OOP-Anim, a system to support learning of basic object oriented programming concepts

Micaela Esteves, António Mendes

Abstract: *This paper presents OOP-Anim, a system to support learning of basic object oriented programming (OOP) concepts. For several reasons, programming is a difficult subject for many students, even in computer science courses. This happens independently of languages and paradigms used. Our system uses program animation to help students overcome some of the difficulties they usually feel to master OOP concepts and associated program dynamic.*

Keywords: *OOP-Anim, Animation, Visualization, Java Animation.*

INTRODUCTION

The option for object oriented languages in introductory programming courses is now common. Our own course has migrated from C to Java in the last few years. However, the difficulties felt by many students in learning how to program correctly haven't changed significantly. Many students continue to find difficult to understand the conceptual issues involved in programming and algorithmic design. Object-oriented programming concepts like classes, objects, references and messages are no exception and often many students fail to master them.

Programs have a dynamic nature, but most learning materials have a static format (e.g. text books) which makes difficult to explain (and understand) program's dynamic behavior. The abstract nature of programming is another source of difficulties, because many students fail to visualize how programming structures work and how problems can be solved using them. These and other reasons can explain the level of student failure common in introductory programming courses.

The object concept is essential to understand object oriented programming paradigm (OOP). However, our experience teaching OOP with Java shows that students have many difficulties to understand and visualize how the program works, especially when they need to work with OOP concepts.

Animation has been proposed as a way to make concrete and visual program's dynamics It can be argued that animated views can help students in three central learning activities: Understand programs; Evaluate existing programs; Develop new programs. This last activity is the most important and also the most difficult. Many students can understand programs, but they fail when they have to develop a program to solve some problem. Based in our experience, we developed an educational environment, OOP-Anim. It has features designed to help students visualize how their own object oriented programs work, allowing them to find and correct errors that may exist. Correcting their own mistakes is a good educational activity, since normally students reach a higher competence and confidence level after being able to correct all errors and have the program running correctly.

RELATED WORK

Several software tools have been proposed to support programming teaching and learning activities. Animation/visualization software systems have been used trying to take advantage of the potential of human visual system. Those systems are rooted in the conviction that programs can be better understood when represented graphically when compared with textual descriptions and representations.

For example, BlueJ [6] is an integrated system including an object-oriented language and an object-oriented development environment. It was developed specifically for teaching object-oriented programming to beginners. BlueJ uses UML- like class diagrams to present a graphical overview of a project structure. It allows the interactive creation of

objects from any given class present in the project. We used a similar approach in OOP-Anim, as we also present the project structure using UML. More detailed information about BlueJ is available in the literature [1, 2, 7].

Several studies have been published trying to evaluate how animation could help programming learning. For example, Kehoe et al. [4] concluded that when students were allowed to freely use a program animation and other study materials, they tended to use the animation together with the source code and/or a written explanation. Stasko et al. [5], used a system that animated data structures, without showing the source code. The students who used this system said they would prefer to have also a written explanation of what the algorithm is doing in each moment, and why. This study also concluded that the students wanted to have control over animation, for example running the program step by step instead of continuously, or going back to previous steps.

Miyadera et al. [3] reached similar conclusions. They concluded that most students think it is very important to have written explanations in addition to animations, allowing them to observe the code, its data structures representation, a brief explanation of what the program is doing at each step

The approach followed by Miyadera et al. was a major influence in our work, since we adopted a similar philosophy in our system. This means that OOP-Anim shows the program code, a representation of its classes and objects and a brief explanation of what the program is doing at each step.

OOP-ANIM - OBJECTIVES

As stated before, the main objective of OOP-Anim is to support learning of basic concepts of object orientated programming. It is essentially a visualization and animation tool that can animate programs created by the students and other common programs, allowing a better understanding of how they work and facilitating error detection and correction, since the student can compare how she/he thought the program would work with how the program really works.

In the first stage of learning OOP-Anim can be useful to animate example programs previously created by teachers or other students, allowing the student to better understand the fundamental concepts and how they operate to solve a problem. However, in a later stage, it is fundamental that students can animate their own programs. Most students find easy to understand correct basic programs, but things change when they have to create their own programs. At this stage OOP-Anim can help showing the student how her/his program works and help to locate, understand and correct bugs. This type of work generally is very useful for learning.

