Special Issue of National Conference NCIEME 2016 Organised by Rajarambapu Institute of Technology and Walchand College, Sangli ISSN 2394 - 1707

Teaching C++ Using Real Life Analogies

Shashikala V. Budni, Deepa Mulimani, S.V.Seeri, P.RPatil

MCA Department, KLE Technological University, HUBLI.
shashikala@bvb.edu,deepamulimani@bvb.edu,seeri@bvb.edu,prakashpatil@bvb.edu

Abstract: Over the years Object oriented programming (OOP) has gained the dominance in both the software industryand education. There are a number of OOP versions released by Software companies and the same are adopted by the educational institutions at various levels in their curricula. The main reason for this inclination is driven by the fact that OOP solves any problem by imitating the mental model of humans. It is usually a challenge for teachers to imbibe OOP concepts in novice programmers. Teaching fundamentals of OOP at an introductory level is challenging. This paper presents a pedagogical approach used for teaching/learning process of Object Oriented Programming in C++ course offered at MCA II semester. The approach uses real life analogies to effectively enhance the learning of OOP concepts among students. Students might program in an OOP language but fail to program in an object-oriented style. This is because students are unsuccessful to model given problem in terms of objects. To ease this, single real life example is used to help students acquire these OOP concepts. The use of a single example from start to finish facilitated deepening of teaching. It mainly helped to join with ease the new concepts and mechanisms, such as class combination, inheritance and polymorphism. Eventually the example expanded into a kind of system. The proposed methodology comprised of five stages in which students are introduced to classes and objects, constructors and destructors, abstraction and encapsulation, inheritance and polymorphism concepts. This approach demonstrated effectiveness in motivating students to incorporate OOP concepts for developing simple real world applications. It has also improved the quality of teaching C++ programming course. An evaluation of the problems experienced in teaching OOP is presented and some possible approaches for improving the quality and success of such courses are discussed.

Keywords: Object Oriented Programming, Object-oriented technologies, Real life Analogies, Class.

1. Introduction

Over the last ten years Object Oriented Programming (OOP) has become the leading programming style in both education and software industry. We have number of OOP languages which are released by software companies and most of the educational institutions have included these in their curricula. Most university courses on programming have moved from C to object oriented languages like C++, Smalltalk, Eiffel, Java and C#. The main reasons for this tendency are related to having advantages like: abstraction, encapsulation, inheritance and polymorphism. Object oriented methods scale up very well with the increasing sizes of real life software projects and enable design of complex software products. However, because of the paradigm shift in OOP languages, switching from the older crucial procedural programming style to object oriented style remains to be a challenging task. Students who havebeen exposed to procedural programming find it a little difficult to move towards OOP. It takes the average programmer 6 to 18 months to switch her mind-set from a procedural to an object-oriented view of the world [1].

The pedagogy of traditional imperative programming languages is built upon sequence; selection and iteration. They are mostly based on the von Neumann architecture. Hence, teaching traditional popular languages such as C is directed by the design and implementation of common algorithms and data structures. This pedagogy requires teaching the syntax and basic structures of the language as early as possible. The early stages of teaching are dominated by the peculiarities and syntactical details of the language. On the other hand, the main focus of OOP teaching is problem solving, rather than language details. Since C++ is a descendent of C, it has a negative legacy effect of traditional programming which is the emphasis on syntactic and structural details of the language. As a result, our students are inclined towards coding C++ programs in a non-object-oriented style.

Often there are presumptions that OOP training in the earlier stages of programming education is too hard and complex [2][3]. To some extent these prove to be true. Nevertheless, these difficulties can be overcome by choosing effective teaching pedagogies and careful design of the courses [2]. Object oriented software technologies are fundamentally based on the modelling of real life objects. It is therefore expected to place the explanation and identification of objects at the start of an OOP course. Generally speaking, object-orientation is not only a programming style but also a methodology that deduces from general concepts to the special and induces from the special to the general. This paper discusses the pedagogy used for teaching “Object Oriented Programming with C++” course offered during MCA II semester. The course aims at training students to program in OOP language. The objectives of the course are listed below:

1. Articulate the principles and benefits of object-oriented approach in problem solving through C++ programming.
2. Write C++ programs using basic programming constructs and data structures.
3. Apply the concepts of class, method, constructor, instance, data abstraction, function abstraction to implement the encapsulation features in C++ programs.
4. Write modular, secure and reusable C++ codes to exemplify the features of inheritance, polymorphism and generic programming.
5. Perform file-related activities using C++ in an OOP paradigm.

