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

revised Tuesday 22nd October, 2019, 12:11

Quiz 1 (October 23, 2019)

Your name:____

Your Kerberos username:____

You have 50 minutes to complete this quiz. It contains 10 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 Kerberos username 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!

Problem	Points
1: Operations	16
2: Implementations	12
3: Specifications	24
4: Tests	24
5: Abstractions	24
Total	100

Train stations, airports, and other transit hubs often have displays that show upcoming departures or arrivals along with other information: a track or gate number, delays, cancellations, etc.

For this quiz, an *information board* is made of several *information board entries*. Each entry has limited space: 16 characters to display a *destination* and 12 characters for a *status*. Both are restricted to upper-case letters, digits, colons, and spaces. For example, a board with three entries:

```
WASHINGTON DC    11:05 AM
LONDON HEATHROW  11:55 AM
HONG KONG        DELAYED
```

In order to show more information, the board cycles each entry through a looping sequence of up to four statuses. For example, if WASHINGTON DC and LONDON HEATHROW have 2-status loops, and HONG KONG has a 3-status loop, then every few seconds the board will update:

```
WASHINGTON DC    ON TIME
LONDON HEATHROW  ON TIME
HONG KONG        NEW DEPRTURE
```

```
WASHINGTON DC    11:05 AM
LONDON HEATHROW  11:55 AM
HONG KONG         1:40 PM
```

```
WASHINGTON DC    ON TIME
LONDON HEATHROW  ON TIME
HONG KONG        DELAYED
```

```
WASHINGTON DC    11:05 AM
LONDON HEATHROW  11:55 AM
HONG KONG        NEW DEPRTURE
```

```
WASHINGTON DC    ON TIME
LONDON HEATHROW  ON TIME
HONG KONG         1:40 PM
```

. . . and so on.

Problems 1–3 in this quiz refer to the code for mutable `MutInfoEntry` starting on page 9. After constructing a new `MutInfoEntry`, the client sets the destination and status cycle. For example:

```
MutInfoEntry train6031 = new MutInfoEntry();
train6031.setDestination("WASHINGTON DC");
train6031.setStatuses(List.of("11:05 AM", "ON TIME"));
```

Problems 4–5 refer to the code for immutable `ImInfoEntry` on page 10. You may detach the code pages.

Problem 1 (Operations) (**16 points**).
Using **`MutInfoEntry` from page 9** and `train6031` as defined in the quiz intro. . .

A client calls `train6031.setStatuses(Collections.emptyList())`.

(a) Why is this incorrect? State a clear and specific reason in one sentence.

(b) Referring to the *spec*: will this call to `setStatuses(..)` throw an exception?

Circle: YES / MAYBE / NO

(c) Referring to the *code*: will this call to `setStatuses(..)` throw an exception?

Circle: YES / MAYBE / NO

(d) And given the provided code: after that call, which operation of `train6031` will no longer work?

(e) Of the four kinds of ADT operations, what kind(s) of operation is that? Leave extra boxes blank:

(f) What will now happen when you call that operation? State a clear and specific result in one sentence.

(g) Why is that result incorrect? State a clear and specific reason in one sentence.

Problem 2 (Implementations) (**12 points**).
Compare each **buggy** `setStatuses(..)` implementation below to the original code in **`MutInfoEntry`** **on page 9**.

(a)

```
public void setStatuses(List<String> statuses) {
    this.statuses = statuses; // oops, buggy line!
    this.iterator = this.statuses.iterator();
}
```

The rep invariant of `MutInfoEntry` is not provided, but you can infer it from the original code. Give a statement from the rep invariant that a client, *without violating the spec*, can now break using `setStatuses(..)`:

Explain in one clear example how they will break the rep invariant, without violating the spec:

(b)

```
public void setStatuses(List<String> statuses) {
    this.statuses = List.copyOf(statuses); // returns an unmodifiable copy
    // oops, missing line!
}
```

Give a statement from the rep invariant that a client, without violating the spec, will now break:

Explain in one clear example how they will break the rep invariant, without violating the spec:
(you can use `train6031` from the quiz intro in your example)

Problem 3 (Specifications) (**24 points**).
Compare each new `setStatuses(..)` spec below to the original spec in **`MutInfoEntry` on page 9**. Differences are in **bold**.

