Applying the Java ExecutorService to the PrimeChecker App

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 powerful features defined in the Java ExecutorService interface
- Understand other interfaces related to ExecutorService
- Know the key methods provided by ExecutorService
- Be aware of how ThreadPoolExecutor implements ExecutorService

- Learn how to program the PrimeChecker app using an ExecutorService implementation
  - Including a fixed-sized thread pool, a list of Future objects, and a background thread
Overview of the PrimeChecker App
Overview of the PrimeChecker App

- This “embarrassingly parallel” & compute-bound app uses the Java ExecutorService to check if $N$ random #’s are prime

See github.com/douglascraigschmidt/POSA/tree/master/ex/M4/Primes/PrimeExecutorService
Overview of the PrimeChecker App

- This “embarrassingly parallel” & compute-bound app uses the Java ExecutorService to check if $N$ random #'s are prime
- It also shows how to handle runtime configuration changes in Android

See developer.android.com/guide/topics/resources/runtime-changes.html
Overview of the PrimeChecker App

- This “embarrassingly parallel” & compute-bound app uses the Java ExecutorService to check if $N$ random #’s are prime
- It also shows how to handle runtime configuration changes in Android
- As well as thread interruptions

See docs.oracle.com/javase/tutorial/essential/concurrency/interrupt.html
A fixed-size thread pool is tuned to # of processor cores in the computing device

```java
mExecutor = Executors.newFixedThreadPool(Runtime.getRuntime().availableProcessors());
```
Overview of the PrimeChecker App

- A fixed-size thread pool is tuned to # of processor cores in the computing device

```java
mExecutor = Executors
    .newFixedThreadPool
    (Runtime.getRuntime().
    availableProcessors());
```

The UI thread generates random #’s that are processed via the thread pool
A fixed-size thread pool is tuned to # of processor cores in the computing device

```java
mExecutor = Executors.newFixedThreadPool
    (Runtime.getRuntime().availableProcessors());
```

This fixed-size thread pool uses an unbounded queue to avoid deadlocks

See [aszajder.github.io/thread-pool-induced-deadlocks](aszajder.github.io/thread-pool-induced-deadlocks)
Overview of the PrimeChecker App

- A fixed-size thread pool is tuned to # of processor cores in the computing device

```java
mExecutor = Executors.newFixedThreadPool
    (Runtime.getRuntime().availableProcessors());

...mThread = new Thread(...);
...mThread.start();
```

Start a 2nd thread to wait for the completion of all futures in the list of futures

This background thread ensures no blocking occurs in the UI thread
Overview of the PrimeChecker App

- PrimeCallable defines a two-way means of determining whether a # is prime

```java
class PrimeCallable
    implements Callable<PrimeResult> {
    long mPrimeCandidate;
    ... 

    PrimeCallable(Long primeCandidate)
    { mPrimeCandidate = primeCandidate; }

    PrimeResult call() {
        return new PrimeResult
            (mPrimeCandidate,
             isPrime(mPrimeCandidate));
    } ...
```

See `src/main/java/vandy/mooc/prime/activities/PrimeCallable.java`
Overview of the PrimeChecker App

PrimeCallable defines a two-way means of determining whether a # is prime

class PrimeCallable
    implements Callable<PrimeResult> {
        long mPrimeCandidate;
        ...

        PrimeCallable(Long primeCandidate)
        { mPrimeCandidate = primeCandidate; }

        PrimeResult call() {
            return new PrimeResult
                (mPrimeCandidate,
                isPrime(mPrimeCandidate));
        }
    ...

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/Callable.html
Overview of the PrimeChecker App

• PrimeCallable defines a two-way means of determining whether a # is prime

```java
class PrimeCallable
    implements Callable<PrimeResult> {
    long mPrimeCandidate;
    ...

    PrimeCallable(Long primeCandidate)
    { mPrimeCandidate = primeCandidate; }

    PrimeResult call() {
        return new PrimeResult
            (mPrimeCandidate,
             isPrime(mPrimeCandidate));
    } ...
```
Overview of the PrimeChecker App

- PrimeCallable defines a two-way means of determining whether a # is prime

```java
class PrimeCallable
    implements Callable<PrimeResult> {
    long mPrimeCandidate;
    ...

    PrimeCallable(Long primeCandidate)
    { mPrimeCandidate = primeCandidate; }

    PrimeResult call() {
        return new PrimeResult
            (mPrimeCandidate,
             isPrime(mPrimeCandidate));
    } ...
```

