DEVELOPMENT OF AN APPLETS TO SUPPORT THE LEARNING OF SORTING ALGORITHMS

Paola Guindani Cordel, Nícolas Hudson de Mello, Tatiana Montes Celinski, Márcio Augusto de Souza

Universidade Estadual de Ponta Grossa

paolacordel@gmail.com, nicolas_mello@yahoo.com.br, tmontesc@uepg.br, msouza@uepg.br

Abstract: This paper has the objective of proposing a learning object for simulating sorting algorithms, implemented with java applets, which provides interactive, didactic and easy handling to teachers, students and to the community that may be interested in algorithms learning. With this dynamic material it is possible to improve the learning process of the sorting algorithms, because it is possible to see the sequence of operations performed in an interactive form, as well as to check this efficiency.

Keywords: Learning objects; Applets Java; Sorting algorithms

1. INTRODUCTION

In many areas of knowledge there are concepts that are not easy to learn. To help students to understand or to improve the learning of concepts treated in classroom, it is interesting the use of supplementary materials, such as learning objects.

Some researches affirm that freshman computer students encounter problems to understand abstract programming concepts (Tuparov et. al, 2013). In this context, it is important to create learning objects that motivate the students and allow better understanding of such concepts.

In computer science, there are many complex concepts for which the development of learning objects can be helpful. As an example, it can be cited the sorting algorithms, which have some details that can be confusing to students. A learning object that allows to watch the step by step execution of the algorithm visually is very useful for understanding it. Besides that, as pointed out by Tuparov et al. (2013), sorting algorithms can hold the attention even of low interested students.

This work proposes a learning object for sorting algorithms developed as a Java applet. This paper presents and description of the interface of the applet and the details of its implementation. The applet was implemented to be easily adapted for other applications, allowing the code to be reused with efficiency.

2. LEARNING OBJECTS

According to Prata and Nascimento (2007), a learning object can be understood as any digital resource that can be used to support the learning process. They can be created using any kind of medium and format, ranging from a simple slideshow to a complex simulation. Images, animations, applets and text are examples of learning objects when they are created with an educational purpose.

According to Monteiro et al. (2006), an animation is a learning object that can provide an active learning, because it generates an excitement of various cognitive processes, such as perception, memory, language, and reasoning. Furthermore, an interactive

animation allows a more efficient cognitive development, because it forces an individual experience to the user when manipulating the tool.

Interactivity is a fundamental characteristic of learning objects that are attractive, playful, dynamic and useful for classroom or distance teaching. (BRAGA et al., 2012).

By using learning objects, the teacher can improve significantly the student education. As discussed by Wetzel et al. (1994), the verbal information, which is the teacher explanation during the classes, combined with the visual resource, increases the interest and, consequently, the understanding of the student.

According to Braga et al. (2012), there are certain requirements that a learning object must fulfill, such as: pedagogical abilities, availability, accessibility, precision, reliability, ease of installation, portability, interoperability and usability. The more these characteristics are reached, the better the learning object will be.

The accessibility is an important characteristic of a learning object, so it must be available in a location that is easily accessed by students and teachers. For easier use and reuse, the object must be stored in a repository following a meta-data pattern (Nitzke et al., 2012).

A meta-data pattern is used for better organization of components stored in a data repository, it ensures that there are no data errors or inconsistences and it eases search operations (Hillmann; Dushay, 2003).

This work proposes a learning object for simulating sorting algorithms, which are discussed in the next section.

3. SORTING ALGORITHMS

Algorithms are part of the daily routine of people. Instructions to assemble an equipment, prepare a cooking recipe and use a medicament are examples of their use. An algorithm can be seen as a sequence of actions that must be executed to obtain a solution for some problem.

A common operation that is performed by means of algorithms is sorting. For example, phonebooks and list of clients, products or students are examples of information that is sorted for efficient access. In the cases when there are a great number of elements, the sorting is commonly made by computers.

As said by Ziviani (2002), sorting algorithms are a good example of how to solve problems using computers. There are several sorting techniques available, and all of them do the same task. Depending on the application, each algorithm has advantages and disadvantages when compared to the others.

