Nodes

Nodes
Poll

Q: How can we indicate the absence of an Object in Java code?
Poll

Q: How can we indicate the absence of an Object in Java code?
A: Use a NULL reference
Q: Given the following code, what is printed? (Recall the simple `Point.java` class, with `x` and `y` instance variables & getters/setters)

```java
public class UsePoint {
    public static void main(String[] args) {
        Point p = new Point(3, 2);
        Point q = p;
        p.setX(64);
        System.out.println(q.getX());
    }
}
```
Q: Given the following code, what is printed? (Recall the simple `Point.java` class, with `x` and `y` instance variables & getters/setters)

A: 64

```java
class UsePoint {
    public static void main(String[] args) {
        Point p = new Point(3, 2);
        Point q = p;
        p.setX(64);
        System.out.println(q.getX());
    }
}
```
Previously, if we ever wanted to store a sequence of data, we used arrays (or ArrayLists, which are just arrays that Java manages in fancy ways for you.)

- Arrays store data in contiguous memory (each element is next to each other in memory)
- We could access a specific position with an index

Example array declaration: `int[] values = {2814, 2048, 867, 5309};`

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2814</td>
<td>2048</td>
<td>867</td>
<td>5309</td>
</tr>
</tbody>
</table>
Array Flaws

What if we wanted to add a new value to the beginning of an array?

- All we have is a variable `values` that stores a reference to an array object in memory
- That array object is a contiguous portion of "heap" space that takes up a fixed amount of space

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>2814</td>
<td>2048</td>
<td>867</td>
<td>5309</td>
</tr>
</tbody>
</table>
Array Flaws

What if we wanted to add a new value to the beginning of an array?
Need to:

- create a new array object (using `new`) with a bigger size
- copy over old values to positions 1-4
- and then insert new value at position 0
**Idea: Non-contiguous Storage**

What if we tried to store data in memory that is noncontiguous, where each element is spread apart from one another?

How would we know where the next element is?

Want to:

- Keep track of the first element (just like arrays)
- have each value store a reference to the "next" value
**Introduction: Nodes**

**Node:** a class containing one or more data fields that store data, and a *reference* to another linked node

- connect these objects together to form a structure of linked nodes

Linked nodes are the building blocks of programs (data structures) that store a large amount of data without using an array

- Allow us to more easily modify a collection of data
- Don’t have to worry about knowing the length before hand
Two examples, one for each kind of data that the node will store.

```java
public class Node {
    public Node next; // Point to next node
    public int data; // Value (int) for this node
    // Constructor
    public Node(int data, Node next) {
        this.data = data;
        this.next = next;
    }
    // data fields are public, no need for getters and setters
}
```

```java
public class Node {
    public Node next; // Point to next node
    public Computer data; // Value (Computer) for this node
    public Node(Computer data, Node next) {
        this.data = data;
        this.next = next;
    }
}
```
Chain of Nodes: Starting the Chain

Let's build a chain of nodes! Each node stores an integer value and a reference to another Node.

```java
Node head = new Node(20, null);
```

The next (following) node of head's pointee is a null reference.

The data stored in head's pointee is 20
Chain of Nodes: Adding a Follower

To add a new node at the end of the chain, we must construct a new Node and point the head node to it.

```java
head.next = new Node(30, null);
```

The next (follower) node of head’s pointee is a new Node storing 30 as its data.
Chain of Nodes: Adding Another Follower

Chain of nodes

head.next.next = new Node(10, null);

The next (follower) node of head’s follower is a new Node storing 10 as its data.
Chain of Nodes: All at Once

Putting everything together:

```java
Node head = new Node(20, null);
head.next = new Node(30, null);
head.next.next = new Node(10, null);
```

The last node has `null` as its `next` to mark that there are no more nodes.
Practice Working with Nodes

How can we get from this program state:

\[
\begin{align*}
\text{p} & \rightarrow 8 & \rightarrow 6 \\
\text{m} & \rightarrow 7 & \rightarrow 5 
\end{align*}
\]

To this program state? (Without modifying node data directly...)

\[
\begin{align*}
\text{p} & \rightarrow 8 & \rightarrow 6 & \rightarrow 7 \\
\text{m} & \rightarrow 5 
\end{align*}
\]
Chain of nodes: practice

- Current state:

```
<table>
<thead>
<tr>
<th>P</th>
<th>8</th>
<th></th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>m</td>
<td>7</td>
<td></td>
<td>5</td>
</tr>
</tbody>
</table>
```

No modifying node data directly
Chain of nodes: practice

- Current state:

```
p 8 6
7 5
m
```

No modifying node data directly

```
p.next.next = m;
```
Chain of nodes: practice

- Current state:

```cpp
p.next.next = m;
m = m.next;
```
Chain of nodes: practice

- Current state:

```
p.next.next = m;
m = m.next;
p.next.next.next = null;
```
Chain of nodes: iteration

To iterate through a chain of nodes:

- We don’t need to know how many nodes are in the chain
- Just know that the last node’s next field is a null reference

Steps:

- Create a temporary node variable that starts by pointing to the head of the chain (sharing)
- Iterate/loop by following the next references with each iteration, update the pointee of the temporary node
- Stop when the temporary node points to a null reference
Chain of nodes: iteration

Given the following chain...
Chain of nodes: iteration

Create a temporary node that points to the head of the chain

```java
Node curr = head; // curr and head are aliases for each other
```
Chain of nodes: iteration

Start the loop & stop when `curr` points to the last node in the chain

```java
Node curr = head; // curr and head are aliases for each other
while (curr != null) {
    curr = curr.next; // advance curr to point to the next node.
}
```
Chain of nodes: iteration

curr now points to the node storing 30

Node curr = head; //curr and head are aliases for each other
while (curr != null) {
    curr = curr.next; // advance curr to point to the next node.
}

Note that head did not move.
Chain of nodes: iteration

```
Node curr = head;  // curr and head are aliases for each other
while (curr != null) {
    // while the pointee of curr is not null,
    curr = curr.next;  // advance curr to point to the next node.
}
```

curr now points to the node storing 10
Chain of nodes: iteration

Curr now points to the node storing 5

Node curr = head; // curr and head are aliases for each other
while (curr != null) {
    curr = curr.next; // advance curr to point to the next node.
}
Chain of nodes: iteration

Curr is now a null reference

```
Node curr = head; //curr and head are aliases for each other
while (curr != null) { // while the pointee of curr is not null,
curr = curr.next; // advance curr to point to the next node.
}
```
Chain of nodes: iteration

Putting everything together:

The following code will print all the values stored in our chain

```java
Node curr = head;  // curr and head are aliases for each other
while (curr != null) {
    System.out.println(curr.data);
    curr = curr.next;
}
```

Will print: 20 30 10 5
Chain of nodes: iteration (for loop)

As a for loop instead:

```java
for (Node curr = head; curr != null; curr = curr.next) {
    System.out.println(curr.data);
}
```

Will print: 20 30 10 5