Advanced Programming Techniques in Java

Recursion III & Stacks

Lecture 22

Class Objectives

- Backtracking (Section 12.5)

Stacks (Chapter 14)

Review: Recursive Alg
Towers of Hanoi

Recursive Algorithm for n -Disk Problem:Move
Peg to the Destination Peg `if` n is 1 move disk 1
the starting peg to the destination peg `else`
move the top $n-1$ disks from the starting pe
starting nor destination peg) move disk n (the
starting peg to the destination peg move th
peg to the destination peg

Review: Counting Cel
Design

Algorithm for `countCells(x, y)`

```
if the cell at (x, y) is outside
      the grid the result is o
else if the color of the cell at (x, y
      abnormal color the result is o
else
```

set the color of the cell at (x, y
temporary color the result is 1 pl
cells in each piece of the blob th
nearest neighbor

Review: Counting Cells i

- □ Verify that the code works for th
  - A starting cell that is on the ed

- A starting cell that has no neighb
- A starting cell whose only abnorr diagonally connected to it
- A "bull's-eye": a starting cell w normal but their neighbors ar
- A starting cell that is normal
- A grid that contains all abnormal
- A grid that contains all normal ce

Backtracking

Backtracking

- Backtracking is an approach to i systematic trialand error search f
- An example is finding a path
- Ifyou are attempting to walk throu probably walk down a path
  - Eventually, you will reach yourdestin able to go any farther

- If you can’t go any farther, you w
  alternative paths

- □ Backtracking is a systematic, nonr
  trying alternative paths and elimi
  don’t work

Backtracking (cont.)

- □ Ifyou never try the same path n
  will eventually finda solutionpath

- □ Problems thatare solved by backtrack
  as a set of choices made by s

- Recursion allows you to implemen relatively straightforward manner
    - Each activation frame is used choice that was made at that p
- Aprogram thatplays chess may of backtracking algorithm

Finding a Path throu

- Problem

- Use backtracking to find and displa
- From each point in a maze, next cell in a horizontal or v cell is not blocked

Finding a Path throug

- □ Analysis
  - The maze will consist of a g
  - The starting point is at the t
  - The exit point is at the botto
    `(getNCols() - 1, getNRows()`
  - All cells on the path will be `BACKGROU`
  - All cells that represent barriers will b
  - Cells that we have visited will b
  - If we find a path, all cells on t
    color

Recursive Algorithm f
Maze Path

Recursive Algorithm for findMazePath(x

```
if the current cell is outside the
        maze return false (you are
        out of bounds)
else if the current cell is part of the b
        been visited return false (you are o
        cycle)
else if the current cell is the maze e
        to the pathcolor and return true
        successfully completed the maze)
else // Try to find a pathfrom the c
```

```
mark the current cell as on the pathb
   pathcolor
for each neighbor of the current cell
      path exists from the neighbor t
      maze exit return true
// No neighbor of the current cell is o
recolor the current cell to the temporary
   (visited) and return
false
```

Testing

- Test for a variety of test c
  - Mazes that can be solved

- Mazes that can't be solved
- A maze withno barrier cells
- A maze witha single barrier c

