Pedagogies for Teaching CS1 with Java

Byron Weber Becker
Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada N2L 3G1
bwbecker@uwaterloo.ca

This is a work in progress.

Abstract

The introduction of Java to CS1 has resulted in a proliferation of pedagogical approaches. As a more purely object-oriented language (no methods outside of a class construct), Java has forced educators to confront pedagogical issues that the move to C++ did not.

In this paper I investigate the range of pedagogical options used to teach three crucial parts of almost any CS1 course taught with Java: the introduction of classes and objects, the introduction of I/O (including graphical user interfaces), and software engineering issues. Using textbooks as pedagogical case studies, I develop a classification system for the introduction of classes and objects, as well as I/O. The work on software engineering issues is still preliminary and merely points towards possible future work.

1 Introduction

At last count thirty-four Java textbooks aimed at the CS1 market sit on the author's shelf. A casual glance through them shows that there are many, many differences in how they approach teaching programming via the Java language.

In [18] five general approaches to I/O are proposed and in [17] and elsewhere advice is given regarding the ordering of topics. Are these approaches to I/O and is this ordering advice actually followed in all those textbooks?

A survey of about half of the thirty-four books reveals that the five general approaches to I/O need significant revision and that while most follow the advice in [17] to use objects before writing classes, there is still considerable variation. New classifications for both areas are proposed.

In the course of the survey, observations about the treatment of software engineering topics were made. While this part of the study is too preliminary to form any categorizations, it may be a pointer for some future study. It was also inescapable that some personal opinions would form in the course of closely examining so many books. These are reported in the conclusions.

Choosing which textbooks to include in the study was somewhat arbitrary. Textbooks included met one or more of the following criteria: 1) those that made the best impression when they first came, 2) were relatively recent, 3) written by authors recognized for other books, and 4) books thought to be widely used.

The sixteen textbooks chosen are listed in the Textbook References section at the end of the paper. A separate section lists other references. Also, notes developed in the course of researching this paper are available at www.math.uwaterloo.ca/~bwbecker/papers/ for those who are interested.

The author deeply respects the vast time and energy invested by the authors of these textbooks and does not want to slight them in any way. Nevertheless, some textbooks were found to be more helpful than others and those *opinions* are noted.

2 Pedagogies for Objects

The richness and variation in textbooks is particularly evident in how authors introduce objects and classes. At first glance, there is very little commonality between pedagogies used by the textbooks. However, by looking at a core set of decisions that must be made by each author, either explicitly or implicitly, patterns emerge. Those decisions include 1) the ordering of objects with respect to other topics, particularly control structures, 2) the order of using objects and writing their classes, and 3) the qualities of the first objects and classes presented.

2.1 Ordering of Topics

The order of topics (e.g.: "objects first" vs. "control structures first" vs. something else first) is a fundamental part of a textbook's pedagogy. To get an approximation of the orderings, each topic covered (in broad categories) was listed in the order presented. The list stopped when classes had been implemented and control structures had been covered. These orderings are given in the second column of Table 1 (which also includes data discussed in section **Error! Reference source not found.**).

2.1.1 Keywords and Definitions

The following keywords and definitions were used to capture the ordering of topics.

Applets: Writing applets. This is not included as object usage or implementing classes (see below) because classes and objects have not yet been discussed and the programmer has not explicitly instantiated the class.

Control: Iteration and selection control structures.

Design: Help in designing classes and their interactions.

Author(s)	Topic Ordering	First Class Used/First Class Written
Arnow & Weiss	UseInstantObj, UseObj, ImplClass, Design, Control	(PrintStream, String), File Laugher – prints "haha" and variations.
Barnes	UseObj, ImplClass, Control	Ship – navigate an ocean-going ship. SimpleNote – a Post-it® note analogy
Bell & Parr	Applets, Methods, Control, UseImpl	(Applets, Graphics), Balloon –a balloon with location and size. Balloon
Bishop	Sample, UseObj, Methods, ImplClass, Control	Date Labels – printing the outline of a box.
Deitel & Deitel	Sample, Applets, Control, Methods, Arrays, UseImpl	(Applet, Graphics), Time[1] Time
Horstmann	Sample, UseImpl, Control	BankAccount – deposit, withdraw or transfer money. BankAccount
Koffman & Wolz	UseImpl, Control	UnitConverter – convert square meters to square yards UnitConverter
Lambert & Osborne	Sample, UseObj, Control, Methods, ImplClass	GUI objects such as buttons Student – with a name, test scores, and average calculations.
Lewis & Loftus	UseInstantObj, UseObj, Control, ImplClass	(PrintStream, String), String, Random, DecimalFormat Coin – simulates flipping a coin.
Morelli	Sample, UseImpl, Control	Rectangle, CyberPet – a pet which "eats" and "sleeps". CyberPet
Nino & Hosch[2]	DiscussObj, SpecifyClass, ImplClass, Control, UseObj	Counter, PlayingCard, Explorer – several examples in parallel Nim
Savitch	Sample, UseInstantObj, Control, UseImpl	(String), Species – estimate population growth in a species Species
Slack	UseObj[3], Control, UseObj, ImplClass	TurtleGraphics, Date, DateFormat SmartTurtle, Counter
Stein[4]	DiscussObj, ImplClassExt, Control, SpecifyClass, ImplClass	(String) StringTransformer – transform strings, Rectangle
Wu	DiscussObj, UseObj, ImplClass,	MainWindow, MessageBox – provided interface package.

