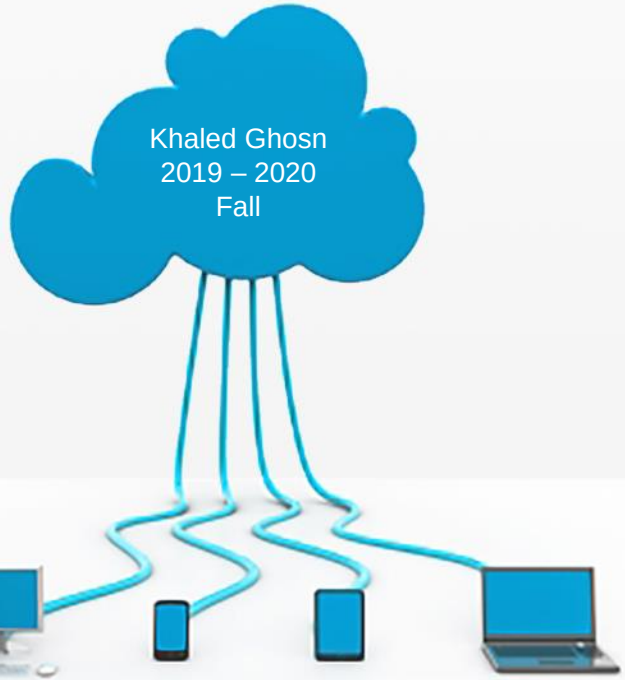
ARTS, SCIENCES & TECHNOLOGY UNIVERSITY IN LEBANON

AUL

Stacks



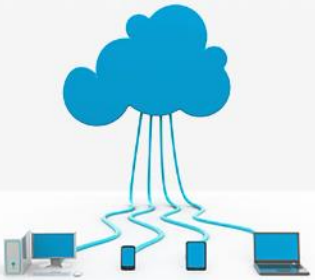
Khaled Ghosn
2019 – 2020
Fall

ADT Stack, Array Stack, Linked Stack

Stack

ADT Stack

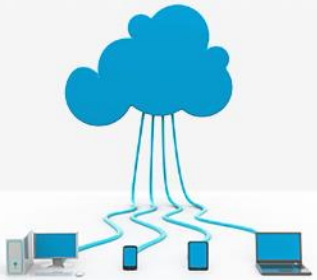
- **STACK** :
  - ✓ is an ordered collection of data items in which *access* is possible only at one end (called the **Top** of the stack)
  - ✓ is a data structure in which all insertions and deletions of entries are made at one end, called the **Top** of the stack
  - ✓ A stack has a **Top** where insertions and deletions are made
- Alternatively, in a stack the element deleted is the most recently inserted. This is also called last-in-first-out (**LIFO**)



Stack

ADT Stack

- Common stack operations:
  - ✓ Constructor
  - ✓ Push (item) – Push item to the top of the stack
  - ✓ Pop ( ) – Remove & return the top item
  - ✓ Peek ( ) – Return the top item without removing it



Stack

Push

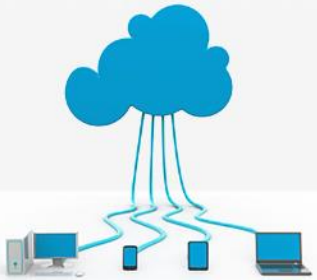
- Insert operation in a stack is often called Push
- Notice that the element pushed to a stack is always placed at the top of the stack

Pop

- Delete operation in a stack is often called Pop
- Notice that the element popped off the stack is always the one residing on top of the stack (LIFO)

