Teaching Programming Subjects with Emphasis on Programming Paradigms

Selvakumar Samuel

Asia Pacific University of Technology and Innovation, TPM, 57000 Bukit Jalil, Kualalumpur, Malaysia
selvakumar@apu.edu.my

Abstract – No holistic approach is available to teach programming subjects, especially for novices. The current practice involves guiding the students to become the users of a programming language or tools. This study explores an alternative approach to teaching programming subjects. The students have to learn programming languages with respect to programming paradigms instead of learning how to write a solution for a problem using a language directly. Solutions are designed by the software engineers, programming paradigms are providing the way to design our thoughts. Programming languages generally provide libraries to implement the solution and provide the platform to run the solution. Students should know languages should be chosen primarily if it has paradigm support, according to the way the solution is designed and the required libraries are available. This makes the learners to understand the program language structure and programming in a better way. This approach has been tested with 30 batches of students in 7 universities. 80% of the students, particularly beginners responded positively and 50% of the students felt that, their fear on programming has been overcome. Almost everyone understands the programming language, architecture and program structure in a better way.

Index Terms - Teaching, Programming Subjects, Programming Paradigms, Programming languages.

1. Introduction

Since 1970's programming subjects are being taught in universities, but there is no holistic approach to teach programming subjects. A programming paradigm is a way of conceptualizing what it means to perform computation, and how tasks that are to be carried out on a computer should be structured and organized [1]. Programming paradigms are the core for any computer programming languages and a program design.

If students understand the role of programming paradigm in a program design, they can design a solution for any requirements with the use of a programming language as every language embodies on one core programming paradigm and support few other programming paradigms.

This report addresses some of the issues in relation to teaching programming subjects for computing and non-computing students and proposed a model to teach programming subjects with the emphasis of programming paradigms. Current teaching practices and the related issues are evaluated briefly in Section 2. Programming paradigms and programming languages are evaluated in view of the proposed model in section 3. The proposed model is explained in section 5.

2. Evaluation on Current Teaching Practices of Programming Subjects and Issues

Programming subjects are being taught in universities; in the meantime, many software development companies and corporate training companies are conducting training programs and workshops for programming languages. This is unusual for almost all other technical and non-technical subjects. As we know the objectives of conducting trainings or conducting workshops by the corporate training centers are characteristically different with university education systems. Generally trainings or workshops introduce a language feature and make the attendees familiar with the features eventually users of a language or a tool. Unfortunately, university students are also learning computer programming languages in a similar way, so that at the end of the day they will become a user of the computer programming languages.

Another issue is, the same approach is being used to teach programming subjects for computing and non-computing students. Basically computing major, students are learning additional programming subjects compared to the non-computing major, students. This is the major difference between them. Computing students should not learn a programming language as a user of the language as like non computing students. They should focus more on language architecture, working principles, and other technical details of a language design. Consequently, students could be a programming language designer, instead of merely a user.

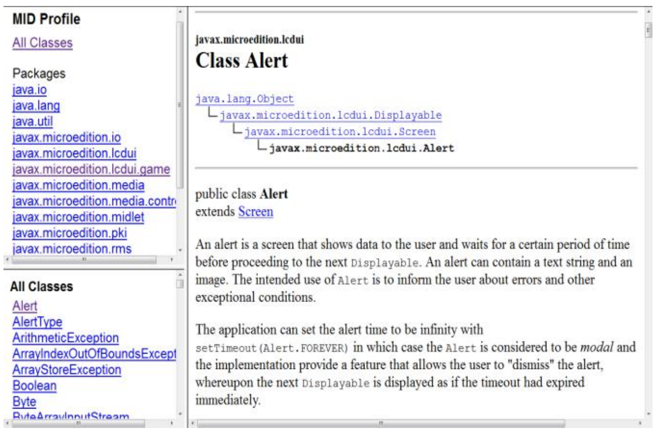
3. Evaluation on Programming Paradigms and Programming Languages

The following discussion is mainly based on the proposed model.

In general programming paradigm is an approach to design a computing solution for a problem. Programming language paradigms such as structured, procedural, object oriented programming, functional programming, logic programming, concurrent programming and event-driven programming [3,4,5] are common and dominant programming paradigms in the current development scenarios. Most of the current development scenarios need multiple programming paradigms to write the computing solutions as single programming paradigm is not sufficient to design a solution for the current problems.

A programming language is actually a collection of libraries or API's, which are based on a core programming paradigm and supports some other programming paradigms.

