Linked Data Structures

MSCI 240: Algorithms & Data Structures

lecture summary

review: value vs. reference semantics

linked data structures and `ListNode`

Topic	Building Java Programs	Algorithms (Sedgewick)
classes, ADTs	chapter 8	1.2
arrays	chapter 7	
ArrayList<T>	chapter 10	1.3
Stack/Queue	chapter 14, (11)	1.3
LinkedList	chapter 16	1.3
Complexity	chapter 13	1.4
Searching		pp. 46-47
Sorting		chapter 2.1-2.3
Recursion	chapter 12	1.1 (p. 25)
BSTs	chapter 17	chapter 3.1-3.2
Dictionaries	chapter 18.1	chapter 3.4
Graphs	N/A (Wikipedia good)	chapter 4.1
Heaps/Priority Queues	chapter 18.2	chapter 2.4

value vs. reference semantics

does the following `swap` method work? why or why not?

```
public static void main(String[] args) {
    int a = 7;
    int b = 35;

    // swap a with b
    swap(a, b);

    System.out.println(a + " " + b);
}

public static void swap(int a, int b) {
    int temp = a;
    a = b;
    b = temp;
}
```

expected output:

```
35 7
```

actual output (test fails!):

```
7 35
```

value semantics: behaviour where values are copied when assigned, passed as parameters, or returned

- all primitive types in Java use value semantics
- when one variable is assigned to another, its value is copied
- modifying the value of one variable does not affect others

reference semantics: behaviour where variables actually store the address of an object in memory

- when one variable is assigned to another, the object is not copied; both variables refer to the same object
- modifying the value of one variable will affect others

arrays and objects use reference semantics—why?

- efficiency—copying large objects slows down a program
- sharing—it's useful to share an object's data among methods

what is the output?

```
Scanner scanner =
    new Scanner("Some tokens!");
Scanner scanner2 = scanner;

String a = scanner2.next();
String b = scanner.next();

System.out.println(a + " " + b);
```

A. Some tokens!

B. Some Some

C. tokens! tokens!

D. tokens! Some

E. does not compile

what is the output?

```
int[] a1 = { 4, 15, 8 };
int[] a2 = a1;
a2[0] = 7;
System.out.println(Arrays.toString(a1));
```

A. [4, 15, 8]

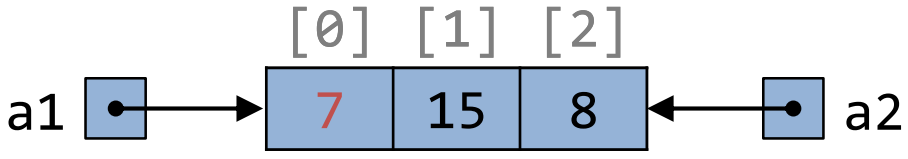
B. [I@15db9742

C. [7, 15, 8]

D. 7

E. does not compile

```
int[] a1 = { 4, 15, 8 };
int[] a2 = a1;
a2[0] = 7;
System.out.println(Arrays.toString(a1));
```



recall

dereference: to access data or methods of an object with the dot notation, such as `s.length()`

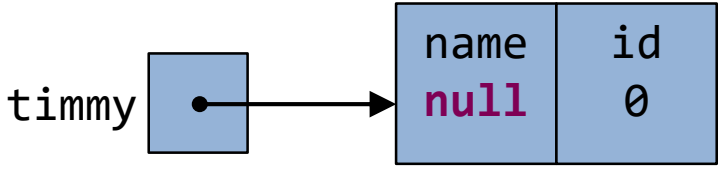
it is illegal to dereference **`null`** (causes an exception)

