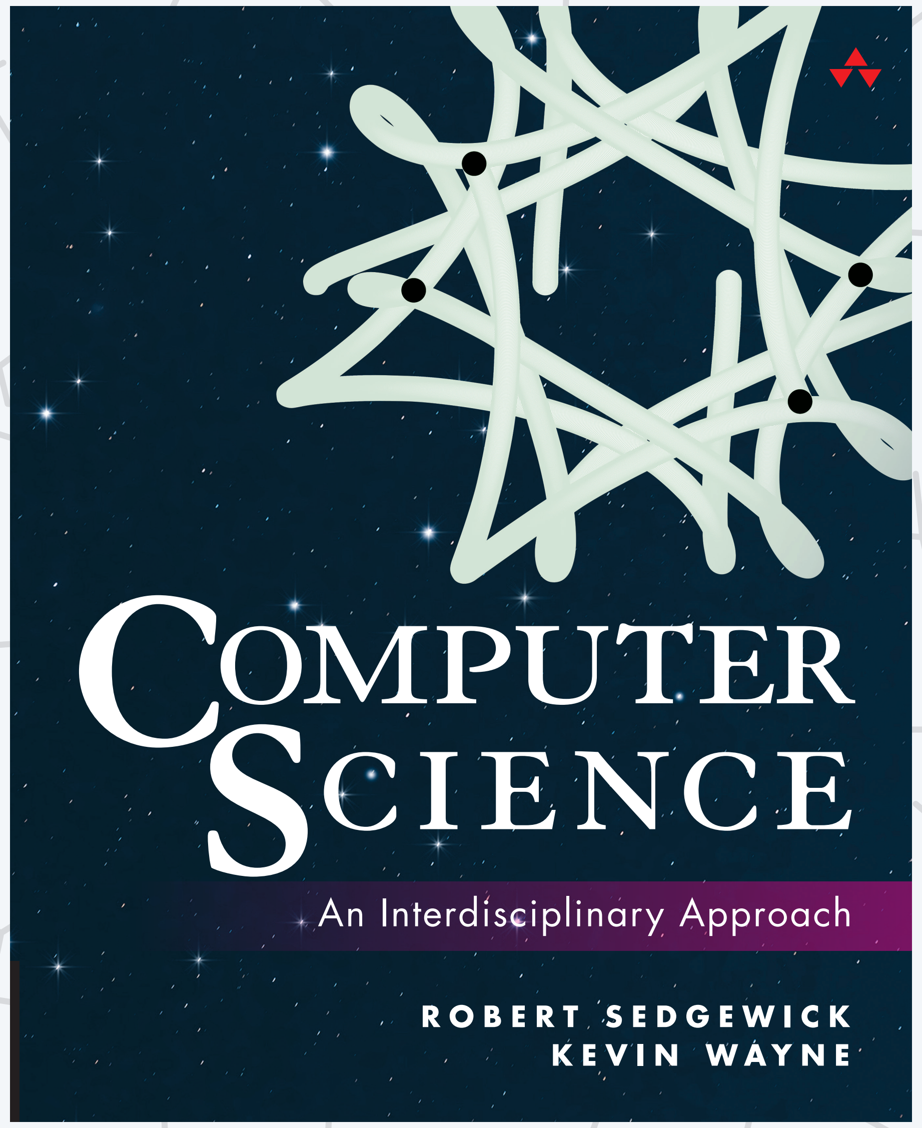
Computer Science

Robert Sedgewick | Kevin Wayne



https://introcs.cs.princeton.edu

4.3 Data Structures

- *collections*
- *stacks and queues*
- *linked lists*
- *symbol tables*
- *Java collections framework*

DRAFT

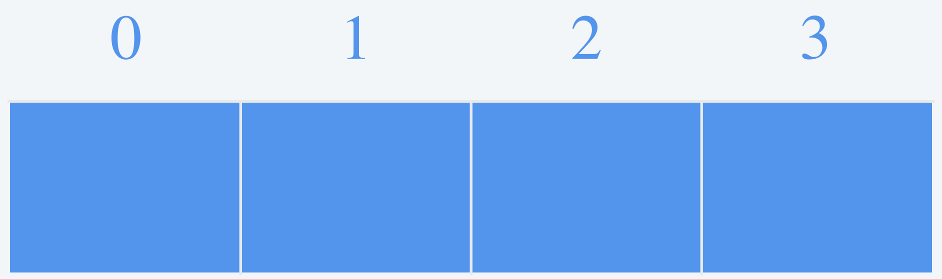
Last updated on 9/10/24 10:33AM

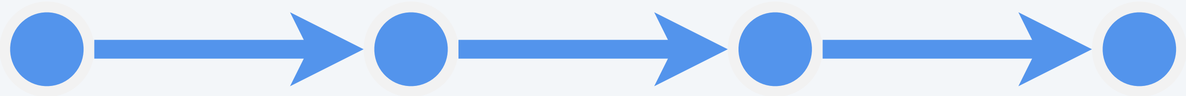
Data structures

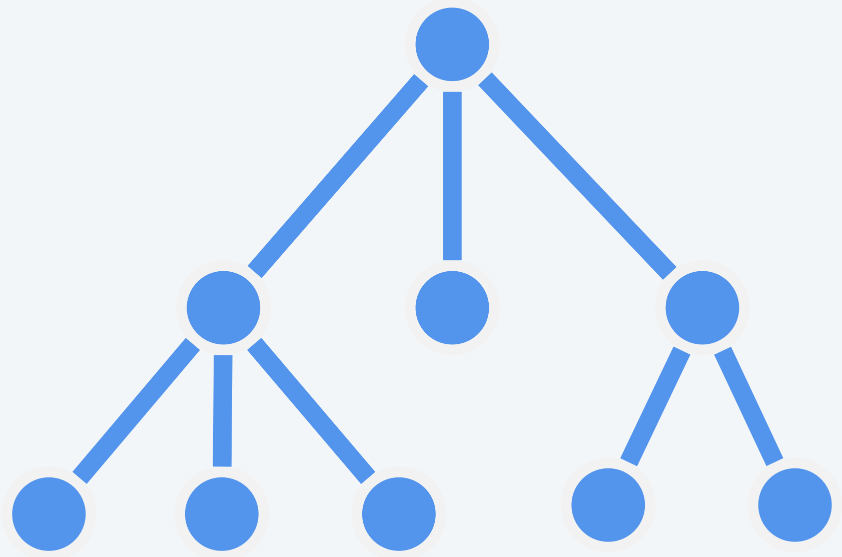
Data structure. Method for organizing data in a computer so that it can be accessed efficiently.

category	data structures
array	1D array, resizing array, binary heap, Bloom filter, ring buffer, …
linked list	singly linked list, doubly linked list, blockchain, …
tree	binary search tree, k-d tree, Merkle tree, B-tree, decision tree, …
composite	2D array, hash table, tensor, sparse matrix, graph, …

Guitar Hero



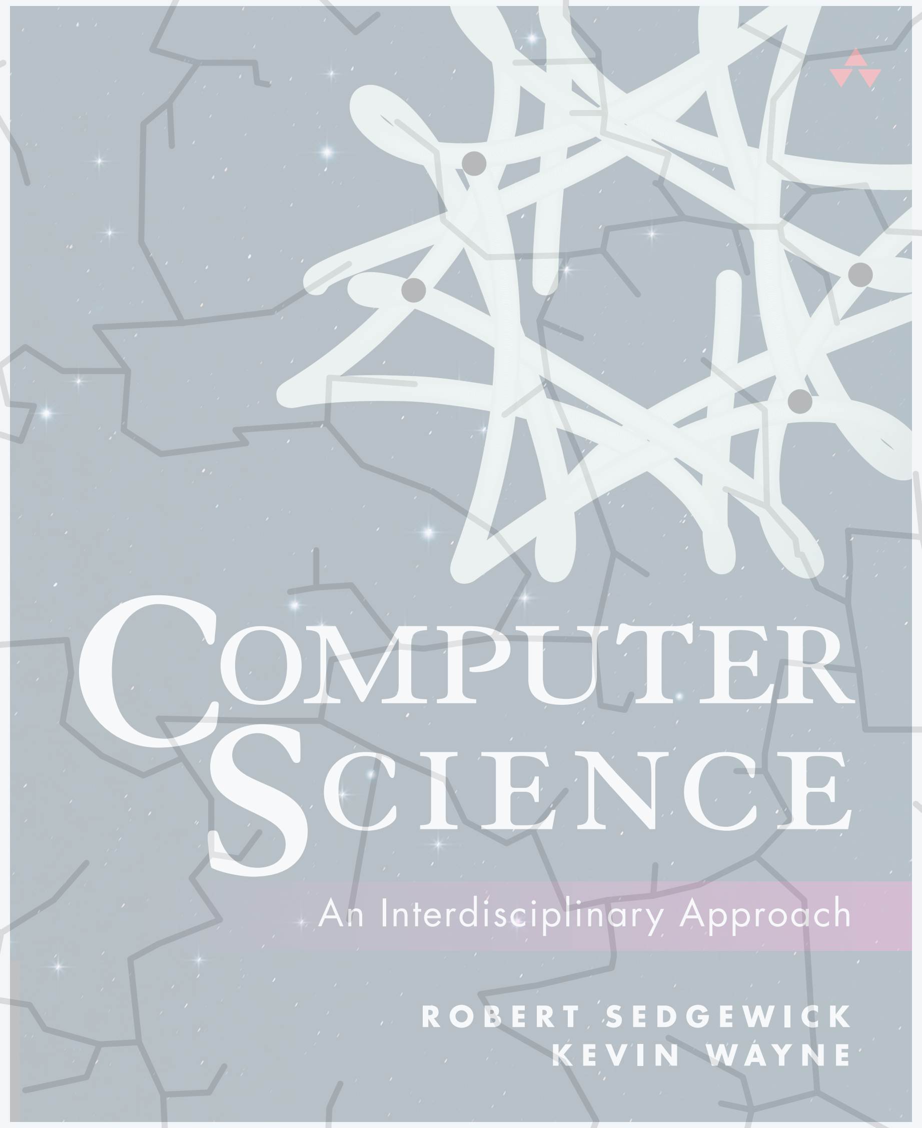




Collections

A collection is a data type that stores a group of related items.

collection	core operations	data structure
stack	Push, Pop	singly linked list resizing array
queue	Enqueue, Dequeue	
symbol table	Put, Get, Delete	binary search tree hash table
set	Add, Contains, Delete	
⋮	⋮	⋮



