Teaching Java with BlueJ – A Sequence of Assignments

Technical Report · September 2002

CITATIONS
4

READS
1,226

1 author:

Michael Kölling
King's College London
104 PUBLICATIONS **2,521** CITATIONS

SEE PROFILE

Michael Kölling

Teaching Java with BlueJ - A Sequence of Assignments

The Maersk Mc-Kinney Moller
Institute for Production Technology
University of Southern Denmark
http://www.mip.sdu.dk
Technical Reports 2002, No 1
September 2002
ISSN No. 1601-4219

Teaching Java with BlueJ – A Sequence of Assignments

Michael Kölling
Mærsk Insitute
University of Southern Denmark
mik@mip.sdu.dk

Abstract

How to teach object orientation in introductory programming courses is still an area not very well understood. Tools, examples and pedagogical issues are regularly discussed by active teachers and researchers. One of the software tools developed specifically to support introductory object-oriented teaching is BlueJ, an integrated environment that allows a different approach to the introduction of object concepts. While it was argued from the first publication about the BlueJ system that this environment lends itself to a different teaching approach, a concrete example of such an approach has not yet been published. In this paper, we will present a sequence of projects and assignments that fully exploits the possibilities of BlueJ and demonstrates the potential inherent in this system. The motivation and learning goals for these assignments is discussed. Teachers may adopt them as they are or use them as inspiration for their own projects.

1 Introduction

Java has now become one of the most popular languages for introductory programming in computing courses. Yet teaching in Java is still not easy, because the pedagogy of teaching object-orientation to beginners has not been fully understood; good examples are hard to find and many software tools currently being used are unsuitable.

Research is currently being undertaken in all of these areas. Researchers are discussing pedagogical approaches [3, 4], presenting assignments and projects (for example [8]), and software tools are being developed.

The tool of special interest to this paper is BlueJ, an integrated software development environment specifically developed for introductory teaching of object-oriented programming. BlueJ has a unique interface that supports a much greater degree of interaction than other environments and allows a fundamentally different teaching approach.

Specifically, BlueJ allows interaction with objects before implementing objects. This leads to a change in the order of introduction of fundamental concepts. Classes, objects, methods and parameters can be introduced as concepts before talking about Java, syntax, variables, types or a "main" method. To exploit the potential benefits fully, a different order of introduction and a different set of assignment projects is needed compared to conventional courses.

This paper builds on previous work on BlueJ. One earlier paper described the functionality and technical aspects of the environment [6], while a second one described a set of abstract guidelines for designing courses with BlueJ [7].

In this paper we present a sequence of assignments built in conformance with the guidelines presented earlier. This assignment sequence differs from those presented by other authors in that it is specifically designed to exploit the facilities of the BlueJ environment.

In section 2, we first give a very brief overview over the BlueJ system. We then go on to present the assignment sequence and discuss the rationale for each project.

2 The BlueJ environment

BlueJ is an integrated Java development environment specifically designed for introductory teaching. BlueJ is a full Java 2 environment: it is built on top of a standard JDK and thus uses a standard compiler and virtual machine. It presents, however, a unique front-end that offers a different interaction style than other environments.

Figure 1 shows the BlueJ main window. A class diagram in the centre shows the application structure. A popup menu on each class icon lets users invoke constructors to interactively create objects of any class. Objects are then displayed on the object bench towards the bottom of the window.

Once an object has been created, any of its public methods can be interactively invoked by selecting it from a popup menu that is provided by each object. Parameters and method results are entered and presented through dialogue windows.

