INTERPRETER AND APPLIED DEVELOPMENT ENVIRONMENT FOR LEARNING CONCEPTS OF OBJECT ORIENTED PROGRAMMING

Ever Mitta[1] and Layla Hirsh[2]

[1]Department of Computer Engineering, Pontificia Universidad Católica del Perú, Lima, Perú.
emitta@pucp.pe
[2]Department of Computer Engineering, Pontificia Universidad Católica del Perú, Lima, Perú.
Lhirsh@pucp.edu.pe

ABSTRACT

The programing languages are classified by its paradigms, some paradigms are easier to understand than others, or at least we have that impression. But when we learn our first language, what paradigm is the best? Many people could say that structured is better, because is a recipe to follow, or maybe an object oriented, because it-s a natural definition, others could say that functional is better. Actually the debate of which should be the first language to learn is growing, and it will continue growing. Nowadays we decide which language to learn as a result of a necessity, more than analyse which one is easier to understand or which should be the correct learning process. This paper will present a tool in Spanish that could be used as an instrument to understand these concepts of object oriented and its management before starting the programming.

KEYWORDS

IDE, Object Oriented Programming, language

1. INTRODUCTION

There are cases where people trying to enter the world of object-oriented programming [OOP] suffer a lack of a theorical basis of concepts in different programming paradigms making it difficult or impossible in some cases to obtain a correct implementation program in any programming language. People who are getting started in the programming of applications based on object-oriented paradigm often have problems because they do not understand the basics of this paradigm, such as encapsulation, hiding, abstract classes, interface, among others.

It is necessary to develop applications / programs efficient and understand precisely these concepts, not just how they should be treated but also when should be used. To clear these concepts could help not only the people who program but also any other people using this paradigm from another perspective such as the analysis or design.

While having prior knowledge of structured paradigm is a good start to function in the object-oriented paradigm, the problem lies in trying to use this knowledge as the sole basis for understanding the OOP paradigm.

DOI : 10.5121/ijpla.2013.3302

Unfortunately, the tools / languages used mostly to get into the object-oriented paradigm does not allow us to understand these concepts fully, however in some cases the language often limits the proper use of all the concepts of this paradigm such as in the case of not allowing multiple inheritance (eg java, C #, Delphi, etc.) or, its opposite, allow this type of inheritance, but not allow the implementation of interfaces (eg C + +, Perl, etc.).

There are some applications that try to do facilitate the learning process, each one for different paradigms, like "Interpreter and development environment for learning structured programming languages"[2], used to learn the structured paradigms. On the other hand, there are also tools for the object oriented paradigm such as jIDEE :: Java IDE for Education4, BlueJ – The Interactive Java Environment and JotAzul Object Oriented all this have their advantages and disadvantages, in the following tables we will see some of them.

Table 1. jIDEE' advantages and disadvantages.

jIDEE :: Java IDE for Education4	
It is a free programming environment designed to meet the educational requirements for the introduction to Java. [3].	
Advantage	**Disadvantage**
Manage source files, files with a ". Class".	Use external text editor
Run with arguments.	Does not contain mechanisms to facilitate the understanding of OOP concepts
Compilation and Execution.	
The operation is performed with a text editor, for which contains an interface to choose.	

Table 2. BlueJ – The Interactive Java Environment'advantages and disadvantages.

BlueJ – The Interactive Java Environment	
It is a development environment designed for the purpose of teaching the paradigm of object orientation with Java.[4]	
Advantage	**Disadvantage**
Manage source files, files with a ". Class".	Use external text editor
Run with arguments.	Does not contain mechanisms to facilitate the understanding of OOP concepts
Compilation and Execution.	
The operation is performed with a text editor, for which contains an interface to choose.	
Contains an interface to choose the Path to the JDK.	

Table 3. JotAzul Object Oriented – The Interactive Java Environment'advantages and disadvantages.

JotAzul Object Oriented	
It is a development environment designed to help beginners in the world of object-oriented programming in Java. [5]	
Advantage	**Disadvantage**
Mechanisms to add attributes and classes	The language used is Java, which does

simple.	not allow multiple inheritances.
Management classes through a structure on the left side.	The interface is not very friendly, since all areas are compressed.
Using Class Diagram	Unit of mechanisms to add attributes and methods, you can modify the code as if it were one.
Abstraction Management	Little known to the public.

