



Konferencie
SILESIAN MOODLE MOOT

Ekonomická fakulta
VŠB-TU Ostrava

Čeladná, Beskydy
12. - 13. 11. 2009

Ako motivovať študentov softvérového inžinierstva

**Miroslav Biñas, Marek Novák, Miroslav Michalko,
František Jakab**

Fakulta elektrotechniky a informatiky, Technická univerzita v Košiciach
miroslav.binas@tuke.sk, marek.novak@tuke.sk,
miroslav.michalko@tuke.sk, frantisek.jakab@tuke.sk

Abstrakt: *Predmety na vysokých školách, ale nie len na nich, ktoré sa venujú výučbe programovania, patria medzi jedny zo základných predmetov najmä na odborne zameraných školách. Preto si vyžadujú značnú pozornosť pri ich príprave a priebehu výučby. Aj napriek tomu však túto problematiku ako učitelia dosť podceňujeme a miesto výučby princípov a technik sa sústreďujeme na výučbu jazyka a technológií a mylne sa domnievame, že si študenti princípy osvoja sami. Existujú však metodiky a nástroje, ktoré umožňujú výučbu programovania zjednodušiť a mnohé abstraktné princípy dokážu podať študentom spôsobom, ktorému budú rozumieť, a ktorý nebudú mať problém si osvojiť.*

Kľúčové slova: *programovanie, metodika, motivácia*

Abstract: *Courses, mainly at universities but not only on them, about programming techniques may be considered as one of the most essential subjects. Therefore remarkable attention should be taken in a preparation and education process. In spite of this fact we as lecturers usually underestimate this problematic. Focus is taken on education of particular computer language and technologies and not on computer programming principles at all. We wrongly assume that students will learn the principles themselves. There are methodologies and tools, which simplifies programming techniques education, presenting the problematic in a friendly way, easier to understand by students.*

Keywords: *programming, methodology, motivation*

1 Chyby, ktorých sa dopúšťame pri výučbe objektovo orientovaného programovania

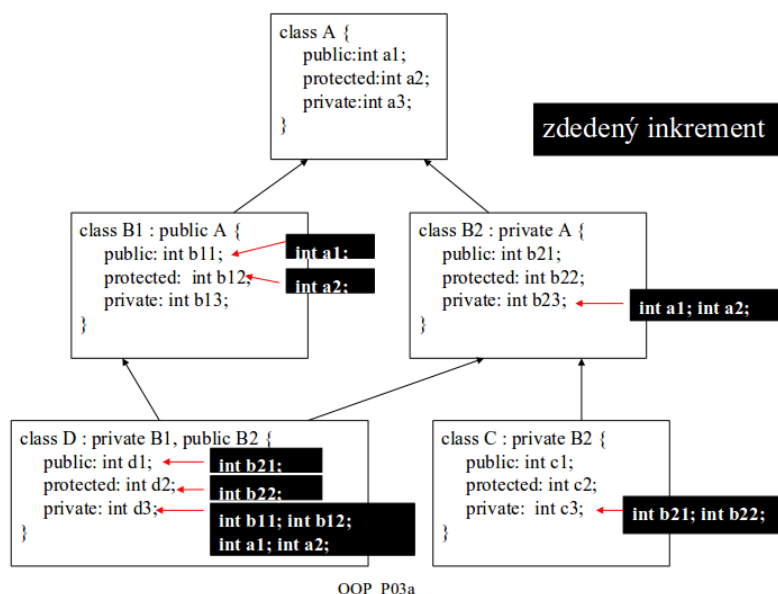
Objektovo orientované programovanie (ďalej len OOP) predstavuje v súčasnom modernom softvérovom priemysle jeden zo základných stavebných pilierov moderných aplikácií. Nejedná sa o technológiu, ale o paradigmu programovania, resp. metodiku, ktorá si vyžaduje odlišnú filozofiu, ako je tomu v prípade procedurálneho prístupu ku programovaniu. A práve skutočnosť, že sa jedná o filozofiu a nie o konkrétny jazyk alebo technológiu, nám pri výučbe OOP často uniká.

Túto skutočnosť si je možné všimnúť v rozličných publikáciách, ktoré sa nazývajú učebnicami OOP, ale rovnako aj v rozličných kurzoch, ktoré si kladú za cieľ naučiť poslucháčov princípom OOP. Ako vedľajší efekt takéhoto prístupu je možné sledovať aj reakcie samotných účastníkov, ktorí si myslia, že tým, že sa zúčastnia kurzu, ktorý sa venuje výučbe OOP, sa súčasne naučia vytvárať aj GUI aplikácie.