[1] Deitel & Dietel do slip in an instantiation of `TextArea` prior to this, but it is really just to get more space for output and is not discussed as an object.

[2] Nino & Hosch is interesting in that they specify and actually implement classes long before they are used – at least in the textbook. The publishers have assured me that students write programs and use objects in lab much earlier than is suggested by reading the textbook alone.

[3] This is a much more thorough use of *UseObj* than is typical. It's really an overview of the entire language, including inheritance.

[4] Stein is similar to Nino & Hosch: both specify and implement classes before putting much emphasis on using objects. In this textbook I don't see a complete program. The sample assignments on the web site do.

	Control	CurrencyConverter – convert one currency to another.

Table 1: Order of Topics and Objects Used/Classes Written

DiscussObj: An in-depth discussion of objects and classes, but without code.

ImplClass: Implementation of classes, including instance variables and methods (unless methods are previously covered as a separate topic).

ImplClassExt: Implementation of classes by extending existing classes.

Methods: Writing methods, but prior to implementing classes.

Sample: A sample program is used as an overview. The sample must be more complex than *HelloWorld* to be noted. (While [17] rightly claims that *HelloWorld* is completely object-oriented and can be a wonderful example, the fact that object instantiation is hidden by using only `System.out` and `String` objects weakens the example substantially.)

SpecifyClass: Developing the specifications of a class, without actually implementing it.

UseObj: Using objects in programs which are explicitly instantiated by the programmer (excludes using class methods and objects covered by *UseInstantObj*).

UseInstantObj: Using objects that were not explicitly instantiated by the programmer. The most common examples are the `PrintStream` object `System.out` (instantiated by the runtime system), `String` objects (instantiated with special language support) and `Graphics` contexts (instantiated by the runtime system and passed to `paint`). If these objects are simply used to get output without a real discussion of objects, then it is classified as a *Sample* or ignored.

UseImpl: Using and implementing objects are presented together.

These keywords are admittedly somewhat loose and there is lots of room for interpretation – both in what the keywords mean and in which keyword best fits a particular section of a textbook. This reflects the rich diversity in how authors approach the topics.

There were several difficulties in constructing this table. First, a text by Decker & Hirshfield initially included in the study is not listed above because it was impossible to classify under this scheme. The text begins by extending `applet` and continues to emphasize applets for the remainder of the text. I could not find a complete class anywhere in the book which did not extend a class in `java.awt` and nowhere were classes and objects thoroughly explained. The text is excluded from the remainder of this section.

Second, a spiral approach used by Bishop made it very difficult to determine the order of topics or even what to count as the first object usage. Topics were introduced but not covered thoroughly only to be revisited again but with different examples.

2.1.2 OO vs. Control Ordering

On the surface, there is very little commonality. Only Horstmann and Morelli show the same set of keywords in the same order.

Deeper analyses are possible, however. One of the most helpful categorizations is locating control structures with respect to objects and classes as represented by the set of keywords $obj = \{ImplClass, UseObj, UseImpl\}$. The keyword *UseInstantObj* is excluded from this set because the examples it represents don't give a complete picture of object-oriented programming. Most of the texts are covered by three possibilities where $o \in Obj$:

1. *Control* precedes *o* (control first). Textbooks in this category include Bell & Parr, Deitel & Deitel, and Savitch.
2. *UseObj* precedes *Control* with *ImplClass* following (control middle). Lambert & Osborne, Lewis & Loftus, and Slack follow this approach.
3. *o* precedes *Control* (control last). This is the largest category, containing all of the remaining textbooks except, perhaps, Stein[5].

The placement of objects with respect to control structures results in very different learning experiences. At the moment it is difficult to say which of these pedagogies, if any, is more correct. However, Joseph Bergin [Bergin] argues that the most important topics should be placed first, if possible. In an object-oriented language a very reasonable position is that using objects and writing classes are among the most important concepts.

Moving objects and classes early echoes the migration of procedures from the end of textbooks to the beginning of textbooks in the Pascal era as documented in [Pattis]. Excerpts of the prefaces of these Pascal textbooks indicate that "procedures early" was driven by a concern with program structure. In an OO language, objects and classes take the place of subprograms as the dominant structuring paradigm.

The control middle and control last pedagogies present an opportunity not available in control first: using objects to illustrate control structures.

[5] Of the three categories, Stein fits most nearly with control last. She has an early section where classes are implemented using extension of a provided class. This, plus an extensive discussion of objects (without code) gives a firm enough foundation to use some objects during discussions of control structures later in the book.

There are several possibilities here. One is placing the control structures in the context of a class and making use of the classes' instance variables. Another is to have the control structures make active use of one or more objects, both in the bodies of the statement and in the condition. By these measures, Arnow & Weiss, Barnes, Horstmann, Morelli, and Wu are strong. The other textbooks implementing the control middle or control late pedagogies use lots of examples with classes containing only `main()` or fragments disembodied from their class. It seems that opportunities are missed.