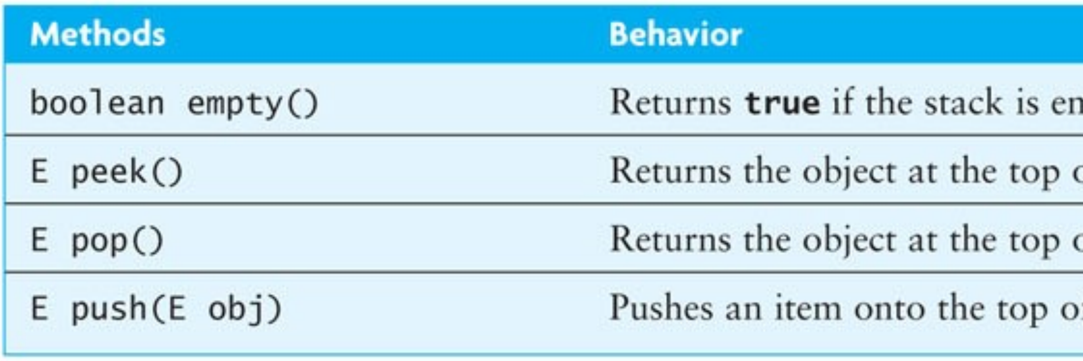
StackAbstract

Stack AbstractData

- A stack is one of the most used data structures in compu
- A stack can be compared to dispenser
  - Only the top item can be acc
  - You can extract only one item
- The top element in the stack added to the stack most recent
- The stack’s storage policy is *First-Out*, or *LIFO*

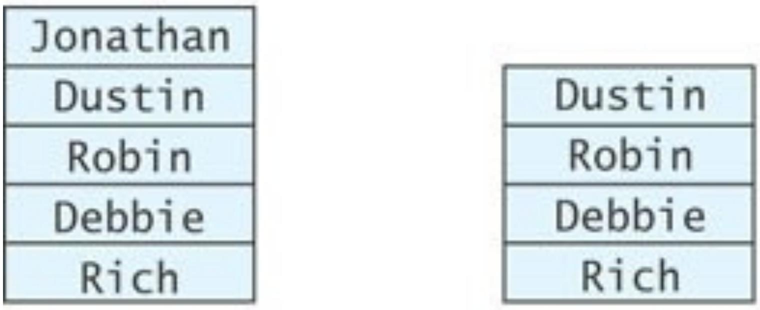
Specification of the St
Data Type

- Only the top element of a stack
  number of operations performed
- We need the ability to
  - test for an empty stack (empty
  - inspect the top element (peek)
  - retrieve the top element (pop)
  - put a new element on the stack



Methods	Behavior
`boolean empty()`	Returns **true** if the stack is en
`E peek()`	Returns the object at the top c
`E pop()`	Returns the object at the top c
`E push(E obj)`	Pushes an item onto the top o:

A Stack ofStrings



(a) (b)

- “Rich” is the oldest element on the stack a
  (Figure a)
- `String last = names.peek();` sto
  “Jonathan” in `last`
- `String temp = names.pop();` rem
  reference to it in `temp` (Figure b)
- `names.push(“Philip”);` pushes “Ph
  c)