https://introcs.cs.princeton.edu

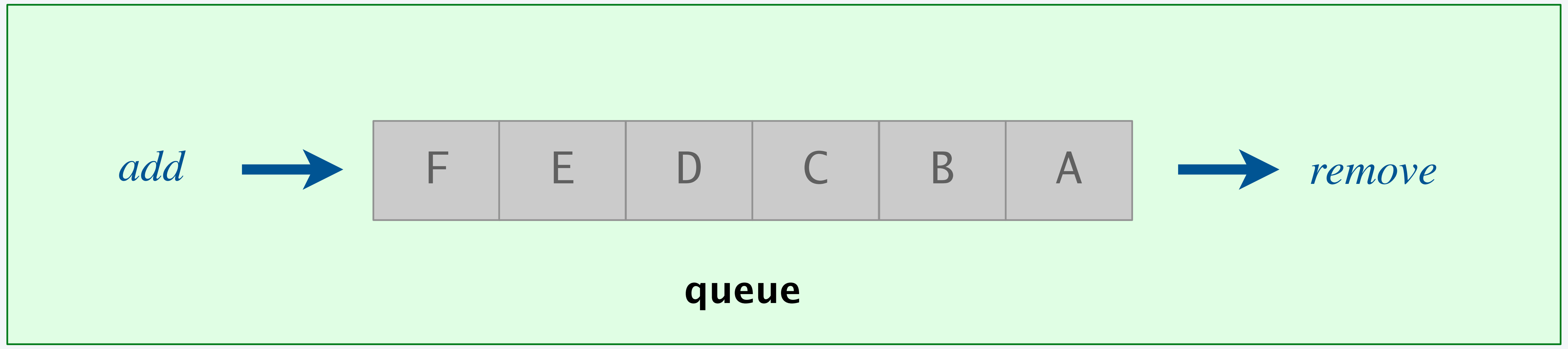
4.3 Data Structures

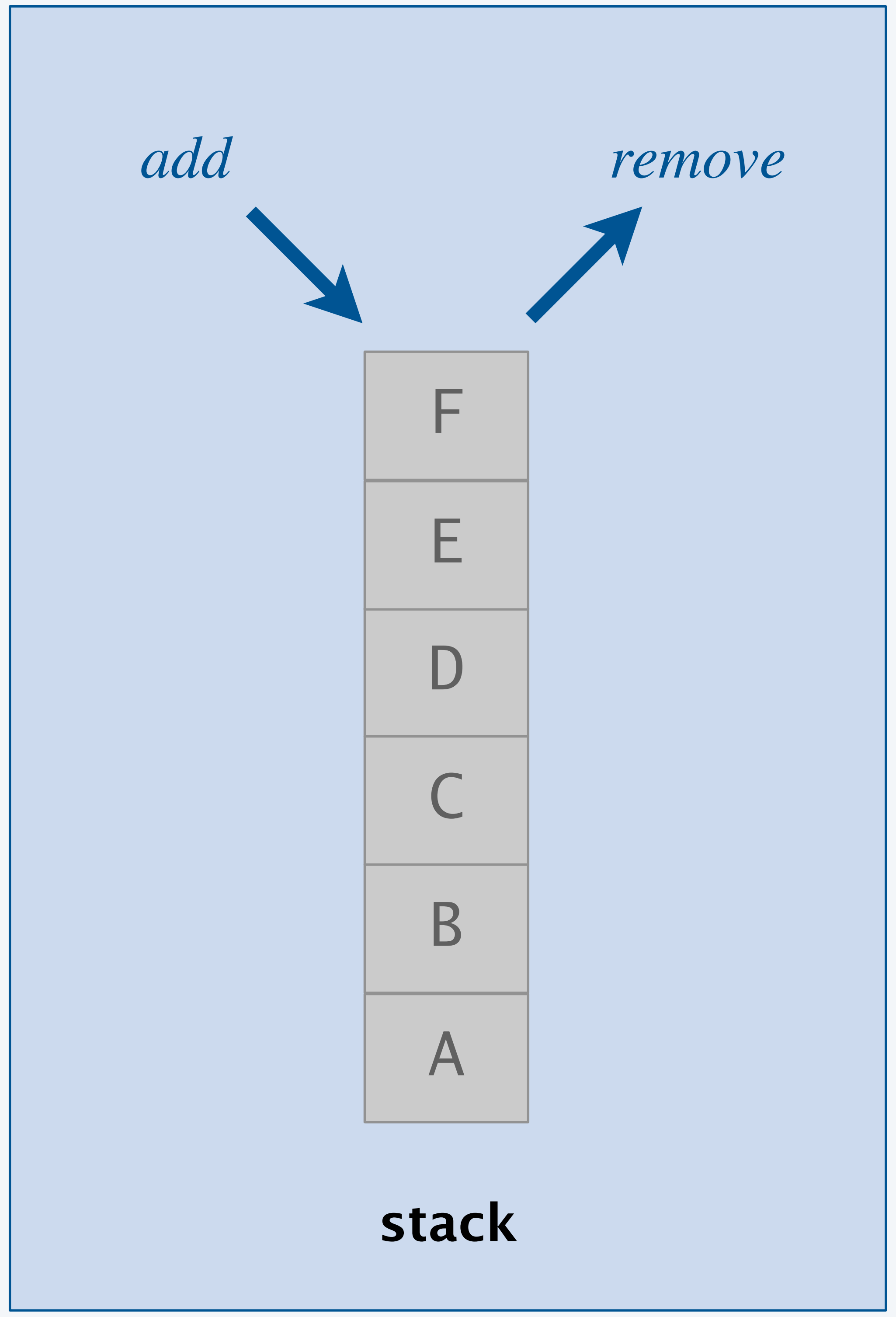
- collections
- stacks and queues
- linked lists
- symbol tables
- Java collections framework

Stacks and queues

Fundamental data types.

- Value: collection of objects.
- Operations: add, remove, iterate, size, test if empty.
- Intent is clear when we add.
- Which item do we remove?





Stack. Remove the item most recently added. ⟵ *LIFO = "last in first out"*

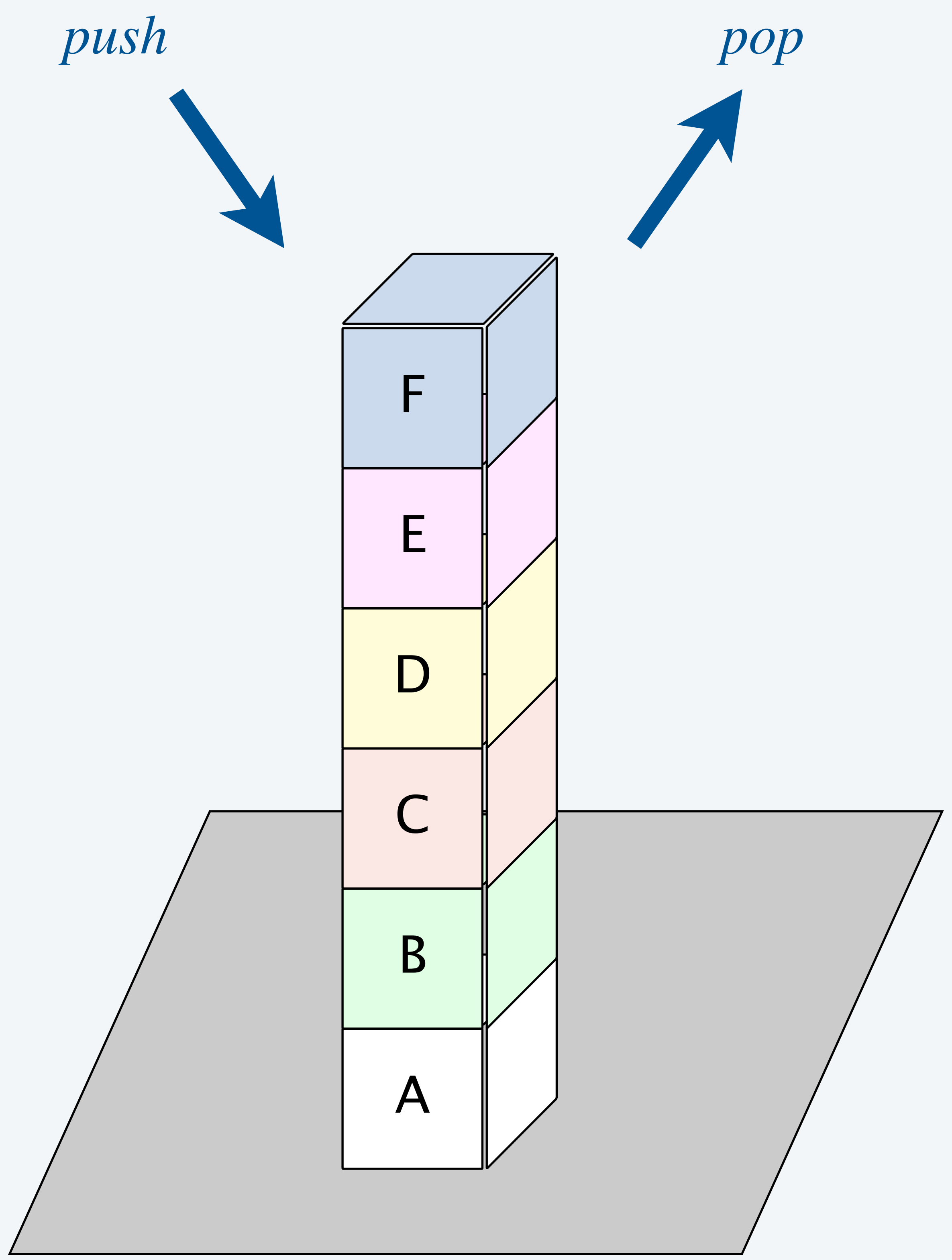
Queue. Remove the item least recently added. ⟵ *FIFO = "first in first out"*

Stack API

Stack data type. Our textbook data type for stacks. ← *available with* `javac-introcs` *and* `java-introcs` *commands*

“generic type parameter”

`public class Stack<Item>`		**description**
	`Stack()`	*create an empty stack*
`void`	`push(Item item)`	*add a new item to the stack*
`Item`	`pop()`	*remove and return the item most recently added*
`boolean`	`isEmpty()`	*is the stack empty?*
`int`	`size()`	*number of items on the stack*



Performance requirements. Every operation takes constant time.

Stack warmup client

Goal. Read strings from standard input and print in reverse order.

- Read strings from standard input and push onto stack.
- Pop all strings from stack and print.

```java
public class Reverse {
   public static void main(String[] args) {
      Stack<String> stack = new Stack<String>();
      while (!StdIn.isEmpty()) {
         String s = StdIn.readString();
         stack.push(s);
      }

      while (!stack.isEmpty()) {
         String s = stack.pop();
         StdOut.print(s + " ");
      }
      StdOut.println();
   }
}
```

"type argument" (can be any reference type)

create stack

push strings onto stack

pop strings from stack and print

```
~/cos126/ds> java-introcs Reverse
I have a dream today
<Ctrl-D>
today dream a have I
```

Data structures: quiz 1

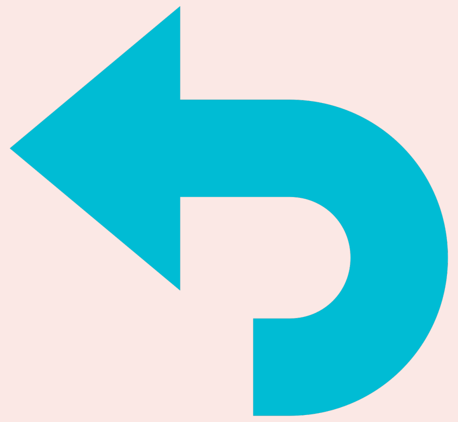
Which would not be implemented with a stack?

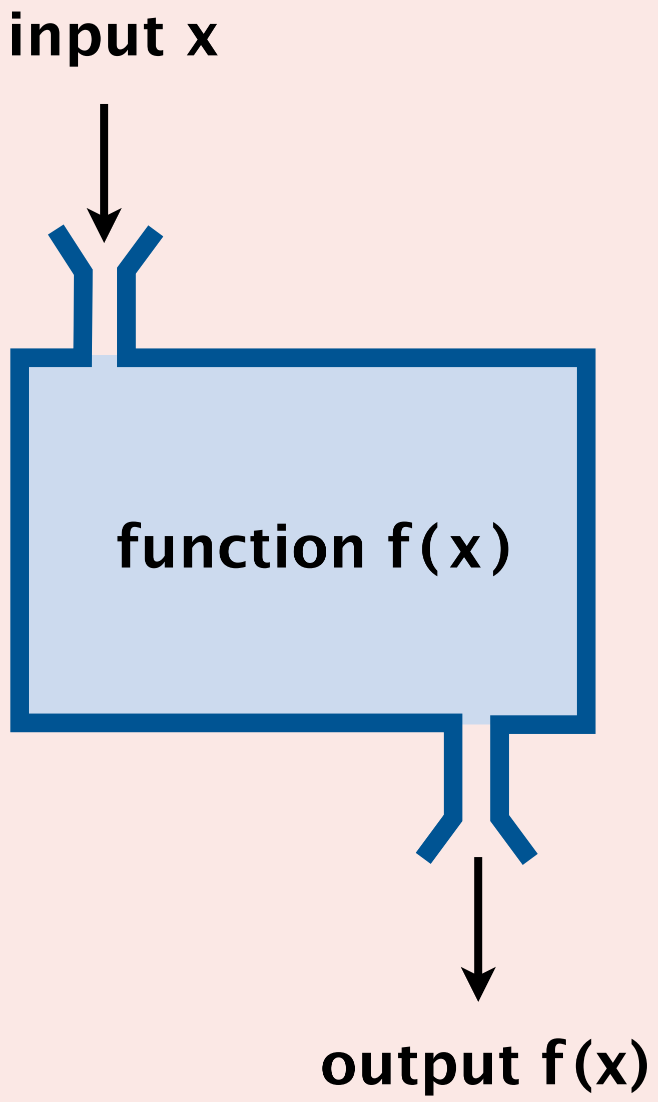
A. Back button in a browser.

