Simple Concurrency in Java

Paul Bauer

Colorado State University Global

CSC450: Programming III

Reginald Haseltine

December 3, 2023

Simple Concurrency in Java

For this project I have built a simple java program demonstrating basic concurrency concepts including launching threads using executor service submit function, using CountDownLatch.await() to wait for resources to finish, and use of the Runnable interface. A common performance concern with concurrency is ensuring that multiple threads remain in sync with each other and execute in the correct order. The threads in this project are coordinated using the CountDownLatch class which allows for a counter to be made available to multiple threads so that it can be used to count how many threads are currently in progress. This ability to count threads enables us to manage and coordinate multiple tasks. The CountDownLatch is created in the main thread and set with a number of threads, then in each thread when the thread exits the counter is decreased. If this program were handling sensitive information like password and username for users then it would be important to manage memory usage as well and make sure that strings are properly cleared before exiting each thread so that information is not left in memory where it could be maliciously retrieved.

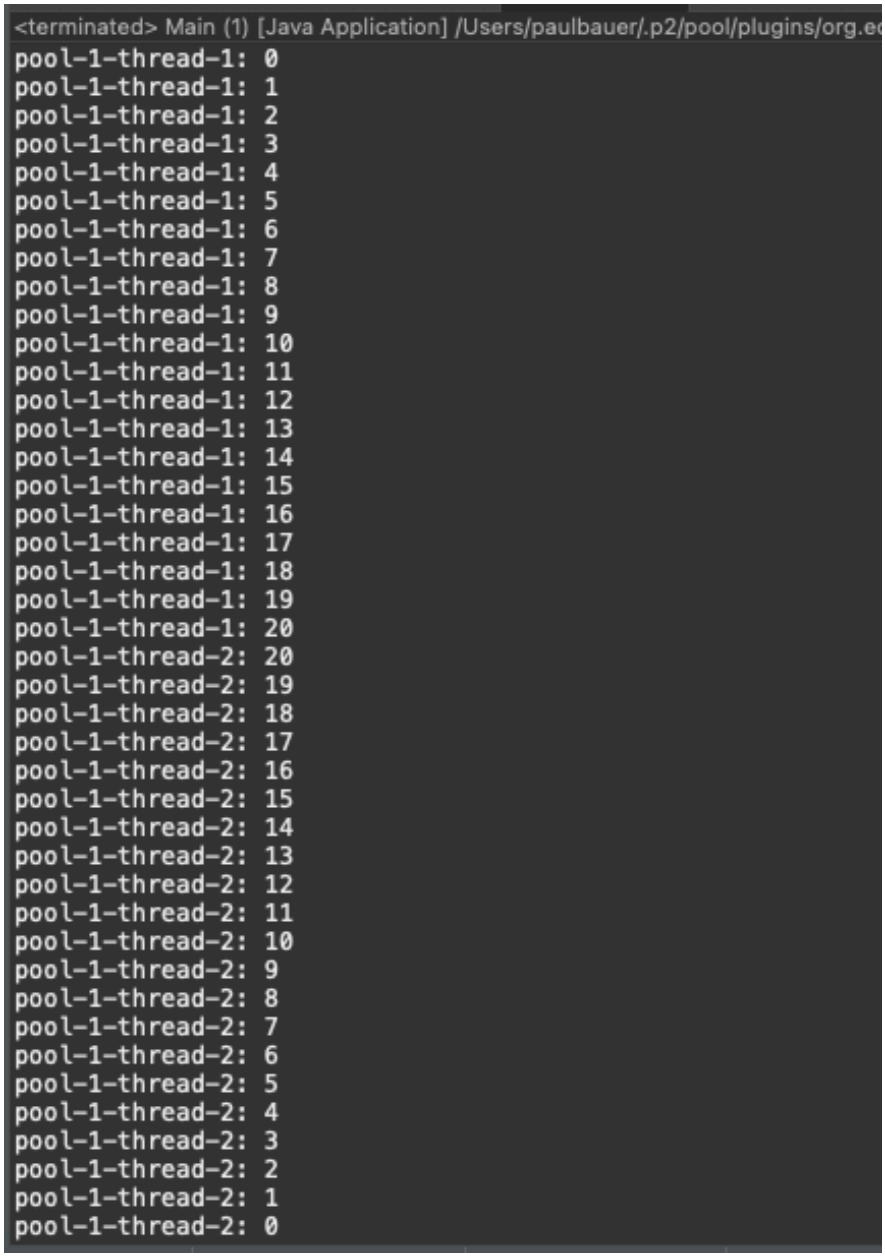
module-info.java | Main.java | CountingRunnable.java

