Lesson 3

Stacks

Chapter Objectives

- To learn about the stack data type and how to use its four methods:
  - push
  - pop
  - peek
  - empty
- To understand how Java implements a stack
- To learn how to implement a stack using an underlying array or linked list
- To see how to use a stack to perform various applications, including finding palindromes, testing for balanced (properly nested) parentheses, and evaluating arithmetic expressions

Stack Abstract Data Type

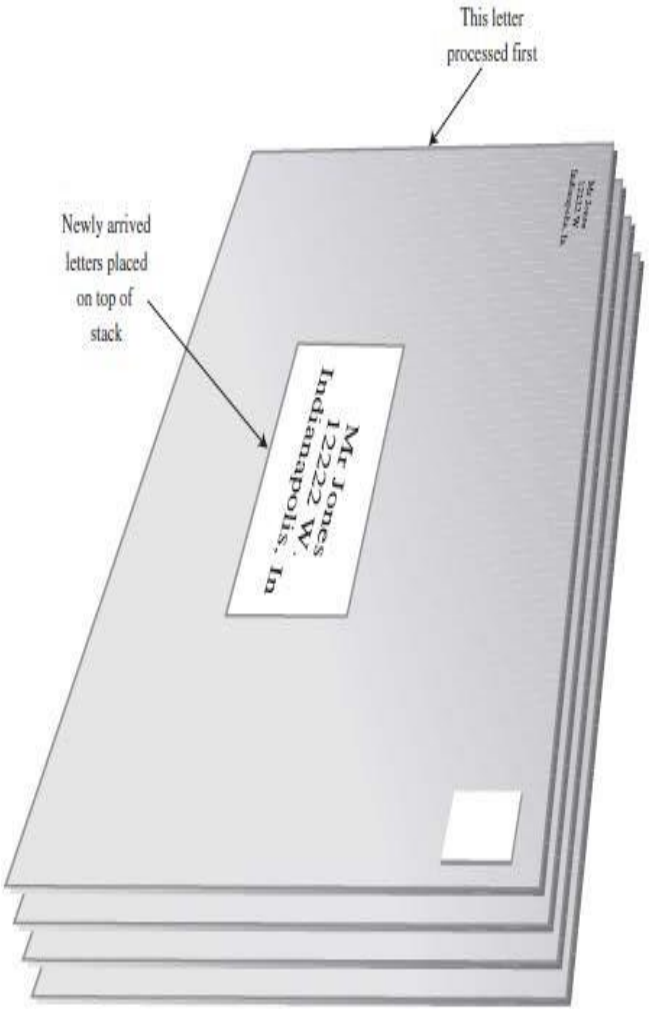
Section 3.1

Stack Abstract Data Type

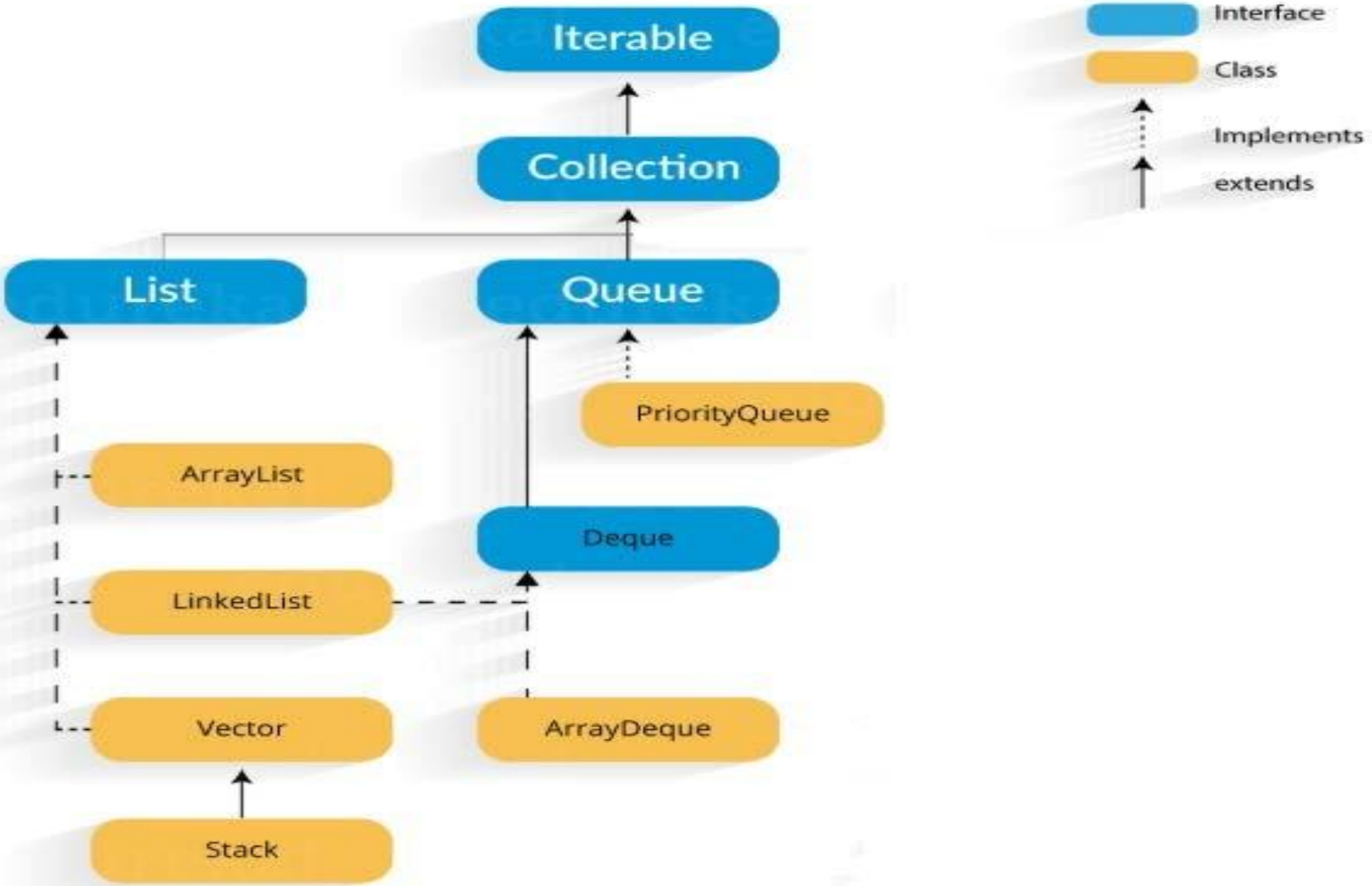
- A stack is one of the most commonly used data structures in computer science
- A stack can be compared to a Pez dispenser
  - Only the top item can be accessed
  - You can extract only one item at a time
- The top element in the stack is the last added to the stack (most recently)
- The stack's storage policy is *Last-In, First- Out*, or *LIFO*

stacks


Fisher-Price


This letter
processed first
Newly arrived
letters placed
on top of
stack
Mr James
1222 W.
Indianapolis, In

Stack and Queue API Hierarchy



Class `Stack` API from Collection Framework

`Stack<`**E**`>()`	constructs a new stack with elements of type **E**
`push(`**value**`)`	places given value on top of stack
`pop()`	removes top value from stack and returns it; throws `EmptyStackException` if stack is empty
`peek()`	returns top value from stack without removing it; throws `EmptyStackException` if stack is empty
`size()`	returns number of elements in stack
`isEmpty()`	returns `true` if stack has no elements

```
Stack<Integer> s = new Stack<Integer>();
s.push(42);
s.push(-3);
s.push(17);       // bottom [42, -3, 17] top
System.out.println(s.pop()); // 17
System.out.println(s.peek)); // -3
```

Specification of the Stack Abstract Data Type

- Only the top element of a stack is visible; therefore the number of operations performed by a stack are few
- 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 (push)

Refer : **package w2l3;** ArrayStack.java,StackInt.java

Methods	Behavior
`boolean empty()`	Returns **`true`** if the stack is empty; otherwise, returns **`false`**.
`E peek()`	Returns the object at the top of the stack without removing it.
`E pop()`	Returns the object at the top of the stack and removes it.
`E push(E obj)`	Pushes an item onto the top of the stack and returns the item pushed.

