

The Java CompletableFuture ImageStreamGang Case Study: Applying Completion Stage Methods (Part 2)

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, e.g.
 - Factory methods
 - Completion stage methods
 - `downloadImageAsync()`
 - `applyFiltersAsync()`

<<Java Class>>	
G CompletableFuture<T>	
•	CompletableFuture()
•	cancel(boolean):boolean
•	isCancelled():boolean
•	isDone():boolean
•	get()
•	get(long,TimeUnit)
•	join()
•	complete(T):boolean
•	supplyAsync(Supplier<U>):CompletableFuture<U>
•	supplyAsync(Supplier<U>,Executor):CompletableFuture<U>
•	runAsync(Runnable):CompletableFuture<Void>
•	runAsync(Runnable,Executor):CompletableFuture<Void>
•	completedFuture(U):CompletableFuture<U>
•	thenApply(Function<?>):CompletableFuture<U>
•	thenAccept(Consumer<? super T>):CompletableFuture<Void>
•	thenCombine(CompletionStage<? extends U>,BiFunction<?>):CompletableFuture<V>
•	thenCompose(Function<?>):CompletableFuture<U>
•	whenComplete(BiConsumer<?>):CompletableFuture<T>
•	allOf(CompletableFuture[]<?>):CompletableFuture<Void>
•	anyOf(CompletableFuture[]<?>):CompletableFuture<Object>

Applying Completion Stage Methods in `ApplyFiltersAsync`

Applying Completion Stage Methods in ApplyFiltersAsync

- Asynchronously filter & store downloaded images on the local file system

```
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();  
}
```

*flatMap() calls behavior
applyFiltersAsync()*



Applying Completion Stage Methods in ApplyFiltersAsync

- Asynchronously filter & store downloaded images on the local file system

*Asynchronous filter images
& store them into files*



```
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();
```

Later operations ignore "empty" optional images

Applying Completion Stage Methods in ApplyFiltersAsync

- Asynchronously filter & store downloaded images on the local file system



"Flatten" all filtered/stored images into a single output stream

```
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 docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#flatMap

Applying Completion Stage Methods in ApplyFiltersAsync

- Asynchronously filter & store downloaded images on the local file system

```
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();  
}
```

Returns a stream of futures to optional images, which have a value if the image is being filtered or are empty if it is already cached

Applying Completion Stage Methods in ApplyFiltersAsync

- applyFiltersAsync() uses the thenApplyAsync() method internally

```
Stream<CompletableFuture<Optional<Image>>> applyFiltersAsync
    (CompletableFuture<Optional<Image>> imageFuture) {
    return mFilters
        .stream()

        .map(filter -> imageFuture
            .thenApplyAsync(imageOpt ->
                imageOpt
                    .map(image ->
                        makeFilterDecoratorWithImage
                            (filter, image).run()),
                        getExecutor())));
}
```

*Asynchronously filter images
& then store them into files*

See [imagestreamgang/streams/ImageStreamCompletableFuture1.java](https://github.com/Netflix/imagestreamgang/tree/master/streams/ImageStreamCompletableFuture1.java)

Applying Completion Stage Methods in ApplyFiltersAsync

- applyFiltersAsync() uses the thenApplyAsync() method internally

```
Stream<CompletableFuture<Optional<Image>>> applyFiltersAsync  
    (CompletableFuture<Optional<Image>> imageFuture) {  
    return mFilters  
        .stream()  Convert the list of filters into a stream  
  
        .map(filter -> imageFuture  
            .thenApplyAsync(imageOpt ->  
                imageOpt  
                .map(image ->  
                    makeFilterDecoratorWithImage  
                        (filter, image).run()),  
                getExecutor()));  
}
```

Applying Completion Stage Methods in ApplyFiltersAsync

- `applyFiltersAsync()` uses the `thenApplyAsync()` method internally

```
Stream<CompletableFuture<Optional<Image>>> applyFiltersAsync  
    (CompletableFuture<Optional<Image>> imageFuture) {  
    return mFilters  
        .stream()  
  
        .map(filter -> imageFuture  
            .thenApplyAsync(imageOpt ->  
                imageOpt  
                .map(image ->  
                    makeFilterDecoratorWithImage  
                        (filter, image).run()),  
                    getExecutor())) ;  
}
```

*Asynchronously apply a filter action
after image completes downloading*

Applying Completion Stage Methods in ApplyFiltersAsync

- `applyFiltersAsync()` uses the `thenApplyAsync()` method internally

```
Stream<CompletableFuture<Optional<Image>>> applyFiltersAsync  
    (CompletableFuture<Optional<Image>> imageFuture) {  
    return mFilters  
        .stream()  
  
