

Applying Java Structured Concurrency:

Case Study ex5

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

- Understand Java's structured concurrency model
- Recognize the classes used to program Java's structure concurrency model
- Evaluate the design & performance of various Java concurrency models
- Know how to implement a custom StructuredTaskScope
 - Case study ex5 shows how to create & apply a custom StructuredTaskScope

```
public class
ShutdownOnNonNullSuccess<T>
    extends StructuredTaskScope<T> {
    private volatile T mResult;
    protected void handleComplete
        (Future<T> future) {
        ...
        T result = future.resultNow();
        if (result != null) {
            mResult = result; shutdown();
        }
    }
    public T result()
    { return mResult; } ...
}
```

Applying Reactive Java Concurrency to Case Study ex5

Applying Reactive Java Concurrency to Case Study ex5

```
ex5 - ShutdownOnNonNullSuccess.java [ex5.main]
ex5 > src > main > java > utils > ShutdownOnNonNullSuccess

Project
  ex5 - ~/Dropbox/Documents/opp/Pearson/LiveLessons
    > .gradle
    > .idea
    > .settings
    > bin
    > build
    > gradle
    > src
      > main
        > java
          > utils
            > ExceptionUtils
            > FutureUtils
            > Options
            > PrimeUtils
            > ShutdownOnNonNullSuccess
            > ex5
          > .classpath
          > .gitignore
          > .project
          > build.gradle
          > gradlew
          > gradlew.bat
          > settings.gradle
        > External Libraries
        > Scratches and Consoles

Commit
Pull Requests
Bookmarks
Structure

17 /**
18  * A {@link StructuredTaskScope} that captures the result of the first
19  * subtask to complete successfully (i.e., without returning a {@code
20  * null}) or returns {@code null} if no subtask completes
21  * successfully. Once captured, it invokes the {@code shutdown()}
22  * method to interrupt unfinished threads and wakeup the owner.
23  *
24  * The policy implemented by this class is intended for cases where
25  * the result of any subtask will do ("invoke any") and where the
26  * results of other unfinished subtask are no longer needed.
27  */
28
29 public class ShutdownOnNonNullSuccess<T>
30     extends StructuredTaskScope<T> {
31     /**
32      * Stores the first computation to match or null if there are no
33      * matches.
34      */
35     private volatile T mResult;
36
37     /**
38      * Creates an unnamed structured task scope that creates virtual
39      * threads.
40      */
41     public ShutdownOnNonNullSuccess() { super("null", Thread.ofVirtual().factory(), 1); }
```

See github.com/douglasraigschmidt/LiveLessons/tree/master/Loom/ex5

End of Applying Java Structured Concurrency: Case Study ex5