OOP-ANIM ENVIRONMENT

OOP-Anim has four essential areas. A screen shot of the system is shown in Figure1. The left side of screen shows the program code. As the program runs, the line currently being executed is highlighted.

There is an area below the program listing that displays program's input and output. This area acts like a terminal window.

The right half of the screen has two main parts. The upper part shows the program animation. This part is subdivided in three areas, one for a representation of the program classes, other for objects created during execution and the references used by the program.

The bottom part shows the buttons used to control the system. It is possible to open a program to animate, to close the program and clean the animation area, to close the environment, to start the animation continuously, to pause the animation, allowing the student to examine its current state, to run the animation step by step (pressing this button repeatedly, allows the student to run the program at any pace), and to go back to examine previous program states that are recorded as the animation runs.

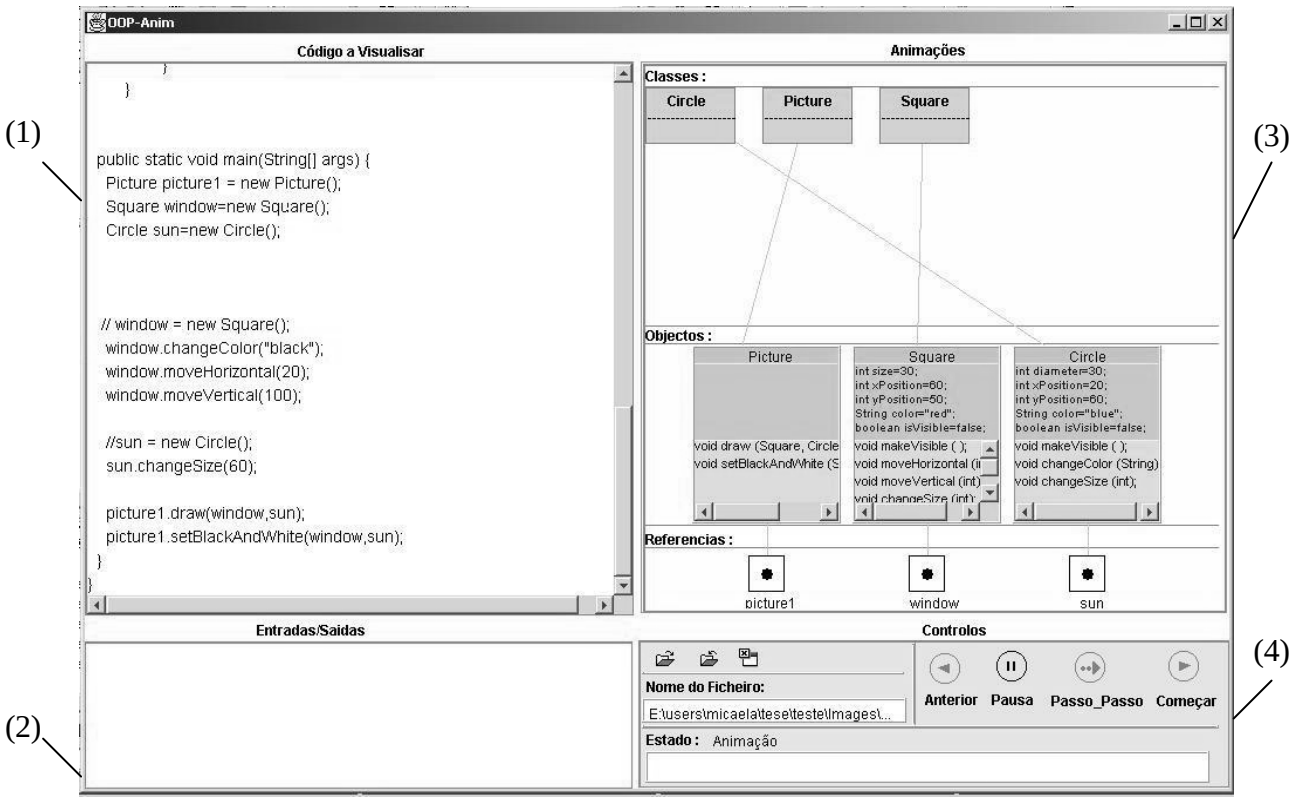

Figure 1: OOP-Anim

Finally, there is a status area that displays the program status (running, stopped, or waiting for input), and a brief explanation of what is happening at every animation stage.

