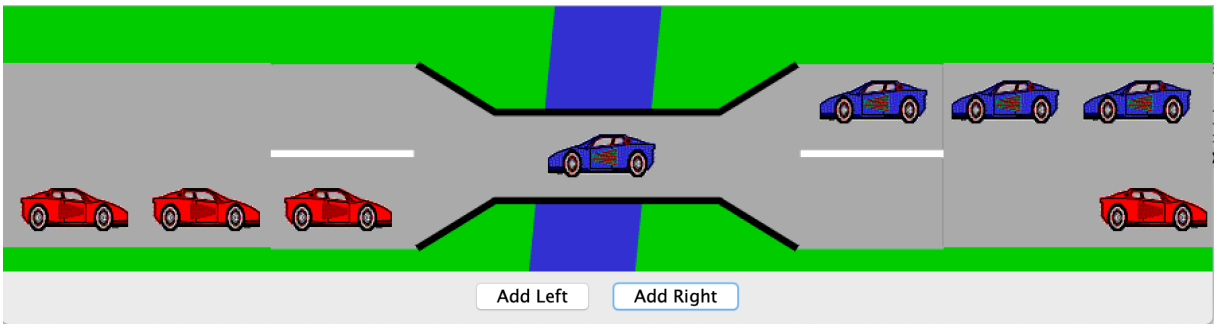
Programmieraufgabe "Traffic Controller"

Es soll ein Traffic Controller entwickelt werden, mit dem der Verkehr über eine einspurige Brücke geregelt werden kann. Es sollen dazu zwei unterschiedliche Implementierungen für den Traffic Controller erstellt werden. Die wesentliche Funktionalität eines Traffic Controllers wird durch das Interface *TrafficController* definiert.



1. Interfaces *TrafficController, Vehicle, TrafficRegistrar*

Das Interface *TrafficController* definiert folgende Methoden mit denen der Zugang zur Brücke geregelt wird.

```
public interface TrafficController {
    void enterRight(Vehicle v); // returns only if bridge is empty; blocks otherwise
    void enterLeft(Vehicle v);   // returns only if bridge is empty; blocks otherwise
    void leaveLeft(Vehicle v);   // must be called after enterRight
    void leaveRight(Vehicle v); // must be called after enterLeft
}
```

Für ein Fahrzeug, das von rechts kommend die Brücke passieren möchte, muss vor dem Auffahren auf die Brücke die Methode *enterRight()*, und vor dem Verlassen der Brücke die Methode *leaveLeft()* aufgerufen werden. Analog dazu muss für ein Fahrzeug, das von links kommend die Brücke passieren möchte, vor dem Auffahren auf die Brücke die Methode *enterLeft()*, und vor dem Verlassen der Brücke die Methode *leaveRight()* aufgerufen werden.

Anmerkung: die Interfaces *TrafficController, Vehicle und TrafficRegistrar* sind vorgegeben und dürfen/können nicht geändert werden.

Das Interface *Vehicle* ist wie folgt definiert:

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

Das Interface *TrafficRegistrar* spezifiziert Methoden zur Erfassung und Protokollierung des Verkehrsflusses über die Brücke durch einen Registrar. Es ist wie folgt definiert:

```
public interface TrafficRegistrar {
    void registerLeft(Vehicle v);
    void registerRight(Vehicle v);
    void deregisterLeft(Vehicle v);
    void deregisterRight(Vehicle v);
}
```

Anmerkung: Das Interface *TrafficRegistrar* muss nicht unbedingt neu implementiert werden. Eine leere Implementierung wird bereitgestellt (Klasse *TrafficRegistrarEmpty*).

2. Klasse *TrafficControllerSimple*

Die Klasse *TrafficControllerSimple* soll das Interface *TrafficController* als Monitor unter Verwendung von *synchronized* und *wait()/notify()/notifyAll()* implementieren. Die Implementierung muss dabei gewährleisten, dass immer nur **maximal ein Fahrzeug** die Brücke passieren kann und alle anderen Fahrzeuge warten, bis die Brücke frei ist. Wenn die Brücke frei wird, kann ein beliebiges Fahrzeug als nächstes die Brücke passieren. (Es muss keine Fairness gewährleistet werden.) Beachten Sie dabei, dass die Methoden des Traffic Controllers von beliebigen vielen Threads (Fahrzeugen) gleichzeitig aufgerufen werden können.

Der Konstruktor der Klasse *TrafficControllerSimple* muss wie folgt implementiert werden, damit ein Registrar via *Constructor Injection* übergeben werden kann.

