyt	a**ab**c abc bc
druhý výskyt	aabc **ab**c bc
všechny výskyty	a**ab**c **ab**c **b**c

Tabulka 2.9: Použití kvantifikátoru ?

U kvantifikátoru `\*` pro libovolný počet výskytů je důležitá jedna vlastnost – opakování je hladové. To znamená, že se snaží pohltit co nejvíce možných znaků, nebo jinak řečeno roztáhnout se na co největší délku.

testovaný řetězec	jelenovi pivo nelej
regulární výraz	`j.*e`
první výskyt	**jelenovi pivo nele**j

Tabulka 2.10: Ukázka hladového opakování

2.3.4 Hranice

Někdy potřebujeme vyhledat řetězec, který se vyskytuje například na začátku nebo na konci řádku. K takovému vyhledávání slouží identifikátory pro specifikaci místa hledání, nazývané také hranice. Existuje řada programů, které zredukují více prázdných řádků na jeden. Toho se dá využít při čištění zdrojových kódů.

Tabulka 2.11 ukazuje nejčastěji používané identifikátory.

zkratka	význam
`^`	začátek řádku
`$`	konec řádku
`\A`	začátek řetězce (pokud je víceřádkový)
`\Z`	konec řetězce (pokud je víceřádkový)
`\b`	na hranici (na začátku či konci) slova (mezi `\w` a `W`)
`\B`	mimo hranici slova

Tabulka 2.11: Hranice

testovaný řetězec	Regulární výrazy jsou prima.
regulární výraz	`\b\w`
první výskyt	**R**egulární výrazy jsou prima.
druhý výskyt	Regulární **v**ýrazy jsou prima.
všechny výskyty	**R**egulární **v**ýrazy **j**sou **p**rima.

Tabulka 2.12: Použití hranice – na začátku či konci slova I

testovaný řetězec	Regulární výrazy jsou prima.
regulární výraz	`\w\b`
první výskyt	Regulárn**í** výrazy jsou prima.
druhý výskyt	Regulární výraz**y** jsou prima.
všechny výskyty	Regulárn**í** výraz**y** jso**u** prim**a**.

Tabulka 2.13: Použití hranice – na začátku či konci slova II

testovaný řetězec	Regulární výrazy jsou prima. Opravdu.
regulární výraz	`^...`
první výskyt	**Reg**ulární výrazy jsou prima. Opravdu
druhý výskyt	Regulární výrazy jsou prima. **Opr**avdu.
všechny výskyty	**Reg**ulární výrazy jsou prima. **Opr**avdu

Tabulka 2.14: Použití hranice – začátek řádku

2.3.5 Závorkové konstrukce, alternativa

Závorky tvoří velice důležitou část v tvorbě regulárních výrazů. Pro sdružování znaků, například kvůli opakování, se používají kulaté závorky `(` a `)`. Do nich se uzavře posloupnost a pokud se za ně přidá kvantifikátor, platí opakování pro celou skupinu znaků.

regulární výraz	`tel(efon)?`
možný výsledek	tel telefon

Tabulka 2.15: Použití závorkové konstrukce s kvantifikátorem I

regulární výraz	`tra(la){,3}`
možný výsledek	tra trala tralala tralalala

Tabulka 2.16: Použití závorkové konstrukce s kvantifikátorem II

Dále se mohou vyskytovat speciální závorkové konstrukce, které umožní zapojení komentáře či aplikaci tvrzení o předcházejícím či následujícím znaku. Jako příklad lze uvést situaci, kdy hledáme v textu trojmístné číslo dělitelné desíti. Regulární výraz by vypadal takto `\d{3}(?<=0)` a odpovídala by mu sekvence tří číslic, přičemž poslední by musela být 0.

Potřebujeme-li použít z určité skupiny znaků právě jeden, uzavře se tato skupina do hranatých závorek `[` a `]`. Pokud se ale chceme vyhnout použití některých znaků, do hranatých závorek doplníme znak `^`.

testovaný řetězec	Regulární výrazy jsou prima.
regulární výraz	`[aei].`
první výskyt	R**eg**ulární výrazy jsou prima.
druhý výskyt	Regulární výr**az**y jsou prima.
všechny výskyty	R**eg**ulární výr**az**y jsou pr**ima.**

Tabulka 2.17: Použití hranatých závorek

testovaný řetězec	Regulární výrazy jsou prima.
regulární výraz	`[^aei]`
první výskyt	**R**egulární výrazy jsou prima.
druhý výskyt	Reg**u**lární výrazy jsou prima.
všechny výskyty	**R**egulární výr**a**zy jsou pri**m**a.

Tabulka 2.18: Použití hranatých závorek s vyloučením

Například v situaci zmíněné v kapitole 2.3.1 lze vylepšit regulární výraz na vyhledávání tagů ohraničujících číslovaný a nečíslovaný seznam takto:

regulární výraz	`<[OU]L>`
výskyt	pouze `<OL>` a `<UL>`

Tabulka 2.19: Použití hranatých závorek, vylepšení příkladu s použitím tečky

Do hranatých závorek lze také zadat rozsah znaků, které jsou požadovány, aby se nemusely zadávat ručně například všechny znaky abecedy. V tomto případě se použije zkrácený zápis `[a-zA-Z]`, což znamená všechna malá i velká písmena v ASCII tabulce. Zde je znak pomlčky v roli oddělovače, v případě že ji chceme také zahrnout do požadovaných znaků, je nutné ji napsat hned za otevírací hranatou závorku, tedy `[-a-zA-Z]`.

Je-li potřeba vybrat z více variant textu, použije se pipe `|`. Platí to jak pro oddělení dílčích výrazů (například `ahoj|nazdar` odpovídá právě jednomu z pozdravů), tak pro oddělení dílčích subvýrazů, jak je ukázáno v tabulce 2.20.

testovaný řetězec	Monday Tuesday Friday
regulární výraz	`(Mon|Tues|Fri)day`
první výskyt	**Monday** Tuesday Friday
druhý výskyt	Monday **Tuesday** Friday
všechny výskyty	**Monday** **Tuesday** **Friday**

Tabulka 2.20: Použití znaku pipe jako alternativy

2.3.6 Speciální znaky

Speciální znaky popsané v předchozích odstavcích vždy plnily nějakou úlohu: `^` jako začátek řádku, `+` jako kvantifikátor nebo `.` jako zástupný symbol pro kterýkoliv znak. Může ale nastat situace, že potřebujeme některý z těchto znaků použít bez jeho speciální funkčnosti (tedy například `+` jako plus a ne jako kvantifikátor). V tomto případě se před speciální znak přidá zpětné lomítko `\` a jeho funkčnost je tímto potlačena.

Mezi tyto speciální znaky patří: `\`, `^`, `$`, `.`, `[`, `]`, `|`, `(`, `)`, `?`, `*`, `+`, `{`, `}`.

2.4 Příklady náročnějších konstrukcí

Následující příklady se týkají vhodných regulárních výrazů na testování zadaných řetězců pro e-mailovou adresu, URL a datum. Byly inspirovány rozborem na [15] a pro ověření jejich správnosti byl použit online nástroj [6].

2.4.1 E-mailová adresa

U jakékoli řetězce, u něhož chceme s pomocí regulárních výrazů testovat jeho správnost, je třeba znát, ze kterých částí se skládá a jaké znaky mohou a nesmí tyto části obsahovat. Rozdělení na části je vidět na obecném příkladu:

```
místní_část@doména
```

a lze prakticky demonstrovat na skutečné e-mailové adrese:

`xherou02@stud.fit.vutbr.cz`

`xherou02`	místní část
`@`	zavináč, oddělovač místní části a domény
`stud.fit.vutbr.cz`	doména (i se subdoménami, ty jsou odděleny tečkami)

doménu lze ještě rozdělit na tři části:

`stud.fit.vutbr`	doména a její subdomény
`.`	tečka, oddělovač subdomény a domény nejvyšší úrovně
`cz`	doména nejvyšší úrovně (např. info, org, uk, museum)

Povolené znaky pro jednotlivé části jsou tyto (pozn. následující příklad neodpovídá plně standardu RFC 822 [4], je uvažována běžná podoba e-mailové adresy):

- místní část: běžně tisknutelné znaky (např. -, ., ;, +) kromě zavináče
- doména: písmena a tečka jako oddělovač, u subdomén je možná i pomlčka , např. `www.tvorba-webu.cz`.

První podoba regulárního výrazu, který by testoval správnost zadané e-mailové adresy, by vypadala takto:

```
[-0-9a-zA-Z._&amp;]+@          (1)
[-0-9a-zA-Z]+.                 (2)
[a-zA-Z]{2,6}                  (3)
```

Tento regulární výraz má ale určité nedostatky. Prvním nedostatkem je, že místní část `(1)` musí začínat a končit alfanumerickým znakem. Druhým je skutečnost, že stávající regulární výraz umožňuje u doménové části `(2)` pouze jednu subdoménu a doménu nejvyšší úrovně (tedy například `vutbr.cz` ano, ale `stud.fit.vutbr.cz` ne), a to bude znamenat další úpravu na následující regulární výraz:

```
([0-9a-zA-Z]+[-._+&amp;])*[0-9a-zA-Z]+@   (1)
([-0-9a-zA-Z]+[.])+                       (2)
[a-zA-Z]{2,6}                             (3)
```

Druhý zápis už je lepší, a to v tom, že v místní části `(1)` se nejdřív vyskytne alespoň jedna číslice nebo písmeno, až poté ostatní znaky a před zavináčem se opět objeví alespoň jeden znak číslice nebo písmena. Doména `(2)` již může obsahovat i libovolný počet subdomén. Část `(3)` zůstává beze změny, protože vyhovuje požadavkům.

