CNT 4714 – Project 2 – Spring 2023

Title: "Project 2: Multi-threaded Programming in Java"
Points: 100

Due Date: Sunday February 16, 2025 by 11:59pm WebCourses time

Objectives: To practice programming an application with multiple threads of execution and synchronizing their access to shared objects.

Description: All Class 1 railroads (BNSF, Union-Pacific, CSX, Norfolk-Southern) in the United States today, use a train scheduling technique known as Precision Scheduled Railroading (PSR). In the past, the USA rail service model focused on moving long trains in order to maximize capacity and yield the greatest efficiency. This technique however, did not always yield the best outcome for either the railroad or its customers. Since the railroads aimed to build trains that would move faster, if a train didn't meet a specific length requirement, it could be cancelled (annulled in railroad lingo). This would often leave the customer without service for the day. This "train focus" technique meant that customer cars could sit for long periods of time before being picked up and delivered to their destination. PSR shifts the focus from moving trains to moving train cars (hereafter referred to as simply cars). Under PSR, instead of waiting for a long train to be built (in what are referred to as "hump yards"), trains are always moving and cars are picked up on schedule, regardless of train length. Speed and train length are still important factors for railroads, but now, the focus on moving cars takes precedence. PSR requires that railroads move trains through their switch yards as quickly as possible in order to avoid unnecessary delays in the network. In addition to promoting network fluidity and more reliable service, this shift in focus also yields another important advantage: As trains continually move through the network, it becomes more balanced — meaning the right resources, like crews, cars and locomotives, are in place when they are needed. For railroads, that means more effective use of resources.

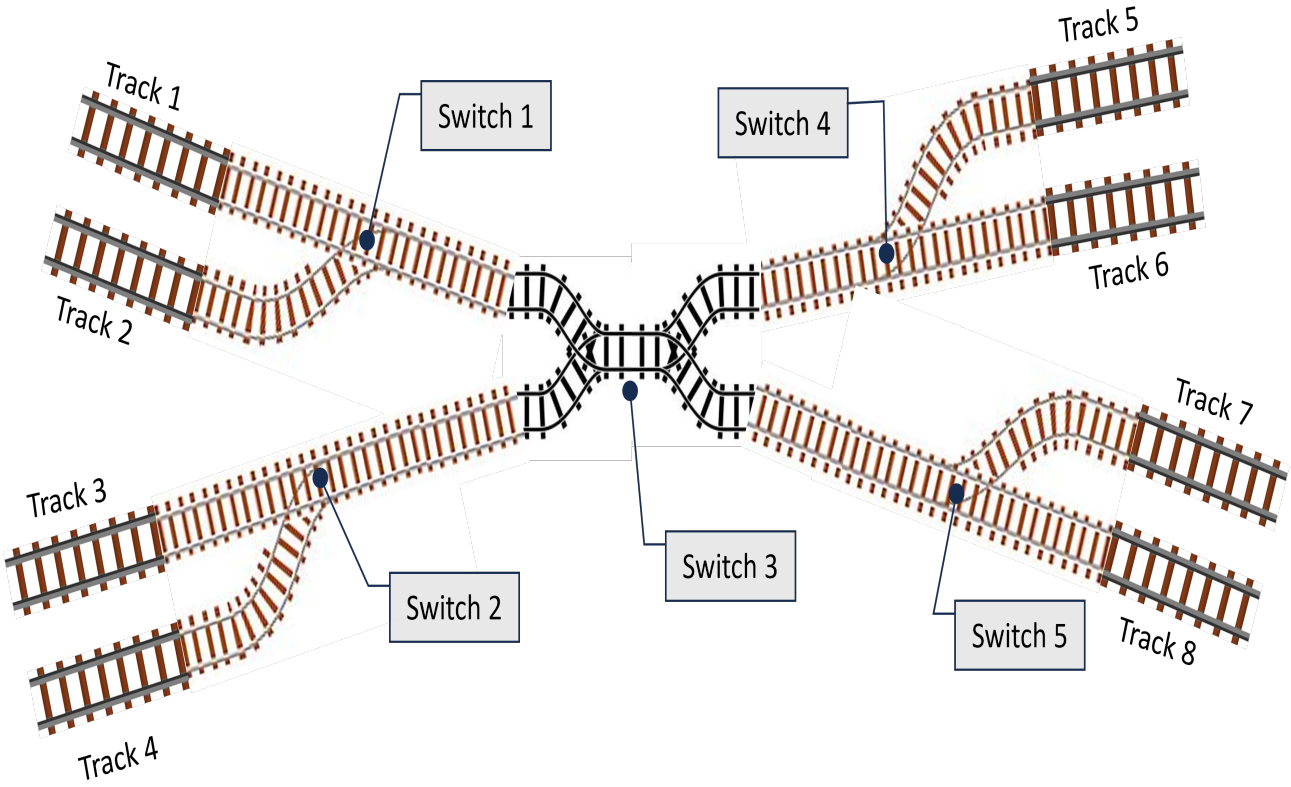
Train scheduling, in the real-world, is an NP-C problem[1], so we are not going to be solving the train scheduling algorithm with this project. Instead, you will experience a small part of PSR by simulating a train switch yard and the movements of trains entering and exiting the yard, similar to the one shown on the next page.