2.1.3 Procedural vs. O-O Paradigm

A variation of the control early and control middle pedagogies is exposed by these textbook case studies: are methods first presented as a construct to control execution (a procedural paradigm) or are methods first presented as an object's behavior (an object-oriented paradigm)? Four of the textbooks (Bell & Parr, Bishop, Dietel & Dietel, and Lambert & Osborne) introduce methods using the procedural paradigm, either as helper methods within an applet or static methods called from `main()`.

2.2 Object Usage vs. Class Authorship

Another pedagogical decision is whether students write programs using objects before they are asked to author their own classes. In Table 1 this is indicated by *UseObj* occurring before one of {*ImplClass*, *UseImpl*}.

Comparing to the Pascal era, this is analogous to writing programs which use existing procedures before writing your own procedures, a pedagogy named "Read/Call Before Write" or "Libraries Early" by Pattis [Pattis]. Pattis advances the following arguments for this pedagogy:

- The experience of using subprograms helps students understand what they are why they should be used.
- Students learn to read and use libraries, a fundamental skill.
- Students become "documentation consumers, not producers" which is more apt to teach the need for quality documentation.
- Students can design and implement more sophisticated (satisfying) programs than they could on their own.

These same arguments can be easily modified into a "Read/Use Before Write" object-oriented pedagogy. Wu does this in his preface, adding the arguments that using a library "shows students how real-world programs are developed" and that it "minimizes the impact of programming language syntax and semantics" (such as I/O). Of the 16 textbooks surveyed, 7 use this approach (Arnow & Weiss, Barnes, Bishop, Lambert & Osborne, Lewis & Loftus, Slack and Wu).

Several of the authors use classes from the standard Java libraries. These include classes such as `PrintStream` `File`, `Date`, `String` (explicitly instantiated with `new`), `Random`, and `DecimalFormat`. Other authors used libraries they provided. These included `Ship` and `Turtle` objects (both reminiscent of turtle graphics and Karel the Robot) and two GUI libraries, one based on an event model and the other simply using dialog boxes instead of reads and writes.

2.3 Object Instantiation

Already in the definition of the keywords used in Table 1 you will have noticed the distinction between using objects which the students instantiate themselves and those which are instantiated for them. Examples of the later are `System.out`, strings, and the `Graphics` context passed to the `paint` method. In spite of the spirited defense of "HelloWorld" as an object-oriented program in [17], programs using pre-instantiated objects really do have a different feel in that either there is only one instance or they do not have a readily discernable state.

Examples which contain only single instances of a class are problematic because students typically stumble on the idea that there can be many instances, each with its own state but a common set of behaviors. This isn't evident in examples relying on `System.out` and the `Graphics` context where students only see one instance in any one program. In my experience students grasped this concept more easily and quickly if the first example program has multiple instances of the same class, each with its own state. Examples? A bank account class with an instance for my account and another for your account. Arnow & Weiss, Horstmann, Lewis & Loftus, and Morelli all have strong examples.

Many early examples use strings as their objects – and use several instances. One problem is that Java provides special support for creating string objects from literals, disrupting the typical object lifecycle. Another problem is that strings are immutable. It's hard to show that different objects have different state because you can't change the state of a string – all you can do is create a new one.

2.4 Interacting Objects

Still another aspect of object pedagogies was exposed in Arnow & Weiss. They have a tollbooth program which has a class containing `main()`, a second class modeling trucks and a third class modeling tollbooths. The `main()` instantiates a tollbooth and several trucks. It then arranges for each truck to "go through" the tollbooth for tolls to be collected.

This example struck me as somewhat unusual in textbooks but not in real life. What are the characteristics which caught my eye?

- Objects from two classes are interacting to accomplish a task. This is different from `main()` interacting with objects from just one class.
- The design process of both classes is shown, including how responsibilities are distributed between them.

Distributing responsibilities between classes is an important skill that we dare not overlook. Yet a number of the textbooks did. [1,2,11,12,14,15] contain examples similar

to the tollbooth example. [4,7,8,9,13,16] have interacting classes in the chapter on arrays where one class contains an array (with a public interface to manipulate it) of objects. A number of texts have no obvious instance of classes that interact in this manner [3,5,6,10].

2.5 Summary of Object Pedagogies

Authors	Control
Arnow & Weiss	Last
Bell & Parr	Middle
Barnes	Last
Bishop	Last
Deitel & Deitel	First
Horstmann	Last
Koffman & Wolz	Last
Lambert & Osborne	Middle
Lewis & Loftus	Middle
Morelli	Last
Nino & Hosch	Last
Savitch	First
Slack	Middle
Stein	Last
Wu	Last

3 Pedagogies for IO

3.1 Approach to I/O

In [18] a five-part classification for handling input and output concerns in a textbook is presented. Briefly, the five possibilities are (quoting)

1. Ignore GUIs entirely. Require students to write programs with input/output constructs from libraries such as C++ ioStream, C stdio, and Java classes system.in and out.
2. Promote non-sequential programming models from the onset by embracing GUI and concurrent programming; for example introduce the Java Abstract Windows Toolkit (AWT) and the Java Thread class immediately.
3. Have students write program fragments rather than complete programs.
4. Include instruction in the use of GUI-design tools such as Visual Basic.
5. Use special purpose libraries.

Reviewing the sixteen textbooks reveals remarkably little overlap in approach and that none of them use options 1, 3 or 4. See **Error! Reference source not found.** for a summary of key differences. Clearly a different classification is needed.