2.4.2 URL

Regulární výraz, který by pokryl většinu možných variant zadaných URL by mohl vypadat například takto:

```
(http|https|ftp)\://([a-zA-Z0-9\.\-]+(\:[a-zA-Z0-9\.&%\$\]+)*@)?
((25[0-5]|2[0-4][0-9]|[0-1]{1}[0-9]{2}|[1-9]{1}[0-9]{1}|[19])\.
 (25[0-5]|2[0-4][0-9]|[0-1]{1}[0-9]{2}|[1-9]{1}[0-9]{1}|[1-9]|0)\.
 (25[0-5]|2[0-4][0-9]|[0-1]{1}[0-9]{2}|[1-9]{1}[0-9]{1}|[1-9]|0)\.
 (25[0-5]|2[0-4][0-9]|[0-1]{1}[0-9]{2}|[1-9]{1}[0-9]{1}|[0-9])|
([a-zA-Z0-9\-]+\.)*[a-zA-Z0-9\-]+\.[a-zA-Z]{2,4})
(\:[0-9]+)?(/[^/][a-zA-Z0-9\.\,\?\'\\/\+&%\$#\=~_\-@]*)*
```

Je ale možné sestavit jednodušší výraz, který sice nepokryje téměř všechny možnosti (například zadat URL jako IP adresu), ale pro ověřování běžně používaných adres (např. `http://www.fit.vutbr.cz` či `http://seznam.cz/`) bude tento výraz stačit.

Nejdříve rozebereme části, ze kterých se URL skládá (opět nebude rozbor úplný podle standardu, jsou voleny jednodušší příklady URL, se kterými se běžný uživatel může setkat). Jsou to tyto části:

```
protokol:doména
```

přičemž protokol je volitelný a může nabývat hodnot: `http`, `https` nebo `ftp`, a doména může, ale nemusí, obsahovat část `www`. Dále za ni může být připojena i lokální cesta.

Regulární výraz kontrolující správnost zadané URL může vypadat takto:

```
((http\://|https\://|ftp\://)|(www.))+   (1)
(([-a-zA-Z0-9\.]+\.                      (2)
[a-zA-Z]{2,4}))                          (3)
(/[-a-zA-Z0-9\%:/_\?\.'~]*)?             (4)
```

Část `(1)` reprezentuje `http://`, `https://`, `ftp://` a `www`. Část `(2)` obsahuje pouze alfanumerické znaky a tečku s pomlčkou, jedná se o znaky povolené pro doménovou část. Ve `(3)` je zahrnuta doména nejvyšší úrovně, na kterou navazuje i nepovinná část `(4)`, a tou je lokální cesta. Po spojení všech čtyři částí vznikne regulární výraz pro kontrolu URL. Bude mu odpovídat například tento řetězec:

```
http://www.fit.vutbr.cz/FIT/history/
```

2.4.3 Datum

Zápis data má tři části – den, měsíc, rok. Tyto části mohou být zapsány v různých tvarech: `DD.MM.RRRR` nebo `DD/MM/RR`. Dále se mohou vyskytovat v jedno- nebo dvouznakových variantách (např. 3.9. a 03.09.). Nejdříve budou rozebrány regulární výrazy pro každou část a z nich se na závěr sestaví jeden ukázkový příklad.

Pro den, měsíc a rok mohou regulární výrazy vypadat následovně:

den:

`0[1-9]|[12][0-9]|3[01]`	dny 01-31
`0?[1-9]|[12][0-9]|3[01]`	dny (0)1-31
`0[1-9]|[12][0-9]|30`	dny 01-30
`0[1-9]|1[0-9]|2[0-8]`	dny 01-28
`0[1-9]|1[0-9]|2[0-9]`	dny 01-29

měsíc:

`0[1-9]|1[0-2]`	dny 01-31
`0?[1-9]|1[0-2]`	dny (0)1-31
`jan|feb|mar|apr|` `may|jun|jul|aug|` `sep|oct|nov|dec`	anglické zkratky pro měsíce

rok:

`[0-9]{2}`	rok 00-99
`20[0-9]{2}`	rok 2000-2099
`(19|20)[0-9]{2}`	rok 1900-2099

Obecný regulární výraz může vypadat takto:

```
den\.měsíc\.rok
den\. měsíc\. rok
den/měsíc/rok
rok-měsíc-den
```

kde se za den, měsíc a rok dosadí některý z výše definovaných regulárních výrazů. Celý regulární výraz může vypadat například takto:

```
(0?[1-9]|([12][0-9])|3[01])\.( )?(0?[1-9]|(1[0-2]))\.( )?(20[0-9]{2})
```

Zadání by odpovídaly tyto řetězce:

`20. 1. 2006`, `01. 2. 2099` či `01.12.2000`.

2.5 Regulární výrazy jako modely pro regulární jazyky

Na regulární výrazy lze nahlížet také jako na jeden ze dvou fundamentálních modelů pro regulární jazyky, druhým modelem jsou konečné automaty, o kterých lze nalézt více informací v [10]. Mezi těmito dvěma modely existuje určitý vztah, a to tento: Regulární výrazy popisují regulární jazyky a ty jsou přijímány konečnými automaty.

Regulární výrazy jsou výrazy s operátory konkatenace (.), sjednocení (+) a iterace(*). Existuje také tato definice:

- $\varnothing$ je regulární výraz značící prázdnou množinu (prázdný jazyk)
- ε je regulární výraz značící jazyk ε
- a, kde $a \in \Sigma$, je regulární výraz značící jazyk a
- nechť r a s jsou regulární výrazy značící po řadě jazyky L_r a L_s, potom:
  - $(r.s)$ je regulární výraz značící jazyk $L = L_r L_s$
  - $(r + s)$ je regulární výraz značící jazyk $L = L_r \cup L_s$
  - $(r*)$ je regulární výraz značící jazyk $L = L_r^*$

Existuje též přesný způsob, jak převést regulární výrazy na konečný stavový automat. O této problematice je možné se podrobně dočíst také v [10].

Kapitola 3

Návrh

V počáteční fázi projektu byla provedena krátká anketa mezi běžnými uživateli, na základě které byla sestavena didaktická koncepce výuky. Dalším krokem ve vývoji byl návrh implementační části aplikace, který je ilustrovám diagramem.

3.1 Anketa

Anketa, vytvořená pomocí nástroje Dokumenty Google, obsahovala 7 otázek a zodpovědělo ji 21 respondentů. Byla zaměřená především na zjištění uživatelských nároků, tedy co běžný uživatel očekává od výukového programu, a také jaké má obecné návyky při práci s aplikací. Otázky se týkaly barevnosti, ovládání programu, přihlašování a bezpečnosti vlastního účtu, dále základní očekávání u výukového programu obecně, styl výuky a prezentace látky. Z ankety byly vybrány tyto nejzásadnější poznatky:

- u otázky zjišťující barevný vkus uživatelů s výraznou převahou zvítězila možnost menšího počtu pastelových barev, výrazné barvy pouze ve výjimečných případech
- otázka zabývající se přihlašováním vydala najevo fakt, že uživatelé mají raději automatické přihlášení, než aby pokaždé vyplňovali nebo i jenom potvrzovali své přihlašovací údaje; zajímavý byl také návrh převzetí uživatele podle přihlášení do systému
- většina uživatelů si u výukového programu přeje obsáhlé vysvětlení látky s dostatečným množstvím testů na probrané učivo
- látka má být rozdělena do více lekcí, které jsou členěny na menší kapitoly, za každou kapitolou následuje malé cvičení, za každou lekcí souhrnný test všech dosud nabytých znalostí
- za velice důležitou považují uživatelé možnost prohlédnout si již vypracované testy nebo si je znovu zkusit; samozřejmostí je přehledné zobrazení výsledků a statistik
- poslední otázka se zabývala formou výuky, kde uživatelé nejvíce preferovali tradiční pojetí, či výuku a testy formou hry; za zmínku stojí, že ani jednou nebyla zvolena možnost animací a jiných vizuálních efektů

3.2 Didaktický návrh

3.2.1 Forma probírané látky

Respondenti v anketě nejvíce hlasovali pro tradiční pojetí, ale také pro výuku a testy formou hry. Snažila jsem se tedy najít kompromis mezi těmito dvěma variantami.

Celá látka bude rozdělena do lekcí a aby si uživatel mohl vše krok za krokem procvičit, každá lekce bude obsahovat několik kapitol – jejich počet bude záviset na šířce aktuálního tématu. V každé kapitole bude část teoretického vysvětlení, ukázka na příkladu a zakončena bude malým testem na procvičení. Lekce bude uzavřena souhrnným testem, ve kterém budou ověřeny znalosti z aktuální lekce, a bude navázáno i na znalosti nabyté v předchozích lekcích.

