ÚVODNÝ KURZ PROGRAMOVANIA V JAVE NA PF UPJŠ[1]

FRANTIŠEK GALČÍK, PETER GURSKÝ, RÓBERT NOVOTNÝ

ABSTRAKT

Obsah i forma úvodného kurzu programovania na vysokých školách patrí ku skúmaným a diskutovaným otázkam - tak na Slovensku ako aj v zahraničí. V tomto článku predstavíme úvodný kurz programovania na PF UPJŠ v Košiciach v programovacom jazyku Java. Kurz je príkladom adaptácie overených a osvedčených postupov z detských programovacích prostredí do Javy v profesionálnom vývojom prostredí. Od úplného začiatku je v tomto kurze realizovaný úvod do objektovo orientovaného programovania súbežne s úvodom do programovania a algoritmizácie. Z dôvodu motivácie je dôraz kladený na vizuálne a interaktívne prvky s využitím korytnačej grafiky. Vizuálne orientovaný úvod neskôr pozvoľne prechádza do systematického výkladu princípov OOP a práce s údajmi v druhej časti kurzu, v ktorej sa majú študenti intuitívne oboznámiť s dobrým objektovým návrhom. Skúsenosti pri výučbe kurzu ukazujú, že prístup „objektové programovanie najprv" a programovací jazyk Java umožňujú zrealizovať kurz programovania spĺňajúci požiadavky kladené na úvodné kurzy programovania.

Kľúčové slová: *programovanie, Java, korytnačia grafika, objektovo orientované programovanie*

ÚVOD

Úvodný kurz programovania je zvyčajne prvým informatickým predmetom, s ktorým sa stretnú študenti bakalárskeho štúdia informatiky. Tento predmet má za cieľ rozvinúť u študentov základné algoritmicko-programátorské vedomosti a zručnosti. Keďže na týchto vedomostiach neskôr stavajú ďalšie predmety ich štúdia a zároveň pre mnohých študentov predstavuje tento kurz „prvý kontakt" s ozajstnejšou informatikou, je jeho kvalitné obsahové i metodologické zvládnutie mimoriadne dôležité. Významným aspektom realizácie tohto kurzu v našich podmienkach je taktiež to, že kurzu sa zúčastňujú študenti, ktorí neprešli nijakou selekciou a často sú aj ich predstavy o informatike ako takej veľmi skreslené. Z realizovaných dotazníkov na začiatku semestra vyplýva, že počiatočné algoritmicko-programátorské skúsenosti väčšiny študentov sú skôr slabé až žiadne (paradoxne, nemalý počet z nich v dotazníkoch deklaruje znalosť programovania v jazyku Pascal alebo C zo stredných škôl). Popri týchto študentoch sa však kurzu zúčastňujú aj študenti s pokročilejšími skúsenosťami. Značná rôznorodosť študentov s prevahou študentov so slabšími predošlými programátorskými skúsenosťami tak robí tento kurz v porovnaní s inými predmetmi oveľa náročnejším. Aj kvôli týmto dôvodom sa otázky obsahu a metodológie úvodného kurzu programovania stali predmetom intenzívneho výskumu a diskusií – či už nás alebo v zahraničí. Dobrý prehľad tejto problematiky poskytuje práca [2]. Doposiaľ však neexistuje (a pravdepodobne ani nebude existovať) jednoznačná odpoveď ako realizovať úvodný kurz programovania. Je to dané najmä tým, že každá vysoká škola musí obsah tohto kurzu prispôsobiť svojim špecifickým podmienkam (tradície, schopnosti prichádzajúcich študentov, profil absolventov štúdia, atď.).

V tomto článku predstavíme úvodný kurz programovania realizovaný na Ústave informatiky PF UPJŠ v Košiciach s kódovým označením predmetu *PAZ1a* [7]. Tento predmet je prvým z trojice predmetov *PAZ1a, PAZ1b* a *PAZ1c*, ktoré obsahovo pokrývajú navrhované kurikulum ACM-IEEE pre kurzy označované ako CS1 a CS2 ([1], [9]). Ako programovací jazyk spomenutej trojice predmetov bol „z politických" dôvodov zvolený jazyk Java – pri výbere sa prihliadalo najmä na obsahové požiadavky ďalších študijných predmetov, požiadavku na znalosť OOP a v neposlednom rade i o požiadavky trhu práce v košickom regióne. Obsah jednotlivých predmetov ovplyvnilo aj to, že predmety *PAZ1a* a *PAZ1b* (realizované v zimnom a letnom semestri počas prvého roku štúdia) sú povinné nielen pre študentov informatiky, ale aj pre študentov medziodborovného („učiteľského") štúdia informatiky a študentov matematiky. Obsahové zameranie týchto predmetov je postavené nasledovne:

- *PAZ1a* (ZS 1. ročník): úvod do programovania a algoritmizácie, základy OOP;
- *PAZ1b* (LS 1. ročník): rekurzia, základné algoritmy a dátové štruktúry, techniky návrhu efektívnych algoritmov;
- *PAZ1c* (ZS 2. ročník, nie je povinný pre študentov matematiky): návrhové aspekty OOP, tvorba rozsiahlejších projektov, pokročilejšie prvky Javy (napr. práca s databázou, tvorba GUI, vlákna, vybrané Java knižnice, …).

Predstavený obsah týchto predmetov i samotný návrh realizácie úvodného kurzu programovania sú dôsledkom zmien vynútených zmenou programovacieho jazyka úvodného kurzu programovania z Pascalu na jazyk Java v roku 2007. Poznamenajme, že pre jazyk Java doposiaľ neexistuje overená a širšie akceptovaná metodika výučby programovania pre začiatočníkov [12]. Obdobie rokov 2007-2009 bolo pre nás obdobím zbierania skúseností s výučbou programovania v Jave. Výsledkom týchto dvoch rokov výučby je koncepcia úvodného kurzu predstavená v tomto článku, podľa ktorej je predmet *PAZ1a* realizovaný od roku 2009.

