CS 61B Spring 2025

Iterators, Asymptotics

Discussion 04: February 17, 2025

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.

```
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 ____________________________________________________ {
    private OHRequest curr;

    public OHIterator(OHRequest request) {

        ________________________________________;

    }

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

    @Override
    _________________ _________________ ______________________________ {

        while (_____________________________________________________________) {

            ___________________________________;

        }

        ___________________________________;

    }

    @Override
    _________________ _________________ ______________________________ {

        if (___________________________________) {

            throw __________ ______________________________________________;

        }

        ___________________________________;

        ___________________________________;

        ___________________________________;

    }
}
```

(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 ____________________________________________________ {
    private OHRequest request;
    public OHQueue(OHRequest request) {

        ___________________________________________;
    }

    @Override
    _________________ _________________ ______________________________ {

        ___________________________________________;
    }
}
```

(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 adjust our implementation of **OHIterator**'s **next()**.

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")**

```
@Override
_____________ ___________ _____________ {

    if (_____________________________________) {

            throw __________ ________________________________________________;

    }
    ____________________________________;

    ____________________________________;

    if (_____________________________________________) {

        ______________________;
    }
    return ______________________;
}
```

(d) Now assume the **OHQueue** uses the modified **OHIterator** 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?

```
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", "Ethan", false, s3);
    OHRequest s1 = new OHRequest("no I haven't tried the debugger", "Ashley", false, s2);

    OHQueue q = ______________________________________;

    for (_________________________________________________) {

        ____________________________________________________;
    }
}
```

2 Asymptotics

(a) Say we have a function **findMax** that iterates through an unsorted int array one time and returns the maximum element found in that array. Give the tightest lower and upper bounds ($\Omega(\cdot)$ and $O(\cdot)$) of **findMax** in terms of **N**, the length of the array. Is it possible to define a $\Theta(\cdot)$ bound for **findMax**?

(b) Give the worst case and best case runtime in terms of M and N. Assume **ping** runs in $\Theta(1)$ and returns an **int**.

```
for (int i = N; i > 0; i--) {
    for (int j = 0; j <= M; j++) {
        if (ping(i, j) > 64) { break; }
    }
}
```

(c) Below we have a function that returns **true** if every **int** has a duplicate in the array, and **false** if there is any unique int in the array. Assume **sort(array)** is in $\Theta(N \log N)$ and returns **array** sorted.

```
public static boolean noUniques(int[] array) {
    array = sort(array);
    int N = array.length;
    for (int i = 0; i < N; i += 1) {
        boolean hasDuplicate = false;
        for (int j = 0; j < N; j += 1) {
            if (i != j && array[i] == array[j]) {
                hasDuplicate = true;
            }
        }
        if (!hasDuplicate) return false;
    }
    return true;
}
```

Give the worst case and best case runtime where $N =$ **array.length**.