

Preface

This text is an introduction to programming using the Java programming language. It differs from most other such texts in several ways.

The livetext ProgramLive

Accompanying the text is a CD, which is itself a complete multimedia text on programming in Java. This CD acts like a book: you can turn the pages, look at the table of contents, the glossary, the index, and so forth. And because it is computerized, you can get to a particular page from the table of contents or the index with a click of your mouse.

Each page of the CD contains recorded lectures with synchronized animation. These lectures teach, using narration and animation, in a way that is impossible in a paper text. For example, students can watch execution of a method call, seeing the frame for the call being drawn and later erased. They can see the step by step development of a method body using step-wise refinement. They can see what abstraction means in a program, as they read a loop body in terms of an English statement and an increment of a counter, then have everything disappear from view except the English statement, and see its implementation.

An instructor can show some of these animated lectures in class —you would be amazed at how much energy that saves. Moreover, these lectures are repeatable: students can watch them at home.

The CD also contains all the programs that are used in this text, as well as material for over 35 guided closed-lab sessions.

Thus, this multimedia CD is a significant step forward in the kinds of materials that we give our students. It has been used successfully in a self-paced (no-lecture) and in a distance-learning course.

In this paper text, the lefthand margin contains many references to the CD (like the one on the left of this paragraph), emphasizing the places where the student might learn better by listening to (and watching) an animated lecture.

Activity
1-1.2

Using DrJava or BlueJ

This text makes a second departure from the norm. Typically in a course, students have several assignments that call for writing and testing Java programs. But students do not practice much outside of those assignments, so they do not become fluent in Java by the end of the semester. The emergence of DrJava, a new IDE from Rice University, can change this mode of operation. DrJava contains an “Interactions pane”, where one can type any expression or statement and have it evaluated or executed immediately, *without having to have a full Java program*. There is no need for a class and no need for a method `main` with a `String[]` parameter. There is no more need for magic! Beginning students can use the Interactions pane to practice; until they write a full class, there is no need to compile. For example, after a lecture on `int` and `double` expressions, they can

practice with such expressions and gain a real understanding of integer division, the remainder operation, and casting. Also, after a first lecture on creating an object and referencing its components, they can practice doing just that, without having to have a full program. Self-help exercises after many sections of this text are designed to get the students to practice with Java more frequently. Studying should involve doing as well as reading.

We have found DrJava to be so useful that we use it almost every day in lecture (with a projector that displays the computer monitor on a screen). It has radically changed our lecture style. We can use it to demonstrate the points mentioned above and much more, such as step-wise refinement.

This preface is not meant to convince you that DrJava is the One True Way. It *is* effective, but other IDEs may provide a similar experience, although, so far we have found them to be less polished and less intuitive. The excellent BlueJ, for example, has an expression evaluator, but at the time of this writing it does not allow variables to be declared and used in later expressions.

DrJava is free. Appendix I explains how to download and use it. Even if you prefer that your students use a more conventional IDE for writing regular programs, DrJava provides a wonderful practice environment, and something like it should be used for the first month of the term.

Objects as manilla folders in filing cabinets

This text differs also in its introduction to classes and objects. A class is viewed as a drawer of a filing cabinet filled with manilla folders, which represent the instances or objects of the class. Each manilla folder has a distinct name on its tab, and it is this name that is stored in variables of the class-type. This approach allows us to dispense with the terms “reference” and “pointer” at the beginning (we introduce them later), for which students have no concrete analogy, and makes the introduction of classes, objects, and variables much easier.

Some reviewers have asked us to use UML diagrams instead of manilla folders. We cannot do that because UML diagrams and manilla folders are used for entirely different purposes. UML diagrams are used when designing or analyzing a program, for example, to show relations between classes. Manilla folders are used when executing statements by hand. For example, evaluating a new-expression requires drawing a new manilla folder that represents the new object. The manilla folder has a “tab”, which contains the name of (or reference to) the object. No such animal exists in the UML world.

This file-cabinet–manilla-folder analogy, together with DrJava, allows us to comfortably introduce the creation and manipulation of objects as early as the second lecture (after a lecture on expressions), and students can begin practicing immediately, creating objects themselves on the computer and interacting with them. This occurs before they see a class definition!

Also, with this analogy, we are able to discuss a general *inside-out* rule that is used in some manner in almost all programming languages: a construct (such as a method body) can reference variables and methods defined not only in that

construct but in any surrounding construct (such as a class definition). This inside-out rule makes it easy to discuss issues of scope in various contexts.

Finally, the analogy extends easily to the discussion of nested classes (both static classes and inner classes), and we are able to show how a “flattened view” of an inner class is similar to how Java implements nested classes. We have not seen a text that does this so easily. Of course, the material on nested classes is beyond the first course, but the fact that we can (and do) treat this material is some indication of the appropriateness of this analogy.