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

Bei der Implementierung der Methoden des Interface *TrafficController* ist zu gewährleisten, dass jeweils die entsprechende Methode des *registrar*-Objekts aufgerufen wird, damit eine korrekte Erfassung der jeweiligen Ereignisse durch den Registrar gewährleistet werden kann.

Beispiel:

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

3. Klasse TrafficControllerFair

Die Klasse *TrafficControllerFair* soll das Interface *TrafficController* als Monitor unter Verwendung von expliziten *Locks und condition variables* implementieren. Die Implementierung muss dabei wiederum gewährleisten, dass immer nur **maximal ein Fahrzeug** die Brücke passieren kann und alle anderen Fahrzeuge warten, bis die Brücke frei ist. Verwenden Sie zur Implementierung die Klasse *ReentrantLock* (mit Parameter *fair* gleich *true*), damit die Auswahl des Fahrzeugs, das als nächstes die Brücke passieren kann, möglichst fair ist. Beachten Sie dabei wieder, dass die Methoden des Traffic Controllers von beliebigen vielen Threads (Fahrzeugen) gleichzeitig aufgerufen werden können.

Der Konstruktor der Klasse *TrafficControllerFair* muss wieder wie folgt implementiert werden:

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

Bei der Implementierung der Interfacemethoden ist ebenfalls wieder zu gewährleisten, dass jeweils die entsprechende Methode des *registrar*-Objekts aufgerufen wird (siehe 2.).

4. GUI

Zu Testzwecken wird ein grafisches User Interface (GUI) bereitgestellt (*BridgeGUI.java, Car.java, CarWindow.java, CarWorld.java*, sowie das Verzeichnis *image),* welches mit der Klasse *BridgeGUI* gestartet wird. (Achten Sie darauf, dass die Bilder aus dem Verzeichnis *image* gelesen werden können).

Dieses GUI dient rein zu Anschauungszwecken und ist nicht Gegenstand der eigentlichen Aufgabenstellung. Beachten Sie zum Beispiel, dass es mit dem GUI nicht möglich ist, den gleichzeitigen Aufruf der *TrafficController*-Methoden durch beliebig viele Threads zu testen. Zum Testen ihrer *TrafficController*-Implementierungen wird es daher sinnvoll sein, geeignete Testprogramme zu entwickeln, z.B. unter Verwendung einer eigenen Implementierung von *TrafficRegistrar*.

5. Achten Sie auch auf folgende Punkte:

- Verwenden sie Java 1.8 und belassen Sie alle Klassen im Defaultpackage.
- Im Zip-File ist die Klasse *TrafficControllerEmpty* enthalten, die eine Leer-Implementierung (ohne die gewünschte Funktionalität) bereitstellt, damit eine Ausführung von *BridgeGUI* möglich ist.
- Beachten Sie, dass es mit dem grafischen User Interface (BridgeGUI) nicht möglich ist, die vorgeschriebene Funktionalität vollständig zu testen.
- Die bereitgestellte Klasse *TrafficRegistrarEmpty* ist eine Leer-Implementierung des Interface *TrafficRegistrar.*
- Bei der Implementierung von *TrafficControllerSimple* dürfen keine expliziten Locks verwendet werden.
- Bei der Implementierung von *TrafficControllerFair* dürfen keine *synchronized*-Methoden und keine *synchronized*-Statements verwendet werden.
- Es dürfen keine Datenstrukturen aus der *Concurrent Collection* verwendet werden.
- Es sind nur die beiden Implementierungsklassen des Interface *TrafficController* auf der Online-Plattform abzugeben.
- Bei der Abgabe auf der Online-Plattform werden unterschiedliche automatisierte Tests durchgeführt, um die Funktionalität Ihrer Klassen zu testen. Zusätzlich kann nach der Abgabe auf der Online-Plattform eine manuelle Kontrolle, sowie ein Plagiatscheck erfolgen.

6. Abgabemodalitäten

Abgabetermin: **Mittwoch, 13.11.2019 12:00** auf der Online-Abgabeplattform

Verwenden Sie als Basis zur Entwicklung Ihres Programms die im Archiv ***TrafficControllerPLC19WS.zip*** enthaltenen Java Klassen/Interfaces. Ändern Sie keinesfalls die vorgegebenen Interfaces und belassen Sie alle Klassen im Default-Package.

Zur Abgabe sind die Klassen *TrafficControllerSimple* und *TrafficControllerFair* **rechtzeitig vor dem Abgabetermin** auf der Online-Plattform erfolgreich abzugeben. Weitere Informationen dazu in den VU-Einheiten und auf Moodle.