A Study of Misconceptions and Missing Conceptions of Novice Java Programmers

Chiu-Liang Chen[1], Shun-Yin Cheng[2], and Janet Mei-Chuen Lin[3]
[1]Department of Information Management, National Taipei College of Business, Taipei, Taiwan
[2, 3]Graduate Institute of Information and Computer Education, National Taiwan Normal University, Taipei, Taiwan

ABSTRACT - *To enhance learning effectiveness in object-oriented programming courses, many researchers have tried to identify difficulties faced by students and the misconceptions they may harbor. As a further attempt along this line, we conducted one-on-one clinical interviews with 22 freshmen who were taking their first Java programming courses. The objective was to investigate not only which programming concepts/constructs most students had difficulties with, but why they found them difficult. We focused on fundamental OO concepts and Java programming constructs such as: (1) classes vs. objects, (2) static data members vs. constant data members, (3) constructors, (4) access modifiers, (5) syntax for method calls (e.g., for calling static vs. non-static methods, for calling methods in other classes vs. within the same class), (6) parameter passing, (7) method overloading, (8) inheritance, (9) polymorphism, (10) for-each vs. for loops, and (11) the use of standard Java libraries. Students' misconceptions and missing conceptions in each of these concepts/constructs are described in detail in this paper.*

Keywords: Java Programming, Missing Conceptions, Misconceptions, Greenfoot, Clinical Interview.

1. INTRODUCTION

Java programming is a required course for many computer science majors, but it is often found difficult by students. To better understand the difficulties encountered by novice Java programmers, many studies have tried to identify their misconceptions [3, 6, 11]. Whether Java instructors should adopt an object-early or object-late approach has also been widely discussed [e.g., 2, 11]. To support the object-early approach so that students can engage in learning OO concepts from the beginning, various pedagogical tools, such as DrJava [1], BlueJ [9], and Greenfoot [7], have been developed.

One common approach to identifying difficulties or misconceptions that students may have typically asks what they do or do not know. However, as Perkins and Martin [10] have pointed out, it is inappropriate to oversimplify knowledge as simply knowing or not knowing something. Students may have *fragile* knowledge about a specific programming concept, which results in programming errors rather than not knowing anything about it at all. Students' fragile knowledge of BASIC statements was then analyzed by Perkins and Martin [10].

One-on-one clinical interviews were conducted in this study to probe students' understating of Java programming constructs and fundamental object-oriented concepts. Students were observed closely as they solved a set of Java programming problems, each centered on a specific programming concept/construct. For example, a problem may require students to modify a Java program by declaring some of the data member(s) of a class to be *static*. Through close observation and interviews we intended to determine different levels of students' understanding about a concept/construct. For example, does a student know what a data member is? Can s/he describe the properties of a *static* data member? Can s/he correctly identify which data member(s) should be made *static* in a specific problem?

The remainder of this paper is divided into four sections. Section 2 discusses related work; Section 3 describes the research method and procedure; Section 4 presents our findings; and Section 5 is the conclusion.

2. RELATED WORK

2.1 Misconceptions of Novice Java Programmers

Eckerdal and Thuné [3] performed a phenomenographic analysis to identify different understandings of object-oriented concepts exhibited by students. They then applied variation theory to the results to pinpoint what students need to be able to discern in order to gain a "rich" understanding of those concepts. For the purpose of building a concept inventory of programming fundamentals, Kaczmarczyk, Petrick, East and Herman investigated students' misconceptions in a series of core CS1 topics [6]. Their investigation revealed four themes, with each theme containing several misconceptions. Three of those misconceptions were described in detail, including two misconceptions about memory models, and a

misconception involving data assignment when primitives were declared. Ragonis and Ben-Ari [11] investigated difficulties encountered and conceptions built by novice object-oriented programmers. Their study identified 58 conceptions and difficulties, which were grouped into four primary categories: class vs. object, instantiation and constructors, simple vs. composed classes, and program flow.

