**/\*\***

**\* Test of tick method, of class Intersection.**

**\*/**

@Test

public void testTick() {

System.out.println("tick");

Intersection instance = new Intersection();

Assert.assertEquals("Initial state check", "N:GREEN S:GREEN W:RED E:RED", instance.getState());

instance.tick();

Assert.assertEquals("One tick check", "N:YELLOW S:YELLOW W:RED E:RED", instance.getState());

instance.tick();

Assert.assertEquals("Two tick check", "N:RED S:RED W:RED E:RED", instance.getState());

instance.tick();

Assert.assertEquals("Three tick check", "N:RED S:RED W:GREEN E:GREEN", instance.getState());

instance.tick();

Assert.assertEquals("Four tick check", "N:RED S:RED W:YELLOW E:YELLOW", instance.getState());

instance.tick();

Assert.assertEquals("Five tick check", "N:RED S:RED W:RED E:RED", instance.getState());

instance.tick();

Assert.assertEquals("Six tick check", "N:GREEN S:GREEN W:RED E:RED", instance.getState());

instance.tick();

Assert.assertEquals("Seven tick check", "N:YELLOW S:YELLOW W:RED E:RED", instance.getState());

instance.tick();

Assert.assertEquals("Eight tick check", "N:RED S:RED W:RED E:RED", instance.getState());

instance.tick();

Assert.assertEquals("Nine tick check", "N:RED S:RED W:GREEN E:GREEN", instance.getState());

instance.tick();

Assert.assertEquals("Ten tick check", "N:RED S:RED W:YELLOW E:YELLOW", instance.getState());

instance.tick();

Assert.assertEquals("Eleven tick check", "N:RED S:RED W:RED E:RED", instance.getState());

instance.tick();

Assert.assertEquals("Twelve tick check", "N:GREEN S:GREEN W:RED E:RED", instance.getState());

}

**/\*\***

**\* Test of tick method, of class Signal.**

**\*/**

@Test

public void testTick() {

System.out.println("tick");

Signal instance = new Signal();

Assert.assertEquals("Initial value check", SignalColour.GREEN, instance.getColour());

instance.tick();

Assert.assertEquals("One tick value check", SignalColour.YELLOW, instance.getColour());

instance.tick();

Assert.assertEquals("Two tick value check", SignalColour.RED, instance.getColour());

instance.tick();

Assert.assertEquals("Three tick value check", SignalColour.GREEN, instance.getColour());

instance.tick();

Assert.assertEquals("Four tick value check", SignalColour.YELLOW, instance.getColour());

instance.tick();

Assert.assertEquals("Five tick value check", SignalColour.RED, instance.getColour());

instance.tick();

Assert.assertEquals("Six tick value check", SignalColour.GREEN, instance.getColour());

}

**INTERSECTION**

* ***Constructor Detail***
  + **Intersection**

public Intersection()

Default constructor for the Intersection class, which models a four-way  
intersection where N,S signals remain in sync and E,W signals remain in sync.

Preconditions: N/A.  
Postconditions: N, S, E, and W Signal objects are created,  
and E, W Signal objects are set to Red.  
Cleanup: N/A.

* ***Method Detail***
  + **getState**

public java.lang.String getState()

Retrieve the current colour of all signal lights as a string.

Preconditions: N/A.  
Postconditions: N/A.  
Cleanup: N/A.

**Returns:**

a string indicating the current colour of each signal light.

* + **tick**
  + public void tick()

throws java.lang.RuntimeException

Increments the intersection state to transition one direction-pair  
from green, to yellow, to red (two ticks) and then the other direction-pair  
from red to green (one tick) - a full direction change cycle is three ticks.

Preconditions:  N/A.  
Postconditions: A pair of Signal objects (N/S or E/W) will increment to  
the next colour (GREEN > YELLOW > RED > GREEN) Cleanup: N/A.

**Throws:**

java.lang.RuntimeException - if source code fails a sanity check.

**SIGNAL**

* ***Constructor Detail***
  + **Signal**

public Signal()

Default constructor for the Signal class, which models a three-state  
signal light with the colours GREEN, YELLOW, and RED.

Preconditions: N/A.  
Postconditions: A Signal object is created and the colour is set to GREEN.  
Cleanup: N/A.

* ***Method Detail***
  + **tick**

public void tick()

Changes the Signal object's colour attribute to the next listed colour  
in the order GREEN > YELLOW > RED > GREEN.

Preconditions: N/A.  
Postconditions: The Signal object's colour will have incremented.  
Cleanup: N/A.

* + **getColour**

public [Signal.SignalColour](../lab03/Signal.SignalColour.html) getColour()

Retrieves the current colour of the Signal object.

Preconditions: N/A.  
Postconditions: N/A.  
Cleanup: N/A.

**Returns:**

the current colour of the Signal object.

**ENUMERATORS**

public static enum **Signal.SignalColour**

extends java.lang.Enum<[Signal.SignalColour](../lab03/Signal.SignalColour.html)>

Enumeration of possible signal colours in order: GREEN, YELLOW, RED.

Preconditions: N/A.  
Postconditions: An enumerated list of signal colours will be created.  
Cleanup: N/A.