TECHNICKÁ UNIVERZITA V KOŠICIACH

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Automatizovaná kontrola postupu študenta v prípadových štúdiách pre výučbu objektového programovania

2011 Juraj MAZÁK

TECHNICKÁ UNIVERZITA V KOŠICIACH

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

AUTOMATIZOVANÁ KONTROLA POSTUPU ŠTUDENTA V PRÍPADOVÝCH ŠTÚDIÁCH PRE VÝUČBU OBJEKTOVÉHO PROGRAMOVANIA

Diplomová práca

Študijný program:	Informatika
Študijný odbor:	9.2.1 Informatika
Školiace pracovisko:	Katedra počítačov a informatiky (KPI)
Školiteľ:	doc. Ing. Jaroslav Porubän, PhD.
Konzultant:	doc. Ing. Jaroslav Porubän, PhD.

Košice 2011 **Juraj MAZÁK**

Analytický list

Autor:	Juraj Mazák
Názov práce:	Automatizovaná kontrola postupu študenta v prípadových štúdiách pre výučbu objektového programovania
Podnázov práce:	
Jazyk práce:	slovenský
Typ práce:	Diplomová práca
Počet strán:	85
Akademický titul:	Inžinier
Univerzita:	Technická univerzita v Košiciach
Fakulta:	Fakulta elektrotechniky a informatiky (FEI)
Katedra:	Katedra počítačov a informatiky (KPI)
Študijný odbor:	9.2.1 Informatika
Študijný program:	Informatika
Mesto:	Košice
Vedúci práce:	doc. Ing. Jaroslav Porubän, PhD.
Konzultanti práce:	doc. Ing. Jaroslav Porubän, PhD.
Dátum odovzdania:	29. apríl 2011
Dátum obhajoby:	24. máj 2011
Kľúčové slová:	anotácia, reflexia, nástroj podpory výučby, výučba objektovo-orientovaného programovania, automatizácia, prírastková štúdia
Kategória konspekt:	Výpočtová technika; Programové vybavenie. Programové prostriedky. Softvér. Programovanie. Dáta. Databázy
Citovanie práce:	Mazák, Juraj: Automatizovaná kontrola postupu študenta v prípadových štúdiách pre výučbu objektového programovania. Diplomová práca. Košice: Technická univerzita v Košiciach, Fakulta elektrotechniky a informatiky, 2011. 85 s.
Názov práce v AJ:	Automated Monitoring of Students Approach in Case Study for Teaching of Object-oriented Programming
Podnázov práce v AJ:	
Kľúčové slová v AJ:	anotation, reflexion, education support tool, object-oriented programming education, automation, incremental study

Abstrakt v SJ

Diplomová práca pojednáva o podpore výučby objektového programovania v kontexte Technickej univerzity v Košiciach. Nástrojová podpora výučby má umožniť automatizovanú kontrolu postupu študenta v prípadových štúdiách používaných vo výučbe objektovo-orientovaného programovania. Má spĺňať podmienky v oblastiach štandardizovaných rozhraní vstupov a výstupov, efektivity práce, možnosti integrovania s existujúcimi systémami a univerzálnosti v nasadení. Kontrola postupu študenta má priebežne počas semestra poskytovať spätnú väzbu o jeho postupe študentovi samotnému i vyučujúcemu. Štúdia, podľa ktorej študent v krokoch po týždňoch postupuje, je modulárne organizovaná usporiadaná množina úloh, ktoré ho prevedú od základných vlastností a konštrukcií objektovo-orientovaného jazyka až k zložitejším prvkom aplikácie ako sú používateľské rozhranie alebo databázy. Cieľom tejto práce je teda vytvorenie sady nástrojov, ktoré umožnia testovanie postupu pri splnení definovaných vlastností a podmienok. Úvod práce opisuje teoretické odporúčania vychádzajúce z dlhoročnej tradície kurikula vydávaného odborníkmi z priemyslu a praktické skúsenosti vysokoškolských učiteľov s výučbou objektovo-orientovaného programovania vo forme štúdií. Obsahuje aj charakteristiku štúdie Minesweeper používanej vo výučbe na Technickej univerzite v Košiciach na predmete Technológie Java a pedagogické východiská, ktoré podporujú myšlienku automatizovanej kontroly a nástrojovej podpory vo výučbe. Hlavná časť práce je venovaná analýze, syntéze a budúcemu použitiu nástrojov podľa opísaných cieľov. V závere práce je opis experimentov a zhodnotenie dosiahnutých výsledkov.

Abstrakt v AJ

The thesis deals with the promotion of teaching object-oriented programming in context of the Technical University in Kosice. A tool support of teaching ought to enable automated control of students‘ progression in case studies used in teaching of object-oriented programming. The tool support of teaching ought to satisfy the conditions in the areas of standardized interfaces of inputs and outputs, work efficiency, abilities to be integrated with existing systems and versatility in deployment. Control of the students‘ procedures should continuously throughout the semester provide feedback on their progression for teacher and the students themselves. A study, according to which each student moves forward in steps week after week, is a modularly organized ordered set of tasks that leads the students through from the fundamental features and structures of object-oriented language to more complex components of applications such as a user interface or a database. Therefore, the aim of this work is to create a set of tools that enables to test the procedure compliance with defined properties and conditions. In the beginning of the thesis we describe theoretical recommendations based on long-standing tradition of curriculum published by industry experts and practical experience of university teachers with teaching object-oriented programming in the form of studies. The thesis also includes characteristics of a study of Minesweeper used in teaching at the Technical University in Košice in the subject called Java Technology and pedagogical assumptions that support the idea of automated control and a tool support in teaching. The main part is devoted to analysis, synthesis and future use of the instruments according to the objectives described. In conclusion of this work, there is a description of the experiments and assessing of the reached achievements

TECHNICKÁ UNIVERZITA V KOŠICIACH

Fakulta elektrotechniky a informatiky

Katedra počítačov a informatiky

ZADANIE DIPLOMOVEJ PRÁCE

Študijný odbor: **9.2.1 Informatika**

Študijný program: **Informatika**

Názov práce:

Automatizovaná kontrola postupu študenta v prípadových štúdiách pre výučbu objektového programovania

Automated Monitoring of Students Approach in Case Study for Teaching of Object-oriented Programming

Študent (tituly, meno, priezvisko): **Bc. Juraj Mazák**

Školiteľ (tituly, meno, priezvisko): **doc. Ing. Jaroslav Porubän, PhD.**

Školiace pracovisko: **Katedra počítačov a informatiky**

Konzultant práce (tituly, meno, priezvisko):

Pracovisko konzultanta:

Pokyny na vypracovanie diplomovej práce:

1. **Teoreticky analyzovať problematiku výučby objektovo-orientovaného programovania na vysokých školách s dôrazom na nástrojovú podporu.**
2. **Navrhnúť integrovanú sadu nástrojov umožňujúcu automatizovanú kontrolu postupu študenta v prípadových štúdiách pre výučbu objektového programovania.**
3. **Implementovať navrhnutú sadu nástrojov.**
4. **Experimentálne overiť prínos a použiteľnosť navrhnutých nástrojov a vyhodnotiť dosiahnuté výsledky.**
5. **Vypracovať dokumentáciu podľa pokynov vedúceho práce.**

Jazyk, v ktorom sa práca vypracuje: slovenský

Termín pre odovzdanie práce: 29.04.2011

Dátum zadania diplomovej práce: 31.10.2010

prof. Ing. Ján Kollár, CSc.
vedúci garantujúceho pracoviska

prof. Ing. Liberios Vokorokos, PhD.
dekan fakulty

Čestné vyhlásenie

Vyhlasujem, že som celú diplomovú prácu vypracoval samostatne s použitím uvedenej odbornej literatúry.

Košice, 29. apríl 2011

.....

vlastnoručný podpis

Poďakovanie

Ďakujem svojmu konzultantovi a zároveň vedúcemu práce doc. Ing. Jaroslavovi Porubänovi, PhD. za odbornú pomoc a rady pri usmernení vo vypracovaní aplikácie a textu práce. Ďakujem aj mojej manželke Lenke za obetavú starostlivosť a pochopenie počas obdobia jej prípravy.

Predhovor

Kvalitné vzdelávanie je kľúčovou úlohou procesu výučby na všetkých stupňoch škôl. Prepojenie teórie a praxe, problémové a projektové vyučovanie, kreativita a individuálny prístup môžu výrazne napomôcť snahe o efektívne dosahovanie výsledkov výučby.

Inšpirácia nástrojovej podpory výučby začala pri mojej bakalárskej práci, na ktorú táto diplomová práca nadväzuje. Vízia odbúrania rutinnej kontroly, možnosti venovania sa talentovaným a snaživým študentom a včasného odhalenie kritických chýb sa na základe prechádzajúceho výskumu ukázala byť reálna a tak je rozšírenie pôsobnosti prirodzeným vyústením snahy o zlepšenie podmienok výučby na univerzite. Z aktuálnych potrieb a možností sa automatizovaná kontrola postupu študenta v prípadových štúdiách výučby programovania ukázala ako vhodná cesta pre diplomovú prácu.

Pod vedením skúseného pedagóga a odborníka na danú oblasť som sa rozhodol prepojiť moje doplňujúce pedagogické štúdium s pedagogike blízkou témou podpory výučby a zvolil som si programovanie integrovanej sady nástrojov, ktorá má za ambíciu napomôcť ďalším generáciám študentov i učiteľov pri vyučovacom procese.

Obsah

Zoznam obrázkov **12**

Zoznam symbolov a skratiek **13**

Slovník termínov **14**

Úvod **15**

1 Formulácia úlohy **17**

2 Analýza výučby programovania a nástrojovej podpory **18**

2.1 Kurikulum „Computing Curricula 2001 Computer Science“ 18

2.1.1 História 19

2.1.2 Odporúčania základov a rozšírení 19

2.1.3 Stratégie realizácie 21

2.1.4 Miesto programovania vo výučbe 21

2.1.5 Prístup „programovanie ako prvé“ – výhody a nevýhody 22

2.1.6 Objektovo-orientované programovanie 24

2.2 Výučba na univerzite v Pretórii, Južná Afrika 25

2.3 Výučba na univerzite v Sydney, Austrália 29

2.3.1 Východiská a okolnosti 29

2.3.2 Predstavenie myšlienky 30

2.3.3 Systém predmetu a hodnotenia 31

2.3.4 Predstavenie sveta 32

2.3.5 Skúsenosti z realizácie štúdie 34

2.4 Výučba programovacích jazykov na TUKE 36

2.4.1 Chronológia programovacích jazykov v predmetoch 36

2.4.2 Porovnanie s odporúčaniami CC2001 – prvý semester 36

2.4.3 Porovnanie s odporúčaniami CC2001 – druhý semester 37

2.4.4 Porovnanie s odporúčaniami CC2001 – tretí semester 37

2.4.5 Porovnanie s odporúčaniami CC2001 – ďalšie semestre 39

2.5 Výučba jazyka Java na TUKE 39

2.6 Štúdia Minesweeper 42

2.7 Porovnanie výučby – Minesweeper a Pretória 46

2.8 Porovnanie výučby – Minesweeper a PigWorld 47

2.9 Možnosti a vlastnosti nástrojovej podpory výučby 49

2.9.1 Interaktivita výučby ... 49
2.9.2 Dynamické multimediálne zdroje pre výučbu ... 50
2.9.3 Flexibilita výučbových materiálov ... 50
2.9.4 Automatizácia vo výučbe ... 51
2.9.5 Motivačné faktory vo výučbe ... 51
2.9.6 Vývojové prostredie ako nástroj podpory výučby programovania ... 52
2.9.7 On-line zdroje pre podporu výučby programovania ... 52
2.9.8 Prenositeľnosť nástrojov ... 53
3 Nástroje pre testovanie postupu v štúdii ... 54
3.1 Ciele nástrojovej podpory ... 54
3.2 Požadované vlastnosti ... 55
3.2.1 Miesto nasadenia ... 55
3.2.2 Forma práce nástrojov ... 56
3.2.3 Zložky, časti a umiestnenia sady nástrojov ... 56
3.2.4 Rozhrania a formáty súborov komunikácie ... 58
3.2.5 Univerzálne použitie na rôznych platformách a predmetoch ... 59
3.2.6 Prepojenie foriem výučby ... 60
3.2.7 Overenie postupu na základe štruktúry ... 60
3.3 Nástroj tvorby nepriamej reprezentácie vzorového riešenia ... 61
3.3.1 Anotácie označenia vzorového riešenia ... 61
3.3.2 Štruktúra nástroja ... 63
3.3.3 Práca a výstup nástroja ... 64
3.4 Nástroj porovnania študentského zadania a vzoru ... 66
3.4.1 Štruktúra a práca nástroja ... 67
3.4.2 Štruktúra súborov logu ... 69
3.5 Nástroj spracovania chybového logu ... 70
3.6 Webová služba testovania ... 73
4 Experimentálne overenie a vyhodnotenie riešenia ... 74
4.1 Experiment A – výsledná aplikácia porovnania ... 74
4.2 Experiment B – prostredia ... 76
4.3 Zhodnotenie a budúcnosť aplikácie ... 77
5 Záver ... 79
Zoznam použitej literatúry ... 81
Prílohy ... 85

Zoznam obrázkov

Obr. 1: Príklad diagramu formy pre if-else konštrukciu 26
Obr. 2: Príklad diagramu formy triedy 27
Obr. 3: Príklad neformálneho diagramu algoritmu 27
Obr. 4: Príklad diagramu tried 28
Obr. 5: Príklad PigWorld-u 32
Obr. 6: PigWorld ako diagram tried (bez označenia použitia triedy) 34
Obr. 7: Ukážka spracovania materiálov pre cvičenia 41
Obr. 8: Ukážka hry Minesweeper 42
Obr. 9: Diagram tried minimálnej predpísanej aplikácie Minesweeper 43
Obr. 10: Náčrt toku informácií medzi nástrojmi 57
Obr. 11: Ukážka štruktúry XML súbor, časť konštruktory triedy Clue 64
Obr. 12: Ukážka výpisu chýb kontroly v prostredí NetBeans 68
Obr. 13: Detail ukážky chyby – URL odkaz na triedu s chybou 68
Obr. 14: Ukážka logovacieho súboru 70
Obr. 15: Ukážka časti výstupu spracovania logovacích súborov 72
Obr. 16: Výsledky experimentu A – typy chýb 75
Obr. 17: Výsledky experimentu A – počty chýb 76

Zoznam symbolov a skratiek

TUKE	Technická univerzita v Košiciach
KPI	Katedra počítačov a informatiky
XML	eXtensible Markup Language
XSD	XML Schema Definition
HTML	HyperText Markup Language
URL	Uniform Resource Locator
MAC	Media Access Control
IEEE	Institute of Electrical and Electronic Engineers, Inc.
resp.	respektíve
tzv.	takzvaný
ACM	Association for Computing Machinery
DOM	Document Object Model
CC2001	Computer Curricula 2001
GUI	Graphical User Interface, grafické používateľské rozhranie
PDF	Portable Document Format
MS	Microsoft

Slovník termínov

IEEE je skratka pre Institute of Electrical and Electronics Engineers, Inc. Tento inštitút sa zaoberá okrem iného tvorbou noriem pre elektronické a elektrotechnické inžinierske smery.

Java je multiplatformový programovací jazyk vyvinutý spoločnosťou Sun Microsystems. Podporuje rôzne elektronické zariadenia a operačné systémy. Je to objektovo-orientovaný, jednoduchý a rozšírený moderný programovací jazyk nezávislý na architektúre.

Override je prekrytie metódy triedy, od ktorej nižšia trieda dedí, za účelom zmeny funkcionality alebo prvotnej implementácie tela metódy. Prekrývaná i prekrývajúce metóda majú zhodnú deklaráciu, teda parametre, typ návratovej hodnoty a názov – iné je len umiestnenie v hierarchii tried.

Overload je viacnásobné použitie mena metódy v rámci triedy, pričom všetky použitia sa líšia aspoň v jednom z parametrov alebo v type návratovej hodnoty.

Swing je štandardná knižnica prvkov pre grafické rozhranie v prostredí jazyka Java.

XML je skratka pre Extensible Markup Language, čo je všeobecný a rozšíriteľný značkovací jazyk vyvinutý W3C konzorciom. Používa sa hlavne na výmenu údajov medzi aplikáciami, obsahuje len údaje a ich význam, nie formát vzhľadu.

Metaúdaje sú tzv. údaje o údajoch. Slúžia ako doplňujúca informácia interpretácie údajov, môžu napríklad urýchľovať vyhľadávanie alebo priradzovať údajom nejakú vlastnosť.

Úvod

Technická univerzita v Košiciach bola pôdou pre inšpiráciu témy tejto diplomovej práce. Vo výučbe na predmete Technológie sa používa štúdia Minesweeper, ktorá je spracovaná do formy 10 cvičení, kde študenti postupne pridávajú do dodaného projektu nové elementy a funkcie. Výsledkom je jednoduchá hra známa z prostredia MS Windows – Míny (Minesweeper). Podobným spôsobom je perspektívne možné a vhodné realizovať aj ďalšie predmety venujúce sa praktickému programovaniu v konkrétnom jazyku.

Z praxe výučby vyplynula potreba nástrojovej podpory, ktorá by automatizovane kontrolovala postup študentov počas semestra. Očakávania vychádzajú z potreby previazať kvalitne spracované materiály cvičení so samostatnou aktivitou študenta a tak mu poskytnúť hodnotnú spätnú väzbu v čo najväčšom rozsahu. Prínos pre vyučujúceho spočíva hlavne v získaní plošnej informácie o práci študentov počas semestra a tiež v možnosti ďalších vylepšení problematických miest štúdie.

Nasadenie integrovaných nástrojov pre podporu výučby programovania sa môže odohrať na viacerých miestach a môžu plniť rôzne jednoduché funkcie. V spojitosti budú poskytovať aj zložitejšie riešenia pre potreby vzdelávacieho procesu, ktorý sa v čase vyvíja. Dôležitá je možnosť používania nástrojov v rôznych prostrediach a na rôznych platformách, tiež možnosť webového nasadenia a použitie štandardizovaných rozhraní pre komunikáciu s externými systémami.

Prvá kapitola tejto práce formuluje úlohu a stručne informuje o okolnostiach, ktoré je nutné zohľadniť pri ďalšej práci.

Druhá kapitola analyzuje kontext, z ktorého je potrebné vychádzať pri posudzovaní vysokoškolskej výučby v oblasti programovania a menuje aspekty nástrojovej podpory pre zlepšenie v procesoch výučby. Obsahuje súhrn očakávaných vlastností sady nástrojov a základné prípady ich použitia. Zaoberá sa tiež obsahom štúdie, ktorá slúži ako základné východisko celej práce.

Tretia kapitola opisuje syntézu nástrojov podľa definovaných cieľov a výstupov analýzy. Zahŕňa použitie prostriedkov, ich vzájomnú väzbu a stanovuje metódy dosiahnutia viacerých menších úloh. Detailne opisuje architektúru projektov a procesy komunikácie prebiehajúce medzi jednotlivými časťami. Výstupom diplomovej práce je

teda tiež skupina nástrojov, ktorá je k dispozícii na nasadenie vo výučbe definovanej stanovenými cieľmi.

Záverečná štvrtá kapitola obsahuje experimentálnu časť s výsledkami testov a zdôvodnením významu a prínosov tejto aplikácie a výhľady na jej budúce použitie.

Hlavnými zdrojmi pre prácu boli erudované odporúčania pre výučbu programovania na vysokých školách z dielne komisie ACM a IEEE zloženej zo zástupcov informatických vied i praxe, ďalej dva zvolené materiály používané na iných univerzitách, odborná pedagogická literatúra z oblasti didaktiky informatiky a psychológie a tiež bakalárska práca, na ktorú táto diplomová práca nadväzuje.

1 Formulácia úlohy

Základná stanovená úloha spočíva v návrhu a vypracovaní integrovanej sady nástrojov umožňujúcej automatizovanú kontrolu postupu študenta v prípadových štúdiách pre výučbu objektového programovania.

Z kontextu výučby objektovo-orientovaného programovania na Technickej univerzite v Košiciach vyplynul výber jazyka Java a tak technické prostriedky prináležia tomuto jazyku. Za základné použité technológie boli zvolené technológie reflexie a anotácií; ako štandardizované rozhrania komunikácie pre súbory bol zvolený formát XML.

V práci sú zahrnuté opisy dvoch prírastkových štúdií a odporúčania vychádzajúce z dlhoročných medzinárodných skúsenosti v oblasti výučby programovania. Zahŕňa aj stručný súhrn pedagogických náhľadov na problematiku, hlavne z oblasti didaktiky.

Výsledný návrh a implementovaný systém musí poskytnúť jasné a merateľné výsledky. Jeho súčasťou je potrebná dokumentácia vypracovaná podľa pokynov vedúceho práce.

2 Analýza výučby programovania a nástrojovej podpory

Analytická časť tejto diplomovej práce pojednáva o východiskovej situácii vyučovania objektovo-orientovaného programovania na vysokých školách a možnostiach nástrojovej podpory pre zlepšenie vyučovacieho procesu a to automatizáciou kontroly postupu študentov v prípadových štúdiách.

Vhodnou pomôckou pri tvorbe študijných programov na vysokej škole v odboroch informačných technológií je zbierka odporúčaní a dlhoročných skúseností z priemyselného prostredia opísaná v časti 2.1. Popisuje, ktoré poznatky, aspekty a spôsoby majú a môžu byť použité pri výučbe v rámci vysokoškolského vzdelania v odbore informatiky. Zahrnuli sme ju pre obsahové porovnanie s výučbou na Technickej univerzite v Košiciach.

Z dostupných materiálov, ktoré sa venujú problematike výučby objektovo-orientovaného programovania na vysokých školách, bol vybraný vedecký článok z univerzity v Pretórii a štúdia k cvičeniam z univerzita v Sydney. Článok o problematike výučby sa venuje kritériám kladeným na materiály pre študentov a spôsobom usporiadania výučby, zvlášť cvičení. Rozsiahla austrálska štúdia bola vybraná pre blízku podobnosť so štúdiami, ktoré sa v rámci výučby realizujú na Technickej univerzite v Košiciach. Porovnanie štúdií môže viesť k inšpirácii a objaveniu ďalšieho potenciálu na vylepšenie výučby na TUKE. Pre porovnanie je samozrejme zahrnutý aj opis súčasného stavu.

Aktuálne sa vo výučbe používa štúdia Minesweeper, ktorá je použitá ako východisko pre špecifikáciu cieľov a požiadaviek na nástrojovú podporu, ktorá výrazne môže prispieť k skvalitneniu vyučovacieho procesu. Jej modulárnosť umožňuje nástrojovú kontrolu postupu študenta v prírastkovom programovaní malej aplikácie počas semestra v náväznosti na materiály cvičení. Opis očakávaného príspevku tejto nástrojovej podpory je opísaný v poslednej časti kapitoly.

