CS 106 INTRODUCTION TO DATA STRUCTURES

SPRING 2020

PROF. SARA MATHIESON

HAVERFORD COLLEGE

ADMIN

- **Midterm 1 in-class on Thursday**
    - Create one-page (front & back) "cheat-sheet"
- **Office Hours TODAY! 4:30-6pm (H110)**
- **Remind me to hand back Handout 12 and Lab 2**
- **We DO have lab this week and attendance is still required. You do NOT need to do anything in advance though, you may begin Lab 4 during lab.**
    - Lab 4 will be posted Wed or Thurs

MAR 3 OUTLINE

- Queues (theory and implementation)
- Review arrays and ArrayLists
- Review Nodes and Linked Lists
- Practice Problems

MAR 3 OUTLINE

- **Queues (theory and implementation)**
- Review arrays and ArrayLists
- Review Nodes and Linked Lists
- Practice Problems

QUEUES

How would you want a data structure to work for waiting in line at a store?

What is the rate of input is different than the rate of output?

> **Example: people show up to the DMV at random times, but processing takes about the same time for each person**

Define an abstract data type (ADT).

THE QUEUE ADT

Insertions and deletions are First In First Out (FIFO)

-Insert at the back

-Delete from the front

Operations:

- `enqueue(Object)`
- `Object dequeue()`
- `Object first()`
- `int size()`
- `boolean isEmpty()`

IMPLEMENTING A QUEUE

Brainstorm: using the data structures we know about, how could we implement this ADT?

IMPLEMENTING A QUEUE

Brainstorm: using the data structures we know about, how could we implement this ADT?

Many ways to implement a Queue! Underneath, we can use:

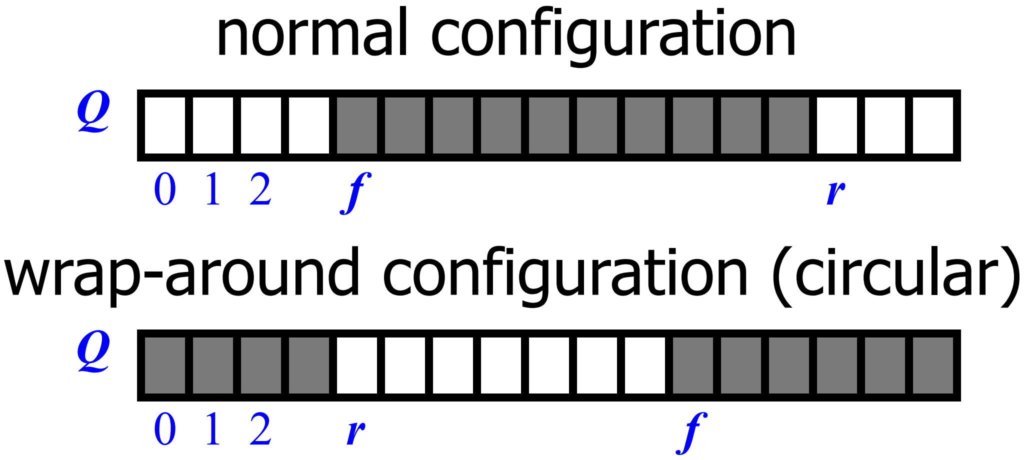
* Arrays
* Lists
* Stacks

ARRAY-BASED QUEUE IMPLEMENTATION

An array of size `n` in a circular fashion

Two `int`s to track front and size

- `f`: index of the front element
- `size`: number of stored elements



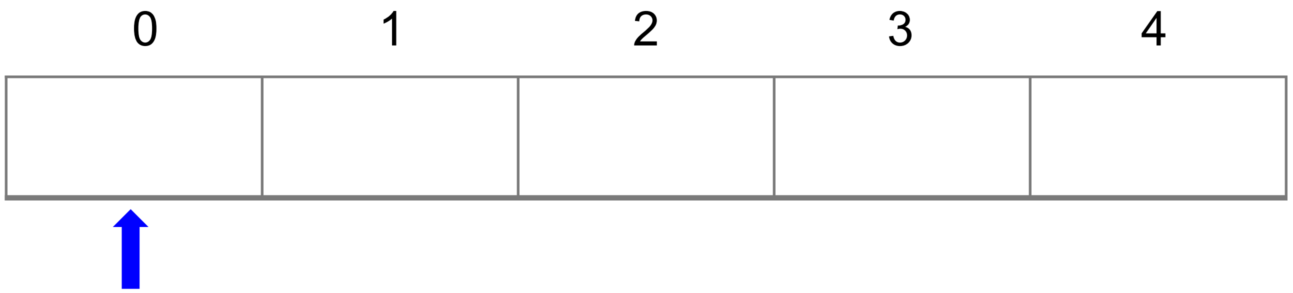
normal configuration

wrap-around configuration (circular)

EXAMPLE

```
Queue<Integer> testQ = new ArrayQueue<Integer>(5);
testQ.enqueue(3);
testQ.enqueue(4);
testQ.enqueue(5);
testQ.enqueue(2);
testQ.enqueue(1);
testQ.dequeue();
testQ.dequeue();
testQ.enqueue(-9);
testQ.dequeue();
testQ.dequeue();
testQ.dequeue();
testQ.dequeue();
testQ.enqueue(-8);
```

Size: 0

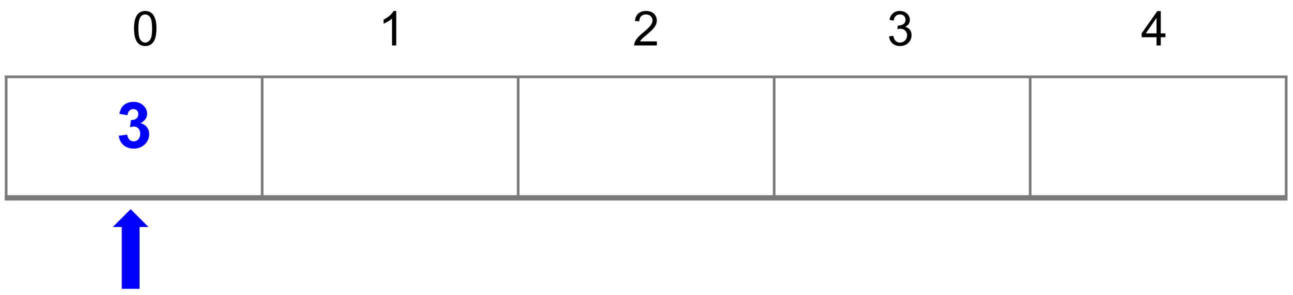


Arrow is front

EXAMPLE

```
Queue<Integer> testQ = new ArrayQueue<Integer>(5);
testQ.enqueue(3);
testQ.enqueue(4);
testQ.enqueue(5);
testQ.enqueue(2);
testQ.enqueue(1);
testQ.dequeue();
testQ.dequeue();
testQ.enqueue(-9);
testQ.dequeue();
testQ.dequeue();
testQ.dequeue();
testQ.dequeue();
testQ.enqueue(-8);
```

Size: 1



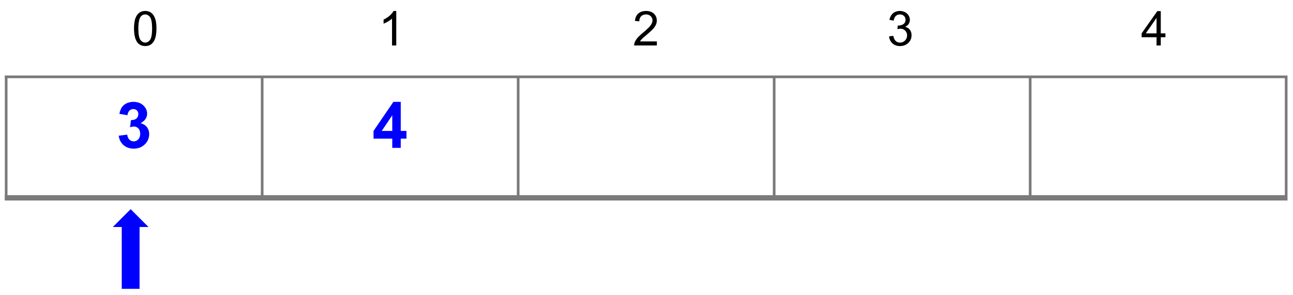
0	1	2	3	4
3				

Arrow is front

EXAMPLE

```
   Queue<Integer> testQ = new ArrayQueue<Integer>(5);
   testQ.enqueue(3);
➡ testQ.enqueue(4);
   testQ.enqueue(5);
   testQ.enqueue(2);
   testQ.enqueue(1);
   testQ.dequeue();
   testQ.dequeue();
   testQ.enqueue(-9);
   testQ.dequeue();
   testQ.dequeue();
   testQ.dequeue();
   testQ.dequeue();
   testQ.enqueue(-8);
```

Size: 2



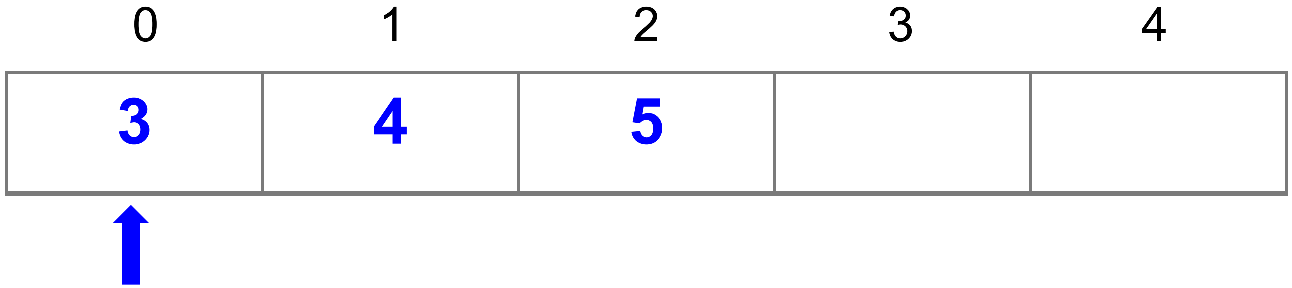
0	1	2	3	4
3	4			

Arrow is front

EXAMPLE

```
Queue<Integer> testQ = new ArrayQueue<Integer>(5);
testQ.enqueue(3);
testQ.enqueue(4);
testQ.enqueue(5);
testQ.enqueue(2);
testQ.enqueue(1);
testQ.dequeue();
testQ.dequeue();
testQ.enqueue(-9);
testQ.dequeue();
testQ.dequeue();
testQ.dequeue();
testQ.dequeue();
testQ.enqueue(-8);
```

Size: 3



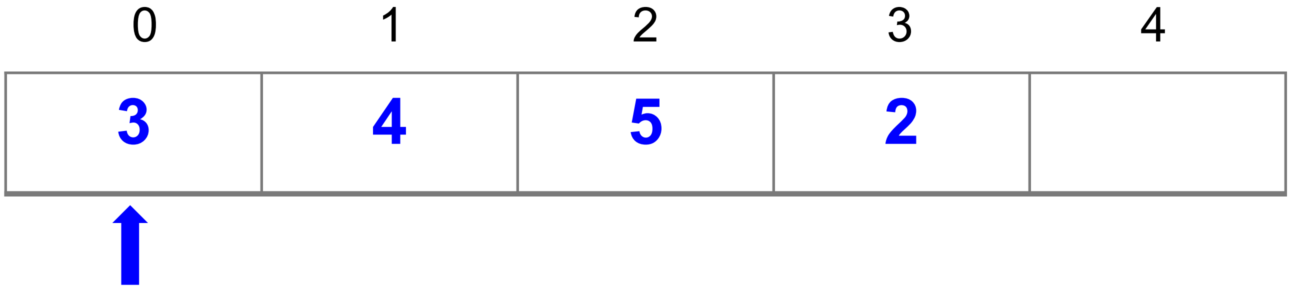
0	1	2	3	4
3	4	5		

Arrow is front

EXAMPLE

```
Queue<Integer> testQ = new ArrayQueue<Integer>(5);
testQ.enqueue(3);
testQ.enqueue(4);
testQ.enqueue(5);
testQ.enqueue(2);
testQ.enqueue(1);
testQ.dequeue();
testQ.dequeue();
testQ.enqueue(-9);
testQ.dequeue();
testQ.dequeue();
testQ.dequeue();
testQ.dequeue();
testQ.enqueue(-8);
```

Size: 4



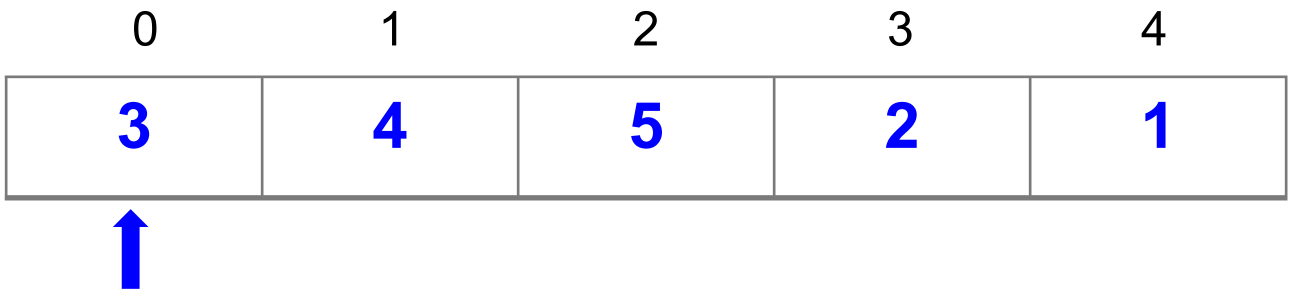
0	1	2	3	4
3	4	5	2	

Arrow is front

EXAMPLE

```
Queue<Integer> testQ = new ArrayQueue<Integer>(5);
testQ.enqueue(3);
testQ.enqueue(4);
testQ.enqueue(5);
testQ.enqueue(2);
testQ.enqueue(1);
testQ.dequeue();
testQ.dequeue();
testQ.enqueue(-9);
testQ.dequeue();
testQ.dequeue();
testQ.dequeue();
testQ.dequeue();
testQ.enqueue(-8);
```