In examining the differences and similarities it becomes evident that the key question is how soon event-driven interfaces are introduced. Two secondary concerns are ordering of material and whether simplified libraries will be used. One way of organizing these concerns is as follows:

1. Use event-driven interfaces from the beginning of the text. Console I/O is either not covered at all or covered briefly in a chapter with other I/O streams.
   a. Use a simplified GUI library early in the book, introducing `java.awt.*` after programming foundations have been laid. Lambert & Osborne follow this approach.
   b. Use `java.awt.*` from the very beginning of the text. Bell & Parr and Decker & Hirshfield use this approach.
2. Use console-style I/O until students are ready for event-driven interfaces. There are three variations:
   a. Use only `java.io.*` throughout the text. Morelli uses this strategy.
   b. Use a simplified console I/O class. Most authors explain the class in a later chapter covering I/O streams. Barnes, Bishop, Horstmann, Nino & Hosch, Savitch, Slack, and Stein use this approach. Horstmann and Slack explain the provided class soon after it is first used.
   c. A third approach is to use a library providing dialog boxes for input and output with the user. The style of programming is like a console program but the result looks more like a GUI. Koffman & Wolz and Wu use this approach. Deitel & Deitel's variation is using `JOptionPane` for I/O rather than a provided library.
3. The third approach is to present console I/O and GUIs in parallel. These textbooks rely on console-style I/O early in the book but include a "GUI Supplement" at the end of most chapters. The supplement attempts to put the chapter's topic in a graphical or GUI context. It generally takes about 6 chapters to build the groundwork for simple event-driven programs. As with the other approaches, there are two variants:
   a. Use a simplified console I/O class. Lewis & Loftus do this.
   b. Use `java.io.*`; Arnow & Weiss.

There are several other much more minor differences in approach. About half of the authors present graphics early as a fun application of the other concepts being discussed. Event-driven GUIs come later.

Another issue is how to address applets. Five of the sixteen textbooks focus almost entirely on applets rather than graphical applications. Five others focus on applications, providing only a few pages on the basics of applets.

3.2 Graphical I/O Libraries

Among the provided libraries which offer a graphical interface, there are clear differences. Among the three, Lambert & Osborne's `BreezyGUI` library is the only one which simplifies event-driven I/O. Their library has the feel of the Java 1.0 event model in which programmers override specific methods to handle specific kinds of events. If the same kind of event can arrive from more than one source, the programmer must distinguish the cases with a selection statement.

	A & W	Barnes	B & P	Bishop	D & H	D & D	Horstman	K & W	L & O	L & L	Morelli	N & H	Savitch	Slack	Stein	Wu
Is console I/O used?	Y	Y	N	Y	N	N	Y	N	N	Y	Y	Y	Y	Y	Y	N
Is a console class provided?	N	Y		Y			Y			Y	N	Y	Y	Y	Y	
Is the console class explained?		IO		IO			E			IO		IO	IO	E		
When are graphics introduced?	E	L	E	L	E	E	E	E	M	E	M	L	M	L	L	E
When are event-driven GUIs used?	M	L	E	L	E	M[1]	M	M	E[1]	M	E	L	M	L	L	L
Are libraries provided for GUIs?	N	N	N	N	N	N	N	Y	Y	N	N	N	N	N	N[1]	Y
Is the focus on applets?	Y	N	Y	N	Y	N	N	Y	N	Y	N	N	N	N	N	N
Do chapters have GUI supplements?	Y	N	Y	N	Y	N	N	N	N	Y	N	N	N	N	N	N

A feature of `BreezyGUI` is a simplified version of `GridBagLayout` which makes it easy to create reasonably sophisticated-looking interfaces.

Wu's `javabook` library is a collection of instantiable classes which provide input from textboxes, lists, yes/no choices, etc. from users via dialog boxes. Other classes provide options for output. One of the stated goals of the class is to give students practice in using objects, both instantiation and method invocation.

Programmers using Koffman and Wolz's library use the *is-a* relationship rather than the *uses* relationship between classes. Classes which need to do I/O extend `SimpleIO`, a provided class. One perceived advantage is that beginners can simply use the I/O methods without needing to know about objects or dot notation. It becomes a liability later on if students need to extend a different class and do console I/O or when they simply realize that `Employee`, for instance, fails the *is-a* test with respect to `SimpleGUI`.

4 Pedagogies for Software Engineering

In examining the sixteen textbooks, a number of admirable qualities related to software engineering came forward. "Software engineering" is used rather liberally here to mean anything that might guide the software development process towards a better product. This encompasses discussions of the software development process, or case studies which illustrate the decisions made by an expert developer or model a software development process. It also includes discussions of testing, debugging, coupling/cohesion, assertions or designing by contract, class specification, or design tools such as UML.

Nino & Hosch are clearly the best of the sixteen in this regard. A working title for their book hints strongly at this bias: *Introduction to Software Construction with Java*. Areas where their book stands out include:

- An early and careful discussion of abstraction.
- Detailed specification of classes before implementation.
- Two chapters on programming by contract and testing.
- A detailed case study where life cycle issues are discussed and CRC cards used, although the life cycle could be stronger.
- A chapter discussing software quality, both from an external view (such as reliability) and an internal view (such as cohesion and coupling).
- Using the Model-View-Controller pattern thoughout.

