MIT
6.031: Software Construction
Prof. Rob Miller & Max Goldman

revised Thursday 4th May, 2017, 12:59

Quiz 2 (May 8, 2017)

Your name:____

Your Athena username:__________________________________

You have 50 minutes to complete this quiz. It contains 7 pages (including this page) for a total of 100 points.

The quiz is closed-book and closed-notes, but you are allowed one two-sided page of notes.

Please check your copy to make sure that it is complete before you start. Turn in all pages, together, when you finish. Before you begin, write your name on the top of every page.

Please write neatly. **No credit will be given if we cannot read what you write.**

For questions which require you to choose your answer(s) from a list, do so clearly and unambiguously by circling the letter(s) or entire answer(s). Do not use check marks, underlines, or other annotations – they will not be graded.

Good luck!

Once the quiz begins, you may detach the last page, which contains the code used by all questions. Keep the other pages attached together, and hand in all pages when you leave.

Problem	Points
1: Code review	20
2: AF & RI	21
3: Thread safety	24
4: Testing	20
5: Abstraction	15
Total	100

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`**.

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

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

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

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

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`.

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

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

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.

(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).

(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.

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

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` $\times$ `Music` $\rightarrow$ `Music`)

`TrafficLight :`

`nextColor :`

`color :`

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

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

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

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() {





    }

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





    }

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









    }
}
```

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 ...
}
```