Based in all this advantages and disadvantages and also all the found requirements, a new proposal has been developed

The person interested in learning this paradigm must have a tool / language whose purpose is not to develop complex applications (Netbeans, Eclipse, Visual Studio, etc.), but rather should use simple tools that help the understanding and then migrate to tools / languages specialized in applications developing.

2. PROPOSAL DESCRIPTION

A tool that let us understand the concepts, that sets the focus on the object oriented idea more than in the language statements, or the component that should be drag and drop. She should remember that is really important to understand what is going on inside our code when we drag and drop component, because usually a big amount of code is added and we don't realized that maybe there is code that could be deleted or used.

The proposal is to create a language self-described in Spanish and an integrated development environment that work together with the help of an interpreter. Also this IDE will include an assistant that will show the concepts and will have many dialogs that will guide the user while making the different programs. This is why this proposal description is divided in three main parts, the grammar design, the interpreter design and the integrated development environment described as follow.

2.1. Grammar design

We tried to develop a grammar that could help to define the main concepts of OOP. The first characteristic to define was the language, Spanish was better because is the mother language. Spanish is a complicated language, could make the statements kind of long, maybe more than desired, but against this buts, it could be easily understood.

Another important characteristic is that it should be in the middle between a pseudo code and a programming language. Thinking always in these two characteristics, a Backus-Naur form (BNF) methodology was used to define the grammar.

The decisions of what productions were needed were defined based on Java, and C#, also using the help of professors that lectures courses of programming languages object oriented and structured. Also with the help of books, a set of possible productions were defined and then using a BNF methodology the following productions were defined.

Another really important factor was the lexemes, and they could affect directly to the final result. A middle point should be found, a point where the lexeme should be easy to understand and also easy to remember and of course easy to write and use. This is why all the lexemes and tokens are

really descripting and use the Hungarian notation or a special char that helps the user to understand what it is.

Table 4. Some lexemes and descriptions.

Lexeme	Description
#metodo	Method start
$publico	Access modifier
#atributo	A class attribute
@principal	The main program
$noRetorno	No return value, a procedure
#ejecutar	Run

Part of the grammar using BNF form is shown in the following figure. As can be read the tokens and the non-terminals are really descripting.

```
cuerpo :        |
                clase cuerpo
                | interfaz cuerpo

interfaz :      TK_INTERFAZ  ID_CLASE '{' seccionAtributos
                seccionDefMetodos '}'

seccionDefMetodos : | defMetodo seccionDefMetodos

defMetodo  :  TK_METODO ':' modificadorAcceso tipoDato ID '('
                seccionParametros ')' ';'

clase  :  TK_CLASE  abstraccion  ID_CLASE seccionHerencia
          seccionInterfaces '{' seccionAtributos seccionMetodos  '}'

abstraccion  : | siAbstraccion

siAbstraccion : TK_ABSTRACCION

seccionHerencia  :  | TK_HEREDA   listaClasesHerencia ;

listaClasesHerencia : ID_CLASE
                    | listaClasesHerencia ',' ID_CLASE
```

Figure 1. Some grammar productions

BOOP language has not much reserved words, each word has a related token, all are shown in the following table.

Table 5. Tokens.

Token	Token	Token	Token
TK_INTERFAZ	TK_ABSTRACCION	TK_PROTEGIDO	TK_DEC
TK_ATRIBUTO	TK_POTENCIA	TK_CASO	TK_IMPLEMENTA
TK_METODO	TK_CONCATENAR	TK_IMPRESION	TK_RETORNA
TK_EJECUTAR	TK_DIVENTERA	TK_PUBLICO	TK_NADA
TK_ENTERO	TK_RESIDUO	TK_SI	TK_COMO
TK_DECIMAL	TK_HEREDA	TK_PARA	TK_ENTORNO
TK_MIENTRAS	TK_SINO	TK_VERDADERO	TK_IMPRESION2
TK_VOF	TK_FALSO	TK_MET	TK_PRIVADO

2.2. Interpreter design

The design of the interpreter was in Java language, this decision was taken because it makes easier the management of the data. Of course, a lexical analyser was designed, and for the syntax analyses YACC was used. For this two phases, the idea is to use the code written in the IDE's text area and use it as input for the lexical analysis, then the output for this phase is used as input for the Syntax analysis. After those two interpreter phases, the semantics actions were defined. The hardest part after this front-end process was to set the data structure that will help us to manage all the OOP concepts and also allow us to compile all the programs that could be created[1].

To manage the data java language was really helpful, the array lists help to manage all the data from every class, and of course, java and its object oriented design made this development really easy to elaborate. The main problem was to manage all the idea of activation registers, because for an instance of an object a new activation register should be used, and of course an original version of the class definition should be saved. Internally all this is well managed thanks to java objects and of course previous experiences.

