Výuka objektově orientovaného programování žáků základních a středních škol

Rudolf Pecinovský[1]

[1]Amaio Technologies, Inc., Třebohostická 14, 100 00, Praha 10
rpecinovsky@amaio.cz

Abstrakt. Příspěvek se zabývá hlavními zásadami správné výuky programování a předvádí možnosti jejich implementace při výuce OOP ve školách a zájmových zařízeních navštěvovaných žáky základních a středních škol. Demonstruje, jak lze přirozeně začít výklad přímo prací s třídami a objekty a od ní pak postupně a plynule přejít k dalším konstrukcím. Ukazuje na důležitost správné volby použitého vývojového prostředí a předvádí, jak lze vytčených cílů dosáhnout při výuce jazyka Java využitím vývojového prostředí BlueJ.

Klíčová slova: OOP, výuka, Java, BlueJ

1 Trocha historie

Svůj první program jsem napsal v roce 1975. Učebnice, které mne tehdy uváděly do tajuplného světa programování, začínaly všechny skoro úplně stejně. Pomineme-li úvodní kapitoly o historii daného programovacího jazyka, historii počítačů, základních principech překladu apod., začínali jejich autoři výkladem jednotlivých datových typů a možných způsobů zápisu literálů, pokračovali vlastnostmi proměnných aby se po následném výkladu programových konstrukcí vítězně probojovali k procedurám a funkcím, které pasáž o základních vlastnostech jazyka uzavíraly. Vzhledem k použitému postupu výkladu měly tyto učebnice několik nevýhod:

- Student se musel dlouho prokousávat vstupními pasážemi než se toho dozvěděl dostatek k tomu, by mohl vytvořit fungující program a mohl přímo na počítači ověřit získané poznatky.
- Ukázkové příklady většinou řešily pouze triviální, nezajímavé problémy, které pro studenty nebyly nijakou výzvou.
- Tento postup neumožňoval vštěpovat studentům od samého začátku zásady strukturovaného a modulárního programování, protože základní kameny tohoto přístupu, procedury a funkce, byly probírány až na závěr úvodního bloku (byly to učebnice jazyků a v těch se tehdy o strukturovaném a modulárním programování nehovořilo).

První učebnice, která tento přístup alespoň částečně změnila (přesněji první z těch, s nimiž jsem se setkal), byla kniha Programovací jazyk C ([15]). Ta začínala prográmkem *Hello World*, který se rychle stal „povinným“ úvodním programem učebnic libovolného programovacího jazyka.

Tato drobná změna sice umožnila, aby si student mohl hned od počátku všechny vysvětlované konstrukce vyzkoušet, ale další dvě uvedené výhody nijak neřešila. Nemám jí to za zlé, protože byla určena především těm, kdo již o programování něco věděli a chtěli se seznámit s programovacím jazykem C. Nicméně pro vstup do světa programování nepovažuji ani tento přístup za optimální.

Později jsem se setkal s několika převážně americkými učebnicemi, jejichž autoři zvolili trochu odlišný přístup. Začínali většinou klasickým *Hello World*, avšak nepokračovali systematickým výkladem jazyka, ale místo toho se snažili předkládat k řešení různé příklady a v průběhu jejich řešení vysvětlovat ty rysy jazyka, které bylo možno při řešení dané úlohy využít.

Na tomto způsobu výkladu je krásné, že autor vše vysvětluje na praktických příkladech. Když si však některou z těchto učebnic prohlédnete pozorněji, zjistíte, že tento přístup je v řadě směrů nakonec ještě méně vhodný, než přístupy, o nichž jsem hovořil na počátku. Výklad je často zmatený a student při něm nikdy neví, zda je již daný rys jazyka probrán, anebo zda bude ještě v budoucnu doplněn. A to již vůbec nemluvím o tom, že se v takovéto knize velice špatně hledá výklad vlastnosti, jejíž některé rysy si potřebuji osvěžit.

Kromě toho nepovažuji za vhodný postup, při němž před žáky předložím problém, který nemohou vyřešit, a pak tahám z klobouku rysy jazyka, s jejichž pomocí problém řeším, ale které žáci doposud neznají. Spíš zastávám opačný přístup: nejprve vysvětlím (samozřejmě na příkladu) nový rys a pak si ukážeme, kde jej můžeme s výhodou využít a zkusíme si vyřešit jiný příklad, kde právě naučenou látku aplikujeme.

2 Zásady správné výuky programování

Dosti ale již o tom, jak si myslím, že výuka programování probíhat nemá. Podívejme se nyní na to, jak by bylo možné vyučovat programování tak, abychom se s žádnou z uvedených nevýhod nesetkali.

Proberme si nejprve zásady, které by měl správný výklad splňovat. Jsem přesvědčen o tom, že i vám budou tyto zásady připadat naprosto zřejmé; nejspíše dokonce natolik zřejmé, že budete považovat za zbytečné se o nich vůbec zmiňovat. Nicméně podíváte-li se do učebnic programování, zjistíte, že je jejich autoři často nedodržují.

2.1 Co nejdříve umožnit tvorbu programů

Aby se žák mohl něco doopravdy naučit, nestačí mu, když se dozví základní pravdy, ale musí si vše sám „osahat“. Pokud vykládanou konstrukci několikrát sám nepoužije, za chvíli vše zapomene (přesnější by bylo říci: zapomene ji daleko rychleji).

