Algorithms

Robert Sedgewick | Kevin Wayne

http://algs4.cs.princeton.edu

1.3 Bags, Queues, and Stacks

- *stacks*
- *resizing arrays*
- *queues*
- *generics*
- *iterators*
- *applications*

Java collections library

List interface. `java.util.List` is API for an sequence of items.

`public interface List<Item> implements Iterable<Item>`

	`List()`	*create an empty list*
`boolean`	`isEmpty()`	*is the list empty?*
`int`	`size()`	*number of items*
`void`	`add(Item item)`	*append item to the end*
`Item`	`get(int index)`	*return item at given index*
`Item`	`remove(int index)`	*return and delete item at given index*
`boolean`	`contains(Item item)`	*does the list contain the given item?*
`Iterator<Item>`	`iterator()`	*iterator over all items in the list*
`...`		

Implementations. `java.util.ArrayList` uses resizing array; `java.util.LinkedList` uses linked list.

caveat: only some
operations are efficient

Java collections library

`java.util.Stack`.

- Supports `push()`, `pop()`, and iteration.
- Extends `java.util.Vector`, which implements `java.util.List` interface from previous slide, including `get()` and `remove()`.
- Bloated and poorly-designed API (why?)

Java 1.3 bug report (June 27, 2001)

> The iterator method on java.util.Stack iterates through a Stack from the bottom up. One would think that it should iterate as if it were popping off the top of the Stack.

status (closed, will not fix)

> It was an incorrect design decision to have Stack extend Vector ("is-a" rather than "has-a"). We sympathize with the submitter but cannot fix this because of compatibility.

Java collections library

java.util.Stack.

- Supports push(), pop(), and iteration.
- Extends java.util.Vector, which implements java.util.List interface from previous slide, including get() and remove().
- Bloated and poorly-designed API (why?)

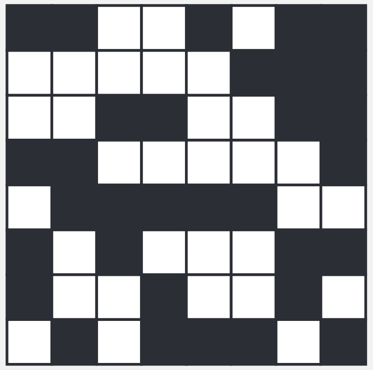
java.util.Queue. An interface, not an implementation of a queue.

Best practices. Use our implementations of Stack, Queue, and Bag.

War story (from Assignment 1)

Generate random open sites in an N-by-N percolation system.

- Jenny: pick (i, j) at random; if already open, repeat.
  Takes ~ $c_1 N^2$ seconds.
- Kenny: create a `java.util.ArrayList` of N^2 closed sites.
  Pick an index at random and delete.
  Takes ~ $c_2 N^4$ seconds.

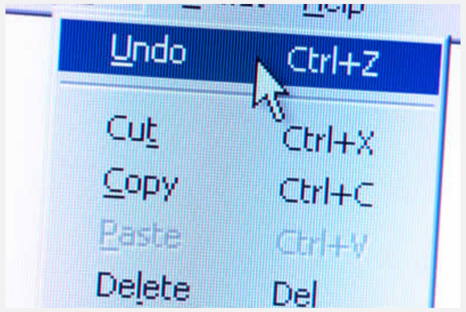


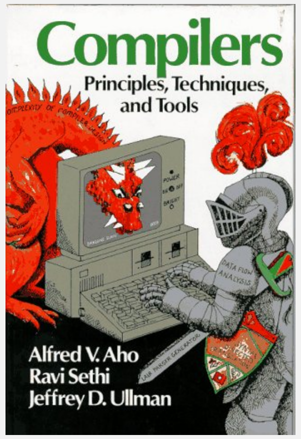
Lesson. Don't use a library until you understand its API!

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

Stack applications

- Parsing in a compiler.
- Java virtual machine.
- Undo in a word processor.
- Back button in a Web browser.
- PostScript language for printers.
- Implementing function calls in a compiler.
- ...





