https://algs4.cs.princeton.edu

1.3 Stacks and Queues II

- *linked lists*
- *stack implementation*
- *queue implementation*
- *iterators*
- *Java collections*

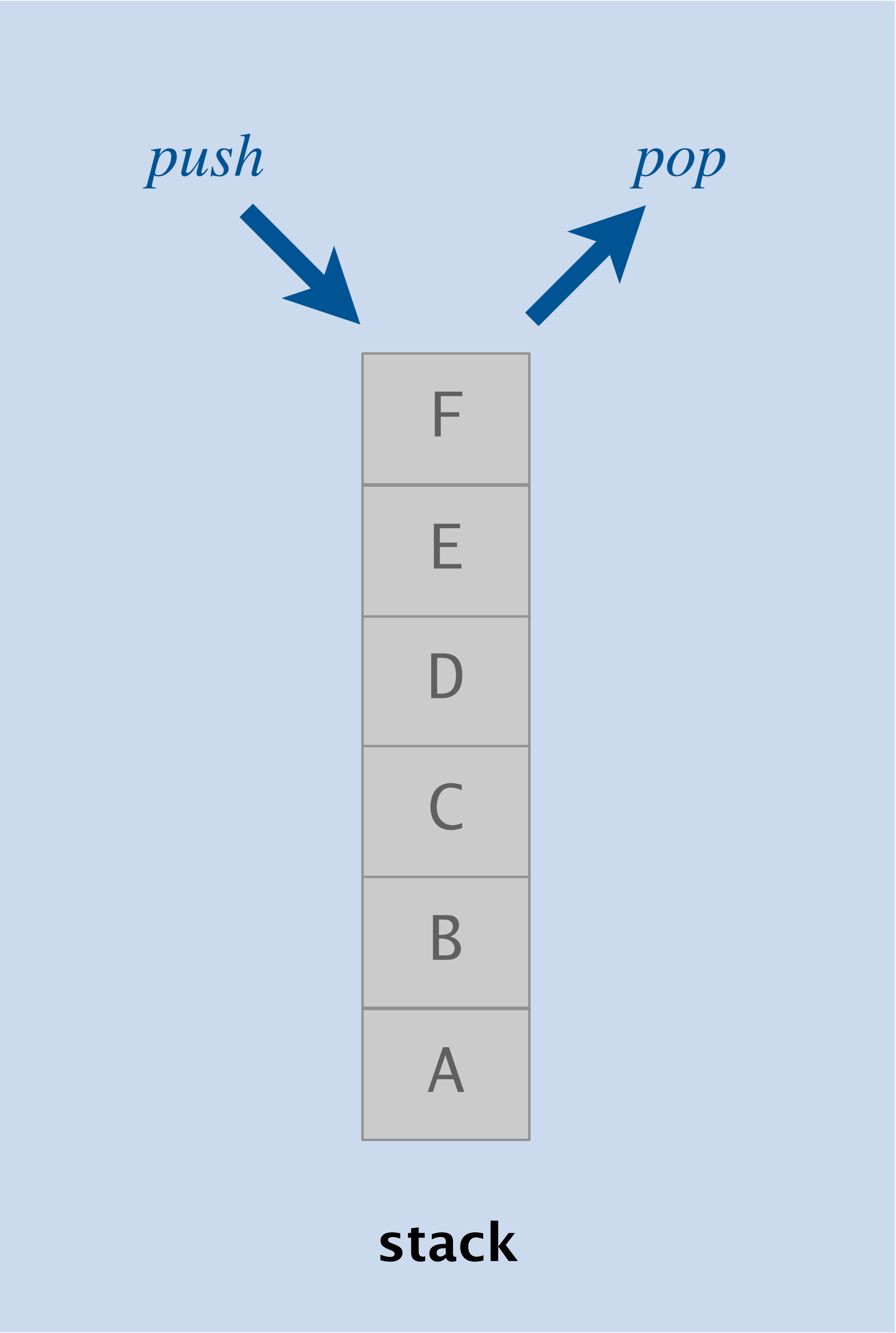
Stacks and queues

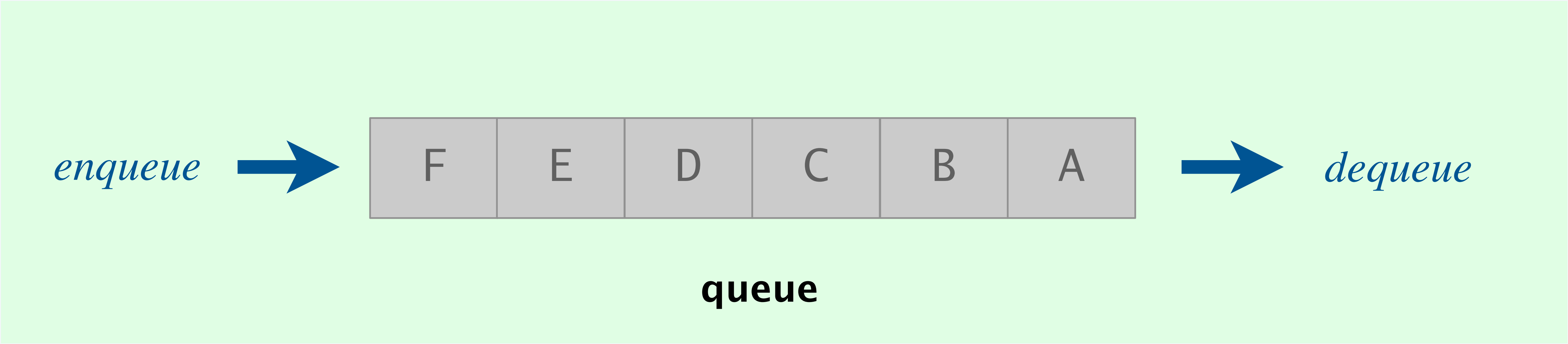
Fundamental data types.

- Value: collection of objects.
- Operations: add, remove, iterate, size, test if empty.

Stack. Remove the item most recently added.

Queue. Remove the item least recently added.





Programming assignment 2

Deque. Remove either the most recently or the least recently added item.

Randomized queue. Remove a random item.

Your job.

- Step 1. Identify a data structure that meets the performance requirements.
- Step 2. Implement it from scratch.

← *think carefully about step 1 before proceeding to step 2*

https://algs4.cs.princeton.edu

1.3 Stacks and Queues II

- *linked lists*
- *stack implementation*
- *queue implementation*
- *iterators*
- *Java collections*

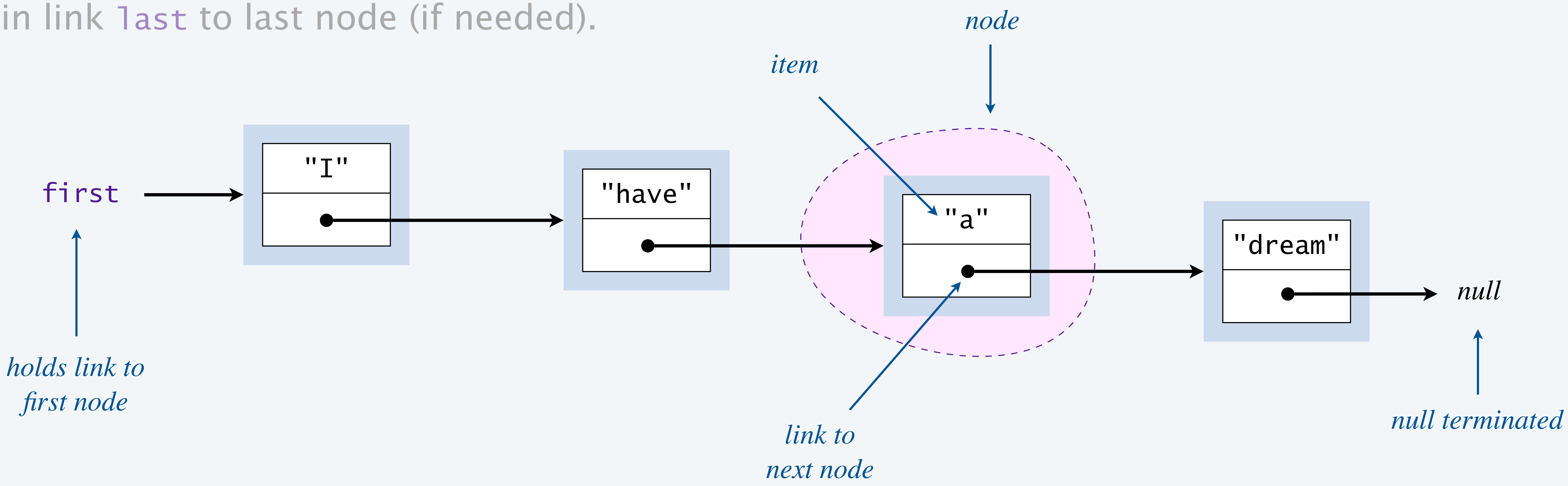
Linked lists

Last lecture. Use a resizable array to implement all operations in amortized $\Theta(1)$ time.

This lecture. Use a singly linked list to implement all operations in $\Theta(1)$ time in the worst case.

Singly linked list.

- Each node stores an item and a link/pointer to the next node in the sequence.
- Last node links to `null`.
- Maintain link `first` to first node.
- Maintain link `last` to last node (if needed).



Possible memory representation of an array

Java array. The elements in an array are stored contiguously in memory.

Consequences.

- Accessing array element i takes $\Theta(1)$ time.
- Cannot change the length of an array.
- When passing an array to a function, the function can change array elements.

memory representation
(using poetic license)

Possible memory representation of a singly linked list

Java linked list. The nodes in a linked list are stored non-contiguously in memory.

Consequences.

- Accessing i^{th} node in linked list takes $\Theta(i)$ time.
- Easy to change the length of a linked list.

