1 OHQueue

Meshan is designing the new 61B Office Hours Queue. The code below for OHRequest represents a single request. 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, and isSetup marks the ticket as being a setup issue or not.

```java
public class OHRequest {
    public String description;
    public String name;
    public boolean isSetup;
    public OHRequest next;

    public OHRequest(String description, String name, boolean isSetup, OHRequest next) {
        this.description = description;
        this.name = name;
        this.isSetup = isSetup;
        this.next = next;
    }
}
```

(a) Create a class `OHIterator` that implements an `Iterator` over `OHRequest`s and only returns requests with good descriptions (using the `isGood` function). Our `OHIterator`'s constructor takes in an `OHRequest` that represents the first `OHRequest` on the queue. If we run out of office hour requests, we should throw a `NoSuchElementException` when our iterator tries to get another request, like so:

```
throw new NoSuchElementException();
```

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

    public OHIterator(OHRequest request) {

        curr = request;

    }

    public static boolean isGood(String description) { return 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;

    }
}
```

(b) Define a class `OHQueue` below: we want our `OHQueue` to be `Iterable` so that we can process `OHRequest` objects with good descriptions. Our constructor takes in the first `OHRequest` object on the queue.

```
public class OHQueue implements Iterable<OHRequest> {
    private OHRequest request;
    public OHQueue(OHRequest request) {

        this.request = request;
    }

    @Override
    public Iterator<OHRequest> iterator() {
        return new OHIterator(request);
    }
}
```

(c) Suppose we notice a bug in our office hours system: if a ticket's description contains the words "thank u", it is put on the queue twice. 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, because we know the next item is an accidental duplicate from our buggy system. 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.

To check if a `String s` contains the substring "thank u", you can use: `s.contains("thank u")`

*Hint - we've already enforced good descriptions with our regular `OHIterator`. Using inheritance, how can we reuse that functionality without repeating ourselves? Also, notice that `OHIterator`'s instance variables are **private**, so we can't access them from subclasses of `OHIterator`.*

```
public class TYIterator extends OHIterator {
    public TYIterator(OHRequest queue) {

        super(queue);
    }

    @Override
    public OHRequest next() {
        OHRequest result = super().next;
        if (result != null && result.description.contains("thank u")) {

            result = super.next()
        }
        return result;
    }
}
```

(d) Now assume the OHQueue uses a TYIterator as its iterator. Fill in the blanks to print only the names of tickets from the queue beginning at s1 with good descriptions, skipping over duplicate descriptions that contain "thank u". What would be printed after we run the main method?

```java
public static void main(String[] args) {
    OHRequest s5 = new OHRequest("I deleted all of my files, thank u", "Elana", true, null);
    OHRequest s4 = new OHRequest("conceptual: what is Java", "Mihir", false, s5);
    OHRequest s3 = new OHRequest("git: I never did lab 1", "Kevin", true, s4);
    OHRequest s2 = new OHRequest("help", "Angel", false, s3);
    OHRequest s1 = new OHRequest("no I haven't tried stepping through", "Ashley", false, s2);

    OHQueue q = new OHQueue(s1);

    for (OHRequest o : q) {
        System.out.println(o.name);
    }
}
```

TYIterator already takes care of "thank u"s

(e) Meshan would like to find a way to prioritize setup tickets on the queue so that they appear at the top. He wants to implement this based on the isSetup field of each OHRequest, but sometimes students forget to set it to true, so he decides to use description as backup to break ties.

Fill in the compare method of OHRequestComparator below. First, if one but not both of the OHRequests have their isSetup set to **true**, the one with isSetup set to **true** should take priority (ie. earlier on the queue). If both or neither of the OHRequests have their isSetup set to **true**, tiebreak with the description: the description has to **exactly match "setup"** in order to be counted as a setup issue. If both requests have such descriptions, it's a true tie and return 0.

*Hint - if o1 is prioritized over o2, then o1 is considered **less** than o2*

```java
public class OHRequestComparator implements Comparator<OHRequest> {
    @Override
    public int compare(OHRequest o1, OHRequest o2) {
        // use as many lines as you need
        boolean o1DescMatchesSetup = o1.description.equals("setup");
        boolean o2DescMatchesSetup = o2.description.equals("setup");
        if (o1.isSetup = true && o2.isSetup = false){
            return -1;
        } else if (o1.isSetup = false && o2.isSetup = true){
            return 1;
        } else if ( o1DescMatchesSetup && !o2DescMatchesSetup){
            return -1;
        } else if (!o1DescMatchesSetup && o2DescMatchesSetup){
            return 1;
        }
        return 0;
    }
}
```