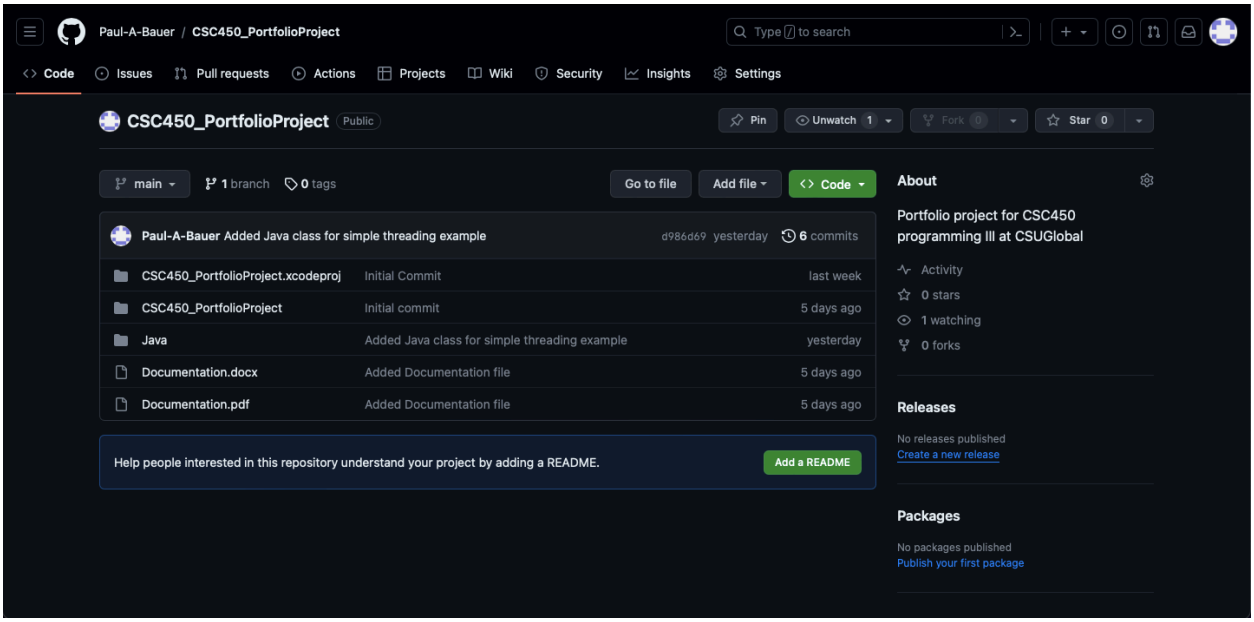
```java
import java.util.concurrent.CountDownLatch;

public class Main {

    public static void main(String[] args) {
        //Set number of threads to be used
        int totalThreads = 2;

        // Create an ExecutorService with a fixed thread pool of size 2
        ExecutorService executorService = Executors.newFixedThreadPool(totalThreads);
        CountDownLatch latch = new CountDownLatch(1);

        // Submit the first thread for execution
        executorService.submit(new CountingRunnable(latch, 0, 20));

        try {
            latch.await();
        } catch (InterruptedException e) {
            // Print messagef
            e.printStackTrace();
        }

        // Submit the second thread for execution
        executorService.submit(new CountingRunnable(latch, 20, 0));

        // Shut down the ExecutorService
        executorService.shutdown();

    }

}
```

```java
package com.CSC450_PortfolioProject;

import java.util.concurrent.CountDownLatch;

public class CountingRunnable implements Runnable {

    int start = 0;
    int end = 0;
    int increment = 1;
    CountDownLatch latch;

    public CountingRunnable(CountDownLatch latch) {
        super();
        this.latch = latch;
    }

    public CountingRunnable(CountDownLatch latch, int start, int end) {
        super();
        this.start = start;
        this.end = end;
        this.latch = latch;
    }

    public CountingRunnable(CountDownLatch latch, int start, int end, int increment) {
        super();
        this.start = start;
        this.end = end;
        this.increment = increment;
        this.latch = latch;
    }

    @Override
    public void run() {

        //set increment
        if(start > end && increment > 0) {
            increment *= -1;
        }

        // Count
        for (int i = start; i != (end + increment); i+=increment) {
            System.out.println(Thread.currentThread().getName() + ": " + i);
            try {
```

```java
    @Override
    public void run() {

        //set increment
        if(start > end && increment > 0) {
            increment *= -1;
        }

        // Count
        for (int i = start; i != (end + increment); i+=increment) {
            System.out.println(Thread.currentThread().getName() + ": " + i);
            try {
                Thread.sleep(1000); // Sleep for 1 second to simulate work
            } catch (InterruptedException e) {
                e.printStackTrace();
            }

        }
        latch.countDown();
    }

}
```