2. Teaching Pedagogy

A. Analogies and Demonstrations.

Class is the foundation of C++ programming. Hence, the process began with dealing of real life examples and class as the core in order to trigger the object-oriented thinking among students [4]. It is the fundamental mechanism of the C++ language where it tries to achieve the organizational form of data encapsulation and information hiding.

Therefore, the course delivery began by using class as a core and further adding features. The sooner the understanding and grasp of the conception of class, the students will learn effectively.

A single example is used throughout the teaching process. It started with the definition and description of a simple class through an example to illustrate the analogy. Further, the other concepts and mechanisms such as abstraction, encapsulation, inheritance and polymorphism joined the example. It progressively formed a kind of system, and finally expanded into a complete C++ example. The object-oriented knowledge was taught tightly around the concept of class using real life example.

a. Classes and objects

First, students were introduced with the concept of class through real world example say concept of car design. They were asked to imagine themselves as (engineers) architects and give the prototype model (plans and other build information) of a generic car. This acted as a base line for the class definition of a car. By using this prototype the different car companies (TATA, Maruti Suzuki, NISSAN, TOYOTA, etc) would make a new car. They use the “new” method on the car class to build a new instance of it and then a new “object” of class car will be ready. Once a car “instance” is ready, we expect it to do something say it is to be driven by a person. This would be considered as a method of the class car. Figure 1 depicts the real life designing of the car whereas Figure 2 depicts the OOP implementation of the car design.

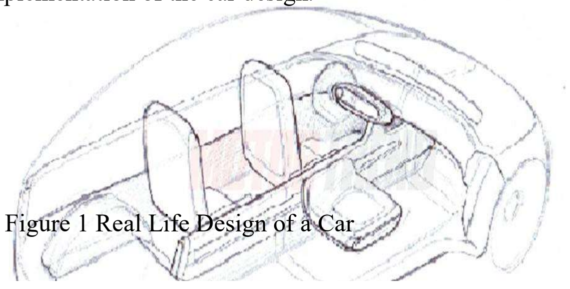


Figure 1 Real Life Design of a Car

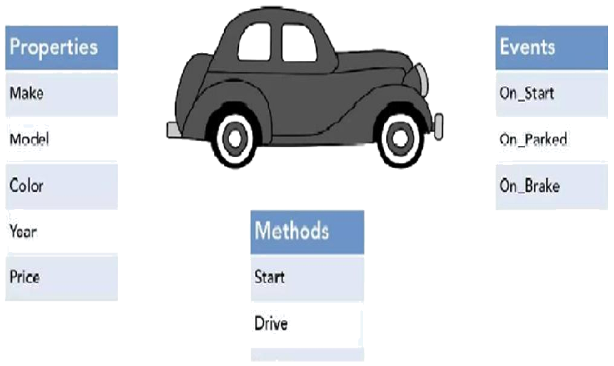


Figure 2 OOP Implementation of a Car Design

Code in object-oriented programming is organized around objects. Once the objects are ready, they can interact with each other to make something happen. Let's say we want to have a program where a person gets into a car and drives it from place A to place B. We would start by describing the objects, such as a person and car. That includes methods: a person knows how to drive a car and a car knows what it is like to be driven. Once we have our objects, we bring them together so the person can get into the car and drive.

b. Introduction of constructor and destructor

As inputs needed for the car are provided, they will be processed by the manufacturing plant (constructor) and a new car (object) is returned. So whenever an object is being created a function called constructor is called to create that object. Car plans (prototypes / the "class" definition) don't take space at all. It is the "objects" that are built from those plans that take space. When a Car "object" is not required anymore, we need to get rid of it. The cars cemetery is where they wreck used cars to "free" up the space used by these old "objects". It is also called the "garbage collector" in other OOP languages like Java, C# and the like.