Stack Operations

Method	Return Value	Stack Contents
push(5)	–	(5)
push(3)	–	(5, 3)
size( )	2	(5, 3)
pop( )	3	(5)
isEmpty( )	false	(5)
pop( )	5	( )
isEmpty( )	true	( )
pop( )	null	( )
push(7)	–	(7)
push(9)	–	(7, 9)
peek ( )	9	(7, 9)
push(4)	–	(7, 9, 4)
size( )	3	(7, 9, 4)
pop( )	4	(7, 9)
push(6)	–	(7, 9, 6)
push(8)	–	(7, 9, 6, 8)
pop( )	8	(7, 9, 6)

Implementing a Stack Using an Array

- If we implement a stack as an array, we would need . . .

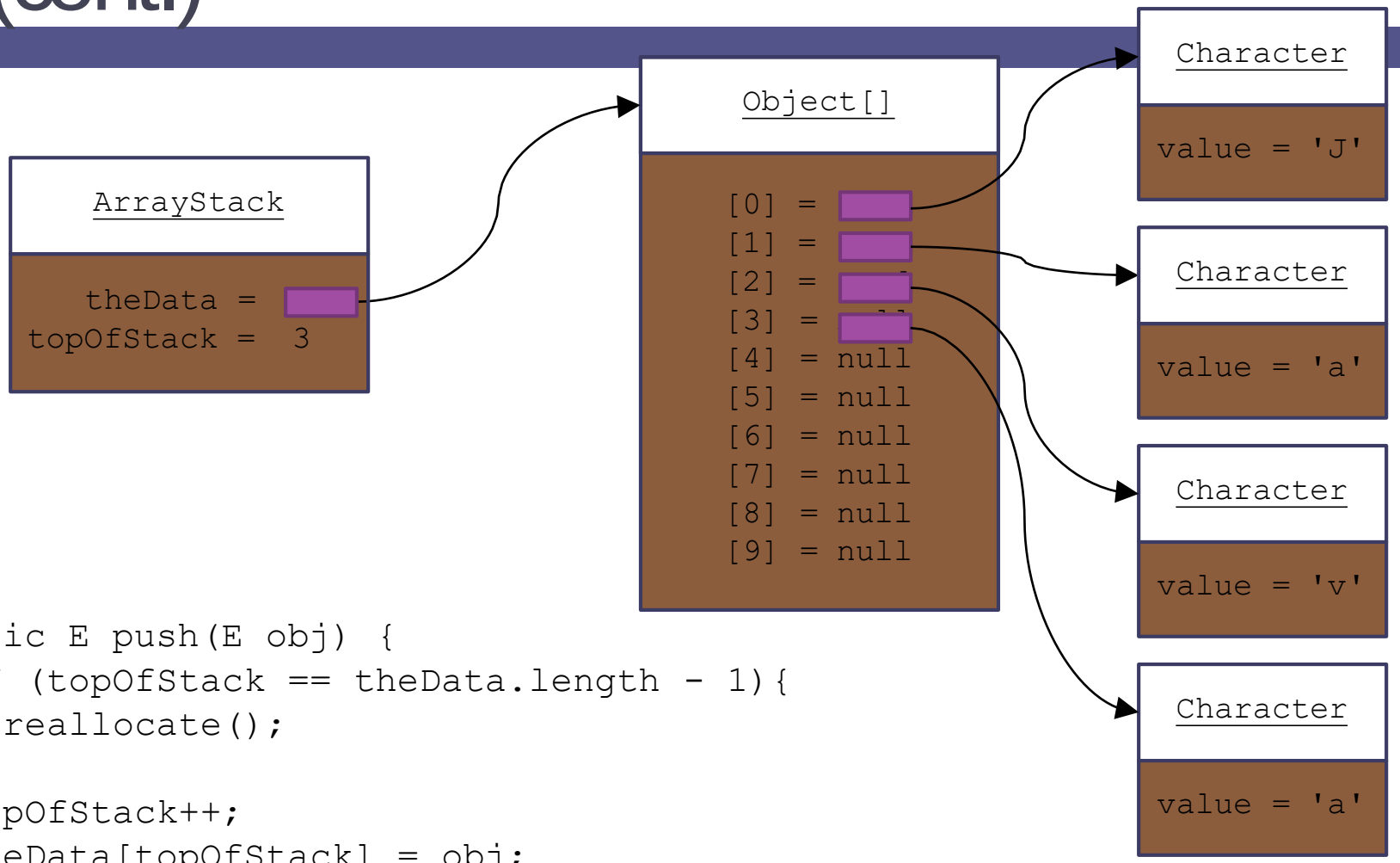
```
public class ArrayStack<E> implements StackInt<E> {
  private E[] theData;
  int topOfStack = -1;
  private static final int INITIAL_CAPACITY = 10;

  @SupressWarnings("unchecked")
  public ArrayStack() {
    theData = (E[])new Object[INITIAL_CAPACITY];
  }
```