Tuto zásadu se naštěstí většina současných učebnic snaží dodržovat. Jejich úvodní programy se však často ani nepokoušejí vyhovět zásadám, o nichž budu hovořit vzápětí.

2.2 Nepředbíhat, tj. nepoužívat prvky jazyka, které ještě nebyly vyloženy

S touto zásadou budou asi všichni po prvním přečtení souhlasit. Jakmile ji však budeme chtít aplikovat do důsledků, řada vyučujících se začne bránit. Typickým příkladem takovéhoto přebíhání je většina učebnic začínajících tvorbou onoho výše zmíněného „povinného“ programu *Hello World*. V tomto programu se totiž používá záplava konstrukcí, které dosud nebyly vysvětleny. Studenti proto je neznají a nemusí být proto vždy schopni detekovat a odstranit případnou chybu.

Mnozí to řeší tak, že na počátek knihy vloží kapitolu, v níž stručně popíší nejzákladnější rysy použitého jazyka a vývojového prostředí. Je to jeden ze způsobů, jak se z výše popsaného problému vylhat. Vede ale bohužel k tomu, že žáci pak používají řadu prvků, jejichž skutečný význam se dozví až mnohem později. Přiznám se, že já jsem zastánce uspořádaného výkladu, při němž je každá vlastnost jazyka vysvětlena až ve chvíli, kdy je ji možné vyložit v rozumné šíři a hloubce.

2.3 Informace je třeba předávat po malých soustech

Všichni se jistě shodneme i na tom, že čím méně nových poznatků svým žákům předložíme, tím větší je naděje na to, že si je studenti zapamatují a naučí používat. Různé názory se objeví až při debatě o tom, co budeme rozumět oním soustem zmíněným v nadpisu. Zde bych použil svůj oblíbený příměr: přirovnáme-li výklad ke schodišti, pak jedním soustem nebude jeden schod, ale skupina schodů mezi jednotlivými odpočívadly. Jinými slovy – soustem bude látka probraná mezi dvěma zastávkami sloužícími k jejímu důkladnému procvičení.

Jak jsem již řekl, se zásadou prakticky všichni souhlasí. Málokdo si však uvědomí, že její důsledné dodržení nedovoluje začít výklad klasických jazyků představením dat a práce s nimi. Abychom totiž mohli procvičovat práci s daty, musíme nejprve vysvětlit konstrukci programu a navíc potřebujeme používat i procedury a funkce sloužící pro výstup dat.

Dalším problémem je to, že pokud nejsou vyloženy základní programové konstrukce, těžko se nám podaří vymyslet nějaké příklady, které by studentům připadaly zajímavé. Drilové úlohy, kterými bývá počáteční výklad základních datových typů většinou provázen, nesplňují zásadu 2.5, a proto je žáci řeší většinou mechanicky bez dostatečné motivace si vše zapamatovat. (Proto také metodika Karel, s níž jsem přišel v osmdesátých letech, začínala výkladem algoritmických konstrukcí.)

S objektově orientovaným programováním je to na první pohled ještě složitější. Současné objektově orientované jazyky však naštěstí nabízejí prostředky, jak se s tímto problémem úspěšně vyrovnat. Časem o nich budu hovořit.

2.4 Doprovodné příklady musí vyžadovat aktivní použití nových poznatků

V učebnicích se setkáváme se třemi druhy příkladů:

- s drilovými příklady procvičujícími syntaxi,
- s příklady z rodu „Hele, ono to funguje!“, jejichž jediným cílem je demonstrovat vyložený rys jazyka,
- s příklady, které se snaží ukázat použití konstrukce v programu, řešícím nějaký praktický problém. Bohužel, z jejich zadání způsob použití vysvětlovaného rysu často „neodkapává“ a navíc bývají většinou „zašuměné“.

Pokud bych měl uvést příklady, tak bychom do první skupiny zařadili např. příklady vyžadující označit správně a špatně zapsané literály, správně a špatně zapsané příkazy apod. Do druhé skupiny by pak patřil známý program „*Hello World*“ nebo příklady typu: „*Napište program, který zjistí, zda zadané číslo je menší než číslo 5*“, „*Napište program, který uloží do souboru čísla do 1 do 100 a pak je z něj přečte*“ apod. Divili byste se, kolik učebnic s podobnými příklady vystačí – jejich krystalickým příkladem jsou např. Eckelovy bestsellerové učebnice *Thinking in C++* ([8]) a *Thinking in Java* ([9]). Ty však jsou určeny spíše pro programátory, kteří přecházejí na další jazyk a které opravdu zajímají spíše podrobnosti syntaxe než nějaké hlubokomyslné úvahy o užitečnosti a možnostech aplikace té které konstrukce.

Z příkladů, které bychom mohli zařadit do třetí skupiny, se v učebnicích většinou objevují „příklady ze života“ typu „*Naprogramujte výpočet určitého integrálu Simpsonovou metodou*“ (starší učebnice klasických jazyků) nebo „*Vytvořte jednoduchý systém pro správu účtů*“ (většina novějších zahraničních učebnic). Bohužel, příklady tohoto druhu velice často nevyhovují následujícím dvěma zásadám.

