

1 Equals (Lecture 10)

from Fall 2025 Midterm 1

Fill in the `equals` method below. Two `GoatMeasurement` objects are considered equal if and only if they have the same `name`, and their `sizes` differ by at most 0.1. Assume `name` is never null.

Hint: You may find `Math.abs` useful.

Syntax hint: `if (o instanceof Dog uddaDog)`

```
public class GoatMeasurement {
    public String name;
    public double size;

    @Override
    public boolean equals(Object o) {

        if (_____) {

            return (_____)
                _____);

        }

        return _____;

    }
}
```

2 DefaultQueue (Lecture 9)

When we write **default** methods, we have no choice but to use our existing abstractions. This is because we don't have any knowledge of or access to the underlying base reality. Suppose we have a **MyQueue** interface that we want to implement. We want to include three default methods in the interface: **clear**, **remove** and **appendAll**. Fill in these methods in the code below.

```
public interface MyQueue<E> {
    void enqueue(E element); // adds an element to the end of the queue
    E dequeue();             // removes and returns the front element of the queue
    boolean isEmpty();       // returns true if the queue is empty
    int size();              // returns the number of elements in the queue
    // removes all items from the queue
    default void clear() {

    }

    // removes all items equal to item from the queue
    // the remaining items should be in the same order as they were before
    // use .equals to compare items rather than ==
    default void remove(E item) {

    }

    // appends all items from the other queue into this queue
    // this method should be non-destructive on the otherQueue!
    default void appendAll(MyQueue<E> otherQueue) {

    }
}
```

3 Every Other Character (Lecture 10)

Implement an `EveryOtherCharacterIterator` that iterates over every other character of a `String`, starting with the character at index 0.

```
import java.util.Iterator;
import java.util.NoSuchElementException;

public class EveryOtherCharacterIterator implements Iterator<Character> {
    private _____;
    private _____;

    public EveryOtherCharacterIterator(String s) {
        _____;
        _____;
    }

    @Override
    public boolean hasNext() {
        return _____;
    }

    @Override
    public Character next() {
        if (!hasNext()) { throw new NoSuchElementException(); }
        char result = _____;
        _____;
        return result;
    }
}
```

As an example, the code below creates an `EveryOtherCharacterIterator` and uses it to print every other character of "ghooautt":

```
EveryOtherCharacterIterator eoci = new EveryOtherCharacterIterator("ghooautt");
while (eoci.hasNext()) {
    IO.print(eoci.next()); // prints goat
}
```

(a) Why does our class implement `Iterator<Character>` and not `Iterator<String>`?

(b) Why can't we write the following? What would we need to change about the `EveryOtherCharacterIterator` to make this work?

```
for (char c : eoci) {
    IO.print(c);
}
```

4 Default (Lecture 9)

Suppose we have a **MyStack** interface that we want to implement. We want to add two default methods to the interface: **insertAtBottom** and **flip**. Fill in these methods in the code below.

```
public interface MyStack<E> {
    void push(E element); // adds an element to the top of the stack
    E pop();               // removes and returns the top element of the stack
    boolean isEmpty();     // returns true if the stack is empty
    int size();            // returns the number of elements in the stack

    // inserts the item at the bottom of the stack using push, pop, isEmpty, and size
    private void insertAtBottom(E item) {

    }

    // flips the stack upside down (hint: use insertAtBottom)
    default void flip() {

    }
}
```