STUDIA UNIVERSITATIS MOLDAVIAE, 2016, nr.7(97)
Seria "Științe exacte și economice" ISSN 1857-2073 ISSN online 2345-1033 p.181-188

CZU: 004.4: 004.43

THE NATURE, THE BEAUTY AND THE DIFFICULTY IN JAVA PROGRAMMING

Dror BENAMI

Universitatea de Stat din Moldova

JAVA language in recent years is widely used for the reason that integrates multiple information technologies. JAVA benefits are not fully exploited. The article discusses some aspects of the design of Data Mining algorithms in Java.

Keywords: *Object Oriented Programming (OOP), Experiential Learning Cycle, Procedural Paradigm, Inheritance, Polymorphism, Portability, High Performance Computing* (*HPC*), *Deployment, Abstraction*.

JAVA: NATURA, FRUMUSEȚEA ȘI DIFICULTĂTILE PROGRAMĂRII

Limbajul JAVA în ultimii ani se utilizează pe scară largă dat fiind că integrează mai multe tehnologii informaţionale. Avantajele JAVA nu sunt pe deplin exploatate. În articol sunt discutate unele aspecte de proiectare a algoritmilor de Data Mining în limbajul JAVA.

Cuvinte-cheie: *Programare Orientată pe Obiect (POO), ciclu experienţial de învăţare, paradigmă procedurală, moştenire, polimorfism, portabilitate, High Performance Computing (HPC), implementare, abstractizare.*

Introduction

There are a growing number of worldwide Java developers and users in recent years. It is difficult to assess the precise number of developers and users of the language, but estimates based on the number of software developers in the world; If by number of JAVA courses (frontal and pertaining to teleprocessing), and if by the Internet stuff (YouTube, Google, Amazon, forums, search engines, etc.). All these indicate a significant increase in Java language usability in various passageways. Tiobe Software [1] company vote on Java as the preferred language for developers. The language was ranked in the first place in February 2016, compared to second place in 2015. An estimation given at 2012 shows that there are between 9-10 million Java developers [2]. This estimation was given by Oracle and Wikipedia companies. Of course, this evaluation only grew over the years.

I will present below two research questions relevant to this article:

- Can we point to a direct link between learning Java and higher/lower performances than those previously studied procedural programming?
- What is the degree of internalization of concepts (inheritance, polymorphism) and the use of object-oriented programming, by actual programmers?

So, let's try to understand what makes Java language popular and useful?

Different Programming Paradigms

We have to understand first what the meaning of OOP is (object-oriented programming) and the perceptual difference between different methods and programming paradigms [3, 4]. The scope of this subject is too wide and therefore the discussion and example should focus JUST on the difference between the OOP-paradigm and procedural programming paradigm.

The two paradigms belong to imperative paradigms. The guiding principle in object-oriented programming is the abstraction of the programmatic ideas problem, abstract processes and structures, which other frame-works integration and/or change of a problem (or other common ownership) will be easy. The OOP paradigm reflects and indicates the analyzed process of the problem and the required application: from narrow-scope of a problem view into wide-broader scope and broader point of view. This observation has many donors below the development process and change the code, depending on the varying requirements. In procedural program-ming paradigm, the observation is mainly based on deconstruction. The observation known as top-down analysis, till we get into kernels programmable bases. Any change in the definition of the problem (functional changes, additions, updates etc.) would require a significant change, and sometimes the software should be rewritten.

An illustrated example

Suppose we plan a house; its windows, doors and rooms. Under the procedural programming paradigm, we can look at every detail planned, independently of other information. For example: the main entrance door, entrance door to the room, door to courtyard etc. However, in object-oriented programming, we can define the class of "*DOORS*" and thus, all the doors can belonged and classified into one common type of doors; objects, which might have a lot of common features. For example, all made of wood, all given to dismantling and assembly, high temperature resistant, etc. So, incidentally, we can also create differentiation in relation to a particular door had features in relation to the rest of the doors (see "*inheritance*" below). Any changes in planning the house will be much simpler under object oriented paradigm rather than using and implementing it in procedural programming paradigm. Add or drop off of the door, is simple and almost immediately, as well as adding or updating an attribute. Not so in the procedural paradigm that requires sometimes dipper programming entries, written code that requires wide code changes, mostly. If the programming perception of object-oriented paradigm is implemented correctly, the developed-written code is very effective, while shorter race, exploit system resources effectively and efficiently, require less time to debug and be well prepared for changes, updates and future uses. We need to remember that software development cost a lot of money, as the human resources are very expensive resource. Using Java effectively can save a lot of money to companies.

