Programming in Java: Student-Constructed Rules

Ann E. Fleury
Aurora University
afleury@admin.aurora.edu

Abstract

Java is becoming a popular first programming language for university students. One reason for its popularity is its power as an object-oriented language. This study examined beginning students' understanding of the construction and use of objects in Java. During tape-recorded interviews, students were asked to predict which programs from a collection of similar programs would work according to specification and which would not. This paper will discuss those interviews, including the most common false assumptions or "student-constructed rules" invoked by the students and the implications of the interviews for instruction.

1 Introduction

Java is becoming a popular first programming language for university students. Its advocates claim that, "It takes the best ideas developed over the past 10 years and incorporates them into one powerful and highly supported language." [16, p.41]. Now instructors need more information about the difficulties that students are likely to encounter when learning Java. This study investigated some of those difficulties.

2 Background

2.1 Previous Research

Educational researchers have studied beginning programming students learning procedural programming. Student difficulties with fundamental concepts such as input/output, assignment, and control structures have been well documented [5, 12, 14]. Students' false assumptions about scope and parameter passing in Pascal have been investigated [6, 11]. Students' difficulties learning recursion have been extensively examined [8].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advant -age and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SIGCSE 2000 3/00 Austin, TX, USA
© 2000 ACM 1-58113-213-1/00/0003...$5.00

Many research studies have focused on the design of object-oriented programs by professional programmers [4]. Only a few research studies have centered on college students learning object-oriented programming. Holland, Griffiths, and Woodman [9] identified some misconceptions of students taking a distance education course in Smalltalk. Ramalingam and Wiedenbeck [13] studied program comprehension of student C++ programmers working with small programs. Few researchers have studied beginning Java students [7].

2.2 Constructivism

This study assumed a constructivist theoretical framework. Constructivism assumes that people create new knowledge by combining new experiences with current knowledge and by reflecting on their knowledge. It assumes that "all learning involves the interpretation of phenomena, situations and events, including classroom instruction, through the perspective of the learner's existing knowledge." [15, p.116]. Constructivism serves as an appropriate theoretical basis for discussion of issues in computer science education [1].

3 Research Question

The research question for this study asked how beginning programming students construct an understanding of the syntactical and semantic rules involving the construction and use of objects in Java. This study was part of a more thorough study examining how beginning programming students experience object-oriented programming in Java.

4 Method

4.1 Participants and Course

The participants in this study were 28 volunteers from an introductory Java programming course at a large state university. This was the second semester that the course had been taught in Java. The text used was by Lewis and Loftus [10]. Although the course did not have a programming prerequisite, about three quarters of the students had had some previous programming experience, most commonly in BASIC, Pascal, or C++. However, the students with C++ background had taken a mini-course that did not include object-oriented programming. Thus, the concept of objects in Java was new to the students. I was not involved in designing or teaching the Java course.

4.2 Interviews

I interviewed each student participant during the last weeks of the semester. The interviews were structured around a set of programs based on a working Java program, called the "original" program in this paper. After pointing out the differences between the original program and a program variation, I asked the students to predict whether the variation would run and generate the same output as the original program. More importantly, I asked the students to explain in detail the reasons for their predictions. Finally, I asked the students the main sources of their information about Java. All interviews were audio-taped. I had successfully used a similar interview format previously with students learning Pascal [6] and with a smaller group of students learning Java [7].

4.3 Programs

The original Java program was a timer simulation program based on a C++ program by Decker and Hirshfield [3]. It was conceptually simple, but illustrated the power of object-oriented programming by having more than one class declaration and more than one object of one of the classes. It was just complicated enough for my needs without introducing unnecessary distracting features. In this paper I will discuss four variations of this original program used in the interviews. (See the Appendix for a copy of the code for the original program.)

To create program NAM, the original program was modified so that two method names of one class were made identical to two method names of another class. That was a legal change. However, I hypothesized that some students may not fully understand the concept of encapsulation and may attribute more importance to the choice of method names than does the Java compiler.

