The FileCount Case Study: Performance & Evaluation

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

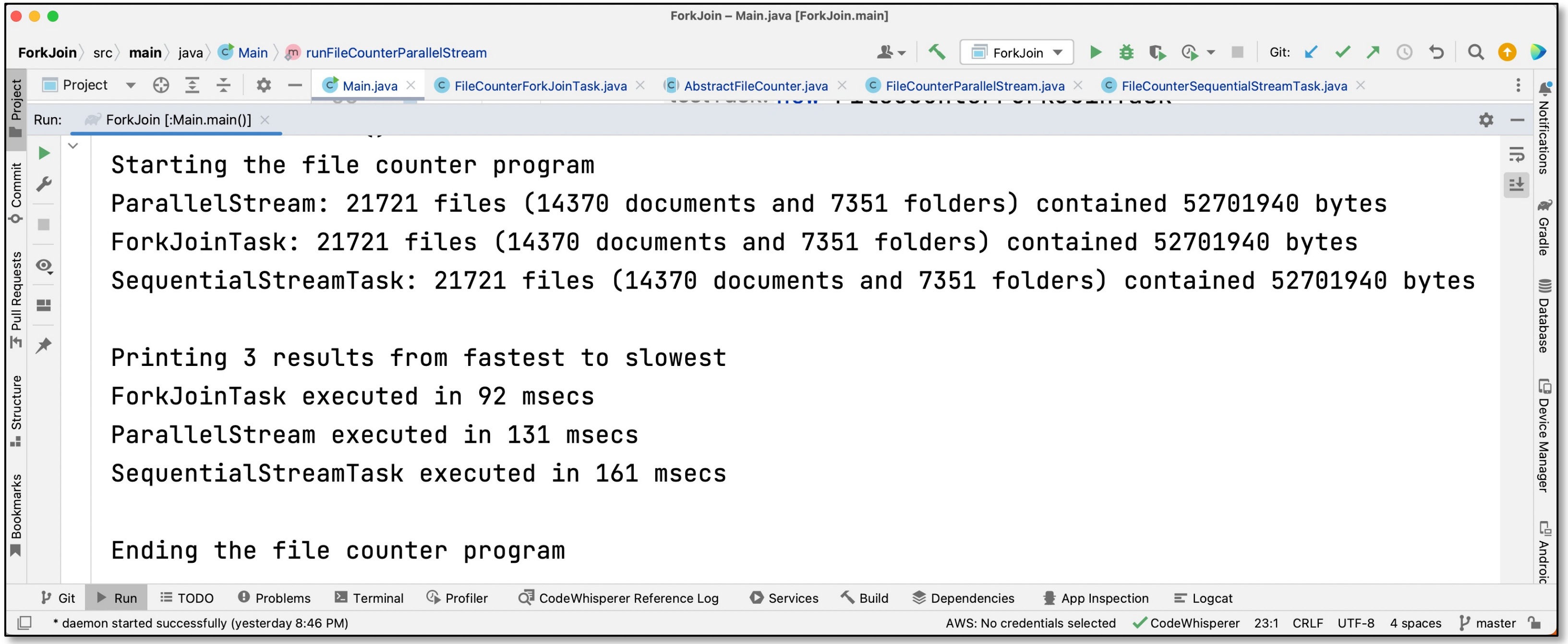
Professor of Computer Science

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
- Walkthrough the program implementation
- Benchmark the performance & evaluate the results


ForkJoin – Main.java [ForkJoin.main]
ForkJoin › src › main › java › Main › runFileCounterParallelStream
Main.java
FileCounterForkJoinTask.java
AbstractFileCounter.java
FileCounterParallelStream.java
FileCounterSequentialStreamTask.java
Run: ForkJoin [:Main.main()]
Git Run TODO Problems Terminal Profiler CodeWhisperer Reference Log Services Build Dependencies App Inspection Logcat
* daemon started successfully (yesterday 8:46 PM)
AWS: No credentials selected CodeWhisperer 23:1 CRLF UTF-8 4 spaces master