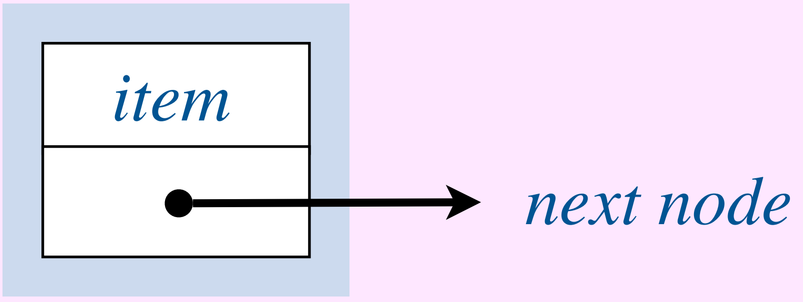
Creating a linked lists in Java

Node data type. Each Node object contains:

- An item.
- A reference to the next Node in the sequence.

```
public class Node {
   private String item;
   private Node next;
}
```

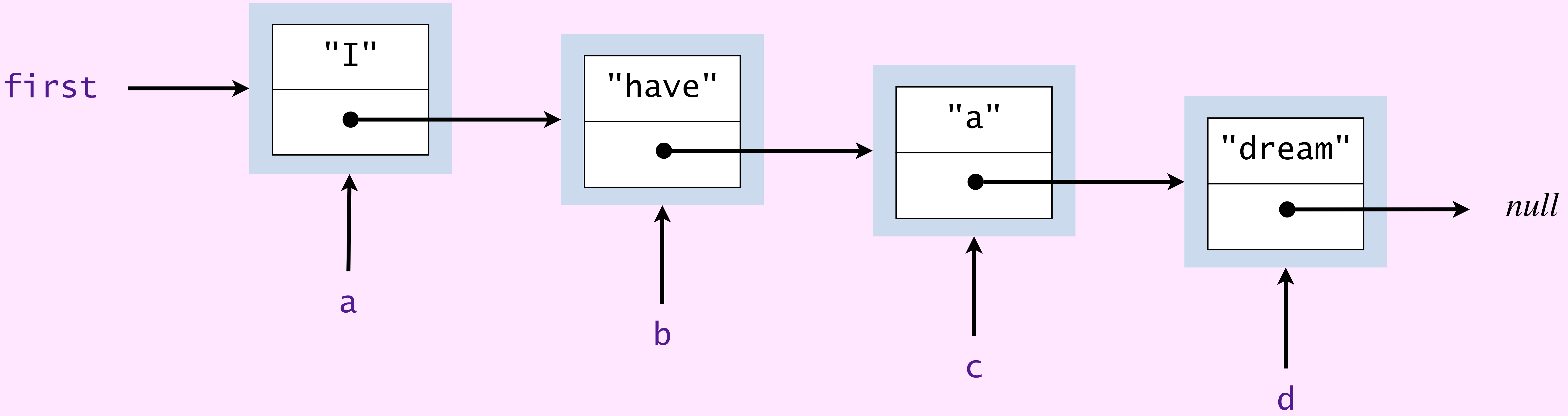
Node data type



Node object

```
Node a = new Node();
Node b = new Node();
Node c = new Node();
Node d = new Node();
a.item = "I";
b.item = "have";
c.item = "a";
d.item = "dream";
a.next = b;
b.next = c;
c.next = d;
d.next = null;
first = a;
```

creating a 4-node linked list

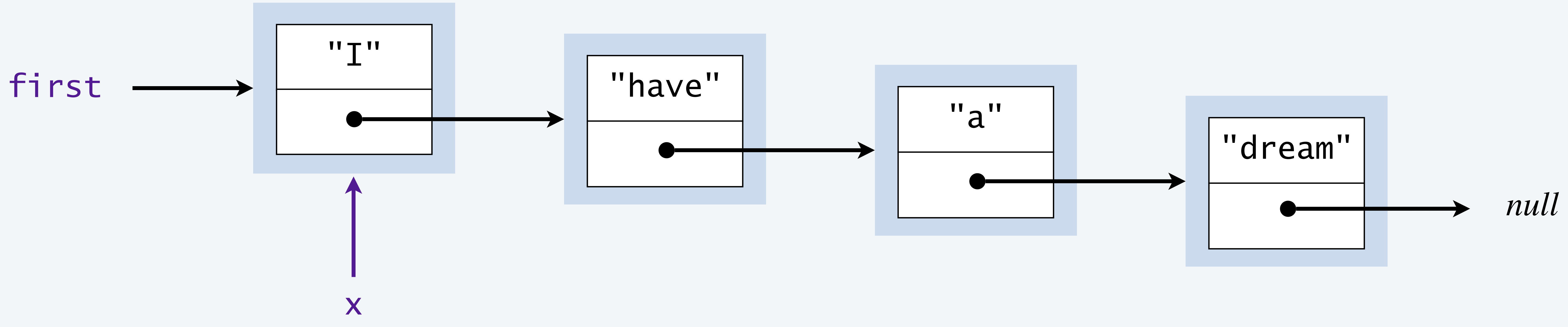


Traversing a singly linked list

Goal. Systematically process each element in a singly linked list.

Solution. For loop idiom.

```
for (Node x = first; x != null; x = x.next) {
   StdOut.println(x.item);
}
```



What does the following code fragment do to the linked list below?

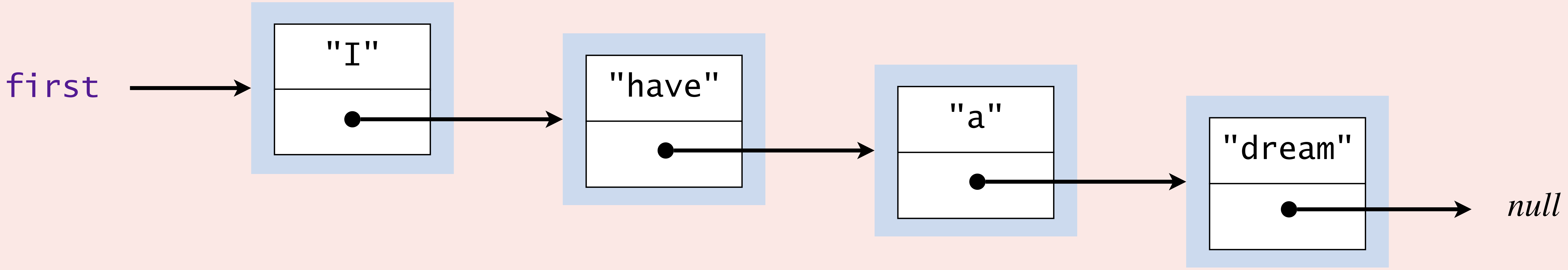
```
first.next = first.next.next;
```

A. Deletes node containing "I".

B. Deletes node containing "have".

C. Deletes node containing "a".

D. Leaves the linked list unchanged.



What does the following code fragment do to the linked list below?

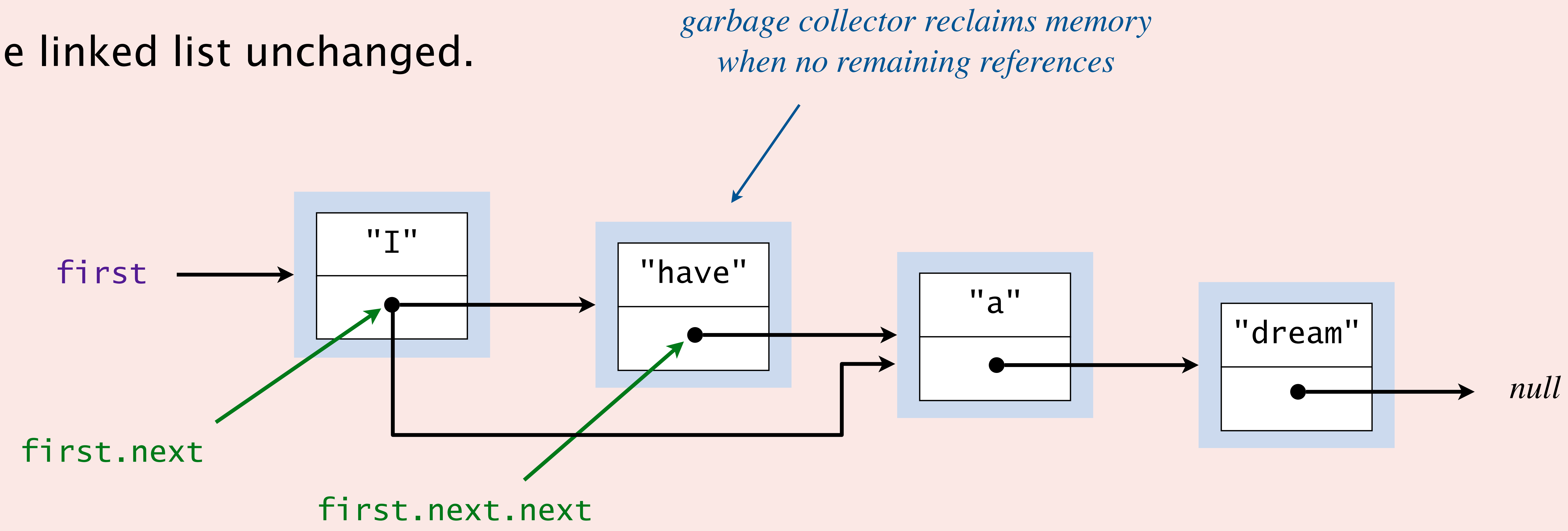
```
first.next = first.next.next;
```

A. Deletes node containing "I".

B. Deletes node containing "have".

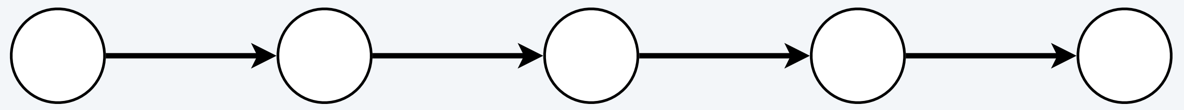
C. Deletes node containing "a".

D. Leaves the linked list unchanged.

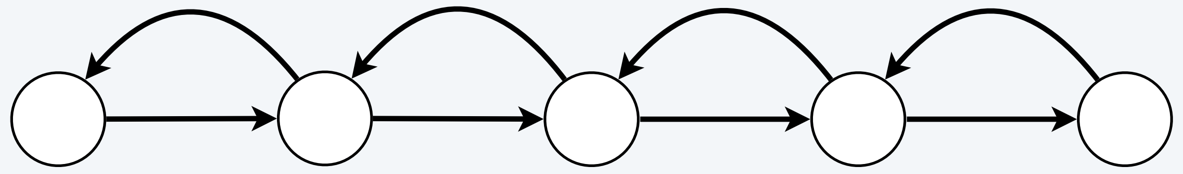


Linked data structures: context

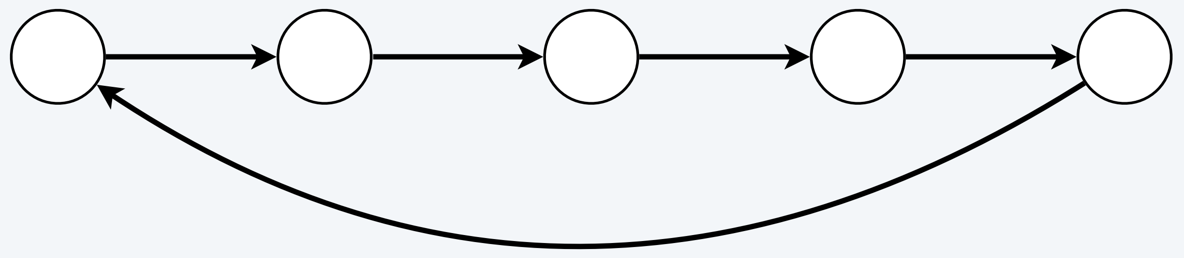
Null-terminated linked list.



