

# PROČ A JAK UČIT OOP ŽÁKY ZÁKLADNÍCH A STŘEDNÍCH ŠKOL

Rudolf Pecinovský

**Klíčové slova:** programování, výuka, OOP, objektově orientované programování

## 1 Úvod

### 1.1 Trocha historie

Výukou programování žáků základních a středních škol se zabývám od roku 1979, kdy jsem v průběhu vojenské presenční služby působil jako učitel programování. Za tu dobu se dramaticky změnila nejenom používaná výpočetní technika, ale neméně dramaticky se změnila i technologie programování a s ní i názory na to, jak programování co nejlépe vyučovat.

V osmdesátých letech minulého století, v době vrcholícího Basicu, jsme usilovně bojovali za to, aby se ve výuce prosadilo strukturované programování. Již tehdy se ale ve světě začalo pomaličku prosazovat programování objektově orientované (OOP). V devadesátých letech se OOP prosadilo jako klíčová programátorská technologie, takže jsme své žáky začali učit jak programovat objektově. Učili jsme je různé objektově orientované konstrukce a vysvětlovali mechanismy jejich funkce. Vše jsme však pouze dodatečně roubovali na jejich předchozí „strukturovanou zkušenost“. Takovýto styl výuky ale není výukou objektově orientovaného programátorského myšlení. (Přiznejme si, že nám k tomu tehdy používané jazyky pomocnou ruku příliš nepodávaly.) Žáci proto i nadále přemýšleli „strukturovaně“ a po objektových vlastnostech jazyků sáhli pouze v případě, kdy vhodnost tohoto přístupu z řešení úlohy nepřehlédnutelně odkapávala. Ostatně i v současné době většina učebnic neučí objektově programování, ale pouze syntaxi objektově orientovaných jazyků.

Zkušenost ukázala, že přeškolení klasického „strukturovaně orientovaného“ programátora na programátora orientovaného objektově trvá 6 až 18 měsíců. Čím je programátor zkušenější, tím delší dobu se musí přeškolovat. Přitom současné požadavky na vysokou efektivitu vývoje a na komplexnost (troufám si říci oblidnost), robustnost a spravovatelnost výsledných programů již není možno bez objektově orientovaného přístupu splnit. V současné době již proto před námi nestojí otázka **zda** učit objektově orientované programování, ale **jak** je učit.

Nezbylo mi proto, než koncepci výuky znovu přebudovat tak, abych žáky již od první hodiny vychovával k objektovému myšlení a abych následně vykládané nadstavbové znalosti a dovednosti zasazoval do dříve připraveného „objektového podhoubí“.

### 1.2 Co vlastně dělám

Na rozdíl od řady z vás mne neživí výuka žáků základních a středních škol, ale výuka a doškolování profesionálních programátorů. 80 % mých studentů jsou přitom programátoři, kteří již nějaký ten rok programují v klasicky orientovaných programovacích jazycích a zjistili, že bez znalosti objektově orientovaného programování již nedosáhnou na platy, na které by rádi dosáhli. Přihlašují se k nám proto na kurzy, ve kterých je přeškolují z klasického programování na programování objektově orientované.

Vedle toho (mohli bychom říci jako sponzorský dar) učím děti v kroužcích na základních a středních školách a ve Stanici techniků Vyšehrad. Obě činnosti se prolínají: s dětmi mohu experimentovat a to, co se v kroužcích osvědčí, převezmu do kurzů pro profesionální programátory.

### 1.3 O čem si budeme povídat

V tomto příspěvku se pokusím ukázat, jak při svých hodinách postupuji, a podělit se s vámi o některé zkušenosti z této výuky. Zároveň bych vás chtěl přesvědčit o tom, že objektově orientované programování není nic tak komplikovaného, aby je nebylo možno učit na základních a středních školách. Naopak si trůfám tvrdit, že při tomto přístupu se žáci naučí řešit složité úlohy daleko dříve, než se nám to dařilo v dobách, kdy jsme je učili programovat „jenom strukturovaně“.

## 2 Kdy a jak začít

### 2.1 Kdy začít

Jakmile začnu někoho přesvědčovat, že bychom měli i na základních a středních školách učit žáky hned od začátku programovat objektově, začne mi okamžitě oponovat, že objektové paradigma je na jeho žáky příliš těžké. Opak je však pravdou: svět objektů, které si navzájem posílají zprávy, je pro žáky daleko pochopitelnější než svět posloupností příkazů. Žáci vychovávaní v klasickém světě příkazů jsou sice schopni vytvořit jednoduché programky pro Karla nebo Baltíka, avšak postavíme-li před ně složitější úlohu, většinou jsou před ní bezradní, nebo se do ní sice pustí, ale během několika prvních kroků zabloudí.

