Web Application to Support the Learning of Programming Through the Graphic Visualization of Programs

https://doi.org/10.3991/ijet.v15i06.12157

Carlos R. Jaimez-González(✉), Miguel Castillo-Cortes
Universidad Autónoma Metropolitana, Ciudad de México, México
cjaimez@correo.cua.uam.mx

Abstract—This paper presents a web application to support the learning of programming at the undergraduate level, which allows students to graphically visualize through animations the execution of programs written in the Java programming language. The web application supports the understanding of programs and the basic concepts of programming, such as declaration of variables, assignment of values to variables, use of control structures, and calls to functions with parameters. The development of the web application, its architecture and the three systems that compose it are presented: data collection, processing and representation. The operation of the web application is shown through three programs, for which their execution is visualized graphically.

Keywords—Educational technology, education computing, software understanding, software visualization

1 Introduction

Understanding a program is having the ability to understand how the source code performs the task for which it was created. The understanding of programs is an area of software engineering focused on developing techniques and tools, based on cognitive and engineering processes, which are used for tasks of reuse, inspection, maintenance, migration, software extension, among other applications; but they can also be used in areas such as education or training. The aim of this area is to achieve a better understanding of computer applications as stated in [1].

One of the main challenges in the area of program understanding is to be able to determine the relationship between the domain of the problem and the domain of the program [2]. The first concept is the result of the execution of the program, for example, printing on the screen the result of an operation; the second concept comprise all the components of the program, which produce the behavior of the system, for example, the methods used to solve an operation [2]. In order to be able to represent these relationships, between domain of the problem and domain of the program, the use of software visualization is used, which is an area of software engineering whose objective is

to generate, from certain aspects of the system, one or more multimedia representations, with the purpose of facilitating the understanding of it [3] [4].

This paper presents a web application to support the learning of programming through the graphic visualization of programs written in the Java programming language. This application complements the teaching-learning process of programming, because it encourages the interconnection of concepts; this way, the knowledge of the student is reinforced through visual representations of the execution of programs that are shown in the web application.

The rest of the paper is organized as follows. Section 2 presents the theoretical framework, where cognitive models and software visualization are addressed. Section 3 describes a series of tools relevant to the implemented application and a comparative analysis is carried out. In section 4 the development of the web application, its architecture and the three systems that compose it are explained. Section 5 shows the operation of the web application, along with some examples of programs for which their execution is graphically displayed. Finally, conclusions and future work are presented in section 6.

2 Theoretical Framework

In the development of systems for software understanding it is important to rely on other fields of study, such as cognitive psychology, psychopedagogy, program visualization, among others. The following sections deal with the topics of cognitive models and software visualization, which are very relevant to the web application presented in this paper.

2.1 Cognitive Models

The integration of Information and Communication Technologies (ICT) in the teaching-learning processes have generated several changes. The role of teachers does not focus on transmitting information anymore, but on encouraging the student's personal search for knowledge [5]. Traditionally, teachers used to take the main role by exposing concepts, while students used to have a passive attitude, playing the role of receiver of those concepts.

In contrast to the traditional model, the constructivist model perceives teaching as a critical activity and not only as the transmission of knowledge, it encourages the innovation of methods that allow students to develop their own cognitive structure. In this approach, the most important thing is learning, for which the teacher takes a role of facilitator, moderator and mediator between the student and the knowledge [6].

Cognitive development is a field of research of cognitive psychology that studies the mental processes involved in the creation of knowledge. A software developer understands a program when it has the ability to understand how the source code performs the task for which it was created, for which it builds a mental model with the information extracted from the program, then it looks for associating the information to its previous knowledge, finally, it will understand the functionality of the system.

Learning is to combine the knowledge already acquired with new concepts through learning processes; for example, the *Bottom Up* model starts from specific concepts obtained from reading the program to generate general abstractions; or the *Top Down* model, which assumes that one already has a notion of the functionality of the program to propose a hypothesis, which will later be verified when reviewing the source code [7] [8].

Cognitive models are composed of three basic elements: knowledge, the process of assimilation and a mental model. The first element proposes two distinctions: internal knowledge, which refers to the knowledge that is already owned; and external knowledge, which is integrated by the new concepts that the system will provide. The second element is formed by learning strategies, such as *Bottom Up* or *Top Down*. Finally, the third element is a representation of the system, which is given in order to explain its behavior [7] [8].

2.2 Software Visualization

The visualization of programs is an area of software engineering related to the visual representation of information, whose objective is to generate, from certain aspects of the system, one or more views that will facilitate the understanding of it. A view is a way of representing the elements of a system and the relationships between these elements [7].

The multimedia representations oriented to the understanding of software must provide three basic views: the output of the system, the elements of the program that generate that output, and the relationship between the previous two. These allow to elaborate representations that associate the domain of the problem with the domain of the program; that makes it easier to relate the knowledge that is owned and the concepts used by the system [4] [9]. These representations are not easy to build, since they are strongly based on cognitive and engineering factors, so it becomes important to rely on several areas of knowledge such as graphic design, cognitive psychology, psychopedagogy, computing and other disciplines related to the creation of multimedia effects and learning [3].

The aspects of software needed to elaborate a representation are obtained using information extraction techniques, that are classified by the type of information that they extract, which can be static or dynamic [2]. The techniques of extracting static information are used to verify that there are no syntactic, lexicographic or even semantic errors, making use of syntactic analysis tools to examine the source code [2]. The techniques for extracting dynamic information are responsible for obtaining information on the behavior of the program and its components (methods, variables, method invocations, etc.) at the time of execution. One of the techniques for obtaining this data is the use of code instrumentation, which consists of placing instructions in the source code in order to monitor their behavior during execution [2].