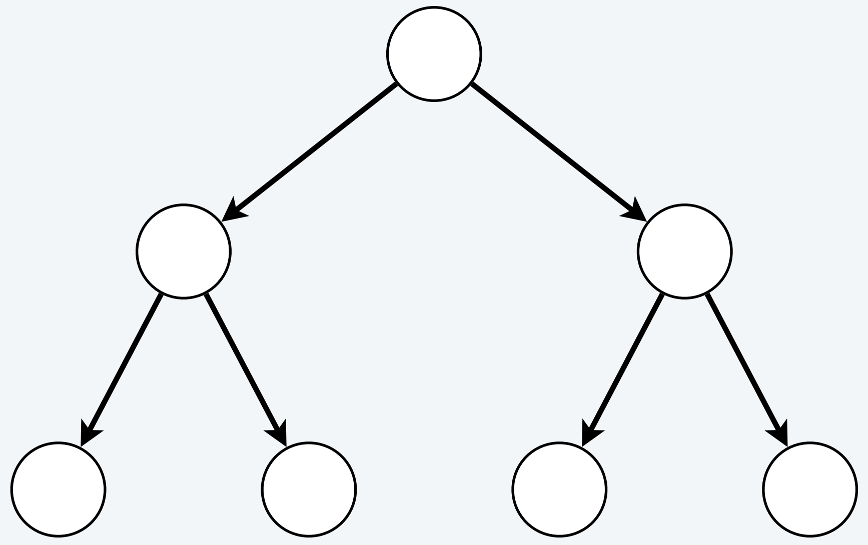
Doubly linked list. [2 links per node]



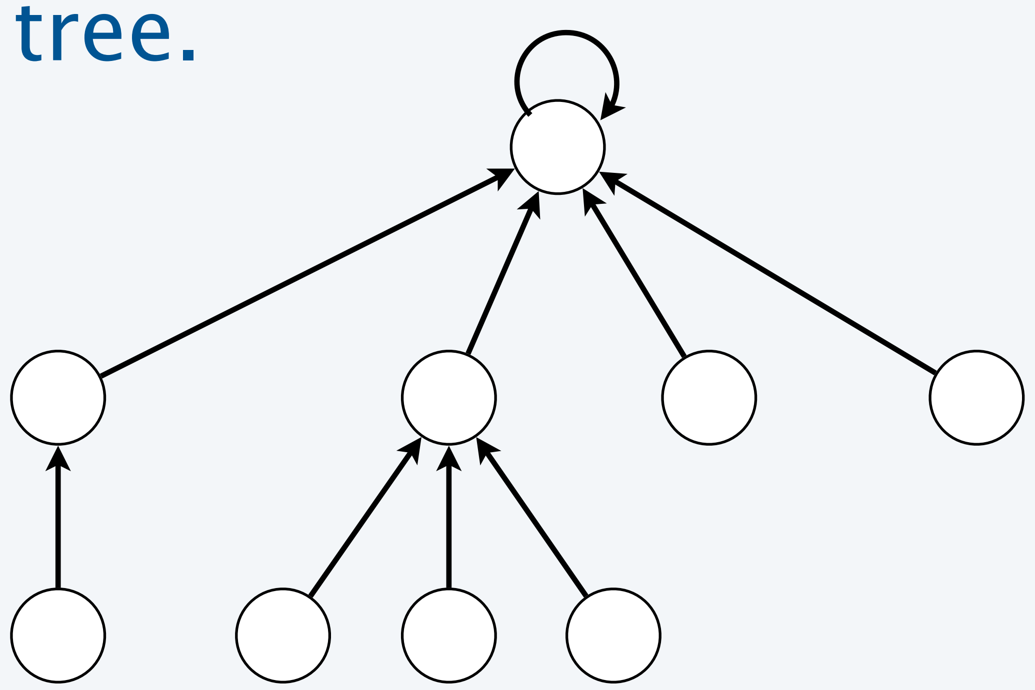
Circular linked list.



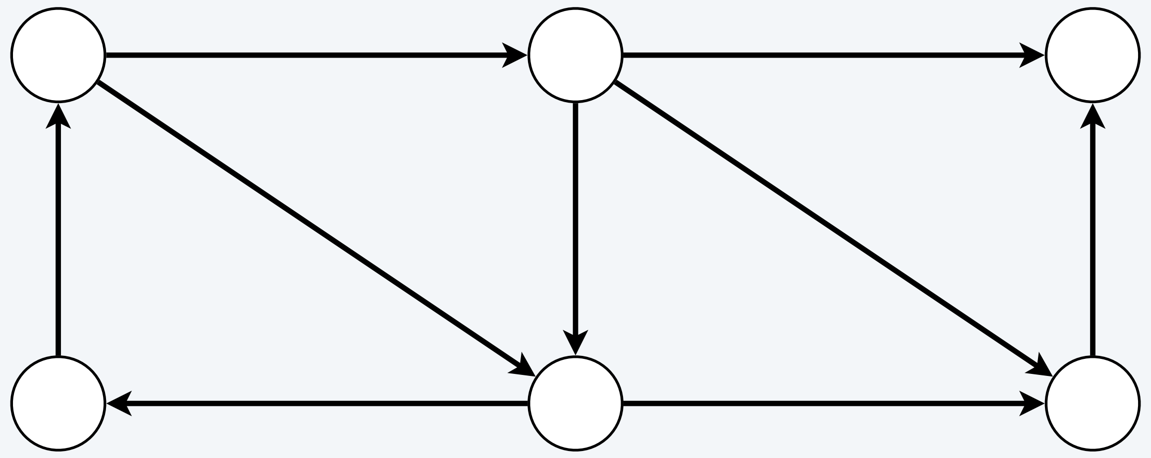
Binary tree. [2 links per node]



Parent-link tree.



Directed graph. [many links per node]



https://algs4.cs.princeton.edu

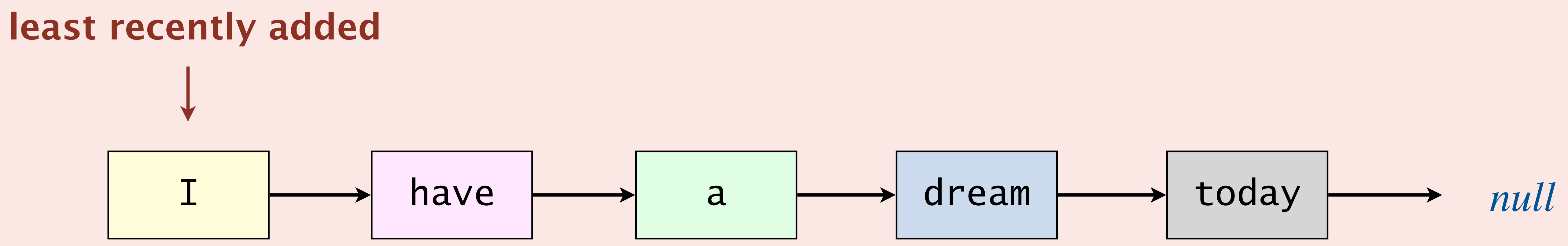
1.3 Stacks and Queues II

- linked lists
- stack implementation
- queue implementation
- iterators
- Java collections

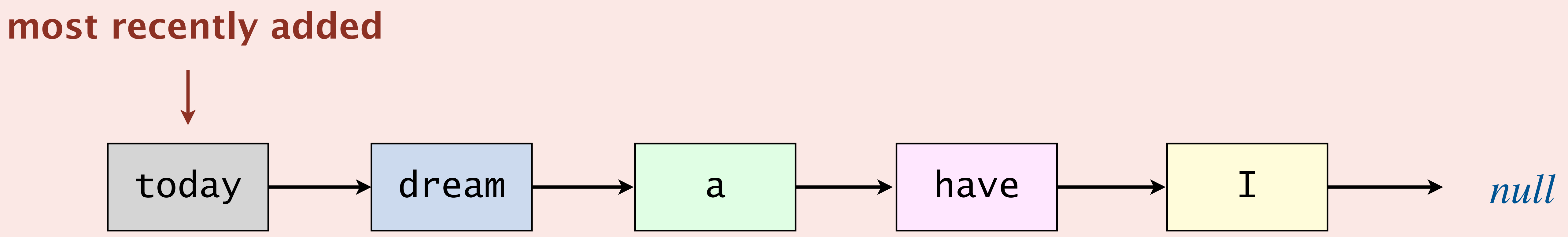
Stacks and queues II: poll 2

How to implement efficiently a stack with a singly linked list?

A.



B.

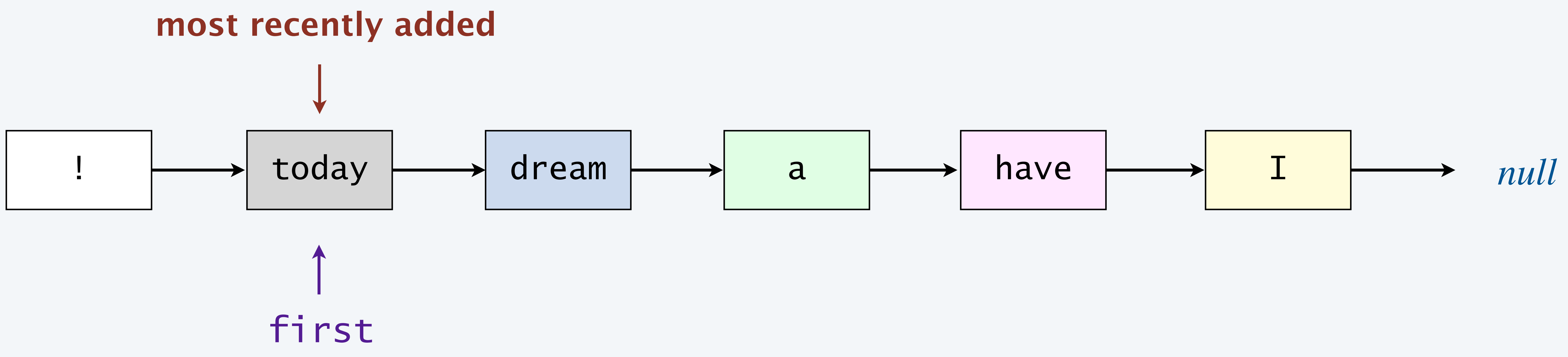


C. *Both A and B.*

D. *Neither A nor B.*

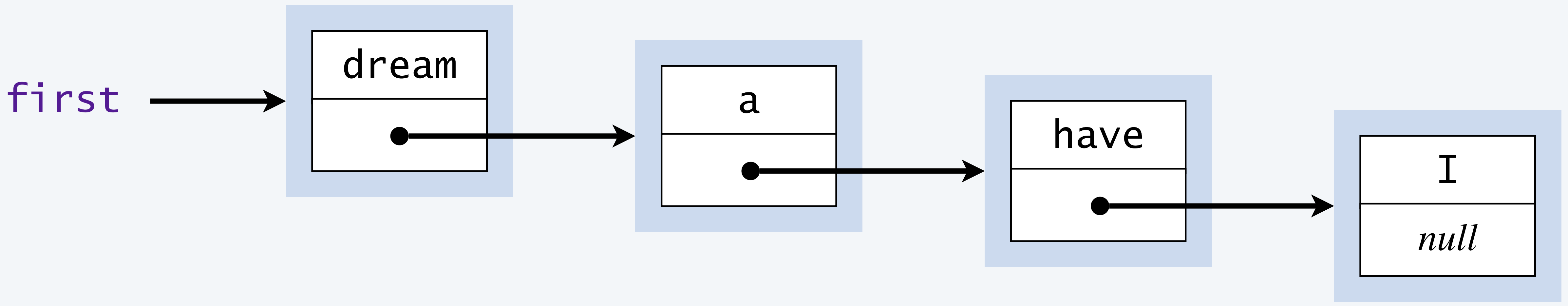
Stack: linked-list implementation

- Maintain link `first` to first node in a singly linked list.
- Push new item before `first`.
- Pop item from `first`.

