Unit 08: Stacks and Queues

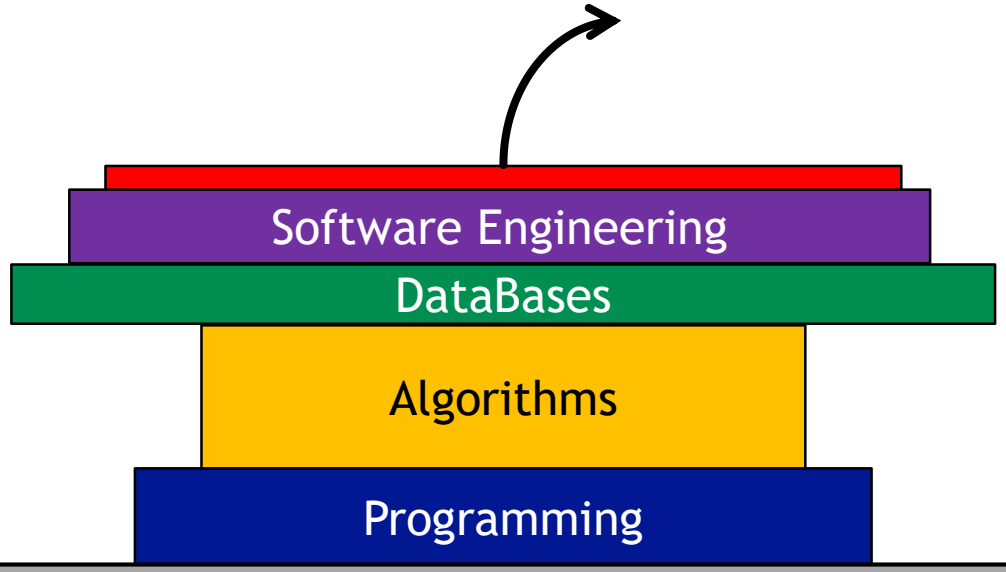
Anthony Estey

CSC 115: Fundamentals of Programming II

University of Victoria

The Notion of a Stack

- Collection of items
  - Items are returned in the ***reverse*** order they were added
  - This is behavior is often abbreviated LIFO (Last In, First Out)



Stack Examples

- Things we use all the time:
  - “Undo” function found in most applications
  - Back button when browsing the web

- Programming:
  - The runtime environment’s handling of nested method calls
  - Recursion

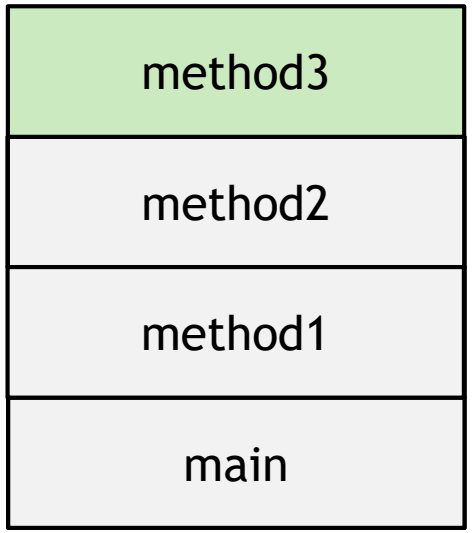
Runtime Environment

```
public static void method3() {}

public static void method2() {
    method3();
}

public static void method1() {
    method2();
    method3();
}

public static void main(String[] args) {
    method1();
}
```



The Stack ADT

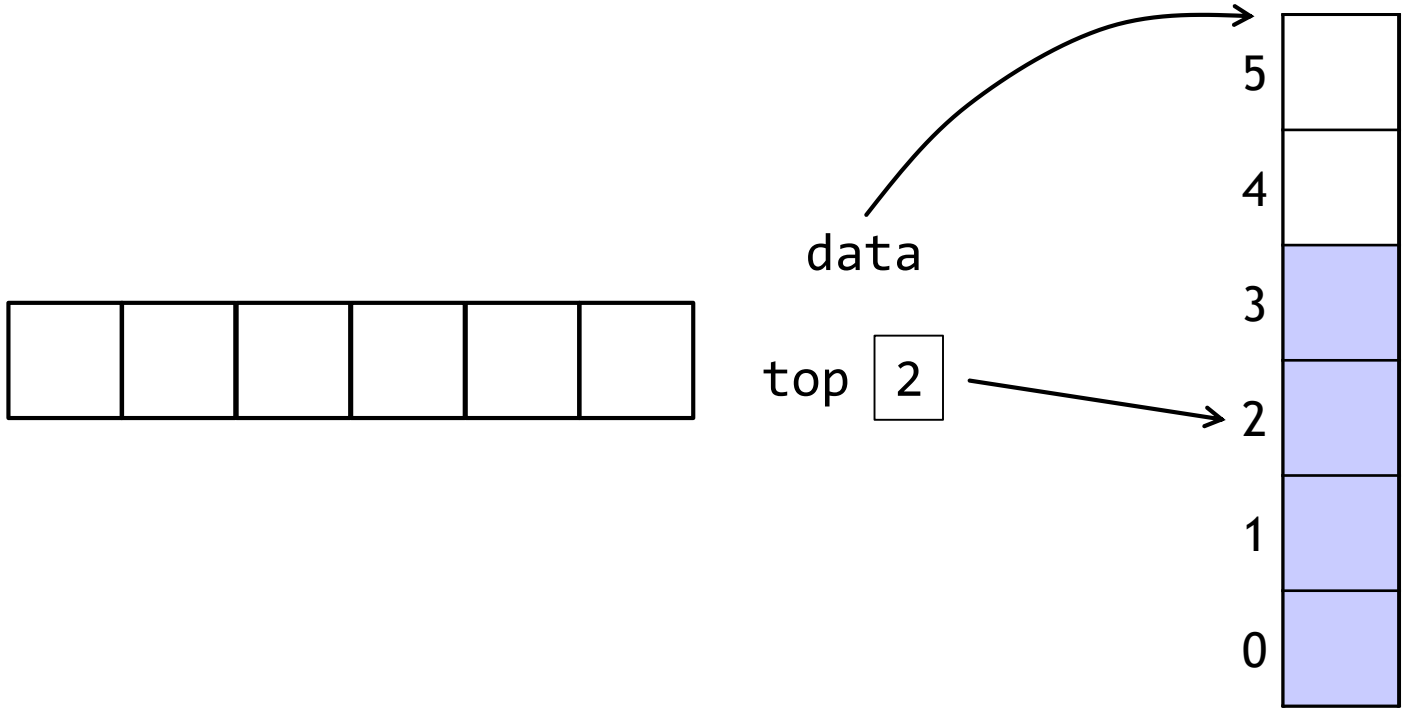
- The Stack ADT specifies the following operations:
  - Create an empty stack
  - Determine whether a stack is empty
  - Insert an object onto the stack
  - Remove the most recently added item from the stack
  - Remove all items from the stack
  - Access the most recently added item from the stack

Stack Interface:

```
isEmpty()
push(o)
pop()
popAll()
top()
```

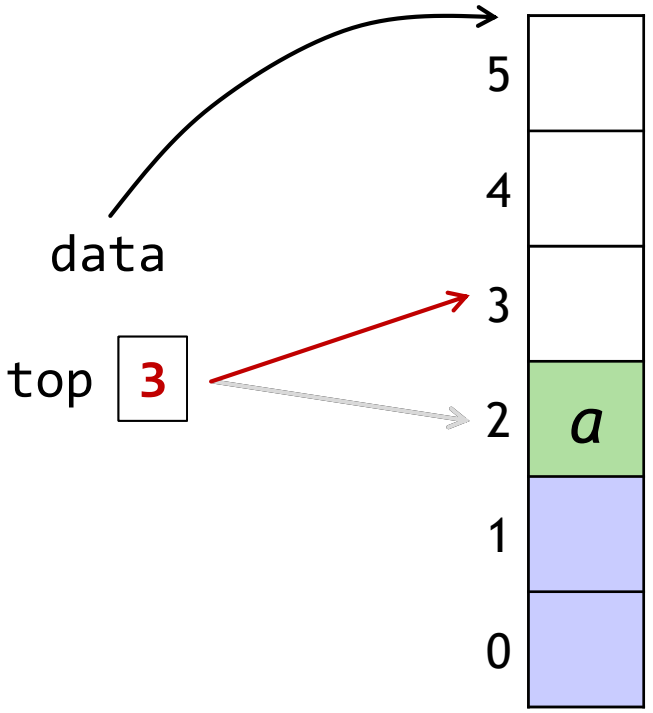
Stack Implementation (Array)

