# CS 61B Spring 2019

# Iterators and Iterables

Discussion 5: February 18, 2018

## 1 Iterators Warmup

1. If we were to define a class that implements the interface `Iterable<Integer>`, what method(s) would this class need to define? Write the function signature(s) below.

   ```
   public Iterator<Integer> iterator()
   ```

2. If we were to define a class that implements the interface `Iterator<Integer>`, what method(s) would this class need to define? Write the function signature(s) below.

   ```
   public boolean hasNext()
   public Integer next()
   ```

3. What's one difference between `Iterator` and `Iterable`?
   We use Iterable to support the enhanced for loop, and Iterator to define custom iterators that maintain some state about iteration.

## 2 OHQueue

The goal for this question is to create an iterable Office Hours queue. We'll do so step by step.

The code below for `OHRequest` represents a single request. Like an `IntNode`, it has a reference to the `next` request. `description` and `name` contain the description of the bug and name of the person on the queue.

```
1  public class OHRequest {
2      public String description;
3      public String name;
4      public OHRequest next;
5
6      public OHRequest(String description, String name, OHRequest next) {
7          this.description = description;
8          this.name = name;
9          this.next = next;
10     }
11 }
```

First, let's define an iterator. Create a class `OHIterator` that implements an iterator over `OHRequest` objects that only returns requests with good descriptions. Our `OHIterator`'s constructor will take in an `OHRequest` object that represents the first `OHRequest` object on the queue. We've provided a function, `isGood`, that accepts a `description` and says if the description is good or not.

```java
import java.util.Iterator;
import java.util.NoSuchElementException;

public class OHIterator implements Iterator<OHRequest> {
    OHRequest curr;

    public OHIterator(OHRequest original) {
        curr = original;

    }

    public boolean isGood(String description) {
        return description != null && description.length() > 5;
    }

    @Override
    public boolean hasNext() {
        while (curr != null && !isGood(curr.description)) {
            curr = curr.next;
        }
        return curr != null;
    }

    @Override
    public OHRequest next() {
        if (!hasNext()) {
            throw new NoSuchElementException();
        }
        OHRequest currRequest = curr;
        curr = curr.next;
        return currRequest;
    }
}
```

Now, define a class `OfficeHoursQueue`. We want our `OfficeHoursQueue` to be iterable, so that we can process `OHRequest` objects with good descriptions. Our constructor will take in an `OHRequest` object representing the first request on the queue.

```java
import java.util.Iterator;
import java.util.NoSuchElementException;

public class OfficeHoursQueue implements Iterable<OHRequest> {
    OHRequest queue;

    public OfficeHoursQueue (OHRequest queue) {
        this.queue = queue;
    }
```

```
10
11      @Override
12      public Iterator<OHRequest> iterator() {
13          return new OHIterator(queue);
14      }
15  }
```

Fill in the main method below so that you make a new `OfficeHoursQueue` object and print the names of people with good descriptions. Note : the main method is part of the `OfficeHoursQueue` class.

```
1  public class OfficeHoursQueue ... {
2      ....
3
4      public static void main(String[] args) {
5          OHRequest s1 = new OHRequest("Failing my test for get in arrayDeque, NPE", "Pam", null);
6          OHRequest s2 = new OHRequest("conceptual: what is dynamic method selection", "Michael", s1);
7          OHRequest s3 = new OHRequest("git: what does checkout do.", "Jim", s2);
8          OHRequest s4 = new OHRequest("help", "Dwight", s3);
9          OHRequest s5 = new OHRequest("debugging get(i)", "Creed", s4);
10         OfficeHoursQueue q = new OfficeHoursQueue(s5);
11         for (OHRequest o : q) {
12             System.out.println(o.name);
13         }
14     }
```

# 3 Thank u, next

Now that we have our `OfficeHoursQueue` from problem 2, we'd like to add some functionality. We've noticed that occasionally in office hours, the system will put someone on the queue twice. It seems that this happens whenever the description contains the words "thank u." To combat this, we'd like to define a new iterator, `TYIterator`.

If the current item's description contains the words "thank u," it should skip the next item on the queue. As an example, if there were 4 `OHRequest` objects on the queue with descriptions `["thank u", "thank u", "im bored", "help me"]`, calls to `next()` should return the 0th, 2nd, and 3rd `OHRequest` objects on the queue. Note: we are still enforcing good descriptions on the queue as well!

hint : to check if a description contains the words "thank u", you can use

```
curr.description.contains("thank u")
```

```
1
2  public class TYIterator extends OHIterator {
3      public TYIterator(OHRequest original) {
4          super(original);
5      }
6
7      @Override
8      public OHRequest next() {
9          OHRequest result = super.next();
10         if (curr != null && curr.description.contains("thank u")) {
11             curr = curr.next;
12         }
13         return result;
14     }
15 }
```