        .map(filter -> imageFuture  
            .thenApplyAsync(imageOpt ->  
                imageOpt  
                .map(image ->  
                    makeFilterDecoratorWithImage  
                        (filter, image).run()),  
                    getExecutor())));  
}
```

This completion stage method registers an action that's not executed immediately, but runs only after imageFuture completes

Applying Completion Stage Methods in ApplyFiltersAsync

- applyFiltersAsync() uses the thenApplyAsync() method internally

```
Stream<CompletableFuture<Optional<Image>>> applyFiltersAsync  
    (CompletableFuture<Optional<Image>> imageFuture) {  
    return mFilters  
        .stream()  
  
        .map(filter -> imageFuture  
            .thenApplyAsync(imageOpt ->  
                imageOpt  
                .map(image ->  
                    makeFilterDecoratorWithImage  
                        (filter, image).run()),  
                    getExecutor())));}
```

If an image is present then perform the action & return optional containing result; otherwise return an empty optional

Applying Completion Stage Methods in ApplyFiltersAsync

- applyFiltersAsync() uses the thenApplyAsync() method internally

```
Stream<CompletableFuture<Optional<Image>>> applyFiltersAsync  
    (CompletableFuture<Optional<Image>> imageFuture) {  
    return mFilters  
        .stream()  
  
        .map(filter -> imageFuture  
            .thenApplyAsync(imageOpt ->  
                imageOpt  
                .map(image ->  
                    makeFilterDecoratorWithImage  
                        (filter, image).run()),  
                    getExecutor())));}
```

If image is non-empty then asynchronously filter the image & store it in an output file

Applying Completion Stage Methods in ApplyFiltersAsync

- `applyFiltersAsync()` uses the `thenApplyAsync()` method internally

```
Stream<CompletableFuture<Optional<Image>>> applyFiltersAsync  
    (CompletableFuture<Optional<Image>> imageFuture) {  
    return mFilters  
        .stream()  
  
        .map(filter -> imageFuture  
            .thenApplyAsync(imageOpt ->  
                imageOpt  
                .map(image ->  
                    makeFilterDecoratorWithImage  
                        (filter, image).run()),  
                    getExecutor())));}
```



`thenApplyAsync()` runs action in a worker thread from the common fork-join pool

Applying Completion Stage Methods in ApplyFiltersAsync

- applyFiltersAsync() uses the thenApplyAsync() method internally

```
Stream<CompletableFuture<Optional<Image>>> applyFiltersAsync  
    (CompletableFuture<Optional<Image>> imageFuture) {  
    return mFilters  
        .stream()  
  
        .map(filter -> imageFuture  
            .thenApplyAsync(imageOpt ->  
                imageOpt  
                .map(image ->  
                    makeFilterDecoratorWithImage  
                        (filter, image).run()),  
                    getExecutor())) ;  
}
```

It also returns a new completable future that will trigger when the image has been filtered/stored

Applying Completion Stage Methods in ApplyFiltersAsync

- `applyFiltersAsync()` uses the `thenApplyAsync()` method internally

```
Stream<CompletableFuture<Optional<Image>>> applyFiltersAsync
    (CompletableFuture<Optional<Image>> imageFuture) {
    return mFilters
        .stream()

        .map(filter -> imageFuture
            .thenApplyAsync(imageOpt ->
                imageOpt
                    .map(image ->
                        makeFilterDecoratorWithImage
                            (filter, image).run()),
                        getExecutor())));
}
```

*applyFiltersAsync() returns
a stream of completable
futures to optional images*

Applying Completion Stage Methods in ApplyFiltersAsync

- `applyFiltersAsync()` uses the `thenApplyAsync()` method internally

```
Stream<CompletableFuture<Optional<Image>>> applyFiltersAsync
    (CompletableFuture<Optional<Image>> imageFuture) {
    return mFilters
        .stream()

        .map(filter -> imageFuture
            .thenApplyAsync(imageOpt ->
                imageOpt
                    .map(image ->
                        makeFilterDecoratorWithImage
                            (filter, image).run()),
                        getExecutor())));
}
```

There is no terminal operation here, so a stream is returned!

Applying Completion Stage Methods in ApplyFiltersAsync

- Asynchronously filter & store downloaded images on the local file system



flatMap() merges the streams of futures returned by applyFilters Async() into just one stream

```
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 Completion Stage Methods in ApplyFiltersAsync

- Asynchronously filter & store downloaded images on the local file system

```
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 stream is processed by collect(), as discussed in the next parts of the lesson

End of the Java Completable Future ImageStreamGang Case Study: Applying Completion Stage Methods (Part 2)