Wu stands out for his use of UML-like diagrams. He has a number of good case studies, including a very detailed study as the entire last chapter.

Arnow & Weiss are most noted for presenting and then consistently using a development process in their case studies. There is also a chapter on verifying object behavior.

Lewis & Loftus have a chapter on software engineering which covers a variety of development models including waterfall, iterative and evolutionary. There are also brief descriptions of how to identify classes and objects, and the software life cycle. Unfortunately, the chapter is too late to be reinforced in case studies.

Horstmann also has a chapter late in his book which covers many of these issues. He also includes using nouns and verbs for class/object/method discovery, supplemented by

CRC cards. An earlier chapter covers testing and debugging.

Morelli provides an "Object-Oriented Design" section in almost every chapter. Though short, the tips provided are important and timely.

Stein is notable for her emphasis on specifying behavior through interfaces.

5 Conclusions

Sixteen Java textbooks aimed at CS1 have been examined in terms of how objects and classes are introduced, I/O is handled, and their support for software engineering concepts. For the first two topics, new classification systems which fits the data were developed and presented.

It is obvious, given the many and varied approaches taken, that no one book is likely to meet every need. Nevertheless, the author has formed some opinions from this survey and would like to offer them in hope that they will benefit instructors choosing textbooks and designing courses, as well as future textbook writers.

- Two of the books were exceptionally enjoyable for their fresh approach and the extent to which they challenged the author personally. Nino & Hosch and Stein are highly recommended to instructors, whether or not they are adopted for student use.
- Of those who used GUIs from the very beginning, Lambert & Osborne seemed to be the only one which avoided having fundamental concepts dominated by the AWT. This is, of course, due to their own simplified library (which is available for anyone – even other textbook authors – to use).
- Six of the sixteen textbooks avoid console I/O almost completely. While most of the programs we use may have GUIs, it is not obvious that console I/O is dead. It is too useful for testing and prototyping. Anyone adopting one of these six texts should seriously consider supplementing it with console I/O.
- The most satisfying books were those which used objects and then wrote classes – and did it early.
- Although they each have their problems, the books most readily recommended are Arnow & Weiss, Lewis & Loftus, Horstmann, Morelli, Slack and Wu, in addition to Nino & Hosch and Stein[6].

Textbook References

[1] Arnow, David M. and Gerald Weiss. Introduction to Programming Using Java: An Object-Oriented Approach. Addison-Wesley, 2000.

[2] Barnes, David J.. *Object-Oriented Programming with Java: An Introduction*. Prentice Hall, 2000.

[3] Bell, Douglas and Mike Parr. *Java for Students*. Prentice Hall, 1999.

[4] Bishop, Judy. *Java Gently: Programming Principles Explained*. Addison-Wesley, 1998.

[5] Decker, Rich and Stuart Hirshfield. *Programming.java: An Introduction to Programming Using Java*. Brooks/Cole, 2000.

[6] Deitel, H.M., and P.J. Deitel. *Java: How to Program*. Prentice Hall, 1999.

[7] Horstmann, Cay. *Computing Concepts with Java 2 Essentials*. John Wiley & Sons, 2000.

[8] Koffman, Elliot and Ursula Wolz. *Problem Solving with Java*. Addison Wesley Longman, 1999.

[9] Lambert, Kenneth A. and Martin Osborne. *Java: A Framework for Programming and Problem Solving*. PWS Publishing, 1999.

[10] Lewis, John and William Loftus. *Java Software Solutions*. Addison-Wesley, 2000.

[11] Morelli, Ralph. *Java, Java, Java: Object-Oriented Problem Solving*. Prentice-Hall, 2000.

[12] Nino, Jaime and Frederick Hosch. *Introduction to Programming and Object-Oriented Design Using Java*. John Wiley & Sons, 2001[7].

[13] Savitch, Walter. *Java: An Introduction to Computer Science & Programming*. Prentice Hall, 1999.

[14] Slack, James M.. *Programming and Problem Solving with Java*. Brooks/Cole, 2000.

[15] Stein, Lynn Andrea. *Interactive Programming in Java*. Due to be published by Morgan Kaufmann Publishers in late 2000. Currently available on-line at http://www-cs101.ai.mit.edu/.

[16] Wu, C. Thomas. *An Introduction to Object-Oriented Programming with Java*. WCB/McGraw-Hill, 1999.

Other References

[Bergin] Bergin, Joseph. Pedagogical Pattern #34 Early Bird Pattern at the *Pedagogical Patterns Project*, http://www-lifia.info.unlp.edu.ar/ppp/pp34.htm.

[17] Lewis, John. Myths about Object-Orienteation and its Pedagogy *SIGCSE Bulletin* 1 (2000), p. 245-249.

[Pattis] Pattis, Richard E. "The 'Procedures Early' Approach in CS1: A Heresy" *SIGCSE Bulletin 1* (1993), p. 122 - 126.

[18] Wolz, Ursula and Elliot Koffman. "simpleIO: A Java package for Novice Interactive and Graphics Programming" *SIGCSE Bulletin* 3 (1999), p. 139 - 142.

[6] The author has nothing to gain by anyone adopting any of these books.

[7] This review is based on a draft manuscript dated March, 2000 and checked against the most recent version at http://www.cs.uno.edu/~fred/OOJ/Utilities/.