Initializing an object introduces the concept of constructor in C++. Manufacturing number of different types of cars maps to call of different constructors of the same class Car. A new car occupies a physical space so in memory. Once a car is no longer used by a person then physical space is released in which case the memory gets de-allocated. This results in call to a destructor.

c. Abstraction and Encapsulation

In order to drive a car, the ‘start-engine’ method of the car is called. This method uses some internal properties of the car to do something on it. It uses the power from the car's battery to run an electric motor which is connected to the engine by some gears. In this case, the battery is a "public" property of the car as everyone can see, access and/or change it. The electric motor is a "protected" property, since only its certified mechanics (its "friends") can see and change it. The gears are "private", because nobody other than companies themselves can see or change it. But herethe 'start_engine' method is used to rotate those gears, even while they are "private". In fact, the gears can be accessed only through the 'start_engine' method which is "public".

Encapsulation helps us to show/hide what user can access. Let say the car class has some of its own functions like start-engine, stop-engine, accelerates, change the gear position, apply break, etc. These are the functions which user can access. That means they have been set as Public. There can also be some functions which are required by the car to know the fuel level and gear position status to start the engine. This is handled by the car itself, so these have been set as Private.

Continuing with the above example in the driving classes, the instructor does not teach about the functioning of the engine. This information about the functioning of engine is not necessary for driving. Such things are kept hidden from the drivers. So functionalities are abstracted from us.

In case of a terrible breakdown to the car and a local mechanic tries to check and repair it. The local mechanic is unable to repair it whereas; a chief mechanic from the company is successful. This particular instance can be mapped to OOP concepts. A driver knows how to start the car by pressing the start button. When the start button is pressed there are number of components that start working. These internal details of the start operations are hidden from the driver. And this, can be rightly stated as ‘the start process of a car is encapsulated’. Similarly the wheel of the car and its movement is encapsulated as the process of rotating the wheel is hidden. These instances facilitated in imbibing the abstraction and encapsulation concepts in students. Figure 3 illustrates the concepts of Encapsulation and Abstraction.

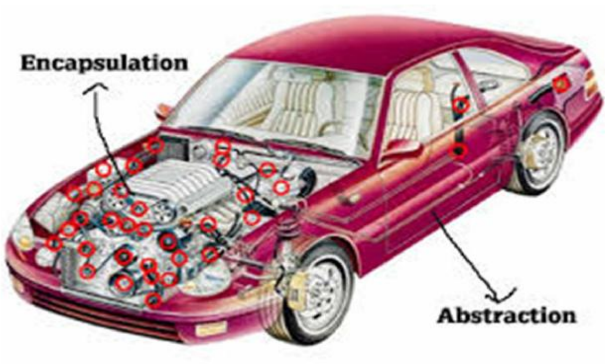


Figure 3 Implementation of Encapsulation and Abstraction in Car Design

d. Inheritance

Suppose a company owns a car model say I10 and after few years its new version is released say I20. This new car "inherits" most of the original i10 properties. It "extends" the old one and also resolves some of the problems existing in older one. The new version might have replaced the ‘start_engine’ method which was weak with a new and more powerful method which overrides the old one. An important point not to be forgotten is that the I10 class is inherited from a more generic class named car. It also inherited from the ‘Hyundai’ class to use some of thestandard design concepts of Hyundai. This is called ‘multiple inheritances’. This was used to introduce the concepts of inheritance and its types.

e. Polymorphism

A car can be used in more than one form – for travelling, transporting goods, as a vanity van and so on. It can be used as any of those subjects and all the laws about all of them are applicable to the car. This is called "polymorphism". Polymorphism illustrates how different objects of the same type respond differently to the same function call. A car can be made to act differently. That is, a car can be transformed to plough fields. The Figure 4 below illustrates this feature. This example helped the students to correlate the real life analogies with polymorphism concepts.