Software visualization includes two fundamental areas: the visualization of programs and the visualization of algorithms, depending on what it is required to understand. The visualization of programs allows to have views of the source code and its structure. A static representation is useful to verify the validity of the source code,

usually code editors are used to give a better presentation and readability of the source code through indentation, differences of colors between reserved words and other identifiers. The dynamic representation shows program information at runtime, for example, highlighting the instructions of the code when they are being executed [10]. The visualization of algorithms focuses on representing the semantics of the program, this means that it generates views of the behavior of the program in execution, without showing the instructions and operations that make this behavior possible [10]. These concepts influence the development of software understanding tools.

3 Related Work

This section carries out a review of the state of the art of tools for the understanding and visualization of programs. A comparative analysis of the tools is also provided at the end of the section.

BlueJ [11] is a development environment designed to learn object-oriented programming with the Java programming language. It uses visualization and interaction techniques to improve the user experience. It is an environment in constant development, which was created by the University of Kent in Canterbury, United Kingdom, and the University of La Trobe, in Melbourne. BlueJ colors the background of each block of code to visually identify the sections of the program, it helps in the detection of misplaced keys and shows details of the instantiated objects.

Jeliot 3 [12] is a software visualization tool that shows how a program is interpreted in the Java language; the method calls, variables and operations are visualized through a multimedia representation that allows to follow step by step the process of execution of the program. The Jeliot3 interface is composed of three panels: the upper left panel is used to enter the code that will be executed; the lower left panel shows the output of the program execution; while the panel on the right displays the representation of the program in execution, where there are four areas, which identify the initialization of variables, the representation of operations and arrays.

jGrasp [13] is a development environment that automatically generates animated visualizations to improve software understanding. jGrasp is able to generate diagrams of control structures for the Java, C, C++ languages, among others; it also creates UML diagrams. The visualizer represents data structures such as stacks, queues, linked lists, etc. The interface of jGrasp includes an area where the source code is entered; when executing the source code, it shows in a lower left panel the instances of the elements created in the program, such as variables and arrays. In an independent window it can be visualized the representation of these instances graphically.

Scratch [14] is an application to program interactive stories, games and animations, just by defining rules using blocks of instructions. The projects developed in this application can be shared within a project catalog, in this way the user has access to the applications created by third parties. The Scratch platform interface shows a window with three sections, on the left there is an area where the program animation will be displayed and in the central part there is a set of blocks of instructions that can be used by the user; instructions such as advance, rotate, change direction, change value of a

variable, among others. In the third section there is an area where the user will drag the blocks of instruction that he wants to use for his animation and he will be able to accommodate them like a puzzle. In order to execute the animation, the blocks that were joined have to be pressed.

There are other tools for the understanding and visualization of software that were not analyzed in depth, which was because these tools are of less relevance to those previously reviewed. Some of these tools are the following: an algorithmic animation platform [15], Understand [16], CodeSurfer [17], Imagix 4D [18], ShriMP [19], Alma [20], a debugging tool to learn algorithms [21], SeeSoft [22], Extravis [23], some authoring tools reported in [24], among others.

Table 1 shows a comparison of functionalities of the tools discussed previously and the web application developed. The tools shown in Table 1 are the following: H1) BlueJ, H2) Jeliot3, H3) jGrasp, H4) Scratch, H5) Developed web application. The tick indicates that the tool has the functionality, while the cross indicates that the tool does not have it.

Table 1. Functionalities of the analyzed tools and the one developed

Functionality	**H1**	**H2**	**H3**	**H4**	**H5**
1) Code with animations	×	✓	✓	✓	✓
2) Web application	×	×	×	✓	✓
3) Representation of classes	×	✓	✓	×	×
4) Illuminated code blocks	✓	×	×	×	✓
5) Editable code	✓	✓	✓	×	✓
6) Representation of methods	×	×	×	×	✓
7) Representation of control structures	×	×	×	✓	✓
8) Parameter representation	×	✓	✓	×	✓

The first functionality indicates that the representation of the execution of the code is carried out with animations; the second one indicates that the tool analyzed is a web application; the third functionality means that the tool shows a graphic representation of classes; the fourth functionality refers to the syntax highlighting and shading of code blocks; the fifth functionality shows that the tool allows the editing of source code; the sixth functionality indicates that the tool graphically represents the methods; the seventh functionality indicates that the control structures are represented visually in the tool; and finally, the eighth functionality means that the tool performs the visual representation of parameters.

It is difficult to evaluate the impact of this type of tools in the learning process. In the literature there are no conclusive empirical results that support which of these tools best meets their objectives [25]. It should be noted that visual representations are successfully used in different disciplines, such as physics, chemistry, biology, mechanics, among others, to illustrate complex concepts and processes, since they use metaphors that help in understanding [26]. Computer Science is a discipline that also makes use of visual representations for different processes, which is why the web application that was developed and presented in this paper makes use of them.

4 Development of the Web Application

This section presents the development of the web application, describes its architecture, and details the operation of each of the systems and modules that comprise it.