Interruptible isPrime() based on "Java Executor: Application to PrimeChecker App"
PrimeCallable defines a two-way means of determining whether a # is prime.

```java
class PrimeCallable
    implements Callable<PrimeResult> {
    long mPrimeCandidate;
    ...
    
    PrimeCallable(Long primeCandidate)
    { mPrimeCandidate = primeCandidate; }
    
    PrimeResult call() {
        return new PrimeResult
        (mPrimeCandidate,
         isPrime(mPrimeCandidate));
    } ...
}
```

**PrimeResult** is a tuple that matches the prime # candidate with the result of checking primality.

These two-way semantics eliminate the need for a dependency on MainActivity!
PrimeCallable defines a two-way means of determining whether a # is prime

class PrimeCallable
    implements Callable<PrimeResult> {
    long mPrimeCandidate;
    ...

    PrimeCallable(Long primeCandidate)
    { mPrimeCandidate = primeCandidate; }

    PrimeResult call() {
        return new PrimeResult
            (mPrimeCandidate,
             isPrime(mPrimeCandidate));
    }

These two-way call semantics eliminate the need for any dependency on MainActivity!
PrimeCallable defines a two-way means of determining whether a # is prime

```java
class PrimeCallable implements Callable<PrimeResult> {
    ...

    long isPrime(long n) {
        if (n > 3)
            for (long factor = 2; factor <= n / 2; ++factor)
                if (Thread.interrupted()) break;
        else if (n / factor * factor == n)
            return factor;
        return 0L;
    }
    ...
}
```

Overview of the PrimeChecker App

Returns 0 if n is prime or smallest factor if it’s not
PrimeCallable defines a two-way means of determining whether a number is prime.

```java
class PrimeCallable
    implements Callable<PrimeResult> {
...

    long isPrime(long n) {
        if (n > 3)
            for (long factor = 2;
                factor <= n / 2; ++factor)
                if (Thread.interrupted()) break;
            else if (n / factor * factor == n)
                return factor;
        return 0L;
    }
...
```

Overview of the PrimeChecker App

- `isPrime()` repeatedly checks to see if it's been interrupted.

See lesson on "Managing the Java Thread Lifecycle"
MainActivity creates a list of futures that store results of concurrently checking primality of “count” random #'s within a range

```
List<Future<PrimeResult>>
futures = ...
```

This list of futures is initialized via a Java sequential stream

See `src/main/java/vandy/mooc/prime/activities/MainActivity.java`
Overview of the PrimeChecker App

- `MainActivity` creates a list of futures that store results of concurrently checking primality of "count" random #’s within a range.

```java
List<Future<PrimeResult>>
futures = new Random()
    .longs(count,
        sMAX_VALUE - count,
        sMAX_VALUE)
```

Generates "count" random #’s ranging from sMAX_VALUE - count to sMAX_VALUE.

The goal of this range is to generate a fair number of duplicates!
Overview of the PrimeChecker App

- MainActivity creates a list of futures that store results of concurrently checking primality of "count" random #’s within a range

```java
List<Future<PrimeResult>>
futures = new Random()
    .longs(count, 
        sMAX_VALUE - count, 
        sMAX_VALUE)
    .mapToObj(PrimeCallable::new)
```

See docs.oracle.com/javase/tutorial/java/javaOO/methodreferences.html
Overview of the PrimeChecker App

• MainActivity creates a list of futures that store results of concurrently checking primality of “count” random #’s within a range

List<Future<PrimeResult>>

futures = new Random()

    .longs(count, sMAX_VALUE - count, sMAX_VALUE)

    .mapToObj(PrimeCallable::new)

    .map(mRetainedState.mExecutorService::submit)

Submit a two-way task for execution & return a future representing pending task results

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/ExecutorService.html#submit
MainActivity creates a list of futures that store results of concurrently checking primality of "count" random #s within a range.

