KOPENOGRAMY
A JEJICH IMPLEMENTACE V *NETBEANS*

Rudolf Pecinovský
ICZ a.s., Na hřebenech II 1718/10, 147 00 Praha 4,
VŠE Praha, Fakulta informatiky a statistiky, Katedra informačních technologií
rudolf@pecinovsky.cz

ABSTRAKT:

V současné éře OOP občas zapomínáme na to, že na konci objektové analýzy nás občas čeká návrh složitějších algoritmů. Pro jejich znázornění existuje několik grafických jazyků. Jedním nejnázornějších způsobů zobrazení strukturovaných algoritmů jsou kopenogramy. Příspěvek v první části podrobně probere způsob zápisu kopenogramů a ve druhé části pak seznámí s editorem, který je vyvíjen jako plug-in do vývojového prostředí *NetBeans*.

KLÍČOVÁ SLOVA:

Výuka programování, vývoj programů, kopenogramy, *NetBeans*.

1 ÚVOD

Objektové programování je v současné době hlavním paradigmatem používaným při vývoji rozsáhlejších aplikací. Učí se již nejenom na univerzitách, ale v mnoha případech i na středních školách. Můžeme diskutovat o tom, zda se učí skutečné objektově orientované programování či pouze nějaká jeho náhražka, nicméně musíme přiznat, že většina škol učí programovat v jazycích, které jejich autoři vydávají za objektově orientované.

Vedle objektově orientovaných jazyků se na řadě škol učí i alespoň základy objektově orientovaného návrhu. Při něm se jako jeden ze základních výrazových prostředků používá modelovací jazyk UML. Tento jazyk přichází s plejádou diagramů zachycujících jednotlivé fáze vývoje programového systému.

Jednou z fází je i návrh algoritmu, který řeší daný problém. Pro ten se většinou používá diagram aktivit. To ale není optimální řešení. Diagram aktivit totiž svoji koncepcí vychází z vývojových diagramů, které byly dlouho kritizovány za svoji přílišnou volnost, která svádí autory návrhu k porušování zásad strukturovaného programování.

Tuto jejich nevýhodu se pokusili odstranit Isaac Nassi a Ben Schneiderman ve svém návrhu způsobu zápisu strukturovaných algoritmů ([3]), který začal být označován jako Nassi-Schneidermanovy diagramy.

Nevýhodou Nassi-Schneidermanových diagramů byl způsob zápisu podmínek využívající šikmých čar. Na počátku osmdesátých let minulého století se začaly rozšiřovat mikropočítače a následně pak osobní počítače, které zpočátku používaly především alfanumerické displeje. Na nich se šikmé čáry zobrazovaly špatně. Proto byl vyvinut alternativní způsob zápisu, který se obešel bez šikmých čar a naopak využíval barvu, která se začala stávat běžnou. Tento způsob zápisu algoritmů byl nazván **kopenogramy**.

Kopenogramy umožňují zapsat celý algoritmus na alfanumerickém displeji. Je-li v použité znakové sadě k dispozici i čárová grafika zavedená u prvních IBM PC, lze na tomto displeji bez problému znázornit i potřebné rámečky.

Kopenogramy byly využívány především v různých učebnicích – např. v [4] až [11]. Vzhledem k nechuti programátorů dokumentovat vytvářené programy, pracnosti jejich ručního zakreslování a neexistenci vhodného univerzálního editoru se však jejich používání v praxi příliš nerozšířilo. Nyní by však mohl nastat zlom, protože je ve vývoji editor koncipovaný jako plug-in do jednoho z nejpoužívanějších vývojových nástrojů – *NetBeans*.

