

Applying Java Structured Concurrency: Case Study ex4 (Part 1)

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

- Case study ex4 compares & contrasts the programming models & performance results of Java parallel streams, completable futures, Project Reactor, RxJava, & Java structured concurrency when applied to download, transform, & store many images from a remote web server

```
try (var scope = new
    StructuredTaskScope
        .ShutdownOnFailure()) {
    var transformedImages = ...;

    for (var imageFuture :
        downloadedImages)
        transformedImages.addAll
            (transformImage(scope,
                imageFuture.resultNow()));

    rethrowRunnable(scope::join);

    return transformedImages;
}
```

Learning Objectives in this Part of the Lesson

- Case study ex4 compares & contrasts the programming models & performance results of Java parallel streams, completable futures, Project Reactor, RxJava, & Java structured concurrency when applied to download, transform, & store many images from a remote web server
- Part 1 of this case study focuses on the Java structured concurrency implementations

```
try (var scope = new
    StructuredTaskScope
        .ShutdownOnFailure()) {
    var transformedImages = ...;

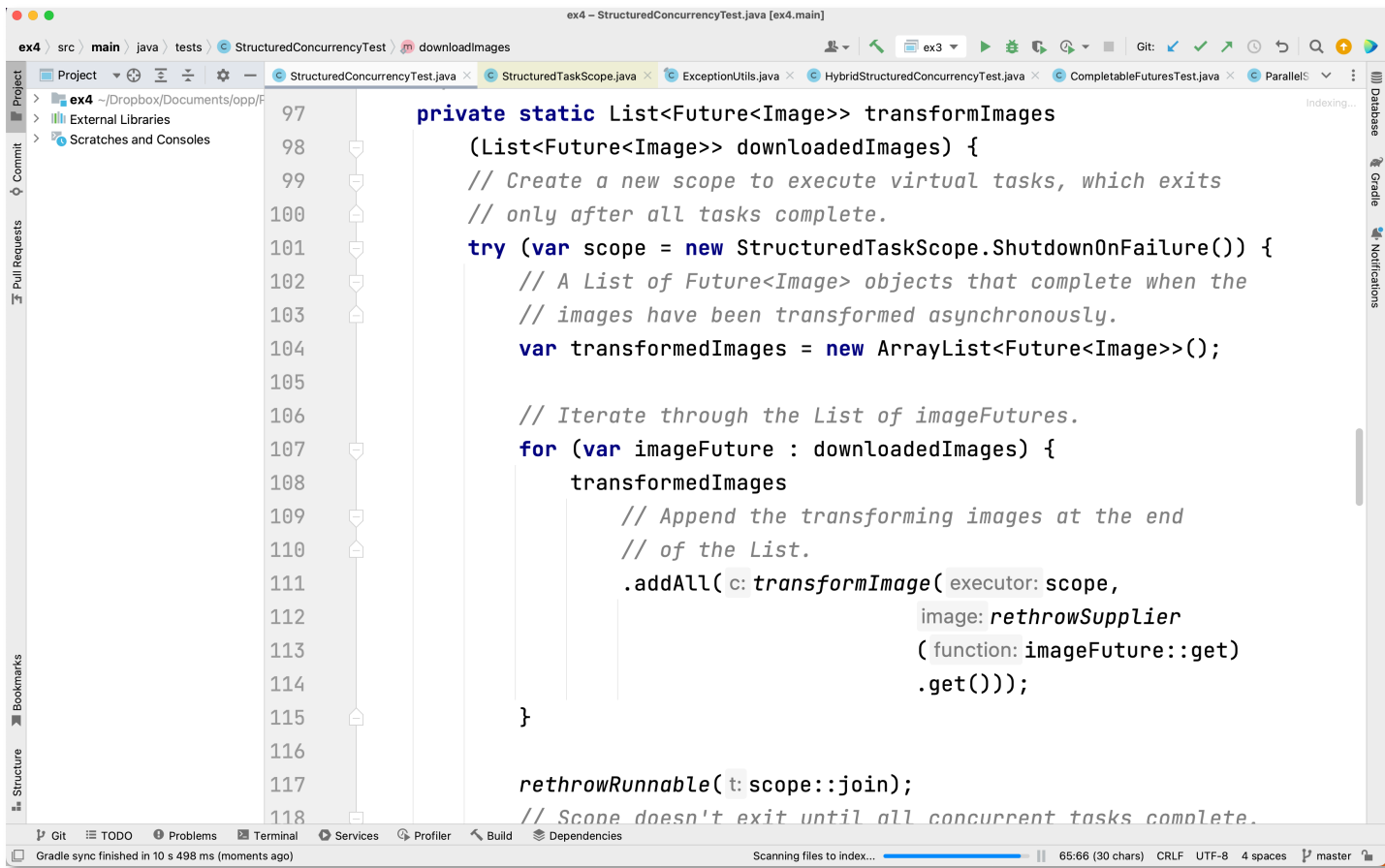
    for (var imageFuture :
        downloadedImages)
        transformedImages.addAll
            (transformImage(scope,
                imageFuture.resultNow()));

    rethrowRunnable(scope::join);

    return transformedImages;
}
```

Applying Java Structured Concurrency to Case Study ex4

Applying Java Structured Concurrency to Case Study ex4



The screenshot shows an IDE window titled "ex4 - StructuredConcurrencyTest.java [ex4.main]". The code is in Java and defines a method `transformImages` that takes a `List<Future<Image>>` and returns a `List<Future<Image>>`. The method uses `StructuredTaskScope` to manage a scope of tasks. It creates a `StructuredTaskScope.ShutdownOnFailure()` and then iterates over the `downloadedImages` list, adding each `Future<Image>` to the scope. The scope is then joined, and the transformed images are returned. The code is as follows:

```
97 private static List<Future<Image>> transformImages
98     (List<Future<Image>> downloadedImages) {
99     // Create a new scope to execute virtual tasks, which exits
100    // only after all tasks complete.
101    try (var scope = new StructuredTaskScope.ShutdownOnFailure()) {
102        // A List of Future<Image> objects that complete when the
103        // images have been transformed asynchronously.
104        var transformedImages = new ArrayList<Future<Image>>();
105
106        // Iterate through the List of imageFutures.
107        for (var imageFuture : downloadedImages) {
108            transformedImages
109                // Append the transforming images at the end
110                // of the List.
111                .addAll(c::transformImage(executor: scope,
112                                         image: rethrowSupplier
113                                         (function: imageFuture::get)
114                                         .get()));
115        }
116
117        rethrowRunnable(t: scope::join);
118        // Scope doesn't exit until all concurrent tasks complete.
```

The IDE interface includes a Project view on the left showing the file structure, a Run and Debug toolbar at the top, and a status bar at the bottom showing the current file position (65:66) and encoding (UTF-8).

See github.com/douglascraigschmidt/LiveLessons/tree/master/Loom/ex4

End of Applying Java Structured Concurrency: Case Study ex4 (Part 1)