Week 9 Assignment: Make the Parking System Application Server Multithreaded

For Graduate Certificate

Information and Communication Technology

Christopher Jackson

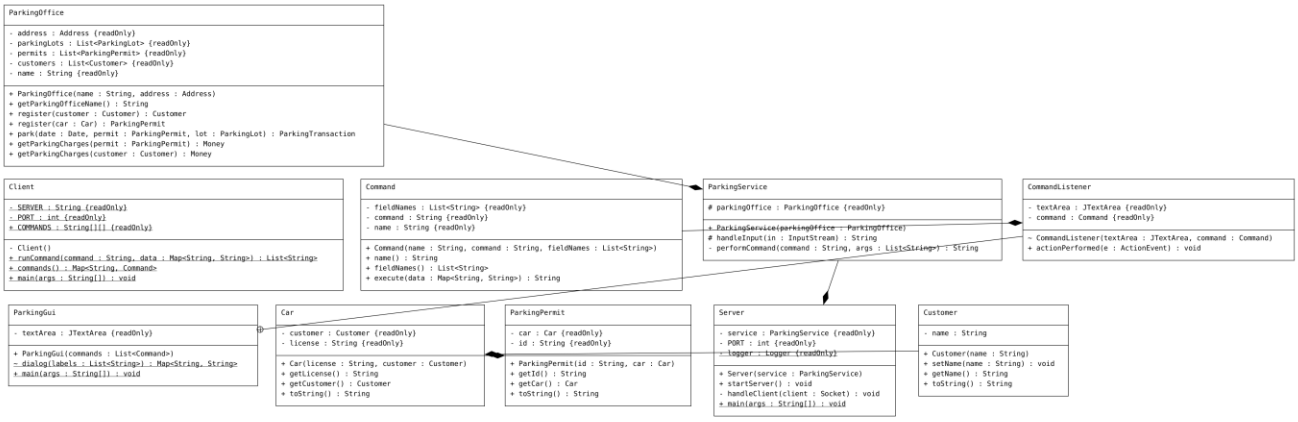
University of Denver University College

05/30/2020

Faculty: Robert Judd, PhD.

Dean: Michael J. McGuire, MLS

UML Class Diagram



Creating Thread-Safe Code

This project focuses on creating thread-safe code and testing that code with a multi-threaded test. The code used is a Client-Server example. The goal is to create a thread-safe server side to handle multiple clients simultaneously. The final product can handle 2,000 Client threads accessing the server without error.

I took a different approach with this project than previous projects. I used test-driven development. When designing thread safe code my main concern was

being able to test the code. If I missed one mutable object it could spell trouble for my code running in production.

I noticed within Golden Rules of Threading there was a for loop that went through 0-Integer.MAX_VALUE. This loop was able to break the code as demonstrated. I used that loop as the basis of my design for ClientTest. I created additional classes within the test package whose sole purpose was to create new threads. With my code finally broken, I could now go about fixing it.

There were several methods I used to create thread-safe code. I landed on creating all immutable objects to product thread-safe code. Prior to that I used synchronized. Synchronized brands itself as an easy way to create thread-safe code. Prior to using synchronized I attempted to use locks. Java generated unusually helpful error messages when my code was breaking. An error message I was given was line 39 in ParkingService.java threw a NullPointerException. I focused in on that line. It appeared that when data.remove(0) was called, a NullPointerException was thrown. I assumed that either data.add(token) or the Scanner method were being skipped by one of the threads and therefore data had no value. I inserted a lock at the beginning of handleInput(). This did not work. I tried different ways of implementing a lock. One method, tryLock(), was

interesting so I attempted to use it in my code. The tryLock() method lets a thread attempt to lock a piece of code. If the thread cannot lock the code, it defaults to the catch in a try-catch block. This did not work, either. I wanted to see if my threads would respond to the lock at all, so I placed a one-minute timer in the tryLock() by using TimeUnit.MINUTES. Within the catch scope I tried a few different methods, namely Thread.sleep(2000). None of my threads were worried about the lock as it seemed none waited 1 minute. This caused me to lose faith in locks and implement synchronized in my code. I later found out the threads were accessing code prior to the handleInput() method.

When using synchronized to create thread-safe code, I noticed a reduction in errors generated in my ClientTest as I targeted more mutable methods and objects. I succeeded in creating thread-safe code using synchronized. The server side could handle up to ~2000 threads. The only error I would experience after 2000 is OutOfMemoryError to be able to create new threads. To do my due diligence, I set about making every object immutable. I wanted to use best practices in my project, and seeing a List be manipulated in a multi-threaded context did not seem appropriate. I used ConcurrentHashMap for HashMap, CopyOnWriteArrayList for List, and ConcurrentSkipListMap for LinkedHashMap. As I did this, nothing happened. However, at some point I wanted to test my

synchronized methods and make sure the appropriate ones were synchronized. During this time, something amazing happened. I was able to remove all my synchronized designations and the code still handles a multi-threaded environment. This initially had me very worried. I thought I may have fundamentally changed my ClientTest so it was not breaking mutable code. When thinking about it, I realized immutability is a solution to multi-threading.

Survey Of Unit Tests

The main test created is ClientTest and its method testMultiThread. The method testMultiThread uses two classes in the test package. These two classes both extend Thread so they can use the run() method. Both classes accept a payload of instructions. To assert my test passes, I added a CopyOnWriteArrayList to my MultipleClientTest which tracked the return from Client::runCommand(). Each loop I asserted that the return value form runCommand() was the same values as the values in the instructions. There were no additional tests used to help create multi-threaded code nor assert that the code is multi-threaded.