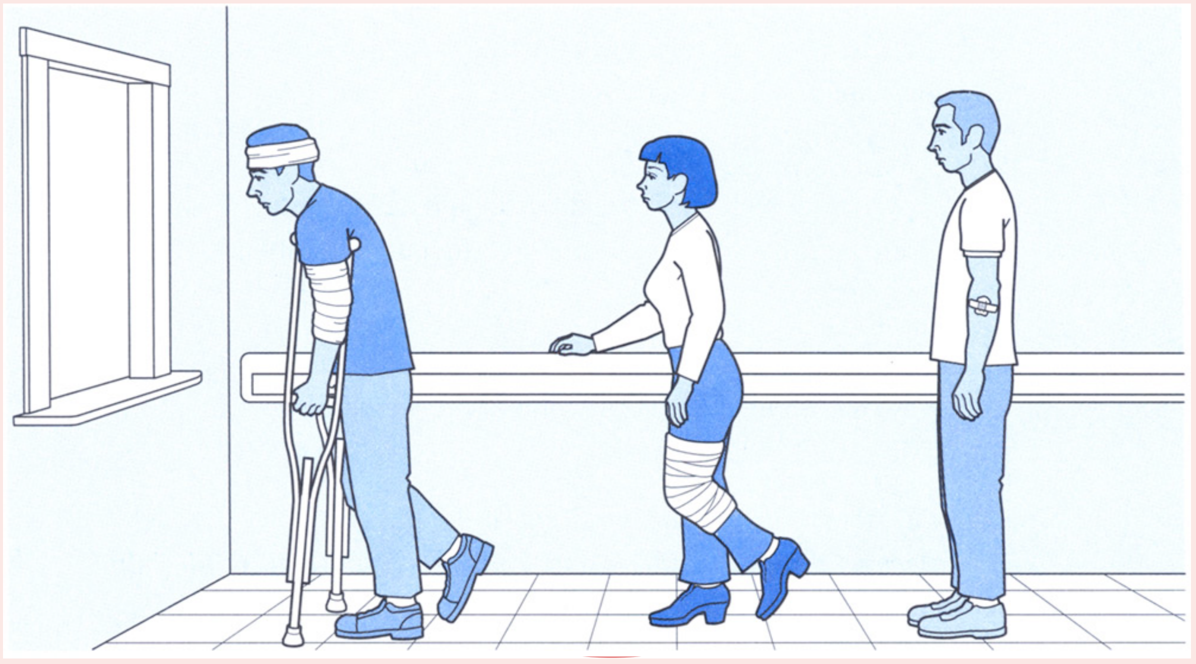
B. Undo in a word processor.

C. Function-call stack.

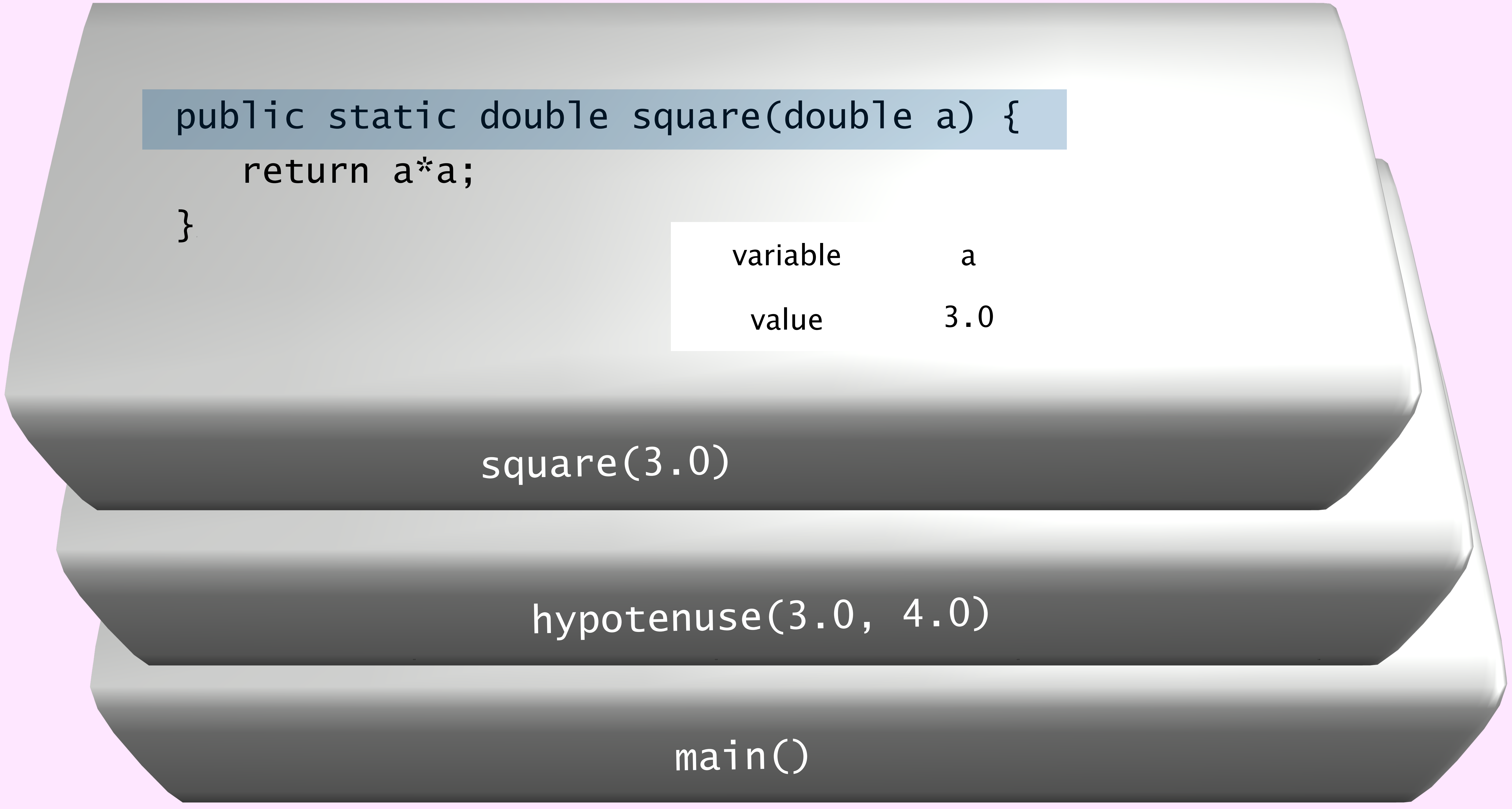
D. Triage in a hospital.







Function-call stack demo



```
public static double square(double a) {
   return a*a;
}
```

variable	a
value	3.0

function-call stack

Arithmetic expression evaluation

Goal. Write a program to evaluate infix expressions.

```
( 1 + ( ( 2 + 3 ) * ( 4 * 5 ) ) )
```

for simplicity, fully parenthesized and whitespace between elements

operand (value)

operator

Solution. Dijkstra's two-stack algorithm. [see demo]

Context. An interpreter!

a program that executes instructions (e.g., infix expressions) without compiling to machine language

Dijkstra's two-stack algorithm demo

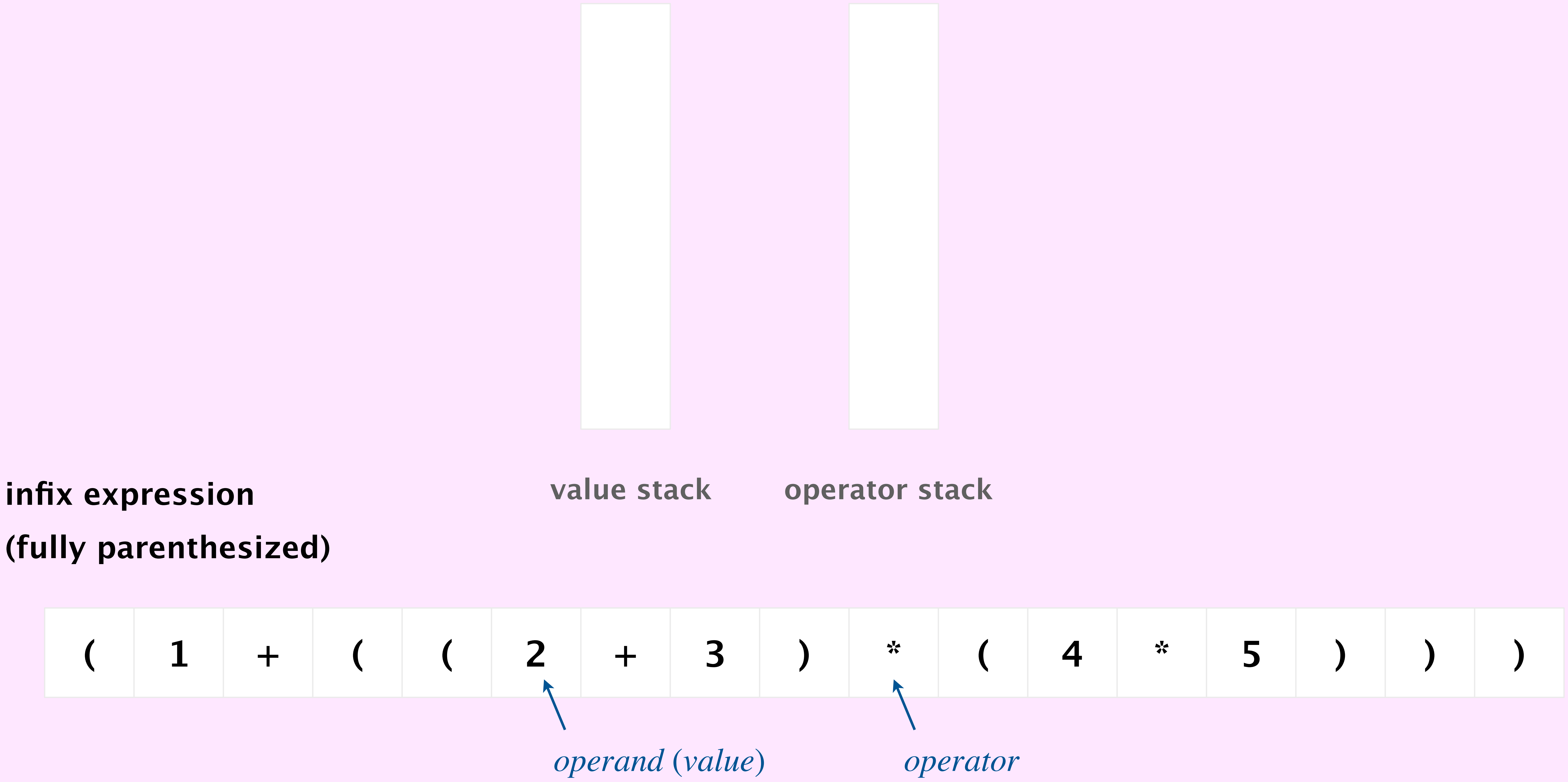
Value: push onto the value stack.

Operator: push onto the operator stack.

Left parenthesis: ignore.

Right parenthesis: pop operator and two values; push the result onto the value stack.

of applying that operator to those two values



infix expression
(fully parenthesized)

(	1	+	(	(	2	+	3	)	*	(	4	*	5	)	)	)

Data structures: quiz 2

How to declare and initialize a stack of integers in Java?

