An OOP w/ Java Course Using an Inductive Approach

Duygu Çakır[1], Selvihan N. Kaptan[2], and Adem Karahoca[3]

Abstract— **In the first class of CE and SE departments, all of the students get a compulsory software language course. This paper indicates that using a procedural and inductive approach in teaching object oriented languages is far more better than using the object-first approach with regard to the experiences gained during 5 years of an introductive OOP Java course. Our supporting ideas are based on student feedbacks and their successes not just during their studentship but also in the industry.**

***Index Terms*— Computer Science Education, OOP, Programming Languages.**

INTRODUCTION

Bahçeşehir University has been teaching programming languages in the first year of Computer Engineering, Software Engineering, Industrial Engineering, Electrical and Electronical Engineering and Mechatronics Engineering. Despite the procedural approaches for beginners in the other departments, Computer Engineering (CE) and Software Engineering (SE) are the ones focusing not just on procedural but also on object oriented programming.

Our CE department teaches this first year course using C++, a hybrid language which derives from C, and our SE department uses Java.

Our SE department is a newer department (5 years) in comparison to the CE department (12 years), hence the SE graduates are newer in the industry, which gives us a chance to observe them more objectively.

During this paper, we will mention the experiences of the SE students during / after the school and our supporting points by giving examples by their pros and cons.

Manuscript received March 3, 2010.

Ç., Duygu is studying as a master's degree student in Computer Engineering and also working as a teaching assistant in Software Engineering, Bahçeşehir University, Besiktas, IST 34353 TURKEY (phone: 0090-212-381-0542; fax: 0090-212-381-0550; e-mail: duygu.cakir@bahcesehir.edu.tr).

K., Adem, Assoc. Prof. is the head of the Software Engineering Department of Bahçeşehir University, Engineering Department. (phone: 0090-212-381-0560; e-mail: akarahoca@bahcesehir.edu.tr).

K., Selvihan is a lecturer in the Computer Engineering Department at Bahçeşehir University. (phone: 0090-212-381-0586; e-mail: syavuzer@bahcesehir.edu.tr)

WHY JAVA?

As Kölling stated before, "for a first year student, the language should support clean, simple and well-defined concepts, it should have an easily readable and consistent syntax and also the language should have an easy-to-use development environment including a debugger so that the students can concentrate on learning programming concepts rather than the environment itself." [2, 5]

Java is an open-source language which enables the user to work in any platform (s)he wants. Its platform-free structure attracted us for we know that although most of the students use Windows platform, some still prefer Unix/Linux OS.

According to our student surveys, students like Java because they can easily implement games for mobile phones and the ability to use Java to create web applications (of course not during their freshman year, but the dream of implementing mobile and web applications attract them).

Besides these, we face with Java in our every day life. Decoders, printers, games, navigation systems, web cams, medical devices and parking machines also use Java coding.

Because Java is a clean, readable, platform-free, user-friendly, preferred-in-the-industry, object-oriented language, it became on top of our choice list. The reason our SE department sticked to it was that the students loved working with it, and made no negative criticism even though they were taught C++ and C# in their junior and sophomore years too and still preferred Java.

COURSE STRUCTURE

"Introduction to Programming with Java" and "Object Oriented Programming with Java" is given to SE freshmen. During the fall semester, we give the basics of Java and in the spring semester we teach OOP and its applications. We use Eclipse IDE (Classic) as the Java editor. [13]

Each semester, we make 2 midterms, 1 lab exam, 4 in-class or take home quizzes and a final exam. 70% attendance for theoretical and 80% attendance for lab sessions are required. Also, each semester, up to 10 points of bonus is given to encourage the students' efforts.

The questions of the midterms and the finals come from the basics. Generally, in the questions, all the keywords and pre-defined functions are given to the student, asking him to fill in the functions or blanks according to the problem by implementing his/her unique algorithm.