Obávám se, že daleko větší problém s přechodem z klasického na objektové paradigma budou mít vyučující. Co si budeme namlouvat, už nám to nezapaluje tak rychle jako dětem. To, co ony rychle pochopí, s tím se my musíme nějakou dobu vypořádávat. Neschovávejme se proto při posuzování vhodnosti či nevhodnosti tohoto typu výuky za děti, těm objektové paradigma problémy nedělá.

Kdy tedy začít? Nejmladší dítě v mých kroužcích navštěvovalo 3. třídu základní školy. (Přiznávám, že takto ranný věk je opravdu časný, ale dotyčný se už nad Baltíkem nudil.) Většina však byla ze 4. a 5. třídy. Za standardní věk, kdy lze se současnými nástroji rozumně začít s výukou v kroužku, bych považoval přibližně pátou až šestou třídu.

Moje zkušenost dokonce říká, že starší věk je pro dobrý začátek spíš horší. Děti, které do kroužku nastupují jako starší (8. třída a víc), již většinou mají nějaké zkušenosti s klasickým programováním a pořád se snaží vysvětlovanou látku nějak navléknout na to, co znají. Někakou dobu proto trvá, než se těchto stereotypů zbaví, takže mají na počátku horší výsledky než ti mladší, klasickým programováním doposud nezkažení.

### 2.2 Zásady, které bychom měli dodržovat

Dovolte, abych před vlastním popisem navrhované metodiky zopakoval zásady, které bychom měli při výuce programování dodržovat. Kdyby někoho zajímal jejich podrobnější rozbor, najde jej např. v [1] nebo [2].

#### 1. Co nejdříve umožnit tvorbu programů

Programováním přitom nemyslím jenom klasické psaní kódu, ale i přímé zadávání instrukcí nějakému ovládanému subjektu.

#### 2. Nepředbíhat, tj. nepoužívat prvky jazyka, které ještě nebyly vyloženy

Nejznámějším porušením této zásady je známý program *Hello World*.

#### 3. Informace je třeba předávat po malých soustech

Jinými slovy: výklad hojně prokládat příklady, při jejichž řešení mají žáci možnost látku zažít.

#### 4. Studenti se musí naučit programy nejen vytvářet, ale také ladit

Budeme-li jenom obcházet studenty a opravovat za ně jejich chyby, tak je pro samostatné programování moc dobře nepřipravíme. Studenti se musí naučit své chyby nacházet a opravovat sami.

#### 5. Doprovodné příklady musí vyžadovat aktivní použití nových poznatků

Jinými slovy: příklady by pokud možno neměly být drilové, ale raději takové, při nichž musí žáci trochu „zapnout hlavu“.

#### 6. Příklady musí být zajímavé

Je-li příklad zajímavý, chtějí jej žáci řešit sami od sebe bez nucení.

### 7. Řešení nesmí být příliš zašuměná

Řešení problému, který demonstruje použití vysvětlované konstrukce, by nemělo zabírat pouze malou část celkového řešení. Chceme-li s žáky řešit rozsáhlejší problém kvůli nějakému malému, ale zajímavému jádru, je výhodné jim „omáčku“ předem naprogramovat, aby se pak mohli soustředit na to, co je chceme naučit.

### 8. Od začátku vštěpovat žákům zásady moderního programování

Chceme-li, aby se žákům vryly zásady moderního programování pod kůži, musíme je vysvětlovat od samého začátku, a ne napřed programovat trochu jinak a pak je přeučovat.

### 9. Předkládat řešení netriviálních problémů

Největší slabinou řadového absolventa není neznalost programových konstrukcí, ale neschopnost řešit složité úlohy. V životě se přitom s jinými úlohami téměř nesetká. Musíme je proto naučit řešit složité úlohy již v rámci výuky.

## 2.3 Jak začít

Tady narážíme na velký problém: abychom žáky 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, při níž si děti s objekty nejprve „hrají“ a když pochopí jejich základní vlastnosti a seznámí se s nejdůležitějšími termíny, tak si zkusí nějaký ten vlastní objekt naprogramovat.

Metodika vychází ze stejných počátečních premis, z jakých jsem vycházel při tvorbě metodiky výuky za pomoci Karla a později Baltíka. Platí známé pravidlo: čím nezkušenější je uživatel, tím chytřejší musí být počítač.