The pace of the text

Some reviewers of this text felt that the first chapter has a great deal of material and is fast-paced. Yes, there is a lot of material, but our experience is that students can handle it. We believe this is due to a combination of factors:

1. The use of DrJava in the classroom to demo concepts and Java features.
2. The self-review exercises in almost each section of the first chapter (and often in later chapters). Practice is needed to gain ease and fluency.
3. Weekly mandatory closed labs, where the students are guided in using DrJava. (The CD contains over 35 guided labs.)
4. The file-drawer–manilla-folder analogy, which provides the students with a concrete idea of what an object is.

An execution model

The text has another difference from most texts. We introduce a “model of memory”, which includes not only drawing objects but also executing method calls, including pushing a frame on a call stack and popping the frame when the method call is finished. We have several assignments in which the students have to execute method calls by hand, and they may be asked to do the same on tests. Without such a model of memory, the whole idea of a method remains vague, and nested method calls such as $f(g())$ are, to some students, bewildering. With the model of memory, students have a more concrete understanding, and we have found that a later explanation of how recursive calls work is trivial.

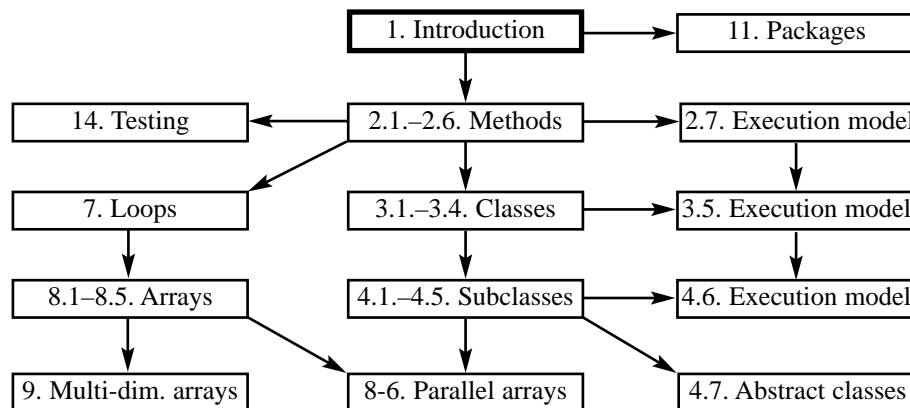
Knowing that some instructors will not want to introduce it, we have structured the text so that the sections on the model can be skipped.

Prerequisite structure of this text

Different instructors have different ideas on when to teach what in introductory programming (using Java). Some teach procedural programming first; others do object-oriented programming first. Some teach for-loops early; others wait until classes and subclasses are finished. Some teach class `Vector` early so that students can write programs that deal with collections of objects; others never mention `Vector`. Some teach file I/O; others do not.

This textbook is organized to allow for such variation. Chapters 1 and 2 give basic introductions to expressions, types, assignment, if-statements and for-loops, methods and method calls, and classes and subclasses (not in the order list-

ed) and are prerequisites for the rest of the text. The following prerequisite chart (for part of the text) shows the order in which material can be taught after (or at the same time as) the first two chapters.

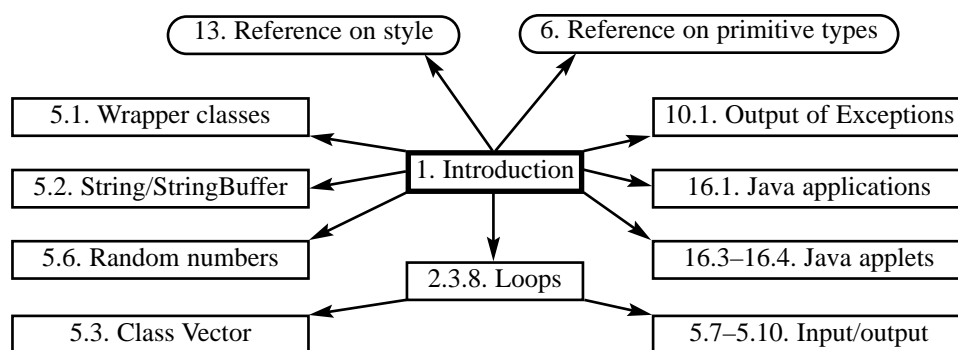


The short chapter on packages is needed to explain the import statement; we do not expect students to write their own packages.

Testing should be integrated throughout the course. However, for organizational purposes, it is best to have a separate chapter on testing and debugging, which the instructor and student can refer to from time to time. It is up to the instructor just how and when to cover this topic.