2.1 Kurikulum „Computing Curricula 2001 Computer Science“

Rozsiahla štúdia vhodného obsahu informatických vied Computing Curricula 2001 Computer Science [1], ktorá je predmetom tejto kapitoly, bola venovaná komplexnej analýze výučby informatiky na prvom a druhom stupni univerzitného štúdia. Rozbor

problematiky bol vytvorený za účasti mnohých pedagogických a vedeckých odborníkov v oblasti počítačových vied. Jeho vytvorenie zastrešili IEEE Computer Society a Association for Computing Machinery, známa aj pod skratkou ACM a bol podporený grantmi pre vedecký výskum.

2.1.1 História

Podobné výskumy sa organizujú od šesťdesiatych rokov minulého storočia pravidelne približne každých desať rokov. Komisia pozostáva z približne 20 zástupcov IEEE a ACM. Výstupom je vždy kurikulum, ktoré v prehľadnej a systematickej forme poskytuje možnosti, odporúčania a postupy pre koncepciu výučby informatiky na univerzitách po celom svete.

Posledná verzia bola ukončená v decembri 2001 [1], aktuálne pripravovaná by mala vyjsť koncom roka 2013. Vzhľadom k rýchlym a rozsiahlym zmenám v oblasti informatiky bola pre rozšírenie posledného a aktuálne už 10 rokov starého kurikula v roku 2008 vydaná dočasná revízia a usmernenie [2].

Obdobné výskumy podporené priemyselnými a medzinárodnými štandardizačnými organizáciami poskytujú prehľad o medzinárodnom kontexte a perspektíve. Pre podklad k analýze bola zvolená táto štúdia z dôvodu relevantnosti jej garantov a najširšiemu rozsahu spracovania témy výučby.

2.1.2 Odporúčania základov a rozšírení

Štruktúra výučby by mala pozostávať zo zastúpenia rozličných teoretických a odborných predmetov. Bližšie opisované sú len oblasti informatiky, ktoré súvisia s objektovo-orientovaným programovaním.

Z celkového minimálneho času stráveného na prednáškach a cvičeniach počas univerzitného štúdia mala obsahovať približne 15% hodín základov programovania, ktoré majú ako prípravu zahŕňať:

- základné konštrukcie jazykov,
- algoritmy a riešenie problémov,
- základné údajové štruktúry,
- rekurzia a jej využitie,
- programovanie na základe udalostí.

Ďalej má výučba v ďalších 10% hodín obsahovať pojmy a vzťahy z oblasti algoritmov a zložitosti:

- základné analýzy algoritmov,
- stratégie algoritmov,
- základné výpočtové algoritmy,
- distribuované algoritmy,
- vypočítateľnosť.

Nezanedbateľné 3% hodín majú študenta naučiť čosi o používateľskom rozhraní:

- základy interakcie človek – počítač,
- tvorba jednoduchého grafického rozhrania.

A z nášho pohľadu najdôležitejších 8% hodín má študenta viesť k poznatkom z oblastí programovacích jazykov vrátane praktickej skúsenosti s vývojom:

- prehľad programovacích jazykov,
- virtuálne stroje,
- preklad jazyka,
- typy a deklarácie,
- abstraktné mechanizmy,
- objektovo-orientované programovanie.

Objektovo-orientované programovanie má v oblasti programovacích jazykov až polovičné zastúpenie a je hlavnou náplňou tejto kapitoly. Súvisí to s uplatnením absolventov po ukončení štúdia; väčšina z nich sa umiestni práve v pozícii programátora v objektovo-orientovanom jazyku, v súčasnosti najčastejšie Java alebo C#.

V kapitole programovacie jazyky navrhuje CC2001 ako možné a vhodné rozšírenia minimálneho štúdia uvedeného v 2.1.2 nasledovné predmety a oblasti:

- funkcionálne programovanie,
- typová kontrola,
- prekladače,
- sémantika programovacích jazykov,
- dizajn programovacích jazykov.

2.1.3 Stratégie realizácie

CC2001 navrhuje za vhodné nasledovné prístupy k predmetom úvodných informácií a zoznámenia sa s určitou oblasťou:

- najprv predstavenie hardvéru,
- najprv ukážky algoritmov,
- najprv demonštrácia funkcionality,
- najprv opis objektov v doméne,
- najprv zoznam pravidiel,
- najprv širší pohľad na kontext.

Pre rozširujúce predmety nasledujúce po predstavení základov CC2001 odporúča nasledovné možnosti prístupu:

- tematické zameranie,
- zameranie na systémy ako celok,
- zameranie na web, online prezentáciu a služby,
- zhustený skrátený prístup, určený pre iných ako denných študentov v štandardnej dĺžke štúdia.

Ďalšie predmety, ktoré poskytnú študentom hĺbkový pohľad do problematiky, majú byť prevažne voliteľného charakteru. Je dôležité, aby štúdium zahŕňalo viaceré smery a možnosti špecializácie, aby mal študent prístup k oblasti, ktorá ho zaujíma.

Kurikulum CC2001 jednoznačne neodporúča žiaden z prístupov, metód alebo programovacích jazykov ako správne, najlepšie a univerzálne riešenie – ponúka viaceré možnosti a uvádza ich bez subjektívneho hodnotenia alebo opisu skúseností.

Predmet Technológie Java, ktorým sa táto diplomová práca zaoberá, sa vyučuje ako rozširujúci predmet objektovo-orientovaného programovania a úvodný predmet z hľadiska konkrétneho jazyka.

2.1.4 Miesto programovania vo výučbe

Programovanie má v oblasti informatiky jednoznačne dôležité zastúpenie a tak sa väčšina študijných programov už desaťročia zameriava práve na budovanie schopností dobrého programátora ako na kľúčovú úlohu v rámci štúdia.

Už Computing Curricula z roku 1968 odporúčala ako vhodný úvodný kurz „Úvod do výpočtových operácií“, ktorý zahŕňal prevažne témy o programovaní. Ďalšia edícia z roku 1978 tento trend potvrdila aj premenovaním kurzu na „Úvod do programovania“.

Budovanie schopnosti efektívneho programovania je možné začať už v úvode štúdia, čo v sebe zahŕňa určite výhody a nevýhody opísané v nasledujúcej časti 2.1.5.

V súčasnosti je tento prístup najrozšírenejší a ukázal sa byť aj všeobecne vhodný pre rôznorodé skupiny študentov. V rámci tohto prístupu CC2001 opisuje tri možné spôsoby, ako učiť programovanie:

- procedurálne programovanie,
- objektovo-orientované programovanie,
- funkcionálne programovanie.

Počas desaťročí bol v odporúčaniach úvodný kurz k programovaniu v dĺžke dvoch semestrov. V súčasnosti to pomaly prestáva stačiť na pokrytie zväčšujúcej sa množiny základných údajov a postupov pre ďalšiu výučbu programovacích jazykov. CC2001 odporúča zvážiť troj-semestrový rozsah a poskytnúť lepšie základy, na ktorých možno ďalej budovať.

2.1.5 Prístup „programovanie ako prvé“ – výhody a nevýhody

Poďla CC2001 je výhodou prístupu, kde je programovanie vyučované od prvého semestra, hlavne dĺžka času, ktorý máme k dispozícii a veľký počet za sebou zaraditeľných predmetov, ktoré na seba môžu nadväzovať a postupne budovať predtým položené základy.

Umožňuje to takisto zoznámenie sa študentov s hlavnou náplňou ich štúdia okamžite po vstupe na univerzitu a znižuje to tak množstvo neskôr málo motivovaných ľudí, ktorí chcú školu dokončiť len preto, že sa v nej dostali už relatívne ďaleko. Takisto to umožňuje študentom stredných škôl, ktoré neboli zamerané na informatiku alebo programovanie, rýchlo dobehnúť náskok spolužiakov, ktorí pochádzajú z elektrotechnických a podobných stredných škôl.

Zaradenie programovania do úvodu výučby študentom dáva tiež príležitosť rýchlo pochopiť základy potrebné pre ďalšie predmety, ktoré nevyhnutne očakávajú poznatky na úrovni základného pochopenia štruktúr a algoritmov. Môžu sa tak zúčastniť predmetov, ktoré by inak pre nich boli nepochopiteľné alebo neúmerne zložité.

Zo skúseností je známe, že programovanie študentov zaujíma viac než iné oblasti informatiky – menej atraktívne sú zvyčajne napríklad matematika či elektrotechnika.

Motivácia u študentov je pri predmetoch s náplňou programovania vyššia aj z dôvodu, že vidia vo firmách priamo toto pracovné uplatnenie – na rozdiel napríklad od

fyziky. Potenciálni zamestnávatelia hľadajú budúcich pracovníkov práve na základe schopností v konkrétnom programovacom jazyku. Abstraktné vedomosti ako napríklad schopnosť zostaviť efektívny algoritmus sú ťažko sformulovateľné a overiteľné, tiež sa u absolventov informatiky automaticky predpokladajú, a tak sa im nepripisuje pri nábore veľká dôležitosť.

Často spomínaným problémom prístupu „programovanie ako prvé“ je nesprávne pochopenie informatických vied, ktoré málo znalý človek jednoducho zhrnie do slova „programovanie“. Obmedzuje to neskôr aj prístup a motiváciu študentov k neprogramátorským predmetom a tým aj ich teoretické základy pre ďalšie vyššie vzdelávanie. Výsledkom sú „programátori – robotníci“, ktorí sa dobre orientujú len v jednom programovacom jazyku, nemajú schopnosť vidieť súvislosti komplexne a teda ani nemôžu riadiť projekty či iných spolupracovníkov.

Pokiaľ zahrnutie programovania do začiatku výučby znamená, že je tým odsunuté vysvetlenie teoretického pozadia do neskorších semestrov, študenti často vnímajú teóriu ako zbytočnú, pretože získali programátorské znalosti aj bez nej.

Programovanie začiatočníkov je často obmedzené na jeden jazyk, čo vyvoláva zameranie viac na syntax a implementačné detaily daného jazyka ako na získanie schopností navrhovania vhodných algoritmov. Študenti myslia viac v možnostiach daného jazyka a nie v abstraktných matematických a analytických modeloch. Prístup tak potláča komplexnosť činností programovania.

Študenti, ktorí už mali predošlú skúsenosť s programovaním, sú v značnej výhode oproti tým, ktorí túto skúsenosť nezískali. To môže vyvolávať dojem neschopnosti alebo nadradenosti u časti kolektívu. Zvyčajne tiež nie je možné sformovať správne návyky u neskúsených študentov kvôli vplyvu ich spolužiakov, ktorým sa snažia vyrovnať a kopírujú ich zaužívané spôsoby.

Častým dôsledkom prístupu „programovanie ako prvé“ je aj viera študentov v to, že jediné vhodné riešenie problému spočíva v napísaní programu a tak je znalosť programovacieho jazyka nevyhnutnosťou – pričom je všeobecne známe, že je ťažšou časťou programovania vytvorenie algoritmu riešenia než jeho zápis do programovacieho jazyka.

2.1.6 Objektovo-orientované programovanie

CC2001 odporúča v úvode študentom objasniť základnú predstavu o pojmoch ako sú objekt a dedičnosť – aj na jednoduchých príkladoch interakcie. Pri prvom dotyku s vývojovým prostredím je samozrejme nutné osvetliť študentom základné údajové štruktúry, ale vždy so zameraním na objektovo-orientovaný dizajn.

Neskôr, po zvládnutí základného dizajnu, je vhodné predstaviť algoritmy metód a primitívne údajové štruktúry špecifické pre zvolený objektovo-orientovaný jazyk.

Je vhodné citlivo voliť vybrané prvky jazyka do výučby, pretože moderné objektovo-orientované jazyky majú široké možnosti a v porovnaní s procedurálnymi sú omnoho zložitejšie.

Rovnako môžu mať žiaci problémy aj so zvládnutím vývojového prostredia, ktoré ponúka už v základe nepreberné množstvo rozličných funkčných prvkov. Preto je už pred samotným „Hello, World!“ programom dôležité aspoň stručne opísať jeho viditeľné časti a ich funkcie.

Každý zo študentov by vzhľadom k požiadavkám praxe na programátorov mal zvládnuť aspoň tieto základy. Dokladuje to aj rastúci význam objektovo-orientovaného prístupu na trhu.

Z dlhoročných skúseností členov komisie CC2001 vychádzajú nasledovné témy, ktoré by mal obsahovať kurz objektovo-orientovaného programovania v rozsahu 10 vyučovacích hodín (teda približne jeden semester alebo trimester):

- objektovo-orientovaný dizajn,
- zapuzdrenie (encapsulation) a skrývanie informácií,
- oddelenie vlastností a implementácie,
- triedy a podtriedy,
- dedenie a tzv. „overriding“,
- mnohotvarosť (polymorfizmus),
- hierarchia tried,
- kolekcie a iterácie,
- vnútorné reprezentácie objektov a diagramy.

Odporúčané ciele výučby podľa CC2001:

- zdôvodniť filozofiu objektovo-orientovaného dizajnu, koncepcie zapuzdrenia, abstrakcie, dedenia a polymorfizmu,

- navrhnúť, implementovať, otestovať a odladiť (debug) jednoduché programy v objektovo-orientovanom jazyku,
- opísať, ako mechanizmus tried a balíkov podporuje zapuzdrenie a skrývanie informácií,
- navrhnúť, implementovať a otestovať vzťahy objektov v hierarchii tried a v rámci dedenia,
- porovnať a opísať rozdiely „override“ a „overload“ (viď slovník termínov) v použití v objektovo-orientovanom jazyku,
- vysvetliť vzťahy statickej štruktúry triedy a dynamickej štruktúry objektov – inštancií triedy,
- opísať možnosti iterácie elementov v kolekcii.

Z opísaného dokumentu vychádzajú mnohé študijné programy informatiky po celom svete. Vzdelanie v tomto odbore len málo podlieha lokálnym špecifikám jednotlivých krajín, zjednotenie obsahu je teda namieste. Priebežná aktualizácia znamená prispôsobovanie sa dynamickým zmenám a tak je potrebné brať opísané návrhy do úvahy pri posudzovaní výučbe programovania na vysokých školách. V ďalšej časti sa pozrieme na dva v praxi používané prístupy k výučbe programovania.

2.2 Výučba na univerzite v Pretórii, Južná Afrika

Viac-semestrovú skúsenosť s programom výučby jazyka Java prezentuje v dokumente o výsledkoch tejto snahy Judith Bishop z univerzity v Pretórii [3].

Posledné desaťročie ukázalo, že Java získala dlhodobé a stabilné zastúpenie na trhu informačných technológií. V oblasti vzdelávania je teda neopomenuteľnou súčasťou prípravy každého informatika na povolanie. Pre efektívnu zmenu študenta na zdatného programátora je potrebné zvoliť správnu cestu, zvlášť ak ide o študenta a aj vyučujúceho s predchádzajúcimi skúsenosťami s jazykmi inej paradigmy [4].

Oproti jazykom, ktoré Javu v čase predišli (napríklad Pascal alebo C++), má určité odlišnosti:

- samotný jazyk je malý a objektovo-orientovaný,
- má málo zabudovaných štruktúrnych typov (len pole – array), ale poskytuje množstvo rôznych štandardných knižníc – tried, ktoré pokrývajú všetky bežné potreby,

- poskytuje plné grafické rozhranie a podporu multimédií, tiež podporu pre web a applety (doplnkové interaktívne Java aplikácie pre web stránky),
- poskytuje podporu pre súbežné vykonávanie a umožňuje programovanie v sieti.

Z týchto vlastností môžeme vyvodiť dva možné prístupy k výučbe:

- začiatok ako práca s grafickým používateľským rozhraním (GUI) a multimédiami, ktoré sú pre študentov prirodzene atraktívne kvôli praktickosti, interaktivite a zaujímavej forme.
- začiatok klasickým prístupom, vysvetlenie základov a postupné budovanie na krátkej úvodnej teórii.

Nevýhodou začiatku priamo s GUI je skúsenosť, že študenti majú problémy s porozumením podstaty toho, čo vlastne robia [5].

Odpoveďou na dilemu „ako zaujať študentov už na začiatku“ je kombinácia týchto prístupov. Keďže má Java v porovnaní s inými jazykmi malé množstvo zabudovaných štruktúrnych typov, nie je nutné v úvode obšírne vysvetľovanie konštrukcií jazyka samotného. To umožňuje po rýchlom teoretickom úvode čoskoro prejsť k zaujímavým témam.

Judith Bishop ďalej odporúča aplikovať nasledovné myšlienky [6]:

- diagramy a schémy všade, kde je to možné,
- vysvetlenie na príklade,
- samo-opravné kontroly a pravidelné praktické cvičenia.

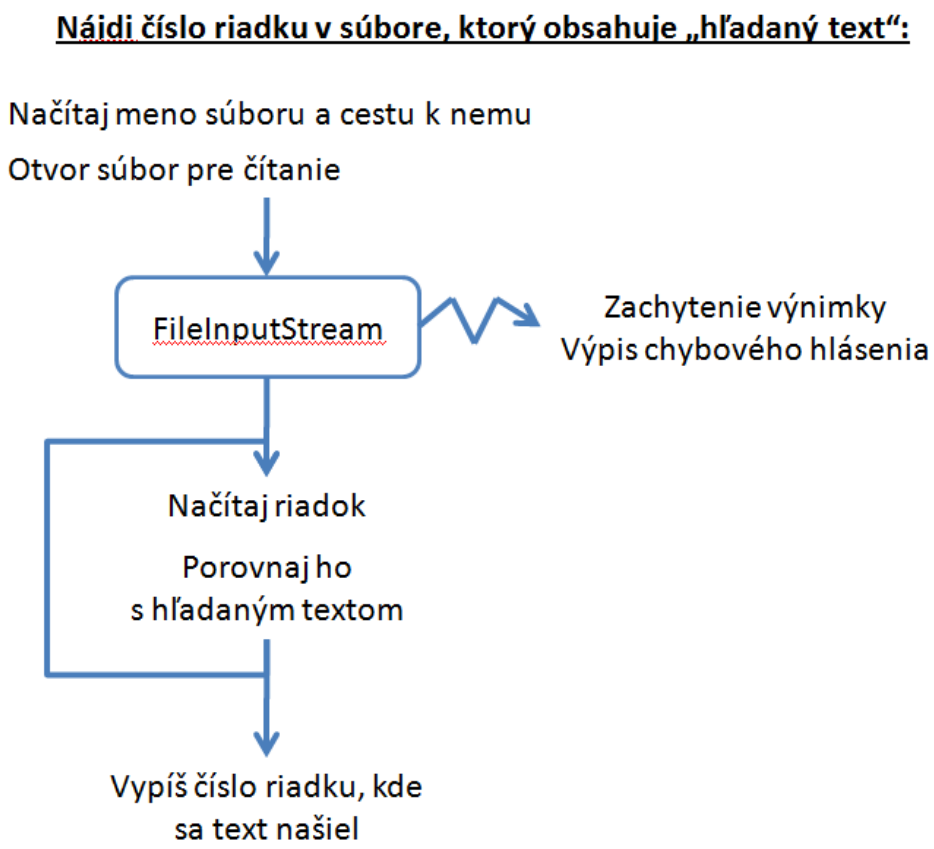
if – príkaz
if (podmienka) *tak-potom-vykonaj*; **else** *vykonaj-toto*;

Obr. 1: Príklad diagramu formy pre if-else konštrukciu

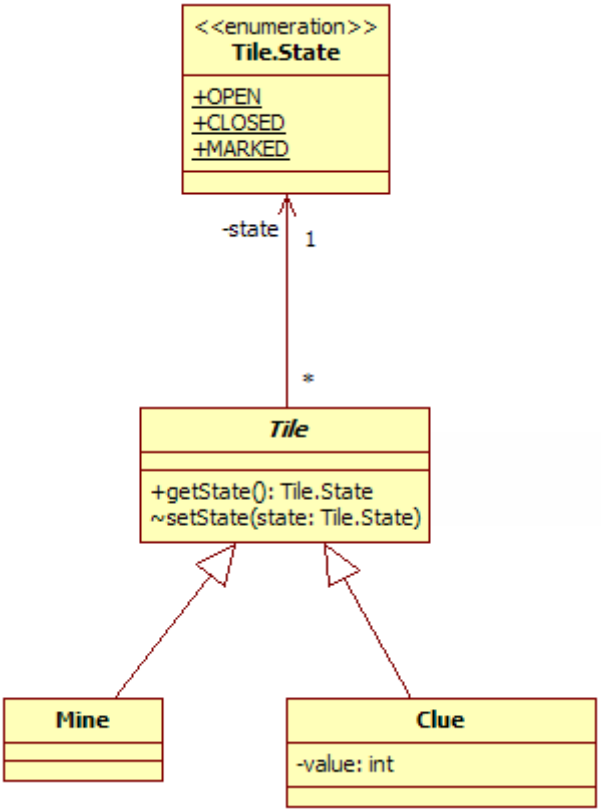
Program

```
class menoTriedy {
   public static void main (String [] menoArgumentu) {
      deklarácie a príkazy;
   }
}
```

Obr. 2: Príklad diagramu formy triedy



Obr. 3: Príklad neformálneho diagramu algoritmu



Obr. 4: Príklad diagramu tried

Je vhodné používať diagramy nasledovných typov:

- diagramy formy – pre každú novú konštrukciu jazyka je vhodné zaviesť diagram syntaktickej formy a príklad použitia (viď Obr. 1 a Obr. 2).
- diagramy algoritmov – na ukážku toku riadenia programu alebo toku údajov v programe; sú vhodné aj pre zavádzanie pojmov ako výnimka alebo vlákna; najlepšie na malých a úplne jednoduchých programoch (viď Obr. 3).
- diagramy tried – jasné schémy pre utvrdenie štruktúry a hierarchie tried programu, tiež pre systematické a zrozumiteľné priradenie metód a premenných jednotlivým triedam a ich interakcii medzi nimi (viď Obr. 4).

Diagramy nemusia byť, hlavne spočiatku, úplne detailné alebo formálne – postačí, keď ponúknu začiatočníkom stručný pohľad na podstatné časti a prvky. Jednoduché diagramy tiež vedú študentov k tomu, aby počas procesu tvorby aplikácie používali nimi vlastnoručne vytvorené a na papieri načrtnuté diagramy. Nestanú sa tak len

súčasťou nikým neštudovanej dokumentácie, ktorú zvyčajne študenti vytvárajú až po ukončení vývoja aplikácie.