Peek

- Returns the element on the top of the stack, but does not remove it



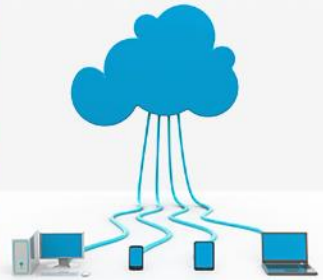
Stack

Implementation

2 ways to implement a Stack

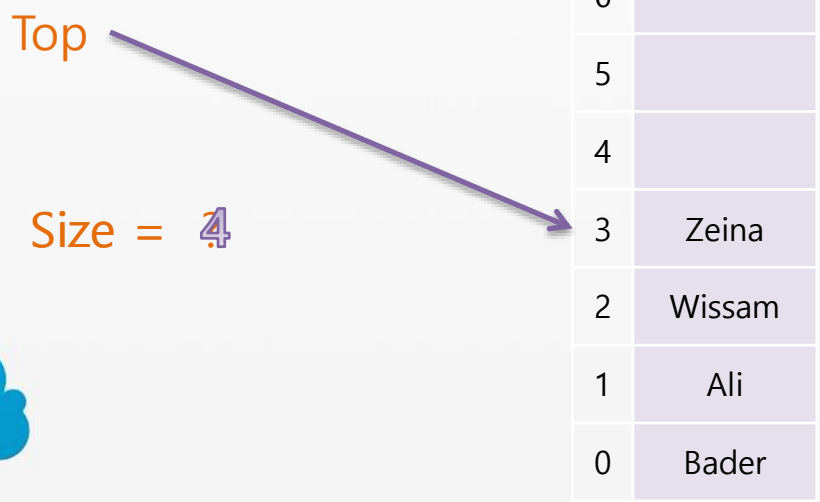
1- Using an array:
ArrayStack

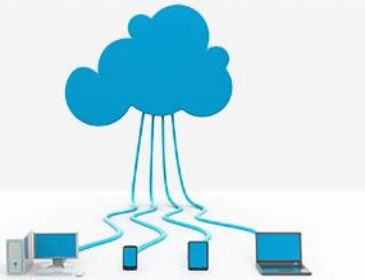
2- Using a linked list:
LinkedStack



Array Stack

ArrayStack operations





Stack
- *Anytype [ ]* **theArray** - *int* **Top** - *int* **Size**
+ **Stack** ( ) + **Stack** (*int* size) + *boolean* **isEmpty** ( ) + *boolean* **isFull** ( ) + *void* **makeEmpty** ( ) + *int* **Length** ( ) + *void* **Push** (*Anytype* value) + *Anytype* **Pop** ( ) + *Anytype* **Peek** ( ) + *void* **Print** ( )

Array Stack

Stack ( )

- ✓ **Initialize the stack**
  1. **Initialize the array**
  2. **Set Top to -1**
  3. **Set Size to 0**

Size = 0

Top

6	
5	
4	
3	
2	
1	
0	

Stack

- *Anytype* **theArray**
- *int* **Top**
- *int* **Size**

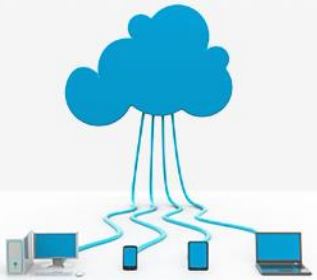
+ **Stack** ( )
+ **Stack** (*int* capacity)

+ *boolean* **isEmpty** ( )
+ *boolean* **isFull** ( )
+ *void* **makeEmpty** ( )
+ *int* **Length** ( )
+ *void* **Push** (*Anytype* value)
+ *Anytype* **Pop** ( )
+ *Anytype* **Peek** ( )
+ *void* **Print** ( )

Array Stack

Stack ( )

```
public ArrayStack()
{
    theArray = (Anytype[]) new Object [ maxSize ];
    Top = -1;
    Size = 0;
}
```

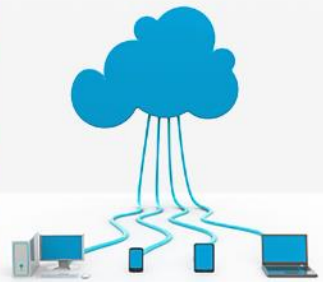


Array Stack

isEmpty ( )

- ✓ Check if Size is 0
  or if Top is -1

```
public boolean isEmpty()
{
    return Size == 0;
// return Top == -1;
}
```