[1] Výskum je podporovaný KEGA grantom č. 326-009UPJŠ-4/2010

1 VÝCHODISKÁ KURZU

Pre úvodný kurz programovania na PF UPJŠ je charakteristických niekoľko špecifík, ktoré vyplývajú z rôznych regionálnych špecifík a organizačno-technických obmedzení:

- spoločná výučba pre študentov informatiky, matematiky a medziodborového štúdia informatiky, t.j. väčšina študentov kurzu nemá implicitne vlastnú motiváciu naučiť sa „programovanie“, resp. o význame programovania pre svoju kariéru pochybujú;
- väčšina študentov má slabé alebo žiadne skúsenosti s programovaním (mnohí pokročilejší študenti z košického a prešovského regiónu preferujú štúdium v Bratislave alebo v Českej republike);
- slabé matematické základy (nielen) u študentov medziodborového štúdia.

Z uvedených špecifík vyplýva, že pri návrhu úvodného kurzu bolo nutné sústrediť sa najmä na motiváciu študentov – aby si programovanie obľúbili, alebo aspoň, aby sme ich príliš neznechutili. Z hľadiska kategorizácií rôznych prístupov v úvodných kurzoch programovania [2], možno nami navrhnutý úvodný kurz charakterizovať ako kurz realizujúci prístup „objektové programovanie najprv“ („**OO first**“). Z pohľadu práce [4] ide v úvodnej časti kurzu o kombináciu prístupu „**Using Objects**“ súbežne s prvkami „**Creating Objects**“, ktoré je nasledované časťou využívajúcou prístup „**Creating Objects**“ a ukončený prístupom „**Concepts**“.

Úvodný kurz v predmete *PAZ1a* je obsahovo i formou rozdelený na dve časti. V prvej časti (6 týždňov) postavenej na korytnačej grafike sa študenti oboznámia so základnými stavebnými prvkami programovacích jazykov (premenné, podmienky, cykly, polia), jednoduchou algoritmizáciou sa intuitívne naučia vytvárať vlastné triedy rozširovaním už existujúcich tried. Táto časť je plná (gradovateľných) metafor a mnohé pojmy sú vysvetlené len na intuitívnej úrovni. Prevládajú zadania orientované na čiastkové úlohy. Pre podporu výučby počas tejto časti využívame framework pre **korytnačiu grafiku** – ***JPAZ2*** [6], ktorý je rozšírený aj o ďalšie podporné prvky. V druhej časti kurzu (6-7 týždňov) sa študentom už systematicky predstaví práca s textovými súbormi, výnimky, základné koncepty a princípy OOP. Objekty sú v tejto časti najmä nositeľom údajov, preberané témy sa podrobne vysvetľujú – už nielen na intuitívnej úrovni, ako to je realizované počas prvej časti kurzu. V tejto časti prevládajú prvky projektového vyučovania.

Pred návrhom kurzu bolo prijatých niekoľko rozhodnutí, ktoré ovplyvnili celkové sformovanie kurzu:

- Java v profesionálnom vývojovom prostredí – pri výučbe sme chceli používať voľne dostupné profesionálne vývojové prostredie, ktoré by neobsahovalo veľa „rozptyľujúcich“ prvkov;
- Žiadne vyučovanie Java GUI (Swing, AWT, SWT) – i napriek tomu, že výučbu je možné zatraktívniť tvorbou vlastných okienkových aplikácií (v prostredí NetBeans [10] dokonca s odtienením viacerých technických detailov), nechceli sme v úvode pozornosť študentov smerovať na prvky GUI na úkor hlavného predmetu výučby (aj keď tento smer by bol určite atraktívnejší pre skúsenejších študentov). Medzi hlavné negatíva tvorby vlastného GUI patrí automaticky generovaný zdrojový kód a nevyhnutnosť práce s vláknami. Výsledný zdrojový kód, ktorý začiatočník dopĺňa vlastnými príkazmi, býva veľmi neprehľadný a odpútava pozornosť nežiaducim smerom.

Alternatívnou možnosťou bolo realizovať kurz v prostredí *BlueJ* [3] na základe metodík, ktoré boli preňho navrhnuté, resp. aplikovaním metodiky úvodných kurzov programovania realizovaných v prostredí *BlueJ*. Dôvod, prečo sme sa nerozhodli pre *BlueJ*, bol najmä ten, že toto prostredie je primárne orientované na objektovo orientované programovanie a vizualizáciu objektov – má formu akéhosi mikrosveta. My sme sa rozhodli namiesto vizualizácie nevizuálnych objektov uprednostniť prirodzene vizuálne objekty a rozvoj algoritmizácie s využitím intuitívnych konceptov OOP uprednostniť pred samotným rozvojom princípov OOP. Zároveň sme chceli kurz postaviť tak, aby nebol zviazaný s konkrétnym vývojovým prostredím. Po zvážení dostupných vývojových prostredí sme rozhodli demonštrovať jednotlivé časti kurzu v prostredí *Eclipse* [5]. Opätovne však poznamenajme, že navrhnutý obsah kurzu, ako aj vytvorený podporný softvér (javovská knižnica), nie sú s prostredím *Eclipse* zviazané a je možné ich použiť v ľubovoľnom „pokročilejšom“ vývojovom prostredí pre jazyk Java (s automatickým manažovaním projektov a kompilovaním).

2 KONCEPCIA KURZU

V tejto kapitole si postupne predstavíme najdôležitejšie koncepcie a prvky predstavovaného kurzu.

Korytnačia grafika ako východisko kurzu

Jedným z kľúčových prvkov celej koncepcie predstaveného úvodného kurzu je jeho prvá časť postavená na korytnačej grafike s využitím frameworku *JPAZ2*. Z didakticko-metodického hľadiska je najväčším problémom Javy to, že vytvorenie čo i jednoduchého (napr. graficky orientovaného) programu je pre úplného začiatočníka veľmi zložité a stretne sa pri ňom s početnými technickými detailmi (vrátane OOP), ktoré je v úvode ťažko rozumne vysvetliť. Klasický prístup postavený na konzolových aplikáciách je pre študentov neatraktívny a nijako nezľahčuje zoznámenie sa s programovaním (je istotne motivujúcejšie, ak je výstupom nejaká vizuálno-interaktívna činnosť, než len jeden alebo viacero výpisov do konzoly ako výsledok činnosti programu). Našim cieľom bolo preto navrhnúť metodiku podporenú vlastným frameworkom, ktorá by mala nasledovné vlastnosti:

- vizuálne a interaktívne programy už od začiatku (korytnačia grafika);

- intuitívne uchopenie základného konceptu objektového programovania – objekty vytváram, s vytvorenými objektmi sa „rozprávam" cez volanie metód, vlastné triedy (objekty) viem vyrobiť vylepšením (rozšírením) existujúcich tried;
- objekty už od prvej prednášky;
- všetky objekty vytvorené v programe majú svoju vizuálnu reprezentáciu (objekty je vidieť: okno, objekt v kresliacej ploche), celkový dôraz na vizuálne vnímanie (aj nesprávnej) činnosti vytvorených programov;
- technické detaily sú ľahko automatizovateľné a je za nimi jasná intuícia, ktorá môže byť neskôr rozvinutá do skutočného poznatku.