For example Java programming language is based on object oriented programming paradigm, but it supports structural, procedural, functional, concurrent programming and event driven programming. Mainly object oriented approach is blended with event driven programming. Since the javas' core paradigm object oriented all the libraries are made as classes and interfaces. Figure 1 lists JavaMe Packages, all classes, abstract classes and interfaces in the left side panels and members details in the right side panel. Complete details can be found in Ref. [2].



MID Profile

All Classes

Packages
java.io
java.lang
java.util
javax.microedition.io
javax.microedition.lcdui
javax.microedition.lcdui.game
javax.microedition.media
javax.microedition.media.contr
javax.microedition.midlet
javax.microedition.pki
javax.microedition.rms

All Classes
Alert
AlertType
ArithmeticException
ArrayIndexOutOfBoundsExcept
ArrayStoreException
Boolean
Byte

javax.microedition.lcdui

Class Alert

```
java.lang.Object
  └javax.microedition.lcdui.Displayable
      └javax.microedition.lcdui.Screen
          └javax.microedition.lcdui.Alert
```

public class **Alert**
extends Screen

An alert is a screen that shows data to the user and waits for a certain period of time before proceeding to the next `Displayable`. An alert can contain a text string and an image. The intended use of `Alert` is to inform the user about errors and other exceptional conditions.

The application can set the alert time to be infinity with `setTimeout(Alert.FOREVER)` in which case the `Alert` is considered to be *modal* and the implementation provide a feature that allows the user to "dismiss" the alert, whereupon the next `Displayable` is displayed as if the timeout had expired immediately.

Fig. 1 JavaMe API lists

Figure 2 in section 5 demonstrates the role of programming paradigms in a simple Java code.

If a student knows how to use a class and its members to write a computing statement, he can apply it to using another class. There is no difference in using another class. If a student knows how to use the 'if..then..else..' statement in one programming language, conceptually there is no difference for the student to use it in another language program design. The 'class' or 'if..then..else..' is not Java or C++ or C#. The 'class' and 'if..then..else..' are programming paradigm concepts. Therefore language provides syntax and platform to implement, compile and run the solutions. The main difference between the programming languages are syntax, number of libraries, name of the libraries, application of the libraries and programming paradigm supports. So a student who knows programming paradigm concepts clearly along with how to implement it in a programming language, should be able to implement it in another language with no major difference except some syntactical differences, different library names and different IDE's. These differences can be easily identified when students become familiar with programming paradigm concepts. The size of a project does not matter, as designers are just going to repeat the programming concepts depending on their requirements.

Students have to choose a programming platform for a solution design provided that the particular language has that library. For example, if a student wants to create a Bluetooth application, the student has to check which language has libraries for Bluetooth application development. If there is no library support from a particular platform, then there is no point using that particular platform for the Bluetooth application development.

If a student knows object oriented programming paradigm concepts thoroughly and knows how to implement in a programming language, consequently they would know how to implement in another object oriented language provided if they know the syntactical differences. For eg. To implement inheritance Java uses "extends" keyword, c++ use ":", VB dotnet uses "Inherits" key word but the meaning is same.

Any program is made up of "Function + Data + Programming Paradigm Concepts + Logic". Based on the programming paradigm concepts the data and functions will be adopted in a program. For e.g. in an object oriented design program we will keep all the functions and data members in a class mechanism which will be covered by encapsulation mechanism. Programming logic is based on our functional requirements. All the functional requirements will be written as function definitions.

Students have to learn programming languages through programming paradigm driven approach, i.e. while writing a code student have to recall the programming paradigm concepts. For e.g. in order to use a private member of a class, students should know the access rights of a private member. This concept is not a programming language concept, it's a programming paradigm concept, it is common for any language platform provided it supports that particular programming paradigm concepts. Therefore a language provides a platform to implement our solution design.

4. Research Methodology

Observation and feedback methods have been used. As an instructor and as a participant, programming subject classes have been observed in 7 universities. To test the proposed model, verbal and written feedback have been collected from students and analyzed. The analysis results are briefly discussed in section 6.