For a computer science student, analyzing the differences among the sorting algorithms is an efficient approach for learning and understanding algorithms in general. The correct comprehension of a sorting algorithm depends on a graphical representation of its execution, which is the goal of the learning object. The classic sorting algorithms that will be available in the learning object are: selection sort, bubble sort, insertion sort, selection sort, bubble sort, insertion sort, heap sort, merge sort, quick sort, radix sort and shell sort.

The next section presents the interface and the details of implementation of the learning object proposed in this work.

4. RESULTS AND DISCUSSION

To create the learning object, it was necessary to research for a technology that would be available for most students, regardless of the computer used by them. A java applet was chosen because it can be transparently executed in browsers with Java support, which are easily accessible for students.

The applet was developed with Eclipse platform, which is a programming tool that provides features that ease the implementation and test of the applet being implemented.

There are many learning tools developed using java applets that are available to be used in the internet, and some of them were studied during the development of this work. This analysis of related works was important for the definition of the resources, colors and size of components of the proposed applet.

The applet developed by Lafore[1], for example, has resources that are helpful for better interactivity. Another one that has an interesting structure is the applet from Eck[2], which defines a schema of colors for animated bars that have been adopted in this work. The applet made available by Mohammadi[3] shows that it is important to have a frame which shows the algorithm code.

Based on the qualities of the related work, it was proposed an interactive applet for simulating sorting algorithms with a simple and intuitive graphical interface, which use colors that contrast and provide good visibility of the operations being performed during the execution of the sorting algorithm.

The Figure 1 represents the interface of the applet, and a description of each of its components.

[1] Available in http://sites.fas.harvard.edu/~cscie22/resources/lafore/. Access on June 2014.

[2] Available in http://math.hws.edu/TMCM/java/xSortLab/. Access on june 2014

[3] Avalaible in http://www.cs.oswego.edu/~mohammad/classes/csc241/samples/sort/Sort2-E.html/. Access on june 2014.

