GLOBAL EDITION

Building Java™ Programs

A Back to Basics Approach

FOURTH EDITION

Stuart Reges • Marty Stepp

Pearson

Fourth Edition
Global Edition

Building Java Programs

A Back to Basics Approach

Stuart Reges
University of Washington

Marty Stepp
Stanford University

Pearson

Boston Columbus Indianapolis New York San Francisco Hoboken
Amsterdam Cape Town Dubai London Madrid Milan Munich Paris Montreal Toronto
Delhi Mexico City Sao Paulo Sydney Hong Kong Seoul Singapore Taipei Tokyo

Vice President, Editorial Director: Marcia Horton
Acquisitions Editor: Matt Goldstein
Editorial Assistant: Kristy Alaura
Acquisitions Editor, Global Editions: Sourabh Maheshwari
VP of Marketing: Christy Lesko
Director of Field Marketing: Tim Galligan
Product Marketing Manager: Bram Van Kempen
Field Marketing Manager: Demetrius Hall
Marketing Assistant: Jon Bryant
Director of Product Management: Erin Gregg
Team Lead, Program and Project Management: Scott Disanno
Program Manager: Carole Snyder
Project Editor, Global Editions: K.K. Neelakantan
Project Manager: Lakeside Editorial Services L.L.C.
Senior Specialist, Program Planning and Support: Maura Zaldivar-Garcia
Senior Manufacturing Controller, Global Editions: Kay Holman
Media Production Manager, Global Editions: Vikram Kumar
Cover Design: Lumina Datamatics
R&P Manager: Rachel Youdelman
R&P Project Manager: Timothy Nicholls
Inventory Manager: Meredith Maresca
Cover Art: © Westend61 Premium/Shutterstock.com
Full-Service Project Management: Apoorva Goel/Cenveo® Publisher Services

Pearson Education Limited
Edinburgh Gate
Harlow
Essex CM20 2JE
England

and Associated Companies throughout the world

Visit us on the World Wide Web at:
www.pearsonglobaleditions.com

British Library Cataloguing-in-Publication Data
A catalogue record for this book is available from the British Library
10 9 8 7 6 5 4 3 2 1

ISBN 10: 1-292-16168-X
ISBN 13: 978-1-292-16168-6

Typeset in Monotype by Cenveo Publisher Services
Printed and bound in Malaysia.

Preface

The newly revised fourth edition of our *Building Java Programs* textbook is designed for use in a two-course introduction to computer science. We have class-tested it with thousands of undergraduates, most of whom were not computer science majors, in our CS1-CS2 sequence at the University of Washington. These courses are experiencing record enrollments, and other schools that have adopted our textbook report that students are succeeding with our approach.

Introductory computer science courses are often seen as "killer" courses with high failure rates. But as Douglas Adams says in *The Hitchhiker's Guide to the Galaxy*, "Don't panic." Students can master this material if they can learn it gradually. Our textbook uses a layered approach to introduce new syntax and concepts over multiple chapters.

Our textbook uses an "objects later" approach where programming fundamentals and procedural decomposition are taught before diving into object-oriented programming. We have championed this approach, which we sometimes call "back to basics," and have seen through years of experience that a broad range of scientists, engineers, and others can learn how to program in a procedural manner. Once we have built a solid foundation of procedural techniques, we turn to object-oriented programming. By the end of the course, students will have learned about both styles of programming.

Here are some of the changes that we have made in the fourth edition:

- **New chapter on functional programming with Java 8.** As explained below, we have introduced a chapter that uses the new language features available in Java 8 to discuss the core concepts of functional programming.
- **New section on images and 2D pixel array manipulation.** Image manipulation is becoming increasingly popular, so we have expanded our `DrawingPanel` class to include features that support manipulating images as two-dimensional arrays of pixel values. This extra coverage will be particularly helpful for students taking an AP/CS A course because of the heavy emphasis on two-dimensional arrays on the AP exam.
- **Expanded self-checks and programming exercises.** Many chapters have received new self-check problems and programming exercises. There are roughly fifty total problems and exercises per chapter, all of which have been class-tested with real students and have solutions provided for instructors on our web site.

Since the publication of our third edition, Java 8 has been released. This new version supports a style of programming known as functional programming that is gaining in popularity because of its ability to simply express complex algorithms that are more easily executed in parallel on machines with multiple processors. ACM and IEEE have released new guidelines for undergraduate computer science curricula, including a strong recommendation to cover functional programming concepts.

