Early Introduction of Inheritance Considered Harmful

Rudolf Pecinovsky[1, 2]

[1] ICZ, a.s., Hvězdova 1632/2a, 142 00 Prague 4
[2] University of Economics, Prague, Winston Churchill Sq. 4, 130 67 Prague 3
rudolf@pecinovsky.cz

Abstract. Most textbooks about object-oriented programming start explaining the OOP triumvirate, inheritance – abstract classes – interface, with inheritance and overriding, then continues with abstract classes and ends with interfaces and their implementation. The article explains why the *Design Patterns First* methodology changes this order of explanation. It suggests explaining the interface at the beginning of course and not continuing with inheritance of classes until the students include active incorporation of interfaces in their design. The article gives reasons for this changed order, expresses its benefits and accompanies this explanation with some examples of students' assignments demonstrating usage of these principles.

1 Introduction

Almost all introductory textbooks on object-oriented programming we have met start the explanation of object-oriented constructions with an explanation of inheritance between standard classes including method overriding. They continue by explaining abstract class as a special sort of class, which can have methods that are only declared and whose implementation is left to child classes. The explanation of these principles ends with introducing the concept of interfaces, which are however, often presented as Java's substitution of C++'s multiple inheritance. The only exception we have met is in [4] (and naturally in [9] and [10]), where the explanation starts with interfaces and only then continues with the explanation of inheritance and the principles of method overriding.

This standard order of explanation breaks the *Early Bird* pedagogical pattern ([2]) that says: "*Organize the course so that the most important topics are taught first.*" In addition it complicates the introducing and especially exercising of two important object oriented design principles:

- Programming to an Interface, not an Implementation
- Favoring object composition over class inheritance

It is not so wise to start with the explanation of inheritance and after some chapters to continue by suggesting using this construction as seldom as possible and even then using it with special care. Moreover often it is possible to favor usage of the interface

and composition over inheritance (not counting textbooks whose authors forget to mention it).

Similarly it is not best to attempt to put the explanation of interface concept near the end of the programming part of the textbook[1] and then declare that the usage of interfaces belongs to the main OOP principles. If we want to put some knowledge deep into the student's blood, we should place the explanation of the subject as early as possible to give students enough time and opportunity to absorb it.

2 Methodology *Design Patterns First*

In our courses we have modified the standard *Object First* methodology presented e.g. in [1] into slightly different methodology which we name *Design Patterns First* ([6]). This modified methodology orders the explained subjects in such a way as to introduce the key principles as soon as possible and so offer sufficient opportunities to learn, practise and absorb them.

This methodology divides the whole course into two sub-courses: the basic course and the continuation course. In courses at university we combine them into one one-semester course, in the high-schools each of them take one year.

2.1 The Basic Sub-course

The basic sub-course divides the explanation into three stages:

1. In the first stage the students don't program. They work in the interactive mode, where the user acts as one of objects in the program. It sends messages to the other objects and checks their responses. In this stage the students familiarizes with the basic principles of OOP without distracting them with some syntactic rules that they would otherwise keep in mind. In this stage the students also acquaint themselves with first design patterns, interfaces and inheritance of interfaces.
2. We follow by teaching the fundamental syntactic rules. The students become familiar with basic syntactic rules of the used language and learn how to define the simple classes, interfaces and enumeration types. We revise all the rules and constructs explained in the first stage and show, how they should be encoded into a program.
3. The third stage assumes that students have mastered the basic syntactic rules excluding the inheritance of classes. Here we solve more complicated problems needing more sophisticated ways of particular objects cooperating and also more sophisticated architecture of the designed program.

[1] We notice that programming textbooks are divided in two parts: the programming part, which explains the basic programming rules, and the technology part, which presents the libraries and its classes and methods.

2.2 The Continuation Sub-course

The continuation sub-course further deepens the students' knowledge of the syntax of the used language together with their skills how to design the solution of various problems. This sub-course is also divided into three stages:

1. The first stage begins with short summary and revision of the subjects discussed in the previous course. We revisit some design problems which we met before while solving some tasks. We show that many of them can be solved by the introduction of abstract classes, which we present as hybrids between standard (concrete) classes and the interfaces. The hybrids that "want" to combine the ability of both therefore get some restrictions from each of them.