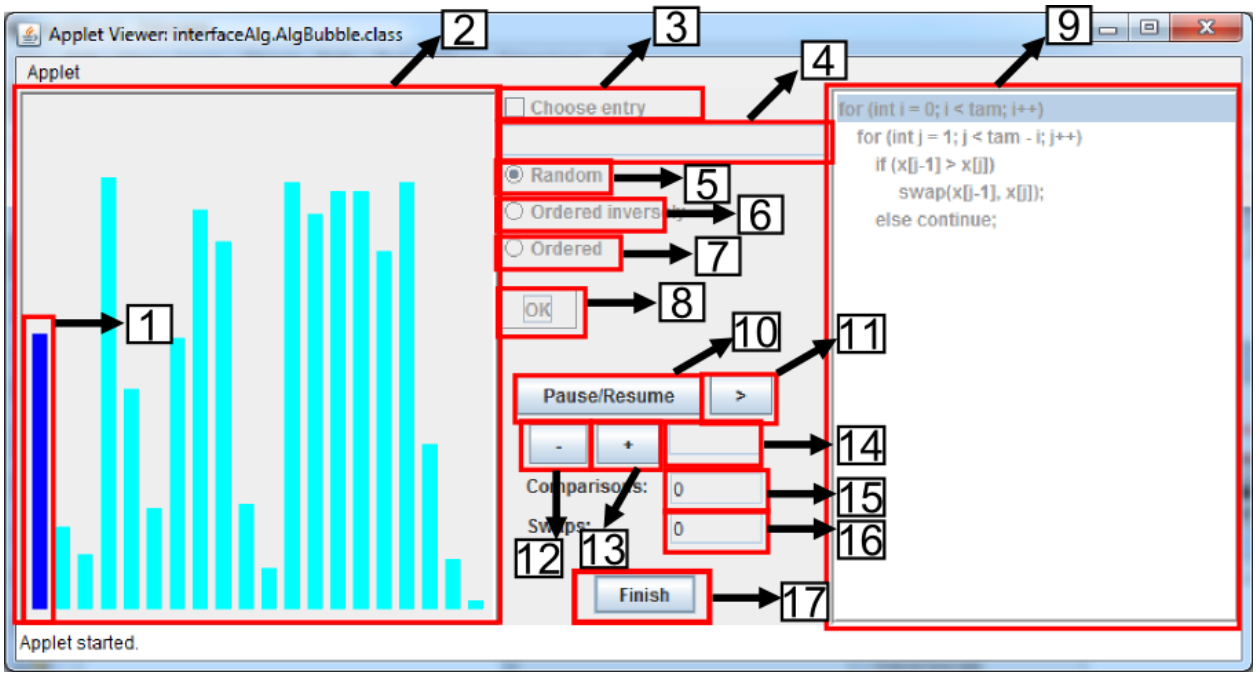

Figure 1 - Applet interface

1) Bar: Each number is represented in form of a bar. Bigger numbers are represented by bigger bars. The colors represent the current state of the number, and they represent:

a. Ciano: Initial position.
b. Green: The bar is in the correct position.
c. Blue: Bar selected for comparison.
d. Red: Bar being compared.
e. Orange: Bar changed.
f. Gray: Bar not changed.

2) Bars frame: Shows the graphical simulation of the sorting algorithm
3) Choose entry: The user can choose the numbers to be sorted.
4) Data field: Numbers typed by the user.
5) Random: Generates a random number set.
6) Ordered Inversely: Generates an inversely ordered set.
7) Ordered: Generates an ordered set.
8) OK button: Initiates the simulation
9) Code frame: Algorithm representation, where it is shown which instruction is being executed.
10) Pause/resume button: Enables or disables the automatic execution.
11) Button >: Executes the next operation. This feature is enabled when the automatic execution is not active.
12) Button -: Decreases the execution speed.
13) Button +: Increases the execution speed.
14) Velocity field: Shows the duration of a pause between two consecutive operations in milliseconds.
15) Comparisons field: Counts the number of comparisons performed.
16) Swap field: Counts the number of swaps performed.
17) Finish button: Ends the simulation.

When producing software, is very important to think in reuse. Newly written code is more likely to have bugs than reused code (Binder, 2000). With the goal of defining a basic structure for the different sorting algorithm applets, it was defined an abstract class named **MainClass** with common methods for all the sorting algorithms and one abstract method named **defineAlgorithm**. This method must be implement by any class that extends the **MainClass**. The Figure 2 shows an example of a class created for simulating the insertion sort algorithm. This will vary according to the algorithm to be implemented, for example, **new Insertion()** for insertion sort, **new Bubble()** for bubble sort and so on.

Four parameters are passed by reference when the Insertion class is instantiated, and they are related to specific information controlled by the sorting algorithm: the position of the animated bars, the algorithm code and the number of comparisons and exchanges made.

```
Public class AlgInsertion extends MainClass {

	@Override
	Public void defineAlgorithm() {
		alg = new Insertion(bars, algorithmField,
comparisonsField, chagesField);
	}
}
```

Figure 2 – Example of a class that initiates the insertion sort simulation.

The algorithm being simulated is implemented by extending an abstract class called **Algorithm**. Figure 3 shows an example of the insertion sort implementation.

```
public class Insertion extends Algorithm {

	public Insertion(Bar[] bars, JList<String> field, JTextField
comparisonsField, JTextField changesField) {
		super(bars, field, comparisonsField, changesField);
	}

	public void sort() {
		// sorting code
	}

	public String[] getStr() {
		// pseudo-code to be showed in the code frame
	}
}
```

Figure 3 – Insertion sort class

The abstract class **Algorithm** has two abstract methods: **sort( )**, which is responsible for moving the bars and refreshing the interface when it is necessary (for example, a color

exchange or the update of the number of comparisons made by the algorithm); **getStr( )**, which returns the algorithm code, which is showed in the item 9 in Figure 1.

