Data Structures and Algorithms

Chapter 2
Stacks and Queues

2.1 ADT

- An **Abstract Data Type** (**ADT**) is a specification of a collection of data and the operations (methods) that can be performed on it.
- The definition of an ADT only mentions what operations are to be performed but not how these operations will be implemented. The process of providing only the essentials and hiding the details is known as **abstraction**.
- A **data structure** is the implementation for an ADT.
- The Linked List (Chapter 1) and the Stack (chapter 2) are examples of ADTs or data structures.

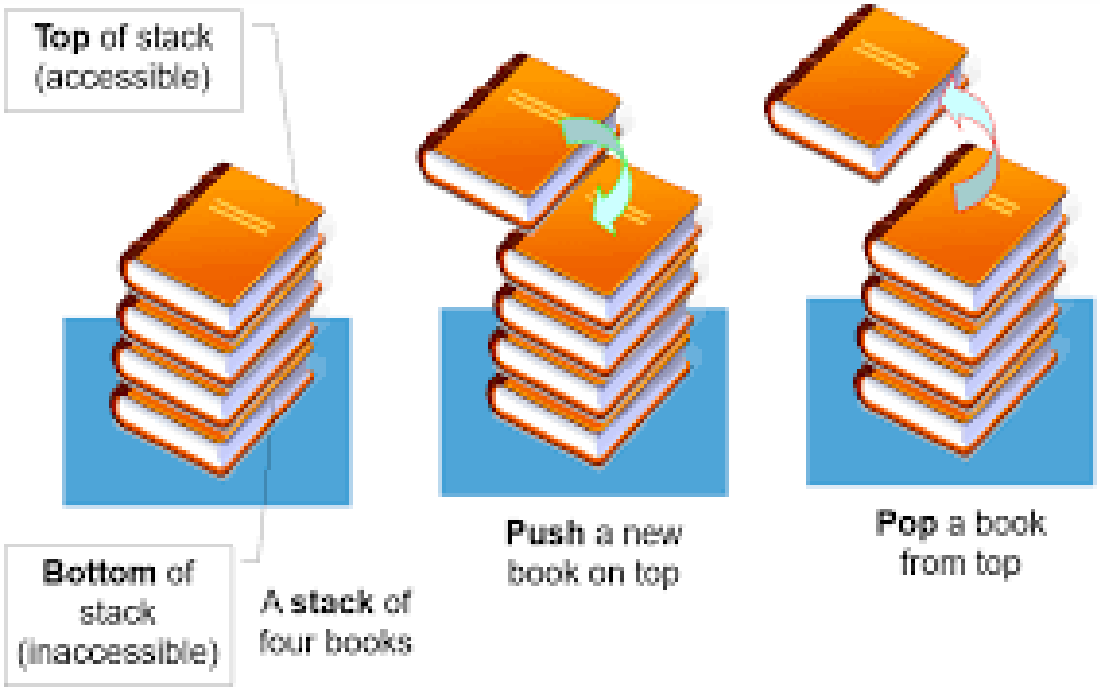
2.2 Stacks

- A **stack ADT** is an **abstract data type** that serves as a collection of elements, with two principal operations:
- **push**, which adds an element at the top of the stack.
- **pop**, which removes the element at the top of the stack (which is the most recently added element that was not yet removed).
- It is therefore a **LIFO** (Last In First Out) **ADT**.
- Note that items are added and removed from only one end of a stack which is called the top of the stack.
- Analogies: a stack of plates, a stack of coins, or a stack of books.