We have added a new Chapter 19 that covers most of the functional concepts from the new curriculum guidelines. The focus is on concepts, not on language features. As a result, it provides an introduction to several new Java 8 constructs but not a comprehensive coverage of all new language features. This provides flexibility to instructors since functional programming features can be covered as an advanced independent topic, incorporated along the way, or skipped entirely. Instructors can choose to start covering functional constructs along with traditional constructs as early as Chapter 6. See the dependency chart at the end of this section.

The following features have been retained from previous editions:

- **Focus on problem solving.** Many textbooks focus on language details when they introduce new constructs. We focus instead on problem solving. What new problems can be solved with each construct? What pitfalls are novices likely to encounter along the way? What are the most common ways to use a new construct?
- **Emphasis on algorithmic thinking.** Our procedural approach allows us to emphasize algorithmic problem solving: breaking a large problem into smaller problems, using pseudocode to refine an algorithm, and grappling with the challenge of expressing a large program algorithmically.
- **Layered approach.** Programming in Java involves many concepts that are difficult to learn all at once. Teaching Java to a novice is like trying to build a house of cards. Each new card has to be placed carefully. If the process is rushed and you try to place too many cards at once, the entire structure collapses. We teach new concepts gradually, layer by layer, allowing students to expand their understanding at a manageable pace.
- **Case studies.** We end most chapters with a significant case study that shows students how to develop a complex program in stages and how to test it as it is being developed. This structure allows us to demonstrate each new programming construct in a rich context that can't be achieved with short code examples. Several of the case studies were expanded and improved in the second edition.
- **Utility as a CS1+CS2 textbook.** In recent editions, we added chapters that extend the coverage of the book to cover all of the topics from our second course in computer science, making the book usable for a two-course sequence. Chapters 12–19 explore recursion, searching and sorting, stacks and queues, collection implementation, linked lists, binary trees, hash tables, heaps, and more. Chapter 12 also

received a section on recursive backtracking, a powerful technique for exploring a set of possibilities for solving problems such as 8 Queens and Sudoku.

Layers and Dependencies

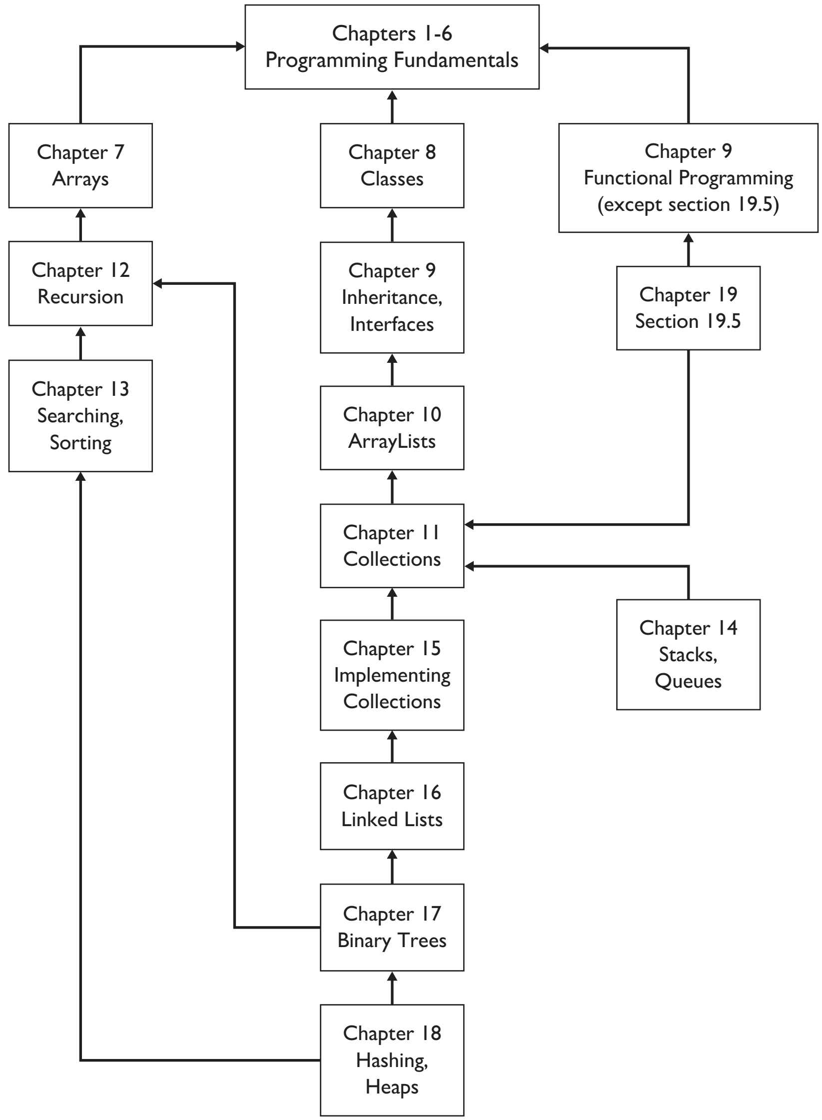
Many introductory computer science books are language-oriented, but the early chapters of our book are layered. For example, Java has many control structures (including for-loops, while-loops, and if/else-statements), and many books include all of these control structures in a single chapter. While that might make sense to someone who already knows how to program, it can be overwhelming for a novice who is learning how to program. We find that it is much more effective to spread these control structures into different chapters so that students learn one structure at a time rather than trying to learn them all at once.