In the animation area the classes are represented by a pink rectangle with the name of the class written. The objects created during program execution are represented by a red rectangle that shows the object's class name, its instance variables (with current values) and the methods defined in the object (inherited methods and own methods). The references are represented by a black rectangle with a black ball in the middle.

OVERVIEW

After opening the Java code file, the animation can be started by pressing the *Começar* (Start) button. After that the system highlights the main method header. Simultaneously all the classes that are part of the program are represented in the animation area (3). This representation includes the hierarchical relations that may exist between classes. The right mouse button, when pressed over a class, gives access to the class instance variables and methods (Figure 2).

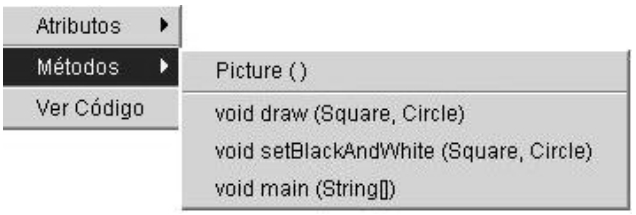

Figure 2: The class menu

After a few moments the highlight moves to the next code line indicating that the first instruction will be interpreted and executed. If the instruction implies the creation of an object, its representation appears in the animation area (3). This process is animated, first appears the reference representation, then it is the object representation that is created. As the class – object relation is a central concept, it is very important that the student

understand it well. The system animates the object creation as follows: Starting from the object's class representation, one red rectangle with the same dimensions of the class moves through the animation area until the objects zone. As mentioned before, the object representation contains: its class name, its instance variables (with values) and the methods defined in the object's class (Figure 3).

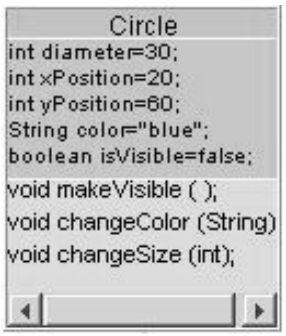

Figure 3: Object representation

Another important concept is message processing. When an object receives a message, this fact is highlighted by a change to red of its reference color. The method name in the object changes to green stressing that the object is responsible to answer the messages it receives. After the method is run and its results, if any, are shown in the corresponding object representation and/or in the output/input area (*Entradas/Saídas(2)*), if appropriate. This way the students can follow all the message sending and processing included in the program.

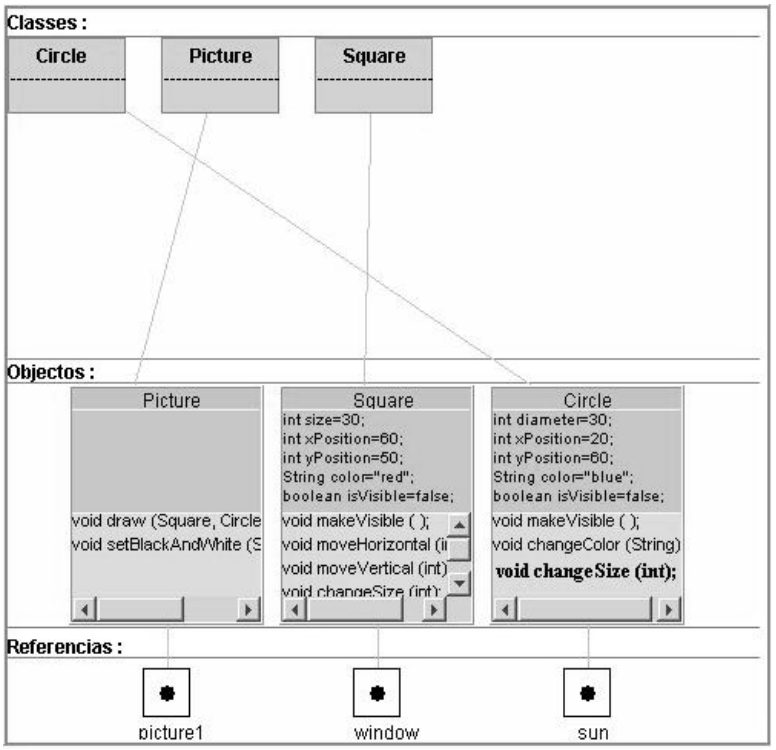

Figure 4: Message processing

The animation goes through all the program instructions, processing them in turn and showing its consequences in terms of instance variables values and/or program output. During execution the student can read some explanatory messages in the control area. The idea is to give some textual explanations that stress the most important execution aspects (Figure 5).

