

Learn How to Implement Behaviors in the Java Parallel ImageStreamGang Case Study

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 purpose of the ImageStreamGang app
- Recognize patterns applied in the ImageStreamGang app
- Know how the structure of the ImageStreamGang app
- Visualize how Java parallel streams are applied to the ImageStreamGang app
- Learn how the parallel stream behaviors of ImageStreamGang are implemented

```
void processStream() {  
    List<Image> filteredImages =  
        getInput()  
            .parallelStream()  
            .filter(not(this::urlCached))  
            .map(this::blockingDownload)  
            .map(this::applyFilters)  
            .reduce(Stream::concat)  
            .orElse(Stream.empty())  
            .collect(toList());  
  
    System.out.println(TAG  
        + "Image(s) filtered = "  
        + filteredImages.size());  
}
```

See github.com/douglasraigschmidt/LiveLessons/blob/master/ImageStreamGang

Implementing a Parallel Stream in ImageStreamGang

Implementing a Parallel Stream in ImageStreamGang

- We focus on `processStream()` in `ImageStreamParallel.java`

```
void processStream() {  
    List<Image> filteredImages =  
        getInput()  
        .parallelStream()  
        .filter(not(this::urlCached))  
        .map(this::blockingDownload)  
        .map(this::applyFilters)  
        .reduce(Stream::concat)  
        .orElse(Stream.empty())  
        .collect(toList());  
  
    System.out.println(TAG  
        + "Image(s) filtered = "  
        + filteredImages.size());  
}
```

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

Implementing a Parallel Stream in ImageStreamGang

- We focus on processStream() in ImageStreamParallel.java

Get a list of URLs

```
void processStream() {  
    List<Image> filteredImages =  
        getInput()  
        .parallelStream()  
        .filter(not(this::urlCached))  
        .map(this::blockingDownload)  
        .map(this::applyFilters)  
        .reduce(Stream::concat)  
        .orElse(Stream.empty())  
        .collect(toList());  
  
    System.out.println(TAG  
        + "Image(s) filtered = "  
        + filteredImages.size());  
}
```

getInput() is defined by the underlying StreamGang framework

Implementing a Parallel Stream in ImageStreamGang

- We focus on `processStream()` in `ImageStreamParallel.java`

*Convert a collection
into a parallel stream*

```
void processStream() {  
    List<Image> filteredImages =  
        getInput()  
        .parallelStream()  
        .filter(not(this::urlCached))  
        .map(this::blockingDownload)  
        .map(this::applyFilters)  
        .reduce(Stream::concat)  
        .orElse(Stream.empty())  
        .collect(toList());  
  
    System.out.println(TAG  
        + "Image(s) filtered = "  
        + filteredImages.size());  
}
```

Implementing a Parallel Stream in ImageStreamGang

- We focus on `processStream()` in `ImageStreamParallel.java`

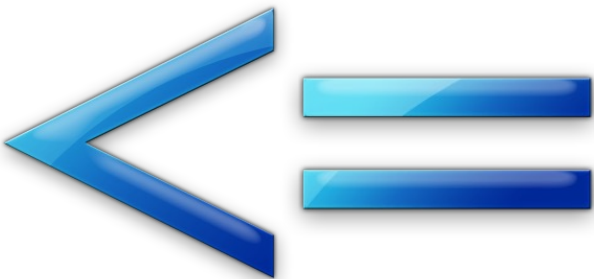
Return an output stream consisting of the URLs in the input stream that are not already cached

```
void processStream() {  
    List<Image> filteredImages =  
        getInput()  
        .parallelStream()  
        .filter(not(this::urlCached))  
        .map(this::blockingDownload)  
        .map(this::applyFilters)  
        .reduce(Stream::concat)  
        .orElse(Stream.empty())  
        .collect(toList());  
  
    System.out.println(TAG  
        + "Image(s) filtered = "  
        + filteredImages.size());  
}
```