- A stack can be implemented in multiple ways
- In an array implementation, we typically have the following:
  - **`data`**: an n-element array
  - **`top`**: an integer to keep track of the index to insert the next element
- Example:



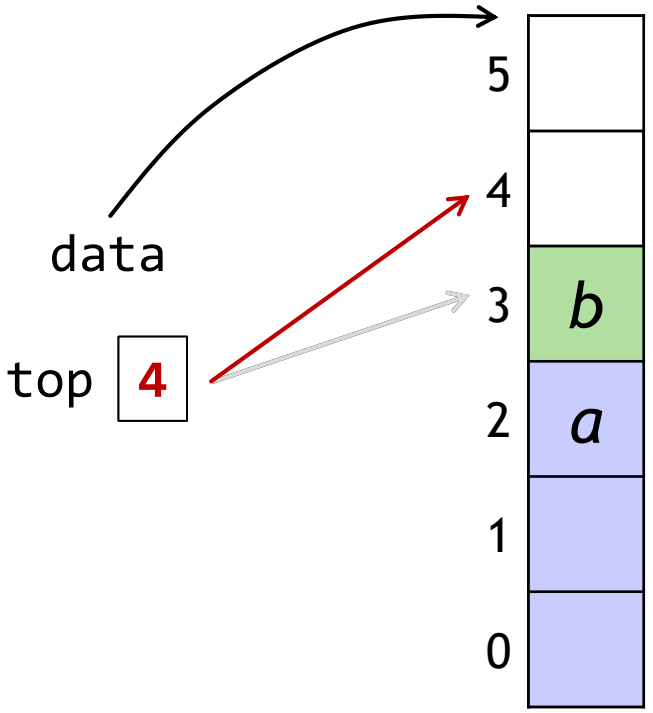
Stack Implementation (Array)

- A stack can be implemented in multiple ways
- In an array implementation, we typically have the following:
  - **`data`**: an n-element array
  - **`top`**: an integer to keep track of the index to insert the next element
- Example:
  - **push(*a*)**



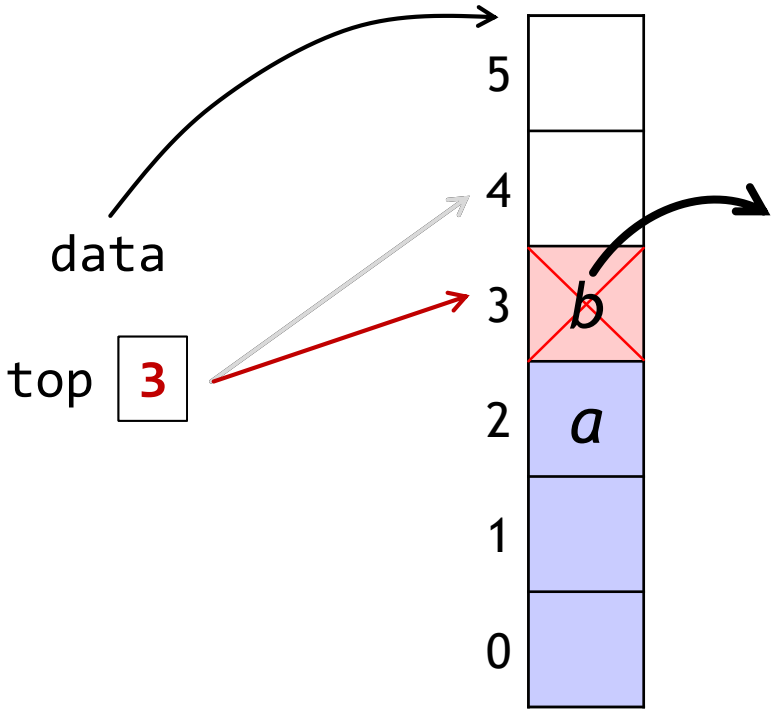
Stack Implementation (Array)

- A stack can be implemented in multiple ways
- In an array implementation, we typically have the following:
  - **`data`**: an n-element array
  - **`top`**: an integer to keep track of the index to insert the next element
- Example:
  - push(a)
  - **push(b)**



Stack Implementation (Array)

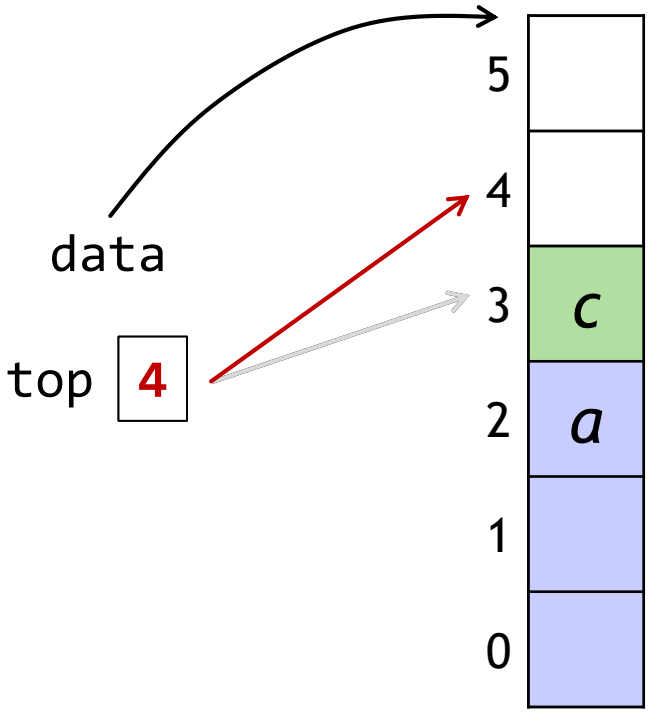
- A stack can be implemented in multiple ways
- In an array implementation, we typically have the following:
  - **`data`**: an n-element array
  - **`top`**: an integer to keep track of the index to insert the next element
- Example:
  - push(a)
  - push(b)
  - **pop()**



Stack Implementation (Array)

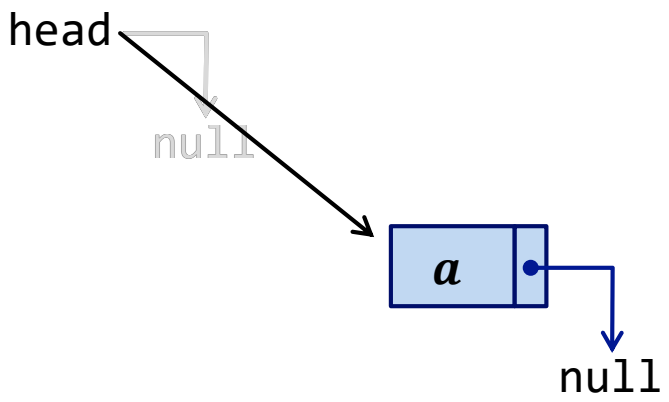
- A stack can be implemented in multiple ways
- In an array implementation, we typically have the following:
  - **`data`**: an n-element array
  - **`top`**: an integer to keep track of the index to insert the next element
- Example:
  - push(a)
  - push(b)
  - pop()
  - **push(c)**

Key observation:
We've done this before (**addBack** and **removeBack**)



