The FileCount Case Study: Overview

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

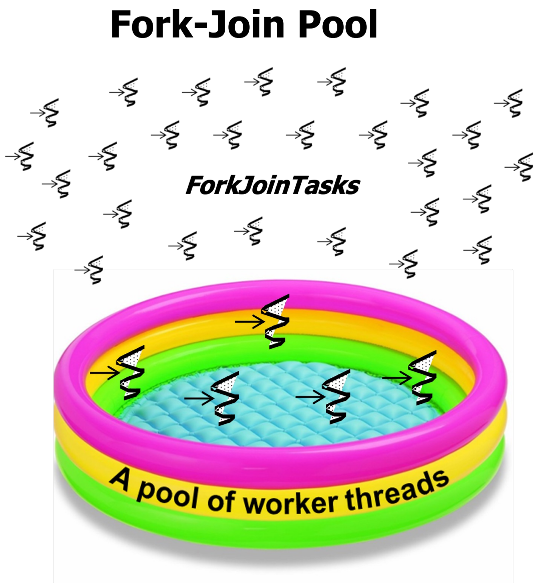
Professor of Computer Science

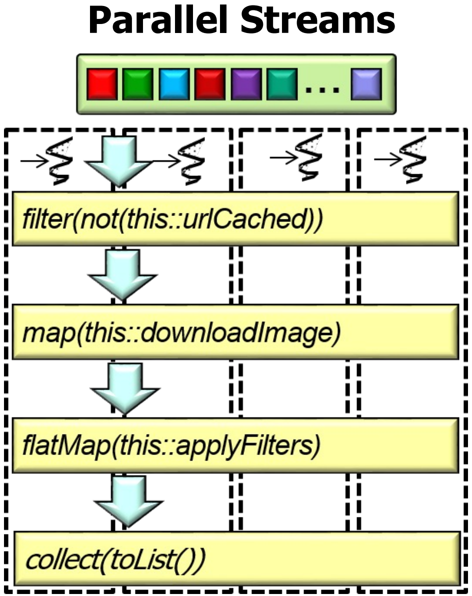
Institute for Software
Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

- Understand the design of the FileCounter case study
  - Evaluates different Java parallel programming models in practice