Implementing a Parallel Stream in ImageStreamGang

- We focus on `processStream()` in `ImageStreamParallel.java`

Return an output stream consisting of the URLs in the input stream that are not already cached



```
void processStream() {  
    List<Image> filteredImages =  
        getInput()  
        .parallelStream()  
        .filter(not(this::urlCached))  
        .map(this::blockingDownload)  
        .map(this::applyFilters)  
        .reduce(Stream::concat)  
        .orElse(Stream.empty())  
        .collect(toList());  
  
    System.out.println(TAG  
        + "Image(s) filtered = "  
        + filteredImages.size());  
}
```

of output stream elements will be $\leq$ # of input stream elements

Implementing a Parallel Stream in ImageStreamGang

- We focus on processStream() in ImageStreamParallel.java

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

Determine whether this url has been downloaded to an image & had filters applied to it yet

```
void processStream() {  
    List<Image> filteredImages =  
        getInput()  
            .parallelStream()  
            .filter(not(this::urlCached))  
            .map(this::blockingDownload)  
            .map(this::applyFilters)  
            .reduce(Stream::concat)  
            .orElse(Stream.empty())  
            .collect(toList());  
  
    System.out.println(TAG  
        + "Image(s) filtered = "  
        + filteredImages.size());  
}
```

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

Implementing a Parallel Stream in ImageStreamGang

- We focus on processStream() in ImageStreamParallel.java

```
boolean urlCached(URL url,
                  String filterName) {
    File file =
        new File(getPath(),
                 filterName);

    File imageFile =
        new File(file,
                 getNameForUrl(url));

    return imageFile.exists();
}
```

```
void processStream() {
    List<Image> filteredImages =
        getInput()
            .parallelStream()
            .filter(not(this::urlCached))
            .map(this::blockingDownload)
            .map(this::applyFilters)
            .reduce(Stream::concat)
            .orElse(Stream.empty())
            .collect(toList());

    System.out.println(TAG
        + "Image(s) filtered = "
        + filteredImages.size());
}
```

Check if a file with this name already exists

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

Implementing a Parallel Stream in ImageStreamGang

- We focus on processStream() in ImageStreamParallel.java



```
void processStream() {  
    List<Image> filteredImages =  
        getInput()  
        .parallelStream()  
        .filter(not(this::urlCached))  
        .map(this::blockingDownload)  
        .map(this::applyFilters)  
        .reduce(Stream::concat)  
        .orElse(Stream.empty())  
        .collect(toList());  
  
    System.out.println(TAG  
        + "Image(s) filtered = "  
        + filteredImages.size());  
}
```

There are clearly better ways of implementing an image cache!

Implementing a Parallel Stream in ImageStreamGang

- We focus on processStream() in ImageStreamParallel.java

Return an output stream consisting of the images that were downloaded from the URLs in the input stream



```
void processStream() {  
    List<Image> filteredImages =  
        getInput()  
        .parallelStream()  
        .filter(not(this::urlCached))  
        .map(this::blockingDownload)  
        .map(this::applyFilters)  
        .reduce(Stream::concat)  
        .orElse(Stream.empty())  
        .collect(toList());  
  
    System.out.println(TAG  
        + "Image(s) filtered = "  
        + filteredImages.size());  
}
```

of output stream elements must match the # of input stream elements

Implementing a Parallel Stream in ImageStreamGang

- We focus on processStream() in ImageStreamParallel.java

```
Image blockingDownload
    (URL url) {
    return BlockingTask
        .callInManagedBlock
        (( ) ->
            downloadImage(url));
}
```

*Downloads content from a url
& converts it into an image*

```
void processStream() {
    List<Image> filteredImages =
        getInput()
            .parallelStream()
            .filter(not(this::urlCached))
            .map(this::blockingDownload)
            .map(this::applyFilters)
            .reduce(Stream::concat)
            .orElse(Stream.empty())
            .collect(toList());

    System.out.println(TAG
        + "Image(s) filtered = "
        + filteredImages.size());
}
```

See imagestreamgang.streams/ImageStreamParallel.java

Implementing a Parallel Stream in ImageStreamGang

- We focus on processStream() in ImageStreamParallel.java

```
Image blockingDownload
    (URL url) {
    return BlockingTask
        .callInManagedBlock
            (() ->
                downloadImage(url));
}
```

Uses a "managed blocker" to ensure sufficient threads are in the common fork-join pool

```
void processStream() {
    List<Image> filteredImages =
        getInput()
            .parallelStream()
            .filter(not(this::urlCached))
            .map(this::blockingDownload)
            .map(this::applyFilters)
            .reduce(Stream::concat)
            .orElse(Stream.empty())
            .collect(toList());

    System.out.println(TAG
        + "Image(s) filtered = "
        + filteredImages.size());
}
```

