Tentamen
EDA698 – Realtidssystem (Helsingborg)

2016–12–22, 14.00–19.00

Det är tillåtet att använda Java snabbreferens och miniräknare, samt ordbok.
Det går bra att använda både engelska och svenska uttryck för svaren.

Den här tentamen består av två delar: *teori*, frågor 1-5 (18 poäng), och *programmering & design*, frågor 6-7 (12 poäng). För godkänd (betyg 3) krävs sammanlagt ca hälften av alla 30 möjliga poäng samt att det krävs ca en tredjedel av poängen i varje del för sig. För högsta betyg (5) krävs dessutom att en del av poängen uppnås med (del)lösningar till båda uppgifterna i *programmering & design* (preliminärt).

---

1. **System analysis**
   In the course literature we find the following inequality:

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(2^{1/n} - 1)$$

   a) What entities do the symbols C_i, T_i, and n represent? *(1p)*

   b) For the inequality to be applicable, the real-time system involved must have some specific properties. Give example of two such properties. *(1p)*

   c) Which conclusions are we able to draw in the following cases?
      1. The inequality is satisfied.
      2. The inequality is *not* satisfied. *(2p)*

2. **Concurrency correctness**
   When doing code review of a monitor class you come across the following method. Overall, the software you are looking at can be assumed to be concurrency correct. Which line or lines in the following method can you eliminate while still guaranteeing concurrency correctness? *(3p)*

```
1  synchronized int f(int x) throws InterruptedException {
2      attribute1 = x*x;
3      notifyAll();
4      attribute2 = SecretSource.g(x, this);
5      notifyAll();
6      while (!attribute3){ wait();}
7      notifyAll();
8      double z = Math.sin((double)x);
9      while (!attribute3){ wait();}
10     attribute4 = (double)attribute2 * z;
11     notifyAll();
12     while (attribute2==0){ wait()};
13     attribute2++;
14     notify();
15     attribute2++;
16     notifyAll();
17     return attribute2*attribute2;
18 }
```

3. **Deadlock analysis**
Elisa is inspecting a small module of a larger system, which consists of a number of threads (T1, T2 and T3) and monitors (M1, M2, M3, M4 and M5), where packets are moved around and processed. Below are the code segments that show the interactions between the threads and monitors, other parts of the code have been omitted. Currently there is only one instance of each thread and monitor, but Elisa suspects that performance could be improved by introducing another T2 and T3 thread. She is, however, uncertain whether this can cause the system to deadlock.
Can a deadlock happen if she applies the change? Could deadlocks occur even before the change? Motivate your answers with a resource allocation graph. *(3p)*

```
// THREAD CLASSES

public class T1 {
    public void run() {
        m1.move();
    }
}


public class T2 {
    public void run() {
        m2.move();
        // ...
        m3.move();
    }
}


public class T3 {
    public void run() {
        m4.move();
    }
}
```

```
// MONITOR CLASSES

public class M1 {
  public synchronized void move() {
    m2.receive();
  }
  public synchronized void receive() {
    // ..
  }
  public synchronized void fail() {
    // ...
    receive();
  }
}
```

```
// MONITOR CLASSES CONT'D

public class M2 {
  public synchronized void move() {
    m3.receive();
  }
  public synchronized void receive() {
    // ..
  }
  public synchronized void fail() {
    // ...
    receive();
  }
}

public class M3 {
  public synchronized void move() {
    m4.receive();
  }
  public synchronized void receive() {
    // ..
  }
}

public class M4 {
  public synchronized void move() {
    // ...
    m5.receive();
    // ...
    m1.fail();
    // ..
    m2.fail();
  }
  public synchronized void receive() {
    // ..
  }
}


public class M5 {
  public synchronized void receive() {
    // ..
  }
}
```

4. **Semaphores and deadlock**

   a) Show, using schematic code, how to use a semaphore to achieve mutual exclusion between two threads. *(1p)*

