Comparators and Iterators

Exam-Level 04

Announcements

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
	9/23 Weekly Survey Due				9/27 Midterm 1 (7-9 PM)	
					10/04 Lab 5 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 `Dog`s 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"

Fill in `AlienComparator` class so that it compares strings lexicographically, based on the order passed into the `AlienComparator` 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.` Then, using `compare("dab", "bad")` with this comparator 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"`.

```java
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"

```java
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 _____________ {
    private List<Iterator<Integer>> iterators;
    private int curr;

    public IteratorOfIterators(List<Iterator<Integer>> a) {
        iterators = new LinkedList<>();
        for (________________________________) {
            if (________________________________) {
                ________________________________;
            }
        }
        curr = 0;
    }

    public boolean hasNext() {
        return ___________________________________________;
    }
```

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

    Iterator<Integer> currIterator =
________________;

    int result = ________________;

    if (________________) {
        __________________;
    } else {
        curr = __________________;
    }
    return result;
}
```

2 Iterator of Iterators

```java
private class IteratorOfIterators implements Iterator<Integer>{
    private List<Iterator<Integer>> iterators;
    private int curr;

    public IteratorOfIterators(List<Iterator<Integer>> a) {
        iterators = new LinkedList<>();
        for (________________________________) {
            if (________________________________) {
                ________________________________;
            }
        }
        curr = 0;
    }

    public boolean hasNext() {
        return ___________________________________________;
    }
```

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

    Iterator<Integer> currIterator =
________________;

    int result = ________________;

    if (________________) {
        __________________;
    } else {
        curr = __________________;
    }
    return result;
}
```

2 Iterator of Iterators

```
private class IteratorOfIterators implements Iterator<Integer>{
    private List<Iterator<Integer>> iterators;
    private int curr;

    public IteratorOfIterators(List<Iterator<Integer>> a) {
        iterators = new LinkedList<>();
        for (Iterator<Integer> iterator : a) {
            if (________________________________) {
                ________________________________;
            }
        }
        curr = 0;
    }

    public boolean hasNext() {
        return ___________________________________________;
    }
```

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

    Iterator<Integer> currIterator =
________________;

    int result = ________________;

    if (________________) {
        __________________;
    } else {
        curr = __________________;
    }
    return result;
}
```

2 Iterator of Iterators

```java
private class IteratorOfIterators implements Iterator<Integer>{
    private List<Iterator<Integer>> iterators;
    private int curr;

    public IteratorOfIterators(List<Iterator<Integer>> a) {
        iterators = new LinkedList<>();
        for (Iterator<Integer> iterator : a) {
            if (iterator.hasNext()) {
                ________________________________;
            }
        }
        curr = 0;
    }

    public boolean hasNext() {
        return ___________________________________________;
    }
```

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

    Iterator<Integer> currIterator =
________________;

    int result = ________________;

    if (________________) {
        __________________;
    } else {
        curr = __________________;
    }
    return result;
}
```

2 Iterator of Iterators

```
private class IteratorOfIterators implements Iterator<Integer>{
    private List<Iterator<Integer>> iterators;
    private int curr;

    public IteratorOfIterators(List<Iterator<Integer>> a) {
        iterators = new LinkedList<>();
        for (Iterator<Integer> iterator : a) {
            if (iterator.hasNext()) {
                iterators.add(iterator);
            }
        }
        curr = 0;
    }

    public boolean hasNext() {
        return __________________________________________;
    }
```

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

    Iterator<Integer> currIterator =
________________;

    int result = ________________;

    if (________________) {
        __________________;
    } else {
        curr = __________________;
    }
    return result;
}
```

2 Iterator of Iterators

```
private class IteratorOfIterators implements Iterator<Integer>{
    private List<Iterator<Integer>> iterators;
    private int curr;

    public IteratorOfIterators(List<Iterator<Integer>> a) {
        iterators = new LinkedList<>();
        for (Iterator<Integer> iterator : a) {
            if (iterator.hasNext()) {
                iterators.add(iterator);
            }
        }
        curr = 0;
    }

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

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

    Iterator<Integer> currIterator =
________________;

    int result = ________________;

    if (________________) {
        __________________;
    } else {
        curr = __________________;
    }
    return result;
}
```

2 Iterator of Iterators

```
private class IteratorOfIterators implements Iterator<Integer>{
    private List<Iterator<Integer>> iterators;
    private int curr;

    public IteratorOfIterators(List<Iterator<Integer>> a) {
        iterators = new LinkedList<>();
        for (Iterator<Integer> iterator : a) {
            if (iterator.hasNext()) {
                iterators.add(iterator);
            }
        }
        curr = 0;
    }

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

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

    Iterator<Integer> currIterator =
iterators.get(curr);

    int result = ________________;

    if (________________) {
        __________________;
    } else {
        curr = __________________;
    }
    return result;
}
```

2 Iterator of Iterators

```
private class IteratorOfIterators implements Iterator<Integer>{
    private List<Iterator<Integer>> iterators;
    private int curr;

    public IteratorOfIterators(List<Iterator<Integer>> a) {
        iterators = new LinkedList<>();
        for (Iterator<Integer> iterator : a) {
            if (iterator.hasNext()) {
                iterators.add(iterator);
            }
        }
        curr = 0;
    }

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

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

    Iterator<Integer> currIterator =
iterators.get(curr);

    int result =
iterators.get(curr).next();

    if (________________) {
        _________________;
    } else {
        curr = _________________;
    }
    return result;
}
```

2 Iterator of Iterators

```java
private class IteratorOfIterators implements Iterator<Integer>{
    private List<Iterator<Integer>> iterators;
    private int curr;

    public IteratorOfIterators(List<Iterator<Integer>> a) {
        iterators = new LinkedList<>();
        for (Iterator<Integer> iterator : a) {
            if (iterator.hasNext()) {
                iterators.add(iterator);
            }
        }
        curr = 0;
    }

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

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

    Iterator<Integer> currIterator =
iterators.get(curr);

    int result =
iterators.get(curr).next();

    if (!currIterator.hasNext()) {
        __________________;
    } else {
        curr = __________________;
    }
    return result;
}
```

2 Iterator of Iterators

```
private class IteratorOfIterators implements Iterator<Integer>{
    private List<Iterator<Integer>> iterators;
    private int curr;

    public IteratorOfIterators(List<Iterator<Integer>> a) {
        iterators = new LinkedList<>();
        for (Iterator<Integer> iterator : a) {
            if (iterator.hasNext()) {
                iterators.add(iterator);
            }
        }
        curr = 0;
    }

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

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

    Iterator<Integer> currIterator =
iterators.get(curr);

    int result =
iterators.get(curr).next();

    if (!currIterator.hasNext()) {
        iterators.remove(curr);
    } else {
        curr = __________________;
    }
    return result;
}
```

2 Iterator of Iterators

```
private class IteratorOfIterators implements Iterator<Integer>{
    private List<Iterator<Integer>> iterators;
    private int curr;

    public IteratorOfIterators(List<Iterator<Integer>> a) {
        iterators = new LinkedList<>();
        for (Iterator<Integer> iterator : a) {
            if (iterator.hasNext()) {
                iterators.add(iterator);
            }
        }
        curr = 0;
    }

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

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

    Iterator<Integer> currIterator =
iterators.get(curr);

    int result =
iterators.get(curr).next();

    if (!currIterator.hasNext()) {
        iterators.remove(curr);
    } else {
        curr = (curr + 1) %
iterators.size();
    }
    return result;
}
```