O rovnakých problémoch hovorí vo svojich článkoch aj Rudolf Pecinovský [5], ktorý je považovaný za jedného z najväčších odborníkov na výučbu OOP v Českej republike. S podobnými závermi sa je však možné stretnúť aj v ďalších článkoch, ako [2], [3].

1.1 „Killer examples“

Jeden zo základných problémov, ktorého sa dopúšťame pri výučbe OOP, sú nevhodné ilustrácie princípov OOP. Mnohokrát sa dokonca jedná o príklady, ktoré nielenže nesprávne modelujú realitu, ale častokrát sú absolútne abstraktné, nič nehovoriace a dá sa predpokladať, že im rozumie iba sám autor. Ak teda chceme študentov naučiť rozumieť a osvojiť si princípy OOP, musíme sa snažiť používať príklady z reálneho života, ktorým budú študenti rozumieť.



Obr. 1 Diagram tried ako ilustrácia pri výučbe dedičnosti. Autor sa však dopustil niekoľkých chýb pri pomenovaní tried a členských premenných a taktiež použité šípky na vyjadrenie vzťahu dedičnosti nie sú správne. Ako príklad je teda tento obrázok absolútne nevhodný a pre pochopenie príliš abstraktný.

Problémom cvičení zasa býva, že v snahe o čo najjednoduchšie podanie problematiky a odskúšanie si princípov OOP v praxi, sa snažíme aj úroveň úloh čo najviac znížiť. Dostávame sa tak na úroveň príkladov typu “Hello world”, ktoré síce ilustrujú danú problematiku v najjednoduchšej forme, ale nepredstavujú pre študentov žiadnu výzvu. V snahe o čo najväčšie zjednodušenie samotného problému sa teda dostávame ku čiastkovým úlohám, ktoré riešia len čiastkový problém. Študenti tak ale nemajú možnosť vidieť komplexnejší pohľad na problém, kde je práve možné vidieť výhody objektového prístupu k riešeniu problému. Zostávajú tak len na úrovni jednej až troch tried, kde je ich úlohou stále len vytvorenie istého algoritmu, ktorý bude riešiť potrebnú úlohu. Keď však študentom neukážeme komplexný návrh problému (objektový model) a jeho rozdelenie do menších celkov (tried), nemôžeme očakávať, že študenti na túto skutočnosť a výhodu objektového prístupu prídu sami bez predchádzajúcich znalostí.

1.2 Sústreďenie sa na jazyk, technológiu a nástroje

Súčasťou výučby OOP je aj programovací jazyk, v ktorom je problematika prezentovaná, a v ktorom študenti precvičujú svoje zručnosti a implementujú záverečný projekt. Ak sa však výklad problematiky OOP začne zaoberať výkladom jazyka, kurz sa mení na štandardný kurz výučby procedurálneho programovania, kde sa výklad venuje

základným dátovom typom, riadiacim štruktúram a podobne. Objektové princípy sa vo výklade ocitnú niekde na konci a nie je im venovaný dostatočný čas a priestor.

Podobným problémom je taktiež prezentácia technológií daného jazyka ako súčasť predmetu o objektovom programovaní. Keďže sa od študentov očakáva, že využijú danú technológiu vo svojich projektoch, je potrebné, aby sa taktiež dostatočne venovali štúdiu danej technológie. Študenti sú tým nútení stráviť viac času štúdiom danej technológie ako premýšľaním nad správnym objektovým návrhom. V konečnej implementácii záverečných projektov tak navzájom súťažia, kto spravil „krajšiu“ aplikáciu.

Kurzy, kde sa nevyhneme samotnému výkladu o jazyku, je možné charakterizovať ako „*Objektové programovanie bez predchádzajúcich znalostí*“ alebo „*Objektové programovanie pre úplných začiatočníkov*“. Poslucháčmi týchto kurzov sú teda študenti, ktorí nemajú s programovaním žiadne skúsenosti a sú teda úplní začiatočníci. Na technických vysokých školách však obyčajne predmetu venujúcemu sa OOP predchádza predmet, ktorý si kladie za cieľ naučiť poslucháčov základom algoritmizácie a programovania.

