A Method for Your First Object-Oriented Project 17
Ari Jaaksi
Object-oriented literature provides a plethora of methods and notations, many of which are complicated and difficult to learn and use. Ari Jaaksi discusses a pragmatic and simple approach—just two notations and five steps—to finding a method that is right for your project.

Modeling Transport Objects with Patterns 26
Ted Foster and Liping Zhao
Design patterns have provided a way to describe frequently occurring components within public transport object models. They have also delivered solutions for previously difficult problems. The Role (State) and Strategy patterns are used to model transport objects that play multiple concurrent roles and contain multiple solutions.

Semantic-Based Concurrency Control in Object-Oriented Databases 33
Woochun Jun and Le Gruenwald
Supporting concurrency control in OODBs is more complex than it is in its relational equivalents. To overcome this problem, runtime information is used to increase concurrency. The authors look at semantics of methods, nested method invocation, and referentially shared objects; and show how to automate communicativity of methods.

Reduced-Conflict Objects 40
Jay Almarode and Robert Breti
By carefully defining semantics, implementors can prevent concurrency conflicts for particular sequences of operations. While locking solves part of the problem, it is also an expensive solution. Object databases provide better alternatives because they allow a higher level of semantic manipulation; operations on objects can be more sophisticated than simple read and writes.

Re-Engineering Object-Oriented Legacy Systems 45
Eduardo Casais
A drawback to the rising popularity of object-oriented technology is the indiscriminate use of OO mechanisms, and weak analysis and design methods. This leads to large legacy systems that are inflexible and hard to maintain. Eduardo Casais examines O0 re-engineering experiences and identifies ways to better maintain legacy systems.

About Abstract Classes 53
E. Marcos, A. de Miguel, and M. Piattini

Quality Metrics for Object-Oriented Design 56
D. Gillibrand and K. Liu
About Abstract Classes

"...the level of abstraction that we can speak directly affects the size of the problem we can solve."

Object-orientation has increased the level of abstraction with respect to previous paradigms, appearing as a powerful mechanism of knowledge representation that makes the resolution of complex problems easier. In addition, a higher level of abstraction facilitates software reuse, extending this concept beyond the traditional to admit other types, such as knowledge reuse.

Two concepts that contribute substantially to increase the level of abstraction and reuse arise in the scope of object-orientation: On one hand is the concept of classes that do not implement all their methods, but delegate this task to their subclasses; on the other is the concept of noninstantiable classes. To refer to these two concepts, two terms appear in object terminology: deferred class and abstract class. However, there is some confusion between these two terms, and also between the concepts that they represent: On many occasions these terms are considered synonyms, and the concepts they represent, equivalent; some authors, though, only speak of abstract classes, others only of deferred classes, etc. We strive to contribute to the clarification of this controversy, giving a separate meaning to each of these terms, and establishing the difference that exists between the concepts they represent.

ABSTRACT AND DEFERRED CLASSES IN THE LITERATURE

To illustrate the controversy we referred to in the previous paragraphs, we will give examples of some of the definitions that can be found in the literature.

The concept of deferred class was introduced by Meyer, who defined deferred class as "a class that contains deferred routine." An immediate consequence of this definition is that a deferred class cannot be instantiated directly, but only through its subclass. Meyer posed the following deferred class noninstantiation rule: "Create may not be applied to an entity whose type is given by a deferred class."

The concept of deferred class has been extended in the object paradigm, but some authors refer to it under the name of "abstract class." This is the case in C++. In the context of C++, Bjarne Stroustrup writes, "a class with one or more pure virtual functions is an abstract class, and no objects of that class can be created."

However, other authors give different definitions for the concept of abstract class. For Wirfs-Brock et al. "classes that are not intended to produce instances of themselves are called abstract classes." This is also the case with the ODMG-93 model, and with STEP/EXPRESS, where an abstract class is one that can only be instantiated through its subclasses.

Another variant in the use of these two terms is the one employed in the Unified Method (UM), where a deferred class is a noninstantiable class: "deferred class means that it must be subclassed and may not be instantiated (also known as an abstract class, but we use Meyer's term here because it is more descriptive)." The UM considers abstract class and deferred class concepts equal, referring to both concepts as deferred classes.

BUT ARE ABSTRACT CLASSES AND DEFERRED CLASSES THE SAME THING?

If we rest on the definitions that can be found in object terminology, it seems that there is no uniform criteria. In UM there is no difference between the two concepts, and the definition of deferred class as a class that may not be instantiated is attributed to Meyer. Nevertheless, in our opinion, the concept of deferred class in the UM (noninstantiable class) is a direct consequence of the definition given by Meyer for this same term (a class that delegates the implementation of some of its methods to its subclasses). Therefore we cannot say that the concepts of deferred class in the UM and in Meyer are strictly equivalent.

We believe that there is a substantial difference between the two terms in discussion. It is true that abstract and deferred classes are noninstantiable (and so, both have to be subclassed); however,

---

E. Marcos, A. de Miguel, and M. Piattini work in the Departamento de Informática, at the Universidad Carlos III de Madrid. They can be reached by email at {cuca, mpiattin, admiguel}@inf.uc3m.es.
applies. Metrics may be a way forward. The metrics evolving for a particular design can guide the developer to refine the design to meet the quality factors outlined here. There are other software quality metrics such as readability, e.g., the number of commented methods, which this article hasn’t addressed.

References