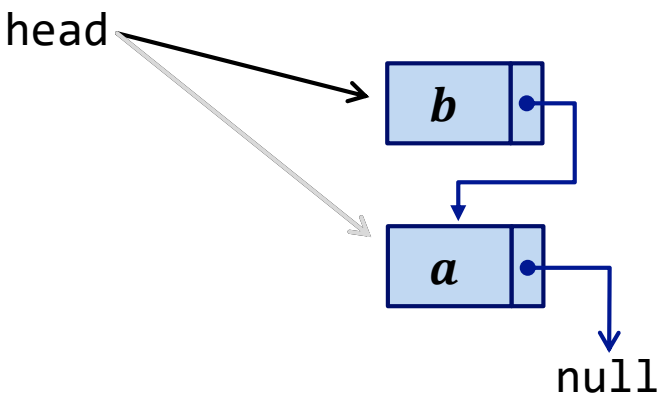
Stack Implementation (Linked List)

- A stack can be implemented in multiple ways
- In a linked list implementation, a stack can easily be implemented using singly-linked nodes and a **head** reference (**tail** isn't necessary!)
- Example:
  - **push(*a*)**



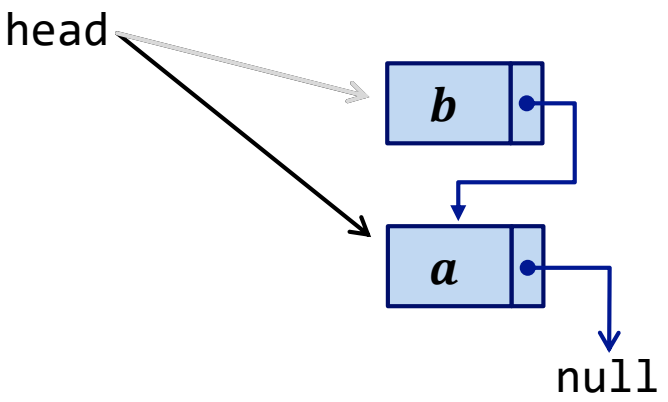
Stack Implementation (Linked List)

- A stack can be implemented in multiple ways
- In a linked list implementation, a stack can easily be implemented using singly-linked nodes and a **head** reference (**tail** isn't necessary!)
- Example:
  - push(a)
  - **push(b)**



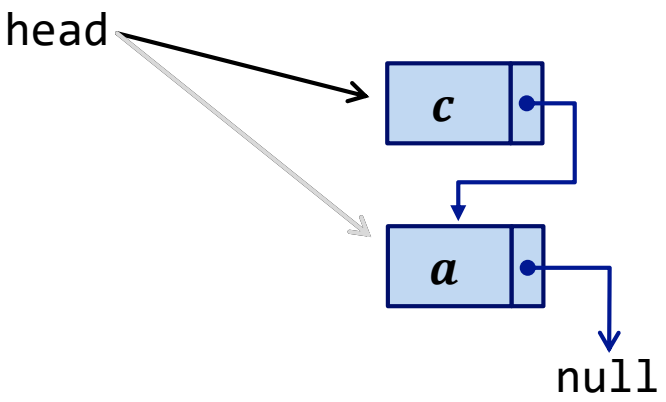
Stack Implementation (Linked List)

- A stack can be implemented in multiple ways
- In a linked list implementation, a stack can easily be implemented using singly-linked nodes and a **head** reference (**tail** isn't necessary!)
- Example:
  - push(a)
  - push(b)
  - **pop()**



Stack Implementation (Linked List)

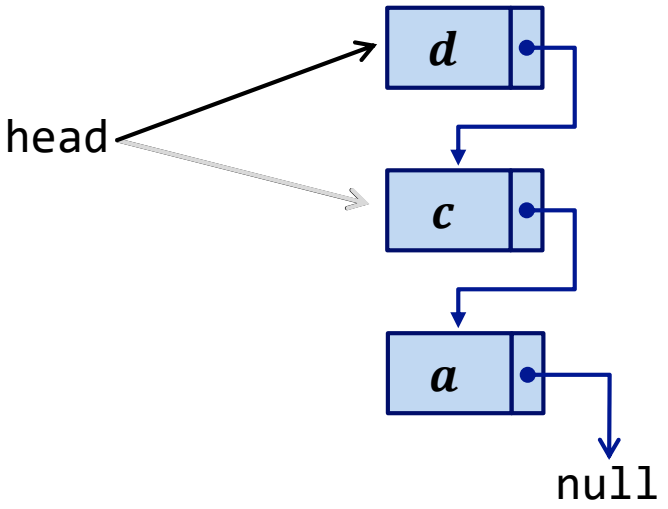
- A stack can be implemented in multiple ways:
- In a linked list implementation, a stack can easily be implemented using singly-linked nodes and a **head** reference (**tail** isn't necessary!)
- Example:
  - push(a)
  - push(b)
  - pop()
  - **push(c)**



Stack Implementation (Linked List)

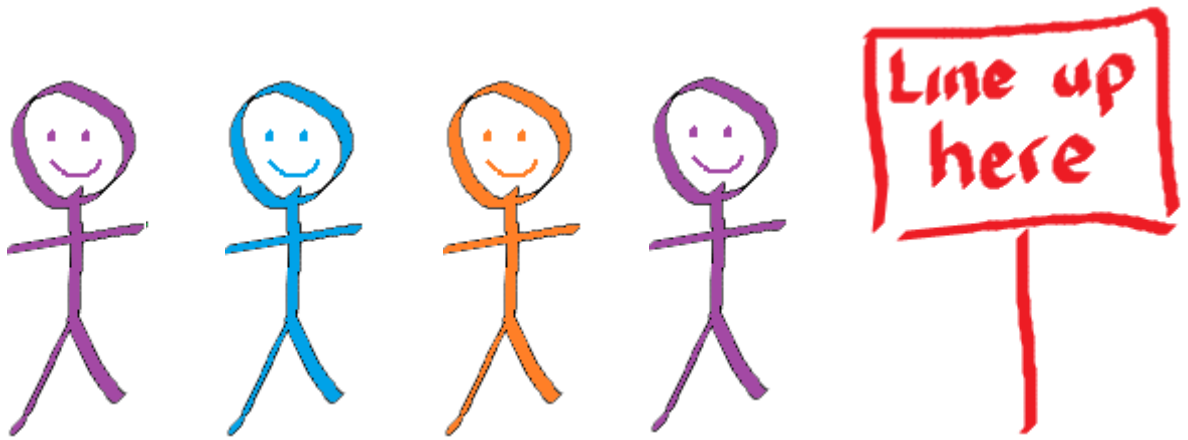
- A stack can be implemented in multiple ways
- In a linked list implementation, a stack can easily be implemented using singly-linked nodes and a **head** reference (**tail** isn't necessary!)
- Example:
  - push(a)
  - push(b)
  - pop()
  - push(c)
  - **push(d)**

Key observation:
We've done this before (**`addFront`** and **`removeFront`**)



The Notion of a Queue

- Collection of items
  - Items are returned in the ***same*** order they were added
  - This is behavior is often abbreviated FIFO (First In, First Out)



Queue Examples

- Any time people wait in line for something
  - the bank, the cafeteria, etc.

- Waitlists for classes here at Uvic!