2 Škola hrou

Autorom na svoju dobu revolučného a naozaj neobvyklého pojmu „škola hrou“, je *Ján Ámos Komenský*, ktorý je v našich česko-slovenských končinách známy aj ako učiteľ národov. Hry majú v oblasti IT priemyslu svoje význačné postavenie a tiahnu sa celou históriou počítačov. Mnoho osobností v IT priemysle sa k počítačom a programovaniu dostalo práve prostredníctvom hier a vlastným, či už úspešným alebo neúspešným, pokusom podobnú hru vytvoriť. Dá sa teda predpokladať, že podobným spôsobom sa ku počítačom dostali aj súčasní študenti.

Fred Martin vo svojom článku [2] hovorí, že pokiaľ študenti po skončení cvičenia povedia, že vytvorili niečo, čo je ich, my ako učitelia sme uspeli. Toto tvrdenie len podporuje myšlienku uvedenú vyššie, že príklady na úrovni „*Hello world*“ sú pre študentov nezaujímavé, nepredstavujú pre nich žiadnu výzvu a na výsledok študenti nemajú byť prečo „hrdí“. Ak teda budeme ako učitelia schopní študentom predložiť problém, ktorý ich dostatočne zaujme, a ktorý si budú vedieť dostatočne prisvojiť za svoj, vyhrali sme. A práve zadanie, ktorého výsledkom je počítačová hra, sa tomuto cieľu značne približuje, pretože študenti vytvárajú niečo, čo je ich - niečo, s čím sa môžu pochváliť aj pred svojimi kamarátmi.

Skrytým problémom je však otázka vhodných a použiteľných technológií, ktoré môžu študenti v danom predmete pre vývoj svojej hry použiť. V prípade výučby OOP sa v prvom rade jedná o výučbu a zvládnutie princíпов a nie technológie, takže vyžadovať od študentov, aby zvládli potrebné technológie pre prácu s grafikou alebo zvukom, nie je dobré. Vhodný kompromis v tomto prípade predstavuje prostredie *Greenfoot* [3], ktoré sa snaží práve technologickú časť čo najviac zakryť. V prípade výučby technologických predmetov a kurzov je však situácia iná, pretože tu sa dá s technológiami naozaj pracovať. Pri vytváraní podmienok pre vypracovanie zadania je preto potrebné na túto skutočnosť nezabudnúť.

3 Výučba OOP na Technickej univerzite v Košiciach

Výučba OOP na Technickej univerzite v Košiciach prešla od svojho vzniku niekoľkými zmenami. Najčastejšie sa jednalo o zmeny v jazyku a technológiách, pričom záverečné zadania zostávali nezmenené (informačné systémy pre evidenciu položiek rozličných typov). Od roku 2008 však došlo k zásadnej zmene v koncepcii cvičení predmetu, kedy sme sa prvýkrát snažili z cvičení vypustiť všetky technológie a venovať sa výučbe na základe princípu *Objects First* (najprv objekty).

3.1 Štruktúra cvičení

Najväčší problém pri zostavovaní cvičení je výber vhodných príkladov, na ktorých sa má daná vlastnosť alebo princíp OOP prezentovať. Opustili sme však spôsob výučby pomocou *Killer examplev* a *AHA príkladov* a počas cvičení si študenti osvojujú preberanú problematiku prácou na čiastkových úlohách, ktoré vytvárajú väčší celok. Cvičenia sú rozdelené do dvoch väčších celkov.

V prvom celku si študenti postupne osvojujú základné princípy OOP, pri čom im značne pomáha aj použitie prostredia *BlueJ* [1] [4]. Študenti postupne prejdú od vytvorenia jednej triedy, cez riadenie viditeľnosti a preťažovanie metód, až ku problematike abstraktných tried, prekryvaniu metód a dedičnosti. Postupne teda vytvoria triedu reprezentujúcu obdĺžnik, následne pomocou nej dokážu vytvoriť pohybujúci sa výt'ah s ovládačom nie len v kabínke, ale aj na jednotlivých poschodiach, stavový riadok, ktorý informuje o aktuálnom stave kabínky výt'ahu a končia implementovaním ďalších tvarov, ktoré výsledný projekt príjemne graficky dopĺňajú. Grafické operácie sú pritom pred študentmi ukryté a vykresľovanie grafických primitív je dopredu pripravené.