6 Appendix A: A Survey of Java Textbooks

6.1 Arnow and Weiss

Reference: Arnow, David M. and Gerald Weiss. *Introduction to Programming Using Java: An Object-Oriented Approach*. Addison-Wesley, 2000. 723 + 82 pages.

Objects: Chapter 1 contains a discussion of objects and classes followed by a helloWorld program. Chapter 2 is about using objects, but they are all objects that are created with hidden constructors – `System.out` and strings. Using constructors to create an instance comes in the next chapter. After a brief discussion of String constructors, the entire chapter is spent on file and console I/O.

Classes are first written in chapter 4 where the first example is a "laugher" class. It contains a single method which prints "haha" to System.out. It quickly evolves to include a parameter specifying the sound to make and an instance variable to store the default laugh. This background is used to develop an interactive I/O class.

An extended example in chapter 5 uses two cooperating classes.

I/O: I/O streams are the *big* thing in the first four chapters of the book. In addition, each chapter includes a "GUI supplement". Early supplements focus on graphics. Event handling is introduced in chapter 6 to create a calculator.

Design: Design issues are introduced immediately after the mechanics of writing a class. Much of chapter 5 is an example where a tollbooth collects tolls from trucks. Extensive time is spent identifying the objects, defining the interfaces, etc. The design process shown is a reasonable introduction and they do use it often throughout the text.

Opinions: One of the strongest texts in illustrating the design process. Other highlights include looping patterns and a clear example distributing responsibilities among multiple classes. I wish I/O streams would be downplayed in the early chapters.

6.2 Barnes

Reference: Barnes, David J.. *Object-Oriented Programming with Java: An Introduction*. Prentice Hall, 2000. 951 + 77 pages.

Objects: HelloWorld is used to introduce the syntactic elements of a program but is quickly followed by an extensive discussion of creating and using a predefined ship object. A second example, bank accounts, already introduces two interacting objects – one for the bank and one for accounts. The next chapter discusses writing classes (a class implementing reminder notes).

I/O: Console I/O (via a provided class) is used for most of the text. There is substantial coverage (270 pages) of GUI applications and applets.

Design: There is little support for design issues other than several case studies.

Opinions: One of the more comprehensive texts.

6.3 Bell and Parr

Reference: Bell, Douglas and Mike Parr. *Java for Students*. Prentice Hall, 1999. 524 + 62 pages.

Objects: Applets are used from the beginning and thus students write and see classes from the beginning. The concepts of object and class are not discussed, however, until after graphics, numeric calculations, methods, events, selection and repetition have all been discussed. At this point a balloon class is introduced and used. There are no examples of distributed responsibilities.

I/O: The text uses applets extensively. Event-driven programming is introduced by page 68 to respond to scrollbars. Other components are added later. Console programs are covered briefly towards the end.

Design: Design issues are left to the end of the book where there are chapters on OO design, testing and debugging. There is no discussion of the software lifecycle.

Opinions: I think OO programming gets lost in the midst of all the GUIs.

6.4 Bishop

Reference: Bishop, Judy. *Java Gently: Programming Principles Explained*. Addison-Wesley, 1998.

Objects: Chapter 2 provides a survey of Java, ending with a few examples using the Date class. Chapter 3 is devoted to methods, most of which are static and called from main. The last two sections discuss classes and show a case study. The next three chapters, however, are full of programs which use little more than `main()`.

Chapter 7 uses objects extensively and includes a case study where responsibilities are distributed.

I/O: Console I/O is done with a provided class which is then developed half way through the book. GUIs (both applications and applets) are discussed in chapters 9 and 10.

Design: There are six case studies but very little on software engineering issues.

Opinions: I dislike the overuse of `main()` and static methods as well as the way objects are de-emphasized until late in the book.

6.5 Decker and Hirshfield

Reference: Decker, Rich and Stuart Hirshfield. *Programming.java: An Introduction to Programming Using Java*. Brooks/Cole, 2000. 593 +24 pages.

Objects: The authors begin with GUIs – and emphasize them throughout the textbook. They do not appear to develop any classes that do not extend a class from `java.awt.*`.

I/O: Nearly everything is done with applets. There is a chapter on file I/O but nothing on console I/O.

Design: The authors advocate designing the user interface first and then filling in the details to make it work. There is a large ATM case study in chapter 7 which uses this ap-

proach. The result, however, only handles the interface for the ATM.

Opinions: I think students would finish this text with a very skewed idea of programming. I fear they would not be able to handle any programming task which is not part of a graphical user interface and would not even really understand what objects and classes are.

6.6 Deitel and Deitel

Reference: Deitel, H.M., and P.J. Deitel. *Java: How to Program*. Prentice Hall, 1999. 1,228 + 116 pages.

Objects: Classes and objects do not appear to be used in depth until chapter 8 where the first example is a time class. However, students extend `JApplet` starting with chapter 2, and use some existing classes such as `JLabel` and class methods much earlier. Chapters on methods and arrays also appear before the in-depth chapter on classes.

I/O: `JOptionPane` is used extensively for I/O early in the text. GUIs are covered extensively in chapters 11-13. Console I/O is not discussed.

Design: There are a few examples of pseudo-code, but little else to illustrate design principles or the software life-cycle.