List<Future<PrimeResult>> futures = new Random().longs(count, sMAX_VALUE - count).mapToObj(PrimeCallable::new).map(mRetainedState.mExecutorService::submit).collect(toList());
Overview of the PrimeChecker App

- FutureRunnable runs in a background thread & gets the results of all futures as they complete.

```java
class FutureRunnable implements Runnable {
    List<Future<PrimeResult>> mFutures;

    MainActivity mActivity;

    FutureRunnable(MainActivity a, List<Future<PrimeResult>> f) {
        mActivity = a; mFutures = f;
    }
}
```
Overview of the PrimeChecker App

- FutureRunnable runs in a background thread & gets the results of all futures as they complete

```java
class FutureRunnable implements Runnable {
    List<Future<PrimeResult>> mFutures;

    MainActivity mActivity;

    FutureRunnable(MainActivity a, List<Future<PrimeResult>> f) {
        mActivity = a; mFutures = f;
    }
}
```

See [docs.oracle.com/javase/8/docs/api/java/lang/Runnable.html](docs.oracle.com/javase/8/docs/api/java/lang/Runnable.html)
Overview of the PrimeChecker App

- FutureRunnable runs in a background thread & gets the results of all futures as they complete

```java
class FutureRunnable implements Runnable {
    List<Future<PrimeResult>> mFutures;
    MainActivity mActivity;

    FutureRunnable(MainActivity a, List<Future<PrimeResult>> f) {
        mActivity = a; mFutures = f; }
}
```
Overview of the PrimeChecker App

- FutureRunnable runs in a background thread & gets the results of all futures as they complete

```java
class FutureRunnable implements Runnable {
    List<Future<PrimeResult>> mFutures;
    MainActivity mActivity;

    FutureRunnable(MainActivity a, List<Future<PrimeResult>> f) {
        mActivity = a; mFutures = f;
    }
}
```

Reference back to enclosing activity
Overview of the PrimeChecker App

- FutureRunnable runs in a background thread & gets the results of all futures as they complete

```java
class FutureRunnable implements Runnable {
    List<Future<PrimeResult>> mFutures;

    MainActivity mMainActivity;

    FutureRunnable(MainActivity a, List<Future<PrimeResult>> f) {
        mMainActivity = a; mFutures = f;
    }
}
```

Constructor initializes the fields
FutureRunnable runs in a background thread & gets the results of all futures as they complete ...

```java
public void run() {
    mFutures.forEach(future -> {
        PrimeCallable.PrimeResult pr =
            rethrowSupplier(future::get).get();

        if (pr.mSmallestFactor != 0)
            ...
        else ...
    });

    mAActivity.done(); ...
```
Overview of the PrimeChecker App

• FutureRunnable runs in a background thread & gets the results of all futures as they complete...

```java
public void run() {
    mFutures.forEach(future -> {
        PrimeCallable.PrimeResult pr =
            rethrowSupplier(future::get).get();

        if (pr.mSmallestFactor != 0)
            ...
        else ...
    });

    mActivity.done(); ...
}
```

Iterate thru all futures

FutureRunnable runs in a background thread & gets the results of all futures as they complete...
Overview of the PrimeChecker App

• FutureRunnable runs in a background thread & gets the results of all futures as they complete...

```java
public void run() {
    mFutures.forEach(future -> {
        PrimeCallable.PrimeResult pr = rethrowSupplier(future::get).get();
        if (pr.mSmallestFactor != 0) {
            ...
        } else {
            ...
        };
    });
    mActivity.done();...
```

future::get blocks if async processing associated with future hasn’t completed

mActivity.done();...

This is an example of the “synchronous future” processing model
Overview of the PrimeChecker App

- FutureRunnable runs in a background thread & gets the results of all futures as they complete...

```java
public void run() {
    mFutures.forEach(future -> {
        PrimeCallable.PrimeResult pr = rethrowSupplier(future::get).get();
        if (pr.mSmallestFactor != 0)
            ...
        else ...
    });
    mAActivity.done(); ...
}
```

Convert checked exception to a runtime exception

See [stackoverflow.com/a/27644392/3312330](https://stackoverflow.com/a/27644392/3312330)
Overview of the PrimeChecker App

- FutureRunnable runs in a background thread & gets the results of all futures as they complete...

```java
public void run() {
    mFutures.forEach(future -> {
        PrimeCallable.PrimeResult pr =
            rethrowSupplier(future::get).get();

        if (pr.mSmallestFactor != 0)
            ...
        else ...
    });

    mActivity.done(); ...
}
```

See [docs.oracle.com/javase/8/docs/api/java/util/function/Supplier.html#get](https://docs.oracle.com/javase/8/docs/api/java/util/function/Supplier.html#get)
Overview of the PrimeChecker App

- FutureRunnable runs in a background thread & gets the results of all futures as they complete

```java
public void run() {
    mFutures.forEach(future -> {
        PrimeCallable.PrimeResult pr =
            rethrowSupplier(future::get).get();
        if (pr.mSmallestFactor != 0)
            ... else ...
    });

    mActivity.done(); ...
```

*Process each result & produce output*
Overview of the PrimeChecker App

• FutureRunnable runs in a background thread & gets the results of all futures as they complete

```java
public void run() {
    mFutures.forEach(future -> {
        PrimeCallable.PrimeResult pr = rethrowSupplier(future::get).get();
        if (pr.mSmallestFactor != 0)
            ...
        else ...
    });

    mActivity.done(); ...
}
```

Inform MainActivity that we’re all done

mActivity.done(); ...

