Java Executor:
Evaluating Pros & Cons

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 simple/single feature provided by the Java Executor interface
- Understand various implementation choices for the Executor interface
- Learn how to program a simple prime checker app using the Java Executor interface
- Evaluate the pros & cons of the prime checker app
Evaluating the PrimeChecker App
Evaluating the PrimeChecker App

- The Java Executor interface enables the transparent tuning & replacement of # & type of threads wrt the prime checker app logic itself

```java
new Random().longs(count, sMAX_VALUE - count, sMAX_VALUE)
  .forEach(randomNumber -> mExecutor.execute
    (new PrimeRunnable(this, randomNumber)));
```

- Fixed-sized Thread Pool
- Cached (Variable-sized) Thread Pool
- Work-stealing Thread Pool
Evaluating the PrimeChecker App

• However, Java Executor has some restrictions
Evaluating the PrimeChecker App

- However, Java Executor has some restrictions, e.g.
- One-way semantics of runnables tightly couple PrimeRunnable with MainActivity

```java
class PrimeRunnable implements Runnable {
    ...
    private final MainActivity mActivity;
    ...
    public PrimeRunnable(MainActivity activity) {
        mActivity = activity; ...
    }

    public void run() {
        ...
        mActivity.done(); ...
    }
}
```

This tight coupling complicates runtime configuration changes
Evaluating the PrimeChecker App

- However, Java Executor has some restrictions, e.g.
  - One-way semantics of runnables tightly couple PrimeRunnable with MainActivity
  - isPrime() tightly coupled w/PrimeRunnable

```java
class PrimeRunnable implements Runnable {
  ...
  long isPrime(long n) {
    if (n > 3)
      for (long factor = 2;
          factor <= n / 2; ++factor)
        if (n / factor * factor == n)
          return factor;
    return 0;
  }
  ...
```

- e.g., non-extensible & primality check is applied even if results are computed
Evaluating the PrimeChecker App

- However, Java Executor has some restrictions, e.g.
  - One-way semantics of runnables tightly couple `PrimeRunnable` with `MainActivity`
  - `isPrime()` tightly coupled w/`PrimeRunnable`
  - The lack of lifecycle operations on Java Executor
- However, Java Executor has some restrictions, e.g.
  - One-way semantics of runnables tightly couple PrimeRunnable with MainActivity
  - isPrime() tightly coupled with PrimeRunnable
  - The lack of lifecycle operations on Java Executor, e.g.
    - Can’t shutdown the executor or interrupt/cancel running tasks
Evaluating the PrimeChecker App

• However, Java Executor has some restrictions, e.g.
  • One-way semantics of runnables tightly couple PrimeRunnable with MainActivity
  • isPrime() tightly coupled w/PrimeRunnable
  • The lack of lifecycle operations on Java Executor, e.g.
    • Can’t shutdown the executor or interrupt/cancel running tasks
    • Can’t handle runtime configuration changes gracefully
      • e.g., must restart processing from the beginning
Evaluating the PrimeChecker App

- However, Java Executor has some restrictions, e.g.
  - One-way semantics of runnables tightly couple PrimeRunnable with MainActivity
  - isPrime() tightly coupled w/PrimeRunnable
  - The lack of lifecycle operations on Java Executor, e.g.
    - Can’t shutdown the executor or interrupt/cancel running tasks
    - Can’t handle runtime configuration changes gracefully
  - The Java Executor is often too simple for its own good!
End of Java Executor: Evaluating Pros & Cons