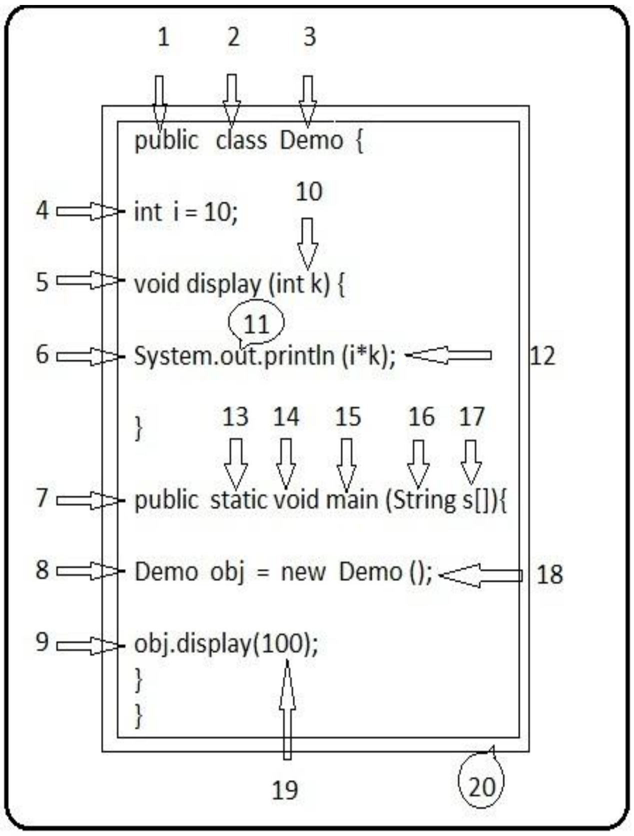
5. An Approach with Emphasis of Programming Paradigm Concepts

Following are the steps of this approach:

Step1: Type a code as Fig. 2 demo code:

The instructor has to use a text editor to type a simple program. It is important for the instructor to type the code (without referring any documents) in front of the students. This will motivate the students and increase the confidence about the instructor. This approach will grab the students' attention. Here Java is considered as an example. The instructor can use notepad++ to type the code, and can use a simple IDE such as JCreator Pro to test the code.

Instructors should avoid using advanced IDE's as it is not suitable for the beginners. Particularly beginners need to spend some time to understand the usage of IDE's. Current IDE's generate some codes which help the programmers to complete their tasks faster and in an organized way. Generally, students don't understand and don't bother about the codes which could be generated by the IDE's.



```
public class Demo {

int i = 10;

void display (int k) {

System.out.println (i*k);

}

public static void main (String s[]){

Demo obj = new Demo ();

obj.display(100);
}
}
```

Fig. 2 Demo Java program indicating related programming concepts.

Once students are familiar with the programming concepts and the respected programming language, advanced IDE's can be used for program development. In the workplace, students can use the advanced IDE's but in the learning environment (universities), it is better to use simple IDE's just to type, debug, compile and run the codes. This will help the students to learn in a better way.

Step 2: Start with some questions:

The instructor can ask some questions such as shown below, before beginning to explain the programming concepts.

Question 1: What does a program have?

Question 2: What is a programming language?

The usual questions are: 'What is a program?' and 'What is a programming language?' The instructor should explain the answers for these questions then should explain the answers for the two questions above as discussed in **Section 3** about programs and programming languages.

Step 3: Explain the demo program concepts

The instructor should explain the concepts and principles behind each and every keyword of the program as indicated in numbers (1 to 20 in Fig 2).

Instructors can consider the following points against the numbers (1 to 20) to explain the programming concepts:

Number 1 & 7: Keyword is public – access specifiers and encapsulation.

Number 2 & 3: Keyword is class – class concept.

Number 4: Keyword is int i – data types, variables and initialization.

Number 5 & 14: Keyword is void display () – function declarations and non-return function.

Number 6, 11 & 12: Keywords are System, out and println() – Built-in class, class libraries, static concepts (11), output function (12) & related principles behind System.out.println().

Number 8: Keyword is 'Demo obj' – object concept, deriving object and 'new' key word.

Number 9: obj.display() – accessing class members.

Number 10 & 19: Keywords are (int k) & (100) – parameters, arguments, cal by reference, cal by name, etc.

Number 13 & 15: Keywords are 'static' and 'main' – static concepts, main function and relate with 'out (11)'.

Number 16 & 17: Keywords are 'String' and 's[]' – String class, how string class and character data type, arrays and object arrays (17).

Number 18: Key word is 'Demo ()' – Constructor concepts.

Number 20: Data hiding, encapsulation and class member access rights could be explained as the double line box symbolically represents the said concepts.

While introducing the concepts behind this code, instructors do not necessarily have to follow this order (1 to 20) as indicated in the demo program Fig 2. Orders are up to their convenience and depend on their class duration.