Stack

- *Anytype* **theArray**
- *int* **Top**
- *int* **Size**

+ **Stack** ( )
+ **Stack** (*int* size)
+ *boolean* **isEmpty** ( )
+ *boolean* **isFull** ( )
+ *void* **makeEmpty** ( )
+ *int* **Length** ( )
+ *void* **Push** (*Anytype* value)
+ *Anytype* **Pop** ( )
+ *Anytype* **Peek** ( )
+ *void* **Print** ( )

Array Stack

isFull ( )

- ✓ Check if Size equal to array capacity
  or if Top refers to the last index in the array

```
public boolean isFull()
{
    return Size == theArray.length;
// return Top == theArray.length -1;
}
```

Stack

- *Anytype* **theArray**
- *int* **Top**
- *int* **Size**

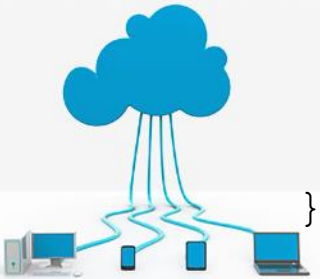
+ **Stack** ( )
+ **Stack** (*int* size)
+ *boolean* **isEmpty** ( )
+ *boolean* **isFull** ( )
+ *void* **makeEmpty** ( )
+ *int* **Length** ( )
+ *void* **Push** (*Anytype* value)
+ *Anytype* **Pop** ( )
+ *Anytype* **Peek** ( )
+ *void* **Print** ( )

Array Stack

makeEmpty ( )

1. Check if the stack is not empty
   (if empty stop here)
2. Set Top to -1
3. Set Size to 0

```
public void makeEmpty() {
    if (!isEmpty()) {
        Top = -1;
        Size = 0;
    }
}
```



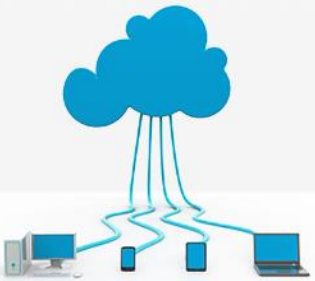
Stack
- *Anytype* **theArray** - *int* **Top** - *int* **Size**
+ **Stack** ( ) + **Stack** (*int* size) + *boolean* **isEmpty** ( ) + *boolean* **isFull** ( ) + *void* **makeEmpty** ( ) + *int* **Length** ( ) + *void* **Push** (*Anytype* value) + *Anytype* **Pop** ( ) + *Anytype* **Peek** ( ) + *void* **Print** ( )

Array Stack

Length ( )

✓ **Return number of filled elements**

```
public int Length()
{
    return Top + 1;
// return Size;
}
```



Stack

- *Anytype* **theArray**
- *int* **Top**
- ~~*int* **Size**~~

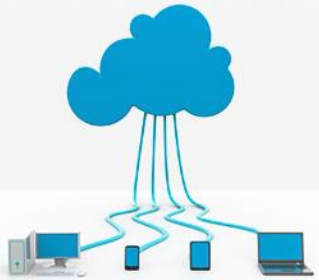
+ **Stack** ( )
+ **Stack** (*int* size)

+ *boolean* **isEmpty** ( )
+ *boolean* **isFull** ( )
+ *void* **makeEmpty** ( )
+ *int* **Length** ( )
+ *void* **Push** (*Anytype* value)
+ *Anytype* **Pop** ( )
+ *Anytype* **Peek** ( )
+ *void* **Print** ( )

Array Stack

Push (value)

1. Check if the stack is not full (if full stop here)
2. Increment Top index
3. Let the element (having the index Top) hold the value
4. Increment Size



6	
5	
4	Alaa
3	Zeina
2	Wissam
1	Ali
0	Bader

Top

Size = ~~4~~ 5

Stack

- *Anytype* **theArray**
- *int* **Top**
- *int* **Size**

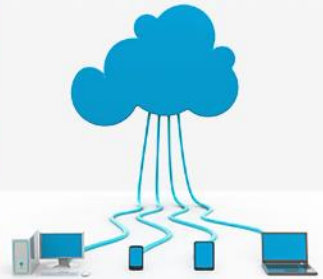
+ **Stack** ( )
+ **Stack** (*int* size)