(a)

```
/** Set the statuses to a single empty status if the given list is empty;
 *  otherwise set the statuses to the given list, and the first status in the
 *  list will be displayed next.
 *  @param statuses new statuses, a list of at most 4 strings of at most
 *                  12 upper-case letters, digits, colons, and spaces each  */
```

This spec's precondition is...
Circle one: STRONGER than / WEAKER than / the SAME as / INCOMPARABLE to the original

This spec overall is...
Circle one: STRONGER than / WEAKER than / the SAME as / INCOMPARABLE to the original

If the overall specs are different, give an example input that demonstrates the difference:

(b)

```
/** Set the statuses. The first status in the list will be displayed next.
 *  @param statuses new statuses, a 4-item list of strings of at most
 *                  12 upper-case letters, digits, colons, and spaces each  */
```

This spec's precondition is...
Circle one: STRONGER than / WEAKER than / the SAME as / INCOMPARABLE to the original

This spec overall is...
Circle one: STRONGER than / WEAKER than / the SAME as / INCOMPARABLE to the original

If the overall specs are different, give an example input that demonstrates the difference:

(c)

```
/** Set the statuses. The last status in the list will be displayed next.
 *  @param statuses new statuses, a 1- to 4-item list of strings of at most
 *                  12 upper-case letters, digits, colons, and spaces each  */
```

This spec's precondition is...
Circle one: STRONGER than / WEAKER than / the SAME as / INCOMPARABLE to the original

This spec overall is...
Circle one: STRONGER than / WEAKER than / the SAME as / INCOMPARABLE to the original

If the overall specs are different, give an example input that demonstrates the difference:

Problem 4 (Tests) (**24 points**).
Start building a testing strategy for **`ImInfoEntry` on page 10**.

(a) Give a *valid but useless* partition, with at least 2 subdomains, of the `destination` input to the `ImInfoEntry` constructor. Write one subdomain per box, and leave extra boxes blank:

And say why this partition is *not useful*:

(b) Give an excellent partition of the `destination` input to the `ImInfoEntry` constructor. Write one subdomain per box, and leave extra boxes blank:

And say why this partition is useful:

(c) What is the type signature of the `nextEntry` operation? Input(s) on the left, output(s) on the right:

	→	

(d) What kind(s) of ADT operation is `nextEntry()`? Leave extra boxes blank:

(e) Give an excellent partition of the input(s) to `nextEntry()`. Write your partition in terms of the *abstract value*, not the rep. Write one subdomain per box, and leave extra boxes blank:

Problem 5 (Abstractions) (**24 points**).
Looking at immutable **`ImInfoEntry` on page 10**, the current implementation rotates the items in `statuses`, which means every `ImInfoEntry` in a cycle uses a different list. Change the implementation so that `ImInfoEntry` instances that are part of the same cycle can share one `statuses` list:

(a) Write an excellent declaration for a third field in the rep of `ImInfoEntry`:

And use that field to implement a new `private` constructor, `status()`, and `nextEntry()`.
You may assume the `public` constructor is updated correctly as well.

```
private ImInfoEntry(String destination, List<String> statuses,                ) {

    this.destination = destination;
    this.statuses = statuses;
    this.               =                                        ;
    checkRep();
}

public String status() { return                                        ; }

public ImInfoEntry nextEntry() {
    return new ImInfoEntry(destination, statuses,
                                                                       );
}
```

(b) Write the strongest assertion you can include in `checkRep()` to constrain the new field:

(c) Write a complete, correct abstraction function for your new rep and implementation:

AF(destination, statuses,   ) =

continued on next page →

Suppose we implement `sameValue(..)` by comparing the outputs of the `destination()` and `status()` methods for equality:

```
return destination().equals(that.destination()) && status().equals(that.status());
```

(d) Does `equals(..)` using this `sameValue(..)` define an equivalence relation? Circle: YES / NO

If no, which of the three properties of an equivalence relation does it violate?