3.2.2 Rozdělení probírané látky

Je empiricky prověřeno, že nejlepší postup je od známého a jednoduššího k neznámému a složitějšímu. V prvních lekcích se uživatel seznámí se základy, jako jsou předdefinované skupiny znaků či tečka jako zástupný symbol kteréhokoli znaku, poté přejde na kvantifikátory, hranice, závorky a speciální znaky. Vybaven těmito základy již může uživatel pracovat se složitějšími konstrukcemi a příklady z praxe. Především rozpoznávání e-mailové adresy, url či zadaného data. Tyto příklady budou podrobně rozebrány v daném kontextu.

Tématické rozdělení lekcí a testů je následující:

1. lekce: Úvod
    - úvod
    - tečka jako zástupný symbol kteréhokoli znaku
2. lekce: Předdefinované skupiny znaků
    - `\d` – číslice
    - `\D` – nečíslice
    - `\w` – alfanumerické znaky
    - `\W` – nealfanumerické znaky
    - `\s` – prázdné znaky
    - `\s` – neprázdné znaky
3. lekce: Kvantifikátory
    - `*` – libovolný počet výskytů
    - `+` – alespoň jeden výskyt
    - `?` – nejvýše jeden výskyt
    - `{M}` – přesný počet výskytů
    - `{M,N}` – rozsah počtu výskytů
    - `{M,}` – minimální počet výskytů
    - `{,N}` – maximálně počet výskytů

4. lekce: Hranice
    - `^` – začátek řádku
    - `$` – konec řádku
    - `\A` – začátek víceřádkového řetězce
    - `\Z` – konec víceřádkového řetězce
    - `\b` – na hranici slova
    - `\B` – mimo hranici slova
5. lekce: Závorkové konstrukce, alternativy
    - `()` – kvantifikace
    - `[]` – výběr právě jednoho znaku
    - `[^]` – výběr mimo znaky
    - `|` – alternativy
    - `\` – speciální znaky
6. lekce: Praktické příklady
    - e-mailová adresa
    - URL
    - datum

3.2.3 Testy

Je známo, že teorie je základní a nezbytnou částí výuky, avšak teprve po absolvování praktického cvičení lze konstatovat, zda byla látka správně pochopena či nikoliv. Na testy proto bude kladen velký důraz. Jak bylo popsáno výše, testovat se budou moci uživatelé na konci každé kapitoly i lekce a tato část bude pojata formou hry.

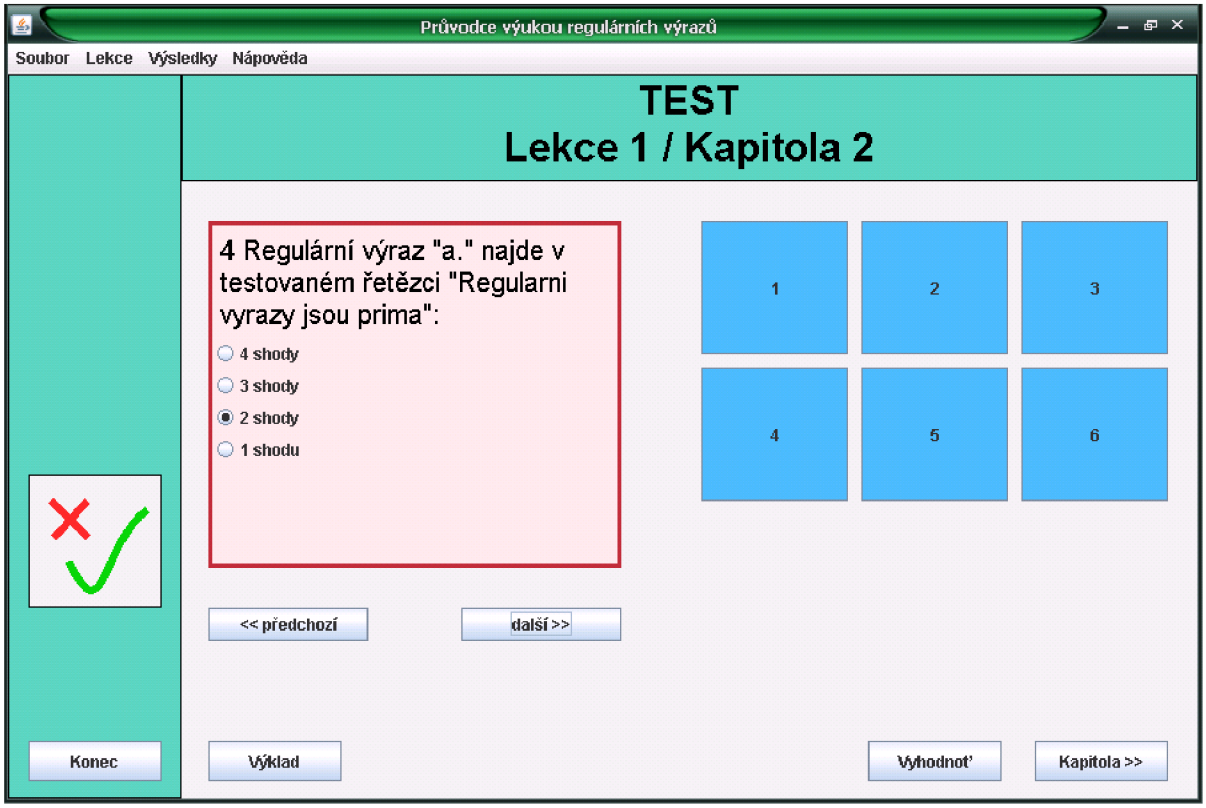
Zvolena byla tématická variace na pexeso. Otázky si lze představit jako obdélníkové karty, které jsou lícem dolů k pracovní ploše. Ve výchozím stavu má tato karta neutrální modrou barvu a po kliknutí na ni se „otočí“ a v levé části obrazovky se zobrazí otázka i s možnostmi. Po vyplnění testových otázek se všechny vyhodnotí a na základě správnosti odpovědi se karta s každou otázkou zabarví zeleně (otázka byla zodpovězena správně) nebo červeně (otázka byla zodpovězena špatně).

Testy nebudou ohraničeny žádným časovým odpočtem či limitem, uživatel tím tedy získá prostor si své odpovědi dostatečně promyslet. Během vyplňování testu zde bude možnost se jedním kliknutím vrátit na výklad teorie a zase zpět.

Ukázka uživatelského rozhraní testů je vidět na obrázku 3.1.

3.2.4 Zpětná vazba

Pod tímto pojmem je skryto hlavně přehledné zobrazení úspěšnosti - ve formě barevných grafů a procentuálního vyjádření. Vyznačení správné, případně špatné odpovědi při testování, bude plnit další pedagogickou funkci; uživatel tím získá plnohodnotnou zpětnou vazbu ke svým znalostem. Stojí za zamyšlení, zda by v principu byla lepší možnost změny odpovědi či znovuvyplnění celého testu, nebo poučení se z chyb a vygenerování testu nového. V tomto případě byla zvolena první možnost, tedy kdykoliv možná změna odpovědi.



Obrázek 3.1: Uživatelské rozhraní testů

Toto spolu s absencí časového limitu na vyplnění testů se může zdát velice mírným až nemotivujícím faktorem. Lze ale předpokládat, že uživatel, který bude pracovat s touto aplikací, má o téma regulárních výrazů zájem a jeho hlavním cílem není získat v testech stoprocentní úspěšnost za každou cenu, například vyplněním jednoho testu natřikrát, ale pochopení a zvládnutí tématu tak, aby s regulárními výrazy mohl pracovat efektivně a spolehlivě.

3.3 Návrh z pohledu uživatele

Po spuštění aplikace se nejdříve objeví uvítací stránka, na které si uživatel může vybrat ze dvou možností – buď začít celý výukový kurs od začátku (Start), nebo navázat tam, kde posledně skončil (Pokračuj). Po vybrání jedné z těchto dvou možností se zobrazí plocha s příslušnou lekcí a kapitolou (jejich číslo záleží na tom, jestli uživatel začíná nebo navazuje). Tato plocha se skládá, stejně jako plocha pro testy a výsledky, ze tří základních částí: levý navigační panel, horní titulek a hlavní plocha.