Java Language Characteristics

Java supports and implements object-oriented programming paradigm [5, 6]. The language has many advantages, but just the main of them would be mentioned here.

a) OOP based & Inheritance attribute

Based on this significant OOP [7] advantage, Java includes the basics bricks of classes, objects and actions (Operations). The term "*object oriented*" by itself is wide, and reflects wide range of subjects which relate to this term and concept. Object-Oriented is not only computers and software term. The basic feature/attribute of *inheritance* is built-in in the language kernel. Objects in Java language are simple to implement, in which we can apply attributes to class or object, and it can be applied to other objects or groups of objects.

Suppose we want to define a class of "*Clocks*". This class can include multiple objects, each of which is a "*clock*". I.e.: hourglass, a digital clock and a sundial. All these objects have common features of being "*watches*" about: their measuring instruments (in general), time measurement capability (in particular) and more. On the other hand, despite belonging to the same class, there is a significant differentiation between objects of the same class, as the manner of measuring time, accuracy, and other operating mechanism.

Practically, non-object oriented languages can be expensive in the time taken to write the code and allocating computer resources at all.

b) Polymorphism attribute

Another important feature of the language is the *Polymorphism* attribute. In this feature we can define an object, object properties, and define it as a prototype of another object (*subtype*). This feature integrates with methods (virtual method), used in and by the language, and in the process of transferring parameters.

As simple example for clarification, suppose we need to exchange the wheel drive in our car. Proposed algorithms should be looked like this:

- Pull out the jack (Jack)
- Remove the spare tire
- Release the wheel bolts
- Raise the vehicle using a jack
- etc.

This algorithm is a "*genetic*" algorithm. It means that it could be suitable for ***any*** type of vehicle: private, commercial, truck and more. The process of lifting, loosening the screws, etc. - is different from the vehicles; At least among some of them. Therein lie the feature of "*multi-way*". The code is written as abstract code rules, "*wearing" multiple forms with respect to a category of the vehicle*"; depending on individual object it refers to.

c) Portability

Java language is "*riding*" on the operating system, up above its running environment. This attribute is well known as "*Cross Platform or Portability*" feature; which empower the language ability to run in different

environments and a variety of operating systems. In languages that have not this feature, we are required to adapt software and/or custom code changes for each OS individually; which requires human resources, time and costs. This attribute is mostly significant advantage and critical, as it comes quite prominently manifested in the running of applications (apps) for cell phones, famously operated under several different operating systems.

d) High Performance Computing (HPC)

This feature derived from the process of compiling and testing written code. This can be done before and while running the application, and methods of language creates executable code optimization; including disposal processes (*Garbage Collection*). This means quick, well-debugged runtime code and optimal utilization of the system resources.

e) Deployment Process

This feature is derived from the ability of the language to be portable (mentioned above). The capability of the final product to run, activate under any environment, makes it simple and easy, and it is very significant for code developers. This might be relevant when we need to distribute, publish, deploy and handle ready-made applications' codes.

There are additional advantages to the stated Java language, appearing in a wide range of academic papers or Internet professional forums. Such article example: High-performance Java [8]; Internet sites: [9, 10]; and Java Professional Internet Forums, as follows in Table:

Table

Java internet forums

Forum Name	Java Forums
Java programming forums	http://www.javaprogrammingforums.com/
stackoverflow	http://stackoverflow.com/questions/tagged/java
Oracle Java Forum	https://community.oracle.com/community/java
Java Dream in Code	http://www.dreamincode.net/forums/forum/32-java/

The difficulty of learning and using the language

Learning programming is not always simple. A lot of references show that; i.e.: [11, 12]. Despite the significant benefits that were count and primarily mentioned, it's only natural to understand that there is no "*perfect language*". It is not always easy to learn and apply correctly the Java language, its properties and attributes [13]. Here are some main difficulties about learning the Java language and implement progressive and advanced code.