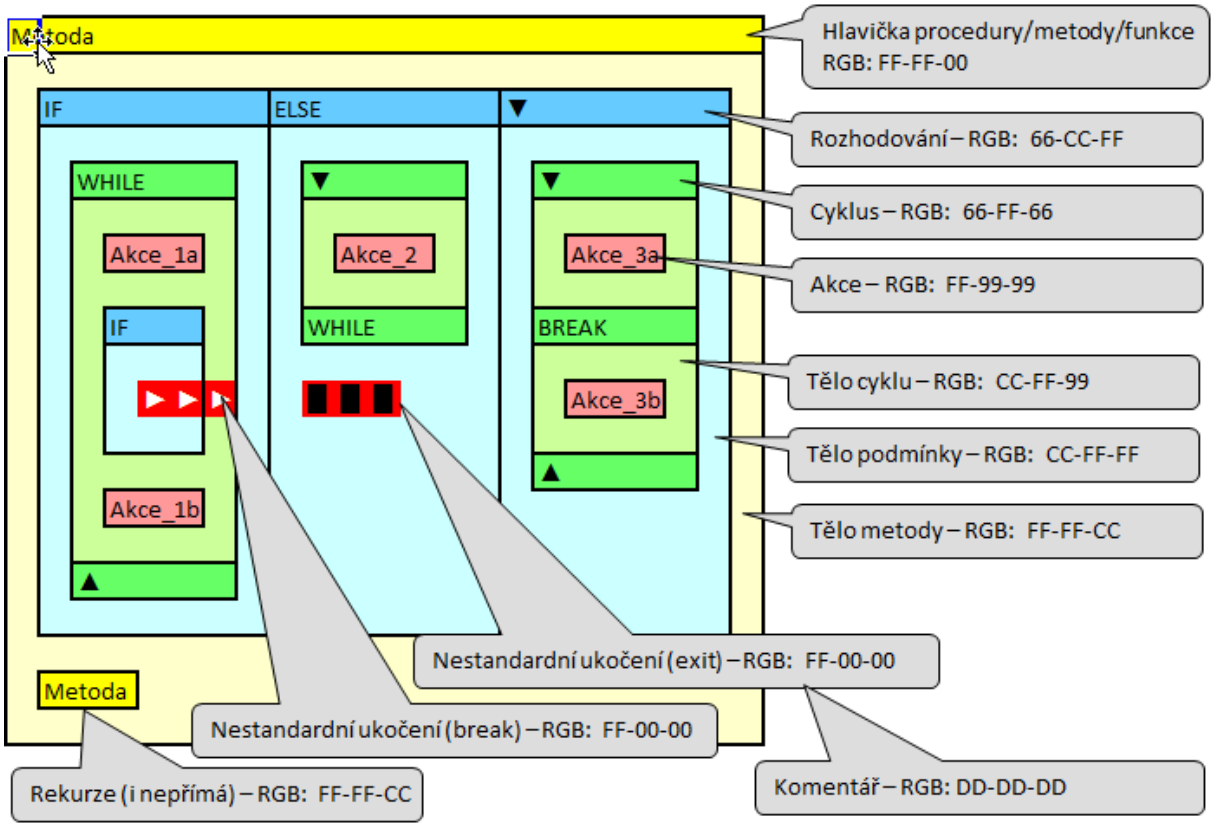
2 SYNTAXE KOPENOGRAMŮ

2.1 Základní algoritmické bloky

Kopenogramy využívají čtyři základní barvy (viz obr. 1):

- Žlutou barvou se vybarvují hlavičky procedur / funkcí / metod. Žlutá barva se používá i k zvýraznění rekurzivního volání.
- Červená barva se používá k vybarvení bloků akcí s výjimkou rekurzivních volání, pro něž se, jak již bylo řečeno, používá barva žlutá.
- Zelená barva se používá k vybarvení záhlaví a zápatí cyklů. V případě cyklů s podmínkou uprostřed se používá i vybarvení bloku s touto podmínkou.
- Modrá barva se používá k vybarvení záhlaví s podmínkami v podmíněných příkazech a přepínačích.

Pro zvýšení přehlednosti se pak světlejším odstínem barvy záhlaví vybarvuje i vnitřek příslušného bloku.



Obrázek 1: *Znázornění základních programových konstrukcí v kopenogramech*

Barvy použité na obrázku 1 nejsou povinné. Jejich RGB složky popsané v komentářích jsou pouze orientační a slouží především k předání informace, kterou klasický černobílý tisk sborníku příspěvků odstraní.