Celá navrhnutá koncepcia vychádza z predpokladu, že objektovo orientované programovanie je pre študentov prirodzené. Všimnime si, že aj detské programovacie prostredia (mikrosvety) sú objektovo orientované. Ich základná myšlienka spočíva v interaktívnom ovládaní nejakého vizualizovaného objektu vo forme príkazov. Pri tom platí, že ovládaný objekt pozná len nejakú základnú sadu príkazov, ktorú je neskôr možné rozširovať. Tieto príkazy pre objekt sú formou komunikácie s objektom. Inšpirujúc sa detskými programovacími prostrediami sme sa rozhodli v úvodnej časti kurzu hojne využívať analogické metafory, aby sme uľahčili získanie prvých skúseností u študentov s malými a žiadnymi programátorskými skúsenosťami. Hlavným zdrojom inšpirácie pri návrhu obsahu prvej časti kurzu sa stali predmety *Interaktívne programovanie a vizuálne modelovanie* [8] a *Programovanie 1* [2] vyučované na FMFI UK. Našou snahou bolo pokúsiť sa overenú a osvedčenú metodiku výučby programovania v prostredí *Imagine* nejakým spôsobom pretransformovať do „sveta Javy" tak, aby žiadna z adaptovaných alebo vytvorených metafor nepôsobila v Jave „umelo". Zároveň sme chceli, aby pred študentom-programátorom neostali skryté žiadne súvislosti, či nediali sa žiadne „dôležité" veci sa jeho chrbtom. Predovšetkým posledne spomenuté bolo problémom pri výučbe programovania na UPJŠ v rokoch 2007-2009 s využitím frameworku *JPAZ*, ktorý vznikol prispôsobením frameworku *Java Power Tools* [11] pre potreby kurzu na UPJŠ. Aby sme pri návrhu koncepcie neboli obmedzení existujúcimi riešeniami, s ohľadom na potreby vytváraného kurzu a použitých metafor bol vytvorený framework ***JPAZ2*** [6] uvoľnený pod licenciou GNU/GPL.

JPAZ2 framework a jeho objekty

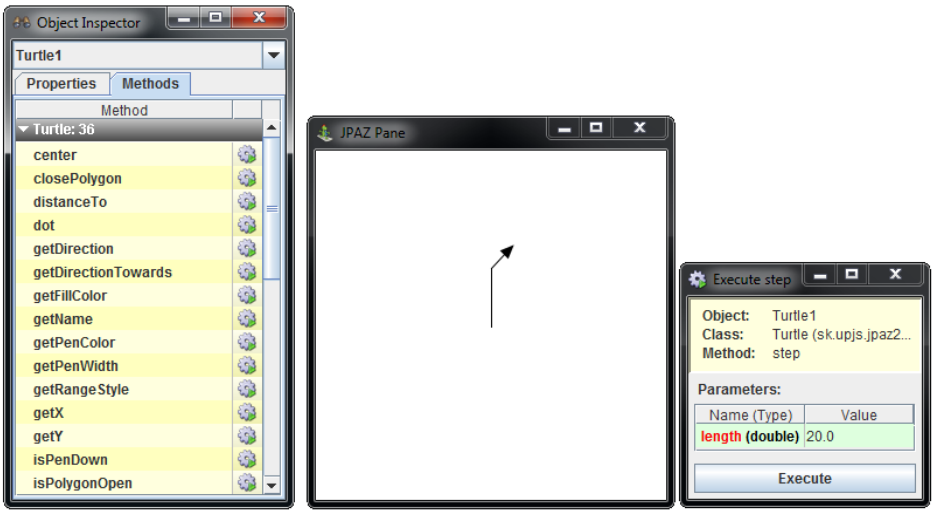
Základnou metaforou v úvodnej časti kurzu sú korytnačky (objekty triedy ***Turtle*** a tried od nej oddedených), ktoré žijú v kresliacej ploche (objekt triedy ***WinPane***). Knižnice realizujúce korytnačiu grafiku (pre Javu a mnohé ďalšie programovanie jazyky), využité v úvodných kurzoch programovania, nie sú ničím novým. Neštandardným prvkom frameworku *JPAZ2* je tzv. **inšpektor objektov** (objekt triedy ***ObjectInspector***). Inšpektor objektov vzhľadom i funkcionalitou vychádza z inšpektorov objektov v IDE prostrediach (Delphi, Lazarus, Visual Studio, …). V porovnaní s nimi však tento inšpektor neslúži na nastavovanie vlastností komponentov pri dizajnovaní formulárov (okien). Cieľom inšpektora objektov v *JPAZ*e je zobrazovať a meniť stav objektu (cez setter-y a getter-y) v dobe behu programu a zároveň slúži na volanie metód objektu – komunikáciu s objektom. Inšpektor objektov tak umožňuje interaktívnu „komunikáciu" s objektmi vytvorenými počas behu programu, čím sa približuje interaktívnym detským programovacím prostrediam. Implementačne inšpektor objektov využíva najmä vlastnosť Javy zvanú *reflection*, ktorá je skombinovaná s čítaním ladiacich informácií z bajt-kódu tried. Inšpektor objektov nie je zviazaný s vytvorenými triedami *JPAZ2* framework-u. Študenti tak majú možnosť použiť ho na volanie metód objektov vlastných tried (najčastejšie rozširujúcich triedu *Turtle*). Tento prvok interaktívnosti je ďalším z významných aspektov navrhnutého úvodného kurzu programovania v Jave. Prvý kód, s ktorým sa študenti stretnú, je nasledovný:

```
import sk.upjs.jpaz2.*;

public class Spustac {

        public static void main(String[] args) {
                WinPane plocha = new WinPane();
                Turtle jozko = new Turtle();
                plocha.add(jozko);

                ObjectInspector oi = new ObjectInspector();
                oi.inspect(jozko);
        }
}
```