a) First difficulty: cognitive ability of abstraction

The Java language (as other programming languages) is a way, a method and set of tools, which emphasize and enable the developers to abstract their ideas into real implemented source code. Abstraction [14] ability cannot be developed quickly. It requires time, and based on a long-term process. In order to develop this capability we are required to a preliminary knowledge and background, at least in even a basic level, mainly in the field of logic; logic theory and deductive/inductive theory of groups. A lot of examples, which at first glance are not even related to the subject of "*software*" or "*computer science*", have to be demonstrated to the Java learners. As an example, dealing with groups of animals: mammals-reptiles-predators and the like; finding common and distinguishing between groups of objects; which allegedly had nothing between them. For example: what are the common things for a "*jewel*" and a "*cat*"? The differences are usually obvious. At first glance it seems that there are no common attributes to these objects. But there are a lot, of course: weight, value, mass, color, size, etc. At the same time, it is important to specify how the language itself can create relations or relevancy of structural or abstract identity, as well as how the differentiation between groups of objects and attributes can be specified easily.

b) Second difficulty: the way of teaching Java

This difficulty stems essentially from waive the significant differences between programming and critical paradigms. All Java study must be illustrated and demonstrated by examples and comparisons: not just as theory on one hand, but when an implemental section is presented, on the other hand – the theory should be presented as well. This is like a "*packa ge deal*" between theory and practice. We need to take a wide broader view and also get into details. Learn Java isn't just to learn the language, the various command, ways to

write methods etc. we need to face with the principles of the basic difference, including: understanding the paradigm basics, and make sure we recognize and use during our development procedures. This principle is not unique just to Java, but also to other/all computer languages and the ways we learn and teach them. It is much relevant to abstract OOP languages, as learners need to have wide understanding; wide set of tools – to understand the OOP concept and the ways Java implements them.

We mostly easy identify that people with previous acquaintance with the programming paradigms excluding the OOP paradigm; may have difficulties to internalize and assimilate the OOP paradigm and the differences. This is a significant way of thinking. The thinking change is not easily being done. We have to face, first, with the concept, the OOP model and just then getting into the practical dimensions of the language. Many exercises are required – while learning, emphasis to different modes and analysis solution, under the criteria of the paradigm. "*Do's and Don'ts*" rules have to be specified. Those are quite a few didactic challenges facing with the courses matter. All these require a lot of energy and efforts while teaching the paradigm and the Java language itself. One of the main didactic question is: "*Is it better to teach procedural programming first, over directly applying Object-Oriented Programming?*". In other words, "*suppose someone learned procedural programming previous to learn object-oriented programming directly: does it demonstrate understanding and better internalization of the subject matter?*".

The answer really is not unequivocal. Fall into that quite a few articles. Between those who one way or another, there is also a neutral approach [15], showing the difficulty of learning programming paradigms in general does not depend at all. However, results of the study population, show a strong linkage in this forward and procedural programming studied, absorbed and implemented the characteristics of OOP better, and actually implemented intelligent coding and more appropriate language. This reference section presents a broad section "*Recommendations*".

c) Third difficulty: knowledge barriers

There are quite a few applications and subject-areas requiring focused and in-depth information on specific content. Not infrequently, they are based on a mathematical background which might be missing (Surakka [16]). A series of articles and references were written on this subject, clearly shows the correlation between applications based on a mathematical model/s and its/their effectiveness. Quite a few applications, commercial and scientific, complex technology environments, require such knowledge and background. We can assume that if those are missing in "*real time*" (during the development process), this may affect the professional level of the entire application.

d) Fourth difficulty: possible solution vs. optimal solution

A further indication that came from research to realization: a problem we need to analyze and implement as a code, may become very difficult. Often, programmers and developers prefer the shortest way, when they know that is not always immediate effective or optimal solution. The effective solution may take time, requires more resources and may cost much more money. Although, this is understandable, especially in high-tech companies. This is when the time factor to provide the solution is usually critical. The distance between the "*possible proposed solution*", which may provide satisfactory and acceptable solution, to the "*optimal-effective solution*" is usually significant in its scope. In addition; it is well known and also can be assumption, that we do not need (as hypothesis) the effective solution, as it does not always worth the investment or not always even required. It is important to note that research not required viewing optimal solution. In spite this the solution brings to the fore the capabilities and characteristics of Java language, which programmers are supposed to bring and express in their development, as it reflects in their proposed solution.