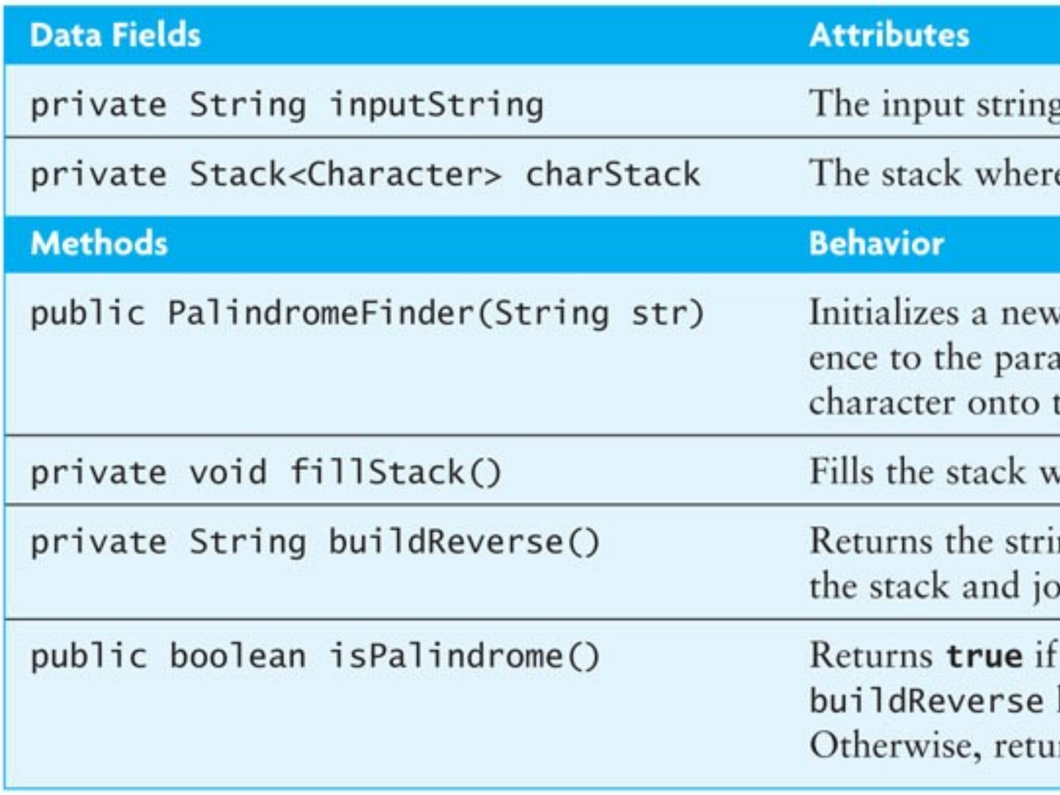
StackApplicati

Finding Palindrome

- Palindrome: a string that rea either direction, letter by let
  - kayak
  - "I saw I was I"
  - “Able was I ere I saw Elba”
  - "Level, madam, level"

- Problem:Write a program tha
  and determines whether

Finding Pal



Data Fields	Attributes
`private String inputString`	The input string
`private Stack<Character> charStack`	The stack where
Methods	**Behavior**
`public PalindromeFinder(String str)`	Initializes a new ence to the para character onto t
`private void fillStack()`	Fills the stack w
`private String buildReverse()`	Returns the strin the stack and jo
`public boolean isPalindrome()`	Returns **true** if `buildReverse` I Otherwise, retur

Finding Pal

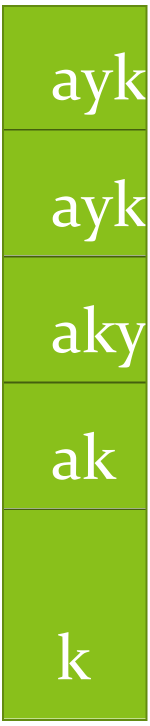
```
import java.util.*;

public class PalindromeFinder {
  private String inputString;
  private Stack<Character> charStack = ne

 public PalindromeFinder(String str) {
   inputString = str;
   fillStack();
```

Finding Pal

```
}
. . .
```

- Solving using a stack:
  - Push each string characte onto a stack

Finding Pal



ayk

ayk

aky

ak

k

k a y a k

```
private void fillStack() {
for(int i = 0; i < inputString.leng
  charStack.push(inputString.charAt
}
```

- Solving using a stac
  - Pop each characte each to the `StringBuilder result`

Finding Pal

ay

aa

yak

ak

k

kayak

```
private String buildReverse(){
 StringBuilder result = new Str

while(!charStack.empty()) {result.appen
 } return
 result.toString();
}
```

. . .

Finding Pal

```
   public boolean isPalindrome() {
     return inputString.equalsIgno
   }
}
```

Testing

- We can test this class u inputs:
  - a single character (always a p
  - multiple characters in a word
  - multiple words
  - different cases
  - even-length strings
  - odd-length strings
  - the empty string (considered a