(e) Explain in one clear example why this implementation violates observational equality. Give a specific observation or observations clients can make that disagrees with `equals(..)`:

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

```
/** An information board entry that shows a destination (e.g. "WASHINGTON DC")
 *  and cycles through a list of 1 to 4 statuses (e.g. [ "11:05 AM", "ON TIME" ],
 *  or [ "NOW BOARDING", "TRACK 3" ]).
 *  Destinations are limited to 16 characters, and each status to 12 characters.
 *  They may only contain upper-case letters A-Z, digits, colons, and spaces.  */
public class MutInfoEntry {

    private String destination;
    private List<String> statuses;
    private Iterator<String> iterator;

    /** Create a new information board entry with empty destination and
     *  a single empty status. */
    public MutInfoEntry() {
        destination = "";
        statuses = List.of("");
        iterator = statuses.iterator();
    }

    /**  @return the destination */
    public String destination() { return destination; }

    /** @return the next status to display, infinitely cycling through this
     *          info board entry's statuses in order  */
    public String nextStatus() {
        if ( ! iterator.hasNext()) {
            // iterator.next() would throw a NoSuchElementException, so
            //   loop around by getting a fresh iterator for the statuses
            iterator = statuses.iterator();
        }
        return iterator.next();
    }

    /** Set the destination.
     *  @param destination new destination, a string of at most 16 upper-case
     *                     letters, digits, colons, and spaces  */
    public void setDestination(String destination) {
        this.destination = destination;
    }

    /** Set the statuses. The first status in the list will be displayed next.
     *  @param statuses new statuses, a 1- to 4-item list of strings of at most
     *                  12 upper-case letters, digits, colons, and spaces each */
    public void setStatuses(List<String> statuses) {
        this.statuses = List.copyOf(statuses); // returns an unmodifiable copy
        this.iterator = this.statuses.iterator();
    }
}
```

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

```
/** An information board entry that shows a destination (e.g. "WASHINGTON DC")
 *  and current status (e.g. "DELAYED") in a cycle of up to 4 statuses
 *  (e.g. [ "DELAYED", "NEW DEPRTURE", "11:55 AM" ]).
 *  Destinations are limited to 16 characters, and each status to 12 characters.
 *  They may only contain upper-case letters A-Z, digits, colons, and spaces.  */
class ImInfoEntry {

    private final String destination;
    private final List<String> statuses;

    /** Create a new information board entry.
     *  @param destination the destination, a string of at most 16 upper-case
     *                     letters, digits, colons, and spaces
     *  @param statuses statuses, a 1- to 4-item list of strings of at most
     *                  12 upper-case letters, digits, colons, and spaces each,
     *                  where statuses[0] is the status of this info board
     *                  entry; statuses[1] is the status shown next; and so on */
    public ImInfoEntry(String destination, List<String> statuses) {
        this.destination = destination;
        this.statuses = List.copyOf(statuses); // returns an unmodifiable copy
        checkRep();
    }

    private void checkRep() { ... }

    /** @return the destination */
    public String destination() { return destination; }

    /** @return the currently-shown status */
    public String status() { return statuses.get(0); }

    /** @return the entry with the same destination and statuses,
     *          showing the next status in the cycle  */
    public ImInfoEntry nextEntry() {
        List<String> rotated = new ArrayList<>(statuses);
        rotated.add(rotated.remove(0)); // move first status to last
        return new ImInfoEntry(destination, rotated);
    }

    @Override public boolean equals(Object obj) {
        return obj instanceof ImInfoEntry && sameValue((ImInfoEntry)obj);
    }

    private boolean sameValue(ImInfoEntry that) { return ...; }

    @Override public int hashCode() {
        return Objects.hash(destination(), status());
    }
}
```