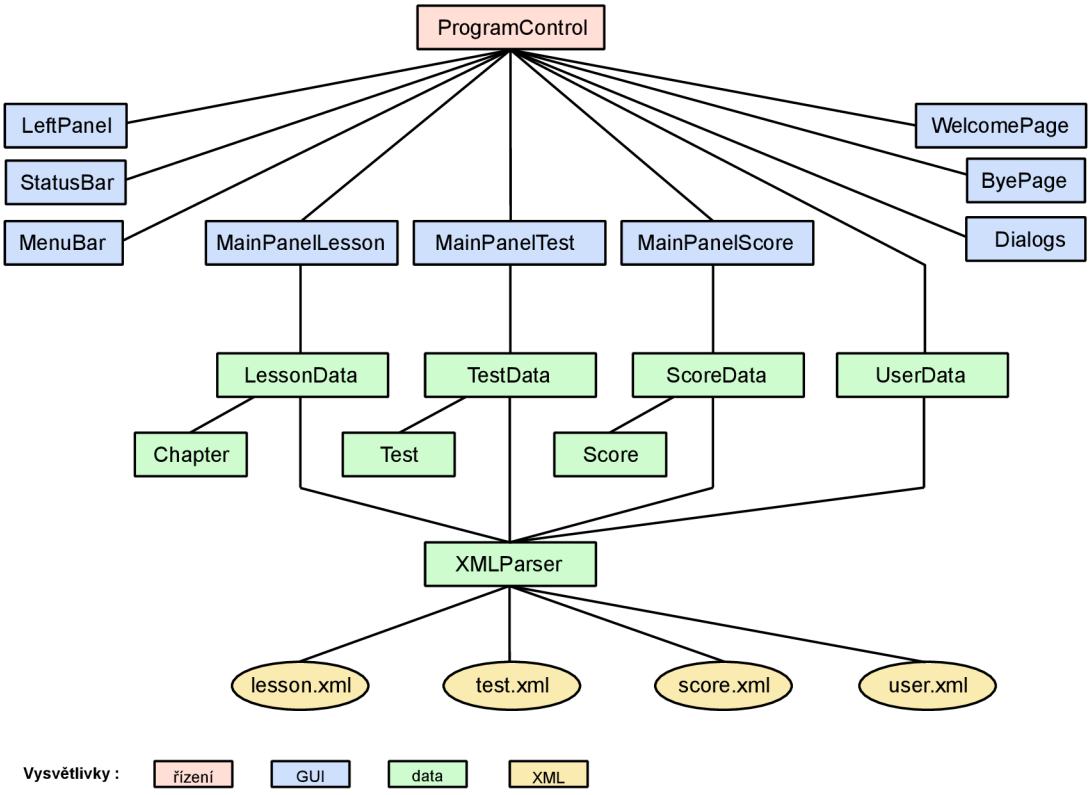
Levý navigační panel obsahuje tlačítko pro ukončení programu a dále ikonu pro snadnější přechody na výsledky. *Horní titulek* zobrazuje informaci o aktuální poloze uživatele v rámci programu, tedy jestli se právě zobrazuje lekce, test či výsledky testů.

Obsah *pracovní plochy* je různý pro každou ze tří tématických částí. V *lekci* je zde prostor pro výklad teorie a ukázku na příkladu a dále navigační odkazy pro přechod na test, nebo na další výklad. Pracovní plocha pro *testy* se skládá opět z navigačních odkazů

a tlačítek pro vyhodnocení, navrácení do výkladu, přechod na další výklad a pro pohyb po otázkách (předchozí a následující). Většinu prostoru zabírá jedna zobrazená otázka s odpověďmi ve formě radio tlačítek a menší „karty“ s dalšími otázkami. Vždy po kliknutí na kartu se příslušná otázka zobrazí. *Výsledky* jsou zobrazeny ve dvou velkých oblastech. První jsou výsledky pro každou lekci (číslo a název lekce, barevný graf a procenta úspěšnosti), druhá oblast se zobrazí po kliknutí na jednu z lekcí a obsahuje stejné vyjádření úspěšnosti kapitol dané lekce. Hlavní roli zde hrají tři barvy – zelená a červená instinktivně představují správné a špatné, modrá dosud nevyplněné.

3.4 Návrh z pohledu programátora

Programová část se dá rozdělit na čtyři základní oblasti, a to řízení logiky programu, realizace grafického uživatelského rozhraní, práce s daty a jejich uložení, jak je vidět na obrázku 3.2:



Obrázek 3.2: Diagram tříd

Řízení programu je umístěno ve třídě `ProgramControl`, kde se udržují veškeré informace nezbytné pro běh programu. Jedná se hlavně o čísla aktuální lekce a kapitoly, dále data uložená v paměti a také prvky GUI (panely). Tyto prvky jsou neustále střídány a také se mění jejich obsah (nejen textový). Třída `ProgramControl` se tedy stará o správné dodání a vykreslení potřebných komponent, hlavně nadpisu a obsahu hlavního panelu – lekcí, testů, výsledků. Důležitou část, která by se ovšem dala zařadit i do GUI, tvoří `MenuBar`, díky kterému se dá aplikací procházet i mimo optimální tématické pořadí.

GUI se dá rozdělit na tři velké části. První obsahuje třídy `LeftPanel` a `StatusBar`, jejichž instance existují po celou dobu běhu programu a mění se pouze jejich obsah (u instance `StatusBar`u text a u instance `LeftPanel`u jsou to tlačítka a odkazy). Druhá část obsahuje třídy `MainPanelLesson`, `MainPanelTest` a `MainPanelScore`, jejich instance jsou vytvořeny ihned při spuštění programu a v zobrazování se neustále střídají a opět mění svůj interní obsah. Třídy `WelcomePage`, `ByePage` a `Dialogs` jsou nezařaditelné mezi předchozí dvě části. Instance třídy `WelcomePage` se zobrazí pouze při spuštění programu a po přechodu na kterékoliv jiné zobrazení se již neobjeví, uvítací stránka by tak ztratila svůj smysl. Instance třídy `ByePage` se zobrazí pouze při ukončení aplikace. Třída `Dialogs` slouží ke specifickému účelu – zobrazuje dialogová okna.

Datová část aplikace je zcela oddělena od logiky a zobrazení programu. Data jsou uložena ve formátu XML v souborech `lesson.xml`, `test.xml`, `score.xml` a `user.xml`. Přístup k nim je umožněn prostřednictvím třídy `XMLParser` a ve třídách `LessonData`, `TestData` a `ScoreData` se ukládá jejich reprezentace do map. Ke zpřístupnění a uložení načtených dat do paměti využívají třídy `Chapter`, `Test` a `Score`, které reprezentují jednotlivé položky těchto map. Poslední je třída `UserData`, v níž jsou načtena uživatelská data při ukončení aplikace.

Kapitola 4

Implementace

Popis implementace vychází z programového návrhu, je tedy rozdělena na stejné tři části. Během celé tvorby programu byly jako hlavní zdroje informací použity [7] jako kompletní přehled tříd a jejich metod a [5] s ukázkami zdrojových kódů a jejich použití v praxi.

4.1 Řízení logiky programu

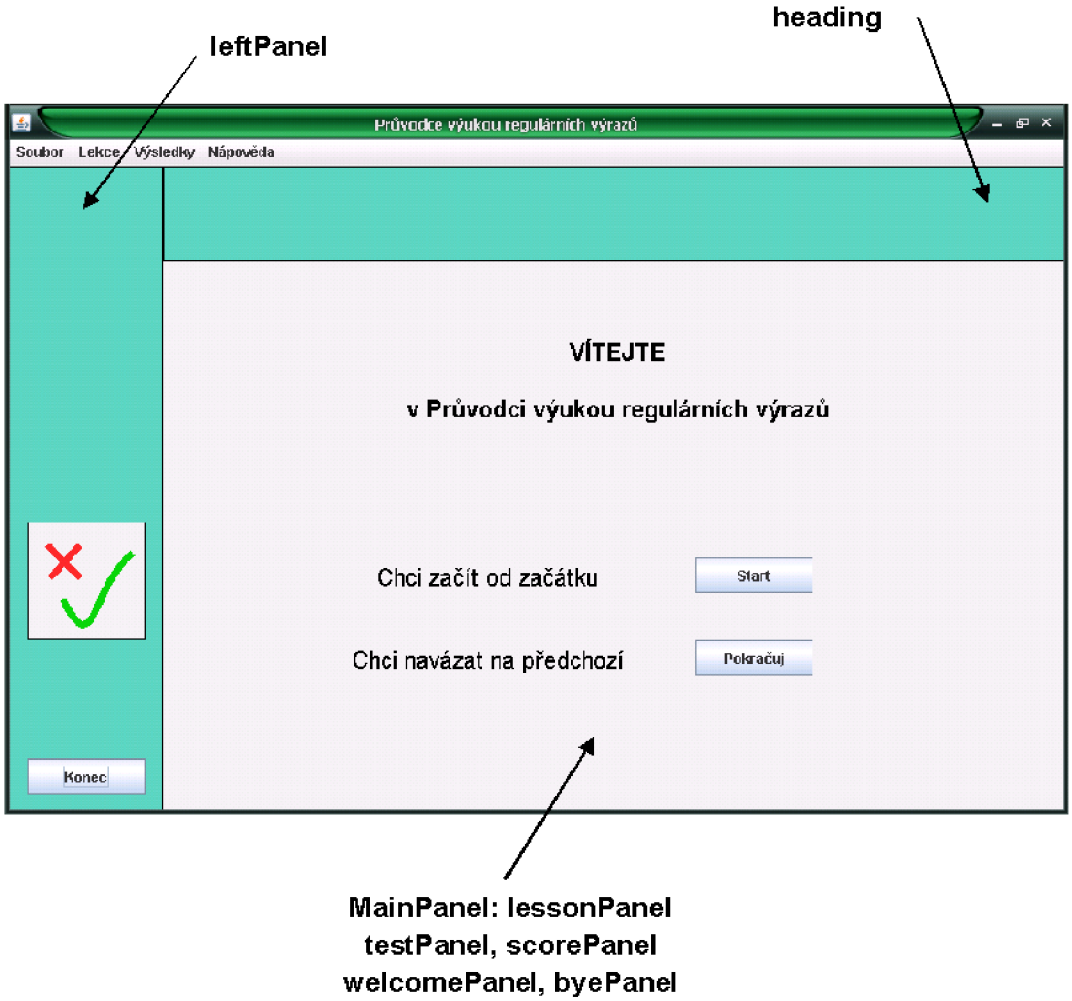
Veškerá logika je řízena jedinou třídou `ProgramControl`, v níž se nachází hlavní metoda `main()`. Po spuštění vytvoří instanci třídy `ProgramControl`, kde jsou v konstruktoru načtena data z formátu XML a těmto datům je přiřazena sémantika, dále je zde volána metoda `createAndShowGUI()` pro vytvoření a zobrazení grafického rozhraní.