   In explaining the inheritance of classes we don't explain for some time the possibility of overriding concrete parent's methods. We only show, that class inheritance, which means the inheritance of implementation, allows taking the common code out of the child classes and placing it into the parent class.
2. The second stage closes the subject of inheritance with a detailed explanation of class inheritance, which we explain as a special kind of object composition where the object includes the parent object together with its interface. Here we also explain all the common problems appearing when some class inheritances are not designed properly.
3. The third stage finalizes the course with explanation of remaining key parts of the standard library together with some further design principles.

3. Conclusion

In this paper we have noticed that the classical order of explanation of key object-oriented constructions explaining first the inheritance with overriding, second abstract classes and finally interfaces, is not optimal. It breaks the *Early Bird* pedagogical pattern and complicates the introduction and especially the exercising of important object oriented design principles.

We have shown the order that is suggested by *Design Patterns First* methodology, and explained why we believe that the suggested order of explanation is much better. We have described some benefits of this approach and we have presented two assignments for our students to solve.

We have verified in practice that using this methodology we can teach children even from as young as 12 years. These very young students absorb the matter very quickly, mostly with fewer conceptual problems than some experienced professional programmers, because the children don't need to incorporate the new information into their previous knowledge and change their habits.

This methodology is used in a broad spectrum of courses from courses for young children through the high school and universities courses to courses for adult professional programmers. In all the mentioned courses we have better results than we had before applying this methodology.

References

[1] BARNES, D., KÖLLING, M. *Objects First With Java: A Practical Introduction Using BlueJ (2nd edition)*. Prentice Hall, 2004. ISBN 0-131-24933-9.

[2] BERGIN, Joseph: Fourteen Pedagogical Patterns. *Proceedings of Fifth European Conference on Pattern Languages of Programs.* (EuroPLoP™ 2000) Irsee 2000.

[3] GAMMA, Erich; HELM, Richard; JOHNSON Ralph; VLISSIDES, John. *Design Patterns. Elements of Reusable Object-Oriented Software.* Addison-Wesley, © 1995. 396 s. ISBN 0-201-30998-X.

[4] HORSTMANN, C.: *Big Java, 2nd Edition*. John Wiley & Sons, Inc., © 2005. ISBN 0-471-69703-9

[5] KÖLLING, M. ROSENBERG, J.: Guidelines for Teaching Object Orientation with Java, *Proceedings of the 6th conference on Information Technology in Computer Science Education (ITiCSE 2001)*, Canterbury, 2001.

[6] PECINOVSKÝ Rudolf, PAVLÍČKOVÁ Jarmila, PAVLÍČEK Luboš: Let's Modify the Objects First Approach into Design Patterns First, *Proceedings of the Eleventh Annual Conference on Innovation and Technology in Computer Science Education,* University of Bologna 2006.

[7] PECINOVSKÝ Rudolf: Výuka programování podle metodiky Design Patterns First. *Tvorba softwaru 2006 – sborník přednášek*. ISBN 80-248-1082-4.

[8] PECINOVSKÝ Rudolf: Aplikace metodiky Design Patterns Fist. *Objekty 2006 – sborník příspěvků desátého ročníku konference*, ČZU, Praha 2006. ISBN 80-213-1568-7.

[9] PECINOVSKÝ Rudolf: *Myslíme objektově v jazyku Java – kompletní učebnice pro začátečníky, 2. aktualizované a rozšířené vydání.* Grada Publishing, 2008. ISBN 978-80-247-2653-3.

[10] PECINOVSKÝ Rudolf: *OOP – příručka pro naprosté začátečníky*. Computer Press, 2009, ISBN 978-80-251-2126-9.

[11] SCHMOLITZKY, A.: "Objects First, Interfaces Next" or Interfaces Before Inheritance. In *Proc. OOPSLA '04 (Companion: Educators' Symposium), (Vancouver, BC, Canada, 2004)*, ACM Press.

[12] SCHMOLITZKY, A.: Teaching Inheritance Concepts with Java. *Proceedings of International Conference on Principles and Practices of Programming In Java (PPPJ 2006)* Manheim, 2006