Michael Kölling
John Rosenberg

BlueJ - The Hitch-Hikers Guide to Object Orientation

The Maersk Mc-Kinney Moller
Institute for Production Technology
University of Southern Denmark
http://www.mip.sdu.dk
Technical Reports 2002, No 2
September 2002
ISSN No. 1601-4219

BlueJ - The Hitch Hiker's Guide to Object Orientation

Michael Kölling
School of Network Computing
Monash University
mik@monash.edu.au

John Rosenberg
Faculty of Information Technology
Monash University
johnr@infotech.monash.edu.au

Abstract

Teachers of introductory object-oriented programming face numerous challenges in designing their courses. In this paper, we discuss some of the challenges and introduce the BlueJ system. We discuss the functionality of BlueJ and show how it can provide solutions to some of the problems that teachers of introductory object-oriented programming courses face.

1 Introduction

Teaching object orientation in first year computing courses has now become commonplace. However, there is no general consensus about how object-oriented programming should be taught. Discussion on approaches ranges from countless details to fundamental educational and technical principles. At the root of the discussion is the fact that we do not have a solid body of experience with teaching object orientation to beginners.

While many institutions and teachers (the authors included) have been teaching object orientation to beginners for some considerable time, the collective experience is nowhere near that gathered for procedural programming.

An additional problem is that many of us were distracted for some time. C++ was, until very recently, the most commonly used language for teaching object orientation. During that time, many teachers struggled with problems peculiar to C++ rather than trying to solve the fundamental questions.

C++ adds a huge amount of external complexity (complexity not inherent in the problem at hand, but introduced by the nature of the language itself) which makes it hard to concentrate on the fundamental issues of object-oriented programming. The proceedings of the ACM Special Interest Group in Computer Science Education (SIGCSE) between 1994 and 1999 are full of papers suggesting ways of coping with the pitfalls of C++ in educational contexts (for example [1, 2, 7]). While these papers provided valuable insights at the time, many of them concentrated on C++ problems and did not address general problems of teaching object orientation. For us as a discipline, this means that we are still very much at the start of learning how to teach object orientation well.

Java, which was only released in 1995, has managed to become widely accepted as a better language to teach OO and has been adopted by the majority of institutions. Java removes much of the external complexity and lets us focus on the real problems at hand.

Many teachers of object-oriented first year courses have reported that they find it more difficult to teach these courses than it was with procedural languages. We believe that this is not due to any intrinsic complexity associated with OO, but is caused by a set of other factors which include:

- A lack of experience and knowledge about how to teach these courses well which results in mistakes being made.
- Many teachers are not as familiar with object orientation as they are with the procedural language they taught earlier and have technical difficulties.
- The teaching materials are not as mature as they were for earlier languages. Many books were produced very quickly with clear compromises in quality. Also, the authors often do not have the necessary teaching experience in this new paradigm themselves.

- The software tools are often not suited to the new paradigm. Many development environments are variations of procedural environments that fail to capture the full potential of object orientation.
- In conjunction with the introduction of object orientation into first year courses there has been a shift in focus from an algorithms centred view to a software engineering centred view. This introduces new topics and new material into the course.
- Technological advances have forced additional material into the course.

In this paper we will introduce BlueJ, an integrated Java development environment that was specifically designed to support teaching and learning object-oriented programming in beginners courses and to address some of the issues identified above. We first further discuss some of the problems being encountered by teachers of OO and then describe the technical aspects of the BlueJ system, the educational context in which it is being used and our experience with using it in class for five semesters. This will provide answers to some of the problems mentioned above, both technical and educational.

2 The problem of the teaching approach

As is always the case with new technologies, there is initially considerable discussion about the "best" way to teach the principles. Generally, teachers agree that the goal is to teach the underlying principles of programming and the programming paradigm at hand, rather than idiosyncratic details of a particular system. But *how* this is best done is far from clear.

When Pascal was the most important teaching language, the focus of such debates was whether procedures should be taught early or late. In the time of object orientation the corresponding argument is held over whether objects should be introduced earlier or later.

It appears, however, that this question has effectively been decided (judging by sheer numbers of public statements). The C++ teaching community fought passionately over this question, while Java teachers seem to have come down clearly on the "objects early" side. At least, almost every new textbook *claims* to introduce "objects early".

