Journal of Statistical Physics, Vol. 93, Nos. 3/4, 1998

Chaos, Complexity, and Computers: Object-Oriented Programming and Physics Concepts for Undergraduates

S. N. Coppersmith[1]

Received December 1, 1997; final January 5, 1998

This article discusses the issues that have arisen as we have updated a course for undergraduates on computers and physics developed at the University of Chicago by Leo Kadanoff, Michael Vinson, Amy Kolan, and Marcelo Magnasco. This course uses interactive computation to teach concepts in nonlinear dynamics, and, more generally, scientific hypothesis testing. Changing the computer language used from THINK Pascal to the object-oriented programming language Java has led not only to "cosmetic" changes (an increased emphasis on animations), but also to conceptual changes in the course (expansion of the discussion of the concept of universality).

KEY WORDS: Chaos; complexity; computers.

1. INTRODUCTION

It is a great privilege to have the opportunity to contribute to this issue honoring Leo Kadanoff's sixtieth birthday. Although my scientific research has been influenced greatly by Leo's work, I have chosen to focus here on another topic of great concern to Leo, teaching physics. This article discusses the issues that have arisen over the past year as a group of us have revised and updated an undergraduate course that Leo and his collaborators developed between 1988 and 1991. Incidentally, one important motivating factor for choosing Java as the computer language for the new version of the course is another issue of great concern to Leo—the improvement of employment opportunities for physics Ph.D.'s.

[1] James Franck Institute, University of Chicago, Chicago, Illinois 60637; snc@control.uchicago.edu.

0022-4715/98/1100-1009$15.00/0

In this paper I will first discuss the overall goals and strategy of the course that Leo and his collaborators developed, which are also those of the present version. I will then discuss how changes and improvements in computational capabilities made substantial revisions necessary, and how the decision to use an object-oriented programming language had unexpected consequences for the choice of physics subject matter, in particular, increased emphasis on the concepts of universality and renormalization.

2. GOALS OF THE COURSE

The course developed by Leo Kadanoff and his collaborators[2, 3] aims to teach juniors and seniors in the physical sciences: (1) how chaotic behavior arises in simple deterministic dynamical systems, and (2) how to use computers, and in particular graphical methods, for hypothesis testing. For example, students were taught how to formulate and answer effectively questions like: *Given a particle trapped in a central force potential that goes like* $1/r^4$, *is a typical orbit closed*?,[2] and *What can we say about the frequency dependence of the response of a forced pendulum*?[3] No previous programming experience was assumed, and yet students were able to develop the skills they needed to answer these questions. Thus the course enabled students to develop broadly applicable quantitative skills, and they were then able to attack effectively problems in fields far removed from the scientific subjects covered in the course syllabus.

The choice to use THINK Pascal and to teach programming as well as physics (rather than a package such as Matlab or Mathematica) followed naturally from the goal of helping students to develop skills that are most broadly applicable. Pascal is a computer language designed for pedagogical purposes, and THINK Pascal has sophisticated and easy-to-use debugging capabilities. These advantages enabled the students to learn to program as painlessly as possible.

Although several excellent courses exist that cover concepts in nonlinear dynamics,[4] the course that Leo and his collaborators developed integrated in a unique fashion concepts of both computation and nonlinear dynamics.

3. GOALS OF THE REVISIONS

Although various versions of the course taught during the period 1989–1993 were extremely successful, by 1997 it was clear that the course

[2] No.

[3] It is very complex and quite likely to be chaotic.

materials needed to be updated. The most pressing issue was the computer language. Updating the language led to unanticipated revisions of physics content in the course.

3.1. Computer Programming Language[5]

Several factors contributed to the decision to change the computer language, the most immediate being that THINK Pascal was no longer being supported in the computer facility to be used for the course laboratory. We again decided not to use a mathematics package but rather to teach the students a programming language. An effective graphics capability was clearly necessary. Our choice needed to be compatible with the available university resources, which had changed considerably since 1991 and continue to evolve rapidly. Although the majority of computers owned by the University are made by Apple (including those to be used by the students in the course laboratory), new university purchases are now overwhelmingly PC's and workstations, and most students own PC's. Therefore, we wanted to revise the course so that it be compatible with but not dependent on Apple hardware. We also wanted to make the course as attractive as possible to students, and wanted to exploit of the popularity of the world wide web.

3.1.1. Advantages of Java. Our two biggest reasons for choosing Java as the computer language are (1) its relative platform-independence, and (2) its sophisticated threading capabilities that are very useful for animations. The platform-independence is not perfect,[6] but the programs used in the course are fairly simple, and in our experience have tended to run on many different platforms with few if any modifications.[7] The threading capabilities are a major advantage, and we have found it straightforward to write simple, good-looking animations of nonlinear systems (for an example, see http://arnold.uchicago.edu/~snc/Physics_251/DWAnimation.html). Animations are considerably simpler to write in Java than in C++.