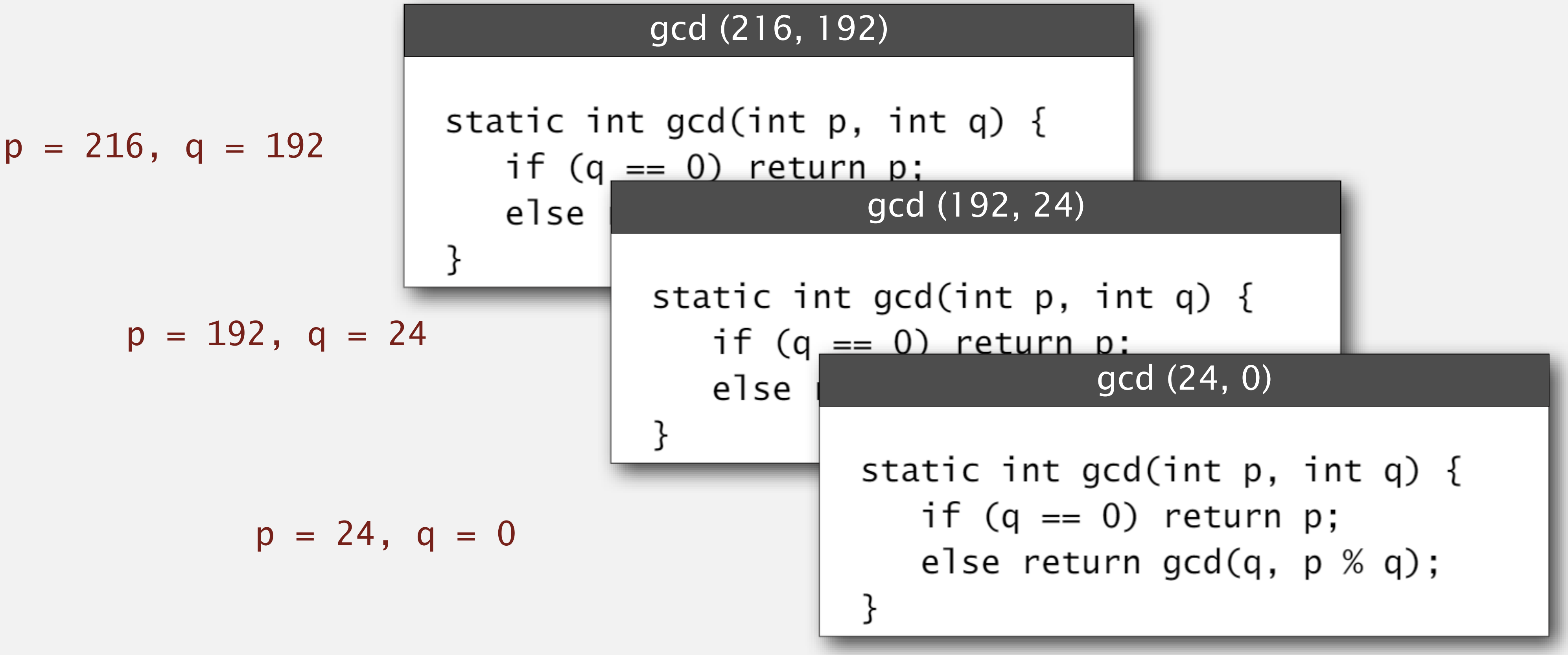
Function calls

How a compiler implements a function.

- Function call: push local environment and return address.
- Return: pop return address and local environment.

Recursive function. Function that calls itself.

Note. Can always use an explicit stack to remove recursion.



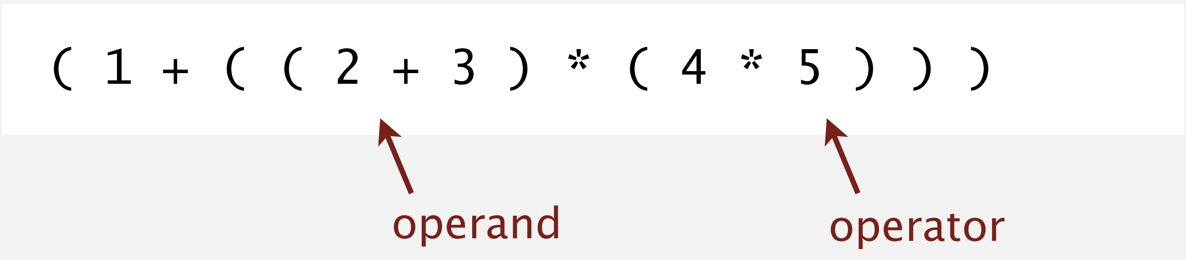
```
static int gcd(int p, int q) {
   if (q == 0) return p;
   else
}
```

```
static int gcd(int p, int q) {
   if (q == 0) return p;
   else
}
```

```
static int gcd(int p, int q) {
   if (q == 0) return p;
   else return gcd(q, p % q);
}
```

