

The Java Tutorials have been written for JDK 8. Examples and practices described in this page don't take advantage of improvements introduced in later releases and might use technology no longer available.
See [JDK Release Notes](#) for information about new features, enhancements, and removed or deprecated options for all JDK releases.

Lesson: Introduction to Collections

A *collection* — sometimes called a container — is simply an object that groups multiple elements into a single unit. Collections are used to store, retrieve, manipulate, and communicate aggregate data. Typically, they represent data items that form a natural group, such as a poker hand (a collection of cards), a mail folder (a collection of letters), or a telephone directory (a mapping of names to phone numbers). If you have used the Java programming language — or just about any other programming language — you are already familiar with collections.

What Is a Collections Framework?

A *collections framework* is a unified architecture for representing and manipulating collections. All collections frameworks contain the following:

- **Interfaces:** These are abstract data types that represent collections. Interfaces allow collections to be manipulated independently of the details of their representation. In object-oriented languages, interfaces generally form a hierarchy.
- **Implementations:** These are the concrete implementations of the collection interfaces. In essence, they are reusable data structures.
- **Algorithms:** These are the methods that perform useful computations, such as searching and sorting, on objects that implement collection interfaces. The algorithms are said to be *polymorphic*: that is, the same method can be used on many different implementations of the appropriate collection interface. In essence, algorithms are reusable functionality.

Apart from the Java Collections Framework, the best-known examples of collections frameworks are the C++ Standard Template Library (STL) and Smalltalk's collection hierarchy. Historically, collections frameworks have been quite complex, which gave them a reputation for having a steep learning curve. We believe that the Java Collections Framework breaks with this tradition, as you will learn for yourself in this chapter.

Benefits of the Java Collections Framework

The Java Collections Framework provides the following benefits:

- **Reduces programming effort:** By providing useful data structures and algorithms, the Collections Framework frees you to concentrate on the important parts of your program rather than on the low-level "plumbing" required to make it work. By facilitating interoperability among unrelated APIs, the Java Collections Framework frees you from writing adapter objects or conversion code to connect APIs.
- **Increases program speed and quality:** This Collections Framework provides high-performance, high-quality implementations of useful data structures and algorithms. The various implementations of each interface are interchangeable, so programs can be easily tuned by switching collection implementations. Because you're freed from the drudgery of writing your own data structures, you'll have more time to devote to improving programs' quality and performance.
- **Allows interoperability among unrelated APIs:** The collection interfaces are the vernacular by which APIs pass collections back and forth. If my network administration API furnishes a collection of node names and if your GUI toolkit expects a collection of column headings, our APIs will interoperate seamlessly, even though they were written independently.
- **Reduces effort to learn and to use new APIs:** Many APIs naturally take collections on input and furnish them as output. In the past, each such API had a small sub-API devoted to manipulating its collections. There was little consistency among these ad hoc collections sub-APIs, so you had to learn each one from scratch, and it was easy to make mistakes when using them. With the advent of standard collection interfaces, the problem went away.
- **Reduces effort to design new APIs:** This is the flip side of the previous advantage. Designers and implementers don't have to reinvent the wheel each time they create an API that relies on collections; instead, they can use standard collection interfaces.
- **Fosters software reuse:** New data structures that conform to the standard collection interfaces are by nature reusable. The same goes for new algorithms that operate on objects that implement these interfaces.