COSI 131a: Spring 2020

Programming Assignment 3

Due date is Wednesday, March 18th

Overview

Your task is to implement a simulation of tunnels and vehicles using Java Threads. There will be few tunnels, but many vehicles. Vehicle types can be cars or sleds. There will be several constraints on how many vehicles of each type a tunnel can contain at any given time.

This assignment requires you to use Java default **synchronized** methods and busy waiting to prevent race conditions.

Tunnel Description

Your task is to implement a class that implements the Tunnel interface. Any implementation of the Tunnel interface must satisfy the following restrictions:

- Each tunnel has only one lane, so at any given time all vehicles must be traveling in the same direction.
- Only three cars may be inside a tunnel at any given time.
- Only one sled may be inside a tunnel at any given time.
- Cars and sleds cannot share a tunnel.

As you know from learning about concurrency, without proper synchronization, these constraints could be violated since a thread may secure permission to enter a tunnel, then be interrupted before it can actually enter, at which point another thread may also secure permission to enter the tunnel and do so. When the first thread to get permission to enter the tunnel does so, one of the invariants may be violated (e.g., maybe a sled and car will be in the tunnel at the same time). Part of your job in this assignment is to use synchronization to prevent such race conditions from occurring regardless of scheduling.

To implement your solutions, you will need to use the code base provided by us, described below. This code base includes a test harness that allows us (and you) to test the correctness of your solutions. You will be therefore instructed to follow certain conventions in your code and will not be allowed to modify certain classes. For example, your code will not be starting or joining client threads. Instead, the threads will be started by the test harness. Please follow the instructions carefully. You will need to understand how the entire provided code works to complete your assignment.

Provided Code

The code for this assignment is divided into the following packages

cs131.pa3.Abstract

The `Abstract` package contains the abstract classes and interfaces that your program must implement. You are not allowed to modify any of the files within this package.

cs131.pa3.Abstract.Log

The `Abstract.Log` package contains classes used to record the actions of vehicles. These classes are used by the test packages to evaluate that constraints have not been violated. The log is not a means for passing messages. You are not allowed to modify any of the files within this package.

cs131.pa3.Test

You may not edit any of the test classes, but we highly encourage you to study them for your understanding and to aid in discovering why a test is failing. Note the `NUM_RUNS` variable in the tests that specifies how many times to run a test. You should make sure to run each test at least 10 times to avoid any errors that appear infrequently.

cs131.pa3.CarsTunnels

The `CarsTunnels` package contains a few mostly empty classes that implement the classes found in the Abstract package. All of the code pertaining to your solution of this assignment must go in this package. You are free to add, edit, rename, and delete files in this package in the course of implementing your solutions.

API Documentations
You will want to inspect the java files yourself, but to get you started we include the API docs generated with JavaDoc from the files we provide.

The Basic Tunnel

Implementation

You are provided with a class called `BasicTunnel` that implements the interface `Tunnel`, and you must implement functionality to carry out the following tasks:

- Enforce the `Tunnel` Restrictions described above.
- Use the Java `synchronized` keyword to prevent race conditions (without introducing deadlock).

When this task is complete your code should pass all tests included in `BehaviorTest`, and `SimulationTest`. Please make sure to run the tests several times as a race condition problem can appear in only some of the runs. We provide the `NUM_RUNS` variable inside each test class that specifies how many times to run each test. It is set to 1 by default but feel free to change it to how many times you want to run each test.

You will need to provide JavaDoc documentation for any new classes, methods, or members you create, and you may be penalized if you fail to do so.

Hints

`instanceof` operator

You might find the `instanceof` operator helpful. `instanceof` can tell you if an object can be downcast from a parent type into a child type. Typically, we don't use it because there are better techniques (polymorphism leverages the type system to do the work for you), but for this problem it will probably help you out.

Here is an example:

```
public class TestInstanceof {

      public static class Parent {}
      public static class Child1 extends Parent {}
      public static class Child2 extends Parent {}

      public static void main(String[] args) {
            Parent p1 = new Child1();
            Parent p2 = new Child2();

            if(p1 instanceof Child1) {
                  System.out.println("p1 is instance of Child1");
            }
            if(p1 instanceof Child2) {
                  System.out.println("p1 is instance of Child2");
            }
            if(p2 instanceof Child1) {
                  System.out.println("p2 is instance of Child1");
            }
            if(p2 instanceof Child2){
                  System.out.println("p2 is instance of Child2");
            }

      }
}
```

This outputs:

```
p1 is instance of Child1
p2 is instance of Child2
```

No “main” method

Note that your only point of entry in the code we provide is through the JUnit tests, which will set up the environment in which your client threads will run. Your tasks for this assignment do not include writing a main method. Rather, you must rely on your understanding of busy waiting, and the JUnit tests to know if you have completed the assignment.

Submission

Please export your project together with JavaDocs as described on LATTE and submit as a zip file. You should not submit source files individually.

Important Note about Disallowed Java Tools

In PA2, you were instructed to consider using a synchronized class provided by Java for inter-thread communication (`LinkedBlockingQueue`) to solve the problem. For this project, that is **not allowed**; you may not use **any** synchronized data structure included in the Java API. You must write your own (using the "synchronized" keyword). Of course, you can and should use non-synchronized data structures in the Java standard library. You can consult the API docs to see if a data structure is synchronized. Finally, you should solve this assignment by using Java 8 and Junit 5. No external packages are allowed, unless you got permission from the TAs.