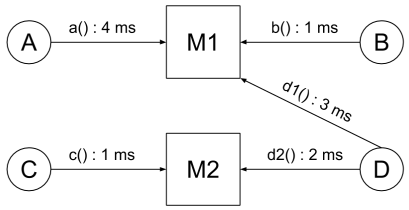
   b) Having mutual exclusion is one of four conditions that must be true for a risk of deadlock to arise. What are the other three? *(2p)*

5. **Schedulability analysis**

   A real-time system consists of four periodic threads, A, B, C, and D. The threads are scheduled using RMS with the basic inheritance protocol applied. The table below shows the worst case execution times (C) and period (T) for each thread.

Thread	C [ms]	T [ms]
A	4	15
B	5	17
C	3	28
D	5	30

   The threads communicate with each other by calling methods in the two monitors M1 and M2 as described in the graph below. Methods are annotated with their corresponding worst case execution time (ms).



   a) What is the blocking factor for each thread? *(2p)*

   b) What is the worst-case response time for each of the threads? *(2p)*

   c) Show that the system is schedulable. *(1p)*

   Hint: $R_i = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$

6. **Programming – Toll station**

   Automatic car toll systems register license plates of passing vehicles and issue fees. Car toll systems are nowadays deployed in several cities in Sweden, such as Gothenburg and Stockholm, in attempts to decrease the number of cars entering the city and therefore lower pollution levels. In this assignment you are to provide part of the implementation of a toll system.

   This particular toll system registers vehicles passing on both lanes of a two lane road (one lane going into the city, the other going out). Two systems are in place to detect and identify vehicles. One is a camera system calibrated to capture license plates. The second way of identification is that a car can be equipped with a transponder unit (a small radio) that registers the paying customer directly at the respective receiver unit of the station. For cars without these the car owner is identified through the license plate and billed.

   The station has also a sensor wire built into the ground for registering an approaching car, so that the identification process can be triggered, and a physical gate system combined with a traffic light and sensor equipped stop line that prevents a car from passing the station without being identified. In case the automatic identification fails, the car is processed manually. This is done by keeping the main lane closed and opening the entrance to a side lane for manual processing, so that other cars can pass.

Your task

Given the (hardware) interfaces for the barriers, sensors, and identification systems (java classes TransponderIDService and LicensePlateIDService implementing IdentificationServiceInterface, interface BarrierControlInterface, and interface SensorInterface in the code skeleton in appendix A, pages 6 and following), you are supposed to develop the core components of the control software for one toll gate according to the following specifications:

1. When a car is registered on the approach sensor wire, both identification services should be activated, i.e., start their identification process. Otherwise they are in "standby".
2. As soon as a transponder unit registration has been successful, the license plate identification process should be aborted, the traffic light switches to green and the gate opens up for the car to pass. The traffic light goes to yellow as soon as the car has started to pass the stop line (light beam is broken) and back to red immediately after the car has passed the stop line fully (light beam clear again). The gate goes back to default ("closed") as soon as the car has passed (also detected by a light beam passage).
3. If no transponder unit registration has happened but the license plate identification responds with "success" within 2 seconds, the transponder registration system should be aborted and the barrier handled as above.
4. If no transponder unit could be registered within 2 seconds, and the license plate identification either reported "failure" (license plate that could not be matched, or was illegible) or did not respond within 2 seconds, an alarm (part of the barrier hardware control) should be triggered for an operator to come and open up the gate to the side lane for the car. The operator then sets back all parts of the system into the default state.
5. While license plate identification and / or transponder unit identification are running, it must still be possible to control other operations (e.g., trigger an abort for either of them).
6. It should be possible to register a new car as soon as the traffic light has been switched to red again, i.e., there is no car at the stop line anymore that could confuse the image processing. It can be assumed that no additional car should pass the sensor wire (approach sensor) before the identification process of the current car is handled (there is an additional traffic light before the wire, that you do not have to care about in this task). However, in case a driver violates this traffic light, the system should still work, i.e., it is acceptable for the system to wait for registration of the detection (the driver simply does not gain anything by driving into the gate early), so that the identification process can be started again after being done (or aborted) for the current car. You are not responsible for catching "bad guys" here, but should make sure that everybody pays!