2.2 Stacks (Continued)

- **Stack of Plates**

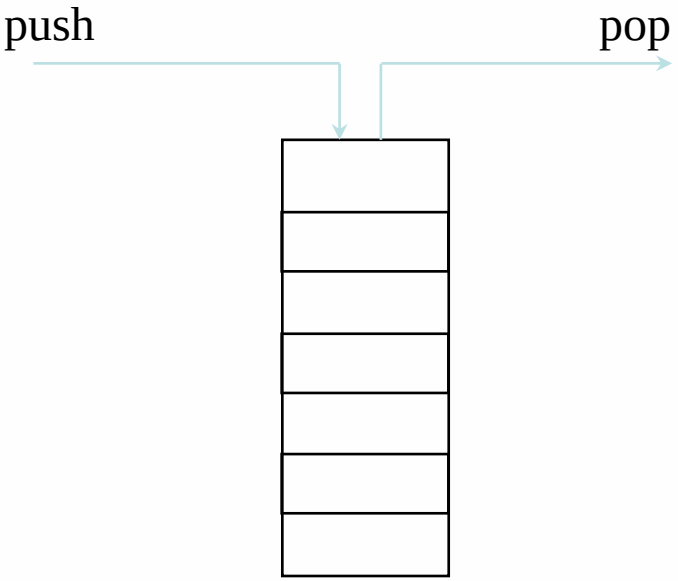
- **Stack of Books**



- **Four stacks of coins**

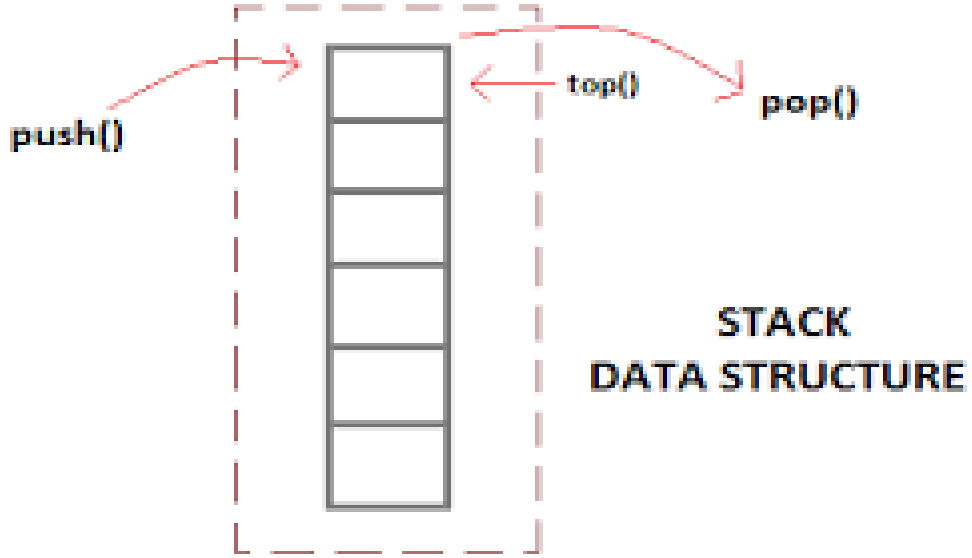
2.2 Stacks (Continued)

- Stacks often are drawn vertically:



2.2 Stacks (Continued)

- Elements are added (pushed) from the top and also removed (popped) from the top.
- We have two main operations (methods): **push()** and **pop()**.
- In addition we have an operation called **top()** which returns the top element in the stack.



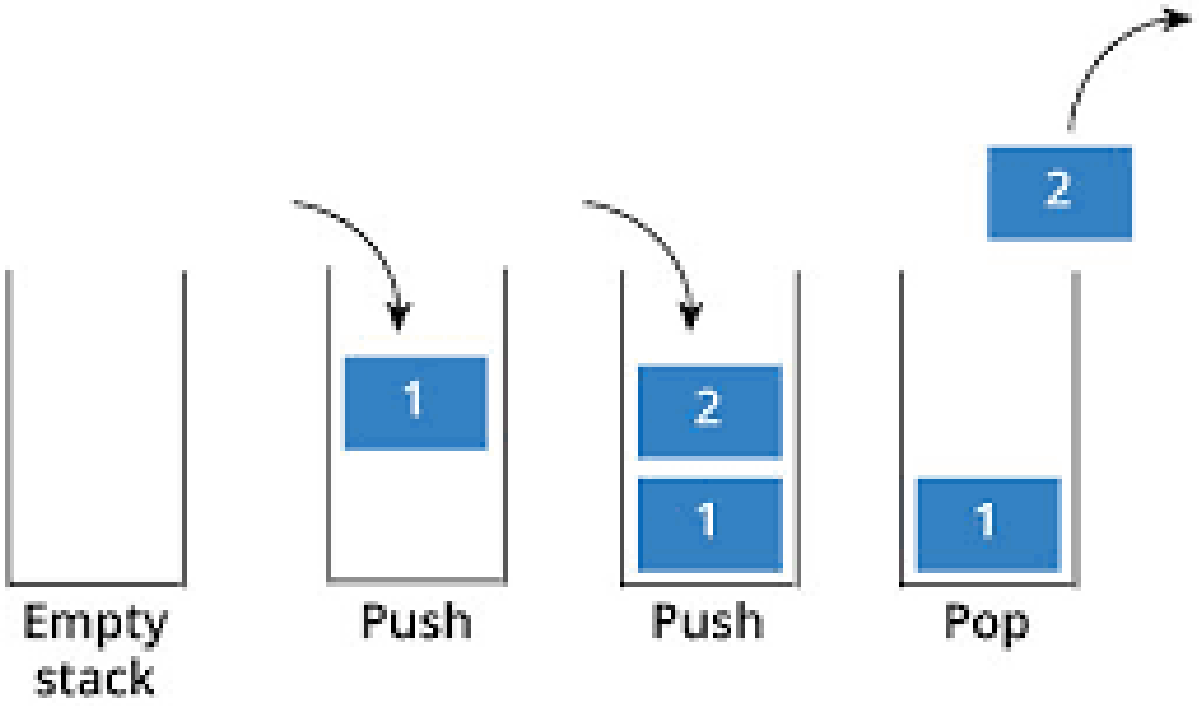
2.2 Stacks (Continued)

