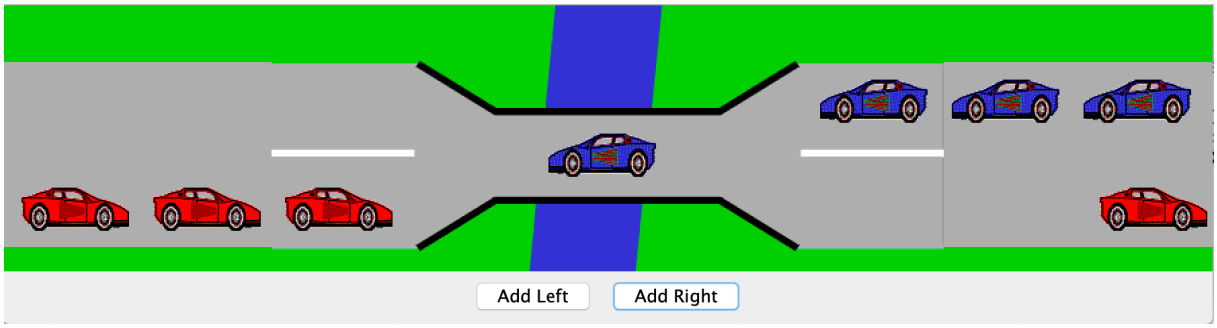
Assignment "Traffic Controller"

Develop a traffic controller for controlling the traffic over a single-lane bridge. Two different versions of the traffic controller need to be implemented. The interface **`TrafficController`**, which is given, specifies the functionality of a traffic controller. Moreover, the interface **`TrafficRegistrar`** is provided to register all vehicles that access the bridge, and the interface **`Vehicle`** to specify common properties of all vehicles.

Note: the interfaces *TrafficController, Vehicle und TrafficRegistrar* are given and must not be modified.



1. Interfaces `TrafficController, Vehicle, TrafficRegistrar`

The interface **`TrafficController`** defines methods for controlling access to the bridge.

```
public interface TrafficController {
    /* returns if bridge can be entered from right; blocks otherwise */
    void enterRight(Vehicle v);

    /* returns if bridge can be entered from left; blocks otherwise */
    void enterLeft(Vehicle v);

    /* must be called upon leaving bridge after entered from right */
    void leaveLeft(Vehicle v);

    /* must be called upon leaving bridge after entered from left */
    void leaveRight(Vehicle v);
}
```

If a vehicle wants to enter the bridge from the right, **`enterRight()`** must be called before entering the bridge, and **`leaveLeft()`** must be called before exiting the bridge. Similarly, if a vehicle wants to enter the bridge from the left, **`enterLeft()`** must be called before entering the bridge, and **`leaveRight()`** must be called before exiting the bridge.

The interface **`Vehicle`** is defined as follows:

```
public interface Vehicle {
    int getId();
}
```

The interface *TrafficRegistrar* specifies methods for capturing the traffic flow and is defined as follows:

```
public interface TrafficRegistrar {
      default void registerLeft(Vehicle v) {
      };

      default void registerRight(Vehicle v) {
      };

      default void deregisterLeft(Vehicle v) {
      };

      default void deregisterRight(Vehicle v) {
      };
}
```

Note: For the interface **`TrafficRegistrar`** an empty implementation (**`TrafficRegistrarEmpty`**) is provided which can be modified if needed.

2. Class `TrafficControllerSimple`

The class **`TrafficControllerSimple`** has to implement the interface **`TrafficController`** such that it becomes a ***monitor*** using **`synchronized`**, **`wait()`**, and **`notify()`** or **`notifyAll()`**. The implementation has to ensure that **only one vehicle at a time can pass** the bridge and all other vehicles have to wait until the bridge is free again.

When the bridge becomes free, an arbitrary waiting vehicle can be selected to enter the bridge. There is no need for ensuring fairness. Please be aware of the fact that the methods of the traffic controller may be called concurrently be multiple threads (vehicles).

The constructor of class *TrafficControllerSimple* has to be implemented as follows in order to allows for injection of a registrar object at runtime (cf., constructor injection).

```
public TrafficControllerSimple(TrafficRegistrar registrar) {
    this.registrar = registrar;
}
```

When implementing the methods of **`TrafficController`** it has to be ensured that the corresponding *register-* and *deregister*-methods of the associated *registrar* object are called such that entry and exit of all vehicles are correctly registered.

Example:

```
public synchronized void enterRight(Vehicle v) {
    ...
    while (...) {
        ...
        wait();
        ...
    }
.   this.registrar.registerRight(v);
    ...
}
```

3. Class `TrafficControllerFair`

The class **`TrafficControllerFair`** implements the interface **`TrafficController`** as monitor using *explicit locks* and *condition variables*. Again, it has to be ensured that only a one vehicle at a time can pass the bridge. To support fairness, the class **`java.util.concurrent.locks.ReentrantLock`** (with argument *fair* set to *true*) should

be used, so that the selection of the car that can pass the bridge next from the set of waiting cars exhibits some degree of fairness.

Again, the constructor of class **`TrafficControllerFair`** has to support *constructor injection* as follows:

```
public TrafficControllerFair(TrafficRegistrar registrar) {
   this.registrar = registrar;
}
```

Again, it has to be ensured that the corresponding *register*- and *deregister*-methods of the corresponding registrar object are called (see Section 2).

4. GUI

To support testing of the controllers, a graphical user interface is provided (**`BridgeGUI.java`**, **`Car.java`**, **`CarWindow.java`**, **`CarWorld.java`**, and directory **`image`**). The GUI can be started using the class **`BridgeGUI`**. (Note: you need to ensure proper access permission for **`image`**.)

Note that the GUI i55s provided for visualization purposes only and not subject of the assignment, as it is does not fully support calling the methods of the traffic controller concurrently by multiple threads. Therefore, you should develop our own test program, for example, using a customized implementation of **`TrafficRegistrar`**.

5. General Comments

- Leave all classes in the default package.
- The class **`TrafficControllerEmpty`** provides an empty implementation of the interface **`TrafficController`** such that **`BridgeGUI`** can be executed.
- Note that the provided GUI only allows for limiting testing of the controllers.
- The class **`TrafficRegistrarEmpty`** provides an empty implementation of the interface **`TrafficRegistrar`**.
- The implementation of **`TrafficControllerSimple`** must not use explicit locks.
- The implementation of **`TrafficControllerFair`** must neither use synchronized-methods nor synchronized-statements.
- Data structures contained in the Java Concurrent Collection must not be used.
- Only the classes **`TrafficControllerSimple`** and **`TrafficControllerFair`** have to be submitted on the online platform.
- When submitting the assignment on the online platform, various tests will be performed to test the functionality of your controllers. In addition, a manual check and a **plagiarism check** may be performed after submission.

6. Submission

Deadline: Wednesday, 6.11.2024 12:00 on the online platform.

You can use the classes and interfaces provided in **TrafficControllerPLC24WS**.zip as basis for your implementation. Do not change the provided interfaces and leave all classes in the default package.

The classes **`TrafficControllerSimple`** and **`TrafficControllerFair`** have to be submitted before the deadline on the online platform after all checks have been passed. Further information is provided in the lectures and on Moodle.