Size: 5



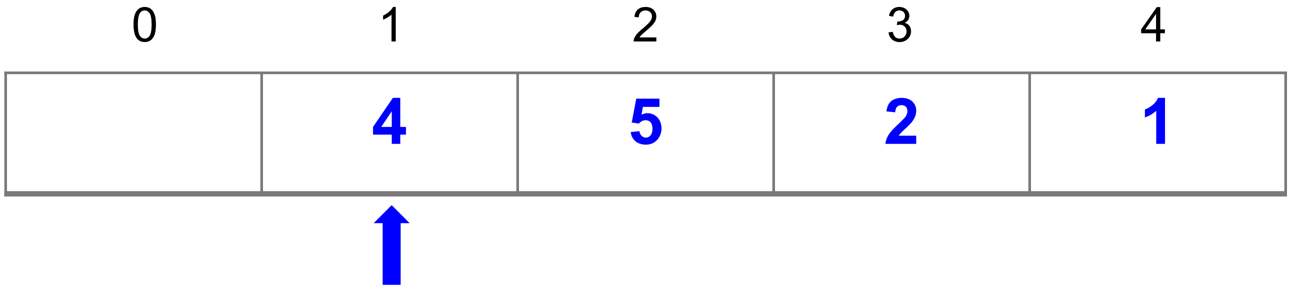
0	1	2	3	4
3	4	5	2	1

Arrow is front

EXAMPLE

```java
Queue<Integer> testQ = new ArrayQueue<Integer>(5);
testQ.enqueue(3);
testQ.enqueue(4);
testQ.enqueue(5);
testQ.enqueue(2);
testQ.enqueue(1);
testQ.dequeue();
testQ.dequeue();
testQ.enqueue(-9);
testQ.dequeue();
testQ.dequeue();
testQ.dequeue();
testQ.dequeue();
testQ.enqueue(-8);
```

Size: 4



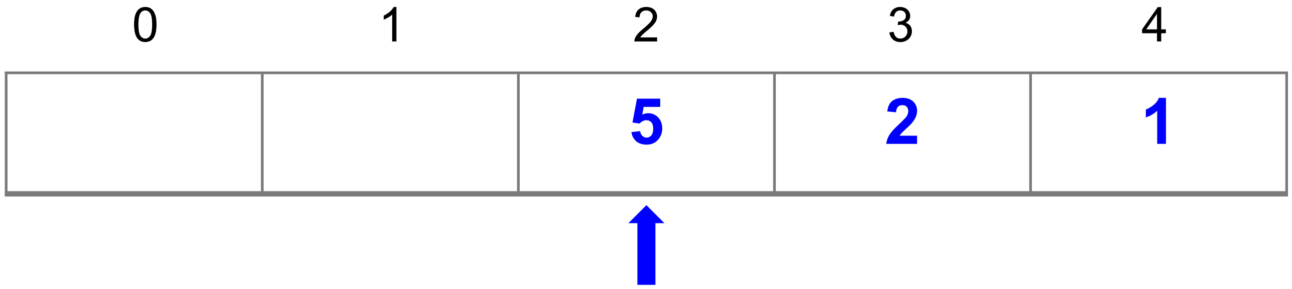
0	1	2	3	4
	4	5	2	1

Arrow is front

EXAMPLE

```
Queue<Integer> testQ = new ArrayQueue<Integer>(5);
testQ.enqueue(3);
testQ.enqueue(4);
testQ.enqueue(5);
testQ.enqueue(2);
testQ.enqueue(1);
testQ.dequeue();
testQ.dequeue();
testQ.enqueue(-9);
testQ.dequeue();
testQ.dequeue();
testQ.dequeue();
testQ.dequeue();
testQ.enqueue(-8);
```

Size: 3



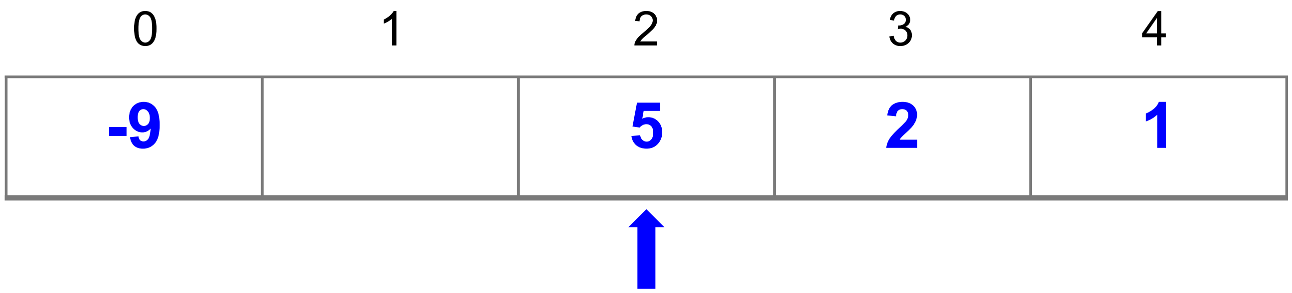
0	1	2	3	4
		5	2	1

Arrow is front

EXAMPLE

```java
Queue<Integer> testQ = new ArrayQueue<Integer>(5);
testQ.enqueue(3);
testQ.enqueue(4);
testQ.enqueue(5);
testQ.enqueue(2);
testQ.enqueue(1);
testQ.dequeue();
testQ.dequeue();
testQ.enqueue(-9);
testQ.dequeue();
testQ.dequeue();
testQ.dequeue();
testQ.dequeue();
testQ.enqueue(-8);
```

Size: 4



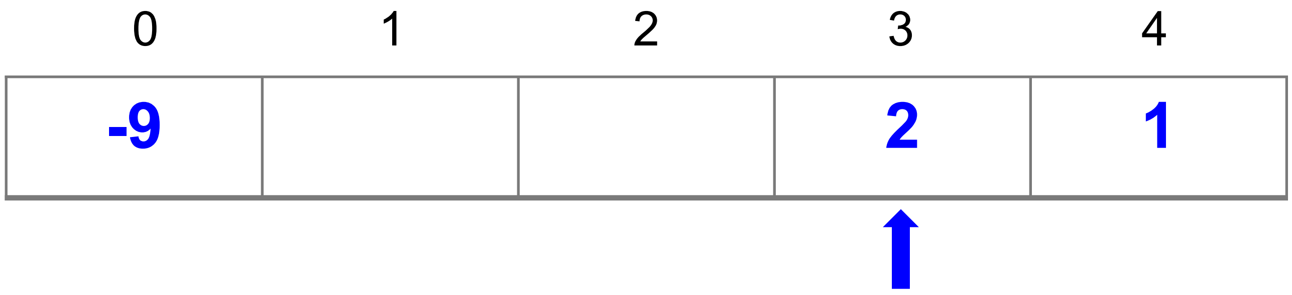
0	1	2	3	4
-9		5	2	1

Arrow is front

EXAMPLE

```
Queue<Integer> testQ = new ArrayQueue<Integer>(5);
testQ.enqueue(3);
testQ.enqueue(4);
testQ.enqueue(5);
testQ.enqueue(2);
testQ.enqueue(1);
testQ.dequeue();
testQ.dequeue();
testQ.enqueue(-9);
testQ.dequeue();
testQ.dequeue();
testQ.dequeue();
testQ.dequeue();
testQ.enqueue(-8);
```

Size: 3



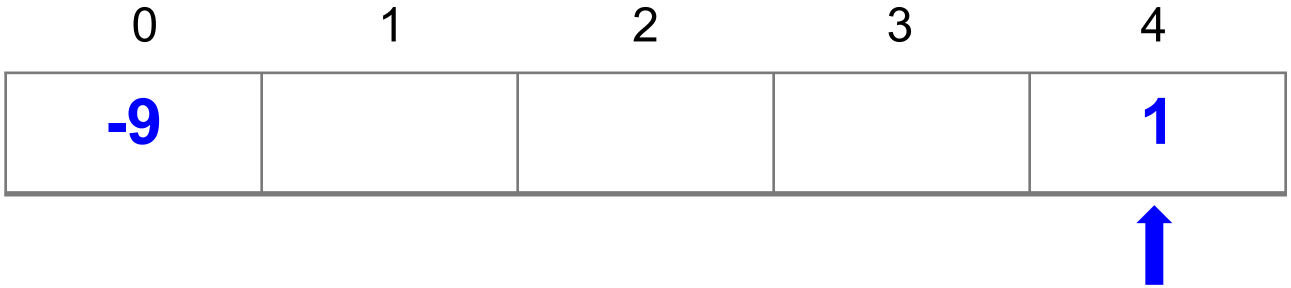
0	1	2	3	4
-9			2	1

Arrow is front

EXAMPLE

```
   Queue<Integer> testQ = new ArrayQueue<Integer>(5);
   testQ.enqueue(3);
   testQ.enqueue(4);
   testQ.enqueue(5);
   testQ.enqueue(2);
   testQ.enqueue(1);
   testQ.dequeue();
   testQ.dequeue();
   testQ.enqueue(-9);
   testQ.dequeue();
➡  testQ.dequeue();
   testQ.dequeue();
   testQ.dequeue();
   testQ.enqueue(-8);
```

Size: 2



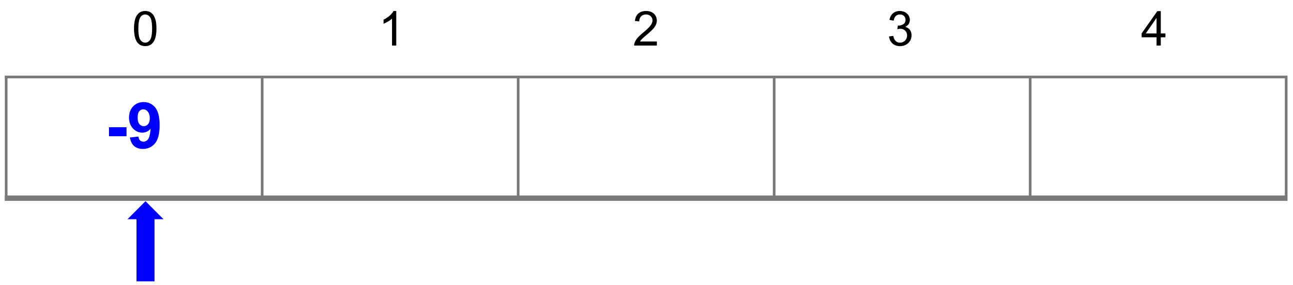
0	1	2	3	4
-9				1
				⬆

Arrow is front

EXAMPLE

```
Queue<Integer> testQ = new ArrayQueue<Integer>(5);
testQ.enqueue(3);
testQ.enqueue(4);
testQ.enqueue(5);
testQ.enqueue(2);
testQ.enqueue(1);
testQ.dequeue();
testQ.dequeue();
testQ.enqueue(-9);
testQ.dequeue();
testQ.dequeue();
testQ.dequeue();
testQ.dequeue();
testQ.enqueue(-8);
```

Size: 1



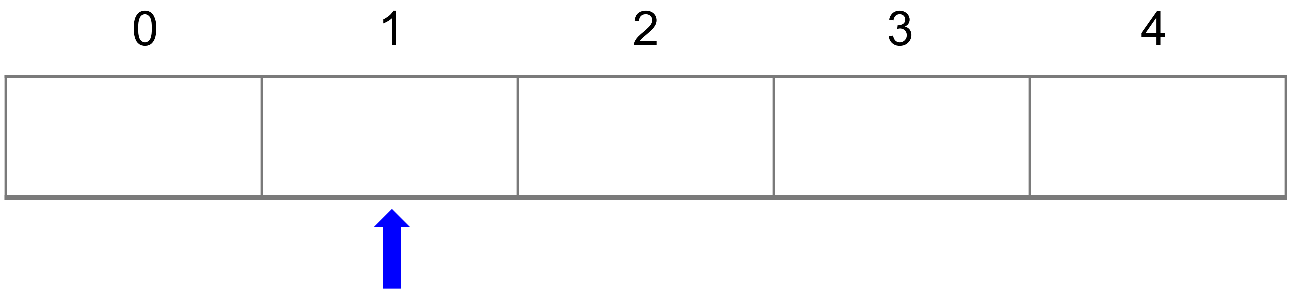
0	1	2	3	4
-9				

Arrow is front

EXAMPLE

```
Queue<Integer> testQ = new ArrayQueue<Integer>(5);
testQ.enqueue(3);
testQ.enqueue(4);
testQ.enqueue(5);
testQ.enqueue(2);
testQ.enqueue(1);
testQ.dequeue();
testQ.dequeue();
testQ.enqueue(-9);
testQ.dequeue();
testQ.dequeue();
testQ.dequeue();
testQ.dequeue();
testQ.enqueue(-8);
```

Size: 0

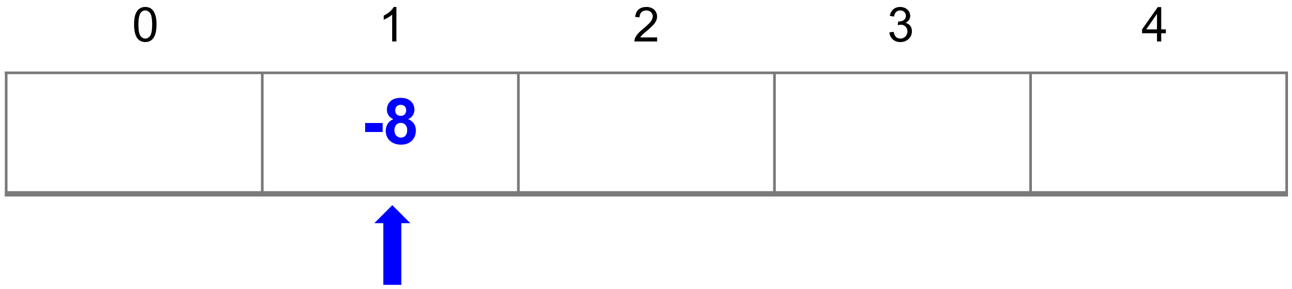


Arrow is front

EXAMPLE

```
Queue<Integer> testQ = new ArrayQueue<Integer>(5);
testQ.enqueue(3);
testQ.enqueue(4);
testQ.enqueue(5);
testQ.enqueue(2);
testQ.enqueue(1);
testQ.dequeue();
testQ.dequeue();
testQ.enqueue(-9);
testQ.dequeue();
testQ.dequeue();
testQ.dequeue();
testQ.dequeue();
testQ.enqueue(-8);
```