Allocate storage for an array with a default capacity

Keep track of the top of the stack (subscript of the element at the top of the stack; for empty stack = -1)

There is no `size` variable or method

Implementing a Stack Using an Array (cont.)



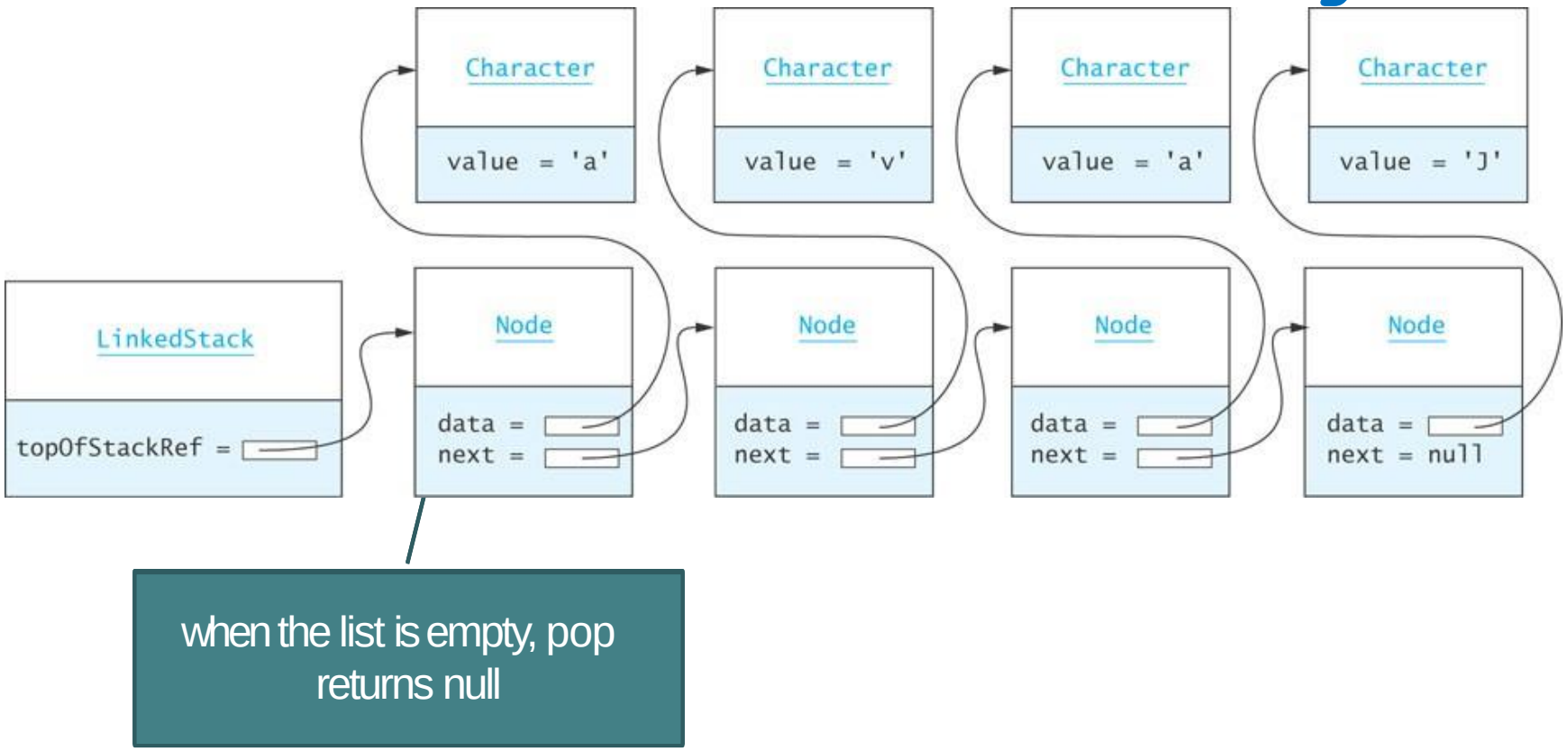

```
public E push(E obj) {
  if (topOfStack == theData.length - 1){
    reallocate();
  }
  topOfStack++;
  theData[topOfStack] = obj;
  return obj;
}
```

Implementing a Stack Using an Array (cont.)