2.5 Příklady musí být zajímavé

Přiznejme si, že většinu začínajících studentů programování nezajímá jak se vypočítává integrál Simpsonovou metodou ani příliš netouží jednoduše spravovat nějaké účty. Moje zkušenost ukazuje, že nejlepšími příklady jsou takové, které studentu předloží zajímavý problém, u nějž student jasně ví, jak by problém řešil on, a potřebuje pouze formalizovat tak, aby jeho řešení zvládl i počítač. Z dob mých studií sem např. patřily úlohy, při nichž jsme měli řešit nějaké jednoduché šachové problémy. Patřil sem i můj první programátorský úkol, v němž dostal za úkol napsat převodník z římských čísel na arabské. Zkušenost s dětmi v zájmových útvarech (ale i s profesionálními programátory, které přeškoluji na OOP) ukazuje, že nejlepší jsou příklady, při nichž se na obrazovce něco kreslí nebo hýbe – všichni si rádi hrají s obrázky.

2.6 Řešení nesmí být příliš zašuměná

Jinými slovy: část programu, řešící zadanou úlohu a tím i procvičující probíraný rys jazyka, se nesmí utápět mezi záplavou příkazů vstupu a výstupu a různými pomocnými operacemi. Ze známých učebnic trpí touto nemocí např. bestsellerová řada *XYZ – How to Program* (např. [5], [6], [7]).

2.7 Od začátku vštěpovat žákům zásady moderního programování

Aby se zásady moderního programování vryly studentům hluboko pod kůži, je třeba je používat od samého počátku výuky. Většina autorů učebnic klasických jazyků sice studentům vysvětluje, že složitější problémy je třeba řešit tak, že se nejprve dekomponují na několik problémů jednodušších. Bohužel jim to však říká až v samém závěru svých učebnic, kdy již není žádný prostor pro důkladné procvičení. Když studenti po celou dobu kurzu řešili problémy vytvořením hlavního programu a výjimečně ještě jedné procedury či funkce, budou obdobným způsobem začínat řešit programy i po ukončení kurzu.

Chceme-li své žáky opravdu naučit řešit i složité problémy, je třeba základy tohoto přístupu vysvětlit již na samém počátku výuky, aby jej pak mohli na příkladech procvičovat v průběhu celého kurzu. Na konci kurzu pak bude tento způsob řešení problémů studentům vlastní.

Obdobné je to i s učebnicemi objektově orientovaných jazyků. Prakticky všechny začnou rozsáhlou ódou na výhody objektově orientovaného přístupu, aby vzápětí sklouzly do klasického výkladu programových konstrukcí. Když posléze přejdou k výkladu objektových rysů jazyka, demonstrují vše na triviálních, účelových příkladech, z nichž sice student pochopí princip fungování dané konstrukce, ale použití konstrukce v praktických příkladech je mu často poněkud vzdálené.

Mají-li si žáci OOP opravdu osvojit, je třeba, aby v jeho světě žili od samého začátku, od první hodiny výuky. Optimální je, když žáci začnou s třídami a objekty pracovat ještě před tím, než napíší svůj první program. Mohou si tak „osahat“ řadů vlastností takovýchto programů, uvědomit si rozdíl mezi třídou a jejími instancemi, pochopit rozdíl mezi metodami třídy a metodami instancí a ujasnit si význam atributů včetně rozdílu mezi atributy instancí a atributy třídy. Do jisté míry se tak mohou seznámit i s některými zákony dědičnosti.

Řada současných učebnic se sice snaží začít s výkladem tříd a objektů (např. [10], [12]), avšak v následném výkladu většinou zůstanou u úloh využívajících jednu třídu a „rozumně objektovému programování“ se dostanou až po několika stech stránkách.

První učebnicí, která se snaží tuto zásady respektovat, je *Object First* [1], jejíž autoři jsou zároveň autory vývojového prostředí *BlueJ*, jež naplnění těchto zásad umožňuje. I ta se však z počátku potýká s naplněním zásady 2.9.

2.8 Studenti se musí naučit programy nejen vytvářet, ale také ladit

Značná část učebnic programování vůbec nepředpokládá, že by v programech mohla být chyba a nijak své čtenáře na tuto skutečnost nepřipravuje. Jen velmi málo učebnic se snaží naučit budoucí programátory programy nejen vytvořit, ale také v nich najít a odstranit případné chyby.

Nutno přiznat, že poslední dobou se situace zlepšuje a stále častěji se začínají objevovat učebnice obsahující speciální kapitoly věnované testování a ladění programů a dokonce se pokoušejí učit i metodiku ladění (viz např. [9], [12]).

2.9 Předkládat řešení netriviálních problémů

Největší slabinou našich žáků a studentů není to, že by si nedokázaly osvojit abstraktní programovací konstrukce, ale to, že nedokáží řešit složitější úlohy. Jak jsem se již mnohokrát přesvědčil, řada učitelů podlehne pokušení naučit své žáky co nejvíce nejrůznějších konstrukcí, obratů a triků, a zapomíná přitom na to nejdůležitější: praktické programy jsou většinou mnohem složitější než ty, které s žáky řešíme. Navíc se domnívám, že právě schopnost řešení složitých úloh je tou nejcennější dovedností, kterou se děti mohou při výuce programování naučit.

Měli bychom prakticky od začátku výuky soustředit na to, abychom své žáky učili řešit složitější problémy a dekomponovat je na problémy jednodušší. Největší problém nám přitom většinou dělá naše vlastní *neschopnost vymyslet dostatečný počet dostatečně zajímavých příkladů*, které bychom mohli s dětmi řešit i při jejich malých počátečních znalostech.