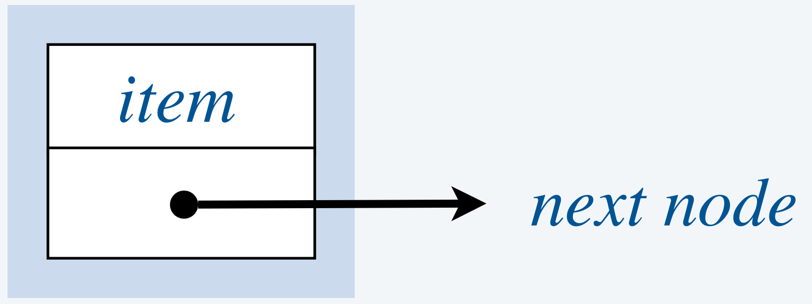


Stack implementation with a linked list: pop

singly linked list




```
public class Node {
   private String item;
   private Node next;
}
```



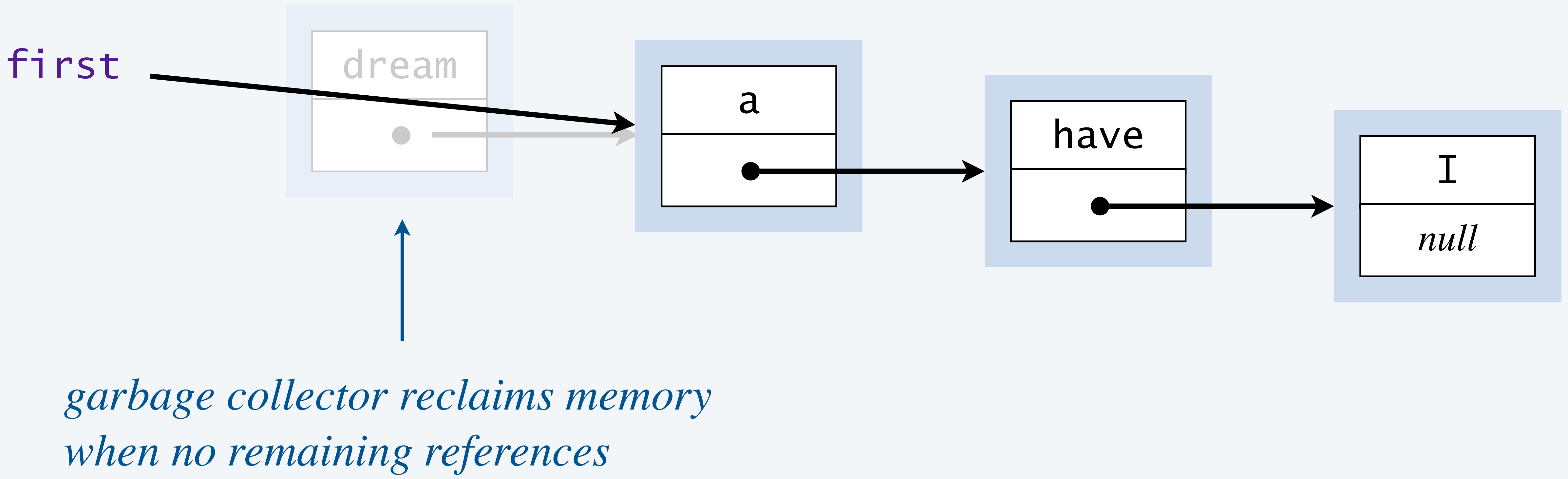
Node object

save item to return

```
String item = first.item;
```

delete first node

```
first = first.next;
```



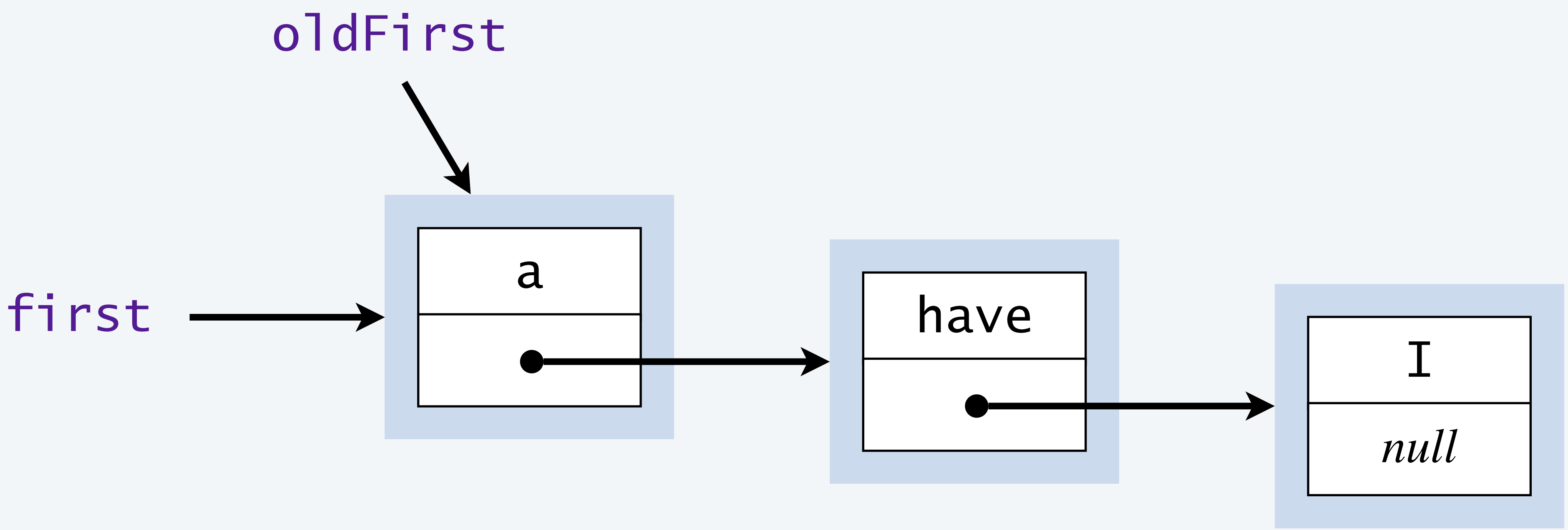
return saved item

```
return item;
```

Stack implementation with a linked list: push

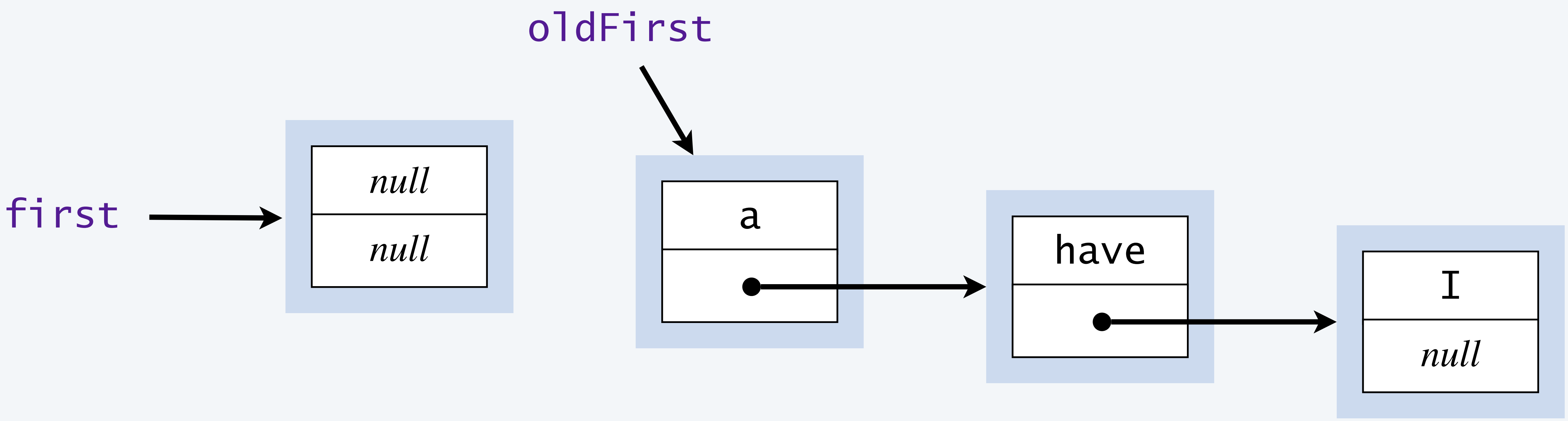
save a link to the list

```
Node oldFirst = first;
```



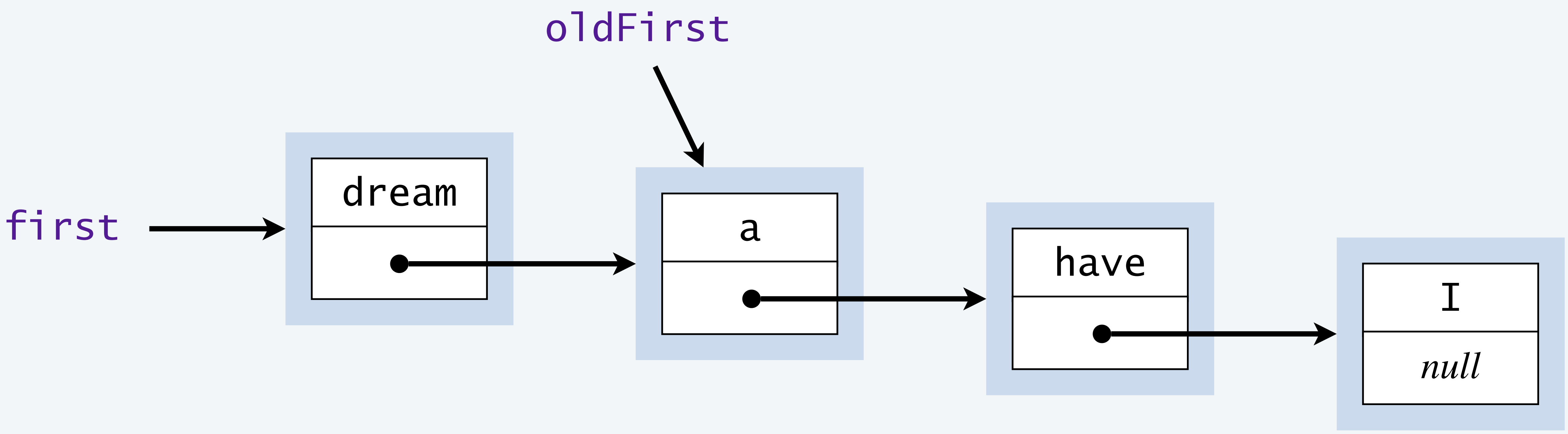
create a new node at the front

```
first = new Node();
```