Predmet z oblasti programovania má študenta nielen naučiť zvládnuť implementáciu programu v určitom programovacom jazyku, ale viesť k riešeniu problému. To znamená zvládnuť návrh údajových štruktúr a algoritmov, ktoré povedú k požadovanému výsledku [7].

Dôležité je viesť študentov k abstrakcii, všeobecnosti a tým aj k možnosti opakovaného použitia časti zdrojového kódu.

Správne usporiadanie práce s objektmi je navrhnuté nasledovne [8][9]:

- použitie objektov hotových preddefinovaných tried, napríklad triedy pracujúce s časom a dátumom, vo vlastnej aplikácii,
- deklarácia vlastných tried s návrhom štruktúry podľa vlastných potrieb.
- dedenie a ďalšie vlastnosti tried, resp. objektov.

2.3 Výučba na univerzite v Sydney, Austrália

V tejto podkapitole sú podrobnejšie rozvedené skúsenosti a závery z univerzity v Sydney prezentované v článku „Teaching Java First: Experiments with a Pigs-Early Pedagogy“ [29]. Viacročný výskum sa zaoberal študentmi, ktorí sa učili jazyk Java ako svoj prvý programovací jazyk a aj tými, ktorí už predtým aktívne poznali programovanie v procedurálnych jazykoch, keďže šlo o veľkú skupinu študentov, rádovo v stovkách.

Existuje výrazná podobnosť so spôsobom výučby na TUKE – žiakom je predkladaná postupne upravovaná štúdia, počas ktorej získavajú prvé skúsenosti s jazykom Java.

2.3.1 Východiská a okolnosti

Podľa CC2001, podrobnejšie rozobranej v 2.1, aplikuje skúmaný predmet prístup programovanie ako prvé (viď 2.1.4), stratégiu tematického zamerania a demonštrácie funkcionality (viď 2.1.3) a pokrýva väčšinu požiadaviek na náplň (viď 2.1.6).

Ako CC2001 konštatuje, moderné objektovo-orientované jazyky ako Java či C++ sú podstatne komplexnejšie ako klasické jazyky. Bez osobitnej starostlivosti venovanej predstaveniu jazyka spôsobom, ktorý obmedzuje túto zložitosť (na pohľad zjednodušuje jazyk), takéto podrobnosti môžu ľahko zahltiť študentov, zvlášť začiatočníkov.

Dokazuje to doteraz pravdepodobne najrozsiahlejšia štúdia konceptuálnych štruktúr začínajúcich programátorov [18], poukazujúca na preťaženie študentov. V tejto štúdii boli študenti požiadaní, aby umiestnili rôzne programovacie konštrukcie a koncepcie do kategórií, na základe ľubovoľných kritérií. Konštrukcia "objekt" bola najčastejšie umiestňovaná spolu s ďalšími troma termínmi, a to "strom", "zoznam" a "pole". To znamená, že mnoho študentov si o objektoch naivne myslí, že sú primárnym miestom, kde sú uložené údaje podobne ako v poliach, zoznamoch a stromoch. Príčina pravdepodobne tkvie v prevahe príkladov tried, ktoré obsahujú len o málo viac okrem súkromných údajov elementov triedy a ich "get" a "set" metód.

Niektorí počítačoví odborníci sa domnievajú, že problém presahuje voľbu programovacieho jazyka. Mnohí sú totiž presvedčení, že výučba procedurálnej programovacej paradigmy musí byť na prvom mieste, keďže objektovo-orientované programovanie "v žiadnom prípade nenahradí staršiu paradigmu – má byť jej doplnkom" [19]. Toto tvrdenie síce bolo vyvrátené [20][21], ale je potrebné pripustiť, že principiálna výučbu objektovo-orientovaného programovania sa ukázala byť náročnejšia, než mnohí pôvodne očakávali.

Dlho pred jazykom Java sa hľadali spôsoby ako učiť programovanie prívetivejšími a pútavejšími spôsobmi. Pôvodný "Robot Karel" [22], používaný na vyučovanie Pascalu, bol dobre známy v rámci programovania virtuálnych robotov na plnenie úloh vo svete reprezentovanom šachovnicou. Prístup Karel bol aktualizovaný pre Javu najmenej v troch nezávislých prípadoch [23][24][25]. Avšak v niektorých prípadoch dôraz na algoritmy v rámci jedinej triedy, často navyše rekurzívne algoritmy, prináša do výučby výraznú procedurálnu príchuť.

2.3.2 Predstavenie myšlienky

PigWorld je pre univerzitu v Sydney a hlavne jej študentov nástroj odhaľovania sveta objektovo-orientovaného programovania. Hlavnou postavou je „bravčové“, ktoré sa pohybuje, kŕmi a miluje sa v mriežkovom mikrosvete. Organická metafora PigWorld-u nie je prvé použitie nerobotickej organickej metafory [26][27][22], ale toto jej poňatie kladie dôraz na interakciu medzi objektmi, najmä inštanciami rovnakej triedy.

Vývoj PigWorldu postupne napredoval v priebehu niekoľkých semestrov. Vyvinul sa v prístup, ktorý nezaobchádza s "objektmi" a "procedúrami" dichotomicky, ale začleňuje algoritmické komponenty pri použití interakcie medzi objektmi.

2.3.3 Systém predmetu a hodnotenia

Na univerzite v Sydney existujú štyri stupne absolvovania predmetu: od najnižšej „pass“ (prešiel, v hodnotení slovenských vysokých škôl zvyčajne známka 3, E alebo 51%) po hodnotenie „s vyznamenaním“.

Štruktúra predmetu [28][29] nevyžaduje od študentov, ktorí sa uspokoja s hodnotením „pass“ ani jedinú úlohu z PigWorld-u. Pre lepšie hodnotenie musia úspešne splniť niektoré z úloh štúdie.

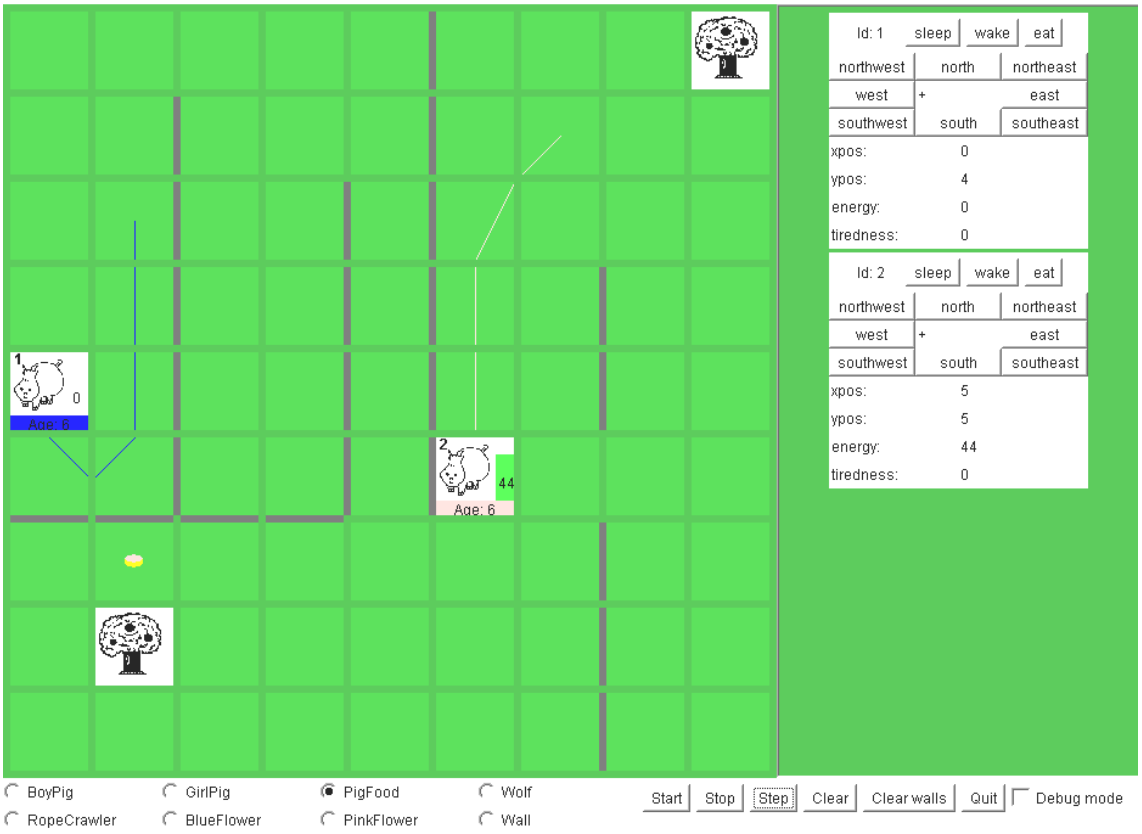
Približne polovica študentov zvyčajne neprejaví žiaden záujem o nič viac ako známku „pass“ aj napriek tomu, že prvá zo série úloh je naozaj jednoduchá.

Keďže ide o voliteľnú časť predmetov, štúdia nie je bližšie predstavovaná ani diskutovaná počas prednášok. Študenti sa snažia splniť úlohy len na základe opisu v špecifikácii zadania a odpovedí na otázky prostredníctvom online diskusného fóra.

Pri jednotlivých úlohách štúdie sa očakávajú a používajú znalosti z už absolvovaných predmetov; tieto skúsenosti sa ďalej rozvíjajú.

Štúdia s opisom úloh je študentom k dispozícii ako textový dokument vo formáte MS Word. Neobsahuje obrázky, len niekoľko náčrtkov a blokov zdrojového kódu. Podrobne opisuje úlohy a stručne aj uvádza do témy PigWorld-u, celkovo na 27 stranách. Štúdia je online dostupná na Internete [15].

Štúdia v čase písania práce nezahŕňala žiadnu nástrojovú podporu, zadania boli kontrolované vyučujúcim na základe praktickej funkčnosti implementácie.



Obr. 5: Príklad PigWorld-u

2.3.4 Predstavenie sveta

Obr. 5 ukazuje obrázok typického scenára PigWorld-u. V tomto scenári je jedno prasa (chlapec označený modrým prúžkom) na ľavej strane a jedna prasnica (dievča, ružové) napravo.

Sú tam dva stromy ako potrava pre ošípané, a v tesnej blízkosti jedného zo stromov je objekt podobný tabletke, ktorý naznačuje, že niektorý zo stromov práve zhodil nejaké krmivo pre ošípané. V PigWorld-e sa hladné ošípané pohybujú po jednom štvorčeku smerom k najbližšiemu jedlu, ktoré potom jedia, aby z neho získali energiu. Pohyb po PigWorld-e odoberá zvieratám energiu. Na Obr. 5 číslo a pás na pravej strane ikonky dievčaťa ukazuje, že má 44 jednotiek energie, zatiaľ čo chlapec nemá žiadnu energiu.

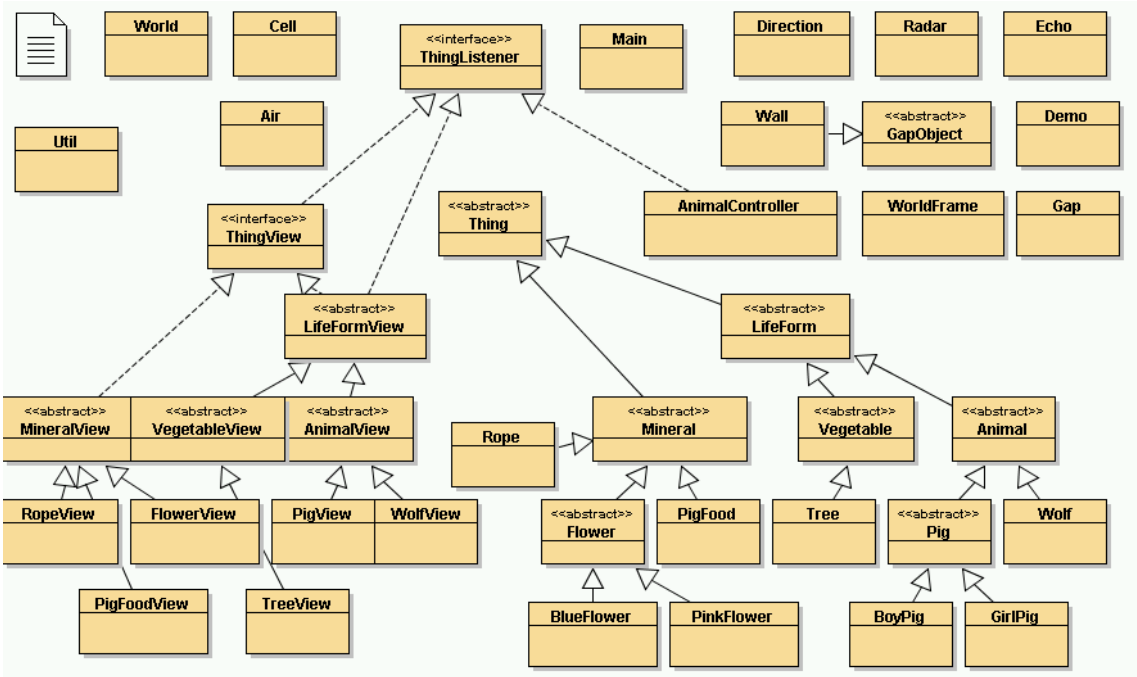
Ako sa ošípané pohybujú po svete, ťahajú za sebou lano. Lano slúži ako pripomienka pre ošípanú, koľko štvorcov už navštívila. Pomocou tohto zariadenia môžu prechádzať bludisko. Na obrázku za sebou obe ošípané ťahajú lano cez niekoľko štvorčekov.

V dolnej časti je panel nástrojov rôznych tzv. rádiových tlačidiel. Tieto tlačidlá umožňujú užívateľovi pridávať do PigWorld-u ďalšie prvky, napríklad múry, cez ktoré ošípané nemôžu prejsť. Na pravej strane sú dva regulátory, jeden na každé prasa. Číselný identifikátor v ľavom hornom rohu každého regulátora zodpovedá číslu zobrazenému vľavo nad prasaťom.

Ošípané majú slabý zrak, takže v PigWorld-e používajú radar na registrovanie svojho okolia. Radary vracajú inštanciu triedy „Echo“, ktorá zahŕňa azimut (uhol oproti severu) a vzdialenosť práve zistenej entity. Radary vidia aj skrz steny, takže hoci prasa vie, že v blízkosti je niečo zaujímavé, musí byť navigované okolo múrov, aby to dosiahlo.

Dostatočným navýšením energie prasaťa prasa stráca záujem o jedlo – namiesto toho sa „naladí na lásku“. Keď je prasa v takejto nálade, objavia sa na jeho ikonke malé srdiečka a usmieva sa, prasa – dievča navyše vydáva virtuálne zvuky, ktorých intenzita postupne klesá ako sa zväčšuje vzdialenosť od zdroja. Taktiež, akýkoľvek chlapec naladený na lásku sa pohybuje zo svojho aktuálneho miesta na najbližšie miesto, odkiaľ počuje dievčenské zvuky. Nakoniec je dievča naladené na lásku obklopené viacerými nápadníkmi.

Ak sú dievča aj chlapec v nálade na lásku, mali by obsadiť susedné polia, kde sa môžu pária, a tým vyrobiť nové prasiatko. Ich Potomok sa objaví na prázdnom mieste vedľa svojich rodičov. Po párení si obaja rodičia užívajú krátky spánok – na ikonke zvieratka sa zobrazí „Zzz“. Život v PigWorld-e však nie je len jedle a sexe. V hre sú prasatá ohrozované vlkmi a inými predátormi, ktoré na ne útočia.



Obr. 6: PigWorld ako diagram tried (bez označenia použitia triedy)

Obr. 6 ukazuje diagram tried PigWorld-u, pričom sú pre jednoduchosť označené len vzťahy dedenia a použitia rozhrania, nie vzťah použitia triedy.

Prvý silný dojem je, že PigWorld obsahuje viac ako 40 tried. Po dobu 4-6 týždňov pred PigWorld-om študenti najskôr pracujú na laboratórnych cvičeniach, ktoré majú nanajvýš 3 triedy. Po počiatočnom šoku zo 40 tried v niekoľkých upokojujúcich vetách študentom vysvetlíme, že v úlohách v PigWorld-e vždy určíme konkrétne metódy a triedy, ktoré majú byť upravené a zvyčajne stanovujeme metódy ďalších tried , ktoré by mohli byť použité. Snažíme sa tak o to [30], aby študenti začali malými zmenami a postupovali na veľké projekty; s dôrazom viac na čítanie tried ako na písanie kódu.

Druhý dojem je použitie výrazného dedenia medzi triedami a abstraktné triedy. Počas semestra spočiatku študenti pracujú podľa vzorov predstavených na prednáškach na triedach dediacich od triedy „Vec“ (Thing), neskôr zapracúvajúc grafiku rozširujú obsah tried vizuálnej reprezentácie (View). To umožňuje sústredenie sa na vzory tried bez toho, aby sme boli rušení grafikou alebo spracovaním jednotlivých udalostí.

2.3.5 Skúsenosti z realizácie štúdie

Zaujímavým momentom bola jedna z prvých úloh – zamedzenie párenia sa prasiat, ktoré sú súrodencami. Študenti mali porovnať odkazy materských objektov v „if“ podmienke. Prekvapením boli problémy tých študentov, ktorí mali predchádzajúcu procedurálnu skúsenosť – videli triedy a inštanciu triedy ako rovnocenné; preto sa im

komunikácia medzi inštanciami rovnakej triedy zdala byť rekurzívnym procesom, v ktorom objekt komunikoval sám so sebou.

Kým v predchádzajúcich Java cvičeniach písali kód pre zasielanie správy medzi objektmi, tieto cvičenia obsahovali objekty rôznych tried, a tak chybné domnienky študentov neboli odhalené. Úprava na zabránenie incestného plodenia sa bola prvým pokusom študentov o napísanie kódu pre posielanie správ medzi inštanciami rovnakej triedy. V ďalších semestroch sa študenti dostanú k úlohám, ktoré vyžadovali metódu komunikácie medzi inštanciami jednej triedy. Skúsenosti poukázali na to, že komunikácia medzi inštanciami jednej triedy by sa mala učiť čo najskôr, a takisto má byť posilňovaná tak často, ako je len možné, lebo je to zásadné pre výučbu objektovo orientovaného programovania.

Ako sme opísali v 2.3.4, cvičenia boli koncipované s cieľom viesť študentov k dôslednému študovaniu už hotového dodaného zdrojového kódu. Niektoré cvičenia pozostávali len z napísania niekoľkých riadkov kódu, na čo ale bolo potrebné pochopiť princíp interakcie s viacerými inými triedami. Študentom to poukazovalo na dobré zvyky a vzory tvorby architektúry programu objektovo-orientovaného jazyka.

Na prvé úlohy v rámci PigWorld-u sa stačí naučiť jeden základný algoritmus: nájdenie najmenšieho alebo najväčšieho prvku v sekvencii prvkov, kde je sekvencia ukončená špecifickou hodnotou (napr. znakom EOF alebo nulou). Tento algoritmus sa vyučuje ako laboratórne cvičenie pred predstavením PigWorld-u.

Študenti učiaci sa objektové programovanie by sa na začiatku semestra nemali učiť nič, čo sa týka procedurálnej podstaty okrem vyššie spomenutého algoritmu. Tento jednoduchý algoritmus vyžaduje slučku obsahujúcu „if“ podmienku.

Študenti by sa mali zamerať na algoritmy interakcie medzi objektmi, nie na implementáciu algoritmov správania sa samotného objektu.

Na základe skúsenosti odporúčame, aby boli z počiatočných cvičení vynechané polia a ďalšie podobné štruktúry známe z procedurálneho programovania, pretože študenti ich nechápu ako objekty, resp. odvádzajú študentov od zásadných prvkov objektovo-orientovaného programovania. Je možné nahradiť ich spájanými zoznamami (linked list), resp. neskôr kolekciami s iterátorom.

2.4 Výučba programovacích jazykov na TUKE

Výučba programovacích jazykov na Technickej univerzite v Košiciach v odbore informatika sa nachádza prvom (bakalárskom) aj druhom (inžinierskom) stupni vysokoškolského vzdelania. Zahŕňa všetky dôležité oblasti súčasného použitia jazykov v praxi a svedomitým študentom tiež poskytuje dobrý základ pre ďalšie učenie sa z vlastnej iniciatívy. Keďže výučba každého predmetu je zasadená v kontexte viacerých nadväzujúcich predmetov, je potrebné brať ohľad aj na súvislosti a prípravu získanú na predchádzajúcich predmetoch.

2.4.1 Chronológia programovacích jazykov v predmetoch

V súčasnosti sa na Technickej univerzite v bakalárskom stupni vyučujú najprv viaceré predmety zamerané na programovanie v nižšom jazyku C (viď 2.4.2 a 2.4.3). Neskôr majú povinný predmet „Objektové programovanie“ vyučovaný v jazyku Java. V nasledujúcom semestri sa študenti zoznámia s nadväzujúcim povinne voliteľným predmetom „Technológie Java“, alternatívne s „Programovaním v prostredí DotNET“. Každý študent teda musí absolvovať minimálne 1 semester jazyka Java.

Objektové programovanie v treťom semestri prvého stupňa vysokoškolského štúdia opísané v 2.4.4 je úvodom do novej paradigmy programovania a nie je zamerané na detailné možnosti a vlastnosti samotného jazyka Java. Študenti sú vedení k pochopeniu nového spôsobu myslenia pri tvorbe programu a algoritmov. Samotný prechod od procedurálnej k objektovej filozofii programovania je náročný a tak je vhodné, že jazyku Java je venovaný samostatný predmet, kde žiaci ďalej rozvíjajú už získanú skúsenosť. Z tohto hľadiska môžeme v rámci rozdelenia predmetov podľa 2.1 zaradiť medzi rozširujúce predmety.

2.4.2 Porovnanie s odporúčaniami CC2001 – prvý semester

V súčasnosti sú na Technickej univerzite pred vyučovaním samotného programovania povinné pre študentov prvého stupňa viaceré predmety. Sú zamerané hlavne na teoretické základy z oblasti hardvéru a matematiku.

Prvým predmetom, kde sa študenti venujú programovaniu, je „Úvod do programovania a sietí“ hneď v prvom semestri. Poskytuje prehľadový popis pojmov, s ktorými sa študenti budú v ďalších semestroch stretávať, čím sa aplikuje stratégia širšieho pohľadu na kontext (viď 2.1.3).

Zároveň sa v prvom semestri vyučujú „Základy elektrotechnického inžinierstva“ a „Základy inžinierstva materiálov“, ktoré žiakom ozrejmia fyzikálnu podstatu javov súvisiacich s elektrotechnikou. Nejde o predmety predstavujúce informatický hardvér ako taký, čiže sú len podporou stratégie pohľadu na kontext.