+ *boolean* **isEmpty** ( )
+ *boolean* **isFull** ( )
+ *void* **makeEmpty** ( )
+ *int* **Length** ( )
+ *void* **Push** (*Anytype* value)
+ *Anytype* **Pop** ( )
+ *Anytype* **Peek** ( )
+ *void* **Print** ( )

Array Stack

Push (value)

```
public void Push(Anytype value)
{
    if(isFull())
        throw new RuntimeException();

    Top++;
    theArray[Top] = value;
    Size++;
}
```



Array Stack

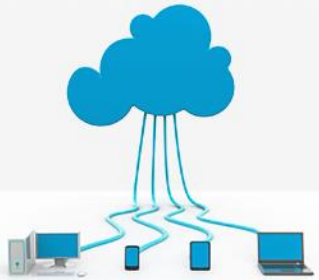
Pop ( )

1. Check if the stack is not empty (if empty stop here)
2. Decrement Top index
3. Decrement Size
4. Return the value stored in element having the old Top index

6	
5	
4	
3	Zeina
2	Wissam
1	Ali
0	Bader

Top

Size = ~~4~~ 3



Stack

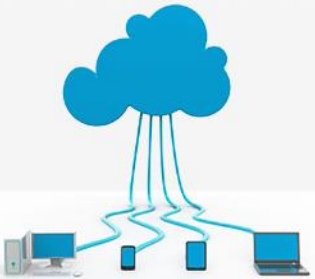
- *Anytype* **theArray**
- *int* **Top**
- *int* **Size**

+ **Stack** ( )
+ **Stack** (*int* size)
+ *boolean* **isEmpty** ( )
+ *boolean* **isFull** ( )
+ *void* **makeEmpty** ( )
+ *int* **Length** ( )
+ *void* **Push** (*Anytype* value)
+ *Anytype* **Pop** ( )
+ *Anytype* **Peek** ( )
+ *void* **Print** ( )

Array Stack

Pop ( )

```
public Anytype Pop()
{
    if (this.isEmpty())
        throw new RuntimeException();

    Top--;
    Size--;

    return theArray[Top+1];
}
```



Array Stack

Peek ( )

1. Check if the stack is not empty
   (if empty stop here)
2. Return the value stored in element having the index (*Top*)

```
public Anytype Peek()
{
    if (isEmpty())
        throw new RuntimeException();

    return theArray[Top];
}
```

Stack
- *Anytype* **theArray** - *int* **Top** - *int* **Size**
+ **Stack** ( ) + **Stack** (*int* size)
+ *boolean* **isEmpty** ( ) + *boolean* **isFull** ( ) + *void* **makeEmpty** ( ) + *int* **Length** ( ) + *void* **Push** (*Anytype* value) + *Anytype* **Pop** ( ) + *Anytype* **Peek** ( ) + *void* **Print** ( )

Array Stack

Print ( )

- ✓ **Print all elements starting from index (Top) till index 0**

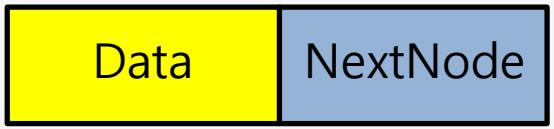
```
public void Print()
{
    for(int i=Size-1; i>=0; i--)
        System.out.println(theArray[i]);
}
```

Stack
- *Anytype* **theArray** - *int* **Top** - *int* **Size**
+ **Stack** ( ) + **Stack** (*int* size) + *boolean* **isEmpty** ( ) + *boolean* **isFull** ( ) + *void* **makeEmpty** ( ) + *int* **Length** ( ) + *void* **Push** (*Anytype* value) + *Anytype* **Pop** ( ) + *Anytype* **Peek** ( ) + *void* **Print** ( )

