Java 8 CompletableFutures

ImageStreamGang Example (Part 2)

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

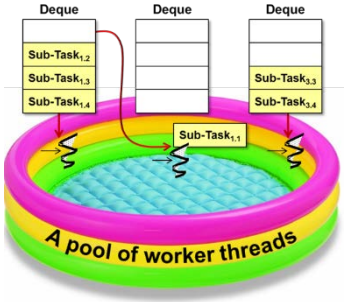
Professor of Computer Science

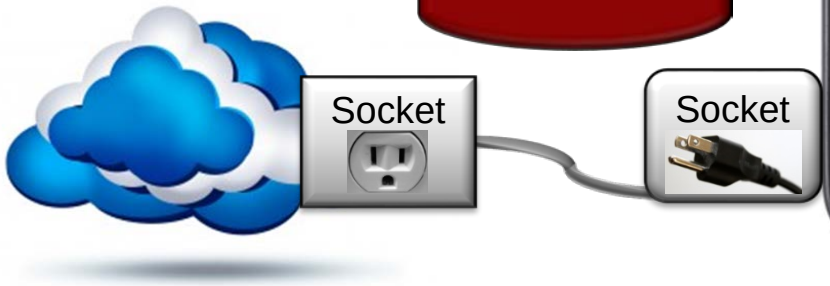
Institute for Software
Integrated Systems

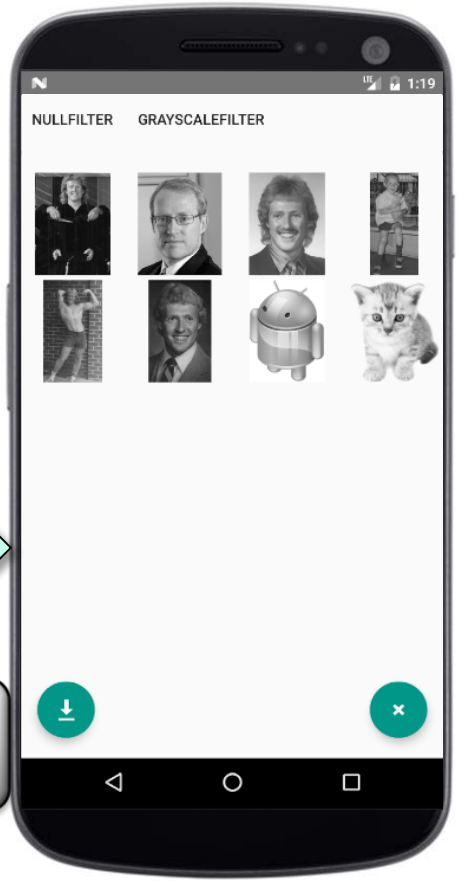
Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

- Understand the design of the Java 8 completable future version of the ImageStreamGang app
- Know how the Java 8 completable future framework is applied to the ImageStreamGang app







Applying Completable Futures to ImageStreamGang

Applying Completable Futures to ImageStreamGang

- We focus on the method processStream()

```
void processStream() {
  List<CompletableFuture<List<Image>>>
  listOfFutures = getInput()
    .stream()
    .filter(not(this::urlCached))
    .map(this::downloadImageAsync)
    .flatMap(this::applyFiltersAsync)
    .collect(toList());
  ...
```

See app/src/main/java/livelessons/imagestreamgang/streams/ImageStreamCompletableFuture1.java

Applying Completable Futures to ImageStreamGang

- We focus on the method processStream()

```
void processStream() {
  List<CompletableFuture<List<Image>>>
  listOfFutures = getInput()
    .stream()
    .filter(not(this::urlCached))
    .map(this::downloadImageAsync)
    .flatMap(this::applyFiltersAsync)
    .collect(toList());
  ...
```

Applying Completable Futures to ImageStreamGang

- We focus on the method processStream()
  - Same as parallel streams

```
void processStream() {
  List<CompletableFuture<List<Image>>>
  listOfFutures = getInput()
    .stream()
    .filter(not(this::urlCached))
    .map(this::downloadImageAsync)
    .flatMap(this::applyFiltersAsync)
    .collect(toList());
  ...
```

Create a stream & ignore cached images

Applying Completable Futures to ImageStreamGang

- We focus on the method processStream()
  - Same as parallel streams

```
void processStream() {
  List<CompletableFuture<List<Image>>>
  listOfFutures = getInput()
    .stream()
    .filter(not(this::urlCached))
    .map(this::downloadImageAsync)
    .flatMap(this::applyFiltersAsync)
    .collect(toList());
  ...
```