Obr. 1 Zdrojový kód prvej aplikácie používajúcej JPAZ2 framework

V úvode metódu *main* nazývame spúšťacou metódou v triede *Spustac* - využíva sa len za účelom spustenia programu a vďaka podpore prostredia *Eclipse* sa vytvára študentmi „zautomatizovanou" klikacou činnosťou zahrňujúcou aj pripojenie knižnice *jpaz2.jar* k projektu. Ako môžeme vidieť v zdrojovom kóde (obr. 1), jednotlivé kroky majú jednoduché a intuitívne vysvetlenie. Najprv sa vytvorí objekt kresliacej plochy, pričom na komunikáciu s ním sa použije premenná *plocha*. Ďalej sa vytvorí objekt korytnačky a na komunikáciu s ním sa použije premenná *jozko*. Potom vytvorenej ploche cez „komunikačnú" premennú *plocha* povieme, aby zatiaľ korytnačku „bezdomovca" pridala na „svoje územie", atď. Poznamenajme, že v úvode kurzu sa študentom nijako „formálne" nevysvetľuje, čo sú to objekty a triedy – začne sa rovno s ich používaním. Pomerne často využívame rôzne metafory. Skúsenosti ukazujú, že úvodná výučba v intuitívnej rovine nie je nijako na škodu a neskôr

po zautomatizovaní, precvičovaní a použití v rôznych iných kontextoch si študenti sami rozšíria poznatok do požadovanej úrovne – vytvoria si vlastný mentálny model objektov a objektovo orientovaného programovania.



Obr. 2 Okno inšpektora objektov, kresliacej plochy a okno na zavolanie metódy objektu

Metafora vylepšovania

Dôležitým konceptom, ktorý využívame už od prvej prednášky je (opäť prirodzený) koncept vylepšovania (rozširovania - *extends*) niečoho, čo už spravil niekto iný. Samotný pojem dedičnosti sa používa až v druhej časti kurzu. Základná motivácia je tá, že objekty triedy *Turtle* nie sú veľmi múdre a preto ich budeme vylepšovať pridávaním vlastných metód (príkazov) – budeme vytvárať nové objekty vlastného „vyšľachteného“ druhu (triedy) korytnačiek, ktoré budú vedieť všetko to, čo vedeli pôvodné, avšak naučíme ich aj niečo nové s využitím toho, čo už vedia. Jedným z úvodných príkladov prestaveným v kurze je vytvorenie triedy *MojaKorytnacka*, ktorej objekty už budú vedieť metódu na nakreslenie rovnostranného trojuholníka (v ľahšom variante je vhodnejší príklad štvorca kvôli uhlu otočenia). V postupne vygradovanej verzii (začne sa bez parametrov, neskôr sa pridá „for“ – cyklus ako „magická“ formulka na opakovanie skupiny príkazov bez bližšieho vysvetlenia a s ukázaním toho, ako sa „nastavuje“ počet opakovaní) bude kód triedy *MojaKorytnacka* vyzerať nasledovne:

```
import sk.upjs.jpaz2.*;

public class MojaKorytnacka extends Turtle {

        public void trojuholnik(double strana) {
                for (int i=0; i<3; i++) {
                        this.step(strana);
                        this.turn(120);
                }
        }
}
```

Obr. 3 Príklad rozširovania triedy *Turtle*

Všimnime si aj použitie kľúčového slova **this**, aby každé volanie metódy v zdrojovom kóde bolo v jednotnom tvare: *volanýObjekt.**volanáMetóda***. Ďalším dôvodom pre použitie *this* je zdôraznenie toho, že objekt *this*, ktorý metódu vykonáva, pri jej vykonávaní „volá“ iné „svoje“ metódy, ktoré už pozná.

Väčšina úloh v úvodnej časti je študentom zadávaná vo forme: „Vytvorte triedu *MojaKorytnacka*, ktorá rozširuje (vylepšuje) triedu *Turtle* tak, aby korytnačky triedy *MojaKorytnacka* mali metódy, ktoré ...“. Neskôr z dôvodu spracovania udalosti myši sa pridáva vytváranie tried rozširujúcich triedu kresliacej plochy – *WinPane*. Samotný koncept triedy sa študentom vysvetľuje ako akási šablóna („genetická informácia“) – vzor správania sa, podľa ktorého sa objekty danej triedy správajú, keď im zadáme nejaký príkaz.

Metafora rodného čísla

Neprítomnosť smerníkov v jazyku Java je často prezentovaná ako nedostatok jazyka pri jeho použití v úvodnom kurze programovania. Zrejme príčinou tohto názoru môže byť neraz snaha vyučujúcich (v snahe zjednodušiť pochopenie vysvetľovaných pojmov) stotožňovať premennú uchovávajúcu referenciu na objekt s objektom samotným. Rozhodli sme sa preto v tomto smere veľmi dôsledne prezentovať jednotlivé pojmy. Počas vysvetľovania zdôrazňujeme, že lokálne premenné a objekty sa nachádzajú v „rôznych svetoch“. O (lokálnych) premenných, s ktorými sa študenti oboznamujú ako prvými, hovoríme, že vznikajú a zanikajú postupne počas vykonávania metód, t.j. žijú akoby „v programe“. Naopak pre objekty používame metaforu „sveta objektov“, v ktorom objekty, vytvorené použitím príkazu **new**, vznikajú a žijú nezávisle na