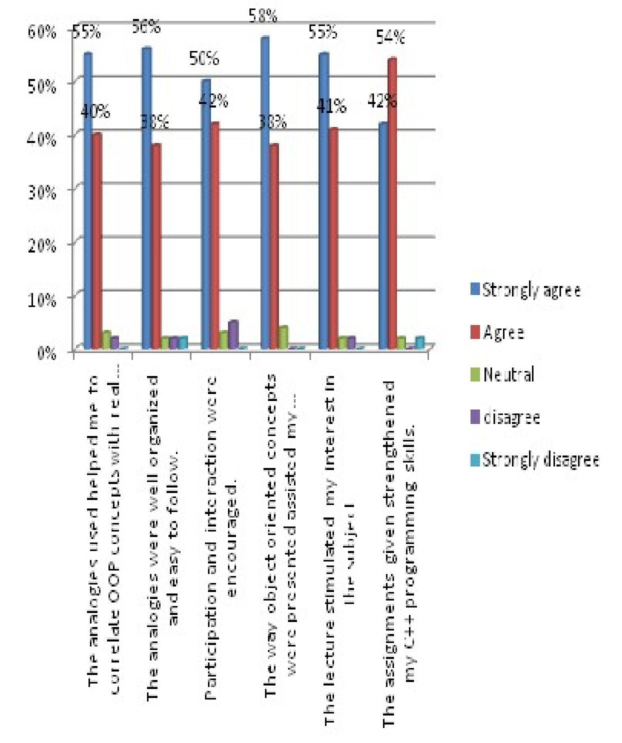
Figure 4 Polymorphism Illustrations by a Car

B. Supplementary learning materials.

In teaching this course using real time analogies it was most evident that extra assignments of similar kind were very effective. Plenty of practice assignments were given to students in groups to solve. After completing each concept students were asked to add that concept in to the existing code to eventually to form a complete program. The practice assignments probed the students to relate their real life experiences with the OOP concepts.

3. Evaluation and Results

During the process students were evaluated for a number of practice assignments. These assignments which were given in groups about the similar examples to solve were evaluated and allotted marks. The results showed that students fared well in most of the assignments. Some of the students were unable to manage with assignments on Abstraction, Encapsulation and Polymorphism. For such students repeat exercises were given and trained. At the end of the course, a feedback about the approach adopted was given by the students. The same is depicted in the graph below:



Graph 1 Student Feedback

4. Conclusion

In teaching the course “Object Oriented Programming with C++” to second semester MCA students, analogies were used to relate object oriented concepts to familiar real life scenarios. Joining with ease most of these concepts and mechanisms in the same example facilitated deepening of teaching/learning process. Using class as core helped students to code C++ programs in OOP style. The significance of this approach is driven by its applicability and usefulness in teaching OOP concepts for beginners. The approach has been tested with students were it agreed to be easy to acquire and consequently apply OOP concepts in problem solving. It also aids students in visualizing the OOP concepts provided by the real life scenarios. Through this approach it was found that, students learnt to correlate their real life experiences with OOP concepts which is a pressing requirement in problem solving. The course outcomes where successfully achieved and Program Outcome 1, 2 and 3 where attained.

References

Kölling, M. "The Problem of Teaching Object-Oriente d Programming, Part 1: Languages," Journal of Object-Oriented Programming, 11(8): 8-15, 1999.

Xiao-dong Zhu: Teaching Adaptability of Object-oriented Programming Language Curriculum International Education Studies; Vol. 5, No. 4; 2012 ISSN 1913-9020 E-ISSN 1913-9039 Published by Canadian Center of Science and Education..

Biju, S. Mathew. 2013, 'Difficulties in understanding object oriented programming concepts', in K. Elleithy& T. Sobh (eds), Innovations and Advances in Computer, Information, Systems Sciences, and Engineering, Springers, Netherlands, pp. 319-326.

http://www.celt.iastate.edu/teaching/teaching-format/14-creative-ways-to-engage-students.