The architecture of the web application consists of three systems: data collection, data processing and data representation. The data collection system is responsible for taking the information provided by the user as input to the application, and validates the lexical part of the source code entered by the user. The data processing system takes the collected data, analyzes it and processes it for later use; this system provides an interface that is combined with the graphical interface that allows the user to interact with the animation generated by the application. The data representation system shows the user the result of processing the data that was entered as input to the application; it is the system responsible for generating the visual representation of the execution of the source code entered.

4.1 Data Collection System

The data collection system is composed of various modules from which the input data is obtained, it provides mechanisms for communication and interaction with the user and defines most of the graphic interface of the application. Figure 1 shows the distribution of the application's interface, where four areas can be seen: the text editor that covers the entire screen and has a transparent background; the representation area, just below the text editor, where the animation will take place and has a black background; on the right side is the control panel; and finally, in the lower part of the screen there is a panel containing the syntax tree and the call tree. The modules that compose the data collection system are explained below.

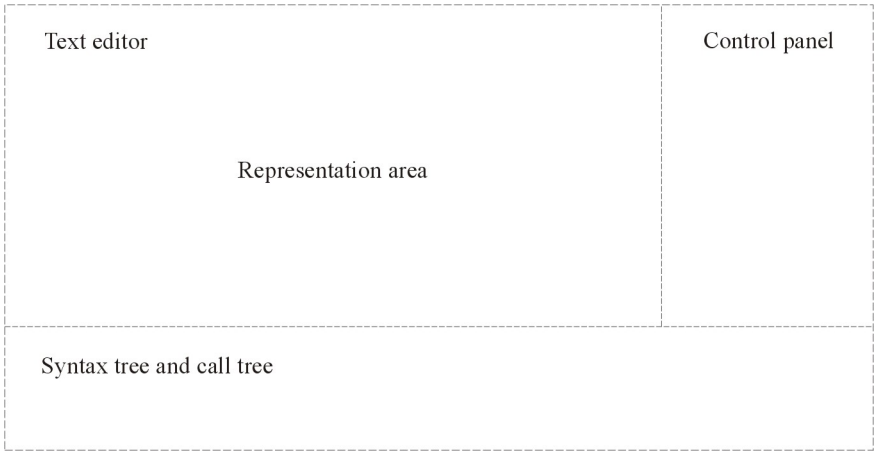

Fig. 1. Distribution of the web application's interface

Text editor module. It is responsible for obtaining the input data for the web application, it is an essential communication mechanism with the user, since it is where the text corresponding to the source code in the Java programming language is entered. The text editor displays the text in a format that facilitates the reading of the source code, it uses different colors for the reserved words of the Java language. The text editor highlights the source code at the moment in which the animation is carried out by the student, in order to indicate the code instruction that is being executed or the instruction that will be executed.

The text editor was created as a table with rows and columns, where each character that is entered is placed internally within a specific cell to be able to locate it by means of its row and column coordinates. For the implementation, it was used the Hypertext Markup Language (HTML) to create the text area, Cascading Style Sheets (CSS) for the presentation and the JavaScript language for the interactivity.

Controls module. It consists of several controls of the web application, which interact with the data collection system and the data processing system. The user interacts with the application through the controls, which are organized in a menu, shown in the right panel of Figure 1. The menu handles two colors for the controls: green for those that can be used at that moment, and red for those that are disabled because there is an animation process that requires them to be temporarily blocked.

Some of the controls found in this module are the following: *speed* control determines how quickly the animation is executed; *stop* control allows to stop the animation at a particular step; the *theme* control changes the background color and text of the source code; the *opacity* control modifies the degree of transparency of the text editor; the *full screen* control defines whether the text editor will occupy the entire screen or only the default space; the *autocomplete* control activates the mechanism that displays a list of words as it is written in the text editor; the *current line* control activates the mechanism to highlight the text that represents at execution time the instruction that has been executed; the *next line* control activates the mechanism to highlight the text that represents at runtime the instruction that will be executed in the next step; the *examples* control allows to load in the text editor several examples of source code in the Java programming language; finally, the *panel* control allows to show or hide the panel that is in the lower part of Figure 1, where the representation of the associated syntactic tree and the call tree are.

Lexical analyzer module. It is used to verify the source code that the user entered in the text editor and check that the words contained are typical of the Java programming language. The lexical analyzer generates a list of objects called tokens from the words analyzed, where each object contains the word analyzed, a symbol that identifies the word, the line where the word was found, the column of the first character and the column of the last character of the word analyzed. The output of the lexical analyzer is the list of tokens or lexical components, which serve as input to the parser described below.

4.2 Data Processing System

The data processing system consists of several modules that are the backbone of the operation of the web application; it is responsible for dealing with the input data, uses the mechanisms designed for data representation, as well as allows the user to interact directly with the animation. The modules that compose this system are explained below.

Syntactic analyzer module. Its function is to verify that the syntactic structure of the source code entered in the text editor is correct; in this case for the Java programming language. The operation of the syntactic analyzer starts from the list of tokens generated by the lexical analyzer, which is a module of the data collection system, and ends with the creation of a syntactic tree that specifies the order that will be followed for the data visualization process, which represent the execution of the input data for the application.

Program execution script module. It is where the syntactic tree nodes are stored, which represent the instructions of the valid source code. This module has an area for storage and provides the functions for insertion, deletion and obtaining instructions.