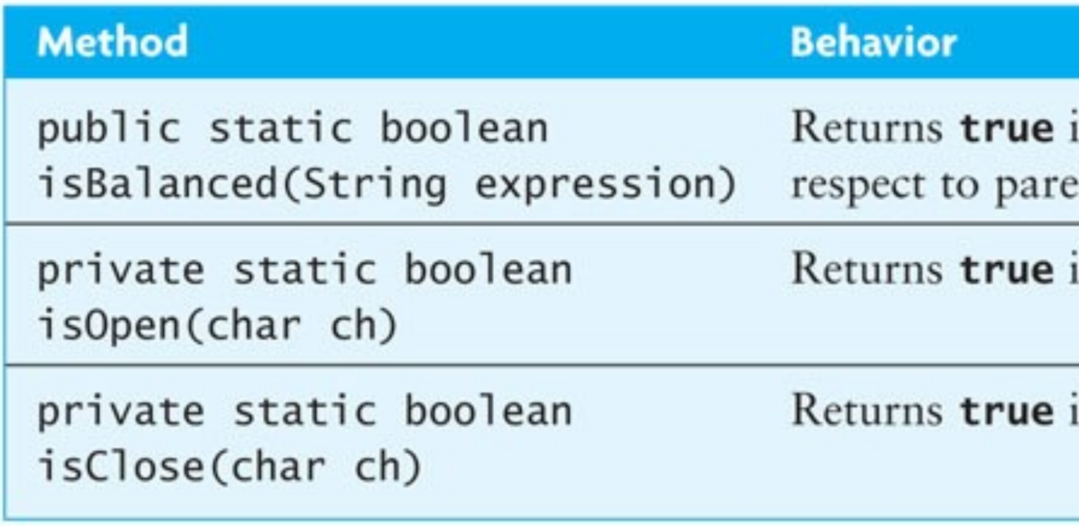
Balanced Parenthese

- When analyzing arithmetic e importantto determinewhethe balanced with respect to p

```
( a + b * ( c / ( d - e
```

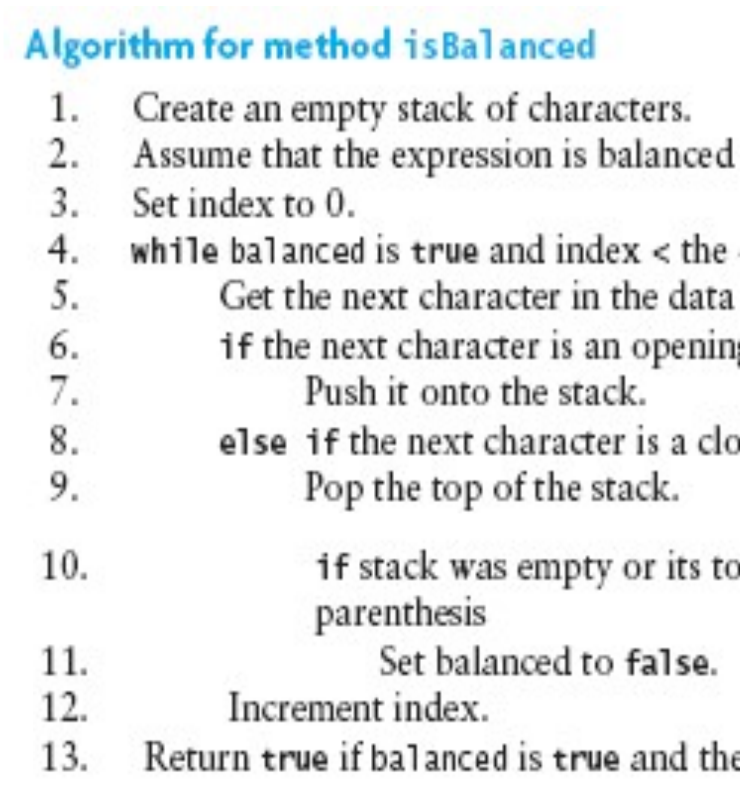
- The problem is further com brackets are used in conjun parentheses
- The solution i

Balanced Pa



Method	Behavior
`public static boolean isBalanced(String expression)`	Returns **true** i respect to pare
`private static boolean isOpen(char ch)`	Returns **true** i
`private static boolean isClose(char ch)`	Returns **true** i

