Programming Assignment Report

Program 3

Multi-Threaded Programing

Abstract

The purpose of this lab was to learn and understand how to use the GetOpt Utility using Java. We were also tasked to understand and then program a multi-threaded java program. The multithreaded program ran two racers at the same time and would output its path as it went on. When both racers were ready the age would be counted and would output each step. In the end the races would output when each racer was finished.

Introduction

3.1 Problem Statement

Understand how GetOpt works and program a multithreaded java program using the provided code.

3.2 Objective

To understand how GetOpt can be used in Linux and to program a multithreaded Racer program.

3.3 Background Information

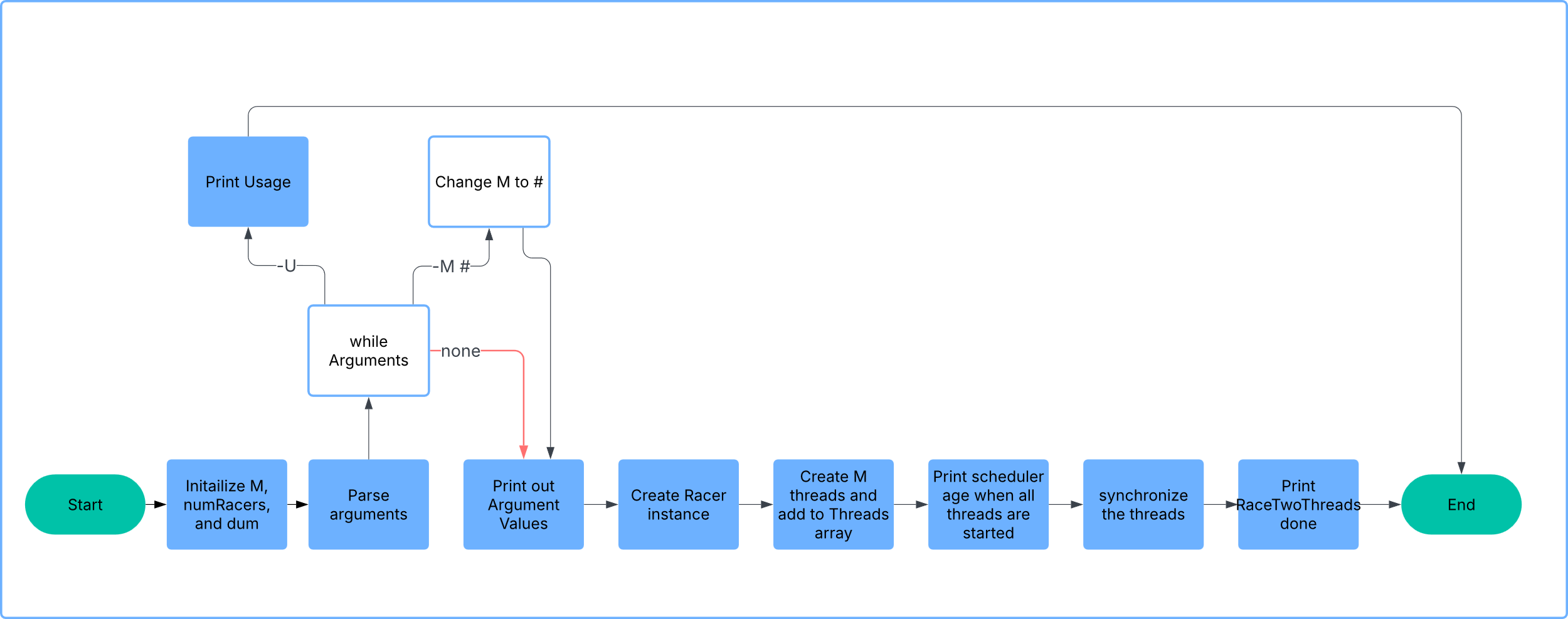
The concepts used within the lab relate to Multithreading in Java programing and argument input validation in Linux.

