

Understand the Java CompletableFuture ImageStream

Gang Case Study: Applying Factory Methods

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
 - `supplyAsync()`

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

Applying Factory Methods in ImageStreamGang

Applying Factory Methods in ImageStreamGang

- Initiate an async check to see if images are cached locally

*map() calls the behavior
checkUrlCachedAsync()*

```
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 Factory Methods in ImageStreamGang

- Initiate an async check to see if images are cached locally

Asynchronously check if a URL is already downloaded

```
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 Factory Methods in ImageStreamGang

- Initiate an async check to see if images are cached locally

Returns a stream of completable futures to optional URLs, which have a value if the URL is not cached or are empty if it is cached

```
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 behaviors simply ignore "empty" optional URL values

Applying Factory Methods in ImageStreamGang

- `checkUrlCachedAsync()` uses the `supplyAsync()` factory method internally

```
CompletableFuture<Optional<URL>> checkUrlCachedAsync(URL url) {  
    return CompletableFuture  
        .supplyAsync(() ->  
            Optional.ofNullable(urlCached(url)  
                ? null  
                : url),  
            getExecutor()) ;  
}
```

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

Applying Factory Methods in ImageStreamGang

- `checkUrlCachedAsync()` uses the `supplyAsync()` factory method internally

```
CompletableFuture<Optional<URL>> checkUrlCachedAsync(URL url) {  
    return CompletableFuture  
        .supplyAsync(() ->  
            Optional.ofNullable(urlCached(url)  
                ? null  
                : url),  
            getExecutor());  
}
```

This factory method registers an action that runs asynchronously

Applying Factory Methods in ImageStreamGang

- `checkUrlCachedAsync()` uses the `supplyAsync()` factory method internally

```
CompletableFuture<Optional<URL>> checkUrlCachedAsync(URL url) {  
    return CompletableFuture  
        .supplyAsync(() ->  
            Optional.ofNullable(urlCached(url)  
                ? null  
                : url),  
            getExecutor());  
}
```

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

```
void initiateStream() {  
    // Set the executor to the common fork-join pool.  
    setExecutor(ForkJoinPool.commonPool());  
    ...  
}
```



See dzone.com/articles/common-fork-join-pool-and-streams

Applying Factory Methods in ImageStreamGang

- `checkUrlCachedAsync()` uses the `supplyAsync()` factory method internally

```
CompletableFuture<Optional<URL>> checkUrlCachedAsync(URL url) {  
    return CompletableFuture  
        .supplyAsync(() ->  
            Optional.ofNullable(urlCached(url)  
                ? null  
                : url),  
            getExecutor());  
}
```

ofNullable() is a factory method that returns an optional URL, which has a value if the URL is not cached or is empty if it is already cached

Applying Factory Methods in ImageStreamGang

- checkUrlCachedAsync() uses the supplyAsync() factory method internally

```
CompletableFuture<Optional<URL>> checkUrlCachedAsync(URL url) {  
    return CompletableFuture  
        .supplyAsync(() ->  
            Optional.ofNullable(urlCached(url)  
                ? null  
                : url),  
            getExecutor());  
}
```

*Returns true if the image has
already been filtered before*

```
boolean urlCached(URL url) {  
    return mFilters.stream()  
        .anyMatch(filter -> urlCached(url,  
            filter.getName()));  
}
```

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

Applying Factory Methods in ImageStreamGang

- checkUrlCachedAsync() uses the supplyAsync() factory method internally

```
CompletableFuture<Optional<URL>> checkUrlCachedAsync(URL url) {  
    return CompletableFuture  
        .supplyAsync(() ->  
            Optional.ofNullable(urlCached(url)  
                ? null  
                : url),  
            getExecutor());  
}
```

*Returns true if image
file already exists*

```
boolean urlCached(URL url, String filterName) {  
    File file = new File(getPath(), filterName);  
    File imageFile = new File(file, getNameForUrl(url));  
    return !imageFile.createNewFile();  
}
```

See [imagestreamgang/streams/ImageStreamGang.java](https://imagestreamgang.streams/ImageStreamGang.java)

Applying Factory Methods in ImageStreamGang

- `checkUrlCachedAsync()` uses the `supplyAsync()` factory method internally

```
CompletableFuture<Optional<URL>> checkUrlCachedAsync(URL url) {  
    return CompletableFuture  
        .supplyAsync(() ->  
            Optional.ofNullable(urlCached(url)  
                ? null  
                : url),  
            getExecutor());  
}
```



ClearlyBetter[®]
SOLUTIONS

```
boolean urlCached(URL url, String filterName) {  
    File file = new File(getPath(), filterName);  
    File imageFile = new File(file, getNameForUrl(url));  
    return !imageFile.createNewFile();  
}
```

There are clearly better ways of implementing an image cache!

End of Understand the Java CompletableFuture ImageStreamGang Case Study: Applying Factory Methods