Balanced Pa



Algorithm for method isBalanced

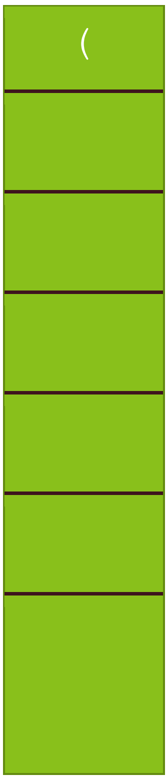
```
1.  Create an empty stack of characters.
2.  Assume that the expression is balanced
3.  Set index to 0.
4.  while balanced is true and index < the
5.      Get the next character in the data
6.      if the next character is an openin
7.          Push it onto the stack.
8.      else if the next character is a clo
9.          Pop the top of the stack.
10.         if stack was empty or its to
            parenthesis
11.             Set balanced to false.
12.     Increment index.
13. Return true if balanced is true and the
```

Balanced Pa

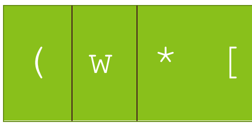
Expression: `(w * [x y] / z)`

0	1	2	3
(	w	*	[

Balanced Pa

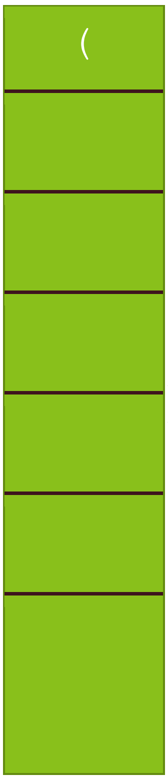

(

Expression: (w [x + y] / z)


(w * [

0 1 2 3

Balanced Pa

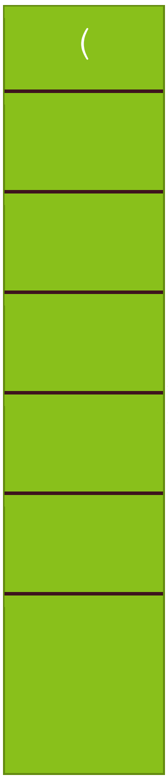


balanced : tru

Expression: (w [x + y] / z)

0 1 2 3

Balanced Pa



balanced : 2

↑ **t:**

Expression: (w [x + y] / z)

(	w	*	[
0	1	2	

Balanced Pa

[(

(

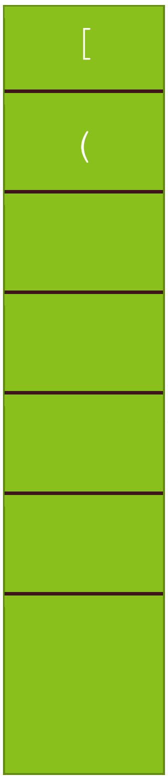
balanced : **true**

Expression: (w [x + y] / z)

(	w	*	[

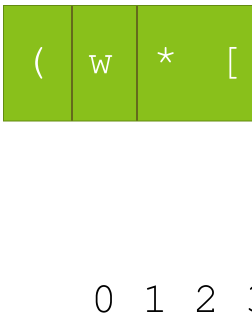
0 1 2 3

Balanced Pa

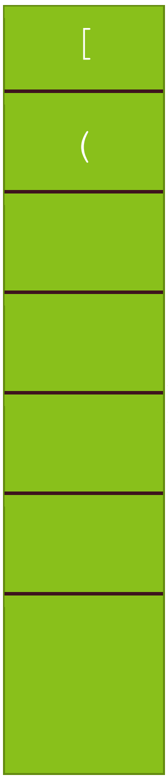


balanced : **true**

Expression: (w
[x + y] /
z)



Balanced Pa

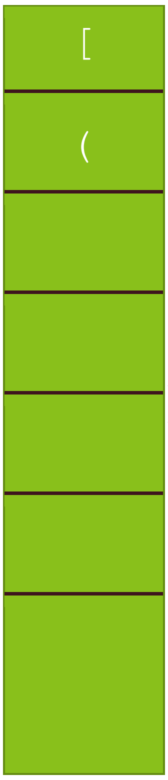


balanced : **true**

5

Expression: (w [x + y] / z)