- Main stack operations:
  - **push** - add an item to the top of the stack
  - **pop** - remove an item from the top of the stack
  - **top** (or peek) – returns the top item without removing it
  - **isEmpty** (or empty) - returns true if the stack is empty

2.2 Stacks (Continued)

Example 1

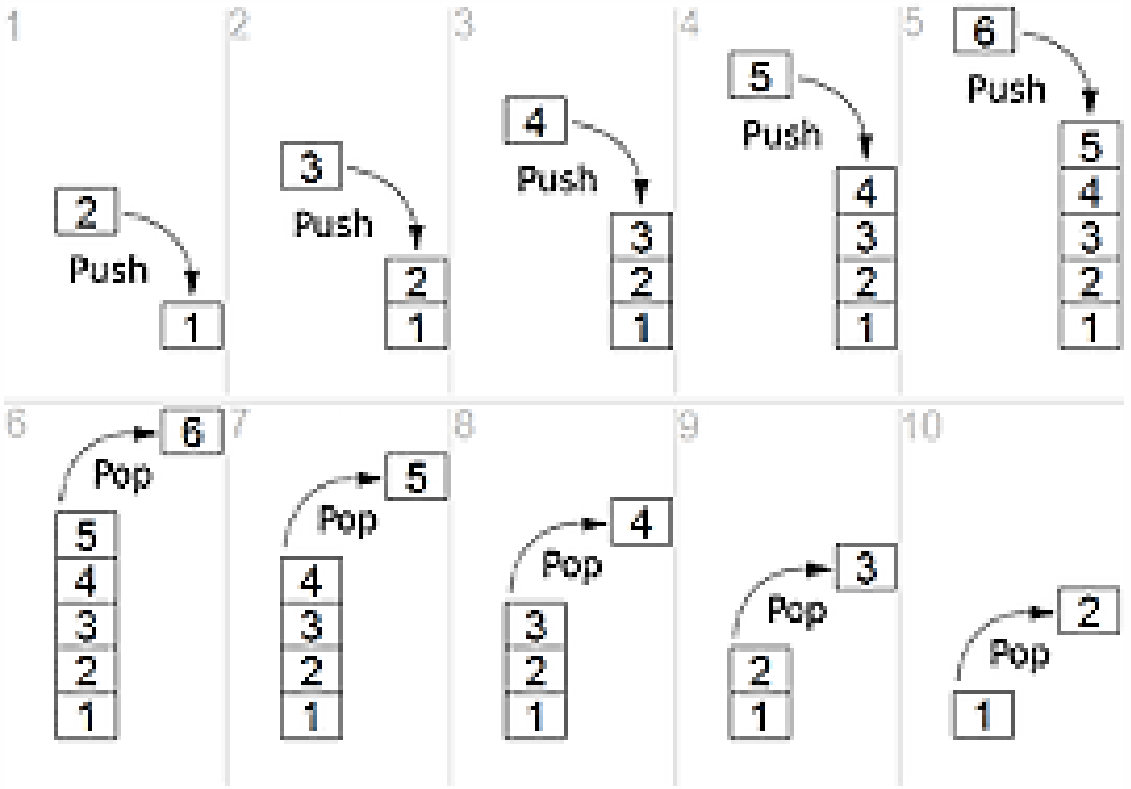
The stack is empty initially, then do the following sequence of operations: push 1, push 2, pop