Opinions: The key selling point is the comprehensiveness. It covers threads, images, audio, the database connectivity API, servlets, etc. On the downside, I find it very hard to read (a combination of typography, verbosity, and organization). Classes and objects are used a long time before they are adequately explained.

6.7 Horstmann

Reference: Horstmann, Cay. *Computing Concepts with Java 2 Essentials*. John Wiley & Sons, 2000. 681 + 181 pages.

Objects: Classes are described early and used sparsely (strings, a console reader, `System.out`) in chapters 1 and 2. Chapter 3 develops and uses a bank account class. A coin class provides a second example. Distributing responsibilities in several classes occurs in the array chapter (11).

I/O: A very simple console class is introduced early. This is followed immediately with an "Advanced" section on reading input with standard Java. The next chapter is on writing classes and includes a section on writing the custom class.

Chapter 4 discusses applets and graphics. Event handling and the AWT comes much later in chapters 10 and 12.

Design: Class discovery using nouns and verbs is introduced just after writing classes. It's picked up again in much more detail in Chapter 14. There is an entire chapter devoted to testing and debugging (8).

Opinions: I find the writing to be very clear. One of my favorite texts.

6.8 Koffman and Wolz

Reference: Koffman, Elliot and Ursula Wolz. *Problem Solving with Java*. Addison Wesley Longman, 1999. 681 + 143 pages.

Objects: Using and writing classes are introduced together on page 44 with a class to convert fabric measurements from square meters to square yards. Converting numbers of nickels and pennies to dollars and change is a second example. Distribution of responsibilities between two classes occurs in Chapters 6 (a bill-paying program) and 7 (an employee database using arrays).

I/O: Programs requiring I/O extend (rather than use) a provided GUI package. `System.out` is used for debugging; console input is not discussed while file I/O is only via provided classes.

GUIs are discussed in chapters 9 and 10.

Design: The waterfall model is discussed early in Chapter 2. Many later case studies reinforce the model. Chapter 8 discusses the design implications of inheritance, abstract classes, etc..

Opinions: Many of classes are simply containers for `read()`, `process()`, and `output()` (suitably renamed). Experienced Java programmers would not create a class to solve these problems.

6.9 Lambert & Osborne

Reference: Lambert, Kenneth A. and Martin Osborne. *Java: A Framework for Programming and Problem Solving*. PWS Publishing, 1999. 494 + 39 pages.

Objects: Objects and classes are explored in chapter 8 via a student class. Readers have seen and written many classes prior to that, but they all follow a template which makes use of the author's GUI package. An example in chapter 9 (arrays) distributes responsibilities between three classes.

I/O: Graphical user interfaces are used throughout the text. The authors provide a package, BreezyGUI, which makes fairly sophisticated interfaces pretty reasonable. The feel is much like the Java 1.0 event model where specific methods are overridden rather than registering listeners. Chapter 17 introduces the AWT.

Console I/O is buried in chapter 13 along with file I/O.

Design: There are a large number of short case studies, each of which is broken into specification, analysis, design and specification. These sections don't often have a lot of elaboration, however. Each chapter has a short section on design, debugging and testing hints. They are set in a smaller font, however, making them seem less important.

Opinions: The `BreezyGUI` package allows the authors to use sophisticated interfaces from the beginning without getting sidetracked by the interfaces. I wish the students had more opportunities to write their own classes which don't follow the author's GUI cookie-cutter.

6.10 Lewis and Loftus

Reference: Lewis, John and William Loftus. *Java Software Solutions*. Addison-Wesley, 2000. 515 + 265 pages.

Objects: Objects are introduced with `System.out` and strings early in chapter 2. Later in the chapter instances of `Random` and `NumberFormat`, along with static methods from `Math` are used. All of the classes shown at this point have a single method: `main`. A GUI supplement shows how to draw a snowman in an applet.

Objects and classes reappear in chapter 4 (after a chapter on control structures) where classes modeling coins, rational numbers and dice are developed. There are very few, if any, examples where responsibilities are distributed between classes, except for using collection classes and inheritance.

I/O: A custom class for console I/O is introduced in chapter 2. It's explained in chapter 8 along with other I/O streams.

Each chapter includes a "GUI Supplement". Early supplements focus on graphics. Event handling for mouse and keyboard listeners makes up the bulk of chapter 5. Input from text fields is covered in chapter 7. Throughout, only applets are discussed.

Design: The waterfall model and pseudocode are briefly described in chapter 3 in the context of developing algorithms but is not reinforced in the rest of the text. Chapter 10 includes a more extensive discussion of development process models and a partial case study.

Opinions: A very competent text. I wish the object usage in chapters 2 and 3 was stronger.

6.11 Morelli

Reference: Morelli, Ralph. *Java, Java, Java: Object-Oriented Problem Solving*. Prentice-Hall, 2000. 912 + 52 pages.

Objects: Chapter 1 walks through a program using `System.out` and a rather sophisticated animated applet. After a brief discussion in chapter 2 of how objects can interact to solve a problem, a pair of classes are written. First a `Rectangle` class and then a `RectangleUser` class (containing `main`). This pair then illustrates how classes are used. A "cyberpet" is then developed. This case study is picked up again in chapters 3-8 and 10, becoming ever more complex.

A case study in chapter 8 shows an application which distributes responsibilities between two classes.

