Kopenograms and their implementation in BlueJ

Rudolf Pecinovský, Marek Chadim
Department of Information Technologies
University of Economics, Prague
Prague, Czech Republic
rudolf@pecinovsky.cz, Marek.Chadim@seznam.cz

***Abstract*—Although currently the bulk of the most common algorithmic tasks are included in libraries of programming languages, it is necessary to realize, that upon completion of object oriented design of application, we still not avoid of using more complex algorithmic constructions. For its visual projection and easier understanding several graphic languages are used. *Architecture First* methodology for its purpose prefers kopenograms, as one of the most suitable method of displaying structured algorithm. This paper deals with the tool, which was added to IDE BlueJ in order to improve support of Architecture First methodology by this IDE and which allows students to show kopenogram of selected method in a simple manner in interactive mode.**

Keywords—kopenograms; Architecture First; OOP; algorithm; BlueJ

I. Introduction

Objected oriented programming languages are currently the most widely used around the world. Simultaneously, everything suggests, that this situation will persist. It is therefore natural, that the most widespread methodology of teaching programming is *Objects First*. For this purpose was developed educational IDE *BlueJ*. This IDE allows to start teaching of programming in really objective way. It means, that teaching starts in interactive mode, which allows students to better understand objected oriented principles. Unfortunately, this method of education is quickly abandoning its premise and interpretation is starting to go towards the syntax an algorithmic constructions. So, even though the *Objects First* methodology is based on the excellent idea, untapped potential is obvious at first sight.

Architecture First methodology, developed on Department of Information Technologies at The University of Economics in Prague, is trying to eliminate these drawbacks. This methodology claims, that if students have to learn how to make really good objected oriented architecture of application, they have to come into contact with this from the beginning, in order to have enough time to acquire thinking in objects. Therefore students are on the layer of architecture since the first lessons. This ensures that students are not unnecessarily distracted by syntax and programming construction, so they can fully concentrate on architecture design of application.

All the code is created by code generator, which allows to work in the interactive mode [2]. This generator is currently also integrated to modified version of *BlueJ*, which is developed in University of Economics, in Prague in order to better meets the needs of *Architecture First* methodology. The textbook [4] can serve as an example of how teaching is designed in accordance with this methodology. However it is important to say, that in time, when this book was written, IDE BlueJ was not modified for the needs of *Architecture First* methodology, so the book does not use the advanced code generator features, which were added later.

Though the Architecture First methodology tries interpretation of algorithmic construction delay as much as possible, it is obvious, that students will sooner or later come into contact with it. After the object analysis is completed, it is often inevitable to design more complex algorithmic construction.

In order to allow students to better absorb interpretation of algorithm is for this purpose used several method of projection of its structure. According to the *Architecture First* methodology the best way is to use kopenograms. Reasons, why kopenograms are the best graphical interpretation of algorithms are described in next section.

II. Kopengrams

A. History of kopenograms

Currently the most common form, which is used for representation of algorithm, is flowchart. Disadvantage of flowcharts lies in the fact, that they do not coerce users to construct algorithm in accordance with the principles of structured programming.

One of the responses to elimination of this drawback was the emergence of Nassi-Schnneiderman diagrams that came with it, but brought another problem. For representation of condition it uses oblique lines, which was difficult to display on formerly common used, alphanumeric displays. In the eighties this shortage resolved the creation of kopenograms, which use for its representation colored rectangles.

B. Syntax of kopenograms

The basic structural element of kopenogram is block, which presents specific element of algorithm. This element is represented by rectangle, filled with color depending on meaning of displayed element. Individual elements can contain other elements in its bodies. It means that elements are nested into each other, which shows structure of algorithm. The form

of some algorithmic block varies, depending on their meaning. Individual elements can contain following parts [3]:

Header – forms the upper section of the block and is tinged with a darker shade of its color. It also contains text representing name of this element or another form of description of its meaning.