See lesson on "The Java Fork-Join Pool: Applying the ManagedBlocker Interface"

Implementing a Parallel Stream in ImageStreamGang

- We focus on processStream() in ImageStreamParallel.java

```
Image blockingDownload
    (URL url) {
    return BlockingTask
        .callInManagedBlock
            (() ->
                downloadImage(url));
}
```

I/O-bound tasks on an N-core CPU typically run best with $N \cdot (1 + WT/ST)$ threads (WT = wait time & ST = service time)

```
void processStream() {
    List<Image> filteredImages =
        getInput()
            .parallelStream()
            .filter(not(this::urlCached))
            .map(this::blockingDownload)
            .map(this::applyFilters)
            .reduce(Stream::concat)
            .orElse(Stream.empty())
            .collect(toList());

    System.out.println(TAG
        + "Image(s) filtered = "
        + filteredImages.size());
}
```

Implementing a Parallel Stream in ImageStreamGang

- We focus on `processStream()` in `ImageStreamParallel.java`

Return an output stream consisting of the images that were downloaded from the URLs in the input stream

```
void processStream() {  
    List<Image> filteredImages =  
        getInput()  
        .parallelStream()  
        .filter(not(this::urlCached))  
        .map(this::blockingDownload)  
        .map(this::applyFilters)  
        .reduce(Stream::concat)  
        .orElse(Stream.empty())  
        .collect(toList());  
  
    System.out.println(TAG  
        + "Image(s) filtered = "  
        + filteredImages.size());  
}
```

Implementing a Parallel Stream in ImageStreamGang

- We focus on processStream() in ImageStreamParallel.java

Return an output stream-of-streams containing results of applying a list of filters to each image in the input stream & storing results in the file system

```
void processStream() {  
    List<Image> filteredImages =  
        getInput()  
        .parallelStream()  
        .filter(not(this::urlCached))  
        .map(this::blockingDownload)  
        .map(this::applyFilters)  
        .reduce(Stream::concat)  
        .orElse(Stream.empty())  
        .collect(toList());  
  
    System.out.println(TAG  
        + "Image(s) filtered = "  
        + filteredImages.size());  
}
```

Implementing a Parallel Stream in ImageStreamGang

- We focus on `processStream()` in `ImageStreamParallel.java`

Return an output stream-of-streams containing results of applying a list of filters to each image in the input stream & storing results in the file system



```
void processStream() {  
    List<Image> filteredImages =  
        getInput()  
        .parallelStream()  
        .filter(not(this::urlCached))  
        .map(this::blockingDownload)  
        .map(this::applyFilters)  
        .reduce(Stream::concat)  
        .orElse(Stream.empty())  
        .collect(toList());  
  
    System.out.println(TAG  
        + "Image(s) filtered = "  
        + filteredImages.size());  
}
```

of output stream elements must match the # of input stream elements

Implementing a Parallel Stream in ImageStreamGang

- We focus on processStream() in ImageStreamParallel.java

```
Stream<Image> applyFilters
              (Image image) {
    return mFilters
        .parallelStream()
        .map(filter ->
            makeFilterWithImage
            (filter,
             image).run())
}
```

Apply all filters to an image in parallel & store on the device

```
void processStream() {
    List<Image> filteredImages =
        getInput()
            .parallelStream()
            .filter(not(this::urlCached))
            .map(this::blockingDownload)
            .map(this::applyFilters)
            .reduce(Stream::concat)
            .orElse(Stream.empty())
            .collect(toList());

    System.out.println(TAG
        + "Image(s) filtered = "
        + filteredImages.size());
}
```

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

Implementing a Parallel Stream in ImageStreamGang

- We focus on `processStream()` in `ImageStreamParallel.java`

Perform flattening manually by replacing `flatMap()` with `map()` + `reduce(Stream::concat)`



```
void processStream() {  
    List<Image> filteredImages =  
        getInput()  
        .parallelStream()  
        .filter(not(this::urlCached))  
        .map(this::blockingDownload)  
        .map(this::applyFilters)  
        .reduce(Stream::concat)  
        .orElse(Stream.empty())  
        .collect(toList());  
}
```

```
System.out.println(TAG  
    + "Image(s) filtered = "  
    + filteredImages.size());  
}
```