Our third reason to use Java is that knowing an object-oriented programming language such as C++ or Java is a significantly more marketable skill than knowing Fortran, Basic, or Pascal. This fact is relevant not only for the undergraduates taking the course, but also for our graduate teaching assistants. Evidence that the graduate students find learning Java attractive is that five of them (three of whom have available other means of support, either fellowship or RA) contacted me six months in advance to volunteer for the course's two teaching assistant positions.

The absolutely clearcut advantage of increased employability for students who know an object-oriented programming language can be distinguished from the question of the relative merits of object-oriented programming for scientific computation. There are two ways to do the calculations that scientists need to do, one of which is substantially more marketable than the other. If the more marketable method is not substantially worse for our calculations, we do not need to be convinced that it is better in order to encourage our students to learn it. The employment factor aside, my views of the relative advantages and disadvantages of object-oriented and procedural programming are outlined below.

3.1.2. Disadvantages of Java. Although on the whole our experience with Java has been positive, there are substantial disadvantages to its use.

Speed: The most obvious disadvantage of Java is that it is significantly slower than most computer languages because it is interpreted when it is executed.[8] Just-in-time compilers (such as in the Apple AppletRunner, Internet Explorer 4 or higher, and Netscape 4 or higher) speed up execution substantially, but even with this improvement, execution is substantially slower for Java programs than for programs written in other computer languages such as C or Pascal. We were quite concerned about this, but we found that improvements in computer hardware since the original programs for the course were written more than compensated for this slowdown, so execution speed was never a major factor for us. Animations, which in the old version looked marginal even when written using sophisticated, platform-dependent, bit-mapping and masking techniques, now were straightforward to program successfully.[9] In short, we found the speed limitations of Java were not a serious limitation for this course.

Textbook: A serious drawback to using Java is that it is new enough that there are as yet very few textbooks about it, and those that do exist often do not emphasize the language features that are needed for a nonlinear dynamics course. We decided to require two Java books, one at a beginning level, and a second, more advanced text for explaining animation techniques. Neither book is a textbook, so we needed to include a significant number of programming exercises and problems in the course laboratory manual. New textbooks are being published at a rapid rate, so hopefully this situation will improve quickly.

Language Stability: The newness of Java gives rise to other disadvantages in using it. One is that new versions of the language are coming out fairly rapidly. Another is that one is more likely to run into bugs. We have found that platform-dependence difficulties arise largely because of the platform-dependent nature of the bugs. Java for the Macintosh is in an

earlier version than for PC's and workstations (1.0.2 versus 1.1.4), so it is even more likely to have bugs. We expect these problems to ameliorate as Java matures.

Getting started has complications: For us, the most serious disadvantage of Java turned out to be that it is relatively quite complicated to write even the simplest programs. To create any Java applet, one must write a Java program following a strict naming convention, compile the program into a class file, write an HTML file which calls the applet, and then finally run the applet from a web browser, an appletviewer, or an integrated development environment. Directions on how to write a simple applet that displays the message Hello world! can be found at[10] http://java.sun.com/docs/books/tutorial/getStarted/applet/index.html. The program itself contains the lines:

```
import java.applet.Applet;
import java.awt.Graphics;
public class HelloWorld extends Applet {
    public void paint(Graphics g) {
      g.drawString("Hello world!," 50, 25);
    }
}
```

For readers who know no Java, it is probably obvious that this program is not completely trivial. For those who do know Java, note that this program imports the two class libraries java.applet.Applet and java.awt.Graphics, uses inheritance in the declaration of the applet HelloWorld, and uses a Graphics object in the paint method. Writing a very basic Java application (for instructions, see http://java.sun.com/docs/books/tutorial/getStarted/application/index.html) is slightly simpler, but then one must open a frame in order to do simple graphics. We have opted to provide the students with working applets at the beginning of the course, and then gradually explain the underlying concepts, with the hope that they can get started quickly and fill in the gaps in their understanding of the programming language later.

Writing simple programs in Java can be more confusing than in procedural languages until one gets the hang of objects. We discuss this, as well as other advantages and disadvantages of object-oriented programming, in the next section.

4. OBJECT-ORIENTED PROGRAMMING AND PHYSICS CONCEPTS

Books about object-oriented programming tend to have a lot of discussion of software reusability and modifiability, data hiding, inheritance,

and polymorphism.[11] These discussions can be quite opaque to the uninitiated, and yet the concepts in question are quite similar to those that physicists use frequently.

