Design Project Two

Schneider Campfort Jean-Pierre

CSIS 3810: Operating Systems

Gregory Simco

November 21, 2022

INTRODUCTION

In this project, it is expected to produce a solution to the problem described in the assignment. It is expected to have separate threads representing each of the race cars according to the three different types of cars available. Depending on the type of car, it will determine which thread class the car will be created using. Each thread class will have a specific specification defining which of the resources it will need when a car stops at the replenishment area. It will also have specifications to calculate at random the number of laps in between stops. In addition to the threads for the cars, there will be a thread representing the supply coordinator. This thread will handle the constant replenishment of the different resources; tires, refreshments, and batteries. It will do this using a counter system, depending on the overall laps completed by the cars, and using time delays. It will also utilize random values to ensure that the distribution of the resources is randomly iterated throughout the race, only allowing a set of two different resources at a time. The resources will be represented using semaphores while having the car threads use the semaphore primitive acquire to use the resources for replenishment in their stops. The supply coordinator would distribute the resources using the release semaphore primitive. It is expected to carry out each of these threads at the same time without any deadlocks or errors, it should synchronize so that each car makes it to the end of the race after having a certain number of stops. It is also expected to have the activity of the race printed out on the console to ensure that the project is meeting expectations.

DESIGN

With the design of this project, it was important to have it as close to the expected solution to the problem as possible. It was important to include multiple thread classes, semaphores, implementations of tracking the laps, and overall synchronization. Although the final product did not meet all the expectations planned, it still produced a workable program close to what the problem detailed.

SCOPE/OVERVIEW

The project will be able to provide an accurate simulation of how a race with different cars stopping and gathering different resources at a time should play out. Having implementations of various random variables with java.util.Random, it should allow the program to run with different outcomes each time, similar to how an actual race wouldn’t have a set outcome. With having asset amount of laps for the cars to iterate through, the race car thread classes set with having the three different types requiring two different sets of resources in accordance to what they have an unlimited amount of, each car thread randomizing a set amount of laps for each stop that is specific to each car, the supply coordinator thread restocking if the supplies are not used after a set amount of time. This will be all achieved by not only by using java.util.Random, but by also using java.util.concurrent.Semaphore for semaphore implementation, java.util.ArrayList, and other key elements.

DATA DESIGN

For the data design, a lot of the data used within the problem are hard coded. The number of laps is hard coded to twenty, allowing a lot of opportunities for each race car to make a stop without the program running too long. This value can be modified later on if needed for more laps or to turn into a user input value. The semaphores for battery, refreshment, and tires are defined at the beginning and set to zero. The semaphores are also set to true to ensure that the first race car thread to call it, is the first car to receive it. An empty array is also defined to hold the names of all the racers when they finish all their laps and “cross the finish line”. It is used to keep track of the overall progress of the cars finishing, as well as for output purposes. After the overall variables are defined, separate threads are established. There is a different thread for each of the three types of race cars and one for the supply coordinator with different time specifications for certain tasks for each. In the main, the supply coordinator is defined and started. In addition, there are six cars defined to a specific thread class and given a name to recognize it during the console’s output. These are hard-coded, however just like the laps can be modified to add more, remove a few, or turned into user input values. These cars are also all started as well after each definition.

ARCHITECTURAL DESIGN

INITIALIZATION

On the start of the java program all the key elements of java needed to carry out the code is imported. This includes java.util.ArrayList, java.util.Random, and java.util.concurrent.Semaphore. After that all the overall variables needed for the program is defined. This includes the laps that each car must drive, the semaphores for the resources (battery, refreshment, tires), and an ArrayList to hold the names of the cars when they finish their laps in order. Following that, the different thread classes are defined with a specification for each. In addition to the thread classes, in the main class is where the Supply Coordinator is defined and started. It is also where each of the race cars is started with a specific name and is defined as well. Immediately after all of this takes place, the program begins to start the race by having each car set a random total value (between three and six) of laps till they need to stop for a replenishment.

JOBS/SCHEDULER

For the jobs of the car threads, it uses a for-loop iteration’s as a representation of a lap around the track. Each of the laps is subtracted from a counter with a random int value that is set to represent how many laps must be driven before stopping. Once the counter has reached zero, the car thread tries to acquire the specific two resources that it needs for replenishment. If it is available it takes it, if it is not, it waits. Once the resources are acquired, the thread is put to sleep for 2000 milliseconds, representing the time it takes to apply the resources. An output statement stating the car has finished is displayed. Then a new random value for the number of laps till the next stop (between 3 and 6) is generated and set for a new counter. This cycle continues until the race car has finished the race by running through all the total laps.