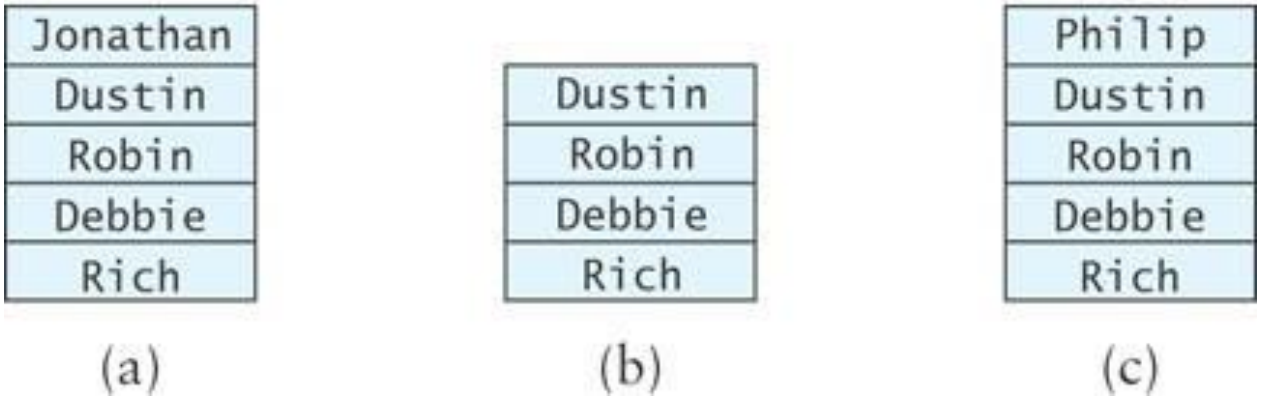
```
@Override
public E pop() {
  if (empty()) {
    throw new EmptyStackException();
  }
  return theData[topOfStack--];
}
```

Implementing a Stack as a Linked Data Structure

- We can also implement a stack using a linked list of nodes. Refer **`LinkedStack.java`**



A Stack of Strings



- "Rich" is the oldest element on the stack and "Jonathan" is the youngest (Figure a)
- `String last = names.peek();` stores a reference to "Jonathan" in `last`
- `String temp = names.pop();` removes "Jonathan" and stores a reference to it in `temp` (Figure b)
- `names.push("Philip");` pushes "Philip" onto the stack (Figure c)

