

G51PRG: Introduction to Programming Second semester Lecture 13

Natasha Alechina
School of Computer Science & IT
nza@cs.nott.ac.uk

Previous lecture:networking

How to make your program to make network connections
(java.net package)

- Connecting via URLs
- Connecting via ports and sockets
- ChatServer program

This lecture

- Dynamic arrays and lists
- Implementing a dynamic array
- Implementing a very simple linked list in Java
- Inner classes

Next lecture

- Iterators
- Synchronised data structures (back to ChatServer)
- Serialisation (saving objects)

Lecture 13: linked list

3

Two kinds of sequential collections

- Array-like collections
- Linked collections
- In what follows, we assume that they all have the following methods:
 - **Object get(int i)** - return the ith element
 - **void add(Object o)** - add Object o
 - **boolean remove(Object o)** - remove the first occurrence of o in the collection; returns true if o was in the collection, and false otherwise.

Lecture 13: linked list

4

Array-like collections

- Array itself:
 - Advantages: easy to use, very fast; equally fast access to any index (constant time).
 - Disadvantages: fixed size
- Vector, ArrayList from java.util: generally known as dynamic arrays.
 - Advantages: almost as fast as array, can grow if more items need to be inserted
 - Disadvantages: resizing expensive; still a bit inflexible.

Dynamic arrays: idea

- Has an array inside to store Objects
- If array is not large enough, makes a larger one and copies the values to the new array.
- If an element is removed, the gap is closed by moving subsequent values one place to the left.

Example

- Dynamic array of initial capacity 5 is created:

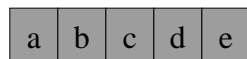


Lecture 13: linked list

7

Example

- Adding objects a,b,c,d,e:

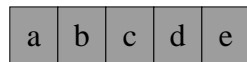


Lecture 13: linked list

8

Example

- Adding object f:



Lecture 13: linked list

9

Example

- Double the capacity:

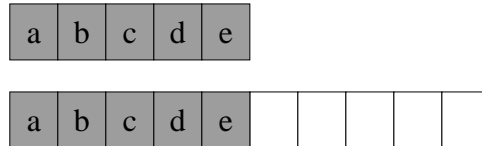


Lecture 13: linked list

10

Example

- Copy the contents to the new array:

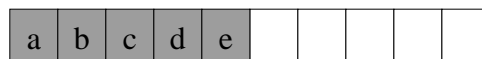


Lecture 13: linked list

11

Example

- Make new array the store:

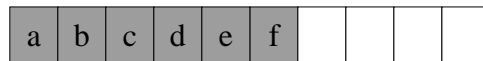


Lecture 13: linked list

12

Example

- Add f:

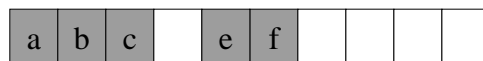


Lecture 13: linked list

13

Example

- Remove d:

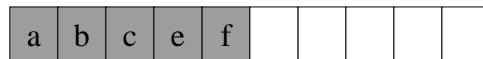


Lecture 13: linked list

14

Example

- Close the gap:



Lecture 13: linked list

15

Implementation of “dynamic array”

```
public class DynamicArray {  
    Object [] store;  
    int capacity;//how many items can store  
    int size; // how many items stored  
    public DynamicArray() {  
        store = new Object[10];  
        capacity = 10;  
        size = 0;  
    }  
}
```

Lecture 13: linked list

16

Implementation of “dynamic array”

```
public void add(Object o) {  
    // if the store has space, insert o  
    if (size < capacity){  
        store[size] = o;  
        size++;  
    }  
    // otherwise need to enlarge the store
```

Lecture 13: linked list

17

Implementation of “dynamic array”

```
else {  
    capacity = capacity*2;//or whatever  
    Object[] newStore = new  
                        Object[capacity];  
    for (int i = 0; i < size; i++) {  
        newStore[i] = store[i];  
    }  
    store = newStore;  
    store[size] = o;  
    size++;  
}
```

Lecture 13: linked list

18

Implementation of “dynamic array”

```
public Object get(int i) {  
    // check if i is within bounds  
    if ((0 <= i)&&(i < size)) {  
        return store[i];  
    } else {  
        throw new  
            ArrayIndexOutOfBoundsException();  
    }  
}
```

Lecture 13: linked list

19

Implementation of “dynamic array”

```
public boolean remove(Object o) {  
    int index = -1;  
    // look for o in store  
    for (int i = 0; i < size; i++) {  
        if (store[i].equals(o)) {  
            index = i; break;  
        }  
    }  
    // if o not in store, return false  
    if (index == -1) return false;  
}
```

Lecture 13: linked list

20

Implementation of “dynamic array”