Body – it is the biggest part of the element and is filled with lighter shade of appropriate color. It can contain another blocks. They are on the given level always displayed bellow each other, which clearly show their sequential execution.

Footer - lower part of the block, also uses a darker shade of that color is used to represent the end of the body cycle.

These parts are separated by horizontal parting line. Moreover some elements consist of several separate parts, which are horizontally or vertically connected (which means not in the sense described above). Example of kopenogram is shown on Fig. 1

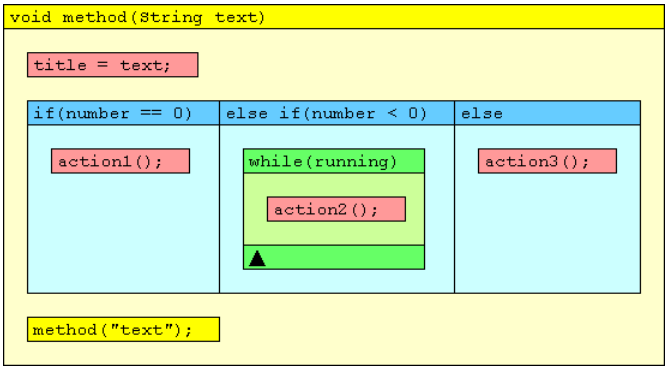

Fig. 1. Example of kopenogram

C. Basic blocks

As was mentioned, one of the integral and very important part of kopenograms are colors. They carry the advantage of easy orientation in algorithm, even from the distance, which means without having to read the labels of individual elements. The base consists following four colors [1]:

- **Yellow**, which shades illustrate methods (procedures/ functions) and recursive calls.
- **Green** is used for representation of cycles.
- **Blue** color shows conditional commands and switches.
- **Red** color is used to tint blocks representing individual elements, except that in the form of recursive call, as was mentioned.

D. Another blocks

Blocks described on Fig. 1 are basic, because they fully comply with structured writing of algorithm. However, sometimes can occur situations, in which is better to violate principles of structured design of algorithm. Therefore into structured programming were added representations of commands, which on the one hand violate principles of structured programming, however on the other hand, they can often simplify whole algorithm [1].

The premature termination of loop (command **break** or **continue**) and premature termination of whole algorithm (command **return**) are two of them. Representation of these elements in kopenogram shows Fig. 2, where the red rectangle with white triangles oriented to the right represents the `break` command, for leaving endless `while` loop in case of running is set to false.

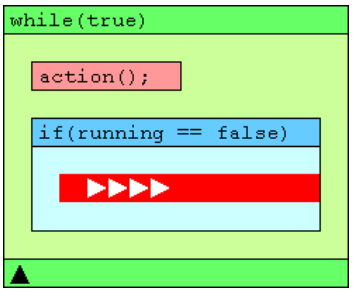

Fig. 2. Representation of premature loop termination

The `continue` statement is usually drawn similarly, except that the white triangles are facing upwards.

E. Exception handling

With the advent of Java has become a regular part of beginner programming courses work with exceptions. Therefore, it was necessary to take this into account in kopenograms [1]. An exception-handling mechanism can be viewed as a special composite block, consisting of part, in which it can be expected, that exception will be thrown, and of one or more block, which this exception catch and process. Kopenogram showing the exception handling mechanism is on Fig. 3.

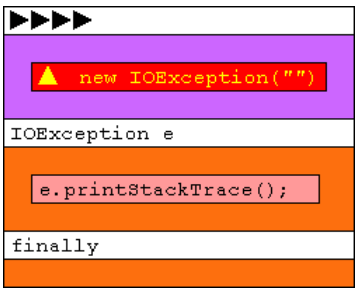

Fig. 3. Representation of exception throwing and catching

The block, in which can we expect exception throwing, is colored purple. Its header is white, with black triangles, illustrating entry into the body of this block, which contain statements, potentially throwing exception. Deep red block with yellow text means, that the exception is thrown, which is processed in an immediately downstream orange block with white header.