2.2 The Object-Early Approach and the Greenfoot Programming Environment

Many previous studies have focused on how to teach introductory programming courses using Java, in particular whether an object-early approach should be adopted [e.g., 2, 11]. As opposed to an object-late approach, an object-early approach introduces classes and objects very early in the course and emphasizes core OO concepts from the beginning. Bruce [2] claimed that using an object-late approach to teach Java would do a disservice to students because students would learn a programming style that does not fit the language itself. He further suggested that instructors should use pedagogical tools such as BlueJ and DrJava to support object-first instruction. After teaching Java to high school students for an academic year, Ragonis and Ben-Ari also concluded that classes and objects should be introduced first, but they suggested using diagrams to enhance students' understanding of OO concepts [11].

The Greenfoot programming environment used in this study is yet another pedagogical environment for Java, like BlueJ and DrJava. In fact, it was created by the same research team that developed BlueJ. The design goals of Greenfoot were to make programming engaging, creative and satisfying for the students, and to actively help teachers in teaching important, universal programming concepts [7]. With Greenfoot, students can easily visualize important object-oriented concepts.

2.3 The Clinical Interview Research Method

Clinical interviews were originally used by Jean Piaget to reveal children's thinking processes [4]. Perkins & Martin employed the method to investigate students' difficulties in learning BASIC programming [10]. In their clinical interviews, an experimenter interacted with students as they worked, systematically providing help as needed progressing from general strategic *prompts* to specific *provides*. Based on the data collected from clinical interviews, students' fragile knowledge was classified into four types: (1) missing knowledge: knowledge that a student had either forgotten or never learned; (2) inert knowledge: knowledge that a student failed to retrieve but in fact possessed; (3) misplaced knowledge: knowledge suitable for some roles invaded occasions where it did not fit; and (4) conglomerated knowledge: students produced code that jammed together several disparate elements in a syntactically or semantically anomalous way.

3. METHOD

3.1 Procedure

A total of 84 students enrolled in two introductory Java programming courses offered by the Department of Information Management at National Taipei College of Business in Fall semester of 2010, which lasted 18 weeks from September 2010 to January 2011. There were three 50-minute class periods per week for each course. Students were offered a mixture of lectures and hands-on exercises that took place in a computer lab. Both courses were taught by the first author of this paper.

The midterm exam, which included a written test and a programming test, was administered in the 9th week, whereas the written test and the programming test of the final exam were given in the 16th and the 17th week respectively. Based on students' test scores, the instructor selected 22 low-achievers to participate in the one-on-one clinical interviews, which were conducted in the 18th week. The first two authors served as the interviewers.

During each interview session, the interviewee was closely observed as s/he worked on a set of assigned problems. S/he was allowed to take as much time as needed to solve each problem. Whenever the interviewee encountered an impasse and did not know how to proceed, the interviewer would intervene by providing a predetermined series of guidance, ranging progressively from asking questions, to providing general strategic prompts, and finally to giving specific syntax for a statement. Each interview session lasted from 45 minutes to 2 hours, depending on how much time a student needed to solve the assigned problems.

The content taught in the two courses was the same. It was adapted from [8] and covered three major parts:

- The Greenfoot programming environment: the class diagram for examining the classes used in the scenario and the inheritance relationship among them, the Act button and the Run button for execution control, the Greenfoot API, and how objects can be created and methods invoked interactively from the pop-up menu.

- Basic programming constructs: variables and constants, arithmetic, relational and logical operators, conditional statements, loops, arrays, method invocation, and packages.

- Object-oriented concepts: classes and objects, access modifiers, constructors, static class members, inheritance, polymorphism, method overloading, the super keyword,

the this keyword, collections vs. the for each statement, and Java standard class library.

3.2 Data Collection

The data-collection instruments used in this study included the programming problems for students to solve during one-on-one clinical interviews, the guiding questions given by the interviewer to the students for each programming problem, and a screen-capture software tool. Detailed contents of these instruments are elaborated below.