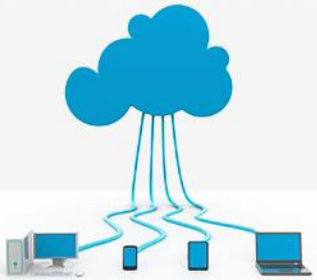
Linked Stack

Node operations

Node



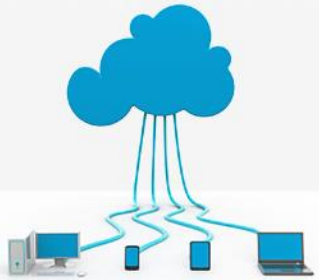
Node
- *Anytype* **Data** - *Node* **NextNode**
+ **Node** ( ) + **Node** (*Anytype* Data) + **Node** (*Anytype* Data, *Node* nextNode) + *Anytype* **getData** ( ) + *void* **setData** (*Anytype* Data) + *Node* **getNextNode** ( ) + *void* **setNextNode** (*Node* nextNode)



Linked Stack

LinkedStack operations

Stack
- *Node* **Top** - *int* **Size**
+ **Stack** ( ) + *boolean* **isEmpty** ( ) + *void* **makeEmpty** ( ) + *int* **Length** ( ) + *void* **Push** (*Anytype* value) + *Anytype* **Pop** ( ) + *Anytype* **Peek** ( ) + *void* **Print** ( )



Linked Stack

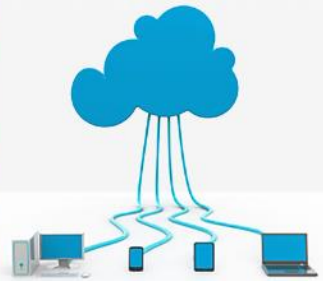
Top → null

Size = 0

Stack ( )

- ✓ Initialize the stack
  1. Set Top to null
  2. Set Size to 0

```
public LinkedStack()
{
    Top = null;
    Size = 0;
}
```



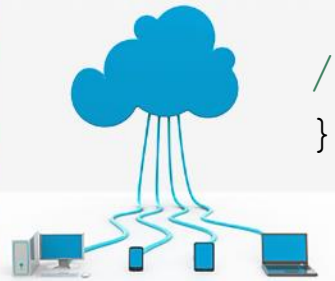
Stack
- *Node* **Top** - *int* **Size**
+ **Stack ( )** + *boolean* **isEmpty ( )** + *void* **makeEmpty ( )** + *int* **Length ( )** + *void* **Push** (*Anytype* value) + *Anytype* **Pop ( )** + *Anytype* **Peek ( )** + *void* **Print ( )**

Linked Stack

isEmpty ( )

- ✓ Check if Top is null
  or if Size is 0

```
public boolean isEmpty()
{
    return Top == null;
// return Size == 0;
}
```



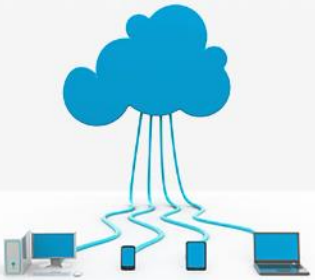
Stack
- *Node* **Top** - *int* **Size**
+ **Stack** ( ) + *boolean* **isEmpty** ( ) + *void* **makeEmpty** ( ) + *int* **Length** ( ) + *void* **Push** (*Anytype* value) + *Anytype* **Pop** ( ) + *Anytype* **Peek** ( ) + *void* **Print** ( )

Linked Stack

makeEmpty ( )

1. Check if the stack is not empty
   (if empty stop here)
2. Set Top to null
3. Set Size to 0

```
public void makeEmpty() {
    if(!isEmpty()) {
        Top = null;
        Size = 0;
    }
}
```

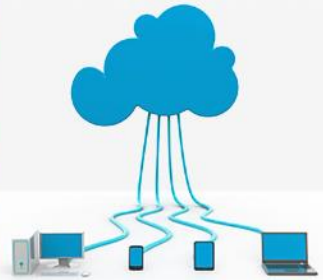


Stack
- *Node* **Top** - *int* **Size**
+ **Stack** ( ) + *boolean* **isEmpty** ( ) + *void* **makeEmpty** ( ) + *int* **Length** ( ) + *void* **Push** (*Anytype* value) + *Anytype* **Pop** ( ) + *Anytype* **Peek** ( ) + *void* **Print** ( )

Linked Stack

Length ( )

1. Start from Top node
2. If Node is not null:
   a. Add counter
   b. Move to next node
3. Repeat step 2 until Node is null