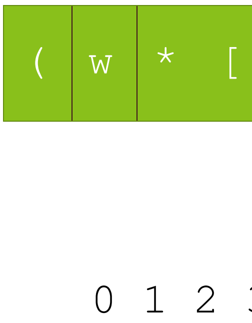
0 1 2



Balanced Pa

balanced : **true** inc

Expression: (w [x + y] / z)



[(
(

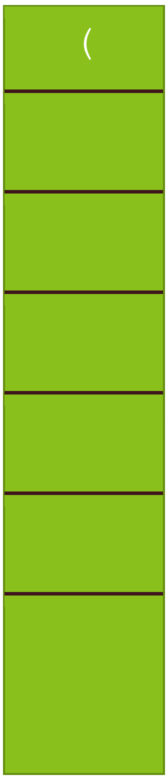
d Pa

Matches!
Balanced still
true

balanced : **true** ind

Expression: (w [x + y] / z)

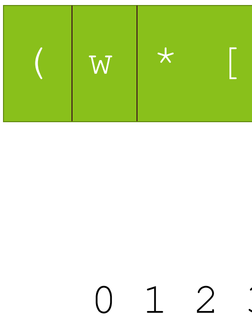
(	w	*	[
0	1	2	3

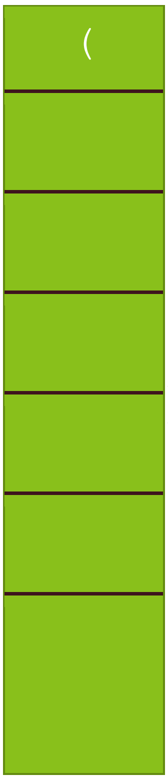


Balanced Pa

balanced : **true** inc

Expression: (w
[x + y] /
z)