3.2.1 The Programming Questions

The scenario used in this study was a simulation of number representation in binary using cards. It was derived from the Binary Numbers scenario created by Lenton and Brown [5]. Figure 1 and figure 2 show the main window and the class diagram of our revised Binary Number scenario respectively. As shown in Fig. 1, there are 5 cards on upper side of the scenario's world. Each card has two states: either shown or not shown. When a card is shown, its face up image is displayed and the binary digit direct below it should be 1. On the other hand, when a card is not shown, its back image is displayed and the binary digit right below it should be 0. Theses cards share a common back image, but each has its own unique face up image. The 5 cards starting from left to right contain 2^4, 2^3, 2^2, 2^1, 2^0 spades respectively on their face up images, and the number of spades on each card's face up image represents the associated value of that card. Total value of the face-up cards is shown to the right of the equal sign. For example, the total value of the three shown cards in Fig. 1 is 19, and its binary representation is 10011.

When the scenario is running, the user can click any card or binary digit to flip it. When a card switches its state, the associated binary digit changes correspondingly (and vice versa). Simultaneously, the total value for the new set of shown cards updates immediately.

The students were provided with a partially implemented version of our revised Binary Number scenario and required to work on 9 programming questions for the clinical interview. The questions as well as their related programming concepts are summarized in Table 1. As shown in Table 1, only question 6 asks for adding new functionality to the existing scenario, the other 8 questions ask for modifying code to more properly implement the scenario. For example, in question 1, students had to find out one data member in PokerCard/BNCard class that could be shared among all PokerCard/BNCard objects and declare it as static. Figure 3 shows a code segment of the PokerCard class, since all PokerCard objects share the same back image, the backImg data member (see line 6) should be declared as:

```
private static String backImg="card_back.png";
```

3.2.2 Guiding questions

The guidance series for all programming questions was proposed by the interviewers in advance. For each particular programming question, guidance was planned according to the sub goals (Table 1) that students should achieve for answering that question. There were two types of guidance: questioning and giving explanations. When students encountered problems, the interviewer first questioned to make sure if they understood related concepts. If they didn't, the interviewer then explained those concepts to the students. For example, guidance for the first sub goal of programming question 1 was:

Can you tell me where the data members are?

If the student had no idea how to answer the question, the interviewer then gave some further explanations, such as:

Generally speaking, a class is composed of method members and data members, and these are method members. Can you find out where the data members are?

If the student still had no idea, the interviewer then gave the exact answer:

These are the data members.

Figure 1. Main window of our revised Binary Number scenario for this study.

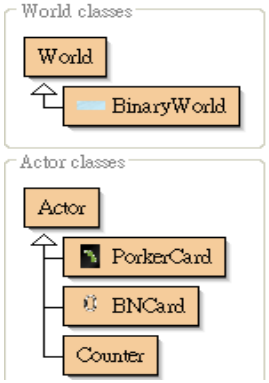

Figure 2. Class diagram of the partially implemented version of our revised Binary Number scenario

Table 1. Programming questions for the clinical interview.

