Java 8 CompletableFutures
ImageStreamGang Example (Part 3)

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 8 completable future version of ImageStreamGang
- Know how completable futures are applied to ImageStreamGang
- Factory methods & completion stage methods
- Arbitrary-arity methods
Applying Arbitrary-Arity Methods in ImageStreamGang
Applying Arbitrary-Arity Methods in ImageStreamGang

- collect() creates a list of futures to images that are in the process of being downloaded, filtered, & stored

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

    List<CompletableFuture<Image>>
    listOfFutures = urls
        .stream()
        .filter(not(this::urlCached))
        .map(this::downloadImageAsync)
        .flatMap(this::applyFiltersAsync)
        .collect(toList());

    ...
```
Applying Arbitrary-Arity Methods in ImageStreamGang

- `collect()` creates a list of futures to images that are in the process of being downloaded, filtered, & stored
- These images must be displayed after their processing completes

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

    List<CompletableFuture<Image>>
    listOfFutures = urls
        .stream()
        .filter(not(this::urlCached))
        .map(this::downloadImageAsync)
        .flatMap(this::applyFiltersAsync)
        .collect(toList());

    ...
```
Applying Arbitrary-Arity Methods in ImageStreamGang

- `StreamsUtils.joinAll()` wraps the "arbitrary-arity" method `allOf()`

```java
void processStream() {
    ... 
    List<CompletableFuture<Image>> futureList = ... 
    CompletableFuture<List<Image>> allImagesDone = StreamsUtils .joinAll(futureList); 
    int imagesProcessed = allImagesDone .join() .stream() .collect(summingInt(List::size)); 
}
```

See [AndroidGUI/app/src/main/java/livelessons/utils/StreamUtils.java](AndroidGUI/app/src/main/java/livelessons/utils/StreamUtils.java)
Applying Arbitrary-Arity Methods in ImageStreamGang

- StreamsUtils.joinAll() wraps the "arbitrary-arity" method allOf()

```java
void processStream() {
    ... 
    List<CompletableFuture<Image>> futureList = ... 

    CompletableFuture<List<Image>> allImagesDone = StreamsUtils.joinAll(futureList);

    int imagesProcessed = allImagesDone.join().stream().collect(summingInt(List::size));
}
```

Wait for all $n$ of the futures to complete
Applying Arbitrary-Arity Methods in ImageStreamGang

- StreamsUtils.joinAll() wraps the "arbitrary-arity" method allOf()

```java
tuple

```
Implementing StreamUtils.joinAll() in ImageStreamGang
Implementing `StreamUtils.joinAll()` in `ImageStreamGang`

- The `StreamUtils.joinAll()` method is a wrapper that encapsulates `allOf()`

```java
static <T> CompletableFuture<List<T>> joinAll(List<CompletableFuture<T>> fList) {
    CompletableFuture<Void> dFuture = CompletableFuture.allOf(fList.toArray(new CompletableFuture[fList.size()]));

    CompletableFuture<List<T>> dList = dFuture.thenApply(v -> fList.stream()
        .map(CompletableFuture::join)
        .collect(toList()));
    return dList;
}
```

See `AndroidGUI/app/src/main/java/livelessons/utils/StreamUtils.java`
Implementing `StreamUtils.joinAll()` in `ImageStreamGang`

- The `StreamUtils.joinAll()` method is a wrapper that encapsulates `allOf()`

```java
static <T> CompletableFuture<List<T>>
joinAll(List<CompletableFuture<T>> fList) {
    CompletableFuture<Void> dFuture = CompletableFuture.allOf
        (fList.toArray(new CompletableFuture[fList.size()]));
    CompletableFuture<List<T>> dList =
        dFuture.thenApply(v -> fList
            .stream()
            .map(CompletableFuture::join)
            .collect(toList()));
    return dList;
}
```

The return value converts a list of completed futures into a list of joined results
Implementing StreamUtils.joinAll() in ImageStreamGang