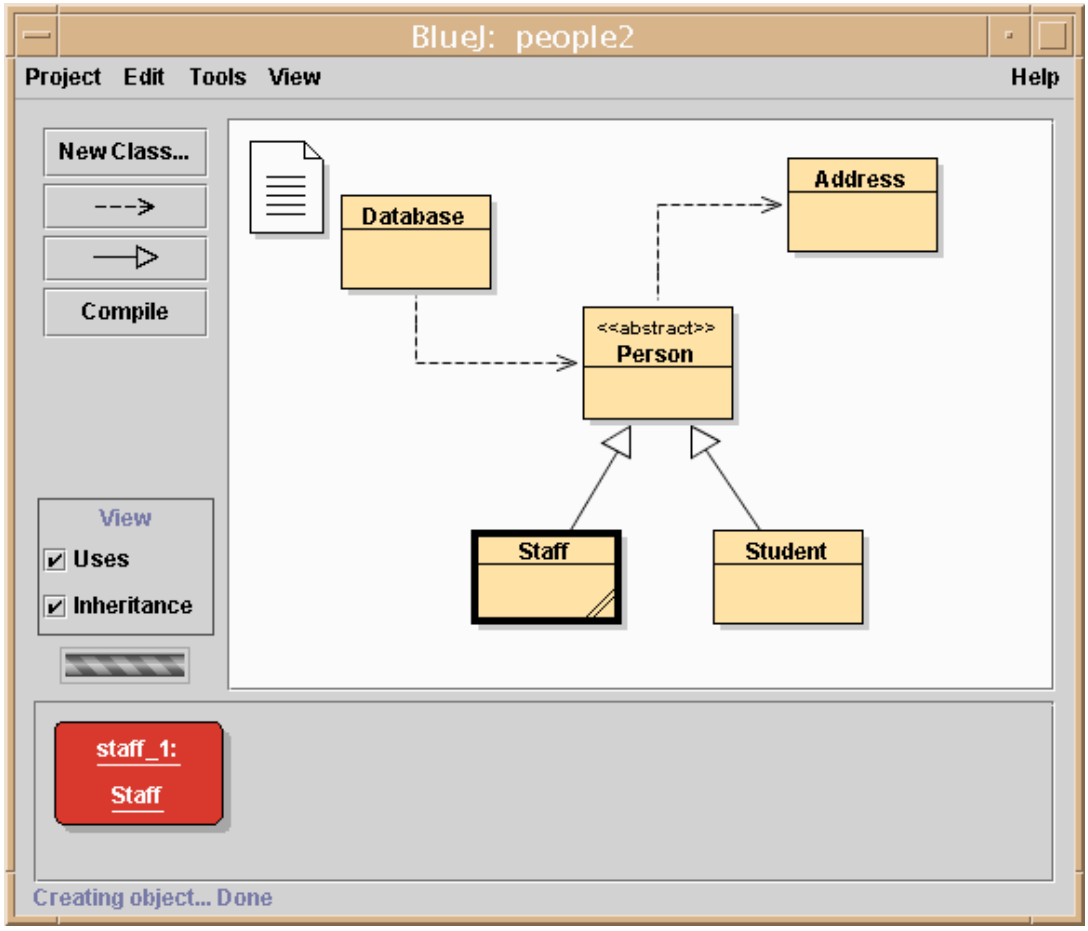

Figure 1: The BlueJ main window

This interaction style is similar in functionality to traditional Smalltalk environments, but more direct through the use if graphical direct interaction techniques.

The environment is carefully designed to be very simple to use. Its major strength lies in the areas of visualisation, interaction and simplicity of use.

More detailed information about BlueJ is available in [6]. The software, documentation and support are available at the BlueJ web site [5].

3 Introducing programming with BlueJ

In an earlier paper [7] we have outlined the principle ideas behind the pedagogy for teaching with BlueJ. The main points were presented as a sequence of eight guidelines for developing programming environments for BlueJ. They were:

Guideline 1: Objects first.

Guideline 2: Don't start with a blank screen.

Guideline 3: Read code.

Guideline 4: Use "large" projects.

Guideline 5: Don't start with "main".

Guideline 6: Don't use "Hello World".

Guideline 7: Show program structure.

Guideline 8: Be careful with the user interface.

In summary, these guidelines suggest a way of teaching that starts by presenting to students reasonably large projects from the start. Students would then be expected to execute, read, modify and extend the projects (in that order). Writing completely new projects from scratch is seen as an advanced exercise.

In the following section, we present a sequence of assignment projects that implements these ideas.

4 The assignment sequence

We have used BlueJ for teaching introductory programming courses in Java in two different degrees (Bachelor of Computing and Bachelor of Network Computing) at Monash University since first semester 1999. BlueJ is used for the first two semesters in each degree. In total, more than 800 students were taught with BlueJ in these courses within that time. The assignment sequence here is used in the Network Computing degree.

The course consists of two lectures and two hours lab/tutorial per week, with 13 weeks per semester. Students see example programs in lectures, do small scale weekly exercises in tutorials and in their own time and complete two to three larger assignments per semester.

This paper describes only the sequence of assignments over the first two semesters. Full descriptions and source code for all assignments is available from the authors web site (see below for address).

The first step: execution

The purpose of the first project is to convey a feeling of the basics to students. These are an impression of objects, classes and methods and of a program as a collection of interacting classes.

For this we use a project called "shapes". This project contains classes for creating circles, squares and triangles, which are represented on screen and can be moved, resized and changed in colour by interactively invoking methods on the separate shape objects.

