Fundamentals of Data Structure

Mahesh Shirole

VJTI, Mumbai-19

Slides are prepared from

1. Data Structures and Algorithms in Java, 6th edition, by M. T. Goodrich, R. Tamassia, and M. H. Goldwasser, Wiley, 2014

2.Data Structures and Algorithms in Java, by Robert Lafore, Second Edition, Sams Publishing

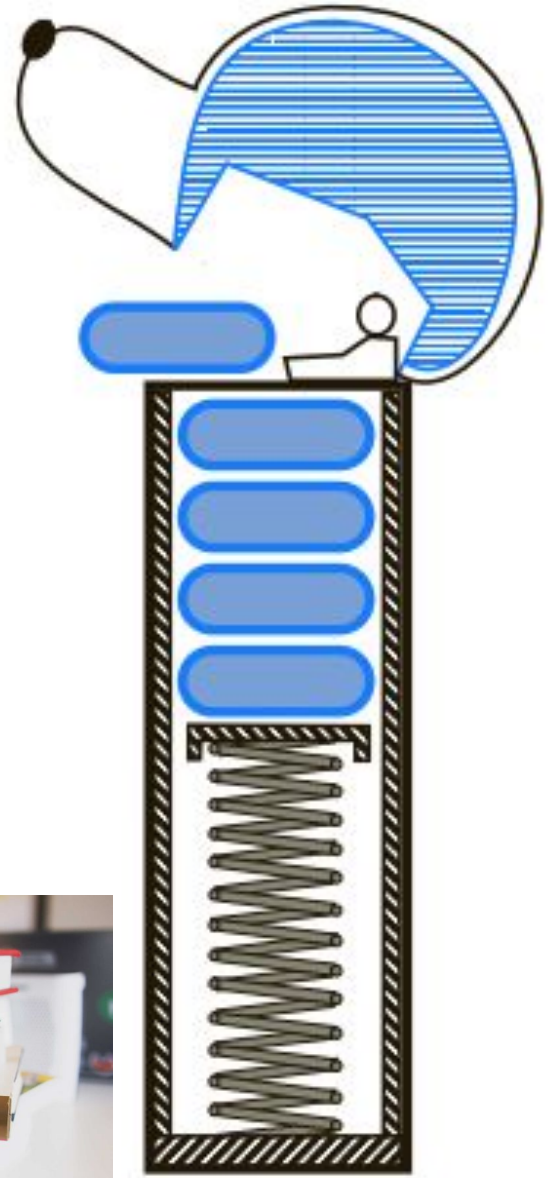
Stack

- Stacks are the simplest of all data structures
- Stacks are also among the most important, as they are used in a host of different applications, and as a tool for many more sophisticated data structures and algorithms
- A stack is a basic data structure that can be logically thought of as a linear structure represented by a real physical stack or pile
- A structure where insertion and deletion of items takes place at one end called ***top*** of the stack
- One of the applications of stack is converting a decimal number into a binary number

Stacks

- A stack is a collection of objects that are inserted and removed according to the ***last-in, first-out*** (LIFO) principle
- A stack allows *access to only one data item*: the last item inserted
- If you remove this item, you can access the next-to-last item inserted, and so on
- A user may *insert objects into a stack at any time*, but may *only access or remove the most recently inserted object* that remains (at the so-called "top" of the stack)
- The fundamental operations involve the "pushing" and "popping" of a data-element on the stack

