INTRODUCING OO CONCEPTS FROM A CLASS USER PERSPECTIVE[*]

Philip A. Smith
University of Ballarat
Ballarat, Victoria
5327 9237
p.smith@ballarat.edu.au

Geoffrey Boyd
University of Ballarat
Ballarat, Victoria
5327 9239
g.boyd@ballarat.edu.au

ABSTRACT

The use of an object-oriented language as an introductory language is becoming more widespread (Biddle & Tempero, 1998). However, paedogical issues relating to the incorporation of such a language are still not understood properly (Kolling, 2001). Approaches to incorporating an object-oriented language into a teaching program vary greatly. Some approaches avoid the issue of object-orientation by putting emphasis on the procedural aspects of the language (Koffman & Wolz, 1999). Others approach the subject from the perspective of a class developer, especially making use of the appeal graphical user interfaces and applets have for students.

The approach that we take at the University of Ballarat is to introduce students to programming from the perspective of a class user. This approach is facilitated by the availability of BlueJ (Kolling & Rosenberg, 2001), a program development

[*]Copyright © 2001 by the Consortium for Computing in Small Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing in Small Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

environment designed explicitly for teaching object-oriented principles using Java. This paper describes this approach and the students' reactions to it.

INTRODUCTION

The University of Ballarat is situated in regional Victoria, Australia. It is a small University which attracts students from the local area and from a geographically large catchment area incorporating a large part of Western Victoria. A large percentage of these students are considered disadvantaged on account of coming from sparsely populated rural areas which cannot supply many facilities available in the city. The proportion of women attending the University of Ballarat is approximately 50% but this figure drops markedly for computing courses offered by the School of Information Technology and Mathematical Sciences (SITMS).

In 2000, SITMS decided to introduce Java as the introductory programming language for all of its courses. We chose Java because it a true (albeit not pure) object-oriented language, because student feedback told us that they believed it to be a useful language to learn and because of the availability of BlueJ, a novice programming environment developed at Monash University, Melbourne. Using BlueJ, we are able to introduce students to object-oriented principles immediately. We teach our introductory programming unit to approximately two hundred students.

Koffman and Wolz (1999) identified two extremes of using an object-oriented language as an introductory teaching language. The first is to dive headfirst into GUIs and applets, embracing the features of the language. They rightly claim that all but the strongest students would struggle with this approach. The second is to play safe and teach the object-oriented language as a procedural language, ignoring the object-oriented design that the language was developed to promote.

Our School's policy is to introduce object-oriented principles immediately but to also be mindful of the needs of weaker students. To this end we introduce basic object-oriented principles but defer more problematic (and peripheral!) issues such as GUI development until later in the course. Instead of teaching these concepts through the design and development of new classes, we teach them through the incorporation of pre-written classes into Java code. With this approach students learn how to discover the functionality of classes and how to use objects of these classes in their code. We feel that this approach is a straightforward introduction into object-oriented programming. We also feel that we are instilling an important skill into the students. Class reuse is an extremely important part of object-oriented programming.

The main design decisions that were taken with regard to this new unit were:

1/ To separate the roles of class user and class developer. While this distinction is probably a little artificial it is useful as a method of introducing the concepts of class and object.

2/ To insist that students become involved in examining the APIs of a class in order to gain an understanding of its behaviour. This is done by experimenting with the class's methods directly in conjunction with examination of the appropriate Javadoc documentation created for these classes.

3/ To insist that students, when they take on the later role of class developers, write their classes with the needs of class users in mind. As a result they must create adequate APIs and create Javadoc documentation for them.

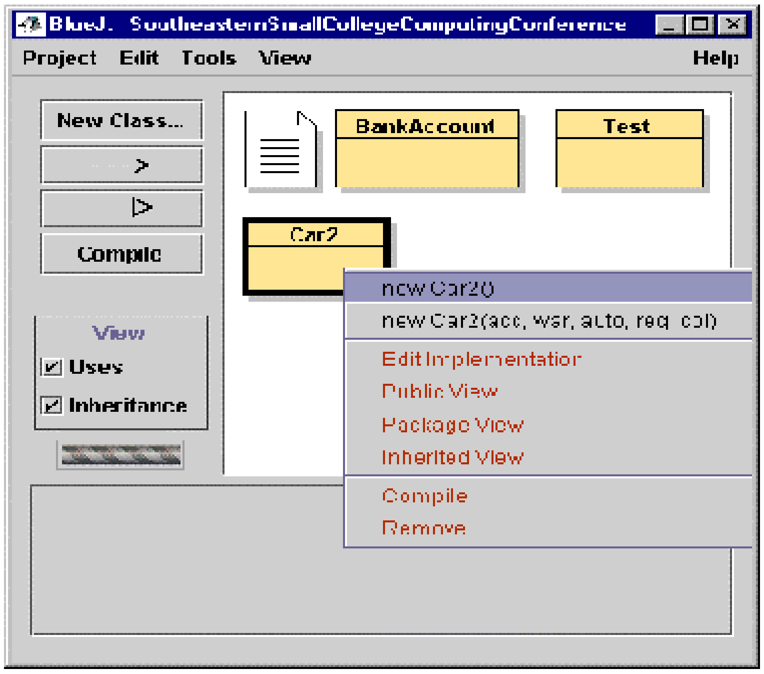
4/ To introduce the details of the syntax of Java after the students have become comfortable with the concepts of classes and objects. The syntax is taught in the context of these object-oriented principles.