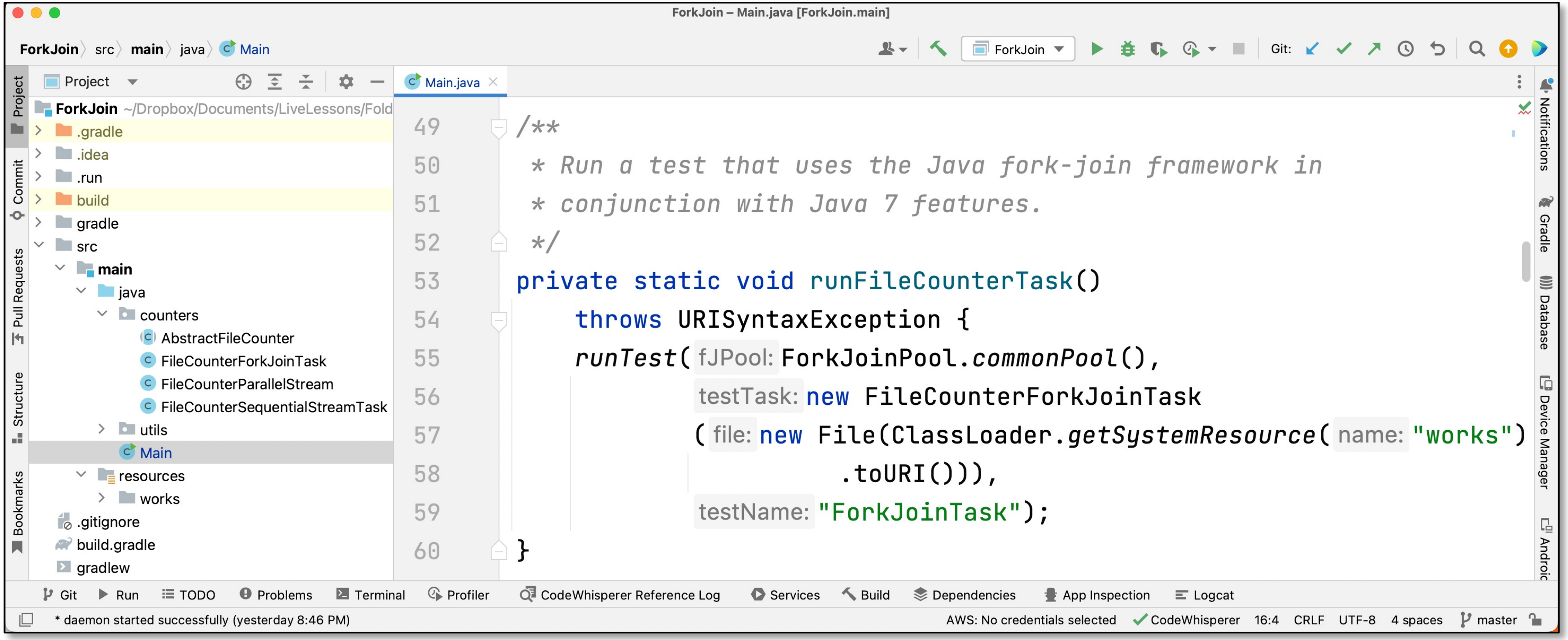




Overview of the FileCounter Case Study

Overview of the FileCounter Case Study

- Different Java parallel programming models are applied on common data & benchmarked to determine tradeoffs between conciseness & performance

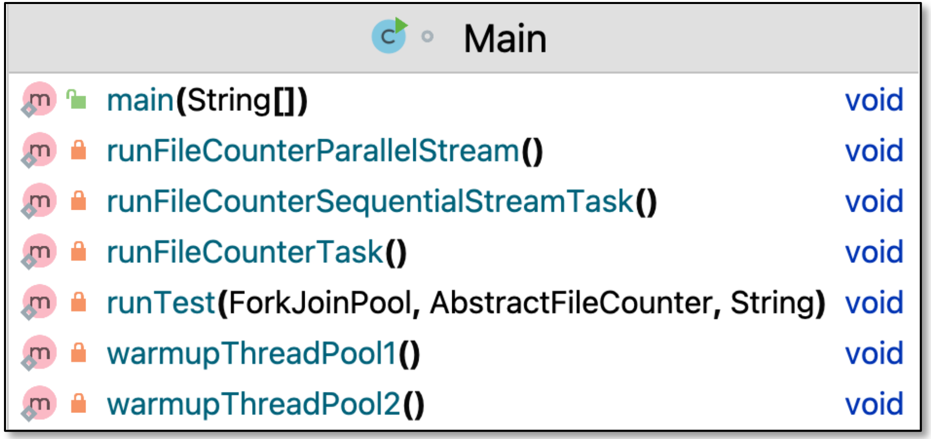


```java
/**
 * Run a test that uses the Java fork-join framework in
 * conjunction with Java 7 features.
 */
private static void runFileCounterTask()
    throws URISyntaxException {
    runTest( fJPool: ForkJoinPool.commonPool(),
             testTask: new FileCounterForkJoinTask
             ( file: new File(ClassLoader.getSystemResource( name: "works")
                          .toURI())),
             testName: "ForkJoinTask");
}
```

See github.com/douglascraigschmidt/LiveLessons/tree/master/Folders/ForkJoin

Overview of the FileCounter Case Study

- Different Java parallel programming models are applied on common data & benchmarked to determine tradeoffs between conciseness & performance
  - **`Main`**
    - Evaluates three Java parallel programming models
      - e.g., fork-join framework & sequential/parallel streams



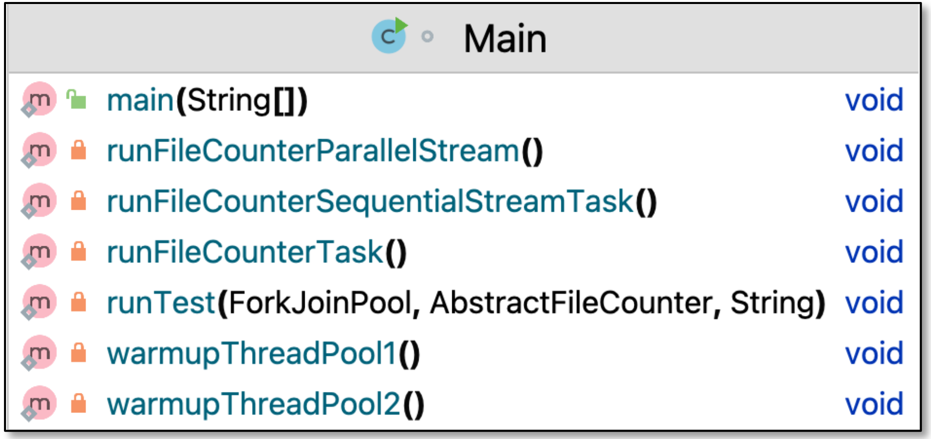
Main

main(String[])	void
runFileCounterParallelStream()	void
runFileCounterSequentialStreamTask()	void
runFileCounterTask()	void
runTest(ForkJoinPool, AbstractFileCounter, String)	void
warmupThreadPool1()	void
warmupThreadPool2()	void