A. `Stack<int> stack = new Stack();`

B. `Stack<int> stack = new Stack<int>();`

C. `Stack stack = new Stack();`

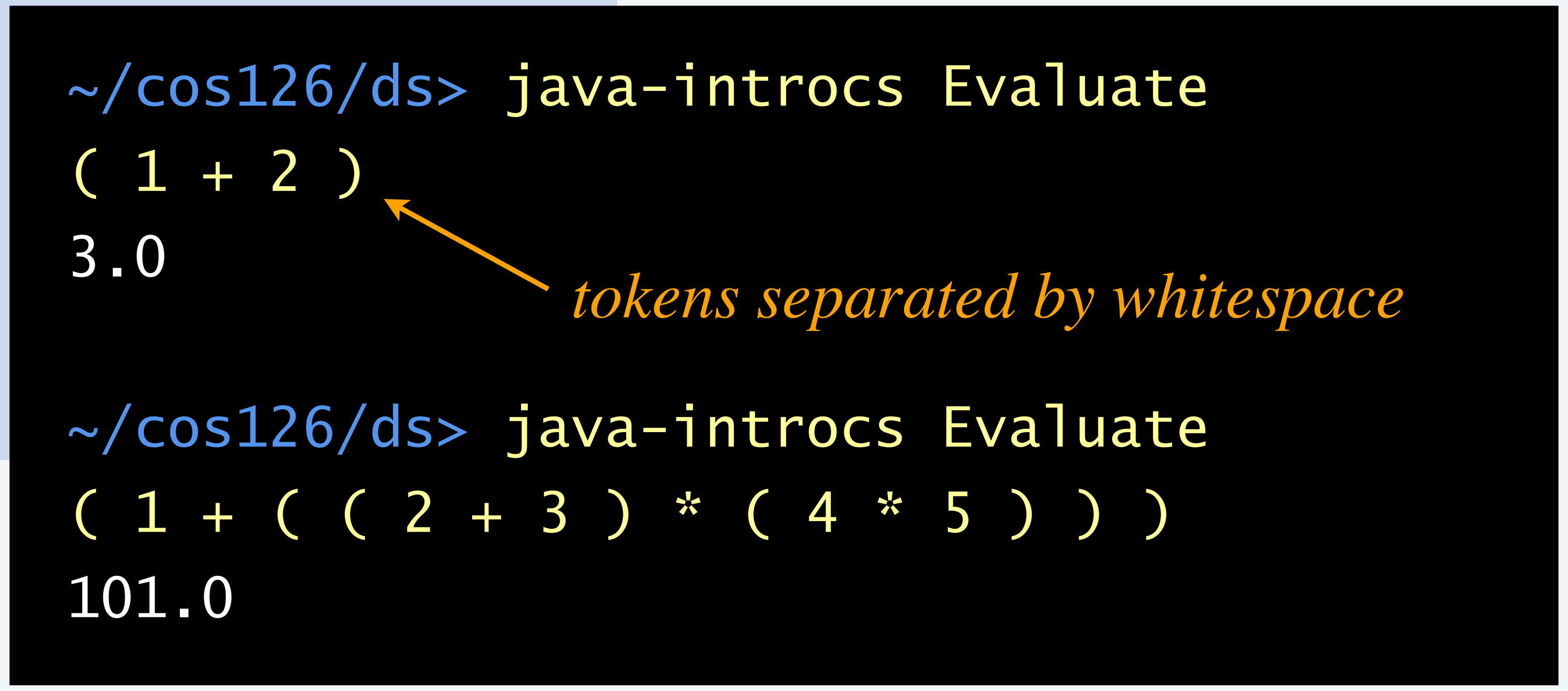
D. None of the above.

Arithmetic expression evaluation: Java implementation

```java
public class Evaluate {
   public static void main(String[] args) {
      Stack<String> ops  = new Stack<String>();
      Stack<Double> vals = new Stack<Double>();

      while (!StdIn.isEmpty()) {
         String s = StdIn.readString();
         if      (s.equals("(")) /* no-op */ ;
         else if (s.equals("+")) ops.push(s);
         else if (s.equals("*")) ops.push(s);
         else if (s.equals(")")) {
            String op = ops.pop();
            if      (op.equals("+")) vals.push(vals.pop() + vals.pop());
            else if (op.equals("*")) vals.push(vals.pop() * vals.pop());
         }
         else vals.push(Double.parseDouble(s));
      }

      StdOut.println(vals.pop());
   }
}
```

for stack of primitive type, need to use "wrapper" type

careful with non-commutative operators such as - and /



```
~/cos126/ds> java-introcs Evaluate
( 1 + 2 )
3.0

~/cos126/ds> java-introcs Evaluate
( 1 + ( ( 2 + 3 ) * ( 4 * 5 ) ) )
101.0
```

tokens separated by whitespace

Arithmetic expression evaluation: correctness

Q. Why correct?

A. When algorithm encounters an operator surrounded by two values within parentheses, it leaves the result on the value stack.

$$(1 + (\underline{(2 + 3)} * (4 * 5)))$$

as if the original input were:

$$(1 + (\underline{5} * (4 * 5)))$$

Repeating the argument:

$$(1 + (5 * 20))$$

$$(1 + 100)$$

$$101$$

Extensions. More operators, precedence order, associativity, …

Stack-based programming languages

Observation 1. Dijkstra’s two-stack algorithm computes the same value if each operator occurs after the two corresponding operands.

```
( 1 + ( ( 2 + 3 ) * ( 4 * 5 ) ) )

( 1 ( ( 2 3 + ) ( 4 5 * ) * ) + )
```

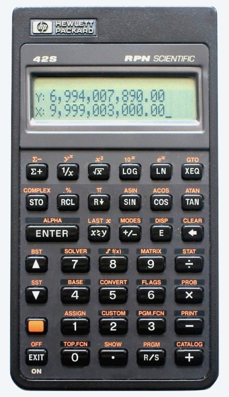
← *operator after operands*

Observation 2. All of the parentheses are redundant! ← *every right parenthesis is now preceded by an operator*

```
1 2 3 + 4 5 * * +
```

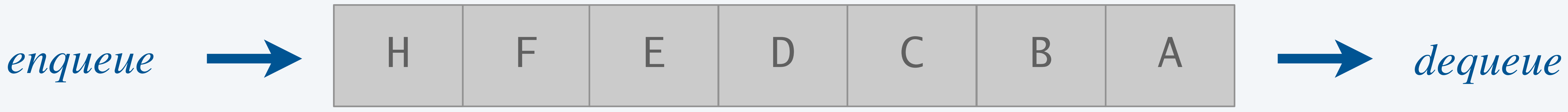
Bottom line. Postfix or “reverse Polish” notation (RPN).

Applications. PostScript, PDF, Java virtual machine, RPL, …



Queue API

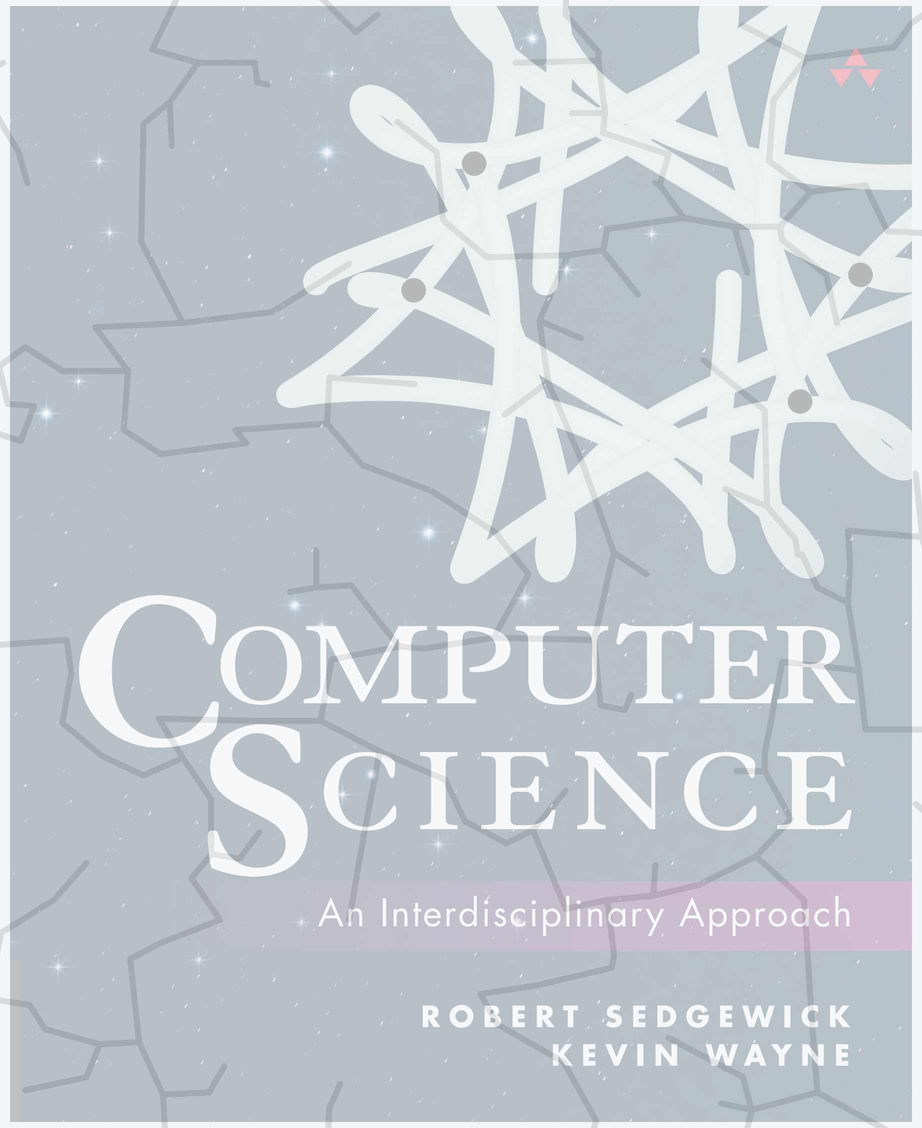
Queue data type. Our textbook data type for queues.



enqueue → H F E D C B A → *dequeue*

public class Queue<Item>		description
	`Queue()`	*create an empty queue*
`void`	`enqueue(Item item)`	*add a new item to the queue*
`Item`	`dequeue()`	*remove and return the item least recently added*
`boolean`	`isEmpty()`	*is the queue empty?*
`int`	`size()`	*number of items on the queue*

Performance requirements. Every operation takes constant time.



https://introcs.cs.princeton.edu

4.3 Data Structures

- *collections*
- *stacks and queues*
- *linked lists*
- *symbol tables*
- *Java collections framework*

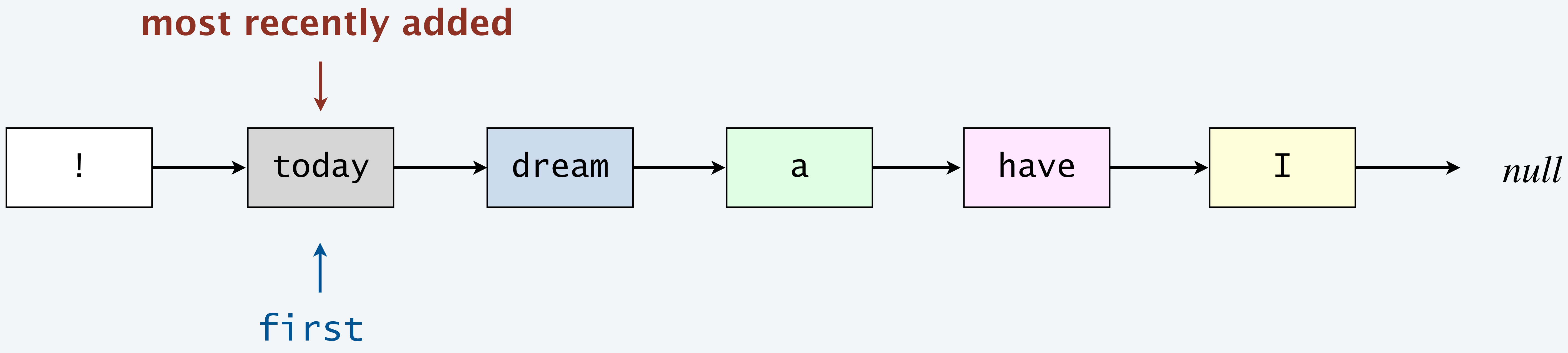
Stack implementation with a linked list

Q. How to implement a stack (or queue)?

Main challenge. Don’t know how many items will be on the stack. ← *otherwise, could used an array*

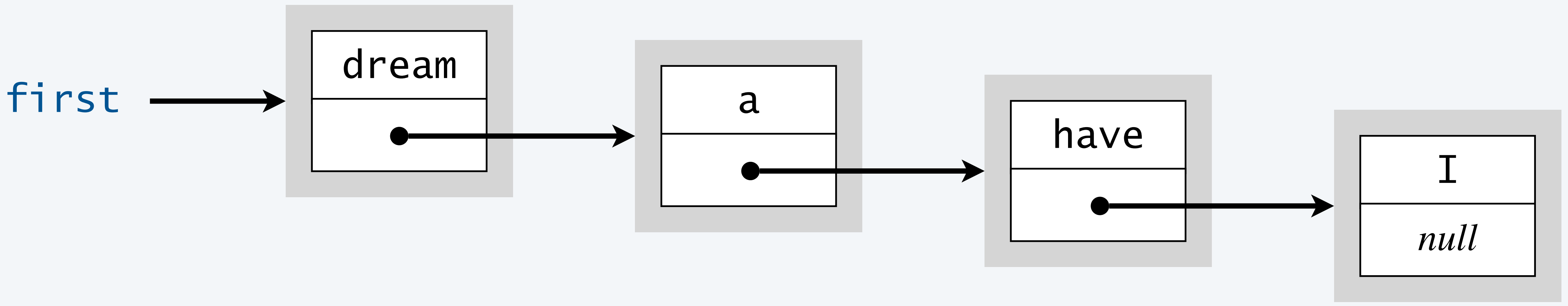
An elegant solution. Use a singly linked list.