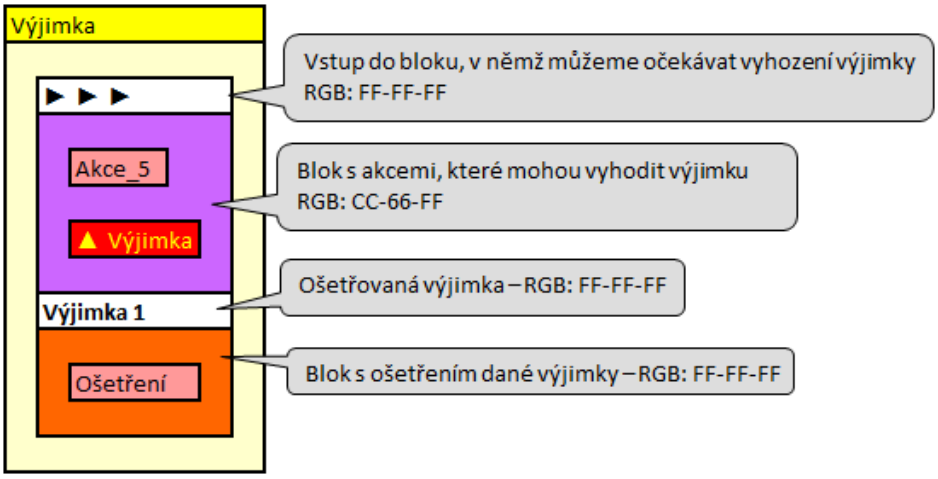
2.2 Výjimky a jejich ošetření

V době vzniku kopenogramů ještě nebylo zvykem používání výjimek běžné. Různé programovací jazyky je řešily různě. Syntaxe programovacích jazyků, které se v tehdejší době používaly při výuce programování, na ně většinou nemyslela a v mnoha případech se ošetřování

výjimek nepřednášelo ani při používání jazyků, které měly příslušné příkazy ve své syntaxi (např. některé verze jazyka Basic).

S nástupem jazyka Java v druhé polovině devadesátých let se však používání a ošetřování výjimek dostalo do základních kurzů programování, a bylo proto vhodné zanést tuto programovou konstrukci i do syntaxe kopenogramů. Pro znázornění bloku s očekávatelným vyhozením výjimky i bloku, který vyhozenou výjimku ošetřuje, opět použita barva (viz obr. 2):

- Hlavička bloku s očekávatelným vyhozením výjimky i následného bloku s jejím ošetřením se vybarvuje bíle.
- Tělo bloku s očekávatelným vyhozením výjimky se vybarvuje fialově.
- Tělo bloku s ošetřením vyhozené výjimky se vybarvuje oranžově.



Obrázek 2: *Zakreslení bloků, vyhazujících a ošetřujících výjimku*

2.3 Nestrukturované příkazy

Teoreticky je strukturované programování definované jako způsob tvorby algoritmů, při němž se používá pouze lineární sekvence příkazů, přičemž vedle jednoduchých příkazů může být členem této posloupnosti podmíněný příkaz `if … then … else` a cyklus. Jinými slovy program by měl být tvořen posloupností bloků, které mají vždy pouze jeden vstup a jeden výstup.

V praxi se však postupně ujal názor, že je vhodné povolit i příkazy, které sice narušují výše uvedená pravidla, ale po jejichž zavedení je program celkově přehlednější a pochopitelnější. Mezi takovéto příkazy patří předčasné ukončení bloku (příkaz `break` doprovázený někdy příkazem `continue`) a předčasné ukončení podprogramu (předčasný `return`). Svým způsobem mezi ně patří i vyhození výjimky.

No obrázku 1 je v levé sekci vnějšího podmíněného příkazu znázorněno předčasné ukončení bloku – v tomto případě cyklu. v Pravé sekci je pak znázorněna obdobná programová konstrukce, avšak tentokrát zakreslená jako cyklus s podmínkou uprostřed. Levá konstrukce je sice obecnější, ale v případě, kdy ji lze znázornit jako cyklus s podmínkou uprostřed, dáváme tomuto způsobu zobrazení přednost, protože většinou lépe vystihuje myšlenku celé konstrukce.

Ve střední sekci je pak znázorněno předčasné ukončení dané metody. Teoreticky bychom je mohli znázornit také jako předčasné opuštění bloku, přičemž tímto blokem by v tomto případě byla celá metoda, ale to je vhodné pouze tehdy, nevyskytují-li se vpravo od tohoto ukončení bloky, jejichž strukturu by probíhající „opouštěcí“ čára narušila.

Znázornění příkazu `continue` vypadá obdobně jako znázornění příkazu `break`, jediným rozdílem je to, že „opouštěcí trojúhelníky“ nesměřují doprava, ale nahoru.