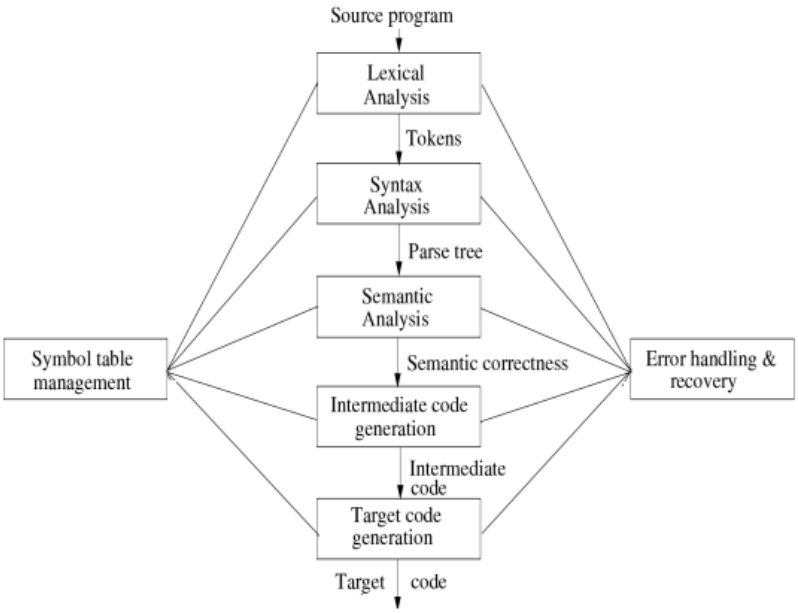

Figure 2. Compiler phases

For the back-end process the intermediate code generation was defined in Spanish and uses simple object oriented definitions, finally the code is interpreted and the output is shown in the IDE's output area.

As can be seen in figure 2, the optimization phase is not taken into account, because the code we will generate should be simple and the optimization won't achieve a great improvement in our code or a great saving of resources.

2.3. Integrated development environment

For the interaction between language/interpreter users an IDE was created. The idea of this tool is to allow the user to create programs in an OOP language in Spanish and with the help of the assistant. The principal function of this IDE is that allows the user to create programs without thinking of the characteristics of the language but thinking in the main paradigm concepts. This

tool interacts with the interpreter and of course the grammar, always using the basic concepts of the paradigm.

The IDE presents five sections. The first section is the text area, where the user is allowed to write code or to insert it, then the button area, that presents special language options to insert the code. The three other sections are the menu bar, the output area and the assistant. More detail of all these parts will be described in the next section.

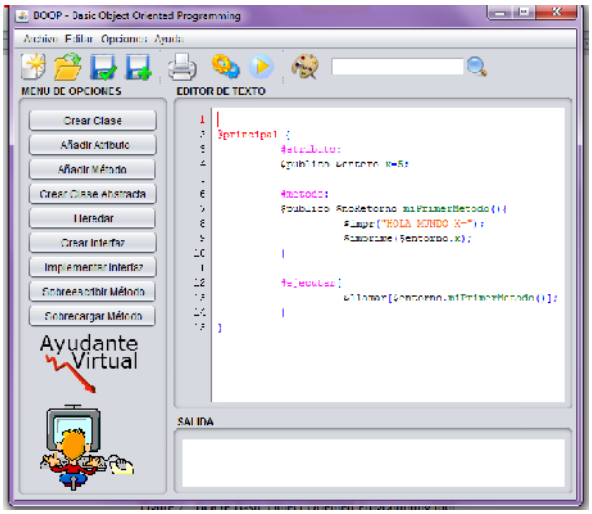

Figure 3. BOOP Basic Object Oriented Programming tool

3. PROPOSAL IMPLEMENTATION

This section will be divided in two, IDE/interpreter and IDE assistant. Each one with many functionalities some will be described and others will only be mentioned.

3.1. IDE and interpreter

The language let us differentiate between some concepts, for example: main class, comments, attributes, methods, variables, types of data, access modifiers, value assignation, class definitions, use of attributes, and use of methods, basic programming structures as selective and iterative structures.

Something particular for the language is the relation with the IDE, and how it let you create a program just following the instructions and using the buttons. When a new file is created automatically a main class in created.