Pre upevnenie základov a zjednotenie úrovne študentov je v prvom semestri zaradený predmet „Matematika I“.

Doplnkovými predmetmi sú „Podnikanie a manažment“, telesná výchova a voliteľný seminár z matematiky.

2.4.3 Porovnanie s odporúčaniami CC2001 – druhý semester

Už v letnom semestri prvého ročníka prichádzajú „skutočne programátorské“ predmety. Podľa 2.1.4 a ich náplne spadajú do spôsobu výučby, ktorý začína procedurálnym programovaním, z čoho plynú výhody opísané v 2.1.5. Počas výučby je zreteľný dôraz u pedagógov aj na teoretické pozadie týchto vedomostí, vďaka čomu sú úspešne potláčané viaceré z nevýhod spomenuté v 2.1.5.

Pre študentov, ktorým viac vyhovuje prístup opisu hardvéru ako prvého, sú zahrnuté v tomto semestri predmety „Elektrotechnika“, „Fyzika“ a všeobecnejšie zamerané „Princípy počítačového inžinierstva“, ktoré sa teoreticky venujú analýze a syntéze funkcií obvodov.

Doplnkom sú voliteľné predmety cudzieho jazyka a semináre z matematiky, fyziky alebo elektrotechniky. Tie podľa kritérií CC2001 spĺňajú kritériá prístupu tematického zamerania rozširujúcich predmetov (viď 2.1.3).

2.4.4 Porovnanie s odporúčaniami CC2001 – tretí semester

Tretí semester ponúka študentom pre rozvinutie algoritmického myslenia a matematického pohľadu na riešenie problémov v informatike „Matematickú logiku“, pre získanie praktických zručností prvú časť série „Administrácia operačného systému Linux“ a ako doplnkové predmety cudzí jazyk a jeden predmet spoločenských vied.

Pre ďalšiu výučbu programovania ponúka hneď 3 predmety a to:

- Údajové štruktúry a algoritmy,
- Formálne jazyky a prekladače,
- Objektové programovanie.

„Formálne jazyky a prekladače“ poučia študentov v klasifikácii podľa 2.1.2 obsahujú vedomosti z nasledovných oblastí:

- základné konštrukcie jazykov,
- základné dátové štruktúry,
- základné analýzy algoritmov,
- stratégie algoritmov,
- základné výpočtové algoritmy,
- preklad jazyka,
- typy a deklarácie,
- funkcionálne programovanie,
- typová kontrola,
- prekladače,
- sémantika programovacích jazykov,
- dizajn programovacích jazykov.

Predmet „Údajové štruktúry a algoritmy“ poskytne pokrytie nasledovných oblastí podľa 2.1.2:

- algoritmy a riešenie problémov,
- základné dátové štruktúry,
- rekurzia a jej využitie,
- základné analýzy algoritmov,
- stratégie algoritmov,
- základné výpočtové algoritmy,
- distribuované algoritmy,
- vypočítateľnosť,
- dizajn programovacích jazykov.

„Objektové programovanie“ sa dotýka nasledovných oblastí podľa 2.1.2:

- základné konštrukcie jazykov,
- algoritmy a riešenie problémov,
- základné dátové štruktúry,
- základy interakcie človek – počítač,
- objektovo-orientované programovanie.

Predmet „Objektové programovanie“ študentov oboznámi s novou paradigmou programovania, princípmi ako zapuzdrenie, zakrývanie, dedičnosť, hierarchia a polymorfizmus. Predstaví tiež základné vzory a celý postup od návrhu aplikácie až po jej implementáciu. Na cvičeniach študenti získavajú zvyčajne prvé praktické skúsenosti s týmto spôsobom programovania. Aktuálne sú cvičenia vedené v jazyku Java, v minulosti sa používal jazyk C++, resp. C#. Výborným aspektom prístupu vyučujúcich je, že študenti majú voľnosť vo výbere jazyka pre zadanie, ktoré spracúvajú za pomoci cvičiacich počas semestra a na jeho konci je súčasťou hodnotenia zápočtu.

Prvá polovica prednášok predmetu je v súlade s odporúčaniami CC2001 v 2.1.6 venovaná vlastnostiam objektovo-orientovaného dizajnu, neskôr sú na vzoroch predstavené algoritmy a na cvičeniach si študenti všetko môžu dobre overiť. „Objektové programovanie“ tvorí výborný základ pre predmet Technológie Java, ktoré ho nasledujú v štvrtom semestri.

2.4.5 Porovnanie s odporúčaniami CC2001 – ďalšie semestre

V ďalších semestroch po absolvovaní predmetu „Technológie Java“ je možné tento jazyk v hojnej miere využívať na viacerých rozširujúcich voliteľných predmetoch, hlavne na predmetoch zaoberajúcich sa webovými technológiami a službami ako Technológie budovania informačných systémov I. a II., Technológie softvérových projektov I. a II. a Webové technológie, tiež na niektorých predmetoch zameraných na databázové technológie.

2.5 Výučba jazyka Java na TUKE

Cieľom výučby je pripraviť študenta na základné spektrum možností, vlastností a spôsobov použitia jazyka Java v neskoršom pracovnom uplatnení. Pre splnenie tohto cieľa je nutné položiť dobré základy už pri prvom kontakte s jazykom, aby študenti správne používali jeho možnosti, osvojili si ním používané konvencie a tvorili efektívne programy s dobrou architektúrou a zrozumiteľným zdrojovým kódom.

Výučba jazyka Java sa na Technickej univerzite v Košiciach realizuje na predmete „Objektové programovanie“ a predmete „Technológie Java“. Tieto predmety sa zaoberajú len časťou platformy jazyka Java a to Java Standard Edition, skrátene Java SE. Ďalšie samo-štúdium je ponechané na študentoch popri jeho využití v rozširujúcich predmetoch (viď 2.1.2 a 2.4.5).

Študenti majú možnosť použiť voliteľne jazyk Java nielen na vyššie spomenutých predmetoch, ktoré sú naňho priamo zamerané, ale aj pri riešení ďalších projektov a semestrálnych prác v ďalšom štúdiu, napríklad na predmetoch zaoberajúcich sa webovými technológiami alebo väčšími aplikáciami založenými na trojvrstvovej architektúre (viď stručný opis predmetov v 2.4.5). Tam môžu rozšíriť svoje poznatky o platformu Java EE, teda Java Enterprise Edition pre aplikácie typu klient – server, respektíve Java ME, teda Micro Edition pre prenositeľnosť aplikácie klienta na mobilné zariadenia. Toto delenie je študentom spomenuté v úvode predmetu „Technológie Java“.

Ako je bližšie opísané nižšie, náplň predmetu „Technológie Java“ úplne pokrývajú požiadavky CC2001 z 2.1.6 vo všetkých požadovaných oblastiach.

Cieľovým predmetom nasadenia integrovaných nástrojov pre podporu výučby programovania, o ktorých pojednáva táto práca, je predmet „Technológie Java“.

Predmet má v porovnaní s inými predmetmi zvyčajný rozsah, teda 2 hodiny prednášok a 2 hodiny cvičení počas 13-týždňového semestra. Priebežného hodnotenie pozostáva z písomnej previerky v druhej polovici semestra a aktívnej účasti na výučbe. Predmet je ukončený skúškou.

Z hľadiska CC2001 opísaného v 2.1.3 je zameraný tematicky na hru Minesweeper, ktorej stručný opis je uvedený nižšie a v 2.6, ale v skutočnosti poskytuje výborný náhľad na vývoj systému ako celku, pretože zdôrazňuje oddelenie reprezentácie, údajov a logiky aplikácie.

Podľa odporúčaní z 2.1.6 prednášky dôsledne ponúkajú ku každému vysvetľovanému elementu aj drobnú a jednoduchú ukážku použitia; úvodné sú zamerané na dátové štruktúry, základné triedy v duchu objektovo-orientovaného programovania; ďalšie na algoritmy, výnimky a testovanie; posledné sa týkajú používateľského rozhrania a spojenia s databázou.

Prednášky sú príkladmi v nich uvedenými aj návodom na správne riešenie zadaní z cvičení. Sú dostupné po autentifikácii a autorizácii v systéme Moodle vo forme PDF.

2. Oboznámte sa s portálom http://java.sun.com/, troma edíciami technológie Java (SE, EE, ME), zistite spôsob inštalácie prostredia Java Standard Edition.

Úloha: Aký je rozdiel medzi inštaláciou JDK, JRE a NetBeans?

3. Preložte a spustite program `HelloWorld`, zdrojový kód HelloWorld.java.

```
public class HelloWorld {
  public static void main(String[] args) {
    System.out.println("Hello World!");
  }
}
```

Program preložte v príkazovom riadku použitím prekladača `javac`. Argumentom pre prekladač je **cesta k súboru s programom** (`HelloWorld.java`).

Poznámka: Potreba rozlišovať veľké a malé písmená v ceste k súboru je závislá od platformy (MS Windows, Solaris, Linux, Mac OS).

```
c:\cvicenie01>javac HelloWorld.java
c:\cvicenie01>_
```

Úloha: Zistite aký súbor pribudol po preklade v adresári. Čo obsahuje daný súbor?

Obr. 7: Ukážka spracovania materiálov pre cvičenia

Cvičenia sú rozdelené na úvodnú časť a časť venovanú hre Minesweeper (opis viď 2.6). Materiály k cvičeniam sú prehľadne spracované vo forme web stránky, ktorá je verejne dostupná odkiaľkoľvek. V čase písania práce bola umiestnená na katedrových serveroch na adrese http://hornad.fei.tuke.sk/~poruban/java/01. Ukážka spracovania je na Obr. 7.

Úvodné stretnutie okrem organizačných pokynov zahŕňa preklad a spustenie tzv. „Hello, World!“ programu. To sa odohráva len pomocou príkazového riadku, bez asistencie vývojového prostredia.

Druhé stretnutie je určené pre zoznámenie sa s vývojovým prostredím NetBeans. Toto prostredie bude aj cieľom, do ktorého budú testovacie nástroje integrované. Ako odporúča CC2001 podľa 2.1.6, študentom sú predstavené vizuálne časti prostredia, sú tiež oboznámení so základnými klávesovými skratkami. Praktickou súčasťou je vytvorenie nového projektu, pridanie dodaných zdrojových textov, ich preklad a spustenie „Hello, World!“ programu pomocou rozhrania vývojového prostredia.

Ďalšie stretnutia sa venujú nižšie opísanej štúdii, ktorá sa skladá z modulov rozvrhnutých na jednotlivé cvičenia. Nasadenie nástrojovej podpory by malo vyučujúcemu priniesť množstvo výhod [17], hlavne automatizovať nekreatívne činnosti pedagóga, individualizovať prístup k študentovi (šikovnému i lenivému) a poskytnúť plošný, agregovaný a systematický obraz o priebežnej aktivite všetkých študentov. Technická stránka nástrojovej podpory musí umožniť oddelenie formy (vzhľadu, reprezentácie) od obsahu, zdieľanie obsahu medzi jednotlivými formami výučby,

sémantické štruktúrovanie, generovanie materiálov (z jedného zdroja pre študenta i vyučujúceho) a previazanosť častí a organizačných jednotiek aj medzi predmetmi. Záverečné hodnotenie študenta tak môže pozostávať nielen z výkonu na skúške, ale aj z hodnotenia priebežnej aktivity, množstva času venovaného štúdiu a overenia pochopenia a opravy chýb, ktorých sa počas semestra pri programovaní dopúšťal.

2.6 Štúdia Minesweeper

Opis štúdie bol prevzatý a upravený na základe bakalárskej práce, na ktorú táto diplomová práca nadväzuje [10]. Nástroje sa vyvíjajú pre všeobecné použitie v situácii vzor – porovnanie, pilotné nasadenie sa testuje na menovanej štúdii.

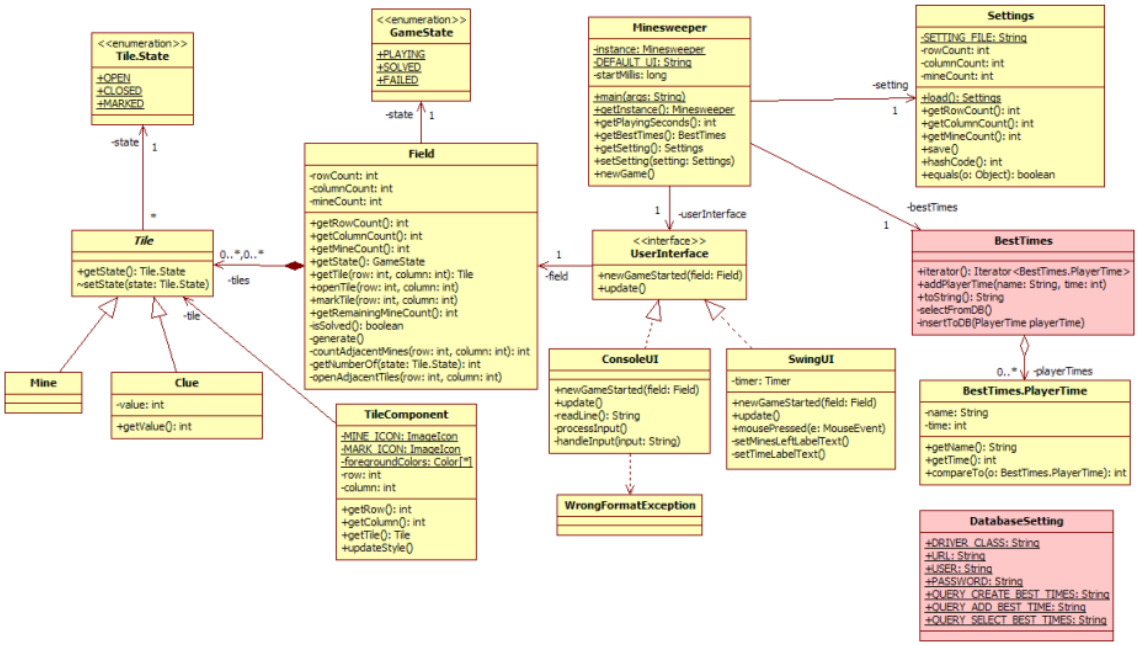
Štúdia Minesweeper je podrobne vypracovaný modulárny materiál pre výučbu študentov bakalárskeho štúdia v oblasti technológií Java. Obsahuje postupnosť krokov vedúcich študentov k vytvoreniu aplikácie, ktorá by poskytovala minimálne funkčnosť definovanú samotnou štúdiou. Navrhuje aj niektoré doplnkové úlohy pre iniciatívnych a šikovných študentov.

Obr. 8: Ukážka hry Minesweeper

Aplikácia Minesweeper je počítačová hra pre jedného hráča. Jej cieľom je správne označenie mín a voľných políčok na hracom poli, pričom sa berie do úvahy čas

a úroveň zložitosti, ktorú si volí samotný hráč. Ukážka neúspešne ukončenej hry je na Obr. 8.

Každá z nasledujúcich častí popisuje jednu úlohu v rámci tejto štúdie. Jednotlivé úlohy sú označované ako moduly s priradením poradového čísla a témy, ktorou sa zaoberajú. V každom kroku sa program postupne rozširuje pridávaním nových komponentov a funkcií.



Obr. 9: Diagram tried minimálnej predpísanej aplikácie Minesweeper

Študent má možnosť a je vyzvaný navrhovať a vytvárať aj vlastné doplňujúce komponenty a detaily programu. Niektoré nápady sú uvádzané pri jednotlivých krokoch. Doplnené komponenty nesmú narušať už vybudovanú štruktúru aplikácie – majú ju výhradne dopĺňať. Celková štruktúra aplikácie vo forme diagramu tried je na Obr. 9.

Pri každom opise modulu je zahrnutý aj stručný komentár, na čo je možné zamerať kontrolu predpísanej štruktúry. Touto kontrolou je možné overiť splnenie požiadaviek kladených na študenta k určitému týždňu semestra. Kontrola tak overí dielčiu prácu študenta na celku, ktorý bude v závere semestra odovzdávať ako funkčnú aplikáciu. Kontrola po implementácii kroku zahŕňa vždy všetky predchádzajúce testy a pridáva nové, aktuálne pre daný týždeň. Vlastné (študentom navrhnuté) doplnenia samozrejme nie je možné testovať, nakoľko predloha ich štruktúry nie je k dispozícii a vytvára si ju študent sám.

Štrukturálna kontrola dopĺňaných prvkov je postačujúca pre učenie postupu. Deklarácia elementu (triedy, metódy) obsahuje a určuje všetky jej vlastnosti podieľajúce sa na komunikácii s okolím: parametre, návratový typ, modifikátory viditeľnosti a podobne. V rámci elementu (napr. tela metódy) má však študent relatívnu voľnosť implementácie – len štruktúra aplikácie je povinná a presne definovaná. Prekladač odhalí všetky syntaktické chyby, ale nedokáže posúdiť súlad programovanej aplikácie so zadaním úlohy. Nie je to ani jeho úlohou, preto je potrebný ďalší samostatný nástroj pokrývajúci túto funkciu. Pre účely zistenia štruktúry aplikácie je vhodné použiť reflexiu (bližšie opísaná v 3.2.7).

Zámerom úvodného modulu (číslo 1) je oboznámiť študenta so všeobecnými požiadavkami na aplikáciu a dodanou vnútornou štruktúrou (balíky a deklarácie tried). Pre lepšie pochopenie sú dodané aj diagramy tried, ktoré sú súčasťou dodaných komponentov budúcej aplikácie. Cieľom testu bude overiť, či študent dodané štruktúry správne zaradil do projektu.

Študent má za úlohu v druhej časti série vytvoriť viacero metód opísaných textovou špecifikáciou. Zvyčajne je dostupné plné znenie deklarácie metódy a slovne opísané jej správne umiestnenie v štruktúre a význam, ktorý má daná metóda pre aplikáciu. Využíva sa generovanie zdrojového kódu prostredím NetBeans pomocou funkcie zapuzdrenia (*Encapsulate Fields*). Táto vygeneruje metódy nazývané *getter* a *setter* pre zvolené prvky triedy. Tieto metódy sprístupnia prvok triedy a umožnia aj jeho zmenu. Výhodou je nepriamy prístup – cez metódu a nie priamo na prvok triedy. Kontrola bude overovať prítomnosť požadovaných metód, ich modifikátory, návratové hodnoty, parametre a ich typy a ďalšie časti deklarácií.

Obsahom tretieho modulu sú rozhrania. Študentovi sú opísané požiadavky na konzolové (textové) rozhranie hry, ktoré bude poskytovať základnú funkcionalitu jadra a umožní prakticky vyskúšať implementované metódy. Študent vytvára vlastné rozhranie a na jeho základe implementuje ďalšiu logiku hry. Test má overiť správnosť deklarácie rozhrania a jeho zavedenia do potrebnej triedy projektu.

Štvrtý modul očakáva od študenta hlavne implementáciu interakcie s používateľom. Zaoberá sa rozoznávaním vstupu a správnym ukončením hry. To vyvoláva potrebu doplnenia ďalších metód podľa dodanej deklarácie a diagramu tried. Zadanie obsahuje aj prvú voliteľnú (doplňujúcu) úlohu. Testovanie bude zamerané na overenie kompletnosti predpísaného riešenia. Doplňujúce úlohy sú voliteľné a teda

nemusia byť implementované všetkými študentmi. Testovanie doplňujúcich úloh je nutné realizovať paralelne s testovaním povinných úloh, pretože sú súčasťou a vytvárajú sa súčasne a v súvislosti s povinnými úlohami. Kontrola musí byť umožnená so spoľahlivými výsledkami pre obe skupiny študentov, preto je v testovacej aplikácii nutné zohľadniť nezávislú možnosť povolenia a zakázania testovania doplňujúcich úloh.

Zámerom piatej časti je oboznámiť študenta s výnimkami, naučiť ho ich deklarovať a používať výnimky. Je potrebné pridanie dodaného zdrojového kódu triedy výnimky do projektu a dodanie ďalších metód, ktoré by zabezpečili napojenie na už hotové komponenty. Druhou časťou je vytvorenie testov (*unit test*) podľa opisu. Overenie práce študenta sa zameria na správnosť použitia výnimiek v deklarácii tried a metód. Keďže v štandardne vytvorenom projekte v prostredí NetBeans testovacie balíky nie sú pridané do skompilovaného programu, nie je možné ich testovanie v tejto fáze. Je možné v budúcnosti rozšíriť pôsobnosť testov aj na študentom vytvorené testovacie balíky, avšak vyžaduje si to upravený konfiguračný súbor. To by študentom mohlo spôsobovať komplikácie a preto sa s testovaním študentom napísaných testov v tejto fáze neráta.

Obsah tejto časti je zameraný na použitie kolekcií a zahrnutie merania času hry. Ráta sa s využitím skúseností z predchádzajúcich modulov a tak už nie sú dostupné kompletné deklarácie metód a tried, ale čiastočnú deklaráciu sprevádza slovný opis vlastností. Úlohou študenta je aj správne zahrnúť dodaný zdrojový kód pre nové triedy (*BestTimes* a *PlayerTime*). Modul vo viacerých častiach využíva znalosti z predchádzajúcich častí a poskytuje ďalšie námety na doplňujúce úlohy. Testovanie bude zamerané na správnosť čoraz rozsiahlejšej štruktúry hry Minesweeper. Okrem pridania nových metód a tried sa bude aj overovať použitie návrhového vzoru *Singleton* v hlavnej triede *Minesweeper*. Spôsob zaradenia kontroly doplňujúcich úloh bol popísaný vyššie.

V siedmom module je študent oboznámený s významom a použitím prúdov a serializácie v jazyku Java. Tieto má využiť pri ukladaní nastavení hry použitím rozhrania *Serializable*. Pre ukladanie nastavení je nutné správne vytvoriť novú triedu Settings a metódy obsluhy s väzbou na ďalšie moduly aplikácie. Zameraním testu bude kontrola pridaných tried a metód s ohľadom na potrebné parametre deklarácií.

Celá táto časť sa zaoberá grafickým rozhraním a jeho zakomponovaním do projektu spolu s ponechaním možnosti pôvodného textového rozhrania. V priebehu

realizácie modulu sa pridáva balík tried *SwingUI*, ktorý má definuje množstvo metód slúžiacich na obsluhu grafického rozhrania ovládaného myšou. Študent samostatne vytvára triedu pre nové dialógové okno a registruje obslužné programy jednotlivých elementov grafického rozhrania. Pri nedbanlivom nasledovaní pokynov je vysoká pravdepodobnosť výskytu formálnej chyby, pretože množstvo kódu namiesto študenta generuje samotné prostredie NetBeans. Testy overujú, či študent podrobne splnil požiadavky tejto úlohy a poskytujú mu priebežnú oporu pri vytváraní každého nového elementu. Dostupná je voľba otestovať projekt vrátane doplňujúcej úlohy, ktorá rozširuje hru o možnosť označenia políčka hracej plochy o otáznik (bežne môže byť neotvorené políčko neoznačené alebo označené ako mína) a doplňujúcej úlohy odkrytia okolitých políčok pri stlačení oboch tlačidiel myši súčasne.