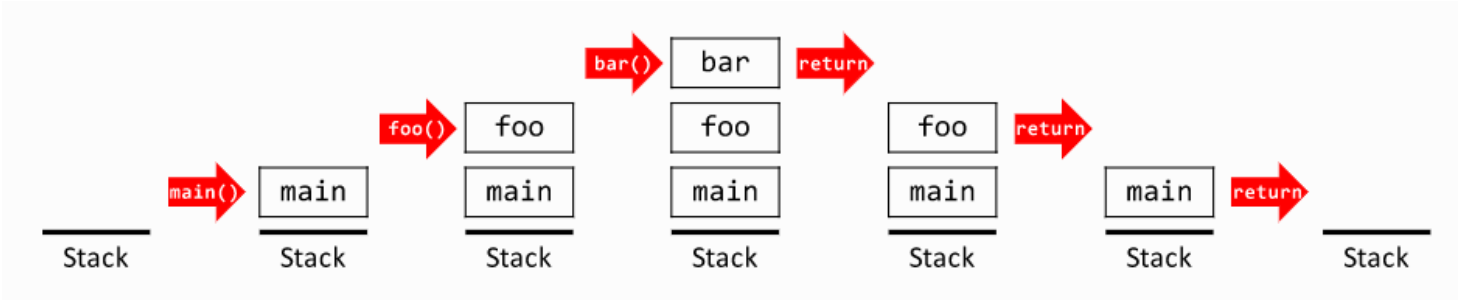
Stack Applications

Section 3.2

Programming languages and compilers

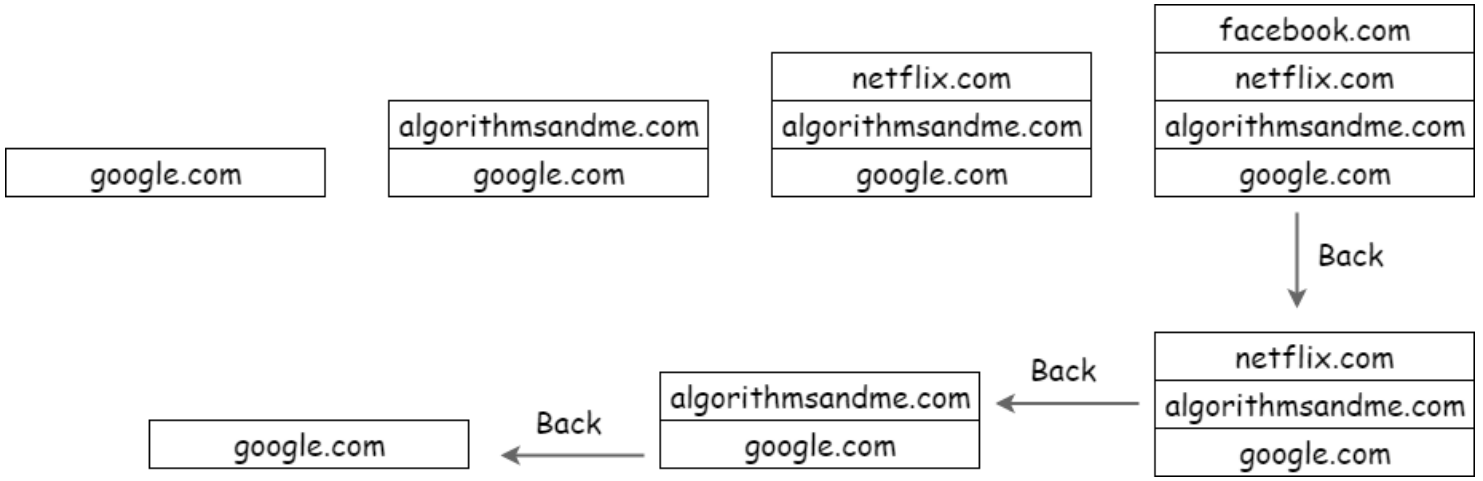
method calls are placed onto a stack (call=push, return=pop)

compilers use stacks to evaluate expressions



Back/Forward stacks on browsers

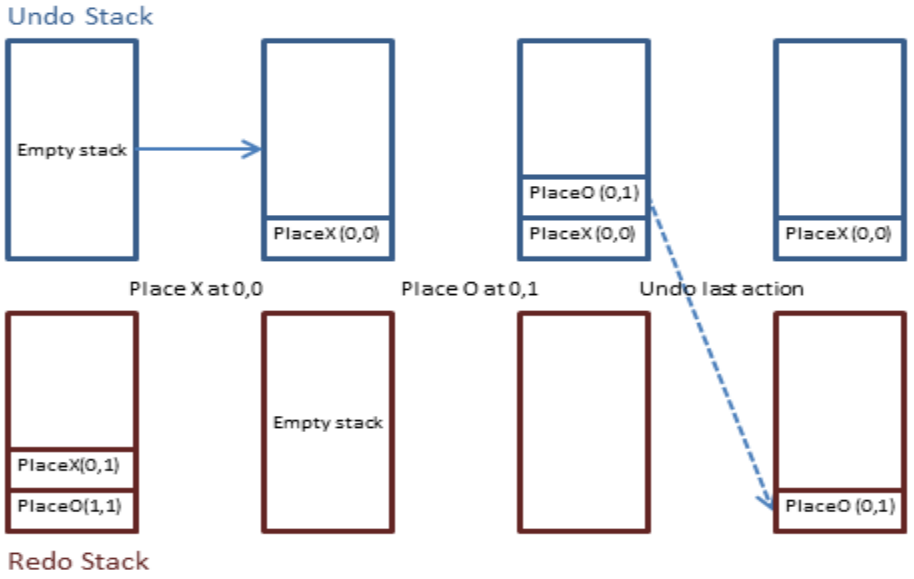
When you go back, you want to land on the last site you visited. What pattern is this? Of course, it last in first out which can be implemented using stack.



