PDHonline Course G195 (2 PDH)

Object-Oriented Thinking

Instructor: Warren T. Jones, Ph.D., PE

2020

PDH Online | PDH Center

5272 Meadow Estates Drive
Fairfax, VA 22030-6658
Phone: 703-988-0088
www.PDHonline.com

An Approved Continuing Education Provider

Object-Oriented Thinking

Warren T. Jones, Ph.D. P.E.

Course Content

Module #1: Motivation for Object-Oriented (OO) Design

The roots of object-oriented design reach back to the 1960's to the Simula language developed at the Norwegian Computing Center. Simula is a simulation language and supports most of the basic concepts seen in OO languages today. The next significant development was the language Smalltalk, designed at Xerox Parc in the early 1970's for use by non-professional programmers. The Department of Defense led the development of the Ada language in the late 1970's and the Eiffel language was introduced in the mid 1980's. C++ appeared on the scene in the early 1980's and was the first OO language to go mainstream. Then in the mid nineties Java began to dominate the OO landscape, driven by the emergence of the Web and the demand for distributed applications in general. Microsoft's C# and the open source Ruby on Rails are recent developments. A language called Alice is now also available based on an interface and objects that resemble a game-playing environment that makes learning OO more visually appealing and intuitive.

So why should engineers be interested in OO technology? It is true that OO is more complex and the transition from languages like C to OO is known to be more challenging than just learning another procedure-oriented language. Making the jump to an OO language like Java may be motivated by requirements for platform independence or networking. Another rationale for considering OO can be given in two words that are very easy for engineers to understand, cost and reliability. The dream from the outset of the OO concept was that the creation of software could become a process of construction from collections of standard components in a library in much the same way as now is commonplace in electrical or civil engineering. Later in this course the mechanisms that have been developed to help move this dream toward reality are described.

During the past several decades the software development process (See PDHOnline course in Software Engineering Concepts) has been studied with the goal of coming up with improvements in the process that will solve the so-called "software crisis" of increasing costs and reliability problems as software systems have increased in complexity. Early attempts, such as structured programming of procedure-oriented code, focused on improving the latter stages of the process, such as the production of modular code. More recently the OO approach has sought to produce improvements on the early stages of the software development process. Another way of saying this is that structured programming addresses issues in the solution set (code implementation) near the end,

whereas OO design and programming seeks to improve the approach to the problem set (requirements and architectural design) at the beginning [1].

Often cited strengths of the OO approach are:

- OO descriptions can be understood by individuals with little technical background in programming thus reducing the communication problem with clients.
- OO programming languages support and promote code reuse which, in turn, increases programmer productivity and can lead to reduced development and testing costs.
- OO designed systems are easier to change.
- OO design can be viewed as a real-world modeling technique in much the same manner as a computer simulation model.
- OO design methodology scales well from simple to very complex systems.
- OO facilitates rapid development by focusing on the early stages of the development process.

Module #2: OO Foundations

The term "A new programming paradigm" is often used when referring to OO programming. The use of the term "paradigm" here is taken from its use in the book Structure of Scientific Revolutions by the historian of science Thomas Kuhn. He used "paradigm" to refer to the way science organizes knowledge into theories and methods that represent a way of viewing the world. This view of the world may change when new paradigms emerge that replace the older views. A programming paradigm is an approach to conceptualizing what it means to carry out a computational process and how computational tasks should be organized. The OO paradigm is very close to how we think of problem solving in our everyday lives. We use a familiar situation to illustrate some basic principles of OO.

Consider the scenario of ordering a pizza for home delivery. Suppose a person named Jim decides he wants pizza for dinner. He has a certain type of pizza in mind and the specific toppings he likes. Suppose further that he doesn't want to prepare and cook the pizza himself. He would like to delegate this responsibility to someone else who has the skills to do this. So Jim makes a telephone call to the local pizza parlor and passes the owner, Alan, the information about the pizza he desires and his home address with the full confidence that Alan will take responsibility to accomplish the task requested. When the pizza is ready, Alan passes it to Walter, the delivery person, who then makes the delivery to the correct address.