The three sections on the “execution model” are separate enough so that they can be skipped, should an instructor choose to do so.

The prerequisite diagram shown above gives the overall, gross structure. But different teachers emphasize different topics within that structure. The following diagram shows when various topics can be introduced.



Much of the material in Chaps. 6 (primitive types) and 13 (programming style) appears in different places throughout the text; these chapters provide more

thorough explanations and serve as references. They do not have exercises.

We provide several alternatives for I/O. Section 5.7 discusses a class, `JLiveRead`, that we have written to provide methods for reading integers, strings, etc, from the keyboard (i.e. Java console). We no longer use this class, preferring in the beginning to use either the DrJava interactions pane or our GUI `JLiveWindow` for input.

`JLiveWindow`, discussed in Sec. 5.8, provides a number of **int**, **double**, and **String** fields and a “ready button”. When the button is pressed, a certain method is called, and this method can be changed to do whatever you want. This GUI is great for providing simple input and for testing methods.

If you prefer to show your students the basics of reading from the keyboard and read/writing files, including appending to a file, using the classes of package `java.io`, use Secs. 5.9 and 5.10.

Style Note
Chap. 13
Introduction to
style issues

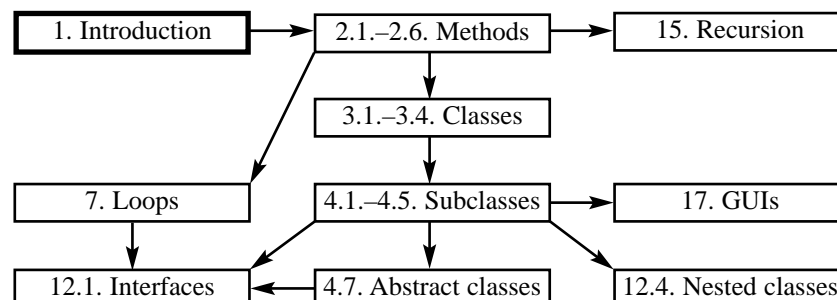
Various issues of style take several paragraphs or even a page to discuss. To place these discussions in the text would dilute the material. For example, placing a discussion of the various ways to indent an if-statement in the section that introduces the if-statement would distract from the topic at hand. We get around this problem by placing all discussions of style in Chap. 13 and placing references to the pertinent style material in the left margin, as in this paragraph.

Section 10.1 can be read at any time to help students understand the output resulting from thrown `Exceptions` and `Errors`.

Because we use DrJava or BlueJ, there is no need to introduce method `main` until well into the semester. Section 16.1 summarizes what one has to know about method `main`, and an instructor can choose to introduce it whenever they want. Similarly, sections in Chap. 16 on applets can be studied at any time.

The other topics in the diagram are discussed in terms of particular Java classes, and they can be studied after Chap. 1. We have suggested studying class `Vector` and I/O after a brief introduction to loops because using these classes in an interesting way usually requires loops.

Several topics covered in this text are not usually included in a first course: recursion, interfaces, nested classes (including anonymous classes), and the GUI packages in Java. We show in the diagram below when these topics can be taught. (The section on interfaces requires loops because one of the major uses of interfaces is in classes `Enumeration` and `Iterator`.)



Acknowledgements

The computer science departments at Cornell, Toronto, and UGA have been extremely supportive of our work. We thank the introductory programming classes—and their instructor—who have used drafts of this book and the CD over the years.

The people at Data Description are great! Matt Clark wrote the software to produce the first livetext, *ActivStats*. As we wrote the CD *ProgramLive*, he responded quickly to our calls for changes and additions to fit our needs. John Sammis, the business manager, has been our constant companion and “encourager” for several years. Paul Vellemen, the author of *ActivStats*, was instrumental in getting us started on *ProgramLive*. And a cadre of other people at Data Description have been supporting the project in many ways, like producing the icons for each activity, cleaning audio files, synching animations to audios, and translating the CD from its Macintosh author-base to the Windows environment. The amount of work to be done to produce a livetext continues to amaze us.

We want to thank reviewers of the text, who helped tremendously.

Ann Kostant, Wayne Wheeler, and the rest of our Springer-Verlag contacts—deserve special mention for their great advice and for getting this book published in near-record time. Laurie Buck did an excellent job of as copy editor for this book, again in record time.

Last—and certainly not least—we thank our wives, Elaine Gries and Petra Hall, for all they have put up with over the years. They have been supportive in countless way and patient in many more during the long, drawn-out months and years of this project.

David Gries
Paul Gries



<http://www.springer.com/978-0-387-22681-1>

Multimedia Introduction to Programming Using Java

Gries, D.; Gries, P.

2005, XVIII, 536 p. 420 illus., Softcover

ISBN: 978-0-387-22681-1