THE CLASS USER PERSPECTIVE

The first semester introductory programming unit requires the student to view programming from two perspectives. In the first half of the semester they view programming as a "Class User". In the second half of the semester they view programming as a "Class Developer".

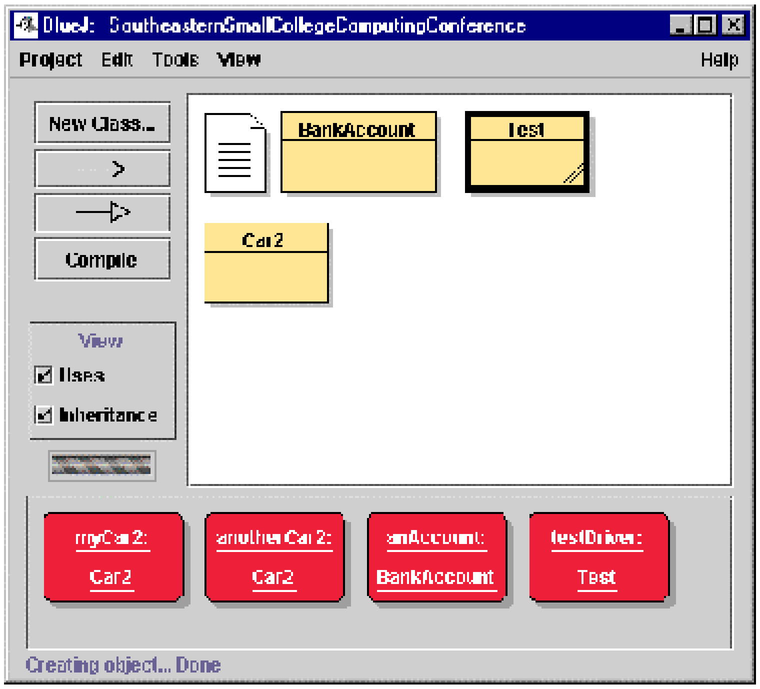
As class users, the students view an object as a "black box" which they come to understand through examination of its API and experimentation utilizing the capabilities of BlueJ. The API is a list of the public methods making up the object's functionality. BlueJ is an excellent tool for promoting this approach because it is simple to learn and gives explicit graphical representations of classes and objects.

BlueJ enables an object's interface to be explored through a pop-up context menu. Public class methods (e.g. constructors) are viewed through a pop-up menu associated with a class itself.

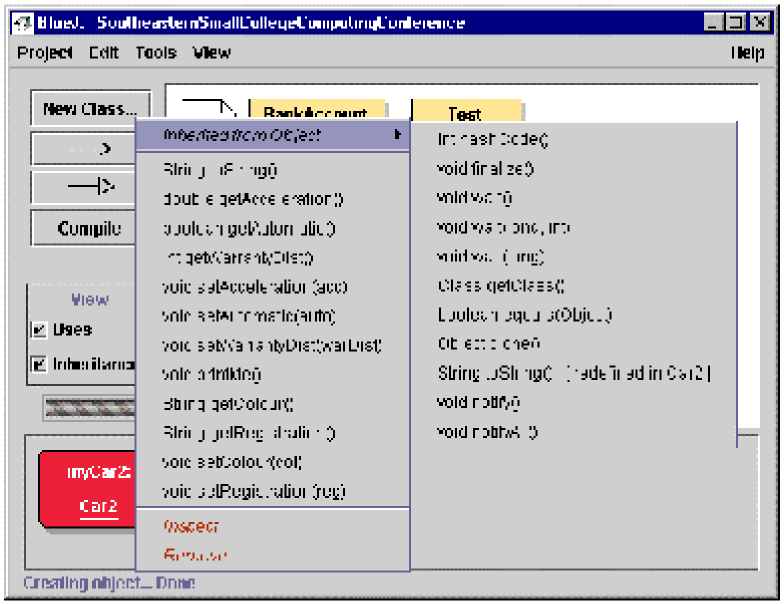


Objects may be instantiated as required via the class's pop-up menu. Objects so instantiated are 'visually placed' on BlueJ's object bench. Consequently, in the one BlueJ window, students see icons representing the classes and different icons representing the objects. The idea of many objects of a single class type is readily visually demonstrated.