Trains can be arriving on any track on either side of the yard and depart on any track on either side of the yard for which there is a track/switch configuration allowed by the yard. Only one train on a given track can move through yard at a time same time regardless of the direction of travel of the train. There may be multiple trains on the same track moving the same direction waiting to go through the yard. Although it is not shown on the main picture, the focused picture below shows the track configuration beyond the yard control that allows multiple trains to move on parallel tracks. CTC (Centralized Traffic Control) controls trains outside of the train yard.

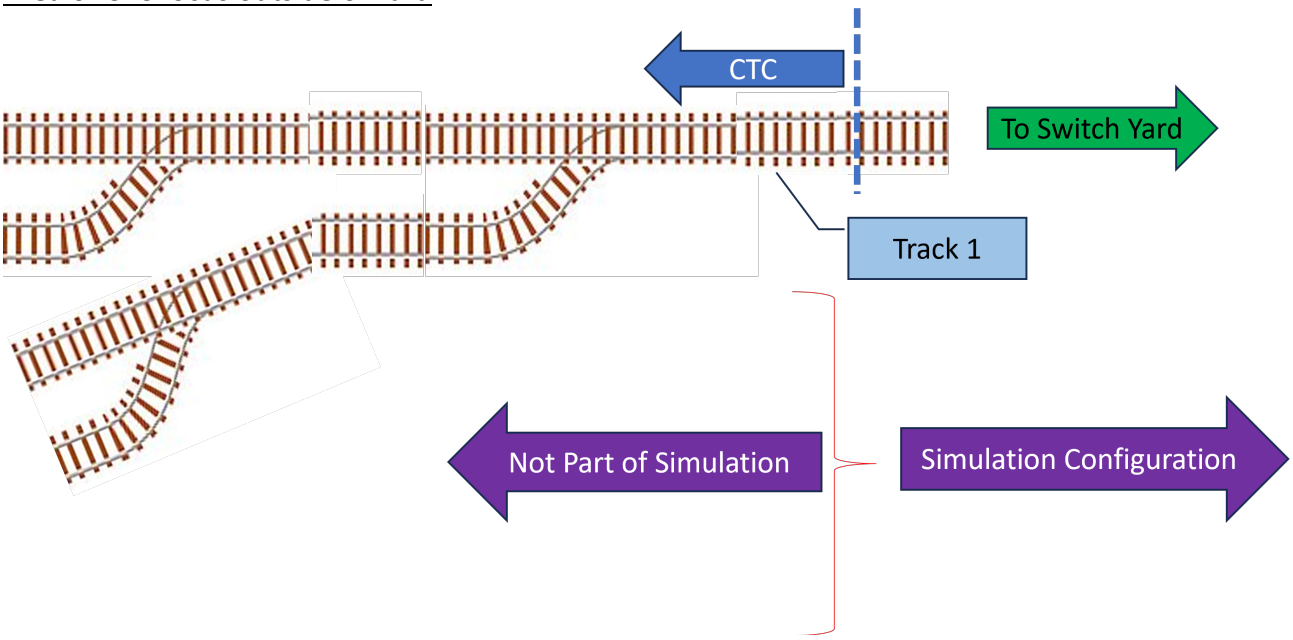
[1] Although a solution to an NP-complete problem can be *verified* "quickly", there is no known way to *find* a solution quickly. That is, the time required to solve the problem using any currently known algorithm increases rapidly (exponentially) as the size of the problem grows. As a consequence, determining whether it is possible to solve these problems quickly, called the P versus NP problem, is one of the fundamental unsolved problems in computer science today.