There are, however, a whole range of other open questions which include:

- Should GUIs be introduced in the first year? Or even the first semester? If so how early, and using what approach? The proponents argue that a GUI is an intrinsic part of every modern program, that GUIs are motivating and fun for students and that event driven programming is quite "natural". The opponents claim that GUIs present a distraction from the fundamental programming and OO principles and that students concentrate on peripheral issues rather than thinking about application structure.
- Should GUI builders be used? A popular example is the use of Visual Basic for teaching. With some teachers it is extremely popular because it allows the construction of professional looking applications easily. With others it is immensely unpopular because it makes it hard to go beyond the interface and visualise the business logic of the program.
- Should we use applets? This can be either argued as applets being an important topic in itself, but is more often used as a simple way to define and create a user interface. As with the argument for GUIs, there is a reluctance to deal with purely text based programs, because they are perceived as outdated technology. In addition, Java's support for text I/O is not ideal for beginners. Do applets provide the answer for easily creating application interfaces? Critics argue that applets often have no interesting object structure to speak of, and thus do not illustrate the principles well. Proponents counter that applets *can* have proper structure if they are just created carefully.
- Should concurrency be a first year topic? While most teachers currently say no, Java introduces some interesting problems. Java is an inherently concurrent system. Every Java application is a concurrent application. This is especially true when garbage collection and GUIs are part of the material being discussed. Java GUIs are always executed by several system threads, and some of the behaviour cannot be correctly understood without understanding threads.
- Should object-oriented design be part of the course? Since an OO application is a collection of classes, design decisions about the identification of the classes, their methods and their relationships have to

be made. So the question should probably better read: *How much* design should be part of a first year course?

- Should a first year course use custom made libraries? This is most prevalent for I/O, and is caused by the same doubts that initiate the argument for applets or GUIs: Text based programs are seen as unsuitable. In addition, if applets are also viewed as not ideal, and the Java GUI library is deemed too complex, many authors of courses or textbooks address these problems by providing their own, simpler version of classes to provide I/O functionality. Critics argue that this reduces the value because students do not learn standard Java.
- Do data structures still have to be taught in first year courses? Surely, part of a course using Java has to deal with using classes from the standard class library. The class libraries also include a reasonably good framework for collections – implementations of a variety of data structures and algorithms. Is it still necessary to teach students how to implement these? And is the first year course the right place to do this?

These are some of the questions that teachers face when designing a first year course.

3 The problem of time

The move from procedural to object-oriented programming is commonly referred to as a "paradigm shift" in programming. This paradigm shift is responsible, to a large extent, for the problems associated with changing first year courses to OO.

However, what is often overlooked is that there is another paradigm shift: the shift from algorithm based courses to software engineering based courses.

Until a few years ago, the first two semesters in a programming course dealt mainly with basics of programming language constructs, algorithms and data structures. Now, in most modern courses, several additional topics have made their way into first year teaching: pre- and post conditions, correctness, testing, group work, software design, interfaces, documentation, using integrated environments and debuggers and working with large class libraries.

In addition to this, new topics have been introduced as a result of advances in technology: GUI programming, applets, concurrency and exceptions are a few.

Clearly, not all of these are new, and not every course will include all of these. But most up-to-date courses now include some of the topics on this list, while they did not some years ago.

Thus one of the problems with teaching today is just that the amount of material being taught in first year (or expected to be taught) has dramatically increased. This results in a problem of time. The courses were not half empty before the shift happened, and material cannot be added indefinitely without causing difficulties.

One of the problems faced by teachers is simply that there is not enough time to deal properly with all the topics deemed important.

4 BlueJ

In this article, we will introduce the BlueJ environment and demonstrate how it may be used in teaching. Using BlueJ for introductory teaching addresses several of the problems outlined above and points to clear solutions and strategies. While not all of the problems are solved by BlueJ, we believe that it represents a clear improvement.

We will in particular discuss the BlueJ interaction interface, which proposes a new solution to the question of how students should interface with their applications. BlueJ offers a unique mechanism of direct parameterised method calls, which initially replaces the need for other interfaces. The BlueJ mechanisms allow teachers to delay the introduction of other interface technologies such as text based, GUI or applet interfaces until a more appropriate point in the course. The approach not only influences the GUI and applet discussion, but also makes custom I/O libraries unnecessary.

Our discussion will also show how BlueJ facilitates the discussion of object-oriented design, and how it aids in using a true "objects first" approach.