Obsah deviatej časti je zameraný na použitie vlákien pri zaradení časovača v hre. Obsahuje aj dve doplňujúce úlohy. V testovacom programe je implementovaná kontrola voliteľnej úlohy umožňujúcej voľbu veľkosti hracieho poľa a počtu mín. Testovanie pozostáva z overenia pridaných metód s ohľadom na potrebné parametre deklarácií.

Posledná časť štúdie pozostáva z doplnenia triedy obsluhujúcej databázovú komunikáciu na základe *SQL*. Hlavná miera implementácie spočíva v pridaní metód zabezpečujúcich komunikáciu s databázovým serverom. Test v poslednom týždni je zameraný na celú hotovú hru Minesweeper, ako by bola distribuovaná konečnému používateľovi. Musí plne zodpovedať minimálnej predpísanej štruktúre podľa priebežne uvádzaných špecifikácií a môže obsahovať doplňujúce úlohy, pre ktoré sú vykonateľné testy.

2.7 Porovnanie výučby – Minesweeper a Pretória

Výhodou Minesweeper-a je kombinácia klasického prístupu (viď 2.2) a skoré zahrnutie výstavby grafického používateľského rozhrania (už v štvrtom a ôsmom module, viď 2.6). Študenti budujú najprv jadro aplikácie a tak rozumejú, čo sa ukrýva za spočiatku konzolovým a neskôr Swing.

Problémom štúdie Minesweeper je fakt, že žiaci až do štvrtého modulu nemajú spätnú väzbu k programovanému zadaniu. Túto nevýhodu je (ako neskôr uvádzame v 2.9.1 a 2.9.5) potrebné odstrániť a práve tu sa naskytá možnosť použitia nástrojov automatizovanej kontroly.

Materiály k štúdii spĺňajú odporúčania na podklady uvedené v 2.2. Faktom je, že obsahujú viac obrázkových diagramov a ukážok ako je jednotlivých krokov v moduloch. Príklady na vysvetlenie sú zahrnuté v prednáškach, v materiáloch štúdie i v samotných zdrojových kódoch, keď je pri implementácii úlohy odporúčané inšpirovať sa inou už hotovou metódou.

Spolu s prednáškovými materiálmi predmet obsahuje v dostatočnom množstve diagramy formy, tried i algoritmov. Možnosťou pre vylepšenie je viesť študentov k tvorbe vlastných neformálnych diagramov, ktoré im pomôžu v návrhovej fáze algoritmov, aby neboli len nepoužitou súčasťou dokumentácie ako varuje Judith Bishop v 2.2.

Minesweeper vedie študentov v duchu odporúčaní ku komplexnému riešeniu problému, nielen implementácii algoritmu v novom programovacom jazyku. Neposkytuje veľa voľnosti v oblasti návrhu údajových štruktúr – táto možnosť je dostupná hlavne vo forme voliteľných doplňujúcich úloh. Študent v nich môže navrhnúť vlastné prvky a im zodpovedajúce údajové štruktúry, ktoré následne použije v doplnených algoritmoch. Istou nevýhodou je požiadavka zachovať minimálnu štruktúru – na druhej strane to študentov učí prakticky používať technológiu „overloadingu“ (viacnásobné použitie názvu metódy s rozličnými parametrami).

Realizácia práce s objektmi je v súlade s poradím navrhovanom podľa náplne predmetu uvedenej v 2.2: študenti najprv v prvom týždni používajú hotové dodané triedy (viď 2.6), neskôr tvoria vlastné triedy a používajú pokročilé prvky dedenia a implementácie rozhraní (viď 2.6 od modulu 3).

2.8 Porovnanie výučby – Minesweeper a PigWorld

Ako odporúčajú tvorcovia PigWorld-u v 2.3.1, aj prístup štúdie Minesweeper sa v počiatkoch nesnaží študentom ukázať všetky možnosti jazyka, ale prezentuje jeho obmedzenú časť. Oproti PigWorld-u spočiatku dokonca neobsahuje ani grafické rozhranie, len konzolové, na ktoré sú študenti zvyknutí z programovania v jazyku C.

Myšlienka a tematika oboch štúdii je podobná: v mriežkovanom, ohraničenom svete existujú objekty, ktoré na seba vzájomne pôsobia. Grafická reprezentácia obsahuje navyše niekoľko ovládacích prvkov. Študenti majú voľnosť v obohacovaní sveta o nové i vlastné prvky. Rozdielom je zameranie PigWorld-u na interakciu medzi objektami

rovnakých tried (prasatá, políčka mapy) a Minesweeper-a na interakciu s používateľom. Bližší opis rozhrania sa nachádza v 2.6 a 2.3.4.

Systém výučby je odlišný – Minesweeper je povinnou súčasťou s viacerými povinnými a voliteľnými úlohami (viď 2.6), PigWorld je koncipovaný pre dobrovoľníkov, ktorí majú záujem o lepšiu než minimálnu známku (viď 2.3.3).

Záber prednášok je podobný (viď 2.5 a 2.3.3) – nevenuje sa samotnej štúdii, ale všeobecným vlastnostiam jazyka. Tie sú ale zoradené tak, aby sa na cvičeniach používali len už vysvetlené prvky.

Obe štúdie používajú do veľkej miery hotové dodané kódy (viď 2.5 a 2.3.5), čo študentom umožňuje najprv analyzovať dobrý vzor a až neskôr vlastnou činnosťou vytvárať nové elementy. Obidve štúdie tiež obsahujú relatívne veľký počet tried, dedenie a implementáciu rozhraní. PigWorld je bohatší o použitie abstraktných tried a širší záber práce s grafickou reprezentáciou a interakciou. PigWorld sa nezameriava na predstavenie niektorých dôležitých konštrukcií jazyka ako sú výnimky, databázové spojenie, vlákna a možnosti testovania (Minesweeper – výnimky a testovanie v module 5, vlákna v module 9 a databázy v module 10; viď 2.6). Nemá za cieľ ani bližšie zoznámiť s vývojovým prostredím (Minesweeper áno – viď 2.5).

Dramatický rozdiel je v spracovaní štúdie – pre Minesweeper sú k dispozícii štruktúrované a vizuálne príjemné on-line opisy k modulom (viď 2.6), PigWorld ponúka len jednoduchý textový dokument. Tento rozdiel je spôsobený pravdepodobne aj tým, že PigWorld je len doplnkom k ostatným vyučovacím materiálom.

Štúdia Minesweeper používa viaceré algoritmy známe z procedurálneho programovania, kým PigWorld len Dijkstrov algoritmus a algoritmus pre vyhľadanie najmenšieho prvku sekvencie. Minesweeper môže vzhľadom k zaradeniu (viď 2.4) stavať na rozsiahlejších skúsenostiach a znalostiach procedurálneho programovania, čo sa v tomto smere ukazuje ako výhoda – študenti dokážu sami vytvoriť algoritmus pre naplnenie určitej funkčnosti.

V prednáškach predmetu Technológie Java je venovaný dostatočný priestor pre vysvetlenie vzťahu základných dátových štruktúr (polia, zoznamy, kolekcie) a objektov (viď moduly 1, 2 a 6 v podkapitole 2.6).

Celkovo je možné zhodnotiť, že štúdia Minesweeper obsahuje dobré vlastnosti zahrnuté v štúdii PigWorld. Ďalej ich rozširuje o kvalitné materiály, variabilitu rôznych algoritmov a náhľadu na programovanie v jazyku Java v podobnosti s procedurálnym

programovaním (použitie vývojového prostredia, konzolové rozhranie a podobne); to aj vďaka zaradeniu po zvládnutí procedurálneho programovania v nižších jazykoch (viď 2.4).

Na druhej strane, skúsenosti z PigWorld-u s interakciou medzi inštanciami jednej triedy môžu žiakom pomôcť lepšie pochopiť objektovú paradigmu programovania a grafické rozhranie s vtipnou myšlienkou môže hneď zaujať väčšie množstvo študentov. Ako sme uviedli v 2.3.3, napriek tomu sa väčšina študentov, žiaľ, zaujíma len o prechod do ďalšieho semestra. Veríme, že je možné dosiahnuť vyššiu motiváciu aj pomocou nástrojov pre podporu výučby, ktoré môžu posunúť doterajšie hranice vo viacerých smeroch (viď 2.9).

2.9 Možnosti a vlastnosti nástrojovej podpory výučby

Pedagógovia na nižších stupňoch vzdelávania (sekundárne, primárne a dokonca i predprimárne školstvo) sa vďaka projektu Infovek postupne zžívajú s novou technikou a didaktickými pomôckami. V prostredí vysokého školstva na Slovensku máme dostatok materiálnych prostriedkov aj vďaka mnohým grantom, avšak v mnohých prípadoch sú využívané len ako náhrada techniky z dôb minulých. Ich potenciál je omnoho vyšší – sú prehliadané nové možnosti ako interaktivita, dynamické multimediálne zdroje, flexibilita, automatizácia a motivačné faktory.

2.9.1 Interaktivita výučby

Slovenský systém školstva preferujúci namemorované vedomosti nevyžaduje veľkú mieru aktivity študenta. Prevažne postačuje vhodne načasované „vypľutie“ množiny faktov bez cieľov na úrovniach porozumenia, analýzy či dokonca syntézy podľa hierarchií v súčasných taxonómiách cieľov[11].

Interaktivita vo vysokoškolskej výučbe nie je obmedzená len na cvičenia – rovnako dobre sa môže odohrávať aj na prednáškach vo forme otázok, riešenia drobných problémových úloh či diskusiách s hľadaním odpovede otázku položenú vyučujúcim.

Neustála komunikácia so študentom prináša obojstranné výhody – vyučujúci má okamžitú spätnú väzbu, vďaka ktorej môže svoj výklad doplniť alebo opakovane objasniť a študent má dojem otvoreného prostredia, čo ho povzbudzuje k aktívnemu zapojeniu sa do procesu výučby. To je dôležité zvlášť vtedy, keď má otázku a otáľa sa prihlásiť, pretože by tam bol jediný „za blbca“.

Je dokázané, že aktivizovaný študent si zapamätá viac ako pasívni poslucháči[12]. Tiež je známym faktom, že ukladanie informácií a súvislostí do dlhodobej pamäte je podmienené porozumením [13].

Nástrojová podpora pre získavanie spätnej väzby v prípade prednášok je bežne dostupná v kancelárskych balíkoch programov používaných pri prezentáciách na prednáškach (napríklad vo forme krátkeho testu s možnosťou označenia vybraných odpovedí). Je možné ďalej rozšíriť ju o ďalšie kanály a automatizované spôsoby spracovania výsledkov, napr. cez mobilné zariadenia alebo ankety v informačných systémoch.

2.9.2 Dynamické multimediálne zdroje pre výučbu

Všeobecným hlavným zdrojom informácii študentov sa stal Internet. Dostupnosť rýchleho pripojenia na univerzitách nie je problémom, takže technická infraštruktúra nie je viac bariérou vo využívaní týchto zdrojov.

Posun od statických textov a obrazov k súčasným možnostiam umožňuje napríklad priame ukážky úpravy programu vo vývojovom prostredí, jeho kompiláciu a spustenie v reálnom čase. Bohatšie zdroje umožňujú žiakom s rôznymi druhmi učenia sa lepšie porozumenie učivu [13][14].

Využitie Internetu ako univerzálneho média na prenos informácie je používané na zverejňovanie prednášok alebo odovzdávanie zadaní. Jeho možnosti ponúkajú viac – napríklad dynamickú zmenu materiálov tak, aby študentom poskytovali vždy aktuálne poznatky s dôrazom na časti podľa ich minulých chýb. Odkazovanie je možné obohatiť o zdieľanie informácií v komunitách a diskusiách. Dôležitým bodom je aj prepojenie materiálov prednášok, cvičení a ich použiteľnosť v praxi.

2.9.3 Flexibilita výučbových materiálov

Elektronické materiály ako podklad výučby sú omnoho flexibilnejšie ako klasické papierové skriptá a učebnice. Zmenu je možné vykonať rádovo v sekundách od napr. zistenia chyby alebo možnosti vylepšenia. Otáznou je miera využitia tejto možnosti. Práve v tom je výhodné použitie automatizovaných nástrojov napr. na zistenie pokrytia učiva pri jeho kontrolovaní, zdôraznenie pre študenta problematickej špecifickej časti učiva, zmenu usporiadania modulov či vizuálnej podoby materiálov, ktorá môže viac upútať alebo sa stať systematickejšou.

Integrované nástroje pre podporu výučby majú teda v takejto situácii za cieľ čo najpresnejšie identifikovať miesta a oblasti, ktoré študenti z vysvetlenia na prednáškach a opisu štúdie na cvičeniach nedostatočne pochopili, prípadne ktorým bol venovaný nedostatočný časový rozsah alebo zle zrozumiteľné príklady.

Automatizáciou a plošným vynútením si spätnej väzby získame oveľa väčšiu vzorku študentov ako pri tradičnom prístupe, kde je vyučujúci obmedzený na priamu konzultáciu so študentom. Vyučujúci by mohol demonštrovať časté chyby, problémy a ich riešenia priamo na prednáškach a do budúcna upraviť svoj výklad či materiály pre použitie v ďalších semestroch s inými študentmi.

2.9.4 Automatizácia vo výučbe

Súčasné prostriedky umožňujú automatizáciu mnohých predtým manuálne vykonávaných úkonov – od opravy jednoduchých testov až po zozbieranie a kontrolu zadaní. Automatizované nástroje v uplynulej dobe vyvinuté pre predmety „Databázové systémy“, „Operačné systémy“ a ďalšie na TUKE poukazujú aj možnosť testovania výstupov študentom vykonanej syntézy, teda nielen poznatky na nižších úrovniach zapamätania a porozumenia.

Automatizácia kontroly a spätnej väzby je mimoriadne dôležitá formatívna súčasť hodnotenia študenta počas semestra. Umožňuje mu to včasnú korekciu správania, chybného poznatku, postupu alebo postoja. Nakoľko pri veľkých množstvách študentov nie je možné poskytovať priebežnú spätnú väzbu každému individuálne počas cvičení a prednášok, je nutné maximálne využiť možnosť automatizácie tejto spätnej väzby pre zabezpečenie včasnej formatívnej nápravy chýb už na ich počiatku.

2.9.5 Motivačné faktory vo výučbe

Dôvodom pre použitie špecifických nástrojov vo výučbe je pôsobenie okamžitej spätnej väzby ako formatívneho motivačného faktoru. Študentovi aj učiteľovi ozrejmuje mieru pochopenia stanovených úloh a cieľov [14].

Rozličné uplatnenia nástrojov kontroly môžu napomôcť v prevencii podvádzania pri overovaní vedomostí alebo autenticity odovzdaného výstupu činnosti študenta.

Nástroje kontroly chýb môžu v počiatočných fázach softvérového projektu, keď ešte nie je možné overiť funkcionalitu odskúšaním, pomôcť pri identifikovaní chýb, ktorých neskoré zistenie a náprava by neskôr stála viac námahy a prostriedkov.

2.9.6 Vývojové prostredie ako nástroj podpory výučby programovania

Inteligentné nástroje podporujúce ľudskú činnosť na prostriedkoch výpočtovej techniky sú tak časté a bežné, až ich ani nevnímame. Stretávame ich v jednoduchých mobilných zariadeniach s prediktívnym doplňovaním textu pomocou slovníka T9 aj pri písaní diplomovej práce v prostredí MS Word.

Takisto môže slúžiť vývojové prostredie programovacieho jazyka ako nástroj pre podporu výučby – poukazuje na chyby, zobrazuje varovania pred možnými problémami, navrhuje optimalizáciu a ponúka ľahko dostupnú stručnú dokumentáciu. Všetky tieto vlastnosti sú bežne dostupné v moderných vývojových prostrediach ako je NetBeans, Eclipse alebo Visual Studio. Pomocou doplnkov a používateľských nastavení je možné prispôsobiť si prostredie pre individuálne použitie v určitom projekte.

Keďže študenti majú počas predmetu Technológie Java zvyčajne prvú skúsenosť s pre nich novým vývojovým prostredím NetBeans, je vhodné upraviť vývojové prostredie ako nástroj podpory pre ich vzdelávanie namiesto nich a poskytnúť im hotové riešenie, ktoré ľahko integrujú do svojho projektu.

Dôležitým prvkom pri výučbe programovania je využitie komerčne používaného prostredia NetBeans. Študenti tak získavajú návyky neskôr potrebné do praxe.

2.9.7 On-line zdroje pre podporu výučby programovania

Ako sme uviedli v 2.9.2 a 2.9.3, Internet ponúka viaceré možnosti a výhody, ako urobiť zdroje on-line dostupnými a aktuálnymi. Pre podporu výučby programovania v prostredí TUKE patria medzi online zdroje diskusné fóra k predmetu: študentské na nezávislých web stránkach a vyučujúcim moderované fórum v rámci katedrového systému Moodle.

On-line nástrojom integrovaným do vývojového prostredia (bližší popis viď 2.9.6) je aj v reálnom čase ponúkaná dokumentácia pri použití určitého prvku z tried jazyka Java. Táto je inštalovaná spolu s vývojovým prostredím aj na pevný disk používateľa, čiže nie je potrebné študovať ju vyhľadávaním na webe, je vždy prístupná z disku.

Za on-line nástroj je možné pokladať aj odkazovanie na štúdiu s úlohami a prednáškové materiály, ktoré sú poskytované študentovi v prípade potreby pri programovaní (napr. ako návod na opravu chyby alebo vylepšenie ním vytvoreného kódu).

2.9.8 Prenositeľnosť nástrojov

Pre dlhodobú a všeobecnú možnosť použitia akýchkoľvek nástrojov podpory výučby v oblasti informatiky je podstatná široká prenositeľnosť a použiteľnosť nástroja na rôznych platformách, operačných systémoch, vývojových prostrediach a viacerých počítačoch počas semestra.

Jazyk Java spĺňa kritéria prenositeľnosti, preto je vhodným pre implementáciu.

3 Nástroje pre testovanie postupu v štúdii

Jadrom tejto kapitoly je analýza a syntéza vedúca k implementácii sady integrovaných nástrojov umožňujúcich automatizovanú kontrolu postupu študenta v prípadových štúdiách pre výučby objektového programovania.

3.1 Ciele nástrojovej podpory

Podpora výučby programovania integrovanými nástrojmi má nasledovné ciele:

- automatizovaná kontrola postupu študenta v prípadových štúdiách,
- lepšie prepojenie materiálov predmetu – prednášok, cvičení, práce študenta,
- okamžitá automatizovaná spätná väzba po kompilácii (pre študenta),
- analýza možností vylepšenia vyučovacieho procesu a materiálov štúdie, analýza častých chýb študentov (pre vyučujúceho).

Cieľom automatizovanej kontroly je odbremeniť vyučujúceho od „ručnej“ kontroly postupu študentov. Len samotná kontrola ešte neprináša žiadnej zo strán užitočnú spätnú väzbu, pretože iba overuje, či študent vôbec pracoval na zadaní počas uplynulého týždňa. Vyučujúci bude mať vďaka automatizácii kontroly prehľad o činnosti všetkých študentov (nielen niekoľkých náhodných vybraných) a študenti budú nútení pravidelne a včas pracovať na zadaní. Zároveň vyučujúci bude môcť venovať svoj čas osobným konzultáciám so študentmi, riešeniu ich problémov alebo otázok.

Lepšie prepojenia materiálov môže vďaka nástrojovej podpore študentom umožniť účinnejšiu fixáciu poznatkov v dlhodobej pamäti (viď 2.9.1). Študentovi by pri výskyte chyby mala byť poskytnutá väzba na zadanie úlohy, resp. detailné pokyny k jej riešeniu.

Okamžitá spätná väzba pôsobí ako motivačný faktor (2.9.5) a umožňuje študentovi uvedomiť si a opraviť si chybu čo najskôr – to tvorí dôležitú časť formatívneho hodnotenia (2.9.4). Prináša tiež do výučby interaktivitu, ktorá aktivizuje jednotlivca (2.9.1).

Hromadná automatizovaná spätná väzba získaná z vývoja programu umožňuje vyučujúcemu identifikovať problémové miesta. Tie môže lepšie vysvetliť alebo upraviť zadanie úlohy na zrozumiteľnejšie alebo jednoduchšie, prípadne presunúť modul úlohy na vhodnejšie miesto v rámci série (2.9.3).

3.2 Požadované vlastnosti

Požiadavky kladené na vývoj a implementáciu nástrojov v nasledovných oblastiach v záujme splnenia vyššie uvedených cieľov (3.1) a v rámci možností opísaných v 2.9 sú nasledovné:

- miesto nasadenia,
- forma práce – on-line, off-line,
- zložky a časti sady nástrojov,
- jednoduchosť konfigurácie,
- rozhrania a komunikácia,
- umiestnenie nástrojov, ich integrácia,
- reprezentácia výsledkov,
- univerzálne použitie,
- lepšie prepojenie prednášok, materiálov cvičení a práce študenta,
- overenie postupu na základe štruktúry.

3.2.1 Miesto nasadenia

Cieľovým predmetom testovacieho nasadenia testovacieho nástroja je predmet Technológie Java. Tento zahŕňa okrem úvodných hodín zoznámenia sa s vývojovým prostredím NetBeans 10 úloh – modulov, rozvrhnutých do desiatich týždňov (chronológia je opísaná v 2.6). V prvom module je dodaná študentovi základná štruktúra vo forme projektu pre prostredie NetBeans – voľba tohto prostredia vychádza z 2.9.6. Počas každého týždňa má študent doplniť nejakú časť do programu podľa detailne vypracovaného návodu. Cieľom je vytvorenie funkčnej hry Minesweeper, ktorej rozsiahlejší opis je v časti 2.6. Nástroj môže byť použitý aj pre iné predmety v obdobnej situácii s hotovým vzorom pre porovnanie s riešením študenta.

Čiastkovú reálnu funkčnosť zadania môže študent sledovať približne od polovice semestra. Počas prvých týždňov je teda spätná väzba obmedzená na chyby a varovania kompilátora, resp. manuálnu kontrolu vyučujúcim. Keďže prakticky nie je možné z časových dôvodov kontrolovať postup, správnosť kódu a problémy u každého študenta individuálne, je vhodné ponúknuť študentovi aspoň automatizovanú spätnú väzbu vo forme štrukturálnej kontroly (opis v 2.6). Ako sme uviedli v 2.9.1, 2.9.4 a

2.9.5, táto spätná väzba môže významne prispieť ako motivačný faktor, zdroj lepšieho porozumenia a uvedomenia si prepojenia teórie z prednášok a praxe.

