16.5 Interface `Collection` and Class `Collections`

Interface `Collection` contains **bulk operations** (i.e., operations performed on an *entire* collection) for operations such as *adding*, *clearing* and *comparing* objects (or elements) in a collection. A `Collection` can also be converted to an array. In addition, interface `Collection` provides a method that returns an `Iterator` object, which allows a program to walk through the collection and remove elements from it during the iteration. We discuss class `Iterator` in Section 16.6.1. Other methods of interface `Collection` enable a program to determine a collection's *size* and whether a collection is *empty*.

Software Engineering Observation 16.1

`Collection` *is used commonly as a parameter type in methods to allow polymorphic processing of all objects that implement interface* `Collection`.

 Software Engineering Observation 16.2

Most collection implementations provide a constructor that takes a `Collection` *argument, thereby allowing a new collection to be constructed containing the elements of the specified collection.*

Class `Collections` provides `static` convenience methods that *search*, *sort* and perform other operations on collections. Section 16.7 discusses `Collections` methods in detail. We also cover `Collections`' **wrapper methods** that enable you to treat a collection as a *synchronized collection* (Section 16.11) or an *unmodifiable collection* (Section 16.12). Synchronized collections are for use with multithreading (discussed in Chapter 23), which enables programs to perform operations *in parallel*. When two or more threads of a program *share* a collection, problems might occur. As an analogy, consider a traffic intersection. If all cars were allowed to access the intersection at the same time, collisions might occur. For this reason, traffic lights are provided to control access to the intersection. Similarly, we can *synchronize* access to a collection to ensure that only *one* thread manipulates the collection at a time. The synchronization wrapper methods of class `Collections` return synchronized versions of collections that can be shared among threads in a program. Chapter 23 also discusses some classes from the `java.util.concurrent` package, which provides more robust collections for use in multithreaded

applications. Unmodifiable collections are useful when clients of a class need to *view* a collection's elements, but they should *not* be allowed to *modify* the collection by adding and removing elements.