2002

Data Structures with Java: A Laboratory Approach

Joe Kent

Lewis Barnett III
University of Richmond, lbarnett@richmond.edu

Preface

Next Steps

After developing basic Java programming skills in an introductory course, students must, in the second course, be introduced to the fundamental ways to organize and manage data. Programs in the first course are often small by the standards of practicing programmers, usually just a few hundred lines. Real applications are much larger, and we need to manage complexity and organize data for efficient access. "Data structures" is a common name given to the second course. While this name emphasizes data organization, abstraction is essential for the management of complexity. That abstraction is provided by classes that support the organization of the data, yet restrict access to the details of that organization. For example, a stack is a linear organization of information that behaves in a last-in, first-out manner. Its information is accessed by a restricted set of methods. Several common implementations of the class are possible, yet the programmer who uses the class need not know the implementation details. On the other hand, the course needs to explore implementation techniques so that the student can create new data structures with efficient implementations. In summary, the next stage in the development of a computer science student's knowledge is a mixture of design, abstraction, and implementation of classic data organizations.

We believe that learning is enhanced by guided hands-on work in a laboratory setting, where new ideas can be explored and implemented. This book emphasizes this philosophy by calling each chapter a "laboratory." Each laboratory has pre-laboratory reading, exercises, and review questions that present the new concepts. The hands-on portion of the laboratory asks the student to write simple programs that illustrate the new ideas, plus complete a component of a larger application program. With the explanatory material, this book can be used as both a textbook

and a laboratory manual for courses following the guidelines for CS2. It can also be used as a self-directed learning tool.

Coverage

This book is designed to present the key topics in the second course for computer science students, commonly called CS2, using the Java programming language. The central topics in such a course include multidimensional arrays, structures created using dynamic memory allocation such as linked lists and trees, stacks, queues, heaps, files, sets, and their applications. Algorithms for hashing and file compression are also included. For the abstract data types (ADTs), stack and queue, we present both array and linked list implementations, and show how they can be used as ADTs. We feel that examining the implementation choices of an ADT provides the foundational understanding and programming skills required for more advanced work.

For convenience, we cover exceptions and file operations in Java, although this may have been covered in a first course. We also include material on the binary representation of data and Java's bitwise operations, with applications. These are topics needed for computer organization and operating systems courses.

Objects are central in Java, but Java can be used without much attention to object-oriented programming concepts. We think that students need a good look at class hierarchies, inheritance, and other fundamental ideas in object-oriented programming. A natural way to do this is by exploring Java's use of graphical components. In a sequence of three labs, we explore the basic ideas of computer graphics and develop a large object-oriented application, the FreeCell solitaire game. The third laboratory reuses many of the classes to develop another card game, illustrating the reusability of a well-designed set of classes.

The capstone laboratory is "File Compression." It uses trees, heaps as priority queues, file operations, and bitwise operations to implement Huffman encoding of data. It requires two laboratory sessions to complete.

Assumptions

We assume that the reader has completed a one-semester first course in computer science that focuses on programming using Java. We assume knowledge of fundamental Java language features, plus basic classes such as String and Math, methods and their use, arrays, recursion, plus algorithms for searching and sorting. Sorting algorithms should include at least one nlog(n) sort such as merge sort or quick sort. Searching should include linear and binary search algorithms. Students should be able to write and test programs using these topics. If our textbook *Basic Java Programming: A Laboratory Approach* was used in the first course, then the material up through Laboratory 12 that does not involve graphics should have been covered. Ideally, the laboratory "I/O and Exceptions" would have been covered, but we reproduce it here in case that was not in the reader's first course.

Students whose first course used the C++ language should be able to make a quick transition to Java. The best way to make that transition is to redo in Java some C++ assignments from the first course. This may take the first week or two of the second course. We do not provide introductory material on basic Java. It is available in many books.

We would like these labs to be independent of any computer environment, but that is not totally possible. We assume the environment to be Microsoft Windows 95/98/NT/2000. The program files were developed in that environment. It should be possible to use these laboratories in general Unix or Linux environments, provided one uses the common dos2unix command to strip the extra character inserted by Windows in the program files on the diskette. Of course, in non-Windows environments the diskette would have to be appropriately mounted.

IDEs

It is possible to create Java programs using a simple editor like Notepad and the tools of the free Java Development Kit (now called the SDK) from Sun Microsystems, but that approach is relatively primitive. A number of integrated development environments, or IDEs, are available. These are applications that use a graphical user interface to provide windows, menus, buttons, and so on to aid in program development and testing. We prefer Kawa by Allaire (**http://www.allaire.com/products/kawa/index.cfm**) because it allows for easy creation of very simple programs and can be used with any version of Java. However, we have made every attempt to make this manual IDE-independent. An appendix covers the basics of several common IDEs that support Java™ 2. Only the appendix on debugging explicitly uses the Kawa IDE for compiling and running a program with the debugger.