Materials and Methods

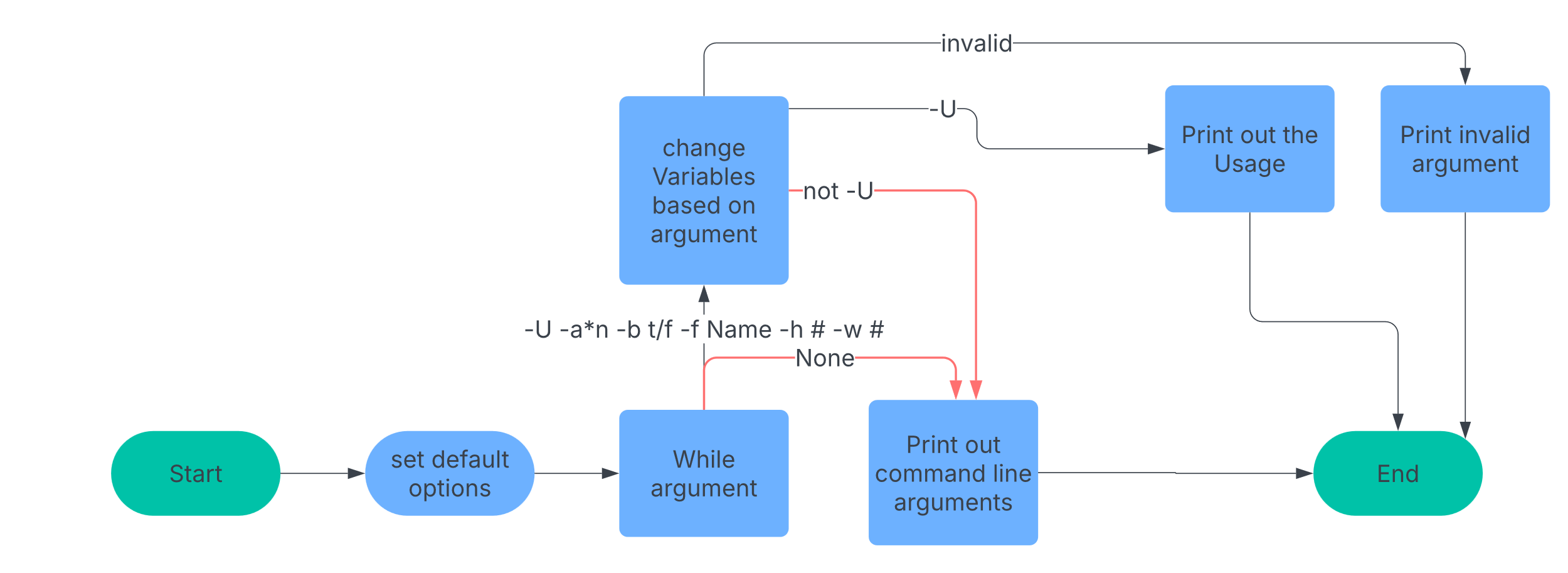
4.1 Tools and Technologies

For the assignment we used several different software types to help complete the lab. For the main part of the assignment, we used a Virtual ubuntu machine to run our Linux environment. Within the environment we used the terminal to access all the operating systems programs. We used the nano text editor to write Java code to then compile using javac compiler. We then were given a script to run as our control for both GetOpt and the racer program.

4.2 Program Design  
Racer.java



GetOpt.java



4.3 Code Implementation

Using the provided race.java file, I added the missing code to the class RaceTwoThreads > Main. I added the missing implements type. Added the abstract function run() to the Racer Class. Set numRacers to 2. The main part of the code that adds the racers threads, I created a racer instance, created an array of Threads to hold each of the threads. And within the loops I created and started each racer thread. I then added a line to join them within another loop to make them synchronized.

Results

5.1 Test Cases

I ran each of the given scripts to test the outputs of each file. The outputs are below.

5.2 Output

Running ans1 to see what the output would be.

A screenshot of a computer

AI-generated content may be incorrect.