Metoda `createAndShowGUI()` vytvoří menu, umístí rámec aplikace do středu obrazovky či na místo, kde bylo okno při posledním ukončení aplikace, a zajistí vytvoření a nastavení obsahu celého rámce metodou `createContentPane()`. Tento rámec se skládá z několika částí. Jednak jsou to pomocné panely `totalGUI` a `rightPanel`, které jsou instancí třídy `JPanel`, a také části finální - `heading`, `leftPanel`, `lessonPanel`, `testPanel`, `scorePanel`, `welcomePanel`. Tyto části jsou podrobně rozebrány v dalších kapitolách. Rozmístění jednotlivých částí je možné vidět na obrázku 4.1.

Třída `ProgramControl` obsahuje metody `runLesson()`, `runTest()` a `runScore()`, které jsou také součástí logického řízení programu. Při jejich zavolání vymění `rightPanel` svůj obsah a nastaví se příslušný text pro titulek i pro otázky (u testů) či výklad s příkladem (u lekcí).

Nejdříve bylo zobrazování jednotlivých panelů paměťově neúsporné. Při každém volání metod `runLesson()` a `runTest()` byly vždy nové instance třídy `JPanel` vytvořeny, použity a zahozeny při následujícím volání jedné z těchto metod. V začátcích vývoje byl tento způsob dostatečný, ale s postupným rozšiřováním aplikace to znamenalo neúspornost. Bylo tedy vytvořeno nové řešení, ve kterém se hned na začátku v metodě `createContentPane` vytvoří stálé objekty, u kterých se pouze mění jejich obsah.

Důležitá je také metoda `incChapNum()`, volá se při přechodu na další kapitolu, v případě poslední kapitoly na další lekci. Čísla `lessNum`, jako číslo lekce, a `chapNum`, jako číslo kapitoly, jsou získána odvozením z mapovací funkce pro ukládání dat do patřičných struktur, jež jsou popsány dále.



Obrázek 4.1: Rozmístění komponent v okně aplikace

4.2 Grafické uživatelské rozhraní

Vytváření grafického uživatelského rozhraní je umožněno třídami `MainPanelLesson`, `MainPanelTest`, `MainPanelScore`, `WelcomePage`, `ByePage`, `LeftPanel`, `StatusBar`, `Dialogs` a `MenuBar`.

První dvě jmenované třídy vytváří tlačítka pro pohyb v programu (např. přechod na další kapitolu, přechod na test) a plochy pro zobrazení textu (výklad a příklad, testy s možnostmi). U třídy `MainPanelLesson` je to panel pro vysvětlení látky a příklad, u třídy `MainPanelTest` je to panel pro zobrazení zvolené otázky s odpověďmi a tlačítka, která představují zbylé karty s otázkami. Třída `MainPanelScore` obsahuje dva základní panely pro zobrazení výsledků lekcí a kapitol.

Třída `WelcomePage` vytváří jednoduchou uvítací plochu, která obsahuje krátký text a dvě tlačítka – Start a Pokračuj. Naproti tomu třída `ByePage` vytváří plochu, která se zobrazí po absolvování celé výuky a obsahuje „rozloučení“ s uživatelem. Instance třídy `LeftPanel` slouží jako navigační boční lišta, která obsahuje tlačítko Konec či ikonu pro přechod na zobrazení výsledků. `Heading` jako instance třídy `StatusBar` je titulek umístěný

v horní části rámce, kde se metodou `setMessage()` nastavuje textový obsah. Třída `Dialogs` není rozsáhlá, obsahuje pouze metody pro zobrazování dialogových oken, například metoda `displayAboutBox` ukáže informační okno „O programu“ nebo metoda `displaySaveDialog` zobrazuje okno s otázkou na ukončení programu. Lišta s menu aplikace je vytvořena třídou `MenuBar`.

4.2.1 Třída MainPanelLesson

Tato třída rozšiřuje třídu `JPanel` a implementuje rozhraní `ActionListener`. V konstruktoru se nastaví velikost panelu a jeho layout, v tomto případě na null (rozmístění jednotlivých komponent je natolik specifické, že nebylo možno využít některý z nabízených), dále se s pomocí metody `createButton()` vytvoří jednotlivá tlačítka pro logický pohyb v programu a konečně metodou `setTextPane()` dva panely pro zobrazení výkladu a ukázkového příkladu. Texty obou položek se nastaví metodou `setTextAndExample`.

Jako reakce na stisk tlačítek jsou volány tyto metody (za použití knihoven `java.awt.event.ActionListener` a `java.awt.event.ActionEvent`): `goToNextChapter()`, ve které se zvýší číslo kapitoly (podle potřeby i číslo lekce) a ihned se nastaví text výkladu a ukázkového příkladu, `switchToTest()` přepne zobrazení na test dané kapitoly, metodu `goBackToTest()` je možno volat pouze v případě, vrátil-li se uživatel z testu na výklad a chce jít zpět na test.

Posledními jsou `get`-metody pro dvě tlačítka, toto souvisí s možností vrátit se z testu na výklad a zpět. Vždy je totiž zobrazeno pouze jedno z těchto tlačítek.

4.2.2 Třída MainPanelTest

Tato třída také rozšiřuje třídu `JPanel` a implementuje rozhraní `ActionListener`, stejně tak se v konstruktoru nastaví velikost panelu a jeho layout na null. Obsahuje ale více prvků než třída `MainPanelLesson`. Jedná se o panel pro zobrazení otázky s odpověďmi, který se skládá z textového panelu `questionText` a panelu pro radio tlačítka `buttonPanel`, a panel sdružující karty s otázkami realizovanými metodou `createTest()` jako pole tlačítek `cardTest` (počet karet závisí na tom, jestli jde o malý test za kapitolou, nebo o velký test za celou lekcí). Tyto karty se postupně skládají do panelu `quesButtonPanel`, který díky nastavenému GridLayoutu automaticky zachová stejné odstupy. Pro vytvoření skupiny radio tlačítek je nezbytná metoda `createButtonGroup()`, která spojí tlačítka do jedné skupiny, lze tedy vždy vybrat právě jedno tlačítko.

Reakcemi na stisk tlačítek jsou tyto metody: `backToSameChapter()` se vrátí zpět na výklad teorie a zároveň se provede odebrání a přidání tlačítka v `lessonPanelu`. Metoda `goToNextChapter()` provede s tlačítky to samé, jen se změní jejich pořadí, a přejde na další kapitolu (či celou lekci). Metody `goToNextQuestion()` a `goToPrevQuestion()` mají podobný význam, zobrazují následující či předchozí otázku. Jejich zakázání či povolení (aby se uživatel nedostal mimo vymezený prostor) zajišťuje metoda `displayQuestion()`. Ta je volána při každém požadavku na zobrazení otázky a dochází zde k nastavení textu otázky s odpověďmi (metodou `setQuestion()`) a volá v sobě také metodu `setButton()`, která nastaví jednu z odpovědí. To se hodí například ve chvíli, kdy uživatel vyplňuje test a není si jist svou odpovědí; jednou otázku zodpoví a tato odpověď je uložena, zkusí další otázky a po chvilce se k té původní vrátí, dříve označená odpověď tak bude znovu vybrána. Pokud otázku vyplňuje úplně poprvé, není označena žádná z odpovědí. Další metodou je `evaluate()`, která ihned po zvolení jakékoli odpovědi vyhodnotí její správnost a tento výsledek uloží do struktury v paměti. Poslední metodou je `setColors()`, která po skončení

testu na základě správnosti odpovědi označí zelenou a červenou barvou karty s otázkami. Při prvním prohlížení označí všechny karty stejnou neutrální modrou barvou.

Metoda `setFirtsQues()` vytváří vždy novou sadu karet s otázkami a nastavuje první otázku pro zobrazení testu. Je volána ve třídě `MainPanelLesson` v metodě `switchToTest()`.

4.2.3 Třída MainPanelScore

Třída rovněž rozšiřuje třídu `JPanel`, ovšem implementuje dvě rozhraní: `ActionListener` a `MouseInputListener`. V konstruktoru jsou vytvořeny dva základní panely – jeden pro zobrazení výsledků lekcí (`lesson`) a druhý pro výsledky kapitol (`chapter`). Poslední komponentou je tlačítko pro návrat do lekce, odkud bylo zobrazení výsledků zavoláno.

