CSE 12 – Basic Data Structures
Prof. Christine Alvarado
[Slides borrowed/adapted from slides by Cynthia Lee]
Announcements

1. HW2 posted! Please start now!
   1. Overview video included
Today’s Topics

1. Iterators
2. Implementing the List interface: Linked Lists
A list is a collection. The Java List interface extends the Java Collection interface.

interface List<E> extends Collection<E>
Which is legal?

A Derived is a Base (A Base is not a Derived)

public class Base {
    protected int x;
}

public class Derived extends Base {
    protected int y;
}

A: Base b=new Base();
    Derived d=b;
B: Derived d=new Derived();
    Base b=d;
C: Base b=new Derived();
    Derived d=new Base();
E: Other/none/more
Lists and Collections

**Collection**

Collection declares an add method:
Ensures that this collection contains the specified element.

**List**

List *inherits* the add method... and changes its contract:
Appends the specified element to the end of this list.
Lists have order

**Note:** \( \text{ls} \) is some object of a type that implements List<\text{MyObjs}>.

```java
int i=0;
while (i<ls.size()) {
    MyObjs x = ls.get(i);
    //do something with x
    i++;
}
```
Iterators

Next!
The Iterator Software Design Pattern

- A common situation: A client needs to inspect the data elements in a collection, without wanting to know details of how the collection structures its data internally

- Solution:
  - Define an interface that specifies how an iterator will behave
  - Design the collection to be able to supply an object that implements that iterator interface
  - A client then can ask the collection for an iterator object, and use that iterator to inspect the collection’s elements, without having to know how the collection is implemented
**Iterable<E> Interface**

- The `Collection<E>` interface extends the `Iterable<E>` interface, which is defined as follows:

  ```java
  public interface Iterable<E> {
    public Iterator<E> iterator();
  }
  ```

- So any class that implements `Collection<E>` must define an instance method `iterator()` that returns an `Iterator<E>` object for that instance.

- And `Iterator<E>` is also an interface in the JCF…
Iterator< E> Interface

• The Iterator< E> interface is defined as follows:

```java
public interface Iterator< E> {
    public E next();
    public boolean hasNext();
    public void remove();
}
```

• So, any object that is-a Iterator< E> will have those operations as part of its API.

• But what are these methods supposed to do? One example
Traversing using an Iterator

Note: `ls` is some object of a type `List<MyObjs>`.

```java
Iterator<MyObjs> i = ls.iterator();
while (i.hasNext()) {
    MyObjs x = i.next();
    // do something with x
}
```
WHY have iterators? What is the difference between these codes?

Note: ls is some object of a type that implements List<MyObjs>

**Iterator version**

```java
Iterator<MyObjs> i = ls.iterator();
while (i.hasNext()) {
    MyObjs x = i.next();
    //do something with x
}
```

**Increment version**

```java
int i=0;
while (i<ls.size()) {
    MyObjs x = ls.get(i);
    //do something with x
    i++;
}
```

A. No difference: they are totally equivalent
B. Difference: Increment will run faster
C. Difference: Iterator will run faster
D. No difference: Java just likes having iterators to make students’ lives harder
E. Other/none/more
WHY have iterators?

- **Linked List**
  - To go to $n^{th}$ element, you must start at the beginning and go through one at a time until you get to the $n^{th}$ one
  - A pain to start over vs just having the iterator keep your place like a bookmark

In your homework you will implement a ListIterator, which is basically just an Iterator.
Linked Lists
Java’s List<E> Interface (a few methods)

boolean **add**(E e) Appends the specified element to the end of this list (optional operation).

void **add**(int index, E element) Inserts the specified element at the specified position in this list (optional operation).

E **get**(int index) Returns the element at the specified position in this list.
Implementing a List with an array

class MyArrayList implements List<Integer>
{
    Integer[] theArray
    int size;

    public MyArrayList()
    {
        size = 0;
        theArray = new Integer[10];
    }

    public boolean add( Integer elem )
    {
        theArray[size] = elem;
        size++;
        return true;
    }
    ...

    We’re using only Integers here because arrays are ugly with generics
Implementing a List with an array

We're using only Integers here because arrays are ugly with generics

The List interface is sort of long...
public class MyArrayList extends AbstractList<Integer>
{
    Integer[] theArray;
    int size;

    public MyArrayList()
    {
        size = 0;
        theArray = new Integer[10];
    }

    public boolean add( Integer elem )
    {
        theArray[size] = elem;
        size++;  
        return true;
    }

    ...

    Java provides a shortcut so that we don’t have to implement ALL the List methods.
An AbstractList “is a” List
A MyArrayList “is a” AbstractList
A MyArrayList “is a” List
Implementing a List with an array

public class MyArrayList implements List<Integer>
{
    Integer[] theArray;
    int size;

    public MyArrayList()
    {
        size = 0;
        theArray = new Integer[10];
    }

    public boolean add( Integer elem )
    {
        theArray[size] = elem;
        size++;
        return true;
    }
    ...

    A MyArrayList “has a” Integer Array
Implementing a List with an array

```java
public class MyArrayList extends AbstractList<Integer> {
    Integer[] theArray;
    int size;

    public MyArrayList() {
        size = 0;
        theArray = new Integer[10];
    }

    public boolean add( Integer elem ) {
        theArray[size] = elem;
        size++;
        return true;
    }
    ...

What is the problem with the add method?
```
Single Linked List: Picture

MySingleLinkedList object

head

size 2

Node object

next

data null

Node object

next

data

Node object

next

data

null

Objects of type E:

Dummy (sentinel) head node
Single Linked List: Picture

MySingleLinkedList object

head
size 2

Node object
next
next

data null

data

Dummy (sentinel) head node

Objects of type E:

What type is next in the node class?
A. Node
B. E
C. LinkedList
D. String
Single Linked List: Abstracted Picture
class Node<E> {
    E data;
    Node next;

    public Node() {
        data = null;
        next = null;
    }

    public Node(E theData, Node newNodePred) {
        data = theData;
        next = newNodePred.next;
        newNodePred.next = this;
    }
}
Single Linked List Node: Code

class Node<E> {
    E data;
    Node next;

    public Node() {
        this.data = null;
        this.next = null;
    }

    public Node(E theData, Node newNodePred) {
        this.data = theData;
        this.next = newNodePred.next;
        newNodePred.next = this;
    }
}

public static void main() {
    Node<Integer> n0 = new Node<Integer>();
    Node<Integer> n1 = new Node(new Integer(1), n0);
}

HW2: Doubly linked lists

MyLinkedList object

head

size 1

tail

Node object

next

data null

prev null

Node object

next

data

prev

Node object

next null

data null

prev

Linked list object stores pointer to the tail

Objects of type E:

Dummy head and tail nodes
Next time

- More practice with linked lists (single and double)
- Iterator implementation