2.2 Stacks (Continued)

Example 2

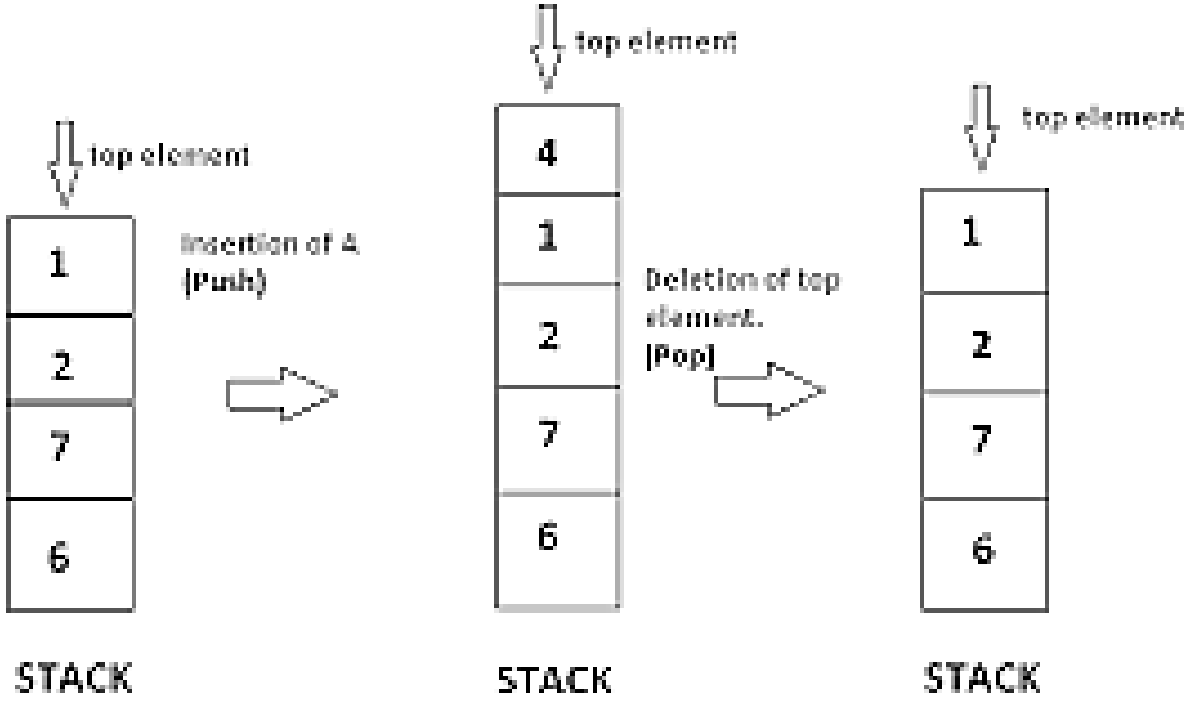
The stack contains 1 initially, then five push operations are done: push 2, push 3, push 4, push 5, push 6, followed by five pop operations.



2.2 Stacks (Continued)

Example 3

The stack contains initially the following numbers (6, 7, 2, 1) with 1 at the top, then 4 is pushed on top of 1, then a pop operation is performed and hence 1 is back at the top



2.2 Stacks (Continued)

- A stack can be represented by an array.
  This will be discussed in section **2.2.1 Implementing a stack using an array**.

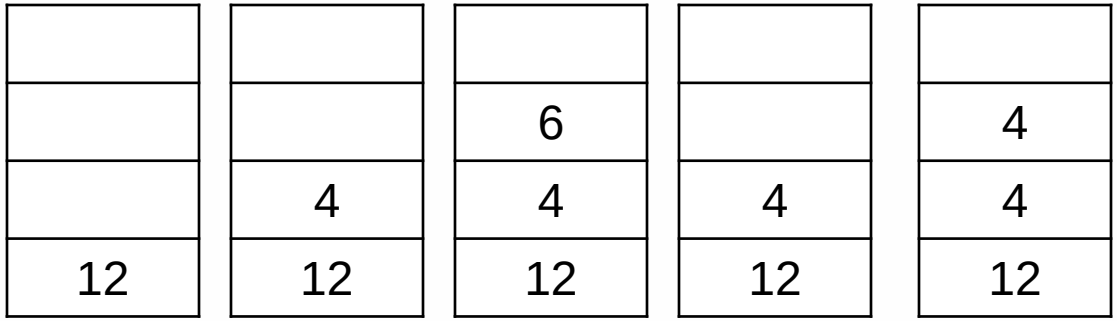
- A stack can be represented by a linked list.
  This will be discussed in section **2.2.2 Implementing a stack using a Linked List**.