Ani objektové programování proto nezačínám učit na zelené louce (ač je to u učebnic programování stále nepsaný standard), ale připravím dětem nějaké prostředí, ve kterém se nejprve pohybují a pro něž později své programy tvoří. Při veškeré výuce využíváme toho, že žáci nemusí hned od počátku vytvářet kompletní programy, ale že stačí, když vytvoří pouze drobný doplněk do nějakého již existujícího světa. Doplněk, který viditelně rozšíří možnosti tohoto světa a zvýší tak jeho „kvalitu“. I když toho žáci na počátku vědí ještě velice málo, už mohou vytvářet relativně efektní (a hlavně pro ně zajímavé) programy.

Vlastní vstup do světa objektově orientovaného programování proto probíhá ve třech na sebe navazujících krocích:

1. V prvním, prohlížecím kroku se žáci seznámí s pojmy třída a objekt a prohlédnou si strukturu nějakého předem připraveného programu. Ujasní si vzájemné závislosti a interakce jednotlivých tříd a jejich instancí. Protože je program předem připravený, nemusí být triviálně jednoduchý (alespoň z jejich pohledu).
2. V druhém, příkazovém kroku si vyzkouší vytvořit instance instančních tříd a zkusí s těmito instancemi pracovat. Volají jejich metody, analyzují jejich chování a zkoumají hodnoty jejich atributů. Hlavním cílem této etapy je ujasnění si základního vztahu mezi třídou a její instancí (objektem). Žáci si sami vyzkouší, že jedna třída může mít (většinou) řadu instancí. Zjistí, že nemohou volat metody instancí, dokud žádná instance neexistuje, ale že metody třídy mohou volat i před vznikem první instance. Seznámí se s atributy instancí a tříd a ujasní si jejich vliv na vlastnosti a následné chování těchto instancí a tříd.

V pozdějších etapách se takto (tj. hraním si s hotovým programem) seznámí se základními projevy polymorfismu a ověří si, že není možné vytvářet instance abstraktních tříd ani rozhraní, i když je možné jejich objekty předávat jako parametry.

3. V třetím kroku začnou programovat. Protože se však v Javě 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 proniknou základy objektově orientovaného přístupu „do krve“, můžeme se s nimi vydat do dalších oblastí a postupně před nimi odkrývat další a další oblasti objektově orientovaného programování.

## 3 Použitý jazyk a vývojové prostředí

Jakmile začneme vážně uvažovat o výuce objektově orientovaného programování, musíme začít uvažovat o použitém programovacím jazyku a vývojovém prostředí. Tyto dvě otázky jdou ruku v ruce a vzájemně se ovlivňují. Zkusím nejprve naznačit, na co bychom se měli při jejich výběru zaměřit a pak vám prozradím, jaký jsem si z toho udělal pro sebe závěr.

### 3.1 Požadavky na programovací jazyk

Použitý programovací jazyk by měl mít následující vlastnosti:

- Měl by být především doopravdy objektově orientovaný. Tím automaticky vypadává starý Visual Basic (Visual Basic .NET již objektově orientovaný je).
- Měl by převzít na svá bedra co největší množství úkolů, které by programátor nemusel bezpodmínečně řešit – např. správu paměti. Tato podmínka vyřazuje Delphi až do verze 7 včetně a C++ (C++ pro .NET již správu paměti zprostředkovává).
- Měl by být maximálně jednoduchý, abychom se při výuce mohli soustředit na programování a nemuseli trávit dlouhé chvíle vysvětlováním vlastností a hlavně záludností jazyka (tady definitivně vypadává C++).
- Měl by být maximálně bezpečný a odhalit co nejvíce chyb již ve fázi překladu.
- Měl by mít minimální nároky na hardware počítače (hardwarové nároky jsou ale silně ovlivněny použitým vývojovým prostředím).
- Měl by být dostupný na všech používaných platformách, tj. v současné době především Windows, Linux a nejspíš také Macintosh.
- Překladače, vývojová prostředí a nástroje by měly být dostupné za minimální cenu, nejlépe zdarma.
- Měl by být dostatečně rozšířený, aby bylo možno vyměňovat zkušenosti v rámci co nejširší komunity.
- Měla by k němu být k dispozici bohatá nabídka knihoven, nástrojů a volně stáhnutelných programů.
- Neměl by to být žádný speciální „exotický“ jazyk, ale jazyk, s nímž se mohu žáci běžně potkat v praxi a nebo by měl být takovému jazyku alespoň velmi podobný (tato podmínka vyřazuje Logo).

Neznám jazyk, který by vyhovoval ve všech parametrech. Jako horcí kandidáti, kteří se u mne probojovali do finále, se ukázaly (abecedně): C#, Delphi pro .NET, Java, Python (možná), Smalltalk a Visual Basic .NET. Neříkám, že by do seznamu neměly patřit ještě další, ale prozatím o nich nevím.

