PROCEEDINGS

of the Union of Scientists - Ruse

Book 5

Mathematics, Informatics and Physics

Volume 11, 2014

RUSE

PROCEEDINGS
OF THE UNION OF SCIENTISTS - RUSE

Union of Scientists - Ruse
16, Konstantin Irechek Street
7000 Ruse
BULGARIA
Phone: (++359 82) 828 135,
(++359 82) 841 634
E-mail: suruse@uni-ruse.bg
web: suruse.uni-ruse.bg

Contacts with Editor
Phone: (++359 82) 888 738
E-mail: zzdravkova@uni-ruse.bg

PROCEEDINGS
of the Union of Scientists – Ruse

ISSN 1314-3077

ISSN 1314-3077

Proceedings

of the Union of Scientists– Ruse

Contains five books:

1. Technical Sciences
2. Medicine and Ecology
3. Agrarian and Veterinary Medical Sciences
4. Social Sciences
5. Mathematics, Informatics and Physics

The Ruse Branch of the Union of Scientists in Bulgariawas foundedin 1956. Its first Chairman was Prof. StoyanPetrov. He was followed by Prof. TrifonGeorgiev, Prof. KolyoVasilev, Prof. Georgi Popov, Prof. MityoKanev, Assoc. Prof. Boris Borisov, Prof. Emil Marinov, Prof. HristoBeloev. The individual members number nearly 300 recognized scientists from Ruse, organized in 13 scientific sections. There are several collective members too– organizations and companies from Ruse, known for their success in the field of science and higher education, or their applied research activities. The activities of the Union of Scientists – Ruse are numerous: scientific, educational and other humanitarian events directly related to hot issues in the development of Ruse region, including its infrastructure, environment, history and future development; commitment to the development of the scientific organizations in Ruse, the professional development and growth of the scientists and the protection of their individual rights.

The Union of Scientists – Ruse (US – Ruse) organizes publishing of scientific and popular informative literature, and since 1998 – the "Proceedings of the Union of Scientists- Ruse".

BOOK 5

"MATHEMATICS, INFORMATICS AND PHYSICS"

VOLUME 11

CONTENTS

Mathematics

Informatics

BOOK 5

"MATHEMATICS, INFORMATICS AND PHYSICS"

VOLUME 11

Physics

web: suruse.uni-ruse.bg

PARSER JAVA-CODE TO XML-FILE

Valentin Velikov, Malvina Makarieva

Angel Kanchev University of Ruse

Abstract: *This paper presents an own parser from Java to XML. Some of XML features are discussed, the need for an own parser is justified, the main algorithm, the basic software modules and some functionalities are presented.*

Keywords: *Computer Science, parser, Java, XML, program generation, software generation*

INTRODUCTION

In many places people work in the automatic software generation area. Some of the crucial matters are related to save human efforts and time for development.

This implies a more widespread use of a variety of systems for automated software creation or separate modules for the subject area modelling, to describe the requirements for custom software design and for documents generation.

In a limited budget a large software system can be developed on a modular basis. Each module can be implemented as a standalone application. Individual subsystems can use a common internal machine representation of the subject area.

In the absence of technological modules one can use an external developer, or open source software.

When the system is constructed as independent of each other subsystems instead of a common internal machine presentation, the separated modules can exchange data between itself using a some common (free) format for data import and export (e.g. - XML), thereby forming a technology chain for obtaining a final product [2].

DETAILED DESCRIPTION

One of the ideas in the developed theory is an existing program (Java- class) to be modified by an external application (or by an external developer, i.e. intermediate processing), in order to add new functionality, for which is currently not existent in a newly created system. This means Java-code (or something else - such as UML, another chart, business structures [1] and others) to be exported to an external system, to be processed and imported back again. For this purpose it is necessary to select some common format for representation of code, data, charts, and so on. The best choice is a text format or some variant thereof, e.g. XML format [4].

This raises the need to create a Java-code parser (converter) to XML-file. So existing Java application (code) will be exported through a universal, cross-platform performance (in this case - XML) to another system, that over the code handled (further processing, existing subprograms updating or new one generation), and the result code (in XML-format) should be imported to our base system again.

It is also possible that at the same time one system is a part of another system (with different developers), which is also used for code precondition. In this case it is also necessary to transfer the code (import / export) to another system.