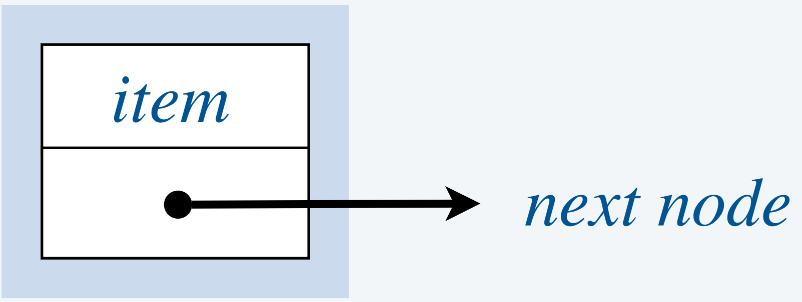


initialize the instance variables in the new Node

```
first.item = "dream";
first.next = oldFirst;
```




```
public class Node {
   private String item;
   private Node next;
}
```



Node object

Stack: linked-list implementation

```
public class LinkedStack<Item> {
   private Node first = null;

   private class Node {
      private Item item;
      private Node next;
   }

   public boolean isEmpty() {
      return first == null;
   }

   public void push(Item item) {
      Node oldFirst = first;
      first = new Node();
      first.item = item;
      first.next = oldFirst;
   }

   public Item pop() {
      Item item = first.item;
      first = first.next;
      return item;
   }
}
```

use generics

private nested class
(access modifiers for instance variables of such a class don't matter)

no `Node` *constructor defined explicitly* ⇒
Java supplies a default no-argument constructor
(which initializes instance variables to default values)

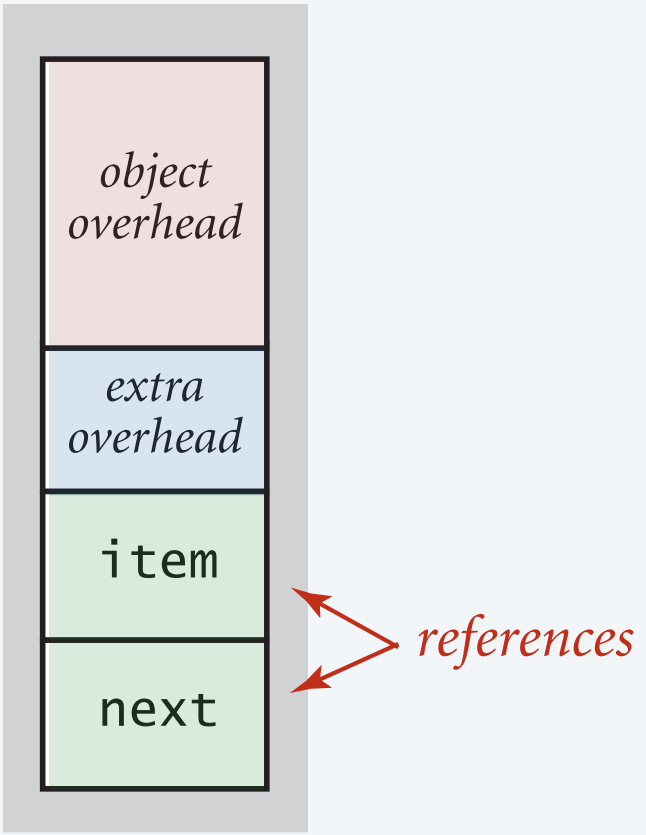
Stack: linked-list implementation performance

Proposition. Every operation takes $\Theta(1)$ time.

Proposition. A `LinkedStack` with n items has n `Node` objects and uses $\sim 40\,n$ bytes.

```
private class Node {
   private Item item;
   private Node next;
}
```

nested class



16 *bytes (object overhead)*

8 *bytes (non-static nested class extra overhead)*

8 *bytes (reference to* `Item`*)*

8 *bytes (reference to* `Node`*)*

40 *bytes per stack* `Node`

Remark. This counts the memory for the stack itself, including the string references.

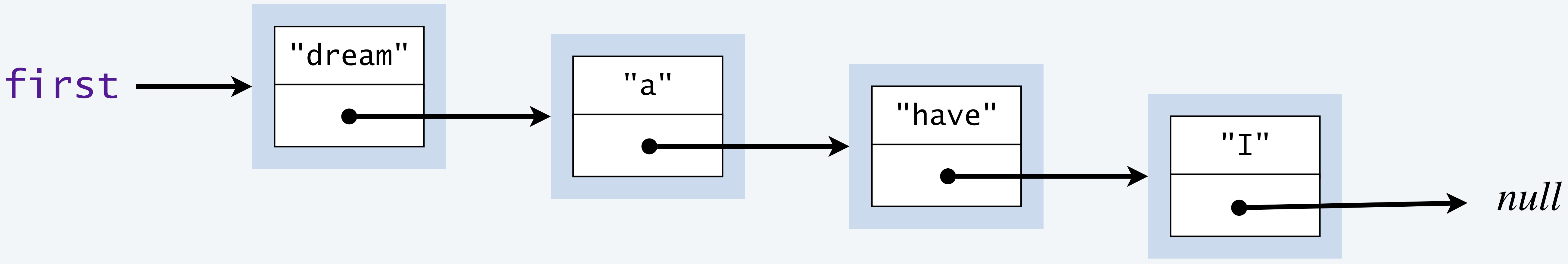
[but not the memory for the string objects, which the client allocates]

Stack implementations: resizable array vs. linked list

Tradeoffs. Can implement a stack with either a resizable array or a linked list; client can use either.

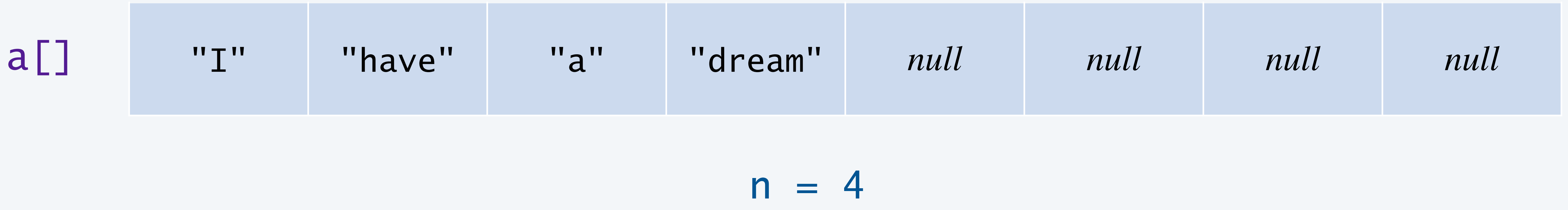
Q. Which is more efficient?

A. It depends.



Linked-list implementation.

- $\Theta(1)$ worst-case performance guarantee.
- More memory.



Resizable-array implementation.

- $\Theta(1)$ amortized performance guarantee.
- Less memory.
- Better use of cache.

accessing nearby memory locations (e.g., in an array) is much faster than accessing scattered memory locations (e.g., in a linked list)

https://algs4.cs.princeton.edu

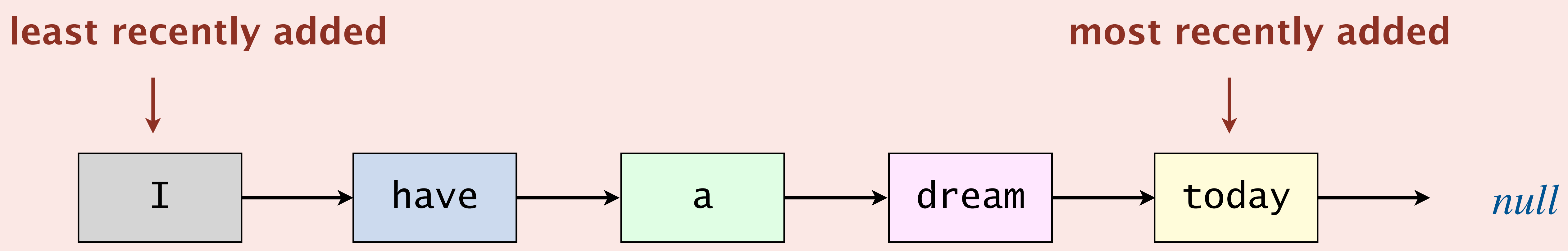
1.3 Stacks and Queues II

- linked lists
- stack implementation
- **queue implementation**
- iterators
- Java collections

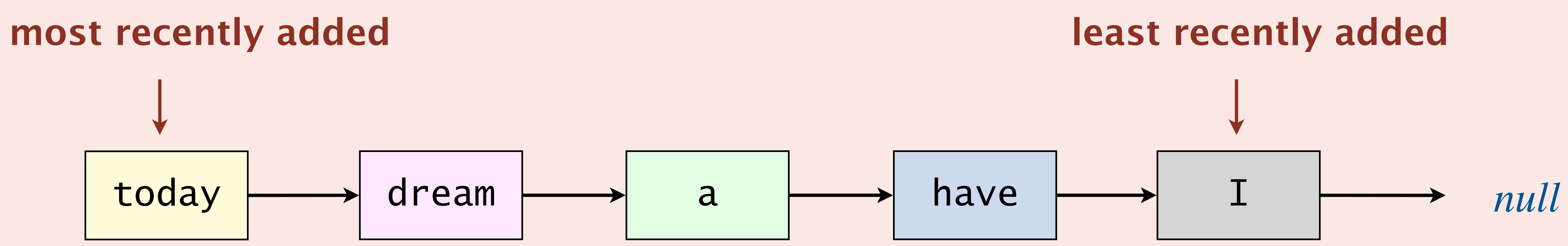
Stacks and queues II: poll 3

How to implement efficiently a queue with a singly linked list?

A.



B.

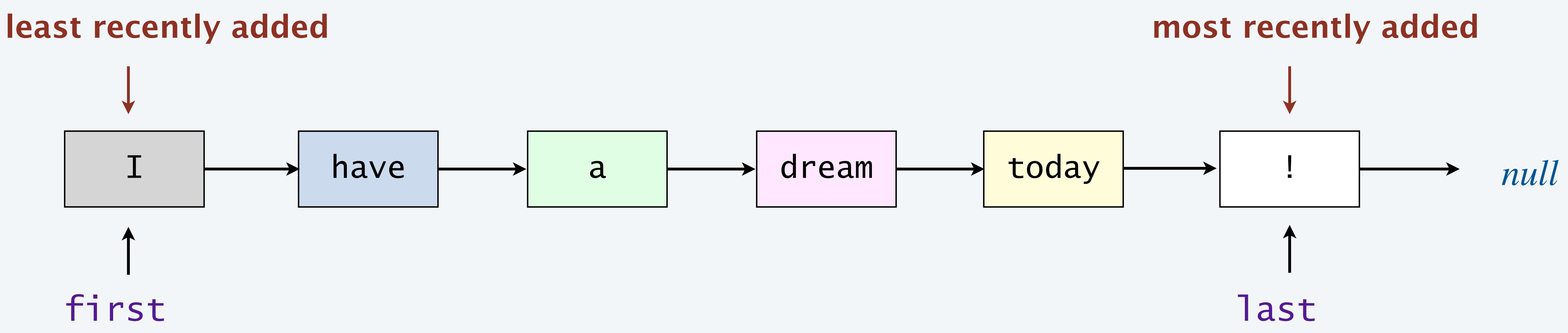


C. *Both A and B.*

D. *Neither A nor B.*

Queue: linked-list implementation

- Maintain one link `first` to first node in a singly linked list.
- Maintain another link `last` to last node.
- Dequeue from `first`.
- Enqueue after `last`.