### 3.2 Požadavky na vývojové prostředí

Vývojové prostředí, které bychom mohli optimálně využít při výuce OOP, by mělo umožnit co nejlepší realizaci výše uvedených tří vstupních kroků. Kromě toho bychom na něj kladli ještě několik dalších požadavků:

- Musí mít maximálně jednoduché ovládání, aby žáci nebadali nad vlastnostmi prostředí a mohli se soustředit na programování.
- Na druhou stranu musí být dostatečně komfortní, protože kombinace Notepad+Překladač, kterou používají mnohé učebnice programování, zdržuje a rozptyluje ještě více, než složitá profesionální prostředí.

U této otázky bych se na chvíli zastavil. Často od zkušených programátorů slyším, že by začátečníci měli začít právě s výše popsanou kombinací, protože jenom tak pochopí některé vnitřní zákonitosti programu. Domnívám se, že začátečníci mají plnou hlavu jiných věcí k pochopení a že „některé vnitřní zákonitosti“ můžeme klidně nechat na pozdější dobu. Případá mi to podobné, jako kdybychom po cestovateli chtěli, aby z Prahy do Vladivostoku neletěl letadlem, ale dal přednost vlaku, protože jedině tak si doopravdy uvědomí, jaká je to dálka.

- Mělo by maximálně usnadňovat použití částí vyvinutých v rámci jiných projektů a tím i spolupráci žákovských týmů při práci na větších projektech.

- Mělo by být maximálně uzpůsobené výuce, tj. mělo by podporovat:
  - vizualizaci struktury programu a dění v něm (tj. umět zobrazit, co se děje uvnitř programu),
  - interakci uživatele s programem (přímé vytváření instancí a volání jejich metod),
  - automatickou tvorbu testů,
  - experimentování.
- 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.

### 3.3 Tak tedy v čem?

Při shrnutí požadavků obou předchozích bloků mi pro moji potřebu vyšla jako jasný vítěz Java ([3]) a vývojové prostředí *BlueJ* ([4]). S výjimkou hardwarových nároků splňuje Java všechny požadavky kladené na programovací jazyk: je multiplatformní, plně objektová, relativně jednoduchá a velmi bezpečná. Kromě toho je vstupním jazykem do světa programování na drtivé většině světových univerzit a navíc je v současné době podle statistik nejrozšířenějším programovacím jazykem.

Při rozhodování měla velkou váhu skutečnost, že veškerý software, který k výuce potřebuji, získám zdarma, a to ne v nějaké příležitostné namlsávací akci, ale vždy. Uvážím-li k tomu, že počítač splňující hardwarové nároky Javy pořídím za 2.500 Kč, nemá z pohledu ceny konkurenci.

Silným argumentem bylo i vývojové prostředí *BlueJ*, které je vyvinuté se speciálním zřetelem na výuku OOP a jako jediné nabízí několik velice užitečných vlastností, které vyučující i žáci velice ocení. Prostředí *BlueJ* se snaží naplnit všechny výše uvedené požadavky. Ne vždy se mu to sice daří optimálně, ale ze všech mně známých prostředí je prozatím optimu nejbližší. Využívá se proto již na téměř 400 universitách a školících centrech po celém světě. Využívat jsem je začal i ve svých kroužcích programování. 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.

Posledním důvodem pro volbu Javy bylo i to, že prakticky všechny děti, které chodí do mých kroužků, mají mobilní telefony programovatelné v Javě, takže si v pokročilých kroužcích mohou připravit pro své telefony aplikace, kterými se mohou chlubit před svými spolužáky a rodiči, což jen zvyšuje jejich motivaci.

To jsou moje důvody. Budete-li chtít minimalizovat investice, asi se k nim přidáte. Budete-li mít sponzora, který vás bude tlačit jinam, asi se necháte přemluvit sponzorem. Mějte ale na paměti, že volba jazyka není vše. Vývojové prostředí optimalizované pro výuku efektivitu výukového procesu výrazně zvýší, kdežto nevhodné vývojové prostředí naopak vše výrazně zkomplikuje.

## 4 Doporučený postup výuky

Tak jsme se vítězně probodovali k těžišti celého příspěvku, kterým je metodika výuky a podrobný popis jejích jednotlivých kroků. Dále uvedená metodika je základem učebnice, která má pracovní název *Myslíme objektivě v jazyku Java 1.5* a měla by se objevit na pultech na podzim letošního roku.

Ti z vás, kteří znají moji metodiku robota Karla (a následně čaroděje Baltíka) vědí, že podle ní 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 trochu zamíchat. Můžeme začít prací s daty a po nějaké době postupně přidávat informace o algoritmických konstrukcích. Tento postup jsem zvolil i v následně popsané metodice.