Programming questions	Related Concepts	Sub goals for each question
Q1: Find a data member that is proper for declaring as static from the PokerCard or the BNCard class and declare it as static.	Static data members.	● Figure out where the data members are. ● Figure out properties of a static data member. ● Determine which data member is appropriate for declaring as static. ● Declare a data member as static.
Q2: Find a data member that is proper for declaring as constant from the PokerCard or the BNCard class and declare it as constant.	Constant data members.	● Figure out properties of a constant data member. ● Determine which data member is appropriate for declaring as constant. ● Declare a data member as constant.
Q3: The BNCard class already contains a constructor with two parameters in its method signature as follows: (int value, boolean isShown) ● Define another constructor with one parameter: (int value) ● Call the original constructor from the newly defined constructor. ● Modify the BinaryWorld class so that when instantiating the BNCard objects, the newly defined constructor will be invoked automatically instead of the original constructor.	● Properties of a constructor. ● Method overloading.	● Define a new constructor in a class. ● Call the original constructor from the newly defined constructor by using the this keyword. ● Modify the way of an argument passing in new BNCard statements of the BirnaryWorld class
Q4: Modify the BinaryWorld class: (1) Define a new method named populateWorld. (2) Remove the code defined in the constructor for instantiating all actor objects to the newly defined method. (3) Call the populateWorld method within the constructor.	Divide the program task into different modules.	● Define a new method. ● Identify related statements existing in the BinaryWorld class constructor and remove them into this newly defined method. ● Call this newly defined method from the BinaryWorld class constructor.
Q5: Modify the Counter class: When the space key is pressed: (1) Switch all cards' states to not shown. (2) Set all binary digits to 0. (3) Reset the total to 0.	● Get a collection (set of specific objects) ● Use for-each loop to process this collection.	● Determine which method this functionality should be defined in. ● Use if-else conditional statement to determine if the space key is pressed. ● Import a class from the Java class library. ● Read all PokerCard Objects. ● Use for-each loop to process all elements of a collection.
Q6: Apply inheritance concept to simplify code defined in both PokerCard class and BNCard class: (1) Define a class named Card that extends the Actor class. (2) Remove appropriate class members from both PokerCard class and BNCard class to the Card class. (3) Let both PokerCard class and BNCard class inherit the Card class.	● Apply the inheritance concept to promote code reuse. ● Define subclasses.	● Define a direct subclass of the Actor class named Card. ● Identify common functionalities among both PokerCard class and BNCard class. ● Define those common functionalities in Card class instead of in both PokerCard class and BNCard class. ● Let both PokerCard class and BNCard class extend the Card class.
Q7: Modify the BinaryWorld class: use only one superclass variable to reference two subclass objects successively.	Apply the polymorphism concept to promote program extensibility.	● Figure out meaning of the polymorphism concept. ● Declare a Card variable to substitute for both PokerCard variable and BNCard variable. ● Use the Card variable to reference PokerCard object and BNCard object successively.
Q8: Find a method that is proper for declaring as private from the PokerCard or BNCard class and declare it as private.	● Data hiding ● Access modifiers of class members	● Figure out what the access modifiers are. ● Figure out differences between public methods and private methods. ● Find out the exact method that could be declared as private.
Q9: The addObjects has been repeatedly called for 5 times, try to simplify the code by placing these method calls in a for loop or a while loop.	Use the loop constructs.	● Figure out syntax of while loop or for loop. ● Figure out how the value of the loop index can be used as arguments in the addObjects method call to determine objects' positions in the scenario world.

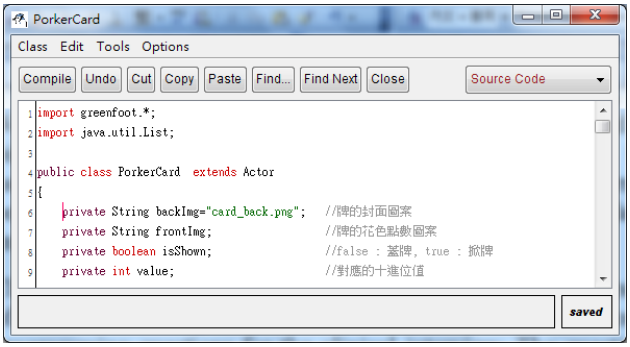

Figure 3. Code segment of the PokerCard class.

3.2.3 The screen-capture software package

A commercially available screen-capture software package and a microphone were installed to capture students' programming processes as well as all discussions between student and the experimenter. Both the programming processes and the audio data were transcribed for further analysis.

4. RESULTS

4.1 Misconceptions harbored by students

This study has unveiled the following misconceptions harbored by many students:

1. Static data members vs. constant data members: Confused the properties of static data members with constant data members (3/22, 14%).

2. Classes vs. Objects: Failed to recognize that classes were actually user-defined data types that could be used to define variables just like built-in data types such as integer and Boolean (8/22, 36%).

3. Constructors: (1) Erroneously thought that constructors of all classes defined in a project were invoked automatically when the project was opened (3/22, 14%); (2) Defined the constructor(s) of a class as its private member(s) when they should be public members instead (8/17, 47%). (3) Did not realize that constructors, as a special type of method, did not require a return type (11/22, 50%).