Jednou z možností, která se uplatní zejména v počátečních etapách výuky, je připravit nějaký složitější projekt, který budou žáci svými programy rozšiřovat a doplňovat. OOP nám v tomto směru nabízí prostředky, které umožňují rozšiřovat stávající projekty velice jednoduchým a elegantním způsobem.

Zde opět vstupuje do hry důležitost zvoleného programovacího nástroje a prostředí. Vhodný nástroj nám totiž práci s vymýšlením úloh a jejich následnou realizací výrazně zjednoduší.

První vlaštovkou, která k nám zabloudila, by Pattisův *Karel* vyvinutý na Stadfordské universitě pro jejich vstupní kurzy programování a upravený Tomášem Bartovským pro počítače, které byly u nás tehdy dostupné. Paradoxem je, že metodiku, kterou jsem kdysi nad touto českou verzí Karlova světa připravil, řada učitelů středních škol odmítla s poukazem, že Karel pro malé děti, a učili proto raději Basic nebo Pascal, přičemž ve svém výkladu porušovali většinu z uvedených zásad.

Dalším českým příspěvkem k výuce klasického programování jsou v této oblasti systémy *Baltazar* a *Baltík* vyvinuté firmou SGP, které jsou přímými pokračovateli robota Karla vylepšenými tak, aby využily grafických možností počítačů řady PC – viz [18], [20], [21]).

Pattisův Karel má i svého objektového nástupce – Karla++ (viz např. [2]), resp. jeho „javovou“ verzi nazvanou Karel J. Robot (on-line verze publikace viz [3]).

Na webu lze najít i řadu dalších knihoven vhodných pro tvorbu příkladů pro začátečnické kurzy. Bohužel, nevím o žádné stránce, která by obsahoval jejich přehled.

3 Jak učit OOP

3.1 Kdy začít

Objektově orientované programování přináší naprosto nové paradigma. To, že nás OOP nutí nově přemýšlet ale ještě neznamená, že by bylo těžké je zvládnout. Těžké je opustit zažité stereotypy a snažit se naučit nové. Pokud ale začneme žáky učit tomuto novému přístupu včas, bude jim připadat OOP jako naprosto přirozený způsob řešení problémů a nad „předobjektovými" postupy se již budou jen usmívat.

Zkušenosti ukazují, že přeškolení zkušeného, klasicky orientovaného programátora na programátora orientovaného objektově trvá 6 až 18 měsíců (čím je zkušenější, tím déle mu změna návyků trvá). Aby děti vzali objektově orientované paradigma hned zpočátku za své, musíme s jeho výukou začít hned od první hodiny. V opačném případě je budeme učit něco, co je budeme za chvíli odnaučovat.

Kdybychom to vzali do důsledků, museli bychom přiznat, že již při učení Baltíka (viz např. [17]) vštěpujeme dětem něco, co později zapadne „na smetiště dějin". Při výuce dětí se tento problém naštěstí neprojeví, protože právě to, co je na OOP zásadně jiné, tj. postup při řešení složitých problémů, vstřebávají velmi pomalu, takže se při přechodu mezi pokročilé, objektově programující se většinou ani nemají co odnaučovat.

Samozřejmě, že by bylo lepší začít hned zpočátku objektově, ale objektově orientovaného Baltíka zatím nikdo nevymyslel (i když ten, kterého učíme, by toho moc nepotřeboval), takže ze všech dostupných vývojových nástrojů určených pro vstup do světa programování je klasický Baltík ([20]) prozatím stále (alespoň podle mne) nejlepší vstupní branou. Navíc dovednosti, které Baltík děti naučil, se jim budou při vstupu do světa OOP velice hodit. Se staršími (7. třída a výše) však můžeme začít s OOP rovnou bez nutné „baltíkovské předehry".

3.2 Jak začít

Tady narážíme na velký problém: abychom děti naučili napsat byť jednoduchý objektový program, museli bychom jim toho vysvětlit tolik, že by to většinu z nich přestalo cestou bavit. Musíme na to oklikou. I já se nyní vydám oklikou a vrátím se na chvíli opět k Baltíkovi a jeho třem základním režimům:

- v kreslícím režimu děti sestavují scény z předem připravených objektů,
- v příkazovém režimu přímo ovládají Baltíka, který ihned vykonává jejich příkazy,
- v programovacím režimu vytvářejí programy, které posléze Baltík vykoná.

Při veškeré výuce přitom využíváme toho, že děti nemusí hned od počátku vytvářet kompletní programy, ale že stačí, když vytvoří pouze drobnou nadstavbu nad předem připravený program simulující Baltíkův svět. Nadstavbu, která upraví chování tohoto programu žádaným směrem (Baltík např. popojde pár kroků a postaví domeček). I když toho děti na počátku vědí ještě velice málo, už mohou vytvářet relativně efektní (a hlavně pro ně zajímavé) programy.

Když bychom tuto myšlenku přesadili do objektově orientovaného světa, získali bychom její aplikací vývojový nástroj, který by nám umožnil vstupovat do světa OOP také ve třech krocích:

- V prvním, „kreslícím" kroku (v našem případě se jedná spíše o krok prohlížecí) si žáci prohlédnou strukturu nějakého předem připraveného programu a ujasní si vzájemné závislosti a interakce jednotlivých tříd a jejich instancí. Protože je program předem připravený, nemusí být triviálně jednoduchý (z jejich pohledu).