To move through the yard shown below, a single train needs to assume control of the three switches required to move the train from its inbound track to its outbound track. At any given moment in time, only one train can be moving through the yard switch complex at a time. There is not sufficient track length between switches to allow trains to idle in between switches.

The Train Yard



Area of CTC focus outside of Yard



Every train yard has a unique track/switch alignment configuration, so we are only simulating this particular track/switch arrangement. Even within this fairly simple yard arrangement, there are multiple possibilities for track/switch alignments. Your application does not need to handle random or unknown configurations. The configuration you are simulating is shown above and consists of eight (8) separate tracks, four on each side of the yard and five switches that allow the various tracks to be aligned. A specific configuration for the switchyard is held in an input file (see below).

As stated above, a train cannot begin moving through the yard until it controls all of the switches that it needs to move from its inbound track to its outbound track. Until a train obtains control of all of the switches that it needs to move through the yard, it will remain idle on its inbound track. To prevent switch control deadlock from occurring, every train must acquire the switches it needs in the order of first switch required, second switch required, third switch required. At any point in time during a train's switch acquisition phase, if a required switch is not available, the train must release all of the switches that it currently controls and wait some period of time before trying again.

For example, if Train A is inbound on Track 1 and needs to be outbound on Track 7, then Train A must acquire Switches 1, 3, and 5, in that order. If all three switches are acquired, then Train A can begin moving through the yard. On the other hand, if Train A has acquired Switch 1, and 3, but fails to acquire Switch 5, then it must release control of Switches 1 and 3 and wait some random period of time before attempting to again acquire all of the switches that it needs to move through the yard.

Under CTC scheduling, most trains move in fleets. A fleet will consist of an unknown number of trains moving from various origins to various destinations. For our purposes here, a fleet schedule will be represented by a file named `TheFleetFile.csv`. This is a comma separated file in which each line of the file contains three items: (1) the Train Number, (2) the inbound track number, and (3) the outbound track number. Thus, a line in this file such as (3,5,4) would indicate train number 3 is inbound on track 5 and will be outbound on track 4.

The switch yard configuration will also be represented by a comma separated file, named `theYardFile.csv`. This file contains a line for each possible track/switch combination allowed by the switch yard tracks and switches. Each line will contain five items in the following order: (1) inbound track number, (2) first required switch number, (3) second required switch number, (4) third required switch number, and (5) outbound track number. Thus, a line in this file such as (5,4,3,1,1) would indicate to go from inbound Track 5 to outbound Track 1, the order in which to acquire the switches to accomplish this move would be (4,3,1). Similarly, a line in the file such as (6,4,3,5,8) would indicate to go from inbound Track 6 to outbound Track 8, the switches would need to be acquired in the order (4,3,5).