The storage area of this module is represented by a tree of objects called the execution tree, which works with three different levels: the first level is the root node, which contains the instructions to be executed; the second level is formed by the objects that represent the declaration of a method and each element of this level contains its set of instructions, that is, it contains the nodes of the syntactic tree belonging to the node that represents the definition of a function; the third level contains objects that represent the control structures, such as *if*, *else*, *for*, *while*, *do while*, and each element of this level contains its set of instructions from the syntax tree.

The execution tree is modified as the execution animation of the program flows. Figure 2 shows a representation of the execution tree.

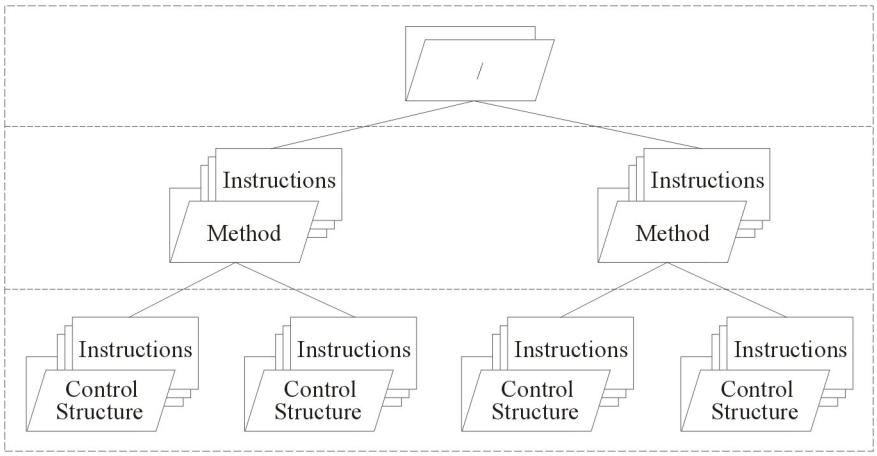

Fig. 2. Representation of the execution tree

The function for insertion of this module is composed of two mechanisms: the first mechanism inserts an object at the second level, together with its set of nodes from the syntactic tree, which represents the instruction of method statement; the second mechanism inserts an object at the third level, together with its set of nodes from the syntactic tree, which represents the instruction of a control structure.

The function for obtaining instructions of this module returns an object that contains the instruction that will be executed; it returns the corresponding instruction in increasing order from the last node of level 2 and its last associated sub-node of level 3. If there is no associated level 3 sub-node, it returns its corresponding instruction in increasing order.

The function for elimination of this module is responsible for deleting nodes from the execution tree. A node is eliminated if the path of its instruction list (nodes of the syntactic tree) comes to an end. A second level node that represents the declaration of a method can be eliminated if in its set of instructions it is possible to execute a return instruction. A third level node that represents the control structures can be eliminated if the evaluation of its condition results in a false result.

Execution steps generator module. It is responsible for adding or removing instructions to the program execution mechanism, as well as calling the appropriate methods for data representation. This module bases its operation on the basis of the syntactic tree, the interaction with the user through the controls module and the result of the execution of instructions of the input data. Two of the most representative steps that rule the operation of this module are: the preparation of the animation, with which you get the first instruction to execute, taken from the syntactic tree; and the execution of the animation that is responsible for simulating the execution of an instruction of the source code entered by the user, which can be carried out step by step or automatically from start to end.

4.3 Data Representation System

The data representation system is responsible for displaying the animation that represents the execution of a program in the Java programming language. The implementation of the modules that comprise it is described below.

Models module. Data representation uses different object models that represent an instruction. Each of these models extends from a main model. There are different types of attributes that represent the types of elements that are displayed in an animation, such as variable names, data types, method names, program flow control instructions, variable values, results of the evaluations of logical expressions, among others. Similarly, there are attributes of the models that allow to draw on the screen the visual representations of the elements, such as the different geometric shapes and text.

Visualization module. It is responsible for displaying elements on the screen, through the use of an application programming interface (API), called Three.js, with which it is possible to draw elements on the screen and apply different textures and colors. This visualization module creates all the basic elements to show on the screen the different geometric figures and text, which represent the execution of the source code that is entered by a user.

5 Operation of the Web Application

In this section the web application is shown in operation, through three examples of programs, for which its execution is graphically visualized: the first program is simple, it contains three declarations of integer variables, with their corresponding assignments; the second program calculates the factorial of a number, which shows how the stack of method calls is represented, as well as the control structures, with their corresponding local variables; finally, the third program performs an ordering through the bubble method, where the stack of method calls, control structures and variable manipulation are again visualized.

5.1 Simple program with declaration of variables

In this example a simple program is demonstrated, its source code is shown in Table 2. Some screenshots are presented with the graphic visualization of the program execution, which was carried out step by step in the web application.

The source code shown in Table 2 contains the declaration of the *MyClass* class in line 1; the declaration of the static method *main()* is inside the class, in line 2. Within the *main()* method, in lines 3 to 5, there are the declarations of three variables of type *int*: *a*, *b* and *c*, which have not been initialized. In line 6 it is the assignment of the variable *a = 3*; in line 7 the assignment of the variable *b = 7*; and finally, in line 8, the variable *c* is assigned the result of the sum of the values contained in variables *a* and *b*. Line 9 contains the key that closes the *main()* method, while line 10 contains the closing key of the *MyClass* class.