premenných v programe. Premenné referenčného typu sú najprv prezentované ako komunikátory s menom, ktoré slúžia na komunikáciu s objektom nejakej triedy (špecifikovanej pri vytvorení) – t.j. ako komunikátory prepájajú „svet programu“ so „svetom objektov“. Premenná-komunikátor sa s vytvoreným objektom „prepája“ príkazom priradenia. Pri volaní metód premenná-komunikátor volá metódu nad objektom, s ktorým je prepojená. V polovici prvej časti kurzu je táto metafora upravená a premenné referenčného typu sú funkcionalitou prirovnávané k premenným primitívneho typu. Tak ako premenná typu *int* uchováva celé číslo, premenná referenčného typu uchováva „rodné číslo“ nejakého objektu určenej triedy, ktoré jedinečným spôsobom (podobne ako ľudí na Slovensku) identifikuje objekty žijúce vo „svete objektov“. Rodné číslo objektu je hodnotou vrátenou pri vytvorení („narodení“) objektu cez konštrukciu **new**. Podobne ako primitívne hodnoty (napr. čísla) aj „rodné čísla“ objektov môžeme „skopírovať“ z jednej premennej do inej. V skutočnosti potom pri volaní metódy nad premennou referenčného typu je táto metóda volaná nad objektom, ktorého „rodné číslo“ je v nej uložené. Metafora „rodného čísla“ zodpovedá konceptu referencie a zároveň je akýmsi ekvivalentom adresy v pamäti. Túto metaforu možno adaptovať aj pri výučbe OOP v jazyku Object Pascal (napr. po „zničení“ objektu cez *Free*, jeho „rodné číslo“ ostáva uložené v premennej a tak volanie metódy po uvoľnení spôsobuje problémy – objekt s daným „rodným číslom“ už neexistuje), kde sa „rodné číslo“ stotožní s adresou objektu. Pri tejto metafore možno zdôrazniť, že pole prvkov referenčného typu neobsahuje samotné objekty, ale len ich „rodné čísla“ – objekty samotné sú vždy „vo svete objektov“ (metafora pre *heap*). Podobne aj samotné polia sú prezentované ako „špeciálne skriňovo-kontajnerové“ objekty, ktoré majú akési privilegované postavenie. Cieľom týchto metafor je myšlienkovo pripraviť študentov na neskoršie dynamické údajové štruktúry v Jave (spájané zoznamy, stromy), či prechod na jazyk C (u informatikov), resp. Pascal (u medziodborových študentov), a prácu so smerníkmi.

Projektové vyučovanie princípov OOP a objektového návrhu

Hlavným zámerom druhej časti kurzu (druhej polovice prvého semestra) je prirodzenou formou naučiť študentov vytvárať dobrý objektový návrh, t.j. vhodne reprezentovať údaje a správne dekomponovať a umiestniť funkcionalitu danú zadaním, a popri tom objasniť princípy OOP. Rozhodli sme sa z prvého semestra úplne vylúčiť algoritmy (triedenia, rekurzie, dynamické programovanie) a zložitejšie údajové štruktúry (stromy, grafy). Zameriavame sa na zoznamy objektov, prácu s nimi (pridávanie, odoberanie, zmena, vyhľadávanie) a ukladanie a nahrávanie dát do/z súboru.

Pri práci s dvojrozmernými poľami a zoznamami objektov je prirodzená potreba ukladania/načítavania týchto dát v nejakej perzistentnej forme, aby nebolo nutné tieto dáta zakaždým generovať alebo ručne pridávať. Rozhodli sme sa tieto dáta ukladať v textových súboroch. Binárne súbory sme z úvodného kurzu vylúčili. Práca s nimi je síce jednoduchšia (cez serializované objekty), ale výsledok uloženia je mimo programu neinterpretovateľný a nemeniteľný. Nakoľko je pri práci s textovými súbormi v Jave nutné pracovať s kontrolovanou výnimkou *FileNotFoundException*, téma o výnimkách je zaradená pred tému textových súborov. Aby študenti lenivo neskĺzavali k vyhadzovaniu výnimiek, ale boli donútení k ich odchyteniu a spracovaniu, je predstavené len spracovanie výnimiek v bloku *try-catch-finally*. Výnimka sa tým pádom využíva od prvej chvíle na to, na čo je určená, t. j. na riešenie výnimočných situácií bez pádu programu. Samotná práca s textovými súbormi je od Javy verzie 5 našťastie výrazne jednoduchšia ako v predchádzajúcich verziách. Už nie je potrebné predstavovať prúdové triedy, ale vystačíme si s triedami *Scanner* a *PrintWriter*.

Pri vysvetľovaní jednotlivých pojmov a konceptov OOP sme sa snažili, aby žiaden príklad nepôsobil umelo, ale aby použitie daného konceptu bolo prirodzené. Na prednáške sme preto zvolili formu projektového vyučovania na projekte „**Zoznam filmov**“, ktorý vytvárame počas 3 prednášok. Napriek tomu, že sa v tomto projekte často využíva pridávanie a odoberanie filmov zo zoznamu, používame ako základnú údajovú štruktúru pole filmov namiesto, v reálnom projekte asi vhodnejšej, implementácie rozhrania *List*. Pokladáme to za dôležitú prípravu pre ďalší semester (predmet *PAZ1b*), v ktorom je pole základným údajovým typom v mnohých algoritmoch, ale aj na programovanie v nižších programovacích jazykoch, ktoré dynamické údajové štruktúry štandardne neposkytujú (jazyk C, jazyk Pascal). Študenti si zároveň neskôr majú možnosť uvedomiť, ako asi môže vyzerať interná implementácia tried *Java Collections Frameworku*.

Projekt začíname víziou vytvorenia systému na spravovanie zbierky DVD filmov. Hlavný dôraz sa kladie na analýzu zadania, v ktorom je potrebné identifikovať dáta, ktoré spracúvame a funkcionalitu, ktorá je požadovaná. Základným cieľom je pochopiť ideu zapúzdrenia, nosného konceptu OOP, a s tým súvisiace spravovanie inštančných premenných cez konštruktory, settery a gettery.