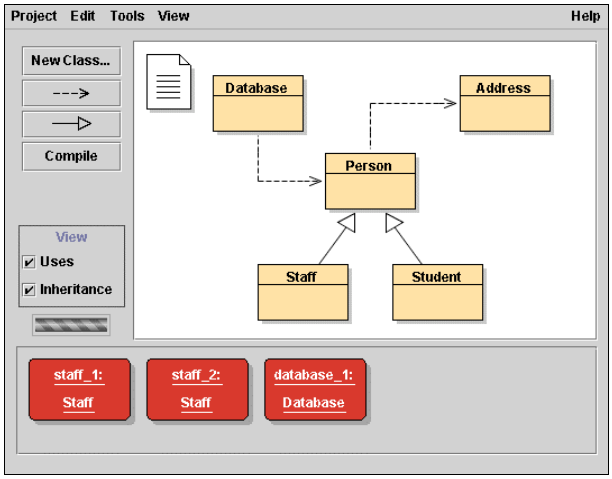

Figure 1: The BlueJ main window

History

BlueJ is a successor of an earlier project called Blue [3, 5]. Blue was an integrated system comprising an object-oriented language and an object-oriented development environment. Blue was developed specifically for teaching object-oriented programming to beginners. The implementation language for this first system was C++. Blue was available only on Solaris and Linux – a port to Windows was never completed.

BlueJ is a system providing an almost identical environment to Blue, but with Java as the supported language. It has the same capabilities as the Blue environment, especially in the areas of interaction and visualisation. BlueJ is, however, a complete re-implementation, this time in Java. One of the benefits of this is cross-platform availability: BlueJ currently runs on all known Java 2 platforms (which includes all Windows and most Unix flavours).

Overview

When a Java project is opened in BlueJ, the main window shows a UML-like class diagram visualising the application structure (Figure 1). Users can then interact directly with classes and objects. A class icon can be used to execute a constructor, which results in an object being created and placed on the object bench at the bottom of the main window (Figure 1). Once the object has been created, any public method can be executed (this is discussed in more detail below).

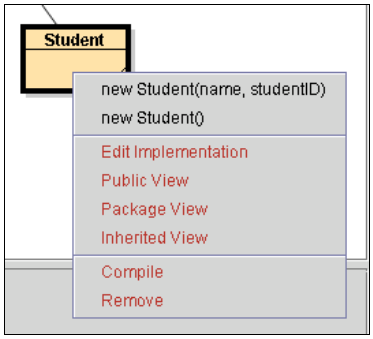

Figure 2: The class menu

A double-click on a class icon opens a text editor that lets users read or edit the class's source code. A simple click on a "Compile" button will recompile all changed classes and execution can start again.

Compile-time errors are displayed directly in the editor by highlighting the corresponding line and showing the text of the error message.

The following sections discuss some of the most important aspects of the system in more detail.

Interaction

Creating objects

Clicking on a class icon with the right mouse button displays a class menu. This contains some environment operations (such as compiling, editing and removing the class) as well as entries to invoke the constructors of the class. Figure 2 shows the menu for a class with two constructors.

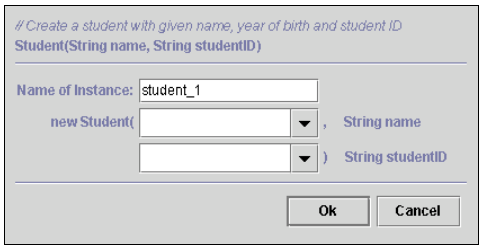

Figure 3: Object creation dialogue

When a constructor is invoked, a dialogue is displayed prompting the user for a name for the object (a default name is supplied) and, if appropriate, the parameters. Figure 3 shows the dialogue for a constructor with two parameters.

Once the dialogue is confirmed, the constructor is executed and the resulting object is placed on the object bench.

Calling methods

A right-click on an object displays an object menu (Figure 4). The object menu contains two environment operations ("Inspect" and "Remove") and an entry for each public method defined in the object's class. Inherited methods are placed in submenus. Selecting one of the methods results in that method being executed. If the method expects parameters, a dialogue similar to that shown for object creation is displayed to let the user specify the parameter values. Parameters can either be typed in (any valid Java expression is allowed) or other objects from the object bench can be chosen. Objects are specified as parameters by supplying their name (a simple click on the object is a shortcut to inserting the object's name into the parameter dialogue).