Note that if an track/switch alignment combination is not represented in `theYardFile.csv`, then any train attempting to use that track/switch alignment will be placed on a permanent hold and will not be allowed to proceed from its inbound track to its outbound track. All possible combinations of track/switch alignments are possible, but not necessarily represented in any specific instance of `theYardFile.csv`. Only track/switch alignments represented in `theYardFile.csv` file can be utilized by any train.

Restrictions:

1. Your source files should begin with comments containing at least the following information:

```
/*
        Name: <your name goes here>
        Course: CNT 4714 Spring 2025
        Assignment title: Project 2 – Multi-threaded programming in Java
        Date:  February 16, 2025

        Class:  <name of class goes here>

        Description:  a description of what the class provides would normally be expected.
*/
```

2. You must use a the `java.util.concurrent.locks.ReentrantLock` interface. In other words, do not create your own locking system nor implement a Boolean semaphore-like system to control the locking.
3. **DO NOT** use a monitor to control the synchronization in your program (i.e., do not use the Java synchronize statement).
4. You must use an `ExecutorService` object to manage a `FixedThreadPool(MAX)`, where `MAX` is the upper limit on the number of trains in a fleet, which we'll set to be 30 (see below under Input Specification).
5. Your Train threads must implement the `Runnable` interface and not extend the Thread class in order to utilize the `ExecutorService` object mentioned in 4 above.
6. Following good OO design principles will require that you have at minimum, three (3) separate class files for this project. **DO NOT** incorporate your lockable objects and thread objects into the same class.

Input Specification:

Your application must read the simulation configuration specifications from the two files: `theFleetFile.csv` and `theYardFile.csv` as each are described above. You can assume that the maximum number of trains in any fleet will be 30. The maximum number of switch configurations (alignments) that can occur within the train yard will be set to 60. The maximum number of switches that will be located in the train yard will be set to 10.

Output Specification:

Your simulator must output **at least** the following text to let the user know what the simulator is doing in each of these situations:

1. When a train obtains the lock on a needed switch:
   Train #: HOLDS LOCK on Switch #.

2. When a train is unable to acquire the lock on the first required switch:
   Train #: UNABLE TO LOCK first required switch: Switch {first}. Train will wait…

3. When a train holds the lock on the first required switch but cannot acquire the lock on the second required switch:
   Train #: UNABLE TO LOCK second required switch: Switch {second}.
   Train #: Releasing lock on first required switch: Switch {first}. Train will wait…

4. When a train holds both the first and second required switch locks but cannot acquire the lock on the third switch:
   Train #: UNABLE TO LOCK third required switch: Switch {third}.
   Train #: Releasing locks on first and second required switches: Switch {first} and Switch {second}.
   Train will wait…

5. When a train has obtained all required locks required to move through the yard:
   Train #: HOLDS ALL NEEDED SWITCH LOCKS – Train movement begins.

6. After event #5 above has occurred (after a suitable time to simulate the time required to move the train through the yard):
   Train #: Clear of yard control.
   Train #: Releasing all switch locks.
   Train #: Unlocks/releases lock on Switch {first}.
   Train #: Unlocks/releases lock on Switch {second}.
   Train #: Unlocks/releases lock on Switch {third}.
   Train #: Has been dispatched and moves on down the line out of yard control into CTC.
   @ @ @ TRAIN #: DISPATCHED @ @ @

7. When a train's inbound track to outbound track does not match any allowed switch configuration in the train yard:
   ****
   Train # is on permanent hold and cannot be dispatched.
   ****

8. As the simulation begins:
   $ $ $ TRAIN MOVEMENT SIMULATION BEGINS……. $ $ $

9. After all trains, not on permanent hold have been dispatched:
   $ $ $ SIMULATION ENDS $ $ $

10. When all trains that are dispatchable, have been dispatched (i.e., the simulation ends). Your application must print a final status for all the trains that appeared in this simulation run. An example of this output is shown on page 9. The example shows only a partial listing as it would not fit on one page.

