Carleton University
Department of Systems and Computer Engineering
SYSC 3303A RealTime Concurrent Systems Winter 2023

Assignment 4 — State Machines

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

You are to use the state pattern shown in Figure 1 below. If you need some help with the state pattern, I suggest you go to https://sourcemaking.com/design_patterns/state. This site explains the pattern and has sample code to help get you started.

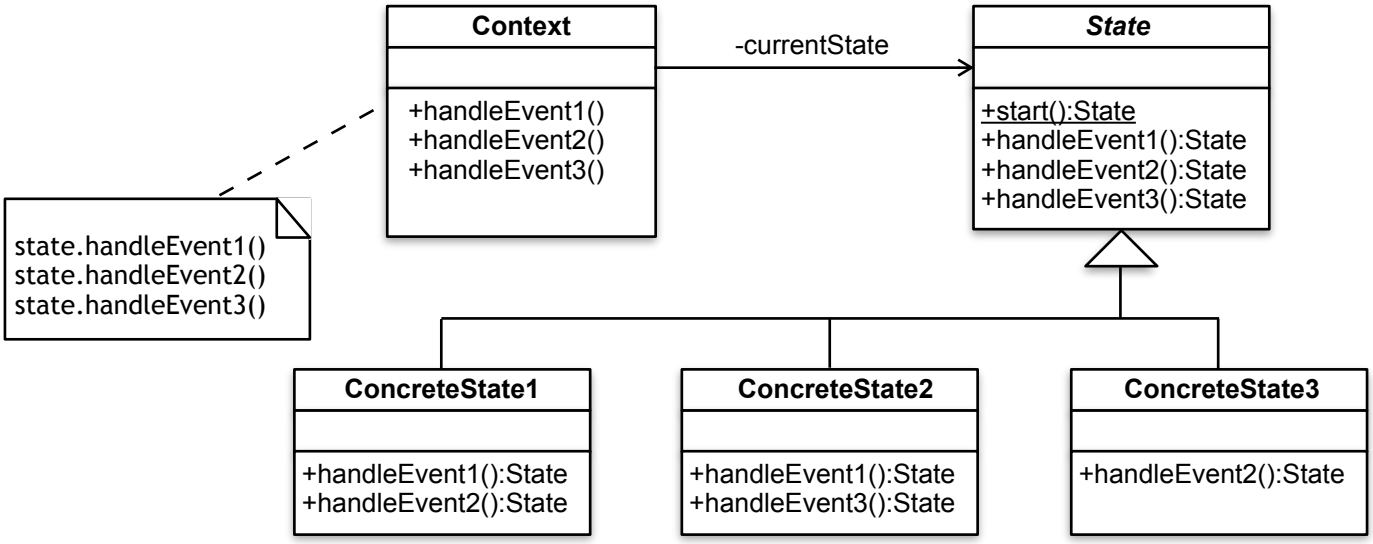

Figure 1: State pattern

1. Be sure that the interface to the state machine is called "`Context`" as this code will be coupled to a test harness used to test the machine. If your code doesn't match the interface exactly, you will get zero for this assignment. See Figure 3 below.
2. There are two events: TIMEOUT and PEDESTRIAN_WAITING. Define the methods `timeout()` and `pedestrianWaiting()` in the `Context` class to signal these events to your state machine. Note that the test harness will not send TIMEOUT events, rather, those events are to be started by your code from your state machine (they are actions after all).
3. Set the timeout for going from a GREEN light to a YELLOW light at 10 seconds, and the timeout for going from a YELLOW light to a RED light to 3 seconds. The WALK light should be enabled for 15 seconds and DONT_WALK should flash on and off for a period of 2 seconds (i.e, one second on, on second off). You can output these actions using `System.out.println()` statements. You don't have to get fancy regarding timing — just use `Thread.sleep()`. Don't worry about timer

drift. However, you have to be able to handle events while you are sleeping so be sure that you can do so. Also ensure that the test program can space events out as necessary.