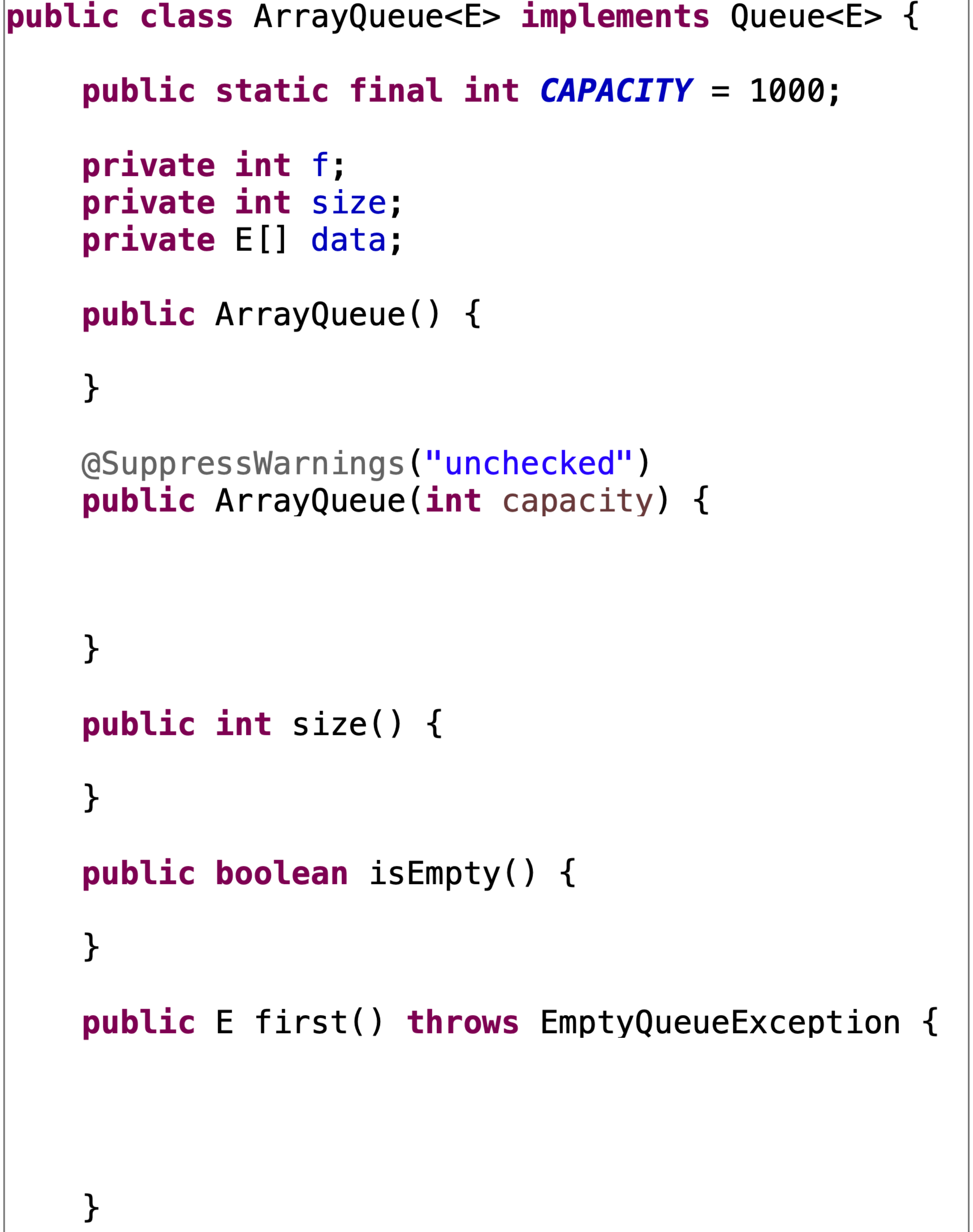
Size: 1



0	1	2	3	4
	-8			

Arrow is front

QUEUE WITH CIRCULAR ARRAY



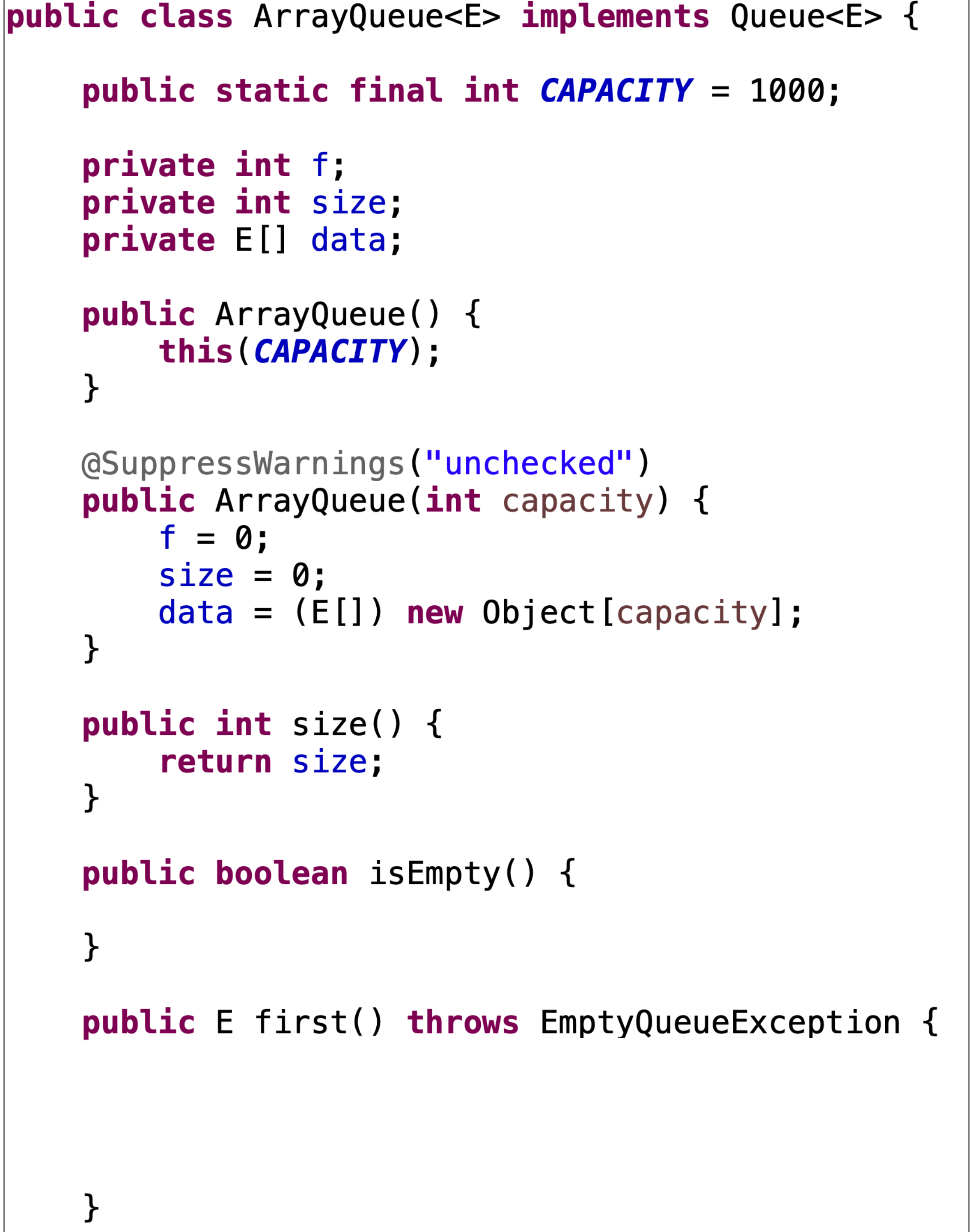
```java
public class ArrayQueue<E> implements Queue<E> {

    public static final int CAPACITY = 1000;

    private int f;
    private int size;
    private E[] data;

    public ArrayQueue() {

    }

    @SuppressWarnings("unchecked")
    public ArrayQueue(int capacity) {



    }

    public int size() {

    }

    public boolean isEmpty() {

    }

    public E first() throws EmptyQueueException {




    }
```

QUEUE WITH CIRCULAR ARRAY

Two constructors (allow user to select capacity, or use default.

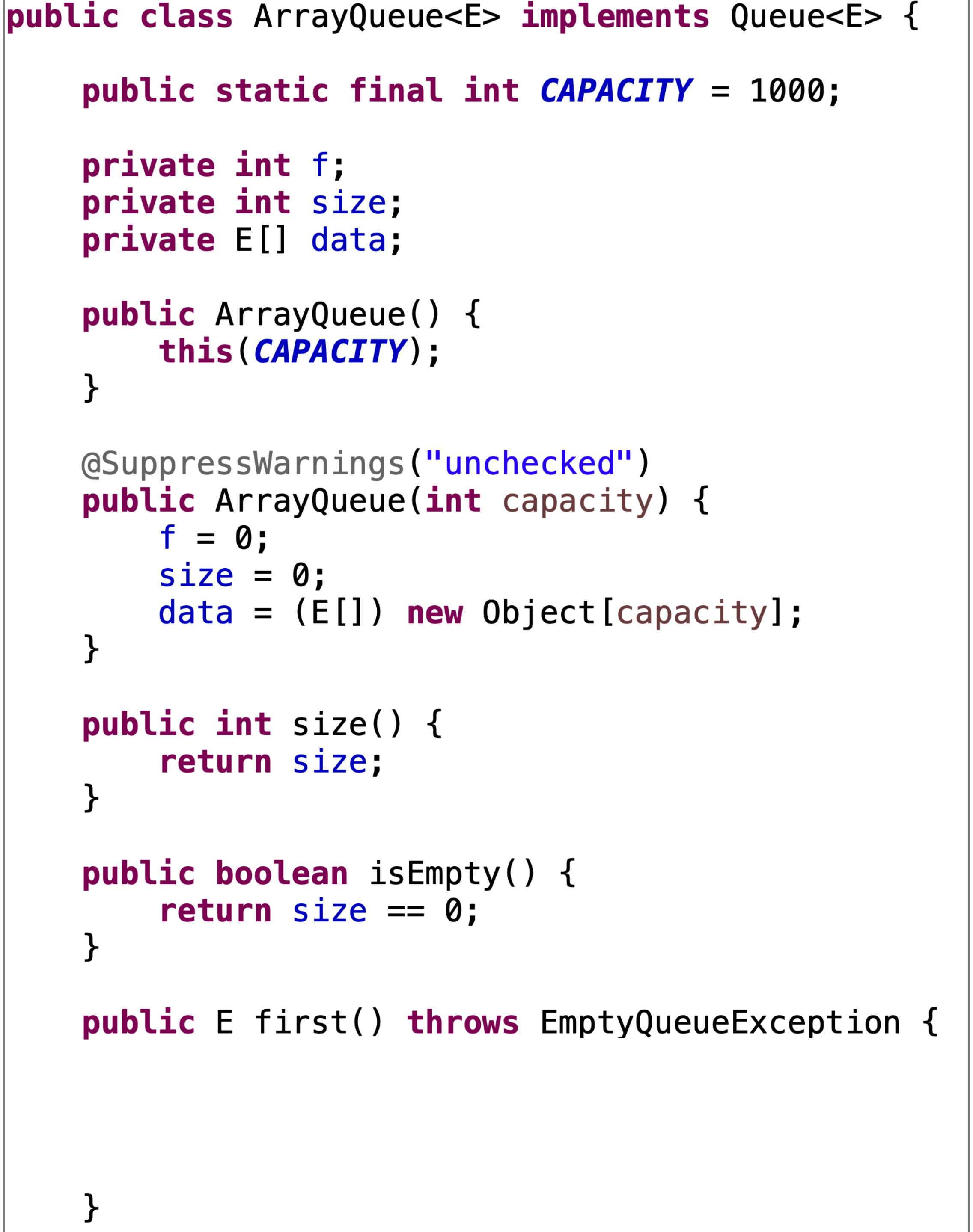
```
public class ArrayQueue<E> implements Queue<E> {

    public static final int CAPACITY = 1000;

    private int f;
    private int size;
    private E[] data;

    public ArrayQueue() {
        this(CAPACITY);
    }

    @SuppressWarnings("unchecked")
    public ArrayQueue(int capacity) {
        f = 0;
        size = 0;
        data = (E[]) new Object[capacity];
    }

    public int size() {

    }

    public boolean isEmpty() {

    }

    public E first() throws EmptyQueueException {



    }
```

QUEUE WITH CIRCULAR ARRAY



```
public class ArrayQueue<E> implements Queue<E> {

    public static final int CAPACITY = 1000;

    private int f;
    private int size;
    private E[] data;

    public ArrayQueue() {
        this(CAPACITY);
    }

    @SuppressWarnings("unchecked")
    public ArrayQueue(int capacity) {
        f = 0;
        size = 0;
        data = (E[]) new Object[capacity];
    }

    public int size() {
        return size;
    }

    public boolean isEmpty() {

    }

    public E first() throws EmptyQueueException {




    }
```