Implement / extend the central monitor class *TollStation*, whose methods are called by the respective threads.

a) Specify suitable attributes for the class to reflect the state of the toll gate. Add parameters to the constructor if needed.

b) Implement the method *carDetected*, called by an *ApproachHandler* thread, that handles an incoming car. The method should be blocking as long as a previous car is processed.

c) Implement the method *waitForCarDetected*, called by various threads. Should block its caller(s) until a car has been detected.

d) Implement the methods *transponderUnitIdentified* and *licensePlateIdentified*, called by the respective service handler threads, that handle the result of an identification service run.

e) Implement the method *waitForIdentification*, called by the *BarrierHandler* thread, that blocks until there was a result for identification or the identification timed out and handles the necessary steps. Implement also the calling thread's run method. Make sure that the specifications regarding timeout and a second car approaching and other issues are satisfied.

f) Implement the *reset* method, that sets the state of the monitor back so that new cars can be processed.

You may change the parameters for the monitor methods or add other methods, if you feel this is necessary. Please explain your thoughts in such a case. *(9p)*

7. **Design - Toll gate station with automatic "side lane" handling and speed recommendation**
The toll gate you worked with in the previous task proved to handle standard situations quite well; drivers trying to speed into the gate learned that that did not help them, and for cars with transponders it turned out that there were virtually no delays at all. However, in case the transponder system fails / a car does not have any and in addition the image processing does not deliver any useful output, there are significant delays due to the manual processing from the barrier and onward. It is decided that the side lane barrier should also be controlled automatically, i.e., when the identification attempt times out, the side lane barrier should be opened, a traffic light should show a green arrow to the right and everything switches back to normal as soon as the problematic car has been driven into the side lane. Also, it seems that a simple traffic light showing much "red" before the approach sensor is a bit harsh in normal working flow as drivers approaching the toll gate often slow down a little too much, which makes the traffic flow less smooth than desired. There should simply be a more sophisticated "optimal speed indicator" instead of a plain red light, that obviously needs to be connected to the toll station, so that the speed recommendations can be adapted in case there is a long time needed for processing a car.

   Explain, how you would add these functionalities into the system, where you would add new monitors, threads, sensors / hardware access, whether you consider to change communication or synchronization mechanisms between threads, etc!
Illustrate your thoughts with a sketch (schematic class or activity diagram, for example). *(3p)*

End of tasks, code skeleton and course specific Java reference follow

A Code skeleton for task 6

The monitor and thread classes which you should extend

```
/**
 * Supervise one lane of traffic
 */
public class TollStation {
    // Add your attributes here....

    /**
    * Create TollStation object using the provided service classes (interfaces)
    * Change parameters if necessary
    */
    public TollStation( TransponderIDService tS,
                        LicensePlateIDService lPS) { /* YOUR TASK */ }

    /**
     * Register a vehicle's approach and set timer
     */
    public synchronized void carDetected() { /* YOUR TASK */ }

    /**
     * Wait until a car has been detected to trigger the id process
     */
    public synchronized void waitForCarDetected() { /* YOUR TASK */ }

    /**
     * Handle successful identification by transponder unit
     */
    public synchronized void transponderUnitIdentified() { /* YOUR TASK */ }

    /**
     * Handle result of identification attempt by license plate
     */
    public synchronized void licensePlateIdentified( int success) { /* YOUR TASK */ }

    /**
     * Handle identification or timeout
     */
    public synchronized void waitForIdentification() { /* YOUR TASK */ }

    /**
     * Reset state
     */
    public synchronized void reset() { /* YOUR TASK */ }
}

/**
 * BarrierHandler repeatedly waits in TollStation first for a car approaching,
 * then for identification or timeout
 * to occur and handles the traffic light and barrier according to specifications
 */
public class BarrierHandler extends Thread {
    private TollStation logic;
    private BarrierControlInterface bC;
    private SensorInterface;

    public BarrierHandler( TollStation logic, BarrierControlInterface bC,
                           SensorInterface sI) {/*...*/}

    public void run() { /* TOUR TASK */ }
}
```

