Object-oriented programming is an exceptionally bad idea, which could only have originated in California.

—Edsger Dijkstra

The object has three properties, which makes it a simple, yet powerful model building block. It has state so it can model memory. It has behavior, so that it can model dynamic processes. And it is encapsulated, so that it can hide complexity.

—Trygve Reenskaug, Working With Objects

Well, yes, we’ve all learned about the object-oriented programming paradigm before. But it never hurts to go over some basic definitions so that we’re all on the same page for our discussion about object-oriented analysis and design.

First of all, objects are things. They have an identity (i.e., a name), a state (i.e., a set of attributes that describes the current data stored inside the object), and a defined set of operations that operate on that state. A stack is an object, as is an Automobile, a Bank Account, a Window, or a Button in a graphical user interface. In an object-oriented program, a set of cooperating objects pass messages among themselves. The messages make requests of the destination objects to invoke methods that either perform operations on their data (thus changing the state of the object), or to report on the current state of the object. Eventually work gets done. Objects use encapsulation and information hiding (remember, they’re different) to isolate data and operations from other objects in the program. Shared data areas are (usually) eliminated. Objects are members of classes that define attribute types and operations.

Classes are templates for objects. Classes can also be thought of as factories that generate objects. So an Automobile class will generate instances of autos, a Stack class will create a new stack object, and a Queue class will create a new queue. Classes may inherit attributes and behaviors from other classes. Classes may be arranged in a class hierarchy where one class (a super class or base class) is a generalization of one or more other classes (sub-classes). A sub-class inherits the attributes and operations from its super class and may add new methods or attributes of its own. In this sense a sub-class is more specific and detailed than its super class; hence, we say that a sub-class extends a super-
class. For example, a priority queue is a more specific version of a queue; it has all the attributes and operations of a queue, but it adds the idea that some queue elements are more important that others. In Java this feature is called inheritance while in UML it’s called generalization.\(^1\) Go figure.

There are a number of advantages to inheritance. It is an abstraction mechanism which may be used to classify entities. It is a reuse mechanism at both the design and the programming level. The inheritance graph is a source of organizational knowledge about domains and systems.

And, of course, there are problems with inheritance, as well. It makes object classes that are not self-contained; sub-classes cannot be understood without reference to their super classes. Inheritance introduces complexity and this is undesirable, especially in critical systems. Inheritance also usually allows overloading of operators (methods in Java) which can be good (polymorphism) or bad (screening useful methods in the superclass).

Object-oriented programming (OOP) has a number of advantages, among them easier maintenance, because objects can be understood as stand-alone entities. Objects are also appropriate as reusable components. But, for some problems there may be no mapping from real-world objects to system objects, meaning that OOP is not appropriate for all problems.

## An Object-Oriented Analysis and Design Process

Object-oriented analysis (OOA), design (OOD) and programming (OOP) are related but distinct. OOA is concerned with developing an object model of the application domain. So, for example, you take the problem statement, generate a set of features and (possibly) use cases,\(^2\) tease out the objects and some of the methods within those objects that you’ll need to satisfy the use case, and you put together an architecture of how the solution will hang together. That’s object-oriented analysis.

OOD is concerned with developing an object-oriented system model to satisfy requirements. You take the objects generated from your OOA, figure out whether to use inheritance, aggregation, composition, abstract classes, interfaces, and so on, in order to create a coherent and efficient model, draw the class diagrams, and flesh out the details of what each attribute is and what each method does, and describe the interfaces. That’s the design.

Some people like object-oriented analysis, design, and programming\(^3\) and some people don’t.\(^4\)

So object-oriented analysis allows you to take a problem model and re-cast it in terms of objects and classes and object-oriented design allows you to take your analyzed requirements and connect the dots between the objects you’ve proposed and to fill in the details with respect to object attributes and methods. But how do you really do all this? Well, here is a proposed process that starts to fill in some of the details.\(^5\) We’ll figure out the rest as we go along.

---

\(^1\) Fowler, M. *UML Distilled.* (Boston, MA: Addison-Wesley, 2000.)

\(^2\) Cockburn, A. *Writing Effective Use Cases.* (Boston, MA: Addison-Wesley, 2000.)