2.2 Stacks (Continued)

Exercise:

```
Stack s = new Stack();
s.push(12);
s.push(4);
s.push(s.top() + 2);
s.pop();
s.push(s.top());
```

What are contents of stack after each operation?



2.2 Stacks (Continued)

- **Stack Limitations:**
- You cannot loop over a stack in the usual way.

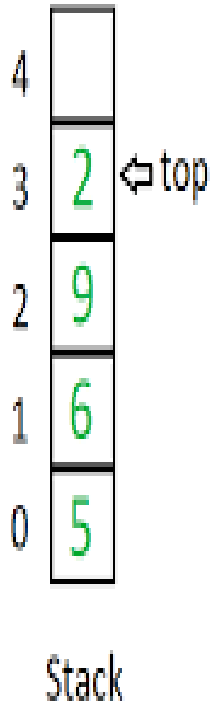
```
for (int i = 0; i < s.size(); i++) {
    do something with s.get(i);
}
```

- Instead, pop each element until the stack is empty.

```
// process (and empty) the entire stack
while (!s.isEmpty()) {
    do something with s.pop();
}
```

2.2.1 Implementing a stack using an array

- A simple way of implementing the Stack ADT uses an array.
- A variable called top keeps track of the index of the top element in the array.
- Hence the class stack will have the following to data fields assuming an array of integers:




```
public class Stack {

    private int stackArray[];
    private int stackTop;
   …
}
```

2.2.1 (Continued)

- We will add two constructors to the class Stack:

```
// This constructor creates an empty stack
public Stack() {
    stackArray = new int[100]; //can use any size (e.g. 5)
    stackTop = -1;
}

//This constructor creates an empty stack of a specific size
public Stack(int size){
    stackArray = new int[size];
    stackTop = -1;
}
```

Note: stacktop is initially -1 meaning that no elements exist in any cell. It cannot be 0 because it will mean that the top element is in cell 0.

2.2.1 (Continued)

- Operations isEmpty( ) and isFull( ):

- The stack is empty when stackTop is -1.

```
// This method checks if the stack is empty
public boolean isEmpty() {
    return (stackTop == -1);
}
```

The stack is full when stackTop is equal to the index of the last cell in the array. If the array size is 100 then the last index is 99.

```
//This method checks if the stack is isFull
public boolean isFull() {
    return (stackTop == stackArray.length-1);
}
```

2.2.1 (Continued)

- Operation push( ):

```
// Method used to add a new element at the top of the stack
public void push(int item) {
  if(!isFull()) { // if the stack still has places
     stackTop++; // move to the next index
     stackArray[stackTop] = item; // add the item
  }
  else
     System.out.println("The Stack is full.");
}
```

After moving to the next index (the index of an empty cell) using `stackTop++` then item is added in the new index.

2.2.1 (Continued)

- Operation pop( ):

```
 // Method used to remove and return the top element from the stack
public int pop() {
   if(isEmpty()) {
      System.out.println("The stack is empty.");
      System.exit(1);
   }
   return stackArray[stackTop--];
}
```

The statement `System.exit(1);` is used to exit from (stop) the whole program to avoid getting a syntax error since the method must return a value and when the stack is empty there is nothing to return.

`stackTop--` is used to decrease the index to the previous cell and this is how the top element is removed. The popped element is returned before stackTop is decremented since we are using postdecrement.

2.2.1 (Continued)

- Operation top( ):

```
// Method used to return the top element in the stack without removing it
public int top() {
   if(isEmpty()) {
      System.out.println("The stack is empty.");
      System.exit(1);
   }
   return stackArray[stackTop];
}
```

This method is similar to pop( ) with the difference that it does not remove the top element. It only returns it if the stack is not empty.