TABLE 7.2 `ExecutorService` methods

Method Name	Description
`void execute(Runnable command)`	Executes a `Runnable` task at some point in the future
`Future<?> submit(Runnable task)`	Executes a `Runnable` task at some point in the future and returns a `Future` representing the task
`<T> Future<T> submit(Callable<T> task)`	Executes a `Callable` task at some point in the future and returns a `Future` representing the pending results of the task
`<T> List<Future<T>> invokeAll( Collection<? extends Callable<T>> tasks) throws InterruptedException`	Executes the given tasks, synchronously returning the results of all tasks as a `Collection` of `Future` objects, in the same order they were in the original collection
`<T> T invokeAny( Collection<? extends Callable<T>> tasks) throws InterruptedException, ExecutionException`	Executes the given tasks, synchronously returning the result of one of finished tasks, cancelling any unfinished tasks

TABLE 7.3 `Future` methods

Method Name	Description
`boolean isDone()`	Returns `true` if the task was completed, threw an exception, or was cancelled.
`boolean isCancelled()`	Returns `true` if the task was cancelled before it completely normally.
`boolean cancel()`	Attempts to cancel execution of the task.
`V get()`	Retrieves the result of a task, waiting endlessly if it is not yet available.
`V get(long timeout, TimeUnit unit)`	Retrieves the result of a task, waiting the specified amount of time. If the result is not ready by the time the timeout is reached, a checked `TimeoutException` will be thrown.

TABLE 7.5 ScheduledExecutorService methods

Method Name	Description
schedule(Callable<V> callable, long delay, TimeUnit unit)	Creates and executes a Callable task after the given delay
schedule(Runnable command, long delay, TimeUnit unit)	Creates and executes a Runnable task after the given delay
scheduleAtFixedRate(Runnable command, long initialDelay, long period, TimeUnit unit)	Creates and executes a Runnable task after the given initial delay, creating a new task every period value that passes.
scheduleAtFixedDelay(Runnable command, long initialDelay, long delay, TimeUnit unit)	Creates and executes a Runnable task after the given initial delay and subsequently with the given delay between the termination of one execution and the commencement of the next

TABLE 7.7 Atomic classes

Class Name	Description
AtomicBoolean	A boolean value that may be updated atomically
AtomicInteger	An int value that may be updated atomically
AtomicIntegerArray	An int array in which elements may be updated atomically
AtomicLong	A long value that may be updated atomically
AtomicLongArray	A long array in which elements may be updated atomically
AtomicReference	A generic object reference that may be updated atomically
AtomicReferenceArray	An array of generic object references in which elements may be updated atomically

TABLE 7.8 Common atomic methods

Class Name	Description
`get()`	Retrieve the current value
`set()`	Set the given value, equivalent to the assignment = operator
`getAndSet()`	Atomically sets the new value and returns the old value
`incrementAndGet()`	For numeric classes, atomic pre-increment operation equivalent to `++value`
`getAndIncrement()`	For numeric classes, atomic post-increment operation equivalent to `value++`
`decrementAndGet()`	For numeric classes, atomic pre-decrement operation equivalent to `--value`
`getAndDecrement()`	For numeric classes, atomic post-decrement operation equivalent to `value--`

TABLE 7.9 Concurrent collection classes

Class Name	Java Collections Framework Interface	Elements Ordered?	Sorted?	Blocking?
`ConcurrentHashMap`	`ConcurrentMap`	No	No	No
`ConcurrentLinkedDeque`	`Deque`	Yes	No	No
`ConcurrentLinkedQueue`	`Queue`	Yes	No	No
`ConcurrentSkipListMap`	`ConcurrentMap` `SortedMap` `NavigableMap`	Yes	Yes	No
`ConcurrentSkipListSet`	`SortedSet` `NavigableSet`	Yes	Yes	No
`CopyOnWriteArrayList`	`List`	Yes	No	No
`CopyOnWriteArraySet`	`Set`	No	No	No
`LinkedBlockingDeque`	`BlockingQueue` `BlockingDeque`	Yes	No	Yes
`LinkedBlockingQueue`	`BlockingQueue`	Yes	No	Yes

TABLE 7.10 `BlockingQueue` waiting methods

Method Name	Description
`offer(E e, long timeout, TimeUnit unit)`	Adds item to the queue waiting the specified time, returning `false` if time elapses before space is available

TABLE 7.10 `BlockingQueue` waiting methods *(continued)*

Method Name	Description
`poll(long timeout, TimeUnit unit)`	Retrieves and removes an item from the queue, waiting the specified time, returning `null` if the time elapses before the item is available

TABLE 7.11 `BlockingDeque` waiting methods

Method Name	Description
`offerFirst(E e, long timeout, TimeUnit unit)`	Adds an item to the front of the queue, waiting a specified time, returning `false` if time elapses before space is available
`offerLast(E e, long timeout, TimeUnit unit)`	Adds an item to the tail of the queue, waiting a specified time, returning `false` if time elapses before space is available
`pollFirst(long timeout, TimeUnit unit)`	Retrieves and removes an item from the front of the queue, waiting the specified time, returning `null` if the time elapses before the item is available
`pollLast(long timeout, TimeUnit unit)`	Retrieves and removes an item from the tail of the queue, waiting the specified time, returning `null` if the time elapses before the item is available

TABLE 7.12 Synchronized collections methods

Method Name
`synchronizedCollection(Collection<T> c)`
`synchronizedList(List<T> list)`
`synchronizedMap(Map<K,V> m)`
`synchronizedNavigableMap(NavigableMap<K,V> m)`
`synchronizedNavigableSet(NavigableSet<T> s)`
`synchronizedSet(Set<T> s)`
`synchronizedSortedMap(SortedMap<K,V> m)`
`synchronizedSortedSet(SortedSet<T> s)`