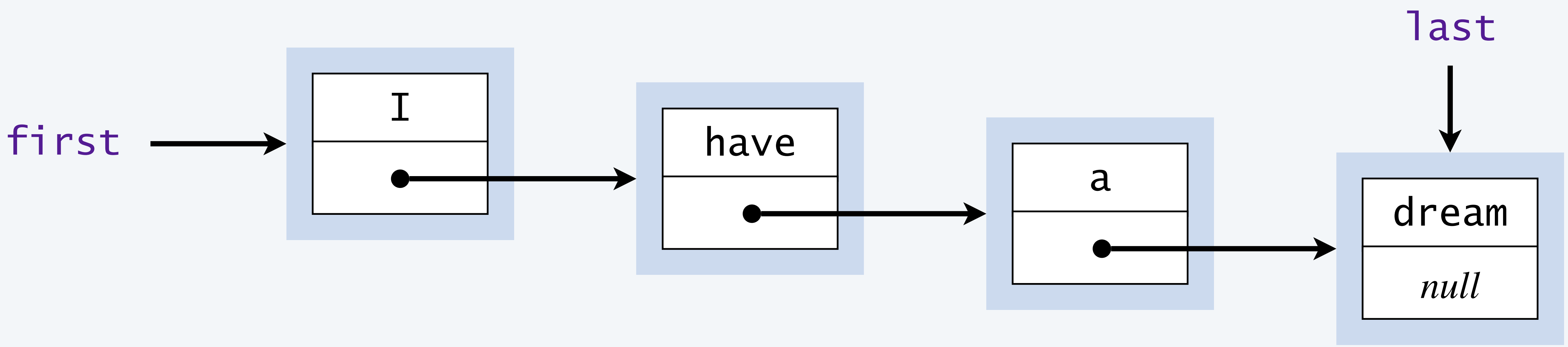
Queue dequeue: linked-list implementation

Remark. Code is identical to `pop()`.

```
public class Node {
   private String item;
   private Node next;
}
```

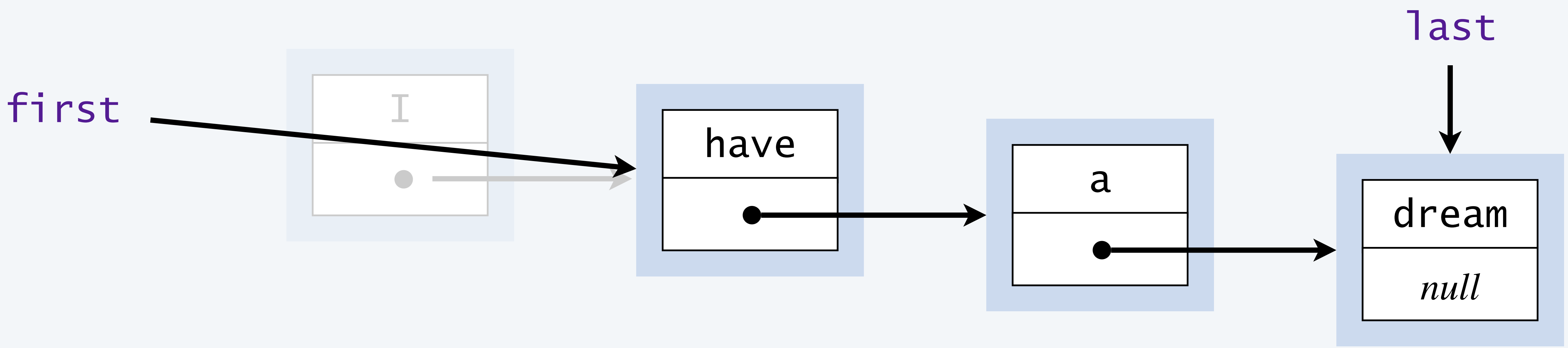
nested class

singly linked list



save item to return

```
String item = first.item;
```



delete first node

```
first = first.next;
```

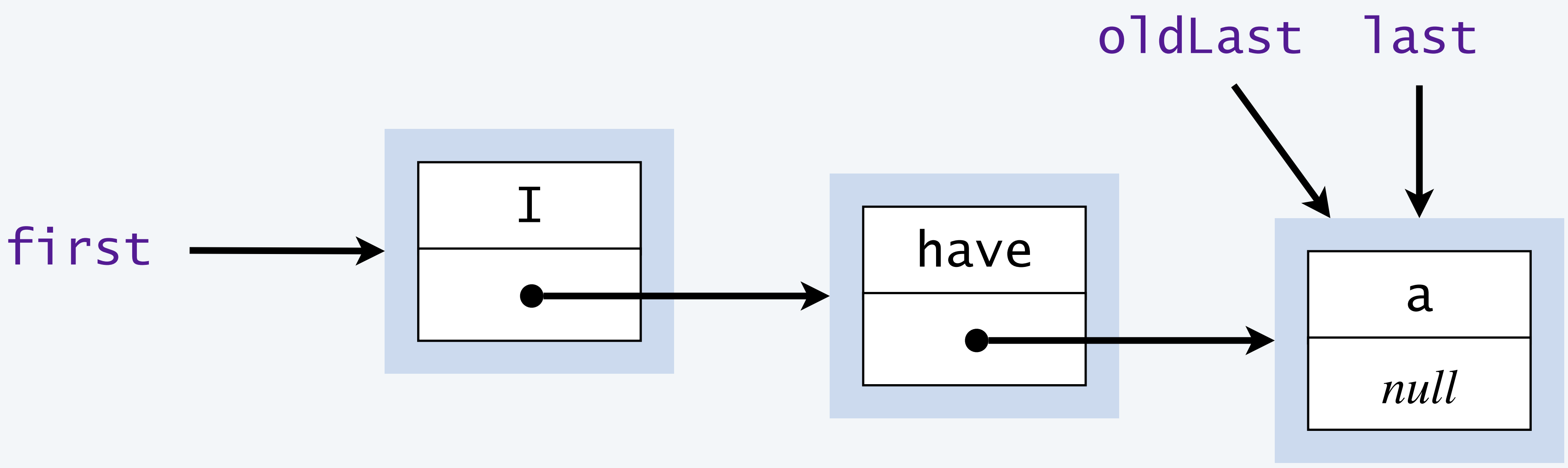
return saved item

```
return item;
```

Queue enqueue: linked-list implementation

save a link to the last node

```
Node oldLast = last;
```

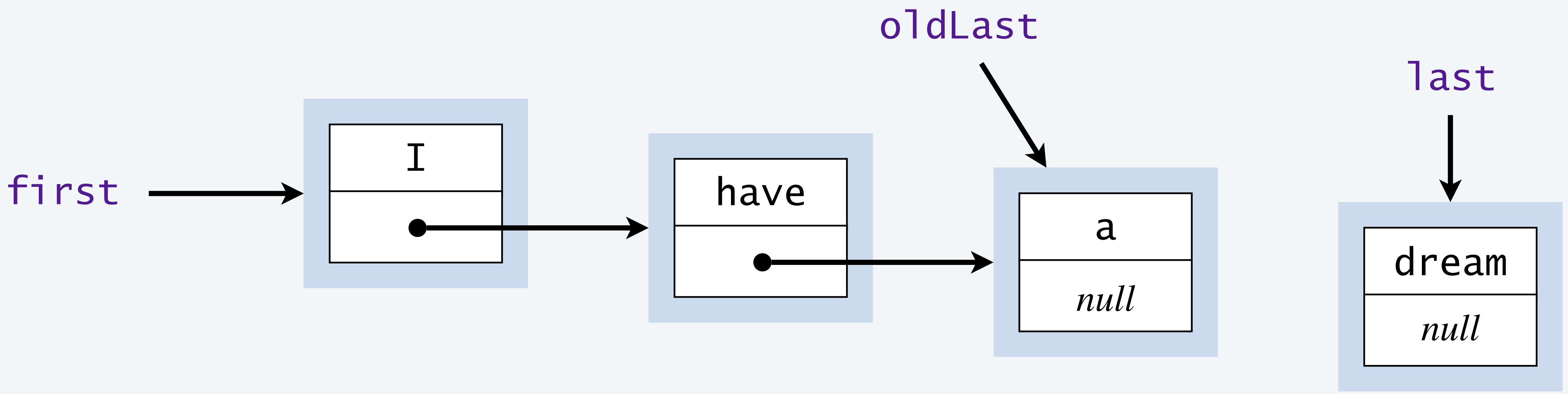



```
public class Node {
   private String item;
   private Node next;
}
```

nested class

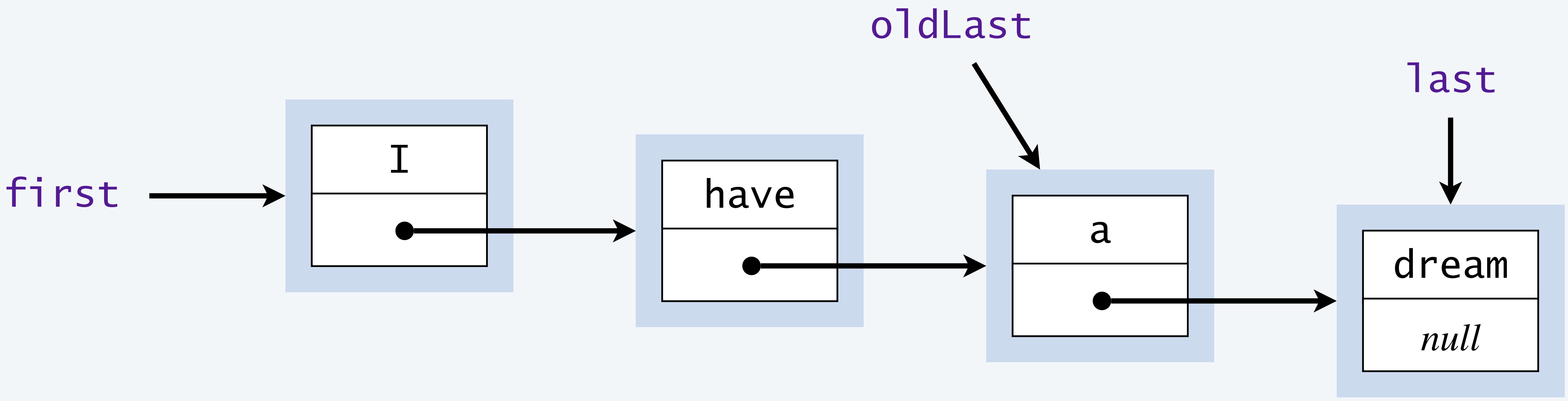
create a new node at the end

```
last = new Node();
last.item = "dream";
```



link together

```
oldLast.next = last;
```



Queue: linked-list implementation

```
public class LinkedQueue<Item> {
   private Node first, last;

   private class Node {
      /* identical to LinkedStack */
   }

   public boolean isEmpty() {
      return first == null;
   }

   public void enqueue(Item item) {
      Node oldLast = last;
      last = new Node();
      last.item = item;
      last.next = null;
      if (isEmpty()) first = last;
      else           oldLast.next = last;
   }

   public Item dequeue() {
      Item item = first.item;
      first     = first.next;
      if (isEmpty()) last = null;
      return item;
   }
}
```

