Understand the Java CompletableFuture ImageStream

Gang Case Study: Applying Arbitrary-Arity 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
  - Completion stage methods
  - Arbitrary-arity methods

### CompletableFuture<T>

- `completableFuture()`: boolean
- `cancel(boolean)`: boolean
- `isCancelled()`: boolean
- `isDone()`: boolean
- `get()`: 
- `get(long, TimeUnit)`: 
- `join()`: 
- `complete(T)`: boolean
- `supplyAsync(Supplier<T>)`: CompletableFuture<T>
- `supplyAsync(Supplier<T>, Executor)`: CompletableFuture<T>
- `runAsync(Runnable)`: CompletableFuture<Void>
- `runAsync(Runnable, Executor)`: CompletableFuture<Void>
- `completedFuture(T)`: CompletableFuture<T>
- `thenApply(Function<? super T>)`: CompletableFuture<T>
- `thenAccept(Consumer<? super T>)`: CompletableFuture<Void>
- `thenCombine(CompletionStage<? extends U>, BiFunction<?>)`: CompletableFuture<V>
- `thenCompose(Function<? super T>)`: CompletableFuture<U>
- `allOf(CompletableFuture<T>[]): CompletableFuture<Void>
- `anyOf(CompletableFuture<T>[]): CompletableFuture<Object>`
Applying Arbitrary-Arity Methods in ImageStreamGang
Applying Arbitrary-Arity Methods in ImageStreamGang

- `collect()` returns a future to a stream of futures to images being processed asynchronously

```java
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()` outputs a stream of futures associated with processing that’s running asynchronously
Applying Arbitrary-Arity Methods in ImageStreamGang

- `collect()` returns a future to a stream of futures to images being processed asynchronously.

```java
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();
}
```

Provides a single means to reactively await the completion of a stream of futures before continuing.
Applying Arbitrary-Arity Methods in ImageStreamGang

- collect() returns a future to a stream of futures to images being processed asynchronously

```java
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();
}
```

collect() also triggers processing of all the intermediate operations
Applying Arbitrary-Arity Methods in ImageStreamGang

- `collect()` returns a future to a stream of futures to images being processed asynchronously.
- `StreamOfFuturesCollector` wraps “arbitrary-arity” `allOf()` method.

```java
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();
}
```

Return a future that completes when all futures in the stream complete.

See AndroidGUI/app/src/main/java/livelessons/utils/StreamOfFuturesCollector.java
Applying Arbitrary-Arity Methods in ImageStreamGang

- collect() returns a future to a stream of futures to images being processed asynchronously
- StreamOfFuturesCollector wraps "arbitrary-arity" allOf() method

```java
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();
}
```

Log the results after the final future completes
Applying Arbitrary-Arity Methods in ImageStreamGang

- `collect()` returns a future to a stream of futures to images being processed asynchronously
- `StreamOfFuturesCollector` wraps “arbitrary-arity” `allOf()` method

```java
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();
}
```

*Remove empty optional values from the stream in Java 9+

See [docs.oracle.com/javase/9/docs/api/java/util/Optional.html#flatMap](docs.oracle.com/javase/9/docs/api/java/util/Optional.html#flatMap)
Applying Arbitrary-Arity Methods in ImageStreamGang

- `collect()` returns a future to a stream of futures to images being processed asynchronously
- `StreamOfFuturesCollector` wraps “arbitrary-arity” `allOf()` method

```java
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
            .filter(Optional::isPresent)
            .map(Optional::get),
            urls.size()))
        .join();
}
```

Remove empty optional values from the stream in Java 8
Applying Arbitrary-Arity Methods in ImageStreamGang

- collect() returns a future to a stream of futures to images being processed asynchronously
- StreamOfFuturesCollector wraps “arbitrary-arity” allOf() method

```java
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
            .filter(Optional::isPresent)
            .map(Optional::get), urls.size()))
        .join();
}
```

Java 8 is more verbose..
Applying Arbitrary-Arity Methods in ImageStreamGang

- `collect()` returns a future to a stream of futures to images being processed asynchronously
- Images are displayed after async processing completes
- `StreamOfFuturesCollector` wraps "arbitrary-arity" `allOf()` method

```java
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();
}
```

Wait until all the async processing is completed

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html#join](http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html#join)
Applying Arbitrary-Arity Methods in ImageStreamGang

- `collect()` returns a future to a stream of futures to images being processed asynchronously.

```java
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 is the one & only call to `join()` in this async stream pipeline!
Applying Arbitrary-Arity Methods in ImageStreamGang

- `collect()` returns a future to a stream of futures to images being processed asynchronously.

```java
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();
}
```

Images are displayed after all the async processing completes.
End of Understand the Java Completable Future ImageStreamGang Case Study: Applying Arbitrary-Arity Methods