Executor Framework

This lesson discusses thread management using executors.

Creating and running individual threads for small applications is acceptable however if you are writing an enterprise-grade application with several dozen threads then you'll likely need to offload thread management in your application to library classes which free a developer from worrying about thread house-keeping.

Task

A task is a logical unit of work. Usually, a task should be independent of other tasks so that it can be completed by a single thread. A task can be represented by an object of a class implementing the `Runnable` interface. We can consider HTTP requests being fielded by a web-server as tasks that need to be processed. A database server handling client queries can similarly be thought of as independent tasks.

Executor Framework

In Java, the primary abstraction for executing logical tasks units is the Executor framework and not the Thread class. The classes in the Executor framework separate out:

- Task Submission
- Task Execution

The framework allows us to specify different policies for task execution. Java offers three interfaces, which classes can implement to manage thread lifecycle. These are:

- `Executor Interface`
- `ExecutorService`
- `ScheduledExecutorService`

The `Executor` interface forms the basis for the asynchronous task execution framework in Java.

You don't need to create your own executor class as Java's `java.util.concurrent` package offers several types of executors that are suitable for different scenarios. However, as an example, we create a dumb executor which implements the Executor Interface.

```
import java.util.concurrent.Executor;
class ThreadExecutorExample {

  public static void main( String args[] ) {
     DumbExecutor myExecutor = new DumbExecutor();
     MyTask myTask = new MyTask();
     myExecutor.execute(myTask);
  }

  static class DumbExecutor implements Executor {
    // Takes in a runnable interface object
    public void execute(Runnable runnable) {
      Thread newThread = new Thread(runnable);
      newThread.start();
    }
  }

  static class MyTask implements Runnable {
    public void run() {
      System.out.println("Mytask is running now ...");
    }
  }

}
```

> The `Executor` requires implementing classes to define a method `execute(Runnable runnable)` which takes in an object of interface `Runnable`. Fortunately, we don't need to define complex executors as Java already provides several that we'll explore in following chapters.