QUEUE WITH CIRCULAR ARRAY



```java
public class ArrayQueue<E> implements Queue<E> {

    public static final int CAPACITY = 1000;

    private int f;
    private int size;
    private E[] data;

    public ArrayQueue() {
        this(CAPACITY);
    }

    @SuppressWarnings("unchecked")
    public ArrayQueue(int capacity) {
        f = 0;
        size = 0;
        data = (E[]) new Object[capacity];
    }

    public int size() {
        return size;
    }

    public boolean isEmpty() {
        return size == 0;
    }

    public E first() throws EmptyQueueException {




    }
```

QUEUE WITH CIRCULAR ARRAY

```
public class ArrayQueue<E> implements Queue<E> {

    public static final int CAPACITY = 1000;

    private int f;
    private int size;
    private E[] data;

    public ArrayQueue() {
        this(CAPACITY);
    }

    @SuppressWarnings("unchecked")
    public ArrayQueue(int capacity) {
        f = 0;
        size = 0;
        data = (E[]) new Object[capacity];
    }

    public int size() {
        return size;
    }

    public boolean isEmpty() {
        return size == 0;
    }

    public E first() throws EmptyQueueException {
        if (isEmpty()) {
            throw new EmptyStackException();
        }
        return data[f];
    }
```

QUEUE WITH CIRCULAR ARRAY

```java
public void enqueue(E element) throws FullQueueException {







}

public E dequeue() throws EmptyQueueException {







}
```

QUEUE WITH CIRCULAR ARRAY

```
public void enqueue(E element) throws FullQueueException {
    if (size == data.length) {
        throw new FullQueueException();
    }







}

public E dequeue() throws EmptyQueueException {
    if (isEmpty()) {
        throw new EmptyStackException();
    }







}
```

QUEUE WITH CIRCULAR ARRAY

```java
public void enqueue(E element) throws FullQueueException {
    if (size == data.length) {
        throw new FullQueueException();
    }

    int end = (f + size) % data.length;
    data[end] = element;

    size += 1;

}

public E dequeue() throws EmptyQueueException {
    if (isEmpty()) {
        throw new EmptyStackException();
    }






}
```

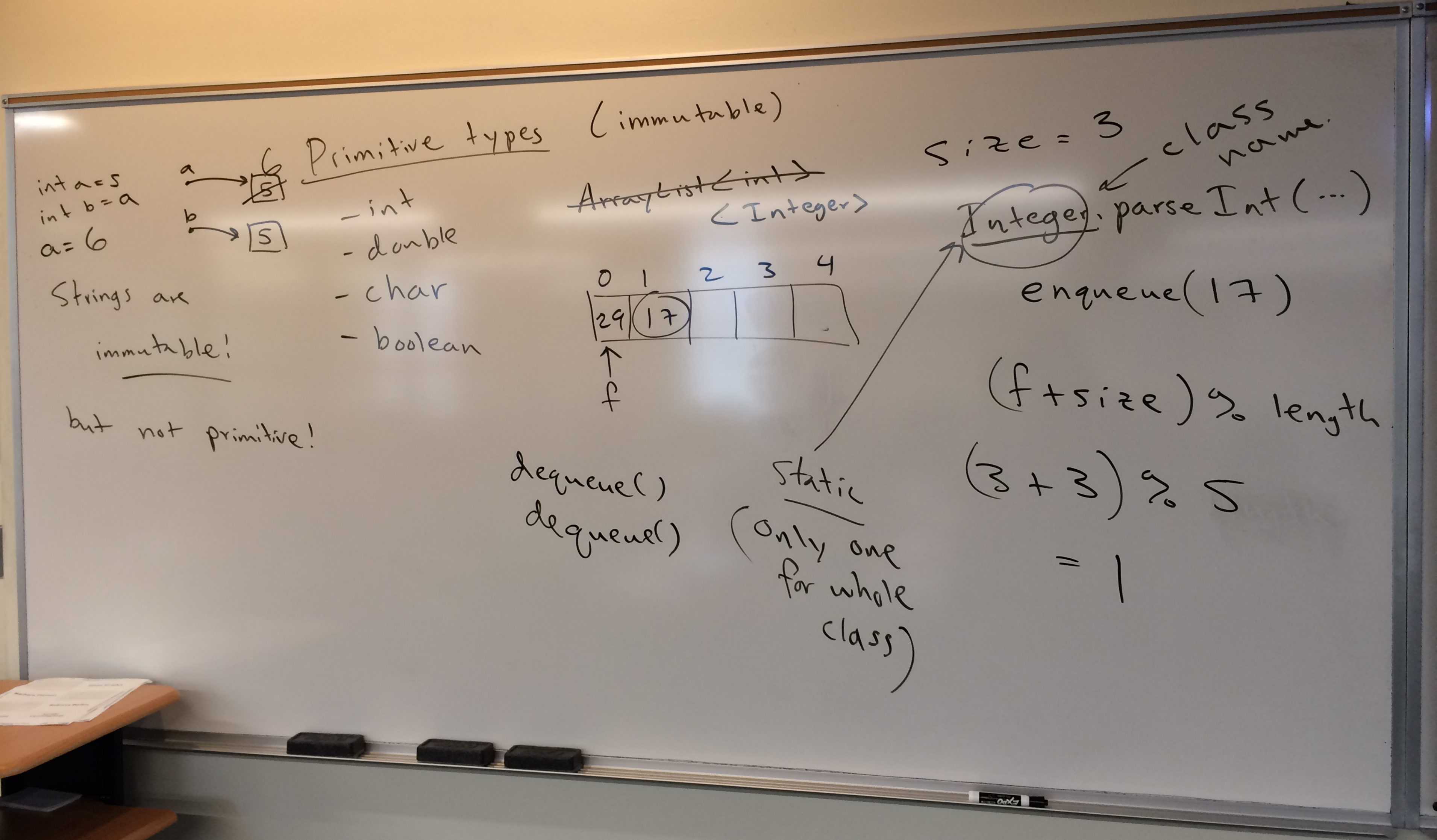
QUEUE WITH CIRCULAR ARRAY

```java
public void enqueue(E element) throws FullQueueException {
    if (size == data.length) {
        throw new FullQueueException();
    }

    int end = (f + size) % data.length;
    data[end] = element;

    size += 1;

}

public E dequeue() throws EmptyQueueException {
    if (isEmpty()) {
        throw new EmptyStackException();
    }
    E result = data[f];
    data[f] = null; // optional
    f = (f + 1) % data.length;

    size -= 1;
    return result;
}
```

DESIGNING DATA STRUCTURES

1. Make a Course object that can store a name and list of students. Include relevant constructors, getters, and setters.
2. Make a LimitedEnrollmentCourse that has a cap on the number of students who can enroll. Have it inherit from Course.
3. Make addStudent, removeStudent, and getEnrolled methods that correctly handle limited versus unlimited enrollment.

Extra practice!



int a=5
int b=a
a=6

a → ~~5~~ 6

b → 5

Strings are immutable!

but not primitive!

Primitive types (immutable)

- int
- double
- char
- boolean

~~ArrayList<int>~~
<Integer>

0	1	2	3	4
29	17			

↑
f

dequeue()
dequeue()

Static
(Only one for whole class)

size = 3

class name

Integer.parseInt(...)

enqueue(17)

(f+size) % length

(3+3) % 5

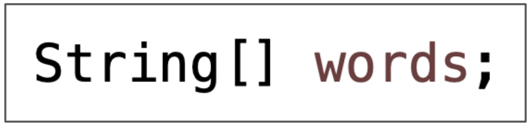
= 1

MAR 3 OUTLINE

- Queues (theory and implementation)
- Review arrays and ArrayLists
- Review Nodes and Linked Lists
- Practice Problems

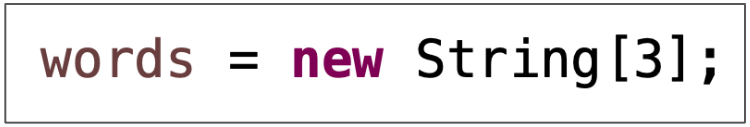
ARRAYS

- **Fixed length**
  - Pro: all operations O(1)
  - Con: cannot resize or move around elements easily
- **Declare**



```
String[] words;
```

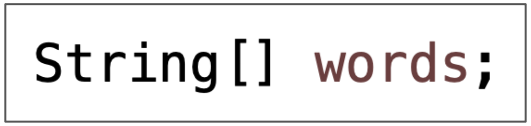
- **Initialize (allocate)**



```
words = new String[3];
```

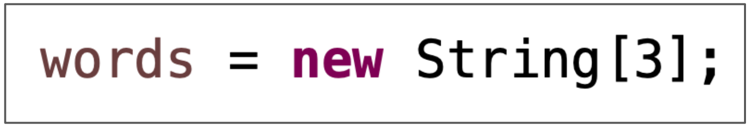
ARRAYS

- **Fixed length**
  - Pro: all operations O(1)
  - Con: cannot resize or move around elements easily
- **Declare**



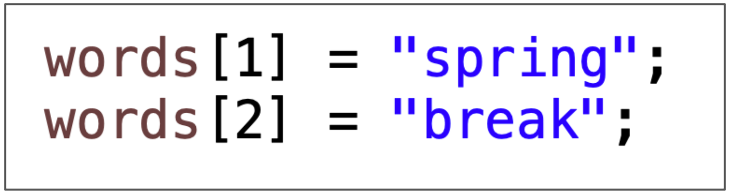
```
String[] words;
```

- **Initialize (allocate)**



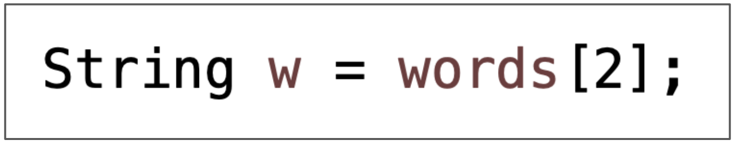
```
words = new String[3];
```

- **“set”:**



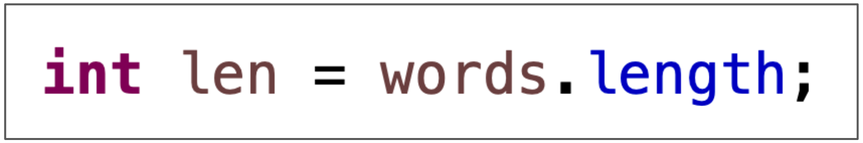
```
words[1] = "spring";
words[2] = "break";
```

- **“get”:**



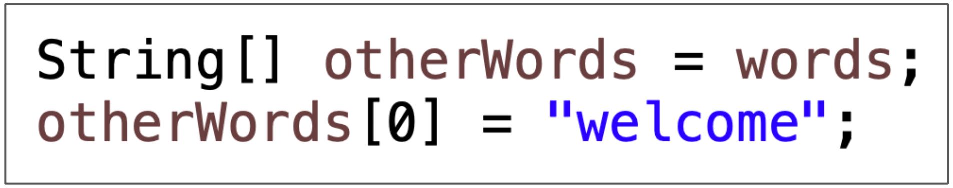
```
String w = words[2];
```

- **length**



```
int len = words.length;
```

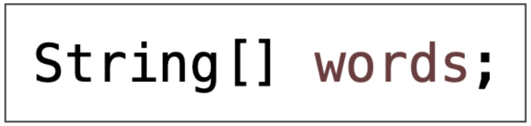
ARRAYS



```
String[] otherWords = words;
otherWords[0] = "welcome";
```

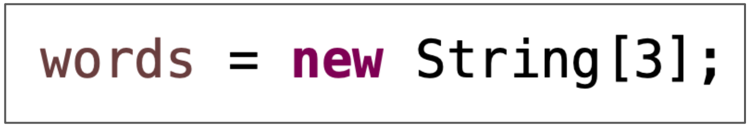
Q: what is happening here??

- **Fixed length**
  - Pro: all operations O(1)
  - Con: cannot resize or move around elements easily
- **Declare**



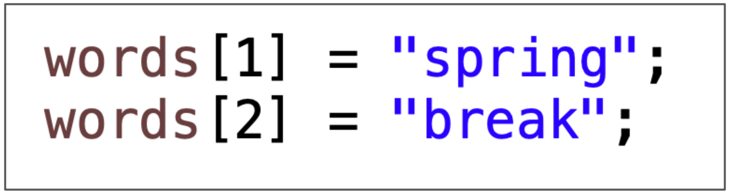
```
String[] words;
```

- **Initialize (allocate)**



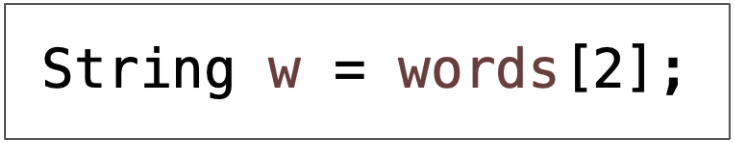
```
words = new String[3];
```

- **“set”:**



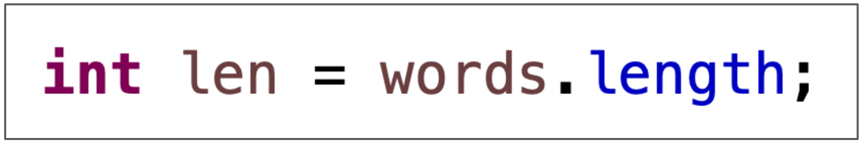
```
words[1] = "spring";
words[2] = "break";
```

- **“get”:**



```
String w = words[2];
```

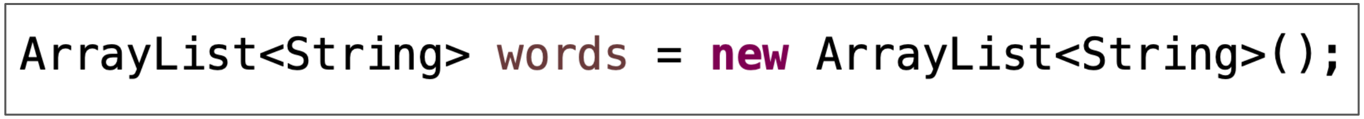
- **length**



```
int len = words.length;
```