- V druhém, „příkazovém“ kroku si vyzkouší vytvořit instance instančních tříd a zkusí s těmito instancemi pracovat – volají jejich metody a zkoumají hodnoty jejich atributů. Hlavním cílem této etapy je ujasnění si základního vztahu mezi třídou a její instancí (objektem). Žáci si samy vyzkouší, že jedna třída může mít řadu instancí. Zjistí, že nemohou volat metody instancí, dokud žádná instance neexistuje, ale že metody třídy mohou volat i před vznikem prvé instance. Seznámí se atributy instancí a tříd a ujasní si jejich vliv na vlastnosti a následné chování jejich instancí a tříd.

  V pozdějších etapách se takto (tj. hraním si s hotovým programem) seznámí se základními projevy polymorfizmu a ověří si, že není možné vytvářet instance abstraktních tříd ani rozhraní, i když je možné jejich objekty předávat jako parametry.
- V třetím kroku začnou programovat. Protože se však v OOP neobejdeme (na rozdíl od Baltíka) bez relativně složité syntaxe, začnou nejprve upravovat existující programy a postupně se na nich naučí vše pro to, aby mohly co nejdříve vytvářet programy vlastní. OOP nám v tomto snažení vychází velmi vstříc, protože umožňuje, aby děti již na samém počátku vytvářely programy, které se přirozeně zapojí do předem připravené větší aplikace a rozšíří tak její možnosti a schopnosti. Navíc hned od počátku pracují se složitými (alespoň pro ně) programy.

Jakmile dětem proniknout základy objektově orientovaného přístupu „do krve“, můžeme se s nimi vydat do dalších oblastí a postupně před nimi odkrývat další a další oblasti světa moderního programování.

3.3 Požadavky na vývojové prostředí

Na vývojové prostředí, které bychom mohli optimálně využít při výuce OOP, bychom kromě podpory výše uvedených tří kroků měli ještě několik dalších požadavků:

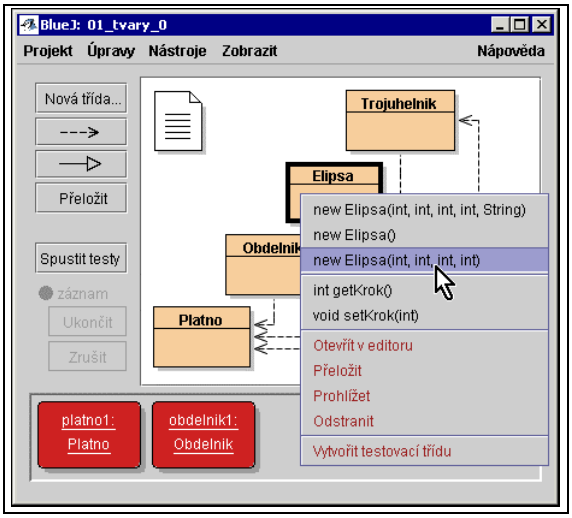
- Muselo by být výrazně komfortnější, než mnohde oblíbená dvojice Notepad + Překladač. Mělo by v sobě integrovat nástroje pro tvorbu programu, jeho překlad i následné ladění.
- Mělo by být co nejjednodušší a snadno použitelné. Většina dostupných IDE je totiž optimalizována pro profesionální vývojáře a začátečníci v nich pak zbytečně bloudí.
- Mělo by mít co nejunifikovanější rozhraní, aby se každá jeho část neovládala trochu jinak a aby se tak minimalizovala námaha potřebná k jeho osvojení.
- Mělo by maximálně usnadňovat opětovné použití částí programu vyvinutých v rámci jiných projektů (mezi hlavní zásady OOP přeci patří, že programy se nemají znovu vyvíjet, ale znovu použít).
- Mělo by být maximálně uzpůsobené výuce; mělo by podporovat
  - vizualizaci struktury programu i dění v něm (tím nemyslím zobrazit to, že např. Baltík chodí, ale zobrazit to, co se děje uvnitř programu),
  - interakci uživatele s programem (např. možnost zjištění hodnot atributů a lokálních proměnných za chodu programu, vytvoření další instance apod.),
  - experimentování.
- Mělo by umožňovat práci v týmu, kdy každý člen skupiny pracuje na své části programu, aby pak na společných schůzkách ladili jejich vzájemné interakce.
- Mělo by být dostupné i studentům, tj. nemělo by být drahé (nejlépe zdarma) a nemělo by mít velké hardwarové nároky.
- Pro výuku mladších žáků by mělo být pokud možno lokalizované.

3.4 Prostředí BlueJ

Podobné myšlenky napadly i Michaela Köllinga a jeho spolupracovníky v australské Monash University v Melbourne a v Mærsk Institute dánské University of Southern Denmark v Odense. Výsledkem jejich několikaletého snažení je vývojové prostředí BlueJ, které je nyní (zdarma) dostupné ve verzi 1.3.0. na adrese [4].

Prostředí BlueJ se snaží naplnit všechny výše uvedené požadavky. Ne vždy se mu to sice daří optimálně, ale je prozatím ze všech mně známých prostředí optimu nejblíže. Využívá se proto již na více než 300 universitách a školících centrech po celém světě. Využívat jsem je začal i ve svých kroužcích programování, jejichž nejmladší loňský účastník chodil do 3. třídy. Musím přiznat, že pro vstup do světa OOP se toto prostředí osvědčilo skvěle.