Notice that the above problem of obtaining a pizza for dinner was solved by finding an appropriate agent Alan and passing a message to him containing the request. Alan then has the responsibility to carry out the request. Also notice that there is a set of operations or methods that Alan will use to appropriately prepare the pizza. These preparation and

cooking details, that Jim need not know, are "hidden" from him. Similarly, when Alan gives the pizza to Walter with the message to deliver, Alan need not know all the details of how to get the delivery accomplished. These are hidden from him. He has delegated this task to Walter and Walter has the method for carrying out the delivery.

This scenario illustrates several aspects of OO:

- An object-oriented program can be viewed as a set of agents called objects which interact by appropriate message passing. Each object provides a service that is used by other objects.

In the above illustration, Jim the requestor, Alan the cook and Walter the delivery person, represent agents in the pizza delivery process. Notice that the problem of getting the requested pizza to Jim is solved by passing messages from a requesting agent to a receiving agent in two stages. The message is sent first from Jim to Alan the cook and then from Alan to Walter the delivery person. In each case the request was given to the receiving agent with the responsibility to carry out the request with the methods available to him. The methods used are of no concern to the requestor and are hidden from him. More than one method may be available to an agent. For example, Walter could choose to use an SUV or a sedan for delivery. He could consult a city map in the pizza parlor or he could use a GPS navigation system in his vehicle.

- The concept of information hiding, frequently called encapsulation is an important aspect of OO programs.

When Jim envisioned ordering a pizza, he had knowledge of pizza parlors in general. That is, he knew what type of products they sold and how they operated. In other words, pizza parlors are a class of restaurants and the particular one he called at some address, for example 56 Main Street, was an instance of this class.

- All objects are instances of some class. The methods used by an object are defined in the class associated with that object. Elements encapsulated inside a class are not accessible to code outside the class. We sometimes say that classes have a public interface for message communication and a private hidden implementation for method details and attributes. Classes are also referred to as blueprints for objects.

Moreover, classes can be organized into class hierarchies of more general knowledge. For example, one could think of moving up a class hierarchy of pizza parlor, restaurant, retail store, etc. in which each is a subclass of the next one in the list.

- Class hierarchies can be constructed in which a subclass inherits attributes from the class above it.

In the OO world view, programming does not begin with variables, assignments or addresses. Instead, the language and framework is that of objects, messages and

responsibility for some action. The program design process begins with an analysis of system behavior. A distinct advantage of this approach is that the behavior of a system is typically understood earlier than any other aspect of the system. The Unified Modeling Language (UML) is a popular graphic tool for OO design. UML emerged in the late 1990's as an attempt to draw the best from the wide variety of modeling languages and methodologies and create an industry standard. This effort was largely successful since the Object Management Group (OMG) for industry standardization adopted UML as an industry standard in 1997.

Various UML diagrams can be used to model the behavior of a system. The three most commonly used graphic tools are use case, sequence, and class diagrams. Use case diagrams are closely related to user scenarios of interaction with the system. After delivery of the system each category of user will have a different view and interest in the system. The use cases seek to model these various views. Sequence diagrams capture the temporal ordering of messages sent between objects and class diagrams model the structural elements of the system.

A UML class diagram is shown in Figure 1.

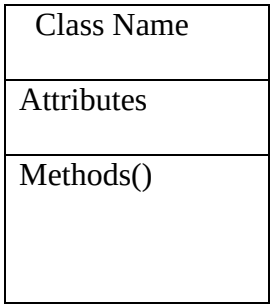

Figure 1: A Class Diagram

Module #3: The Power of Reuse

OO facilitates the construction of new classes by using existing classes. This means that changes to a system to incorporate a new requirement can be made in a very controlled manner, at the right level and with fewer errors. Reusing from a class library for new development also saves time and hence reduces cost. The design of reasonable classes with an eye for later reuse is nontrivial, however, and must be considered as part of the investment in OO technology infrastructure. Derived classes can be created in two ways.

Inheritance

First, we can use the concept of inheritance in which a class and a derived class are in the relationship is-a . In the above pizza delivery example, we mentioned a class hierarchy in which the classes are related in this way. For example, a pizza parlor is-a restaurant. The

pizza parlor inherits all the attributes and methods in the restaurant class but adds some specialized attributes and methods and is said to extend the restaurant class. So when we define the pizza parlor class we only need define the attributes and methods in the pizza parlor class that make it different from the restaurant class since the attributes and methods of the more general restaurant class are automatically inherited.

