Chapter 2 Data Structures Defined

The primary purpose of most computer programs is to store and retrieve data rather than to perform calculations. There are many different ways to organize data for storage and retrieval, and each type of organization is well suited to solving certain kinds of problems and poorly suited to solving problems of other kinds. Consequently, the way in which a program's data is organized may have a profound effect on the program's running time and memory requirements. A software designer must be educated about data organization, for one kind of organization may yield a program that requires days or weeks to execute, while another organization may yield a program that runs in only a few seconds. The study of data organization is called *data structures* and is the primary focus of this book.

2.1 Abstract Data Types and Data Structures

In the chapters to come we will examine the vector, set, heap, stack, queue, deque, binary tree, and binary search tree *abstract data types* and their applications.

abstract data type (ADT): a specification that describes the members of a data set and the operations that can be performed on those members; the inclusion of the term *abstract* means that an ADT does not mention implementation details; an ADT is *conceptual*, not *concrete*

An example ADT already familiar to you appears below.

```
integer: a whole number

operations:

    addition (+)
    subtraction (-)
    multiplication (*)
    division (/)
    modulus (%)
```

Notice that the integer abstract data type specifies neither how an integer value is to be represented nor how any of the operations is to be carried out. The ADT tells us *what*, not *how*. Computer integers may be represented using any of several schemes including sign-and-magnitude, one's complement, and two's complement. Each of these representation schemes is a *data structure*.

data structure: an implementation of an ADT; a data structure is concrete, i.e., a data structure makes an ADT a reality by specifying how to represent instances of the data type and how to perform operations on those instances

2.2 Collections and Containers

The ADTs that will concern us in later chapters–vector, set, heap, stack, queue, deque, binary tree, and binary search tree–are of a special kind. These ADTs are called *collections*.

collection: an ADT for a group of data items, rather than for the set to which the individual members of the group belong; the operations specified by a collection apply to the group and have no meaning at the level of the group's individual members

We will discuss one or more data structures for each of these collections. Each data structure will take the form of a *container*.

container: a class that implements a collection, i.e., a collection data structure in the form of a class

Java's `ArrayList` class, for example, is a container. Class `ArrayList` implements Java's list abstract data type. Java's list ADT specifies operations like `add` and `get`, which add an element to a list or retrieve an element, respectively. Per the definition of collection, these operations are not applied on or between elements of a list, but on the list itself. As its name implies, an `ArrayList` stores its collection of data items in an array. An instance of the container uses its array to carry out the many operations of the list ADT. Some of the containers presented in subsequent chapters will also use arrays, and some will use alternative means of data organization.

2.3 Our Destination: The Final Software Model

The UML class diagram on the next page shows the software design at which we will arrive in Chapter 14. The diagram is incomplete in that it shows neither attributes nor operations, but it does show all of the classifiers that will be relevant to us. We will create the model incrementally; each chapter will show the model up to that point, and one of the chapter exercises will be to add attributes and operations to the current diagram. Adding attributes and operations to the diagram will help you to better understand the unfolding design.

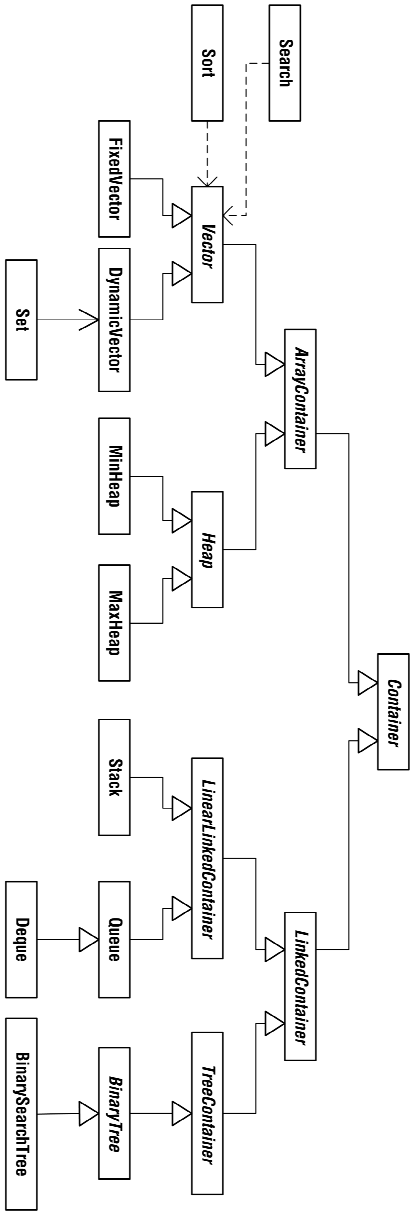

Fig. 2.1. Our final software model sans attributes and operations

Exercises

1. Compare and contrast the one's complement and two's complement data structures for the integer ADT. Why is two's complement more commonly used?
2. Use the Java SDK documentation to create a UML class diagram with class `ArrayList` as your focus of attention. Show only realizations and generalizations in your diagram. Is class `ArrayList` a data structure for any Java ADTs besides list?