

# Teaching and Learning OO

## Extended Abstract

**Jürgen Börstler**

Department of Computing Science  
Umeå University, SE-901 87 Umeå, Sweden  
e-mail: [jubo@cs.umu.se](mailto:jubo@cs.umu.se)

### 1. OO is different

It is widely accepted that object-oriented development requires a different way of thinking than traditional structured development. Nevertheless, we still use essentially the same pedagogical approaches and examples that were developed to suit the teaching and learning of traditional structured development.

Many introductory programming courses focus on syntax instead of concepts, code instead of models, and construction instead of structure (Becker, 2000). Traditionally program elements are introduced step by step, strictly building on the elements taught before. The structuring of programs into suitable entities is taught at the "next level". This level however is at the heart of object-oriented development.

There are (at least) three important factors that make the teaching and learning of object-oriented development different and more difficult:

- high-level and abstract program elements need to be taught early;
- the basic programming language concepts are highly interrelated, making it impossible to teach them in isolation; and
- there is typically no central control in object-oriented programs, making it difficult to find the "right" location for a piece of code (Guzdial, 1995).

### 2. Principles for Success

A consistent result from many workshops on the topic of Teaching and Learning Object-Oriented is that objects should be taught from the very beginning (Börstler & Sharp, 2003). To make teaching objects from the very beginning possible we propose several guiding principles for the design of introductory programming courses (Börstler et al, 2002; Kölling & Rosenberg, 2001).

**No magic.** Students must be able to understand new topics or example by means of the knowledge gained so far. Unnecessary details and excuses ("you'll understand later") must be kept to a minimum. There are many common examples that ignoring the no magic principle. The popular 'Hello World' example is only the tip of an iceberg (Forum of the CACM, 2002). GUI- or applet-based introductions to OO in Java do often ignore explaining the "magic" of calling the 'paint'-method in a level of detail that can be appreciated by beginner's students.

Early examples must be constructed very carefully. They must not contradict any general rules that we want to instil in our students. Using the Java `String`-class as the main example for discussion the concepts of class and objects should for example be avoided, since `String` objects are immutable (Thimbleby, 1999). Even the `main`-method should be used with care. Students do not easily understand where it is called from and why it has parameters.

**Easy-to-use tools.** Students need easy-to-use tools that support object-oriented thinking and enable them to focus on the problems at hand. Such tools should furthermore integrate a wide range of development activities, like modelling, coding, testing, and program execution.

**Strategy.** Students should be taught a systematic way to develop software. A simple methodology for analysis and design will help them to understand a problem and evaluate alternative solutions *before* starting to code. We use CRC cards and scenario role plays as tool for early "testing" (Nordström & Börstler, 2002).

**Practise, practise, practise.** Programming is a skill and skills are learned by practise. Lectures should be accompanied and followed up by examples and exercises. For the first few lectures it might be advisable to provide step-by-step instructions to walk students through successively more complex exercises.

### 3. Properties of Good Examples

From the discussion in the previous section it should be clear that examples need to be designed carefully. Examples should not be constructed solely to exemplify language specific details. They should be of at least the same quality we would expect from our students. Preferably examples should be non-trivial and well designed according to appropriate quality criteria (whatever these might be).

To avoid misconceptions examples should furthermore follow certain more specific criteria (Holland et al, 1997; Fleury, 2000). Good examples should involve multiple classes. At least one class should implement state and behaviour (except printing/drawing) to make clear the difference between objects and variables. For the same reason it is advisable to use attributes of class type (and not numbers or strings only). At least one class should have multiple instances to make clear the difference between classes and objects. Since message passing is at the heart of object-oriented programming there should also be communication between objects and not user I/O only.

### References

- Becker, B. W. (2000). Pedagogies for CS1: A Survey of Java Textbooks. Manuscript, <http://www.math.uwaterloo.ca/~bwbecker/papers/javaPedagogies.pdf>, last visited 2003-10-06.
- Börstler, J., Johansson, T. and Nordström, M. (2002). Teaching OO Concepts—A Case Study Using CRC-Cards and BlueJ. Proceedings of the 32<sup>nd</sup> ASEE/IEEE Frontiers in Education Conference.

- Börstler, J., Sharp, H. (Eds) (2003). Learning and Teaching Object Technology. Special issue of *Computer Science Education* **13**(4).
- Fleury, A. (2000). Programming in Java: Student-Constructed Rules. *Proceedings of the 31<sup>st</sup> SIGCSE Technical Symposium on Computer Science Education*, pp.197-201.
- Forum of the CACM (2002). 'Hello, World' Gets Mixed Greetings. *Communications of the ACM* **45**(2), pp. 11-15.
- Guzdial, M. (1995). Centralized Mindset: A Student Problem with Object-Oriented Programming. *Proceedings of the 26<sup>th</sup> SIGCSE Technical Symposium on Computer Science Education*, pp. 182-185.
- Holland, S., Griffiths, R., Woodman, M. (1997). Avoiding Object Misconceptions. *Proceedings of the 28<sup>th</sup> SIGCSE Technical Symposium on Computer Science Education*, pp 131-134.
- Kölling, M. and Rosenberg, J. (2001). Guidelines for Teaching Object Orientation with Java. *Proceedings of the 6<sup>th</sup> Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education*, pp. 33-36.
- Nordström, M., Börstler, J. (2002). Object-Oriented Analysis and Design with CRC-Cards. In Swedish, Technical Report UMINF-02.19, Department of Computing Science, Umeå University, Sweden.
- Thimbleby, H. (1999). A Critique of Java, *Software—Practice and Experience*. **29**(5), pp. 457-478.