The Queue ADT

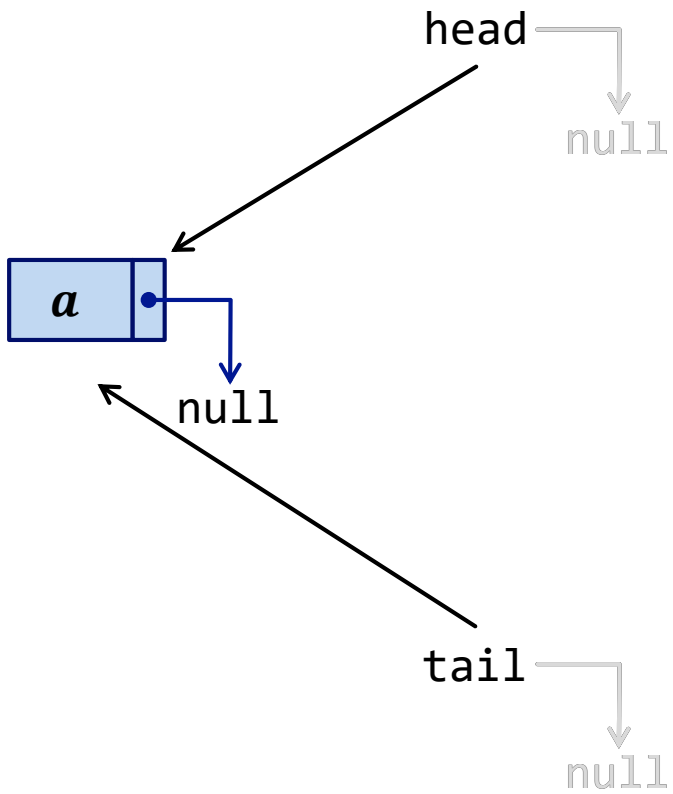
- The Queue ADT specifies the following operations:
  - Create an empty queue
  - Determine whether a queue is empty
  - Add an object to the back of the queue
  - Remove the object from the from of the queue
  - Remove all objects from the queue
  - Access the object at the front of the queue

Queue Interface:

```
isEmpty()
enqueue(o)
dequeue()
dequeueAll()
front()
```

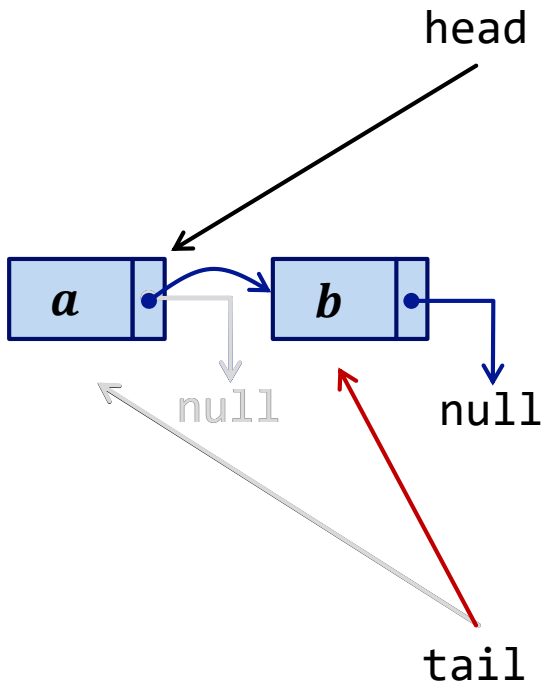
Queue Implementation (Linked List)

- A queue can be implemented in multiple ways
- In a linked list implementation, a queue can easily be implemented using singly-linked nodes with **head** and **tail** references
- Example:
  - **enqueue(*a*)**



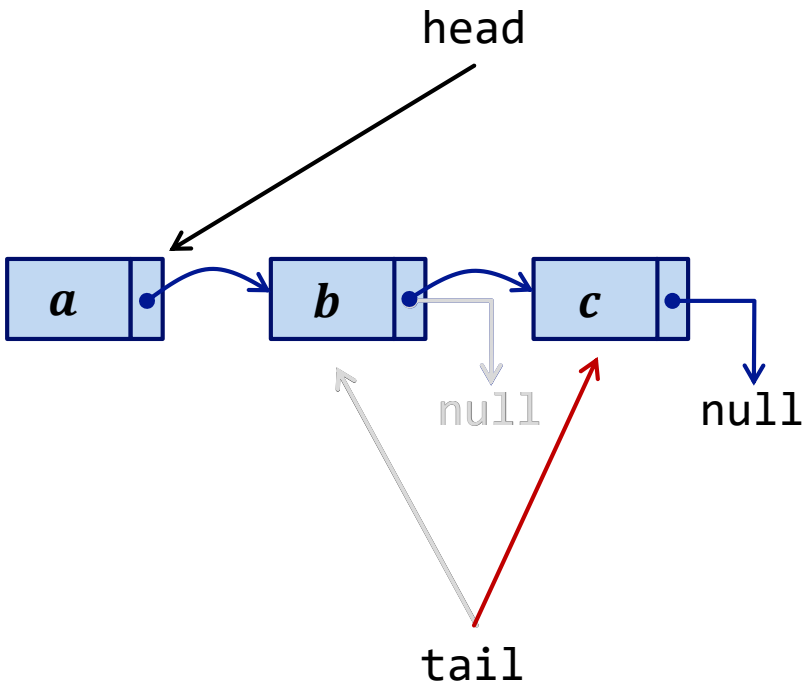
Queue Implementation (Linked List)

- A queue can be implemented in multiple ways
- In a linked list implementation, a queue can easily be implemented using singly-linked nodes with **head** and **tail** references
- Example:
  - enqueue(*a*)
  - **enqueue(*b*)**



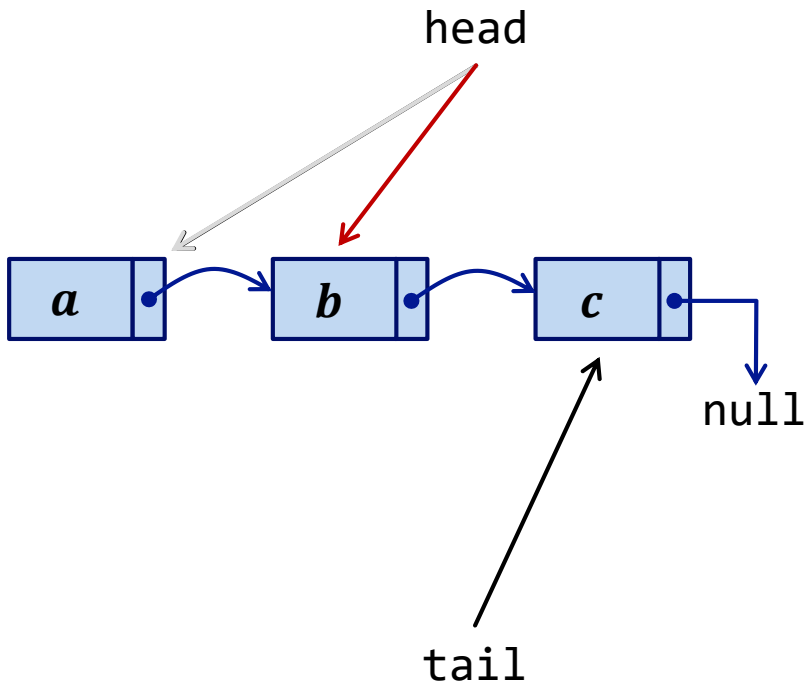
Queue Implementation (Linked List)

- A queue can be implemented in multiple ways
- In a linked list implementation, a queue can easily be implemented using singly-linked nodes with **head** and **tail** references
- Example:
  - enqueue(a)
  - enqueue(b)
  - **enqueue(c)**



Queue Implementation (Linked List)