Popisovaná metodika ukazuje postup, který používám při výuce dětí v kroužcích. Naprosto stejný postup však používám i při výuce dospělých ve svých kurzech. Jediný rozdíl je v tom, že tito dospělí absolvují týdenní intenzivní kurz, kdežto děti se učí programovat 90 minut týdně. Dále uvedenou látku přitom s dospělými probíráme v prvních dvou až třech dnech (záleží na zkušenostech a bystrosti posluchačů).

Zdůrazňuji to proto, že jsem se již několikrát setkal s tím, že středoškolští učitelé odmítli pracovat podle mojí metodiky Karla a Baltíka s poukazem na to, že je to metodika určená pro malé děti. Není tomu tak. Metodiku jsem vyvíjel pro dospělé se záměrem, aby podle ní bylo možno učit i malé děti a s dětmi také všechny postupy nejprve zkouším. Prověřené postupy pak používám v kurzech pro profesionální programátory.

#### 4.1 Seznámení s prostředím, třídami, objekty a zprávami

Na počátku dětem povím, že moderní programování je založeno na třídách a objektech a jejich vzájemné komunikaci. Vysvětlím jim že program již dávno není pouhá posloupnost příkazů realizujících nějakou funkci, ale že to je v nějakém programovacím jazyku zapsaný popis tříd, jejich instancí a zpráv, které si instance mezi sebou posílají.

Děti si pak na počítačích spustí malý projekt, který umožňuje umisťovat na plátno jednoduché grafické objekty a manipulovat s nimi. Tento projekt jim hned na počátku ukáže, že objektem je v současných programech nejenom to, co běžný člověk jako objekt chápe (v příkladě našeho grafického projektu je to plátno a geometrické tvary), ale i takové na první pohled abstraktní vlastnosti, jako např. směr nebo barva.

Dětem, které s programováním začínají, takováto abstrakce nečiní nejmenší potíže. Horší je to s těmi, kteří již někdy něco programovali – ti odmítají přijmout barvu jako objekt a neustále se pídí po tom, jak je to s vnitřní reprezentací konkrétních barev, kde se pamatují jednotlivé barevné složky a další detaily, které se řeší o několik hladin abstrakce níž. Jak jsem již několikrát řekl, programátoři načichlí klasickým programováním se učí pomaleji.

Vraťme se ale k metodice. Nad oknem projektu si vysvětlíme, co to jsou třídy a objekty, ukážeme si, jak vypadá diagram tříd a popíšeme si jeho základní prvky. Při té příležitosti si povíme, co je to vlastně projekt a v jakém vztahu je projekt, program a třída.

Pak si děti v interaktivním režimu vyzkoušejí vytváření instancí, kterým pošlou několik zpráv. Během první hodiny tak jasně pochopí, jaký je rozdíl mezi třídou a instancí, co to jsou metody a jak reagují na zaslání zprávy. Naučí se, že s instancemi musí komunikovat posíláním zpráv.

V této počáteční fázi je vše postaveno na interaktivním režimu prostředí *BlueJ*, které umožňuje veškerou vykládanou látku názorně předvést.

#### 4.2 Návrátové hodnoty, zprávy s parametry

V následující hodině si ukážeme, že můžeme instancím posílat zprávy, v nichž je žádáme o nějakou informaci a předvedeme si, jak v prostředí *BlueJ* instance na zprávu požadující informaci reagují. Při tom si vysvětlíme základní vlastnosti datových typů Javy včetně jejich rozdělení na primitivní a objektové. Ukážeme si, jak postupovat, je-li návratovou hodnotou zprávy odkaz na instanci, který se následně umístí do zásobníku odkazů. Protože odkaz musíme před uložením do zásobníku odkazů nejprve pojmenovat, probereme zároveň pravidla pro tvorbu identifikátorů.

Ve výkladu pokračujeme zprávami, s nimiž můžeme objektu předávat i parametry, na základě jejichž hodnot objekt upravuje svoji reakci na zprávu. Ukážeme si, že parametry mohou být i objektových typů předvedeme si, jak je možné předat jako parametr odkaz uložený v zásobníku odkazů.

#### 4.3 Metody a atributy instancí a tříd, zdrojový program a definice první třídy

Pokračuji tím, že dětem prozradím, 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. Dopředu je připravuji na to, že ani v programu to nebude jiné.

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 dozví, že vedle atributů instancí existují i atributy tříd a povíme si, ve kterých situacích je vhodnější definovat atribut instance a kdy atribut třídy.