Mezi jeho důležité vlastnosti usnadňující nasazení ve školách a zájmových kroužcích patří mimo jiné jeho relativní hardwarová nenáročnost (pod Windows 98 si vystačí se 64 MB paměti), možnost stažení zdarma a také to, že je lokalizováno do češtiny, takže s ním bez problémů pracují i ti, kteří s angličtinou teprve začínají. Jedinou jeho nevýhodou je, že si prozatím nerozumí s češtinou v identifikátorech a komentářích (program se přeloží a rozběhne, ale při použití diakritiky začnou zlobit některé jeho užitečné pomocné funkce).



4 Postup výuky

Po tomto sáhodlouhém úvodu se konečně dostáváme k vlastnímu tématu: návrhu metodiky výuky OOP pro žáky základních a středních škol, kterou jsem vyzkoušel ve svých kroužcích programování a kterou se nyní snažím vtělit do připravované učebnice, na jejíž první kapitoly se můžete podívat na adrese [19].

Ti z vás, kteří znají metodiku robota Karla (a následně čaroděje Baltíka) vědí, že podle této metodiky učíme děti nejprve základy algoritmizace (jazyk robota Karla ani začátečnický režim Baltíka neznají proměnné) a až poté děti seznámíme se syntaxí nějakého vyššího jazyka (v Baltíkovi pouze přejdeme do pokročilého režimu) a pomalu začneme přidávat práci s daty. Chceme-li při výuce klasického programování předávat informace po malých soustech, ani to jinak nejde.

Objektově orientované jazyky a dobře navržené vývojové prostředí nám nyní umožňují tuto posloupnost otočit. Můžeme začít prací s daty a po nějaké době postupně přidávat informace o algoritmických konstrukcích. Projděme si nyní takový postup.

4.1 Seznámení s prostředím, třídami a objekty

Na počátku si s dětmi vysvětlíme, co to jsou třídy a objekty a děti si v zápětí v interaktivním režimu osahají práci s nimi. Spustí si malý projekt (viz obrázek), který umožňuje umisťovat na plátno jednoduché grafické objekty a posouvat je. Naučí se

konstruovat instance tříd, volat jejich metody a během první hodiny jasně pochopí, jaký je rozdíl mezi třídou a instancí, co to jsou metody a jak reagují na zadané parametry. Naučí se, že s instancemi musí komunikovat pomocí metod. V této počáteční fázi je vše postaveno na interaktivním režimu prostředí BlueJ, které umožňuje veškerou vykládanou látku názorně předvést.

4.2 Metody a atributy instancí a tříd,

V následující hodině (hodina v kroužku má 90 minut) si ukážeme, že i třídy mají svoje metody a děti si ujasní, pro které úkoly se používají metody instancí a pro které metody třídy. Zároveň si na příkladu ukazujeme, že nikdy nemají v ruce přímo instanci, ale vždy pouze odkaz na ni (programujeme v Javě – v jiných jazycích to může být jinak).

Poté se děti naučí vyvolat prohlížeč objektů, vysvětlíme si, co to jsou atributy a ukážeme si, jak se jejich hodnoty mění s měnícím se stavem objektu. Povíme si o přístupových metodách, pomocí nichž je možno stav atributu zjistit nebo jej naopak nastavit. Zároveň se děti doví, že vedle atributů instancí existují i atributy tříd.

4.3 Zdrojový program, definice vlastní třídy, první volání metod

Třetí hodinu nahlédneme programu pod pokličku a ukážeme si jak takový program vypadá uvnitř. Děti požádají prostředí, aby pro ně vytvořilo prázdnou třídu (opravdu prázdnou) a následně i její instanci. Poté si nechají zobrazit její zdrojový kód.

Vysvětlíme si, co je to konstruktor a ukážeme si, jak se vytvářejí instance jiných tříd. Děti definují svoji první metodu. Při té příležitosti dostanou svůj první domácí úkol: vytvořit třídu, jejíž instance nakreslí na plátně předem zadaný obrázek (její konstruktor pouze zavolá konstruktory dříve definovaných tříd v projektu).

4.4 Přiřazovací příkaz, metody s parametry, komentáře

Další hodinu si děti své výtvory porovnávají, spojují je dohromady a vytvoří třídu, jejíž instance jsou sestaveny z instancí tříd, které vytvořily za domácí úkol. Aby se jejich výtvory nepřekrývaly, definují ve svých třídách konstruktory s parametry, umožňujícími umístit jejich obrázky na libovolné místo plátna.

Při práci s cizími programy si vysvětlíme důležitost komentářů a seznámíme se s pravidly, kterými je třeba se při psaní komentářů řídit. Za domácí úkol dostanou doplnit své třídy o dokumentační i obyčejné komentáře, které se od této chvíle stanou povinnou součástí jejich programů.

4.5 Definice atributů

Ukážu dětem, že současné definice tříd neumožňují vytvořený obrázek někam přesunout. Děti doplní své třídy o první atributy a zároveň definují několik metod, které přesun dané instance zprostředkují.

4.6 Rozhraní

Seznámit děti s konceptem rozhraní je trochu složitější, tak se o tomto problému rozhovořím podrobněji. Na počátku dětem vysvětlím, že se jejich překrývající se obrázky doposud navzájem umazávaly, a že proto přejdeme k jinému prostředí, které ohlídá, aby se tak nedělo. Pak změním sadu výchozích tříd, s nimiž děti pracují. Plátno od této chvíle nebude pasivním objektem, na který se instance třídy sama nakreslí, ale stane se manažerem, který se sám stará o to, aby se při přesunu jednoho objektu přes

druhý spodní objekt neodmazával. Instance se však již nemůže kresli sama jako doposud, ale chce-li být zobrazena na plátně, musí se u tohoto manažera přihlásit. Manažer ji přijme mezi spravované pouze tehdy, bude-li se umět na požádání nakreslit dodaným „kreslítkem“ (objekt třídy *Graphics*).