It is simple to create new sorting algorithms applets based on the model presented in this work. It is just necessary to instantiate the class responsible for the sorting and implement the methods **sort( )** and **getStr( ),** as it can be seen in Figure 3.

After creating the prototype interface and the required classes, the sorting algorithms were implemented adapting their codes for proper operation at the interface. A recurring problem of graphical components that need constant updating is flickering, which is a *blink* that occurs in the redraw of components of an applet. This problem has been observed in many similar applets studied, and in this work, it occurred in the drawing of animation bars.

This flickering problem was solved by the use of the following technique:

1) An object (called **frame)** was created for representing the frame where the bars will be drawn.
2) The creation of the object was made by the code showed in Figure 4.

```
frame = new JPanel() {

        @Override
        protected void paintComponent(Graphics g) {
                super.paintComponent(g);
                drawBars(g);
        }
};
```

Figure 4 – Code portion responsible for drawing bars

The **paintComponent( )** method was overridden, and it kept a call for the overridden method (using the *super* keyword), which is used to draw the previously configured features (background color, border, border color, etc.).
In the next line it is called the method **drawBars( )**, which draw the bars in the context defined by the parameter **g**.
As everything is inside of the **paintComponent( )** method, backbuffering is used, avoiding the flickering. If the bars were printed inside the applet **paint()** method, the bars would be printed sequentially, resulting in the unpleasant flashes of the window.
3) Finally, to avoid updating the whole window, it was implemented the method **updateFrame( )**, showed in Figure 5, which is called from the sorting algorithm when it is necessary to update the positions or the color of the bars.

```
public void updateFrame() {
        frame.repaint();
}
```

Figure 5 - Code portion responsible for updating the bars frame

5. CONCLUSIONS

Learning objects can be used in many areas of education, serving as support material for teachers and as study material for students. Interactive learning objects permit more effective learning, as they allow more interaction with users.

In the study of sorting algorithms, it is a practice very useful. Through visual resources, the student can follow the process of sorting values graphically, which provides a better understanding of the methods.

This work presented a description of a learning objects for sorting algorithms implemented by means of Java applets. It was presented the interface and the methodology used to develop the applet, and a solution for the problem of flickering, common on related projects.

For better use of the developed material it is intended to make it available in a public repository. The code can be adapted for other uses, and may be used for the creation of learning objects for another problems in computer science.

REFERENCES

BINDER, R. V. **Testing Object-Oriented Systems: Models, Patterns, and Tools**. 2000.

BRAGA, J. C. et al. **Desafios para o Desenvolvimento de Objetos de Aprendizagem Reutilizáveis e de Qualidade. UFABC**. Workshop de Desafios da Computação Aplicada à Educação, 2012.

HILLMANN, D. I.; DUSHAY N. **Analyzing Metadata for Effective Use and Reuse**. Cornell University. National Science Digital Library, USA, 2003.

MONTEIRO, B. S. et al**., Metodologia de desenvolvimento de objetos de aprendizagem com foco na aprendizagem significativa**, João Pessoa: UFPB, 2006.

NITZKE, J. A.; CARNEIRO, M. L. F.; PASSOS, P. C. S. J. **Gestão do desenvolvimento de objetos de aprendizagem digitais**. UFRGS. Núcleo de Apoio Pedagógico à Educação a Distância, 2012.

PRATA, C. L.; NASCIMENTO, A. C. A. A. **Objetos de aprendizagem: uma proposta de recurso pedagógico**. Brasília: MEC, SEED, 2007.

TUPAROV G.; TUPAROVA D.; JORDANOV V. **Teaching sorting and searching algorithms through simulation-based learning objects in an introductory programming course**. 5^{th} World Conference on Educational Sciences – WCES, 2013.

WETZEL C.; RADTKE P.; STERN H. **Instructional effectiveness of video media**. Hillsdale, NJ: Lawrence Erlbaum Associates, 1994.

ZIVIANI, N. **Projeto de Algoritmos Com Implementações em Pascal e C**. 4ª ed. São Paulo: Editora Pioneira, 1999.