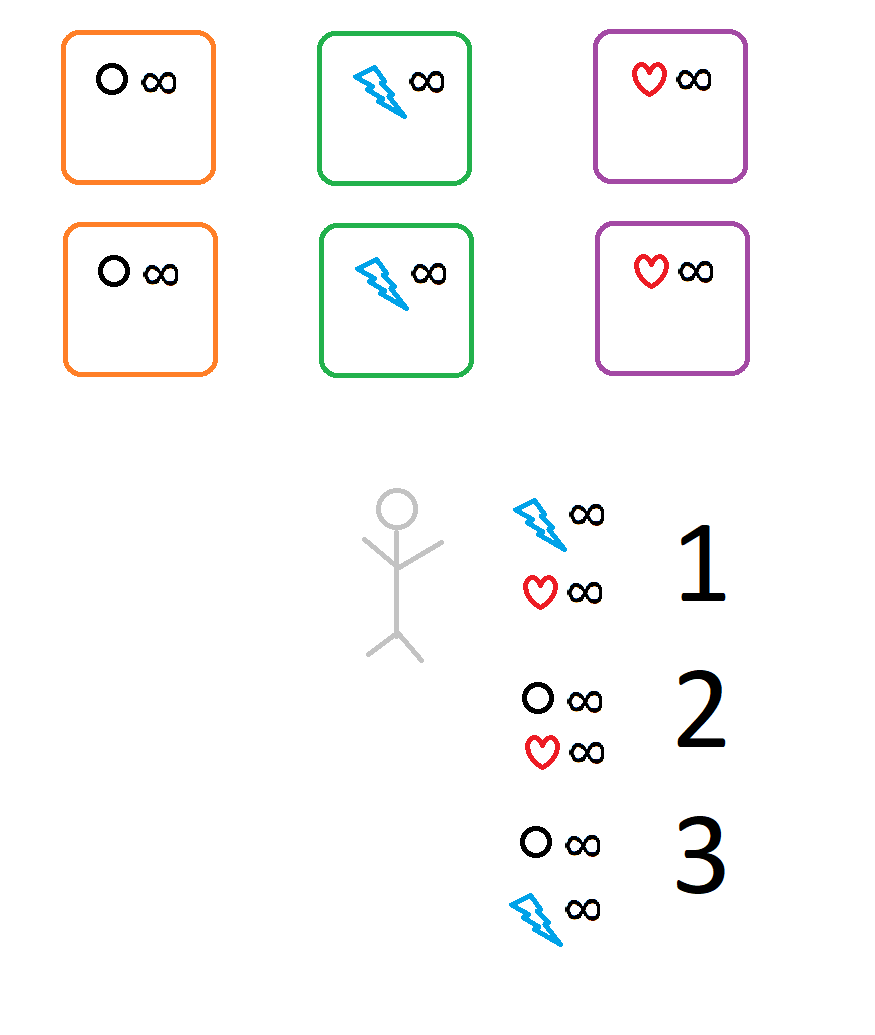
The job of the supply coordinator thread uses a while loop to keep track of how many cars are in the race. If all the cars have finished the race, then the supply coordinator doesn’t need to put out any resources, however, until then the supply coordinator must constantly put out resources. In the while loop it has an if else statement to track the availability of the resources, according to the number of resources available, the supply coordinator will randomly release a set of two different resources at a time for the car threads to acquire. If there is an instance where resources are not taken after 5000 milliseconds, the supply coordinator will take back the resources and release a new set of resources. Once again, it continues to do this cycle until all the cars have finished the race.

TERMINATION

Each of the racing threads is set to end once they have reached 20 total laps that they have driven. This is represented by the laps variable and used as a comparison for the counter for the iterations of the laps. Once a car has completed the total number of laps, its name is added to the finished ArrayList. Once all the cars have finished their total number of laps, the supply coordinator can stop putting out resources. It uses the ArrayList filling up with the total number of cars as a signal to stop. Once the supply coordinator stops, it exits its loop. This means that all the threads are no longer running and processing.

CLASS AND OBJECT DESIGN/MODULES

The overall public class main contains the overall code for this whole program. In this main it begins with all the overall variables that the program needs in order to run processes and tasks. It then has the CarTypeA, CarTypeB, and CarTypeC thread classes that are used to create threads for each of the three different types of cars that the problem describes. Each class is set to have the race car drive through a set number of laps using a for loop. Each lap lasts 1000 milliseconds to resemble the time it takes to complete a full lap. Each class although similar have specifications on the type of resources they try to acquire. In addition to those thread classes, there is a thread class that represents the supply coordinator, named SupplyC. In this class it uses a while loop to track the number of racers that have completed the race. Until all racers are done racing, the supply coordinator continues to put out supplies for the racers to use. In public static void main, this is where the supply coordinator is defined and started. It is also where all the race cars are defined and started as well. Picture below depicts the different race cars with the resources that they have an unlimited amount of and the supply coordinator with the different sets of outputs that can satisfy each type of car. The code follows this same depiction.