Po tomto úvodu si vysvětlíme význam a účel rozhraní a ukážeme si, jak třídu k implementaci daného rozhraní přihlásit. Pak děti přihlásí své třídy k implementaci rozhraní *IKresleny* a vyzkouší, že plátno opravdu pracuje tak, jak slibovalo.

Poté spolu definujeme rozhraní *IPosuvny* a třídu *Drc*, jejíž instance umějí plynule posunout instance typu *IPosuvny* p úevně danou vzdálenost. Děti mají za úkol samy upravit definice svých tříd tak, aby mohly být po plátně posouvány. Všechno jsou to drobné operace, takže je můžeme bez větších problémů za dvouhodinovku stihnout.

Pak jim ukážu třídu *Presouvac* která umí instance typu *IPosuvny* přesouvat na libovolnou vzdálenost (i mimo plátno a zpět) a děti si její funkci vyzkouší.

Za domácí úkol mají definovat rozhraní *INafukovaci* a třídu *Psouk* (česky pšouk – takovýto název třídy je vyprovokuje k tomu, že domácí úkol určitě udělají), jejíž instance jemně zvětší, resp. zmenší zadaný objekt. Zároveň mají potřebně upravit i své třídy s obrázky. Další hodinu pak vyzkouší funkci svých programů ve spolupráci s třídou *Kompresor*, jejíž instance umí nafukovací objekty libovolně „nafouknout“ nebo naopak „vypustit“.

4.7 Dědění rozhraní

Při procvičování rozhraní si ukážeme, že to, že třída může implementuje nafukovací nebo posunovací rozhraní ještě neznamená, že bude nakreslitelná na plátno. Vysvětlíme si princip dědění rozhraní a ukážeme si, jak nám tento princip umožní obejít nemožnost zadání dvou typů téhož parametru.

4.8 Balíčky

Doposud musely děti zakomponovat do názvů svých tříd i svoje jméno. Tím bylo zaručeno, že názvy tříd nebudou kolidovat. Po seznámení s balíčky si každé dítě definuje svůj balíček, ve kterém bude svobodně definovat své třídy nezávisle na ostatních.
Přiznejme si, že se prostředí BlueJ s balíčky moc nekamarádí, takže řadu věcí, které bychom rádi řešili interaktivně, musíme naprogramovat. Nicméně i tak je zavedení balíčků vítaným krokem k osvobození dětí od nucené definice dlouhých jmen.

4.9 Knihovny a knihovní třídy

Zavedení balíčků umožní mimo jiné začít používat třídy ze standardní knihovny z jiných balíčku než *java.lang* a také třídy z vlastních knihoven. Děti se seznámí s knihovnou *Robot* (volné pokračování robota Karla).

4.10 Vzor Strategie

Zavedením balíčků umožníme dětem definovat složitější sady objektů aniž by pak byl výsledný program nepřehledný. Vrátíme se proto ještě jednou k dědění rozhraní a ukážeme si, jak je možné vhodnou definicí hierarchie tříd a rozhraní obejít podmíněný příkaz (návrhový vzor *Strategie*). Definujeme třídu, jejíž objekty se umějí posouvat vpřed a otáčet se o 90°. Ukážeme si, že instance této třídy mohou implementovat rozhraní *IRobot* z knihovny *Robot*, a že proto mohou zastoupit roboty v příkladech, které jsem vytvářeli v minulé hodině.

4.11 Dědičnost tříd, abstraktní třídy

Děti se seznámí s podtřídou jako speciálním případem rodičovské třídy. Na příkladu robotů si ukážeme, jak vhodně definovaný systém tříd a podtříd může výrazně ušetřit práci a zmenšit počet metod, které je nutné napsat. Upozorním je na to, že dceřinné třídy by měly používat opravdu pouze k definici speciálních případů rodičovské třídy a neopakovat začátečnické chyby některých autorů učebnic, kteří definují geometrické tvary jako dceřinné třídy bodu (viz např. [11], [14]).

V další hodině si pak na příkladu grafických objektů, s nimiž jsme pracovali na začátku kurzu, ukážeme situace, kdy by se nám hodilo definovat společnou rodičovskou třídu skupiny tříd, avšak nejsme schopni definovat pro tuto třídu některé povinné metody. Zavedeme abstraktní třídy a ukážeme si, jak lze s jejich pomocí tento problém řešit.

Vysvětlíme si, že jednou z důležitých zásad současného je pokud možno nekopírovat do jiných míst jednou napsaný kód a ukážeme si, jak je s použitím doposud vysvětlených konstrukcí dosáhnout toho, abychom mohli napsat kód jen jednou a zařídit, aby všichni, kteří kód potřebují, používali tuto jedinou „instanci“ kódu.

4.12 Podmíněný příkaz, vyvolání výjimky

Již je nejvyšší čas, abychom začali také s výkladem klasických algoritmických konstrukcí. Seznámíme se s podmíněným příkazem a vedle dalších použití si také ukážeme, jak lze chybové situace řešit vyhozením výjimky. Výjimky prozatím pouze vyhazujeme, avšak neošetřujeme.