4. Arguments passing for method calls: Confused formal parameters with actual parameters (7/22, 31%). This is demonstrated by the code segment shown in Figure 4: code at line 16 is right for calling the constructor located at line 11 to line 14 while code at line 17 is not.

```
11  public PorkerCard(int value, boolean isShown)
12  {
13
14  }
15  public PorkerCard(int value) {
16      this(value, isShown);
17      this(int value, boolean isShown);
18  }
```

Figure 4. Mixed the use of formal parameters with actual parameters.

5. Polymorphism: (1) Mistakenly considered that declaring one superclass variable instead of two subclass variables means declaring two variables at the same line (3/22, 14%); (2) Mistakenly considered that referencing two subclass objects successively by the same superclass variable means substituting two superclass objects for two subclass objects (10/22, 45%). For example, the 7th question asked for applying polymorphism to modify related code defined in BinaryWorld class (as shown in Figure 5) such that both PokerCard object and BNCard object were referenced by the same Card variable (as shown in Figure 6). Some students mistakenly considered that just one superclass variable is defined in the following code:

   private Card pkCard, bnCard;

   Additionally, some mistakenly considered that polymorphism meant substituting superclass objects for subclass objects, with their code demonstrating as follows:

   private Card pkCard;
   private Card bnCard;
   pkCard = new Card(…);
   bnCard = new Card(…);

```
10  private PokerCard pkCard;
11  private BNCard bnCard
12  public BinaryWorld()
13  {
14      // Create a new world with 600x240 cells with a cell size of 1x1 pixels.
15      super( 720, 240, 1 );
16      pkCard = new PokerCard(16);
17      addObject(pkCard, 60, 80);
18      bnCard = new BNCard(16);
19      addObject(bnCard, 60, 200);
```

Figure 5. Reference subclass objects with their own variable types.

```
10  private Card card;
11  public BinaryWorld()
12  {
13      // Create a new world with 600x240 cells with a cell size of 1x1 pixels.
14      super( 720, 240, 1 );
15      card = new PokerCard(16);
16      addObject(card, 60, 80);
17      card = new BNCard(16);
18      addObject(card, 60, 200);
```

Figure 6. Reference subclass objects polymorphically.

6. Access modifiers of methods: Mistakenly considered that a private method could not invoke methods defined in other classes (2/17, 12%).

4.2 Missing Conceptions

Missing conceptions refer to knowledge that students have either not retained or never learned. In this study, students' missing conceptions have been identified as follows:

1. Could not recall the correct syntax of certain Java Programming constructs: (1) Static data members (12/21, 57%): Did not know the keyword for defining a static data member and/or the position for this keyword; (2) Constants (12/22, 55%): Did not know the keyword for defining a constant and/or the position for this keyword; (3) Instantiate Objects (12/22, 55%): Did not know the purposes of the new keyword were to instantiate an object and to initialize this object by passing arguments to the constructor; (4) Loop constructs (6/13, 46%): Did not how to control the execution flow by writing proper initialization expression, termination expression as well as increment expression for the for loop head; (5) Constructors (6/22, 27%): Did not know that a constructor is a special kind of method with the same name as the class it's defined in; (6) Method calls: Did not know syntax for calling non-static methods of other classes (12/17, 71%), syntax for calling static methods of other classes (10/19, 53%) and syntax for calling methods within the same class (4/22, 18%); (7) Inheritance (3/21, 14%): Did not know the syntax, i.e., using the extends keyword, for defining a class that inherit another class.

2. Did not understand how certain Java programming constructs were executed internally: (1) The for-each construct (17/17, 100%): Did not know the execution flow of using the for each construct to process all elements of a collection; (2) The for-loop construct (8/13, 62%): Did not know the control flow of the for loop construct; (3) Arguments vs. return value (11/22, 50%): Did not know how to provide additional information for a method to execute via arguments as well as how to get execution results from a method call via return values.

