Java: Language and Environment as Important Tools in Object-Oriented Design and Programming Instruction

Andrew H. Schulak

Abstract

As noted in *Requirements For A First Year Object-Oriented Teaching Language* [Kolling95] the language used in an introductory programming course is very important. The language should be robust enough to allow students the complete feel of an object-oriented language and also be able to pick up on other object-oriented languages, and should be light enough so that the students are not bogged down with language artifacts which are not conducive to learning object-oriented design.

Two years ago we, at Brown University, switched to the Java programming language for our introductory course. As in [Kolling95] we also conducted a language survey. Our results pointed us to the then brand new language Java. Java meets more of the requirements posed by [Kolling95] then any of the languages surveyed by them or us. Some of the better points of Java that led to its adoption have been its portability, robustness, cleanliness and range of use.

In this paper I will discuss how we have used Java in the last two years, how we have succeeded and how we have failed and where I think we can go with it. I will also discuss how we teach object-oriented design and programming should be modeled in a programming environment that students use to create and program their projects.

Current Uses of Java

We currently use the Java programming language as the main tool to teach object-oriented design. Our approach to teaching programming is very different from most casual approaches.
Other universities have focused in on teaching straight programming (i.e. syntax) in their introductory courses. Design is either something to be picked up on indirectly or is left for later courses down the road. Our approach is to make design the most important aspect of programming, syntax is picked up during the process.

Teaching Design With Java

As mentioned our primary concern in teaching an introductory programming course is object-oriented design. Within the first four weeks of class students have received all the knowledge necessary to understand and implement full object-oriented programs. Where we differ from most other introductory programming classes is that we *do not* focus in on syntax at all in these first four weeks.

At first this might seem very difficult. Such languages as C++, where one needs a good knowledge of the, at times quirky, syntax would not allow such a direction to be taken. However with Java the

syntax is simple enough that we can fully teach object-oriented design and programming without focusing in on syntax.

One thing must be noted here, however. That is that the students are given a very comprehensive set of graphical libraries to use for their assignments. In the beginning assignments are scaled such that the amount of real syntax (parameter passing, loops, conditionals, etc.) used is non existent. Support code is written such that syntax matters which might be relevant to solving a problem are taken care of a level above the students view. As programs, and knowledge of Java, progress the amount the support code does for the student decreases. Eventually students are writing code that uses a minimal amount of support code, that is to say code which hides parts of the language from the student.

Design is initially given to them in the form of help sessions. In these sessions TAs walk the students through an example design session. “Wrong turns” as well as “right turns” are taken throughout the sessions to demonstrate to the students how designs are created. In smaller sessions, called section, students become involved in group design sessions and start to fully realize how good designs, and bad designs, emerge. Eventually students are left to design their own programs with no TA intervention.

Another feature are in class demos. Every lecture in our introductory programming class has an accompanying demo or so. Demos, written in Java as well, are used to demonstrate the concepts being taught in the lecture. Full code is given out to students as well so as to perpetuate the understanding of how code is put together to embody the constructs of object-oriented design. Since demo code is available to the students, the demos must also follow this line of complexity progression. That is, in the beginning the code that the students see should not contain any concepts or artifacts that they have not seen. This presents a problem because in order to present interesting and motivating demos we have to use all of Java’s robustness. Writing demos that do interesting things yet hide constructs not familiar to the students has become something of a black art amongst the TA staff that works on the course.

Java Interfaces

For our needs Java is a perfect match for us. Java does not support multiple inheritance (as does C++) and provides for us the idea of *interfaces* native in the language. Inheritance is usually taught as an “is a” relationship. This is usually the only relationship (besides the more common “has a” and “knows about” relationships) that students learn about and use. Having native interfaces in Java allows us to teach the “acts as” relationship. Using interfaces, objects can take on capabilities of many things besides the one thing that they happen *to be*. *Figure 1* below shows a class, SmallUFO, which happens to inherit from class UFO, *but also* takes on the capabilities of something which can Move. It thus appears as being two things at once. In fact, Java interfaces can be used more than once and a class may implement more than one interface at any given time. This language construct allows us to teach a more robust idea of design to first year students who have never programmed before. Also, because it is native in the language, we do not have to spend extra time motivating an abstract design idea that has no basis in “real code.” Interfaces provide an intuitive, easy to teach work around to multiple inheritance.

