Java Executor Implementation Choices

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science
Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA
Learning Objectives in this Part of the Lesson

- Recognize the single simple feature provided by the Java Executor interface
- Understand various implementation choices for the Executor interface
Implementation Choices for the Java Executor Interface
Overview of the Java Executor Interface

- The Executor interface can be implemented via different types of thread pooling mechanisms.

**Fixed-sized Thread Pool**

**Cached Thread Pool**

**Work-stealing Thread Pool**

**A Custom Thread Pool**

<<Java Interface>>

1. Executor

   - execute(Runnable): void
Overview of the Java Executor Interface

- Executor configuration is often performed just once to select the “execution policy” for tasks passed to it.

<<Java Interface>>

```
<Executor>
  execute(Runnable):void
```

Fixed-sized Thread Pool

Cached Thread Pool

Work-stealing Thread Pool

A Custom Thread Pool
Overview of the Java Executor Interface

- The “execution policy” for a group of tasks defines several properties
Overview of the Java Executor Interface

- The “execution policy” for a group of tasks defines several properties, e.g.
  - In which thread will a task be executed
    - e.g., a existing thread in the pool, a new thread created/added to the pool, etc.

There’s even a single threaded implementation of Executor!
The “execution policy” for a group of tasks defines several properties, e.g.

- In which thread will a task be executed
- In which order will tasks be executed
  - e.g., FIFO, LIFO, priority order, etc.
Overview of the Java Executor Interface

• The “execution policy” for a group of tasks defines several properties, e.g.
  • In which thread will a task be executed
  • In which order will tasks be executed
  • How many tasks can run concurrently
    • e.g., is the maximum # of tasks limited by the # of CPU cores or by some other factor?
The “execution policy” for a group of tasks defines several properties, e.g.
- In which thread will a task be executed
- In which order will tasks be executed
- How many tasks can run concurrently
- If not all tasks can be executed due to system overload which task(s) should be rejected & how should an app be notified
  - e.g., should execute() fail silently vs. throw RejectedExecutionException

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/RejectedExecutionException.html
The “execution policy” for a group of tasks defines several properties, e.g.

- In which thread will a task be executed
- In which order will tasks be executed
- How many tasks can run concurrently
- If not all tasks can be executed due to system overload which task(s) should be rejected & how should an app be notified
- What actions (if any) should be performed before and/or after executing a task
  - e.g., Android AsyncTask’s onPreExecute() & onPostExecute() hook methods

End of Java Executor Implementation Choices