corner case: add to an empty queue (don't forget to update `first`*)*

corner case: remove down to an empty queue (avoid loitering)

https://algs4.cs.princeton.edu

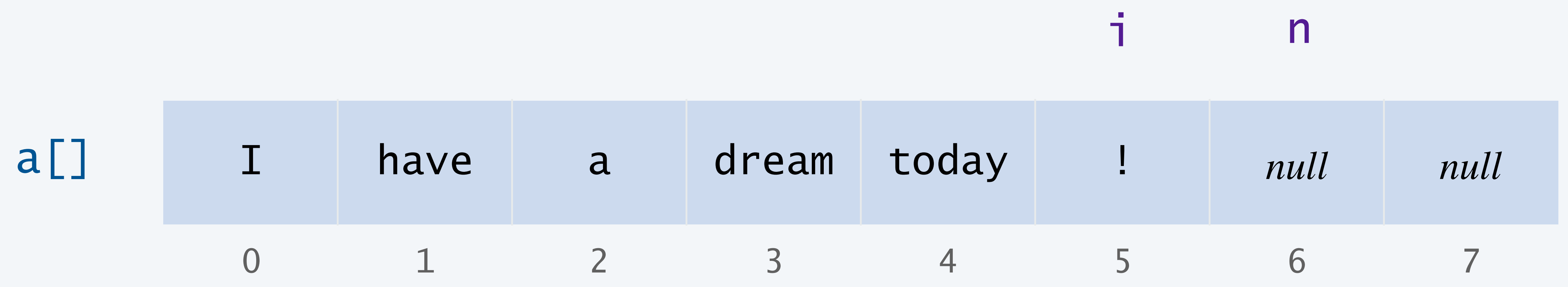
1.3 Stacks and Queues II

- linked lists
- stack implementation
- queue implementation
- ***iterators***
- Java collections

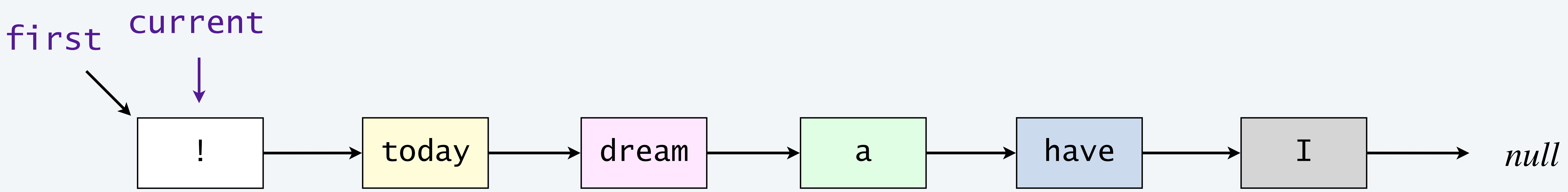
Iteration

Design challenge. Allow a client to access sequentially (iterate over) the items in a collection, without exposing the collection's internal representation.

stack (resizable-array representation)



stack (linked-list representation)



Java solution. Use a foreach loop.

Foreach loop

Java provides elegant syntax for iterating over the items in a collection.

"foreach" loop (shorthand)

```
Stack<String> stack = new Stack<>();
...

for (String s : stack) {
    // do something with s
}
```

equivalent code (longhand)

```
Stack<String> stack = new Stack<>();
...

Iterator<String> iterator = stack.iterator();
while (iterator.hasNext()) {
   String s = iterator.next();
   // do something with s
}
```

To provide clients the ability to iterate with a foreach loop:

- Collection must have a method `iterator()`, which returns an `Iterator` object.
- An `Iterator` object represents the state of a traversal. ← *e.g., current spot in sequence*
  - the `hasNext()` returns `true` unless the traversal is complete
  - the `next()` method returns the next item in the traversal

Iterator and Iterable interfaces

Java defines two interfaces that facilitate foreach loops. ← *Java interface = set of related methods that define some behavior (partial API)*

- `Iterable` interface: `iterator()` method that returns an `Iterator`.
- `Iterator` interface: `next()` and `hasNext()` methods.
- Each interface is parameterized using generics.

java.lang.Iterable interface

```
public interface Iterable<Item> {
   Iterator<Item> iterator();
}
```

"I am a collection that can be traversed with a foreach loop."

java.util.Iterator interface

```
public interface Iterator<Item> {
   boolean hasNext();
   Item next();
}
```

"I represent the state of one traversal."

Type safety. Foreach loop won't compile unless collection is `Iterable` (or an array). ← *ensures that the (implicit) call to* `iterator()` *will succeed at run time*

Stack iterator: resizable-array implementation

```
import java.util.Iterator;
import java.util.NoSuchElementException;

public class ResizableArrayStack<Item>implements Iterable<Item>  {
   private int n;      // number of items in the stack
   private Item[] a;   // stack items
   ...

   public Iterator<Item> iterator() {
      return new ReverseArrayIterator();
   }

   private class ReverseArrayIterator implements Iterator<Item> {
      private int i = n-1;   // index of next item to return

      public boolean hasNext() {
         return i >= 0;
      }

      public Item next() {
         if (!hasNext()) throw new NoSuchElementException();
         return a[i--];
      }
   }
}
```

collection implements the `Iterable` *interface*

object you return must implements the `Iterator` *interface*

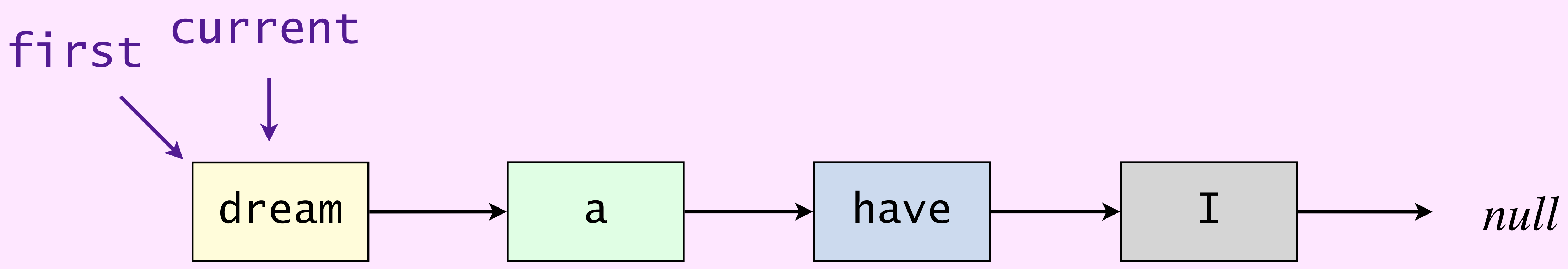
code in inner class can access instance variables in outer class

`Iterator` *API says to throw this exception if called after traversal is complete*

						i	n	
a[]	I	have	a	dream	today	!	*null*	*null*
	0	1	2	3	4	5	6	7

Stack iterator: linked-list implementation (in IntelliJ)

```
import java.util.Iterator;
import java.util.NoSuchElementException;

public class LinkedStack<Item> implements Iterable<Item>  {
   private Node first;
   ...

   public Iterator<Item> iterator() {
      return new LinkedIterator();
   }

   private class LinkedIterator implements Iterator<Item> {
      private Node current = first;

      public boolean hasNext() {
         return current != null;
      }

      public Item next() {
         if (!hasNext()) throw new NoSuchElementException();
         Item item = current.item;
         current   = current.next;
         return item;
      }
   }
}
```



Suppose that you add A, B, and C to a stack (linked list or resizable array), in that order.

What does the following code fragment do?

```
for (String s : stack)
   for (String t : stack)
      StdOut.println(s + "-" + t);
```

A. Prints A-A A-B A-C B-A B-B B-C C-A C-B C-C

B. Prints C-C C-B C-A B-C B-B B-A A-C A-B A-A

C. Run-time exception.

D. Depends on the implementation.

Stacks and queues II: poll 5

Suppose that you add A, B, and C to a stack (linked list or resizable array), in that order.

What does the following code fragment do?

```
for (String s : stack) {
   StdOut.println(s);
   StdOut.println(stack.pop());
   stack.push(s);
}
```

modifies stack

A. Prints C C B B A A

B. Prints C C B C A B

C. Prints C C C C C C C C ...

D. Run-time exception.

E. Depends on the implementation.

Iteration: concurrent modification

Q. What should happen if a client modifies a collection while traversing it?

A. A fail-fast iterator throws a `java.util.ConcurrentModificationException`.

concurrent modification

```
for (String s : stack)
   stack.push(s);
```

Java iterators summary

Iterator and Iterable. Two Java interfaces that allow a client to iterate over the items in a collection, without exposing the collection's internal representation.

```
Stack<String> stack = new Stack<>();
...

for (String s : stack) {
  ...
}
```

This course.

- Yes: use iterators in client code.
- Yes: implement iterators (Assignment 2 only).

https://algs4.cs.princeton.edu

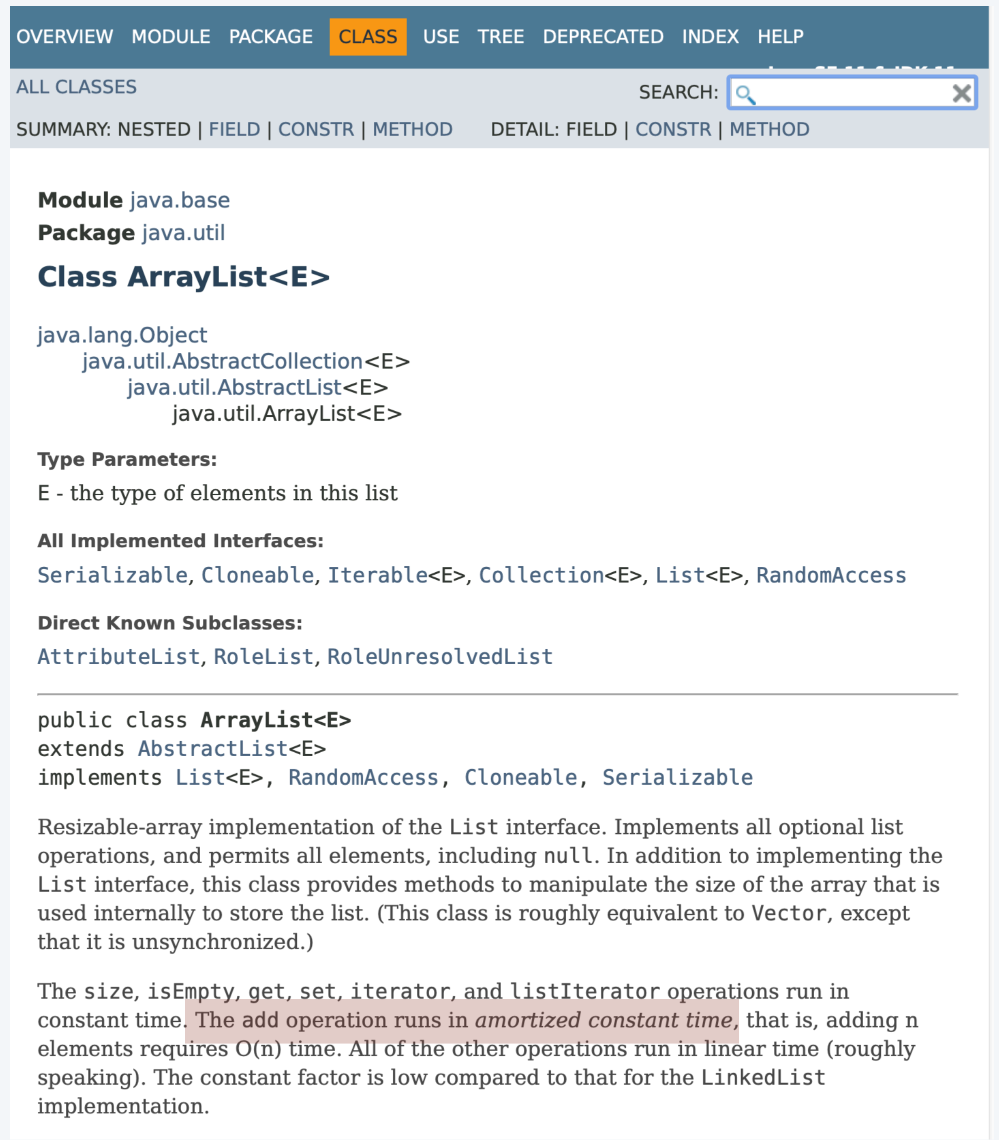
1.3 STACKS AND QUEUES II

- *linked lists*
- *stack implementation*
- *queue implementation*
- *iterators*
- ***Java collections***

Java collections framework

Java's libraries for collection data types.