The following table shows how the layered approach works in the first six chapters:

Chapter	Control Flow	Data	Programming Techniques	Input/Output
1	methods	`String` literals	procedural decomposition	`println`, `print`
2	definite loops (`for`)	variables, expressions, `int`, `double`	local variables, class constants, pseudocode	
3	return values	using objects	parameters	console input, 2D graphics (optional)
4	conditional (`if/else`)	`char`	pre/post conditions, throwing exceptions	`printf`
5	indefinite loops (`while`)	`boolean`	assertions, robust programs	
6		`Scanner`	token/line-based file processing	file I/O

Chapters 1–6 are designed to be worked through in order, with greater flexibility of study then beginning in Chapter 7. Chapter 6 may be skipped, although the case study in Chapter 7 involves reading from a file, a topic that is covered in Chapter 6.

The following is a dependency chart for the book:



Supplements

Answers to all self-check problems appear on the web site and are accessible to anyone. Our web site has the following additional resources for students:

- **Online-only supplemental chapters**, such as a chapter on creating Graphical User Interfaces

- **Source code and data files** for all case studies and other complete program examples
- The **`DrawingPanel` class** used in the optional graphics Supplement 3G

Our web site has the following additional resources for teachers:

- **PowerPoint slides** suitable for lectures
- **Solutions** to exercises and programming projects, along with homework specification documents for many projects
- **Sample exams** and solution keys
- **Additional lab exercises** and **programming exercises** with solution keys
- **Closed lab creation tools** to produce lab handouts with the instructor's choice of problems integrated with the textbook

The materials are available at www.pearsonglobaleditions.com/reges.

MyProgrammingLab

MyProgrammingLab is an online practice and assessment tool that helps students fully grasp the logic, semantics, and syntax of programming. Through practice exercises and immediate, personalized feedback, MyProgrammingLab improves the programming competence of beginning students who often struggle with basic concepts and paradigms of popular high-level programming languages. A self-study and homework tool, the MyProgrammingLab course consists of hundreds of small practice exercises organized around the structure of this textbook. For students, the system automatically detects errors in the logic and syntax of code submissions and offers targeted hints that enable students to figure out what went wrong, and why. For instructors, a comprehensive grade book tracks correct and incorrect answers and stores the code inputted by students for review.

For a full demonstration, to see feedback from instructors and students, or to adopt MyProgrammingLab for your course, visit the following web site: http://www.myprogramminglab.com/

VideoNotes

VideoNote

We have recorded a series of instructional videos to accompany the textbook. They are available at the following web site: www.pearsonglobaleditions.com/reges.

Roughly 3–4 videos are posted for each chapter. An icon in the margin of the page indicates when a VideoNote is available for a given topic. In each video, we spend

5–15 minutes walking through a particular concept or problem, talking about the challenges and methods necessary to solve it. These videos make a good supplement to the instruction given in lecture classes and in the textbook. Your new copy of the textbook has an access code that will allow you to view the videos.

Acknowledgments

First, we would like to thank the many colleagues, students, and teaching assistants who have used and commented on early drafts of this text. We could not have written this book without their input. Special thanks go to Hélène Martin, who pored over early versions of our first edition chapters to find errors and to identify rough patches that needed work. We would also like to thank instructor Benson Limketkai for spending many hours performing a technical proofread of the second edition.