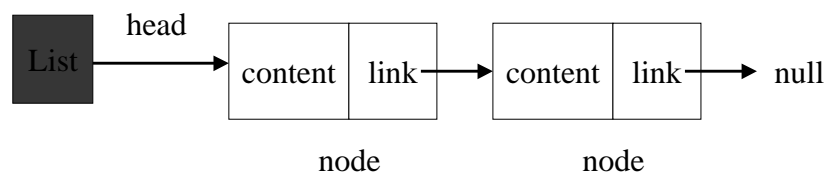
```
// else replace o by the next value and
// shift remaining values to the left
for (int i = index; i < size-1; i++){
    store[i] = store[i+1];
}
size--;
return true;
}
```

Lecture 13: linked list

21

Linked list

- A linked list consists of nodes .
- Each node has a contents fields which stores data (an Object) and a link field which says what the next item in the list is. A list has a head field which refers to the head of the list.

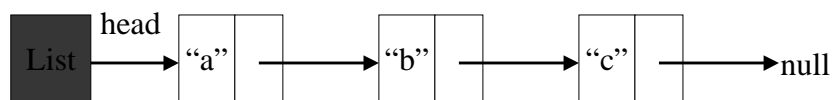


Lecture 13: linked list

22

Example: (“a”, “b”, “c”) list

- This list contains three strings (“a”, “b”, “c”).



Lecture 13: linked list

23

Linked list

- Really dynamic data structure - don't have to know initial size at all
- No problems with resizing
- Slow (sequential) access to elements in the middle of the list

Lecture 13: linked list

24

Linked list: how to add things

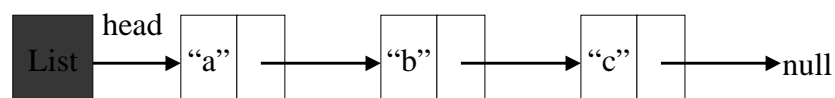
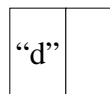
- We add things at the beginning of the list. To add a new node with an object d inside it to the list above, we
 - create a new node with d as its contents
 - link this node to the present head of the list (so that the present head is the next element)
 - make the new node to be the new head of the list

Lecture 13: linked list

25

Example: adding “d”

- Create a new node:

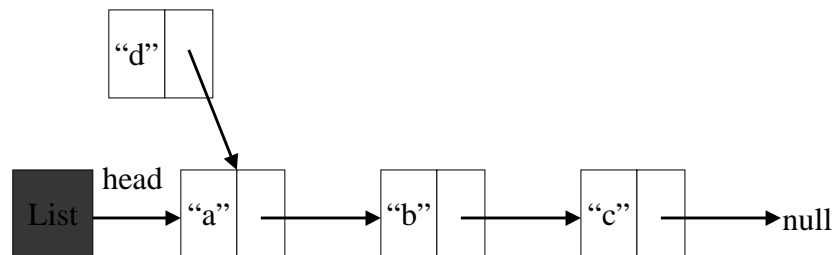


Lecture 13: linked list

26

Example: adding “d”

- Link it to the head of the list:

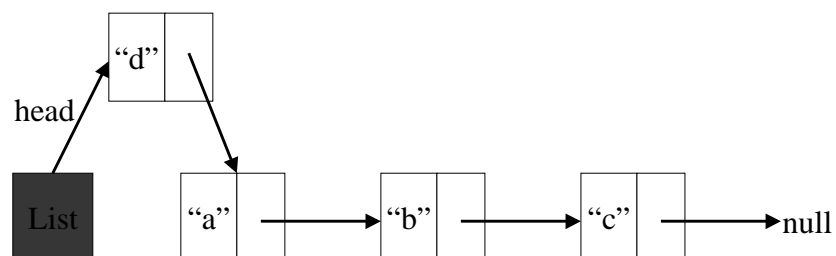


Lecture 13: linked list

27

Example: adding “d”

- Make it the new head of the list:



Lecture 13: linked list

28

Linked list: how to remove things

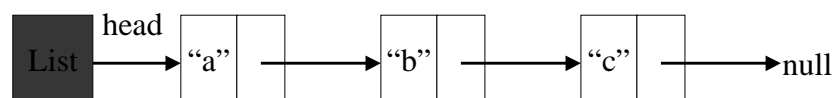
- We find the first node which has the object to be removed as contents, and bypass it: get the node which was linked to the removed node to link to the o-node's successor in the list.
- No other object now has a reference to the removed node so it will be garbage-collected by Java's garbage collector eventually. You don't have to destroy it yourself.

Lecture 13: linked list

29

Example: removing a node

- To remove “b”...