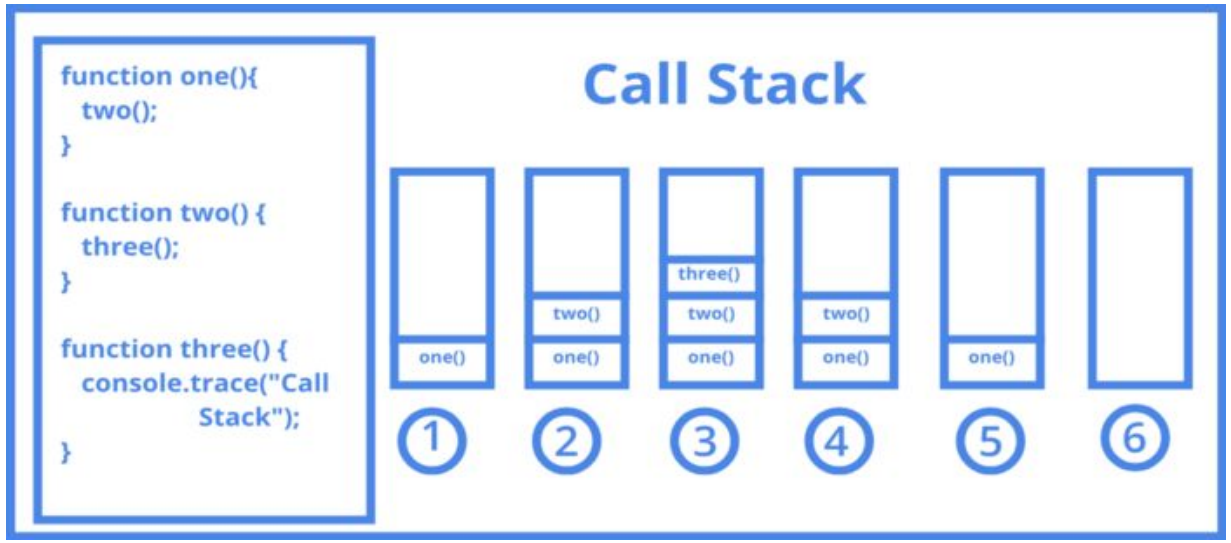
Examples of stack - general



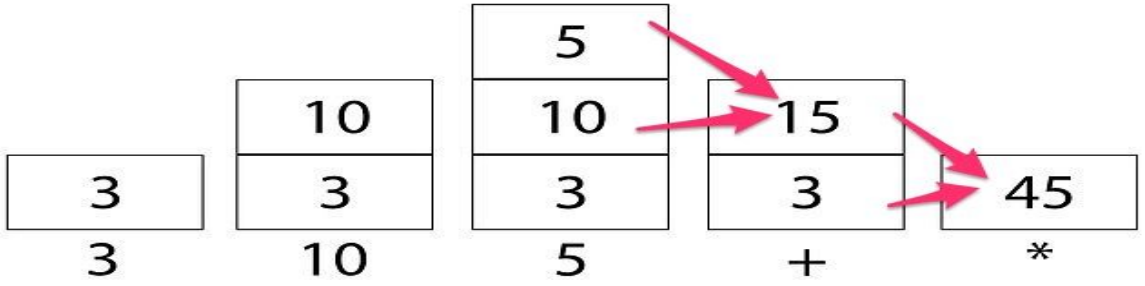
- A stack of plates in a spring-loaded, cafeteria plate dispenser
- A PEZ® candy dispenser, which stores mint candies in a spring-loaded container that “pops” out the topmost candy in the stack when the top of the dispenser is lifted
- In library, a stack of received books is restored by employee from top recent book

Stack Example- Computer Architecures

1. Microprocessor
   - Most microprocessors use a stack-based architecture
   - When a method is called, its return address and arguments are pushed onto a stack, and when it returns, they're popped off
   - The stack operations are built into the microprocessor
2. Calculators
   - Some older pocket calculators used a stack-based architecture
   - Instead of entering arithmetic expressions using parentheses, you pushed intermediate results onto a stack



Equation: 3 10 5 + *



Stack Example - Software Applications

- Internet Web browsers
  - Internet Web browsers store the addresses of recently visited sites on a stack
  - Each time a user visits a new site, that site's address is "pushed" on to the stack of addresses
  - The browser then allows the user to "pop" back to previously visited sites using the "back" button
- Text editors
  - Text editors usually provide an "undo" mechanism that cancels recent editing operations and reverts to former states of a document
  - This undo operation can be accomplished by keeping text changes in a stack

The Stack Abstract Data Type

- Formally, a stack is an abstract data type (ADT) that supports the following two update methods:
  - **push(e)**: Adds element '***e***' to the top of the stack
  - **pop()**: Removes and returns the top element from the stack (or null if the stack is empty)
- Additionally, a stack supports the following accessor methods for convenience:
  - **top()**: Returns the top element of the stack, without removing it (or null if the stack is empty)
  - **size()**: Returns the number of elements in the stack
  - **isEmpty()**: Returns a boolean indicating whether the stack is empty

A Stack Interface in Java

- Application programming interface (API) in the form of a *Java interface*, which describes the **names of the methods** that the ADT supports and how they are to be declared and used
- Java includes support for writing **generic classes** and methods that can *operate on a variety of data types* while often avoiding the need for explicit casts

