The Java CompletableFuture ImageStreamGang Case Study: Applying Completable Futures

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

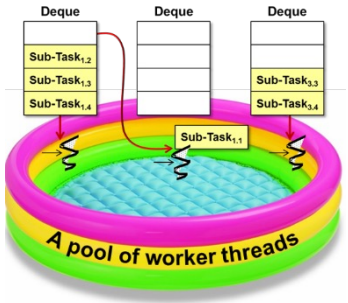
Professor of Computer Science

Institute for Software Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

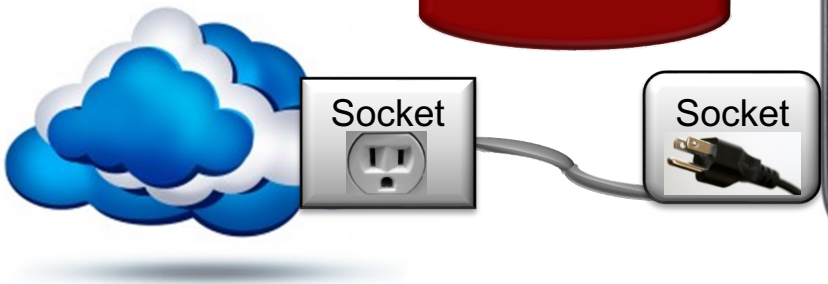
- Understand the design of the Java completable future version of ImageStreamGang
- Know how to apply completable futures to ImageStreamGang



Process a list of URLs to images that aren't already cached & download/ transform/store images asynchronously

List of URLs to Download

List of Filters to Apply



Applying Completable Futures to ImageStreamGang

Applying Completable Futures to ImageStreamGang

- Focus on processStream()

```
void processStream() {
  List<URL> urls = getInput();

  CompletableFuture<Stream<Image>>
    resultsFuture = urls
    .stream()
    .map(this::checkUrlCachedAsync)
    .map(this::downloadImageAsync)
    .flatMap(this::applyFiltersAsync)
    .collect(toFuture())
    .thenApply(stream ->
                log(stream.flatMap
                    (Optional::stream),
                     urls.size()))
    .join();
```

See imagestreamgang/streams/ImageStreamCompletableFuture1.java

Applying Completable Futures to ImageStreamGang

- Focus on processStream()

```
void processStream() {
  List<URL> urls = getInput();

  CompletableFuture<Stream<Image>>
    resultsFuture = urls
    .stream()
    .map(this::checkUrlCachedAsync)
    .map(this::downloadImageAsync)
    .flatMap(this::applyFiltersAsync)
    .collect(toFuture())
    .thenApply(stream ->
                 log(stream.flatMap
                     (Optional::stream),
                      urls.size()))
    .join();
```

Combines a Java sequential stream with completable futures

Applying Completable Futures to ImageStreamGang

- Focus on processStream()
  - This implementation begins like parallel streams version

Get the list of URLs input by the user

```
void processStream() {
  List<URL> urls = getInput();


  CompletableFuture<Stream<Image>>
    resultsFuture = urls
    .stream()
    .map(this::checkUrlCachedAsync)
    .map(this::downloadImageAsync)
    .flatMap(this::applyFiltersAsync)
    .collect(toFuture())
    .thenApply(stream ->
                log(stream.flatMap
                    (Optional::stream),
                     urls.size()))
    .join();
```

Applying Completable Futures to ImageStreamGang

- Focus on processStream()
  - This implementation begins like parallel streams version

```
void processStream() {
  List<URL> urls = getInput();

  CompletableFuture<Stream<Image>>
    resultsFuture = urls
    .stream()
    .map(this::checkUrlCachedAsync)
    .map(this::downloadImageAsync)
    .flatMap(this::applyFiltersAsync)
    .collect(toFuture())
    .thenApply(stream ->
                 log(stream.flatMap
                     (Optional::stream),
                      urls.size()))
    .join();
```

Factory method creates a stream of URLs

Applying Completable Futures to ImageStreamGang

- Focus on processStream()
  - This implementation begins like parallel streams version
  - However, it then becomes very different from parallel streams

```
void processStream() {
  List<URL> urls = getInput();

  CompletableFuture<Stream<Image>>
    resultsFuture = urls
    .stream()
    .map(this::checkUrlCachedAsync)
    .map(this::downloadImageAsync)
    .flatMap(this::applyFiltersAsync)
    .collect(toFuture())
    .thenApply(stream ->
                 log(stream.flatMap
                     (Optional::stream),
                      urls.size()))
    .join();
```

