Learning Objectives in this Part of the Lesson

- Understand parallel stream internals, e.g.
  - Know what can change & what can’t
  - Partition a data source into “chunks”
  - Process chunks in parallel via the common fork-join pool

- Configure the Java parallel stream common fork-join pool
  - Know the performance impact of configuring the common fork-join pool size

Entering the test program with 12 cores
- ex20: testDefaultDownloadBehavior() downloaded and stored 42 images using 12 threads in the pool
- ex20: testAdaptiveMBDownloadBehavior() downloaded and stored 42 images using 43 threads in the pool
- ex20: testAdaptiveBTDownloadBehavior() downloaded and stored 42 images using 43 threads in the pool

Printing 3 results from fastest to slowest
- testAdaptiveBTDownloadBehavior() executed in 3598 msecs
- testAdaptiveMBDownloadBehavior() executed in 3910 msecs
- testDefaultDownloadBehavior() executed in 4104 msecs

Leaving the test program

See [github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex20](https://github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex20)
Demo’ing Impact of Configuring Common Fork-Join Pool
Demo’ing Impact of Configuring Common Fork-Join Pool

• The common fork-join pool size can be controlled programmatically

See prior lesson on “Java Parallel Stream Internals: Configuring the Common Fork-Join Pool”
Demo'ing Impact of Configuring Common Fork-Join Pool

- The common fork-join pool size can be controlled programmatically
- This demo applies the Managed Blocker interface to adaptively add new worker threads to the Java common fork-join pool

```java
File downloadAndStoreImageMB(URL url) {
  final Image[] image = new Image[1];
  ...
  ForkJoinPool
    .managedBlock(new ForkJoinPool
      .ManagedBlocker() {
      public boolean block() {
        image[0] =
          downloadImage(url);
        return true;
      } ... });

  return image[0].store(); ...
}
```

This program shows the performance difference of using ManagedBlocker versus not using ManagedBlocker for an I/O-intensive app.

```java
void testDownloadBehavior(Function<URL, File> downloadAndStoreImage, String testName) {
...
List<File> imageFiles = Options.instance()
    .getUrlList()
    .parallelStream()
    .map(downloadAndStoreImage)
    .collect(Collectors.toList());
printStats(testName, imageFiles.size()); ...
```
Demo’ing Impact of Configuring Common Fork-Join Pool

- This program shows the performance difference of using ManagedBlocker versus not using ManagedBlocker for an I/O-intensive app

```java
void testDownloadBehavior(Function<URL, File>
downloadAndStoreImage,

String testName) {

...
List<File> imageFiles = Options.instance()
  .getUrlList()
  .parallelStream()
  .map(downloadAndStoreImage)
  .collect(Collectors.toList());

printStats(testName, imageFiles.size()); ...
```

This function param is used to pass different strategies for downloading & storing images from remote websites.

See en.wikipedia.org/wiki/Strategy_pattern
• Results show increasing worker threads in the pool improves performance

Entering the test program with 12 cores
ex20: testDefaultDownloadBehavior() downloaded and stored 42 images using 12 threads in the pool
ex20: testAdaptiveMBDownloadBehavior() downloaded and stored 42 images using 43 threads in the pool
ex20: testAdaptiveBTDownloadBehavior() downloaded and stored 42 images using 43 threads in the pool

Printing 3 results from fastest to slowest
testAdaptiveBTDownloadBehavior() executed in 3598 msecs
testAdaptiveMBDownloadBehavior() executed in 3910 msecs
testDefaultDownloadBehavior() executed in 4104 msecs

Leaving the test program

See upcoming lessons on “The Java Fork-Join Pool: the ManagedBlocker Interface”
Demo’ing Impact of Configuring Common Fork-Join Pool

```java
// Warm up the common fork-join pool.
warmUpThreadPool();

// Runs the tests using the using the Java fork-join
// framework’s default behavior, which does not add new worker
// threads to the pool when blocking occurs.
runTest(this::downloadAndStoreImage,
        testName: "testDefaultDownloadBehavior();");

// Run the tests using the using the BlockingTask wrapper
// around the Java fork-join framework’s ManagedBlocker
// mechanism, which adds new worker threads to the pool
// adaptively when blocking occurs.
runTest(this::downloadAndStoreImageMB,
        testName: "testAdaptiveMBDownloadBehavior();");

// Run the tests using the using the BlockingTask wrapper for
// the Java fork-join framework’s ManagedBlocker mechanism,
// which adds new worker threads to the pool adaptively when
// blocking occurs.
runTest(this::downloadAndStoreImageBT,
        testName: "testAdaptiveBTDownloadBehavior();");

// Print the results.
```

See github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex20
End of Java Parallel Streams Internals: Demo’ing How to Configure the Common Fork-Join Pool