For this purpose converters / parsers from Java-code (Java class) to XML-format and vice versa are needed.

Why XML-format?

XML (e**X**tensible **M**arkup **L**anguage) is a type of text-file (i.e. – a set of character codes). It is standardized and is widely known, it can be opened and processed with any text editor, not with a specializing systems only. It was created by World Wide Web Consortium (W3C) [6]. It resembles HTML, which is widely used in web applications design, but is somewhat limited. The main advantage of XML is its flexibility. It has no restrictions on the tags, used by the creator. It has a simple and logical structure. Its second advantage is the structure richness. Despite its simplicity, it is powerful enough to express complex data structures. Another XML advantage is the possibility to handle multiple languages. This option can solve some problems in certain web applications.

The main purpose, for which it was designed, is data transmission and storage [5]. Data is stored in text format that is making it easy to use in new operating systems, new applications, etc., without losing data due to the code sets standardization and characters. XML is a descriptive language that is not visualised. When using HTML the information is visualized, while with XML it is stored, transmitted, and in the new location it is restored again. This is done by using the user-defined service words, also called markers or tags, like these in HTML, placed at key points, i.e. it is a self-descriptive language. Thus XML defines a set of rules for documents encoding in text format that is both readable for people and for machines.

Tags in this language are defined by the creator [3], it is not like HTML, where tags are pre-defined. This makes the language suitable to describe various data types and at the same time be readable not by machines only but by people, as each name follows a certain logic.

XML converters have been created in many languages, and for different languages or parts of them, but only for some basic constructions. As a part of a developed software system a Java-class converter was required for covering basic language constructs at the beginning. As a newly created tool it was written in Java, and enables one java-file (describing not the class only) to be saved in a text form, with an XML tree structure.

Subsystem design

The system has to convert a Java-class (program) to XML-file. The logic can be separated into several parts (modules / classes) that implement different functionalities. The first class, tentatively called Parser, will contain all processing methods of the individual class elements, which are: class, package, import, methods, variables, method calls and comments.

Each XML-file has a hierarchical tree structure. In order to provide it in the final result of the subsystem, a class will be created which will format the converted text (tentatively called **XmlFormatter**). It will format the final XML-code into suitable tree structure, optionally using internal Eclipse XML libraries.

The created subsystem should be able to easily modify, i.e. it will be built on the Model-View-Controller (MVC) principle, i.e. it should be clearly separated into interface-model-controller. This would facilitate future changes to the parser. With this purpose, a third class controller, called **Messenger,** is created. It will transmit information between **the interface** and **the model**.

Modern systems do not imply using console applications. Such usually requires additional knowledge, related to specific operating systems. Therefore it is recommended to implement a modern graphical user interface, using a mouse or other interactive device, and a menu to make all necessary actions or settings. This idea will be realized by one class (tentatively called **MainWindow**), which will build the GUI. To simplify the application, it will have two text boxes (areas): the first place will store the Java-class

source code, and the second will receive the generated XML-code. Several buttons will trigger corresponding actions: **start** conversion, **save** the ready XML-file, **import** from a text file (the reverse converting - under development), **clean** the fields. Another class (tentatively called **Commander**) is required, which will read each line of code, understand what this line contain and forward it to the appropriate method in the **Parser** class.

1. Main process algorithm:

Key points in it are the following:

1.1 The necessary variables for the code processing and its output are input data;

1.2 Next is the processing cycle, in which:

1.2.1 It checks if a line exists. If there are no more, this means that the code conversion is finished. The output is formatted to be in XML-file form and it exits the program. If the condition is met (i.e. there are lines), it starts processing line by line and checking them for any of the elements: package, import, comment, class and variables;

1.2.2 The next condition which leads in itself to many other checks, is whether the series ends with ";". If it is not satisfied, then it checks for the presence of conditional statements, such as: if-else, while, for, return, etc.

2. The Class-diagram (fig. 1) provides a comprehensive view of all classes, relations between them, as well as variables and methods in them. The main related class is ***MainWindow***. It is connected with the next three classes: The first is ***Commander***, which in turn controls the other two - ***Messenger*** and ***Parser***. At the end of its work, after the XML-generation, ***Commander*** calls ***XmlFormatter***, carrying out the arrangement of newly generated code. The very object of the ***MainWindow*** class is created by its invocation, and visualizes the GUI.

