Carleton University
Department of Systems and Computer Engineering
SYSC 3303A RealTime Concurrent Systems Winter 2026

Assignment 3 — State Machines

Overview

For this assignment, you are going to implement a state machine used to control a pedestrian crossing similar to the one found at Colonel By Drive and the University of Ottawa, shown in Figure 1 on the next page. The specific type of crossing modelled in this assignment is the "Pelican" crossing (Pedestrian Light Controlled crossing). You can find background information about different pedestrian crossings at: https://www.2pass.co.uk/crossing.htm

The purpose of this assignment is to give you practical experience implementing **event-driven state machines**, including:

- hierarchical states,
- well-defined transitions,
- and deterministic behavior suitable for automated testing.

Design and Implementation

You must implement the Pelican crossing controller in **Java** as a finite state machine.

You may use either of the following implementation approaches:

- a **classic enum/switch-based state machine**, or
- the **State Pattern** (object-oriented state classes).

Both approaches are acceptable and will be graded equally, provided the observable behavior of the system is correct.

Starter Code

- A starter code file is provided separately on Brightspace.

  This starter code defines:
  - the required class name,
  - required enums (events, states, and signals),
  - required method signatures.
- **You must use the provided starter code exactly as given.**
- Do not rename classes, enums, enum values, or public methods.
- The TA test harness depends on this interface to automatically test your solution.
- If your code does not compile or does not match the provided interface, it will not be graded.

Test Harness

- You are provided a test harness to make sure that your code runs with no issues.
- You are provided with a subset of the automated JUnit tests. Additional tests will be used during grading by TAs to verify correctness and robustness.

