KOPENOGRAMS
AND THEIR IMPLEMENTATION IN *NETBEANS*

Rudolf Pecinovský
ICZ a.s., Na hřebenech II 1718/10, 147 00 Praha 4,
VŠE Praha, Faculty of Informatics and Statistics, Department of Information Technologies
rudolf@pecinovsky.cz

ABSTRACT:

In contemporary period of OOP we tend to forget that sometimes a proposal of more complex algorithms is expecting us at the end of object analysis. There are several graphic languages existing for their projection. One of the most illustrative ways of structured algorithms projection uses kopenograms. This paper deals with the detailed way of kopenograms´ transcript in the first part, the second part is devoted to an editor, which is developed as a plug-in into the developmental environment *NetBeans*.

KEY WORDS:

Lessons of programming, development of programs, kopenograms, *NetBeans*.

1 INTRODUCTION

The object programming is at present the main paradigm used for the development of more extensive applications. They are presented not only at universities, but in many cases also in secondary schools. We can discuss if really the object oriented programming is presented or only its imitation, however, we have to admit that majority of schools teach programming in languages proclaimed as object oriented by their authors.

Besides object oriented languages a lot of schools teach also the basics of object oriented proposal. The pattern language UML is used for it as one of the basic expressional means. This language brings a wide range of diagrams depicting individual phases of the development of the programming system.

One of these phases is an algorithm proposal that solves the given problem. Mostly the diagram of activities is used for it. But this is not the optimal solution. The diagram of activities comes out of development diagrams in its concept which were criticized for its excessive latitude for a long time that diverts the authors to break the principles of structured programming.

Isaac Nassi and Ben Schneiderman tried to remove this disadvantage in their proposal for transcription of structured algorithms ([3]), and it started to be called as Nassi-Schneiderman diagrams.

The disadvantage of Nassi-Schneiderman diagrams was the way of transcription of conditions using oblique lines. At the beginning of the 80´s of the last century the micro-computers and subsequently the personal computers started to spread which at the beginning used above all alpha-numeric displays. Depicting of oblique lines at these computers was bad. Therefore an alternative way of transcription has been developed dispensing the oblique lines and using on the contrary a color which started to be a current matter. This way of transcription was named **kopenograms**.

Kopenograms enable to write down the whole algorithm at an alpha-numeric display. If, in the used set of attributes, you have at your disposal also the linear graphic established at the first IBM PCs, also the needed frames can be depicted at this display.

The kopenograms were used above all in various textbooks – e.g. in [4] to [11]. With regards to a repulsion of the programmers to make the documentation of the developed programs, laboriousness of the hand-depicting and missing the suitable universal editor, using them in practice was not much spread. Nowadays, a turn could occur because the editor outlined as plug-in into one of the mostly used development tools – *NetBeans* is being developed.