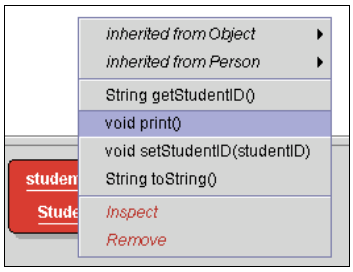

Figure 4: The object menu

If a method has a non-void return type, the result is displayed in a method result dialogue. If the result value itself is an object type, the object can be placed on the object bench for further interaction.

Inspection

The interaction mechanisms allow very sophisticated and detailed testing of classes. Once a method has been written it can immediately be tested without the overhead of writing test drivers. Sometimes, however, a user wants to test a method that alters the state of the object, while no accessor methods are available to directly observe the state. For example, a constructor has just been written, and no other methods are implemented yet, but we would like to test the constructor before proceeding.

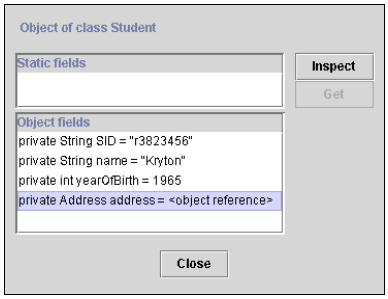

Figure 5: The object inspector

In this case, object inspection can be used to check the effect of the method. The "Inspect" operation from the object menu opens the object inspector, which displays the values of all static and instance fields of the object (Figure 5).

Visualisation

One of the central aspects of the BlueJ environment is the class structure display in its main window. This forces students to recognise and think about structure from the very first time they see a Java program. When showing students the very first example program, it becomes immediately clear that an application is a set of cooperating classes.

Traditionally, one of the hard-to-explain (but very important) issues is the difference between classes and objects, and their relationships. Using BlueJ, a teacher can interactively create multiple objects of a class and inspect and interact with every one of them. The relationship between classes and objects usually becomes clear very quickly. Without the need to talk much about it, students see that the class is used to create objects (as many objects as desired), and that the objects contain data. They also notice that the type of data in each object of the same class is the same, while the actual values are different.

It also becomes apparent that objects are manipulated by invoking operations on them (which they provide) that alter their state. Some operations return information about the state.

Thus, visualising the important abstraction entities of object orientation (classes and objects) and allowing direct interaction with each serves to illustrate the OO concepts in a powerful and easy-to-understand manner without the need for long, dry explanations.

Simplicity

The third cornerstone of the BlueJ architecture (besides interaction and visualisation) is simplicity. The major problem with many existing environments is their complexity. Most environments were designed primarily for professional programmers, and the complexity of their tools overwhelms beginners. Beginning students need different tools than professional software engineers. This issue has been discussed in detail in the context of the original Blue system [4, 6].

BlueJ is designed specifically for beginners. The central aim is that we want to teach about OO programming, not about using a particular environment.

With BlueJ, students can start using the environment on their own ten minutes after having seen it for the first time. After the first tutorial, we never talk about the environment again, and students are able to competently use it. We have traded some advanced functionality not needed in first year courses for ease-of-use, resulting in an environment not necessarily suitable for professional development, but absolutely appropriate for first year teaching.

Other BlueJ features

BlueJ includes a wide variety of other features, which we will not discuss in detail here. Some of the most important are an integrated, easy-to-use debugger, integrated support for *Javadoc* generation, sophisticated support for generating and executing applets and an *export* function that can create executable jar files.

The applet support includes automatic generation of an applet skeleton, automatic generation and loading of an HTML page and the ability to run the applet in web browsers and applet viewers.

Details can be found on the BlueJ web page (see URL at the end of this article) and in the BlueJ documentation (available from that web page).

5 Teaching with BlueJ

Objects first - yes, really!

Teaching with BlueJ enables teachers, for the first time, to use a genuine "objects first" approach. Most books claiming to use "objects first" still delay the first real object interaction until chapter three or four. The reason for this is that it is, in fact, not easy to create an object in Java.

In traditional Java environments, users have to at least

- create a class source file;
- write a "main" method;
- understand (or ignore, or otherwise deal with) various concepts in the main method's signature: static, public, void, parameters and arrays;
- write some Java statements;
- use the "new" construct to create a first object;
- use a variable to store the object;
- use a type for the variable;
- use dot notation to make a method call (possibly with parameters).

Going through this process, until a first object is readily available to interact with, typically involves a considerable amount of non-trivial explanation or a similar amount of hand-waving. Usually it is a little of each. Neither of these is satisfying, though. Either students are swamped with information that the teacher would not normally choose to introduce at this point, or they are sent on their way knowing that starting to program involves concepts that they cannot possibly understand (yet).

