compact1, compact2, compact3
java.util.concurrent

Interface Executor

All Known Subinterfaces:

ExecutorService, ScheduledExecutorService

All Known Implementing Classes:

AbstractExecutorService, ForkJoinPool, ScheduledThreadPoolExecutor, ThreadPoolExecutor

---

```
public interface Executor
```

An object that executes submitted Runnable tasks. This interface provides a way of decoupling task submission from the mechanics of how each task will be run, including details of thread use, scheduling, etc. An Executor is normally used instead of explicitly creating threads. For example, rather than invoking new Thread(new(RunnableTask())).start() for each of a set of tasks, you might use:

```
Executor executor = anExecutor;
executor.execute(new RunnableTask1());
executor.execute(new RunnableTask2());
...
```

However, the Executor interface does not strictly require that execution be asynchronous. In the simplest case, an executor can run the submitted task immediately in the caller's thread:

```
class DirectExecutor implements Executor {
  public void execute(Runnable r) {
    r.run();
  }
}
```

More typically, tasks are executed in some thread other than the caller's thread. The executor below spawns a new thread for each task.

```
class ThreadPerTaskExecutor implements Executor {
  public void execute(Runnable r) {
    new Thread(r).start();
  }
}
```

Many `Executor` implementations impose some sort of limitation on how and when tasks are scheduled. The executor below serializes the submission of tasks to a second executor, illustrating a composite executor.

```
 class SerialExecutor implements Executor {
   final Queue<Runnable> tasks = new ArrayDeque<Runnable>();
   final Executor executor;
   Runnable active;

   SerialExecutor(Executor executor) {
     this.executor = executor;
   }

   public synchronized void execute(final Runnable r) {
     tasks.offer(new Runnable() {
       public void run() {
         try {
           r.run();
         } finally {
           scheduleNext();
         }
       }
     });
     if (active == null) {
       scheduleNext();
     }
   }

   protected synchronized void scheduleNext() {
     if ((active = tasks.poll()) != null) {
       executor.execute(active);
     }
   }
 }
```

The `Executor` implementations provided in this package implement `ExecutorService`, which is a more extensive interface. The `ThreadPoolExecutor` class provides an extensible thread pool implementation. The `Executors` class provides convenient factory methods for these Executors.

Memory consistency effects: Actions in a thread prior to submitting a `Runnable` object to an `Executor` *happen-before* its execution begins, perhaps in another thread.

Since:

1.5

Method Summary

All Methods Instance Methods Abstract Methods

Modifier and Type	Method and Description
void	**execute**(**Runnable** command) Executes the given command at some time in the future.

Method Detail

execute

```
void execute(Runnable command)
```

Executes the given command at some time in the future. The command may execute in a new thread, in a pooled thread, or in the calling thread, at the discretion of the `Executor` implementation.

Parameters:

`command` - the runnable task

Throws:

`RejectedExecutionException` - if this task cannot be accepted for execution

`NullPointerException` - if command is null