- A node contains an item and a reference to the next node in the sequence.
- Maintain reference `first` to first node.
- Push new item before `first`.
- Pop item from `first`.



Stack implementation with a linked list: pop

singly linked list

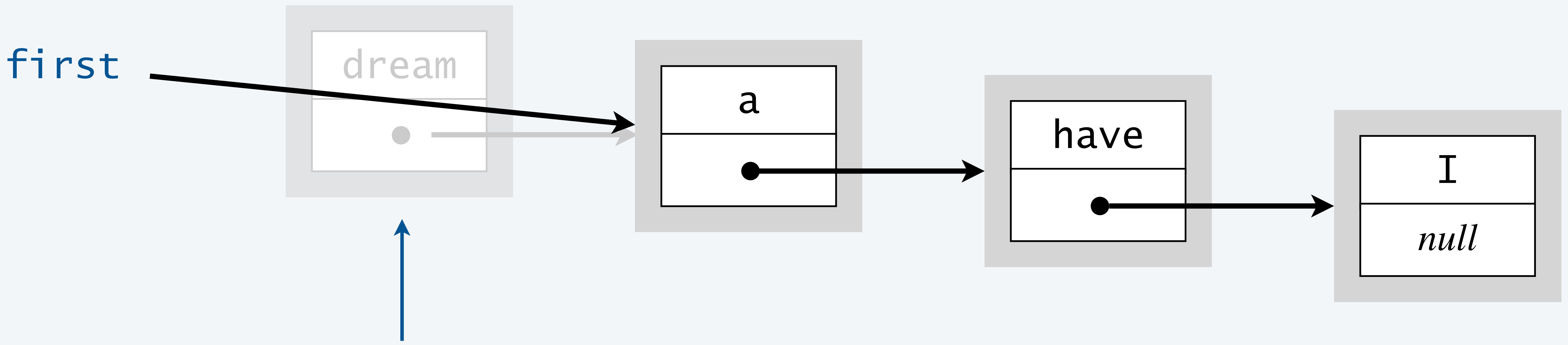


save item to return

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

delete first node

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



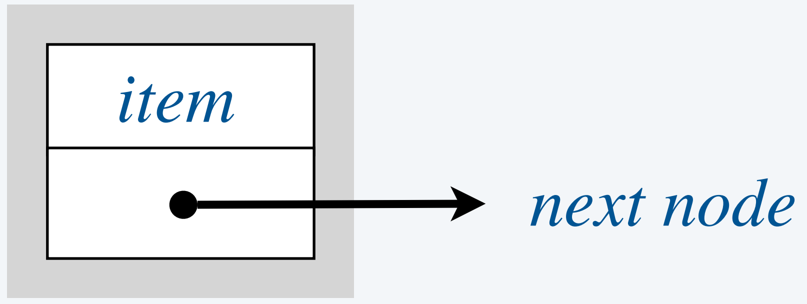
garbage collector reclaims memory when no remaining references

return saved item

```
return item;
```

nested class

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

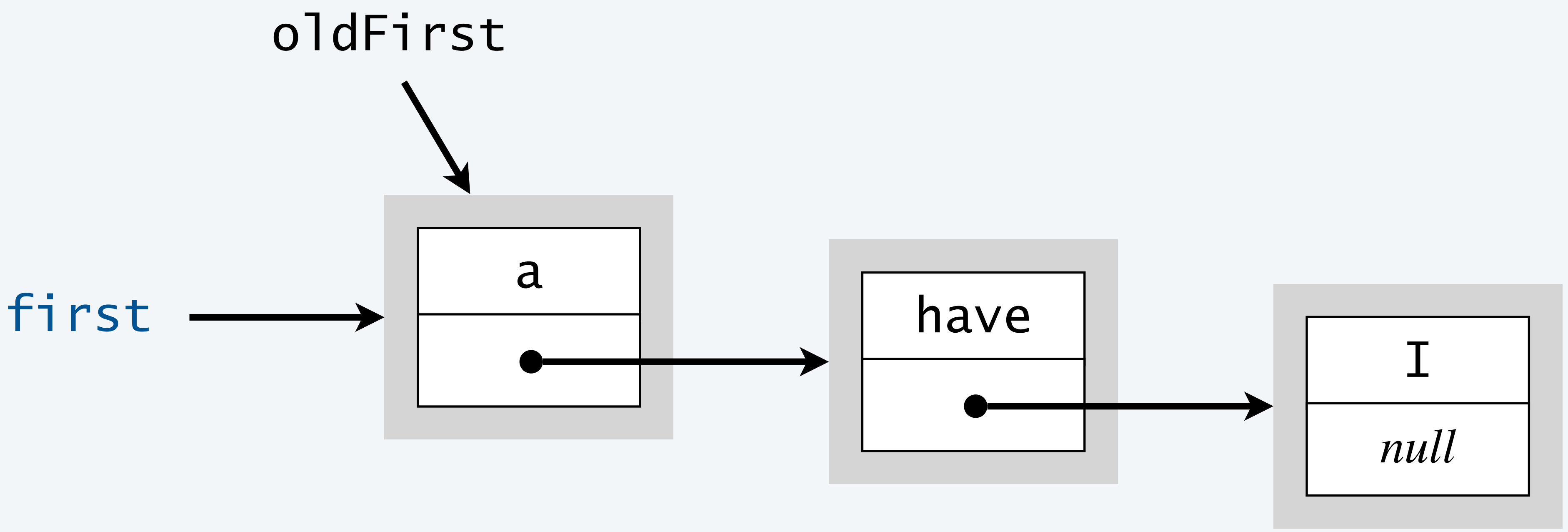


Node

Stack implementation with a linked list: push

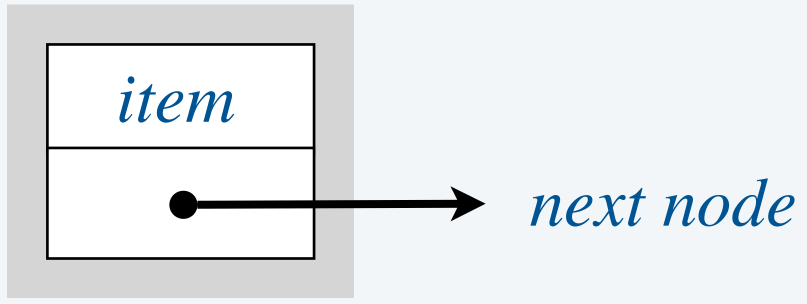
save a link to the list

```
Node oldFirst = first;
```



nested class

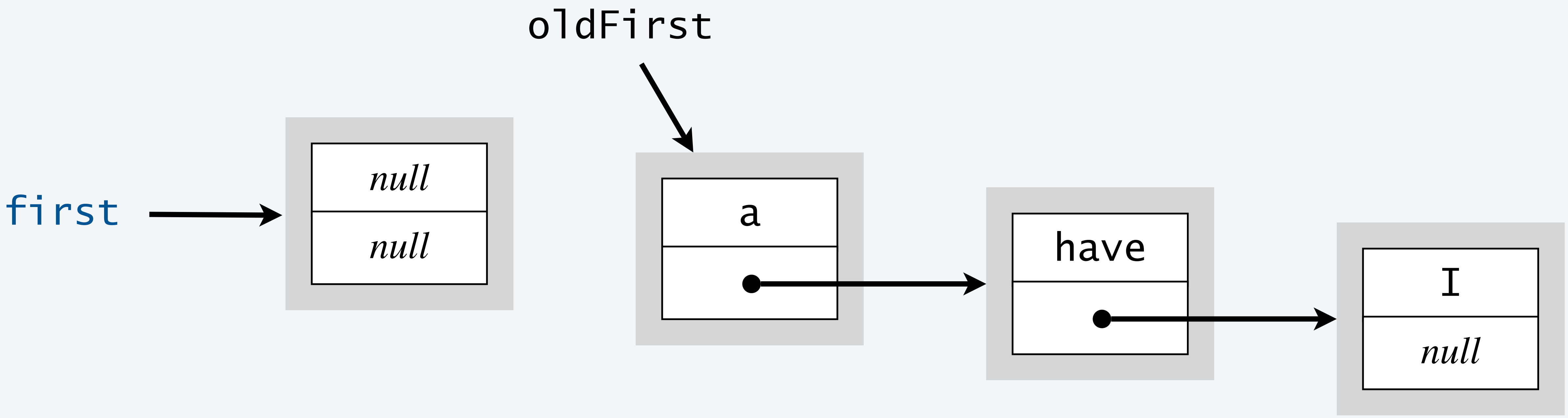
```
private class Node {
   private String item;
   private Node next;
}
```