The main class

This is the first executed class; inside this we could create methods, attributes. This class has a particular method,"ejecutar" that is the starting method. Also in this class, attributes and other methods could be created. This is the first step to create a program using BOOP. Something important to mention is that all the reserved words are shown in a special colour that could be changed by the user, and also have a special character at the beginning. Furthermore, there are other special words like the name of the methods and the name of the classes that also are painted in a different colour

In the next figure a basic example is shown.

```
@principal {
        #atributo:
        $publico $entero x=5;

        #metodo:
        $publico $noRetorno miPrimerMetodo(){
                $impr("HOLA MUNDO X=");
                $imprime($entorno.x);
        }

        #ejecutar{
                $llamar[$entorno.miPrimerMetodo()];
        }
}
```

Figure 3. BOOP Main class example

In the previous section we mentioned a button section; this button section allows the user to insert code just by pressing a button. See Figure 4.

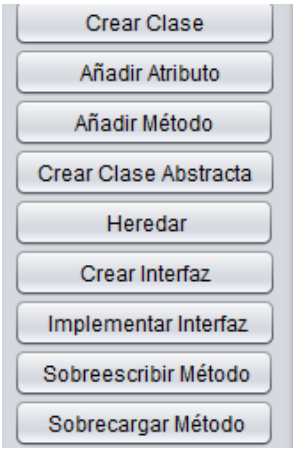

Figure 4. BOOP Buttons section

Using the different options the user just need to follow the instructions, for example, to create a new class we could press the button “Crear Clase” and a helpful dialog will guide us.

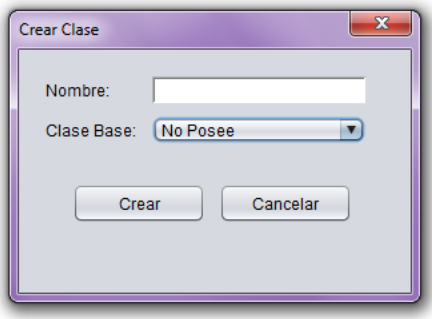

Figure 5. BOOP Create class

As we can see in Figure 5 the user just need to write the name of the class and the code will be updated, also as is shown in the same figure we can define heritage from another created class,

also using another buttons we could add attributes and methods. In figure 6 the adding method dialog is shown, all the parameters are set in the dialog of figure 7.

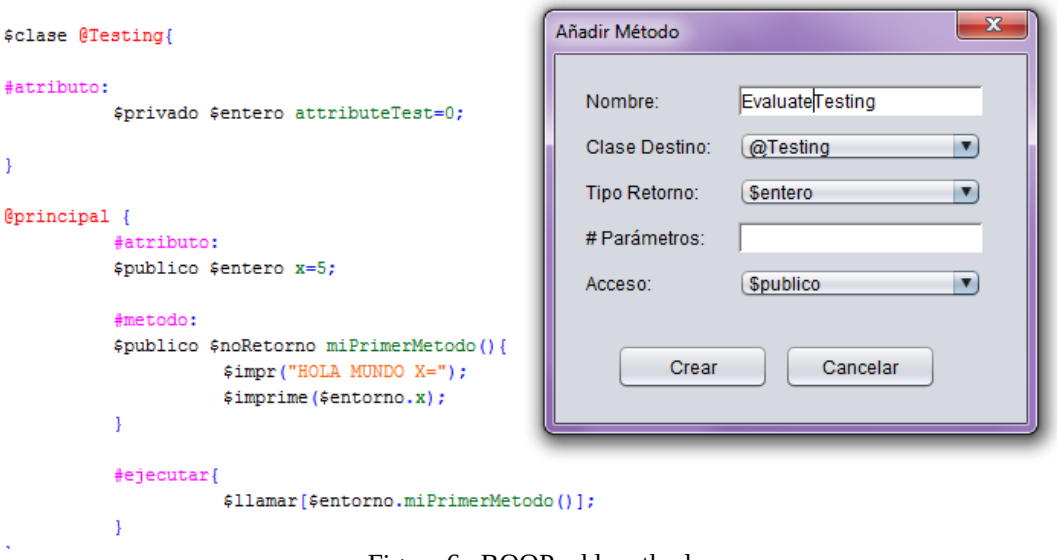

Figure 6. BOOP add method

As completing the numbers of parameters, the next dialog will appear asking not only for the name of each parameter but also for the data type, of course they are part of the previous created class.

This dialogs gives all the instructions needed, of course there are concepts that should be learned, but there is where the assistant completes the learning cycle. As the IDE and the interpreter are taking care of the programming, the statements, and language syntax, the assistant will complement the work.

