

Object-Oriented Programming: Helping Beginning Programmers See the Forest and the Trees

BY RICK MERCER

First, two definitions:

Forest: Object-Oriented Design and Programming

Trees: Traditional topics such as Primitive Types, Input/Output, Methods, Parameters, Selection, Loops, Arrays, . . .

Over the past dozen years I have been integrating object-oriented design and programming into the first computer science course (CS1). One theme that has persisted is a focus on retaining the topics that are traditionally taught to first-time programmers. This paper offers some suggestions for integrating object-oriented concepts and technology (the forest) while retaining the traditional topics (the trees).

Object-oriented programs have more than one object. Most CS1 textbook examples start with main methods that do little. And little attention is paid trying to convince students that

- object-oriented programming is about building systems
- object-oriented programs are collections of interacting objects

While the traditional CS1 topics continue to be important, it is certainly possible to acquaint students with object-oriented programming. This could occur at the end of CS1 or in a later course, but it need not wait. It could happen at the beginning of a course where the first assignment is to “find the objects.” Or it could be integrated throughout. Here are some suggestions to help students experience object-oriented programming while learning traditional topics.

It is certainly possible to acquaint students with object-oriented programming. This could occur at the end of CS1 or in a later course, but it need not wait.

1. Define an object as an entity in the memory of the computer that has a name state (the set of values that the object remembers) a set of services that the object can provide
2. Specify a small system and find the objects. Just ask students to find the things in the problem specification. Suggest underlining the nouns. I describe a bank teller system and students invariably find BankAccount. Then I use class BankAccount to introduce objects and classes. I remind students that it is part of something bigger.
3. Emphasize that objects know things and can do things. Objects provide services. You deposit to and withdraw from BankAccounts. They can also know the account holder and the balance.
4. The first classes that students see and write should have more than once instance variable, and they should have at least two types of instance variables. Avoid BankAccounts that are little more than a wrapper around a number. Example programs should have more than one instance of one class. Instantiate the class at least twice.
5. Early programs can be a combination of classes provided by the language's libraries, classes supplied by the instructor or the textbook, and classes implemented by the student. Even a simple program with an input object, an output object, and a string can be sold as object oriented, especially if you throw in domain objects such as BankAccounts. The programs will have several objects working together to get things done. Although there may not be many objects, emphasize that objects are sending and receiving messages.
6. Show an example program that has a larger collection of classes. For example, I show a simple BankTeller program and explain that there are some domain-specific objects written by a programmer for this specific application. These include BankAccount, GraphicBankTeller, BankAccountCollection, and GraphicBankTeller. Also necessary to get this going are objects constructed from the Java library of classes. These include String, TreeSet,

Even a simple program with an input object, an output object, and a string can be sold as object oriented.

Iterator, JFrame, JButton, JTextField, JLabel, Object, ObjectInputStream, FileInputStream, ObjectOutputStream, FileOutputStream, and IOException. Students need not be aware of what all of these objects are responsible for, but it helps for them to be aware that there are many objects interacting with each other to get things done. Here is a screen shot of the program I demonstrate in lecture.

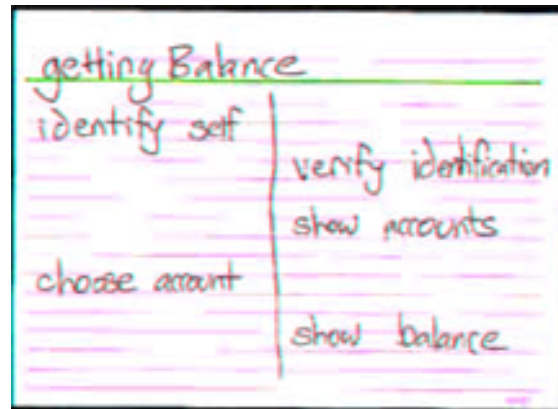


7. Teach methods and parameters in the context of a class. Emphasize that objects—instances of the class that is being written- know things. This can be done through instance variables or information provided by helper objects. Emphasize that objects can do things. This can be through the class's own methods or by asking other objects for help.
8. Teach the “trees” in the context of classes. Methods now require selection or repetition. Some classes need arrays to store a collection of things (a BankAccountCollection class, for example).
9. Unless you are teaching procedural rather than object-oriented programming, avoid static methods in Java. One class with several static methods is procedural—and procedural design rules apply. In C + + , teach classes rather than separate functions.
10. At some point during the first course, introduce object-oriented design. Consider using cases, responsibility-driven design, CRC card development, role playing, and perhaps a team programming project. More on this in the next section.

Actively Learning Object-Oriented Design and Programming

Now I would like to tell you about an in-class activity that I have been doing for several years. It is an active learning exercise to help students partition a system into objects rather than functions. It involves providing a small system specification. It helps to begin analysis with a use case¹. This is a person who is pretending to use the system (any student) speaking to the system (the person who specified the system). This helps everyone to understand what the

system is supposed to do without getting bogged down in the internal design. This can be captured on an analysis card² that is any titled piece of paper that has the user in the left column and the system on the right.



Robert Biddle's Use Case Card from the Workshop "Active Learning for OO Design," OOPSLA, 2000

Then the entire group "finds the objects." The nouns in the problem specification are a source of objects that may become part of the system. We get students to play the role of these objects up at the front of the room. Four, five, or six students work well. We'll use CRC cards³, Rebecca Wirfs-Brock's responsibility-driven design approach⁴, and role playing to determine what objects are needed, what the objects are responsible for, and how the objects communicate with each other. The objects become alive. At the end, we have artifacts that capture some of the responsibilities of the objects.

Now students can see an object-oriented program (the forest) as a collection of interacting objects. And they use the things they have learned (the trees) to implement this system as a collection of interacting objects.

1. Essential Use Cases. Constantine and Lockwood. 1999.

2. "BMW: 2000 Workshop: Active Learning for OO Design." Biddle, Mercer, and Wallingford. OOPSLA, 2000.

3. "A Laboratory for Teaching Object-Oriented Thinking." Beck and Cunningham. OOPSLA, 1989.

4. Designing Object-Oriented Software. Wirfs-Brock, Wilkerson, and Wiener. Prentice Hall, 1990.