The instantiated objects may now be either inspected to confirm their individuality or invited to execute their functionality via their pop-up menu. The pop-up menu also indicates the methods inherited from the superclasses allowing the concepts of inheritance and overriding to be discussed.



Students are taught about the need for adequate documentation. In Java, the process of documentation is partially automated by the Javadoc utility. As class users, the students are provided with API documentation created by Javadoc, which describes both classes developed by the teacher and a sub-set of the classes provided by the Software Development Kit (SDK). They are taught how to use this documentation to understand the behaviour of the given classes. They are also taught how to use Javadoc and, in particular, Javadoc comments and tags to create this documentation for themselves. When they become class developers they are required to use Javadoc to document their own classes. Fortunately, BlueJ facilitates this approach with a menu item to generate the documentation for the package's classes.



When they have been shown how to explore the functionality of a given class, they are then shown how to use these classes in their own Java code. To this end, they are provided with a template Test class which has a single method called testMethod. They import the Test class into their package and write driver code which performs tasks using the classes that they have examined. In this way, they are introduced to Java coding as class users.

These concepts are drawn together in the first assignment. The students are given two classes that they have not seen before. They are not told anything about these classes. They need to create Javadoc documentation of these classes. They can read the documentation for the classes and explore the functionality of the classes using BlueJ. They are then required to apply what they have gathered about the functionality of classes to perform given tasks in the Test class. (While strictly speaking this makes them class developers, the Test class is not intended to be incorporated into other classes by class users. Hence it is appropriate to consider this assignment an exercise as a class user.)

In the second half of the unit the students approach programming from the perspective of a class developer. In this section they are taught principles such as iterative design which they can use to develop their own classes. They are told that they are expected to develop classes with the understanding that they might be used by other programmers acting as class users. Hence, they must give careful consideration to the development and documentation of the class's API as well as come to terms with issues concerning the implementation of the API.

The second assignment is an extension of the first. The students are required to use the two classes which they studied in the first assignment to develop their own new classes. While this makes them, in the strictest sense, class users, the hard work of learning about the functionality of the classes was achieved in the first assignment. Hence it is appropriate to consider this second assignment as being mainly an exercise in class development.

STUDENTS' REACTIONS TO THE UNIT

We felt that students reacted well to the unit and all but the weakest students were able to make progress. We received a lot of positive student feedback about the unit from course evaluation questionnaires that we handed out at the end of the semester. Many felt that we had made the right decision in teaching Java using BlueJ and many expressed the view that the material was enjoyable and understandable.

There were some concerns, however. Some of the students, especially those with previous programming experience, felt that the BlueJ system was too simplistic. One of them declared that BlueJ was "embryonic!". We take this statement as actually being an indication that our choice of BlueJ was correct since simplicity was one of our guiding factors in our choice of a development environment. We wanted to give the students one challenging thing to learn - not two.

Some students expressed disappointment that GUIs and applets were deferred until later in their course. This is understandable since the experience of most students will be of programs with GUIs. They are likely to consider these to be "real" programs. Most of them will have had experience with applets in their dealings on the World Wide Web and be impatient to be able to write their own. We decided to defer the introduction of GUIs and applets until later in the course. It was felt that students should have a thorough grounding in inheritance concepts before learning GUIs and applets. As we mentioned earlier, Koffman and Wolz (1999) claim that introducing these features too early risks confusing all but the strongest students.

CONCLUSION

More and more institutions are using object-oriented languages such as Java as introductory programming languages for their courses. Many of them ignore the object-oriented features of the language and introduce the language in a similar manner to procedural languages. Others jump headlong into discussion of GUIs and applets.

The approach we have used is to introduce object-oriented concepts immediately by placing the student in the role of a class user. We feel that this is appropriate because programmers need to make heavy use of classes written by other programmers and especially classes in the SDK. The availability of the novice program development environment BlueJ, together with the documentation produced by the utility Javadoc, has facilitated this approach.

In this report, we have recounted our experiences of using Java and BlueJ in an introductory programming unit. We feel that the combination of Java and BlueJ has been a success and has provided our students with a straightforward yet useful introduction to programming.

BIBLIOGRAPHY

Biddle, R. & Tempero, E. (1998). Java Pitfalls for Beginners. SIGCSE Bulletin, June, 30(2), 48-52.

Koffman, E. & Wolz, U (1999). CS1 Using Java Language Features Gently. ITiCSE '99, Cracow, Poland, 40-43.

Kolling, M. (2000). The BlueJ Tutorial: Version 1.2. Technical Report Number 2000-01, Monash University, November.

Kolling, M. & Rosenberg, J. (2001). BlueJ - The Hitchhikers Guide to Object Orientation, accepted for publication in Journal of Object-Oriented Programming, 2001.

Rosenberg, J. & Kolling, M. (1997). I/O Considered Harmful (At Least for the First Few Weeks). Proceedings of the Second Australasian Conference on Computer Science Education, ACM, Melbourne, 216-223, July.

SITMS Annual Course Reports (1998).