16.11 Synchronized Collections

In Chapter 23, we discuss *multithreading*. The collections in the collections framework are *unsynchronized* by default, so they can operate efficiently when multithreading is not required. Because they're unsynchronized, however, concurrent access to a `Collection` by multiple threads could cause indeterminate results or fatal errors—as we demonstrate in Chapter 23. To prevent potential threading problems, **synchronization wrappers** are provided for collections that might be accessed by multiple threads. A **wrapper** object receives method calls, adds thread synchronization (to prevent concurrent access to the collection) and *delegates* the calls to the wrapped collection object. The `Collections` class provides `static` methods for wrapping collections as synchronized versions. Method headers for some synchronization wrappers are listed in Fig. 16.18. Details on these methods are available at `http://docs.oracle.com/javase/8/docs/api/java/util/C` Each method takes a collection and returns its *synchronized view*. For example, the following code creates a synchronized `List` (`list2`) that stores `String` objects:

```
List<String> list1 = new ArrayList<>();
List<String> list2 = Collections.synchronizedList(lis
```

More robust collections for concurrent access are provided in the `java.util.concurrent` package, which we introduce in Chapter 23.

`public static` **method headers**
`<T> Collection<T> synchronizedCollection(Collection<T> c)`
`<T> List<T> synchronizedList(List<T> aList)`
`<T> Set<T> synchronizedSet(Set<T> s)`
`<T> SortedSet<T> synchronizedSortedSet(SortedSet<T> s)`
`<K, V> Map<K, V> synchronizedMap(Map<K, V> m)`
`<K, V> SortedMap<K, V> synchronizedSortedMap(SortedMap<K, V> m)`

Fig. 16.18

Some synchronization wrapper methods.