Inheritance hierarchies can sometimes be problematic. Over a period of time as changes are required and classes added using the is-a relation, difficulties can sometimes encountered. For example, suppose we want to define a bird class hierarchy. We want to define the root class of the tree to carefully contain the attributes most common to birds. One of the most natural is the ability to fly. You can already guess the dilemma. Later as subclasses are added and the tree becomes somewhat complex with many types of birds, the time arrives when we need to add the class for the unanticipated ostrich category. If we add an ostrich class with the is-a relation it will inherit all the attributes of the base class bird, which includes fly. So in order to accommodate this exceptional situation the class hierarchy must be modified to define bird at the root level omitting fly and the next level to be flying birds and non-flying birds, a major disruption in the system. As mentioned earlier, the appropriate construction of classes for reuse is non-trivial. It is difficult indeed to design a class anticipating all possible extensions and thus avoid this problem. Thus the initial great enthusiasm for dominant reuse with the is-a relation has cooled considerably after real world experiences of applications. Designing with the is-a relation is actually a trade-off between system complexity and more functionality. Because of this potential problem of inheritance, some would say that one should only use the is-a relation as a last resort.

Composition

An alternative, and many believe a more desirable, way to create classes from other classes uses a mechanism called composition which represents the has-a relationship. The automobile is the classic example for this mechanism. For example, a car has-an engine. The inheritance mechanism requires that the derived class and superclass be in the is-a relation. In creating a new class using composition, the relationship between classes is not important. All that is required is that the superclass be in the scope of the derived class and that the superclass have accessible public members needed by the derived class.

Polymorphism

Some consider polymorphism to be the most important aspect of OO design for reuse. The word polymorphism literally means “many forms”. That is, a name can have different meanings. This idea is used very commonly in everyday life. Consider the word “ride”. This has many meanings. One can ride on a bicycle, in a car, in a boat, etc. The same name is used but the meaning takes a different form in each context.

There a several varieties of polymorphism in OO languages. We will focus on overloading and overriding. These techniques are similar in that both are mechanisms for selecting from many different candidate code bodies for execution. However, they also differ in several ways. The most important difference is that overloading is done at compile time, sometimes referred to as early binding, whereas overriding is carried out at run time, sometimes called late binding.

The concept of overloading may be familiar to C programmers since C allows overloading of function names. The important idea here is signatures. A method has a signature consisting of its unique name and parameter list. The signature establishes the context for the particular meaning intended. For example, a class may define a method called "add" with a parameter list of (integer k1, integer k2) and a subclass use the same name "add" with the parameter list (double p1, double p2). The code selected is dictated by the unique signature at compile time.

When defining new subclasses within an inheritance hierarchy the methods may need to be the same name and even the same signature but require a different implementation code. One possible solution to this problem is to rewrite the method in the superclass. Modifying classes internally in this way is to be avoided since it will affect other subclasses in the hierarchy. So a better solution is to use overriding polymorphism by defining the method code needed with the same name in the subclass.

The Future of Software Reuse

In spite of all of the excitement and enthusiasm for OO over the last decade, most would agree that it has not met the expectations that the proponents envisioned of ushering in the "software industrial revolution" of components based software. Why has this been the case? Several reasons are given below [2].

- The production of quality and reusable software components is almost always a greater investment of time and resources than the development of the customized software needed initially.
- The benefits of reuse must be amortized over multiple projects.
- The lack of immediate benefits of producing reusable components reduces motivation.
- The variability of problems across projects slows the development of reusable components.
- "Not invented here" can sometimes slow development of reuse across projects.
- Lack of knowledge of OO and its potential for accomplishing the goals of reuse will certainly reduce the chances of success.

Module #4: The Java Language

Since Java is arguably the most popular OO language some introductory information is included here.

Java and C

If your programming background is the language C, you will find the syntax of Java and C to be similar. C is of course not object-oriented and not a collection of classes. Rather C is a collection of functions that are very similar to static methods in Java.

Java and C++

C++ has object-oriented features but it is still possible to do procedure-oriented programming as well. Java is actually a simplification of C++ in many ways. There is no preprocessor or operator overloading and no independent functions, global variables, goto statements, structures or pointers. The unit of programming in Java is the class description. No variables can exist outside of class boundaries. All Java programs are constructed from objects.