Na ďalšej prednáške je projekt rozšírený o filmy na videopáskach a filmy v počítači. Pre každý nosič máme spoločné aj špecifické vlastnosti. Z toho prirodzene vyplynie potreba nadpojmu – predka v hierarchii dedičnosti (trieda *Film*), ktorý bude spravovať iba spoločné vlastnosti. Metóda na výpis informácií o médiu je prirodzene polymorfnou metódou volanou v cykle cez všetky médiá napríklad pri výpise filmov v zbierke. Na upevnenie poznatkov o dedičnosti a polymorfizme je zoznam filmov ukladaný a načítaný do/z textového súboru. Je potrebné spomenúť, že na cvičeniach sa tento projekt nerealizuje, ale využívajú sa iné príklady na dedičnosť a polymorfizmus (napr. „klasický“ príklad: tvary – kruh, obdĺžnik, trojuholník).

Na poslednej „projektovej“ prednáške, po vysvetlení abstraktných tried a metód, sa rodičovská trieda *Film* vyhlási za abstraktnú a podobne sú za abstraktné vyhlásené aj niektoré jej metódy, pre ktoré sme si v predchádzajej prednáške povedali, že „sa dohodneme, že ich volať nebudeme“.

Pri vysvetľovaní pojmu rozhrania (*interface*) si pripomenieme, že za začiatku projektu sme si navrhli nejakú reprezentáciu dát do štruktúr, ktorá nemusí byť nutne ideálna pre všetky prípady. Iná implementácia by mohla využívať iné pamäťové dátové štruktúry, alebo aj štruktúry v databáze, či inom type úložiska. Predstavíme tak rozhranie ako spôsob zapísania požadovanej funkcionality zo zadania úlohy, teda ako kontrakt, ktorý je potrebné dodržať. Spôsob naplnenia týchto požiadaviek je závislý od programátora a spôsobu riešenia, ktorý si zvolil. Pojem rozhraní a ich rôznych implementácií je neskôr upevňovaný pri vysvetľovaní *Java Collections Framework*.

Prirodzenou požiadavkou pre projekty typu „zoznam objektov“ je triedenie. Nakoľko sme sa rozhodli nepreberať v prvom semestri žiadne náročnejšie algoritmy, využívame predpripravenú triediacu metódu *Arrays.sort()* resp.

Collections.sort(). V reálnom programátorskom živote si už asi nik triediaci algoritmus neprogramuje a práve použitie takejto metódy je prirodzené. Je to tiež pekná ukážka využitia rozhraní *Comparable* resp. *Comparator* ako role pri triedení zoznamu ľubovoľných objektov.

Dobré návyky pri programovaní

Až do okamihu vysvetlenia mnohých pojmov a konštrukcií jazyka Java a OOP všeobecne, sa prednášajúci a cvičiaci, a tým pádom aj študenti, počas kurzu riadia množinou pravidiel písania zdrojového kódu. Tieto pravidlá majú za cieľ vytvoriť „dobré návyky" písania programov.

- Metóda *main* je vždy v samostatnej triede bez ďalších metód (zvyčajne nazývanej *Spustac*).
- Každá trieda, ktorá nie je spúšťacou, je rozšírením nejakého potomka triedy *Object* (hlavne rozšírením tried *Turtle* alebo *WinPane*). To platí až do 9. týždňa, kedy začíname vytvárať nezávislé vlastné triedy.
- Volania metód a pristupovanie k inštančným premenným rovnakej triedy alebo triedy predka sa realizuje cez „*this*". Od tohto pravidla sa upúšťa zhruba 2 týždne po predstavení inštančných premenných.
- Všetky inštančné metódy majú viditeľnosť „*public*". Toto pravidlo je užitočné pre inšpektor objektov, ktorý zobrazuje iba verejné metódy tried. Samozrejme po vysvetlení viditeľnosti sa viditeľnosť mnohých metód obmedzuje a verejné už ostávajú len niektoré vhodné metódy.
- Všetky inštančné premenné majú viditeľnosť „*private*". Tohto pravidla sa držíme aj po vysvetlení viditeľnosti z dôvodov osvojenia si konceptu zapúzdrenia. K inštančným premenným sa z iných tried pristupuje iba prostredníctvom konštruktorov, setterov a getterov.
- Statické metódy a premenné sa nepoužívajú (metóda *main* je chápaná ako „čosi" špeciálne) až do posledného týždňa kurzu, kedy sa vysvetlí, kedy je ich použitie vhodné.
- Výnimky sa nevyhadzujú z metód až do posledného týždňa kurzu. Snažíme sa študentom prizvukovať, že ak sa dá, výnimočné situácie by sa mali riešiť a nie nechať ukončiť program s vyhodením výnimky do konzoly.

3 SYLABUS KURZU

Obsah kurzu a hlavných tém po jednotlivých týždňoch (3-hodinová prednáška, 2+2 hodiny cvičení):