When defining a concept, instructors could ask the questions as discussed in STEP 2. For e.g.: To define a 'class' instructors can ask questions like "What does the class consist of?" Then let the students see the demo program class. Possible answers could be – "A class consists of members such as data and methods", "Methods has instructions each instruction is written on the basis of certain programming concepts".

Step 3 will take a few classes and lab / tutorial sessions to complete as it covers many programming concepts.

At the end of the session the students will understand the role of programming paradigms and the respective programming concepts in the program design. Although given the demo program has only a few lines of code and the outcome is just to display the result of (i*k), but it is made up of many core programming concepts. Thus, students will understand the importance of learning programming paradigm concepts and how to use it while writing a code in a programming language.

While teaching object oriented languages such as 'Java', the instructors should focus on object oriented programming concepts. In order to demonstrate I/O concepts procedural or structural programming language such as 'C' should be used.

Finally the instructor could ask the students, in this code where is "Java"? Or the role of "Java"? Hence the instructor can conclude that the Java language is the platform for development and it is made up of certain libraries. If students know the programming paradigm well, they can easily write the code.

Step 4: Practical/tutorial sessions

The instructor should introduce one or two basic IDE's to the students. The advantages and disadvantages of using IDE's should be discussed as step 1.

During every practical / tutorial session, the instructor should give a demo code to the students by focusing on one or two programming concepts.

The instructor has to ask the students to practice the given code by editing and adding new similar functionalities or similar codes, where by students will understand the common errors and concepts behind the codes clearly.

At the end of the session the instructor can ask any one or two students to present their work, which will increase the confidence and decrease the programming anxiety of the students.

Step 5: Introduce and discuss related and alternative concepts:

Once programming concepts are explained with respect to the fig. 2, the related concepts can be introduced. The instructor should edit the fig. 2 demo code to introduce the related and alternative concepts by adding new methods and instructions.

For e.g.: return type functions should be introduced as an alternative method for void type functions (point 5 in step 3), data types (point 4 in step 3) can be related with parameter types (point 10 in step 3).

For e.g.: return type functions should be introduced as an alternative method for the void type functions (point 5 in step 3), data types (point 4 in step 3) can be related with parameter types (point 10 in step 3).

Step 6: Compare syntaxes of different programming languages:

While explaining the concepts, the instructor should compare the syntax of one particular concept (e.g.: inheritance) with other similar programming language as discussed in Section 3.

Step 7: Explain the role of multiple programming paradigms:

The instructor can explain how different programming paradigms are blended in a program, although there is a core programming paradigm for every programming language. In our example (Fig. 2), Java is the language, object oriented programming is the core and the procedural and structural concepts are blended.

The above steps can be repeated if it is necessary, until the end of the course.

6. Testing Results and Discussion

This approach has been tested with 30 batches of computing and non-computing students in 7 universities. Verbal and written feedbacks have been received from the students. 50% of the students felt that, their fear on programming has been overcome. Most of the non-computing students said that, their interest in programming languages has increased. 80% of the students, particularly beginners responded positively and almost everyone said that they understood the programming language, architecture and program structure in a better way.

This approach also been introduced with some programming subject instructors. They have acknowledged as this approach is quite interesting and relatively one of the fastest way to teach programming subjects and also they have commented that the students are idle while the instructor is typing the code. To rectify this issue instructor should ask the students to follow them by copying the code on paper, to be tested later in their lab or tutorial sessions.

7. Conclusion

Based on the empirical understanding of the current teaching approaches, teachers should focus on programming concepts instead of teaching the programming languages directly. Different languages should be used to demonstrate the programming concepts while teaching the programming concepts. Students should be trained as a designer of a new language or new programming concepts instead of being trained as a user of a language.

Acknowledgement

Thanks to God for making this paper successful. Sincere thanks to our university management for the moral support and resources provided during the research.

References

[1] T. Budd, "Multi programming Paradigm in Leda," Addison Wesley, pp. 3, 1993.
[2] http://docs.oracle.com/javame/config/cldc/ref-impl/midp2.0/jsr118/index.html.
[3] http://www.eecs.ucf.edu/~leavens/ComS541Fall97/hw-pages/paradigms/major.html.
[4] http://www.iue.tuwien.ac.at/phd/heinzl/node32.html.
[5] http://people.cs.aau.dk/~normark/prog303/html/notes/paradigms_themes-paradigm-overview-section.html.