(

Balanced Pa

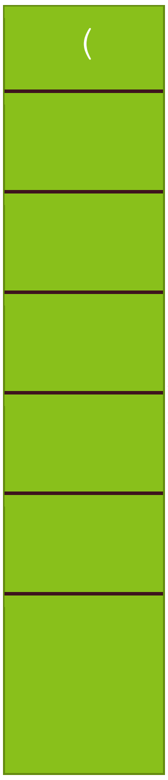
```
balanced : true inc
```

Expression: `(w * [x +`
`z)`

```
0 1 2 3
```

d Pa

Matches!
Balanced still true



balanced : **true** index 10

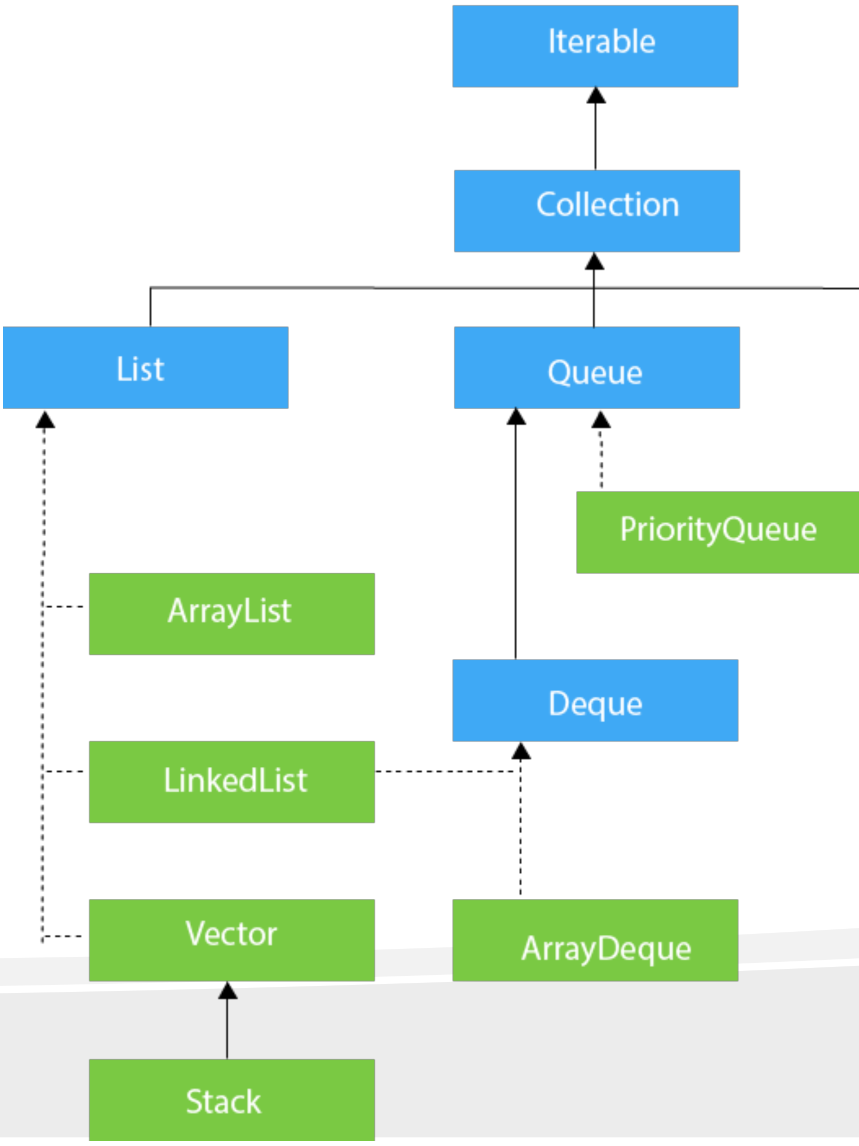
(w * [

Testing

- Provide a variety of input expr
  true or false
- Try several levels of nested pare
- Try nested parentheses where corre
  not of the same type
- Try unbalanced parentheses

- PITFALL:attempting to pop an emp
  `EmptyStackException`. You
  either testing for an empty stac
  exception