Table 2. Source code of the example program

Line number	Source code
1	public class MyClass {
2	public static void main() {
3	int a;
4	int b;
5	int c;
6	a = 3;
7	b = 7;
8	c = a + b;
9	}
10	}

Figure 3 shows a screenshot with the application's interface, where three main areas can be seen, described in a previous section. In the left panel is the text editor to enter the source code of the program, which by default occupies the full screen; however, it can be configured to change its size, degree of transparency and background color. The central panel is where the animation of the program execution will be displayed; this panel occupies the full width of the screen. Finally, the panel on the right side contains

the controls menu, where there are several controls that allow manipulating the animation of the program execution: *Prepare* to be able to start executing the animation; *Animate* to run the animation continuously without stopping; *Pause* to pause the animation that is currently running continuously; *Step by step* to execute the animation step by step (instruction by instruction); among others.

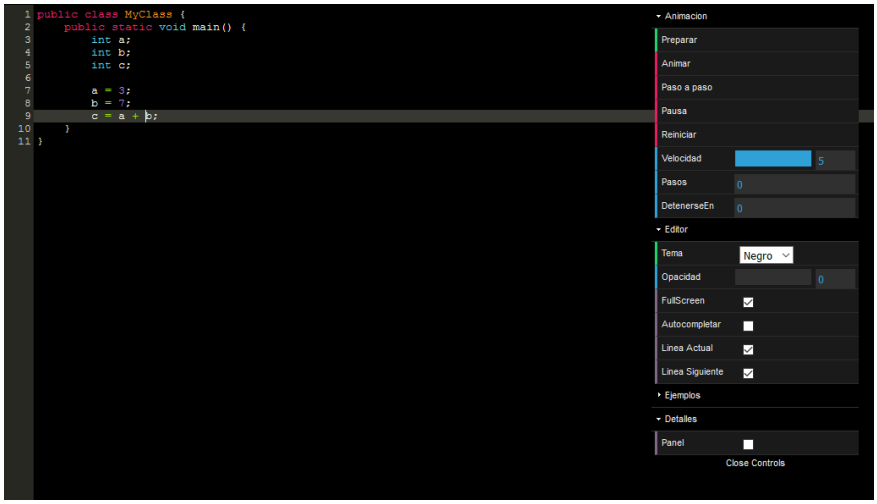

Fig. 3. Interface to start the application

Figure 4 shows a screenshot of the web application once the execution of the program has begun and after having pressed the *Step by step* control several times, up to the point of having created the three variables of the program: *a*, *b* and *c*, that is, until it has reached line 5 of the source code. In the screenshot of Figure 4 it can be seen that several blocks have already been created in the animation: the black and gray blocks on the right side represent the *MyClass* class and the *System* object of the Java language; the dark brown block at the center represents the call to the *main()* method of the class; while the light brown blocks, which are located above the dark brown block, represent the three variables that have been created in the program: *a*, *b* and *c*. The idea of the blocks is to try to simulate the space that the elements occupy in memory, as well as the dependencies that exist among them.

In order for the student to follow step by step the execution of the program in the code, the text editor indicates the last instruction or sentence executed with orange background color, while the next instruction that will be executed is indicated with pink background color. In the screenshot of Figure 4 the last sentence executed was the declaration of the variable *c*, which is found with an orange background color; while the next instruction to be executed is the assignment *a = 3*, which has a pink background and will be executed once the student presses the *step by step* control of the controls menu again.

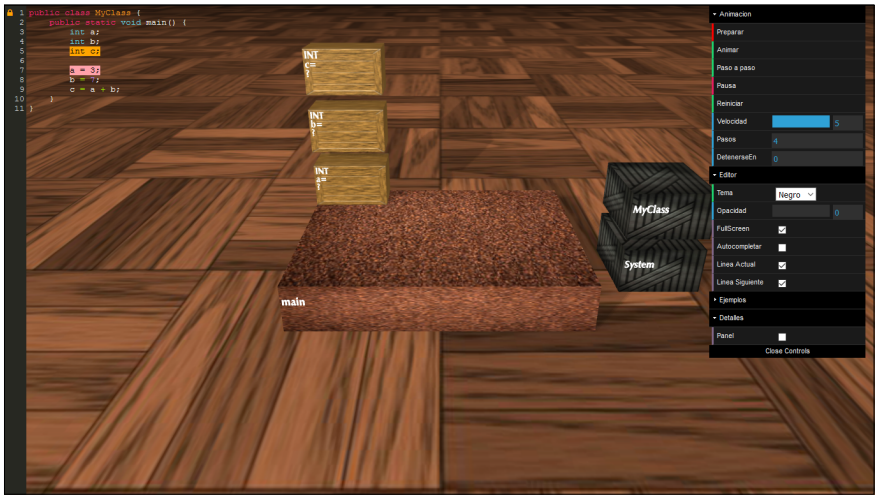

Fig. 4. Visualization of the animation after creating variables

In Figure 5 a screenshot of the same animation is shown, once values have been assigned to variables *a* and *b*. The last sentence that is running, with orange background color, is the assignment of the variable *c*, which is the sum of *a* + *b*. It is observed in the animation how the value *7* is traveling from the box of variable *b* to the area where the operation will be performed.

