Last time’s big ideas

1. When we want an array of objects, we store their references in the array

2. It is important to distinguish between the specification and implementation of a class

3. public and private control access to fields and methods

answer the Socrative Quiz: Pick a pair of the above three big ideas. In 1-2 sentences tell how the two ideas you picked are related to each other.
CS 2230
CS II: Data structures

Meeting 5: more references(pointers), linked lists
Brandon Myers
University of Iowa
First let’s finish the PatientDatabase major point here:

• **public** and **private** control access to fields and methods
// Register a new Patient in the database (if we have space)
boolean registerNewPatient(String name) {
    if (numPatients == patients.length) return false;

    // since they haven't been measured we will give height=0
    Patient newp = new Patient(name, 0);

    // start with the new patient at the end of the list
    patients[numPatients] = newp;

    numPatients+=1;

    // keep swapping the patient with the previous patient
    // until it is in alphabetical order
    int i = numPatients-1;
    while (i > 0 &&
        patients[i].name.compareTo(patients[i-1].name) < 0) {

        swapPatients(i, i-1);
        i--;
    }

    return true;
}

we’ll get to swapPatients() next

"Alice".compareTo("Bob") \rightarrow -1
"Bob".compareTo("Alice") \rightarrow 1
class PatientDatabase {
    private Patient[] patients;

    private void swapPatients(int a, int b) {
        Patient pa = patients[a];
        Patient pb = patients[b];
        patients[a] = pb;
        patients[b] = pa;
    }

    boolean registerNewPatient(String name) {
        while (true) {
            swapPatients(i, i-1);
        }
    }
}
class PatientDatabase {
    private Patient[] patients;

    private void swapPatients(int a, int b) {
        Patient pa = patients[a];
        Patient pb = patients[b];
        patients[a] = pb;
        patients[b] = pa;
    }

    boolean registerNewPatient(String name) {
        while ( ) {
            swapPatients(i, i-1);
        }
    }
}
Making patients and swapPatients private is most an example of which object-oriented design principle?

A) Abstraction  
B) Encapsulation  
C) Modularity

```java
class PatientDatabase {
    private Patient[] patients;

    private void swapPatients(int a, int b) {
    }

    boolean registerNewPatient(String name) {
        while ( ) {
            swapPatients(i, i-1);
        }
    }
}
```
Implementation of the other method

```java
// Print all patient names in alphabetical order
void printNamesAlphabetically() {
    for (int i=0; i<numPatients; i++) {
        System.out.println(patients[i].name);
    }
}
```

(see the whole PatientDatabase class in PatientDatabase.java, and run the program for yourself)
Today’s big ideas

• understand reference manipulation better

• we can build a list out of ”Nodes” linked by references (a linked list)

• references that don’t point to anything store the value null
What are some cons of Arrays?

(i.e., downsides, as in pros and cons)
Arrays can be difficult to work with

**EXAMPLE 1:** I want to insert the String “A” at the front of the array...move everything down!!

<table>
<thead>
<tr>
<th>“B”</th>
<th>“C”</th>
<th>“D”</th>
<th>“E”</th>
<th>“F”</th>
</tr>
</thead>
</table>

the good ol’ days (lists in Python)

```python
alphabet = ['B', 'C', 'D', 'E', 'F']
alphabet.insert(0, 'A')
```

**EXAMPLE 2:** I want to add the number 400 at the end...out of room so copy to a bigger array??

<table>
<thead>
<tr>
<th>100</th>
<th>200</th>
<th>300</th>
</tr>
</thead>
</table>

the good ol’ days (lists in Python)

```python
hundreds = [100, 200, 300]
hundreds += [400]
```
New data structure: *linked list*

\[100, 200, 300\]

Let’s implement the python list like this:
New data structure: *linked list*

```
[100,200,300]
Let's implement the python list like this:
```

```
hundreds = [100,200,300]
hundreds += [400]
```

solves the out-of-space problem that arrays have just add a new node to the end
hundreds = [100, 200, 300]
hundreds += [400]

solves the out-of-space problem that arrays have just add a new node to the end

hundreds = [200, 300]
alphabet.insert(0, 100)

solves the insert at the front problem that arrays have just add a new node to the front
Peer instruction

We are going to make a linked list from `ListNode` objects.

The type for `data` is `int`. What should be the type for `next`?

a) `int[]`
b) `int`
c) `double`
d) `ListNode[]`
e) `ListNode`
Creating a new ListNode

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

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

    public static void main(String[] args) {
        ListNode head = new ListNode(100);
    }
}
```

The convention is to denote null as a slash \.
Let’s play with references

ListNode p1 = new ListNode(100);

ListNode p2 = new ListNode(200);

p2 = p1;

p2.next = new ListNode(300);
Let’s play with references

ListNode p1 = new ListNode(100);

ListNode p2 = new ListNode(200);

p2 = p1;

p2.next = new ListNode(300);
Let’s play with references

ListNode p1 = new ListNode(100);

ListNode p2 = new ListNode(200);

p2 = p1;

p2.next = new ListNode(300);
Let’s play with references

ListNode p1 = new ListNode(100);

ListNode p2 = new ListNode(200);

p2 = p1;

p2.next = new ListNode(300);
Let’s play with references

ListNode p1 = new ListNode(100);

ListNode p2 = new ListNode(200);

p2 = p1;

p2.next = new ListNode(300);
Peer instruction

How do you know if a ListNode is the last one in the list?

a) it has no next field
b) its next field points to itself
c) its next field points to the beginning of the list
d) its next field is null
e) its next field and data field are equal
The *append* method

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

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

    public void append(int d) {
        if (next == null) {
            next = new ListNode(d);
        } else {
            next.append(d);
        }
    }
}
```
Today’s big ideas

• understand reference manipulation better

• we can build a list out of ”Nodes” linked by references (a linked list)

• references that don’t point to anything store the value null

next week: ENCAPSULATION for ListNodes, searching for data stored in arrays and linked lists!!!
What to do now

• Quiz 2 then HW 2

• You will soon receive your Socrative id as an entry in your ICON gradebook (creative use of technology 😊)

• Collect your Change of Registration form at the front right now if you’ve submitted one