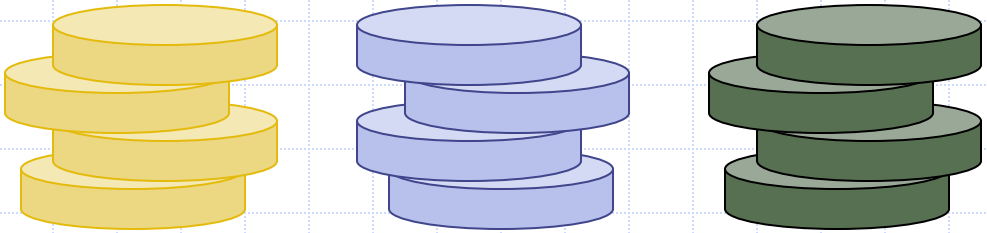
Presentation for use with the textbook Data Structures and Algorithms in Java, 6th edition, by M. T. Goodrich, R. Tamassia, and M. H. Goldwasser, Wiley, 2014

Stacks



Reading

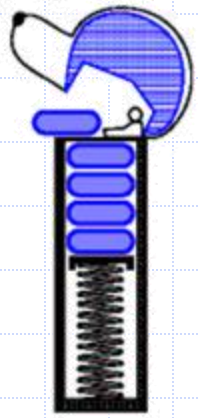
M. T. Goodrich, R. Tamassia and M. H. Goldwasser, *Data Structures and Algorithms in Java*, 6th Edition, 2014.

- **Chapter 6. Stacks and Queues**

Abstract Data Types (ADTs)

- An abstract data type (ADT) is an abstraction of a data structure
- An ADT specifies:
  - Data stored
  - Operations on the data
  - Error conditions associated with operations
- Example: ADT modeling a simple stock trading system
  - The data stored are buy/sell orders
  - The operations supported are
    - order buy(stock, shares, price)
    - order sell(stock, shares, price)
    - void cancel(order)
  - Error conditions:
    - Buy/sell a nonexistent stock
    - Cancel a nonexistent order

The Stack ADT



- The Stack ADT stores arbitrary objects
- Insertions and deletions follow the *last-in first-out* scheme
  插入和删除操作遵循 **后进先出**（LIFO, Last-In-First-Out） 的规则，也就是说，最后插入的元素最先被删除。
- Think of a spring-loaded plate dispenser
- Main stack operations:
  - push(object): inserts an element
    将一个元素插入栈的顶部。
  - object pop(): removes and returns the last inserted element
    从栈的顶部移除元素并返回它。
- Auxiliary stack operations:
  返回栈顶元素，但不移除它。
  - object top(): returns the last inserted element without removing it
  - integer size(): returns the number of elements stored
    返回栈中存储的元素数量。
  - boolean isEmpty(): indicates whether no elements are stored
    检查栈是否为空。

    此操作返回一个布尔值（true 或 false），指示栈是否为空。如果栈为空，返回 true，否则返回 false。

Stack Interface in Java

- Java interface corresponding to our Stack ADT
- Assumes null is returned from top() and pop() when stack is empty
- Different from the built-in Java class java.util.Stack

```
public interface Stack<E> {
  int size();
  boolean isEmpty();
  E top();
  void push(E element);
  E pop();
}
```

在栈为空时，top() 和 pop() 操作将返回 null。这与一些栈的实现方式不同，在这些实现中，当栈为空时可能会抛出异常。

```
public interface Stack<E> {
    int size();  // 返回栈中元素的个数

    boolean isEmpty();  // 检查栈是否为空

    E top();  // 返回栈顶元素，但不移除它

    void push(E element);  // 将元素压入栈中

    E pop();  // 从栈中移除并返回栈顶元素
}
```

Example

push: 插入尾部
size: 元素数量
pop: 移除最新加入的
isEmpty: 是否为空，空为true，非空为false
top: 返回最新加入，但不移除

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)
top( )	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)

Exceptions vs. Returning Null

- Attempting the execution of an operation of an ADT may sometimes cause an error condition
- Java supports a general abstraction for errors, called exception
- An exception is said to be “thrown” by an operation that cannot be properly executed
- In our Stack ADT, we do not use exceptions
- Instead, we allow operations pop and top to be performed even if the stack is empty
- For an empty stack, pop and top simply return null

Applications of Stacks

- Direct applications 直接应用
  - Page-visited history in a Web browser 网页浏览器中的访问历史
  - Undo sequence in a text editor 文本编辑器中的撤销序列
  - Chain of method calls in the Java Virtual Machine Java虚拟机中的方法调用链
- Indirect applications 间接应用
  - Auxiliary data structure for algorithms 作为算法的辅助数据结构
  - Component of other data structures 作为其他数据结构的组件