Second, we would like to thank the talented pool of reviewers who guided us in the process of creating this textbook:

- Greg Anderson, Weber State University
- Delroy A. Brinkerhoff, Weber State University
- Ed Brunjes, Miramar Community College
- Tom Capaul, Eastern Washington University
- Tom Cortina, Carnegie Mellon University
- Charles Dierbach, Towson University
- H.E. Dunsmore, Purdue University
- Michael Eckmann, Skidmore College
- Mary Anne Egan, Siena College
- Leonard J. Garrett, Temple University
- Ahmad Ghafarian, North Georgia College & State University
- Raj Gill, Anne Arundel Community College
- Michael Hostetler, Park University
- David Hovemeyer, York College of Pennsylvania
- Chenglie Hu, Carroll College
- Philip Isenhour, Virginia Polytechnic Institute
- Andree Jacobson, University of New Mexico
- David C. Kamper, Sr., Northeastern Illinois University
- Simon G.M. Koo, University of San Diego
- Evan Korth, New York University
- Joan Krone, Denison University
- John H.E.F. Lasseter, Fairfield University

- Eric Matson, Wright State University
- Kathryn S. McKinley, University of Texas, Austin
- Jerry Mead, Bucknell University
- George Medelinskas, Northern Essex Community College
- John Neitzke, Truman State University
- Dale E. Parson, Kutztown University
- Richard E. Pattis, Carnegie Mellon University
- Frederick Pratter, Eastern Oregon University
- Roger Priebe, University of Texas, Austin
- Dehu Qi, Lamar University
- John Rager, Amherst College
- Amala V.S. Rajan, Middlesex University
- Craig Reinhart, California Lutheran University
- Mike Scott, University of Texas, Austin
- Alexa Sharp, Oberlin College
- Tom Stokke, University of North Dakota
- Leigh Ann Sudol, Fox Lane High School
- Ronald F. Taylor, Wright State University
- Andy Ray Terrel, University of Chicago
- Scott Thede, DePauw University
- Megan Thomas, California State University, Stanislaus
- Dwight Tuinstra, SUNY Potsdam
- Jeannie Turner, Sayre School
- Tammy VanDeGrift, University of Portland
- Thomas John VanDrunen, Wheaton College
- Neal R. Wagner, University of Texas, San Antonio
- Jiangping Wang, Webster University
- Yang Wang, Missouri State University
- Stephen Weiss, University of North Carolina at Chapel Hill
- Laurie Werner, Miami University
- Dianna Xu, Bryn Mawr College
- Carol Zander, University of Washington, Bothell

Finally, we would like to thank the great staff at Pearson who helped produce the book. Michelle Brown, Jeff Holcomb, Maurene Goo, Patty Mahtani, Nancy Kotary, and Kathleen Kenny did great work preparing the first edition. Our copy editors and the staff of Aptara Corp, including Heather Sisan, Brian Baker, Brendan Short,

and Rachel Head, caught many errors and improved the quality of the writing. Marilyn Lloyd and Chelsea Bell served well as project manager and editorial assistant respectively on prior editions. For their help with the third edition we would like to thank Kayla Smith-Tarbox, Production Project Manager, and Jenah Blitz-Stoehr, Computer Science Editorial Assistant. Mohinder Singh and the staff at Aptara, Inc., were also very helpful in the final production of the third edition. For their great work on production of the fourth edition, we thank Louise Capulli and the staff of Lakeside Editorial Services, along with Carole Snyder at Pearson. Special thanks go to our lead editor at Pearson, Matt Goldstein, who has believed in the concept of our book from day one. We couldn't have finished this job without all of their hard work and support.

Stuart Reges
Marty Stepp

Acknowledgments for the Global Edition

Pearson would like to thank and acknowledge the following people for their contributions to the Global Edition.

Contributor

Ankur Saxena, Amity University

Reviewers

Arup Bhattacharya, RCC Institute of Technology

Soumen Mukherjee, RCC Institute of Technology

Khyat Sharma

BREAKTHROUGH
To improving results

MyProgrammingLab™

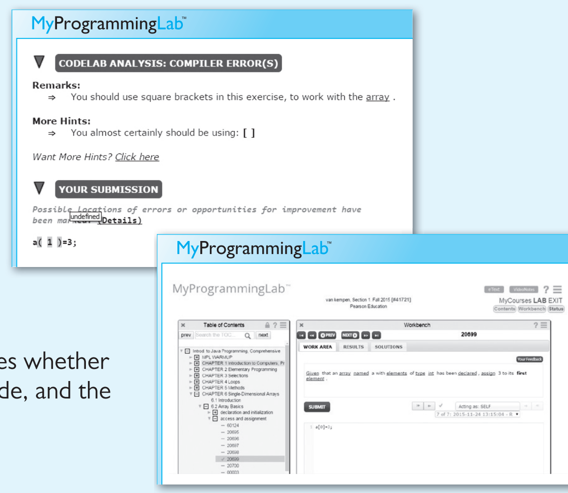
Through the power of practice and immediate personalized feedback, MyProgrammingLab helps improve your students' performance.