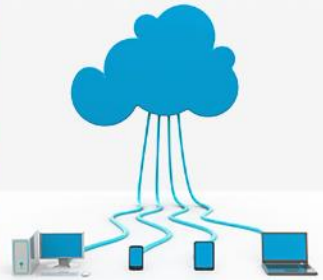


Stack
- *Node* **Top** - ~~*int* **Size**~~
+ **Stack** ( ) + *boolean* **isEmpty** ( ) + *void* **makeEmpty** ( ) + *int* **Length** ( ) + *void* **Push** (*Anytype* value) + *Anytype* **Pop** ( ) + *Anytype* **Peek** ( ) + *void* **Print** ( )

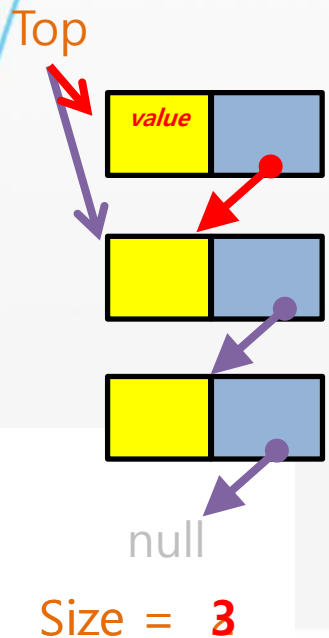
Linked Stack

Length ( )

```
public int Length()
{
    int Size=0;
    Node cn=Top;
    while(cn!=null)
    {
        cn = cn.getNextNode();
        Size++;
    }
    return Size;
}
```



Linked Stack



Push (value)

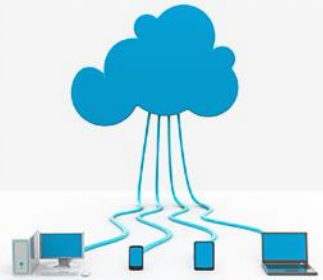
1. Create a new node storing value
2. Let the nextNode reference of the new node refers to Top node
3. Let Top refers to the new node
4. Increment Size

Stack
- *Node* **Top** - *int* **Size**
+ **Stack** ( ) + *boolean* **isEmpty** ( ) + *void* **makeEmpty** ( ) + *int* **Length** ( ) + *void* **Push** (*Anytype* value) + *Anytype* **Pop** ( ) + *Anytype* **Peek** ( ) + *void* **Print** ( )

Linked Stack

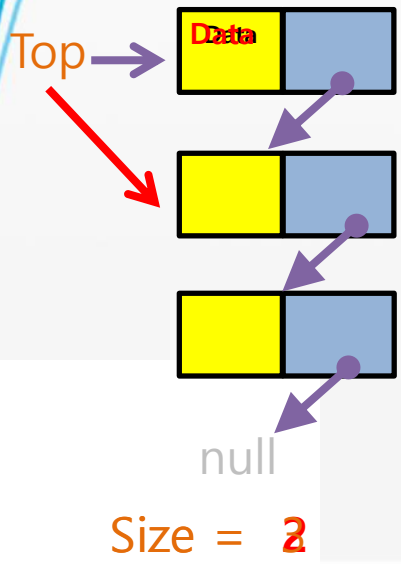
Push (value)

```
public void Push(Anytype value)
{
    Top = new Node<Anytype>(value, Top);
    Size++;
}
```



Linked Stack

Pop ( )



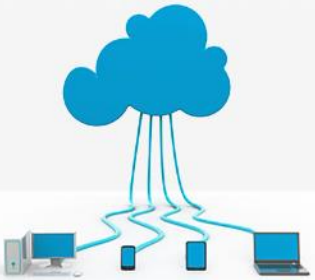
1. **Check if the stack is not empty**
   (if empty stop here)
2. **Get the data stored in Top node**
3. **Let Top refers to its next node**
4. **Decrement Size**
5. **Return the data** (from step 2)