Java

Java was developed by James Gosling of Sun Microsystems. The initial motivation in 1991 was to develop a programming language that could be processed by a wide variety of home entertainment devices, each of which has its own processor and machine language. The universal programming language needed was designed so that programs written in the language would be translated into an intermediate language called byte code. Each device would be provided with another program that would then translate the byte code into the local machine language of the device. Although the market originally envisioned for this universal language failed to materialize, the opportunity to transfer these ideas to the Internet, a system of many different hardware platforms connected together into a single network. The result was explosively successful.

- The Java Virtual Machine (JVM) executes the byte code, so Java code can run on any machine to which the JVM has been ported.
- A JVM execution environment contains a collection of base classes that are foundational for applications development
- Memory access is limited to particular controlled locations and variables cannot be used before they are initialized.
- There are no pointers in Java. Memory is managed by reference and therefore cannot be manipulated arithmetically. Pointers are often cited as the source of

many bugs in C and C++ programs. (See PDHOnline course on Software Security for other features of Java that make it a good choice for security reasons.)

- In 2006 Sun Microsystems moved Java source code into the public domain by adopting a version of the General Public License, the same license that covers Linux.

The Java Development Kit (JDK)

The Java Development Kit is available free on the web at the Sun Microsystems website http://java.sun.com/docs/books/tutorial/ . This site includes Java tutorials, tools and demos. The popular "Hello World!" simple Java example is given at http://java.sun.com/docs/books/tutorial/getStarted/cupojava/win32.html#win32-2 for beginners.

After you have run your first program in Java, you may want to move into the Java tutorials on the JDK website. Another particularly interesting resource is a free online textbook called Thinking in Java (http://www.mindview.net/Books/TIJ/) . Other selected websites and textbook resources are given below.

Course Summary

This course provides a conceptual overview of object-oriented (OO) technology. Historical roots, strengths and weaknesses, and the mechanisms of encapsulation, inheritance, composition and polymorphism are introduced. Some basic features of the popular Java language are also discussed. It is intended as a first step for those interested in making the transition from procedure-oriented programming to the object-oriented paradigm. This transition requires a new way of thinking. There are a number of reasons why engineers might want to consider acquiring OO knowledge and skills in a language such as Java. The investment in OO technology has the potential for reducing cost and increasing reliability.

Selected References

1. Armstrong, Deborah J., "The Quarks of Object-Oriented Development, Communications of the ACM, 49, 123-128, 2006. /* The OO literature is replete with variations on the definitions of concepts. This article offers a taxonomy of concept definitions */
2. Bishop, Judith and Nigel Bishop, "Java Gently for Engineers and Scientists", Addison-Wesley, 2000.
3. Budd, Timothy, "An Introduction to Object-Oriented Programming", Addison-Wesley, 2002.

4. Ekel, Bruce, "Thinking in Java", 4th Edition, Prentice-Hall. /* This is the hard copy edition of the free online book listed in the Selected Web Resources below. This book is not only available online, but was written online with many improvement suggestions from the web community as it was being written */
5. Hahn, Brian D. and Katherine M. Malan, "Essential Java for Scientists and Engineers", Butterworth Heinemann, 2002.
6. Sanchez, Julio and Maria P. Canton, "JAVA Programming for Engineers", CRC Press, 2002.
7. Zakhour, Sharon, Scott Hommel, Jacob Royal, Isaac Rabinovich, Tom Risser and Mark Hoeber, "The Java Tutorial: A Short Course on the Basics, 4th Edition (The Java Series) /* This is the Java online tutorial at the Sun Microsystems website in enhanced hard copy form */

Selected Web Resources

Online tutorial of the Java language:
http://java.sun.com/docs/books/tutorial/java/index.html

The Object-Oriented Programming Web
http://www.oopweb.com/CPP/Files/CPP.html

Writing Object-Oriented Software Using C#
http://www.devhood.com/training_modules/dist-a/WritingOOCSharp/?module_id=3

Free Online Book: Thinking in Java
http://www.mindview.net/Books/TIJ/

The Java Development Kit (JDK): A free download
http://java.sun.com

Unified Modeling Language (UML)
http://www.uml.org/