4.13 Cykly

Na příkladu robotů si ukážeme typické problémy řešené pomocí cyklů. Seznamuji přitom dětem jak s cykly s počáteční a koncovou podmínkou, tyk s cykly s podmínkou uprostřed, které umožňují relativně jednoduché řešení situací, kdy je třeba provést nějaké operace před testem počáteční podmínky, resp. po testu ukončovací podmínky.

4.14 Kontejnery, pole, metody s proměnným počtem parametrů

Výklad kontejnerů začínám výkladem dynamických kontejnerů, s nimiž se v řadě příkladů pracuje snadněji než s klasickými statickými poli. Definujeme třídu *Supertvar*, jejíž instance mohou být složeny z předem neurčeného počtu tvarů.

Při následném výkladu klasických polí si ukážeme, jak je možné využít pole při definici metody s předem neznámým počtem parametrů.

4.15 Ošetření výjimek

Základní běh zakončujeme výkladem ošetření výjimečných situací.

5 Závěr

Uvedená metodika vychází ze zkušeností z vedení kroužků programování na ZJŠ. Kroužky navštěvovalo 8 až 10 dětí ve věku od 10 do 13 let. Většina dětí však byla 12letých. Byl jsem překvapen, jak snadno děti chápou objektové paradigma a jak lehce vše vstřebávají. Jejich náhled na řešení problémů byl mnohem přirozenější, než u programátorů, které přeškoluji z klasického programování na programování objektově orientované.

Reference

1. Barnes D. J., Kolling M. *Objects First with Java: A Practical Introduction Using BlueJ*. Prentice Hall, 2002, ISBN: 0-13-044929-6.
2. Bergin J., Stehlik M., Roberts J., Pattis R. *Karel++, A Gentle Introduction to the Art of Object-Oriented Programming*, John Wiley & Sons, 1977, ISBN 0-471-13809-6
3. Bergin J., Stehlik M., Roberts J., Pattis R. *Karel J. Robot, A Gentle Introduction to the Art of Object-Oriented Programming*
   http://csis.pace.edu/~bergin/KarelJava2ed/Karel++JavaEdition.html
4. BlueJ – The Interactive Java Environment. *http://www.bluej.org/*
5. Deitel H. M., Deitel P. J. Listfield J. A., Nieto T. R., Yager C. H., Zlatkina M. *C# How to Program*. Prentice Hall, 2001, ISBN: 0-13-062221-4.
6. Deitel H. M., Deitel P. J. *C++ How to Program (4th Edition)*. Prentice Hall, 2002, ISBN: 0-13-038474-7.
7. Deitel H. M., Deitel P. J. *Java How to Program, Fifth Edition*. Prentice Hall, 2002, ISBN: 0-13-101621-0.
8. Eckel B. *Thinking in C++, Volume 1: Introduction to Standard C++ (2nd Edition)*. Prentice Hall PTR, 2000, ISBN 0-13-979809-9. Ke stažení na adrese *http://mindview.net/Books/TICPP/ThinkingInCPP2e.html*. Český překlad: *Myslíme v jazyku C++*. Grada, 2000, ISBN 80-247-9009-1.
9. Eckel B. *Thinking in Java (3rd Edition)*. Prentice Hall PTR, 2002, ISBN: 0-13-100287-2. Ke stažení na adrese *http://www.mindview.net/Books/TIJ/*. Český překlad 2. vydání: *Myslíme v jazyku Java*. Grada, 2000, ISBN 80-247-9010-6, 80-247-0027-1.
10. Fu C. T. *An Introduction to Object Oriented Programming with Java*, McGraw-Hill, 2001, ISBN 0-07-239684-9.
11. Hála T. *Pascal. Učebnice pro střední školy.* Computer Press, 2002. ISBN 80-7226-733-7.
12. Horstmann C. S. *Big Java*. John Wiley & Sons, Inc., 2002, ISBN 0-471-40248-6.
13. Horstmann C. S., Cornell G. *Core Java, Volume I – Fundamentals*. Sun Microsystem Press, 2003, ISBN 0-13-047177-1.
14. Kačmář D. *Úvod do jazyka C#, díl IV*. Softwarové noviny, 6/2003.
15. Kernighan B.W., Ritchie D. M. *The C Programming Language*. Prentice Hall PTR, 1988, ISBN: 0-13-110362-8. Český překlad prvního vydání: *Programovací jazyk C*. SNTL, 198?, ISBN 80-?.
16. Merunka V. Výuka objektově orientovaného programování. Sborník konference Objekty '97.
17. Pecinovsky R. *Baltík – Učebnice programování nejen pro děti*. SGP Systems, 2001.
18. Pecinovský R. *Programujeme s Baltazarem.* Seriál, ABC 1998/1999, ke stažení na *http://vyuka.pecinovsky.cz/Baltazar.pdf*.
19. Pecinovský R. ??? (Název jsem ještě nevymyslel). Připravovaná učebnice programování s prvními kapitolami na *http:vyuka.pecinovsky.cz/Obsah.htm*
20. SGP: Multimediální tvůrčí systém SGP Baltík 3 (stránka s popisem vlastností a možností programu Baltík) *http://www.sgp.cz/cz/produkt-b3/Baltik3-popis.htm*
21. SGP: Soubory ke stažení zdarma (mimo jiné je zde ke stažení demo programů Baltazar a Baltík). *http://www.sgp.cz/new/index1.asp?PAGE=detail&ID=DOWN-01*