Figure 1: ***Class implementing an interface takes on capabilities of the interface as well as the class it inherits from***

```
public class SmallUFO extends UFO implements Mover {

  public SmallUFO(GP.Container container) {
    super(container);
    _body = new UFOBody(container,this);
    _bubble = new UFOBubble(container);
    _location = new Position(500,500);
    this.SetPosition(_location);
  }

  public void Move(Position placeToGo) {
    this.FlyToLocation(placeToGo);
   }

}
```

The Wave Programming Environment

Currently students create their programs using the Wave programming environment created by Ben Boer at Brown University. Wave combines the hyperlink capabilities of the Netscape browser and the powerful editing capabilities of the Emacs editor to provide an easy to use environment for beginning students.

Students beginning a program select a program stencil from a menu of projects in the Wave console. This brings up some stencil code for the assignment, if any is needed, in the Emacs editor, and brings up documentation for any support classes in the Netscape browser. Students can browse documentation, select files to edit, and compile their code from the Netscape window. among other things.

Some of the benefits beginning programmers get from using Wave are that no knowledge of UNIX, the system our lab machines run on, is necessary, documentation for otherwise unfamiliar support classes is provided and easy to locate, commands to build and run code are provided in an easy to use interface.

Currently Wave has been used for two years and has received very encouraging reviews. Students who have never programmed before, and those who have, find it easy to use, fast and helpful in program construction.

Successes In Teaching With Java

Our introductory programming class, CS15, prides itself by taking a class of mostly college freshmen and graduating about 85% of the starting population. What graduation from CS15 means is that the student has received a comprehensive study of object-oriented design and programming. has learned a substantial windowing toolkit (not AWT), and is able to write fully object-oriented programs of up to 3,000 lines of code.

As Java is a very marketable language to know, we have also graduated many of our students on to good internships with such companies as Marimba and Comunica. It is also important to note that because of our methodology in approaching object-oriented programming (and because of our whole department) many of the graduates from the department are note worthy for their excellence in object-oriented programming ability.

Aside from these issues we feel, as educators, that Java has been, and will continue to be, a very good teaching language. It is robust enough that any student who wishes to go on and learn other object-oriented languages can and will have no problem with them, and it is light enough that it's inherent nature does not get in the way of our teaching of design.

Failures in Teaching With Java

One failure of how we teach object-oriented programming really has very little do to with Java at all. Many people, including TAs, students and faculty, have criticized us for teaching design and object-oriented concepts first instead of following the "natural" course and teaching syntax first. Having gone through the system, and teaching it for two years, I can say that it does indeed have its drawbacks, which will be outlined shortly, but that it does have merit.

Object-oriented programming is more than learning a language or syntax. It involves a whole new way of thinking. Many students who come into an object-oriented programming class with prior procedural knowledge are very confused in the beginning. It is very difficult to switch trains of thought. Starting up from the beginning with object-oriented ideas and designs allows students of all caliber's to become acquainted with the system and more importantly, to think within the bounds of the system.

However this does and has led to some problems. First off, new students and experienced students do not start off on the same level. New students have to pick up object-oriented ideas as well as syntax. Even though the syntax is minimal in the beginning the learning curve is still tough enough that it proves to still be a problem. Experienced users, in contrast, only battle with the object-oriented paradigm. Once they pick that up it is smooth sailing for them.

One of the main arguments for object-oriented programming is for code reuse. Once objects are written you should just be able to reuse them over and over again. Aside from the support code we give students, no code reuse is done in CS15. Nothing that is written early on in the semester is ever used again. Even though students are taught that this is an important aspect of object oriented programming they never experience it themselves.

Another failure I see in our use of Java is that we have totally forsaken the world-wide-web. It has often been remarked how it is funny that our webpage contains no Java applets. While there is indeed a real world reason for this (our graphics package, GP, uses Java 1.1 which, at this time, is not supported) I believe that it is due to a more fundamental reason. That is, no one really knows how to use the web for educational purposes.

The Future Of Teaching With Java

CS15 has everywhere to go. With new Java structures becoming available (Java3D, beans) there are many new areas for us to explore. First I will discuss the last two problems mentioned in the above section, then I will discuss further ideas.

Code Reuse

As mentioned code reuse is something in CS15 that students do not actively take place in. Each project stands in and of itself (aside from our aforementioned support code) and has nothing to do with any other projects the students have written. This is detrimental because students never experience why they should be writing code “as extensible” as possible. Constantly on TA hours our TAs must try to motivate extensibility abstractly, always saying that in some far away future it will be important.

With other concepts we teach, such as inheritance and polymorphism, students get first hand knowledge of them. Students classes extend each other, and polymorphism is always used. the idea of extensibility gets no exposure. Because of this students only get a second-hand education of what extensibility really means.