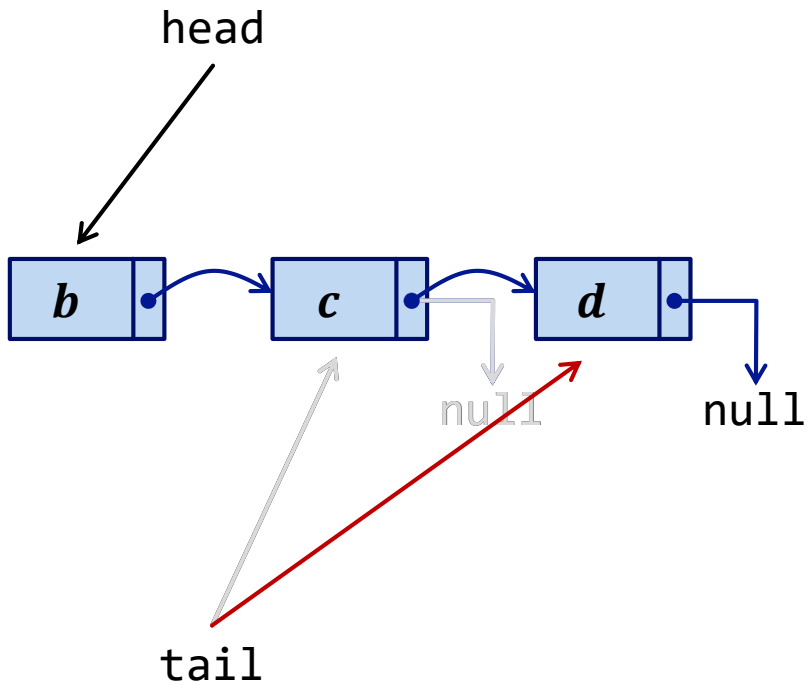
- A queue can be implemented in multiple ways
- In a linked list implementation, a queue can easily be implemented using singly-linked nodes with **head** and **tail** references
- Example:
  - enqueue(a)
  - enqueue(b)
  - enqueue(c)
  - **dequeue()**



Queue Implementation (Linked List)

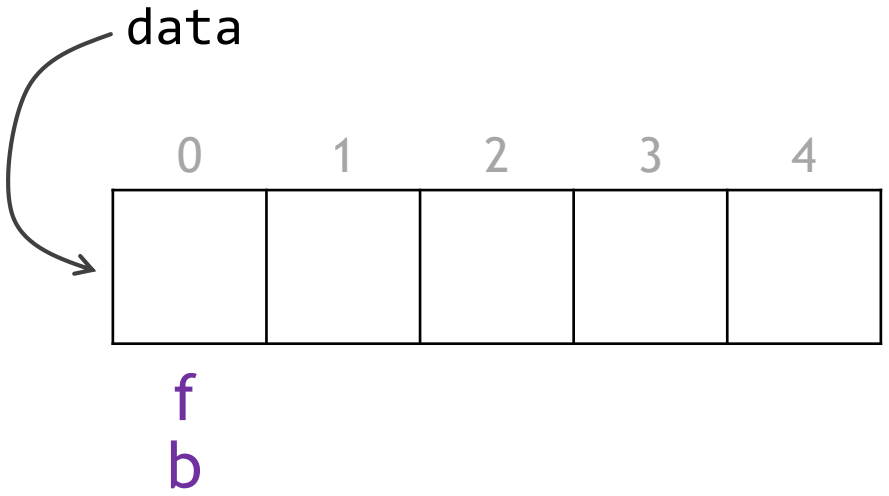
- A queue can be implemented in multiple ways
- In a linked list implementation, a queue can easily be implemented using singly-linked nodes with **head** and **tail** references
- Example:
  - enqueue(a)
  - enqueue(b)
  - enqueue(c)
  - dequeue()
  - **enqueue(d)**

Key observation:
We've done this before (**`addBack`** and **`removeFront`**)



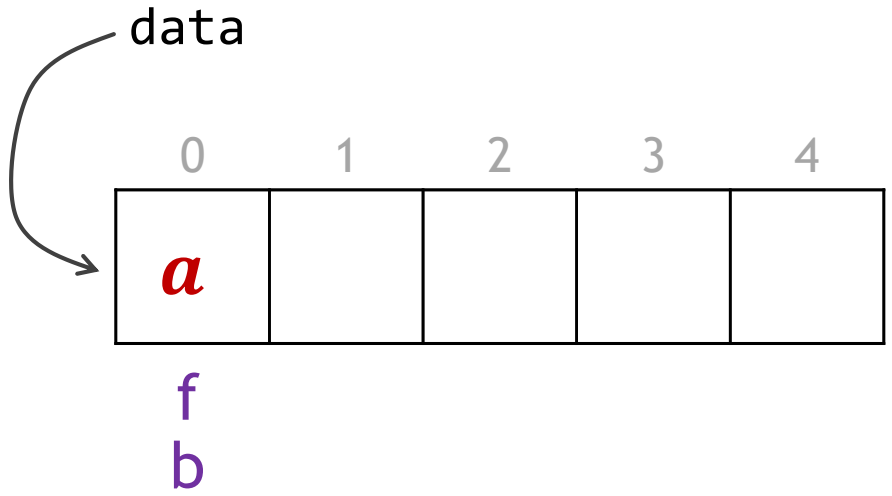
Queue Implementation (Array)

- A queue can be implemented in multiple ways
- In an array linked list implementation, we typically have the following:
  - **`data`**: an n-element array
  - **`f`**: an integer representing the index of the front element in the queue
  - **`b`**: an integer to keep of the index to insert the next element
- Example:



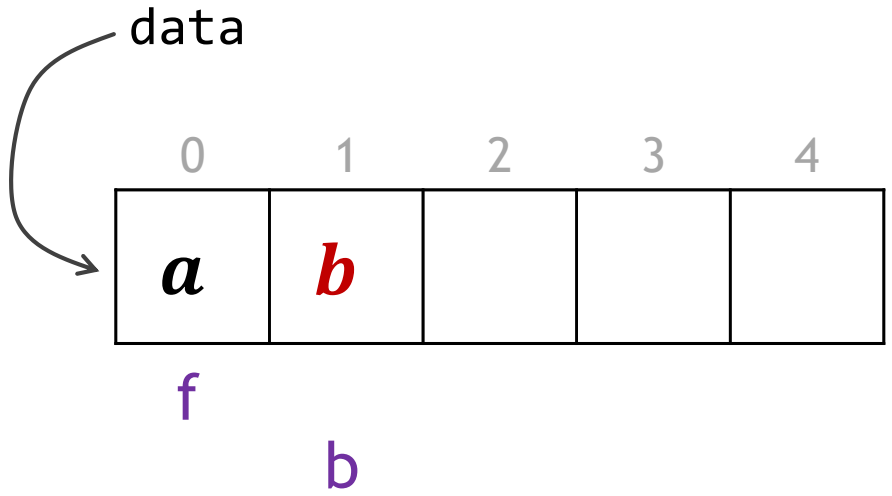
Queue Implementation (Array)

- A queue can be implemented in multiple ways
- In an array linked list implementation, we typically have the following:
  - **`data`**: an n-element array
  - **`f`**: an integer representing the index of the front element in the queue
  - **`b`**: an integer to keep of the index to insert the next element
- Example:
  - **enqueue(*a*)**



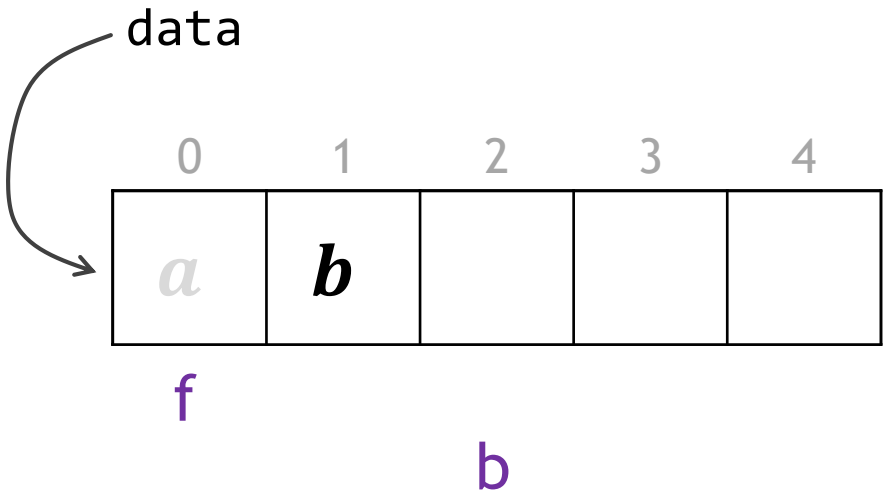
Queue Implementation (Array)