See Folders/ForkJoin/src/main/java/Main.java

Overview of the FileCounter Case Study

- Different Java parallel programming models are applied on common data & benchmarked to determine tradeoffs between conciseness & performance
  - **`Main`**
    - Evaluates three Java parallel programming models
    - Counts all the files in a recursive folder hierarchy & calculates cumulative sizes of files

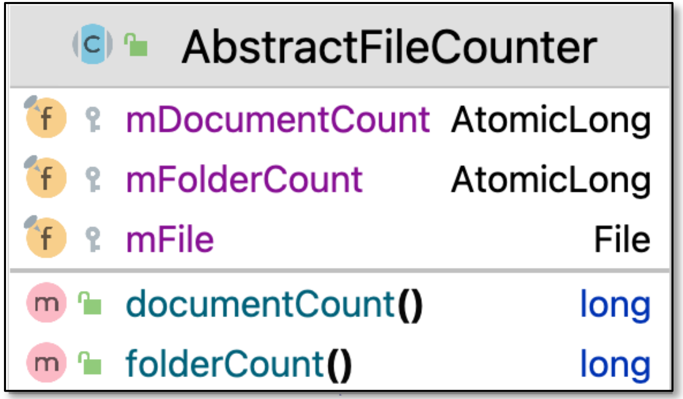


Main	
main(String[])	void
runFileCounterParallelStream()	void
runFileCounterSequentialStreamTask()	void
runFileCounterTask()	void
runTest(ForkJoinPool, AbstractFileCounter, String)	void
warmupThreadPool1()	void
warmupThreadPool2()	void

See Folders/ForkJoin/src/main/java/Main.java

Overview of the FileCounter Case Study

- Different Java parallel programming models are applied on common data & benchmarked to determine tradeoffs between conciseness & performance
  - **`AbstractFileCounter`**
    - Provides foundational functionality for subclasses that compute the size of files in folders



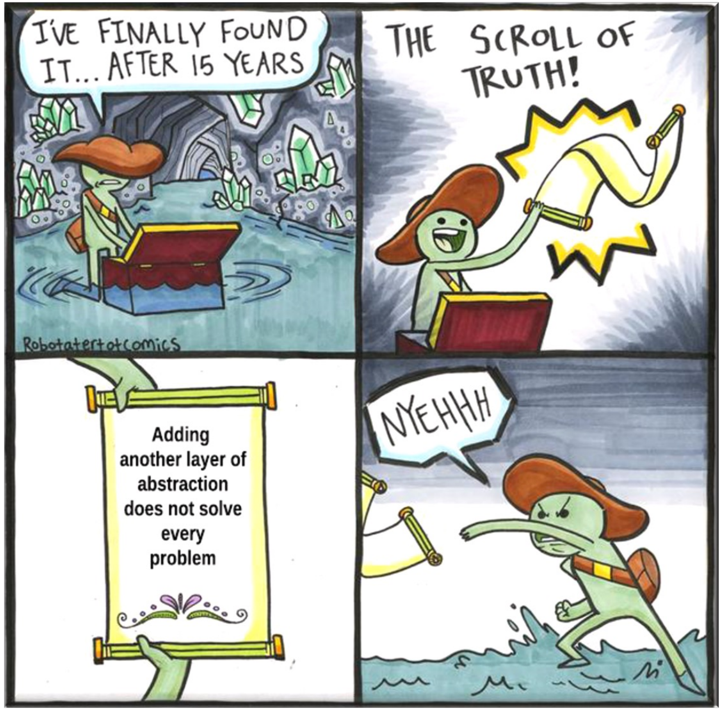
AbstractFileCounter

mDocumentCount	AtomicLong
mFolderCount	AtomicLong
mFile	File
documentCount()	long
folderCount()	long