Arithmetic expression evaluation

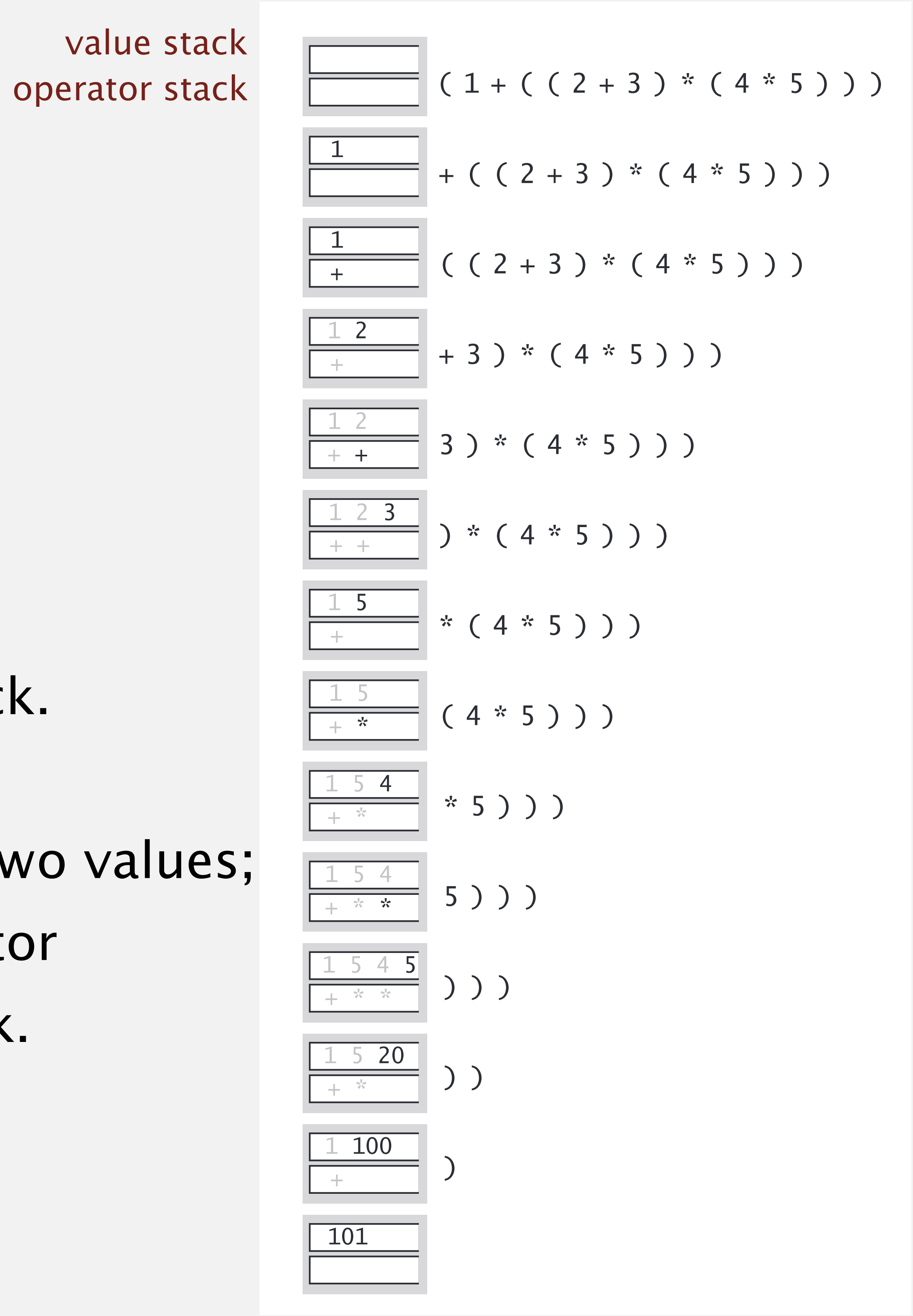
Goal. Evaluate infix expressions.