Zobrazování obsahu panelů pro výsledky lekcí i kapitol je velice podobné. Je zajištěno metodami `createRowsForLesson()` a `createRowsForChapter()`. Obě obsahují tři základní zobrazované části: pole nápisů `less-` a `chap-label[]`, ve kterém se vypisují čísla a jména lekcí a kapitol, dále panel `result`, který vyjadřuje úspěšnost graficky (ta je vykreslena metodou `setResult()`), a percentage jako instanci třídy `JLabel`, kde se zobrazuje procentuální úspěšnost. Důležitou metodou je zde `calculateRatio`, která onu úspěšnost vypočítá.

U obou panelů je nastaven GridLayout, což znamená, že se jednotlivé části skládají za sebou do předem nastaveného počtu sloupců a řádků. Původně byl nastaven BorderLayout, vyskytl se ale problém se zarovnáním do sloupců u grafického zobrazení úspěšnosti.

Jsou zde potřeba reakce na stisk tlačítka, ale pracuje se hlavně s reakcemi na pohyb myši a kliknutí na nápis (knihovna `java.awt.event.MouseEvent`). Aby se dosáhlo efektu měnící se barvy písma při přejetí myši po nápisu, jsou využity metody `mouseEntered()` a `mouseExited()`. Pro reakci na kliknutí na nápis (pouze pro lekce) je zde metoda `mouseClicked()`, která zjistí počet kapitol v dané lekci, na základě toho zobrazí nápisy, vypočítá poměr úspěšnosti pro každou kapitolu a nastaví barvu textu lekce, aby šlo jasně odlišit, o kterou lekci se zrovna jedná.

4.3 Data

Přístup k datům zajišťuje třída `XMLParser`, která s pomocí tříd `LessonData`, `TestData` a `ScoreData` naplní struktury `Chapter`, `Test` a `Score`. Třída `UserData` slouží k načtení a uložení uživatelských dat při ukončení aplikace.

Třída `Chapter` sestává ze dvou proměnných typu `String`, a to `lecture` a `example`, které představují text pro výklad látky a příklad. Třída `Test` obsahuje proměnnou text, která je typu `String` a představuje zadání otázky, a `answer`, což je pole typu `String` a představuje možné odpovědi na danou otázku. Třída `Score` má proměnnou `selected` typu `int`, která obsahuje číslo uživatelem zvolené odpovědi, a proměnnou `correct` typu `String`, jež nese informaci, zda byla odpověď zvolena správně. Všechny tyto třídy obsahují také po dvou metodách: `getLecture()` a `getExample()` pro třídu `Chapter`, `getText()` a `getAnswer()` pro třídu `Test` a `getSelected()` a `getCorrect()` pro třídu `Score`, které vrací obsahy výše zmíněných proměnných.

Třídy `LessonData`, `TestData`, `ScoreData` a `UserData` obsahují konstruktory, ve kterých se vytvoří prázdné datové struktury (ty jsou popsány dále), a také `add-` a `get-metody` pro ukládání a získávání obsahu. Jako pomocné metody sloužily `print-metody`, které byly hlavně v začátcích použity pro kontrolu správnosti načtení `.xml` souborů.

4.3.1 Uložení dat ve formátu XML

Veškerá data, která má aplikace k dispozici, jsou před zpracováním uložena ve formátu XML. Jsou rozdělena do několika souborů (`lesson.xml` pro lekce, `test.xml` pro testové otázky, `score.xml` pro ukládání a načítání výsledků testů) a nezbytného `DTD.dtd` [12]. Specifickým souborem je `user.xml`, kde jsou při ukončení aplikace uložena uživatelská data.

Způsob, kterým je rozčleněn soubor `lesson.xml`, lze popsat následovně. Kořenový element `L` obsahuje minimálně jeden element `Lesson`, reprezentující lekci, který má atributy `number` a `name`. V těchto atributech jsou uloženy informace o čísle lekce a jejím názvu. Dále element `Lesson` obsahuje alespoň jeden element `LChapter`, který představuje kapitolu, a jeho atributy jsou rovněž `number` a `name`, se stejným významem jako u lekce. Element `LChapter` obsahuje elementy `Text` a `Example`, což jsou řetězce s textem pro daný výklad.

Soubor `lesson.xml` tedy vypadá takto:

```
<L>
  <Lesson number=,,1‘‘ name=,,Úvod‘‘>
    <LChapter number=,,1‘‘ name=,,Úvod‘‘>
      <Text>text prvni kapitoly </Text>
      <Example>priklad prvni kapitoly </Example>
    </LChapter>
    <LChapter number=,,2‘‘ name=,,Zástupný symbol kteréhokoli znaku‘‘>
      <Text>text druhe kapitoly </Text>
      <Example>priklad druhe kapitoly </Example>
    </LChapter>
  </Lesson>
  <Lesson number=,,2‘‘ name=,,Předdefinované skupiny znaků‘‘>
    ...
  </Lesson>
  ...
</L>
```

Soubor `test.xml` má kořenový element `T` a je rozčleněn na lekce (`TLesson`), lekce na kapitoly (`TChapter`) a kapitoly na jednotlivé otázky (`TQuestion`). Všechny tyto elementy mají atribut number, který určuje jejich pořadové číslo. Element `TQuestion` obsahuje jeden element `Text` a čtyři elementy `Answer`, reprezentující text otázky a její možné odpovědi. Element `Answer` má také jeden nepovinný atribut `right`, který může nabývat hodnoty „ano“ a určuje, zda je odpověď správná nebo ne. Tento atribut se smí vyskytnout vždy pouze u jednoho elementu `Answer`.

Ukázka souboru `test.xml`:

```
<T>
  <TLesson number=,,1‘‘>
    <TChapter number =,,1‘‘>
      <TQuestion number=,,1‘‘>
        <Text>text lekce 1, kapitola 1, otazka 1</Text>
        <Answer>prvni odpoved</Answer>
        <Answer>druha odpoved</Answer>
        <Answer right=,,ano‘‘>treti odpoved, spravna</Answer>
        <Answer>ctvrta odpoved</Answer>
      </TQuestion>
      <TQuestion number=,,2‘‘>
        ...
      </TQuestion>
      ...
    </TChapter>
    <TChapter number =,,2‘‘>
      ...
    </TChapter>
    ...
  </TLesson>
  ...
</T>
```

Soubor `score.xml` obsahuje kořenový prvek `S` a je rozdělen na lekce reprezentované elementy `SLesson`, ty obsahují elementy `SChapter`. Oba nesou atribut `number` s číslem lekce či kapitoly. Poslední element `SQuestion` představuje jednotlivé zodpovězené otázky. Tento element nemá žádný obsah, pouze tři atributy: atribut `correct` nabývající hodnot „ano“ či „ne“, který nese informaci o tom, zda byla otázka uživatelem zodpovězena správně, dále číselný atribut `number` s údajem o pořadí otázky a poslední atribut `selected`, který říká, jaká odpověď na otázku byla uživatelem naposledy zvolena.

Práce se souborem `score.xml` se mírně liší od práce s předchozími dvěma soubory. Jednak při spuštění aplikace nemusí tento soubor existovat (například při úplně prvním spuštění daným uživatelem), a dále je možné a nutné tento soubor měnit, jelikož udržuje informace o aktuálních zodpovězených otázkách.

Ukázka souboru `score.xml`:

```
<S>
  <SLesson number=,,1‘‘>
    <SChapter number=,,1‘‘>
      <SQuestion correct=,,ano‘‘ number=,,1‘‘ selected=,,2‘‘/>
      <SQuestion correct=,,ne‘‘ number=,,2‘‘ selected=,,2‘‘/>
      ...
    </SChapter>
    <SChapter number=,,2‘‘>
      <SQuestion correct=,,ne‘‘ number=,,1‘‘ selected=,,3‘‘/>
      <SQuestion correct=,,ano‘‘ number=,,2‘‘ selected=,,1‘‘/>
      ...
    </SChapter>
    ...
  </SLesson>
  ...
</S>
```

Soubor `user.xml` obsahuje kořenový prvek `U` a každý element je jedinečný. Jsou zde totiž uloženy informace o souřadnicích (`x`, `y`) a rozměrech okna aplikace (`width`, `height`) a také o čísle dosažené lekce a kapitoly (`lessNum`, `chapNum`). Všechny tyto elementy mají atribut `value`, který nese požadovanou informaci.

Ukázka souboru `user.xml`:

```
<U>
  <height value=,,600‘‘/>
  <lessNum value=,,3‘‘/>
  <width value=,,900‘‘/>
  <chapNum value=,,3‘‘/>
  <y value=,,96‘‘/>
  <x value=,,147‘‘/>
</U>
```

4.3.2 Získávání a ukládání dat

Pro manipulaci s daty a k jejich zpřístupnění slouží třída `XMLParser`. V metodě `createDOM()` se XML soubory otevřou pomocí rozhraní DOM a uloží se do paměti [8]. Metody `readLessonXML()`, `readTestXML()`, `readScoreXML()` a `readUserXML()` tyto stromové struktury zpracují [3], [1] a spolu s `add-metodami` tříd `LessonData`, `TestData` a `ScoreData` je uloží jako kolekce typu `Map`, a to s využitím třídy `TreeMap`, pro `userData` je použita `HashMap` (knihovna `java.util`). Důvod k použití právě `TreeMap` je ten, že prvky ukládá vzestupně podle hodnoty klíče, který je vypočítán pomocí následující mapovací funkce:

```
key = 100*less + chap                  (v LessonData)
key = 10000*less + 100*chap + ques     (v TestData, ScoreData)
```

kde `less`, `chap` a `ques` určují číslo lekce, kapitoly a otázky.