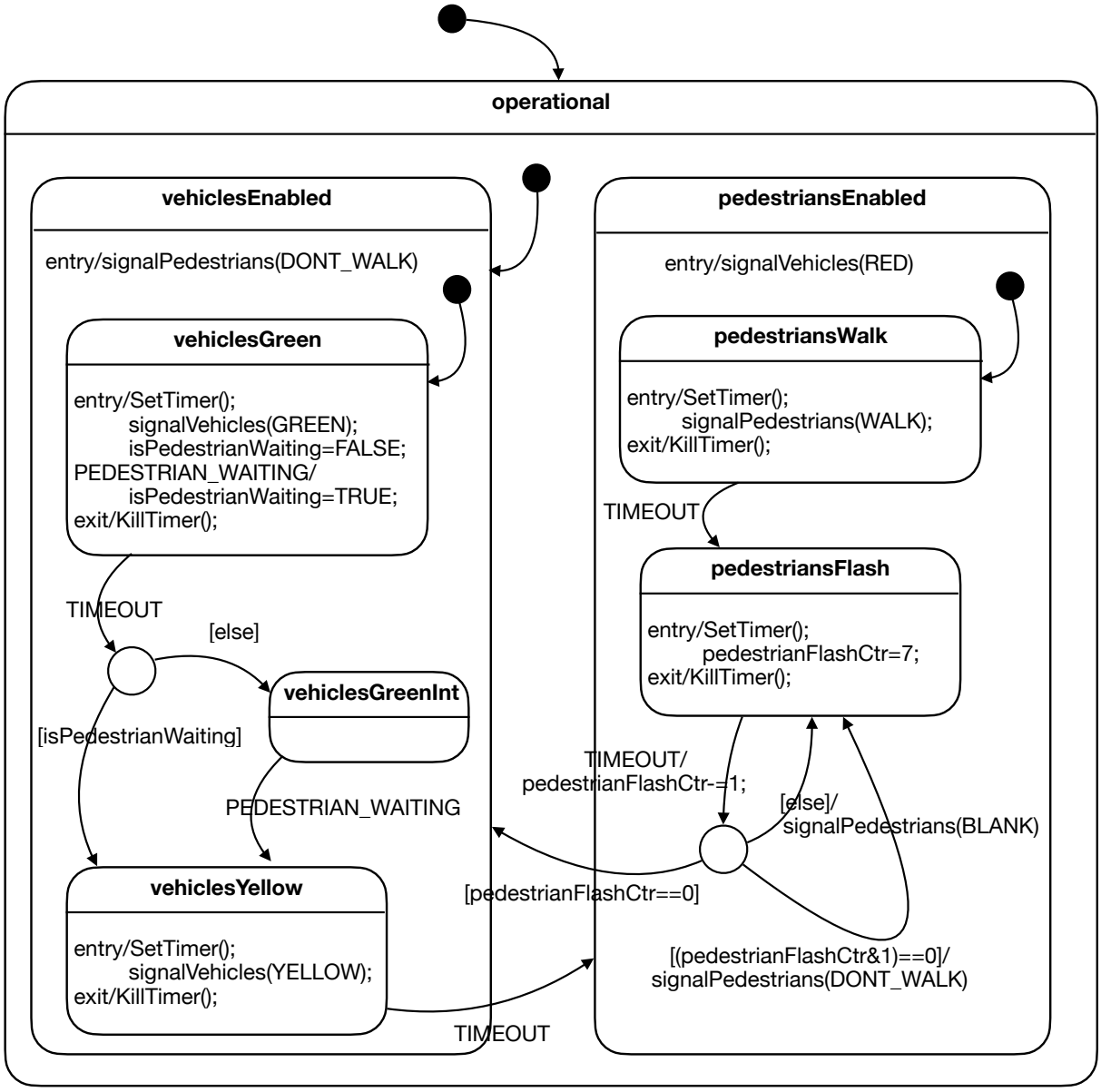

Figure 1: State Machine

See https://barrgroup.com/embedded-systems/how-to/introduction-hierarchical-state-machines. Accessed on Feb 13, 2023

State Machine Behavior

Operation mode

The Pelican crossing alternates between enabling cars and enabling pedestrians.

At a high level, the controller operates in two modes:

- **Operational mode**, in which cars and pedestrians are alternately enabled
- **Offline mode**, in which the crossing displays flashing signals

Within operational mode, the controller enforces:

- a minimum green-light duration for cars,
- a yellow-light transition period,
- a pedestrian walk phase,
- and a pedestrian flashing “Don’t Walk” phase.

Your implementation must correctly model:

- waiting pedestrians,
- interruptible versus non-interruptible car green phases,
- safe transitions between car-enabled and pedestrian-enabled phases,
- and safe entry to and exit from offline mode.

Events

The state machine is driven entirely by **events**.
Events are delivered to the controller through a single dispatch mechanism.

The following events must be supported:

- initialization of the controller,
- pedestrian waiting requests,
- timeout events representing the passage of time,
- transitions into and out of offline mode.

Timeout behavior must be modeled **logically**, using counters or internal variables, rather than relying on wall-clock delays.

Timing and Simulation

You do **not** need to use real-time delays, threads, or timers.
Instead, timing is simulated using timeout events generated by the test harness.
Each timeout event represents a single clock tick and may cause state transitions when internal counters expire.

Your code must be able to:

- handle events in any valid order,
- update internal timers correctly,
- and respond deterministically to injected events.

Signals and Outputs

Your controller must maintain and update:

- the current **car signal** (e.g., red, green, yellow, flashing),
- the current **pedestrian signal** (e.g., walk, don't walk on, don't walk off).

These signals represent the externally observable behavior of the crossing and are used by the test harness to verify correctness.

You may use `System.out.println()` statements for debugging, but the correctness of your solution is determined solely by the state and signal values returned by your code.

Safety Requirements

Your implementation must guarantee that:

- Cars are always shown a **red light** outside of the car-enabled phase
- Pedestrians are always shown "**Don't Walk**" outside of the pedestrian-enabled phase
- Transitions into offline mode leave the system in a safe configuration
- Transitions out of offline mode correctly reinitialize the crossing

The state hierarchy and transition semantics must ensure that these guarantees hold regardless of which state the system is in when an event arrives.

Work Products

You must submit:

1. Java source code implementing the Pelican crossing state machine
2. A README.txt file containing:
   - the names of your source files,
   - brief setup or run instructions (if applicable),
   - a short explanation of your implementation approach (enum-based state machine or State Pattern).

Submitting Assignments

Assignments are to be submitted electronically using BrightSpace. Emailed submissions will not be accepted. See the course outline for the procedure to follow if illness causes you to miss the deadline.