Recommendations

The tips below are practical advices and focus on teaching teams and courses of:

(a) Object-oriented programming

(b) The Java language.

In parallel, there are more practical-oriented recommendations which are not included here; for actual Java-language developers.

1. Courses order and process

It is essential, and even MUST, to edit an introductory course – which deals with the paradigm of object-oriented programming and other programming paradigms. This is one of the research findings. Evidences & similar results and conclusions can be found at [17] reference, as an example. These show that direct study of

the language without a background in programming paradigms to get results and learning products, would be less effective significantly. On the other hand, I must specify that there are different approaches, which show that preliminary Java study is better to understand later the OOP paradigm [18].

In addition, it is important to emphasis the understanding of relationships between groups (foreignness, inclusion & exclusion relations, unification etc.), which are the basis for the understanding of learning and ways of implementation, and to refresh these underlying issues in order to achieve effective teaching/learning results.

Most of the research participants (78%) recommend hierarchical staged study, as illustrated in Fig. 1:

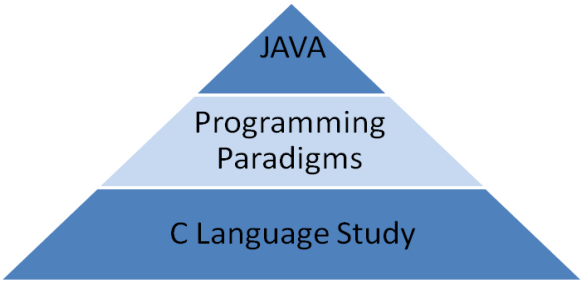

Fig.1. Courses Order.

Thus, the performance level who study in this order are proved to be better by the research results. It makes sense, as when students have the wide and deep understanding, and abilities to compare concepts, paradigms and different tools – they may achieve better practical results in the implementation process.

2. Didactics aspects

(a) Effective learning is very important and affects the ability of internalizing the terms, the complexities and the entire relationships between the Java elements. Therefore, early lessons preparation is required. Although this is correct to say for any other subject, but it is critical in teaching Java, specially, due to the difficulties identified and mentioned above.

(b) It is important to be aware to the processes of abstraction, conceptually and internalization of the courses; as a class or course on one hand, but also in individuals perspective view. We can use the learning cycle model of Kolb [19, 20], as an educational tool, which may simplify the learning processes and experiences. Kolb model (to see Fig. 2) can be implemented and used, of course, in any professional studying matter, and not just when studying OOP or Java programming.

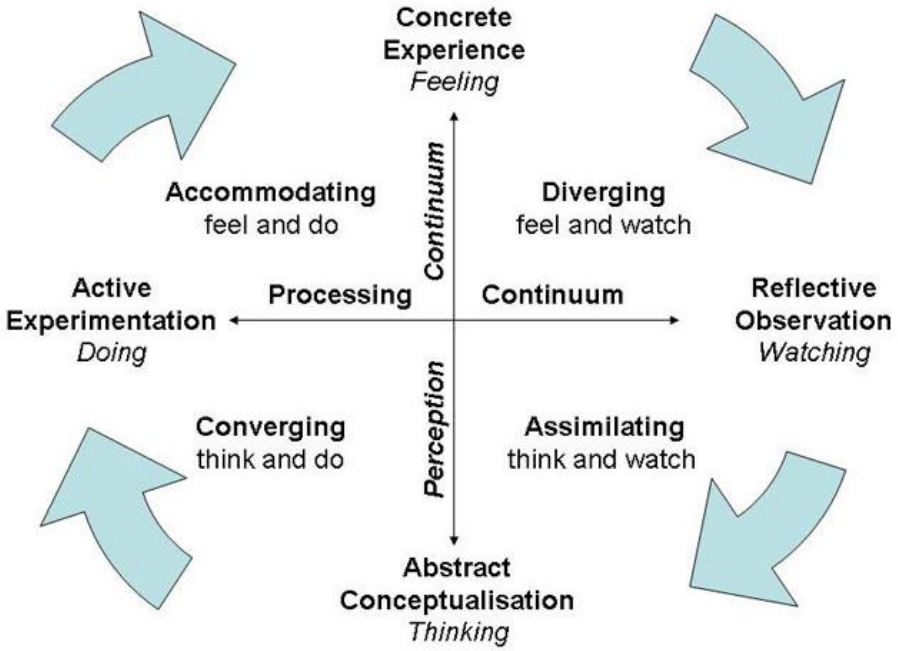

Fig.2. KOLB Experiential Learning Cycle [21].