To amend this problem I propose a new program track that allows for the greatest amount of extensibility. Early programs which are simple could reappear in later assignments as more complex and interesting problems. Properly design and implemented code from the earlier assignments could effortlessly be reused in these assignments. If this earlier code was not design and implemented correctly students would have a first hand chance at seeing how extensibility is an important concept in object-oriented programming and would be able to actively correct the problem.

For students who absolutely failed the original assignment TA provided classes could be created and given to the student provided the student went over their design and implementation with a TA to insure proper understanding.

Something like this has been done casually (without intent I presume) in classes like CS04 and CS16. Students were given the option to reuse data-structures already created in earlier assignments when they were called for in further, more advanced assignments. Having experienced this in CS16 and having taught this in CS04 I feel that it is a very good way of teaching extensibility and is extremely fruitful to the student.

Java and The Web

Utilizing the world-wide-web for educational purposes has not been looked at when it comes to CS15. This seems almost unbelievable seeing as Java's number one asset is its cross-platform capabilities. However it is somewhat understandable as we have only been teaching Java for two years and have just now gotten our two feet on the ground.

The first issue involved is that, as of now, applets created using our graphics package (and further more Java 1.1) does not run inside the Netscape browser. Hopefully browsers such as Netscape will soon support Java 1.1 and this stumbling block will be overcome. Other browsers, such as JWS's HotJava may be able to be incorporated into our environment (which it will as will be shown later).

Aside from this problem no one really knows what could be done. It was suggested, however, in a GISP (group independent study project) last semester, that diagnostics could be provided on-line for instant feedback on assignments. Currently all but one homework in CS15 could be formatted to an on-line version that could provide instant access to students. This would both be an advantage for the students and TAs. An on-line database could be maintained that would keep statistics on problems so that students could see how they were doing compared to the rest of the class. I believe that an addition like this to CS15 would be a very plausible and very helpful.

Teaching Design With No Syntax

It was also brought up in this GISP that design could be taught with *no* syntax at all. That some sort of interpreter with a specific, design-oriented, language could be used to better teach design. This would level the playing field were beginners and experienced programmer's are concerned as well as allow us to focus strictly on design and not have to worry about syntax. Then, once design was learned, the students could jump into learning the syntax and applying it to what they had already learned.

Some objections were immediately raised. Many people felt that one could not learn how to program at all without syntax from day one. It would be impossible to make the jump from design oriented thinking to syntax oriented thinking. It was also argued that once design was learned all one would need to do is learn syntax as another way of expressing the ideas behind design.

I think what can be gleaned from this idea is that it may indeed be helpful to some students to be able to learn design in this way. So rather than force all students to go through this regiment we could provide this tool as a supportive measure for students who wanted extra help or practice. It would also, theoretically, be possible to incorporate a tool like this with web, thus allowing students to work from home, at any time, or perhaps with each other.

Teaching design is something that we feel is very important when it comes to teaching object-oriented programming. Finding other methods of teaching it should be made a top priority.

A New Design-Oriented Environment

Seeing as design is very important to how we teach object-oriented programming, should it not also be an integral part of the students environment, and how they create programs?

The answer, we feel, is a definitive yes. We feel that project creation should be like a shopping trip [Boer97]. That is, students would be able to browse all of the support libraries they have access to, and be able to pick and choose which classes they would like to use in their project. They may choose classes which they will directly use, inherit from, or just classes that provide helpful documentation.

In the current invocation of Wave students are given classes pre-selected by TAs what are deemed helpful. Students are taken entirely out of the project construction phase. Such simple tasks as having to build your project from class libraries can lead to more advanced means of thinking about design.

The shopping model provides us with a great way of seeing design. Normally we go to a grocery store and select ingredients from everything that is available to assemble our breakfasts, lunches and dinners. Design and programming can be seen in the same way [Boer97]. Having a knowledge base of what is available to us, and what is not, we start to envision what is necessary for our program, what we need to get and what we need to create.

One could imagine being able to select a bunch of classes and being able to export them to some other tool, one being the Wave environment which would organize the selected classes into a helpful page of documentation, or perhaps another tool being an OMT style design tool which would create dialogue boxes and start object relations such as inheritance and implementation (interfaces). One could also imagine automatically starting this process with the results from a tool like FOOD [Reiss96].

The students could then proceed to create her design in the OMT tool and then implement that design in the Wave environment. Such selection and automation allows the student to make design choices, and also prods him on his way to making better choices. As of right now, aside from three mandatory design checks for three programs, students do not have to engage the design process in any official way. They are free to go straight into the coding process. Providing tools that guide them into a formal design setting can help infuse them with good design habits.