Node

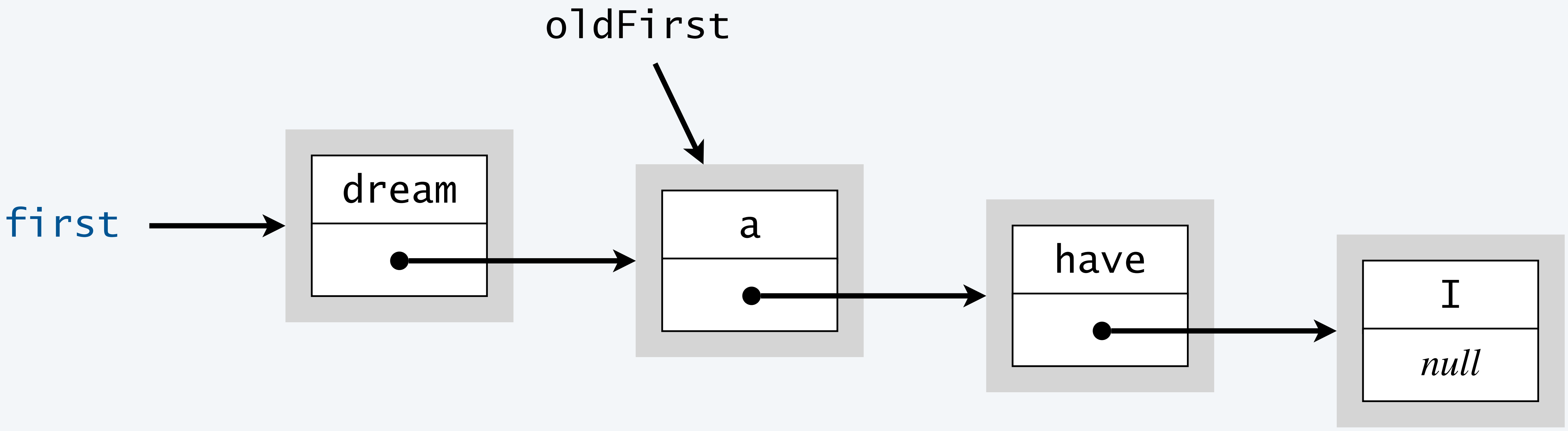
create a new node at the front

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



initialize the instance variables in the new Node

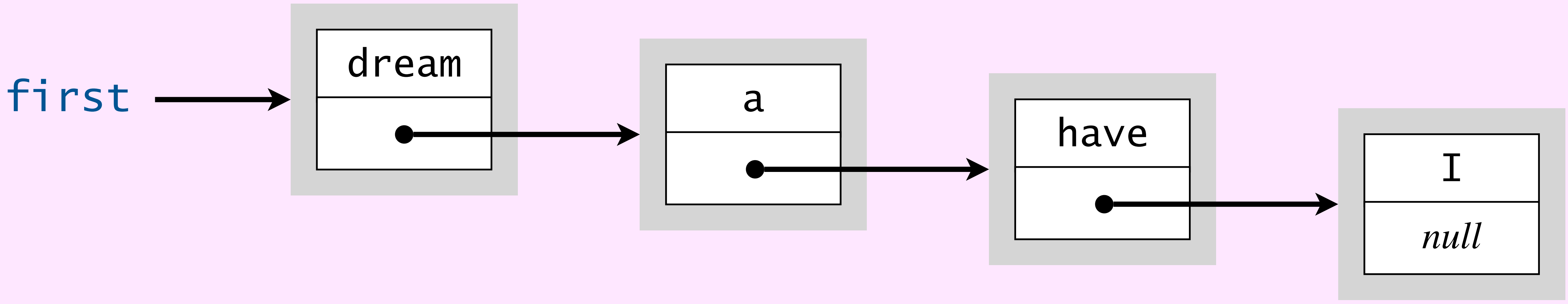
```
first.item = "dream";
first.next = oldFirst;
```



Possible memory representation

Each Node object stores a String and a reference to the next Node in the linked list.

actually, a reference to a String
(poetic license)



memory
address

286
first
304
318
330
a
330
304
dream
286
I
0
have
318

object reference
(holds memory address of Node *object)*

Node *object*

null reference

memory representation
(using poetic license)

Stack implementation with a linked list

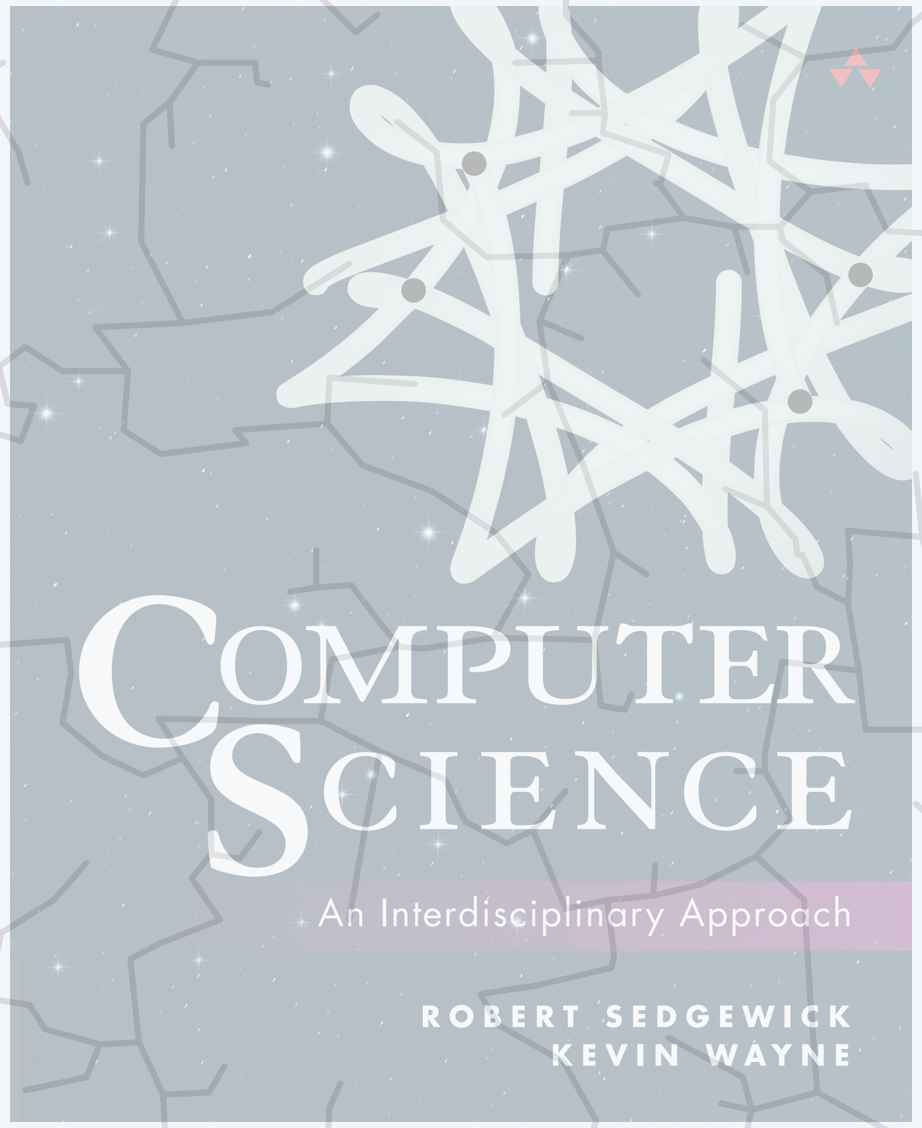
```
public class StackOfStrings {
   private Node first;

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

   public class Stack() {
      first = null;
   }

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

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

⚠ **code beyond scope of COS 126**

for simplicity, we're assume items are of type String

private *nested class*
(not accessible outside this file)

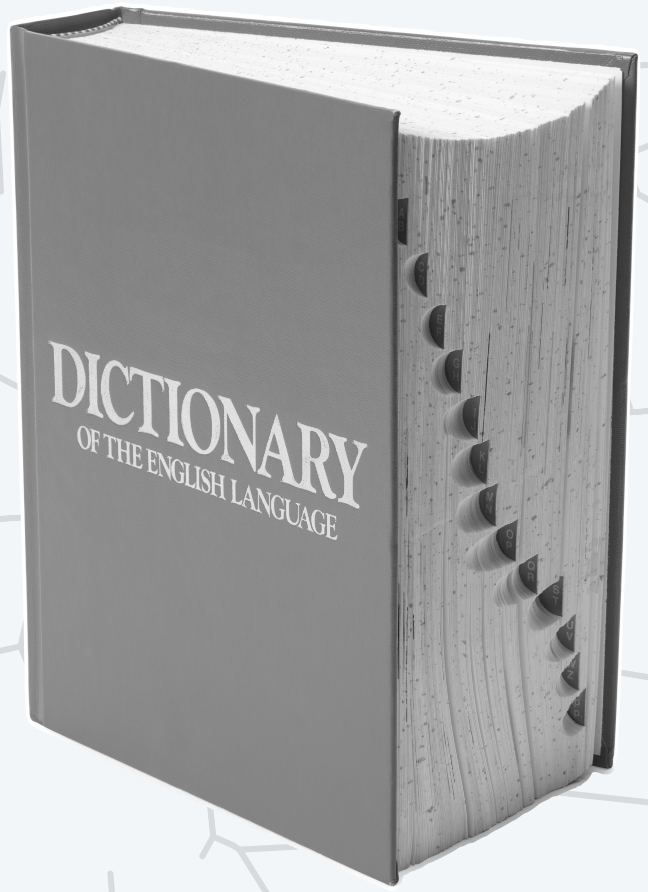
no Node *constructor explicitly defined* ⇒
Java supplies default no-argument constructor



https://introcs.cs.princeton.edu

4.3 Data Structures

- ‣ *collections*
- ‣ *stacks and queues*
- ‣ *linked lists*
- ‣ *symbol tables*
- ‣ *Java collections framework*



Symbol tables

Key-value pair abstraction.

- Insert a value with specified key.
- Given a key, search for the corresponding value.

also known as maps (Java), dictionaries (Python), and associative arrays (Perl)

Ex. DNS lookup.

- Insert domain name with specified IP address.
- Given domain name, find corresponding IP address.

domain name	IP address
www.cs.princeton.edu	128.112.136.61
goprincetontigers.com	67.192.28.17
wikipedia.com	208.80.153.232
google.com	172.217.11.46

key *value*

Symbol table applications

application	purpose of search	key	value
dictionary	*find definition*	word	definition
compiler	*find properties of variables*	variable name	type and value
DNS	*find IP address*	domain name	IP address
reverse DNS	*find domain name*	IP address	domain name
file system	*find file on disk*	filename	location on disk
file share	*find song to download*	name of song	computer ID
web search	*find relevant web pages*	keyword	list of page names

Symbol table API

Symbol table data type. Our textbook data type for symbol tables.

Key *type must be comparable* (String, Integer, Double, …)

public class ST<Key, Value>		description	
	ST()	*create an empty symbol table*	*generalizes arrays (keys need not be integers between 0 and $n-1$)*
void	put(Key key, Value val)	*insert key–value pair*	← a[key] = val;
Value	get(Key key)	*value paired with* key	← a[key]
boolean	contains(Key key)	*is there a value paired with* key*?*	
Iterable<Key>	keys()	*all the keys in the symbol table*	
boolean	isEmpty()	*is the symbol table empty?*	
int	size()	*number of key–value pairs*	
	⋮		

Data structures: quiz 3

What does the following code fragment print?

A. 1.0

B. 1.5

C. 2.5

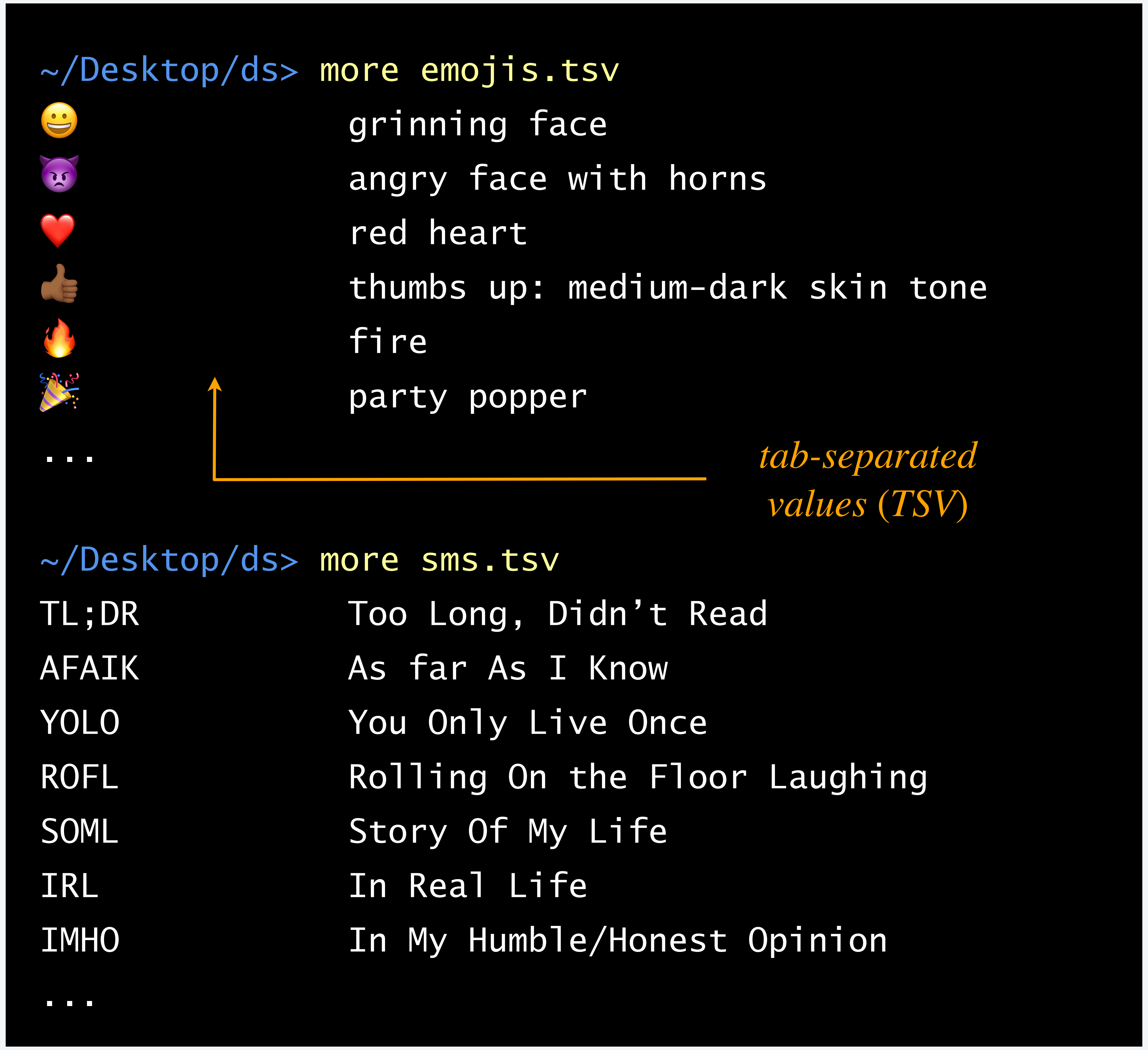
D. Run-time exception.