On the contrary to midterms and the finals, the open-note lab exam is a bit harder. The student is asked to implement the given program in Eclipse using his/her notes and send the

ISBN: 978-988-17012-9-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

source files to the lab TA. Usually the first semester's lab exam is about multidimensional arrays and complex algorithms (mostly games), and the second semester's lab exam contains inherited classes and functions about a daily life sample to stimulate them think of programming as a daily life activity.

A. *Fall Semester*

This semester focuses on algorithms, Java basics, how it works and how its compiled by the JVM, Eclipse environment and so on. The main topics are as follows:

- Algorithm representations with pseudo-code and flow chart
- Algorithm representations using control structures, repetition
- Anatomy of a simple Java program, Java byte codes, Java compiler and Java virtual machine (JVM), Java syntax.
- Basic variables, scope, variable assignment and arithmetic operators, running a Java program both using Eclipse IDE and using the command prompt.
- Logical operators, decision structures, if/else and switch/case blocks
- While, do/while and for loops
- String class and manipulation functions
- Predefined libraries and their functions, function definitions and parameters, function prototype
- Using the Math library and random number generation
- Creating, accessing and using arrays, basic sorting and searching algorithms

After we make sure the students really get these topics by making a review session, we start object orientation by using some basic, every day examples such as a Person or a Vehicle class at the end of the first semester.

B. *Spring Semester*

This semester fully focuses on object orientation topics, encapsulation, inheritance, abstract structures and interfaces. We first make a review of the first semester by writing complex problems for the first 2 weeks and then start OOP.

- Basics of classes, member variables, class methods, constructors
- Inheritance, polymorphism, class abstraction
- Exception handling
- File I/O
- Java swing components, graphical development
- Basic data structures, list implementations, dynamic allocation

As soon as we finish the topics, we make an introduction to data structures to help the students understand next year's topics. Then we give a graphical quiz (mostly a program in which the student makes a painting using the mouse and the keyboard) focusing on the class hierarchy and lists.

DISADVANTAGES OF JAVA

When the student starts writing his very first "Hello World", the first thing he sees is this:

```
public class FirstProgram
{
    public static void main(String[] args)
    {
        System.out.println("Hello World!");
    }
}
```

Figure 2 - Hello World

The first coding they do is extremely hard to understand for a starter using the keywords "public, class, static, void, String, String[], System, out, println"... Of all these keywords, the student just understands the "Hello World" part and gets a bit confused. But as he practices more and more, he, himself, understands the need for the class and the main function. Of course, Eclipse IDE has a big role in his understanding.

When it comes to get an input from the user (usually in the 4th week of fall semester), it gets even more complicated.

At the beginning, in our first year, we used the BufferedReader class to get inputs from the user, but then Scanner seemed easier for the students to understand.

```
import java.util.Scanner;

public class UserInput {

    public static void main(String[] args)
    {
        System.out.println("Please enter your age: ");

        Scanner reader = new Scanner(System.in);
        int age = reader.nextInt();

        System.out.println("WoW! Are you really " + age + "???");

        boolean answer = reader.nextBoolean();

        if(answer)
            System.out.println("You don't seem like " + age);
        else
            System.out.println("I never beleived you were " + age
    }
}
```

Figure 1 - User Input with java.util.Scanner

Besides it's simplicity, we can use the Scanner class everywhere, when getting an input from the user, when reading a text file, or when listening to a port for an input stream. But, no matter how easier it gets, it's always hard for a beginner student to understand all in one.

Although Java has it's disadvantages like these for a beginner, it is far more easier when the program gets more complex. After a while, the student gets used to the syntax and rules of Java.

COUNTER ATTACKS

As can be seen above, we used an inductive approach starting from the variable declarations to OOP. We believe that a healthy combination of theory and practice in balance is necessary for a student to pass on to OOP. They need a mature

ISBN: 978-988-17012-9-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

perspective and algorithmic intelligence before they start OOP.