3 IMPLEMENTACE KOPENOGRAMŮ

3.1 Dosavadní způsob kreslení

Jak již bylo řečeno v úvodu, kopenogramy se doposud používají poměrně omezeně. Za hlavního nepřítele jejich většího rozšíření považujeme především nedostatek literatury, v níž by byly použity a pak především neexistence nástroje, který by umožňoval jejich pohodlnou tvorbu.

Vzhledem k jejich obdélníkovému charakteru se doposud jako nejefektivnější nástroj kreslení kopenogramů jevil tabulkový procesor – nejčastěji *Excel*. Ten umožňuje nejenom snadné vybarvování a orámování obdélníkových bloků, ale současně poskytuje i rozumné prostředky, jak do již nakresleného kopenogramu vložit další blok. Je to sice stále pracné, protože je třeba provést ručně řadu dodatečných úprav, ale na druhou stranu je to neporovnatelně efektivnější způsob, než ten, který by nabízel klasický grafický editor. V *Excelu* jsou ostatně nakresleny oba obrázky v tomto příspěvku, a to včetně komentářů.

3.2 Plug-in do *NetBeans*

Jak již bylo naznačeno v úvodu, v současné době je ve vývoji plug-in do vývojového prostředí *NetBeans*, který umožní zobrazit kopenogram zadané metody. Rozhodli jsme se, že v zájmu co největší jednoduchosti nebude tento plugin umožňovat přímé kreslení kopenogramů, ale bude vždy zobrazovat pouze kopenogram zadané metody. Jakoukoliv požadovanou úpravu kopenogramu proto bude třeba udělat tak, že se upraví zdrojový kód příslušné metody. Dospěli jsme k závěru,

4 ZÁVĚR

Příspěvek v úvodu připomenul, že při přechodu n objektové paradigma nesmíme zapomínat na to, jak znázornit složitější algoritmy. Upozornil na to, že všeobecně používaný jazyk UML neobsahuje mezi svými diagramy žádný, který by umožnil strukturovaně zapsat algoritmus.

V další části pak seznámil se základní syntaxí grafického jazyka kopenogramů a způsoby zápisu jednotlivých algoritmických konstrukcí. Současně ukázal, jak kopenogramy řeší vyhazování výjimek a zobrazování některých dalších tolerovaných nestrukturovaných konstrukcí.

V závěru pak stručně charakterizoval současný způsob kreslení kopenogramů a upozornil na vyvíjený plug-in do prostředí *NetBeans*, který bude schopen zobrazit kopenogram označené metody.

LITERATURA

[1] KOFRÁNEK, Jiří; NOVÁK, Petr: *Kopenogramy – způsob grafické reprezentace programů. In Moderní programování* – Vinné 1987, Dům techniky ČSVTS, díl 4, Žilina, str. 11–161. 1987.

[2] KOFRÁNEK, Jiří; NOVÁK, Petr: *Kopenogramy – způsob grafické dokumentace programů.* In Mikropočítačová technika a výchova mládeže ČVUT, knižnice ČSVTS-FEL, Praha, str. 40–54. 1987.

[3] NASSI Isaac, SHNEIDERMAN Ben: *Flowchart techniques for structured programming,* ACM SIGPLAN Notices, Vol. 8 Issue 8, August 1973. ACM, ISSN: 0362-1340

[4] PECINOVSKÝ R.: Základy algoritmizace I, 602 ZO Svazarmu, 1985.

[5] PECINOVSKÝ R.: Základy algoritmizace II, 602 ZO Svazarmu, 1985.

[6] PECINOVSKÝ R., KOFRÁNEK J.: Jednoduché datové typy, 602 ZO Svazarmu, 1985.

[7] PECINOVSKÝ R., KOFRÁNEK J.: Strukturované datové typy, 602 ZO Svazarmu, 1986.

[8] PECINOVSKÝ R., KOFRÁNEK J.: Vyhledávání a třídění, 602 ZO Svazarmu, 1986.

[9] PECINOVSKÝ R., KOFRÁNEK J.: Dynamické datové struktury, 602 ZO Svazarmu, 1986.
[10] PECINOVSKÝ R., KOFRÁNEK J.: Modulární programování, 602 ZO Svazarmu, 1987.
[11] PECINOVSKÝ R., RYANT I.: Programování paralelní procesů, 602 ZO Svazarmu, 1987.