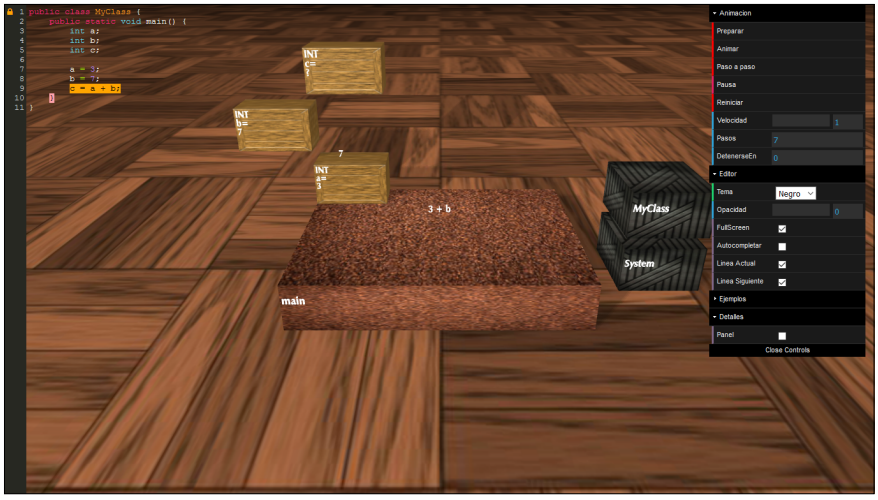

Fig. 5. Assignment of values to variables a and b

Figure 6 shows a screenshot with an animation that evaluates the operation *3* + *7* = *10*, and assigns it to the variable *c*, as indicated in the source code of the program. It should be noted that all the instructions that are executed from the source code are

displayed in the application in an animated way, so that the student can observe how the program is executed step by step, along with the graphic visualization.

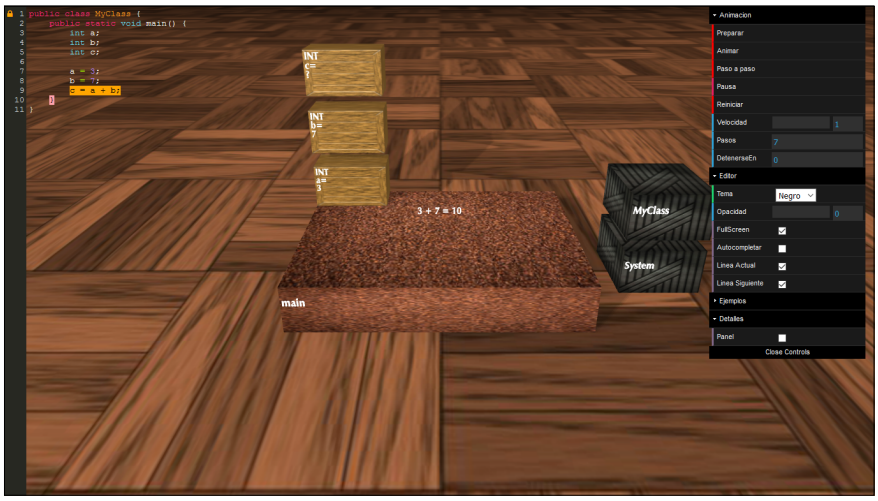

Fig. 6. Evaluation of the operation 3 + 7 = 10

Finally, Figure 7 shows that the value of *10*, which was the result of the operation of *a* + *b*, is travelling towards the box of variable *c*, where the question mark will disappear, indicating that the assignment to the variable has been carried out.

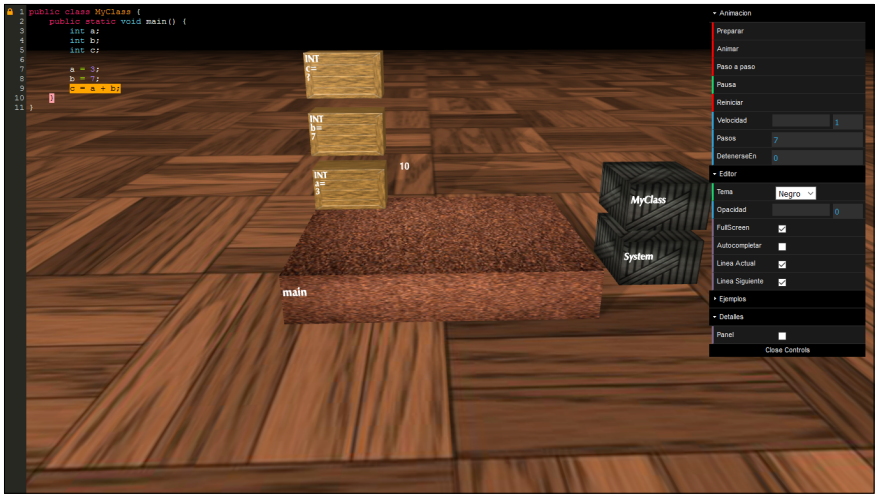

Fig. 7. Assignment of a value to the variable c

The program presented in this section is simple, but it shows the way in which the web application works for the graphic visualization of programs through animations.

This example allows the student to observe how methods, variables, operations and assignments are visually represented in the web application, and how the program is executed step by step. The following two subsections show examples of the execution of more elaborate programs.

5.2 Program to calculate the factorial of a number

Figure 8 shows a screenshot of the web application once it has started the execution of a program that is responsible for calculating the factorial of a number, and after having pressed the *step by step* control several times. The animation of the program is in a step where there are several blocks of instances of method calls (in dark brown color), as well as blocks that represent the conditional control structures *if* and *else* (in black and gray color), which are part of the execution of the program. Additionally, the blocks of variables (in light brown color) are observed, which are stacked on the block of the method or control structure to which they belong. It can also be seen the source code of the program that is running in the text editor, where the last instruction that was executed in the animation is marked, with orange background color, as well as the instruction that will be executed in the next step, with pink background color. This program is more complex than the previous one and shows the way in which the stack of method calls and control structures are visually represented in the application, as well as the creation of variables corresponding to each of them.