Create a stream & ignore cached images

Applying Completable Futures to ImageStreamGang

- We focus on the method processStream()
  - Same as parallel streams
  - downloadImageAsync() uses supplyAsync()

```
void processStream() {
  List<CompletableFuture<List<Image>>>
  listOfFutures = getInput()
    .stream()
    .filter(not(this::urlCached))
    .map(this::downloadImageAsync)
    .flatMap(this::applyFiltersAsync)
    .collect(toList());
...
```

Asynchronously download each URL in the input stream & return completable futures in the output stream

Applying Completable Futures to ImageStreamGang

- We focus on the method processStream()
  - Same as parallel streams
  - downloadImageAsync() uses supplyAsync()

```
void processStream() {
  List<CompletableFuture<List<Image>>>
  listOfFutures = getInput()
    .stream()
    .filter(not(this::urlCached))
    .map(this::downloadImageAsync)
    .flatMap(this::applyFiltersAsync)
    .collect(toList());
...
```

Asynchronously download each URL in the input stream & return completable futures in the output stream

Applying Completable Futures to ImageStreamGang

- We focus on the method processStream()
  - Same as parallel streams
  - downloadImageAsync() uses supplyAsync()

```
void processStream() {
  List<CompletableFuture<List<Image>>>
  listOfFutures = getInput()
    .stream()
    .filter(not(this::urlCached))
    .map(this::downloadImageAsync)
    .flatMap(this::applyFiltersAsync)
    .collect(toList());
...
```

Asynchronously download each URL in the input stream & return completable futures in the output stream

Applying Completable Futures to ImageStreamGang

- We focus on the method processStream()
  - Same as parallel streams
  - downloadImageAsync() uses supplyAsync()
  - applyFiltersAsync() uses thenApply(), thenCompose(), & supplyAsync()

```
void processStream() {
  List<CompletableFuture<List<Image>>>
  listOfFutures = getInput()
    .stream()
    .filter(not(this::urlCached))
    .map(this::downloadImageAsync)
    .flatMap(this::applyFiltersAsync)
    .collect(toList());
...
```

Asynchronously filter & store each downloaded image in the input stream & return completable futures in the output stream

Applying Completable Futures to ImageStreamGang

- We focus on the method processStream()
  - Same as parallel streams
  - downloadImageAsync() uses supplyAsync()
  - applyFiltersAsync() uses thenApply(), thenCompose(), & supplyAsync()

```
void processStream() {
  List<CompletableFuture<List<Image>>>
  listOfFutures = getInput()
    .stream()
    .filter(not(this::urlCached))
    .map(this::downloadImageAsync)
    .flatMap(this::applyFiltersAsync)
    .collect(toList());
...
```

Asynchronously filter & store each downloaded image in the input stream & return completable futures in the output stream

Applying Completable Futures to ImageStreamGang

- We focus on the method processStream()
  - Same as parallel streams
  - downloadImageAsync() uses supplyAsync()
  - applyFiltersAsync() uses thenApply(), thenCompose(), & supplyAsync()

```
void processStream() {
  List<CompletableFuture<List<Image>>>
  listOfFutures = getInput()
    .stream()
    .filter(not(this::urlCached))
    .map(this::downloadImageAsync)
    .flatMap(this::applyFiltersAsync)
    .collect(toList());
...
```

Asynchronously filter & store each downloaded image in the input stream & return completable futures in the output stream

Applying Completable Futures to ImageStreamGang

- We focus on the method processStream()
  - Same as parallel streams
  - downloadImageAsync() uses supplyAsync()
  - applyFiltersAsync() uses thenApply(), thenCompose(), & supplyAsync()

```
void processStream() {
  List<CompletableFuture<List<Image>>>
  listOfFutures = getInput()
    .stream()
    .filter(not(this::urlCached))
    .map(this::downloadImageAsync)
    .flatMap(this::applyFiltersAsync)
    .collect(toList());
...
```

Asynchronously filter & store each downloaded image in the input stream & return completable futures in the output stream

Applying Completable Futures to ImageStreamGang

- We focus on the method processStream()
  - Same as parallel streams
  - downloadImageAsync() uses supplyAsync()
  - applyFiltersAsync() uses thenApply(), thenCompose(), & supplyAsync()

```
void processStream() {
  List<CompletableFuture<List<Image>>>
  listOfFutures = getInput()
    .stream()
    .filter(not(this::urlCached))
    .map(this::downloadImageAsync)
    .flatMap(this::applyFiltersAsync)
    .collect(toList());
...
```