Many disagreed with this approach before, such as Pedroni, Kölling, Rosenberg, Zhu and McLaughlin [11, 3, 4, 5, 12, 7]. Their mutual opinion was that OOP is best taught by teaching objects from the start, rather than starting with a procedural programming approach and adding objects later. Kölling stated that "If we want to teach object-orientation, we should do it first. The path to object-orientation through a procedural programming is unnecessarily complicated."[4] On the contrary, McLaughlin's statement wasn't as moderate as Kölling's: "object technology must be taught as a core theme in order to provide theoretical underplannings. Failure to do so will lead to the worst of all worlds – teaching a procedural approach with an OO language." [7]

We disagreed with the many…

WHAT WE BELIEVE

Even though there are many professors disagreeing, some of them still support the procedural approach. As a matter of fact, Lewis addressed the subject to a whole different topic and suggested that **object-first methodology is not a good *pedagogy* for teaching object orientation** [6].

We argue that if a student masters on the syntax and algorithms first, then (s)he can write more effective and optimized software. He should start writing without any questions in his head about the language and the syntax. That's why we teach the basics first, before everything gets more complicated.

After the student learns the basics, e.g. variables, loops and conditionals, he starts to search for different approaches. He, himself, asks us to lower the number of lines in the program and not repeat similar lines. After he spontaneously searches for a new approach, we start teaching arrays and functions.

When he sees that the lines can be reduced by getting rid of the repetitive ones, he has the need for reducing more and more. Our basic example just before starting object oriented coding is as follows:

```
public class People
{
    public static void main(String[] args)
    {
        String[] names = new String[10];
        String[] surnames = new String[10];
        long[] id = new long[10];
        int[] ages = new int[10];
        double[] weights = new double[10];
        double[] heights = new double[10];
    }
}
```

Figure 3 - Before Transmitting to OOP

As can be seen from the above example, after the student sees that there are so many arrays to store 10 people with 6 properties in the system, he seeks for an easier way to do this and requests a hand on making this simpler. Hence, there are no questions left in his mind about the need for object orientation and everything becomes clear in his head.

More than 80% of the students in their first year start the university as soon as they finish high school. This is a huge fact that their mathematical intelligence is stil fresh. If we choke them with more rules using the object-first approach and make them confront lots of new keywords, the students' interest and appreciation decrease more and more in time. Instead of this approach, we try to teach them how to solve problems using their fresh mathematical intelligence, and then gradually make them memorize more keywords. This way, they barely loose their attention and interest on the subject. This also helps them think more modularly.

70% of the freshmen have seen programming before and at least 50% have tried writing short programs before they came to the university. Therefore most have an idea on what's going on in the course. This is an advantage to progress faster.

Another reason we believe the inductive procedural approach is better than all is that when the student starts with an object-first approach, because of the so many rules about the syntax, classes, encapsulation, modularity and all, he can not concentrate on the indentation, the discipline and the naming. Besides teaching him how to program and use his intelligence, we should give him the discipline on the **"unwritten coding rules."** To change one's habits is hard, but to make him gain a new habit is easier. That's why, we choose to start from the beginning.

AFTERWARDS…

The industry usually complains about the newly graduates being so unaware of the rules of coding undisciplined in writing a program. Their most common grievance is that what they write is un-readable and unable to interfere. The inductive procedural approach helps them earn these abilities and coding disciplines.

As Meyer suggests, we, too, believe that if the basis of functional and procedural programming is given powerfully with lots of practice and time, the students, then, may be able to involve in other object oriented languages and pre-written programs easier [9]. Again, during their studentship with other courses such as Data Structures or Computer Graphics and also in the industry, the students cannot succeed quickly if their algorithmic bases aren't well. They can learn any language they want, write any program they need. But the main problem with them is that they don't learn how to think algorithmic through their studentship if they start with the object-first methodology. A procedural-first kid's vision to the problem won't be the same with an object-first kid's vision. The first will try to find a more optimized, modular and quick-working solution whereas the other will just solve it, no matter how slow or complicated the code works.

