

Introducing programming with objects in CS1

Jaime Nino
Computer Science
University of New Orleans
jaime@cs.uno.edu

Abstract

There are many low level details we feel pressured to present in CS1; on top of these, an objects first approach places higher level topics. So a bewildered instructor will ask: what gives? We strongly feel that there are many details that can give (after all, they will spend 2 years on core issues), but there are higher software development issues, supported by OO, that should be used as a basis on which to present “programming”, which we tend to equate with coding of algorithms in a rather unstructured view of the whole. Among these basic high level issues we have the specify-design-implement-test iterative cycle for software development, and the separation of the problem solution from its user interface.

Introduction

The topics to teach programming to beginners are well known, and there is a whole industry behind support books to do so. Lots of topics here: machine issues, names, variables, assignment, expressions, control structures, primitive data types, typing issues, modularization, arrays, records, parameters, arguments, parameter passing, etc. and etc.

Now we are presented with the idea to do objects first in CS1; here we have another part of such industry producing books by the thousands aiming at supporting our teaching efforts. Lots of high-level topics here: objects, classes, instances, relations, static diagrams, dynamic diagrams, polymorphism, inheritance, etc. and etc. And when doing objects first, we must deal with abstraction from the beginning, and issues with designing the structure of a solution; so a clear and distinct activity must be made between designing a solution around objects, and implementing the algorithms for such solution.

Our desire to introduce objects first (we have been doing this since '96) arose from the recognition that upper level students had a low grasp on abstraction and their uses coupled the methodology used to develop programs whose solution could not “easily” fit in their heads. We also recognized that our purpose is to introduce students to software development, not just programming. We had found ourselves in upper level courses in software development trying to undo the poor software development habits acquired early on supported by a poor view of what software is; in a nut-shell their view, method and practices can not easily accommodate changes in software nor do they scale up when complexity increases.

A new foundation

We decided that Object Orientation gives us the ground work in which to introduce software development and algorithm issues. An early definition gives us a framework for all the issues to be addressed:

Software: a temporary solution to a continuously evolving problem.

From the point of view of a system, we break it into the 3 components consisting of the Model, the User Interface, and the data; and lecture on the model, followed many lectures later by the UI and simple data management via flat files. Objects are introduced from the very beginning in our lectures. From the point of view of “programming”, we initially direct the software cycle we want to enforce: identify objects, specify, design, implement, test and iterate over.

Objects are presented first from the client view point, as a set of functions they perform; we partition this functionality into queries and commands. Students are given simple application where to recognize the model (a simple object initially), to identify queries and commands; we proceed with class specification, followed by class implementation, followed by class testing using a given test harness. Overall we use the spiral approach to teaching, where we present more complex objects and more complex algorithms. Half-way through, we present applications which need more than one object.

We use an in-house implemented class List to deal with collections. It is at this point where we introduce iteration. Although we do an standard cover of sorting and searching, we reexamine this topics where sorting and searching end up being function objects to be used with lists.

Strategy

The basic strategy is to provide programming support to objects crafted by the students; both in the lab setting and in programming homework. When dealing with the user interface, we first recognize it as a client object to the model; “programmatically” we first deal with text based objects. Towards the end we deal with the graphical user interface for an application, and introduce MVC.

Other Issues

We use basic UML when and where needed. We identify patterns as well-defined solutions to a problem we encounter. (We have the opportunity to address factory, strategy, state, observer, wrapper, composite, adapter, bridge, command, decorator, null object). When introducing recursion we cover functional as well as object recursion. We use an in-house i/o package which uses commands and queries to get input of basic data types. (User changes the state of the Reader object by reading, and then queries what was read). When using Java, we only cover what is needed of its syntax and semantics and when it is needed. We use a separate class that contains the main(), and creates the needed object(s) and starts them.

We must also note that we deal with an average population with very minimum acceptance requirements, but we teach this intro to CS majors only; we also cover CS1 and CS2 in 3 semesters. The notes, a soon to be published book by Wiley, covers topics for the first two semesters as taught in our setting.

Conclusion

Although we are not fully satisfied with the student’s ability to deal and use abstraction in upper level courses we are much better satisfied than in earlier years. We noticed that although objects are used in CS1 and CS2, we needed to followed up with a course in Object design as there is much to be covered here and for which there is no room in the core courses. We have decided to address these topics within the course used as an introduction to software engineering.