V závěru hodiny opustíme hrátky s hotovými programy a vytvoříme první vlastní program – prázdnou třídu. Vysvětlíme si, co je to konstruktor a naprogramujeme první skutečnou třídu *Strom*. Definujeme její konstruktory a vytvoříme několik jejích instancí. Zároveň děti dostanou svůj první domácí úkol: vytvořit třídu, která na požádání nakreslí na plátno nějaký jiný zajímavý grafický objekt (panáčka, panenku, šipku, ...).

#### 4.4 Konstruktory a metody s parametry, přiřazovací příkaz

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

Od konstruktorů s parametry pokračujeme k metodám. Při definici první metody ale zjistíme, že instance si musí někde pamatovat, z čeho je složena, a proto deklaruujeme první atributy. Seznámíme se s přiřazovacím příkazem a ukážeme si, jak je možno přiřazovat atributům objektových typů odkazy na instance. Pak postupně definujeme jednoduché přesunové metody, metody s parametry a metody vracející nějakou hodnotu.

#### 4.5 Testovací třída

Celou další hodinu věnujeme testování. Ukážeme si, jak se definuje testovací třída, vysvětlíme si, co je to testovací přípravek (test fixture), ukážeme si, jak jej vytvořit a jak jej můžeme později v případě potřeby upravit, a převedeme si, jak je v *BlueJ* možno generovat testy předvedením požadované akce.

Vysvětlíme si důležitost testování a ukážeme si některé další možnosti, které nám *BlueJ* při definici testů nabízí. Povíme si o programování řízeném testy (Test Driven Development) a ukážeme si, jak je možno splnit požadavek, abychom nejprve definovali test a teprve pak testovaný program.

#### 4.6 Zapouzdření, komentáře, dokumentace

V další hodině začneme programům dávat profesionální vzhled. Vysvětlíme si, co to jsou komentáře a jak se používají. Naučíme se psát jednoduché dokumentační komentáře a ukážeme si, jak *BlueJ* vygeneruje z těchto komentářů dokumentaci k celé třídě a posléze i k celému projektu. Od této chvíle se komentáře stanou povinnou součástí programů. Předvedeme si přepínání mezi implementačním a dokumentačním režimem editoru a vysvětlíme si rozdíl mezi rozhraním a jeho implementací. Při té příležitosti probereme nejdůležitější vlastnost objektově orientovaných programů – zapouzdření.

#### 4.7 Lokální proměnné, vstupy a výstupy, konstanty a literály

Další hodinu si vysvětlíme, jak se definují statické atributy a metody a rozšíříme o ně svoji třídu. Poté se věnujeme lokálním proměnným a jejich zvláštnostem. Vysvětlíme si, že parametry jsou vlastně lokální proměnné, které se od těch obyčejných liší pouze tím, že jim volající program může přiřadit počáteční hodnotu.

V další části si ukážeme dvě jednoduché metody, které pro nás otevrou dialogové okno, v němž uživateli oznámí zadanou zprávu nebo jej požádají o zadání nějaké hodnoty, kterou nám pak předají. Použití těchto metod si hned vyzkoušíme na příkladech.

Na závěr se seznámíme s literály a jejich zápisem a vysvětlíme si, proč je výhodnější používat pojmenované konstanty.

#### 4.8 Počítání vytvořených instancí, metoda `toString`, inkrementační a dekrementační operátory

Naše programy se nám utěšeně rozrůstají. V tuto chvíli již mají okolo 200 až 300 řádek a budou i nadále kynout. V této hodině si ukážeme, jak může třída počítat své instance. Naučíme se k tomu využít kombinaci statického atributu s konstantním atributem instance. Při té příležitosti si vysvětlíme i možné použití inkrementačních a dekrementačních operátorů, které se nám budou k tomuto účelu hodit.

Hned si také ukážeme, jak je možné takto vzniklé „rodné číslo instance“ využít k jejímu jedinečnému pojmenování. Seznámíme se přitom s metodou *toString*, která se používá v okamžiku, kdy je třeba převést danou instanci na textový řetězec.

Průběžně si také ukazujeme, jak pro všechny vytvořené metody hned připravit automatické testy, které je kdykoliv na požádání otestují.

Jako domácí úkol mají děti vytvořit novou třídu *Xkrement* se statickou metodou *test*, která názorně demonstuje chování různých verzí inkrementačních a dekrementačních operátorů ve výrazech.

## 4.9 Standardní výstup a prázdná třída, v útrokách testovací třídy

Ukážu dětem, že vedle výstupu do okna existuje ještě možnost posílat texty na standardní výstup. Metodu, kterou vytvářeli za domácí úkol, upravíme tak, aby používala standardní výstup a ukážeme si okno standardního výstupu, kam *BlueJ* tyto texty odesílá. Při té příležitosti probereme jeho výhody a nevýhody.