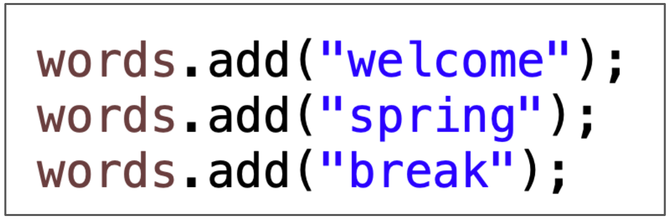
ARRAY LISTS

- We allow the size to change, but we don’t copy over elements every time a new element doesn’t fit
- Use the idea of doubling the size of the array to get an average creation time of O(n) for length n array
- **Declare/Initialize**



```
ArrayList<String> words = new ArrayList<String>();
```

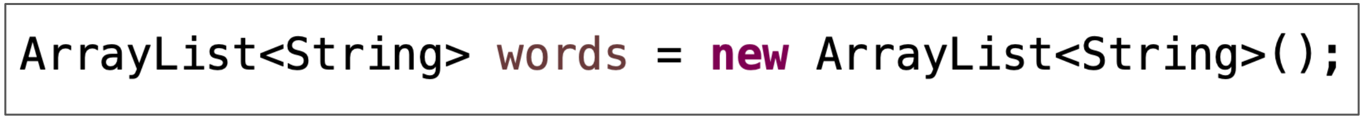
- **add:**



```
words.add("welcome");
words.add("spring");
words.add("break");
```

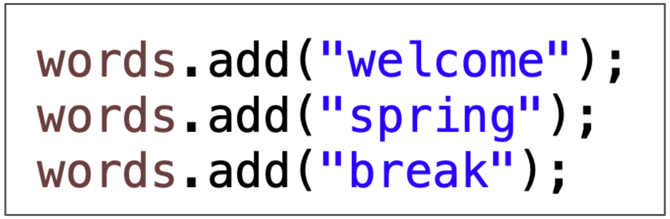
ARRAY LISTS

- We allow the size to change, but we don’t copy over elements every time a new element doesn’t fit
- Use the idea of doubling the size of the array to get an average creation time of O(n) for length n array
- **Declare/Initialize**



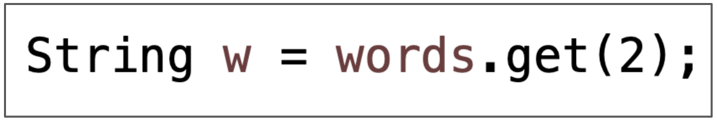
```java
ArrayList<String> words = new ArrayList<String>();
```

- **add:**



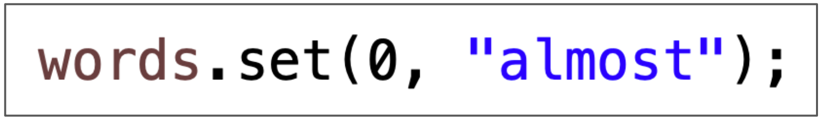
```java
words.add("welcome");
words.add("spring");
words.add("break");
```

- **get:**



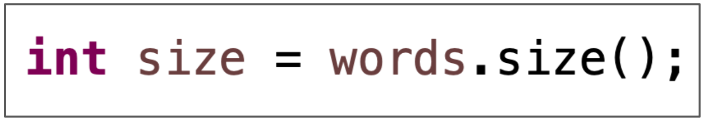
```java
String w = words.get(2);
```

- **set:**



```java
words.set(0, "almost");
```

- **size:**



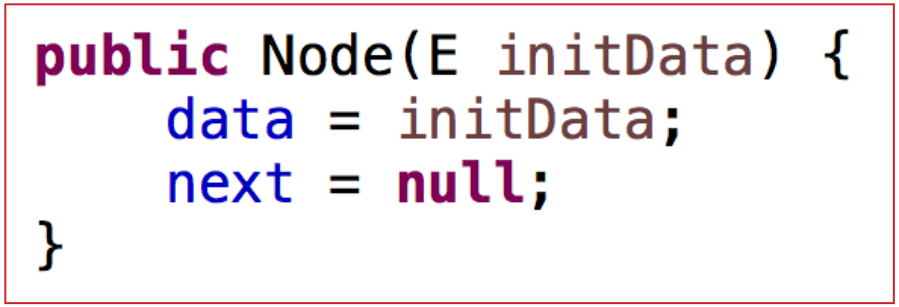
```java
int size = words.size();
```

MAR 3 OUTLINE

- Queues (theory and implementation)
- Review arrays and ArrayLists
- **Review Nodes and Linked Lists**
- Practice Problems

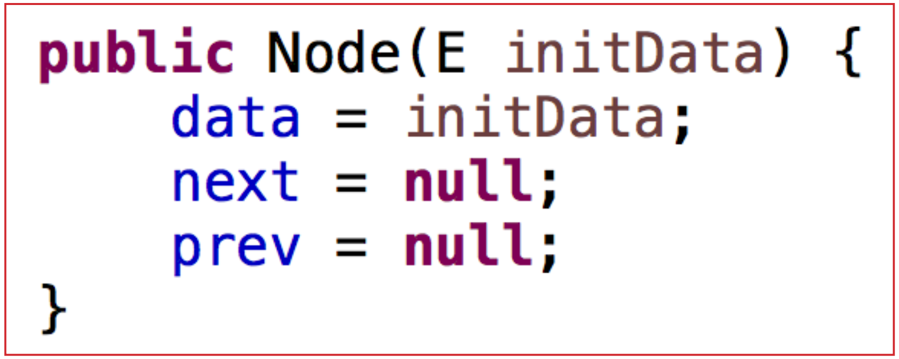
NODE OBJECTS

Match the constructor to the type of list that would contain such Nodes



```
public Node(E initData) {
    data = initData;
    next = null;
}
```

- Doubly linked list

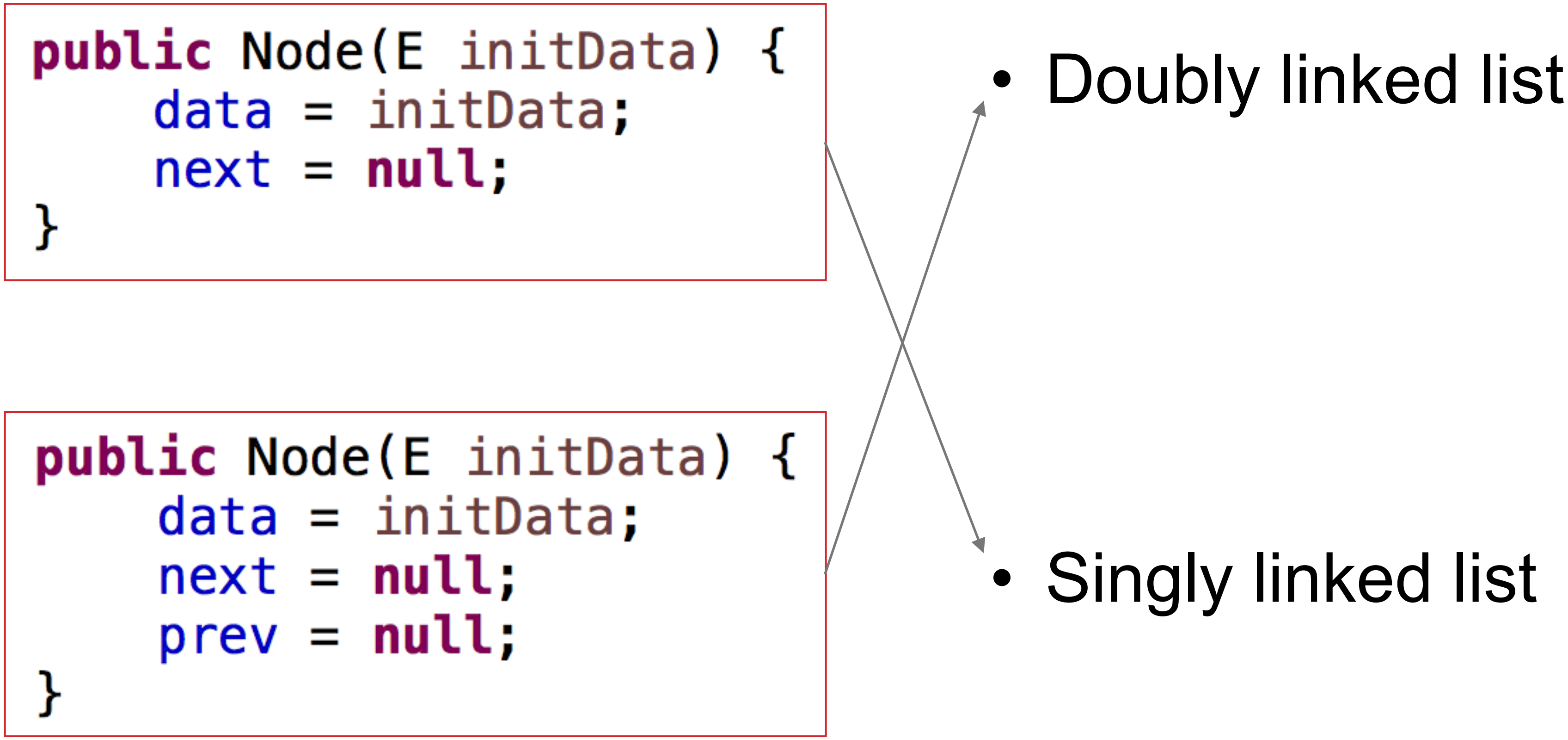


```
public Node(E initData) {
    data = initData;
    next = null;
    prev = null;
}
```

- Singly linked list

NODE OBJECTS

Match the constructor to the type of list that would contain such Nodes



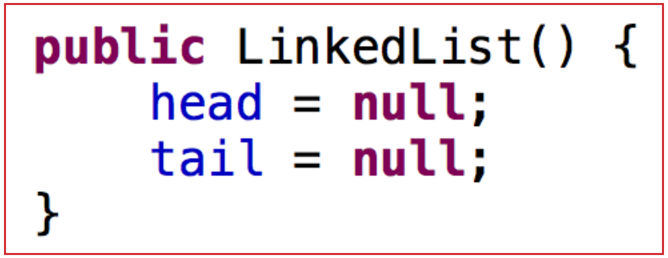
```java
public Node(E initData) {
    data = initData;
    next = null;
}
```

```java
public Node(E initData) {
    data = initData;
    next = null;
    prev = null;
}
```

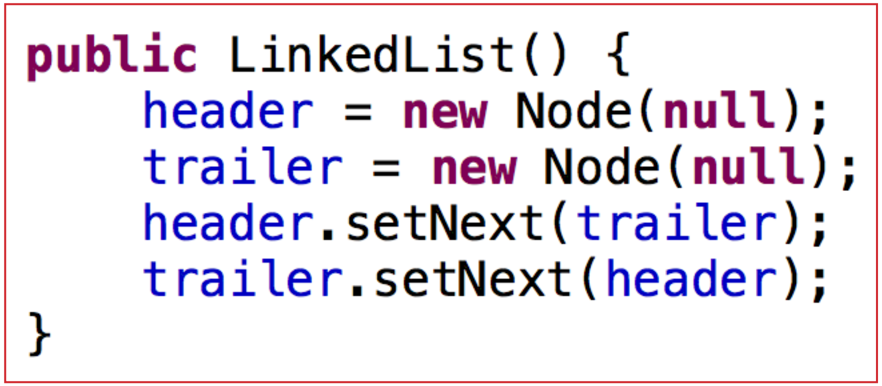
- Doubly linked list
- Singly linked list

LINKED LISTS

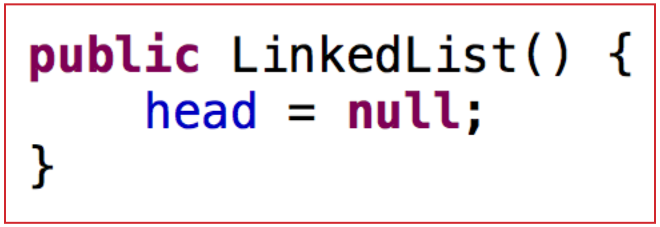
Match the constructor to the appropriate type(s) of list.



```java
public LinkedList() {
    head = null;
    tail = null;
}
```



```java
public LinkedList() {
    header = new Node(null);
    trailer = new Node(null);
    header.setNext(trailer);
    trailer.setNext(header);
}
```