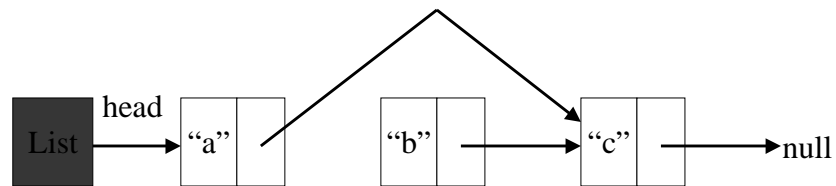


Lecture 13: linked list

30

Example: removing a node

- Get “a” to link to “c”:

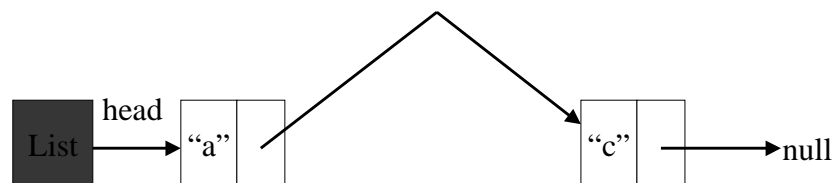


Lecture 13: linked list

31

Example: removing a node

- Eventually “b” will be garbage collected:



Lecture 13: linked list

32

Linked list implementation

- Need a class to represent nodes in a list
- Need a list class to store the head of the list and `add(Object o)`, `remove(Object o)`, `get(int i)` methods.

Lecture 13: linked list

33

Node class

```
class Node {  
    Object contents;  
    Node link;  
  
    public Node(Object o, Node next) {  
        this.contents = o;  
        this.link = next;  
    }  
}
```

Lecture 13: linked list

34

List class

```
public class List{
    Node head;

    public List() {
        this.head = null;
    }
    public void add(Object o){
        Node newHead = new Node (o, head);
        head = newHead;
    }
}
```

Lecture 13: linked list

35

List class

```
public boolean remove(Object o){
    if (head == null) return false;
    if (head.contents.equals(o)){
        head = head.link;
        return true;
    }
    // otherwise look for o in the list
}
```

Lecture 13: linked list

36

List class: remove() continued

```
Node checkNode = head;
while(checkNode.link != null){
    if (checkNode.link.contents.equals(o)){
        checkNode.link = checkNode.link.link;
        return true;
    } else {
        checkNode = checkNode.link;
    }
} // end while
return false;
}
```

Lecture 13: linked list

37

Access to element at index i

- Access to element at index *i* comes less naturally to lists than to dynamic arrays.
- We'll postpone implementing `get(int i)` method until the next lecture, when we look at Iterators.
- Intuitively, Iterators are objects which know how to walk or iterate along the list. A bit like Tokenizers, they have `boolean hasNext()` and `Object next()` methods.
- We will implement `get(int i)` using an Iterator: get it to produce *i* items in order, and then return the *i*th item.

Lecture 13: linked list

38

Digression: inner classes

- This is really a bit of a digression; but the List and Node classes are a good illustration of the use of inner classes.
- Another good illustration are graphical user interfaces, which we will see later in the course.

Inner, or nested, classes

- Classes can be members of other classes. Just like other members, they can be declared public or private, static etc. Why would one do this?
 - one class is auxiliary and only makes sense in the context of the other class (like the Node class)
 - convenience
 - encapsulation

Two kinds of nested classes

- top-level nested classes (static)
- inner classes (non-static)
- Static inner classes have the same status as top-level (ordinary) classes. They just tucked away in another class to keep the code tidy and readable.
- Static nested classes exist relative to the class where they are defined.
- Non-static nested classes exist relative to an object of the class.

Lecture 13: linked list

41

Static nested classes

- Example from Arnold and Gosling:

```
public class BankAccount {  
    private long number;  
    private long balance;  
    public static class Permissions {  
        public boolean canDeposit,  
                        canWithdraw,  
                        canClose;  
    }  
    // class BankAccount continued
```

Lecture 13: linked list

42

Inner classes

- Non-static nested classes are called inner classes. (In Java Gently they are called member classes).
- They cannot have static members.
- They have access to enclosing instance of other class, even to private fields and methods; **this** is used to refer to the inner class members, **this** prefixed by the outer class name to the outer class members.

Lecture 13: linked list

43

Example

```
public class List {  
    class Node {  
        Object contents;  
        Node link;  
        Node(Object o, Node next) {  
            this.contents = o;  
            this.link = next;  
        }  
    } // end of Node class  
    Node head;    // List continued
```

Lecture 13: linked list

44

Example

- When the List class is compiled, so is the inner class Node. Instead of **Node.class**, we get

List\$Node.class

- In general, when an nested (static or non-static) class A of class B is compiled, its name is given as B\$A.class

Summary

- For nested classes, see Sun tutorial
<http://java.sun.com/docs/books/tutorial/java/javaOO/nested.html>
- For various linked lists and flexible arrays, any textbook with data structures in it, and Java API classes.
- Will see more of it in the Algorithms and Data Structures course next year.
- Good exercise: implement a recursive list in Java (like a list in Haskell).
- Snag to watch for: empty list not the same as null; probably will need a special empty list class.