3. Did not realize certain object-oriented concepts: (1) Polymorphism (17/19, 89%): Did not understand the key meaning for polymorphism is that a super class variable can reference all its subclass objects; (2) Static data members (18/21, 86%): Did not know that whenever a data member is declared as static, only one copy of the data is maintained and shared for all objects of the class; (3) Method overloading (17/22, 77%): Did not know that multiple methods with the same name but with different formal parameters can be defined within the same class. Furthermore, some did not know how to pass proper arguments to call the exact overloaded method; (4) Constant (16/22, 73%): Did not know that the value of a constant data member cannot be changed after its initial value is given; (5) Constructors (15/22, 68%): Did not know that a constructor is always automatically executed whenever an object of this class is created, and that the constructor is usually used to initialize the states of that object; (6) Inheritance (11/19, 58%): Many students realized that a class can inherit and access functionalities and properties of its super classes, but still could not apply this concept to define proper class inheritance hierarchy; (7) Method's access modifiers (8/17, 47%): Did not know differences between both public methods and private methods; (8) Class data members (9/22, 41%): Could not figure out which data members are in a certain class.

4. Could point out certain features of specific concepts, but still could not apply them in problem solving: (1) Static data members: Could correctly describe static data members as data members that can be shared for all objects, but still were not able to find the exact data member that is proper for declaring as static; (2) Method's access modifiers: Could give descriptions that are somewhat correct for public methods vs. private methods such as: public methods are out of a class while private methods are inside a class, but still were not able to figure out the exact differences between both.

5. Not familiar with the application of standard Java libraries, such as collection (java.util.List).

5. CONCLUSIONS

In this study, we conducted one-on-one clinical interviews with 22 college freshmen to explore the difficulties encountered by novice Java programmers. In particular, typical missing conceptions and misconceptions that students may have about fundamental object- oriented concepts and certain programming constructs were identified. These findings pointed out specific topics that Java instructors may want to pay special attention to in order to minimize students' learning difficulties.

6. ACKNOWLEDGMENTS

This research has been funded by the National Science Council of Taiwan, the Republic of China, under the grant numbers NSC 97-2511-S-003-014-MY3.

7. REFERENCES

[1] Allen, E., Cartwright, R., and Stoler, B. DrJava: a lightweight pedagogic environment for Java. SIGCSE '02: Proceedings of the 33rd SIGCSE technical symposium on Computer science education

[2] Bruce, K. B. (2005). Controversy on how to teach CS 1: A discussion on the SIGCSE-members mailing list. ACM SIGCSE Bulletin, v.37 n.2, June 2005 [doi>10.1145/1083431.1083477]Greenroom - The Greenfoot Educators Community. Retrieved December 28, 2010 from http://greenroom.greenfoot.org/door.

[3] Eckerdal A., Thuné, M. (2005). Novice Java programmers' conceptions of "object" and "class", and variation theory, Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education, June 27-29, 2005, Caparica, Portugal [doi>10.1145/1067445.1067473]

[4] Ginsburg, H. P., Entering the child's mind : the clinical interview in psychological research and practice. November, 1997. Cambridge: Cambridge University Press

[5] Greenroom - The Greenfoot Educators Community. Retrieved December 28, 2010 from http://greenroom.greenfoot.org/door

[6] Kaczmarczyk L. C., Petrick E. R. , East J. P. , Herman G. L. (2010). Identifying student misconceptions of programming, Proceedings of the 41st ACM technical symposium on Computer science education, March 10-13, 2010, Milwaukee, Wisconsin, USA [doi>10.1145/1734263.1734299]

[7] Kölling, M. (2010). The Greenfoot Programming Environment. ACM Transactions on Computing Education, Vol. 10, No. 4, Article 14.

[8] Kölling, M. (2009). Introduction to Programming with Greenfoot. Upper Saddle River, New Jersey, USA: Pearson Education.

[9] Kölling, M., Quig, B., Patterson, A., and Rosenberg, J. 2003. The BlueJ system and its pedagogy. J. Comput. Sci. Educ. 13, 4.

[10] Perkins, D. N. and Martin, F. (1986). Fragile knowledge and neglected strategies in novice programmers. In E. Soloway & S. Lyengar (Eds.), Empirical Studies of Programmers, 1986, pp. 213-229.

[11] Ragonis, N., & Ben-Ari, M. (2005). A Long-Term Investigation of the Comprehension of OOP Concepts by Novices. Computer Science Education, 15(3), 203-221.