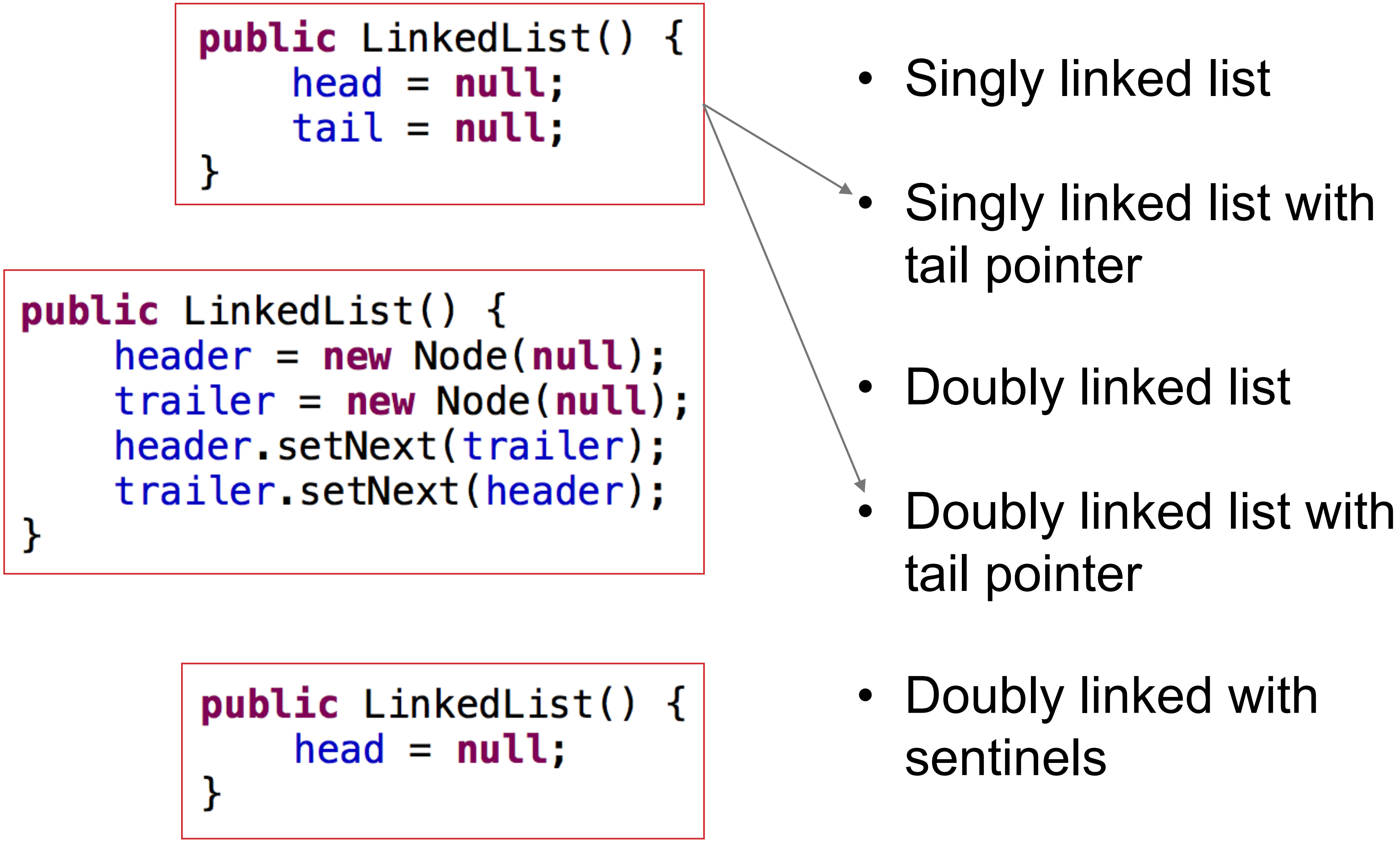


```java
public LinkedList() {
    head = null;
}
```

- Singly linked list
- Singly linked list with tail pointer
- Doubly linked list
- Doubly linked list with tail pointer
- Doubly linked with sentinels

LINKED LISTS

Match the constructor to the appropriate type(s) of list.



```java
public LinkedList() {
    head = null;
    tail = null;
}
```

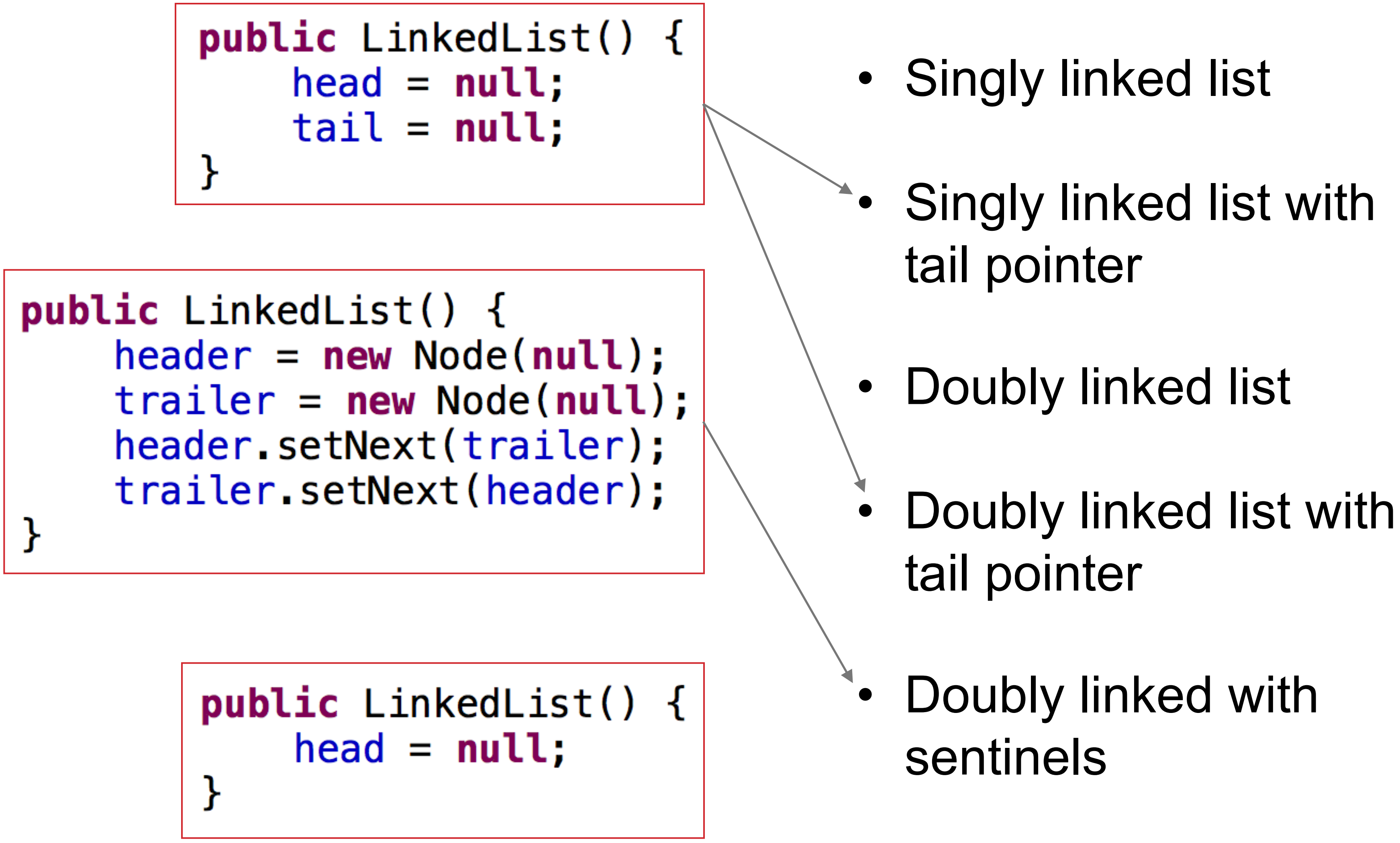
```java
public LinkedList() {
    header = new Node(null);
    trailer = new Node(null);
    header.setNext(trailer);
    trailer.setNext(header);
}
```

```java
public LinkedList() {
    head = null;
}
```

- Singly linked list
- Singly linked list with tail pointer
- Doubly linked list
- Doubly linked list with tail pointer
- Doubly linked with sentinels

LINKED LISTS

Match the constructor to the appropriate type(s) of list.



```java
public LinkedList() {
    head = null;
    tail = null;
}
```

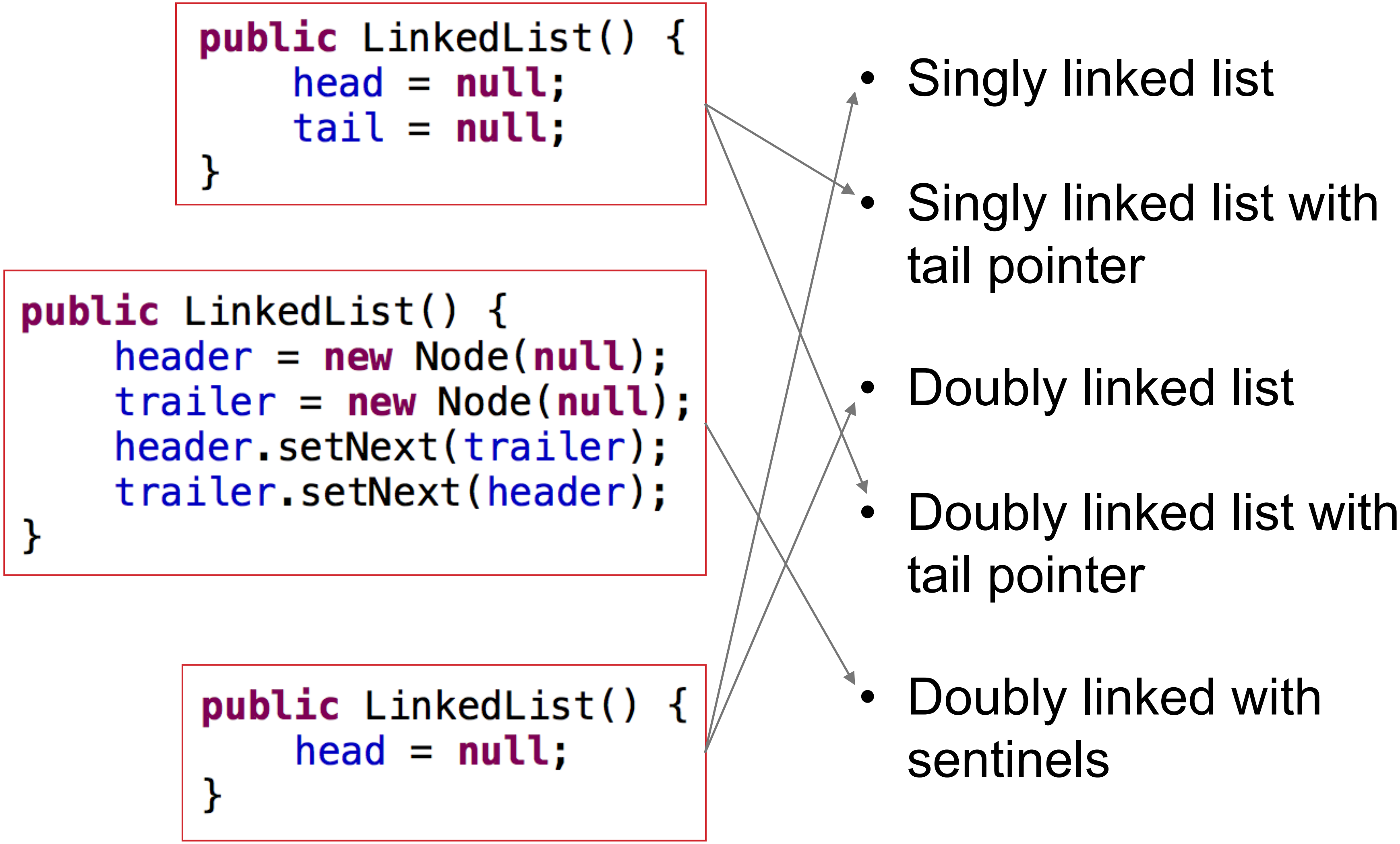
```java
public LinkedList() {
    header = new Node(null);
    trailer = new Node(null);
    header.setNext(trailer);
    trailer.setNext(header);
}
```

```java
public LinkedList() {
    head = null;
}
```

- Singly linked list
- Singly linked list with tail pointer
- Doubly linked list
- Doubly linked list with tail pointer
- Doubly linked with sentinels

LINKED LISTS

Match the constructor to the appropriate type(s) of list.



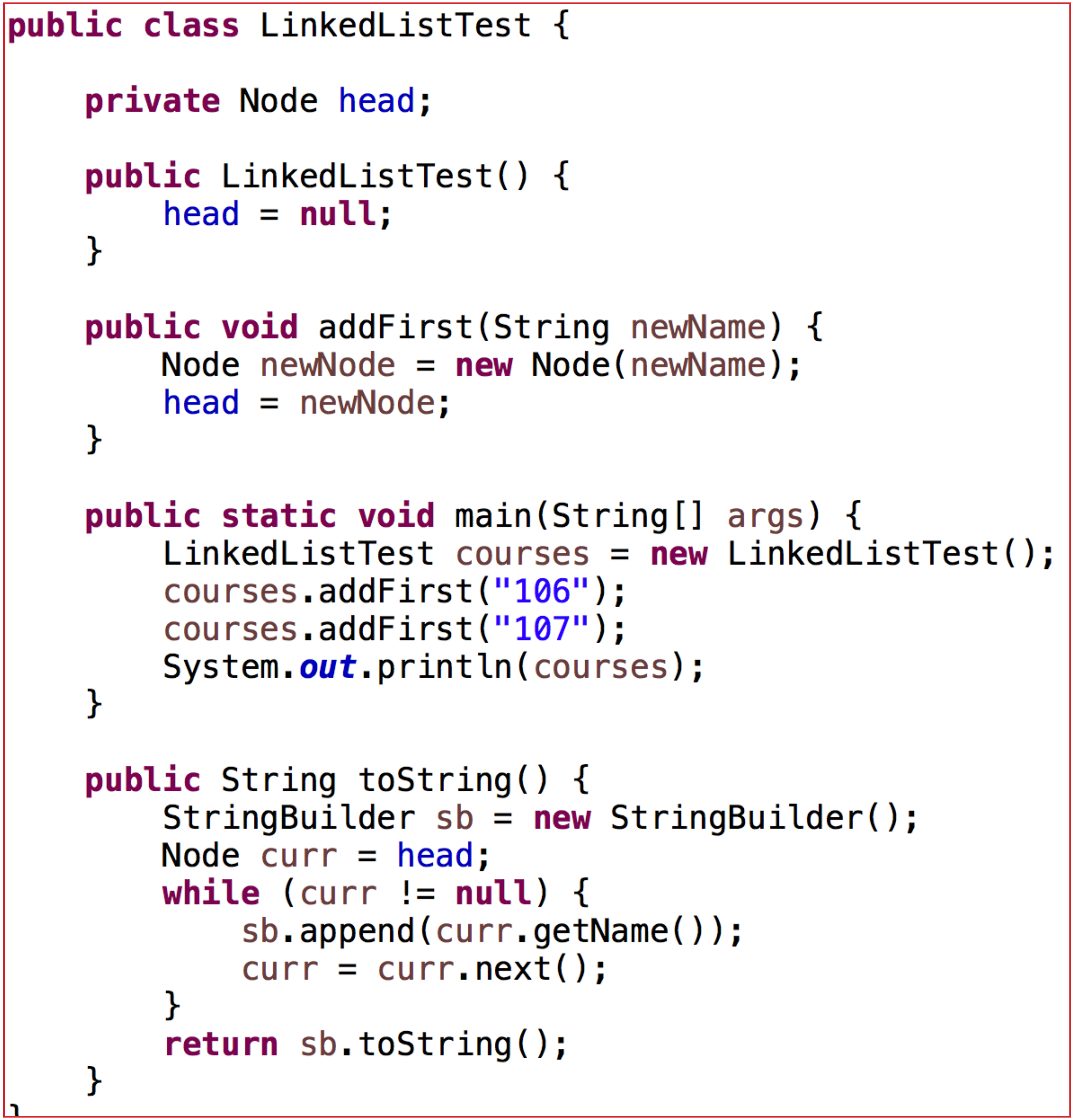
```
public LinkedList() {
    head = null;
    tail = null;
}
```

```
public LinkedList() {
    header = new Node(null);
    trailer = new Node(null);
    header.setNext(trailer);
    trailer.setNext(header);
}
```

```
public LinkedList() {
    head = null;
}
```