The identification service classes and their interface
(LicensePlateIDService and TransponderIDService)

```
public class TransponderIDService implements IdentificationServiceInterface {
    static final int ABORTED = 0;
    static final int TU_OK = 1;
    /* ... */
}

public class LicensePlateIDService implements IdentificationServiceInterface {
    static final int ABORTED = 0;
    static final int LP_SUCCESS = 1;
    static final int LP_FAILURE = 2;
    /* ... */
}

public interface IdentificationServiceInterface {
    /**
     * run the identification process, return status
     * 0: process was aborted
     */
    public int runIdentification();

    /**
     * abort the identification process
     */
    public void abortIDProcess();
}
```

The interface for the sensors

```
public interface SensorInterface {
    /**
     * Blocks the calling thread until a car
     * triggers the approach sensor
     */
    public void waitForCar();


    /**
     * Blocks the calling thread until the light beam at the stop line
     * is interrupted
     */
    public void waitForLightBeamBroken();

    /**
     * Blocks the calling thread until the broken light beam at the stop line
     * becomes clear ("closed") again
     */
    public void waitForLightBeamClear();

    /**
     * Blocks the calling thread until the passage of a vehicle has been detected
     * (a light beam has been broken and cleared again)
     */
    public void waitForBarrierPassed();
}
```

The interface for the barrier and light control

```
public interface BarrierControlInterface {
    /**
     * Opens the associated barrier. Non-blocking.
     */
    public void openBarrier();

    /**
     * Closes the barrier. Non-blocking.
     */
    public void closeBarrier();

    /**
     * Set the lamp of the traffic light to green. Non-blocking.
     */
    public void setLightGreen();

    /**
     * Set the lamp of the traffic light to yellow. Non-blocking.
     */
    public void setLightYellow();

    /**
     * Set the lamp of the traffic light to red. Non-blocking.
     */
    public void setLightRed();

    /**
     * Start the alarm. Non-blocking!
     */
    public void startAlarm();

    /**
     * Set the barriers and lights back to "normal". Blocking!
     */
    public void waitForManualReset();
}
```

Other parts of the system (you may change things, but you should not need to)
Other threads

```
/**
 * ApproachHandler repeatedly waits for the sensor wire to report a
 * vehicle approaching the station, then registers the detection in
 * the TollStation logic
 */
public class ApproachHandler extends Thread {
    private TollStation logic;
    private SensorInterface sI;

    public ApproachHandler( TollStation logic, SensorInterface sI) {
        this.logic = logic;
        this.sI = sI;
    }

    public void run() {
        while( !isInterrupted()) {
            sI.waitForCar();
            logic.carDetected();
        }
    }
}
```

```
/**
 * LicensePlateIDHandler waits in the logic for a car to be detected
 * and handles the license plate identification attempt. It
 * then reports the result (if not aborted)
 */
public class LicensePlateIDHandler extends Thread {
    private TollStation logic;
    private LicensePlateIDService lPS;

    public LicensePlateIDHandler( TollStation logic,
                                  LicensePlateIDService lPS) {
        this.logic = logic;
        this.lPS = lPS;
    }

    public void run() {
        int success;
        while( !isInterrupted()) {
            logic.waitForCarDetected();
            success = lPS.runIdentification();
            if( success != 0)
                logic.licensePlateIdentified( success);
        }
    }
}

/**
 * TransponderIDHandler waits in the logic for a car to be detected
 * and handles the transponder unit identification attempt. It
 * then reports the result (if not aborted)
 */
public class TransponderIDHandler extends Thread {
    private TollStation logic;
    private TransponderIDService tS;

    public TransponderIDHandler( TollStation logic,
                                 TransponderIDService tS) {
        this.logic;
        this.tS = tS;
    }

    public void run() {
        int success;
        while( !isInterrupted()) {
            logic.waitForCarDetected();
            success = tS.runIdentification();
            if( success != 0)
                logic.transponderUnitIdentified();
        }
    }
}
```