Implementing a Parallel Stream in ImageStreamGang

- We focus on `processStream()` in `ImageStreamParallel.java`

*This idiom works-around
a limitation with `flatMap()`
for parallel streams*

```
void processStream() {  
    List<Image> filteredImages =  
        getInput()  
        .parallelStream()  
        .filter(not(this::urlCached))  
        .map(this::blockingDownload)  
        .map(this::applyFilters)  
        .reduce(Stream::concat)  
        .orElse(Stream.empty())  
        .collect(toList());  
  
    System.out.println(TAG  
        + "Image(s) filtered = "  
        + filteredImages.size());  
}
```

Implementing a Parallel Stream in ImageStreamGang

- We focus on `processStream()` in `ImageStreamParallel.java`

Trigger all the preceding intermediate operations

```
void processStream() {  
    List<Image> filteredImages =  
        getInput()  
            .parallelStream()  
            .filter(not(this::urlCached))  
            .map(this::blockingDownload)  
            .map(this::applyFilters)  
            .reduce(Stream::concat)  
            .orElse(Stream.empty())  
            .collect(toList());  
  
    System.out.println(TAG  
        + "Image(s) filtered = "  
        + filteredImages.size());  
}
```

Implementing a Parallel Stream in ImageStreamGang

- We focus on `processStream()` in `ImageStreamParallel.java`

reduce() returns an Optional, so we need to handle that case, e.g., if all images were cached

```
void processStream() {  
    List<Image> filteredImages =  
        getInput()  
        .parallelStream()  
        .filter(not(this::urlCached))  
        .map(this::blockingDownload)  
        .map(this::applyFilters)  
        .reduce(Stream::concat)  
        .orElse(Stream.empty())  
        .collect(toList());  
  
    System.out.println(TAG  
        + "Image(s) filtered = "  
        + filteredImages.size());  
}
```

Implementing a Parallel Stream in ImageStreamGang

- We focus on `processStream()` in `ImageStreamParallel.java`

collect() is a "reduction" operation that combines elements into one result

```
void processStream() {  
    List<Image> filteredImages =  
        getInput()  
        .parallelStream()  
        .filter(not(this::urlCached))  
        .map(this::blockingDownload)  
        .map(this::applyFilters)  
        .reduce(Stream::concat)  
        .orElse(Stream.empty())  
        .collect(toList());  
  
    System.out.println(TAG  
        + "Image(s) filtered = "  
        + filteredImages.size());  
}
```

See docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#collect

Implementing a Parallel Stream in ImageStreamGang

- We focus on processStream() in ImageStreamParallel.java

It's somewhat unusual to see two terminal operations in one stream!

```
void processStream() {  
    List<Image> filteredImages =  
        getInput()  
        .parallelStream()  
        .filter(not(this::urlCached))  
        .map(this::blockingDownload)  
        .map(this::applyFilters)  
        .reduce(Stream::concat)  
        .orElse(Stream.empty())  
        .collect(toList());  
  
    System.out.println(TAG  
        + "Image(s) filtered = "  
        + filteredImages.size());  
}
```

Implementing a Parallel Stream in ImageStreamGang

- We focus on processStream() in ImageStreamParallel.java

```
void processStream() {  
    List<Image> filteredImages =  
        getInput()  
            .parallelStream()  
            .filter(not(this::urlCached))  
            .map(this::blockingDownload)  
            .map(this::applyFilters)  
            .reduce(Stream::concat)  
            .orElse(Stream.empty())  
            .collect(toList());  
  
    System.out.println(TAG  
        + "Image(s) filtered = "  
        + filteredImages.size());  
}
```

*Create a list containing all
the filtered & stored images*

Implementing a Parallel Stream in ImageStreamGang

- We focus on `processStream()` in `ImageStreamParallel.java`

```
void processStream() {  
    List<Image> filteredImages =  
        getInput()  
        .parallelStream()  
        .filter(not(this::urlCached))  
        .map(this::blockingDownload)  
        .map(this::applyFilters)  
        .reduce(Stream::concat)  
        .orElse(Stream.empty())  
        .collect(toList());  
  
    System.out.println(TAG  
        + "Image(s) filtered = "  
        + filteredImages.size());  
}
```

*Logs the # of images that were
downloaded, filtered, & stored*

End of Learn How to Implement Behaviors in the Java ParallelImageStream Gang Case Study