TEST PROVISIONS

Testing began with limiting the code to one type of car and testing how well it interacted with the supply coordinator releasing exactly what it needed. Once that was established and working, then the other types of cars were implemented and tested. They were tested the same way as the one type of car was tested, by setting a set value for the supply coordinator’s if else statement deciding variable. Once they worked fine, then random variables were implemented for the deciding variable of the supply coordinator’s if else statement. This brought in many errors of cars not receiving the resources and staying stuck at the general refreshments table. That is when it seemed crucial to have the supply coordinator basically refresh the options on the table after a certain amount of time had passed so that no car ends up stuck without being able to complete the race. Errors occurred with the random seed tried to be implemented. The same value kept getting generated and wasn’t random every time. This caused for a random with the seed not to be implemented and just regular random to be. Once that was implemented, all the cars were able to successfully complete each of their stops and finished the overall race. It isn’t a complete synchronization as I think that problem would have liked, however, it does provide a working code similar to what the problem described.

OUTPUT

(Random values will cause output to be different each time):

Supply Coordinator has placed battery and refreshments out on the table.

Racer 3 has completed lap #1

Racer 4 has completed lap #1

Racer 6 has completed lap #1

Racer 5 has completed lap #1

Racer 2 has completed lap #1

Racer 1 has completed lap #1

Racer 3 has completed lap #2

Racer 4 has completed lap #2

Racer 6 has completed lap #2

Racer 6 has stopped at the general refreshments table.

Racer 1 has completed lap #2

Racer 2 has completed lap #2

Racer 5 has completed lap #2

Racer 3 has completed lap #3

Racer 4 has completed lap #3

Racer 5 has completed lap #3

Racer 2 has completed lap #3

Racer 2 has stopped at the general refreshments table.

Racer 1 has completed lap #3

Racer 4 has completed lap #4

Racer 3 has completed lap #4

Racer 1 has completed lap #4

Racer 5 has completed lap #4

Racer 4 has completed lap #5

Racer 4 has stopped at the general refreshments table.

Racer 3 has completed lap #5

Racer 3 has stopped at the general refreshments table.

Racer 1 has completed lap #5

Racer 1 has stopped at the general refreshments table.

Racer 5 has completed lap #5

Racer 5 has stopped at the general refreshments table.

Supply Coordinator has placed battery on the table.

Supply Coordinator has placed battery and tires out on the table.

Racer 4 has replenished their needs and has returned to the race.

Racer 4 has completed lap #6

Supply Coordinator has placed battery and refreshments out on the table.

Racer 4 has completed lap #7

Racer 6 has replenished their needs and has returned to the race.

Racer 4 has completed lap #8

Racer 6 has completed lap #3

Supply Coordinator has placed battery and refreshments out on the table.

Racer 1 has replenished their needs and has returned to the race.

Racer 4 has completed lap #9

Racer 6 has completed lap #4

Racer 4 has completed lap #10

Racer 4 is car number 1 to finish the race.

Racer 1 has completed lap #6

Racer 6 has completed lap #5

Racer 1 has completed lap #7

Racer 6 has completed lap #6

Racer 1 has completed lap #8

Racer 6 has completed lap #7

Racer 6 has stopped at the general refreshments table.

Racer 1 has completed lap #9

Racer 1 has stopped at the general refreshments table.

Supply Coordinator has placed battery on the table.

Supply Coordinator has placed tires and refreshments out on the table.

Racer 1 has replenished their needs and has returned to the race.

Racer 1 has completed lap #10

Racer 1 is car number 2 to finish the race.

Supply Coordinator has placed tires and refreshments out on the table.

Racer 2 has replenished their needs and has returned to the race.

Racer 2 has completed lap #4

Supply Coordinator has placed tires and refreshments out on the table.

Racer 5 has replenished their needs and has returned to the race.

Racer 2 has completed lap #5

Racer 5 has completed lap #6

Racer 2 has completed lap #6

Racer 3 has replenished their needs and has returned to the race.

Supply Coordinator has placed battery on the table.

Racer 5 has completed lap #7

Racer 2 has completed lap #7

Racer 3 has completed lap #6

Racer 5 has completed lap #8

Racer 2 has completed lap #8

Racer 2 has stopped at the general refreshments table.

Racer 3 has completed lap #7

Racer 5 has completed lap #9

Racer 3 has completed lap #8

Racer 3 has stopped at the general refreshments table.

Racer 5 has completed lap #10

Racer 5 is car number 3 to finish the race.

Supply Coordinator has placed battery on the table.

Supply Coordinator has placed battery and tires out on the table.

Supply Coordinator has placed tires on the table.

Racer 6 has replenished their needs and has returned to the race.

Racer 6 has completed lap #8

Supply Coordinator has placed tires on the table.

Racer 6 has completed lap #9

Racer 6 has completed lap #10

Racer 6 is car number 4 to finish the race.

Supply Coordinator has placed tires on the table.

Racer 3 has replenished their needs and has returned to the race.

Racer 3 has completed lap #9

Racer 3 has completed lap #10

Racer 3 is car number 5 to finish the race.

Supply Coordinator has placed tires and refreshments out on the table.

Supply Coordinator has placed refreshment on the table.

Racer 2 has replenished their needs and has returned to the race.

Racer 2 has completed lap #9

Racer 2 has completed lap #10

Racer 2 is car number 6 to finish the race.