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:

<table>
<thead>
<tr>
<th>Destination</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>WASHINGTON DC</td>
<td>11:05 AM</td>
</tr>
<tr>
<td>LONDON HEATHROW</td>
<td>11:55 AM</td>
</tr>
<tr>
<td>HONG KONG</td>
<td>DELAYED</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Destination</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>WASHINGTON DC</td>
<td>ON TIME</td>
</tr>
<tr>
<td>LONDON HEATHROW</td>
<td>ON TIME</td>
</tr>
<tr>
<td>HONG KONG</td>
<td>NEW DEPRTURE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Destination</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>WASHINGTON DC</td>
<td>11:05 AM</td>
</tr>
<tr>
<td>LONDON HEATHROW</td>
<td>11:55 AM</td>
</tr>
<tr>
<td>HONG KONG</td>
<td>1:40 PM</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Destination</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>WASHINGTON DC</td>
<td>ON TIME</td>
</tr>
<tr>
<td>LONDON HEATHROW</td>
<td>ON TIME</td>
</tr>
<tr>
<td>HONG KONG</td>
<td>DELAYED</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Destination</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>WASHINGTON DC</td>
<td>11:05 AM</td>
</tr>
<tr>
<td>LONDON HEATHROW</td>
<td>11:55 AM</td>
</tr>
<tr>
<td>HONG KONG</td>
<td>NEW DEPRTURE</td>
</tr>
</tbody>
</table>

... and so on.

Problems 1–3 in this quiz refer to the code for mutable MutInfoEntry starting on page 6. After constructing a new MutInfoEntry, the client sets the destination and status cycle. For example:

```java
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 7. You may detach the code pages.
Problem 1 (Operations) (16 points).
Using MutInfoEntry from page 6 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.

Solution. The empty list does not satisfy the precondition, which requires 1 to 4 elements.

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

Circle: YES / MAYBE / NO

Solution. MAYBE, since the precondition was violated.

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

Circle: YES / MAYBE / NO

Solution. NO.

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

Solution. nextStatus

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

Solution. Observer and mutator.

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

Solution. It will throw a NoSuchElementException.

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

Solution. The postcondition requires it to return the next status, not throw an exception.

Problem 2 (Implementations) (12 points).

Compare each buggy setStatuses(…) implementation below to the original code in MutInfoEntry on page 6.

(a)

```java
public void setStatuses(List<String> statuses) {
    thisstatuses = 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(…):
Solutions to Quiz 1 (October 23, 2019)

Solution. E.g., statuses.size() > 0.

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

Solution. The client has an alias to rep field statuses. They will call setStatuses with a valid list input, then mutate that list; e.g., call clear() so it has zero size.

(b)

```java
public void setStatuses(List<String> statuses) {
    thisstatuses = 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:

Solution. iterator iterates through the elements of statuses.

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)

Solution. In constructing train6031, the call to setStatuses will set the statuses list but fail to change the iterator, which continues to point to an iterator for the initial List.of(" "). Even through the next status should now be "11:05 AM", calling nextStatus() will return "."

Problem 3 (Specifications) (24 points).

Compare each new setStatuses(...) spec below to the original spec in MutInfoEntry on page 6. Differences are in bold.

(a)

```java
/** 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:

Solution. The new precondition is WEAKER and the spec is STRONGER. The empty list is now a legal input.

(b)
Solutions to Quiz 1 (October 23, 2019)

4

Solutions to Quiz 1 (October 23, 2019)

/** 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:

Solution. The new precondition is STRONGER and the spec is WEAKER. E.g. List.of("ARRIVING") is no longer a legal input.

(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:

Solution. The new precondition is the SAME and the spec is INCOMPARABLE.

E.g. for List.of("BOARDING", "TRACK 1"). the next status is now "TRACK 1".

Problem 4 (Tests) (24 points).
Start building a testing strategy for ImInfoEntry on page 7.

(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:

Solution. destination: contains the character Q, does not contain Q.

And say why this partition is not useful:

Solution. Implementation is very unlikely behave differently depending on the presence of this particular character.

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

Solution. destination.length(): 0, 1, 2–15, 16

And say why this partition is useful:
Solution. Includes boundary values where behavior may be different or incorrect.

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

→

Solution. ImInfoEntry → ImInfoEntry

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

Solution. Producer.

(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:

Solution. this has a status cycle of size: 1, 2–3, 4

Problem 5 (Abstractions) (24 points).
Looking at immutable ImInfoEntry on page 7, 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:

Solution. E.g., private final int currentStatus;

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,
    );
}

Solution. E.g.:
Add argument int current
this.currentStatus = current;
return statuses.get(currentStatus);
And call with (currentStatus + 1) % statuses.size()  

(b) Write the strongest assertion you can include in checkRep() to constrain the new field:
Solution. Given the choices above:

```
assert 0 <= currentStatus && currentStatus < statuses.size();
```

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

**Solution.**

```
AF(destination, statuses, currentStatus) =
```

the info board entry with destination destination and current status statuses[currentStatus], then looping through statuses[currentStatus+1..]+statuses[..currentStatus]

*Note: both current status and entire cycle are part of the abstract value, and a correct abstraction function will be unambiguous when statuses contains duplicates.*

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

**Solution.** YES.

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(..)`:

**Solution.** Given a pair of entries that have the same destination and current status but different next statuses in the cycle, `equals(..)` will return true, but calling `nextEntry()` will return entries whose `status()` values are different.

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();
```
public String destination() {
    return destination;
}

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();
}

public void setDestination(String destination) {
    this.destination = destination;
}

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());
}