Interfaces and Linked Lists

4/6/26

Green slides taken (w/ minor modifications) from Cynthia Lee's CS 2 slides on http://www.peerinstruction4cs.org, licensed under Creative Commons Attribution-NonCommercial 4.0 International License.

Administrivia

- HW 2 (test cases for Book) due Wednesday (4/8)

Recall: Abstract data types (ADTs)

- Set of operations supported on a particular kind of data
- Does not include implementation details
- Our example: List (ordered collection of items)
  - add, remove, get(i), size, etc

Implementation of List: ArrayList

- Array-based implementation of the List ADT

```
ArrayList<Point> list = new ArrayList<Point>();

Point p = new Point(x, y);
list.add(p);

for(int i=0; i<list.size(); i++) {
        Point p = list.get(i);
        //do something with p
}
```

Interface: Java’s way to express an ADT

- Essentially a list of method signatures

```
public interface List<T> {
        public void add(T item);
        public int size();
        public T get(int index);
        …
}
```

Using an interface

```
public class ArrayList<T> implements List<T> {
    //must have methods for all promised signatures
}
```

Using an interface

```
public class ArrayList<T> implements List<T> {
	//must have methods for all promised signatures
}
```

- Allows variable to store **<u>any</u>** implementing type:

```
List<String> list = new ArrayList<String>();
```

Using an interface

```
public class ArrayList<T> implements List<T> {
        //must have methods for all promised signatures
}
```

- Allows variable to store **any** implementing type:

```
List<String> list = new ArrayList<String>();
```

- Can define operation on any implementing type:

```
public String mostCommon(List<String> list) {
        //find string with most occurrences for any List
```

We saw this in Lab 2

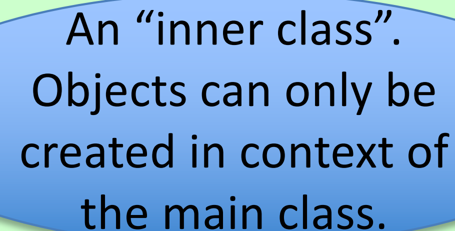
- PrintQueue interface defines some behavior
- WorkingPrintQueue (implements PrintQueue) does this correctly; all the methods work as advertised
- BadPrintQueue (implements PrintQueue) almost does this, but has a bug with some incorrect behavior
- You wrote test cases in terms of a PrintQueue p
  - When p referenced a WorkingPrintQueue, all the tests pass
  - When p referenced a BadPrintQueue, at least one failed

LinkedList

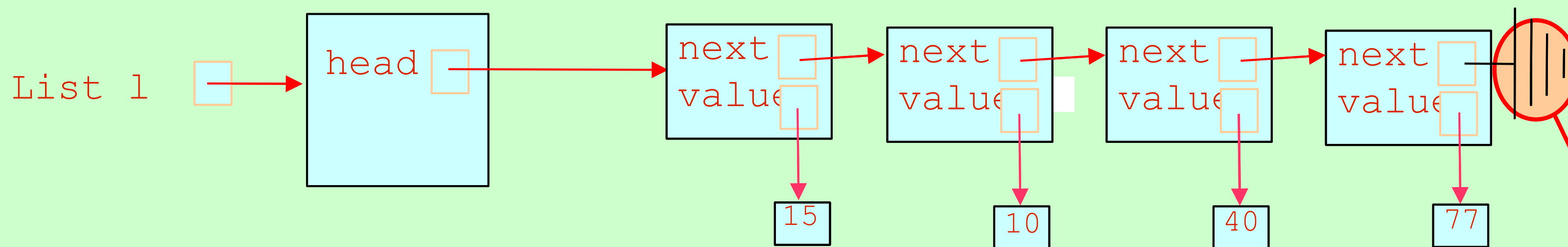
```
public class OurLinkedList<T> implements OurList<T> {
  private Node head;
  //optionally: could have size

  public class Node {
    public T value;
    public Node next;

    public Node(T value, Node next) {
      this.value = value;
      this.next = next;
    }
  }
...
```

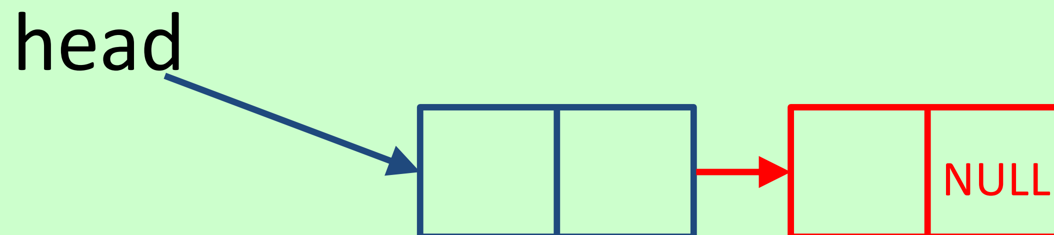


How our class looks in memory



-“head” points to first node

Null next pointer means “end of the list”



What line of code will correctly complete this method?

```
public void addFront(T newItem) {
   Node newNode = new Node(newItem, head);
      ________________________;
}
```

A) No line is needed. The code is correct as written.

