CS 367 - Introduction to Data Structures
Thursday, January 28, 2016

We assume that you are proficient at object-oriented programming in Java.

Websites

- **Lec 3 & 1:** http://pages.cs.wisc.edu/~cs367-1/
- **Lec 2:** http://pages.cs.wisc.edu/~cs367-2/

See syllabus page for online readings and lecture outlines (no textbook).

Waitlisted? Please sign up on the yellow notepad. Continue attending.

Homework 1 REVISED due 10 pm Tuesday, February 2nd
Program 1 due 10 pm Sunday, February 14th, GET STARTED NOW!
Assignment questions? Post on Piazza or consult with a TA during scheduled hours.

Last Time

Lists

- using lists via the ListADT
- implementing the ListADT using an array (SimpleArrayList)

Java API Lists
Iterators Concept

Today

Iterators

- iterators and the Java API
- using iterators
- options for implementing iterators
- making a class iterable

Next Time

Read: *Exceptions*
Handin Info
Exceptions Review

- throwing
- handling
- execution
- practice with exception handling
- throws and checked vs. unchecked
- defining

Interfaces - Iterators in Java API

`Iterable<T>` **interface in** **`java.lang`**

specifies the operation to get an iterator to step through a collection:

- Iterator<T> iterator()

`Iterator<E>` **interface in** **`java.util`**

specifies the operations that iterators can do:

- boolean hasNext()
- E next()
- void remove() //"optional"

Use - Iterators

Suppose `words` is a `SimpleArrayList<String>` that implements the `Iterable` Interface.
→ **Write a code fragment** that gets an iterator, named `itr`, from `words`.

Suppose `words` is a `SimpleArrayList<String>` and `itr` is an iterator for `words`.
→ **Write a code fragment** that uses `itr` to print each item in `words`.

→ **Next write a code fragment** that uses `itr` to print the length of each item in `words`.

Use - Iterators

Assume `SimpleArrayList<String>` implements the `Iterable` Interface.
→ **Challenge: Complete the method** using iterators to determine `list` contains duplicates.

```
public boolean hasDups(SimpleArrayList<String> list) {
```

Implementation - Options for Iterator Classes

Indirect Access

Direct Access

Implementation - Indirect Access `SimpleArrayListIterator` Class

```
import java.util.*;
public class SimpleArrayListIterator<E> implements Iterator<E> {




    public SimpleArrayListIterator(                              ) {




    }

    public boolean hasNext() {


    }

    public E next() {









    }

    public void remove() {


    }
}
```

Implementation - Direct Access `ArrayBagIterator` Class

```
import java.util.*;
public class ArrayBagIterator<E> implements Iterator<E> {









    public ArrayBagIterator(                          ) {




    }

    public boolean hasNext() {


    }

    public E next() {









    }

    public void remove() {
       throw new UnsupportedOperationException();
    }
}
```

→ Could we code this as an indirect access iterator instead?

Making Array Bags `Iterable`

Approach in Readings - Modify the `BagADT` Interface

```
import java.util.*;
public interface BagADT<E> {
    void add(E item);
    E remove() throws NoSuchElementException;
    boolean isEmpty();


}
```

Also Modify the `ArrayBag` Class

```
import java.util.*;
public class ArrayBag<E> implements BagADT<E> {

    // *** Data members (fields) ***
    private E[] items;
    private int numItems;
    private static final int INIT_SIZE = 100;

    //*** required BagADT methods ***
    public void add(E item) { ... }
    public E remove() throws NoSuchElementException { ... }
    public boolean isEmpty() { ... }









}
```

How would we do this using Java's approach?