The most important of all is that our only aim is to raise farsighted programmers. Before we teach them how to implement a code, we first make them think without choking them with extra knowledge. It isn't nice for a programmer to correct his mistakes and bugs all the time. Of course he will make many, but at least before starting to write, if he thinks a bit more, we believe he can eliminate most of the mistakes and exceptions, i.e. handle compilation and runtime errors before they occur. A inductively-raised student will have the chance to think before writing, while the object-first student will not, because he is a student trying to memorize things he has never seen before. This "think before act" thing is a discipline only can be taught when the student is on his early stages of coding.

STUDENT FEEDBACKS AND EVALUATIONS

During our 5 years of experience, 469 students have taken the course (1st and 2nd semesters' average) and only 1 of them

ISBN: 978-988-17012-9-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

withdrew. Over these 5 years, after every semester the university asked the students to fill an evaluation form for every course they had taken. Usually, more than 50% of the students filled these forms with junk, ignorable data whereas the others typed their commands on how they dealed with the course, the lecturer and the TA.

According to these student evaluations, none of them found the course content hard to keep up with. They have problems getting on with the lecturer or the TAs, they have complaints about the number of quizzes but none of them has a negative opinion about the course structure. As we mentioned before, most of them had developed programs on their own. What they say is that our system made it easy for them to improve their coding skills.

After talking to 10 of our best coders, they all agree that the structure of the course is better like this and that they wouldn't have followed the lesson if it was given with an object-first approach. Besides following our notes and labs, they used books (especially Deitel's) [14] and searched for online content. Almost everything they found was written in an object-first manner. According to what they experienced, it was hard for a beginner or an intermediate programmer to learn OOP before they get the basics.

CONCLUSION

We teach the students to always spend more time in project planning in our Software Project Management courses. An object-first approach doesn't let the student to plan how to write a better program, it just teaches how to write one, no matter how. (S)he spends time in learning the syntax rather than thinking on the problem and preparing a more optimized solution.

On the other hand, an inductive procedural approach to OOP helps them memorize the syntax and language rules gradually and spend more time on algorithms. As Lewis suggested, their pedagogy doesn't let them learn everything at once. What we aim is to make use of their mathematical intelligence before they choke up with the rules by repeating the syntax and solving more complicated problems. There is a saying that a tree only bends when it's still sappy. They gain discipline by progressing slowly to OOP when they are still in their learning phase, which helps them hold on to their jobs after they graduate. Looking at the results, every semester more than 85% students in 5 years' average being successful is enough as a proof to inductive, procedural approach's accomplishment.

REFERENCES

[1] Franca, P. B., Software Engineering Education The Shift to Object Oriented Programming, *IEEE*

[2] Goldwasser, M. H., Letcher, D., Teaching an Object-Oriented CS1 – With Python, *ITiCSE '08*

[3] Kölling, M., Koch, B., Rosenberg, J., Requirements for a First Year Object-Oriented Teaching Language, *SIGCSE '95*

[4] Kölling, M., The Problem of Teaching Object-Oriented Programming, Part 1: Languages, *Journal of Object-Oriented Programming*, 11(8): 8-15

[5] Kölling, M., Rosenberg, J., Guidelines for Teaching Object Orientation with Java, *ITiCSE '01*

[6] Lewis, J., Myths About Object-Orientation and its Pedagogy, *SIGCSE '00*

[7] McLaughlin, P., Oh by the way Java is Object Oriented, Monitor 8, *Proceedings of the 1st 'Java in the Computing Curriculum' Conference*

[8] McKim, Jr. J. C., Ellis, H. J. C., Using a Multiple Term Project to Teach Object Oriented Programming and Design, *CSEET '04*

[9] Meyer, B., Towards an Object Oriented Curriculum, *Prentice Hall*

[10] Northrop, L. M., Finding an Educational Perspective for Object-Oriented Development, *OOPSLA '92*

[11] Pedroni, M., Meyer, B., The Inverted Curriculum in Practice, *SIGCSE '06*

[12] Zhu, H., Zhou, M, Methodology First and Language Second: A Way to Teach Object-Oriented Programming, *OOPSLA '03*

[13] eclipse.org, Eclipse Classic 3.5.2

[14] Java – How to Program, 6th ed., *Deitel*

ISBN: 978-988-17012-9-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)