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 the structure/functionality of the ImageStreamGang app
- Visualize how Java parallel streams are applied to the ImageStreamGang app
- Learn how to implement parallel streams behaviors of ImageStreamGang

```java
void processStream() {
    List<URL> urls = getInput();
    List<Image> filteredImages = urls
        .parallelStream()
        .filter(not(this::urlCached))
        .map(this::blockingDownload)
        .flatMap(this::applyFilters)
        .collect(toList());

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

Implementing a Parallel Stream in ImageStreamGang
Implementing a Parallel Stream in ImageStreamGang

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

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

    List<Image> filteredImages = urls
        .parallelStream()
        .filter(not(this::urlCached))
        .map(this::blockingDownload)
        .flatMap(this::applyFilters)
        .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

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

    List<Image> filteredImages = urls
        .parallelStream()
        .filter(not(this::urlCached))
        .map(this::blockingDownload)
        .flatMap(this::applyFilters)
        .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`

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

    List<Image> filteredImages = urls
        .parallelStream()
        .filter(not(this::urlCached))
        .map(this::blockingDownload)
        .flatMap(this::applyFilters)
        .collect(toList());

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

*Convert a collection into a parallel stream*
Implementing a Parallel Stream in ImageStreamGang

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

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

    List<Image> filteredImages = urls
        .parallelStream()
        .filter(not(this::urlCached))
        .map(this::blockingDownload)
        .flatMap(this::applyFilters)
        .collect(toList());

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

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

See docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#filter
Implementing a Parallel Stream in ImageStreamGang

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

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

    List<Image> filteredImages = urls
        .parallelStream()
        .filter(not(this::urlCached))
        .map(this::blockingDownload)
        .flatMap(this::applyFilters)
        .collect(toList());

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

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

# of output stream elements will be <= # of input stream elements
Implementing a Parallel Stream in ImageStreamGang

- We focus on processStream() in ImageStreamParallel.java

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

    List<Image> filteredImages = urls.stream()
        .parallelStream()
        .filter(not(this::urlCached))
        .map(this::blockingDownload)
        .flatMap(this::applyFilters)
        .collect(toList());

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

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

See imagestreamgang/streams/ImageStreamGang.java
Implementing a Parallel Stream in ImageStreamGang

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

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

    List<Image> filteredImages = urls
        .parallelStream()
        .filter(not(this::urlCached))
        .map(this::blockingDownload)
        .flatMap(this::applyFilters)
        .collect(toList());

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

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

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

    return imageFile.exists();
}
```

Check if a file with this name already exists

See `imagestreamgang/streams/ImageStreamGang.java`
Implementing a Parallel Stream in ImageStreamGang

• We focus on processStream() in ImageStreamParallel.java

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

    List<Image> filteredImages = urls
        .parallelStream()
        .filter(not(this::urlCached))
        .map(this::blockingDownload)
        .flatMap(this::applyFilters)
        .collect(toList());

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

See imagestreamgang/streams/ImageStreamGang.java

There are clearly better ways of implementing an image cache!
Implementing a Parallel Stream in ImageStreamGang

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

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

    List<Image> filteredImages = urls.parallelStream()
        .filter(not(this::urlCached))
        .map(this::blockingDownload)
        .flatMap(this::applyFilters)
        .collect(toList());

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

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

See docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#map
Implementing a Parallel Stream in ImageStreamGang

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

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

    List<Image> filteredImages = urls
        .parallelStream()
        .filter(not(this::urlCached))
        .map(this::blockingDownload)
        .flatMap(this::applyFilters)
        .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`

```java
void processStream() {
    List<URL> urls = getInput();
    List<Image> filteredImages = urls.parallelStream()
        .filter(not(this::urlCached))
        .map(this::blockingDownload)
        .flatMap(this::applyFilters)
        .collect(toList());

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

**Image blockingDownload(URL url)**

```java
    return BlockingTask.callInManagedBlock(() ->
        downloadImage(url));
}
```

Downloads content from a url & converts it into an image

See `imagestreamgang/streams/ImageStreamStreamParallel.java`
Implementing a Parallel Stream in ImageStreamGang

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

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

```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

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

```java
void processStream() {
    List<URL> urls = getInput();
    List<Image> filteredImages = urls
        .parallelStream()
        .filter(not(this::urlCached))
        .map(this::blockingDownload)
        .flatMap(this::applyFilters)
        .collect(toList());

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

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

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

Implementing a Parallel Stream in ImageStreamGang

- We focus on processStream() in ImageStreamParallel.java

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

    List<Image> filteredImages = urls
        .parallelStream()
        .filter(not(this::urlCached))
        .map(this::blockingDownload)
        .flatMap(this::applyFilters)
        .collect(toList());

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

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

See docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#flatMap
Implementing a Parallel Stream in ImageStreamGang

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

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

    List<Image> filteredImages = urls
        .parallelStream()
        .filter(not(this::urlCached))
        .map(this::blockingDownload)
        .flatMap(this::applyFilters)
        .collect(toList());

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

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

# of output stream elements may differ from the # of input stream elements
Implementing a Parallel Stream in ImageStreamGang

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

```java
void processStream() {
    List<URL> urls = getInput();
    List<Image> filteredImages = urls
        .parallelStream()
        .filter(not(this::urlCached))
        .map(this::blockingDownload)
        .flatMap(this::applyFilters)
        .collect(toList());

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

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

See imagestreamgang/streams/ImageStreamParallel.java

Apply all filters to an image in parallel & store on the device
Implementing a Parallel Stream in ImageStreamGang

• We focus on processStream() in ImageStreamParallel.java

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

    List<Image> filteredImages = urls
        .parallelStream()
        .filter(not(this::urlCached))
        .map(this::blockingDownload)
        .flatMap(this::applyFilters)
        .collect(toList());

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

*collect()* is a “`reduction`” operation that combines elements into one result

See [docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#collect](https://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

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

    List<Image> filteredImages = urls
        .parallelStream()
        .filter(not(this::urlCached))
        .map(this::blockingDownload)
        .flatMap(this::applyFilters)
        .collect(toList());

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

`Trigger all intermediate operations`
Implementing a Parallel Stream in ImageStreamGang

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

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

    List<Image> filteredImages = urls.parallelStream()
        .filter(not(this::urlCached))
        .map(this::blockingDownload)
        .flatMap(this::applyFilters)
        .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<URL> urls = getInput();

    List<Image> filteredImages = urls
        .parallelStream()
        .filter(not(this::urlCached))
        .map(this::blockingDownload)
        .flatMap(this::applyFilters)
        .collect(toList());

    System.out.println(TAG + "Image(s) filtered = " + filteredImages.size());
}
```
End of Learn How to Implement Behaviors in the Java ParallelImageStream Gang Case Study