`null` is not any object, so it has no methods or data

objects can store references to other objects as fields

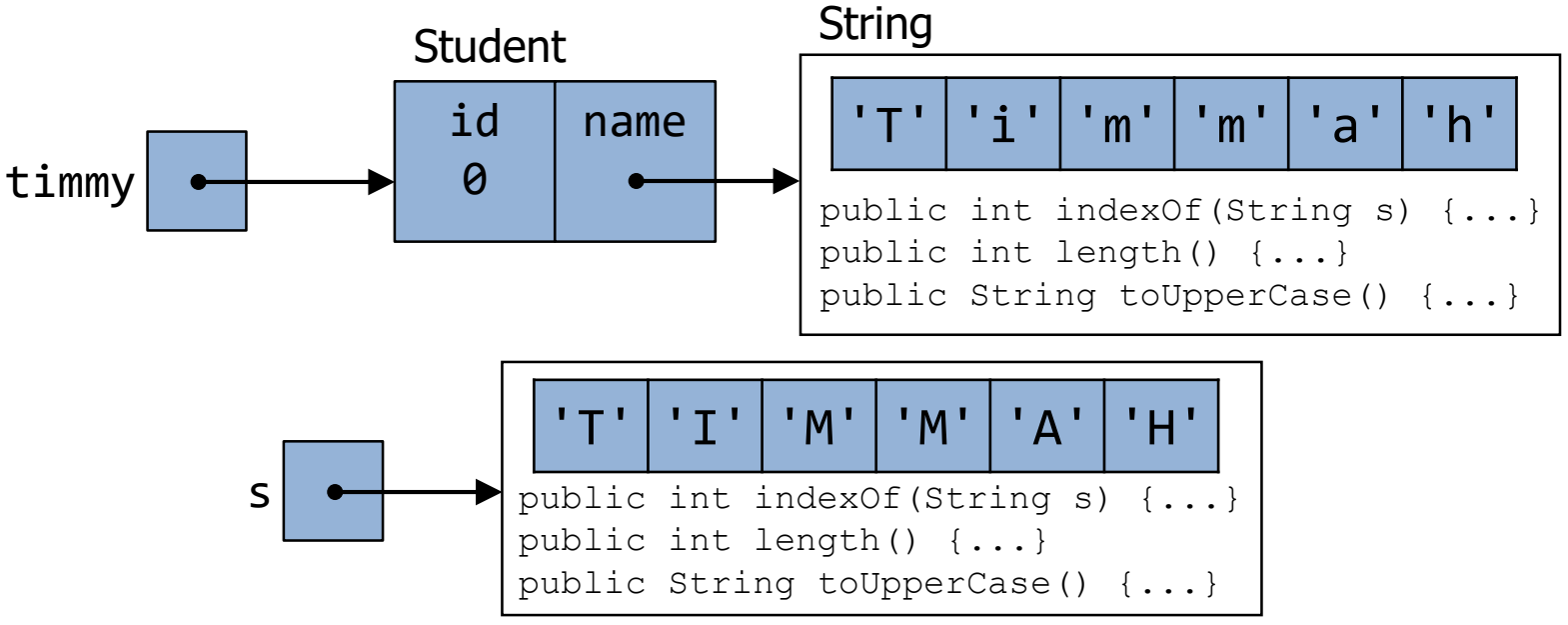
unset reference fields of an object are initialized to **null**

```
public class Student {
    String name;
    int id;
}

Student timmy = new Student();
```



when you use a . after an object variable, Java goes to the memory for that object and looks up the field/method

```
Student timmy = new Student();
timmy.name = "Timmah";
String s = timmy.name.toUpperCase();
```



reference semantics summary

there are two types of semantics used in Java:

value semantics (used by all primitive types): value is copied and changes do not affect the original variable

reference semantics (used by objects & arrays): a reference to the original is used, and changes affect both the new and original variables

objects can store references to other objects as fields

. after an object tells Java to look up field/method being referenced

can keep dereferencing objects (e.g., `timmy.name.toUpperCase()`)

linked data structures

references to same type

what would happen if we had a class that declared one of its own type as a field?

```
public class Strange {
    private String name;
    private Strange other;
}
```

will this compile?

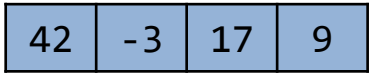
- if so, what is the behaviour of the other field? what can it do?
- if not, why not? what is the error and the reasoning behind it?

linked data structures

all collections we use and implement in this course use one of the following two underlying data structures:

an array of all elements

- `ArrayList`, `Stack`, `HashSet`, `HashMap`



a set of linked objects, each storing one element, and one or more reference(s) to other element(s)

- `LinkedList`, `TreeSet`, `TreeMap`

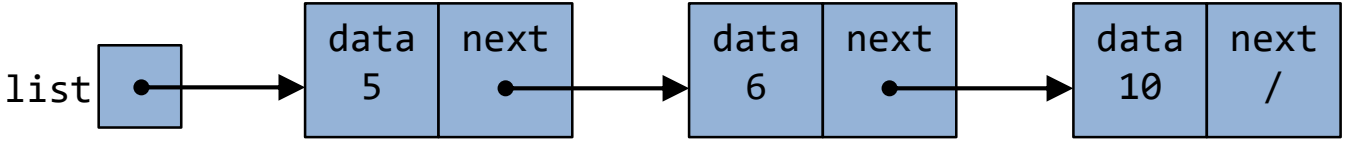
a list node

```
public class ListNode {
    int data;
    ListNode next;
}
```

each `ListNode` object stores:

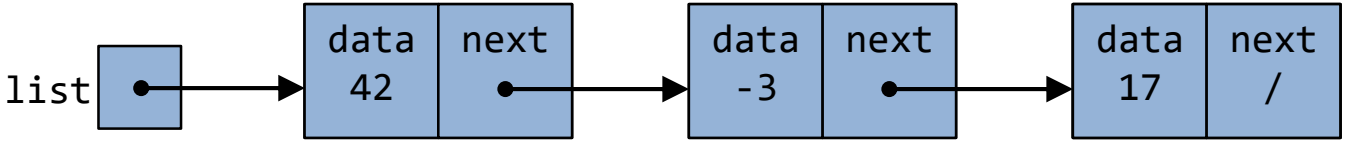
- one piece of integer data
- a reference to another list node

`ListNode`s can be "linked" in chains to store a list of values:



List node client example

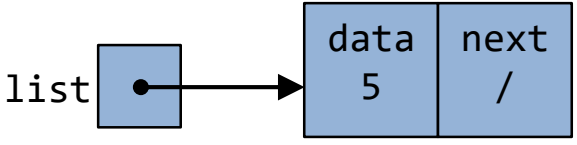
```
public class ConstructList1 {
    public static void main(String[] args) {
        ListNode list = new ListNode();
        list.data = 42;
        list.next = new ListNode();
        list.next.data = -3;
        list.next.next = new ListNode();
        list.next.next.data = 17;
        list.next.next.next = null;
        System.out.println(list.data + " " + list.next.data
                + " " + list.next.next.data); // 42 -3 17
    }
}
```



List node w/ constructor

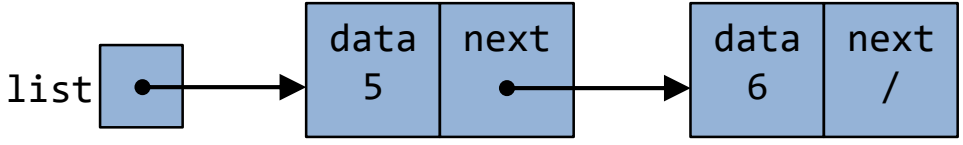
```
public class ListNode {
    int data;
    ListNode next;

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

    public ListNode(int data, ListNode next) {
        this.data = data;
        this.next = next;
    }
}
```

list



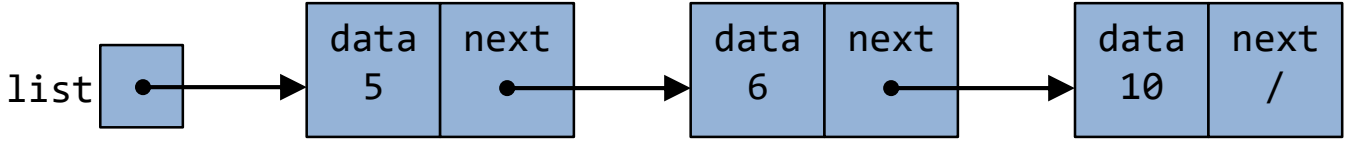
exercise: write Java code to create this structure using only one variable, of type `ListNode`

list



exercise: write code (using previous code) to create this structure without any new variables

list



exercise: write code (using previous code) to create this structure without any new variables

reassigning references

references vs. objects

```
variable = value;
```

a `variable` (left side of =) is an arrow (the base of an arrow)

a `value` (right side of =) is an object (a box; what arrows point at)

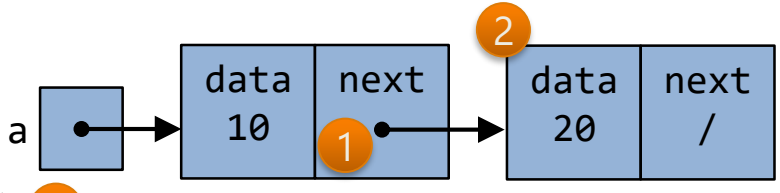
for the list at right:

`a.next = value;`

means to adjust where 1 points

`variable = a.next;`

means to make `variable` point at 2



reassigning references

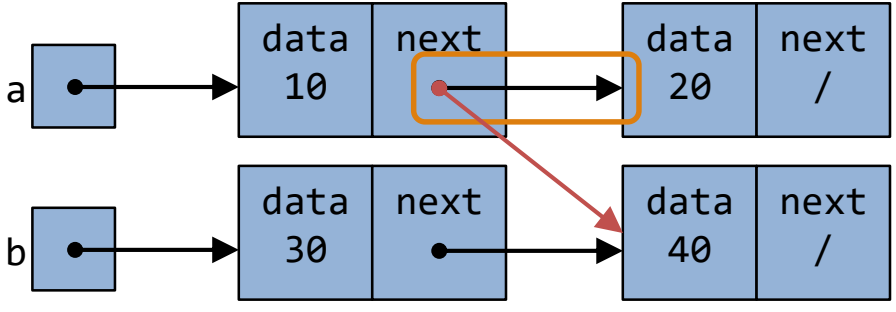
when you say:

```
a.next = b.next;
```

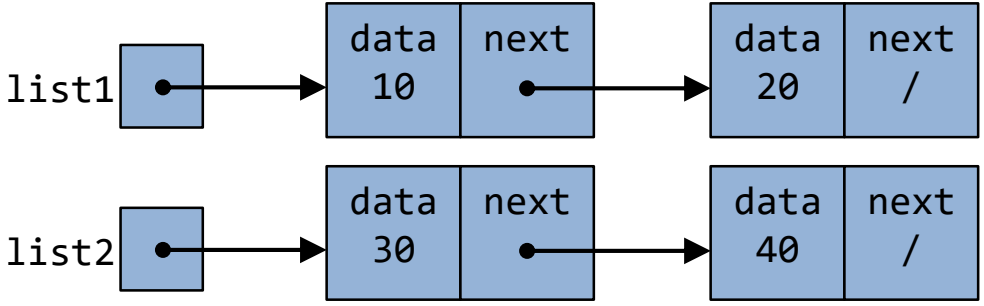
you are saying:

"make the variable `a.next` refer to the same value as `b.next`"

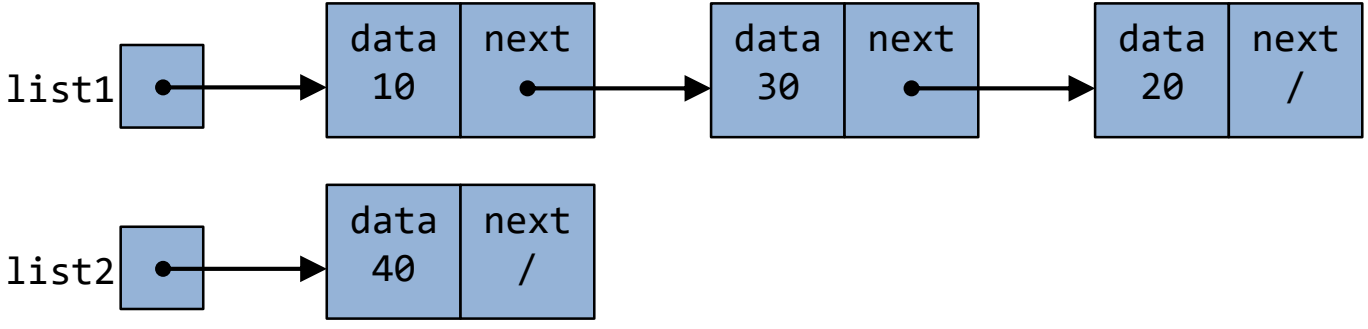
or, "make `a.next` point to the same place that `b.next` points"



what set of statements turns this picture:

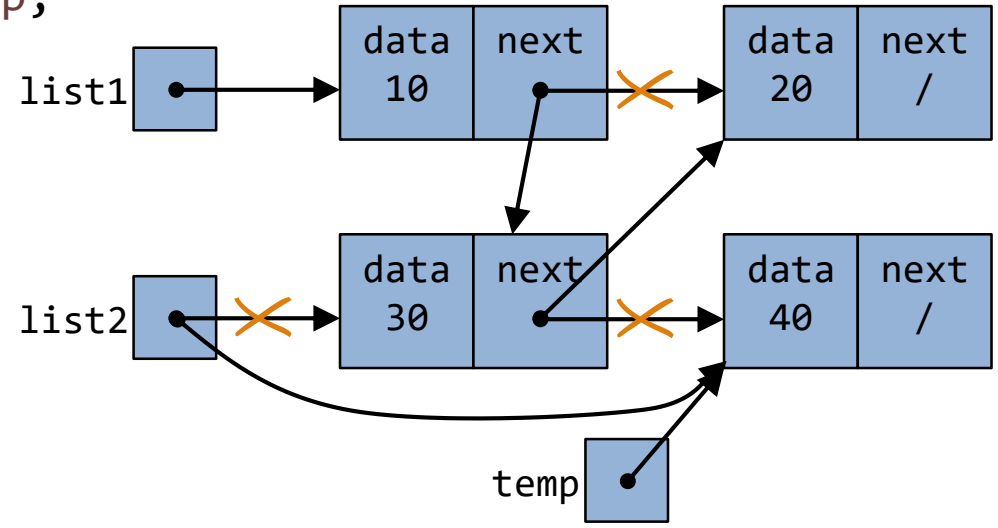


into this?



solution

```
ListNode temp = list2.next;
list2.next = list1.next;
list1.next = list2;
list2 = temp;
```



linked data structure summary

a class can have a field of the same type

linked data structures are made up of nodes (e.g., `ListNode`) that contain a value and a link to another node

these nodes can be chained together to form a list

the links can be assigned and reassigned to other nodes

next class:

linked lists