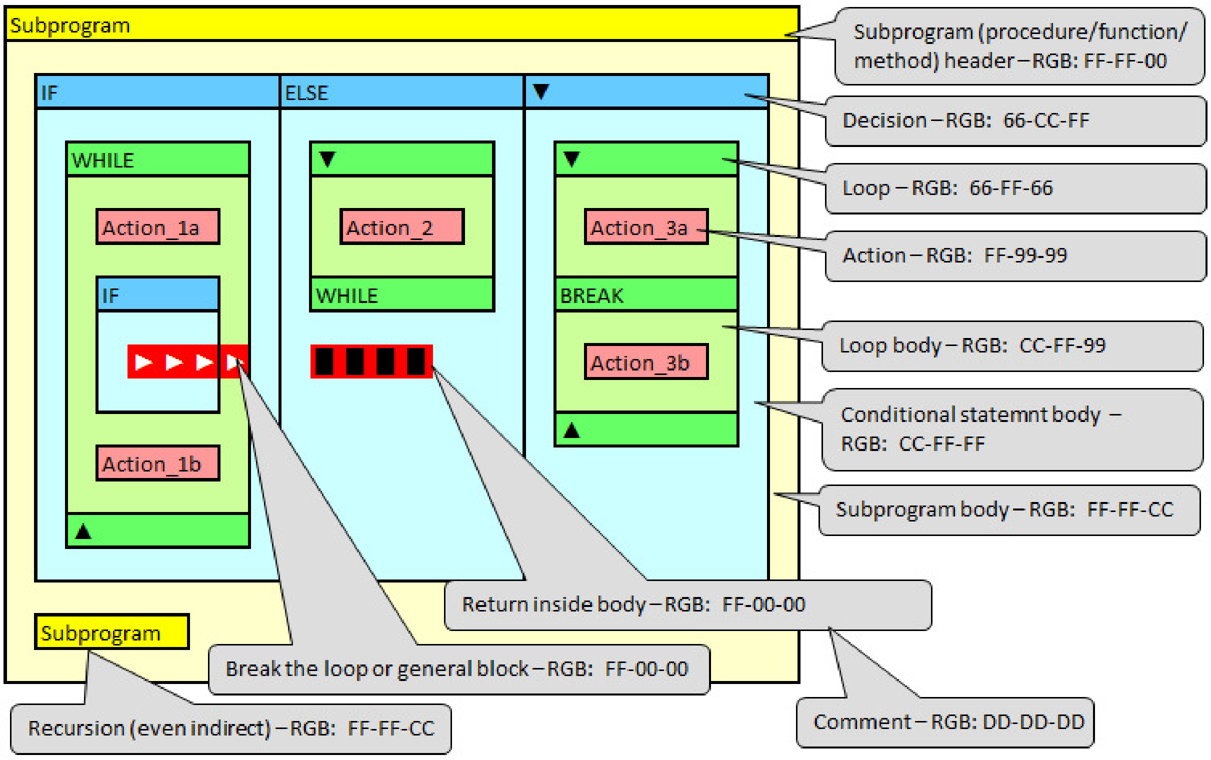
2 SYNTAX OF KOPENOGRAMS

2.1 Basic Algorithmic Blocks

Kopenograms use four basic colors (see figure 1):

- The yellow color is used for headlines of procedures / functions / methods. It is also used for emphasizing the recursive calling.
- The red color is used for coloring the blocks of actions except the recursive callings which are, as was already said, yellow.
- The green color is used for coloring the headings and footings of cycles. In case of cycles with a condition in the middle, it is used also for coloring the block with this condition.
- The blue color is used for coloring the headings with conditions in conditioned commands and switchers.

For increasing the lucidity the brighter shade of the color is used for the interior of the relevant block.



Subprogram
IF
ELSE
▼
WHILE
Action_1a
IF
Action_1b
▲
▼
Action_2
WHILE
▼
Action_3a
BREAK
Action_3b
▲
Subprogram
Subprogram (procedure/function/ method) header – RGB: FF-FF-00
Decision – RGB: 66-CC-FF
Loop – RGB: 66-FF-66
Action – RGB: FF-99-99
Loop body – RGB: CC-FF-99
Conditional statemnt body – RGB: CC-FF-FF
Subprogram body – RGB: FF-FF-CC
Return inside body – RGB: FF-00-00
Break the loop or general block – RGB: FF-00-00
Recursion (even indirect) – RGB: FF-FF-CC
Comment – RGB: DD-DD-DD

***Figure 1:** Demonstration of basic program constructions in kopenograms*

Colors used at figure 1 are not compulsory. Their RGB components described in commentaries are only orientational and serve above all to pass over the information which the classic black-white print of anthology removes.

2.2 Exceptions and Their Treatment

In the time when kopenograms emerged using of exceptions was not current. The various programming languages solved them in various ways. The syntax of programming languages used at that time during programming lessons, did not take them into account and in many cases treating with exceptions was not presented neither in using the languages which have the relevant commands in its syntax (e.g. certain versions of Basic language).

However, when Java language started in the second half of the 90´s using and treating with exceptions came to basic courses of programming, and therefore it was suitable to bring this programming construction into kopenogram syntax. For depicting the block with presumable throwing of an exception as well as of a block that treats with the thrown exception we use again a color (see figure 2):

- The block´s heading with presumable throwing of an exception as well as with a following treating block is colored white.
- The block´s body with presumable throwing of an exception is colored violet.
- The block´s body treated with the thrown exception is colored orange.