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

Two-stack algorithm. [E. W. Dijkstra]

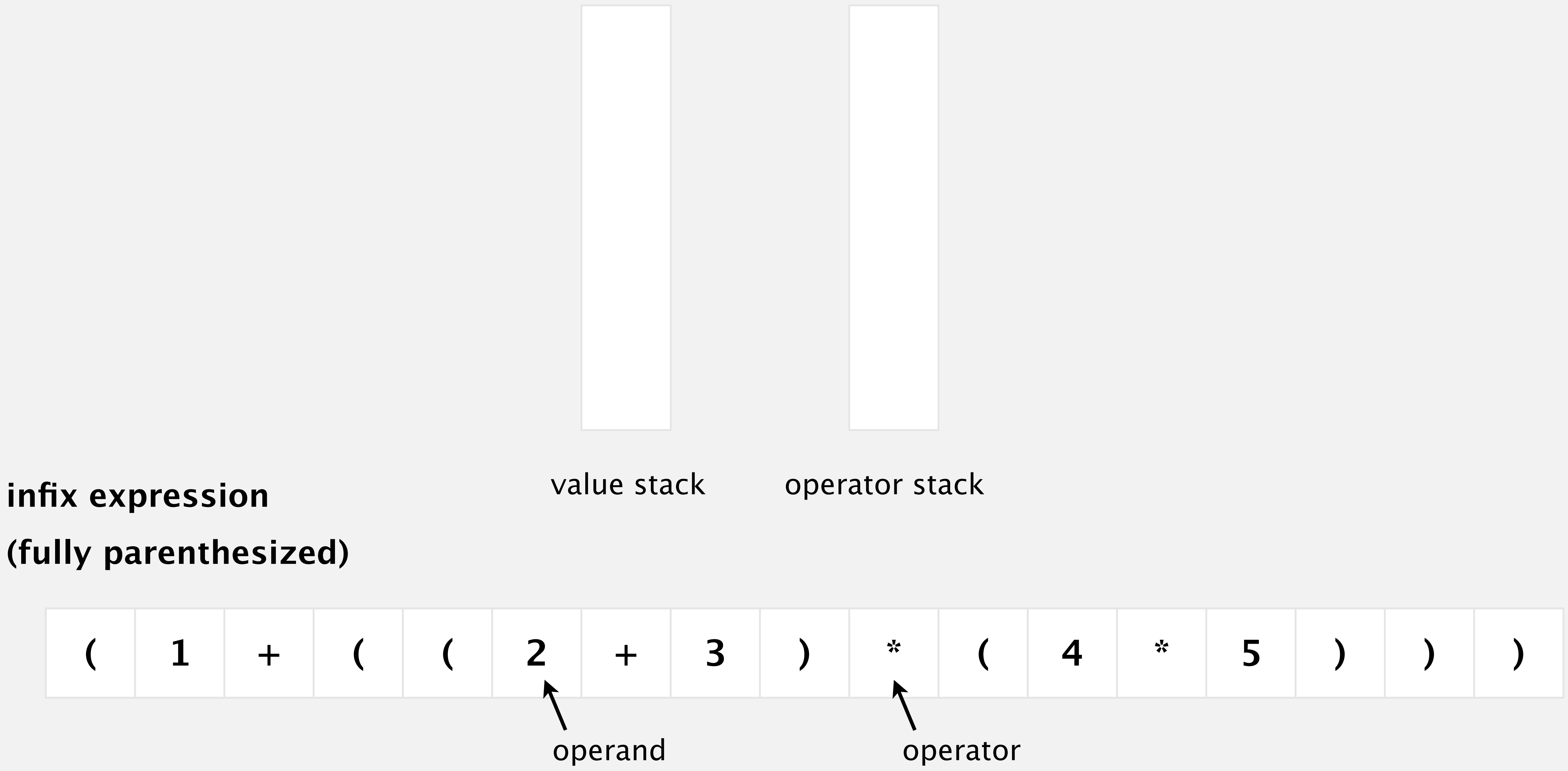
- Value: push onto the value stack.
- Operator: push onto the operator stack.
- Left parenthesis: ignore.
- Right parenthesis: pop operator and two values; push the result of applying that operator to those values onto the operand stack.



value stack	operator stack	
		(1 + ((2 + 3) * (4 * 5)))
1		+ ((2 + 3) * (4 * 5)))
1	+	((2 + 3) * (4 * 5)))
1 2	+	+ 3) * (4 * 5)))
1 2	+ +	3) * (4 * 5)))
1 2 3	+ +	) * (4 * 5)))
1 5	+	* (4 * 5)))
1 5	+ *	(4 * 5)))
1 5 4	+ *	* 5)))
1 5 4	+ * *	5)))
1 5 4 5	+ * *	)))
1 5 20	+ *	))
1 100	+	)
101		

Context. An interpreter!

