Concepts Vs. Programming Skills in Java Learning

Li Chen Steven Foster Hoai Le
Department of Electrical Engineering and Computer Sciences
University of the District of Columbia
4200 Connecticut Avenue, N.W.
Washington, D.C. 20008
lchen@udc.edu

Abstract

This paper focuses on what aspects of Java we should teach in the following computer language classes: Computer Science I (CSI), Computer Science II (CS II), Advanced Programming in Java, and Object-Oriented Design in Java with UML. Based on ABET/CAC Computing Curricula 2001, for CSI and CSII we suggest that instructors cover a minimum amount of the material that concerns Object-Oriented Programming (OOP) and focuses on honing the beginning student on programming skills. We can put the intensive training of OOP in Advanced Programming in Java, and Object-Oriented Design in Java with UML.

1. Introduction

Unlike the procedural/traditional programming courses such as Pascal, C, Fortran and Cobol, learning Java has two major obstacles: understanding concepts and developing programming skills. It does not mean that procedural/traditional programming languages do not have such concepts vs. programming issues. What we want to address here is that Java has more problems than C/C++ [2].

What makes Java different is that you cannot teach Java without introducing complicated concepts (i.e. OOP) because Java contains almost all of the major components of object-oriented programming in its code structure [3]. It is impossible to ignore OOP even for the simplest code example. In general, Java is an easier computer language than C/C++, but it is hard to understand at the first stage.

Traditionally, during the first two-three semesters a computer science student does not need to know much about object-oriented programming [4]. However, if they do not have a relative understanding of OOP, they will not be able to understand the structure of the Java programming language. On the contrary, too much conceptual teaching will discourage students' programming skill development [1].

There are two ways to solve such a contradictory matter: first we can give up Java as the first computer language, second we need to study how to teach Java gradually in both concepts and skills.

Java is now the first selective programming language for computer science and engineering majors at the college level. It will replace C++ as the computer language used for the Advanced Placement Examination in High Schools to test the student understanding of college level computer science course work beginning Spring 2004.

We have the experience in teaching CS1, CS2, and OOP using Java. What should we teach in each level? What is the most important material for the student? How does a student perform if our emphasis is on concepts or programming skill? Since the first author was a professional programmer, we also discuss the expectations from a software engineer point of view.

In this paper, we will focus on CSI and provide a teaching sample on how much the student should know about Java in CSI. We will also cover CSII, Advanced Programming, and object-oriented analysis and design with UML [5-6].

Our goal is to teach students in a gradually improving mechanism for the OOP concept issue while focusing on developing programming skills for most of our Computer Science majors.

2. CS I: How much the student should know in Java and OOP

Computer Science I, the calculus of computer science, is the most important class in CS. ABET/CAC has suggested using an " Imperative-first" approach to implement this model. Other five models are Objects-first, Functional-first, Breadth-first, Algorithms-first, and Hardware-first.

ABET/CAC wrote: "The imperative-first approach is the most traditional of the models we have included." "It is important to note that first course in either sequence -- CS101$_I$ or CS111$_I$ -- may well use an object-oriented language for its programming examples and exercises. What distinguishes this approach from the objects-first model is the emphasis and ordering of the early topics. Even if it is taught using an object-oriented language, the first course focuses on the imperative aspects of that language: expressions, control structures, procedures and functions, and other central elements of the traditional procedural model. The techniques of object-oriented design are deferred to the follow-on course." [1] This section will provide a practical implementation to this model.

What material should we teach in CSI? We suggest that an instructor covers: the typical Java program/code structure, Basic Input/Output statements, Assignment statements and Basic Arithmetic Calculations, Selection Statements, Repetition statements, Functions, and Single dimensional arrays. Some of the following material can be covered as the instructor's perspective vision (As long as you can define these concepts, you may do whatever you wanted): classes, string operations, and file accessing.

In such a case, we have little or nothing to do with OOP. However, when an instructor shows the first program in Java: *HelloWorld.java*, many students will get lost

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

"Why do we not do:

```
        main(String [] args) {
                print("Hello, World");
        }
"
```

What should we answer? (1) Should we go through the entire history of the high level languages? (2) Should we introduce comprehensive OOP first? (3) Should we say: "Just follow this, we will teach you later."