It is important to note here that for what we want to accomplish in CS15 the Wave environment is essential. Without its design assisting capabilities there would be no way to enforce a design-first mentality. Programming environments, in general, are important in that they integrate a number of helpful tools for the student/user to use. If programming is more than code generation, which we feel it is, then we need an environment that supports this notion as well.

Documentation and the Wave Environment

As has been thought for many years programming is indeed much more than just the act of generating code. Much more goes into the creating of a program. One is concerned with design: how extensible is the program, is it easily maintainable, usability: how easy is the program to use, can

anyone use it, and also documentation: what do I know about this program, how can I make this program easier to understand.

This last idea is something we have taken to heart. In the current Wave environment users are presented with a static javadoc page of documentation about a class. This is all the knowledge one gets about and class in contention. The user is unable to interact with the documentation. What we conceived of was the ability of being able to interact with the documentation. Perhaps you only wanted to see the public interface for a particular class? Maybe you wanted to create a page of documentation for classes whose comments contained the keyword “Data.”? Aside from flipping through tons and tons of already generated documentation you could not easily do this.

Taking Steve Reiss’s idea of a fragment database [Reiss95] we have created our own database of documentation. Now the user is able to generate a multitude of requests for documentation. From the fragments contained in the database pages and pages of user specified documentation can be generated. Requests like those mentioned above are now a reality.

Another important aspect of documentation is that it really is an integral aspect of a program. Users who get access to, or pay for, libraries also want to know how to use them properly. Using Java’s new.jar file specification we envision packaging these fragments with the code so anyone who has access to the code files also has access to user specified documentation generation.

Documentation is an important part of not only a program, but of program creation. If a user is building a program and has many large libraries at her fingertips she wants to be able to get the specific knowledge she needs, quickly, to be able to create her project. With the Wave environment this is possible.

Final Remarks on the Wave Environment

It has been shown that an environment that not only aids in the construction of a program, but aids in the process of learning is very important. As a result, we a re striving to make the Wave environment as suitable to the needs of CS15, and users in general, as possible.

The idea of documentation as an important aspect of program creation leads us to see what other ways we can use the documentation fragments for program creation. One aspect of software engineering which is rapidly becoming very important is group work.

Groups of people, local and non-local, need to be able to work on a project or projects simultaneously. People in Seattle need to have access to the same documentation as the people in New York do. With the Wave environment we can effectively share this documentation easily. By packaging the documentation with the source code we can be assured that everyone has access to the same kind of documentation.

Besides documentation sharing, we would like people to be able to use the Wave environment to engage in group projects. Currently projects are available to only individual users. That is, if student A is working on Tetris and student B is working on Tetris they are working on their own, localized versions of the program. With the new Wave environment it would be encouraging to be

able to not only be allowed to start up localized projects, but to also be able to open up group projects and to also join existing group projects.

When a user is a member of a group project that user would only be able to work on code that other users are not currently working on. This would follow a standard check-in/check-out transaction scheme. The current version of the code would be left in the repository so anyone who wanted to create a build at anytime could do so. The code to be modified would be localized at check-out time. Only after the code had been edited and checked back in would the new version be available to all the users of the project.

Other project artifacts could be managed this way as well. Design documents, notes, etc. could all be based on the transaction scheme. What I feel would be really useful would to have a tool which would monitor users editing the system. Selecting a user would show you what they had recently changed, what they currently had checked-out and perhaps any project restrictions (some users can only edit certain files, some users can only edit design documents, etc.). This user manager would allow users to monitor the creation process of the entire project as well as the creation of the parts they are working on, thereby creating a more group oriented process.

Work in this nature has been done by a couple of groups, one such being a team at Columbia which reported positive results in their student classes [Kaiser97].

Conclusions

Using Java for object-oriented design and programming instruction is a very fruitful action and presumes to be even more fruitful in the future. Such extended uses of the language such as electronic WWW-based diagnostic tools, interpreters for design-without-syntax learning, and design-oriented environments that include project construction in the programming process all seem to be viable aspects which will reward those who use Java as their language of choice.

The environment used to teach object-oriented design and programming is the heart of how and where the students learn everything about object-oriented software construction. Keeping this in mind, it is important that this environment be fully integrated with the ideals of the object-oriented paradigm. The Wave environment promises to do just that by providing an easy to use “shopping model” for project construction and design.

Aside from what has been mentioned, other aspects of Java, including the soon to be released Java3D, Java Beans (which have yet to be tested in an educational arena) and Java’s easy to implement networking system, also promise to add to Java’s educational attraction. What is done with Java over the next year or so, especially in CS15, will most likely define how object-oriented programming is taught in introductory courses at colleges. Therefore I feel it is very important that all aspects of the language be closely examined and brought into the educational arena as soon as possible.