Pak se podíváme do zdrojového kódu testovací třídy, vysvětlíme si, jak *BlueJ* generuje automatické testy a jak můžeme tyto testy dodatečně modifikovat. Zároveň si ukážeme, jak je možno definovat testy vlastní.

## 4.10 Návrhové vzory, práce s debuggerem

Další hodina je cele věnována návrhovým vzorům a práci s debuggerem. Vysvětlíme si, co je to návrhový vzor a seznámíme se s třemi nejjednoduššími vzory: přepravkou (Messenger), jedináčkem (Singleton) a tovární metodou (Factory Method).

V závěru hodiny si ukážeme práci s debuggerem a předvedeme si výhody a nevýhody jeho používání.

Tím končí první část. V závěrečném domácím úkolu mají děti za úkol definovat třídu *Zlomek*, která umožní pracovat v programu se zlomky a zlomkovou aritmetikou.

## 4.11 Rozhraní

První část byla věnována především zapouzdření. V druhé části zahrneme do oblasti svého zájmu postupně polymorfismus a dědičnost. Začínáme konceptem rozhraní.

Na počátku dětem připomenou, že se jejich překrývající se obrázky při pohybu doposud navzájem umazávaly, takže změníme naše plátno za takové, které dokáže ohlídat, aby se tak nedělo. Třídu *Plátno* nahradíme třídou *AktivníPlátno*, jejíž instance již nebude pasivním objektem, na který se grafické objekty samy nakreslí, ale stane se manažerem, který se sám stará o to, aby se při přesunu jednoho objektu přes druhý spodní objekt neodmazával. Aby toho mohla dosáhnout, nemohu se již grafické objekty kreslit samy, ale chtějí-li být zobrazeny na plátně, musí se u tohoto manažera přihlásit. Manažer je přijme mezi spravované pouze tehdy, budou-li se umět na požádání nakreslit dodaným „kreslákem“ (objekt třídy *Graphics*).

Po tomto úvodu si vysvětlíme význam a účel rozhraní a ukážeme si, jak třídu k implementaci daného rozhraní přihlásit. Děti přihlásí své třídy k implementaci rozhraní *IKreslený* a vyzkouší si, že plátno opravdu pracuje tak, jak očekáváme. Současně si ověří, že samotná implementace metody požadované rozhraním nestačí a že je potřeba se k implementaci rozhraní v hlavičce třídy explicitně přihlásit. Zároveň si ověří, že toto přihlášení lze realizovat prostým natažením šipky v diagramu tříd – zdrojový kód upraví *BlueJ* automaticky. Děti se přitom začnou nenásilnou formou seznamovat s událostmi řízeným programováním a zpětně volanými funkcemi (callback functions), s nimiž má problémy ne jeden profesionální programátor.

Poté spolu definujeme rozhraní *IPosuvný* a třídu *Drc*, jejíž instance umějí posunout objekty typu *IPosuvný* o předem danou vzdálenost. Děti mají za úkol 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.

Za domácí úkol mají definovat rozhraní *INafukovací* a třídu *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“ a s třídou *Přesouvač*, která umí instance typu *IPosuvný* pomalu a plynule přesouvat do libovolné pozice či na libovolnou vzdálenost (i mimo plátno a zpět).

## 4.12 Dědění rozhraní

Při procvičování rozhraní si ukážeme, že to, že třída 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.

Děti se seznámí s třídou *Multipřesouvač*, která je schopna pohybovat několika objekty současně (hned si to také vyzkoušíme). Současně si umí zjistit, jestli pohybovaná instance implementuje rozhraní *IMultiposuvný*, a pokud ano, tak po skončeném přesunu zavolá její metodu *přesunuto*. Tu může instance využít k provedení nějaké akce svázané s dosažením cíle – např. k požádání o přesun na jiné místo.

Na závěr děti seznámím s třídami *Stanice* a *Linka*. *Linka* je přitom tvořena sadou cyklicky uspořádaných stanic, kde každá má svého předchůdce a následovníka. Za domácí úkol mají definovat třídu *Kabina*, jejíž instance se usadí na počáteční stanici zadané linky, zjistí si, která stanice ji následuje, té se zeptá na její souřadnice a na ně se nechá multipřesovačem přesunout. V cíli vyvolá multipřesouvač její metodu *přesunuto*, která si zjistí pozici následující stanice a opět se na ni nechá přesunout. Tak jezdí stále kolem dokola. To vše se děje na všech linkách zároveň. Děti tak několika málo příkazy vytvoří efektní multiprocesový program.