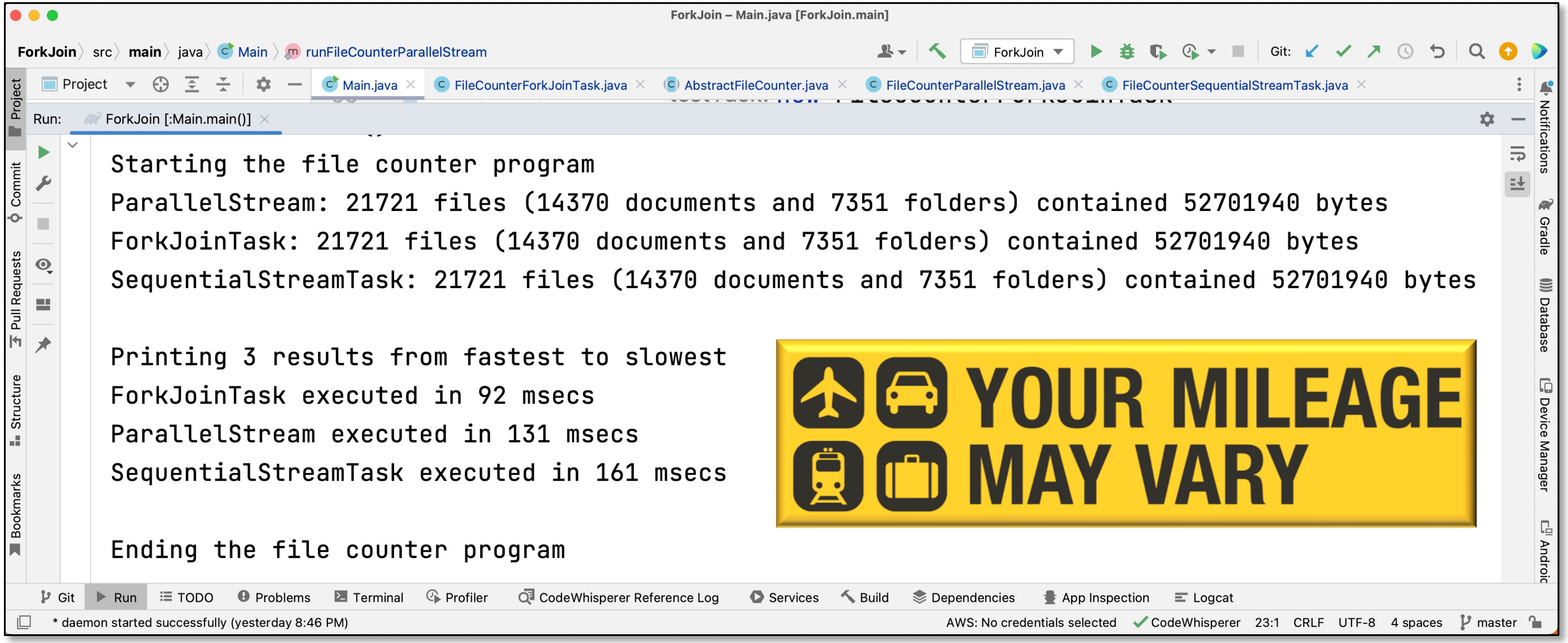
```
Starting the file counter program
ParallelStream: 21721 files (14370 documents and 7351 folders) contained 52701940 bytes
ForkJoinTask: 21721 files (14370 documents and 7351 folders) contained 52701940 bytes
SequentialStreamTask: 21721 files (14370 documents and 7351 folders) contained 52701940 bytes

Printing 3 results from fastest to slowest
ForkJoinTask executed in 92 msecs
ParallelStream executed in 131 msecs
SequentialStreamTask executed in 161 msecs

Ending the file counter program
```

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

Benchmarking the FileCounter Case Study

Benchmarking the FileCounter Case Study

- The benchmark results on my 10-core 64GB MacBook Pro are interesting, though your mileage may vary


ForkJoin – Main.java [ForkJoin.main]
ForkJoin src main java Main runFileCounterParallelStream
Main.java FileCounterForkJoinTask.java AbstractFileCounter.java FileCounterParallelStream.java FileCounterSequentialStreamTask.java
Run: ForkJoin [:Main.main()]
Git Run TODO Problems Terminal Profiler CodeWhisperer Reference Log Services Build Dependencies App Inspection Logcat
* daemon started successfully (yesterday 8:46 PM) AWS: No credentials selected CodeWhisperer 23:1 CRLF UTF-8 4 spaces master

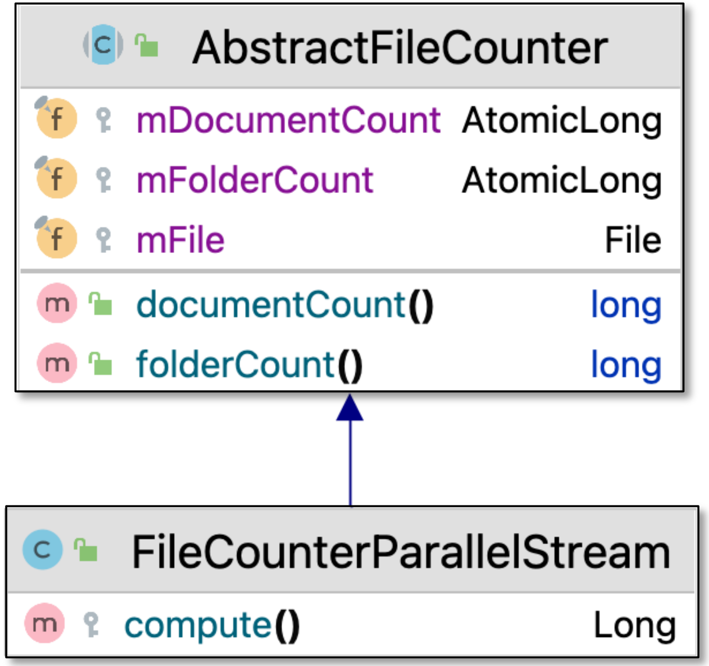
```
Starting the file counter program
ParallelStream: 21721 files (14370 documents and 7351 folders) contained 52701940 bytes
ForkJoinTask: 21721 files (14370 documents and 7351 folders) contained 52701940 bytes
SequentialStreamTask: 21721 files (14370 documents and 7351 folders) contained 52701940 bytes

Printing 3 results from fastest to slowest
ForkJoinTask executed in 92 msecs
ParallelStream executed in 131 msecs
SequentialStreamTask executed in 161 msecs

Ending the file counter program
```