B Short specific Java reference with potentially helpful methods

Timing, threads, waiting, math, etc

- class Math and class System:
  - public static double Math.random(): generates a random value between 0 and 1.
  - public static long System.currentTimeMillis(): current system time in milliseconds.
- klass Thread:
  - public static void sleep(long t): sets a (the current) thread to sleep for t milliseconds.
  - public static void yield(): withdraw running thread and put in queue for rescheduling.
  - public void join(): wait for the thread to die, i.e., terminate execution.
  - public void start(): start the thread.
  - public void run(): the actual work description for the thread.
- class Object:
  - public final void wait(): sets the calling thread in wait state until notified.
  - public final void wait(long t): sets the calling thread in wait state for t milliseconds. Can be woken up early!
  - public final void notify(): notify the thread first in queue of a change in respective monitor.
  - public final void notifyAll(): notify all waiting threads of a change in the respective monitor.

Short solution sketch

1. a) C_i represents the maximum execution time for thread i during one invocation. T_i stands for the period of thread i, and n is the total number of threads in the system.

   b) The threads must not be able to block each other. The threads must be executing periodically. Costs for context switch, etc, are ignored.

   c) If the inequality is satisfied we can conclude that the system is schedulable. If the inequality is *not* satisfied, we can only conclude that the real-time system described by the inequality *might* be schedulable, but it could just as well be unschedulable. Exact analysis is needed to determine the actual case.

2. The lines of code that can safely be removed without violating concurrency correctness have been commented out in the code below. There are two alternative main solutions with subalternatives when it comes to which notifyAll() you choose to comment out, see comments in the code.

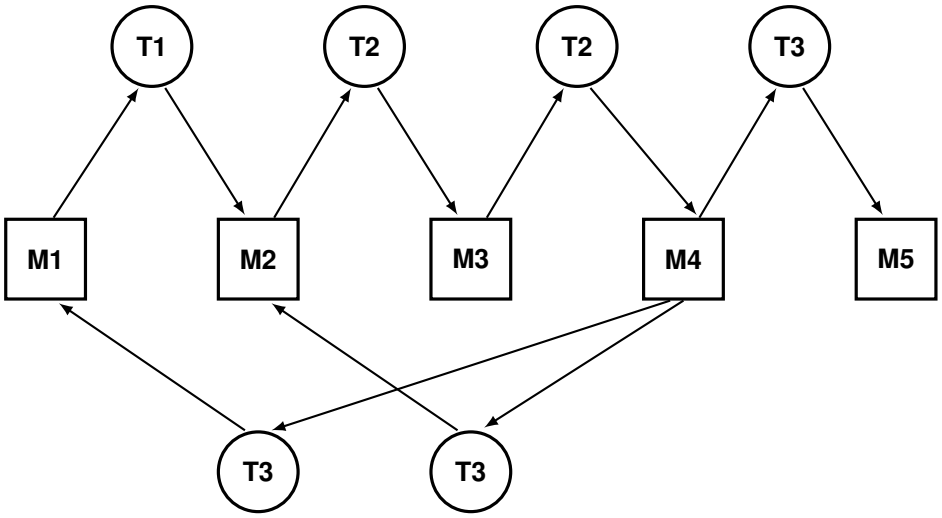
   Solution 1:

```
synchronized int f(int x) throws InterruptedException {
  attribute1 = x*x;
  notifyAll();
  attribute2 = SecretSource.g(x, this);
  notifyAll();
  while (!attribute3) wait();
  notifyAll(); // Could be commented instead of line 11
  double z = Math.sin((double)x);
//  while (!attribute3) wait();
  attribute4 = (double)attribute2 * z;
//  notifyAll(); // Could alternatively comment out line 7
  while (attribute2==0) wait();
  attribute2++;
//  notify();
  attribute2++;
  notifyAll();
  return attribute2*attribute2;
}
```