Deliverables:

Submit the following items via WebCourses no later than 11:59pm Sunday February 16, 2025. Zip all deliverables into a single zip archive.

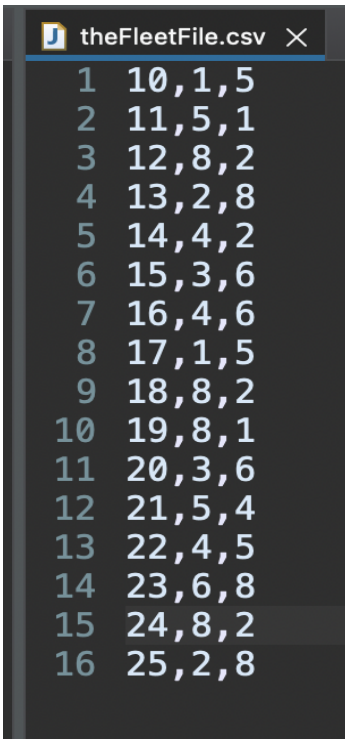
(1) All of your `.java` files.

(2) A copy of a sample execution of your program, i.e., the output produced by your simulator (this should just be a text file). In your IDE redirect console output to a text file, do this and include a complete copy of the output file produced by your application for a sample run of your application. Note that this must be a complete simulation run with a minimum of ten trains in the simulation. This is not a screenshot of a partial simulation run.

Additional Information:

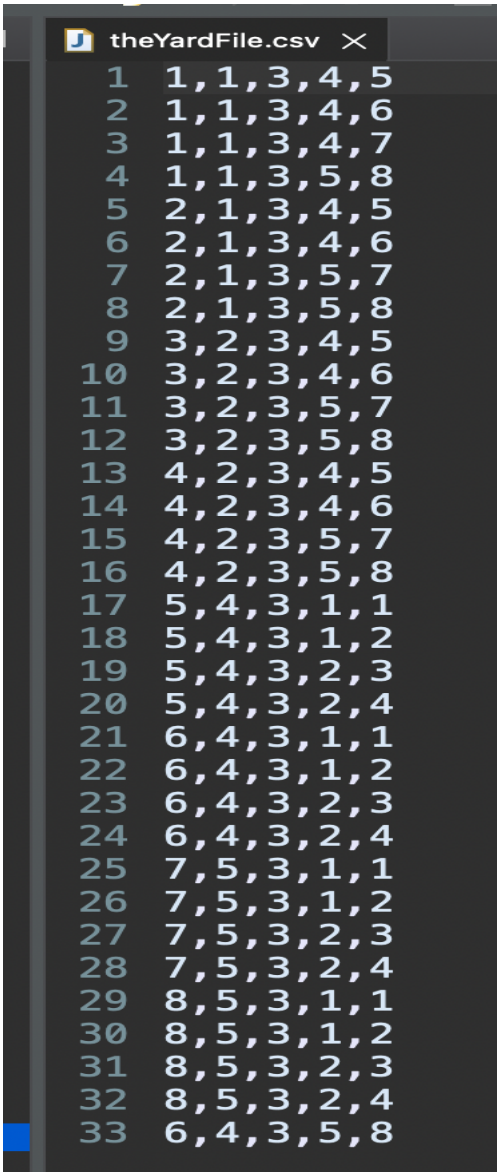
Shown below are two different sample input files for `theFleetFile.csv` and `theYardFile.csv` that you can use when developing your project. A copy of an actual simulation run output file is shown beginning on page 7 (console output redirected in this example). As I will demonstrate in the Q&A sessions for this project, I added a lot of debugging type output so that I can better understand what is shown in the simulation output. I added details printing virtually all of the configuration details that show the details of the fleet and the yard before the trains begin operation, and then also added some final output details that allow me to verify how the simulation proceeded. I would strongly suggest that you add similar debugging code to help you along as you develop the simulation application.