Asynchronously filter & store each downloaded image in the input stream & return completable futures in the output stream

Applying Completable Futures to ImageStreamGang

- We focus on the method processStream()
  - Same as parallel streams
  - downloadImageAsync() uses supplyAsync()
  - applyFiltersAsync() uses thenApply(), thenCompose(), & supplyAsync()

```
void processStream() {
  List<CompletableFuture<List<Image>>>
  listOfFutures = getInput()
    .stream()
    .filter(not(this::urlCached))
    .map(this::downloadImageAsync)
    .flatMap(this::applyFiltersAsync)
    .collect(toList());
...
```

Creates & stores list of completable futures to images that are being filtered & stored

Applying Completable Futures to ImageStreamGang

- We focus on the method processStream()
  - Same as parallel streams
  - downloadImageAsync() uses supplyAsync()
  - applyFiltersAsync() uses thenApply(), thenCompose(), & supplyAsync()

```
void processStream() {
  List<CompletableFuture<List<Image>>>
  listOfFutures = getInput()
    .stream()
    .filter(not(this::urlCached))
    .map(this::downloadImageAsync)
    .flatMap(this::applyFiltersAsync)
    .collect(toList());
...
```

Creates & stores list of completable futures to images that are being filtered & stored

Applying Completable Futures to ImageStreamGang

- We focus on the method processStream()
  - Same as parallel streams
  - downloadImageAsync() uses supplyAsync()
  - applyFiltersAsync() uses thenApply(), thenCompose(), & supplyAsync()

```
void processStream() {
  List<CompletableFuture<List<Image>>>
  listOfFutures = getInput()
    .stream()
    .filter(not(this::urlCached))
    .map(this::downloadImageAsync)
    .flatMap(this::applyFiltersAsync)
    .collect(toList());
...
```

Creates & stores list of completable futures to images that are being filtered & stored

Applying Completable Futures to ImageStreamGang

- We focus on the method processStream()
  - Same as parallel streams
  - downloadImageAsync() uses supplyAsync()
  - applyFiltersAsync() uses thenApply(), thenCompose(), & supplyAsync()
  - Uses “arbitrary-arity” allOf() & thenApply()

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

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

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

Return a completable future that’s used to know when all asynchronous functions have completed

Applying Completable Futures to ImageStreamGang

- We focus on the method processStream()
  - Same as parallel streams
  - downloadImageAsync() uses supplyAsync()
  - applyFiltersAsync() uses thenApply(), thenCompose(), & supplyAsync()
  - Uses “arbitrary-arity” allOf() & thenApply()

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

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

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

Return a completable future that’s used to know when all asynchronous functions have completed

Applying Completable Futures to ImageStreamGang

- We focus on the method processStream()
  - Same as parallel streams
  - downloadImageAsync() uses supplyAsync()
  - applyFiltersAsync() uses thenApply(), thenCompose(), & supplyAsync()
  - Uses “arbitrary-arity” allOf() & thenApply()

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

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

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

Return a completable future that’s used to know when all asynchronous functions have completed

Applying Completable Futures to ImageStreamGang

- We focus on the method processStream()
  - Same as parallel streams
  - downloadImageAsync() uses supplyAsync()
  - applyFiltersAsync() uses thenApply(), thenCompose(), & supplyAsync()
  - Uses “arbitrary-arity” allOf() & thenApply()

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

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

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

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

Applying Completable Futures to ImageStreamGang

- We focus on the method processStream()
  - Same as parallel streams
  - downloadImageAsync() uses supplyAsync()
  - applyFiltersAsync() uses thenApply(), thenCompose(), & supplyAsync()
  - Uses “arbitrary-arity” allOf() & thenApply()

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

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

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

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

Applying Completable Futures to ImageStreamGang

- We focus on the method processStream()
  - Same as parallel streams
  - downloadImageAsync() uses supplyAsync()
  - applyFiltersAsync() uses thenApply(), thenCompose(), & supplyAsync()
  - Uses “arbitrary-arity” allOf() & thenApply()

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

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

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

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

Applying Completable Futures to ImageStreamGang

- We focus on the method processStream()
  - Same as parallel streams
  - downloadImageAsync() uses supplyAsync()
  - applyFiltersAsync() uses thenApply(), thenCompose(), & supplyAsync()
  - Uses “arbitrary-arity” allOf() & thenApply()

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

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

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

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

End of Java 8 Completable Futures ImageStreamGang Example (Part 2)