Java Executors

Thread and Runnable

- java.lang.Thread and java.lang.Runnable are low-level
- There is a close connection
  - The task being done by a thread (defined by a Runnable)
  - The thread (defined by a Thread object)
- What is the return type for run()?
- OK for small applications, but …

Executors

- Sometimes you want first-class control
  - Separate thread management and creation from the rest of the application
- What if you're running an application with 10,000+ concurrent objects? Most machines cannot allocate that many threads.

Executor Interfaces

The `java.util.concurrent` package has:

- `Executor` - base interface
- `ExecutorService` - subinterface of `Executor`
  - adds features that help manage tasks and their `Executor`
- `ScheduledExecutorService` - subinterface of `ExecutorService`
  - Supports future and/or periodic execution of tasks
- Executors has methods for creating instances of Executor and ExecutorService

Executor

```
Runnable r = ...
//Run r in a thread
(new Thread(r)).start();

//Use an executor to run r
Executor e = ...
e.execute(r);
```

- An Executor is similar to a Thread
- An Executor
  - May use an existing thread to run r
  - May put r in a queue and wait for an available thread

ExecutorService

- `ExecutorService` also have a `submit()` method
- submit() can accept `Callable` objects and returns a `Future` value, which can be used to receive the result of the `Callable` object, and manage the status of `Callable` and `Runnable` objects

General Rule

- If you see

```
Runnable r = ...
//Run r in a thread
(new Thread(r)).start();
```

- but you want more control

```
//Use an executor to run r
Executor e = Executors.<see the Javadoc>
e.execute(r);
```