Comparators and Iterators

Exam-Level 04

Announcements

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
	2/12 Project 1B Due Weekly Survey Due			2/15 Midterm 1 (7-9pm)		
		2/20 Lab 4 Due Project 1C Due				

Content Review

Comparables

Comparables are things that **can be compared with each other**.
Any class could implement this interface.
Defines the notion of being “less than” or “greater than”.

```
public class Dog implements Comparable<Dog> {
    private String name;
    private int size;
    @Override
    public int compareTo(Dog otherDog) {
        return this.size - otherDog.size;
    }
}
```


Comparables

Can't use < and > directly on dog objects - undefined for them!
Instead, use the `compareTo` method instead.

~~`if (d1 < d2) {`~~

~~`} else`~~

~~`}`~~

```
if (d1.compareTo(d2) < 0) {
    // Dog 1 "less than" dog
} else {

}
```


Comparators

Comparators are things that **can be used to compare two objects.** Think of it as a "seesaw". Comparables are the things sitting on the seesaw. Not the seesaw itself!

```
public interface Comparator<T> {
    int compare(T o1, T o2);
}

public class DogComparator<Dog> implements Comparator<Dog> {
    public int compare(Dog d1, Dog d2) {
        return d1.size - d2.size;
    }
}
```


Why does compare/compareTo return an integer?

The `Comparator` interface's `compare` function takes in two objects of the same type and outputs:

- A negative integer if `o1` is "less than" `o2`
- A positive integer if `o1` is "greater than" `o2`
- Zero if `o1` is "equal to" `o2`

For `Comparable`, it is the same, except `o1` is `this`, and `o2` is the `other` object passed in.

Think of it as subtracting!

```
compare(T o1, T o2) -> o1 - o2
o1 - o2 < 0 -> o1 < o2
o1 - o2 > 0 -> o1 > o2
o1 - o2 = 0 -> o1 = o2
```

```
o1.compareTo(o2) -> o1 - o2
o1 - o2 < 0 -> o1 < o2
o1 - o2 > 0 -> o1 > o2
o1 - o2 = 0 -> o1 = o2
```


The Iterator & Iterable Interfaces

Iterators are objects that can be iterated through in Java (in some sort of loop).

```
public interface Iterator<T> {
    boolean hasNext();
    T next();
}
```

Iterables are objects that can produce an iterator.

```
public interface Iterable<T> {
    Iterator<T> iterator();
}
```


The Iterator & Iterable Interfaces

The enhanced for loop

```
for (String x : lstOfStrings) // Lists, Sets, Arrays are all Iterable!
```

is shorthand for:

```
for (Iterator<String> iter = lstOfStrings.iterator(); iter.hasNext();) {
    String x = iter.next();
}
```


Check for Understanding

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

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

3. What's one difference between `Iterator` and `Iterable`?

Check for Understanding

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

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

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

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

3. What's one difference between `Iterator` and `Iterable`?

`Iterators` are the actual object we can iterate over, i.e., think a Python generator over a list.
`Iterables` are object that can produce an iterator, i.e., an array is iterable; an iterator over the array could go through the element at every index of the array).

== vs. .equals()

- == compares if two variables point to the same object in memory.
  - null is compared with ==
- For reference types: .equals() (ex. myDog.equals(yourDog))
  - Each class can provide own implementation by overriding
  - Defaults to Object's .equals() (which is the same as ==)
  - Example: We make the Dog .equals() method return true if both Dogs have the same name
    - Dog fido = new Dog("Fido"); Dog otherFido = new Dog("Fido");
    - fido == otherFido -> false, but fido.equals(otherFido) -> true

Worksheet

1 Take Us to Your "Yrnqre"

Given the `AlienAlphabet` class, fill in `AlienComparator` class so that it compares strings lexicographically, based on the order passed into the `AlienAlphabet` constructor. For simplicity, you may assume all words passed into `AlienComparator` have letters present in `order`.

For example, if the alien alphabet has the order `"dba..."`, which means that `'d'` is the first letter, `'b'` is the second letter, and so on. `AlienAlphabet.AlienComparator.compare("dab", "bad")` should return a value less than `0`, since "`dab`" comes before "`bad`".

If one word is an exact prefix of another, the longer word comes later. For example, `"bad"` comes before `"badly"`.

```
public class AlienAlphabet {
    private String order;

    public AlienAlphabet(String o) {
        order = o;
    }
}
```


1 Take Us to Your "Yrnqre"

```
public class AlienComparator implements Comparator<____________> {
    public int compare(String word1, String word2) {
        int minLength = Math.min(__________________, __________________);
        for (_____________________________________________________________) {
            int char1Rank = ____________________________________________;
            int char2Rank = ____________________________________________;
            if (_______________________________________________) {
                return -1;
            } else if (_______________________________________________) {
                return 1;
            }
        }
        return _______________________________ -
_______________________________;
    }
}
```