The objects behaviour can be represented by the following algorithm:

- the user enters the correct code as required by the java-code international conventions;
- The ***Commander*** object accepts information and clears the lines transmission in Java-code;
- Then in a loop it goes through the lines, and each line is analysed and sent to conversion class (***Parser***). It, in turn, having finished with the process, returns the converted code to ***Commander***;
- ***Commander*** adds it to the final result, and once is comes out of converting the lines into XML, it calls the last object - ***XmlFormatter***. This class processes the resulting conversion code and arranges it in a XML characteristic tree structure;
- Processing code returns to Commander and it takes care of the visualization;
- The user saves file in a selected directory.

Some features are implemented with appropriate methods:

1. Class ***Commander:***

a. The method **private** String fixLongLines() is necessary because Java-editors usually wrap long lines of two or more lines automatically, and that is unacceptable for our code. The method returns a String type value, which contains correctly formatted the all code, to be successfully converted.

b. The method **protected** String execute() performs the basic language structures recognition. Using the scanner it will check for the next line availability, and after the removing spaces at the beginning and the end using the function ***trim()***), starts a loop for the code reading line by line until the end is reached. Whereupon:

- If the row size is greater than zero, i.e. line is not empty, the following checks are performed:

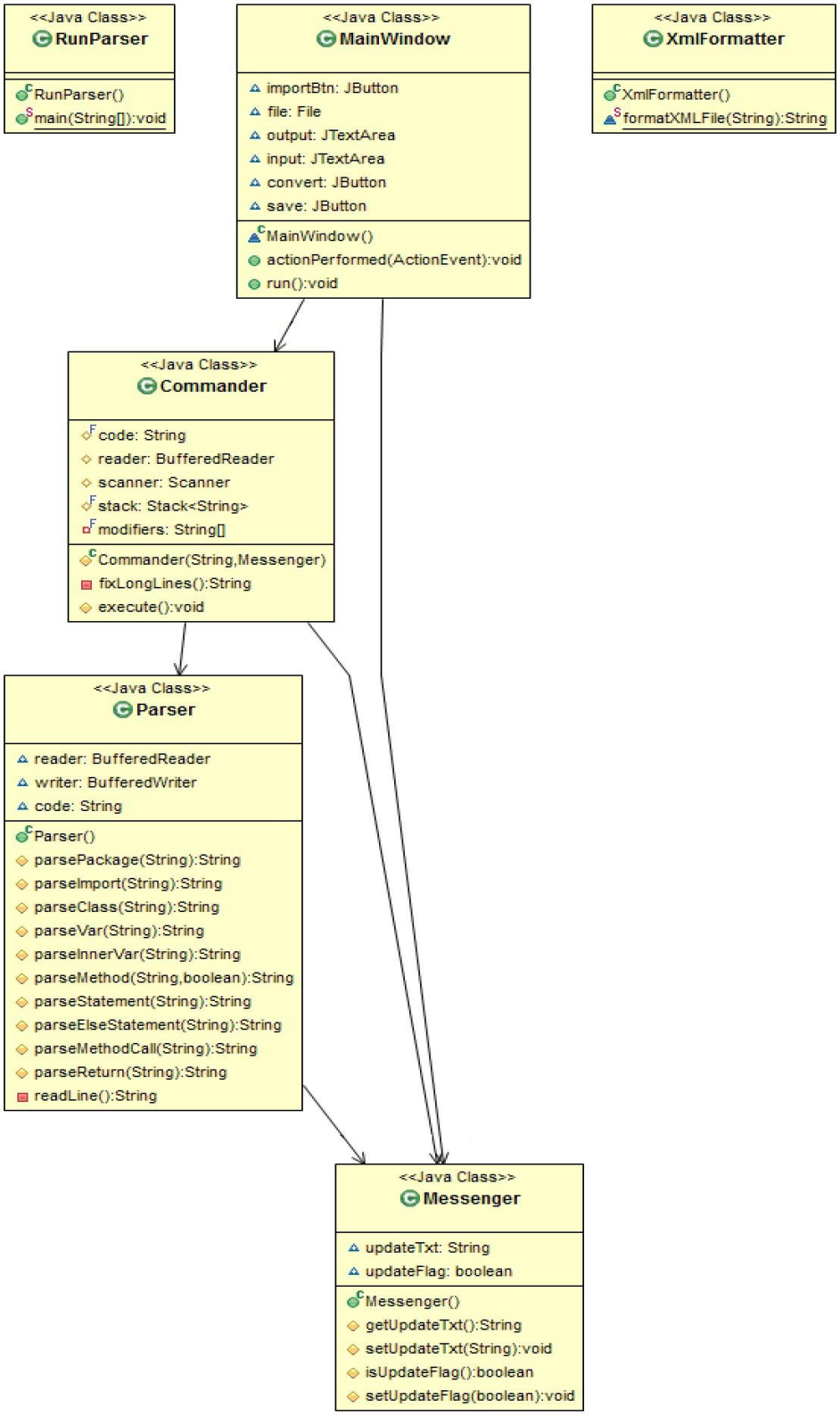

Fig. 1- Class diagram

- If the line starts with the reserved word "package", it runs the parsePackage(line) method, for package recognition, i.e. if the line contains the "package" phrase, it adds an opening tag, adds the symbols after the first whitespace character up to ";" and the package is closed with tag </package>
- If the line begins with "**import**", it runs the **parseImport()** method for the imported libraries description;
- If the line starts with "//", "*", "/*", then this is a comment. The value of the line is added to **result**, as initially the opening tag **<comment>** is added, and closing **</comment>** at the end;
- If the line contains the reserved word **class**, it runs the **parseClass()** method and adds to the stack the closing tag **</class>**;