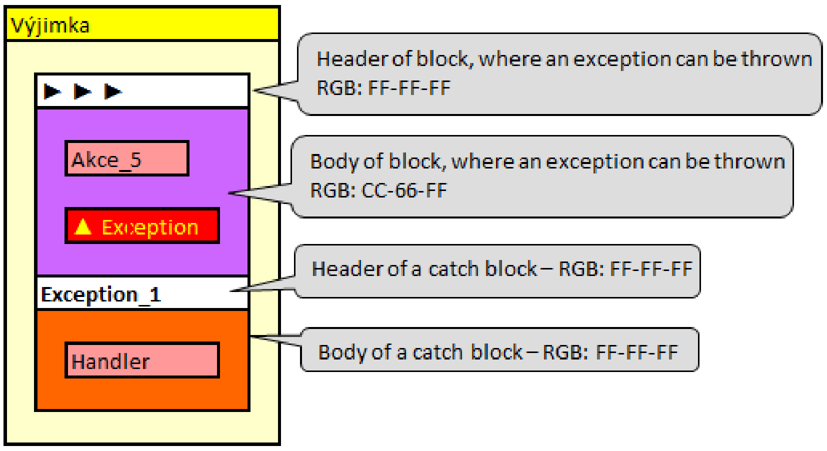

Výjimka
Akce_5
Exception
Exception_1
Handler
Header of block, where an exception can be thrown RGB: FF-FF-FF
Body of block, where an exception can be thrown RGB: CC-66-FF
Header of a catch block – RGB: FF-FF-FF
Body of a catch block – RGB: FF-FF-FF

Figure 2: *Depicting of blocks, throwing and treating with the exception*

2.3 Unstructured Commands

Theoretically the structured programming is defined as a way of treating the algorithms for which the linear sequence of commands is used, and member of this sequence can be, besides simple commands, also a conditioned command `if … then … else` and a cycle. To say it in other words, the program should be created by a sequence of blocks which have – each of them - only one input and one output.

However, gradually an opinion became established in practice that it is suitable to allow using of commands that disrupt the above stated rules, but after their implementation the program is generally more transparent and more understandable. The premature finishing of a block (a command `break` accompanied by a command `continue`) as well as the premature finishing of a sub-program (premature `return`) belong among such commands. Also the throwing of an exception belongs to a certain degree among them.

Figure 1 shows in the left part of the outer conditioned command a premature finishing of a block – at this case a cycle. In the right part a similar programming construction is depicted, but nowadays drawn as a cycle with a condition in the middle. The left construction is more general, but in case when it can be depicted as a cycle with a condition in the middle, we prefer this way of depicting because it gives a true picture of the idea of the whole construction.

In the middle section a premature finishing of a given method is depicted. Theoretically we could demonstrate them also as premature leaving of a block because this block would mean the whole method, but this is suitable only if blocks, whose structure would disrupt the “leaving” line, don´t occur on the right from this finishing.

Depicting of the command `continue` looks similarly as depicting of the command `break`, the only difference is that the „leaving triangles“ are not oriented to the right, but upwards.

3 IMPLEMENTATION OF KOPENOGRAMS

3.1 Current Way of Depicting

As was already said in the introduction, kopenograms are still used to a limited extent. The main „enemy" of more extensive using them we can consider above all a lack of literature, in which they would be used and further a lack of a tool which would enable their comfortable creation.

Regarding to their rectangle character the most effective tool of drawing the kopenograms can be considered a table processor – most often *Excel*. It enables not only easy coloring and framing of rectangle blocks, but at the same time provides also a reasonable means how to include another block into an already outlined kopenogram. It is laborious, because a number of additional adjustments is necessary to be done manually, but on the other side, it is incomparably more effective way than the one offered by a classical graphic editor. Both pictures of this paper including commentaries are drawn in *Excel*.

3.2 Plug-in to *NetBeans*

As was indicated at the introduction, at present the plug-in into the developmental environment *NetBeans* is being developed, which will enable to depict the kopenogram of an assigned method. We have decided that for the biggest simplicity this plug-in will not enable a direct drawing of kopenograms, but will always depict only a kopenogram of an assigned method. Any required adjustment of the kopenogram will therefore be needed to make in such a way that the source code of the relevant method should be modified.

4 CONCLUSION

At the opening this paper reminded that when transferring to an object paradigm we cannot forget the fact how to depict more complex algorithms. It also drew your attention to the fact that the generally used UML language does not continue any diagram which would enable to write down an algorithm in a structured way.

In the next part this paper presented the basic syntax of kopenogram graphic language as well as ways of transcription of separate algorithmic constructions. At the same time it showed how kopenograms solve throwing of exceptions and depicting of some other tolerated non-structured constructions.

At the conclusion it briefly characterized a contemporary way of drawing kopenograms and drew your attention to a developed plug-in into the *NetBeans* environment, that would be able to depict a kopenogram of a marked method.

LITERATURE

[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.