   Solution 2:

```
synchronized int f(int x) throws InterruptedException {
  attribute1 = x*x;
  notifyAll();
  attribute2 = SecretSource.g(x, this);
  notifyAll(); // Could be commented out instead of line 7
//  while (!attribute3) wait();
//  notifyAll(); // Could alternatively comment out line 5
  double z = Math.sin((double)x);
  while (!attribute3) wait();
  attribute4 = (double)attribute2 * z;
  notifyAll();
  while (attribute2==0) wait();
  attribute2++;
//  notify();
  attribute2++;
  notifyAll();
  return attribute2*attribute2;
}
```

3. The initial system was deadlock free. If another instance of T2 were introduced, deadlocks could occur, as seen by the loops in the resource allocation graph (involving threads T2, T2 and T3, or threads T1, T2, T2 and T3).



4. a) Mutual exclusion:
   Thread 1: `m.take(): ... m.give();`
   Thread 2: `m.take(): ... m.give();`
   The semaphore should initially have the value 1 (true).
   1) Hold and Wait - a thread can reserve a resource and wait for another.
   2) No resource preemption - a thread can not be forced to release held resources.
   3) Circular Wait - thread-resources dependencies must be circular.

5. a) First we need to calculate the blocking factor for each thread i.e. the maximum time each thread can be blocked by lower-priority threads:
   $B_D = 0$ - Since there is no thread with lower priority than D.
   $B_C = max(2,3) = 3$ (direct, indirect)
   $B_B = 3$ (direct)
   $B_A = max(3,1) = 3$ (direct)

   Then we calculate worst case response times:
   $R_A^0 = R_A^1 = 4 + 3 < 15$

   $R_B^0 = 5 + 3 = 8 < 17$
   $R_B^1 = 5 + 3 + \lceil 8/15 \rceil \cdot 4 = 12 < 17$
   $R_B^2 = 5 + 3 + \lceil 12/15 \rceil \cdot 4 = 12 < 17$

   $R_C^0 = 3 + 3 = 6 < 28$
   $R_C^1 = 3 + 3 + \lceil 6/15 \rceil \cdot 4 + \lceil 6/17 \rceil \cdot 5 = 15 < 28$
   $R_C^2 = 3 + 3 + \lceil 15/15 \rceil \cdot 4 + \lceil 15/17 \rceil \cdot 5 = 15 < 28$

   $R_D^0 = 5 < 30$
   $R_D^1 = 5 + \lceil 5/15 \rceil \cdot 4 + \lceil 5/17 \rceil \cdot 5 + \lceil 5/28 \rceil \cdot 3 = 17 < 30$
   $R_D^2 = 5 + \lceil 17/15 \rceil \cdot 4 + \lceil 17/17 \rceil \cdot 5 + \lceil 17/28 \rceil \cdot 3 = 21 < 30$
   $R_D^3 = 5 + \lceil 21/15 \rceil \cdot 4 + \lceil 21/17 \rceil \cdot 5 + \lceil 21/28 \rceil \cdot 3 = 26 < 30$
   $R_D^4 = 5 + \lceil 26/15 \rceil \cdot 4 + \lceil 26/17 \rceil \cdot 5 + \lceil 26/28 \rceil \cdot 3 = 26 < 30$

   b) $R_C < T_C$ for all four threads, therefore is the system schedulable.