- If the line contains any modifiers, loaded in the ***modifiers*** array, and ends with "**{**", then the line contains the method beginning. In this line, if the second space is after the first bracket "(" , then the method is a constructor and it calls the method convert function ***parseMethod(line, true)*** with second parameter **true**, which indicates the function should process the line as a constructor. Otherwise it is a method and the function is called as method conversion, with the second parameter **false**. Then it saves in the stack **</constructor>** or **</method>** depending on the methods flag;
- If the line does not begin with one of the modifiers and ends with ")" and contains "=", it is recognized as a method call. Then ***parseMethodCall()*** is called;
- If the line ends with ";", then it checks for modifiers. If no modifiers are recognised, then it runs the method for variable conversion - ***parseVar()*** and the flag is set to **false**, which means that there is no access modifier. Otherwise, the value of the flag becomes **true**, which indicates that the method has access modifier;
- If the line starts with ***if parseIfStatemant()***is called and ***</IfStatemret>*** is added to the into stack;
- If the line continues with ***else*** then it pulls one tag from the stack and adds ***</elseIf>***. Then the construction for converting method is executed – ***parseElseStatement(line)***;
- If the line starts with ***while parseStatement(line)*** is executed, and ***</whileStatement>*** is inserted into the stack;
- If the line starts with ***for*** then ***</forStatement>*** is added into the stack and ***parseStatemant(line)*** is executed;
- If the line starts with ***return*** then the ***parseReturn(line)*** method is called;
- If the line starts with „}“ then it checks if the stack is empty and the tag on the top of the stack is added to ***result***.
- If a non-recognisable line is reached, it is bypassed with an opening and closing tag respectively ***<literal> </literal>***.

- While the stack is not empty an element is taken from it;
- After exiting the loop it closes a class with ***</javaClass>*** tag;
- Finally the converted text is formatted.

2. Class ***MainWindow***.

This is the class of the GUI. It is "extended" with a JFrame for the graphics components and "implement" ActionListener (tracking the buttons).

3. Class ***Messenger***.

It is used for information transmission to the interface;

4. Class ***Parser***.

It defines all the methods for converting the elements of a class:

- ***parsePackage(String line)***
- ***parseImport(String line)***
- ***parseClass(String line)***
- ***parseVar(String line, boolean modifierExists)***
- ***parseMethod (String line, boolean constructor)***
- ***parseStatement(String line)***
- ***parseElseStatement (String line)***
- ***parseMethodCall(String line)***
- ***parseReturn(String line)***

5. Class ***XmlFormatter***

It is used for arranging the already converted XML-code into a form that meets the requirements of a correct XML document structure.