I/O: Each chapter has a "From the Java Library" section. In chapter 2 this covers keyboard input using a `BufferedReader`. In chapter 7, `StringTokenizer`.

Applets and simple event-based programming using buttons is introduced in Ch 4. There is much more on GUI widgets and graphics in chapters 9 and 10.

Design: Chapter 2 contains a 10 page discussion on program design – just after students have seen `helloWorld`. A number of case studies throughout the text reinforce the design methodology. In addition, many chapters contain a section titled "Object-Oriented Design" which offer OO design tips for concepts just introduced.

Opinions: I like this text. One small complaint is that each chapter has a number of threads (object-oriented design, examining a class from the Java library, something to do in the lab, etc.), making the text feel disjointed.

6.12 Nino and Hosch

Reference: Nino, Jaime and Frederick Hosch. *Introduction to Programming and Object-Oriented Design Using Java*. John Wiley & Sons, 2001[8]. 667 + 84 pages.

Objects: Classes and objects are described in depth at the beginning of the text. The two subsequent chapters discuss class specification and implementation. The first program is a test driver; the first complete program is presented in chapter 9 and involves five interacting classes. Students write portions of programs earlier in a lab setting.

I/O: All I/O is done using the model-view-controller architecture. The majority of the text uses provided classes for console I/O (explained in an appendix). GUIs are covered late in the text.

Design: The authors emphasize design issues extensively. Class specification is discussed before implementation. Entire chapters are devoted to programming by contract, testing, and software quality. An extensive case study and the model-view-controller pattern are included.

Opinions: If you are concerned with design issues, this is the text to use. It's worth reading regardless of whether it's used in class.

6.13 Savitch

Reference: Savitch, Walter. *Java: An Introduction to Computer Science & Programming*. Prentice Hall, 1999. 691 + 35 pages.

Objects: Writing classes and using objects are presented together, after primitive types, strings, and control structures. The first example, revised several times, models a species with a name, current population and growth rate. A small example in the arrays chapter uses a pair of interacting classes.

I/O: Most of the text uses console I/O via a provided class. Chapter 7 discusses GUIs and the last chapter discusses applets.

Design: Information hiding is discussed several times, but little to nothing on other design topics.

Opinions: I wish there had been an explanation of why objects are good and a few larger case studies.

---

[8] This review is based on a draft manuscript dated March, 2000 and checked against the most recent version at http://www.cs.uno.edu/~fred/OOJ/Utilities/.

6.14 Slack

Reference: Slack, James M.. *Programming and Problem Solving with Java*. Brooks/Cole, 2000. 1007 + 130 pages.

Objects: Many object-oriented concepts are introduced in Chapter 2 via classes implementing turtle graphics. Method invocation, extending a class with new methods, parameters, selection, and iteration are all covered. These concepts are all revisited in later chapters.

Writing a new class is introduced after students have seen many objects used in the selection and iteration chapters. Example classes include a counter, money and bank account.

I/O: Console input is done with a custom class. Development of the class in shown when writing classes. GUIs are relegated to several appendices.

Design: Chapter 10 discusses design issues, including inheritance, abstract classes and software development methodologies. These issues are not reinforced elsewhere in the text.

Opinions: Chapter 2 is the strength and the weakness of this text: it provides a good example of using objects before students write their own classes. But it introduces a lot of information very quickly. I think students would be left overwhelmed. Karel the Robot [Pattis81] is an obvious inspiration but is not improved upon.

6.15 Stein

Reference: Stein, Lynn Andrea. *Interactive Programming in Java*. Due to be published by Morgan Kaufmann Publishers in late 2000.

Objects: The text begins with an in-depth discussion of OO programming as "coordinating a computational community". This is illustrated describing a string transformation program and writing a number of transformers by extending a provided class. Subsequent chapters discuss statements and flow of control using a mixture of objects and primitive types and specifying classes via their interfaces before the chapter on implementation.

I/O: Most of the text uses console I/O via a provided class. The last two chapters discuss GUIs but the emphasis is clearly on event-driven programming paradigm.

Design: Quite a bit of attention is given to specifying and using interfaces and how objects interact with each other. Chapter 8 discusses OO design, including class discovery with nouns and verbs. Case studies and examples of applying a development methodology are missing.

Opinions: This is one of the most interesting texts due to its emphasis on threads and interacting objects. It contains a large number of helpful analogies.

6.16 Wu

Reference: Wu, C. Thomas. *An Introduction to Object-Oriented Programming with Java*. WCB/McGraw-Hill, 1999. 821 + 37 pages.

Objects: Using objects to solve problems is demonstrated from the beginning using a custom graphical I/O package. The first class development comes in Chapter 4 after a chapter on numerical data. The primary examples are a currency conversion class and a loan calculator.

Responsibilities are distributed between classes from the beginning, but only due to I/O. The first time two classes are written for the same problem is an address book in Chapter 9 on arrays.

I/O: I/O in the majority of the book is handled by the custom GUI I/O package. Chapter 12 is about Java GUI interfaces. Console input is not discussed.

Design: The waterfall model is discussed in Chapter 1. Each chapter includes a sample program which models the process. A more incremental approach is demonstrated in Chapter 14. UML-like diagrams are used.

Opinions: This text has an uncommonly large number of illustrations and is easy to read. I like the extensive use of objects early. A text to consider.