B) `head = head.next;`

C) `head = newNode;`

D) `newNode.next = head;`

Question to think about/discuss: what is the purpose or effect of passing `head` as an argument here?

What line of code will correctly complete this method?

```
public void addFront(T newItem) {
    Node newNode = new Node(newItem, head);
        ________________________;
}
```

A) No line is needed. The code is correct as written.

B) `head = head.next;`

C) `head = newNode;`

D) `newNode.next = head;`

Question to think about/discuss: what is the purpose or effect of passing `head` as an argument here?

We implement size() by traversing the list. Which of the following tests if curr is at the end of the list? (i.e. what should go in the blank?)

```
public int size() {
   Node curr = head;
   int count = 0;
   while(___) {
    count++;
      ...
   }
   return count;
}
```

A. curr == head
B. curr != null
C. curr.next != head
D. curr.next != null
E. Not exactly one of the above

We implement size() by traversing the list. Which of the following tests if curr is at the end of the list? (i.e. what should go in the blank?)

```
public int size() {
  Node curr = head;
  int count = 0;
  while(___) {
   count++;
     ...
  }
  return count;
}
```

A. curr == head
B. curr != null
C. curr.next != head
D. curr.next != null
E. Not exactly one of the above

Which of the following advances curr to point to the next Node? (i.e. what should go in the blank?)

```
public int size() {
    Node curr = head;
    int count = 0;
    while(curr != null) {
        count++;
        ___
    }
    return count;
}
```

A. curr++;
B. curr.next;
C. curr = curr.next;
D. Node temp = curr.next;
   curr = temp;
E. Not exactly one of the above

Which of the following advances curr to point to the next Node? (i.e. what should go in the blank?)

```
public int size() {
    Node curr = head;
    int count = 0;
    while(curr != null) {
        count++;
        ___
    }
    return count;
}
```

A. curr++;
B. curr.next;
C. curr = curr.next;
D. Node temp = curr.next;
   curr = temp;
E. Not exactly one of the above (C & D)

Suppose you want to print the contents of a OurLinkedList for debugging. You want a variable curr to refer to each Node in turn. Which of the following does this correctly?

A.
```
Node curr = head;
while(curr != null)  {
    ...
    curr = curr.next;
}
```

B.
```
Node curr = head;
while(curr.next != null)  {
    ...
    curr = curr.next;
}
```

C.
```
Node curr = head.next;
while(curr != null)  {
    ...
    curr = curr.next;
}
```

D.
```
Node curr = head.next;
while(curr.next != null)  {
    ...
    curr = curr.next;
}
```

E. Not exactly one of the above

Suppose you want to print the contents of a OurLinkedList for debugging. You want a variable curr to refer to each Node in turn. Which of the following does this correctly?

A.
```
Node curr = head;
while(curr != null)  {
    ...
    curr = curr.next;
}
```

B.
```
Node curr = head;
while(curr.next != null)  {
    ...
    curr = curr.next;
}
```

C.
```
Node curr = head.next;
while(curr != null)  {
    ...
    curr = curr.next;
}
```

D.
```
Node curr = head.next;
while(curr.next != null)  {
    ...
    curr = curr.next;
}
```

E. Not exactly one of the above

```
public boolean contains(T s) {
    //returns whether the list contains value s
```

How many of the following work as the body for the clear method (empties list)?

I. `size = 0;`

II. `head = null;`

III. `head.next = null;`

IV.
```
Node curr = head;
while(curr != null) {
    curr.value = null;
    curr = curr.next;
}
```

A. 0
B. 1
C. 2
D. 3
E. 4

How many of the following work as the body for the clear method (empties list)?

I. size = 0;

II. head = null;

III. head.next = null;

IV.
```
Node curr = head;
while(curr != null) {
    curr.value = null;
    curr = curr.next;
}
```

A. 0

B. 1 (II.)

C. 2

D. 3

E. 4

What is wrong with this add method?

```
public void add(T toAdd) {      //add to end of the list
    Node curr = head;
    while(curr.next != null)
        curr = curr.next;
    Node newNode = new Node(toAdd, null);
    curr.next= newNode;
}
```

A. Syntax error
B. Doesn’t add the new value
C. Throws exception if list is empty
D. Removes existing value(s) from the list
E. Not exactly one of the above

What is wrong with this add method?

```
public void add(T toAdd) {      //add to end of the list
    Node curr = head;
    while(curr.next != null)
        curr = curr.next;
    Node newNode = new Node(toAdd, null);
    curr.next= newNode;
}
```

A. Syntax error
B. Doesn’t add the new value
C. Throws exception if list is empty
D. Removes existing value(s) from the list
E. Not exactly one of the above