To create program SET, a constructor was removed from one class of the original program and was replaced by a method to initialize instance variables. This method was called without having allocated space for the objects. That was not a legal change. However I hypothesized that students may not clearly distinguish the steps of allocating space for an object and initializing the instance variables of an object. Also, because few languages require the programmer to allocate space for all objects, previous programming experience would be of little help here.

To create program CON, the original program was modified by including parameters in the constructors of two classes. The main method passed initial values to the Timer constructor. The Timer constructor invoked the NumericDisplay constructor twice, each time passing on one of these initial values and a limit value. That was a legal change. However, I hypothesized that beginning programming students may find parameters confusing.

To create program PUB, the original program was modified by making two data fields of one class public. Also one of these public data fields was accessed directly by an object of another class. That was a legal change. However, I hypothesized that some students may not clearly understand the full impact of public and private designations.

5 Results

The student responses spanned the full range from none correct to four correct. The median number of correct responses was three. This section will describe the students' responses to each of the program variations. Then it will summarize the students' responses regarding the most important sources of their Java knowledge. The students' incorrect assumptions will be called "student-constructed rules."

5.1 Student-Constructed Rule 1

Program variation NAM had matching method names in two different classes. Many students correctly explained that when one of these method names was used, the class of the object involved would determine which of the same-named methods was invoked. However, almost one third of the students provided explanations why they did not think this variation would work. The most common erroneous explanation invoked Student-Constructed Rule 1: The Java compiler can distinguish between same-named methods only if they have differences in their parameter lists.

5.2 Student-Constructed Rule 2

Program variation SET had no constructor for the NumericDisplay class, but had a method named set_values that initialized the instance variables of the class. Many students correctly explained that no memory space was being allocated and no object was being created. However, over one-third of the students thought that this variation would work. Most of these students invoked Student-Constructed Rule 2: The only purpose of invoking a constructor is to initialize the instance variables of an object.

5.3 Student-Constructed Rule 3

Program variation CON correctly included two integer parameters in the Timer constructor and two integer parameters in the NumericDisplay constructor. One formal parameter of Timer was used as an actual parameter for NumericDisplay. Most of the students, some of them after using scratch paper, correctly decided that this variation would work. However, many students who disagreed invoked Student-Constructed Rule 3: Numbers or numeric constants are the only appropriate actual parameters corresponding to integer formal parameters.

5.4 Student-Constructed Rule 4

Program variation PUB had public instance variables for one class. It had no accessor method for one of these public variables and accessed the variable directly from another class. Many students correctly explained that this variation would work. Most of the students who disagreed invoked Student-Constructed Rule 4: The dot operator can only be applied to methods.

5.5 Sources of Student Knowledge

The students most often cited experience with programs as the most important source of their knowledge of Java. Lecture and discussion sections were a close second. The text was a distant third.

6 Discussion

The concept of student construction of knowledge was implicitly acknowledged by the students when they emphasized programming assignments as a main source of their Java knowledge. Because students construct their own meanings during instruction, it is not surprising that students possess only partial conceptions even when provided with complete and accurate information. This section will examine the probable source of each of the student-constructed rules.

6.1 Student-Constructed Rule 1

The Java compiler can, indeed, distinguish between same-named methods with differences in their parameter lists. Student convictions that such different signatures are necessary to distinguish two methods having the same name are quite reasonable. This is the only way to distinguish methods with the same name that belong to the same class. It is not surprising that some students extended the principle to methods of different classes.

6.2 Student-Constructed Rule 2

One purpose of invoking a constructor is, indeed, to initialize the instance variables of an object. The requirement that programmers allocate space for objects, but not for primitive variables, is an aspect of Java that is different from most other programming languages. Thus it is not surprising that even students with programming background in other languages found it difficult to understand. As constructivism would predict, the student explanations emphasized what the students could easily observe, namely initialization of variables, over that which they could not so easily observe, namely allocation of memory space.

6.3 Student-Constructed Rule 3

Numbers and numeric constants are, indeed, appropriate actual parameters corresponding to integer formal parameters. Numbers have been clearly understood by the students for years. Thus it is not surprising that students understood using numbers as actual parameters better than they understood using formal parameters as actual parameters.