PROGRAMMING PRACTICE

With MyProgrammingLab, your students will gain first-hand programming experience in an interactive online environment.

IMMEDIATE, PERSONALIZED FEEDBACK

MyProgrammingLab automatically detects errors in the logic and syntax of their code submission and offers targeted hints that enables students to figure out what went wrong and why.



GRADUATED COMPLEXITY

MyProgrammingLab breaks down programming concepts into short, understandable sequences of exercises. Within each sequence the level and sophistication of the exercises increase gradually but steadily.

DYNAMIC ROSTER

Students' submissions are stored in a roster that indicates whether the submission is correct, how many attempts were made, and the actual code submissions from each attempt.

PEARSON eTEXT

The Pearson eText gives students access to their textbook anytime, anywhere

STEP-BY-STEP VIDEONOTE TUTORIALS

These step-by-step video tutorials enhance the programming concepts presented in select Pearson textbooks.

For more information and titles available with **MyProgrammingLab**, please visit **www.myprogramminglab.com**.

ALWAYS LEARNING

PEARSON

LOCATION OF VIDEO NOTES IN THE TEXT www.pearsonglobaleditions.com/reges	VideoNote
Chapter 1	Pages 57, 66
Chapter 2	Pages 91, 100, 115, 123, 136
Chapter 3	Pages 167, 182, 187, 193
Chapter 3G	Pages 223, 241
Chapter 4	Pages 269, 277, 304
Chapter 5	Pages 350, 353, 355, 359, 382
Chapter 6	Pages 422, 435, 449
Chapter 7	Pages 484, 491, 510, 531
Chapter 8	Pages 561, 573, 581, 594
Chapter 9	Pages 623, 636, 652
Chapter 10	Pages 698, 703, 712
Chapter 11	Pages 742, 755, 763
Chapter 12	Pages 790, 798, 835
Chapter 13	Pages 860, 863, 869
Chapter 14	Pages 915, 922
Chapter 15	Pages 956, 962, 966
Chapter 16	Pages 998, 1005, 1018
Chapter 17	Pages 1063, 1064, 1074
Chapter 18	Pages 1099, 1118

Brief Contents

This page intentionally left blank

Contents

Chapter 1

Introduction to Java Programming

Introduction

This chapter begins with a review of some basic terminology about computers and computer programming. Many of these concepts will come up in later chapters, so it will be useful to review them before we start delving into the details of how to program in Java.

We will begin our exploration of Java by looking at simple programs that produce output. This discussion will allow us to explore many elements that are common to all Java programs, while working with programs that are fairly simple in structure.

After we have reviewed the basic elements of Java programs, we will explore the technique of procedural decomposition by learning how to break up a Java program into several methods. Using this technique, we can break up complex tasks into smaller subtasks that are easier to manage and we can avoid redundancy in our program solutions.

1.1 Basic Computing Concepts

Computers are pervasive in our daily lives, and, thanks to the Internet, they give us access to nearly limitless information. Some of this information is essential news, like the headlines at cnn.com. Computers let us share photos with our families and map directions to the nearest pizza place for dinner.

Lots of real-world problems are being solved by computers, some of which don't much resemble the one on your desk or lap. Computers allow us to sequence the human genome and search for DNA patterns within it. Computers in recently manufactured cars monitor each vehicle's status and motion. Digital music players such as Apple's iPod actually have computers inside their small casings. Even the Roomba vacuum-cleaning robot houses a computer with complex instructions about how to dodge furniture while cleaning your floors.

But what makes a computer a computer? Is a calculator a computer? Is a human being with a paper and pencil a computer? The next several sections attempt to address this question while introducing some basic terminology that will help prepare you to study programming.

Why Programming?

At most universities, the first course in computer science is a programming course. Many computer scientists are bothered by this because it leaves people with the impression that computer science is programming. While it is true that many trained computer scientists spend time programming, there is a lot more to the discipline. So why do we study programming first?

A Stanford computer scientist named Don Knuth answers this question by saying that the common thread for most computer scientists is that we all in some way work with *algorithms*.

> **Algorithm**
> A step-by-step description of how to accomplish a task.