Students interactively manipulate these objects to create a picture on screen. Note that the manipulation is done via method calls, not by dragging picture objects as in graphics program.

In doing this, students practise creating objects, calling methods and passing parameters. They also get a first hint at types: integer and string types are used as parameters. We also let students inspect objects (that is: view the values of the internal variables). This activity illustrates several important concepts:

- Java applications consist of a collection of classes;
- classes can be used to create objects;
- many objects can be created from one class;
- objects have operations (methods);
- methods may have parameters and return values;
- objects have a state (fields with values that may change through method calls).

Note that students can experiment with objects and get a feel for important concepts without being distracted and held back by Java syntax issues.

Writing Java: "picture"

The next step is to give students an impression of program source and Java syntax. In this activity, they start making small modifications to existing code. We use a project named "picture". "picture" is similar to the "shapes" project, but contains an additional "Picture" class that combines various shapes to draw a picture. Students look at the picture by creating a picture object and invoking its "draw" method. This draws the picture on screen.

We then get students to open the source and find and read the "draw" method. The picture, inside its draw method, creates a few rectangles, triangles and circles, changes their size, position and colours, and thus creates what looks like a simple block painting of a house with the sun in the sky.

Students immediately notice that the source code implements exactly what they have just done interactively in the "shapes" exercise. They can very quickly understand the meaning of the source and make modifications.

We start by giving them simple tasks to do, such as "Make the sun blue". We fairly quickly move to creating completely new pictures. Here, students write their first Java code, consisting of object creation, method calls and parameter passing.

Good students, by the way, very quickly come up with quite sophisticated ideas. The shape objects, for example, have methods such as "slowMoveVertical(int)" and "slowMoveHorizontal(int)" to create an animation effect by moving them slowly across the screen. Students regularly start creating animated pictures in which the sun sets or a ball rolls over the screen.

Implementing methods: Calculator, Blocks and MIF

The next step is for students to extend existing classes by implementing methods or adding their own methods to an existing class. Here, the project typically consists of multiple classes, but all the work the student is expected to do is within one single class. All other classes are provided fully implemented. We use three assignments of this kind, The first one is a "calculator" project. Similar projects have been used by other people. A nice one is described in a paper by Reges [8].

The calculator project has a graphical user interface (completely implemented) and a "CalcEngine" class with method stubs, in which students implement the calculator logic. The CalcEngine class has fields to hold the internal values and methods that are invoked when a number or operator button is pressed. Implementations for all these methods is very simple. Loops, for instance, are not required. Students deal mainly with variables, assignments, simple operations (addition, subtraction), instance fields and return values. One of the main aims is for them to understand the interaction of different objects to make a program work.

There are two more assignments of this style, where students are expected to implement slightly more complex methods in an existing class. The first is called "blocks" and is a partial implementation of a Tetris-like game. Again, students are given several classes, including a graphical user interface, but are expected to modify only a single class.

The last example of this kind deals with image manipulation (named "imageviewer"). Images are represented in MIF - Monash Image Format, which is a simple two-dimensional array of bytes. Students implement a set of image operations, such as "brighter", "darker", "smooth", "pixelise", "flip" or "threshold". This assignment is partly designed to practice loops and other control structures as well as

arrays. On the other hand, this assignment also requires students not only to fill in bodies of existing methods, but also to add complete new method definitions.

This is the last assignment in semester 1. The following two projects are semester 2 assignments.

Adding classes: Zork1

The next step is a project where students create complete classes (again as part of an existing project). Here, we have used a simple text based adventure game called "zork1". A basic framework is given to students that implements different rooms, input of commands and movement through rooms.

Students are asked to invent a game scenario, add items to rooms, the ability for players to carry items (up to a certain weight), etc. The scope for challenge tasks is endless.

One crucial aspect is that some of the tasks clearly require the addition of new classes, the most obvious one being an “Item” class. Students also go through exercises reading and understanding the existing code. They have to make changes in most (but not all) of the classes, but they have to figure out themselves what they have to change and where.

This has been one of the most successful assignments, with surprisingly elaborate student submissions both in inventiveness of story telling and technical implementation.

The ultimate challenge: Do it all

The last step is a project where students work in groups and create a whole application from scratch. This time, only a brief problem description is given, and students have to go through the whole development process, including the class design (with a lot of guidance).

We have used a variety of discrete event simulations as projects. They included a supermarket checkout simulation, a traffic intersection with traffic lights, a lift simulation, emergency evacuation from buildings, a marine life simulation and others.