6.4 Student-Constructed Rule 4

The dot operator can, indeed, be applied to methods. Because the dot operator is not needed with private instance variables, the student assumption that the dot operator works only with methods is reasonable. Students can successfully write their programming assignments without understanding the full power of the dot operator.

6.5 Summary

In summary, it is interesting to note that the removal of "only" or "the only" from each of the student constructed rules makes it a true statement. The students have constructed incorrect rules by misapplying correct rules. This is consistent with previous research on student misconceptions in science and mathematics. "Most, if not all, commonly reported misconceptions represent knowledge that is functional but has been extended beyond its productive range of application." [15, p.147].

7 Implications for Instruction

Researchers who have studied students programming in other languages have made recommendations that are also useful when teaching Java programming. Emphasizing reading and debugging activities as well as programming activities can be useful in bringing partial conceptions to light [12]. Presenting the student with a model of the machine can be useful when explaining some of the troublesome hidden side-effects of program statements [5]. By constructing sample programs with care, an instructor can avoid inadvertently fostering common misconceptions [9].

Some students made specific, unsolicited, suggestions for instruction. One student suggested providing a workbook having very simple problems with answers for students to work before tackling more difficult sample programs and programming assignments. Another student suggested the use of transparencies and a text that color-coded the program listings to differentiate Java statement components. Several students said that they were most successful in the course when they had time to "play" with the programs. Several students mentioned the advantages of having individual help available.

Note that all of these student suggestions are consistent with constructivism. The workbook and the color-coding of program listings are intended to strengthen students' prior knowledge before they attempt to decipher complicated programs. Playing with programs allows students to ask and answer the questions that are relevant to their current levels of understanding. One-on-one assistance provides help tuned to students' current levels of understanding.

Some suggestions for instruction were indirectly suggested by the students' responses. Several students were much more concerned with whether their true-false predictions were correct than with whether their explanations were correct. Providing more opportunities for students to explain their reasoning, both in discussions and on quizzes and examinations can only help. Students' should be encouraged to set goals that involve understanding, not just successful masking of confusion.

Constructivism suggests that designing appropriate laboratory experiences for the students is likely to be more effective than just talking to them. "The goal of instruction should be not to exchange misconceptions for expert concepts but to provide the experiential basis for complex and gradual processes of conceptual change." [15, p.154]. Structured laboratory activities involving a collection of program variations could help to provide the basis for such change.

8 Limitations

This study necessarily has some limitations on the generalizability of its results. Only 28 students participated in the study; the participating students were volunteers rather than being randomly selected. This paper only discussed students' understanding of a few syntactical and semantic rules involving objects in Java. However, similar student interviews were also conducted regarding the stylistic conventions of Java programming and the transcriptions from those interviews are currently being analyzed.

9 Conclusion

Being aware of common student difficulties and misconceptions is useful for instructors. However, being aware of how much they can learn from listening to students under nonthreatening circumstances is even more useful for them. "Teaching is most effectively improved when the teacher learns to listen to students' thoughts and to interpret students' actions and thoughts from their perspective." [2, p.8]. These interviews were an attempt to listen to Java students and to remind instructors of the benefits of such listening.

References

[1] Ben-Ari, M. Constructivism in Computer Science Education. *SIGCSE Bulletin 30, 1* (1998), 257-261.

[2] Confrey, J. A Review of the Research on Student Conceptions in Mathematics, Science, and Programming. In C. Cazden, Ed., *Review of Research in Education*, American Educational Research Association, Washington, D.C., 1990, 3-56.

[3] Decker, R. and Hirshfield, S. *The Object Concept: An Introduction to Computer Programming Using C++*. PWS Publishing, Boston, 1995, 25-29.

[4] Detienne, F. and Rist, R. Introduction to this Special Issue on Empirical Studies of Object-Oriented Design. *Human-Computer Interaction 10*, 2&3 (1995), 122-128.

[5] duBouley, B. Some Difficulties of Learning to Program. *Journal of Educational Computing Research 2*, 1 (1986), 57-73.

[6] Fleury, A. Parameter Passing: The Rules the Students Construct. *SIGCSE Bulletin 23*, 1 (1991), 283-286.