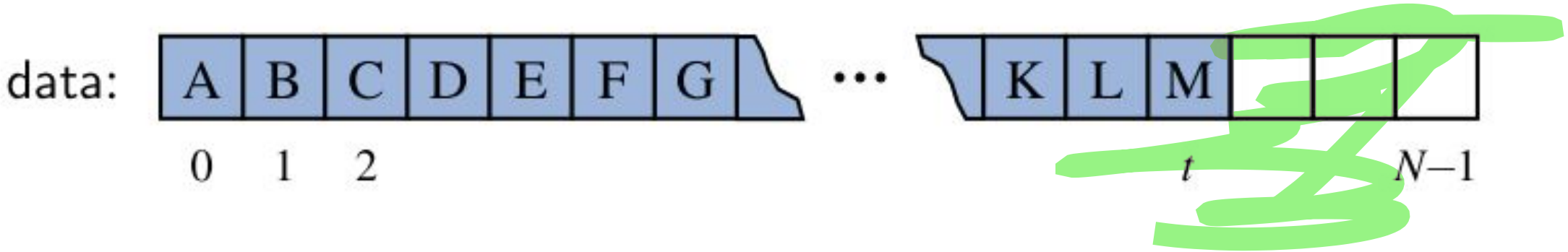
- We rely on **Java's generics framework** , allowing the elements stored in the stack to belong to any object type **<E>**
  - For example, a variable representing a stack of integers could be declared with type **Stack<Integer>**
- The formal type parameter is used as the parameter type for the push method, and the return type for both pop and top
- For the ADT to be of any use, we must provide one or more ***concrete classes*** that implement the methods of the interface associated with that ADT

Stack - Java Interface

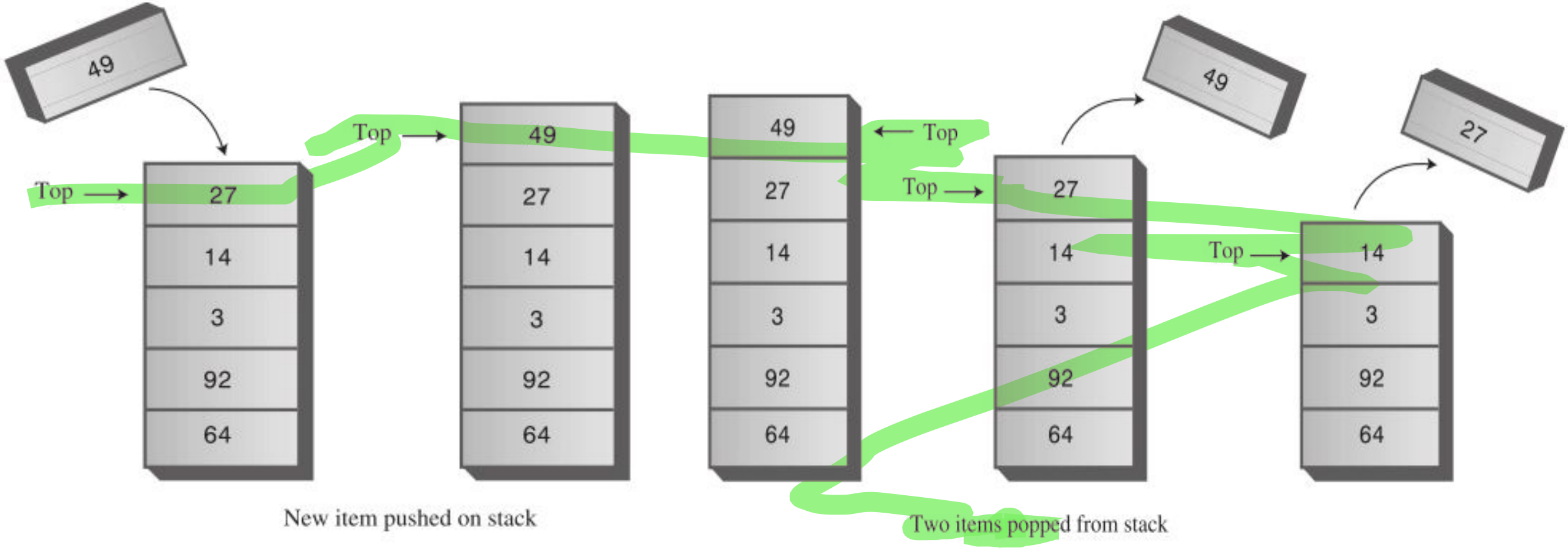
```
public interface Stack<E> {

    int size();              \\Returns the number of elements in the stack.

    boolean isEmpty();       \\return true if the stack is empty, false otherwise

    void push(E e);          \\ Inserts an element 'e' at the top of the stack

    E top();                 \\ Returns, but does not remove, the element at the
                top of the stack (or null if empty).
    E pop();                 \\ Removes and returns the top element from the
                stack (or null if empty).
}
```