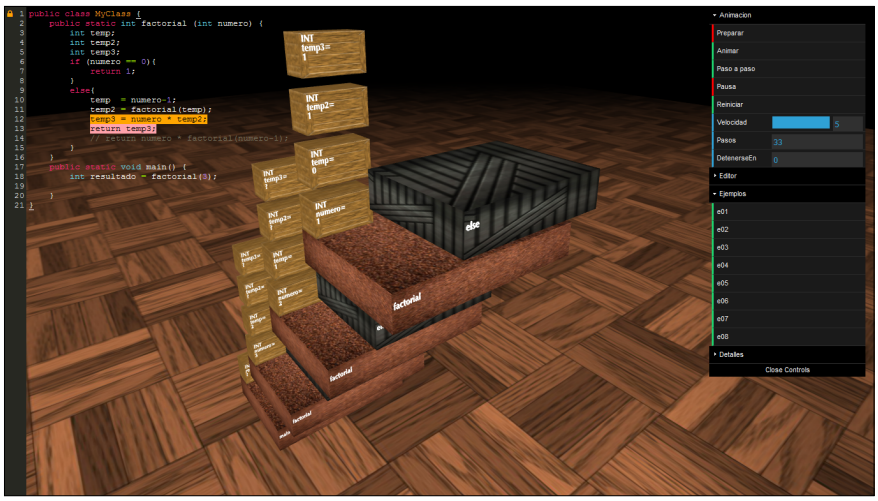

Fig. 8. Program that calculates the factorial of a number

5.3 Program to sort an array of elements

This subsection presents a program that sorts an array using the bubble method. Figure 9 shows a screenshot of the web application once the execution of the program has started, and after having pressed the *step by step* control several times. In the screenshot

of Figure 9 there are three blocks that correspond to the call to the *main()* method, in dark brown color; and to the two control structures *for*, in black and gray color, found inside the *main()* method. In addition to these blocks, the blocks of variables, in light brown color, are observed, which are stacked on the block of the method or control structure to which they belong. As well as in the previous examples, it can also be seen the source code of the program that is running in the text editor, where the last instruction executed in the animation is also marked, with orange background color, as well as the instruction that will be executed in the next step, with pink background color.

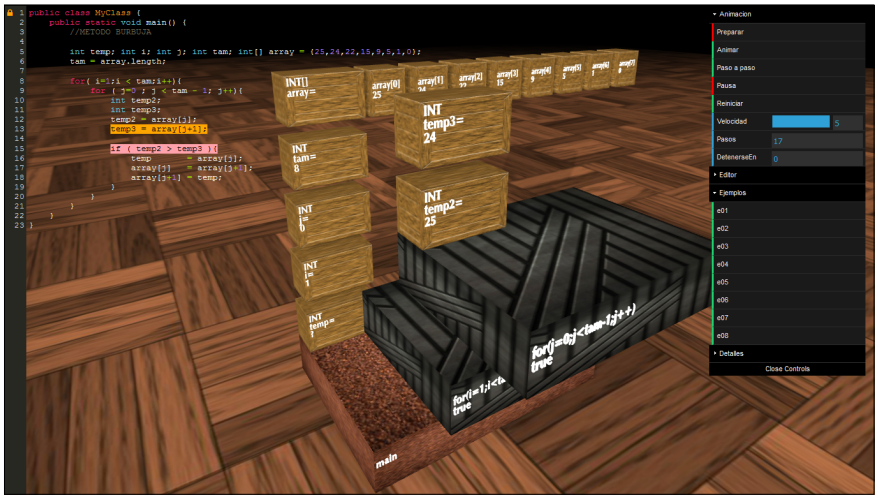

Fig. 9. Program that sorts an array of elements

It should be noted that for the source code that is entered into the text editor, the graphic visualization of the program will be carried out, through the animation that will represent methods, control structures and variables through the blocks that have been shown in these three examples.

6 Conclusions and Future Work

This paper presented a web application to support the learning of programming, which allows students to graphically visualize through animations the execution of programs written in the Java programming language. The web application supports the understanding of programs and understanding of the basic concepts of programming, such as declaration of variables, assignment of values to variables, use of control structures, and calls to methods with parameters.

It was carried out a comparative analysis of tools similar to the web application presented, and the most relevant features for the visualization of programs were highlighted. The implementation of the application was also presented, where its

architecture and the three systems that composed it were explained: data collection, data processing and data representation.

The operation of the web application was illustrated through three programs, for which its execution was graphically visualized. It is important to emphasize that the step by step execution of a program helps the student to understand it, because while it is displayed graphically in the web application, the instruction that is being executed in the source code is also shown.

Further work is required to develop an assessment instrument to formally apply it to students of introductory courses of programming, with the aim of assessing their opinion about the use of the web application that was presented in this paper. Once the instrument is applied to students, the application will be improved according to the feedback received. Additionally, work is being carried out in the web application to develop an interpreter to execute programs written in the C programming language, which will also be graphically visualized.

7 References

[1] Berón, M., Uzal, R., Henriques, P., Pereira, M. (2007). Understanding of programs by visual inspection and animation. IX Workshop of Researchers in Computer Science, Chubut, Argentina.