Pre tento účel je podľa 2.9.6 vhodné integrovať do projektu prostredia NetBeans testovaciu aplikáciu. Tá po každej úspešnej kompilácii, teda pre syntakticky i sémanticky kompilátorom akceptovateľné zdrojové kódy, vráti študentovi výsledky kontroly napísaných častí kódu voči vzorovému riešeniu.

3.2.2 Forma práce nástrojov

V praktickej výučbe sa testovanie študentských zadaní realizuje dvomi spôsobmi: on-line a off-line. On-line kontroly sú zvyčajne riešené cez katedrový systém Moodle. Ten má integrovanú podporu pre ukladanie študentských zadaní a pomocou doplnkov pre tento systém je možné zadania aj testovať. Veľkou pomocou je zníženie administratívy oproti možnosti zbierať študentské zadania formou emailu či dokonca zozbieraním z USB kľúčov na cvičení a podobne.

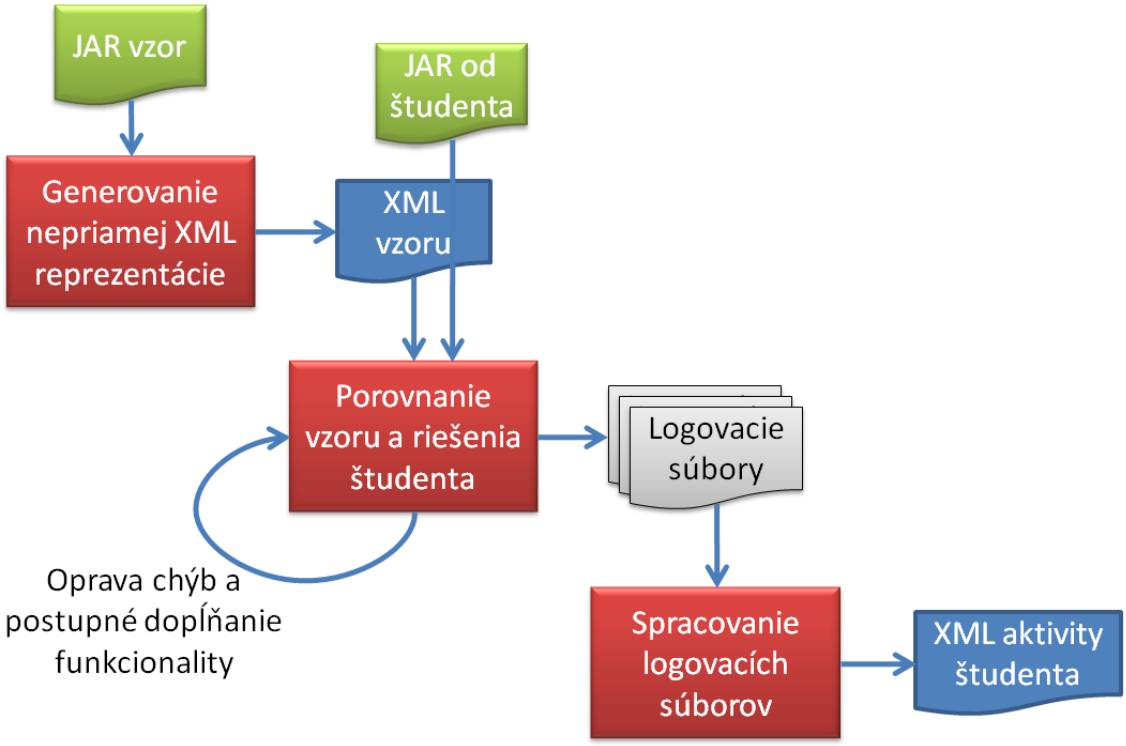
Off-line kontrola je obyčajne obmedzená na overenie funkcionality tvorenej aplikácie, prípadne na jej schopnosť komunikovať s knižnicou alebo rozhraním dodaným treťou stranou, teda zvyčajne cvičiacim, respektíve vyučujúcim.

Podľa cieľa okamžitej automatizovanej spätnej väzby stanoveného v 3.1 a z ďalších technických príčin (rýchlosť, odozva, spoľahlivosť, možnosť pracovať bez internetového pripojenia, osobitné spoplatnenie internetového pripojenia) je vhodné použiť off-line formu práce testovacieho nástroja. Zároveň je potrebné koncipovať ho takým spôsobom, aby bolo v budúcnosti možné jeho nasadenie vo forme webovej služby alebo prepojenie do existujúceho systému Moodle. Off-line kontrola podporuje myšlienku interaktivity opísanú v 2.9.1 a pôsobí tiež ako motivačný faktor (viď 2.9.5).

Ideálnym stavom, ku ktorému je potrebné perspektívne spieť, je prepojenie on-line dostupných materiálov (opísané v 2.9.7) a off-line práce študenta vo vývojovom prostredí (opísané v 2.9.6).

3.2.3 Zložky, časti a umiestnenia sady nástrojov

Z dôvodu rizika, že študenti by hotové vzorové riešenie mohli zneužiť skopírovaním častí zdrojových kódov pomocou metód spätného inžinierstva, nie je vhodné poskytnúť im priamo toto riešenie. Pre účely porovnania štruktúry vzoru a študentského riešenia postačuje nepriama reprezentácia vzorového riešenia a teda vzorové riešenie nie je súčasťou sady nástrojov.



Obr. 10: Náčrt toku informácií medzi nástrojmi

Obr. 10 zobrazuje nadväznosť jednotlivých nástrojov v sade.

Prvým elementom sady je aplikácia vytvárajúca nepriamu reprezentáciu vzorového riešenia. Táto je daná k dispozícii len vyučujúcemu, keďže je naviazaná na vzorové riešenie.

Ďalším elementom je aplikácia porovnávajúca nepriamu reprezentáciu a študentské riešenie. Táto je daná študentovi integrovaná v projekte, ktorý počas semestra rozvíja. Výstup jej práce je zapisovaný pravidelne pri každej kompilácii do logovacieho súboru.

Tretím elementom sady je aplikácia, ktorá vyhodnocuje prácu študenta reprezentovanú logovacími súbormi a transformuje ju do ľudsky lepšie zrozumiteľnej podoby. Môže byť súčasťou projektu daného študentovi, v budúcnosti je možné rozšíriť ju o integráciu do systému Moodle ako webovú službu.

Ďalšími elementmi v možnom rozvoji aplikácie je webová služba. Vždy on-line dostupná študentom bude používať na serveri nasadené aplikácie a výstup kontrol bude zapisovať do systému Moodle, ktorý je dlhodobo nasadený v rámci výučby a je predpoklad jeho ďalšieho používania.

Vzhľadom k zvolenej platforme Java bol v duchu prenositeľnosti podľa 2.9.8 použitý efektívny a všeobecne dostupný štandardný prostriedok Java Properties (*java.util.Properties*). Hodnoty sú uchovávané v textovom súbore, ktorý je ľahko editovateľný a zrozumiteľný.

Od študenta je v úvode použitia porovnávacieho nástroja vyžadované zapísanie jeho unikátneho identifikátora v rámci TUKE práve do konfiguračného súboru tohto typu. Tiež je pomocou toho istého súboru definované, vzhľadom ku ktorému modulu sa riešenie kontroluje.

Praktickou výhodou pre jednoduchosť budúcich úprav aplikácie je možnosť použiť jednotný konfiguračný súbor pre všetky nástroje.

Nakoľko sa testovací nástroj zaoberá analýzou skompilovaného jar súboru, je možné jeho spúšťanie až po úspešnej kompilácii študentského programu. To je možné dosiahnuť doplnením konfigurácie známeho nástroja Apache Ant (z dôvodov opísaných v 2.9.6), ktorý v prostredí NetBeans riadi procesy spojené s kompiláciou aplikácie.

Študentovi dostupný nástroj porovnania so vzorom je potrebné integrovať. Prináša to praktickosť použitia, automatické spúšťanie a elimináciu chýb pri nesprávnom použití.

Ako sme opísali v 2.9.6, vývojové prostredie už niektoré nástroje podpory programovania priamo obsahuje a tak je nástroj vhodné integrovať priamo do prostredia aplikácie – projektu vývojového prostredia NetBeans, ktoré sa používa pri výučbe. Výhodou je, že výstup bude priamo zobrazovaný v štandardnej konzole výstupu kompilácie. Študent si tak po každej úspešnej kompilácii prečíta aj správu porovnávacieho testovacieho nástroja.

V budúcnosti je pre dosiahnutie lepšej prenositeľnosti podľa 2.9.8 ľahko možné aplikáciu samostatne použiť aj pre projekty vyvíjané v prostredí Eclipse alebo inom. Je pre to potrebné doplniť spúšťanie aplikácie a parametrov pre nástroj Apache Maven alebo iný, ktorý sa v prostredí a danom projekte používa.

3.2.4 Rozhrania a formáty súborov komunikácie

Výmena údajov medzi nástrojmi sa uskutočňuje pomocou textových súborov. Rozhranie s používateľom je realizované pomocou konfiguračných súborov a presmerovateľných výstupov (štandardne nastavené na konzolu *System.err*). Programy

neponúkajú žiadne GUI ani konzolové interaktívne rozhranie kvôli predpokladu webového nasadenia.

Pre všeobecnosť a rozsah použitia a množstvo nástrojov pre spracovanie (kritérium opísané v 2.9.6), dobrú prenositeľnosť (kritérium opísané v 2.9.8) je zvolený rozšírený súborový formát XML, ktorý poskytuje všetky potrebné možnosti a je znovu použiteľný aj pre iné zámery. Výhodou je aj možnosť dynamických zmien jednotlivých nástrojov sady (viď 2.9.2 a 2.9.3). Súčasťou definovania formátu XML je súbor XSD pre definovanie jeho štruktúry a vnútorných elementov. Tiež môže slúžiť ako kontrolný nástroj medzi jednotlivými členmi sady, ktoré si odovzdávajú touto formou zapísané informácie.

Formát logovacieho súboru je interne definovaný a hodnoty v ňom sú zapísané v textovej podobe. Neskôr je možné jeho rozšírenie o utajenie (šifrovanie) pre zamedzenie snáh o zmenu študentmi. Formát v hlavičke definuje identifikáciu študenta, identifikáciu počítača, čas kompilácie a v tele pre každý riadok názov chyby, jej doterajší výskyt a číslo modulu, vzhľadom ku ktorému bol spúšťaný test.

Výsledok práce nástrojov je výhradne vo formátoch opísaných v tejto časti. Formát XML môže byť pre lepšiu vizuálnu zrozumiteľnosť človeku obohatený o definovanie vzhľadu pomocou CSS ako odporúčame v časti 2.9.2.

Štruktúra XML súborov je dynamicky generovaná na základe ľahko zmeniteľného poradia alebo povolenia testovania jednotlivých vlastností. Je tiež možné meniť názvy značiek (tagov) v konfiguračných súboroch.

3.2.5 Univerzálne použitie na rôznych platformách a predmetoch

Akademické prostredie, nezávislosť na komerčných platformách a hlavne vyučovaný programovací jazyk prirodzene viedli pre implementáciu nástrojov k jazyku Java. Vďaka nemu je možné spúšťať testovacie nástroje všade tam, kde je možné spúšťať programy napísané v jazyku Java, resp. všade tam, kde je k dispozícii konfiguračný nástroj Apache Ant. Zabezpečí sa tak výborná požadovaná prenositeľnosť (2.9.8).

V rámci výučby je jazyk Java preberaný aj na predmete „Objektové programovanie“ a tak je celkové riešenie koncipované s dôrazom na univerzálnosť a možné použitie aj pre iné predmety, ktoré majú predpísanú vnútornú štruktúru,

napríklad rozhrania, názvy a parametre metód a tried. Riešenie je viazané na programy v jazyku Java vyvíjané za pomoci nástroja Ant.

3.2.6 Prepojenie foriem výučby

Ako sme opísali v 2.9.3 a stanovili za cieľ prepojenie študijných materiálov (3.1), spätná väzba získaná nástrojom umožní skvalitniť materiály štúdie pre cvičenia v nejasných častiach; tiež má viesť prednášajúceho k lepšiemu vysvetleniu problematických častí učiva.

Každé úloha v materiáloch k cvičeniam je označená XML opise, z ktorého je neskôr generovaná web stránka (HTML kód). Je identifikovaná číslom modulu a názvom úlohy. Číslo modulu je súčasťou URL adresy, názov úlohy je definovaný ako kotva (anchor) v rámci HTML súboru.

Zhodná identifikácia je používaná nástrojom pre odkazovanie študenta pri výskyte chyby. Chybové hlásenie obsahuje popis chyby a URL adresu na konkrétnu úlohu v rámci materiálov štúdie, kde študent nájde inšpiráciu pre opravu chyby. Zároveň to umožňuje prepojiť výskyt chýb, resp. návštevnosť URL adries na stránke, s analýzou návštevnosti stránky.

Prepojenie s materiálmi a obsahom prednášok je zatiaľ ponechané na individuálnej práci prednášajúceho.

Celkové modulárne usporiadanie predmetu, prednášok i štúdie vyžaduje zachovanie modularity aj pri testovacích nástrojoch. Tie sú ovplyvnené konfiguračnými súbormi (viď 3.2.3) a implementáciou vzorového riešenia. Pri jeho zmene postačí v okamihu znovu vygenerovať jeho nepriamu reprezentáciu a skopírovať nový súbor do projektu poskytovaného študentom. Podporuje to dobrú flexibilitu, ktorú očakávame na základe argumentov z časti 2.9.3.

3.2.7 Overenie postupu na základe štruktúry

Ako sme opísali v 2.6, postup študenta pri programovaní aplikácie počas semestra je overovaný podľa prítomnosti štrukturálnych prvkov aplikácie, ktoré majú definovaný týždeň (modul), ku ktorému majú byť hotové.

Vzhľadom k stanovenému miestu nasadenia porovnávacieho nástroja (podľa 3.2.3) a voľbe programovacieho jazyka (podľa 3.2.5) je vhodnou technológiou zisťovania štruktúry Java reflexia nad skompilovaným JAR súborom. Reflexia je schopnosť

bežiaceho programu skúmať samého seba a svoje softvérové prostredie a zmeniť svoje správanie na základe získaných informácií [16].

Jednotným spôsobom je takto možné zistiť štruktúru vzorového riešenia a zároveň aj študentského riešenia – reflexia by v prípade štrukturálne zhodného zdrojového kódu mala dávať rovnaké výsledky pre obe riešenia.

3.3 Nástroj tvorby nepriamej reprezentácie vzorového riešenia

Ako je uvedené v časti 3.2.1, pre štúdiu Minesweeper je vytvorené vzorové riešenie. Študentovi nemá byť prístupné v žiadnej forme z dôvodov uvedených v 3.2.3, preto je zvolená nepriama reprezentácia XML súborom (viď 3.2.4). Tento súbor má štruktúru zodpovedajúcu štruktúre zistiteľnej z JAR súboru pomocou reflexie, ktorá je hlavným nástrojom pre automatizovanú tvorbu nepriamej reprezentácie (3.2.7).

Samotný nástroj nenapĺňa žiaden z cieľov stanovených časťou 3.1, pretože je len jedným článkom sady a bez ostatných nemá veľký význam. Dáva však dobrý základ pre naplnenie stanovených cieľov previazaním vzorového riešenia na webovú štúdiu pre študentov. Vďaka automatizácii generovania XML reprezentácie je jednoduché robiť v štúdii priebežné zmeny a je možné jej ďalšie strojové spracovanie.

3.3.1 Anotácie označenia vzorového riešenia

Anotácia je špeciálna forma zápisu meta-údajov v zdrojovom kóde. Je to modifikátor obsahujúci meno anotačného typu a žiaden, jeden alebo viacero párov položka (kľúč) – hodnota, z ktorých každý je priradením určitej hodnoty jednotlivým položkám.

Anotácie môžu byť použité ako modifikátory ľubovoľnej deklarácie vrátane deklarácie balíka možné priradiť akémukoľvek elementu, ktorému je možné priradiť modifikátory (balíčky, triedy, rozhrania, premenné triedy, metódy, parametre, konštruktory, lokálne premenné, enumerácie a aj typy anotácií). Práve tieto elementy a ich vlastnosti je možné identifikovať reflexiou, pomocou ktorej budeme vytvárať nepriamu reprezentáciu JAR vzoru.

Anotácie sa vytvárajú pomocou deklarovania typu anotácie. Jazyk Java ich rozoznáva ako špeciálny typ rozhrania, na rozlíšenie je použitý znak @ pred slovom *interface*. Ich deklarácia môže byť umiestnená na miestach, kde sú prípustné deklarácie rozhraní a tried a zdieľajú aj ďalšie ich vlastnosti (viditeľnosť a podobne).

Pri deklarácii typu anotácie je možné použiť meta-anotácie na bližšie určenie vlastností vytváraného typu anotácie. V prípade reflexie nad vzorovým riešením je potrebné zvoliť zotrvanie anotácií v zdrojovom kóde až do času behu programu, teda aj po kompilácii.

Anotácia navrhnutá pre popis vzorového riešenia má nasledovné zložky:

- názov modulu, údajový typ String, bez východzej návratovej hodnoty,
- textový opis úlohy v rámci modulu, údajový typ String, východzia hodnota je prázdny reťazec,
- prepínač možnosti dedenia, údajový typ boolean, východzia hodnota „pravda“ (true).

Implementácie anotácie podľa vyššie uvedeného návrhu je teda nasledovná:

```
@Retention(RetentionPolicy.RUNTIME)
public @interface Task {
    String module();
    String id() default "";
    boolean inherit() default true;
}
```

Anotácie budú aplikované priamo do vzorového riešenia v textovej forme.

Module identifikuje číslo modulu v rámci štúdie, resp. voliteľné štruktúry doplňujúcich úloh. Číslo modulu je súčasne časť URL adresy cvičenia na webe.

V rámci modulu je potrebné identifikovať jednotlivé časti úlohy, keďže niektoré sú zložené z viacerých krokov. Pokiaľ bude vrátený prázdny reťazec, použije sa ako identifikátor názov elementu, teda napr. jednoduchý názov metódy alebo triedy (metóda *getSimpleName()*). Zhodný identifikátor je použitý vo web stránkach ako kotva (anchor) v HTML kóde. Pomocou neho je potom možné študentovi poukázať priamo na časť úlohy, kde sa vyskytla chyba.

Dedenie umožňuje povoliť alebo zakázať získanie anotácie nižšími elementmi od vyššieho (napr. metóda použije anotáciu nadradenej triedy). Zákaz dedenia má zmysel hlavne pri triedach používajúcich knižnice grafického rozhrania Swing. Tieto triedy často obsahujú metódy, ktoré nie sú predmetom štúdie a nemusia tak byť kontrolované.

3.3.2 Štruktúra nástroja

Nástroj pozostáva z definície anotácie uvedenej v predošlej časti 3.3.1 (balík annotation s jedinou triedou Task), častí načítavania vstupov a zapisovania výstupov (balíky core, examining a export) a časti reflexie (balík testing).

Balík core obsahuje jedinú triedy Main zodpovednú za inicializáciu všetkých premenných a testovacími triedami používaných generických testovacích objektov, tiež za overenie prístupnosti vzorového riešenia a výstupnej cesty pre XML súbor.

Balík examining obsahuje jedinú triedu ExaminedJarFile, ktorá udržiava informácie o skúmanom JAR súbore a poskytuje metódy vyhľadania triedy podľa jej mena.

Balík export obsahuje triedy XmlExporter a XmlParagraph. XmlExporter zapíše hlavičku XML súboru s meta-údajmi o teste, zozbiera a zoradí výstupy testov z triedy, konštruktorov a metód, označí ich hlavičkami a zapíše ich do súboru. XmlParagraph reprezentuje úsek jednej triedy z budúceho XML súboru. V rámci neho sú časti pre výstup z testu triedy, konštruktorov a metód. V XML výstupe sú zapísané len tie časti s booleovskou hodnotou, ktorých hodnota je „pravda“ (true).

Balík testing obsahuje päť tried, po jednej pre test triedy, metódy a konštruktora, triedu kontroly modifikátorov ModifiersChecker a triedu generického testovania vlastností GenericChecker.

Trieda GenericChecker obsahuje metódy spoločné pre triedy, konštruktory i metódy. Prvou je test vlastnosti, ktorého návratová hodnota je pole tried. Druhou je test pre vlastnosti, ktorého návratovou hodnotou je reťazec (typ String). Zvyšné metódy získavajú pre element modifikátory použijúc triedu ModifiersChecker a anotáciu Task.

Nástroj obsahuje rozsiahly konfiguračný súbor, ktorý definuje všetky vlastnosti, názvy XML značiek (tagov) a chybové hlásenia. Je tiež možné vo forme regulárneho výrazu zadefinovať, pre ktoré metódy nebude generované chybové hlásenie (zavedené z dôvodov opísaných v 3.2.7).

```
<isPublic>
  true
</isPublic>
<constructors>
  <constructor>
    <getName>
      minesweeper.core.Clue
    </getName>
    <toGenericString>
      public minesweeper.core.Clue(int)
    </toGenericString>
    <module>
      01
    </module>
    <id>
      src
    </id>
    <inherit>
      true
    </inherit>
    <exceptionsAmount>
      0
    </exceptionsAmount>
    <parametersAmount>
      1
    </parametersAmount>
    <parametersNamed>
      <parameter>
        int
      </parameter>
    </parametersNamed>
    <isPublic>
      true
    </isPublic>
  </constructor>
</constructors>
<methods>
  <method>
    <getName>
      getValue
```


Obr. 11: Ukážka štruktúry XML súbor, časť konštruktory triedy Clue

3.3.3 Práca a výstup nástroja

Pri spustení aplikácie je vyhľadaný vyššie opísaný konfiguračný súbor. Ďalej sú podľa jeho obsahu inicializované cesty k vstupnému JAR súboru, výstupnému XML súboru, chybovému výstupu a podobne.

Po úspešnom zavedení je spustené vyhľadanie JAR záznamov vo vstupnom súbore a pre každý záznam triedy je spustené testovanie triedy. To obsahuje testovanie nasledovných vlastností (identifikovaných názvom metódy alebo slovným opisom):

- getSimpleName (jednoduchý názov triedy),
- getCanonicalName (názov triedy aj s balíkmi),
- zistenie anotácií typu Task (modul, identifikátor v rámci modulu, dedenie),
- deklarované podtriedy,

- triedou implementované rozhrania,
- členy „enum“-ov, ak je trieda „enum“ (vymenovanie prvkov),
- deklarované vnútorné členy triedy,
- modifikátory viditeľnosti a iné modifikátory (isAbstract, isFinal, isInterface, isNative, isStatic, isStrict, isSynchronized, isTransient, isVolatile).