See Folders/ForkJoin/src/main/java/counters/AbstractFileCounter.java

Overview of the FileCounter Case Study

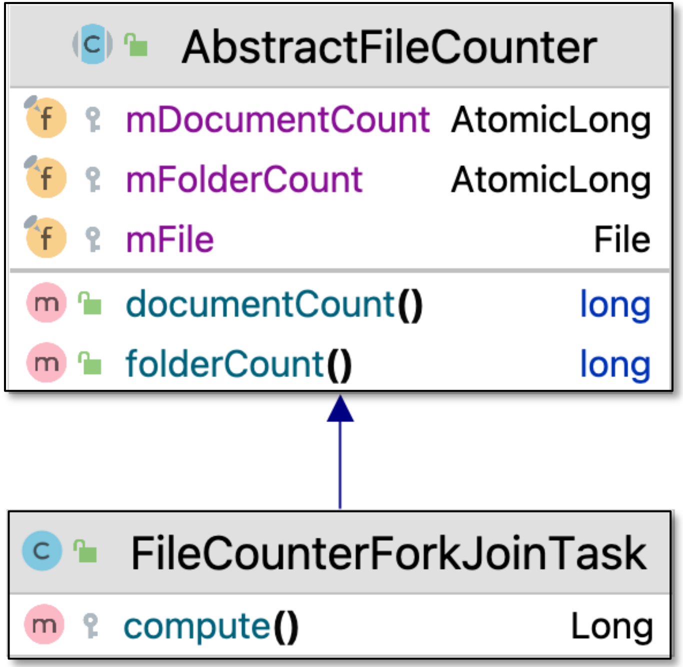
- Different Java parallel programming models are applied on common data & benchmarked to determine tradeoffs between conciseness & performance
  - **`AbstractFileCounter`**
    - Provides foundational functionality for subclasses that compute the size of files in folders
    - This "abstraction layer" offers common methods & fields shared among the various FileCounter implementations


I'VE FINALLY FOUND IT... AFTER 15 YEARS
THE SCROLL OF TRUTH!
Adding another layer of abstraction does not solve every problem
NYEHHH
Robotatertotcomics

See en.wikipedia.org/wiki/Abstraction_layer

Overview of the FileCounter Case Study

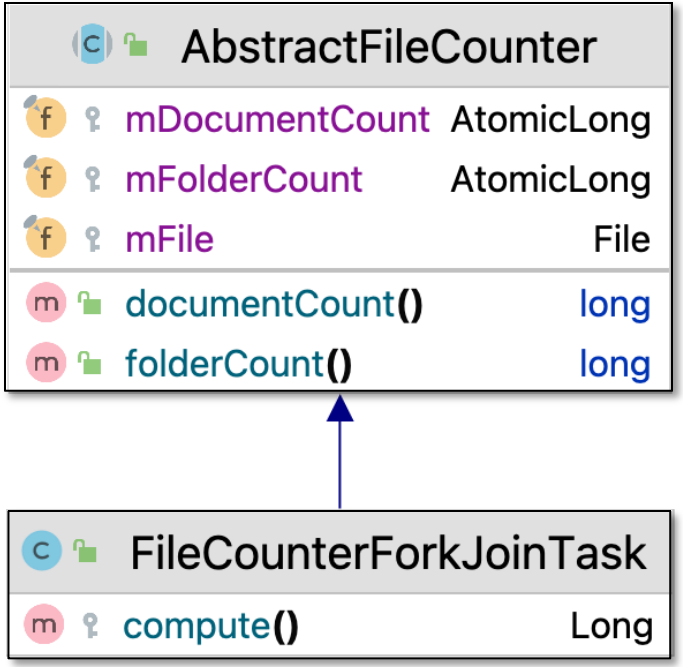
- Different Java parallel programming models are applied on common data & benchmarked to determine tradeoffs between conciseness & performance
  - **`FileCounterForkJoinTask`**
    - Applies the Java fork-join framework & Java 7 features to compute size of a folder & all reachable files



See Folders/ForkJoin/src/main/java/counters/FileCounterForkJoinTask.java

Overview of the FileCounter Case Study

- Different Java parallel programming models are applied on common data & benchmarked to determine tradeoffs between conciseness & performance
  - **`FileCounterForkJoinTask`**
    - Applies the Java fork-join framework & Java 7 features to compute size of a folder & all reachable files
    - Best used for recursive tasks that can be split into smaller sub-tasks
      - i.e., divide-and-conquer algorithms



