CS 61B Spring 2021

Iterators and Iterables

Discussion 5: February 15, 2021

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`?

```
An object that is Iterable can create and Iterator object that can iterate over
the contents of the Iterable object.
For example, suppose we have a Class CSGroup which implements Iterable<T>. And we
create an object sSGroupJr from the Class. sSGroupJr can create an Iterator object
called sSGroupJrIterator. sSGroupJrIterator can iterate over the students in the
sSGroupJr group.
```

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. If we run out of office hour requests, we should throw a `NoSuchElementException` when our iterator tries to get another request.

```
import java.util.Iterator;
public class OHIterator  implements Iterator<OHRequest>  {
    OHRequest curr;

    public OHIterator(OHRequest queue) {
        curr = queue
    }

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

    @Override
    public boolean hasNext() {
        while(curr != null && !isGood(curr.description)){     用while循环来筛选掉description太少的节点
            curr = curr.next;
        }
        if (curr == null) {
            return false;
        }
        return true;
    }
    @Override
    public OHRequest next() {
        if (!hasNext()) {
            throw new NoSuchElementException();
        }

        OHRequest currRequest = curr;
}       curr = curr.next;
        return currRequest;
    }
```

Now, define a class `OHQueue`. We want our `OHQueue` 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.

```
import java.util.Iterator;
public class OHQueue _implements Iterable<OHRequest>_____ {

    OHRequest queue;

    public OHQueue (OHRequest queue) {
         this.queue = queue;
    }

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


}
```

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

```
public class OHQueue ... {
    ....

    public static void main(String [] args) {
        OHRequest s5 = new OHRequest("I deleted all of my files", "Allyson", null);
        OHRequest s4 = new OHRequest("conceptual: what is Java", "Omar", s5);
        OHRequest s3 = new OHRequest("git: I never did lab 1", "Connor", s4);
        OHRequest s2 = new OHRequest("help", "Hug", s3);
        OHRequest s1 = new OHRequest("no I haven't tried stepping through", "Itai", s2);

        OHQueue q = new OHQueue(s1);

        for (_OHRequest o__ : ___q_____________) {
               System.out.println(o.name);
        }

}
```

3 Thank u, next

Now that we have our `OHQueue` from problem 2, we'd like to add some functionality. We've noticed a bug in our office hours system: whenever a ticket's description contains the words "thank u", that ticket 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. 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")
```

```
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();
      }
    }
```

不能写result = result.next。假设result的节点是1，result.next节点为2。即便这里写了result =result.next，把当前result节点设置为2，但这并没有改变OHIterator的节点记录，OHIterator记录的下一个节点还是2，但下一轮iterator还是会用OHIterator得到result的节点为2。2这个节点就会被打印2次。

```
}
```

4 Senior Class *Extra*

For each line in the `main` method of our `testPeople` class, if something is printed, write it next to the line. If the line results in an error, write next it whether it is a compile time error or runtime error, and then proceed as if that line were not there.

```
1  public class Person {
2      public String name;
3      public int age;
4
5      public Person(String name, int age) {
6          this.name = name;
7          this.age = age;
8      }
9
10     public void greet(Person other) {System.out.println("Hello, " + other.name);}
11 }
12
13
14 public class Grandma extends Person {
15
16     public Grandma(String name, int age) {
17         super(name, age);
18     }
19
20     @Override
21     public void greet(Person other) {System.out.println("Hello, young whippersnapper");}
22
23     public void greet(Grandma other) {System.out.println("How was bingo, " + other.name + "?");}
24 }
25
26 public class testPeople {
27     public static void main(String[] args) {
28         Person n = new Person("Neil", 12);
29         Person a = new Grandma("Ada", 60);
30         Grandma v = new Grandma("Vidya", 80);
31         Grandma al = new Person("Alex", 70);   //Compile time error
32         n.greet(a); // Hello, Ada
33         n.greet(v); // Hello, Vidya
34         v.greet(a); // Hello, young whippersnapper
35         v.greet((Grandma) a);  // How was bingo, Ada?
36         a.greet(n);  // Hello, young whippersnapper
37         a.greet(v);  // Hello, young whippersnapper
38         ((Grandma) a).greet(v);  // How was bingo, Vidya?
39         ((Grandma) n).greet(v);   // Runtime error
40     }
41 }
```