## 4.13 Dědění tříd

Ukážeme si, že vedle dědění rozhraní je možno vytvářet také potomka třídy. Děti se seznámí s podtřídou jako speciálním případem rodičovské třídy. Vlastnosti takovýchto potomků si ukážeme na příkladech tříd *Kruh* a *Čtverec*, které jsou potomky tříd *Elispa* a *Obdélník*, u nichž si předvedeme, jak vhodně definovaný systém tříd a podtříd může výrazně ušetřit práci a zmenšit počet metod, které je nutné napsat. Upozorním je na to, že dceřiné třídy by měly používat opravdu pouze k definici speciálních případů rodičovské třídy a neopakovat začátečnické chyby některých autorů učebnic, kteří definují geometrické tvary jako dceřiné třídy bodu.

Děti dostanou za domácí úkol vytvořit třídu, která nakreslí šipku, jež bude schopna se na povel *vpřed* přesunout na sousední políčko plátna. Děti jsou přitom rozděleny do čtveřic, přičemž v každé čtveřici má každý žák za úkol nakreslit jinou šipku. Šipka každého z nich pak bude mít definovanou metodu *vlevoVbok*, která vrátí odkaz na instanci šipky, která je sice na stejném políčku, ale je otočena o 90° vlevo.

## 4.14 Šipky a návrhový vzor Stav

V následující hodině si vyzkoušíme, jak šipky spolupracují. Definujeme společnou třídu *Šipka*, která bude umět kromě přesouvání se vpřed udělat ještě plnohodnotný *vlevoVbok*. Ukážeme si na tomto příkladu, že v řadě situací lze úlohy, které by klasický programátor řešil nejspíše nějakým rozhodováním, řešit elegantně pomocí sady vhodných implementací speciálního rozhraní.

Ukážeme si, že instance této třídy mohou při drobném vylepšení implementovat rozhraní *IRobot* a simulovat chování pohyblivých robotů na dvorku.

## 4.15 Balíčky, knihovny a knihovní třídy

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 *BlueJ* se s balíčky prozatím 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.

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

## 4.16 Dědičnost tříd, abstraktní třídy

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 programátorských zásad je pokud možno nekopírovat do jiných míst jednou napsaný kód, a ukážeme si, jak 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.

#### 4.17 Podmíněný příkaz, vyvolání výjimky

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 vyvoláním výjimky. Přitom děti seznámíme se základní sadou výjimek. Výjimky zatím pouze vyvoláváme, avšak neošetřujeme.

Za domácí úkol dostanou děti naprogramovat robota, který se bude s ostatními roboty pohybovat po dvorku se značkami a jeho úkolem bude sesbírat co největší počet značek. Další hodinu děláme soutěž, kdy na dvorek pustíme všechny roboty najednou a děti soutěží, čí robot bude nejúspěšnější.

#### 4.18 Cykly

Na příkladu robotů si ukážeme typické problémy řešené pomocí cyklů. Seznamujeme přitom děti jak s cykly s počáteční a koncovou podmínkou, tak s cykly s podmínkou uprostřed, které umožňují jednoduché řešení situací, kdy je třeba něco udělat před testem počáteční podmínky, resp. po testu ukončovací podmínky.

#### 4.19 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. Nyní již děti mají potřebné vědomosti k tomu, aby mohly začít vytvářet vlastní plnohodnotné roboty, kteří se pohybují po dvorku, umí zvedat a pokládat značky a dokáží se vyhnout zdem a ostatním robotům.

#### 4.20 Ošetření výjimek

Základní běh zakončujeme výkladem ošetření výjimečných situací.

## 5 Závěr

Doporučená metodika ani její nastíněný harmonogram není dogma. Důležité je, abychom rychlost výkladu vždy přizpůsobili zájmu a rychlosti chápání dětí. Bude-li v kroužku silná skupina aktivních, můžete pokračovat výrazně rychleji. Sejde-li se vám v kroužku skupina flákačů (to bývá tehdy, jsou-li tam silněji zastoupeni děti ze starších tříd), nemusí se vám podařit dodržet ani tento velmi mírný harmonogram.

## 6 Literatura

- [1] Pecinovský R.: Výuka objektově orientovaného programování žáků základních a středních škol, Objekty 2003 – sborník příspěvků konference, VŠB, Ostrava 2003, ISBN 80-248-0274-0
- [2] Pecinovský R.: Zásady správné výuky programování, [www.ceskaskola.cz](http://www.ceskaskola.cz)
- [3] Domovská stránka standardní edice jazyka Java: <http://java.sun.com/j2se/index.jsp>.
- [4] Domovská stránka prostředí *BlueJ*: <http://www.bluej.org>.