theFleetFile.csv



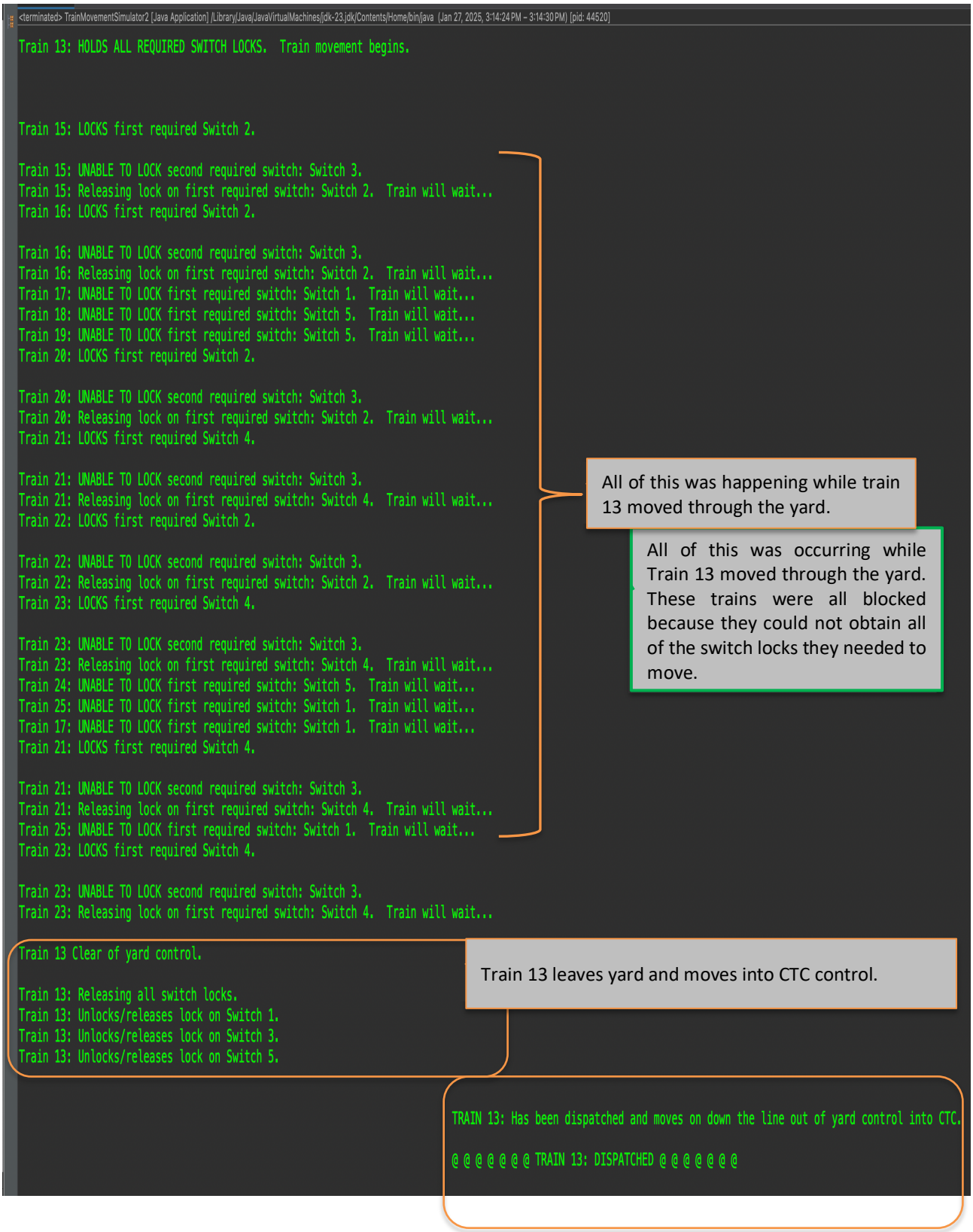
(train number, inbound track, outbound track)

