Filtered List

Here is a video walkthrough of the solutions.

We want to make a FilteredList class that selects only certain elements of a List during iteration. To do so, we're going to use the Predicate interface defined below. Note that it has a method, test that takes in an argument and returns True if we want to keep this argument or False otherwise.

```
public interface Predicate<T> {
        boolean test(T x);
}
```

For example, if L is any kind of object that implements List<String> (that is, the standard java.util.List), then writing

```
FilteredList<String> FL = new FilteredList<>(L, filter);
```

gives an **iterable** containing all items, x, in L for which filter.test(x) is True. Here, filter is of type Predicate. Fill in the FilteredList class below.

```
import java.util.*;
public class FilteredList<T> ________________________________________ {


    public FilteredList (List<T> L, Predicate<T> filter) {



    }
    @Override
    public Iterator<T> iterator() {

    }













}
```

Solution:

```
import java.util.*;
```

```java

class FilteredList<T> implements Iterable<T> {
    List<T> list;
    Predicate<T> pred;

    public FilteredList(List<T> L, Predicate<T> filter) {
        this.list = L;
        this.pred = filter;
    }

    public Iterator<T> iterator() {
        return new FilteredListIterator();
    }

    private class FilteredListIterator implements Iterator<T> {
        int index;

        public FilteredListIterator() {
            index = 0;
            moveIndex();
        }

        @Override
        public boolean hasNext() {
            return index < list.size();
        }

        @Override
        public T next() {
            if (!hasNext()) {
                throw new NoSuchElementException();
            }
            T answer = list.get(index);
            index += 1;
            moveIndex();
            return answer;
        }
        private void moveIndex() {
            while (hasNext() && !pred.test(list.get(index))) {
                index += 1;
            }
        }
    }
}
```

Alternate Solution: Although this solution provides the right functionality, it is not as efficient as the first one. Imagine you only want the first couple items from the iterable. Is it worth processing the entire list in the constructor? It is not ideal in the case that our list is millions of elements long. The first solution is different in that we "lazily" evaluate the list, only progressing our index on every call to `next` and `hasNext`. However, this solution may be easier to digest.

```java
import java.util.*;

class FilteredList<T> implements Iterable<T> {
    List<T> list;
    Predicate<T> pred;

    public FilteredList(List<T> L, Predicate<T> filter) {
        this.list = L;
        this.pred = filter;
    }

    public Iterator<T> iterator() {
        return new FilteredListIterator();
    }

    private class FilteredListIterator implements Iterator<T> {
        LinkedList<T> items;

        public FilteredListIterator() {
            items = new LinkedList<>();
            for (T item: list) {
                if (pred.test(item)) {
                    items.add(item);
                }
            }
        }

        @Override
        public boolean hasNext() {
            return !items.isEmpty();
        }

        @Override
        public T next() {
            if (!hasNext()) {
                throw new NoSuchElementException();
            }
            return items.removeFirst();
        }
    }
```