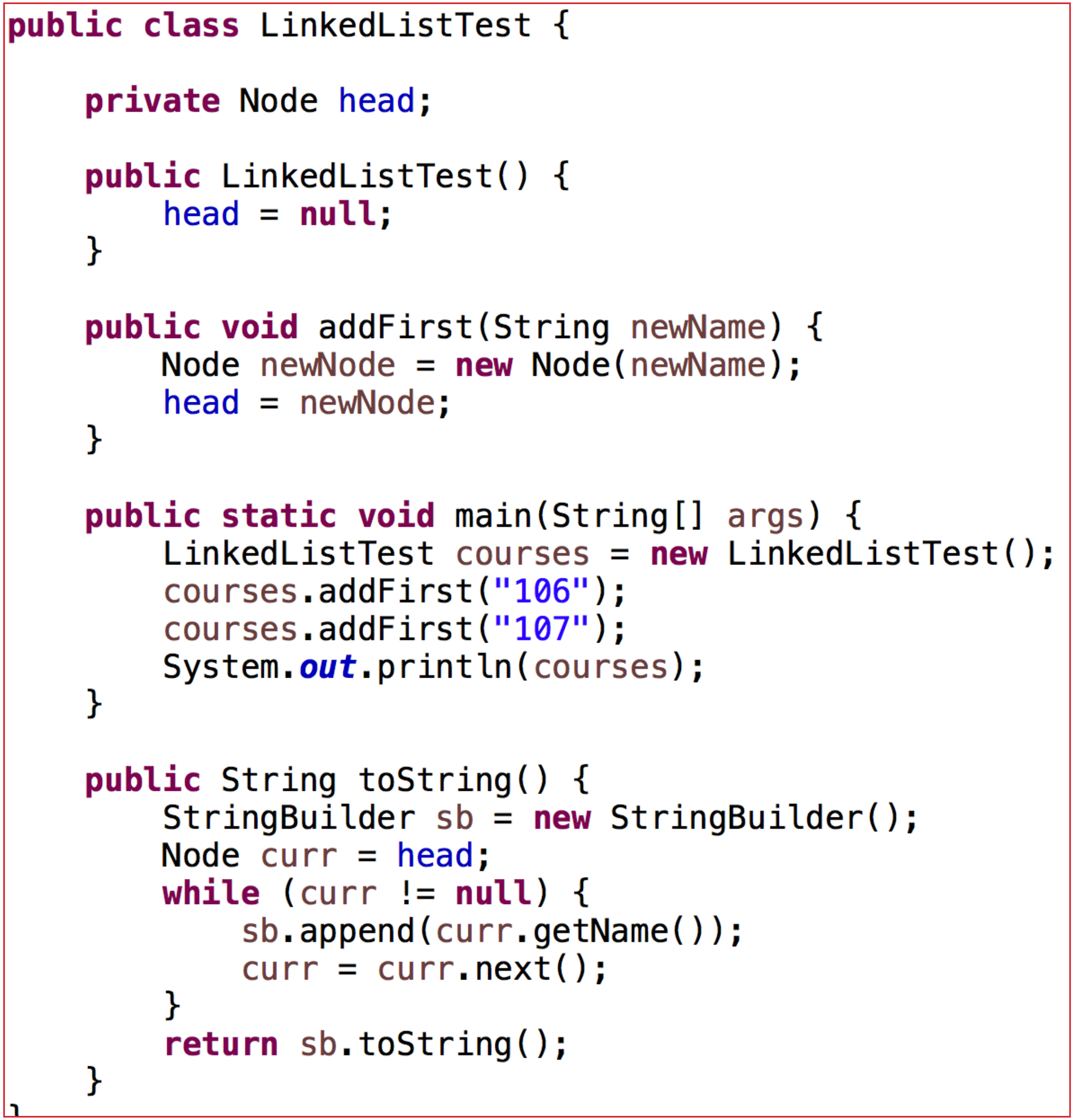
- Singly linked list
- Singly linked list with tail pointer
- Doubly linked list
- Doubly linked list with tail pointer
- Doubly linked with sentinels

What is the issue with the following code? What is printed?



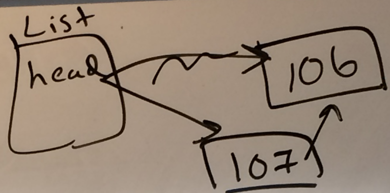
```
public class LinkedListTest {

    private Node head;

    public LinkedListTest() {
        head = null;
    }

    public void addFirst(String newName) {
        Node newNode = new Node(newName);
        head = newNode;
    }

    public static void main(String[] args) {
        LinkedListTest courses = new LinkedListTest();
        courses.addFirst("106");
        courses.addFirst("107");
        System.out.println(courses);
    }

    public String toString() {
        StringBuilder sb = new StringBuilder();
        Node curr = head;
        while (curr != null) {
            sb.append(curr.getName());
            curr = curr.next();
        }
        return sb.toString();
    }
}
```

What is the issue with the following code? What is printed?



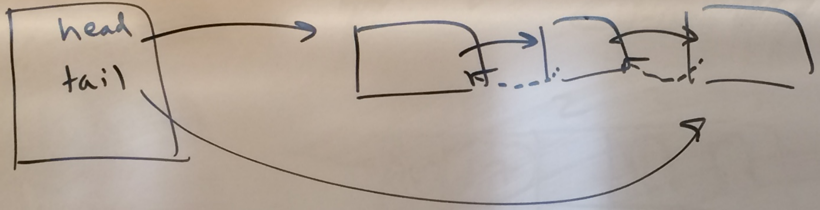
```java
public class LinkedListTest {

    private Node head;

    public LinkedListTest() {
        head = null;
    }

    public void addFirst(String newName) {
        Node newNode = new Node(newName);
        head = newNode;
    }

    public static void main(String[] args) {
        LinkedListTest courses = new LinkedListTest();
        courses.addFirst("106");
        courses.addFirst("107");
        System.out.println(courses);
    }

    public String toString() {
        StringBuilder sb = new StringBuilder();
        Node curr = head;
        while (curr != null) {
            sb.append(curr.getName());
            curr = curr.next();
        }
        return sb.toString();
    }
}
```

Only “107”!



new Node.next = head;

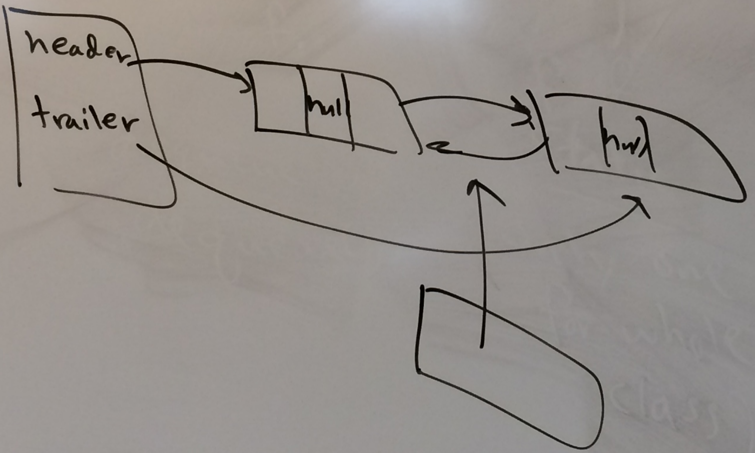
(1)



if size is 0:

after first element inserted.

(2)



Node curr = header.next();

MAR 3 OUTLINE

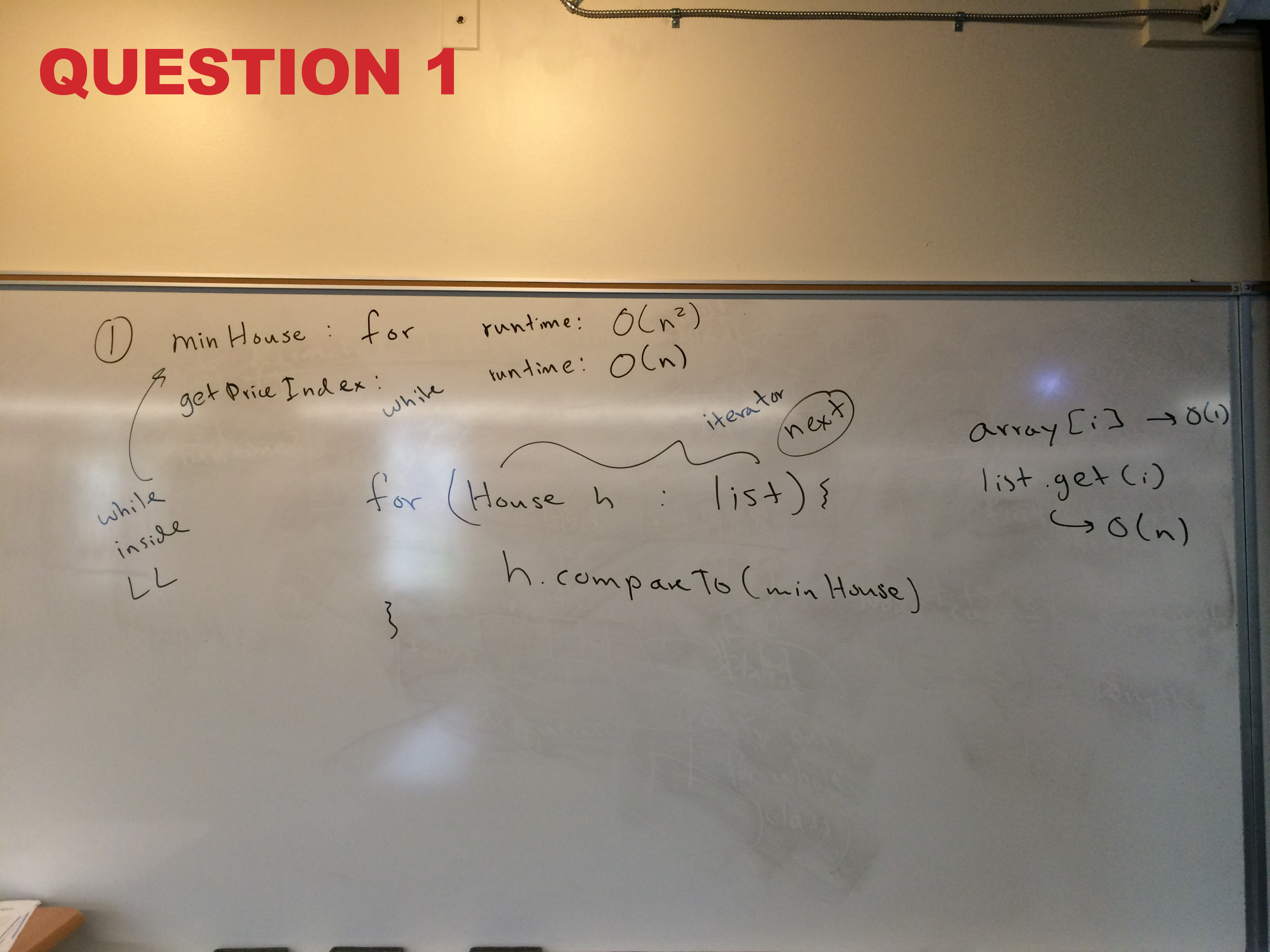
- Queues (theory and implementation)
- Review arrays and ArrayLists
- Review Nodes and Linked Lists
- **Practice Problems**

WORK WITH A PARTNER!