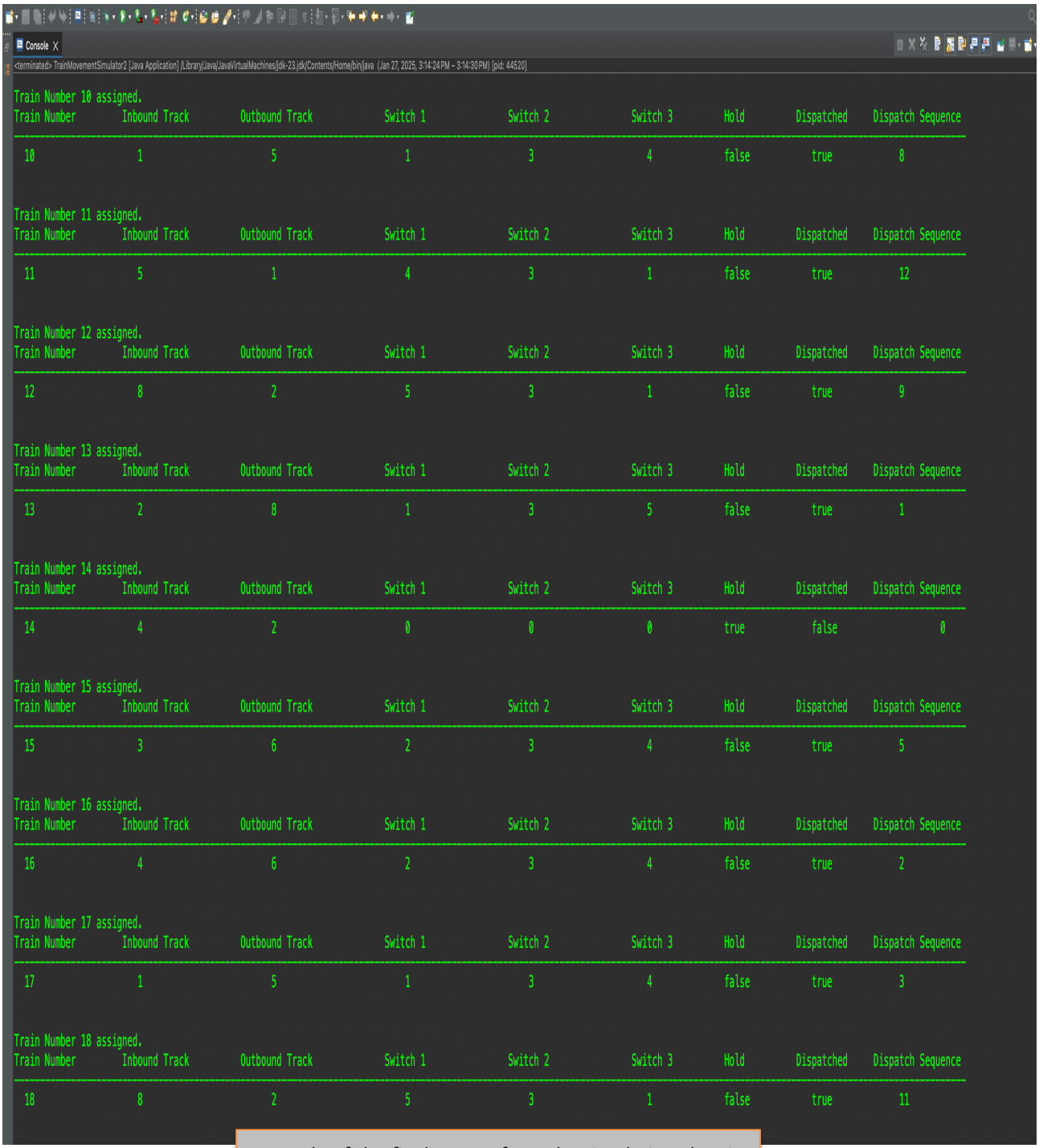
theYardFile.csv



(inbound track, first switch needed, second switch needed, third switch needed, outbound track)

This page and the following two pages show bits and pieces of a sample simulation run output using the two input files shown above. Note that what is shown is not a complete simulation output, but just some isolated screenshots from a simulation run. I will show you the complete output file from simulation runs in the Q&A sessions.


