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`?

   Iterators are the actual object we can iterate over. An example of this would be an array - we can iterate through the object in the array. Iterables are object that can produce an iterator that somehow iterate over their contents. If we have a class called CS61B, it itself cannot be iterated over, but it can produce an iterator that iterates over all of the students in the class.
Iterators and Iterables

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.

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

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

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.

```java
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)) {
            curr = curr.next;
        }
        if (curr == null) {
            return false;
        }
        return true;
    }

    @Override
    public OHRequest next() {
        return curr;
    }
}
```
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);

        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")

```java
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;
    }
}
```
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.

```java
public class Person {
    public String name;
    public int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public void greet(Person other) {System.out.println("Hello, " + other.name);}
}

public class Grandma extends Person {
    public Grandma(String name, int age) {
        super(name, age);
    }

    @Override
    public void greet(Person other) {System.out.println("Hello, young whippersnapper");}
    public void greet(Grandma other) {System.out.println("How was bingo, " + other.name + "?");}
}

public class testPeople {
    public static void main(String[] args) {
        Person n = new Person("Neil", 12);
        Person a = new Grandma("Ada", 60);
        Grandma v = new Grandma("Vidya", 80);
        Grandma al = new Person("Alex", 70); // Compile time error
        n.greet(a); // "Hello Ada"
        n.greet(v); // "Hello Vidya"
        v.greet(a); // "Hello, young whippersnapper"
        v.greet((Grandma) a); // "How was bingo, Ada?"
        a.greet(n); // "Hello, young whippersnapper"
        a.greet(v); // "Hello, young whippersnapper"
        ((Grandma) a).greet(v); // "How was bingo, Vidya?"
        ((Grandma) n).greet(v); // Runtime error
    }
}
```