The following paragraphs are general and not necessarily unique just to Java study. However, as the didactics takes significant central place in the teaching process; we need to mention and refresh these principles, and to stick to these principles, as critical success factors to achieve good results.

(c) Depending on the conclusion suggested from the previous paragraph, it is offered to teach the introductory course in relatively small groups (15-20 students). It is recommended for the Java course itself, as well, if possible.

(d) As part of the structuring didactic process, and more specifically, according to the constructivism [15] in its educational meaning, it would be important to expose the students into desktop application implementations, first. Just after checking and controlling this process, we would be able to lead the students and expose them to client-server implementations, working and merging databases. This must be done after proving the students' abilities to cope with larger and wider conceptual and scoped systems.

(e) Working-teams: it is extremely important to build teams of three students each (optimum 3-5). The team is committed to work, of course, as a team, which has responsible student/team leader. Responsibility will be replaced in the rotation between teammates: If any project/assignment submission, if it is a large project - when completed milestone along the development process. It is also advisable from time to time to replace and change the structure of the team. Similar to what happens in organizations and companies. That is important to achieve cross-fertilization much effectively. It is important to work in teams will also include feedback mechanisms. Each team will visit, examine, and analyze the work of other staff, within precise guidelines of: what is tested, what is right, what should embrace and what is not, what the effectiveness of the code is; the teleological terms, in the first stage, rather than operational efficiency, which is acquired over time. As teachers/lecturers - we must guide the students in the ongoing management of the learning process is mentioned. Through the whole process also acquired additional skills of the students as: task management / project, how to build a feedback process in a team, handling and dealing with different approaches, and more.

Exercises & practice: this is the main issue that must accompany us during teaching the courses. We need to drill down more and more. The exercises must be accompanied by an evaluation process, and by performance analysis of features, characteristics and implementation way, by using the object-oriented programming tools. The assessment can be done by dividing the work teams where each team assesses the work of other staff. Thus, the results, comments, suggestions and conclusions appear occasionally before all course participants, and/or on the course website. Further reading about the teaching process is given on [22] Reference. As a model for the feedback process/procedure I would specify the following (to see Fig. 3).

Fig.3. Evaluation Model.

(f) We need to present examples of problems and their solutions, as well as USING OOP paradigms and such those that are NOT USING OOP programming, while meeting on charting differences. This is quite important as we develop the understanding of the proper ways to use the OOP concepts. These can sharp our "*Good*" or "*Not-Good*" to do.

3. Lecturers and Teachers training

More broadly, may require specific and targeted training of teachers and lecturers in the studied areas and courses. Teaching staff should know and be familiar with the OOP and know exactly the differences between the paradigms [17]. It is important to teach "*how to be a programmer*" rather than teaching "*programming language*". This is quite different mission. As part of that mission, it is recommended to read the article discusses the topic "*What are the qualifications, skills and topics of study required for software developers?*" [16]. We can identify, for example, that it include collection of abilities, from different domains. I will mention two domains, as an example:

(A) Cognitive abilities, as: the ability to understand the problem and define analytical skills, ability to catch the possible solutions and their formulation, implementation capability and more.
(B) Interpersonal skills: teamwork, good interpersonal skills, ability to share and more.
More can be found at report [23].

Conclusions

Java experienced programmers have a special fondness to the language, to the way that application development can be processed and managed, and to the wide scope of the language behavior under high professional standards. However, whoever they face enough basics principles of object-oriented programming, can engage in developing applications, but much less professional in the total result. All these are controlled, of course, under generally accepted well-known standards of performance evaluation.

There is no doubt that Java language will continue to maintain the top places in Programming Languages rank. This is, at least, in the fields of desktop applications and for enterprise business environments. In addition, Java will continue to be a central significant key language platform for cell phones application. Therefore, we, as lecturers and teachers have to improve our teaching methods especially in specific courses: object-oriented programming and Java. In order to get better utilization of the language, we are required to establish organized and well-arranged courses in these fields.