AbstractFileCounter

mDocumentCount AtomicLong

mFolderCount AtomicLong

mFile File

documentCount() long

folderCount() long

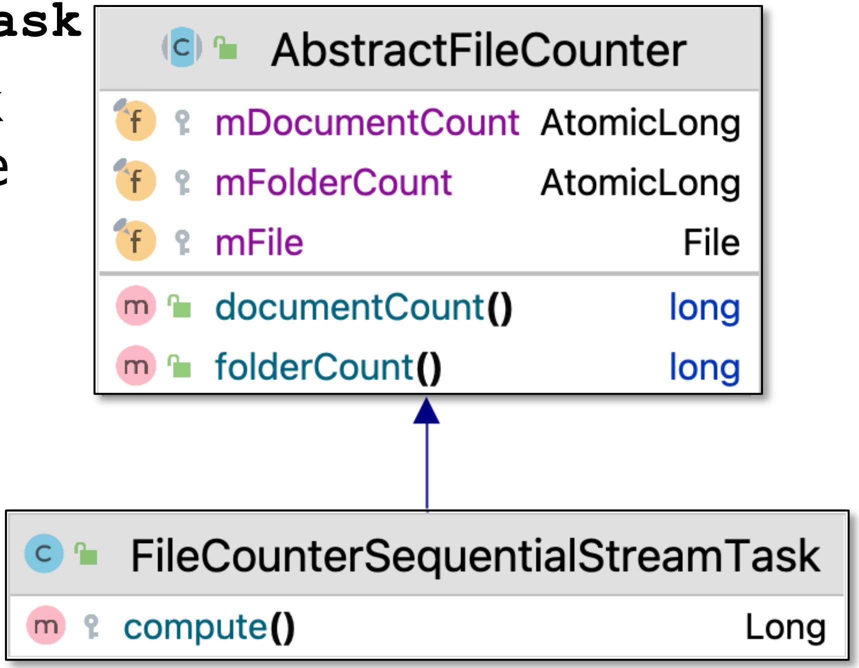
FileCounterForkJoinTask

compute() Long

See en.wikipedia.org/wiki/Divide-and-conquer_algorithm

Overview of the FileCounter Case Study

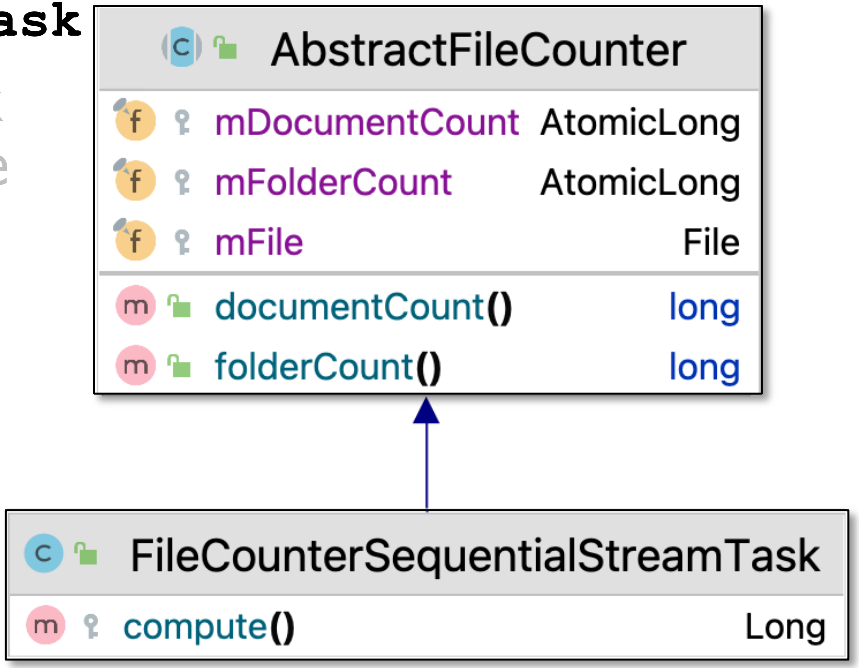
- Different Java parallel programming models are applied on common data & benchmarked to determine tradeoffs between conciseness & performance
  - **`FileCounterSequentialStreamTask`**
    - Applies the Java fork-join framework & sequential streams to compute the size of a folder & all reachable files