[2] Bernardis, H., Berón, M., Riesco, D., Henriques, P., Pereira, M. (2011). Dynamic information extraction in object-oriented programming (Java). XIII Workshop of Researchers in Computer Science, Rosario, Argentina, pp. 413-417.

[3] Berón M., Riesco, D., Montejano, G., Henriques, P., Pereira, M. (2010). Strategies to facilitate the understanding of programs. XII Workshop of Researchers in Computer Science, El Calafate, Santa Cruz, Argentina, pp. 445-449.

[4] Miranda, E., Berón, M., Montejano, G., Riesco, D., Henriques, P., Pereira, M. (2011). Software visualization: concepts, methods and techniques to facilitate the understanding of programs. XIII Workshop of Researchers in Computer Science, Argentina, pp. 519-523.

[5] Morales, M., Trujillo, J., Raso, F. (2015). Perceptions about the integration of ICT in the teaching-learning process of the university. Pixel-Bit Journal, 46, pp. 103-117.

[6] Martínez, A., Torres, L. (2013). Personal learning environments (PLE). From how to teach to how to learn. Edmetic. Journal of Media Education and ICT, 2 (1), pp. 39-57.

[7] Ruiz, J., Hernández, J., Gutiérrez, D., Harvey, G., Salinas, A. (2013). Comparison between software packages to support the teaching of Java programming. Technical Report.

[8] Berón, M., Henriques, P., Pereira, M., Uzal, R. (2006). Tools for understanding programs. VIII Workshop of Researchers in Computer Science, Morón, Buenos Aires, Argentina.

[9] Berón, M., Uzal, R., Henriques, P., Pereira, M. (2008). Code inspection to relate the domains of the problem and program for the understanding of programs. X Workshop of Researchers in Computer Science, Santa Rosa, La Pampa, Argentina, pp. 549-553.

[10] Moroni, N., Señas, P. (2002). SVED: algorithm visualization system. VIII Congreso Argentino de Ciencias de la Computación, Buenos Aires, Argentina, pp. 1175-1184.

[11] BlueJ. A Java development environment. Available: https://www.bluej.org/

[12] Jeliot. A program visualization application. Available: http://cs.joensuu.fi/jeliot/

[13] jGRASP. An integrated development environment with visualizations for improving software comprehensibility. Available: http://www.jgrasp.org/

[14] Scratch. Grupo Lifelong Kindergarten. MIT Media Lab. Available: http://scratch.mit.edu

[15] Esponda, M. (2008). An algorithmic animation platform for the web. International Journal of Emerging Technologies in Learning, 3(4), pp. 29-40.
[16] Scitools. Available: https://scitools.com/
[17] CodeSurfer. A code browser. GrammaTech. Available: https://www.grammatech.com/products/codesurfer
[18] Imagix. A source code analysis tool. Available: http://www.imagix.com/products/source-code-analysis.html
[19] Shrimp. Simple Hierarchical Multi-Perspective visualization tool. Available: http://www.thechiselgroup.com/shrimp/
[20] Alma. A system for program visualization and animation. Available: http://epl.di.uminho.pt/~gepl/ALMA/
[21] Khedr, A., Bahig, H. (2017). Debugging tool to learn algorithms: A case study minimal spanning tree. International Journal of Emerging Technologies in Learning, 12(4), pp. 90-100. https://doi.org/10.3991/ijet.v12i04.6442
[22] Eick, S., Steffen, J., Sumner, E. (1992). Seesoft-a tool for visualizing line oriented software statistics. IEEE Transactions on Software Engineering, 18 (11), pp. 957-968.
[23] ExTraVis. Execution Trace Analysis Through Massive Sequence and Circular Bundle Views. Available: http://swerl.tudelft.nl/bin/view/Main/ExTraVis
[24] Then, M., Wallenborn, B., Ianniello1 B., Vu, D., Fuchs, M., Hemmje1, M. (2016). Innovative authoring tools for online-courses with assignments. International Journal of Emerging Technologies in Learning, 11(2), pp. 29-40. http://dx.doi.org/10.3991/ijet.v11i02.5108
[25] Compañ-Rosique, P., Satorre-Cuerda, R., Llorens-Largo, F., Molina-Carmona, R. (2015). Teaching to program: a direct way to develop computational thinking. Journal of Distance Education, 46.
[26] Naps, T., Rossling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C., Korhonen, A., Malmi, L., McNally, M., Rodger, S., Velazquez-Iturbide, J. (2003). Exploring the role of visualization and engagement in Computer Science Education. Report of the Working Group on "Improving the Educational Impact of Algorithm Visualization", SIGCSE Bulletin, 35, pp. 131-152.

8 Authors

Carlos R. Jaimez-González is an Associate Professor at the Information Technologies Department at the Universidad Autónoma Metropolitana Campus Cuajimalpa, in Mexico City. He received his PhD degree in Computer Science from the University of Essex, United Kingdom in 2011. His research interests include technologies for supporting education, interoperability in distributed systems, XML and related technologies, and the development of web and e-commerce applications. He has a distinction as a national researcher from the Mexican Government.

Miguel Castillo-Cortes is a mobile and web application developer. He received his BSc degree in Information Technologies and Systems from the Universidad Autónoma Metropolitana Campus Cuajimalpa, Mexico in 2017. He obtained the best final project award in his undergraduate studies. His research interests include technologies for supporting education, mobile and web application development.

Article submitted 2019-10-31. Resubmitted 2019-12-07. Final acceptance 2019-12-08. Final version published as submitted by the authors.