In addition to the amount of explanation needed, it is also the wrong type of explanation: the main method itself is not object-oriented. There is no fundamental reason why this code should be in a class – its only purpose is to link the language to the operating system. Thus, teaching about objects begins with an exception to the rule, rather than with a typical example. This is not a beneficial teaching approach. If we want to teach about classes and objects, it does not help to discuss "main".

With BlueJ, students can be given classes that can be instantiated and objects that are used without the need to write a single line of code. This really reverses the order of introduction: students really interact with objects first. Afterwards, when they start to develop an understanding about what a Java object is, they can look at its implementation. They can then start to make changes and observe their effect, eventually arriving at writing their own classes.

Interaction and experimentation

The ease with which objects can be created and methods can be called interactively fundamentally changes the way students approach the programming task. Execution, because it is quicker and more flexible, is used more often in the development process. Thus, easy interaction encourages experimentation. Students tend to test smaller units of code. Since calling a method is easy and little overhead is associated with it, many students test every method directly after it has been implemented. Testing behaviour improves, problems are found earlier and often understanding of concepts is aided by this testing.

The interactive execution also helps to investigate "what if" questions. Our tutors frequently are confronted with questions from students wanting to know what happens when they use certain Java constructs in some specific way. With BlueJ, the answer can very often be "Try it out!". Because trying things out is easy, experimentation is encouraged.

Experimentation, together with immediate feedback, can create a sense of satisfaction and fun that is an immensely valuable aid in teaching. This same immediacy and direct interaction is available in Smalltalk systems and represents one of the strongest arguments that have been brought forward in the literature for the use of Smalltalk as a teaching language.

6 Conclusion

BlueJ offers a unique interface for teaching object-oriented programming to beginners. It visualises class structure, is highly interactive and very easy to use. The BlueJ interface helps to avoid some of the problems commonly encountered with teaching introductory object-oriented programming.

We have seen how the "objects first" approach is supported in a unique way through direct object interaction. The interaction mechanism also suggests answers for a few of the other issues mentioned at the start of this article: GUIs, applets or text I/O do not need to be introduced as one of the first things, since the environment offers an integrated call facility that supports parameters and results. This facility is based entirely on method calls (one of the fundamental concepts to teach), and teaching of other interfaces and I/O can be delayed until students are ready for it.

The question of design is addressed through the class structure visualisation. Having this diagram on the screen at all times facilitates discussion of class design. Discussion is focussed on the fundamental OO principles of classes, objects and methods.

The BlueJ interface is designed to lead teachers' and students' thoughts back to the basics of programming and OO design, instead of distracting them with details. The simplicity of the interface also tries to address the problem of time: While providing a modern tool to discuss important issues, teachers do not need to waste much time discussing the tool itself.

BlueJ has been successfully used by the authors for some time. We also have received very positive feedback from many other teachers using it at various universities around the world.

The BlueJ system and its documentation is freely available at http://www.bluej.org.

References

[1] M. Berman, *et al.*, *Using C++ in CS1/CS2*, in Proceedings of SIGCSE '94, ACM, 383-384, 1994.

[2] M. R. Headington, *Removing Implementation Details from C++ Class Declarations*, in ACM SIGCSE Bulletin, ACM, Nashville, 24-28, March 1995.

[3] M. Kölling, *The Blue Language,* Journal of Object-Oriented Programming, Vol. 12 No. 1, 10-17, March/April 1999.

[4] M. Kölling, *The Problem of Teaching Object-Oriented Programming, Part 2: Environments,* Journal of Object-Oriented Programming, Vol. 11 No. 9, 6-12, February 1999.

[5] M. Kölling, *Teaching Object Orientation with the Blue Environment,* Journal of Object-Oriented Programming, Vol. 12 No. 2, 14-23, May 1999.

[6] M. Kölling and J. Rosenberg, *An Object-Oriented Program Development Environment for the First Programming Course*, in Proceedings of 27th SIGCSE Technical Symposium on Computer Science Education, ACM, Philadelphia, Pennsylvania, 83-87, March 1996.

[7] R. Rasala, *Function Objects, Function Templates, and Passage By Behaviour in C++*, in Proceedings of 28th SIGCSE Technical Symposium on Computer Science Education, ACM, San Jose, Calif., 35-38, February 1997.