See Folders/ForkJoin/src/main/java/counters/FileCounterSequentialStreamTask.java

Overview of the FileCounter Case Study

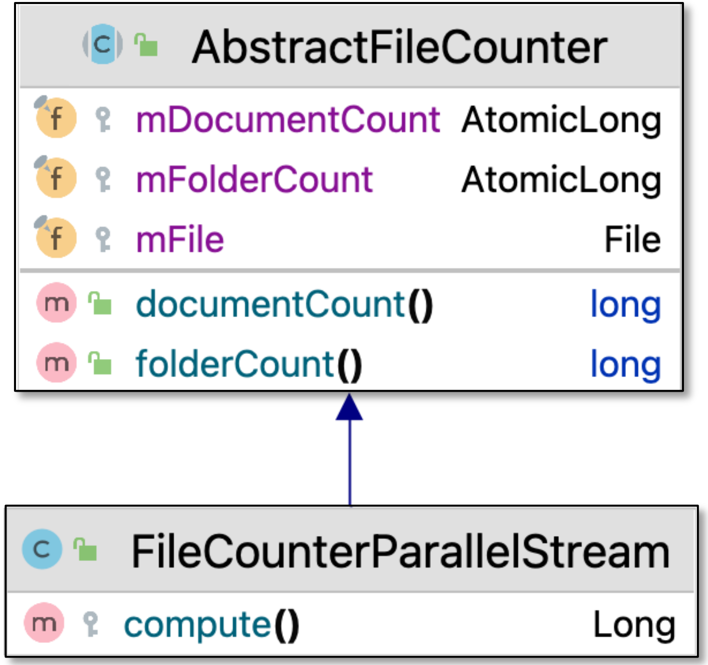
- Different Java parallel programming models are applied on common data & benchmarked to determine tradeoffs between conciseness & performance
  - **`FileCounterSequentialStreamTask`**
    - Applies the Java fork-join framework & sequential streams to compute the size of a folder & all reachable files
    - Best used when the simplicity of streams is desired along with the control of managing parallelism using the fork-join framework



See www.oracle.com/technical-resources/articles/java/ma14-java-se-8-streams.html

Overview of the FileCounter Case Study

- Different Java parallel programming models are applied on common data & benchmarked to determine tradeoffs between conciseness & performance
  - **`FileCounterParallelStream`**
    - Applies Java parallel streams to compute size of a folder & all reachable files



AbstractFileCounter

mDocumentCount AtomicLong

mFolderCount AtomicLong

mFile File

documentCount() long

folderCount() long

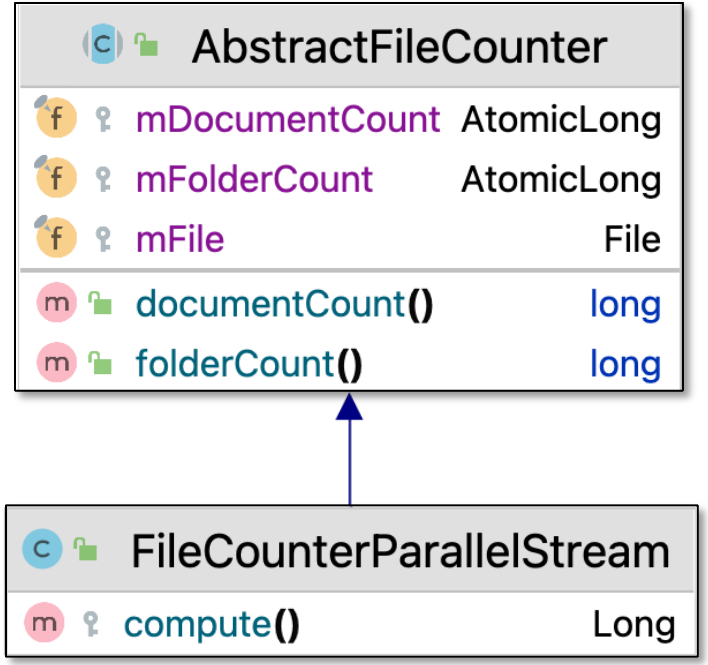
FileCounterParallelStream

compute() Long

See Folders/ForkJoin/src/main/java/counters/FileCounterParallelStream.java

Overview of the FileCounter Case Study

- Different Java parallel programming models are applied on common data & benchmarked to determine tradeoffs between conciseness & performance
  - **`FileCounterParallelStream`**
    - Applies Java parallel streams to compute size of a folder & all reachable files
    - Best used when data-level parallelism is desired, especially when working with collections
      - Parallel streams abstract away low-level threading details



AbstractFileCounter

mDocumentCount	AtomicLong
mFolderCount	AtomicLong
mFile	File
documentCount()	long
folderCount()	long

FileCounterParallelStream

compute()	Long

See www.baeldung.com/java-when-to-use-parallel-stream

End of the FileCounter Case Study: Overview