If we select (1) or (2), we might lose the fundamentals of CSI. If we choose (3), we may get two extreme responses: (a) OK, I will go with that, or (b) No, if you do not tell me why, I will not go any further. That is why we have to explain something about OOP. We have to do some thing to resolve this understanding issue. We will present in this paper what we did in such a case.

In order to provide the student the basic material in Java, an instructor needs to spend some time to explain Classes, Objects, Properties, and Methods. Inheritance, Polymorphism, and Information Hiding can be introduced but it is not mandatory.

3. Teaching Experimental Example: How to Teach CSI With Java

In this section, we will present our opinions and some examples we made specifically for our purposes.

3.1 Start Java

Objective: *Show a true example that student knows that we, the people have the full control over the computers. Build their confidence on Java and programming languages. They are easy to learn.*
Problem and difficulty: *Students have problems understanding the Java Code Structure.*
Strategy: *Promise students that we are going to have a quick lesson on OOP during the next class that will explain everything.*

The first java programming will be quite simple. It prints the words "Hello, World!" on the screen. This is a good example that demonstrates the typical code structure of Java. The code was presented above. As we indicated before, students may have a lot of questions on the code structure.

I usually ask students to be patient and let the instructor finish his lesson by going through the basic arithmetic calculations and display the results. Then, I always promise the students we are going to have a quick lesson on OOP next time that will explain everything. That is a strategic "lie," because no one can do this in a single lesson. However, the purpose is to ask the students to memorize the code structure and not to reject Java at the beginning.

Simple arithmetic problems allow students to know how to use Java statements. The following *ArithOps.java* is used to demonstrate simple arithmetic operators on primative data type.

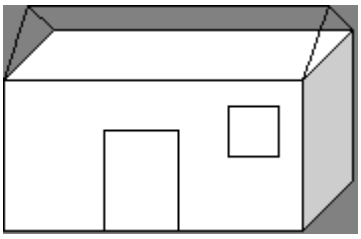
```
class ArithOps {

    public static void main(String [] args) {
        double x,y, result;
        x=3;
        y=5;
        result = x + y;                          // The arithmetic addition
        System.out.println( x + " + " + y + " = " + result);

        result = x - y;                          // The arithmetic subtraction
        System.out.println( x + " - " + y + " = " + result);

        result = x / y;                          // The arithmetic division
        System.out.println( x + " / " + y + " = " + result);

        int re = (int)x % (int)y;      // the arithmetic mod
        System.out.println( (int)x + " % " + (int)y + " = " + re);

    }
}
```

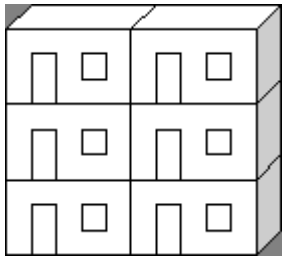
3.2 A Simple Lesson to OOP

Objective: *provide a simple lesson to Object-oriented programming that includes the concepts of Classes, objects, properties (variables and constants), and methods (functions). Explain how the Java code structure fit into these concepts.*
Problem and difficulty: *Students may still have problems understanding the Java Code Structure.*
Strategy: *Using real world examples to explain the "possible" advantages of object-oriented methodology.*

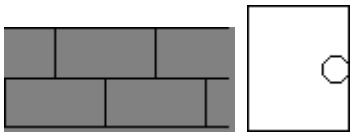
This lesson will solve two problems: (1) trying to explain why the object-oriented methodology is needed, and (2) enhancing the Java code structure.

(1) I often use the following example that relates to the construction of a small, single family-house and a big apartment building

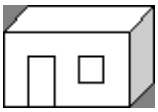




For a Simple House: the basic construction elements would be brakes, doors and windows. A basic action would be "open the door, or close the window."



For a big apartment building: the basic construction element is an apartment. We need to repeatedly build the same apartment and put it at different locations on the building. Our focus has changed to group the brakes, doors, and windows together to be an apartment and then, to use many apartments to construct a building



This repeated pattern becomes the idea of reuse and encapsulation that are the basic terminologies of OOP.