Stack
- *Node* **Top** - *int* **Size**
+ **Stack** ( ) + *boolean* **isEmpty** ( ) + *void* **makeEmpty** ( ) + *int* **Length** ( ) + *void* **Push** (*Anytype* value) + *Anytype* **Pop** ( ) + *Anytype* **Peek** ( ) + *void* **Print** ( )

Linked Stack

Pop ( )

```
public Anytype Pop()
{
    if (this.isEmpty())
        throw new RuntimeException();

    Anytype removedValue = Top.getData();
    Top = Top.getNextNode();

    Size--;
    return removedValue;
}
```

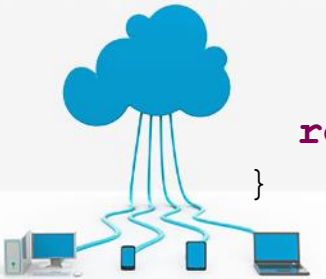


Linked Stack

Peek ( )

1. Check if the stack is not empty
   (if empty stop here)
2. Return the value stored in Top node

```
public Anytype Peek()
{
    if (isEmpty())
        throw new RuntimeException();

    return Top.getData();
}
```

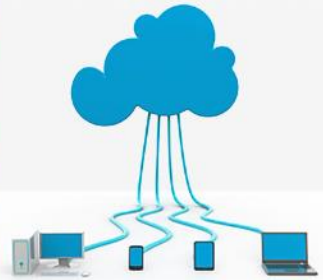


Stack
- *Node* **Top** - *int* **Size**
+ **Stack** ( ) + *boolean* **isEmpty** ( ) + *void* **makeEmpty** ( ) + *int* **Length** ( ) + *void* **Push** (*Anytype* value) + *Anytype* **Pop** ( ) + *Anytype* **Peek** ( ) + *void* **Print** ( )

Linked Stack

Print ( )

1. Check if the stack is not empty
   (if empty stop here)
2. Start from Top node
3. If Node is not null:
   a. Print data stored in the node
   b. Move to next node
4. Repeat step 2 until Node is null

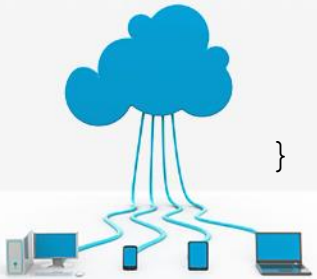


Stack
- *Node* **Top** - ~~*int* **Size**~~
+ **Stack** ( ) + *boolean* **isEmpty** ( ) + *void* **makeEmpty** ( ) + *int* **Length** ( ) + *void* **Push** (*Anytype* value) + *Anytype* **Pop** ( ) + *Anytype* **Peek** ( ) + *void* **Print** ( )

Linked Stack

Print ( )

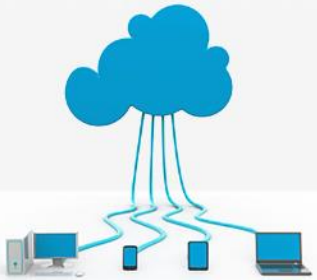
```
public void Print() {
   if (this.isEmpty())
      System.out.println("The Stack is empty.");
   else {
      Node<Anytype> currentNode = Top;

      while (currentNode != null) {
        System.out.print(currentNode.getData().toString());
        currentNode = currentNode.getNextNode();
      }
}
```



Applications

Numerous applications

- ✓ Parsing code
  - Matching:
    - parentheses (...)
    - brackets [...]
    - braces { ... }
  - Matching tags in XHTML
- ✓ Tracking function calls
- ✓ Dealing with undo/redo operations
- ✓ Reverse-Polish calculators
- ✓ Assembly language
- ✓ Converting a decimal number into a binary number



Applications

Numerous applications

- ✓ Towers of Hanoi
  - First Implementation (Without using Stacks)
  - Second Implementation (Using Stacks)
- ✓ Expression evaluation and syntax parsing
  - Evaluation of an Infix Expression that is Fully Parenthesized
  - Evaluation of Infix Expression which is not fully parenthesized
  - Evaluation of Prefix Expression
- ✓ Conversion of an Infix expression that is fully parenthesized into a Postfix expression
- ✓ Rearranging railroad cars
- ✓ Quicksort