<terminated> TrainMovementSimulator2 [Java Application] /Library/Java/JavaVirtualMachines/jdk-23.jdk/Contents/Home/bin/java (Jan 27, 2025, 3:14:24 PM – 3:14:30 PM) [pid: 44520]
Train 13: HOLDS ALL REQUIRED SWITCH LOCKS. Train movement begins.
Train 15: LOCKS first required Switch 2.
Train 15: UNABLE TO LOCK second required switch: Switch 3.
Train 15: Releasing lock on first required switch: Switch 2. Train will wait...
Train 16: LOCKS first required Switch 2.
Train 16: UNABLE TO LOCK second required switch: Switch 3.
Train 16: Releasing lock on first required switch: Switch 2. Train will wait...
Train 17: UNABLE TO LOCK first required switch: Switch 1. Train will wait...
Train 18: UNABLE TO LOCK first required switch: Switch 5. Train will wait...
Train 19: UNABLE TO LOCK first required switch: Switch 5. Train will wait...
Train 20: LOCKS first required Switch 2.
Train 20: UNABLE TO LOCK second required switch: Switch 3.
Train 20: Releasing lock on first required switch: Switch 2. Train will wait...
Train 21: LOCKS first required Switch 4.
Train 21: UNABLE TO LOCK second required switch: Switch 3.
Train 21: Releasing lock on first required switch: Switch 4. Train will wait...
Train 22: LOCKS first required Switch 2.
Train 22: UNABLE TO LOCK second required switch: Switch 3.
Train 22: Releasing lock on first required switch: Switch 2. Train will wait...
Train 23: LOCKS first required Switch 4.
Train 23: UNABLE TO LOCK second required switch: Switch 3.
Train 23: Releasing lock on first required switch: Switch 4. Train will wait...
Train 24: UNABLE TO LOCK first required switch: Switch 5. Train will wait...
Train 25: UNABLE TO LOCK first required switch: Switch 1. Train will wait...
Train 17: UNABLE TO LOCK first required switch: Switch 1. Train will wait...
Train 21: LOCKS first required Switch 4.
Train 21: UNABLE TO LOCK second required switch: Switch 3.
Train 21: Releasing lock on first required switch: Switch 4. Train will wait...
Train 25: UNABLE TO LOCK first required switch: Switch 1. Train will wait...
Train 23: LOCKS first required Switch 4.
Train 23: UNABLE TO LOCK second required switch: Switch 3.
Train 23: Releasing lock on first required switch: Switch 4. Train will wait...
Train 13 Clear of yard control.
Train 13: Releasing all switch locks.
Train 13: Unlocks/releases lock on Switch 1.
Train 13: Unlocks/releases lock on Switch 3.
Train 13: Unlocks/releases lock on Switch 5.
TRAIN 13: Has been dispatched and moves on down the line out of yard control into CTC.
@ @ @ @ @ @ @ TRAIN 13: DISPATCHED @ @ @ @ @ @ @
All of this was happening while train 13 moved through the yard.
All of this was occurring while Train 13 moved through the yard. These trains were all blocked because they could not obtain all of the switch locks they needed to move.
Train 13 leaves yard and moves into CTC control.



Example of the final output from the simulation showin the status of every train in the simulation run.

More pictures of real trains below!


BNSF


Santa Fe
501
Santa Fe
Santa Fe