A Simple Array-Based Stack Implementation

- In array based implementation of the stack ADT, we store elements in an array, named data, with **capacity N** for some fixed ***N***
- The *bottom element* of the stack is always stored in cell **data[0]**
- The *top element* of the stack in cell **data[t]** for *index t* that is equal to one less than the current size of the stack
- When the stack is empty it will have **t** equal to −1



Stack push and pop operations



```
public class ArrayStack<E> implements Stack<E> {
    private E[ ] data;                      // generic array used for storage
    private int t = −1;                     // index of the top element in stack
    public ArrayStack(int capacity) {               // constructs stack with given capacity
        data = (E[ ]) new Object[capacity]; }

    public int size() { return (t + 1); }
    public boolean isEmpty() { return (t == −1); }

    public void push(E e) throws IllegalStateException {
        if (size() == data.length) throw new IllegalStateException("Stack is full");
            data[++t] = e;  }           // increment t before storing new item
    public E top() {
        if (isEmpty()) return null;
            return data[t]; }
    public E pop() {
        if (isEmpty()) return null;
        E answer = data[t];
        data[t] = null;                     // dereference to help garbage collection
        t−−;
        return answer; }
}
```

A Drawback of This Array-Based Stack Implementation

- The array implementation of a stack is simple and efficient.
- Nevertheless, this implementation has one negative aspect—it relies on a **fixed-capacity array**, which limits the ultimate size of the stack

Analyzing the Array-Based Stack Implementation

- Each method executes a **constant number of statements** involving arithmetic operations, comparisons, and assignments, or calls to size and isEmpty, which both run in constant time.
- Thus, in this implementation of the stack ADT, each method runs in constant time, that is, they each run in ***O(1) time***.
- The **space usage is O(N)**, where N is the size of the array, determined at the time the stack is instantiated, and independent from the number ***n ≤N*** of elements that are actually in the stack.

Method	Running Time
size	$O(1)$
isEmpty	$O(1)$
top	$O(1)$
push	$O(1)$
pop	$O(1)$

Stack Example 1: Reversing a Word

- When you run the program, it asks you to type in a word. When you press Enter, it displays the word with the letters in reverse order
- A stack is used to reverse the letters
- First, the characters are extracted one by one from the input string and pushed onto the stack
- Then they're popped off the stack and displayed
- Because of its *Last-In-First-Out* characteristic, the stack reverses the order of the characters

Stack Example 2: Delimiter Matching

- The delimiters are the braces { and }, brackets [and], and parentheses (and)
- Each opening or left delimiter should be matched by a closing or right delimiter; that is, every { should be followed by a matching } and so on
- Also, opening delimiters that occur later in the string should be closed before those occurring earlier.

```
c[d]          // correct
a{b[c]d}e     // correct
a{b(c]d}e     // not correct; ] doesn't match (
a[b{c}d]e}    // not correct; nothing matches final }
a{b(c)        // not correct; nothing matches opening {
```

Stack Example 3: Matching Tags in a Markup Language

- Another application of matching delimiters is in the validation of markup languages such as HTML or XML
- HTML is the standard format for hyperlinked documents on the Internet
- XML is an extensible markup language used for a variety of structured data sets

```
<body>
<center>
<h1> The Little Boat </h1>
</center>
<p> The storm tossed the little
boat like a cheap sneaker in an
old washing machine.  The three
drunken fishermen were used to
such treatment, of course, but
not the tree salesman, who even as
a stowaway now felt that he
had overpaid for the voyage. </p>
<ol>
<li> Will the salesman die? </li>
<li> What color is the boat? </li>
<li> And what about Naomi? </li>
</ol>
</body>
```

Stack Example 4: Converting an Integer number into a binary number

- * Read a number
- * Iteration (while number is greater than zero)
- 1. Find out the remainder after dividing the number by 2
- 2. Push the remainder on stack
- 3. Divide the number by 2
- * End the iteration
- * Iteration (while stack is empty)
- 1. Pop and the element on top of stack
- * End the iteration