(2) To enhance the Java program/code structure. We could first show that the class that it contains two parts: Property/variable-declarations and Methods/functions. For example:

```
class MyClass{
    Property-declarations;
    Methods;
}
```

Since the apartment is a repeated pattern, we could define

```
class apartment{
    Doors, windows, etc; //Property-declarations
    Door_open(), Window_close();//Methods;
}
```

Introduce here "The minimum unit of a java project is called a Class. A Java project may contain several classes." A class is an abstract concept and it can be used to "define" or to

"declare" a real "object" for example Apartment101. Tell students that in order to use the class to construct the building, we can generate another class that is called *building*.

```
class building{
        public static void main(String [] args) {
                apartment a101= new apartment ();
                apartment a102= new apartment ();
                apartment a201= new apartment ();
                ……..
        }
}
```

Depending on students' understanding level, the instructor can add a lesson on inheritance and information hiding. However, it is not necessary to do this for CSI.

3.3 Basic Language Syntax and Practice (Multiple Lessons)

Objective: *To provide students with enough language concepts and Java syntax for programming.*

Problem and difficulty: *Many problems on how to program.*

Strategy: *No specific and easy way. More examples and practices would help.*

The most important things here are: (1) Basic Java Statements: the assignment statement, the selection statement, and the repetition statement, (2) functions, and (3) single dimensional arrays.

After each individual concept and code practice, we should give a complex example to show the internal connection of usage. We choose: Sum-AVG-Min-Max

```
class ArrayOps {
        public int[] A = {5, 7, 9, 4, 10};

        public ArrayOps() {                     // constructor
                                                // Nothing
        }

        public int findSum() {                  // find sum
                int sum = 0;
                for(int i = 0; i<A.length; i++)
                        sum = sum + A[i];
                return sum;
        }

        public double findAvg() {               // Find average
                double avg;
                avg = findSum() / 5;
                return avg;
        }

        public int findMax() {                  // Find max
                int maxIndex = 0;
```

```
                for(int i = 0; i<A.length; i++)
                        if(A[maxIndex] < A[i] )
                                maxIndex = i;
                return A[maxIndex];
        }

        public int findMin() {                  // Find min
                int minIndex = 0;

                for(int i = 0; i<A.length; i++)
                        if(A[minIndex] > A[i] )
                                minIndex = i;
                return A[minIndex];
        }

        public void display() {                 // Print result
                System.out.println(" 1. Sum =  " + findSum() );
                System.out.println(" 2. Avg =  " + findAvg() );
                System.out.println(" 3. Max =  " + findMax() );
                System.out.println(" 4. Min =  " + findMin() );
        }

        public static void main ( String [] args ) {
                ArrayOps arrayOps = new ArrayOps();
                arrayOps.display();
        }
}
```

3.4 Student Knowledge Development on Algorithms (Multiple Lessons)

Objective: *To teach students the basic ability to solve a problem using algorithms, and to show students that logical thinking on his/her own is essential.*
Problem and difficulty: *Many problems on how to use algorithmic/logical thinking and how to translate words to programs/code.*
Strategy: *Use popular and well-known examples that will link and add to the student's previous knowledge.*

Lambert *et al* use four of the popular and well-known examples in their textbook [2]: (1) The quadratic equation, (2) the system of linear equations with 2-variables, (3) the sorting problem and the bubble-sort algorithm, and (4) the searching problem and sequential-search and binary search algorithms.

(1) Solving the quadratic equation needs selection statements and math library. One could involve a function in the process. It is straightforward. Thus, students will no longer have a problem understanding the code.

(2) A system of linear equations with 2-variables can be a great example. There are three ways to solve this problem. A) Substitution, some students choose to use this method since they learned this in high school. After the formula was driven, they can program by themselves. B) Cramer's Rule, students can also understand this method and implement it without many difficulties. C) Gaussian Elimination Algorithm. This algorithm is not at

CSI level. However, I used a similar way to show students how to solve this problem step by step—the key philosophy of algorithms. The code is shown below.

```
//  Solving the system of linear equations using algorithmic steps
//
//  ax + by = c
//  dx + ey = f
//
//  solve for x, y


class Elimination {
        public static void main(String [] args) {
                double a, b, c, d, e, f;
                // get all coefs
                //
                a = Math.random()*100;
                b = Math.random()*100;
                c = Math.random()*100;
                d = Math.random()*100;
                e = Math.random()*100;
                f = Math.random()*100;

                // elimination starts
                //
                if ( d == 0){
                        System.out.println("d=0, extreme situation"  );
                        System.exit(0);
                }

                // a/d * (dx + ey = f)
                //
                double dd =(a* d)/d; // dd=a;
                double ee = a*e/d;
                double ff = a*f/d;

                // using equation (1) to subtract equation (2)
                //  ( ax + by = c ) -  ( ax + ee y = ff )
                //   (b-ee) y = c-ff => eee y = fff
                //
                double eee = b-ee;
                double fff = c-ff;

                // y = fff/eee
                //
                if ( eee == 0){
                        System.out.println("No solution"  );
                        System.exit(0);
                }
                else{
                        System.out.println("y="+ (fff/eee) );
                        System.out.println("x="+ (f –e* (fff/eee)) );
                }
        }
```