At this stage of the course, we don’t discuss small scale programming issues very much anymore. The low level code writing is more or less assumed to be mastered by students, and the project serves as a practice ground for applying these skills. The really new and challenging issues at this stage are application design and group work.

Simulations are an ideal example for practising object-oriented design, because almost all objects needed in the program have corresponding objects in the real world, and are very easy to recognise with fairly simple methods. We use the noun/verb analysis and CRC cards [1] for class discovery.

Small scale problems are usually solved by groups internally, while the lecturer and tutor concentrate on discussing analysis, design and group work issues. It is made very clear that the group work aspect is not a coincidental side issue, but one of the important study topics of this course. Well organised group work processes are expected to be set up and documented.

This is the first time students do design, but not the last. In the following year of study, there is a whole subject about analysis and design. We go through their first design with a lot of advice and attention to make sure that all groups arrive at a solution that is implementable within their given time.

This project is by far the longest of the assignment projects. Students are given eight weeks to complete the project, and the deliverables include a report and a demonstration.

5 Discussion

The projects and assignments presented here implement a sequence of activities that introduce the important concepts of object-orientation in a significantly different order than traditional programming courses. One of the unique aspects is a true “objects first” approach: students start seeing and interacting with objects as the very first thing, even before being confronted with Java syntax or source code.

From there on, students go through a sequence of progressively complex activities. They

- make small modifications to existing methods,
- implement complete method bodies where method signatures were supplied,
- add new methods to existing classes,
- add new classes to existing projects,
- and finally create a complete project.

All of this work is done in the context of relatively "large" projects. Students get used to reading and modifying existing code from the very beginning. Many of these activities conform to an educational pattern called "Fill in the Blanks" [2].

Overall, the assignments present a good mix of topics. We get regular feedback after the "imageviewer" project from students amazed that they can suddenly understand and even implement operations that they have seen in graphics packages such as Photoshop. Both the "zork1" and the simulation projects have proven very popular and successful, to a large extent due to their open nature that allows students to take ownership of the project and implement their own ideas.

It would be interesting to test BlueJ and its capabilities in a real problem-based learning (PBL) course. While the above set of projects has some elements of PBL, the whole course is not run in a PBL mode. We suspect that BlueJ and PBL would complement each other in an ideal way. A trial would be interesting.

One shortcoming of this course that became apparent very quickly is the lack of a textbook that follows a similar style of introduction. Among the negative feedback that we obtain from student surveys, the only regularly mentioned issue was that our currently used textbook was considered not helpful enough. We tried to compensate by referencing numerous other sources of information, but for the students this was clearly not the same.

Another area of possible improvement is group work support in BlueJ. Group processes could probably be significantly improved if the environment allowed some form of concurrent modification of a group project by multiple group members.

Downloads

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

The assignments are available from the author's web site at http://www.mip.sdu.dk/~mik.

Acknowledgements

The "imageviewer" project has been significantly improved after discussions with Gordon Royle from the University of Western Australia, who offers a similar project to his students.

References

[1] K. Beck and W. Cunningham, *A Laboratory For Object-Oriented Thinking*, in OOPSLA '89 Conference Proceedings, ACM, New Orleans, Louisiana, 1-6, 1989.

[2] J. Bergin, *Fourteen Pedagogical Patterns for Teaching Computer Science*, in Proceedings of the Fifth European Conference on Pattern Languages of Programs (EuroPLop 2000), Irsee, Germany, July 2000.

[3] D. Buck and D. J. Stucki, *Design Early Considered Harmful*, in SIGCSE 2000 Proceedings, ACM, Austin, Texas, 75-79, March 2000.

[4] F. Culwin, *Object Imperatives!*, in SIGCSE 1999 Proceedings, ACM, New Orleans, 31-36, March 1999.

[5] M. Kölling, *BlueJ - Teaching Java,* web document at http://bluej.monash.edu, Monash University.

[6] M. Kölling and J. Rosenberg, *BlueJ - The Hitch-Hikers Guide to Object Orientation,* accepted for publication in Journal of Object-Oriented Programming, 2001.

[7] M. Kölling and J. Rosenberg, *Guidelines for Teaching Object Orientation with Java*, in submitted to ITiCSE 2001, ACM, Canterbury, UK, June 2001.

[8] S. Reges, *Conservatively Radical Java in CS1*, in SIGCSE 2000 Proceedings, ACM, Austin, Texas, 85-89, March 2000.