Dijkstra's two-stack algorithm demo



value stack

operator stack

infix expression

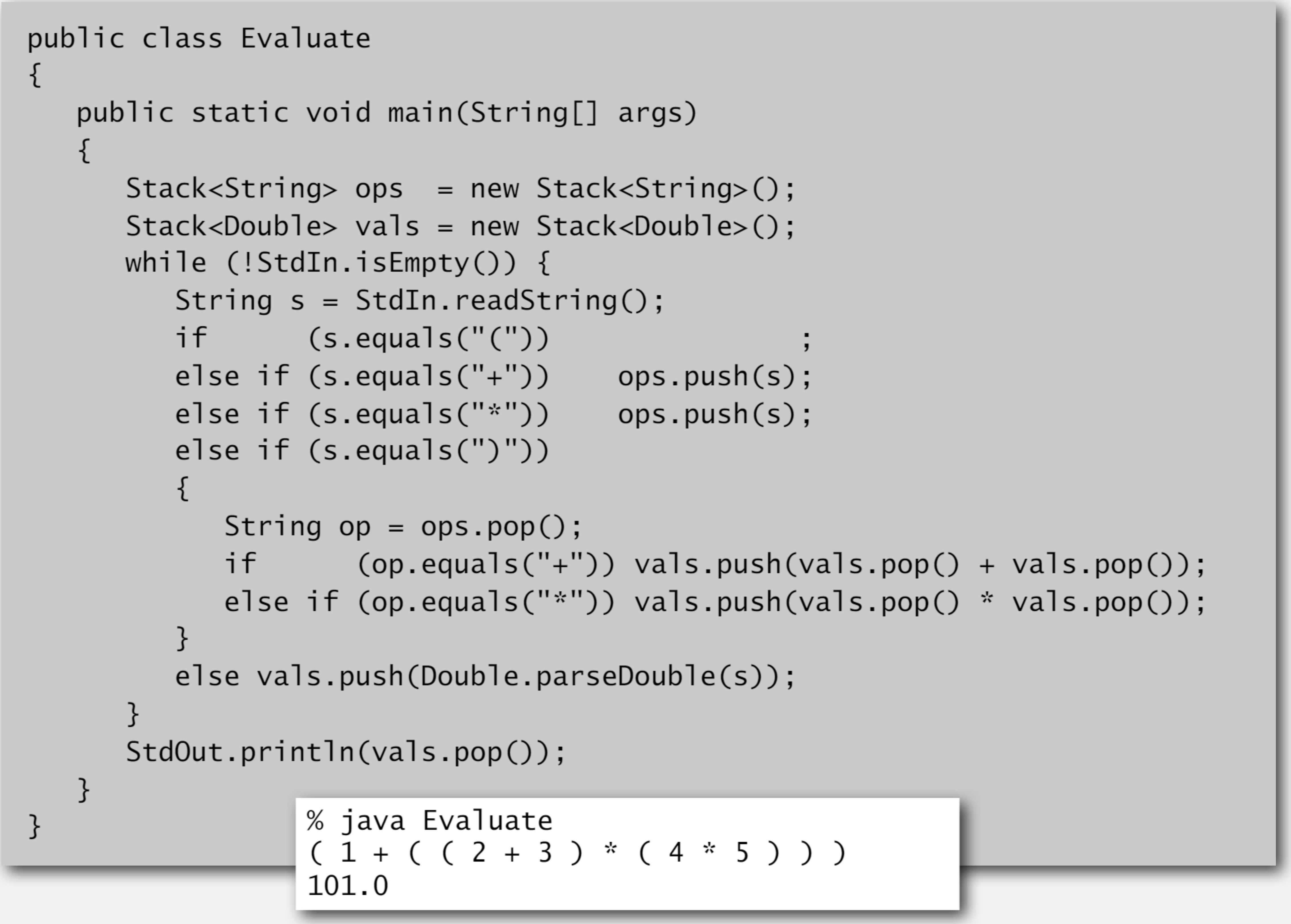
(fully parenthesized)

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

operand

operator

Arithmetic expression evaluation



```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("("))               ;
         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());
   }
}
```

```
% java Evaluate
( 1 + ( ( 2 + 3 ) * ( 4 * 5 ) ) )
101.0
```

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

as if the original input were:

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

Repeating the argument:

```
( 1 + ( 5 * 20 ) )
( 1 + 100 )
101
```

Extensions. More ops, precedence order, associativity.

Stack-based programming languages

Observation 1. Dijkstra's two-stack algorithm computes the same value if the operator occurs after the two values.

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

Observation 2. All of the parentheses are redundant!

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

Jan Lukasiewicz

Bottom line. Postfix or "reverse Polish" notation.

Applications. Postscript, Forth, calculators, Java virtual machine, ...