6.
```
/**
 * Supervise one lane of traffic
 */
```

```
public class TollStation {
    // Add your attributes here....
    int lPSuccess = LicensePlateIDService.LP_ABORTED;
    boolean lPReported = false;

    boolean tPSuccess = false;
    boolean carDetected = false;

    long timeout = 2000;
    long timer;

    TransponderIDService tS;
    LicensePlateIDService lPS;

    /**
    * Create the Toll Station
    */
    public TollStation( TransponderIDService tS, LicensePlateIDService lPS) {
        this.tS = tS;
        this.lPS = lPS;
    }

    /**
     * Register a vehicle's approach to start the timer
     * Should block until no car is currently processed
     */
    public synchronized void carDetected() {
        while( carDetected) wait();
        carDetected = true;
        timer = System.CurrentTimeMillis() + timeout;

        notifyAll();
    }

    /**
     * Wait until a car has been detected to trigger the id process by license plate
     */
    public synchronized void waitForCarDetected() {
        while( !carDetected) wait();
    }

    /**
     * Handle successful identification by transponder unit
     */
    public synchronized void transponderUnitIdentified() {
        tPSuccess = true;
        lPS.abortIDProcess();
        notifyAll();
    }

    /**
     * Handle result of identification attempt by license plate
     */
    public synchronized void licensePlateIdentified( int success) {
        lPReported = true;
        lPSuccess = success;
        if( lPSuccess == lPS.LP_SUCCESS) tS.abortIDProcess();
        notifyAll();
    }

    /**
     * Handle timeout
```

```
     */
    public synchronized boolean waitForIdentification() {

        long t = System.CurrentTimeMillis();

        while( !tPSuccess && lPSuccess != lPS.LP_SUCCESS && (t < timer)) wait( timer-t);


        if( !tPSuccess) tS.abortIDProcess();
        if( !lPReported ) lPS.abortIDProcess();

        return( tPSuccess || (lPSuccess == lPS.LP_SUCCESS));
    }


    /**
     * Reset state after handling
     */
    public synchronized void reset() {
        lPReported = false;
        tPSuccess = false;
        carDetected = false;
        notifyAll();
    }


}

/**
 * BarrierHandler repeatedly waits in TollStation first for a car approaching,
 * then for identification or timeout
 * to occur and handles the traffic light and barrier according to specifications
 */
public class BarrierHandler extends Thread {
    private TollStation logic;
    private BarrierControlInterface bC;
    private SensorInterface sI;

    public BarrierHandler( TollStation logic,
                           BarrierControlInterface bC,
                           SensorInterface sI) {
        this.logic = logic;
        this.bC = bC;
        this.sI = sI;
    }

    public void run() {
        // Your task!
        boolean success = false;
        while( !isInterrupted() {
            logic.waitForCarDetected();
            success = logic.waitForIdentification();
            if( success) {
                bC.switchLight(bC.GREEN);
                bC.openBarrier();
                sI.waitForLightBeamBroken();
                bC.switchLight(bC.YELLOW);
                sI.waitForLightBeamClear();
                bC.switchLight(bC.RED);
                logic.reset();
                sI.waitForBarrierBeamPassed();
                bC.closeBarrier();
```

```
                } else {
                    bC.startAlarm();
                    bC.waitForManualReset();
                    logic.reset();
                }

            }
        }
    }
```

7. A second barrier is needed, and there would be either a different traffic light connected (with the arrow), or another method in the interface would have to be dedicated to the arrow. Either the two barriers are controlled through their own object (i.e., one instance of a class that implements the main barrier control, and one instance for the side lane), or the second barrier is added to the hw-control (not really modular). For the handling this means that problematic cases are handled faster, i.e., the system can guarantee its cycle time better. For the approach speed recommendation, there would be another monitor (or mailbox thread) needed, in which it is checked through a periodic poll whether a) a car was registered and is processed and b) a "success" was reported for this car. The longer the time period that has passed, the lower the speed recommendation for approaching cars needs to be, before eventually the traffic light has to switch to "red" anyway (if something is broken or the gate is jammed with cars in the side lane). As long as there is a "success" reported within appropriate time frames (less than two seconds) the traffic light is kept green and the recommendation at the maximum speed allowed for approaching the gate.