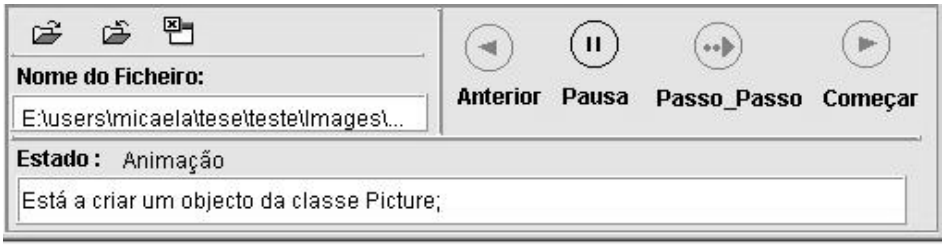

Figure 5: Textual notes about program execution

EXAMPLE

To clarify some OOP-Anim features, in this section we present a small utilization example, using the following main program:

```
public static void main(String[] args) {
    Picture picture1 = new Picture();
    Square window;
    Circle sun;

    window = new Square();
    window.changeColor("black");

    sun = new Circle();
    sun.changeSize(60);

    picture1.draw(window,sun);
}
```

The program creates three objects from classes *Picture*, *Circle* and *Square*. OOP-Anim starts the animation showing the code and the three classes representations as shown in Figure1. After a moment the system highlights the first line of code. In this example it creates an object of the class *Picture*. In the animation area (3) a reference with the name *picture1*, is created, the color of the box that represents the class *Picture* changes to red and a rectangle with the same dimension appears and moves until the objects area (Figure 6).

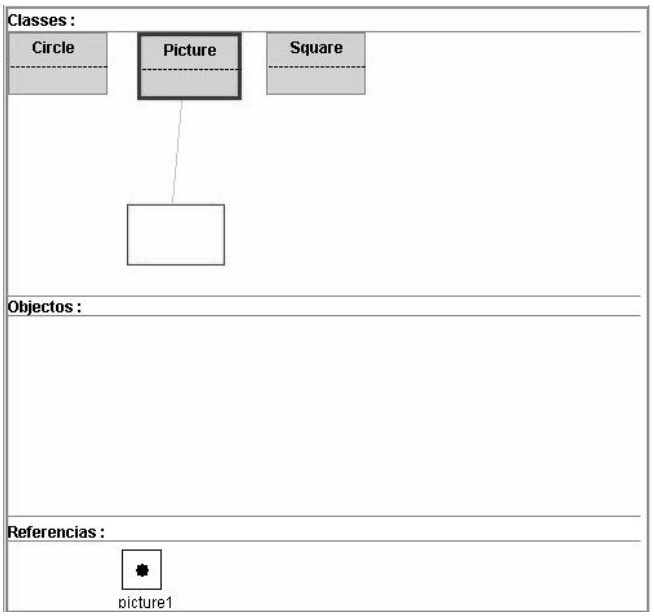

Figure 6: Creating an object

The next two lines of code create non initialized references to objects of classes *Square* and *Circle*. The names of these references are *window* and *sun*. This results in the situation shown in Figure 7. Our objective is to stress the difference between references and objects. These two concepts are often confused by the students.