Tímto způsobem jsou pro soubor `lesson.xml` vytvořeny tři kolekce stejného typu `Map`. Jedna slouží k ukládání obsahu lekcí, tedy výklady a ukázkové příklady, druhá je pro uložení názvů lekcí a kapitol, třetí udržuje informace o počtu lekcí a kapitol. Pro soubor `test.xml` jsou vytvořeny dvě takové kolekce. První má uložená data pro testy (otázky a odpovědi na ně) a druhá má pouze informace o otázce a její správné odpovědi (to slouží k pozdějšímu vyhodnocování testů). Pro soubor `score.xml` stačí jedna taková kolekce, a to pro uložení zvolené odpovědi a její správnosti, to vše pro každou testovou otázku. Stejně tak pro soubor `user.xml` je potřeba jedna kolekce.

Do souborů `score.xml` a `user.xml` lze také zapisovat, to umožňují metody `writeScoreXML` a `writeUserXML`.

4.4 Vývoj

Vývoj celého projektu by se dal rozdělit do tří částí – hrubá podoba, přístup k datům a přesná podoba. Rozdělení není striktně podle časové osy, protože například tvorba GUI byla souběžná s tvorbou logiky aplikace a také pracemi s daty. Má spíš za úkol neplést všechny kroky dohromady a popsat vývoj větších celků.

4.4.1 Hrubá podoba

První etapa, pracovně nazvaná „řešení existuje“, byla hlavně o hlubším seznámení se s jazykem Java a experimenty s knihovnou Swing sloužící k návrhu a práci s grafickým uživatelským rozhraním. Jako výborný průvodce tvorbou GUI v Javě mi posloužil The GUIdebook [9], díky kterému jsem pochopila základní principy tvorby GUI v Javě.

Nejdříve byly úspěšně poskládány jednotlivé panely podle základního návrhu, byla přidána tlačítka a reakce na ně, např. změna barvy pozadí či přepínání mezi lekcí a testem. Dalším krokem bylo vytvoření textových ploch pro výklad a ukázkový příklad v lekci a s tím související způsob úpravy textu. Volba mezi čistým HTML a CSS nakonec padla na HTML, jelikož stačí jednoduché formátování písma, např. velikost, italika, centrování na střed a podobně.

Vývoj pokračoval vytvořením logiky řízení, což znamenalo přidání globálních proměnných udržujících číslo aktuální lekce a kapitoly. Protože řešení právě pomocí globálních proměnných není příliš čisté, byly brzo tyto proměnné nahrazeny get-, set- a inc-metodami. Vzniklo také menu a jeho první položka, která umožňovala ukončení aplikace.

Lekce byly z hlediska vytvoření předběžné grafické podoby hotové, mohlo se tedy přejít na testy. Zde bylo potřeba přidat tlačítka pro návrat do výkladu nebo pokračování na další kapitolu, dále prostor pro otázku i s možnostmi a nakonec tlačítka představující karty s otázkami.

Dalším krokem bylo přidání uvítací obrazovky, doplnění menu o další položky (například výběr lekce ke spuštění), zpřehlednění zdrojového kódu a hlavně vylepšení práce s pamětí. Nebyly už stále vytvářeny nové panely a ihned zahazovány, ale hned při spuštění programu se vytvořily od každého jeden a pouze se měnily jejich obsahy (texty, titulek). U testů

se dodělala funkčnost jednotlivých karet s otázkami, dále se doplnil text aktuální otázky a jejích odpovědí a v závěru bylo vyřešeno i vyhodnocování testů a ukládání výsledků.

Tyto bylo potřeba zobrazit, vznikl tedy grafický prostor pro výsledky. V něm se nejprve zobrazily nápisy a barevný graf s procenty, to vše se staticky zadanou úspěšností. Poté se vyřešilo počítání této úspěšnosti a také doplnění nápisů o přesný název lekce a kapitoly. Dále byly implementovány reakce na pohyb myši a kliknutí, například změna barvy a zobrazení úspěšnosti jednotlivých kapitol dané lekce. Původně byl pro poskládání jednotlivých komponent použit BorderLayout se svým zarovnáním WEST, CENTER a EAST, to ale způsobilo problémy s různou polohou barevného grafu. Jako lepší se tedy ukázal GridLayout, který komponenty seřadil do přehledných sloupců a řádků.

Aplikace byla opět o něco chytřejší, bylo tedy na čase dodělat položky menu, které by umožnily rychlejší orientaci a přesouvání v programu, například zobrazení výsledků, přechod na určitou lekci či zobrazení informací o programu.

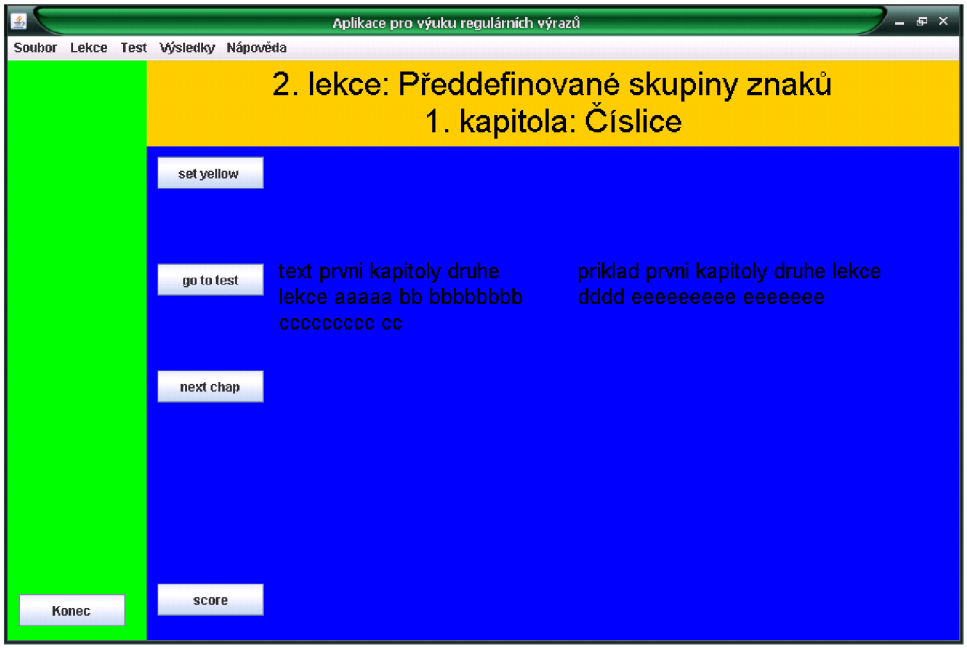
Pracovní název „řešení existuje“ jsem používala hlavně proto, že aplikace již uměla zobrazit různá textová pole, formátovaný text či reagovat na tlačítka, ale stále nebylo jasné a ani podstatné, jaké přesné rozměry mají textová pole mít, jaká bude jejich přesná lokace či optimální velikost písma.

4.4.2 Přístup k datům a jejich zpracování

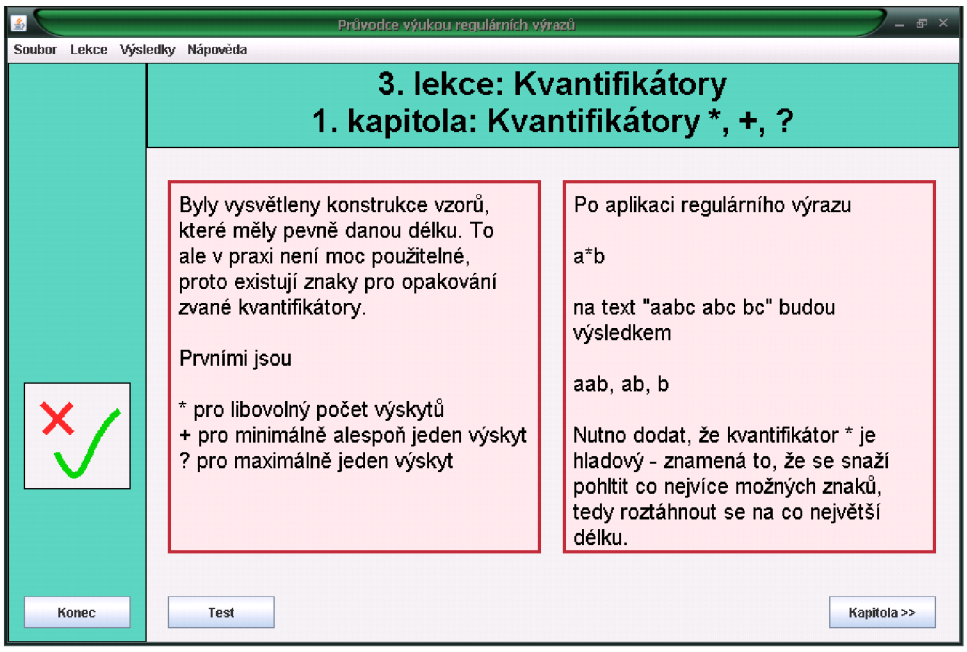
Prvním krokem, který se týkal práce s daty, byl návrh DTD a vytvoření samotných dat. Nejdříve ve zkušební podobě, která stačila k vytvoření a otestování algoritmů pro přístup k nim, a až ke konci dostala data smysluplnou a faktickou podobu. Následovalo otevření souborů pomocí rozhraní DOM a jejich uložení do paměti. Další fází bylo procházení stromové struktury uložené v paměti a její převod na kolekce typu Map. V části vyhodnocování testů bylo také potřeba vyřešit ukládání výsledků do XML souboru, aby bylo možné i zpětně zobrazit statistiky a výsledky uživatelem vyplněných testů. Pro uživatelské nastavení bylo potřeba zjistit údaje o aktuální velikosti a poloze okna aplikace a také poslední absolvované lekce a kapitoly, aby bylo možné při dalším spuštění programu v tomto místě navázat. Finální částí bylo naplnění XML souborů pro lekce a testy reálnými daty.