- The StreamUtils.joinAll() method is a wrapper that encapsulates allOf()

```java
static <T> CompletableFuture<List<T>>
joinAll(List<CompletableFuture<T>> fList) {
    CompletableFuture<Void> dFuture = CompletableFuture.allOf
        (fList.toArray(new CompletableFuture[fList.size()]));
    CompletableFuture<List<T>> dList =
        dFuture.thenApply(v -> fList
            .stream()
            .map(CompletableFuture::join)
            .collect(toList()));
    return dList;
}
```

The parameter is a list of completable futures to some generic type T
Implementing `StreamUtils.joinAll()` in `ImageStreamGang`

- The `StreamUtils.joinAll()` method is a wrapper that encapsulates `allOf()`

```java
static <T> CompletableFuture<List<T>> joinAll(List<CompletableFuture<T>> fList) {
    CompletableFuture<Void> dFuture = CompletableFuture.allOf
        (fList.toArray(new CompletableFuture[fList.size()]));

    CompletableFuture<List<T>> dList = dFuture.thenApply(v -> fList
        .stream()
        .map(CompletableFuture::join)
        .collect(toList()));

    return dList;
}
```

Returns a completable future that completes when all completable futures complete

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html#allOf](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html#allOf)
The StreamUtils.joinAll() method is a wrapper that encapsulates allOf()

```java
static <T> CompletableFuture<List<T>>
joinAll(List<CompletableFuture<T>> fList) {
    CompletableFuture<Void> dFuture = CompletableFuture.allOf
        (fList.toArray(new CompletableFuture[fList.size()]));
    CompletableFuture<List<T>> dList =
        dFuture.thenApply(v -> fList
            .stream()
            .map(CompletableFuture::join)
            .collect(toList()));
    return dList;
}
```
The StreamUtils.joinAll() method is a wrapper that encapsulates allOf()

```
static <T> CompletableFuture<List<T>>
joinAll(List<CompletableFuture<T>> fList) {
    CompletableFuture<Void> dFuture = CompletableFuture.allOf
        (fList.toArray(new CompletableFuture[fList.size()]));
    CompletableFuture<List<T>> dList =
        dFuture.thenApply(v -> fList
            .stream()
            .map(CompletableFuture::join)
            .collect(toList()));
    return dList;
}
```

Creates a completable future to a list of joined results when all completable futures complete
The `StreamUtils.joinAll()` method is a wrapper that encapsulates `allOf()` method of `CompletableFuture`. It takes a list of `CompletableFuture` instances and returns a complete future of the list of joined results.

```java
static <T> CompletableFuture<List<T>> joinAll(List< CompletableFuture<T>> fList) {
    CompletableFuture<Void> dFuture = CompletableFuture.allOf(
        fList.toArray(new CompletableFuture[fList.size()]));
    CompletableFuture<List<T>> dList = dFuture.thenApply(v -> fList
        .stream()
        .map(CompletableFuture::join)
        .collect(toList()));
    return dList;
}
```

Return a completable future to the list of joined results.
Implementing `StreamUtils.joinAll()` in `ImageStreamGang`

- `joinAll()` provides a very powerful wrapper for some complex code!!

```java
static <T> CompletableFuture<List<T>>
    joinAll(List<CompletableFuture<T>> fList) {
    CompletableFuture<Void> dFuture = CompletableFuture.allOf
            (fList.toArray(new CompletableFuture[fList.size()]));
    CompletableFuture<List<T>> dList =
            dFuture.thenApply(v -> fList
                                 .stream()
                                 .map(CompletableFuture::join)
                                 .collect(toList()));
    return dList;
}
```

See www.nurkiewicz.com/2013/05/java-8-completablefuture-in-action.html
Implementing StreamUtils.joinAll() in ImageStreamGang

- joinAll() provides a very powerful wrapper for some complex code!!!

```java
static <T> CompletableFuture<List<T>> joinAll(List<CompletableFuture<T>> fList) {
    CompletableFuture<Void> dFuture = CompletableFuture.allOf(
        fList.toArray(new CompletableFuture[fList.size()]));
    CompletableFuture<List<T>> dList =
        dFuture.thenApply(v -> fList
            .stream()
            .map(CompletableFuture::join)
            .collect(toList()));
    return dList;
}
```

Also see www.nurkiewicz.com/2013/05/java-8-definitive-guide-to.html
End of Java 8 Completable Futures ImageStreamGang Example (Part 3)