Solutions to Quiz 2 (May 8, 2017)

Problem 1 (Code review) (**20 points**).

All questions on this quiz refer to the code on the last page, which you may detach.

For each code review comment below, say whether you agree or disagree and explain why in one sentence.

(a) `myColors` should be declared **`private`**.

Solution. AGREE, prevent rep exposure ■

(b) `myColors` should be declared **`static`**.

Solution. DISAGREE, it is the rep for one traffic light, not shared ■

(c) `myColors` should be declared **`final`**.

Solution. AGREE, defend against reassignment ■

(d) `TrafficLight` should override `equals(..)`.

Solution. DISAGREE, mutable types should use reference equality ■

(e) The spec of the constructor suffers from rep exposure.

Solution. DISAGREE, "green traffic light" is an abstract value ■

Problem 2 (AF & RI) (**21 points**).

Where relevant for this problem, **include explicitly any pieces that 6.031 normally assumes implicitly**.

(a) Write the rep invariant for `TrafficLight`.

Solution. `myColors` is one of:

`[GREEN, YELLOW, RED]`, `[YELLOW, RED, GREEN]`, or `[RED, GREEN, YELLOW]`. ■

(b) Write the abstraction function for `TrafficLight`.

Solution. Represents a traffic light that is currently `myColors.get(0)`. ■

(c) Implement `color()` according to its specification.

Solution. `return myColors.get(0).toString().toLowerCase();` ■

Problem 3 (Thread safety) (**24 points**).
Ben wants to make `TrafficLight` a threadsafe ADT. He proposes to change the initialization on line 13:

```
myColors = Collections.synchronizedList(new ArrayList<>());
```

Unfortunately, that change does *not* make `TrafficLight` threadsafe.

(a) Explain clearly and succinctly a concrete example of one race condition that exists in **`nextColor`** with Ben's change. Do not simply mention pieces of code involved in the race, describe it in clear steps.

Solution. When colors are removed and re-added, they can be re-added out-of-order, breaking the cycle order. ■

(b) Explain clearly and succinctly a concrete example of a *second* race condition that exists in **`nextColor`** with Ben's change. This race may involve the same variables or values, but the kind of error that occurs must be different from part (a).

Solution. When colors are removed and re-added, if 3 threads have removed but not yet re-added, a 4th caller will get an exception when it tries to remove. ■

(c) Propose alternative or additional changes to make `TrafficLight` (with just the 1 constructor and 2 methods we've implemented so far) threadsafe. State clearly and specifically all the changes you will make. If suggestions you agreed with in Problem 1 are needed for thread safety, please include those changes here. If your implementation of `color()` in Problem 2 (c) is hard to make threadsafe, revise and simplify it there.

Solution. Make `myColors` private and final.

Implement the monitor pattern, synchronizing `nextColor` and `color`. ■

(d) Write the thread safety argument for `TrafficLight` with your changes.

Solution. `myColors` cannot be reassigned, and it is private and never exposed to clients. All methods are synchronized, so access to `myColors` is protected by the lock on `this`. ■

Problem 4 (Testing) (**20 points**).
We would like to build a test suite for the `TrafficLight` ADT. Recall:

> We build a test suite for an ADT by creating tests for each of its operations. These tests inevitably interact with each other. The only way to test creators, producers, and mutators is by calling observers on the objects that result, and likewise, the only way to test observers is by creating objects for them to observe.

`TrafficLight` currently has three operations.

(a) Complete their type signatures in function notation (*e.g.* `concat` : `Music` × `Music` → `Music`)

```
TrafficLight :

nextColor :

color :
```

Solution.

`TrafficLight` : *void* → `TrafficLight`

`nextColor` : `TrafficLight` → *void*

`color` : `TrafficLight` → `String` ■

(b) Partition the input space of `TrafficLight`:

Solution. The single partition of no inputs. ■

(c) Partition the input space of `nextColor`:

Solution. The traffic light is green, yellow, or red.

The traffic light has cycled (reached green after the initial state) 0, 1, or >1 time(s). ■

(d) Partition the input space of `color`:

Solution. Same as (c). ■

Problem 5 (Abstraction) (**15 points**).

Alyssa looks at the `TrafficLight` ADT. “Nice infinite iterator!” she says.
Ben is confused, so help Alyssa write the code:

```
/**
 * Iterator that produces the sequence of colors seen on traffic lights,
 * starting with green:
 * green -> yellow -> red -> green -> yellow -> red -> green -> ...
 */
public class TrafficLightsIterator implements Iterator<String> {

    TrafficLight myLight;

    public TrafficLightsIterator() {
```

Solution. `myLight = new TrafficLight();` ■

```
    }

    // @return true iff the iteration has more elements
    @Override
    public boolean hasNext() {
```

Solution. `return true;` ■

```
    }

    // @return the next element in the iteration
    // @throws NoSuchElementException if the iteration has no more elements
    @Override
    public String next() {
```

Solution. `String color = myLight.color(); myLight.nextColor(); return color;` ■

```
    }
}
```

You may detach this page. Write your name at the top, and hand in all pages when you leave.

```
package city;

public enum Color {
    RED, ORANGE, YELLOW, GREEN, BLUE, PURPLE
}
```

```
package city;
import static city.Color.*;
import java.util.*;

/**
 * Represents a traffic light that cycles through the sequence
 * green -> yellow -> red -> green -> yellow -> red -> green -> ...
 * one color at a time.
 */
public class TrafficLight {

    List<Color> myColors;

    /** Create a new green traffic light. */
    public TrafficLight() {
        myColors = new ArrayList<>();
        myColors.add(GREEN);
        myColors.add(YELLOW);
        myColors.add(RED);
    }

    /** Modify this traffic light by going to the next color in the sequence. */
    public void nextColor() {
        Color prev = myColors.remove(0); // remaining colors shift to the left
        myColors.add(prev);
    }
    /** @return current color of this traffic light: "green", "yellow", or "red" */
    public String color() {

        // TODO

    }

    // ... perhaps other observers ...
}
```