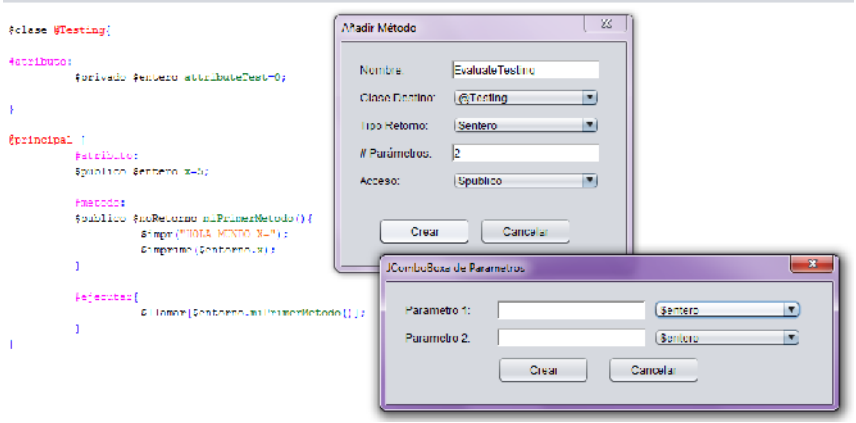

Figure 7. BOOP add parameters

The idea of these dialogs is to allow the user to complete the code, and practice creating new programs. After some time of practice the user can stop using the buttons and try to create code without the help of neither the assistant nor the buttons, but with the font colour to guide him during the process.

In the output section we will see all the results after running the program. And to run the program and create it, save it, and edit it we can use the menu bar. In this section we will also find the icon that will allow the user to define the text colour, as shown in figure 8.

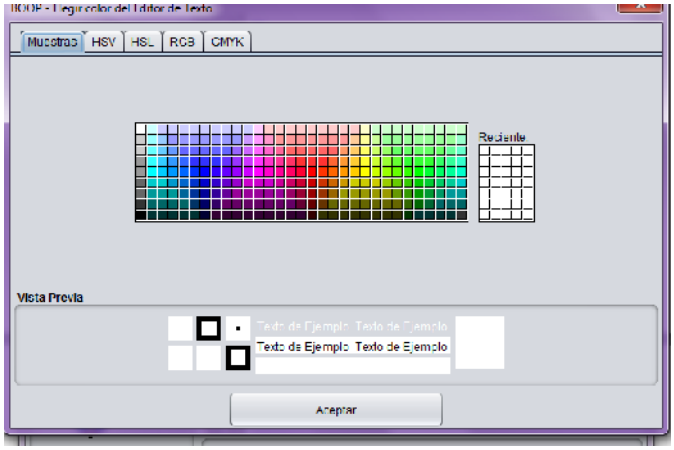

Figure 8. BOOP Colour pallet

2.5. The IDE assistant

During all the creation of the code, there is the assistant ready to explain us the concept, not only by reading but also by hearing, this definitions given by the assistant help us to create code that we understand. And of course understand the concept while using them.

The review of OOP books helps us to define the different concepts that could be needed, for the talking voice a speech synthesizer was used, and of course in the design of the IDE we used NetBeans.