- A queue can be implemented in multiple ways
- In an array linked list implementation, we typically have the following:
  - **`data`**: an n-element array
  - **`f`**: an integer representing the index of the front element in the queue
  - **`b`**: an integer to keep of the index to insert the next element
- Example:
  - enqueue(a)
  - **enqueue(b)**



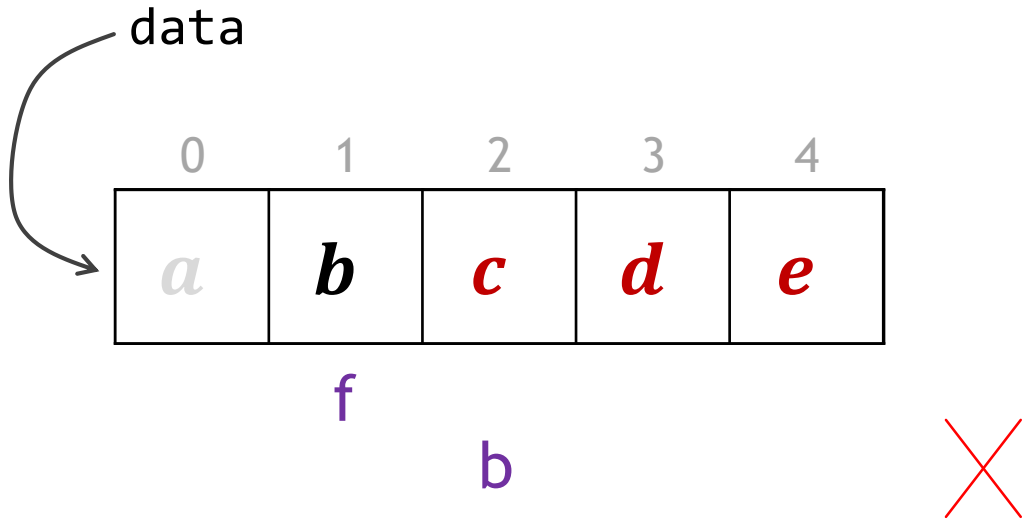
Queue Implementation (Array)

- A queue can be implemented in multiple ways
- In an array linked list implementation, we typically have the following:
  - **`data`**: an n-element array
  - **`f`**: an integer representing the index of the front element in the queue
  - **`b`**: an integer to keep of the index to insert the next element
- Example:
  - enqueue(a)
  - enqueue(b)
  - **dequeue()**



Queue Implementation (Array)

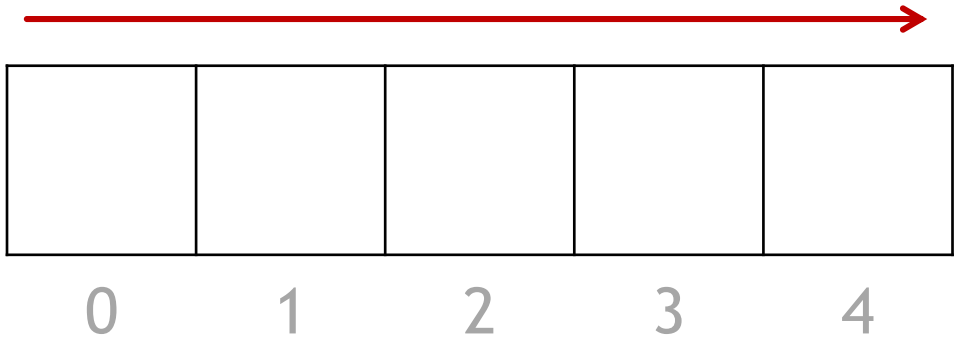
- A queue can be implemented in multiple ways
- In an array linked list implementation, we typically have the following:
  - **`data`**: an n-element array
  - **`f`**: an integer representing the index of the front element in the queue
  - **`b`**: an integer to keep of the index to insert the next element
- Example:
  - enqueue(a)
  - enqueue(b)
  - dequeue()
  - **3*enqueue**

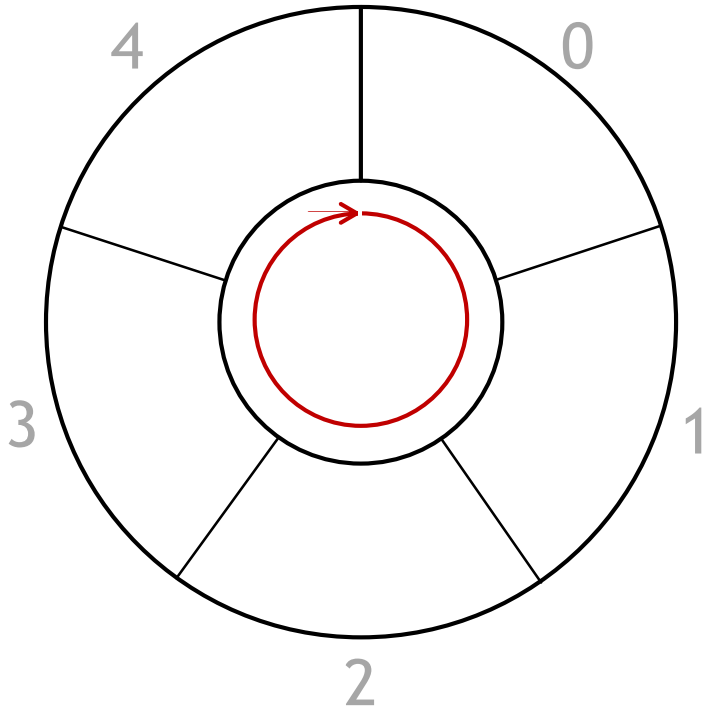


`ArrayIndexOutOfBoundsException`

Idea: Circular Arrays

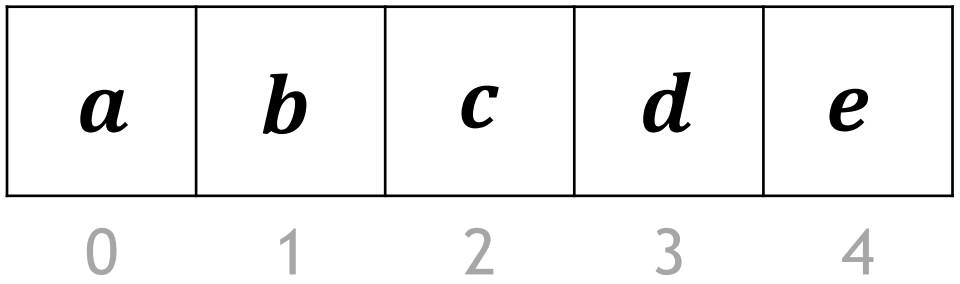
- Problem: ArrayIndexOutOfBoundsException
- Solution: Have the **b** variable go back around to the first index