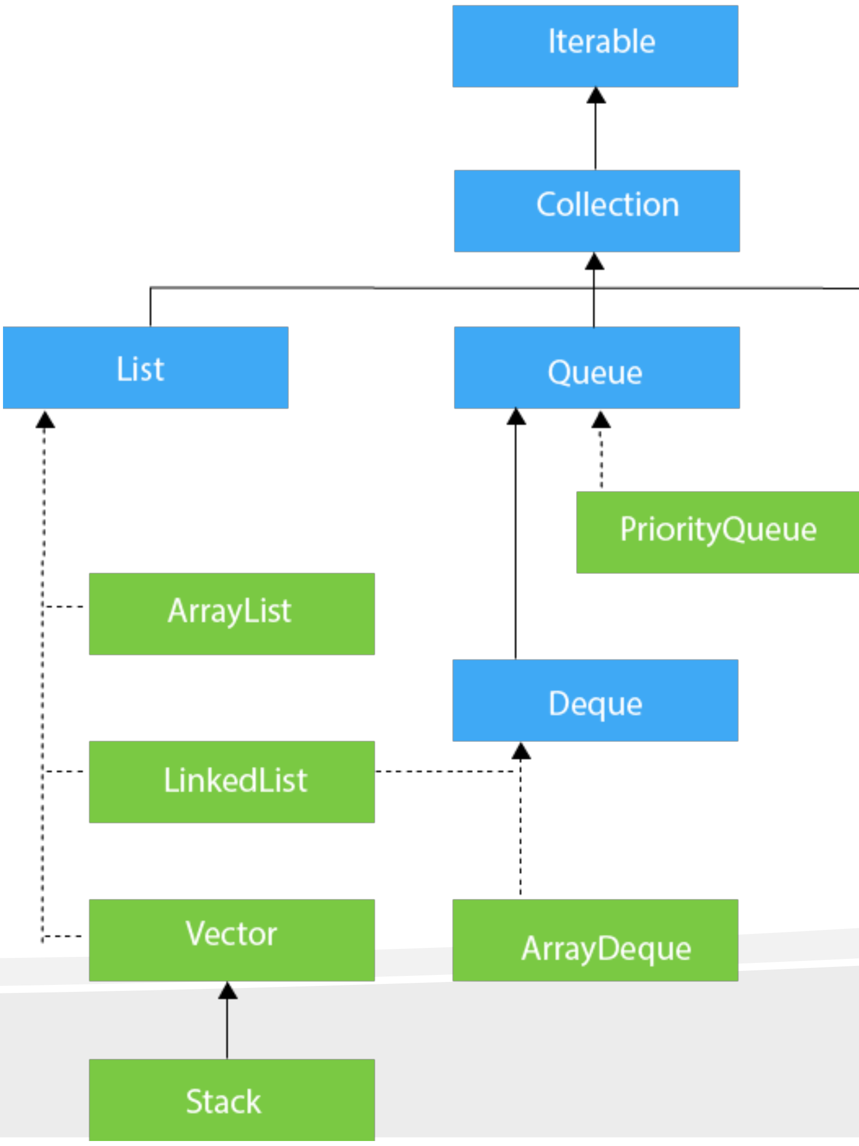
Collections Framework Diagram


Iterable
Collection
List
Queue
PriorityQueue
ArrayList
Deque
LinkedList
Vector
ArrayDeque
Stack

Implementing a Stack a
# of Vector (cont.)

- Becausea `Stack` *is a* `Vecto` operations can be applied searches and access by inde
- But, since only the top elemen be accessible, this violates information hiding

Review: Collections Framework D



Iterable
Collection
List
Queue
PriorityQueue
ArrayList
Deque
LinkedList
Vector
ArrayDeque
Stack

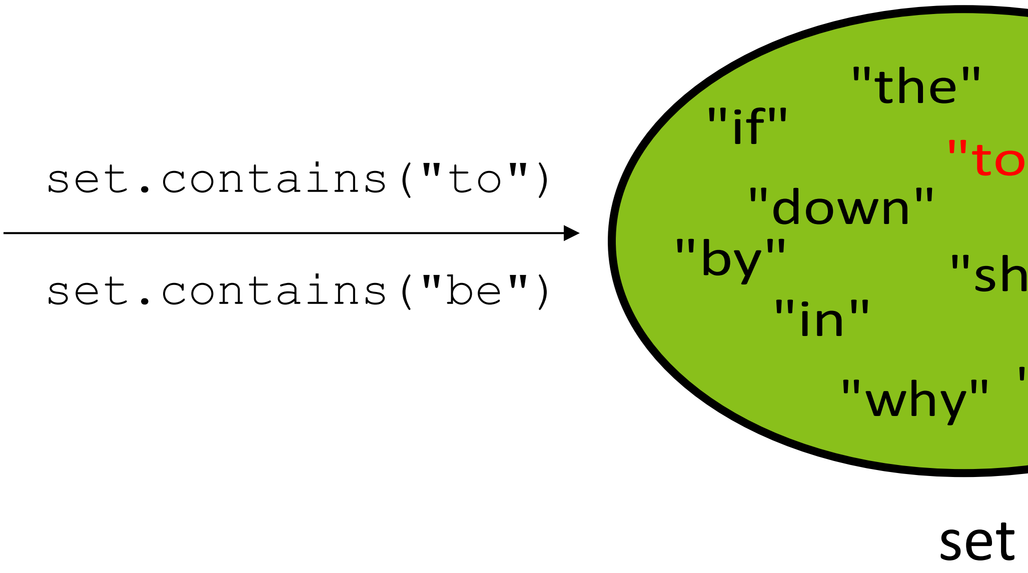
Sets

Words in a book

- Write an application that reads in the text of a boo user type words, and tells whether those words a
- How would we implement this with a List?

Sets

- **Set**: A collection of unique values (no duplicates allow operations efficiently:
  - `add`, `remove`, search (`contains`)
  - We don't think of a set as having indexes; we just adc worry about order



```
set.contains("to")
set.contains("be")
```