1. prvé stretnutie s Javou a *JPAZ2* frameworkom, vytvorenie *JPAZ2* projektu v *Eclipse*, interaktívne ovládanie korytnačky cez inšpektor objektov, intuitívne predstavený pojem objektu, triedy (šablóny pre objekty) a metódy, jednoduchá korytnačia grafika cez rozšírenie triedy *Turtle*, parameter ako náhrada užívateľom zadanej hodnoty, *for*-cyklus s pevným počtom opakovaní (len ako „formulka" bez vysvetlenia);
2. *for*-cyklus s variabilným počtom opakovaní, lokálna premenná a jej typ, aritmetické výrazy, náhodné čísla, náhodné pochôdzky, podmienkový príkaz (demonštrovaný na náhodných pochôdzkach), logické výrazy;
3. *while*-cyklus (špirály), vysvetlenie *for*-cyklu a jeho častí, metódy vracajúce hodnotu (korytnačka-matematik), práca s číslami (napr. práca s ciframi čísla), debugovanie programov, referencia a premenná referenčného typu;
4. rozdiel medzi premennou primitívneho typu a premennou referenčného typu, znaky, práca s objektmi triedy *String* (reťazce a základné algoritmy na prácu s reťazcami), vylepšovanie kresliacej plochy: myšacie udalosti a inštančné premenné;
5. polia („s pevným počtom prvkov"): pole referencií (pole korytnačiek v projekte „Korytnačí futbal" vrátane zmeny tvaru korytnačky) a pole primitívnych hodnôt (projekt „Záhradka s kvetinovými záhonmi"), základné poľové algoritmy (naj-prvok poľa, zistenie, či všetky prvky poľa majú nejakú vlastnosť);
6. „zmena" počtu prvkov poľa (nová lopta v korytnačom futbale), korytnačka-poliarka ako zovšeobecnenie algoritmov na prácu s poľom, dvojrozmerné pole a jednoduché algoritmy (projekt „Piškvorky").
7. výnimky: *stack trace*, spôsob predchádzania výnimočným stavom overovaním vstupných hodnôt, využívanie blokov *try-catch-finally*; práca so súbormi: metadáta o súboroch v objektoch triedy *File* a práca s obsahom textových súborov cez objekty tried *PrintWriter* a *Scanner*; spôsoby konverzie reťazcov do iných typov;
8. praktická práca so súbormi: zápis a načítanie matice čísiel z textového súboru, spracovanie viacerých typov hodnôt vo vstupnom textovom súbore (táto prednáška neprináša nové koncepty, pretože študenti venujú svoj čas hlavne príprave na polsemestrálny test z obsahu prvej časti kurzu, ktorý sa koná v čase praktických cvičení – nová látka by v dôsledku toho nebola precvičená);
9. na projekte „Zoznam filmov na DVD" je vysvetlená zapúzdrenosť, konštruktory s parametrami, hierarchia konštruktorov, koncept getterov a setterov a preťažovanie metód (poznamenajme, že počas tejto prednášky sú prvý krát v kurze vytvorené vlastné triedy, ktoré nie sú rozšírením - oddedením predpripravených tried).
10. dedičnosť a polymorfizmus: pokračovanie projektu s filmami, ktoré už môžu byť uložené aj na videopáskach a v počítači, zápis zoznamu filmov do súboru a jeho načítanie zo súboru;
11. abstraktné triedy a metódy, rozhranie (*interface*) ako kontrakt a ako rola, používanie balíčkov, modifikátory viditeľnosti, triedenie cez *Arrays.sort()* s využitím rozhraní *Comparable* a *Comparator*.
12. *Java Collections Framework*: trieda *ArrayList*, obaľovacie triedy primitívnych typov a autoboxing, rozhranie *List* a jeho implementácie *ArrayList* a *LinkedList*, rozhranie *Set* a jeho implementácia *HashSet*, metódy *equals* a *hashCode*, *for-each* cyklus, rozhranie *Map* a jeho implementácia *HashMap*.

13. výnimky: vyhadzovanie výnimiek, vytváranie vlastných výnimiek, prebaľovanie výnimiek, výnimky a dedičnosť, kontrolované vs. nekontrolované výnimky, chyby, statické metódy a premenné a prípady ich použitia v reálnych projektoch.

4 ORGANIZÁCIA VÝUČBY

Výučba predstaveného úvodného kurzu je z organizačno-historických dôvodov realizovaná vo forme trojhodinovej prednášky nasledovanej dvomi hodinami cvičení v počítačovej učebni a dvomi hodinami v klasickej učebni vybavenej tabuľou, dátovým projektorom a počítačom. Keďže programovanie považujeme za dynamický proces, ktorého výsledkom je zdrojový kód, dôležitým prvkom výučby je „programovanie na živo“. Iba vtedy má študent možnosť uvidieť postupnosť krokov a myšlienok, na základe ktorých program vzniká. Uvidí, že program často nevzniká len „riadok za riadkom“ ako je vo výslednom zdrojovom kóde, ale je to neraz iteratívny proces, počas ktorého sa môžu vyskytnúť aj chyby, ktoré sú priebežne odhaľované a opravované. Pri prednáškach využívame dva dátové projektory. Na jeden je premietaná prezentácia k prednáške a na druhom sa „naživo“ programujú demonštračné programy. Prednáška je tak často vo forme komentovaného vytvárania zdrojového kódu. Pri cvičeniach v počítačovej učebni využívame model, v ktorom je na stránke predmetu zverejnená sada úloh, ktoré sa budú na cvičení riešiť. Šikovnejší študenti ich môžu riešiť samostatne. Priemerní študenti ich často riešia spoločne s lektorom, ktorý jednotlivé úlohy s istým oneskorením naprogramuje na dátový projektor (aby študenti mali priestor vytvoriť aj vlastné riešenia a nielen odpisovali z projektora). Pri cvičeniach v klasickej učebni sa kladie dôraz na diskusiu o riešených úlohách. Študenti však po vyzvaní kód nepíšu na tabuľu, ale programujú ho na dátový projektor. Zdrojové kódy z prednášok i kódy vytvorené na cvičeniach sú zverejňované.

Na hodnotení študentov sme sa rozhodli aplikovať systém, v ktorom bod nie je „jednotkou výkonu“ ale jednotkou odvedenej práce. Známka je súčtom nazbieraných bodov za aktivity troch kategórii. Najväčšia ponuka bodov je za riešenie domácich zadaní (zahrňujúcich aj projekt), potom za body získané počas praktických programátorských testov (midterm + „skúška“ – tie sú tzv. open-book, t. j. študenti majú povolené využívať ľubovoľné pasívne informačné zdroje) a nakoniec za body získané počas kontrolných desaťminútoviek písaných na papier každý týždeň. Aby sa eliminovali podvody pri riešení domácich zadaní, pre získanie hodnotenia musí študent v každej z kategórií získať predpísané povinné minimum bodov.