See en.wiktionary.org/wiki/your_mileage_may_vary

Evaluating the Various Java Parallel Programming Models

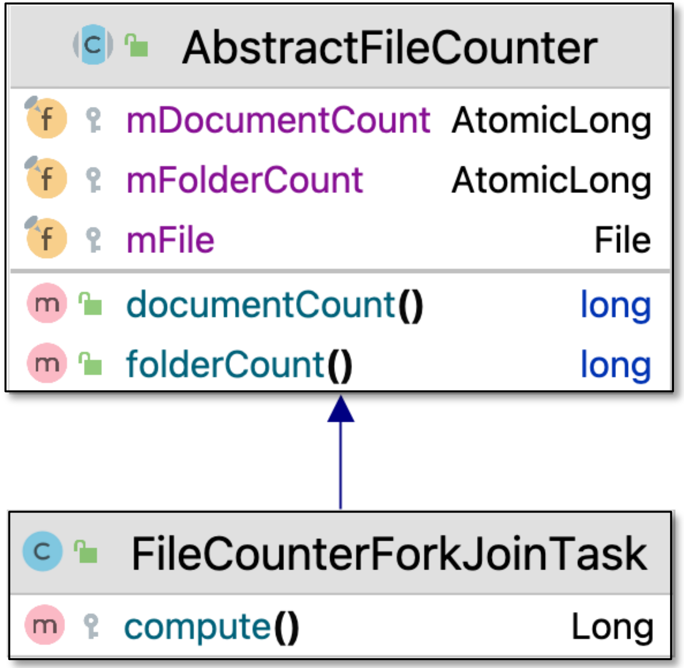
Evaluating the Various Java Parallel Programming Models

- If the goal is to simplify parallel processing without much concern for fine-grained control, the Java Parallel Streams model is a good choice



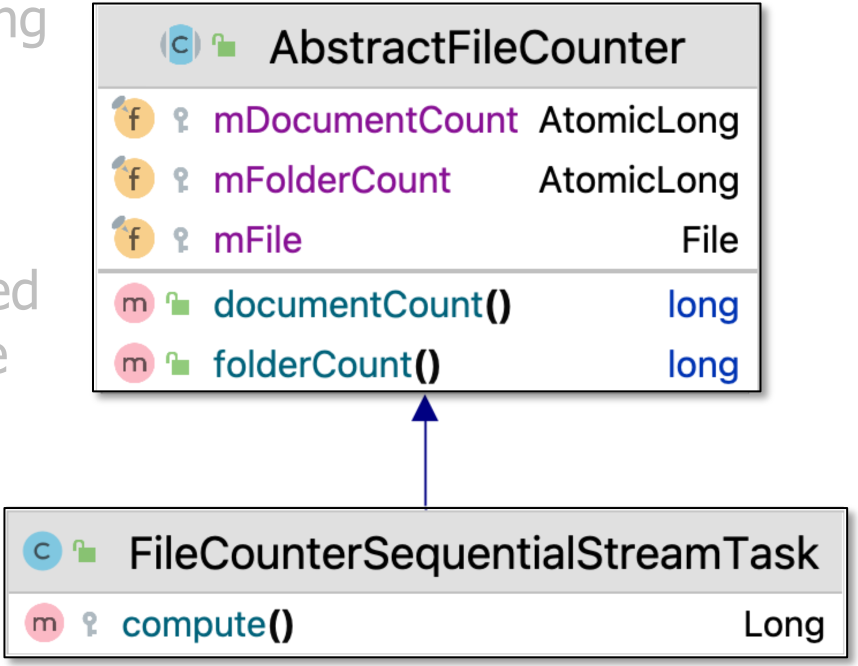
Evaluating the Various Java Parallel Programming Models

- If the goal is to simplify parallel processing without much concern for fine-grained control, the Java Parallel Streams model is a good choice
- For recursive tasks or when there's a need for more control over the parallelism, the Java Fork-Join model is suitable
  - Also doesn't require any modern Java features/JDK/JRE



Evaluating the Various Java Parallel Programming Models

- If the goal is to simplify parallel processing without much concern for fine-grained control, the Java Parallel Streams model is a good choice
- For recursive tasks or when there's a need for more control over the parallelism, the Java Fork-Join model is suitable
- When a blend of simplicity & control is desired, the combining Sequential Streams with Fork-Join is a balanced approach



End of the FileCount Case Study: FileCounterParallelStream