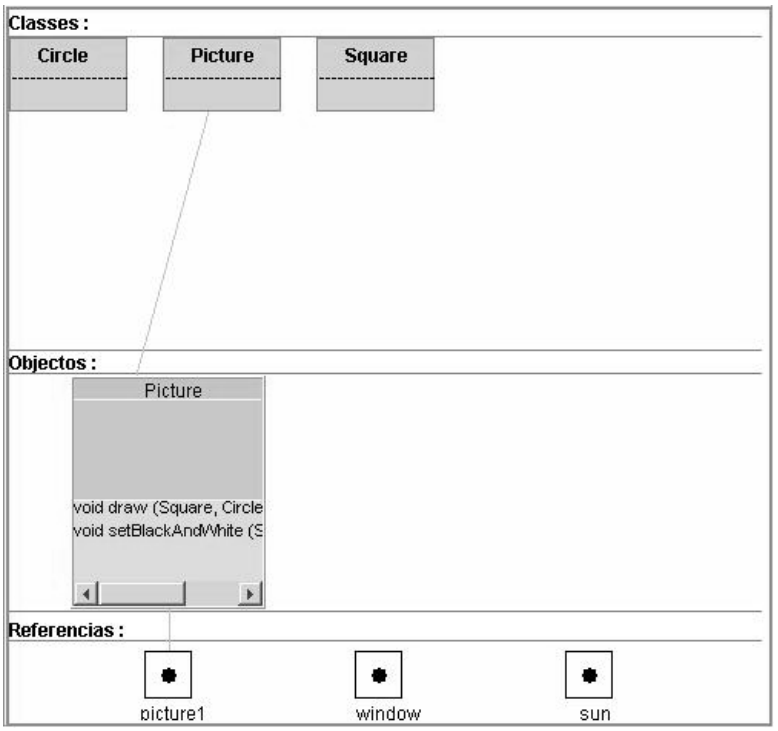

Figure 7: A null reference

Next line creates an object that will be referenced by *window*. The object creation is animated as described before and after it is created a line is draw connecting the reference and the object, showing that *window* now references the new object.

The next line is a message sent to the object referenced by *window* to execute its method *changeColor* with the parameter *"black"*. To show the steps involved the *window* reference color changes to red and a red ball goes from the reference to the object. When it arrives the method *changeColor* in the object representation changes to green and the result of its execution is shown (the instance variable *color* value changes). Finally, the representations involved change back to its original colors.

The next two lines of code have similar representations to those described above. The last line orders the object represented by *picture1* to execute method *draw*. Method execution was already described, but as this method prints the attributes of each object it receives, those attributes values will appear in the *Entrada/Saida* (input/output) area, as shown in Figure 8.

Figure 8: Input/output

CONCLUSIONS

Several studies have been made trying to evaluate the effectiveness of animation/visualization systems to support the development of programming skills. Many of them have verified a positive contribution of such systems to students learning, especially when interactive and multiple representation animation systems are used.

Our work tries to help to overcome some of the most important difficulties our students feel during object oriented programming learning. We think OOP-Anim will prove successful when used with our students. However, presently we are in the later development stage. We plan to have a first version soon and to evaluate that version with university programming students and teachers. This evaluation will probably result in new features and corrections to original ones. No educational package can be thought complete without a good evaluation and field utilization. This is the next step in our project.

REFERENCES

[1] Bergin, J. Fourteen Pedagogical Patterns for Teaching Computer Science, in Proceedings of the Fifth European Conference on Pattern Languages of Programs (EuroPLop 2000), Irsee, Germany, July 2000.

[2] Bergin, Joseph, Mark Stehlik, Jim Roberts, and Richard Pattis. Karel ++: A Gentle Introduction to the art of Object-Oriented Programming. John Wiley & Sons,1997

[3] Miyadera,Y., Huang, N. & Yokoyama, S. (2000). A programming language education system based on program animation. (IFIP-International Federation for Information Processing). Pp 258-261..

[4] Kehoe, C., Stasko, T & Taylor, A (199) Rethinking the Evaluation of Algorithm Animations as Learning Aids: An Observational Study, Graphics, Visualization, and Usability Center in Georgia Institute of Technology.

[5] Stasko, T., Badre, A.&Lewis, C (1993) Do Algorithm Animations Assist Learning? An Empirical Study and Analysis, Proceedings of the ACM INTERCHI'93,61-63.

[6] Kölling, Michael. The blueJ Tutorial- Version 1.4
http//www.bluej.monash.edu/tutorial/tutorial.pdf

[7] Smith, Phillip A. y Webb, Geoffrey I. The Efficacy of a low-level program visualization tool for teaching programming concepts to novice c programmers. Journal of Education Computing Research, Vol 22 No 2,2000, p. 187-216

ABOUT THE AUTHORS

Assist. Prof. Maria Micaela G. P. Dinis Esteves, Escola Superior de Tecnologia e Gestão de Leiria (ESTG), Portugal. Phone: +351 244820300, E-mail: micaela@estg.ipleiria.pt

Assoc Prof. António José Mendes, Centro de Informática e Sistemas da Universidade de Coimbra, Portugal. Phone: +351 239790000, E-mail : toze@dei.uc.pt