Knuth is an expert in algorithms, so he is naturally biased toward thinking of them as the center of computer science. Still, he claims that what is most important is not the algorithms themselves, but rather the thought process that computer scientists employ to develop them. According to Knuth,

> It has often been said that a person does not really understand something until after teaching it to someone else. Actually a person does not *really* understand something until after teaching it to a *computer*, i.e., expressing it as an algorithm.[1]

[1]Knuth, Don. *Selected Papers on Computer Science*. Stanford, CA: Center for the Study of Language and Information, 1996.

Knuth is describing a thought process that is common to most of computer science, which he refers to as *algorithmic thinking*. We study programming not because it is the most important aspect of computer science, but because it is the best way to explain the approach that computer scientists take to solving problems.

The concept of algorithms is helpful in understanding what a computer is and what computer science is all about. The Merriam-Webster dictionary defines the word “computer” as “one that computes.” Using that definition, all sorts of devices qualify as computers, including calculators, GPS navigation systems, and children’s toys like the Furby. Prior to the invention of electronic computers, it was common to refer to humans as computers. The nineteenth-century mathematician Charles Peirce, for example, was originally hired to work for the U.S. government as an “Assistant Computer” because his job involved performing mathematical computations.

In a broad sense, then, the word “computer” can be applied to many devices. But when computer scientists refer to a computer, we are usually thinking of a universal computation device that can be programmed to execute any algorithm. Computer science, then, is the study of computational devices and the study of computation itself, including algorithms.

Algorithms are expressed as computer programs, and that is what this book is all about. But before we look at how to program, it will be useful to review some basic concepts about computers.

Hardware and Software

A computer is a machine that manipulates data and executes lists of instructions known as *programs*.

> **Program**
>
> A list of instructions to be carried out by a computer.

One key feature that differentiates a computer from a simpler machine like a calculator is its versatility. The same computer can perform many different tasks (playing games, computing income taxes, connecting to other computers around the world), depending on what program it is running at a given moment. A computer can run not only the programs that exist on it currently, but also new programs that haven’t even been written yet.

The physical components that make up a computer are collectively called *hardware*. One of the most important pieces of hardware is the central processing unit, or *CPU*. The CPU is the “brain” of the computer: It is what executes the instructions. Also important is the computer’s *memory* (often called random access memory, or *RAM*, because the computer can access any part of that memory at any time). The computer uses its memory to store programs that are being executed, along with their data. RAM is limited in size and does not retain its contents when the computer is turned off. Therefore, computers generally also use a *hard disk* as a larger permanent storage area.

Computer programs are collectively called *software*. The primary piece of software running on a computer is its operating system. An *operating system* provides an environment in which many programs may be run at the same time; it also provides a bridge between those programs, the hardware, and the *user* (the person using the computer). The programs that run inside the operating system are often called *applications*.

When the user selects a program for the operating system to run (e.g., by double-clicking the program's icon on the desktop), several things happen: The instructions for that program are loaded into the computer's memory from the hard disk, the operating system allocates memory for that program to use, and the instructions to run the program are fed from memory to the CPU and executed sequentially.

The Digital Realm

In the last section, we saw that a computer is a general-purpose device that can be programmed. You will often hear people refer to modern computers as *digital* computers because of the way they operate.

Digital

Based on numbers that increase in discrete increments, such as the integers 0, 1, 2, 3, etc.

Because computers are digital, everything that is stored on a computer is stored as a sequence of integers. This includes every program and every piece of data. An MP3 file, for example, is simply a long sequence of integers that stores audio information. Today we're used to digital music, digital pictures, and digital movies, but in the 1940s, when the first computers were built, the idea of storing complex data in integer form was fairly unusual.

Not only are computers digital, storing all information as integers, but they are also *binary*, which means they store integers as *binary numbers*.

Binary Number

A number composed of just 0s and 1s, also known as a base-2 number.

Humans generally work with *decimal* or base-10 numbers, which match our physiology (10 fingers and 10 toes). However, when we were designing the first computers, we wanted systems that would be easy to create and very reliable. It turned out to be simpler to build these systems on top of binary phenomena (e.g., a circuit being open or closed) rather than having 10 different states that would have to be distinguished from one another (e.g., 10 different voltage levels).

From a mathematical point of view, you can store things just as easily using binary numbers as you can using base-10 numbers. But since it is easier to construct a physical device that uses binary numbers, that's what computers use.

This does mean, however, that people who aren't used to computers find their conventions unfamiliar. As a result, it is worth spending a little time reviewing how binary