The interruptComputations() method shuts down all the concurrent computations via the UI thread.

```java
void interruptComputations() {
    mRetainedState.mExecutorService.shutdownNow();
    mRetainedState.mThread.interrupt();
    ...
    mRetainedState.mExecutorService.awaitTermination(500, TimeUnit.MILLISECONDS);
}
```
Overview of the PrimeChecker App

• The interruptComputations() method shuts down all the concurrent computations via the UI thread

```java
void interruptComputations() {
    mRetainedState.mExecutorService.shutdownNow();

    mRetainedState.mThread.interrupt();

    mRetainedState.mExecutorService.awaitTermination(500, TimeUnit.MILLISECONDS);
}
```

Called when user presses the cancel button
Overview of the PrimeChecker App

- The `interruptComputations()` method shuts down all the concurrent computations via the UI thread.

```java
void interruptComputations() {
    mRetainedState.mExecutorService.shutdownNow();
    mRetainedState.mThread.interrupt();
    ...
    mRetainedState.mExecutorService.awaitTermination(500, TimeUnit.MILLISECONDS);
}
```

Abruptly shutdown the executor service, which interrupts all threads running tasks.

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/ExecutorService.html#shutdownNow](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ExecutorService.html#shutdownNow)
The interruptComputations() method shuts down all the concurrent computations via the UI thread:

```java
void interruptComputations() {
    mRetainedState.mExecutorService.shutdownNow();
    mRetainedState.mThread.interrupt();
    ...
    Interrupt the background thread
    mRetainedState.mExecutorService.awaitTermination(500, TimeUnit.MILLISECONDS);
}
```

See [docs.oracle.com/javase/8/docs/api/java/lang/Thread.html#interrupt](docs.oracle.com/javase/8/docs/api/java/lang/Thread.html#interrupt)
Overview of the PrimeChecker App

- The interruptComputations() method shuts down all the concurrent computations via the UI thread

```java
void interruptComputations() {
    mRetainedState.mExecutorService.shutdownNow();
    mRetainedState.mThread.interrupt();
    ...
    mRetainedState.mExecutorService.awaitTermination(500, TimeUnit.MILLISECONDS);
    ...
}
```

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/ExecutorService.html#awaitTermination](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ExecutorService.html#awaitTermination)
Overview of the PrimeChecker App

- RetainedState contains fields that must be preserved across Android runtime configuration changes

```java
class RetainedState {
    ExecutorService mExecutorService;
    FutureRunnable mFutureRunnable;
    Thread mThread;
}
```

These fields store concurrency-related objects
RetainedState contains fields that must be preserved across Android runtime configuration changes...

```java
mRetainedState.mFutureRunnable = new FutureRunnable(this, futures);
```

FutureRunnable is stored in a field so its state can be updated during a runtime configuration change.

```java
mRetainedState.mThread = new Thread(mRetainedState.mFutureRunnable);
```

```java
mRetainedState.mThread.start();
```

See [developer.android.com/guide/topics/resources/runtime-changes.html](https://developer.android.com/guide/topics/resources/runtime-changes.html)
RetainedState contains fields that must be preserved across Android runtime configuration changes.

```java
mRetainedState.mFutureRunnable = new FutureRunnable(this, futures);
```

A background thread is started to wait for all future results to avoid blocking the UI thread.

```java
mRetainedState.mThread = new Thread(mRetainedState.mFutureRunnable);
mRetainedState.mThread.start();
```

See [developer.android.com/training/articles/perf-anr.html](http://developer.android.com/training/articles/perf-anr.html)
Overview of the PrimeChecker App

- Android provides hook methods to store & retrieve app state across runtime configuration changes

```java
Object onRetainNonConfigurationInstance() {
    return mRetainedState;
}
```

void onCreate(...) {
    mRetainedState = (RetainedState) getLastNonConfigurationInstance();
    if (mRetainedState != null) {
        ...
    }
}

See developer.android.com/reference/android/app/Activity.html#onRetainNonConfigurationInstance()
End of Applying the Java Executor Service to the PrimeChecker App