Undo/redo operations

An “undo” mechanism in text editors; this operation is accomplished by keeping all text changes in a stack:

Undo/Redo stacks in Excel or Word.



Finding Palindromes

- Palindrome: a string that reads identically in either direction, letter by letter (ignoring case)
  - kayak
  - "I saw I was I"
  - “Able was I ere I saw Elba”
  - "Level madam level"

- Problem: Write a program that reads a string and determines whether it is a palindrome

Balanced Parentheses

- When analyzing arithmetic expressions, it is important to determine whether an expression is balanced with respect to parentheses

```
( a + b * ( c / ( d – e ) ) ) + ( d / e )
```

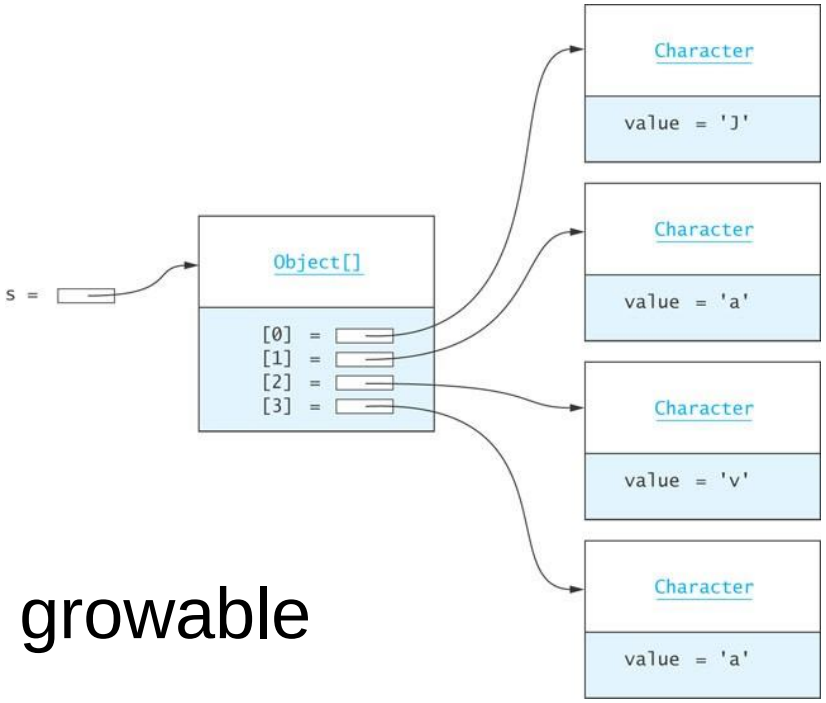
- The problem is further complicated if braces or brackets are used in conjunction with parentheses
- The solution is to use stacks!

Implementing a Stack

Section 3.3

Implementing a Stack as an Extension of Vector

- The Java API includes a `Stack` class as part of the `java.util` package.
  `public class Stack<E> extends Vector<E>`
- The `Vector` class implements a growable array of objects
- Elements of a `Vector` can be accessed using an integer index and the size can grow as needed to accommodate the insertion and removal of elements



Implementing a Stack as an Extension of Vector (cont.)

- We can use `Vector`'s `add` method to implement `push`:

```
public E push(obj E) {
  add(obj);
  return obj;
}
```

- pop can be coded as

```
public E pop throws EmptyStackException {
  try {
    return remove (size() - 1);
  } catch (ArrayIndexOutOfBoundsException ex) {
    throw new EmptyStackException();
  }
}
```

Implementing a Stack as an Extension of Vector (cont.)

- Because a `Stack` *is a* `Vector`, all of `Vector` operations can be applied to a `Stack` (such as searches and access by index)

Comparison of Stack Implementations

- Extending a `Vector` (as is done by Java) is a poor choice for stack implementation, since all `Vector` methods are accessible
- The easiest implementation uses a `List` component (`ArrayList` is the simplest) for storing data
  - An underlying array requires reallocation of space when the array becomes full, and
  - an underlying linked data structure requires allocating storage for links
  - As all insertions and deletions occur at one end, they are constant time, O(1), regardless of the type of implementation used