```
ST<String, Double> st = new ST<String, Double>();
st.put("a", 1.0);
st.put("b", 1.5);
st.put("a", st.get("a") + st.get("b"));
double value = st.get("a");
StdOut.println(value);
```

Text-to-English

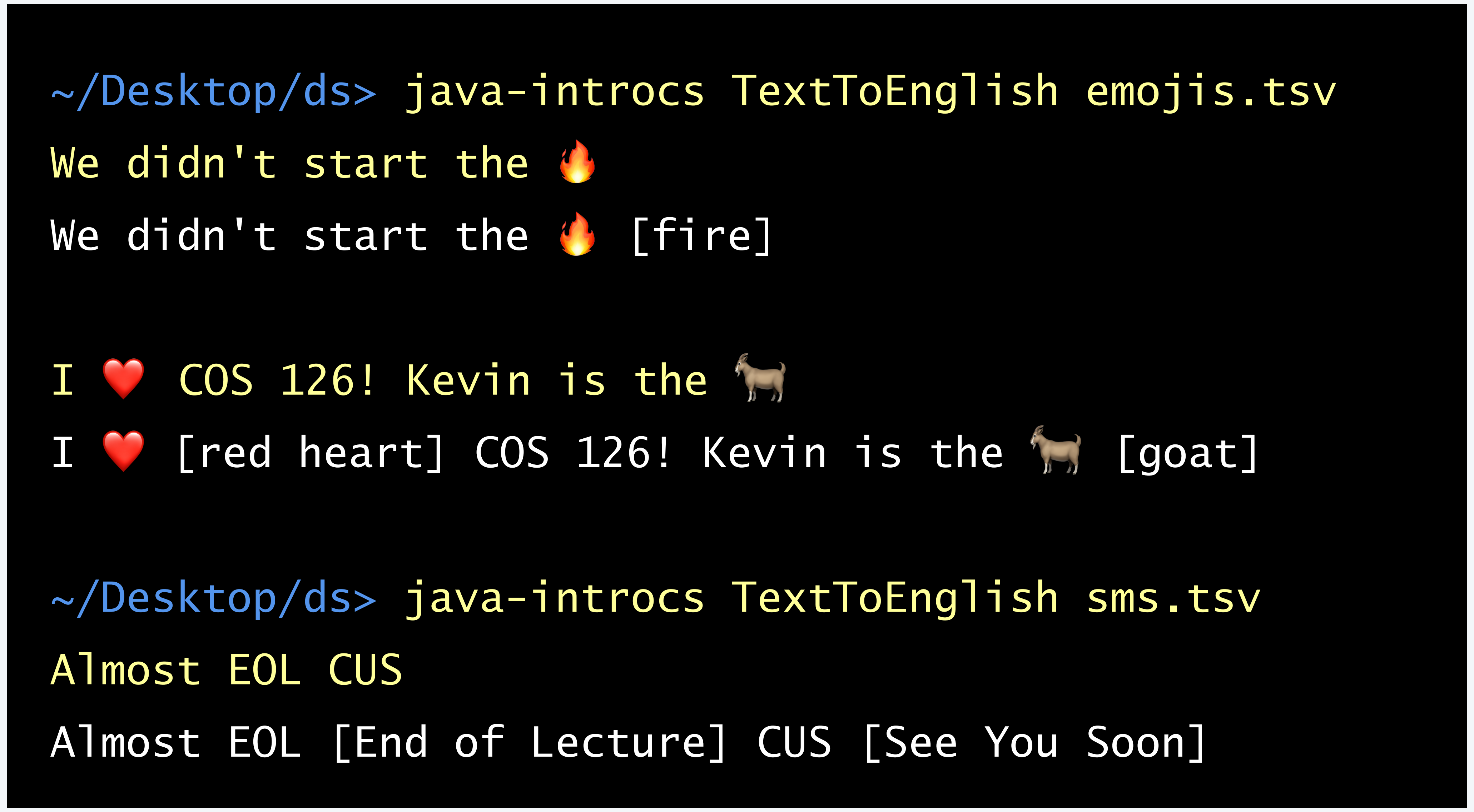
Goal. Convert text message with emojis (or text abbreviations) to English.

- Create symbol table that maps from emoji (or text abbreviation) to English.
- Read lines from standard input, replacing emojis (or text abbreviations) with expansions.



```
~/Desktop/ds> more emojis.tsv
😀              grinning face
👿              angry face with horns
❤️              red heart
👍🏾              thumbs up: medium-dark skin tone
🔥              fire
🎉              party popper
...

~/Desktop/ds> more sms.tsv
TL;DR           Too Long, Didn’t Read
AFAIK           As far As I Know
YOLO            You Only Live Once
ROFL            Rolling On the Floor Laughing
SOML            Story Of My Life
IRL             In Real Life
IMHO            In My Humble/Honest Opinion
...
```

tab-separated values (TSV)



```
~/Desktop/ds> java-introcs TextToEnglish emojis.tsv
We didn't start the 🔥
We didn't start the 🔥 [fire]

I ❤️ COS 126! Kevin is the 🐐
I ❤️ [red heart] COS 126! Kevin is the 🐐 [goat]

~/Desktop/ds> java-introcs TextToEnglish sms.tsv
Almost EOL CUS
Almost EOL [End of Lecture] CUS [See You Soon]
```

Text-to-English converter

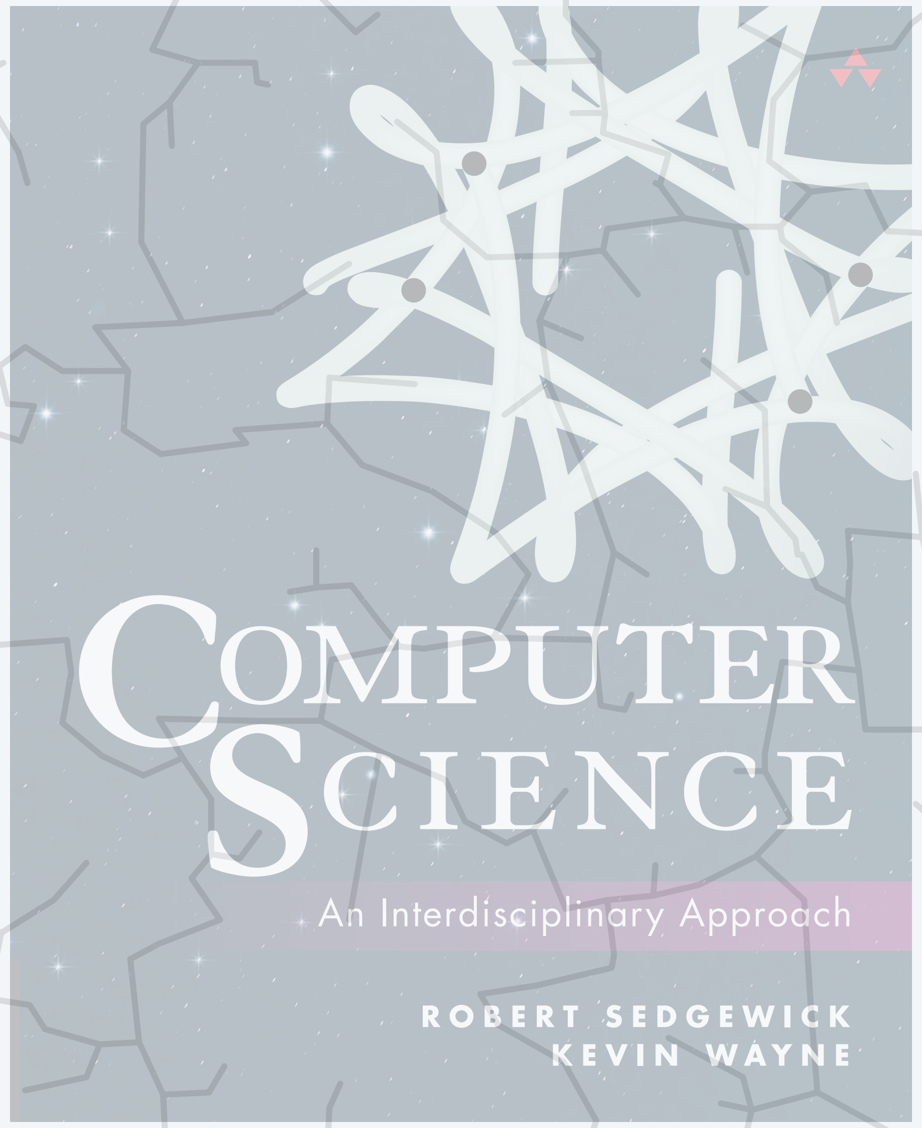
```
public class TextToEnglish {
   public static void main(String[] args) {
      // build symbol table that maps from abbreviation to expansion
      ST<String, String> st = new ST<String, String>();
      In in = new In(args[0]);
      while (in.hasNextLine()) {
         String line = StdIn.readLine();
         String[] fields = line.split("\\t");
         String abbreviation = fields[0];
         String expansion = fields[1];
         st.put(abbreviation, expansion);
      }

      // process lines of text, replacing abbreviations with expansions
      while (StdIn.hasNextLine()) {
         String line = StdIn.readLine();
         String[] words = line.split(" ");
         for (int i = 0; i < words.length; i++) {
            StdOut.print(words[i] + " ");
            if (st.contains(words[i])) {
               StdOut.print("[" + st.get(words[i]) + "]" + " ");
            }
         }
         StdOut.println();
      }
   }
}
```

create symbol table with string keys (abbreviations) and string values (expansions)

break line into fields using tab as delimiter

process one word at a time

print expansion if word is in symbol table (delimiting with square braces)



COMPUTER SCIENCE

An Interdisciplinary Approach

ROBERT SEDGEWICK
KEVIN WAYNE

https://introcs.cs.princeton.edu

4.3 Data Structures

- *collections*
- *stacks and queues*
- *linked lists*
- *symbol tables*
- *Java collections framework*

System libraries

Textbook libraries. Collections for stacks, queues, symbol tables, sets, …

Java collections framework. Collections for lists, symbol tables (maps), sets, …

collection	core operations	introcs.jar	java.util
stack	PUSH, POP	Stack	~~java.util.Stack~~ java.util.LinkedList java.util.ArrayList
queue	ENQUEUE, DEQUEUE	Queue	
symbol table	PUT, GET, DELETE	ST	java.util.TreeMap java.util.HashMap
set	ADD, CONTAINS, DELETE	SET	java.util.TreeSet java.util.HashSet
⋮	⋮	⋮	

← *provides superset of stack/queue operations*

Java collections framework: lists

`java.util.LinkedList`. Java collections framework data type for lists.

public class LinkedList<Item>		description
	`LinkedList()`	*create an empty list*
`void`	`addFirst(Item item)`	*add a new item to the beginning of list*
`void`	`addLast(Item item)`	*add a new item to the end of list*
`Item`	`removeFirst()`	*remove and return item at beginning of list*
`Item`	`removeLast()`	*remove and return item at end of list*
`boolean`	`isEmpty()`	*is the list empty?*
`int`	`size()`	*number of items in the list*
	⋮	

← *generalizes stacks and queues*

Performance requirements. Above operations take constant time. ← *but many other* `LinkedList` *operations do not (!)*

Java collections framework: symbol tables

`java.util.TreeMap`. Java collections framework data type for symbol tables (maps).

public class TreeMap<Key, Value>		description
	TreeMap()	*create an empty symbol table*
Value	put(Key key, Value val)	*insert key–value pair*
Value	get(Key key)	*value paired with key*
Set<Key>	keySet()	*all the keys in the symbol table*
boolean	containsKey(Key key)	*is there a value paired with key?*
void	remove(Key key)	*remove key (and associated value)*
boolean	isEmpty()	*is the symbol table empty?*
int	size()	*number of key–value pairs*
	⋮	

← *similar to API for* ST

Enhanced for loop (foreach loop)

Enhanced for loop. A second form of `for` loop designed to iterate over collections (and arrays).