- Question 1: focus on understanding the code and thinking about what type of loops to use (don’t rewrite the code now)

- Question 2: skip (about Queues)

QUESTION 1



① minHouse : for   runtime: $O(n^2)$

getPriceIndex : while   runtime: $O(n)$

while inside LL

iterator

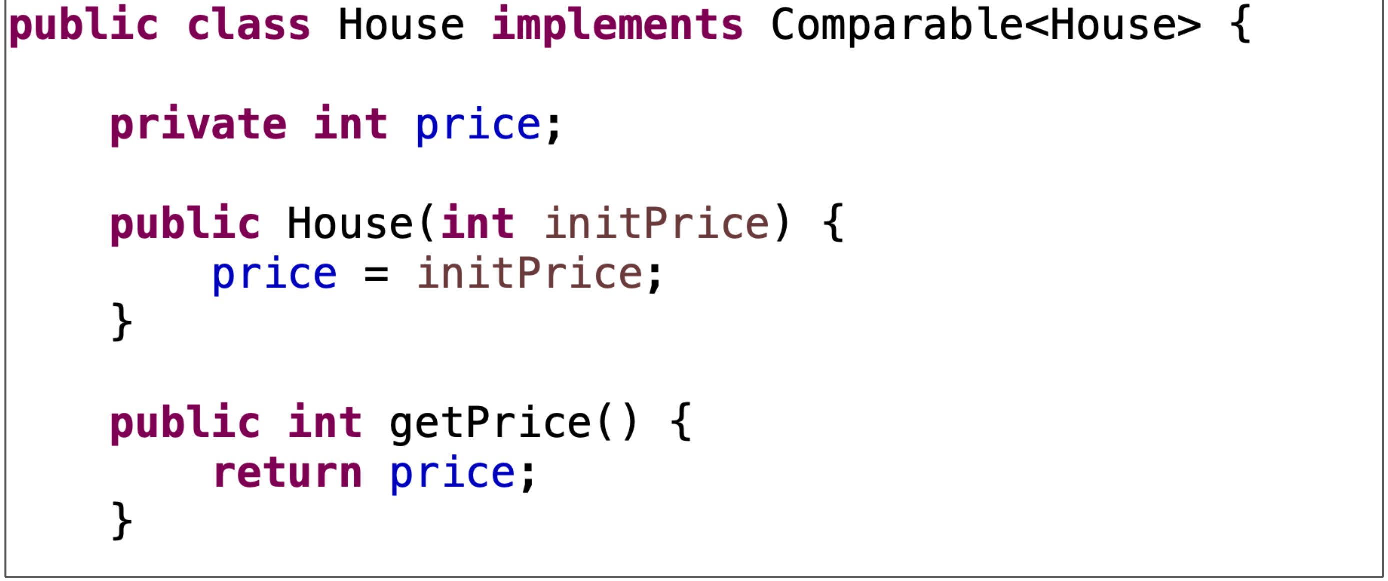
next

```
for (House h : list) {
    h.compareTo(minHouse)
}
```

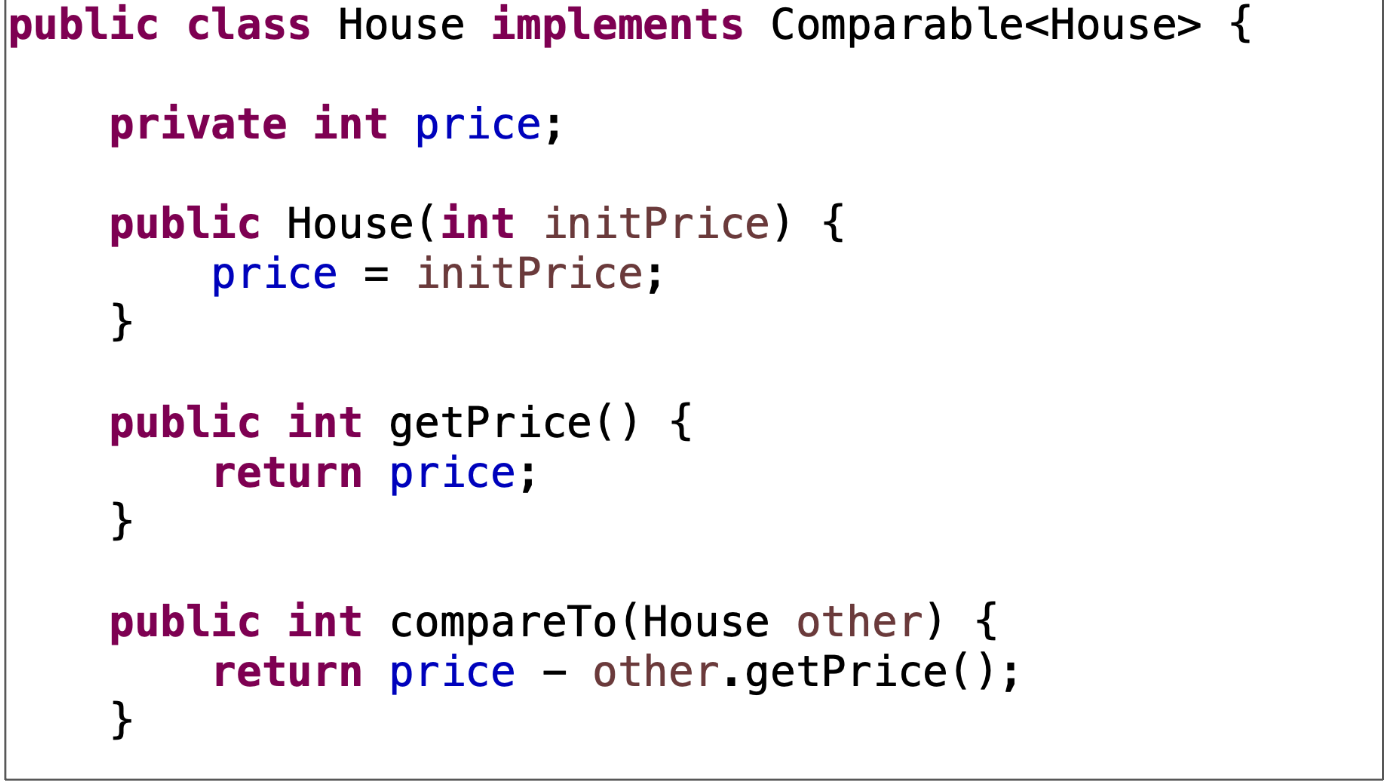
array[i] → $O(1)$

list.get(i)

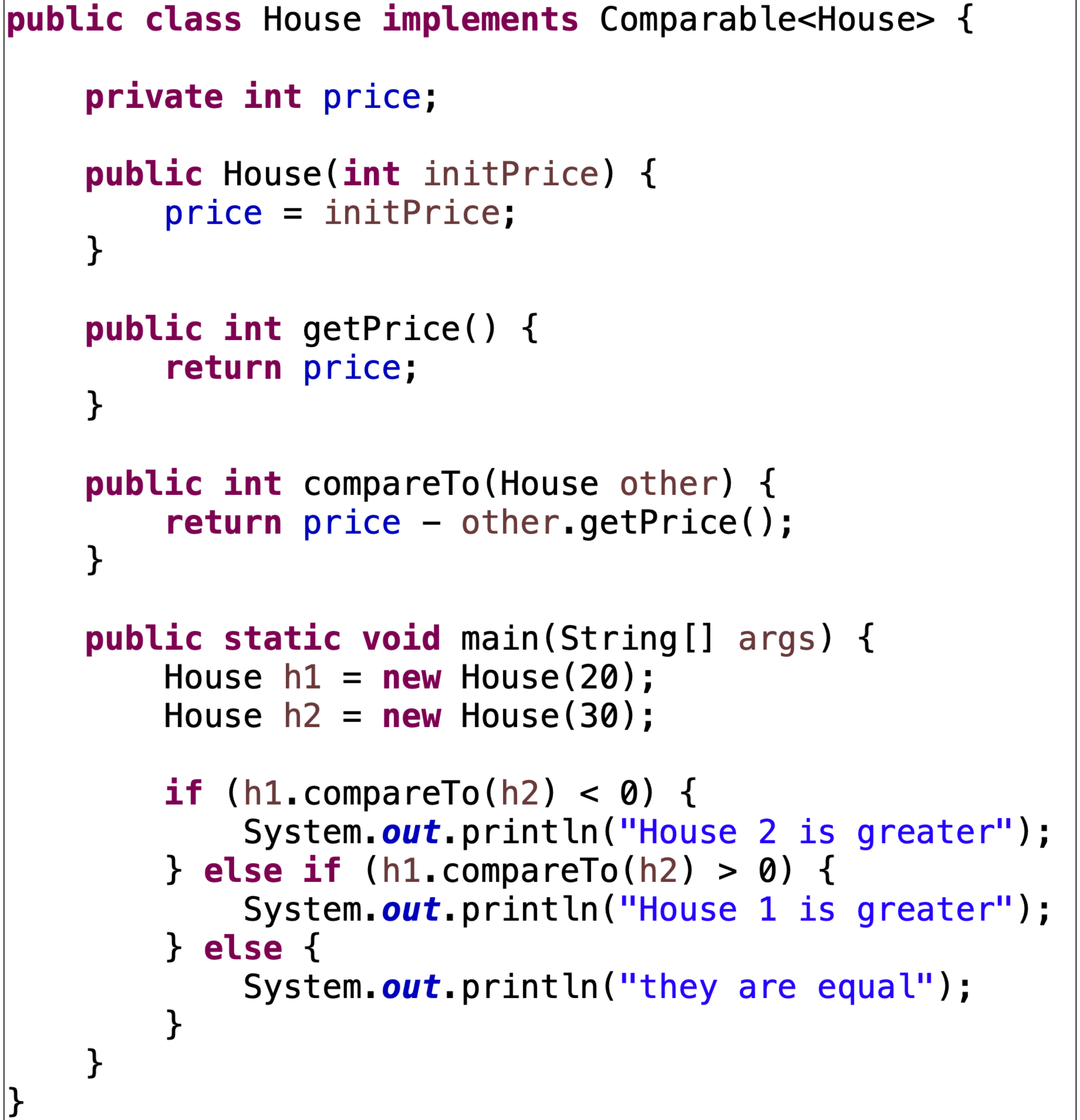
↪ $O(n)$



```java
public class House implements Comparable<House> {

    private int price;

    public House(int initPrice) {
        price = initPrice;
    }

    public int getPrice() {
        return price;
    }
```



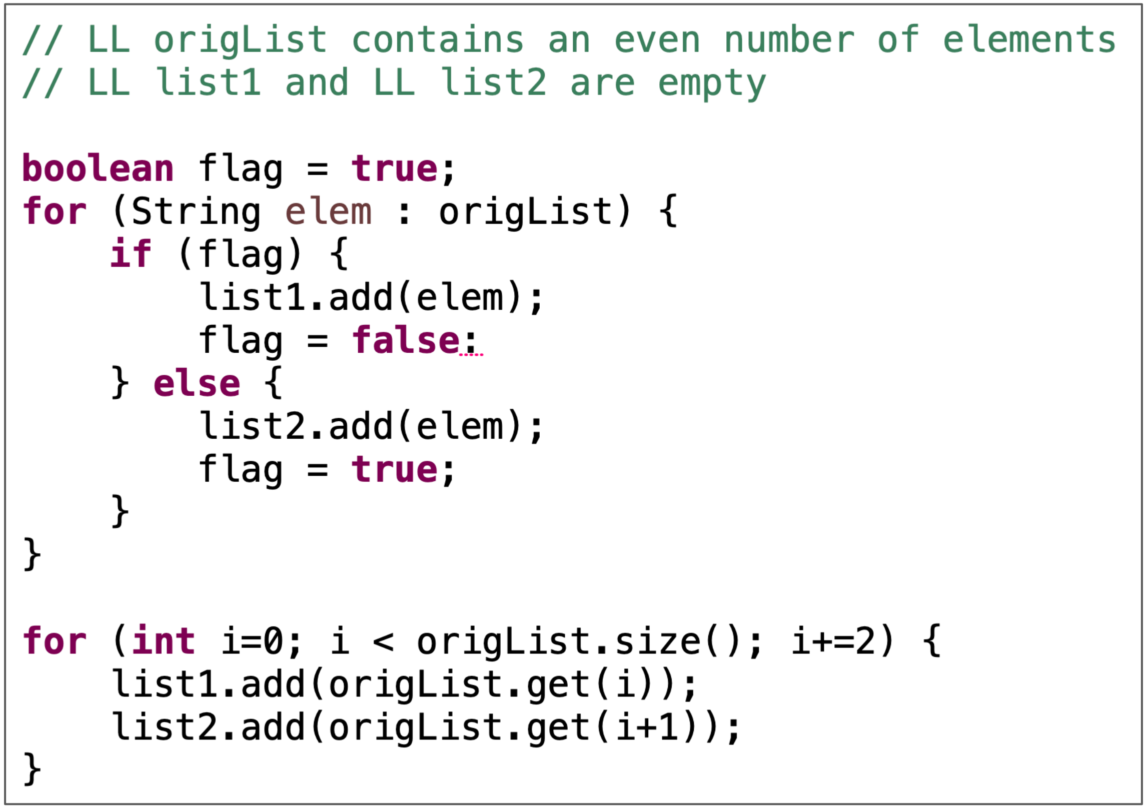
```java
public class House implements Comparable<House> {

    private int price;

    public House(int initPrice) {
        price = initPrice;
    }

    public int getPrice() {
        return price;
    }

    public int compareTo(House other) {
        return price - other.getPrice();
    }
```



```java
public class House implements Comparable<House> {

    private int price;

    public House(int initPrice) {
        price = initPrice;
    }

    public int getPrice() {
        return price;
    }

    public int compareTo(House other) {
        return price - other.getPrice();
    }

    public static void main(String[] args) {
        House h1 = new House(20);
        House h2 = new House(30);

        if (h1.compareTo(h2) < 0) {
            System.out.println("House 2 is greater");
        } else if (h1.compareTo(h2) > 0) {
            System.out.println("House 1 is greater");
        } else {
            System.out.println("they are equal");
        }
    }
}
```

QUESTION 3

There are many ways to do this! Two are shown below. What are the runtimes of these two algorithms?



```java
// LL origList contains an even number of elements
// LL list1 and LL list2 are empty
```

1)

```java
boolean flag = true;
for (String elem : origList) {
    if (flag) {
        list1.add(elem);
        flag = false;
    } else {
        list2.add(elem);
        flag = true;
    }
}
```

2)

```java
for (int i=0; i < origList.size(); i+=2) {
    list1.add(origList.get(i));
    list2.add(origList.get(i+1));
}
```

QUESTION 4

a) get & set (only public methods)

b) Could be a LinkedList or an ArrayList

c) Less efficient than an ArrayList, same efficiency as a LinkedList

d) No! the details are *abstracted* away

Visualization:

Assume that we have stack A with elements 5, 11, -1, 3 (where 3 is on the top). What happens when we call get(2)? Draw the stacks A and B and see what happens.

QUESTION 5

a) Pseudocode:

```
tail = tail.prev
tail.next = null
```

b) Hints: there are 6 pointers that need to be rearranged (2 for A, 2 for B, and head/tail)

d) See notes from Lecture 11