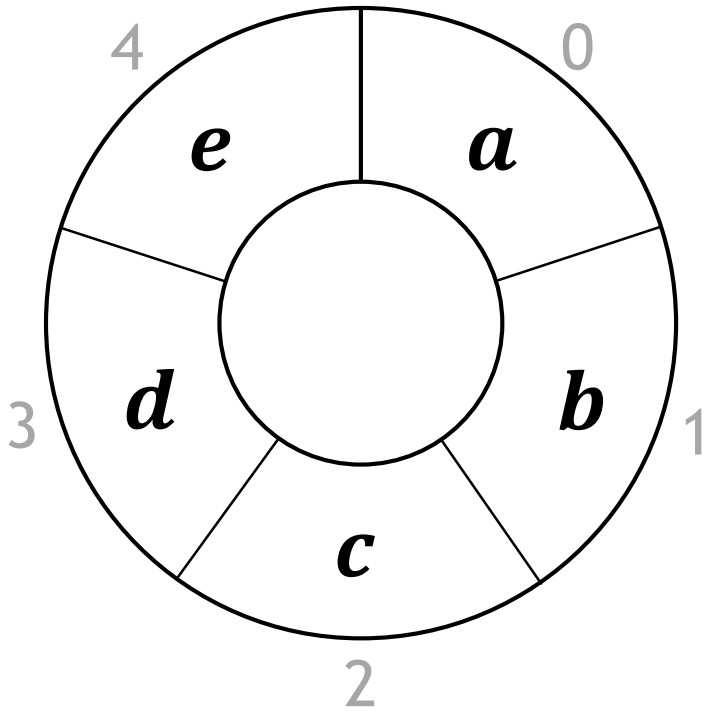




Idea: Circular Arrays

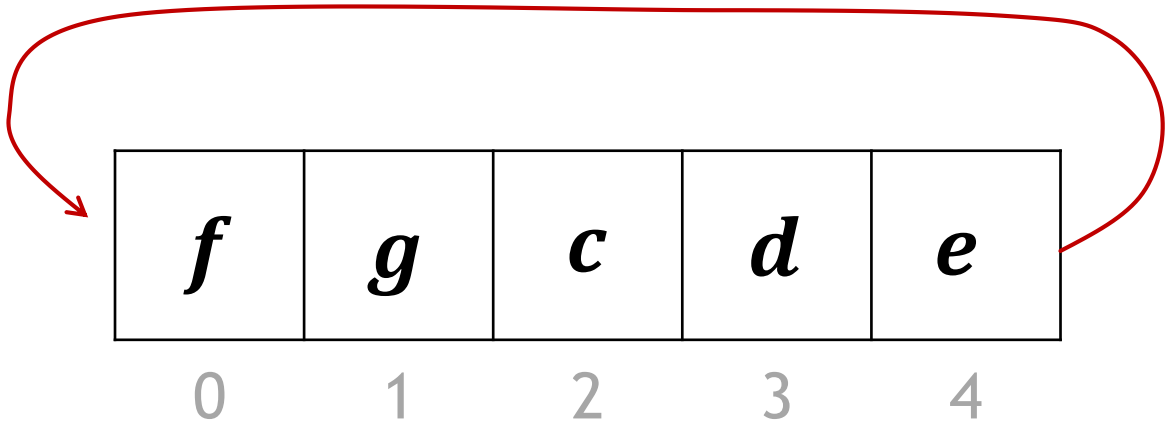
- Problem: ArrayIndexOutOfBoundsException
- Solution: Have the **b** variable go back around to the first index

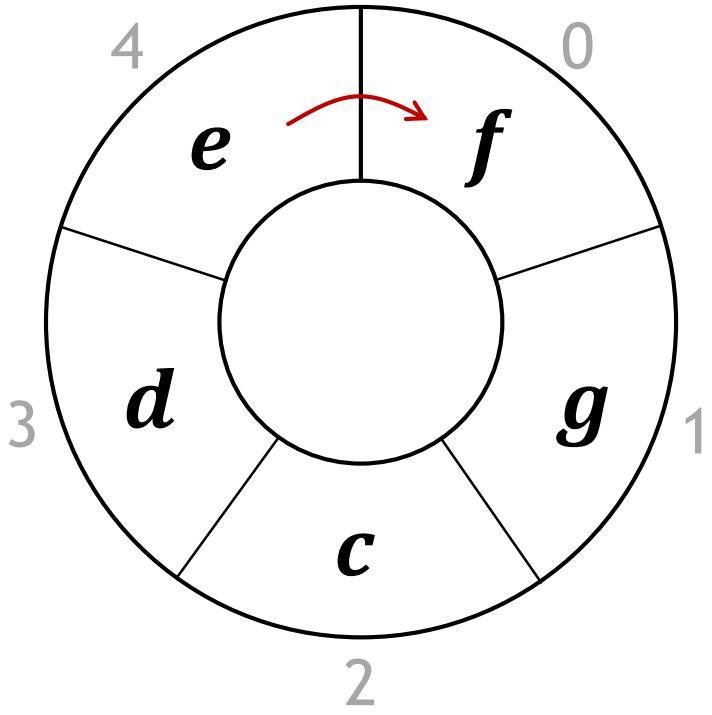




Idea: Circular Arrays

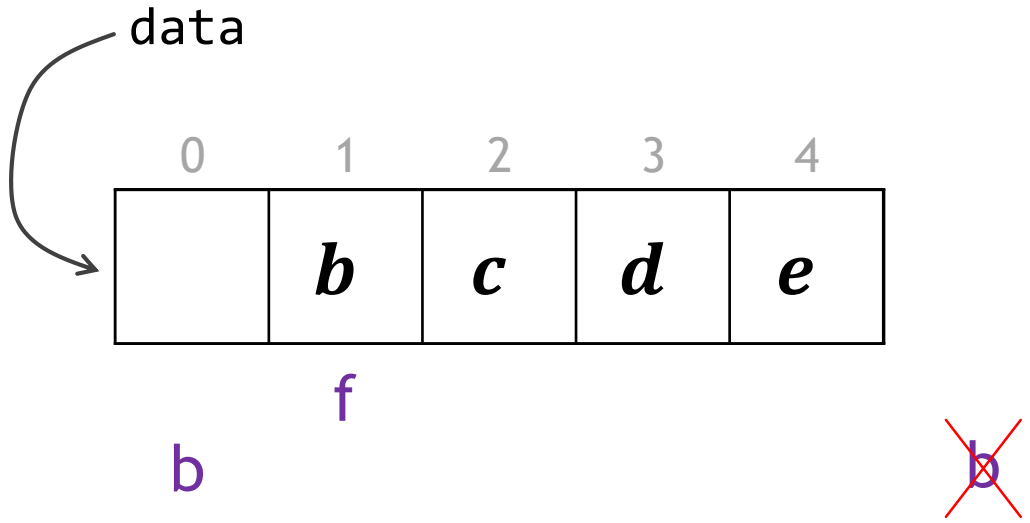
- Problem: ArrayIndexOutOfBoundsException
- Solution: Have the **b** variable go back around to the first index





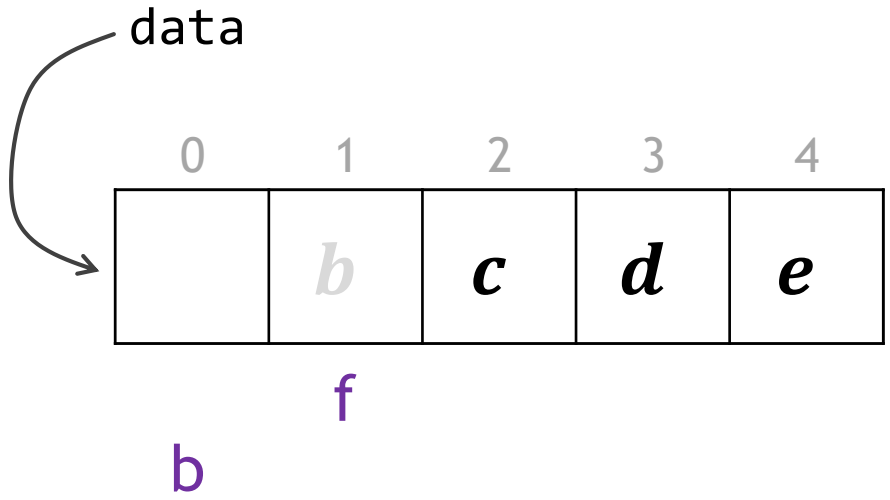
Queue Implementation (Array)

- A queue can be implemented in multiple ways
- In an array linked list implementation, we typically have the following:
  - **`data`**: an n-element array
  - **`f`**: an integer representing the index of the front element in the queue
  - **`b`**: an integer to keep of the index to insert the next element
- Example:



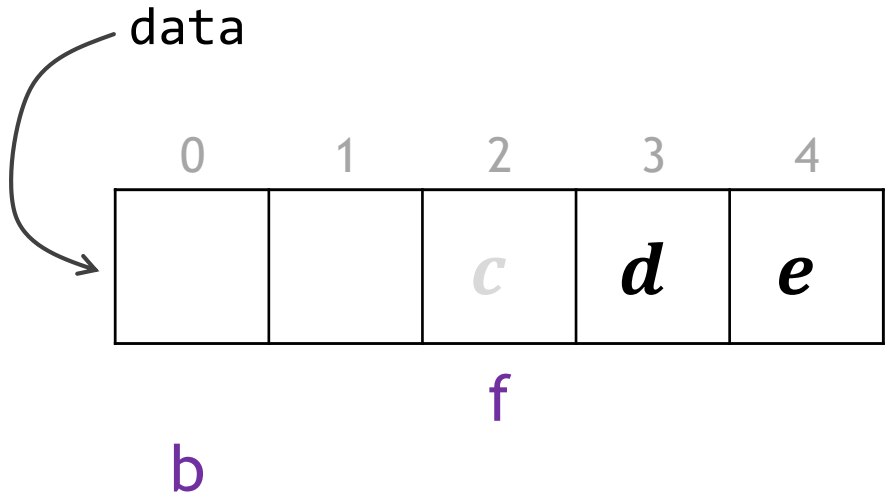
Queue Implementation (Array)

- A queue can be implemented in multiple ways
- In an array linked list implementation, we typically have the following:
  - **`data`**: an n-element array
  - **`f`**: an integer representing the index of the front element in the queue
  - **`b`**: an integer to keep of the index to insert the next element
- Example:
  - **dequeue()**



Queue Implementation (Array)

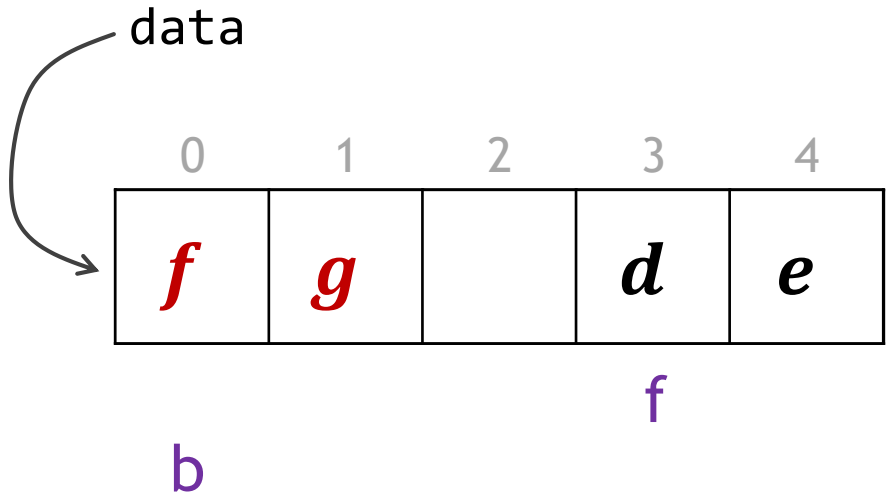
- A queue can be implemented in multiple ways
- In an array linked list implementation, we typically have the following:
  - **`data`**: an n-element array
  - **`f`**: an integer representing the index of the front element in the queue
  - **`b`**: an integer to keep of the index to insert the next element
- Example:
  - dequeue()
  - **dequeue()**



Queue Implementation (Array)

- A queue can be implemented in multiple ways
- In an array linked list implementation, we typically have the following:
  - **`data`**: an n-element array
  - **`f`**: an integer representing the index of the front element in the queue
  - **`b`**: an integer to keep of the index to insert the next element
- Example:
  - dequeue()
  - dequeue()
  - **enqueue(f)**
  - **enqueue(g)**

Key observation:
We've done this before (**`addBack`** and **`removeFront`**)



Analysis

Method	Array		Singly Linked		Doubly Linked	
	average	worst	no tail	tail	no tail	tail
Object get(int pos)						
int size()						
int find(Object o)						
String toString()						
void addAt(Object o, int pos)						
void addFront(Object o)						
void addBack(Object o)						
void removeAt(int pos)						
void removeFront()						
void removeBack()						
void swapElements(int pos1, int pos2)						

Analysis

Method	Array		Singly Linked		Doubly Linked	
	average	worst	no tail	tail	no tail	tail
Object get(int pos)	$O(1)$	$O(1)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
int size()	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$
int find(Object o)	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
String toString()	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
void addAt(Object o, int pos)	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
void addFront(Object o)	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$
void addBack(Object o)	$O(1)$	$O(n)$	$O(n)$	$O(1)$	$O(n)$	$O(1)$
void removeAt(int pos)	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
void removeFront()	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$
void removeBack()	$O(1)$	$O(1)$	$O(n)$	$O(n)$	$O(n)$	$O(1)$
void swapElements(int pos1, int pos2)	$O(1)$	$O(1)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$

Analysis - Stacks

Method	Array		Singly Linked		Doubly Linked	
	average	worst	no tail	tail	no tail	tail
Object get(int pos)	$O(1)$	$O(1)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
int size()	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$
int find(Object o)	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
String toString()	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
void addAt(Object o, int pos)	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
void addFront(Object o)	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$
void addBack(Object o)	$O(1)$	$O(n)$	$O(n)$	$O(1)$	$O(n)$	$O(1)$
void removeAt(int pos)	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
void removeFront()	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$
void removeBack()	$O(1)$	$O(1)$	$O(n)$	$O(n)$	$O(n)$	$O(1)$
void swapElements(int pos1, int pos2)	$O(1)$	$O(1)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$

Array

`push:` `addBack`

`pop:` `removeBack`

$O(1)$ runtime!

Analysis - Stacks

Method	Array		Singly Linked		Doubly Linked	
	average	worst	no tail	tail	no tail	tail
Object get(int pos)	$O(1)$	$O(1)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
int size()	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$
int find(Object o)	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
String toString()	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
void addAt(Object o, int pos)	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
void addFront(Object o)	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$
void addBack(Object o)	$O(1)$	$O(n)$	$O(n)$	$O(1)$	$O(n)$	$O(1)$
void removeAt(int pos)	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
void removeFront()	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$
void removeBack()	$O(1)$	$O(1)$	$O(n)$	$O(n)$	$O(n)$	$O(1)$
void swapElements(int pos1, int pos2)	$O(1)$	$O(1)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$

Linked List

push: `addFront`

pop: `removeFront`

$O(1)$ runtime!

Analysis - Queues

Method	Array		Singly Linked		Doubly Linked	
	average	worst	no tail	tail	no tail	tail
Object get(int pos)	$O(1)$	$O(1)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
int size()	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$
int find(Object o)	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
String toString()	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
void addAt(Object o, int pos)	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
void addFront(Object o)	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$
void addBack(Object o)	$O(1)$	$O(n)$	$O(n)$	$O(1)$	$O(n)$	$O(1)$
void removeAt(int pos)	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
void removeFront()	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$
void removeBack()	$O(1)$	$O(1)$	$O(n)$	$O(n)$	$O(n)$	$O(1)$
void swapElements(int pos1, int pos2)	$O(1)$	$O(1)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$

Linked List

`push: addFront`

`pop: removeFront`

$O(1)$ runtime!

Analysis - Queues

Method	Array		Singly Linked		Doubly Linked	
	average	worst	no tail	tail	no tail	tail
Object get(int pos)	$O(1)$	$O(1)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
int size()	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$
int find(Object o)	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
String toString()	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
void addAt(Object o, int pos)	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
void addFront(Object o)	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$
void addBack(Object o)	$O(1)$	$O(n)$	$O(n)$	$O(1)$	$O(n)$	$O(1)$
void removeAt(int pos)	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
void removeFront()	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$
void removeBack()	$O(1)$	$O(1)$	$O(n)$	$O(n)$	$O(n)$	$O(1)$
void swapElements(int pos1, int pos2)	$O(1)$	$O(1)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$

Array

`push: addBack`

`pop: removeBack`

`circular array`

$O(1)$ runtime!