7.1 Introduction

This chapter introduces **data structures**—collections of related data items. **Array** objects are data structures consisting of related data items of the same type. Arrays make it convenient to process related groups of values. Arrays remain the same length once they're created. We study data structures in depth in Chapters 16–21.

After discussing how arrays are declared, created and initialized, we present practical examples that demonstrate common array manipulations. We introduce Java's *exception-handling* mechanism and use it to allow a program to continue executing when it attempts to access an array element that does not exist. We also present a case study that examines how arrays can help simulate the shuffling and dealing of playing cards in a card-game application. We introduce Java's *enhanced* `for` *statement*, which allows a program to access the data in an array more easily than does the counter-controlled `for` statement presented in Section 5.3. We build two versions of an instructor `GradeBook` case study that use arrays to maintain sets of student grades *in memory* and analyze student grades. We show how to use *variable-length argument lists* to create methods that can be called with varying numbers of arguments, and we demonstrate how to process *command-line arguments* in method `main`. Next, we present some common array manipulations with `static`

methods of class `Arrays` from the `java.util` package.

Although commonly used, arrays have limited capabilities. For example, you must specify an array's size, and if at execution time you wish to modify it, you must do so by creating a new array. At the end of this chapter, we introduce one of Java's prebuilt data structures from the Java API's *collection classes*. These offer greater capabilities than traditional arrays. They're reusable, reliable, powerful and efficient. We focus on the `ArrayList` collection. `ArrayList`s are similar to arrays but provide additional functionality, such as **dynamic resizing** as necessary to accommodate more or fewer elements.

Java SE 8

8

After reading Chapter 17, Lambdas and Streams, you'll be able to reimplement many of Chapter 7's examples in a more concise and elegant manner, and in a way that makes them easier to parallelize to improve performance on today's multi-core systems.

Recall from the Preface that Chapter 17 is keyed to many earlier sections of the book so that you can conveniently cover lambdas and streams if you'd like. If so, we recommend that after you complete Chapter 7, you read Sections 17.1–17.7, which introduce the lambdas and streams concepts and use them to rework examples from Chapters 4–7.