Problems Javadoc Declaration Console


<terminated> Main (1) [Java Application] /Users/paulbauer/.p2/pool/plugins/org.e
pool-1-thread-1: 0
pool-1-thread-1: 1
pool-1-thread-1: 2
pool-1-thread-1: 3
pool-1-thread-1: 4
pool-1-thread-1: 5
pool-1-thread-1: 6
pool-1-thread-1: 7
pool-1-thread-1: 8
pool-1-thread-1: 9
pool-1-thread-1: 10
pool-1-thread-1: 11
pool-1-thread-1: 12
pool-1-thread-1: 13
pool-1-thread-1: 14
pool-1-thread-1: 15
pool-1-thread-1: 16
pool-1-thread-1: 17
pool-1-thread-1: 18
pool-1-thread-1: 19
pool-1-thread-1: 20
pool-1-thread-2: 20
pool-1-thread-2: 19
pool-1-thread-2: 18
pool-1-thread-2: 17
pool-1-thread-2: 16
pool-1-thread-2: 15
pool-1-thread-2: 14
pool-1-thread-2: 13
pool-1-thread-2: 12
pool-1-thread-2: 11
pool-1-thread-2: 10
pool-1-thread-2: 9
pool-1-thread-2: 8
pool-1-thread-2: 7
pool-1-thread-2: 6
pool-1-thread-2: 5
pool-1-thread-2: 4
pool-1-thread-2: 3
pool-1-thread-2: 2
pool-1-thread-2: 1
pool-1-thread-2: 0


Paul-A-Bauer / CSC450_PortfolioProject
Type / to search
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
CSC450_PortfolioProject
Public
Pin
Unwatch 1
Fork 0
Star 0
main
1 branch
0 tags
Go to file
Add file
Code
Paul-A-Bauer Added Java class for simple threading example
d986d69 yesterday
6 commits
CSC450_PortfolioProject.xcodeproj
Initial Commit
last week
CSC450_PortfolioProject
Initial commit
5 days ago
Java
Added Java class for simple threading example
yesterday
Documentation.docx
Added Documentation file
5 days ago
Documentation.pdf
Added Documentation file
5 days ago
Help people interested in this repository understand your project by adding a README.
Add a README
About
Portfolio project for CSC450 programming III at CSUGlobal
Activity
0 stars
1 watching
0 forks
Releases
No releases published
Create a new release
Packages
No packages published
Publish your first package

Conclusion

Using the runnable interface, ExecutorService, and CountDownLatch this project demonstrates how to coordinate two threads and force one thread to wait for another to end before executing. Using CountDownLatch to track when threads have completed I can coordinate the two threads and manage their use of resources.

References