III. Implementation in BlueJ

As was indicated in introduction, for the purpose of improvement support of *Architecture First* methodology in *BlueJ* IDE, was added functionality, which allows users to show algorithm of the selected method by kopenogram. This should support especially the last part of teaching, in which is the more complex algorithmic construction that exceed the possibilities of integrated code generator are designed. For better support of *Architecture First* methodology, this tool is integrated into *BlueJ*.

For this purpose local menus of classes and objects (references to objects) were enriched with item, which invokes the dialog box, for selecting the method, which kopenogram should be shown. After selection, user can display kopenogram of the selected method by pressing the *Show kopenogram* button. Each kopenogram is depicted in a separate window. It allows comparing more algorithms, if necessary (for example in case of editing code of displayed method).

Kopenograms are created during compilation. Invocation dialog box from local menu of types in the class diagram and objects in object bench differs in the set of offered methods: objects show only instance methods, whereas classes show all. For that reason it is not necessary to create instance of appropriate class, in case of need to show kopenogram of its instance method.

Regarding colors, their meaning is given, however it is possible to define own colors for individual elements by editing configuration file *bluej.defs*.

In addition to elements described above, this tool can also display blocks, labels and static blocks (which is in its principle the same as method). An example of complex method, which contains also empty blocks and a label is shown on Fig. 4.

All of the included images were created in described tool and they are also example of its output.