4.4.3 Úpravy GUI do konečné podoby

Úpravy GUI do finální podoby spočívalo hlavně v přeskládání komponent tak, aby orientace v okně byla rychlá a jasná na první pohled. K tomu přispělo hlavně uspořádání tlačítek a také barevné odlišení částí textu (např. výklad a ukázkový příklad v lekci). Ukázka vzhledu GUI před a po úpravě je vidět na obrázcích 4.2 a 4.3.



Obrázek 4.2: Ukázka GUI lekce před úpravou



Obrázek 4.3: Ukázka GUI lekce po úpravě

Kapitola 5

Shrnutí

5.1 Zvolené a použité nástroje

Jako optimální programovací jazyk byla zvolena Java, a to nejen kvůli objektově orientovanému přístupu, který je vhodný pro zpracování tématu projektu, ale také proto, že disponuje knihovnou Swing, což je mocný nástroj pro tvorbu grafických uživatelských rozhraní. Jako vývojové prostředí bylo vybráno prostředí NetBeans.

5.2 Možná rozšíření

Výsledná aplikace sice již dokáže předat uživateli základy práce s regulárními výrazy, její podoba a přístup k výuce ale nemusí být konečné.

Bylo by pozitivní, kdyby se databáze testových otázek rozšířila a existovala možnost náhodného generování testů, uživatel by si je mohl zopakovat s jinými otázkami, čímž by si látku mohl více procvičit. S tím souvisí i rozšíření testů na praktické otázky, tedy nejenom volba z daných možností, ale tvůrčí otázky typu „Vytvořte regulární výraz, který najde shodu s daným řetězcem ...“. Pak by bylo třeba vyřešit kontrolu a vyhodnocování takto zadaných úkolů.

Pro průběžnou možnost opakování znalostí je vhodné mít výklad látky po ruce i mimo počítač, v papírové formě. Látka by byla shrnuta do přehledné, heslovité tabulky (s pravidly i příklady), u které by uživatel nadefinoval, která témata má obsahovat, a tuto tabulku by si mohl vytisknout.

Jak již bylo zmíněno, při výuce je důležitá vlastní praxe. Aplikace by proto mohla nabízet možnost kontroly vytvořených regulárních výrazů a jejich aplikování na zadaný řetězec. Inspiraci lze najít např. u webové služby [6]. Zde uživatel zadá regulární výraz, úsek textu jako nahrazení nalezené pasáže a až deset řetězců, na které bude regulární výraz aplikován. Po vyhodnocení se uživateli zobrazí podrobný rozbor celého řetězce.

Další rozšíření by se týkalo zpřístupnění aplikace více uživatelům naráz. Každý by měl vytvořen svůj účet, byly by zaznamenány výsledky i poslední probrané téma v kurzu, na které by se dalo navázat.

Protože regulární výrazy jsou používány více jazyky a nástroji, mohou se vyskytovat různé mutace pro konkrétní případy. Dalším možným rozšířením je tedy buď nastavení celé aplikace či testů, aby se orientovaly pouze na jednu oblast, nebo podrobnější výklad látky.

Kapitola 6

Závěr

Byl vytvořen výukový program pro regulární výrazy, díky kterému uživatel projde všemi důležitými částmi procesu učení: jedná se o vysvětlení teorie, ukázku na příkladu, procvičení znalostí v testu a také zjištění, do jaké míry bylo pochopení látky úspěšné. Tato látka je navíc tématicky rozdělena na lekce a kapitoly, jež jsou seřazeny s ohledem na postupnost učiva. To vše bylo inspirováno anketou provedenou mezi běžnými uživateli a aplikace by tedy neměla být překážkou, ale pomocníkem k pochopení práce s regulárními výrazy.

Program sice nepostihuje úplně všechny možnosti jejich zápisu či využití, ale měl by hlavně položit pevné základy, na které se dá už snadno navázat. Hlavní výhodou zůstávají informace seskupené pohromadě a procvičení na příkladech s kontrolou správnosti. Program lze samozřejmě rozšířit o výše zmíněné prvky a poté by se mohl stát aplikací s opravdu širokým záběrem.

Během projektu jsem se seznámila s využitím formátu XML v plném nasazení a zaujala mě především tvorba GUI v Javě s pomocí knihovny Swing. Ačkoli se názory na tuto knihovnu výrazně liší a při hledání informací na internetu jsem narazila na některé negativní ohlasy, osobně se mi práce s touto knihovnou i přes krušné začátky (nebo právě díky nim) líbila a přišla mi zajímavá. Také kompletní vývoj projektu, počínaje nastudováním teorie, přes anketu a návrh, až po implementaci a zmíněné GUI, byl zajímavou zkušeností a někdy i zábavou.

Literatura

[1] Developerlife - Tutorials - XML and Java Tutorial, Part 1 [online]. `http://developerlife.com/tutorials/?p=25#howtocreateadomobject`, 1998 [cit. 2009-04-18].

[2] Zvon [online]. `http://www.zvon.org/other/PerlTutorial/Output_cze/contents.html`, 2006 [cit. 2009-03-19].

[3] Java DOM Tutorial [online]. `http://http://www.roseindia.net/xml/dom/`, 2007 [cit. 2009-04-18].

[4] Standard for the format of arpa internet text messages [online]. `http://www.faqs.org/rfcs/rfc822.html`, 2008 [cit. 2009-04-28].

[5] Graphical User Interfaces (The Java Tutorials) [online]. `http://www.j2ee.me/docs/books/tutorial/ui/index.html`, 2008 [cit. 2009-05-13].

[6] RegEx: online regular expression testing [online]. `http://www.fileformat.info/tool/regex.htm`, 2009-01-28 [cit. 2009-03-19].

[7] Overview (Java 2 Platform SE 5.0) [online]. `http://java.sun.com/j2se/1.5.0/docs/api/overview-summary.html`, 2009 [cit. 2009-05-13].

[8] Brůha, L.: *Java Hotová řešení*. Computer Press, 2006, iSBN 80-251-0072-3.

[9] Davidson, S.: The Guidebook - Swing Tutorials [online]. `http://www.macs.hw.ac.uk/guidebook/?`, 2009-04-17 [cit. 2009-04-05].

[10] Dostál, H.: Teorie konečných automatů, regulárních gramatik, jazyků a výrazů. `http://iris.uhk.cz/tein/teorie/konecnyAutomat.html`, 2008 [cit. 2009-05-13].

[11] Herold, H.: *AWK a SED, příručka pro dávkové zpracování textu*. Computer Press, 2004, iSBN 80-251-0309-9.

[12] Kolář, D.: XML aneb nový formát pro nové tisíciletí [online]. `http://www.builder.cz/art/html/xml_uvod.html`, 2001-01-17 [cit. 2009-03-17].

[13] Kukla, M.: *Využití regulárních výrazů při opravách textu, úpravách a generování souborů*. Diplomová práce, Univerzita Pardubice, 2007.

[14] Kysela, M.: *Perl - kapesní průvodce programátora*. Grada, 2005, iSBN 80-247-1170-2.

[15] Pecka, M.: Regulární výrazy [online]. http://www.regularnivyrazy.info/, 2008 [cit. 2009-03-19].

[16] Satrapa, P.: Regulární výrazy [online]. http://i.iinfo.cz/r2/k/Regularni_vyrazy.pdf, 2000 [cit. 2009-03-15], vytvořeno pro on-line magazín root, www.root.cz.

[17] Vacek, R.: Lipava [online]. http://www.lipava.cz/edu/vyuka-linux/fh-regexp.html, 2008-01-02 [cit. 2009-05-13].

Dodatek A

Obsah CD

zdrojové kódy programu, návod jak přeložit a spustit program v prostředí NetBeans, instalační soubory k tomu potřebné, programová dokumentace

Dodatek B

Plakát