- `java.util.LinkedList` [doubly linked list]
- `java.util.ArrayList` [resizable array]
- `java.util.TreeMap` [red-black BST]
- `java.util.HashMap` [hash table]



OVERVIEW MODULE PACKAGE CLASS USE TREE DEPRECATED INDEX HELP
ALL CLASSES SEARCH:
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Module java.base
Package java.util

Class ArrayList<E>

java.lang.Object
java.util.AbstractCollection<E>
java.util.AbstractList<E>
java.util.ArrayList<E>

Type Parameters:
E - the type of elements in this list

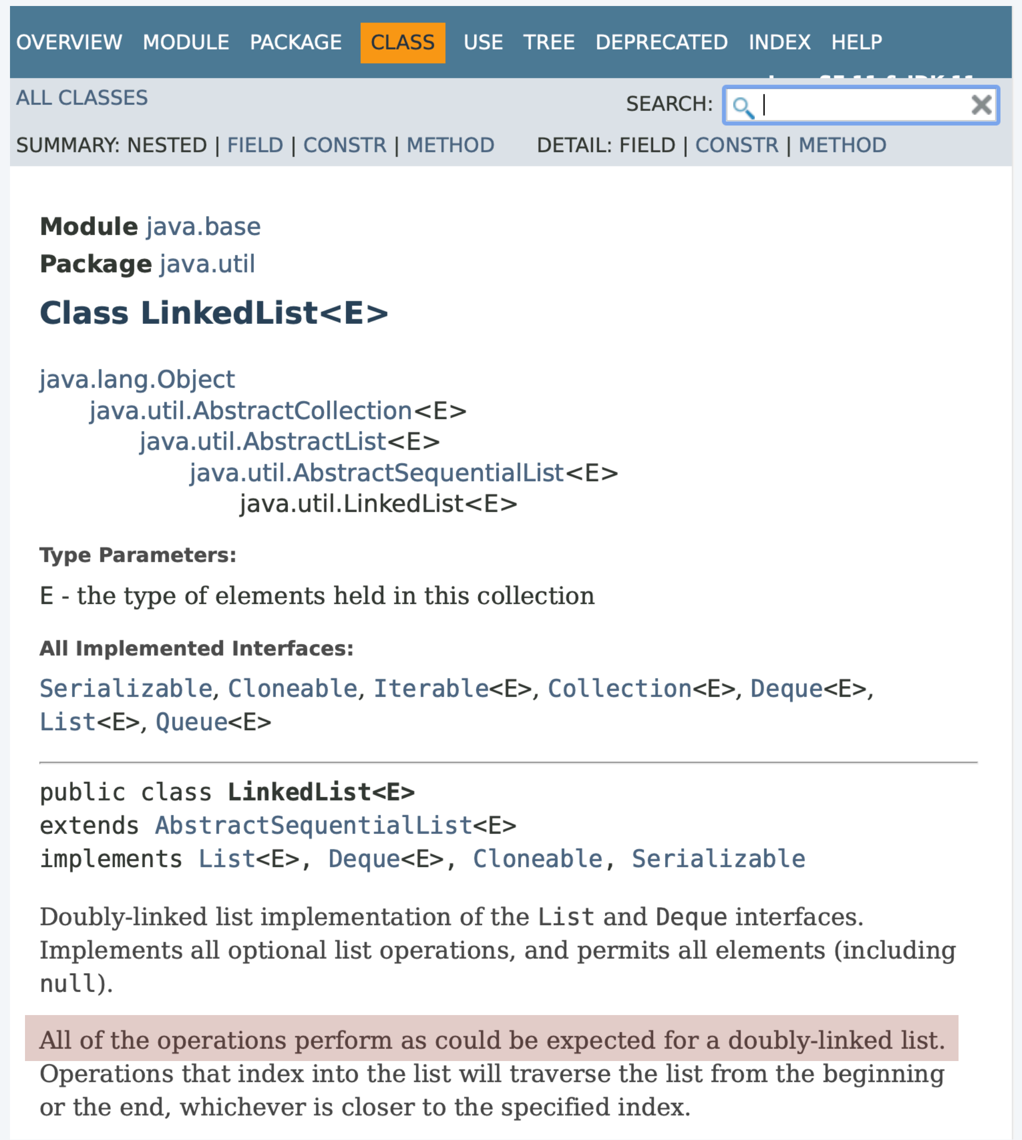
All Implemented Interfaces:
Serializable, Cloneable, Iterable<E>, Collection<E>, List<E>, RandomAccess

Direct Known Subclasses:
AttributeList, RoleList, RoleUnresolvedList

```
public class ArrayList<E>
extends AbstractList<E>
implements List<E>, RandomAccess, Cloneable, Serializable
```

Resizable-array implementation of the List interface. Implements all optional list operations, and permits all elements, including null. In addition to implementing the List interface, this class provides methods to manipulate the size of the array that is used internally to store the list. (This class is roughly equivalent to Vector, except that it is unsynchronized.)

The size, isEmpty, get, set, iterator, and listIterator operations run in constant time. The add operation runs in *amortized constant time*, that is, adding n elements requires O(n) time. All of the other operations run in linear time (roughly speaking). The constant factor is low compared to that for the LinkedList implementation.



OVERVIEW MODULE PACKAGE CLASS USE TREE DEPRECATED INDEX HELP
ALL CLASSES SEARCH:
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Module java.base
Package java.util

Class LinkedList<E>

java.lang.Object
java.util.AbstractCollection<E>
java.util.AbstractList<E>
java.util.AbstractSequentialList<E>
java.util.LinkedList<E>

Type Parameters:
E - the type of elements held in this collection

All Implemented Interfaces:
Serializable, Cloneable, Iterable<E>, Collection<E>, Deque<E>, List<E>, Queue<E>

```
public class LinkedList<E>
extends AbstractSequentialList<E>
implements List<E>, Deque<E>, Cloneable, Serializable
```

Doubly-linked list implementation of the List and Deque interfaces. Implements all optional list operations, and permits all elements (including null).

All of the operations perform as could be expected for a doubly-linked list. Operations that index into the list will traverse the list from the beginning or the end, whichever is closer to the specified index.

This course. Implement from scratch (once).

Beyond. Basis for understanding performance guarantees.

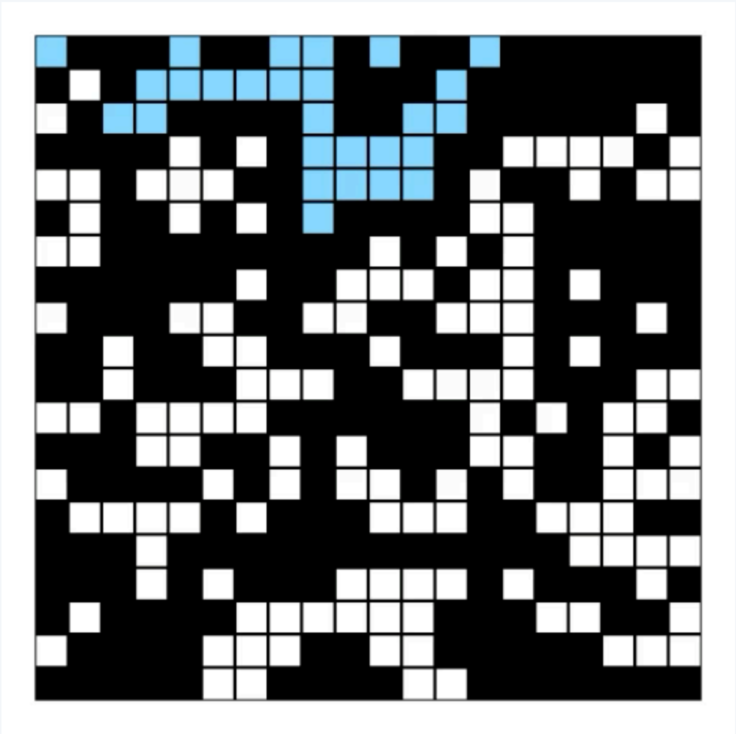
Best practices.

- Use `Stack` and `Queue` in `algs4.jar` for stacks and queues to improve design and efficiency.
- Use `java.util.ArrayList` or `java.util.LinkedList` when other ops needed.

  (but remember that some ops are inefficient)

COS 226 story (from Assignment 1)

Goal. Generate random open sites in an n-by-n percolation system and repeat until system percolates.



Jenny.

- Pick $(row, \ col)$ at random; if already open, repeat.
- Takes $\Theta(n^2)$ time.

Kenny.

- Create a `java.util.ArrayList` to store the n^2 blocked sites.
- Pick an index at random and delete.
- Takes $\Theta(n^4)$ time.

Lesson. Don't use a library until you understand its API!

This course. Can't use a library until we've implemented it in class.

Stacks and queues summary

Fundamental data types.

- Value: collection of objects.
- Operations: add, remove, iterate, size, test if empty.

Stack. [LIFO] Remove the item most recently added.

Queue. [FIFO] Remove the item least recently added.

Efficient implementations.

- Resizable array.
- Singly linked list.

Credits

image	source	license
Assignment Logo	Kathleen Ma '18	by author
Stack of Books	Adobe Stock	Education License
Long Queue Line	Adobe Stock	Education License
People Standing in Line	Adobe Stock	Education License
Stack of Sweaters	Adobe Stock	Education License
Programmer Icon	Jaime Botero	public domain
ChatGPT Phone	Adobe Stock	Education License

A final thought

“ *Linked lists, nodes connected with care,*

Arrays resizing, with memory to spare.

Organizing data, their only need,

Helping us, with efficiency indeed. ”