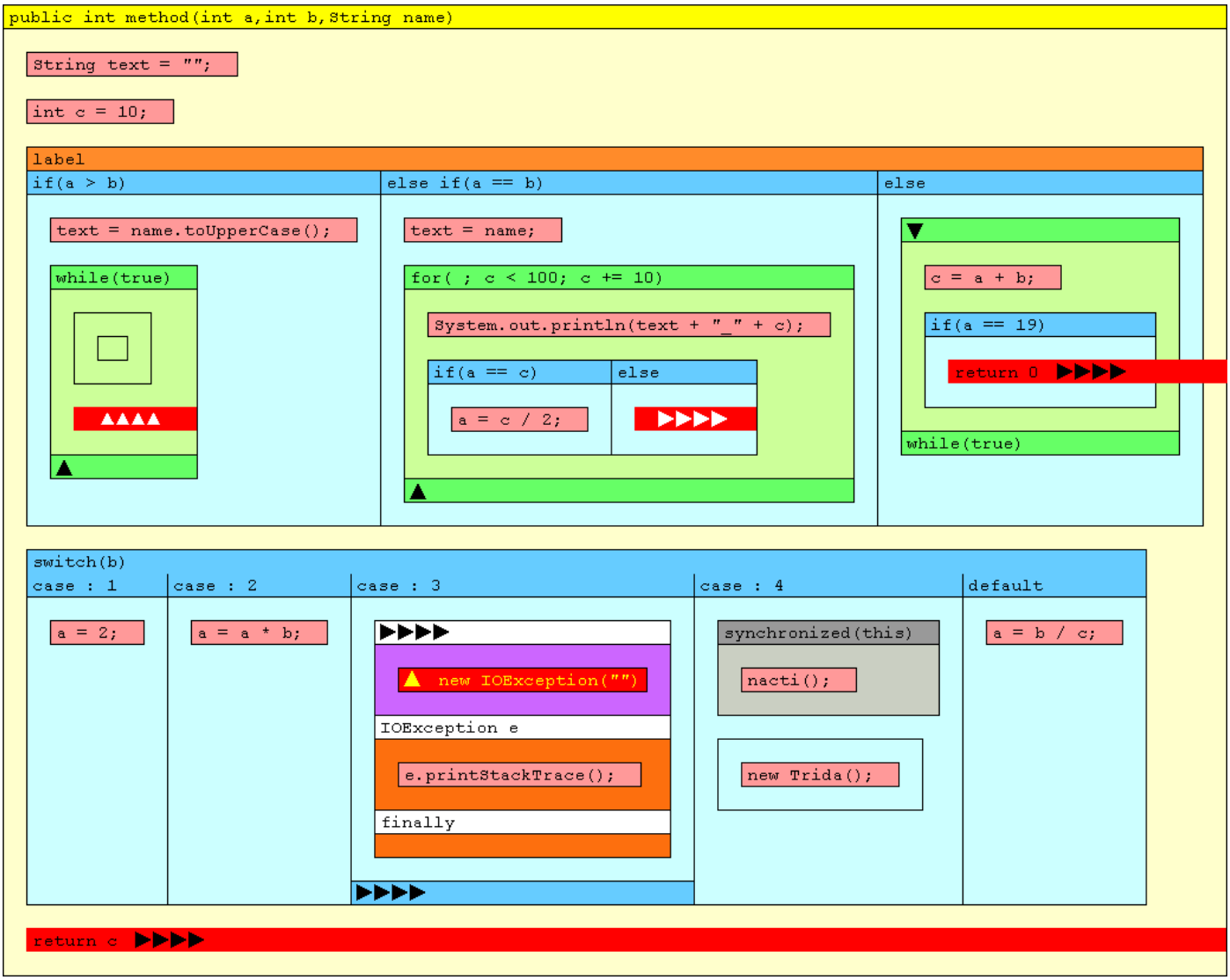

Fig. 4. Kopenogram of a complex method

IV. Future plans

Generator of kopenograms, described in previous section, works reliably, however there is still place for several improvements. One of the first things, that should be resolved, is situation that in case of long line of code is due to this final kopenogram too large to fit on the screen.

Currently is this shortage most significant in case of streams. It is because kopenograms are based on parsing of abstract syntax tree, which means, that even if user writes individual statements bellow each other, the tool represents is as one long command and does not deal with the fact, that user wrote it differently. However, even after this treatment, still a situation can occur that final kopenogram will be unable to fit the screen. Solution for this could be to allow export kopenogram in form of picture, which is than possible to adapt the screen.

Another good feature, which will be good to implement, is possibility of settings, allowing whether to display full syntax of the blocks and statements, or whether to show only simple description of meaning of individual elements. It would be useful in time, when students should not be distracted by syntax.

However, the most important future improvement, is allow to debug program using the kopenograms. It means, that particular element, which is executed, would be highlighted. So the user would have much more vivid idea, how exactly the particular algorithm works. And even without the knowledge of syntax.

V. Conclusion

The first section of this paper describes the reason of creation of *Architecture First* methodology and mentions its main principles. It also reminds that we still not avoid design of more complex algorithm. The next section is about history of kopenograms and explains reason for its creation. In the rest of this capture syntax of kopenograms is explained. Third section describes the tool, which was integrated into *BlueJ* and which allows users to display kopenogram of the selected method. The last capture suggests the future changes and improvements, which can be expected.

References

[1] PECINOVSKÝ R.: Kopenograms and their implementation in NetBeans. Proceedings of the 38th international conference on software development. Ostrava 2012. ISBN 978-80-248-2669-1.

[2] PECINOVSKÝ Rudolf: Principles of the Methodology Architecture First. Objekty 2012 – Proceedings of the 17th international conference on object-oriented technologies, Praha 2012. ISBN 978-80-86847-63-4..

[3] KOFRÁNEK, Jiří, PECINOVSKÝ, Rudolf, NOVÁK, Petr. Kopenograms – Graphical Language for Structured Algorithms. Las Vegas 2012. In: FCS 2012 – Foundations of computer science. [online] Las Vegas: CSREA Press, 2012, s. 90–96. ISBN 1-60132-211-9. URL: http://world-comp.org/proc2012/fcs/papers.pdf

[4] PECINOVSKÝ R.: OOP – Learn Object Oriented Thinking and Programming. Eva&Tomas Bruckner Publishing 2013. ISBN 80-904661-8-4.URL: http://pub.bruckner.cz/titles/oop