There is one method only:

static String formatXMLFile(String file) **throws** Exception

with the following logic:

- A few variables of library types for documents processing are declared;
- It reads the file passed as a parameter, and declares an output stream to the same file but already processed (using a temporary intermediate file)
- The library parser sets parameters, using submitted rules by which the code is processed, such as: coding, type of code, distances and their size, whether to have an initial line in the XML file;
- The code and the output stream are supplied into the parser variable "transformer". It returns the already formatted code into file;
- The streams are closed, and the finished code is returned as a String to the executed method.

The created subsystem could be easily changed to be built on the MVC principle (Model-View-Controller), i.e. to be separated in interface-model-controller. This would facilitate future changes in the parser. For this purpose, a third class (controller) was designed, called **Messenger**. It will transmit information between the interface and the model (classes realizing the conversion).

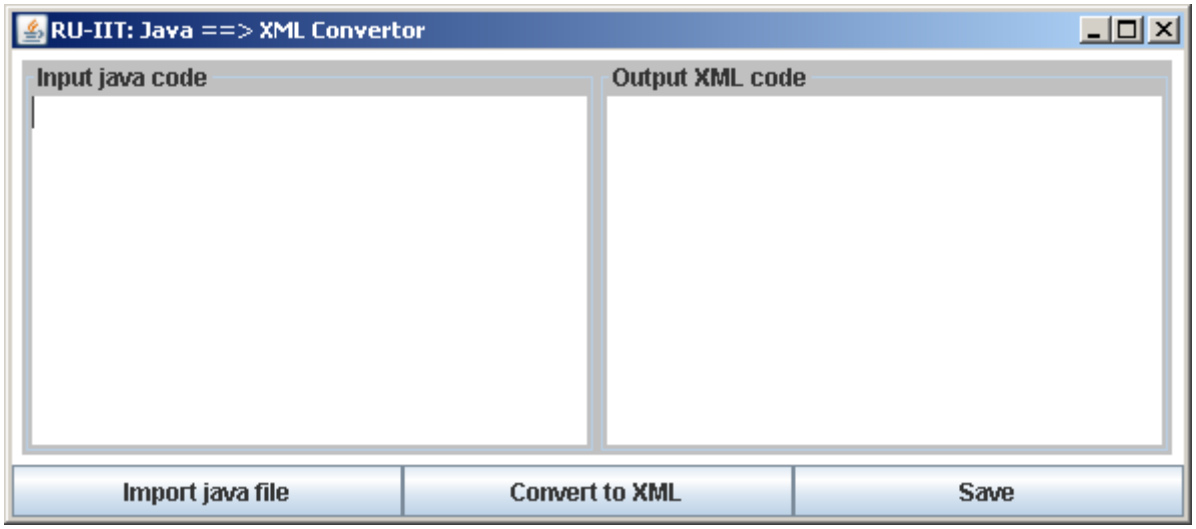

Fig. 2 – MainWindow - GUI

The present document has been produced with the financial assistance of the European Social Fund under Operational Programme "Human Resources Development". The contents of this document are the sole responsibility of "Angel Kanchev" University of Ruse and can under no circumstances be regarded as reflecting the position of the European Union or the Ministry of Education and Science of Republic of Bulgaria.

Project № BG051PO001-3.3.06-0008 "Supporting Academic Development of Scientific Personnel in Engineering and Information Science and Technologies"

REFERENCES

[1] Grigorova K., P. Hristova, G. Atanasova, J. Q. Trelewicz, Extracting Business Structures from XML Documents, Intenational Conferense "Automation and Informatics", Sofia, 2005.

[2] Valentin Velikov, Automatic program generation without internal machine representation, Proceedings of the Union of Scientists – Rousse, book 5, Mathematics, Informatics and Physics, 2013, Vol.10, in English, ISSN 1314-3077, pp. 72 – 77.

[3] Хироши Маруяма, Урамото Н., Тамура К. – XML и Java, Разработване на Web приложения, Addison-Wesley, Инфодар ЕООД-София, 2001, ISBN 954-761-017-1

[4] Wikipedia, XML - http://en.wikipedia.org/wiki/XML, 2014.

[5] W3Schools, XML Tutorial, http://www.w3schools.com/xml/default, 2014.

[6] W2C, XML - http://www.w3.org/XML/, 2014.