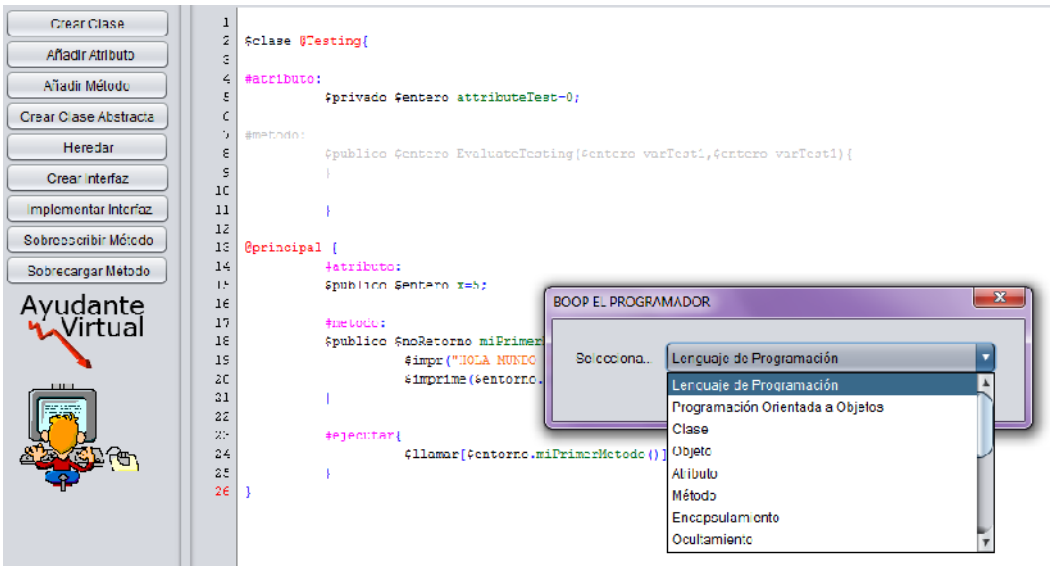

Figure 9. BOOP Assistant

3. CONCLUSIONS

It's not possible to define which language should be the first to learn, nor the paradigm. But we can evaluate which language will be more helpful for each kind of user. Of course if the user will be a Computer Engineer or a system engineer we can start a debate, but what about engineers that will use only some specific software products, that are most object oriented. What about the concepts that it's complicate to understand. A link between the natural thinking and the object oriented programming is needed, more when the programming language is in a different communication language.

By other side, if the learning starts with the structured paradigm, everything is view as a recipe, steps to follow and that's it. The change between one language and another is not hard, some statement may change but the structures and concepts are the same. But when the paradigm changes the difficulty starts to grow.

What if the learning starts with the OOP, the idea of objects seems to be really natural, and an object with characteristics is able to do many things. But the question should be how to integrate these concepts to a correct programming.

Learning a programming language is not as important as learning the paradigm, if the basic concepts of the paradigm have a firm base, learning the language statements is secondary.

The developed tool was created to ensure this learning process by trying to explain concepts during the programming, using a familiar communication language and also a friendly IDE. The product was thought more for people that want to learn to use an object oriented programming language without a previous programming knowledge.

Obviously, if we understand the concepts the practice will be easier to understand. The idea of a tool in our communication language is a good start, and then with just learning the language statements we should be able to create code using object oriented languages as Java or others.

REFERENCES

[1] Aho, Alfred V., (2007) "Compilers: Principles, Techniques, & Tools", 2nd Edition. Pearson Education.

[2] Hirsh, Layla, (2008) "Intérprete y entorno de desarrollo para el aprendizaje de lenguajes de programación estructurada", VII Jornadas Iberoamericanas de Ingeniería de Software e Ingeniería del Conocimiento. Pag. 189-196

[3] Mithat Konar,(2010), "JiDEE :: Java IDE for Education". http://jidee.tuxfamily.org/

[4] University Of Kent (2010)."BlueJ – Teaching Java. Consulta: http://www.bluej.org

[5] Morgade, Marcelo, JotAzul – IDE Java para Aprendizaje de POO. http://jotazul.sourceforge.net/

[6] Joyanes Aguilar, Luis, (2006) "Programación en C++ Algoritmos, estructuras de datos y objetos", 2da Edición. Madrid: McGraw-Hill/Interamericana de España, S.A.U.

[7] Deitel & Associates, Inc, (2005) "Java How To Program", 6ta Edition. New Jersey: Pearson Education, Inc.

[8] Mithat Konar, (2010) "JiDEE :: Java IDE for Education." Consult: 11 de April del 2012. <http://jidee.tuxfamily.org/>

[9] University Of Kent, (2010) "BlueJ – Teaching Java". Consult: 11 de april del 2012. <http://www.bluej.org>

[10] Morgade, Marcelo, (2012) "JotAzul – IDE Java para Aprendizaje de POO". Consult: 12 de April del 2012. <http://jotazul.sourceforge.net/>

[11] Ronda león,Rodrigo, (2007) “Revisión de Técnicas de Arquitectura de Información. No Solo Usabilidad Journal”. Consult: 03 de June del 2012 http://www.nosolousabilidad.com/articulos/tecnicas_ai.htm

[12] Zsuzsanna Papp-Varga, Péter Szlávi, László Zsakó. *ICT teaching methods – Programming languages.D*epartment of Media and Educational Informatics Eötvös Loránd University, Budapest, Hungary. Submitted 7 March 2008; Accepted 9 December 2008 Annales Mathematicae et Informaticae 35 (2008) pp. 163–172

Authors

Ever Mitta F.
student and teaching assistant of the Pontificia Universidad Catolica del Peru

MSc. Eng. Layla Hirsh M.
Professor of the Pontificia Universidad Catolica del Peru