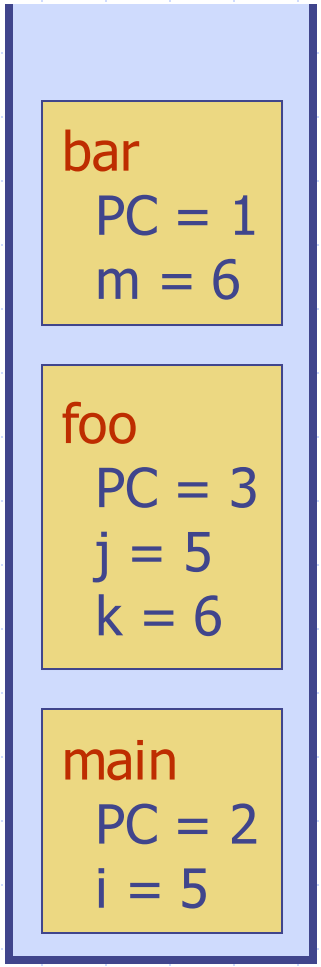
Method Stack in the JVM

- The Java Virtual Machine (JVM) keeps track of the chain of active methods with a stack
- When a method is called, the JVM pushes on the stack a frame containing
  - Local variables and return value
  - Program counter, keeping track of the statement being executed
- When a method ends, its frame is popped from the stack and control is passed to the method on top of the stack
- Allows for recursion

```
main() {
  int i = 5;
  foo(i);
  }

foo(int j) {
  int k;
  k = j+1;
  bar(k);
  }

bar(int m) {
  …
  }
```



Array-based Stack

- A simple way of implementing the Stack ADT uses an array 元素从数组的左侧开始添加，栈的操作由一个变量来管理该位置。
- We add elements from left to right
- A variable keeps track of the index of the top element

Algorithm ***size***()
  return $t + 1$

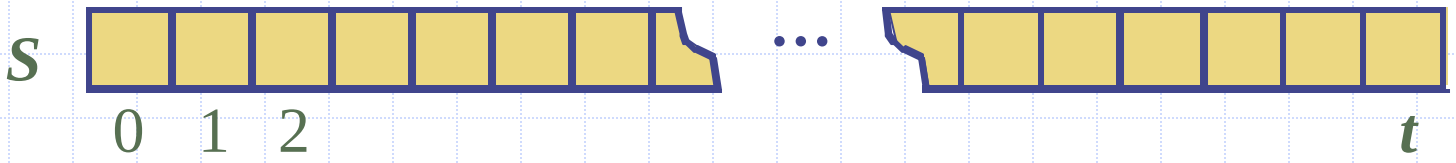
Algorithm ***pop***()
  if *isEmpty*() **then**
    return null
  else
    $t \leftarrow t - 1$
    return $S[t + 1]$

Array-based Stack (cont.)

- The array storing the stack elements may become full
- A push operation will then throw a FullStackException
  - Limitation of the array-based implementation
  - Not intrinsic to the Stack ADT

栈满时的限制：
1. 数组栈存储栈元素的数组可能会变得满。
2. 当数组中的元素数量达到了最大值，栈再也无法容纳更多元素。在这种情况下，执行 push 操作时将抛出一个异常。

Algorithm *push*(o)
  if $t = S.length - 1$ **then**
    throw *IllegalStateException*
  else
    $t \leftarrow t + 1$
    $S[t] \leftarrow o$



性能：栈的空间复杂度是 O(n)，每个操作的时间复杂度是 O(1)，表现非常高效。
限制：栈的大小是固定的，基于数组实现时，栈无法动态扩展。如果栈满了，尝试执行 push 操作会抛出异常。

Performance and Limitations

空间复杂度：假设栈中有 n 个元素，栈所占用的空间是 O(n)。这意味着栈占用的空间是与栈中元素的数量成正比的。当栈中元素的数量增加时，占用的空间也会增加。
时间复杂度：每个操作的时间复杂度是 O(1)，即每次栈的操作（如 push()、pop()、top() 等）都可以在常数时间内完成，不依赖于栈中元素的数量。

- Performance
  - Let n be the number of elements in the stack
  - The space used is $O(n)$ 空间复杂度
  - Each operation runs in time $O(1)$ 时间复杂度
- Limitations
  - The maximum size of the stack must be defined a priori and cannot be changed
  - Trying to push a new element into a full stack causes an implementation-specific exception

栈的最大大小：栈的最大大小必须在使用之前定义，并且在栈的生命周期中不能更改。这是由于基于数组的栈实现的局限性：栈的大小由数组的大小决定，因此栈的容量一开始就被固定下来，无法动态扩展。如果栈满了，无法再添加更多的元素。
栈满时的异常：当我们尝试将一个新的元素推入已满的栈时，会引发一个与实现相关的异常（例如 FullStackException）。这意味着栈有一个最大容量限制，当栈已经满时，任何新的 push 操作都会导致错误。