CS 61B Fall 2020

# Iterators and Iterables

Exam Prep Discussion 5: September 21, 2020

## 1 Filtered List

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<String> (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.Iterator;
import java.util.Iterable;
import java.util.NoSuchElementException;
public class FilteredList<T> ________________________________________ {


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

        }

        @Override
        public Iterator<T> iterator() {

        }









}
```

**Solution:**

```
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(list, pred);
    }

    private class FilteredListIterator<T> implements Iterator<T> {
        List<T> list;
        Predicate<T> pred;
        int index;

        public FilteredListIterator(List<T> l, Predicate<T> f) {
            list = l;
            pred = f;
            index = 0;
        }

        @Override
        public boolean hasNext() {
            while (index < list.size() && !pred.test(list.get(index))) {
                index += 1;
            }
            return index < list.size();
        }

        @Override
        public T next() {
            if (!hasNext()) {
                throw new NoSuchElementException();
            }
            index += 1;
            return list.get(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 has next.

```
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(list, pred);
    }

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

        public FilteredListIterator(List<T> l, Predicate<T> f) {
            list = new LinkedList<>();
            for (T item: l) {
                if (f.test(item)) {
                    list.add(item);
                }
            }
        }

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

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

# 2 Iterator of Iterators

Implement an `IteratorOfIterators` which will accept as an argument a `List` of `Iterator` objects containing `Integers`. The first call to `next()` should return the first item from the first iterator in the list. The second call to `next()` should return the first item from the second iterator in the list. If the list contained `n` iterators, the `n+1`th time that we call `next()`, we would return the second item of the first iterator in the list.

For example, if we had 3 `Iterators` A, B, and C such that A contained the values `[1, 2, 3]`, B contained the values `[4, 5, 6]`, and C contained the values `[7, 8, 9]`, calls to `next()` for our `IteratorOfIterators` would return `[1, 4, 7, 2, 5, 8, 3, 6, 9]`

Feel free to modify the input `a` as needed.

```
import java.util.*;
public class IteratorOfIterators  _______________________________ {


    public IteratorOfIterators(List<Iterator<Integer>> a) {









    }

    @Override
    public boolean hasNext() {




    }



    @Override
    public Integer next() {




    }
}
```

**Solution:**

```
import java.util.*;

public 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);
            }
        }
    }

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

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

# 3 Every $k$th Element (Fall 2014 MT1 Q5)

Fill in the `next()` method in the following class. Do not modify anything outside of `next`.

```
import java.util.Iterator;
import java.util.NoSuchElementException;
/** Iterates over every Kth element of the IntList given to the constructor.
*   For example, if L is an IntList containing elements
*   [0, 1, 2, 3, 4, 5, 6, 7] with K = 2, then
*       for (Iterator<Integer> p = new KthIntList(L, 2); p.hasNext(); ) {
*           System.out.println(p.next());
*       }
*   would print get 0, 2, 4, 6. */
public class KthIntList implements Iterator <Integer> {
    public int k;
    private IntList curList;
    private boolean hasNext;

    public KthIntList(IntList I, int k) {
        this.k = k;
        this.curList = I;
        this.hasNext = true;
    }

    /** Returns true iff there is a next Kth element. Do not modify. */
    public boolean hasNext() {
        return this.hasNext;
    }

    /** Returns the next Kth element of the IntList given in the constructor.
     *  Returns the 0th element first. Throws a NoSuchElementException if
     *  there are no Integers available to return. */
    public Integer next() {
        ____________________________________________________________
        ____________________________________________________________
        ____________________________________________________________
        ____________________________________________________________
        ____________________________________________________________
        ____________________________________________________________
        ____________________________________________________________
        ____________________________________________________________
        ____________________________________________________________
        ____________________________________________________________
    }
}
```

**Solution:**

```
public Integer next() {
    if (!hasNext() || curList == null) {
        throw new NoSuchElementException();
    }

    Integer item = curList.item;
    for (int i = 0; i < k; i++) {
        curList = curList.next;
        if (curList == null) {
            hasNext = false;
            return item;
        }
    }
    return item;
}
```