Teaching Polymorphism Early

Joseph Bergin
Panel Chair
Pace University
One Pace Plaza
New York, NY 10038 USA
1-212-346-1499
berginf@pace.edu

Eugene Wallingford
Panelist
University of Northern Iowa
Cedar Falls, IA, USA
1- 319-273-2618
wallingf@cs.uni.edu

Michael Caspersen
Panelist
University of Aarhus
Aabogade 34
DK-8200 Aarhus N, Denmark
45-2338-2067
mec@daimi.au.dk

Michael Goldweber
Panelist
Xavier University
3800 Victory Parkway
Cincinnati, Ohio 45207 USA
1-513-745-3936
mikeyg@cs.xu.edu

Michael Kölling
Panelist
University of Kent
Canterbury
Kent, CT2 7NF, UK
44-1227-827570
m.kolling@kent.ac.uk

ABSTRACT
Is it possible to teach dynamic polymorphism early? What techniques could facilitate teaching it in Java. This panel will bring together people who have considered this question and attempted to implement it in various ways, some more completely than others. It will also give participants an opportunity to explore the topic and to share their own ideas.

Categories and Subject Descriptors
D.1.5 [**Programming Techniques]** Object-Oriented Programming.

General Terms
Algorithms, Languages.

Keywords
Object-Oriented Programming, Polymorphism, First Course.

1. INTRODUCTION
Dynamic polymorphism is an important component of object-oriented programming in languages like Java. The question then becomes when to teach it, early or late. If a decision is made to teach it early, then we must discover how to do it effectively. This panel will discuss these issues. Though all members generally believe that polymorphism can and should be taught early, each has different ideas about how early and how to do it. Some are, indeed, still exploring this issue. Java, in particular, opens the way to teach dynamic polymorphism before teaching inheritance, via interfaces. On the other hand, teaching polymorphism may require larger examples than people are comfortable using early in a curriculum. Can this be avoided, or if not, leveraged? Elementary design patterns can help, but how. Tools and libraries can help, but again, how. What would occur if we could successfully teach polymorphism very early? These are deep issues that affect how we teach the early courses. The panelists will make very short (5 minute) presentations and most of the available time will be available for an open exploration with the audience on the issues. It is hoped that the panel can open the way for a community to further explore and develop the ideas presented here.

An attempt will be made to capture ideas for presentation on a web site.

ITiCSE'05, June 27–29, 2005, Monte de Caparica, Portugal.
ACM 1-59593-024-8/05/0006.

2. PANELIST POSITIONS AND BIOGRAPHICAL SKETCHES

2.1 Joe Bergin
Joe Bergin believes, following the Early Bird pedagogical pattern, that polymorphism, being the key idea of OO, needs to be taught first, not just early. In this way the students will have the most opportunity to practice and reinforce the ideas. In fact, polymorphism should be taught even before you have code (syntax) to support it, using metaphor and role-play. Polymorphism is an easy topic to explain and it is easy for the students to grasp. It does, however, have deep implications that require a certain thought process that affects how you design programs.

Joe has been teaching since 1972 and objects since 1985. He is the author of four books on aspects of object-oriented programming. The latest is *Karel J Robot: A Gentle Introduction to the Art of Object-Oriented Programming in Java*, with Stehlik, Roberts, and Pattis.

2.2 Eugene Wallingford

Polymorphism serves as the foundation of nearly every pattern of object-oriented programming. As such, students should learn and use polymorphism early and often, so that they can develop a deep understanding of the more complex design ideas that follow. By focusing on message passing from Day 1 of CS1, students can grow into a programming model in which object relationships and communication are the centerpiece of program design, rather than algorithmic patterns.

Fortunately, students can understand and use polymorphism on Day 1 of CS1. They see it in the real world, and they can readily learn to implement the idea in code. The real challenge in teaching polymorphism early lies in identifying and using programming examples in which polymorphism plays a natural part of the solution.

Eugene Wallingford has been writing object-oriented programs for more than fifteen years and teaching object-oriented programming for more than ten. He is a long-time advocate of pattern-driven approaches to learning programming.

2.3 Michael Caspersen

Michael E. Caspersen believes in a model-driven and systematic approach to object-oriented programming where the progression in the first course is shaped by complexity of problems and conceptual models of the problem domain rather than the traditional progression defined by complexity in the programming language. To facilitate a jump start of the introductory course, and in accordance with the Early Bird pedagogical pattern, Michael uses an objectified version of turtle graphics which allows for early (i.e. from day one) exposure of key object-oriented concepts such as class, object, state, behaviour, inheritance, polymorphism, class model, control flow, parameterisation, design by contract (specifications), etc. The use of objectified turtle graphics is combined with the use of LEGO robots to demonstrate a nifty example of polymorphism.

Michael has been teaching since 1984 and objects since 1989. He is the author of two books on programming (in Danish) and several papers on teaching programming published at SIGCSE and ITiCSE.

2.4 Michael Goldweber

Instead of objects first/early or polymorphism early I believe that the initial focus should remain on (return to?) algorithmic problem solving. From this perspective one could argue that polymorphism is more important than objects/encapsulation and therefore deserves greater treatment early on. The difficulty of this is that to focus on algorithmic problem solving one usually uses many small diverse algorithmic/programming tasks to explore different ideas, build confidence, etc. Good examples of polymorphism are, on the other hand, usually larger in scope than what many like to tackle in the introductory course(s). Unfortunately, the traditional algorithmic-focused problems one can realistically expect introductory students to solve as programming (or design) problems do not lend themselves to worthwhile (i.e. unforced) examples of polymorphism.

Mikey has been teaching since 1990 and objects for over 10 years. While he has not (yet) authored any introductory texts he has been involved in many innovative experiments revolving around the introductory course sequence.

2.5 Michael Kölling

Michael Kölling believes that polymorphism is not the very first thing to be covered (since not everything can be first), but one of the important early topics. He also thinks that the fundamental principle of polymorphism can be understood quite easily using general intuition, if presented with the right examples. BlueJ, a software tool for teaching OO principles, has been designed so that polymorphic behavior may be illustrated easily: Interactively invoking the same method on different objects can exhibit different behavior, and objects of distinct (sub-)classes may be passed to a method expecting the superclass as a parameter. Thus, the difference in behavior due to polymorphic dispatch can be directly observed.

Using these tools, an intuitive understanding gained, for instance, from role play, can easily be transferred to Java objects.

Michael Kölling has studied object-oriented systems since the late 1980s, and has worked on developing pedagogical tools for teaching object orientation since the mid 1990s. He is one of the developers of BlueJ.