```
double[] values = { 0.0, 2.0, 3.0, 6.125, 4.5 };
double sum = 0.0;
for (double x : values) {
   sum += x;
}
```

iterates over array elements in order

enhanced for loop with an array

```
LinkedList<String> list = new LinkedList<String>();
list.addLast("I");
list.addLast("have");
list.addLast("a");
list.addLast("dream");
list.addLast("today!");
for (String s : list) {
   StdOut.println(s);
}
```

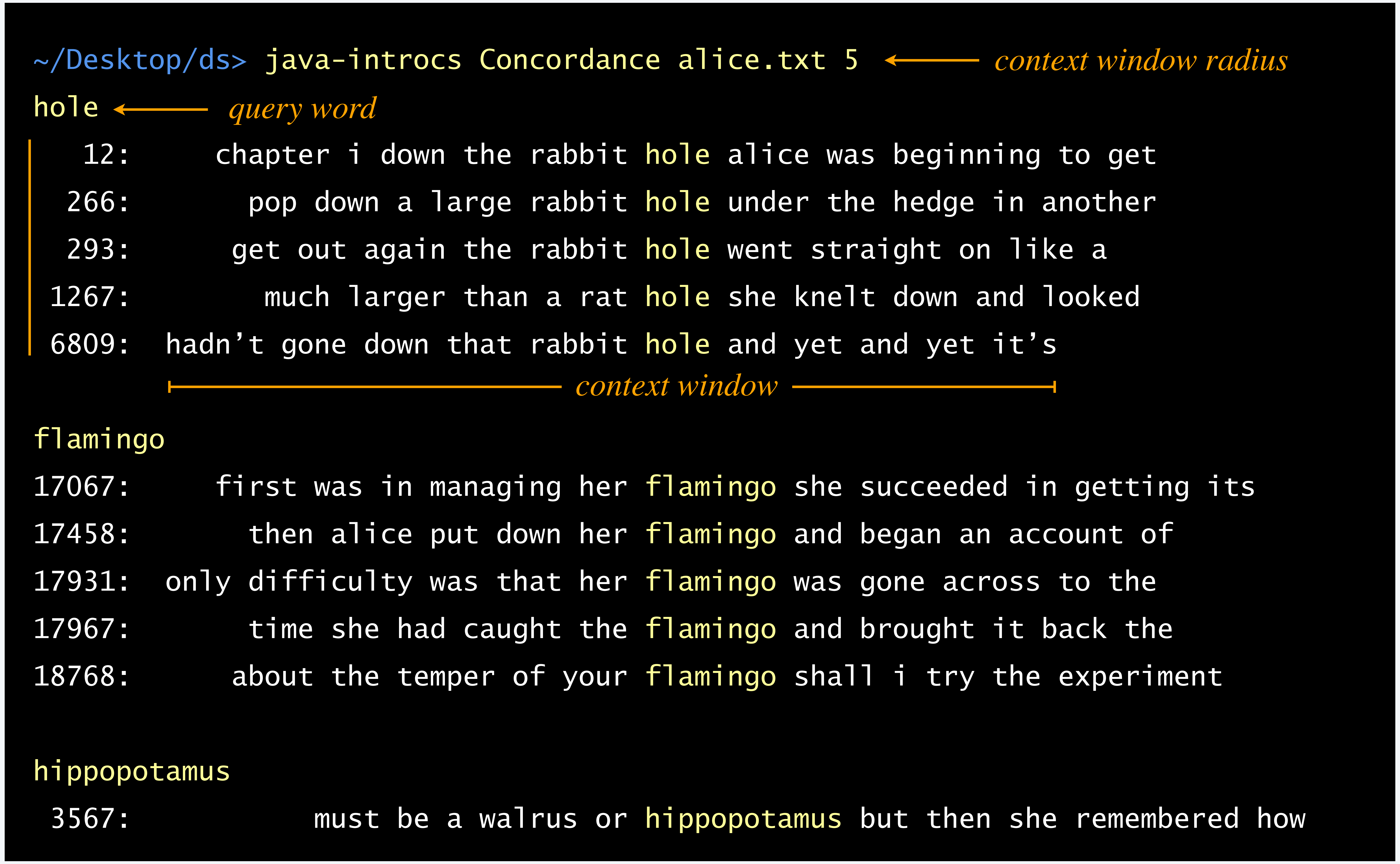
iterates over list elements in list order

enhanced for loop with a collection
(iterates without removing the elements)

Concordance

A concordance is a list of every occurrence of each word in a text, along with surrounding context.

indices where query word appears



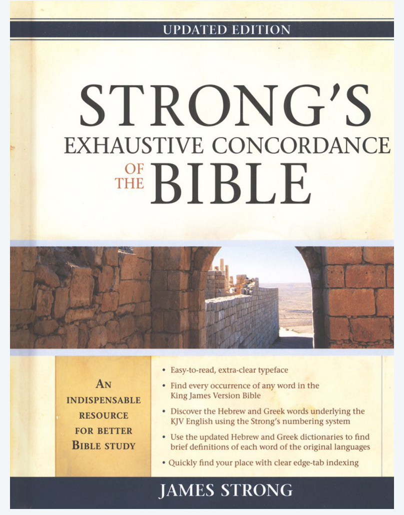
```
~/Desktop/ds> java-introcs Concordance alice.txt 5
hole
   12:     chapter i down the rabbit hole alice was beginning to get
  266:       pop down a large rabbit hole under the hedge in another
  293:      get out again the rabbit hole went straight on like a
 1267:        much larger than a rat hole she knelt down and looked
 6809:  hadn’t gone down that rabbit hole and yet and yet it’s

flamingo
17067:     first was in managing her flamingo she succeeded in getting its
17458:       then alice put down her flamingo and began an account of
17931:  only difficulty was that her flamingo was gone across to the
17967:       time she had caught the flamingo and brought it back the
18768:      about the temper of your flamingo shall i try the experiment

hippopotamus
 3567:         must be a walrus or hippopotamus but then she remembered how
```

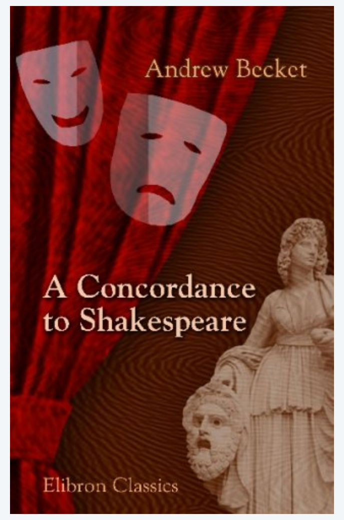
Concordance

A concordance is a list of every occurrence of each word in a text, along with immediate context.

Pre-computational age. Compiled only for works of special importance:

- Vedas.
- Bible.
- Qur’an.
- Works of Shakespeare.
- ...





Computational age. Any COS 126 student can create one!

Spotlight search (iOS or OS X). Essentially a concordance of files on your phone/computer.

Google search. Essentially a concordance of the web.

with clever algorithm to rank results

Data structures: quiz 4

What should the declared type be for a symbol table for concordance?

A. `TreeMap<String, Integer>`

B. `TreeMap<Integer, String>`

C. `TreeMap<String, LinkedList<Integer>>`

D. `TreeMap<LinkedList<Integer>, String>`

Concordance implementation: build concordance

```
import java.util.LinkedList;
import java.util.TreeMap;

public class Concordance {
  public static void main(String[] args) {
     In in = new In(args[0]);
     String[] words = in.readAllStrings();

     // build concordance
     TreeMap<String, LinkedList<Integer>> map = new TreeMap<String, LinkedList<Integer>>();
     for (int i = 0; i < words.length; i++) {
        String s = words[i];

        if (!map.containsKey(s)) {
           map.put(s, new LinkedList<Integer>());
        }

        LinkedList<Integer> list = map.get(s);
        list.addLast(i);
     }

     ⋮
```

access Java collections libraries

read all words in file

first occurrence of word

get list associated with word

add index of word to list

Concordance implementation: process queries

```
public class Concordance {
   public static void main(String[] args) {
            ⋮
      int context = Integer.parseInt(args[1]);

      // process queries
      while (!StdIn.isEmpty()) {
         String query = StdIn.readString();
         if (map.containsKey(query)) {
            LinkedList<Integer> list = map.get(query);
            for (int k : list) {
               int start = Math.max(k - context, 0);
               int end   = Math.min(k + context, words.length - 1);
               for (int i = start; i <= end; i++) {
                  StdOut.print(words[i] + " ");
               }
               StdOut.println();
            }
         }
      }
   }
}
```

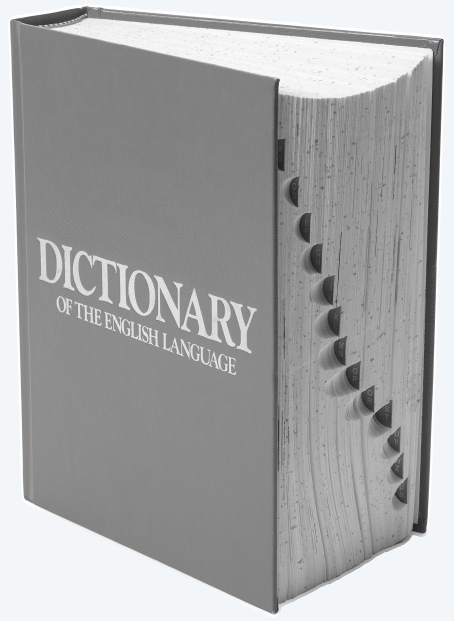
list of indices where word appears

print 5 words before and after (context window)

Collections summary

Fundamental data types.

- Value: collection of objects.
- Operations: add, remove, iterate, size, …



Stack. Remove the item most recently added.

Queue. Remove the item least recently added.

Symbol table. Associate key-value pairs.

…

COS 126. Use pre-existing collection data types.

COS 226. Implement your own collections using linked data structures and resizing arrays.

Credits

media	source	license
Data Structures Icon	Adobe Stock	education license
Bushel of Apples	Adobe Stock	education license
Stack of Sweaters	Adobe Stock	education license
Long Queue Line	Adobe Stock	education license
Stack of Books	Adobe Stock	education license
Red Back Button	Adobe Stock	education license
Undo Icon	Wikimedia	MIT license
Triage in ER	mainjava.com	
Queue of People	Adobe Stock	education license
RPN Calculator	Wikimedia	CC BY 2.0

Credits

media	source	license
Dictionary	Adobe Stock	education license
Java Logo	Oracle	
Alice in Wonderland	Lewis Carroll	
Bible Concordance	James Strong	
Shakespeare Concordance	Andrew Becket	