1 Take Us to Your "Yrnqre"

```
public class AlienComparator implements Comparator<String> {
    public int compare(String word1, String word2) {
        int minLength = Math.min(__________________, __________________);
        for (_____________________________________________________________) {
            int char1Rank = ____________________________________________;
            int char2Rank = ____________________________________________;
            if (_______________________________________________) {
                return -1;
            } else if (_____________________________________________) {
                return 1;
            }
        }
        return _______________________________ -
_______________________________;
    }
}
```

1 Take Us to Your "Yrnqre"

```
public class AlienComparator implements Comparator<String> {
    public int compare(String word1, String word2) {
        int minLength = Math.min(word1.length(), word2.length());
        for (_____________________________________________________) {
            int char1Rank = ____________________________________________;
            int char2Rank = ____________________________________________;
            if (_______________________________________________) {
                return -1;
            } else if (_____________________________________________) {
                return 1;
            }
        }
        return ________________________________ -
_________________________________;
    }
}
```


1 Take Us to Your "Yrnqre"

```
public class AlienComparator implements Comparator<String> {
    public int compare(String word1, String word2) {
        int minLength = Math.min(word1.length(), word2.length());
        for (int i = 0; i < minLength; i++) {
            int char1Rank = ____________________________________________;
            int char2Rank = ____________________________________________;
            if (_______________________________________________) {
                return -1;
            } else if (_____________________________________________) {
                return 1;
            }
        }
        return _______________________________ -
_______________________________;
    }
}
```


1 Take Us to Your "Yrnqre"

```
public class AlienComparator implements Comparator<String> {
    public int compare(String word1, String word2) {
        int minLength = Math.min(word1.length(), word2.length());
        for (int i = 0; i < minLength; i++) {
            int char1Rank = order.indexOf(word1.charAt(i));
            int char2Rank = order.indexOf(word2.charAt(i));
            if (_____________________________________________) {
                return -1;
            } else if (_____________________________________________) {
                return 1;
            }
        }
        return _______________________________ -
_______________________________;
    }
}
```


1 Take Us to Your "Yrnqre"

```
public class AlienComparator implements Comparator<String> {
    public int compare(String word1, String word2) {
        int minLength = Math.min(word1.length(), word2.length());
        for (int i = 0; i < minLength; i++) {
            int char1Rank = order.indexOf(word1.charAt(i));
            int char2Rank = order.indexOf(word2.charAt(i));
            if (char1Rank < char2Rank) {
                return -1;
            } else if (_____________________________________________) {
                return 1;
            }
        }
        return ________________________________ -
_________________________________;
    }
}
```


1 Take Us to Your "Yrnqre"

```
public class AlienComparator implements Comparator<String> {
    public int compare(String word1, String word2) {
        int minLength = Math.min(word1.length(), word2.length());
        for (int i = 0; i < minLength; i++) {
            int char1Rank = order.indexOf(word1.charAt(i));
            int char2Rank = order.indexOf(word2.charAt(i));
            if (char1Rank < char2Rank) {
                return -1;
            } else if (char1Rank > char2Rank) {
                return 1;
            }
        }
        return ________________________________ -
_________________________________;
    }
}
```


1 Take Us to Your "Yrnqre"

```
public class AlienComparator implements Comparator<String> {
    public int compare(String word1, String word2) {
        int minLength = Math.min(word1.length(), word2.length());
        for (int i = 0; i < minLength; i++) {
            int char1Rank = order.indexOf(word1.charAt(i));
            int char2Rank = order.indexOf(word2.charAt(i));
            if (char1Rank < char2Rank) {
                return -1;
            } else if (char1Rank > char2Rank) {
                return 1;
            }
        }
        return word1.length() - word2.length();
    }
}
```


2 Iterator of Iterators

```
private class IteratorOfIterators  _____________ {
    public IteratorOfIterators(List<Iterator<Integer>> a) {

    }

    public boolean hasNext() {



    }

    public Integer next() {



    }

}
```


2 Iterator of Iterators - Solution 1

```
private class IteratorOfIterators implements Iterator<Integer> {
    LinkedList<Iterator<Integer>> iterators;

    public IteratorOfIterators(List<Iterator<Integer>> a) {
        iterators = new LinkedList<>();
        for (Iterator<Integer> iterator : a) {



        }
    }

    ...
}
```


2 Iterator of Iterators - Solution 1

```
private class IteratorOfIterators implements Iterator<Integer> {
    LinkedList<Iterator<Integer>> iterators;

    public IteratorOfIterators(List<Iterator<Integer>> a) {
        iterators = new LinkedList<>();
        for (Iterator<Integer> iterator : a) {
            if (iterator.hasNext()) {
                iterators.add(iterator);
            }
        }
    }

    ...
}
```


2 Iterator of Iterators - Solution 1

```
private class IteratorOfIterators implements Iterator<Integer> {
    LinkedList<Iterator<Integer>> iterators;
    public IteratorOfIterators(List<Iterator<Integer>> a) { ... }

    public boolean hasNext() {
        return !iterators.isEmpty();
    }

    public Integer next() {

    }
}
```