5 VYHODNOTENIE

Výučba úvodného kurzu spôsobom popísaným v tomto článku bola doposiaľ realizovaná počas dvoch akademických rokov. Je ťažké porovnávať úspešnosť navrhnutého kurzu, keďže súbežne nebola realizovaná výučba alternatívnym spôsobom. Avšak na základe našich doterajších skúseností môžeme skonštatovať, že u študentov, ktorí predmet absolvovali, je viac ako dostatočná miera pochopenia konceptov OOP i základov programovania a algoritmizácie. Úvodný kurz počas zimného semestra v akademickom roku 2010/2011 úspešne absolvovalo 52 študentov zo 78 študentov, ktorí na začiatku letného semestra ešte nemali ukončené štúdium (študentov, ktorí ukončili štúdium, nezarátavame, keďže je pravdepodobné, že u nich došlo k neúspešnému absolvovaniu viacerých predmetov). I napriek pôvodným obavám z jazyka Java a „OO first“ prístupu sa ukázalo, že u študentov je prezentovaný úvodný kurz v uvedenej forme úspešný. Na základe anonymných dotazníkov realizovaných na konci každého semestra výučby predmetov *PAZ1a* a *PAZ1b* sme získali niekoľko pozorovaní, ktoré sú v súlade s tým, čo sme vypozorovali aj počas samotnej výučby. Paradoxom vzhľadom k veku študentov je to, že aj keď úvodná časť bola obsahovo (ale aj formou prednášania) plná metafor a „detských“ prvkov, väčšina študentov ju prijala veľmi pozitívne. Bolo to pre nich zábavnejšie, než výučba iných predmetov prezentovaných klasickou formou. Veľa študentov bolo pozitívne motivovaných k programovaniu (diskusie o projektoch i programovaní na sociálnych sieťach, vytvorenie kvalitných a zaujímavých projektov prezentovaných počas verejnej obhajoby projektov). Isté problémy sa ale objavili u niektorých pokročilejších študentov, ktorí sa kvôli zdanlivej nenáročnosti na predmet dostatočne nesústredili, čo sa prejavilo v podstatne horších výsledkoch, než sme očakávali. V dôsledku toho sa objavili prípady, kedy študenti s horšími východiskovými predpokladmi výrazne predbehli študentov s lepšími východiskovými predpokladmi. Riešením by mohlo byť oddelenie týchto študentov do špeciálnych skupín. Na druhej strane by takéto rozdelenie spôsobilo, že v zmiešaných študijných skupinách (odborovo i skúsenosťami) by chýbali pokročilejší študenti, čo by pravdepodobne viedlo k zníženiu súťaživosti ich menej skúsených kolegov. V nezmiešaných skupinách tvorených (najmä medziodborovými) študentmi so slabšími počiatočnými skúsenosťami sa v minulosti zvyklo prejaviť výrazne slabšie napredovanie v porovnaní so zmiešanými skupinami.

ZÁVER

V článku sme stručné predstavili úvodný kurz programovania v jazyku Java na PF UPJŠ. V kurze adaptujeme osvedčené a overené metodiky z detských programovacích jazykov do jazyka Java. Dôraz je kladený na vizuálnu pútavosť a interaktívnosť vytváraných programov. Pri realizácii kurzu sa ukázalo, že objektovo orientované programovanie už od začiatku prepletené s úvodom do programovania a algoritmizácie, ktoré sú nasledované systematickým výkladom princípov OOP a práce s údajmi, je použiteľnou koncepciou pre úvodný kurz programovania v jazyku Java. Študijné výsledky študentov i samotné anonymné hodnotenia študentov v dotazníkoch k predmetu preukazujú, že navrhnutý kurz spĺňa požiadavky kladené na úvodný kurz programovania.

LITERATÚRA

[1] ACM *Computing curricula 2001*. J. Educ. Resour. Comput. 1, 3es (Sep. 2001)

[2] BLAHO, A.: *Analýza vyučovania objektového programovania v bakalárskych programoch štúdia informatiky.* Dizertačná práca FMFI UK, 135 s., 2010.

[3] BARNES, D., KÖLLING, M.: *Objects First with Java: A Practical Introduction using BlueJ*. Prentice Hall / Pearson Education, 2008, ISBN: 978-0-13-606086-4.

[4] BENNEDSEN, J., SCHULTE, C.: What does "Objects-First" Mean? An International Study of Teachers' Perceptions of Objects-First. In: Lister, R. und Simon, Hrsg. (2007), Koli Calling 2007, Finland, ACS.

[5] ECLIPSE FOUNDATION: *Eclipse Classic* [počítačový program], ver. 3.6.1. [citované 24.1.2011] Dostupné na internete: <http://www.eclipse.org/>

[6] GALČÍK, F.: *JPAZ2* [počítačový program], ver. 2, [citované 24.1.2011] Dostupné na internete: <http://code.google.com/p/jpaz2/>

[7] GALČÍK, F. – GURSKÝ, P. *Stránka predmetu PAZ1a* [webová stránka] , [citované 24.1.2011] Dostupné na internete: <http://web.ics.upjs.sk/paz/>

[8] KALAŠ, I.: *Stránka predmetu Interaktívne programovanie a vizuálne modelovanie* [webová stránka] , [citované 24.1.2011] Dostupné na internete: <http://www.edi.fmph.uniba.sk/10311>

[9] Liberal Arts Computer Science Consortium 2007. *A 2007 model curriculum for a libe-ral arts degree in computer science*. J. Educ. Resour. Comput. 7, 2 (Jun. 2007)

[10] ORACLE CORPORATION: *NetBeans* [počítačový program] , ver. 6.9.1 [citované 24.1.2011] Dostupné na internete: <http://netbeans.org/>

[11] RASALA, R. – PROULX, V.: *Java Power Tools* [počítačový program] , ver. 2.7.0 [citované 24.1.2011] Dostupné na internete: <http://www.ccs.neu.edu/jpt/>

[12] SALANCI, Ľ. a kol.: *Didaktika programovania, Ďalšie vzdelávanie učiteľov základných škôl a stredných škôl v predmete informatika*, Bratia Sabovci, Zvolen, 2010, 36 s., ISBN 978–80–8118–065–1.

AUTORI

FRANTIŠEK GALČÍK, RNDR., PHD.

Ústav informatiky,
Prírodovedecká fakulta,
Univerzita Pavla Jozefa Šafárika
v Košiciach,
Jesenná 5,
040 01 Košice
frantisek.galcik@upjs.sk

PETER GURSKÝ, RNDR. PHD.

Ústav informatiky,
Prírodovedecká fakulta,
Univerzita Pavla Jozefa Šafárika
v Košiciach,
Jesenná 5,
040 01 Košice
peter.gursky@upjs.sk

RÓBERT NOVOTNÝ, RNDR.

Ústav informatiky,
Prírodovedecká fakulta,
Univerzita Pavla Jozefa Šafárika
v Košiciach,
Jesenná 5,
040 01 Košice
robert.novotny@upjs.sk