[7] Fleury, A. Student Conceptions of Object-Oriented Programming in Java. *Journal of Computing in Small Colleges 15,1 (1999), 69-78.*

[8] Ginat, D. and Shifroni, E. Teaching Recursion in a Procedural Environment: How Much Should We Emphasize the Computing Model? *SIGCSE Bulletin 31*, 1 (1999), 127-131.

[9] Holland, S., Griffiths, R. and Woodman, M. Avoiding Object Misconceptions. *SIGCSE Bulletin 29*, 1 (1997), 131-134.

[10] Lewis, J. and Loftus, W. *Java Software Solutions: Foundations of Program Design*. Addison-Wesley, Reading, MA, 1998.

[11] Madison, S. and Gifford, J. Teaching Pascal Parameters: Revealing a Novice Misconception. *Proceedings of the National Educational Computing Conference* (1996), 244-246.

[12] Pea, R. Language-Independent Conceptual "Bugs" in Novice Programming. *Journal of Educational Computing Research 2*, 1 (1986), 25-35.

[13] Ramalingam, V. and Wiedenbeck, S. An Empirical Study of Novice Program Comprehension in the Imperative and Object-Oriented Styles. In S. Wiedenbeck and J. Scholtz, Eds., *Empirical Studies of Programmers: Seventh Workshop*. ACM Press, New York, 1997, 124-139.

[14] Sleeman, D., Putnam, R., Baxter, J. and Kuspa, L. Pascal and High School Students: A Study of Errors. *Journal of Educational Computing Research 2*, 1 (1986), 5-23.

[15] Smith, J., diSessa, A. and Roschelle, J. Misconceptions Reconceived: A Constructivist Analysis of Knowledge in Transition. *The Journal of the Learning Sciences 3*, 2 (1993), 115-163.

[16] Tyma, P. Why Are We Using Java Again? *Communications of the ACM 41*, 6 (1998), 38-42.

Appendix

```
// CLASS ORIGINAL
// Original main method to test the use of a timer

class Original {
    private static final int MAX = 100;
    private static Timer clock;

    public static void main (String[] args) {
        clock = new Timer();
        int count = 1;
        while (count <= MAX) {
            clock.increment_time();
            clock.show_time();
            System.out.println();
            count = count + 1;
        } // while
    } // method main

} // class Original

// CLASS TIMER
// A timer is made up of two numeric displays,
// one to hold minutes and one to hold seconds.
// A timer can increment the time by one second,
// and can show the time on the screen.

class Timer {
    private NumericDisplay minutes;
    private NumericDisplay seconds;

    // This constructor instantiates and initializes
    // the minutes and seconds displays of the timer.
    public Timer () {
        minutes = new NumericDisplay ();
        seconds = new NumericDisplay ();
    } // constructor Timer

    // Increment the timer by one second
    public void increment_time () {
        seconds.increment_value();
        if (seconds.get_value() == 0)
          minutes.increment_value();
    } // method increment_time

    // Show the time on the screen
    public void show_time () {
        minutes.show_value();
        System.out.print (':');
        seconds.show_value();
    } // method show_time

} // class Timer
```

```
// CLASS NUMERICDISPLAY
// A numeric display stores a value
// and an upper limit for that value.
// The value will stay between 0 and limit - 1.
// A numeric display can increment its value,
// return its value, and show its value on the screen.

class NumericDisplay {
    private int value;
            // Current value of the numeric display
    private int limit;
            // The value will stay between 0 and limit - 1.

    // Constructor initializes the value and the limit.
    public NumericDisplay () {
        value = 0;
        limit = 60;
    } // constructor NumericDisplay

    // Increase the value by one.
    // The value circles back to 0 so it can never reach limit.
    public void increment_value() {
        value = value + 1;
        if (value == limit)
          value = 0;
    } // method increment_value

    // Gets the value for use of the caller
    public int get_value () {
        return value;
    } // method get_value

    // Shows the value on the screen as a two digit number
    public void show_value () {
        if (value < 10)
          System.out.print ('0');
        System.out.print (value);
    } // method show_value

} // class NumericDisplay
```