Test sa zaoberá len deklarovanými elementmi, nie zdedenými, pretože študenti dopĺňajú len deklarované členy a nie vzťahy dedenia. Ďalej testovanie triedy spustí testovanie všetkých deklarovaných konštruktov a metód v nej obsiahnutých.

Test konštruktora a metódy zahŕňa overenie nasledovných vlastností (identifikovaných takisto názvom metódy alebo slovným popisom):

- getName (jednoduchý názov),
- toGenericString (názov s parametrami, výnimkami a pod.),
- zistenie anotácií typu Task (modul, identifikátor v rámci modulu, dedenie),
- deklarované výnimky,
- parametre,
- modifikátory viditeľnosti a iné modifikátory (isAbstract, isFinal, isInterface, isNative, isStatic, isStrict, isSynchronized, isTransient, isVolatile, isAccessible, isVarArgs).

Triedy, ktoré zabezpečujú testovanie metód a konštruktorov, obsahujú len metódu pre zistenie anotácie elementu, resp. nadradeného elementu. Pre samotné testovanie používajú triedu GenericChecker.

V prípade, že element nemá anotáciu modulu, je vypísané na chybový výstup hlásenie o mieste a type chyby. Generovanie XML v prípade výskytu chyby pokračuje ďalej, chybové miesto je označené. Vyučujúci tak môže doplniť anotáciu na danom elemente a nechať XML vygenerovať znovu. Aplikácia má v konfiguračnom súbore vo forme regulárneho výrazu možnosť zadefinovania, pre ktoré metódy nebude generované chybové hlásenie (zavedené z dôvodov opísaných v 3.2.7).

Zostručnená základná hierarchia XML značiek je nasledovná:

```
<meta-údaje hlavičky>
<vymenovanie tried>
    <informácie o triede>
    <informácie o anotáciách>
    <počet deklarovaných rozhraní, tried, polí triedy>
        <vymenovanie elementov>
    <značka konštruktorov>
        <vymenovanie konštruktorov>
            <informácie o konštruktore>
            <informácie o anotáciách>
            <počet deklarovaných výnimiek>
                <vymenovanie výnimiek>
            <počet deklarovaných parametrov>
                <vymenovanie parametrov>
    <značka metód>
        <vymenovanie metód>
            <informácie o metóde>
            <informácie o anotáciách>
            <počet deklarovaných výnimiek>
                <vymenovanie výnimiek>
            <počet deklarovaných parametrov>
                <vymenovanie parametrov>
            <návratový typ>
```

Úspešne vygenerovaný XML súbor je k dispozícii v rámci Príloha A: , resp. jeho časť je ako ukážka na Obr. 11.

3.4 Nástroj porovnania študentského zadania a vzoru

Kľúčovým nástrojom z pohľadu študenta je aplikácia sprostredkujúca spätnú väzbu po kontrole študentom vyvíjaného projektu. Vyhľadá štrukturálne chyby, vypíše ich zoznam a ponúkne odkazy na časti štúdie, podľa ktorých si môže študent chybu opraviť. Kontrola prebieha vzhľadom k určitému modulu, resp. týždňu v semestri.

Nástroj pre porovnanie na vstupe pracuje s nepriamou reprezentáciou vzorového zadania vo forme XML súboru (bližší opis v predchádzajúcej podkapitole 3.3). Tá obsahuje všetky informácie o povinných elementoch štruktúry.

Porovnanie so študentským riešením prebehne nasledovne: postupným prechádzaním elementov z XML sa reflexiou (viď 3.2.7) vyhľadávajú odpovedajúce elementy študentského riešenia. Pokiaľ sú zdrojové kódy štrukturálne zhodné, reflexia zapísaná v XML a reflexia nad študentským riešením bude dávať rovnaké výsledky.

Každý rozdiel je študentovi vypísaný do konzoly prostredia NetBeans a takisto je poznačený aj do logu (denníka). Tento je podľa stanovených cieľov (3.1) neskôr možné spracovať pre vyhodnotenie priebežnej aktivity študenta aj pre zistenie štatistickej chybovosti v jednotlivých častiach štúdie.

3.4.1 Štruktúra a práca nástroja

V balíku tester sa nachádza trieda Main zodpovedná za inicializáciu všetkých ciest, premenných, vstupného XML súboru (opis vyššie v 3.3.3), konfiguračného súboru, výstupného logovacieho súboru a DOM (Document Object Model). Načítava aj študentom zvolené nastavenie modulu a zadaný identifikátor. Činnosť metódy *main* končí odovzdaním riadenia novému objektu triedy DOM. V prípade neúspešnosti niektorej z kritických činností zavádzania aplikácie sa ďalej v spúšťaní nepokračuje a aplikácia končí s opisom chyby (napr. nebol nájdený súbor s nepriamou reprezentáciou vzoru).

Balík tester.comparator obsahuje triedy testovania, triedu pre prácu s anotáciami a spomenutú triedu DOM. Jej implementácia používa a vychádza z balíka javax.xml.parsers, ktorý obsahuje nástroje pre prácu s XML súbormi. Obsahuje metódy pre získanie opisu tried, konštruktorov a metód z XML reprezentácie.

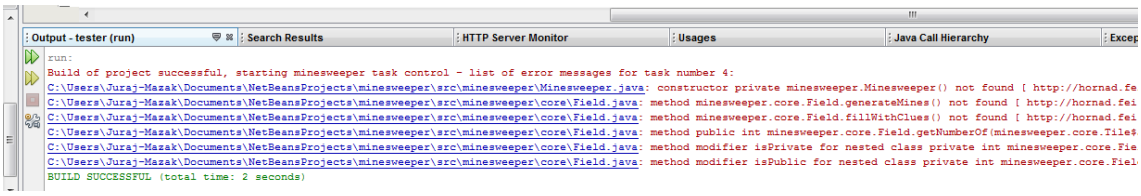
Triedy testovania vlastností porovnávajú hodnoty získané z DOM s hodnotami získanými reflexiou nad študentským riešením, teda v štandardnom režime vlastnosti opísané v časti 3.3.3. Pre každú testovanú triedu sú volané testy konštruktorov a metód.

XML reprezentácia vzoru získaná reflexiou vzhľadom k princípu práce reflexie neobsahuje informácie o vnorených triedach (nested class). Je teda nutné s týmito narábať špeciálnym spôsobom a metódy porovnávania tried preto vo väčšine prípadov obsahujú vetvenie na vnorené a obyčajné triedy.

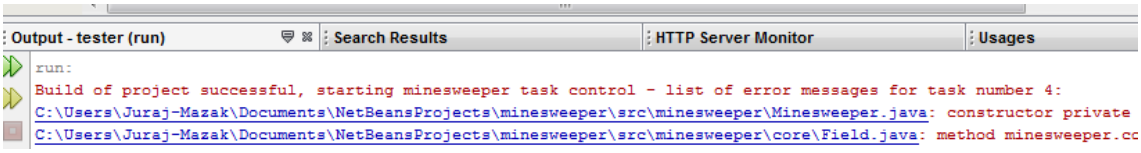
Pred prvým spustením testu je študent vyzvaný zadať svoj identifikátor. Zmena identifikátora a zmena modulu, voči ktorému spúšťame testy, sa robí v textovom súbore. Tento súbor neobsahuje iné hodnoty, je súčasťou dodaného projektu a študentom je prístupný na úpravu obvyklým otvorením napr. v Notepad-e alebo GEdit-e. Štruktúra, ako je opísaná, je nasledovná:

```
minesweeper_task=4
student_id=ab123ce
```

Ak študent zadá ako číslo testovaného modulu nulu, testovanie je vypnuté. Pri zadaní čísla, ktoré nespadá do rozsahu štúdie (aktuálne sú moduly očíslované číslami 1 až 10) je vypísaná chyba. Korektná identifikácia modulu napr. číslom 4 znamená, že budú spustené testy pre prvý až štvrtý modul – testy sú vykonávané vždy aj spätne.



Obr. 12: Ukážka výpisu chýb kontroly v prostredí NetBeans



Obr. 13: Detail ukážky chyby – URL odkaz na triedu s chybou

V prípade výskytu rozdielu porovnávaného študentského riešenia a vzorovej štruktúry je volaná metóda, ktorá zabezpečí výpisy do konzoly a do logovacieho súboru.

Výpis v konzole prostredia NetBeans nasleduje po výpise štandardnej kompilácie – kontrola sa spúšťa len v prípade úspešnej kompilácie bez chýb. Ukážka výpisu chýb k štvrtému Minesweeper modulu je na Obr. 12, detail na Obr. 13.

Modrou farbou označené texty sú odkazy, na ktoré je možné kliknúť a študenta hneď presmerujú na triedu, v ktorej sa chyba vyskytla. Nasledujúci červený text stručne opisuje chybu v angličtine. V prípade potreby je samozrejme možná zmena jazyka a hlásení pomocou konfiguračného súboru. V zátvorkách za opisom je odkaz na štúdiu, kde sa vyskytuje opis modulu, resp. jeho časti. Odkaz je tvorený hlavnou časťou odkazu na štúdiu, teda umiestnením na webe na serveroch KPI, konkrétne v čas písania štúdie na adrese `http://hornad.fei.tuke.sk/~poruban/java/` s pridaním čísla modulu a pridaním kotvy (anchor) v rámci stránky. Pomocou tejto kotvy je možné študentovi presne v rámci web stránky ukázať na chybne splnenú či nesplnenú časť úlohy. Celkovo teda URL adresa vyzerá napríklad takto:

```
http://hornad.fei.tuke.sk/~poruban/java/04/index.html#generateMines
```

Na odkazy na web nie je priamo kliknúť s otvorením v prehliadači, na odkazy v rámci tried áno. Je to spôsobené narábaním prostredia NetBeans s URL odkazmi. Je možné riešiť to doplnkovým modulom (plug-in), čo ale nie je v súlade s požiadavkou prenositeľnosti uvedenej v 2.9.8 a požiadavkou platformovej univerzálnosti z 3.2.5. V prípade doplnku do prostredia NetBeans by ho študenti museli mať vždy a všade nainštalovaný, čo je niekedy komplikované až nemožné (napr. z dôvodu nedostatočným oprávnení používateľského konta). Je možné, že v budúcnosti bude do štandardného prostredia zahrnuté aj rozpoznávanie URL odkazov na web stránky.

3.4.2 Štruktúra súborov logu

Výpis chyby do logovacieho súboru zabezpečuje balík tester.logger s triedami LogManager, LogParser a LogFileFilter. Trieda LogFileFilter slúži len ako implementácia triedy java.io.FileFilter pre filtrovanie obsahu adresára. Trieda LogParser má jedinú zvonku prístupnú metódu getOccurance, ktorá v súbore (parameter metódy) vyhľadá počet výskytov chybového hlásenia, ktoré je tiež parametrom tejto metódy.

Najdôležitejšou triedou je LogManager, ktorý v konštruktore otvára nový súbor, zapisuje hlavičku s meta-údajmi a do tela súboru všetky chyby, ktoré sa v projekte vyskytli. Keďže sme zvolili jednoduchý textový formát, hlavička je reprezentovaná len názvom súboru a každý riadok obsahu znamená jeden výskyt chyby.

Hlavička (názov súboru) obsahuje v poradí nasledovné elementy oddelené podčiarkovníkom:

- unikátny identifikátor študenta v rámci TUKE, ktorý je študent vyzvaný zadať do konfiguračného súboru pred prvým spustením porovnávacieho nástroja
- identifikáciu počítača vo forme MAC adresy, resp. pre počítače bez sieťového pripojenia je použitá náhrada „unknown“
- čas spustenia kompilácie, resp. zápisu chýb do súboru; v tvare zvyčajnej reprezentácie času hlavne na unixových operačných systémoch, teda ako počet sekúnd od 1. januára 1970
- súborová prípona (.jlog – na „zmätenie nepriateľa“, teda študenta, ktorý by sa zaoberal viac testovacími nástrojmi ako samotnými úlohami v Minesweeper-i)

Ukážkou názvu súboru teda môže byť napríklad:

```
ab123ce_0021005378FE_1298990107238.jlog
```

Obsah súboru je zložený z riadkov, kde každý riadok reprezentuje jedno chybové hlásenie zobrazené študentovi v konzole výstupu prostredia. Riadok sa skladá z nasledovných častí, taktiež oddelených podčiarkovníkom, keďže nie je predpoklad výskytu takéhoto znaku v texte chyby:

- text chyby
- číslo modulu, voči ktorému bola prevádzaná kontrola
- počet doterajších výskytov zhodnej chyby v predtým zapísaných logovacích súboroch

```
method minesweeper.core.Field.generateMines()
not found [2: Data types - Mines]_10_4

method minesweeper.core.Field.fillWithClues()
not found [2: Data types - Clues]_10_4

method modifier isPrivate should be true
but is false in method selectFromDB
[10: Database - Best times]_10_4

class modifier isTransient should be true
but is false [1: Data types - Data types]_10_4
```

Obr. 14: Ukážka logovacieho súboru

Ukážku obsahu súboru môžete vidieť na Obr. 14.

Na základe experimentálnej simulovanej práci študenta je možné očakávať na konci semestra až niekoľko stoviek logovacích súborov zozbieraných v príslušnom adresári nástrojov. Ich veľkosť by mala byť zvyčajne nulová (to znamená, že študent programuje bez chýb, resp. chybu si zakrátko po objavení opraví), teda celkovo zanedbateľná. Informačná hodnota je ale vysoká, keďže umožňuje podrobne spätne sledovať aktivitu študenta.

3.5 Nástroj spracovania chybového logu

Tretím a konštrukčne najjednoduchším nástrojom sady je aplikácia spracúvajúca jednoduché textové súbory logu do formy XML zvolenej podľa časti 3.2.4. Jej cieľom stanoveným v 3.1 (automatizovaná kontrola postupu študenta) je jednoduché

a operatívne spustenie s cieľom zistiť nedávnu aktivitu študenta. Vedľajším produktom je možnosť odhalenia kopírovaného zadania, resp. zadania získaného bez aktivity študenta (napríklad odhalením celkovo „až“ niekoľkých kompilácií počas celého semestra).

Obsah logovacieho súboru, ako bol opísaný v 3.4.2, sa skladá zo samotných údajov (riadkov) logu v tele súboru a meta-údajov zapísaných v hlavičke súboru. Keďže sme zvolili jednoduchý textový formát, hlavička je reprezentovaná len názvom súboru a každý riadok obsahu znamená jeden výskyt chyby.

Štruktúra nástroja obsahuje päť tried v jedinom balíku:

- Main – trieda zabezpečujúca spustenie aplikácie, základné nastavenie ciest a inicializáciu premenných používaných v celej aplikácii; obsahuje regulárne výrazy pre kontrolu študentského identifikátora, MAC adresy a času,
- XMLBuilder – trieda zabezpečujúca výstup aplikácie do XML súboru, zapísanie XML hlavičky a údajov podľa nadefinovaného formátovania,
- Student – trieda reprezentujúca entitu študenta identifikovanú zadaným identifikátorom, množinou prislúchajúcich MAC adries a časov kompilácie; určená hlavne pre odlíšenie skopírovaného zadania od iného študenta; na jej základe sú budované rozličné prehľadu podľa MAC adresy, identifikátora študenta a podobne,
- LogFile – trieda predstavujúca objekt jedného súboru zloženého z hlavičky a poľa riadkov,
- LineEntry – trieda reprezentujúca objekt jedného riadku v logovacom súbore, obsahuje text chybového hlásenia, testovaný modul a doterajší výskyt chyby.

Nástroj pre zachovanie jednoduchosti a prehľadnosti neobsahuje samostatný konfiguračný súbor.

```
    <perWeeksByMac>
      <mac address="0021005378FE">
        <week i="4">
          1
        </week>
      </mac>
    </perWeeksByMac>
  </student>
  <student>
    <studentID>
      ab123ce
    </studentID>
    <total12Weeks>
      53
    </total12Weeks>
    <lastWeek>
      0
    </lastWeek>
    <perWeeks>
      <week i="4">
        35
      </week>
      <week i="7">
        18
      </week>
    </perWeeks>
    <perWeeksByMac>
      <mac address="unknown">
        <week i="4">
          19
        </week>
        <week i="7">
          18
        </week>
      </mac>
      <mac address="0021005378FE">
        <week i="4">
          16
        </week>
      </mac>
    </perWeeksByMac>
  </student>
```


Obr. 15: Ukážka časti výstupu spracovania logovacích súborov

Výstupom nástroja je XML súbor rozdelený do častí podľa jednotlivých identifikátorov študenta (pri nekopírovaných študentských zadaniach by mala byť len jedna takáto časť). V rámci každej tejto časti je uvedený hneď v úvode identifikátor študenta, celkový počet kompilácií a počet kompilácií za posledných 7 dní. Nasleduje počet kompilácií v jednotlivých týždňoch a nakoniec počty kompilácií v jednotlivých týždňoch rozdelené podľa MAC adries.

Z ukážky uvedenej na Obr. 15 vieme odčítať, že študent „ab123ce“ zatiaľ použil tento nástroj (úspešne skompiloval hru Minesweeper) 53-krát, z toho ani raz za posledný týždeň, ale 35-krát v štvrtom týždni a 18-krát v siedmom týždni. Jeho

zdrojové kódy boli kompilované na pravdepodobne len jednom počítači s MAC adresou 0021005378FE, pri ostatných kompiláciách sa sieťovú adresu nepodarilo zistiť alebo študent nemal pripojenú žiadnu sieť. Z ukončovaného záznamu (tagu) „student“ na začiatku príkladu vidíme, že študent má pravdepodobne zadanie skopírované od niekoho iného a neskôr len pozmenil svoj identifikátor. Pre zamedzenie spätných zmien je možné súbory a ich hlavičky jednoducho zašifrovať alebo vytvoriť kontrolné súčty nad údajmi.

Práca nástroja po spustení, úvodných inicializáciách premenných a kompilácii vzorov pre regulárne výrazy prebieha nasledovne:

- z adresára logov sa načítajú všetky súbory zodpovedajúceho typu a názvu do objektov LogFile,
- pre každý súbor sa do objektu LogFile načítajú a rozložia (*parse*) riadky obsahu súboru,
- cez XMLBuilder sa vypíšu údaje v požadovanej štruktúre do XML súboru.

Výstupný XML súbor je prehľadnosťou vhodný aj pre námatkovú kontrolu vyučujúcim priamo na cvičení a vďaka zvolenému formátu aj pre ďalšie automatizované strojové spracovanie.

3.6 Webová služba testovania

Súčasťou experimentálnej sady nástrojov je aj webová služba, ktorá dokáže preniesť textový reťazec (meta-údaje napr. s identifikáciou autora) a binárny súbor (očakáva sa JAR súbor študenta) na server a vrátiť hodnotenie vo forme textového reťazca; a tiež samozrejme príslušný konzolový klient pre otestovanie tejto služby. Keďže v čase písania práce nebolo možné z technických príčin nasadenie na katedrových serveroch KPI, nebola nakoniec zahrnutá do hodnotenej sady nástrojov.

4 Experimentálne overenie a vyhodnotenie riešenia

Najpodstatnejšou vlastnosťou programu je možnosť jeho použitia a nasadenia. Pokiaľ by bol program dobrý, ale nebolo by možné používať ho v reálnom prostredí, bol by zbytočný. Preto je dôležité výsledky práce overiť experimentálne pred nasadením na cieľové použitie.

Aplikácia generovania nepriamej reprezentácie je závislá jedine od definovanej anotácie – úspešne je schopná generovať XML reprezentáciu aj pre iné náhodne zvolené JAR súbory (samozrejme ak neobsahujú anotácie zvoleného typu, XML súbor neobsahuje žiadne elementy, ktoré by bolo neskôr možné testovať).

V prípade výskytu chyby je hlásenie nasmerované na definovaný chybový výstup (predvolene *System.err*) a do XML súboru je poznačený výskyt tejto chyby preskočením elementu. Zvyčajne môže byť takouto chybou napríklad metóda bez anotácie, pričom anotácia je definovaná pre triedu a je pre ňu tiež zakázané dedenie (viď opis anotácie v 3.3.1).

Nástroj spracovania logov pri testoch preukázal spoľahlivosť a efektivitu spracovania, keďže informácie zo stoviek súborov dokáže v okamihu premeniť na štruktúrované XML. Návrh štruktúry na prvý pohľad umožňuje zistiť najpodstatnejšie údaje a celkovo je stručný a jednoduchý. Pre dôsledné rozpoznanie častých chýb je nutné zozbierať súbory logu od všetkých študentov, čo môže byť bez automatizácie náročné – toto automatizované zozbieranie je vhodné zveriť systému Moodle v rámci plošného zozbierania študentských zadaní na konci semestra.

Nakoľko nie je možné počas diplomovej práce obsiahnuť dlhodobé (viacsemestrálne) pôsobenie nástrojov vo výučbe, zhodnotenie tejto časti výstupov je možné posúdiť až neskôr s odstupom niekoľkých rokov, keďže predmet je vyučovaný len v letnom semestri.

4.1 Experiment A – výsledná aplikácia porovnania

Prvým prístupom k testovaniu aplikácie a jej opodstatnenia je meranie úspešnosti vo vyhľadávaní chýb v už hotových odovzdaných študentských aplikáciách. Tie boli získané od študentov z predchádzajúcich semestrov.

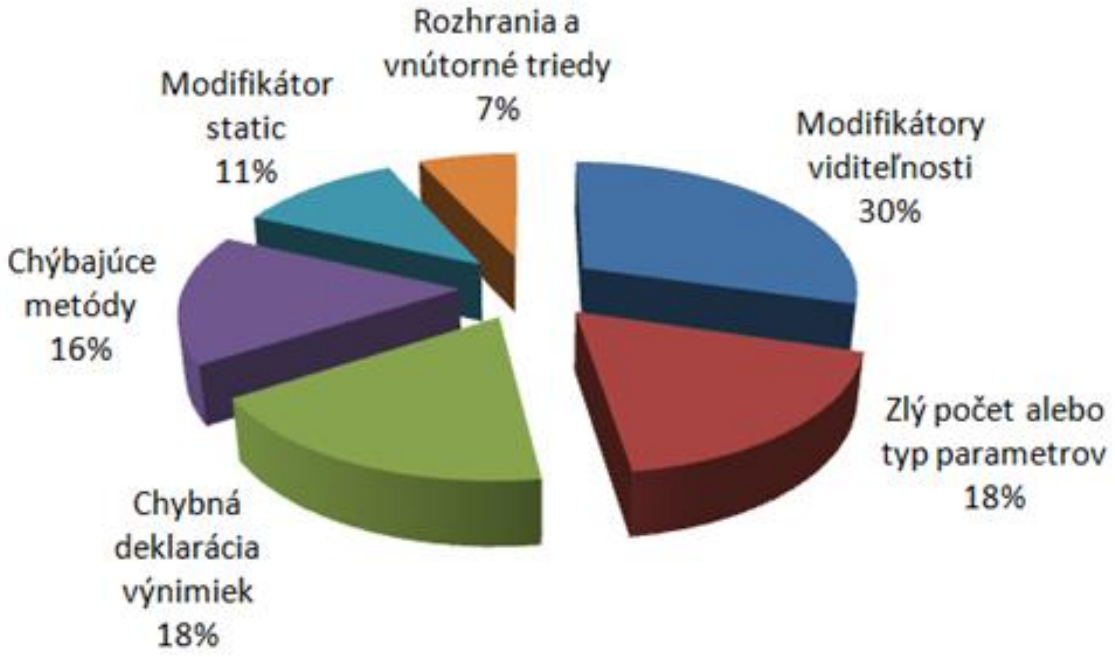
Cieľom experimentu je overiť použiteľnosť a opodstatnenie existencie vytvorenej aplikácie. Program má preukázať merateľné výsledky a obhájiť tak svoj potenciál.