CONTACT ADDRESSES

Pr. Assist.Valentin Velikov,
Department of Informatics and Information Technologies
Faculty of Natural Sciences and Education
Angel Kanchev University of Ruse
8 Studentska Str., 7017 Ruse,Bulgaria
Phone: (++359 82) 888 326,
Cell Phone: (++359) 886 011 544,
E-mail: val @ ami. uni-ruse. bg

BSc. Malvina Makarieva
Department of Informatics and Information Technologies
Faculty of Natural Sciences and Education
Angel Kanchev University of Ruse
8 Studentska Str., 7017 Ruse,Bulgaria
Cell Phone: (++359) 0889 187 911,
E-mail: malvina.v.m@gmail.com

КОНВЕРТОР JAVA-КОД --> XML-ФАЙЛ

Валентин Великов, Малвина Макариева

Русенски университет „Ангел Кънчев"

Резюме: *Статията представя собствен парсър от Java към XML. Разгледани са някои особености на XML, обосновава се необходимостта от собствен парсър, представен е основният алгоритъм, базовите софтуерни модули и някои функционалности.*

Ключови думи*: информатика, парсър, Java, XML, генерация на програми, генерация на софтуер*

Requirements and guidelines for the authors - "Proceedings of the Union of Scientists - Ruse" Book 5 Mathematics, Informatics and Physics

The Editorial Board accepts for publication annually both scientific, applied research and methodology papers, as well as announcements, reviews, information materials, adds. No honoraria are paid.

The paper scripts submitted to the Board should answer the following requirements:

1. Papers submitted in English are accepted. Their volume should not exceed 8 pages, formatted following the requirements, including reference, tables, figures and abstract.

2. The text should be computer generated (MS Word 2003 for Windows or higher versions) and printed in one copy, possibly on laser printer and on one side of the page. Together with the printed copy the author should submit a disk (or send an e-mail copy to: vkr@ami.uni-ruse.bg).

3. Compulsory requirements on formatting:

- font - Ariel 12;
- paper Size - A4;
- page Setup - Top: 20 mm, Bottom: 15 mm, Left: 20 mm, Right: 20mm;
- Format/Paragraph/Line spacing - Single;
- Format/Paragraph/Special: First Line, By: 1 cm;
- *Leave a blank line under Header - Font Size 14;*
- Title should be short, no abbreviations, no formulas or special symbols - Font Size 14, centered, Bold, All Caps;
- *One blank line - Font Size 14;*
- Name and surname of author(s) - Font Size: 12, centered, Bold;
- *One blank line - Font Size 12*;
- Name of place of work - Font Size: 12, centered;
- *One blank line;*
- abstract – no formulas - Font Size 10, Italic, 5-6 lines ;
- keywords - Font Size 10, Italic, 1-2 lines;
- *one blank line;*
- text - Font Size 12, Justify;
- references;
- contact address - three names of the author(s) scientific title and degree, place of work, telephone number, Email - in the language of the paper.

4. At the end of the paper the authors should write:

- The title of the paper;
- Name and surname of the author(s);
- abstract; keywords.

Note: The parts in item 4 should be in Bulgarian and have to be formatted as in the beginning of the paper.

5. All mathematical signs and other special symbols should be written clearly and legibly so as to avoid ambiguity when read. All formulas, cited in the text, should be numbered on the right.

6. Figures (black and white), made with some of the widespread software, should be integrated in the text.

7. Tables should have numbers and titles above them, centered right.

8. Reference sources cited in the text should be marked by a number in square brackets.

9. Only titles cited in the text should be included in the references, their numbers put in square brackets. The reference items should be arranged in alphabetical order, using the surname of the first author, and written following the standard. If the main text is in Bulgarian or Russian, the titles in Cyrillic come before those in Latin. If the main text is in English, the titles in Latin come before those in Cyrillic. The paper cited should have: for the first author – surname and first name initial; for the second and other authors – first name initial and surname; title of the paper; name of the publishing source; number of volume (in Arabic figures); year; first and last page number of the paper. For a book cited the following must be marked: author(s) – surname and initials, title, city, publishing house, year of publication.

10. **The author(s) and the reviewer, chosen by the Editorial Board, are responsible for the contents of the materials submitted**.

Important for readers, companies and organizations

1. Authors, who are not members of the Union of Scientists - Ruse, should pay for publishing of materials.

2. Advertising and information materials of group members of the Union of Scientists – Ruse are published free of charge.

3. Advertising and information materials of companies and organizations are charged on negotiable (current) prices.

Editorial Board

ISSN 1314-3077