References:

1. *TIOBE Index for February 2016*. http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html
2. *How many Java developers are there in the world?* July 2012. https://plumbr.eu/blog/java/how-many-java-developers-in-the-world
3. VILNER, T., ZUR, E., GAL-EZER, J. Fundamental concepts of CS1: procedural vs. object oriented paradigm - a case study. In: *SIGCSE Bulletin*, Volume 39, Issue 3, September 2007, p.171-175.
4. WHITE, G. SIVITANIDES, M. Cognitive Differences Between Procedural Programming and Object Oriented Programming. In: *Information Technology and Management*, Volume 6, Issue 4, October 2005, p.333-350.
5. PENDERGAST, M. Performance, overhead, and packetization characteristics of Java application level protocols. In: *ACM SIGITE Research in IT*, Volume 8, Issue 1, January 2011, p.4-15, NY, USA.
6. TILEVICH, E., SMARAGDAKIS, Y. J-Orchestra: Enhancing Java programs with distribution capabilities. In: *ACM Transactions on Software Engineering and Methodology* (TOSEM), Volume 19, Issue 1, 2009, NY, USA.
7. MARTIN, J., ODELL, J. *Object Oriented Analysis and Design*. Englewood Cliffs: Prentice-Hall, 1992. 515 p.
8. PANCAKE, C., LENGAUER, C. High-performance Java. In: *Communications of the ACM*, Volume 44, no.10, October 2001, p.98-101.
9. *Summary: java language advantages*. http://www.streetdirectory.com/travel_guide/114362/programming/most_significant_advantages_of_java_language.html
10. *The Java Language Environment: Contents.* http://www.oracle.com/technetwork/java/langenv-140151.html
11. GAL-EZER, J., VILNER, T. & ZUR, E. Has the paradigm shift in CS1 a harmful effect on data structures course: a case study. In: *SIGCSE bulletin*, Volume 41, Issue 1, March 2009, p.126-130.
12. KOLLING, M. The problem of teaching Object-Oriented-Programming. Part 1: Languages. In: *Journal of Object-Oriented-Programming*, Volume 11, Issue 8, 1999, p.8-15.
13. MANNILA, L., PELTOMÄKI, M., SALAKOSKI, T. What about a simple language? Analyzing the difficulties in learning to program. In: *Computer Science Education*, Volume 16, Issue 3, September 2006, p.211-227.
14. GUERRAOUI, R., FAYAD, M.E. Thinking objectively: object-oriented abstractions for distributed programming. In: *Communications of the ACM*, Volume 42, Issue 8, August 1999, p.125-127.
15. *Structuring - Constructivism - Learning*. https://en.wikipedia.org/wiki/Constructivism_(philosophy_of_education)
16. SURAKKA, S. What subjects and skills are important for software developers? In: *Communications of the ACM*, Volume 50, Issue 1, January 2007, p.73-78.
17. GOVENDER, I. From Procedural to Object-Oriented Programming (OOP) – An exploratory study of teachers' performance. In: *School of IS & T, University of KwaZulu-Natal, South Africa, Research Article - SACJ*, no.46, December 2010, p.14-23.
18. KOSTER, A. Teaching Object-Oriented Programming: A comparison of Java and Objective-C. In*: International Journal of Management & Information Systems*, Volume 19, Number 1, 2015.
19. KOLB, D.A. *Experiential Learning: Experience as the Source of Learning and Development*. Prentice-Hall, Englewood cliffs, New-Jersey, 1983.

20. KONAK, A., CLARK, T.K., NASEREDDIN, M. Using Kolb's Experiential Learning Cycle to improve student learning in virtual computer laboratories. In: *Computers & Education*, Volume 72, Match 2014, p.11-22.
21. *Kolb Learning Styles*. http://www.businessballs.com/kolblearningstyles.htm
22. YOULIANG, L., CUI'E, X., JIANCHEN, Z. Application of Project-based Teaching Method in Java Language. In: *Teaching Education Research Frontier*, 2013, p.118-121. Volume 3, Issue 3, p.118-121.
23. ACM, Associate-Degree Programs In Computing And Engineering Technology, 2014, http://www.acm.org/education/se2014.pdf

Notă: *Lucrarea a fost efectuată în cadrul tezei de doctorat „Modele, tehnici şi produse program inteligente de asigurare a organizaţiei cu cunoştinţe".*

Prezentat la 15.09.2016