Študenti si na začiatku cvičenia stiahnu do počítača výsledné riešenie projektu z minulého cvičenia, na ktorom potom ďalej pracujú. Môžu teda svoj výsledný kód porovnať s kódom, ktorý im je poskytnutý ako ukážkový. Počas cvičenia študentov sprevádzajú podrobné scenáre, ktoré obsahujú sadu pripravených úloh na riešenie. Pre šikovnejších sú pripravené dodatkové úlohy, ktoré pôvodnú úlohu ďalej rozširujú.

Druhý celok, na ktorého riešenie študenti používajú vývojové prostredie *NetBeans*, sa venuje prípadovej štúdiu – a teda implementácii študentského zadania. Študenti už ale nemajú k dispozícii tak podrobné scenáre, ako tomu bolo v prípade prvého celku, pretože za výsledok prípadovej štúdie sú hodnotení. Nemajú taktiež k dispozícii ukážkové riešenia, ale na cvičenie si musia nosiť vždy výsledok svojho vlastného projektu z predchádzajúceho cvičenia, čo ich núti riešiť úlohy na čas. Pripravené scenáre v tomto prípade predstavujú pre študentov smerovanie počas ich tvorby na zadaní – od prvotnej analýzy až po finálne riešenie.

3.2 Štruktúra scenára

Každý scenár má svoju pevnú štruktúru. Zdrojový súbor scenára predstavuje XML dokument, v ktorom je možné použiť špeciálne značky, čím je možné dokument štruktúrovať (podobne, ako ľubovoľný HTML dokument). Výhodou takéhoto prístupu je, že učiteľ, resp. autor kurzu sa môže viac sústrediť na obsah scenára ako na jeho formu (vzhľad).

Každý scenár sa skladá z týchto častí:

- TITLE – Názov scenára, resp. precvičovanej problematiky.
- OBJECTIVES – Množina cieľov, ktoré majú byť na cvičení dosiahnuté.
- ABOUT – Úvod do problematiky cvičenia, ktorý v krátkosti predstaví problém.
- CONTENT – Samotný obsah scenára, ktorý je tvorený krokmi (značka STEP) a úlohami (značka TASK). Kroky slúžia na členenie scenára do menších logických celkov. Úlohy predstavujú samotné problémy, ktoré majú študenti riešiť, a ktoré majú viesť k dosiahnutiu cieľov uvedených v časti OBJECTIVES.
- ADDITIONAL TASKS – Množina doplňujúcich úloh, ktoré je možné riešiť, a ktoré umožňujú danú problematiku precvičiť ešte viac.
- ADDITIONAL LINKS – Množina referencií na ďalšie zdroje, ktoré môžu slúžiť ako doplnkový alebo odrazový materiál pre zvládnutie problematiky v scenári.

Okrem uvedených značiek je samozrejme možné použiť aj ďalšie značky, ktoré primárne slúžia na formátovanie textu a vkladanie obrázkov a liniek. Výsledný dokument je následne možné transformovať podľa XSLT šablóny do potrebného výstupného formátu, ktorý potom študenti môžu na cvičeniach použiť.

Autorom pôvodnej myšlienky štruktúrovaných cvičení je Doc. Ing. Jaroslav Porubán PhD. z Katedry počítačov a informatiky na Technickej univerzite v Košiciach.

3.3 Prípadová štúdia

Záverečné zadanie, ktoré má preukázať znalosti študentov z problematiky OOP, predstavuje vytvorenie jednoduchej textovej hry (adventúry). Textovka je realizovaná v textovom prostredí, čím je zabezpečené, že študenti pre svoju prácu nepotrebujú poznať žiaden špecifický softvérový rámec alebo technológiu. Nakoľko sa jedná o pomerne rozsiahly projekt z pohľadu počtu implementovaných tried, je potrebné študentov usmerniť a vytýčiť im isté mantinely, ktoré neskôr poslúžia aj pre potreby testovania zadaní pomocou pripravených testov. Tieto mantinely sú reprezentované pomocou vopred špecifikovanej množiny rozhraní, ktoré sú študenti povinní vo svojich zadaniach použiť a implementovať. Takýmto spôsobom je možné študentom priamo ukázať, ako je možné rozčleniť väčší projekt na menšie celky podľa vopred definovaného opisu správania. Vytýčením takýchto mantinelov sa je možné čiastočne vyhnúť problémom súvisiacim s nepochopením objektových princípov. Taktiež núti študentov viac si osvojiť a porozumieť významu a používaniu rozhraní.