LabPkg

We want these labs to be as independent as possible of any particular textbook. For that reason we have developed a Java package, `LabPkg`, that we use for the user interface of our applications. The package allows programs to interact with the user via graphical components. The critical class is `ViewFrame`, which is instantiated as a window with three components:

- a toolbar with a provided Exit button that kills the current application
- a scrolled text area for displaying program output
- a "canvas" for displaying images or drawing figures

The canvas component is optional and is not used in most simple applications. The user may add additional buttons to the toolbar for initiating actions. Interactive input is obtained by pop-up dialog windows generated by calls to simple methods.

If you looked at another textbook, it would have a different package, as every author has a favorite way to do input in Java. Why? Java supports input from the keyboard on a line-by-line basis but forces the user to process the string of characters to extract the desired numeric values and handle errors. For beginners, the details of this task distract from the primary goal of understanding principles of programming that apply generally. Special packages hide the mess. Unfortunately, there is no standard package at this time.

Our package is based on the paradigm of a program as having three components: the model, the view, and the controller. (This is the MVC paradigm developed by Smalltalk programmers more than 20 years ago.) The *model* is simply the data that represents the current state of the program. The *view* is what the user sees of the data. The *controller* is the part that provides interaction of the user with the program. Often the view and controller components

are joined in programs that use a graphical user interface. For most of our programs, a `ViewFrame` object provides the view and controller components.

Many of the provided test programs use this package to create a simple graphical interface. Appendix A provides information about the package and its use, with simple examples. You should install the package so that the Java compiler and interpreter can find it. See the information on personal system setup that comes on the diskette.

Getting Files

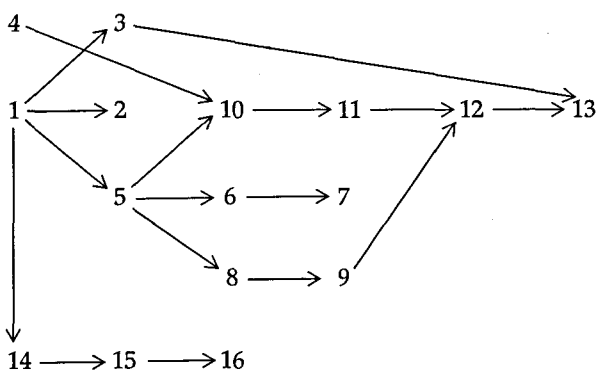
A diskette accompanies the text and contains all of the files required for the laboratories, plus example programs discussed in the pre-lab readings. The file **index.html** provides a guide to the files and documentation and should be opened in a browser.

Using a browser, the files can be "downloaded" by the student from the diskette to a local hard drive or a network drive. It is also possible to use Windows Explorer to copy the files from the diskette. The files are in subfolders of the folder **labfiles.**

The diskette contains information concerning setting up a personal computer to do the hands-on part of the text.

Dependencies among Labs

We assume basic Java programming skills prior to the first laboratory. The chart below gives the content dependencies among the laboratories.



The minimum content for a semester course would include all of the laboratories except possibly 3, 13, 14, 15, and 16. Labs 1 and 2 may be omitted if the material was covered in a previous course. Lab 4 on recursion may be review for some students, but it is worth spending more time on it, if only to cover quick sort and merge sort.

Lab 13 is a capstone laboratory and may be developed as a major project. It takes two sessions to complete and still has a large post-laboratory exercise. Labs 14, 15, and 16 present graphics and a large object-oriented example. These labs are not central to the data structures aspect of the course, but they can provide an enriching experience if the first course had minimal object-oriented programming. If all of the laboratories are to be completed, they require 18 sessions.

The Authors

We are computer science faculty at the University of Richmond, with more than 38 combined years of teaching experience. We developed early Java versions of these laboratories as special experiences for our students more than five years ago. In the fall of 1999, they were expanded and used in a formal laboratory setting. Experience and feedback from other faculty allowed us to refine those initial materials.

Thanks

We would like to thank our faculty colleagues who provided feedback. Special thanks go to Jim Leisy, our editor, who had faith in us. Thanks also to Sue Page, Stephanie Welch, Tom Sumner, Ian Shadburne, Christine Collier, Krista Brown, and Misty Harland at Franklin, Beedle & Associates. Of course, we cannot forget the hundreds of students at UR who provided candid responses to their experiences with the materials.

For supporting us while we worked on the manuscript, we give special thanks to our wives, Mary Kent and Rebekah Barnett.