2 Iterator of Iterators - Solution 1

```
private class IteratorOfIterators implements Iterator<Integer> {
    LinkedList<Iterator<Integer>> iterators;
    public IteratorOfIterators(List<Iterator<Integer>> a) { ... }

    public boolean hasNext() {
        return !iterators.isEmpty();
    }

    public Integer next() {
        Iterator<Integer> nextIter = iterators.removeFirst();









    }
}
```


2 Iterator of Iterators - Solution 1

```
private class IteratorOfIterators implements Iterator<Integer> {
    LinkedList<Iterator<Integer>> iterators;
    public IteratorOfIterators(List<Iterator<Integer>> a) { ... }

    public boolean hasNext() {
        return !iterators.isEmpty();
    }

    public Integer next() {
        Iterator<Integer> nextIter = iterators.removeFirst();
        Integer nextItem = nextIter.next();




    }
}
```


2 Iterator of Iterators - Solution 1

```
private class IteratorOfIterators implements Iterator<Integer> {
    LinkedList<Iterator<Integer>> iterators;
    public IteratorOfIterators(List<Iterator<Integer>> a) { ... }

    public boolean hasNext() {
        return !iterators.isEmpty();
    }

    public Integer next() {
        Iterator<Integer> nextIter = iterators.removeFirst();
        Integer nextItem = nextIter.next();
        if (nextIter.hasNext()) {
            iterators.addLast(nextIter);
        }

    }
}
```


2 Iterator of Iterators - Solution 1

```
private class IteratorOfIterators implements Iterator<Integer> {
    LinkedList<Iterator<Integer>> iterators;
    public IteratorOfIterators(List<Iterator<Integer>> a) { ... }

    public boolean hasNext() {
        return !iterators.isEmpty();
    }

    public Integer next() {
        Iterator<Integer> nextIter = iterators.removeFirst();
        Integer nextItem = nextIter.next();
        if (nextIter.hasNext()) {
            iterators.addLast(nextIter);
        }
        return nextItem;
    }
}
```


2 Iterator of Iterators - Alternate Solution

```
private class IteratorOfIterators implements Iterator<Integer> {
    LinkedList<Integer> l;

    public IteratorOfIterators(List<Iterator<Integer>> a) {
        l = new LinkedList<>();
        while (!a.isEmpty()) {









        }
    }

    ...
}
```


2 Iterator of Iterators - Alternate Solution

```
private class IteratorOfIterators implements Iterator<Integer> {
    LinkedList<Integer> l;

    public IteratorOfIterators(List<Iterator<Integer>> a) {
        l = new LinkedList<>();
        while (!a.isEmpty()) {
            Iterator<Integer> curr = a.remove(0);




        }
    }

    ...
}
```


2 Iterator of Iterators - Alternate Solution

```
private class IteratorOfIterators implements Iterator<Integer> {
    LinkedList<Integer> l;

    public IteratorOfIterators(List<Iterator<Integer>> a) {
        l = new LinkedList<>();
        while (!a.isEmpty()) {
            Iterator<Integer> curr = a.remove(0);
            if (curr.hasNext()) {



            }
        }
    }

    ...
}
```


2 Iterator of Iterators - Alternate Solution

```
private class IteratorOfIterators implements Iterator<Integer> {
    LinkedList<Integer> l;

    public IteratorOfIterators(List<Iterator<Integer>> a) {
        l = new LinkedList<>();
        while (!a.isEmpty()) {
            Iterator<Integer> curr = a.remove(0);
            if (curr.hasNext()) {
                l.add(curr.next());

            }
        }
    }

    ...
}
```


2 Iterator of Iterators - Alternate Solution

```
private class IteratorOfIterators implements Iterator<Integer> {
    LinkedList<Integer> l;

    public IteratorOfIterators(List<Iterator<Integer>> a) {
        l = new LinkedList<>();
        while (!a.isEmpty()) {
            Iterator<Integer> curr = a.remove(0);
            if (curr.hasNext()) {
                l.add(curr.next());
                a.add(curr);
            }
        }
    }

    ...
}
```


2 Iterator of Iterators - Alternate Solution

```
private class IteratorOfIterators implements Iterator<Integer> {
    LinkedList<Integer> l;
    public IteratorOfIterators(List<Iterator<Integer>> a) { ... }

    public boolean hasNext() {
        return !l.isEmpty();
    }

    public Integer next() {

    }
}
```


2 Iterator of Iterators - Alternate Solution

```
private class IteratorOfIterators implements Iterator<Integer> {
    LinkedList<Integer> l;
    public IteratorOfIterators(List<Iterator<Integer>> a) { ... }

    public boolean hasNext() {
        return !l.isEmpty();
    }

    public Integer next() {
        if (!hasNext()) {
            throw new NoSuchElementException();
        }

    }
}
```


2 Iterator of Iterators - Alternate Solution

```
private class IteratorOfIterators implements Iterator<Integer> {
    LinkedList<Integer> l;
    public IteratorOfIterators(List<Iterator<Integer>> a) { ... }

    public boolean hasNext() {
        return !l.isEmpty();
    }

    public Integer next() {
        if (!hasNext()) {
            throw new NoSuchElementException();
        }
        return l.removeFirst();
    }
}
```