The point of defining objects is to keep the description of a problem as abstract as possible for as long as possible. This concept is completely analogous to things that physicists do all the time. For example, freshman physics students are instructed to do problems in terms of variables and only plug in numbers at the end. For instance, consider the following simple physics problem: *A projectile is launched from a flat surface at a* 45° *angle to the horizontal and is observed to reach its maximum height at a time three seconds after launching. How far from the launch point is the projectile when it has fallen halfway from its maximum height*? We teach the students to call the angle θ and not 45°, keeping the problem as general as possible until the very end. We motivate this by telling them that they will save a lot of work if they have to redo the problem with a different initial angle. The analogous procedure in object-oriented programming is to define an object (say, a launcher object) which has a method (say, getInitialAngle()) which one calls to determine the initial angle. If the process by which the initial angle is determined changes, then only the launcher object need be modified; the rest of the program is completely unaffected.

A second illustration of the role of abstraction is particularly relevant to a course in dynamical systems. Recall that a dynamical system is a rule that determines the configuration at a later time given the configuration at some earlier time. Newton's laws work this way: given the velocity and position of a particle at one instant of time, these laws enable the prediction of the velocity and position at future times. However, one can imagine many different types of rules, encompassing different numbers of variables, including continuous differential equations as well as discrete-time maps. It is clear that systems described by continuous differential equations comprise a subset of dynamical systems, and that systems which are single particles obeying Newton's laws comprise a subset, and, in turn, the damped pendulum is a yet smaller subset. Object-oriented programming is particularly useful in this situation, for one can define the general class dynamicalSystem and then use inheritance to define subclasses corresponding to the more specialized cases (e.g., diffEqSystem, dampedPendulum). Therefore, when programming in Java it is natural and useful to consider the whole ensemble of dynamical systems together. In physics, considering the whole ensemble of dynamical systems is important because of *universality*, where large classes of different dynamical systems have some properties that are *quantitatively* identical. The similarity and importance of these concepts led us to expand greatly the emphasis on universality in the course, focusing on the universal properties of the period-doubling route to

chaos.[12] Java's object-oriented nature makes it easy and natural for students to examine many different dynamical systems, and thus observe empirically that they all obey scaling laws with the same exponents.

4.1. Teaching Object-Oriented Programming

In our course the programming is used as a means to an end in the sense that the highest priority is to write working programs that students can use to understand dynamical systems rather than to teach excellent object-oriented programming technique. However, to write even fairly simple Java programs one must have some understanding of how objects work. This necessity is pressing, in contrast to the situation with C++, because C++ is a superset of the procedural language C. This aspect of Java means that those already proficient in a procedural programming language go through a frustrating period (which for the author lasted about a month) when programs that were straightforward to write in the "old" language now seem difficult.[14] The advantage of this feature is that one more quickly understands the differences between procedural and object-oriented programming.

4.2. Object-Oriented Programming and Scientific Computation

The section discusses a few aspects of the usefulness of object-oriented programming languages for scientific computation.

Despite Java's advantages compared to C++ for use in an undergraduate nonlinear dynamics course (simpler graphics and animations, plus the language itself is less confusing), currently Java is not a good option for scientific computation.[5] Disadvantages include: (1) execution is slower, a factor which is discussed above, (2) input and output facilities tend to be more rudimentary in Java than in other computer languages,[13] and (3) the mathematical libraries available for Java are much less extensive than those available for Fortran and C (and thus also C++). Therefore, currently the most practical object-oriented programming language for scientific programming is C++.

Anecdotal evidence seems to indicate that C++ programs run somewhat (roughly 20%) slower than Fortran programs. The author has never done a computation where such a difference in speed was important, and therefore does not consider this factor significant. In addition, at least one expert claims that this factor arises only because the flexibility of C and C++ makes optimization of a program more complex, and that a fully

optimized C or C++ program typically runs as fast or faster than the same program written in Fortran.[15] Memory management in C and C++ tends to be much more efficient than in Fortran, and often memory rather than time is the limiting factor in a calculation.[15] During the "learning curve," the time needed to write simple programs is longer with an object-oriented language, but reasonably bright individuals get over this quickly. Therefore, there is no compelling reason to avoid object-oriented programming.

For simple computations the advantages of object-oriented programming are minimal (just as the advantages of using θ instead of 45° in the projectile problem described above are not so great), but for massive computer programs that will undergo revisions by many people over a long period, the author believes that object-oriented programming encourages useful abstractions that lead to significant advantages. This usefulness for large computations, together with the employability factor, are significant reasons for encouraging the use of object-oriented programming in the physical sciences.

