Java 8 CompletableFutures

ImageStreamGang Example (Part 3)

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

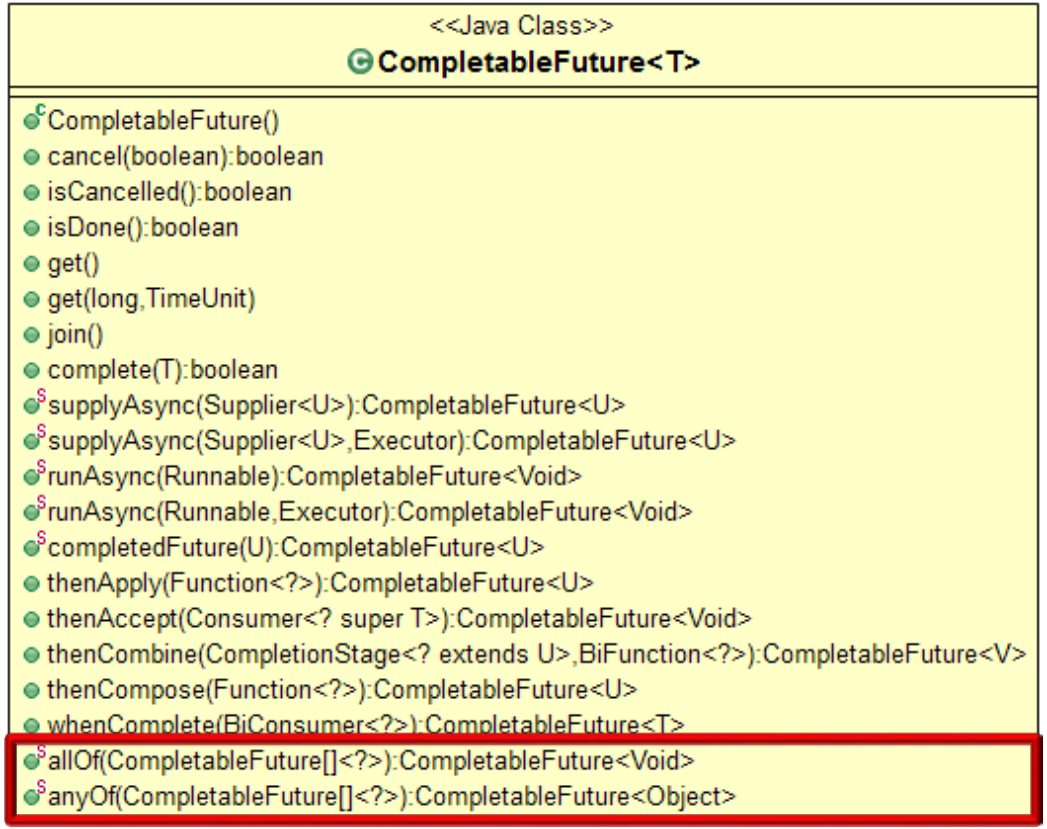
Professor of Computer Science

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

```
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 must be displayed after their processing completes

```
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()

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

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

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

Return a completable future that triggers when all async computation complete

See AndroidGUI/app/src/main/java/livelessons/utils/StreamUtils.java

Applying Arbitrary-Arity Methods in ImageStreamGang

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

```
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()

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

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

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

Sum up the total count of images after they are downloaded, filtered, & stored asynchronously

Implementing StreamUtils.joinAll() in ImageStreamGang

Implmenting StreamUtils.joinAll() in ImageStreamGang

- 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;
}
```

See AndroidGUI/app/src/main/java/livelessons/utils/StreamUtils.java

Implmenting StreamUtils.joinAll() in ImageStreamGang

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

The return value converts a list of completed futures into a list of joined results

```
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;
}
```

Implmenting StreamUtils.joinAll() in ImageStreamGang

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

The parameter is a list of completable futures to some generic type T

```
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;
}
```

Implmenting StreamUtils.joinAll() in ImageStreamGang

- 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;
}
```

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

Implmenting StreamUtils.joinAll() in ImageStreamGang

- 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;
}
```

Create an array that stores the list of completable futures

Implmenting StreamUtils.joinAll() in ImageStreamGang

- 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

Implmenting StreamUtils.joinAll() in ImageStreamGang

- 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;
}
```

Return a completable future to the list of joined results

Implmenting StreamUtils.joinAll() in ImageStreamGang

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

```
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

Implmenting StreamUtils.joinAll() in ImageStreamGang

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

```
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)