This output shows that running the script each line produces different outputs when using getOpt. Line one just outputs the usage of command line arguments. Line 2 shows that all of the given arguments are valid arguments. Line 3 shows which argument is invalid ‘-x’ which is not defined in the usage. Line 4 shows a valid parsing of the repeated argument “-aaaaa’ which is parsed to ‘5’. Line 5 shows the arguments ‘-af theFile’ get parsed out separately even though they are together. Also the -w8 parsed out to its proper value as well.

Running ans2 to see each of the different lines that the scrip runs  
A screenshot of a computer program

AI-generated content may be incorrect.

This is the output from the racer.java class file. Line 1 shows the usage of the file. Line 2 runs the file with the default values of M=10 and a timeSlicingEnsured=False. This shows both threads run at the same time and finish around the same time. Line 3 runs the program with an M value of 50. The output is similar to the first with a small variance when each racer finishes. Line 4 runs the program with an M value of 70, showing a large difference in sum at the end. The output shows the ages are out of order which is due tot eh multithreaded nature of the program.

Discussion

6.1 Post-Lab Questions

N/A

6.2 Challenges Encountered and Improvements

A challenge I faced was attempting to run the scripts the first time, I had forgotten to allow them to be executed. This was solved by running chmod on the files, changing them to executable. a possible improvement would be to add more arguments to the racer class such as thread count.

Conclusion

I understand how the GetOpt command works and how to use it in Java programs, I also understand how Threads work when making a multithreaded program. The output of the racer program does match the way a multithreaded program should act by showing different start and stop times, but not in ascending order.

Screenshots

N/a

Source Code:

Racer.java

|  |
| --- |
| class Racer implements Runnable { // input the interface name  private String name;  private int M = 0; // these fields are shared by both  private volatile long sum = 0; // threads since there is one object  public Racer(String name, int M) {  this.name = name;  this.M = M;  System.out.println("age()=" + Scheduler.age() + ", "  + name + " is alive, M=" + M);  }  private long fn(long j, int k) {  long total = j;  for (int i = 1; i <= k; i++)  total += (2 \* i - 1) \* (2 \* i - 1);  return total;  }  public void run() { // input the abstrac function name  System.out.println("age()=" + Scheduler.age() + ", "  + name + " is running");  for (int m = 1; m <= M; m++) {  /\*  \* "N = N + 1" type lost update (race condition) in following line  \*/  sum = fn(sum, m);  }  System.out.println("age()=" + Scheduler.age() + ", "  + name + " is done, sum = " + sum);  }  }  class RaceTwoThreads {  private static int M = 10;  private final static int numRacers = 2; // fill in a number that how many threads you want to run  public static void main(String[] args) {  // parse command line arguments, if any, to override defaults  GetOpt go = new GetOpt(args, "UtM:");  go.optErr = true;  String usage = "Usage: -t -M m";  int ch = -1;  boolean timeSlicingEnsured = false;  while ((ch = go.getopt()) != go.optEOF) {  if ((char) ch == 'U') {  System.out.println(usage);  System.exit(0);  } else if ((char) ch == 'M')  M = go.processArg(go.optArgGet(), M);  else {  System.err.println(usage);  System.exit(1);  }  }  System.out.println("RaceTwoThreads: M=" + M + ", timeSlicingEnsured="  + timeSlicingEnsured);  // start the two threads, both in the same object  // so they share one instance of its variable sum  Racer r = new Racer("Racer", M); // create a thread function pobject  Thread[] threads = new Thread[numRacers]; // create an array of threads  for (int i = 0; i < numRacers; i++)  threads[i] = new Thread(r, "RacerThread-" + i); // create a racer thread  for (int i = 0; i < numRacers; i++) {  threads[i].start(); // start each thread  }  System.out.println("age()=" + Scheduler.age() +  ", all Racer threads started");  // wait for them to finish if not forced consecutive  try {  for (int i = 0; i < numRacers; i++) {  threads[i].join(); // synchronize threads using join()  }  } catch (InterruptedException e) {  System.err.println("interrupted out of join");  }  // correct race-free final value of sum is 2\*220 = 440 for M of 10  // and 2\*1335334000 = 2670668000 for M of 2000 (so `long sum' needed)  System.out.println("RaceTwoThreads done");  System.exit(0);  }  } |

Appendix

n/a