Môže sa však zdať, že keď sú študenti nútení použiť vo svojich projektoch rovnaké rozhrania, ich zadania sú identické. Problém duplicity zadaní je odstránený vytvorením vlastného scenára hry. Scenár predstavuje jednoznačnú postupnosť krokov, ktoré vedú k úspešnému vyriešeniu hry. Ten je preto pre študentov záväzný a študenti ho nemôžu po odovzdaní meniť. Dovoľené sú len čiastočné úpravy po dohode s cvičiacim - nemôže však dôjsť ku zmene scenára ako celku. Študenti pri odovzdávaní scenára musia dodržať aj isté minimálne limity, ktoré sa týkajú jeho veľkosti (počet príkazov), počtu vytvorených miest herného sveta ako aj počet predmetov nachádzajúcich sa v hre.

Inšpiráciu pre tento typ zadania sme čerpali z publikácie [1], ktorou sa inšpiroval aj Rudolf Pecinovský a zo skutočnosti, že tento formát hier bol pomerne obľúbený počas éry 8b mikropočítačov.

3.4 Reakcie študentov

Najväčšou motiváciou pre študentov je skutočnosť, že môžu konečne vytvoriť niečo vlastné. Mnoho študentov je taktiež motivovaných spôsobom vedenia cvičení, ktoré považujú za značne motivujúce v porovnaní s predmetmi, ktoré už absolvovali. Z pohľadu učiteľa je pozitívne sledovať, ako práca na vlastnej hre motivuje aj slabších študentov k tomu, aby boli aktívnejší a využívali všetky dostupné prostriedky pre diskusiu o problémoch, s ktorými sa pri implementácii stretli. Je potešujúce sledovať diskusie študentov aj počas prednášok iných predmetov, ako sa snažia so zápalom riešiť svoje implementačné problémy pri oživovaní rozličných príšer alebo riešení rozličných logických úloh, aby cestu svojmu hlavnému hrdinovi dostatočne „spríjennili“.

4 Záver

Spôsob, ako motivovať študentov akéhokoľvek predmetu, stále predstavuje isté „umenie“ a vyžaduje si istú dávku trpezlivosti a skúseností. Pokiaľ sa nám to však ako učiteľom podarí, môžeme povedať, že sme vyhrali. Nadchnúť študenta znamená motivovať a inšpirovať ho, aby sa snažil dosiahnuť cieľ, aby sa snažil byť samostatný, a aby sa snažil byť lepší.

Tento článok prezentuje jeden z možných spôsobov, ako motivovať a nadchnúť študentov pre štúdium objektového programovania a nenásilnou formou si osvojiť jeho princípy. Začlenenie hier do procesu výučby sa javí ako vhodný spôsob ako tento cieľ splniť. Túto myšlienku podporuje aj existencia nástrojov ako *Greenfoot*, ktoré je možné na dosiahnutie tohto cieľa vhodne využiť a rovnako aj skúsenosti z ďalších predmetov na Technickej univerzite v Košiciach, ktoré do svojej prípadovej štúdie hru začlenili.

Literatura

1. Barnes, D. J., Kolling, M. *Objects First With Java: A Practical Introduction Using BlueJ.*, Prentice Hall, 2008, 560s, ISBN 0136060862
2. Fred, M. TOY PROJECTS CONSIDERED HARMFUL. New York, NY, USA : ACM, 2006, Commun. ACM, Zv. 49, s. 113-116. ISSN:0001-0782.
3. Henriksen, P., Kölling M. *Greenfoot - The Java Object World.* [online] [cit. 2009-10-18]. Dostupné na www: <<http://www.greenfoot.org/>>.
4. Kölling, Michael. *BlueJ - Teaching Java - Learning Java.* [online] [cit. 2009-10-18]. Dostupné na www: <<http://www.bluej.org/>>.
5. Meyer, B. SOFTWARE ENGINEERING IN THE ACADEMY. 5, Los Alamitos, CA, USA : IEEE Computer Society Press, 2001, Computer, Zv. 34, s. 28-35. ISSN 0018-9162.
6. Pecinovský, R. JAK EFEKTIVNĚ UČIT OOP, 2005. Dostupné na www: <http://vyuka.pecinovsky.cz/prispevky/2005-SW_Jak_efektivne_ucit_OOP.pdf>.
7. Pecinovský, R. *Myslíme objektově v jazyku Java*, Grada, 2008, 576s, ISBN 9788024726533

ACKNOWLEDGEMENT

This publication is particularly the result of the project implementation “*Building of virtual and remote experiments for network of on-line laboratories*”, (project number: KEGA, 3/7245/09, 2009-2011.)