5. DISCUSSION

This article has discussed issues that have arisen as we have updated a course on computers and physics developed by Leo Kadanoff and his collaborators. A major focus is the consequences of the decision to change the computer programming language to Java. This decision led not only to increased use of animations in the course for basically cosmetic reasons, but also the increased emphasis on the concept of universality in dynamical systems in the subject matter. This latter change was motivated by the importance of abstraction, classification, and hierarchy in object-oriented programming languages, and the analogy to important physics concepts.

Unfortunately, the deadline for submission of this article came before the course was taught, so the article could not include any information about the reaction of the students taking the course. Those interested in developments occurring after this writing are invited to contact the author (snc@control.uchicago.edu).[16]

ACKNOWLEDGMENTS

I thank the authors of ref. 3 for allowing me, along with the other authors of ref. 1, free access to the material that they prepared and for the benefit of their insight. I acknowledge a very useful conversation with Steve White, and hospitality and support from the Institute for Theoretical

Physics at Santa Barbara (NSF Grant Now PHY94-07194), where this article was prepared in part.

REFERENCES

1. Benjamin Blander, Susan Coppersmith, Leo P. Kadanoff, Michael J. Vinson, Amy J. Kolan, Marcelo Magnasco, and Scott Wunsch, Chaos, Complexity, and Computers, unpublished class notes (1998).
2. Leo P. Kadanoff, Interactive computation for undergraduates, *Physics Today*, December 1988, page 9.
3. Leo P. Kadanoff, Michael J. Vinson, Amy J. Kolan, and Marcelo Magnasco, Chaos, Computers, and Physics, unpublished class notes (1991).
4. Other texts integrating programming and physics include Nicholas J. Giordano, *Computational Physics* (Prentice-Hall, Englewood Cliffs, NJ, 1997); Harvey Gould and Jan Tobochnik, *An Introduction to Computer Simulation Methods*, 2nd ed. (Addison Wesley, Reading, MA, 1996); G. L. Baker and J. P. Gollub, *Chaotic Dynamics: An Introduction*, 2nd ed. (Cambridge University Press, 1996). An extensive and very useful list of resources available for teaching courses in nonlinear dynamics is Robert C. Hilborn and Nicholas B. Tufillaro, *Am. J. Phys.* **65**:822 (1997). In addition, Mike Cross has some interesting and sophisticated Java applets illustrating concepts of nonlinear dynamics on his web page at http://www.cmp.caltech.edu/~mcc/chaos_new/Chaos_demos.html.
5. For a useful discussion of the advantages and disadvantages of Java for scientific applications, see Paul Dubois, Is Java for Scientific Programming, *Comput. Phys.* **11**(6):611 (1997).
6. For instance, see Mike Cross' comments at: http://www.cmp.caltech.edu/~mcc/chaos_new/experience.html. We have addressed the issue of putting technical text on the web by using Adobe Acrobat, and the 1.1 version of Java is supposed to address the component layout issues raised by Cross. However, as of this writing (November 1997), Java 1.1 was not yet available for Macintosh computers.
7. One problem of this type is that the resize() command does not work at all when running applets using Netscape 3.0. One can work around this problem by setting the applet size in the html file.
8. Java programs are compiled into bytecodes, which in turn are executed by a Java runtime interpreter. For a fuller discussion of this point, see Patrick Niemeyer & Joshua Peck, *Exploring Java* (O'Reilly, Sebastopol, CA, 1996), p. 4.
9. The most important improvement leading to more attractive animations is increased computer memory sizes, which allow for double-buffering (see, e.g., http://java.sun.com/docs/books/tutorial/ui/drawing/doubleBuffer.html).
10. Our course uses the integrated development environment (IDE) Codewarrior, largely because it is used in many of the other classes taught in the Computer Science department here. Using an IDE makes getting started somewhat simpler, but comes far from eliminating all the problems.
11. See, e.g., Grady Booch, *Object-oriented Analysis and Design with Applications*, 2nd ed. (Benjamin-Cummings, Redwood City, CA, 1994).
12. M. J. Feigenbaum, The universal metric properties of nonlinear transformations, *J. Stat. Phys.* **21**:669–706 (1979).
13. The inconvenient input and output facilities in Java can be overcome by writing class libraries. Some useful utilities of this type can be found in Gary Cornell and Cary Horstmann, *Core Java* (Prentice-Hall, Upper Saddle River, NJ, 1996).

14. For example, in Java a method can only return zero or one objects, in contrast to a Fortran subroutine where large numbers of both inputs and outputs can be handled simultaneously. In Java one must bundle the outputs into an object in order to have them all returned by a single method.
15. S. R. White, private communication.
16. Readers interested in examining the course lab manual can look at: http://arnold uchicago.edu/~snc/Physics_251/Lab_Manual.