Asynchronously check if images have already been cached locally

map() converts a stream of URLs to a stream of futures to optional URLs

Applying Completable Futures to ImageStreamGang

- Focus on processStream()
  - This implementation begins like parallel streams version
  - However, it then becomes very different from parallel streams

Asynchronously download an image at each given URL

```
void processStream() {
  List<URL> urls = getInput();

  CompletableFuture<Stream<Image>>
    resultsFuture = urls
    .stream()
    .map(this::checkUrlCachedAsync)
    .map(this::downloadImageAsync)
    .flatMap(this::applyFiltersAsync)
    .collect(toFuture())
    .thenApply(stream ->
                log(stream.flatMap
                    (Optional::stream),
                     urls.size()))
    .join();
```

map() converts URL futures (completed) to image futures (downloading)

Applying Completable Futures to ImageStreamGang

- Focus on processStream()
  - This implementation begins like parallel streams version
  - However, it then becomes very different from parallel streams

```
void processStream() {
  List<URL> urls = getInput();

  CompletableFuture<Stream<Image>>
    resultsFuture = urls
    .stream()
    .map(this::checkUrlCachedAsync)
    .map(this::downloadImageAsync)
    .flatMap(this::applyFiltersAsync)
    .collect(toFuture())
    .thenApply(stream ->
                log(stream.flatMap
                    (Optional::stream),
                     urls.size()))
    .join();
```

Asynchronously filter & store downloaded images on the local file system

flatMap() converts image futures (completed) to filtered image futures (xforming/storing)

Applying Completable Futures to ImageStreamGang

- Focus on processStream()
  - This implementation begins like parallel streams version
  - However, it then becomes very different from parallel streams

```
void processStream() {
  List<URL> urls = getInput();

  CompletableFuture<Stream<Image>>
    resultsFuture = urls
    .stream()
    .map(this::checkUrlCachedAsync)
    .map(this::downloadImageAsync)
    .flatMap(this::applyFiltersAsync)
    .collect(toFuture())
    .thenApply(stream ->
                 log(stream.flatMap
                     (Optional::stream),
                      urls.size()))
    .join();
```

Trigger all intermediate operations & create a future used to wait for all async operations associated w/the stream of futures to complete

See lesson on "*Java CompletableFutures ImageStreamGang Example: StreamOfFuturesCollector*"

Applying Completable Futures to ImageStreamGang

- Focus on processStream()
  - This implementation begins like parallel streams version
  - However, it then becomes very different from parallel streams

```
void processStream() {
  List<URL> urls = getInput();

  CompletableFuture<Stream<Image>>
    resultsFuture = urls
    .stream()
    .map(this::checkUrlCachedAsync)
    .map(this::downloadImageAsync)
    .flatMap(this::applyFiltersAsync)
    .collect(toFuture())
    .thenApply(stream ->
                 log(stream.flatMap
                     (Optional::stream),
                      urls.size()))
    .join();
```

This lambda logs the results when all the futures in stream complete their async processing

Applying Completable Futures to ImageStreamGang

- Focus on processStream()
  - This implementation begins like parallel streams version
  - However, it then becomes very different from parallel streams

```
void processStream() {
  List<URL> urls = getInput();

  CompletableFuture<Stream<Image>>
    resultsFuture = urls
    .stream()
    .map(this::checkUrlCachedAsync)
    .map(this::downloadImageAsync)
    .flatMap(this::applyFiltersAsync)
    .collect(toFuture())
    .thenApply(stream ->
                  log(stream.flatMap
                    (Optional::stream),
                     urls.size()))
    .join();
```

This call removes all the empty Optional objects

See blog.knoldus.com/java-9-enhance-your-java-8-code-with-java-9-optional-api-enhancement

Applying Completable Futures to ImageStreamGang

- Focus on processStream()
  - This implementation begins like parallel streams version
  - However, it then becomes very different from parallel streams

```
void processStream() {
  List<URL> urls = getInput();

  CompletableFuture<Stream<Image>>
    resultsFuture = urls
    .stream()
    .map(this::checkUrlCachedAsync)
    .map(this::downloadImageAsync)
    .flatMap(this::applyFiltersAsync)
    .collect(toFuture())
    .thenApply(stream ->
                log(stream.flatMap
                    (Optional::stream),
                     urls.size()))
    .join();
```

Block until all images have been downloaded, processed, & stored

This join() is the one & only call in this implementation strategy!

End of the Java Completable Future ImageStreamGang Case Study: Applying Completable Futures