V prípade preukázania úspešnosti je predpoklad reálneho nasadenia vo výučbovom procese v rámci predmetu Technológie Java na jeho určený účel – testovanie postupu v prípadovej štúdii Minesweeper.

Do experimentu bolo zahrnutých 19 hotových aplikácií zozbieraných od študentov s ukončeným predmetom Technológie Java. Tieto aplikácie boli odobrené cvičiacim ako dostatočné pre získanie zápočtu. Každá z nich má rozdielneho autora, i keď nie je možné vylúčiť čiastočnú spoluprácu pri tvorbe aplikácie.

Pre všetky aplikácie test úspešne prebehol až do konca a nevyskytli sa žiadne neočakávané situácie alebo chyby v aplikácií porovnávania.

Experiment prebiehal ako statické otestovanie štruktúr a zanesenie súčtových výsledkov do výsledného grafu. Meraný bol relatívny pomer chýb podľa najčastejších typov chýb.



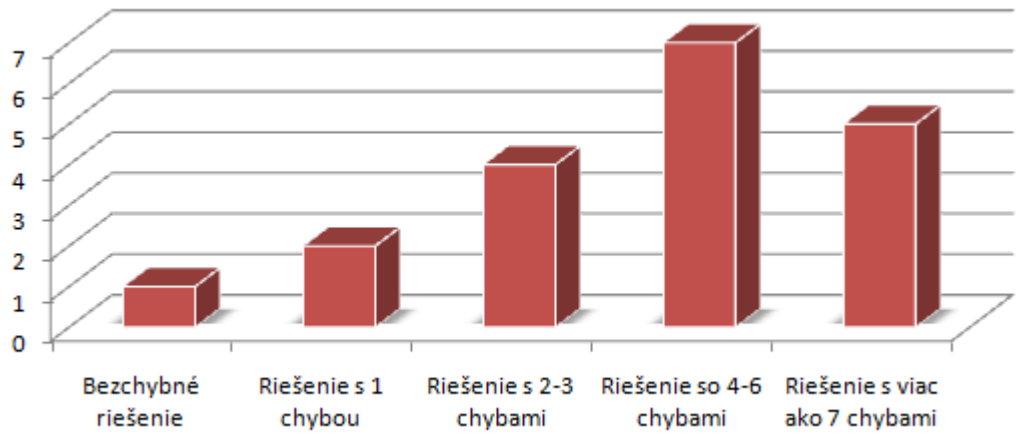
Obr. 16: Výsledky experimentu A – typy chýb

Výsledky experimentu sú zobrazené na Obr. 16. Chyby sú sčítané pre najčastejšie nájdené chyby a vyjadrené v percentách.

Výsledky jasne poukazujú na problematické oblasti. Študenti dávajú malý dôraz na dodržanie princípov zapuzdrenia. Problémy im čiastočne spôsobuje aj nepochopenie princípov objektového programovania – používajú modifikátor *static* viac, než je nutné a žiadané. Túto chybu je dôležité zachytiť už pri prvom výskyte, keďže u študenta sa

práve na začiatku formuje predstava o práci s objektmi. Správne porozumenie objektovej paradigme programovania je jedným z najdôležitejších prínosov výučby v tomto smere.

Zo spracovania podkladov grafu na Obr. 16 tiež vyplýva, že študenti často odovzdávajú viac parametrov, ako je potrebné. To nespôsobil zlý návrh aplikácie, keďže bol vykonaný skúseným programátorom a vyučujúcim – testovaní programátori málo využívali metódy typu *getter* a *setter* a namiesto nich si vypomáhali barličkou odovzdania ďalšieho parametra, čím znižujú prehľadnosť a spoľahlivosť aplikácie.



Obr. 17: Výsledky experimentu A – počty chýb

Počty chýb v jednotlivých riešeniach zobrazené na Obr. 17 ukazujú, že väčšina hodnotených programátorov nejaký detail prehliadla a vniesla do svojho riešenia štrukturálny rozdiel oproti vzorovému riešeniu. Najvyššiu hodnotu blízku aj strednej hodnote reprezentuje množina riešení so štyrmi až šiestimi chybami. Toto množstvo je očakávateľné a tolerovateľné pri projekte tohto rozsahu.

Vytvorená testovacia aplikácia by mala pomôcť vyučujúcemu identifikovať najčastejšie chyby a eliminovať ich upozornením, správnym vysvetlením alebo doplnením študijných materiálov.

4.2 Experiment B – prostredia

Implementované nástroje boli otestované pod v súčasnosti najbežnejšími a najrozšírenejšími operačnými systémami. Funkčnosť bola experimentálne overená na 32-bitových aj 64-bitových MS Windows 7 SP1, Ubuntu 10.04, MacOS 10.6 a OS Solaris. Aplikácie nie sú závislé na veľkosti písmen ani na oddeľovači v cestách pri

používaní súborového systému. Študenti tak môžu aplikácie používať všade tam, kde je prístupné prostredie NetBeans.

Zaťaženie systému je najvýraznejšie pri behu porovnávacej aplikácie, ale aj na najpomalších počítačoch vhodných pre menované operačné systémy je činnosť úspešne ukončená do niekoľkých sekúnd. Pomalšia činnosť môže nastať pri on-line nasadení, ktoré je pri prenose Minesweeper JAR súboru (zvyčajne do 2 MB) závislé od aktuálnej rýchlosti spojenia koncových bodov.

Výsledný počet logovacích súborov pri experimentoch nepresiahol niekoľko stoviek, čo pri priemernej veľkosti súboru pod 1 kB predstavuje prijateľné číslo do 1 MB. XML výstupný súbor vyhodnocovacieho nástroja je vždy prepisovaný aktuálnou verziou, preto je jeho veľkosť do 20 kB zanedbateľná. Veľkosť nepriamej reprezentácie vzoru je do 100 kB. Celková veľkosť, o ktorú je rozšírený prázdny počiatočný projekt dodaný študentom, je približne 150 kB.

Celkovo aplikácia nezaťažuje systém a preto je po technickej stránke možné jej bezproblémové nasadenie aj na vyťažené systémy (napr. katedrový server pre Moodle).

4.3 Zhodnotenie a budúcnosť aplikácie

Aplikácia spĺňa ciele a požiadavky na vlastnosti uvedené v 3.1 a 3.2. Spracúva zadaný typ vstupu, používa štandardizované rozhrania a súborové štruktúry, spĺňa funkčné požiadavky na prírastkové testovanie povinných aj voliteľných úloh v štúdii, poskytuje štandardizovaný modulárny výstup spracovateľný skriptovacími prostriedkami. Je vhodná na webové nasadenie, ľahko rozšíriteľná a upraviteľná aj pre prípadné zmeny štúdie. Po dodaní iného vzorového riešenia s anotáciami navrhnutého typu je možné použiť ju aj na testovanie štruktúry iných projektov. Podporuje testovanie voliteľných úloh, rozdelenie modulov na menšie časti. Vlastnosti sú definované textovým konfiguračným súborom vrátane možnosti presmerovania výstupov (napr. do súboru).

Všetky tieto vlastnosti sú dosahované štandardnými programátorskými postupmi a prostriedkami v jazyku Java. Aplikácia obsahuje štandardné komentáre a integrovanú dokumentáciu vo forme *Javadoc*. Jej štruktúra je prehľadná s dôrazom na zrozumiteľnosť funkcií metód a tried v ich názvoch.

Celkovo je teda aplikácia pripravená na nasadenie. Spĺňa kritériá, ktoré na ňu boli kladené v zadaní a návrhu. Očakáva sa jej nasadenie v reálnom prostredí prístupnom študentom, kde bude slúžiť požadovanému účelu.

Budúce doplnenie a rozšírenie aplikácie spočíva vo využití prostriedkov na spracovanie XML výstupu a po nasadení na katedrové servery aj prepojením so systémom Moodle.

Vyslovujeme presvedčenie, že aplikácia je navrhnutá a vytvorená s ohľadom na procesy a ciele, ktorými nástrojová podpora výučby môže zlepšiť znalosti študentov. Príspevkom k interaktivite praktickej činnosti, zvýšením flexibility a prepojenia študijných materiálov, automatizáciou procesov vyhodnocovania postupu študentov, pridaním nových motivačných faktorov a integráciou do moderného vývojového prostredia spĺňa aj pedagogické nároky kladené odborníkmi z oblasti prípravy študentov na prax programovania v priemyselnom využití.

5 Záver

Predmet Technológie Java je po materiálnej, technickej i personálnej stránke na vysokej úrovni. Navrhovanou možnosťou vylepšenia, ktorou sa táto práca zaoberá, je nástrojová podpora výučby. Vďaka dobrej celkovej myšlienke prírastkovej štúdie a výbornému spracovaniu materiálov sa nástroje pre podporu výučby majú o čo oprieť. Štruktúrované návody na cvičenia vo formáte XML a z neho generovaného HTML umožnili jednoduché prepojenie analýzy úloh a práce študenta na tvorbe aplikácie.

Základným kameňom pre úspešné použitie implementovaných nástrojov je kvalitné a správne vzorové riešenie, ktoré korešponduje s požiadavkami prezentovanými v materiáloch. Automatizácia tvorby nepriamej reprezentácie vzoru umožňuje flexibilne prispôsobovať vzorové riešenie a v náväznosti aj materiály cvičení.

Postupné kroky študenta počas tvorby aplikácie podľa návodu v jednotlivých týždňoch semestra vyžadujú aj inkrementálnu kontrolu. Vďaka technológii anotácií je možné jednoducho priradiť štruktúrne elementy ku konkrétnym modulom štúdie. Navyše sme zahrnuli a implementovali možnosť identifikovať pomocou URL odkazu priamo tie riadky textu modulu, ktoré opisujú cieľ študentovho riešenia.

Keďže technológia reflexie nad skompilovaným kódom má určité obmedzenia, kontrola zahŕňa hlavne štruktúrne prvky. Štruktúrne chyby vyskytujúce sa v študentských riešeniach sú však v takom množstve, že už len vedenie k náprave napr. modifikátorov viditeľnosti by u študentov rozšírilo lepšie porozumenie jazyku.

Možnosť kontroly postupu v štúdii je možné prevádzať jednoduchým spustením nástroja spracovania logov, čo môže študenta včas viesť k formatívnej spätnej väzbe. Vďaka nej je ešte počas semestra priam nútený k aktivite a tak štúdia dostáva nový rozmer priebežnej práce.

V prípade prepojenia nástrojov s katedrovým systémom Moodle je možné sledovať prácu študentov napríklad aj v prípade neprítomnosti vyučujúceho na cvičení. Priebežná spätná väzba dáva študentom väčšiu samostatnosť, príležitosť na včasnú opravu chýb a prispieva k celkovo vyššej kvalite výučby aj výsledných aplikácií.

Konečným výsledkom diplomovej práce je skupina aplikácií implementovaná podľa požiadaviek stanovených počas jej návrhu v súlade s princípmi pedagogiky a odporúčaniami CC2001 v kontexte skúseností s realizáciou štúdií vo výučbe. Experimenty preukázali, že nástroje sú vhodným doplnkom pre prepojenie študijných

materiálov s praxou získavanou samostatnou prácou na zadaní. Tiež poukazujú na pripravenosť testovacej aplikácie na nasadenie do vyučovacieho procesu, kde študentom pomôže odhaliť chyby v štruktúre programovanej hry. Vyučujúcemu sprehľadní a zjednoduší hodnotenie postupu jednotlivcov a tiež poskytne cennú spätnú väzbu o možných vylepšeniach výučby a jej materiálov.

Sada nástrojov má široké možnosti prispôsobenia a rozličných foriem nasadenia. Vďaka všeobecne akceptovaným priemyselným štandardom, ktoré využíva, je možné jej ďalšie prepojenie s rôznymi systémami alebo ďalšími nástrojmi. Je koncipovaná nielen pre predmet Technológie Java, ale môže nájsť širšie uplatnenie vo výučbe objektovo-orientovaného programovania, ktoré je založené na štúdii s hotovým vzorom s cieľom kontroly postupu študentov.

Zoznam použitej literatúry

[1] The Joint Task Force on Computing Curricula. Computing Curricula 2001 Computer Science [online]. December 15, 2001 by IEEE Computer Society. [cit. 2010-11-27]. Dostupné na internete: <http://www.acm.org/education/education/education/curric_vols/cc2001.pdf>.

[2] The Joint Task Force on Computing Curricula. Computer Science Curriculum 2008: An Interim Revision of CS 2001. [online]. November 15, 2008 by ACM Education Council. [cit. 2010-11-07]. Dostupné na internete: <http://www.acm.org/education/curricula/ComputerScience2008.pdf>.

[3] Computer Science Department, University of Pretoria. A philosophy of teaching Java as a first teaching language: An Interim Revision of CS 2001. [online]. November 15, 2001. [cit. 2010-11-01]. Dostupné na internete: <http://www.acm.org/education/curricula/ComputerScience2008.pdf>.

[4] KAY, D. G. 1996. Bandwagons considered harmful, or the Past as Prologue in Curriculum Change. In ACM SIGCSE Bulletin. [online]. 1996, vol. 28 no. 4, p. 55–58, 64. [cit. 2011-02-10]. Dostupné na internete: < http://portal.acm.org/citation.cfm?id=242666>.

[5] DEITEL, H. M. - DEITEL, P. J. 1997. Java: how to program. New Jersey : Prentice Hall, 1997. ISBN 9780138993948.

[6] BISHOP, J. M. 2001. Java Gently programs. [online]. Harlow : Pearson Education, 2001, 655 p. [cit. 2011-11-01]. Dostupné na internete: < http://books.google.sk/books?id=PvqjOO6nPrwC&lpg=PP1&ots=kMlZPiD0NI&dq=Java%20Gently%20programs%20bishop&pg=PR4#v=onepage&q=Java%20Gently%20programs%20bishop&f=false>. ISBN 0-201-71050-1.

[7] ASTRACHAN, O. 1997. Computer science: the first year - Beyond language issues. In: ACM SIGCSE Bulletin of the 27th Technical Symposium on Computer Science Education [online]. 1996, vol. 28, no. 1, p. 389–390. [cit. 2011-02-10]. Dostupné na internete: < http://www.cs.duke.edu/~ola/slides/lang96.html>.

[8] ADAMS, J. - FRENS, J. 1996. Object-centered design: a five-phase introduction to object-oriented programming in CS1-2. In: ACM SIGCSE Bulletin of the 27th Technical Symposium on Computer Science Education [online]. 1996, vol. 28, no. 1, p. 78-82. [cit. 2011-03-05]. Dostupné na internete: < http://www.google.sk/url?sa=t&source=web&cd=1&ved=0CBgQFjAA&url=http%3A%2F%2Fciteseerx.ist.psu.edu%2Fviewdoc%2Fdownload%3Fdoi%3D10.1.1.89.298%26rep%3Drep1%26type%3Dpdf&ei=FdW2TZzPJY_Dswaz79zZDQ&usg=AFQjCNH5Sp0f1LpquX844UlPtZkIXfVyyw>.

[9] WOODMAN M. - DAVIES, G. - HOLLAND S. The Joy Of Software – Starting With Objects,. In: ACM SIGCSE Buletin: 27th Technical Symposium on Computer Science Education. [online]. Charlotte : ACM Press, 1996, vol. 28 no.

1, p. 88-92. [cit. 2010-12-03]. Dostupné na internete: < http://portal.acm.org/citation.cfm?doid=236452.236515>. ISBN 0-89791-757-X.

[10] MAZÁK, J. 2009. Minesweeper – testovanie postupu v prípadovej štúdii. Bakalárska práca. Košice: TU FEI, 2009. 58 s.

[11] PETLÁK, E. 2004. Všeobecná didaktika. Bratislava : Iris, 2004, 311 s. ISBN 80-89018-64-5.

[12] PETTY, G. 2008. Moderní vyučování. Praha : Portál, 2008, 380 s. ISBN 978-80-7367-427-4.

[13] BLAŠKO, M. 2008. Úvod do modernej didaktiky. Košice : KIP TU, 2008, 248 s. ISBN 978-80-8073-973-7.

[14] ČÁP, J. – MAREŠ, J. 2001. Psychologie pro učitele. Praha : Portál, 2001, 656 s. ISBN 80-7178-463-X.

[15] LISTER, R. 2003. Assignment on Object Oriented Programming. [online]. Sydney : FEIT, UTS, 2003. [cit. 2010-10-10]. Dostupné na internete: <www-staff.it.uts.edu.au/~raymond/oop/assignment2003Spring.doc>

[16] FORMAN, I. R. - FORMAN, N. 2005. Java Reflection in action. Greenwich : Manning Publications Co., 2005, 644 p. ISBN 1-932394-18-4

[17] PORUBÄN, J. a kol. 2011. Nástroje pre podporu výučby programovania - I. časť. 13.4.2011, Košice : KPI FEI TU.

[18] PETRE, M., et al. 2004. A large-scale elicitation of students' knowledge of programming constructs. [online]. [cit. 2010-11-11]. Paper submitted to ITiCSE 2004. Dostupné na internete: <http://www.sciweavers.org/conference/iticse-2004>.

[19] BURTON, P. J. – BRUHN, R. E. 2003. Teaching Programming in the OOP Era. In: ACM SIGCSE Bulletin [online]. 2003, vol. 35, no. 2, p. 111-115. [cit. 2011-02-10]. Dostupné na internete: <http://academic.research.microsoft.com/Publication/1245382/teaching-programming-in-the-oop-era>. DOI 10.1145/782941.782993.

[20] CULWIN, F. 1999. Object Imperatives! In: ACM SIGCSE Bulletin: 30th Technical Symposium on Computer Science Education. [online]. New Orleans : ACM Press, 1999. s. 31-36. [cit. 2011-03-15]. Dostupné na internete: <http://www.informatik.uni-trier.de/~ley/db/indices/a-tree/c/Culwin:Fintan.html>. ISBN:1-58113-085-6.

[21] MITCHELL, W. 2001. A Paradigm Shift to OOP has Occurred... Implementation to Follow. In Journal of computing in Small Colleges [online]. 2001, Vol. 16, no. 2, p. 95-106. [cit. 2011-03-15]. Dostupné na internete:

<http://portal.acm.org/ft_gateway.cfm?id=374876&type=pdf&coll=DL&dl=GUIDE&CFID=19589801&CFTOKEN=39563839 >. ISBN 9780898713664.

[22] PATTIS, R. E.: Karel the Robot: A Gentle Introduction to the Art of Programming. New York : Wiley, 1981, 106 p.

[23] BECKER, B. 2001: Teaching CS1 with Karel the Robot. ACM SIGCSE: 32nd Technical Symposium on Computer Science Education. [online]. Charlotte : ACM Press, 2001, p. 50-54 [cit. 2011-03-16]. Dostupné na internete: <http://www.cs.uwaterloo.ca/~bwbecker/papers/javaPedagogies.pdf>.

[24] BERGIN, J. et al. 2003. Karel J. Robot: A Gentle Introduction to the Art of Object Oriented Programming. [online]. New York : Wiley, 2003, 226 p. [cit. 2011-03-16]. Dostupné na internete: <http://csis.pace.edu/~bergin/KarelJava2ed/Karel++JavaEdition.html>. ISBN 978-0-97057951-5.

[25] BUCK, D. – STUCKI, B. 2001. JKarelRobot: A Case Study in Supporting Levels of Cognitive Development in the Computer Science Curriculum. ACM SIGCSE: 32nd Technical Symposium on Computer Science Education. [online]. Charlotte : ACM Press, 2001, p. 16-20. [cit. 2011-03-16]. Dostupné na internete: < http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.119.9150&rep=rep1&type=pdf>.

[26] BARNES, D. J. - KÖLLING, M. 2003. Objects First With Java: A Practical Introduction Using BlueJ [online]. Prentice Hall : Pearson Education, 2003. [cit. 2011-01-07]. Dostupné na internete: <http://www.bluej.org/objects-first/description.html>. ISBN 0-13-044929-6.

[27] BRADY, A. 2002. The Marine Biology Simulation Case Study. In AP Computer Science [online]. 2002, no. 8, 108 p. [cit. 2011-01-07]. Dostupné na internete: <http://kbucaria.com/apcomp/jmbs.pdf>.

[28] LISTER, R. - LEANEY, J. 2003. Introductory Programming, Criterion Referencing and Bloom, ACM SIGCSE: 34th Technical Symposium on Computer Science Education. [online]. [cit. 2010-11-01]. Reno : ACM Press, 2003, p. 143 – 147. Dostupné na internete: <https://portal.acm.org//purchase.cfm?id=611954&CFID=19589801&CFTOKEN=39563839>. ISBN 1-58113-648-X.

[29] LISTER, R. 2007. Teaching Java First: Experiments with a Pigs-Early Pedagogy. [online]. [cit. 2010-11-01]. Dostupné na internete: <http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=ABF055817C9B5B8DDF611EF8C4B0D6DD?doi=10.1.1.6.4304&rep=rep1&type=pdf>.

[30] KÖLLING, M. - ROSENBERG, J. 2001. Guidelines for Teaching Object Orientation with Java. ACM SIGCSE Bulletin. [online]. 2001, vol. 33, no. 3, p. 33-36. [cit. 2011-02-10]. Dostupné na internete:

<http://www.cs.kent.ac.uk/pubs/2001/2187/content.pdf>. DOI 10.1145/507758.377461.

Prílohy

Príloha A: CD médium – diplomová práca v elektronickej podobe, JavaDoc dokumentácia k aplikáciám, implementované nástroje, príslušné konfiguračné a iné súbory.