(3) Some professors are not very optimistic on requiring the sorting and search coding from CSI students. However, we do need to teach them the algorithm for sorting, especially the bubble-sort algorithm.

(4) Again, we need to teach the students sequential search and binary search. We may not ask students to code the binary search algorithm because of the difficulty.

4. CSII, Advanced Programming In Java, and OOD in Java

Computer Science II (CSII), Advanced Programming in Java, and Object-Oriented Design in Java can be selected as the series of Java/computer language courses [7-8]. CSII may be a first semester class at the sophomore level. Advanced Programming in Java can be selected by sophomores or juniors. Object-Oriented Design in Java is a junior level course, but it can be a senior level course if we add UML.

4.1 Focus of CSII

According to [1], CSII **"**Continues the introduction of programming begun in CSI with a particular focus on the ideas of data abstraction and object-oriented programming. Topics include recursion, programming paradigms, principles of language design, virtual machines, object-oriented programming, fundamental data structures, and an introduction to language translation." That is to say, CSII must be taught with an object-oriented programming language.

In the authors' opinion, the principles of language design, virtual machines, and language translation can be introduced, but they are not the focus in CSII. Our focus should be at Data structure/abstraction and OOP incorporated with basic algorithm design.

For data structure/abstraction, we want to cover Arrays, Linked Lists, Stacks, Queues, Sorting, Trees and Binary Search Trees. In OOP, we want to cover Inheritance, Polymorphism including overloading and overriding, and Information Hiding in Java. In algorithm design, we want to cover Big-O, Iteration, Recursion, Sorting that includes some $O(n \log n)$ sorting algorithms, and Search that includes binary search. The algorithms on the binary search trees will be a plus.

4.2 Advanced Programming in Java

Advanced Programming (in Java) is not listed by ABET/CAC as a core class. It can be an intermediate level class for the students who want to be professional programmers after college. This class will focus on the programming skill training in Java. The next course, Object-Oriented Design, will focus on methodology training.

In order to use the Java language proficiently, we must concentrate on the object-oriented programming skills combined with extensive coding practice. After reviewing the elements of Java, we should include:

(1) Inheritance for construction, extension and its combination with "implements" as well as their relation to public, private and protected.
(2) Abstract method for overriding
(3) Different uses of overloading
(4) GUI and the AWT Class.
(5) Input/Output Streams and File Processing
(6) Exception Handling
(7) Basic Graphics
(8) Use of built in classes such as Vector, String, etc.
(9) Multiple Threading

4.3 Object-Oriented Design in Java

Object-Oriented Design in Java can be taught with Unified Modeling Language (UML). After extensive conceptual learning, we should focus on relatively big projects in which students will be able to boost their knowledge level in the sense of general computer programming. Students will be required to finish personal projects and team-based projects. Intensive and cutting edge reading materials should be provided in this class.

Many aspects can be involved in this Java/UML class: Design Patterns, Class-Diagrams, State-Diagrams, Collaboration-Diagrams, Sequence-Diagrams, Internet and Web Programming Practice, Network Programming, Databases and Data Mining Practice.

References

1. ACM, Computing Curricula 2001, http://www.computer.org/ education/cc2001/ final/index.htm.
2. Lambert, Lance, and Naps, Introduction to Computer Science with C++, Brooks/Cole, 2000.
3. Deitel & Deitel, Java How to Program, Prentice Hall, 1999.
4. Brookshear, Computer Science: an overview, Addison-Wesley, 1999.
5. T. Budd, Understanding Object-Oriented Programming with Java, Addison Wesley, 2000.
6. M. Page-Jones, Fundamentals of Object-Oriented Design in UML, Addison Wesley, 2000
7. Goodrich and Tamassia, Data Structures and Algorithms in Java, John Wiley & Sons, 1997.
8. Standish, Data Structures in Java, Addison-Wesley, 1997.