4. You can assume that your state machine runs until the end of time (there is no exit state).

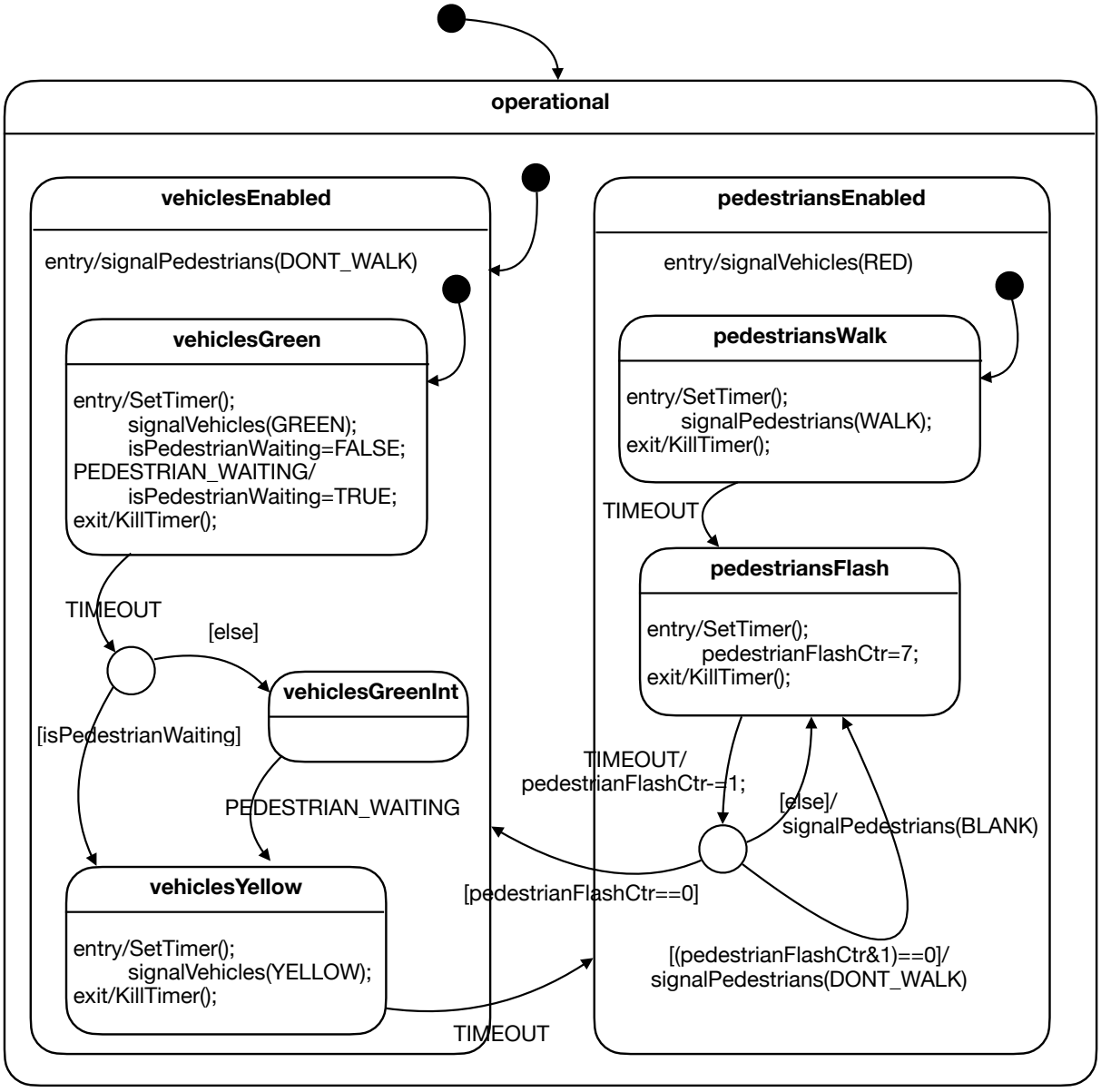

Figure 2: State Machine

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

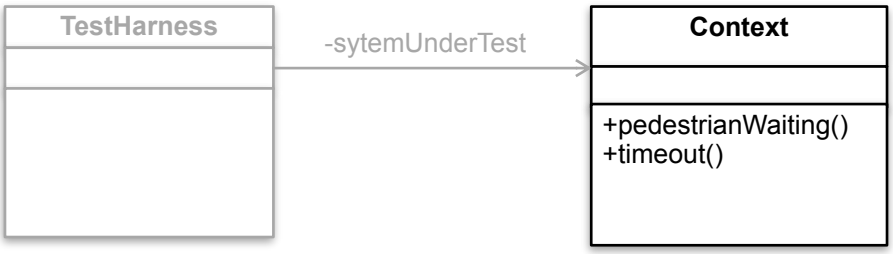

Figure 3: Interface

Notes:

1. Flashing of the DONT_WALK signal is implemented by a one-second timer that turns the signal on and off. `pedestrianFlashCtr` is used to count down. When the counter is even, the light is ON and when odd, off. This is shown using the bit-wise '`&`' operator in Figure 2. If the counter is odd, then the least significant bit is 1 which when ANDed with 1, yields one. This is more efficient than using the modulus operator this way: `pedestrianFlashCtr % 2 == 0`, though any decent compiler would optimize the modulus to a bit-wise AND anyway.
2. There is a defect in the design which will annoy law-abiding pedestrians. For full marks, correct this defect in your implementation. The test harness will check for this.
3. There is second error in the design. Can you spot it? If so, indicate what the error is in the README file included with your submission.
4. Be sure your state machine operates on wall-clock time. The test harness will take a minute or so to run to exercise all the states.

Work products:

1. The Java Source code for the state machine.
1. A "README.txt" file explaining the names of your files, set up instructions, and the answers to the two questions above.
2. Test code that invokes the state machine. The test code need only send events. Actions generated by the state machine should be output using `System.out.println()`

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.