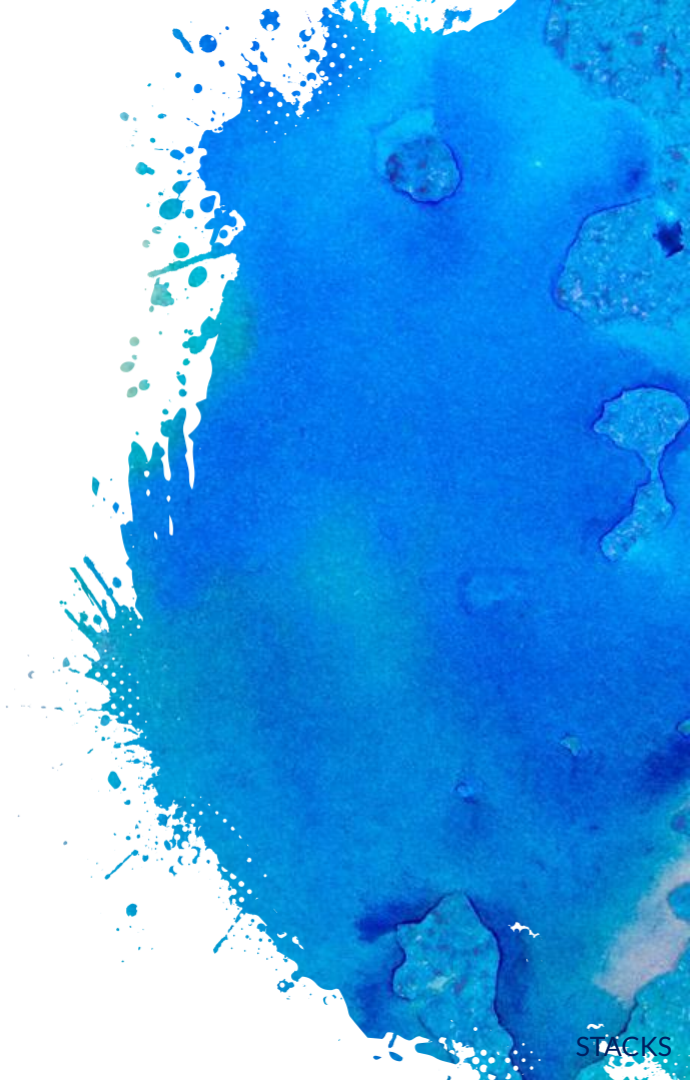
RIZAL TECHNOLOGICAL UNIVERSITY
COLLEGE OF ENGINEERING ARCHITECTURE AND TECHNOLOGY

DATA STRUCTURE AND ALGORITHM

STACKS

- What is Stacks?
- How Stacks are used?
- Stack Application
- Operations on Stacks
- Representation of Stacks
- Evaluation of Expressions
- Stack Implementation



Objectives:

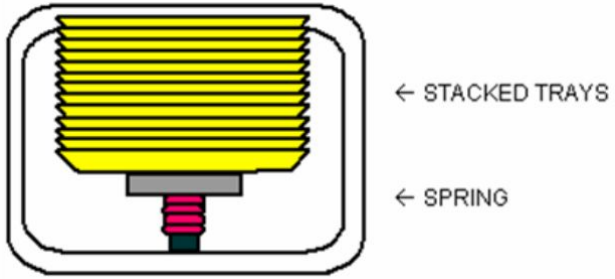
- Define stacks,
- Identify stacks are used in data structure,
- Enumerate the applications and operations in stacks,
- Identify how stacks are represented,
- Identify how expressions are evaluated, and
- Convert infix notation to postfix notation and vice versa.

STACKS

- A stack is an ordered list where all operations are restricted at one end of the list known as the **top**.
- Stacks are also used by compilers in the process of evaluating process and expressions and generating machine code language.

STACK is a linear data structure which follows a particular order in which the operations are performed. The order is LIFO.
LIFO or LAST-IN-FIRST-OUT. The last object inserted in the list will be the first one to be retrieved.

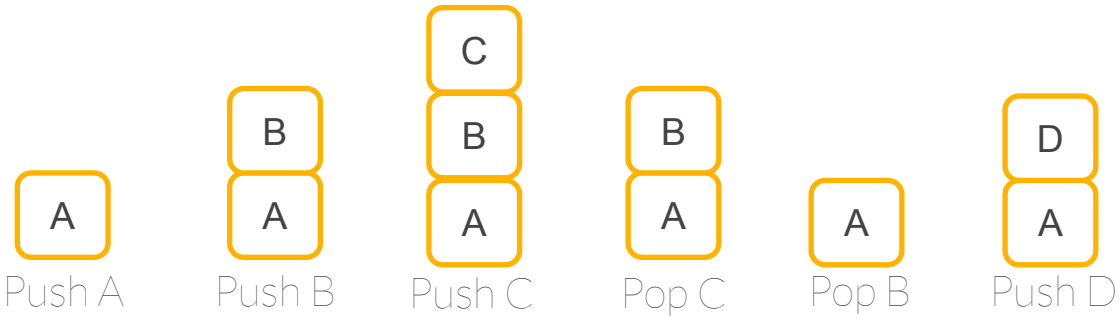
Example:



Stack Operations

- **PUSH.** Add items at the top
- **POP.** Removes element from the top
- **PEEK.** Accesses value at the top
- **isEmpty.** Returns true value if stack is empty, else false
- **isFull.** Tests if the stack is full or not
- **size.** Returns the number of elements present in the stack

Example:



Stack Applications

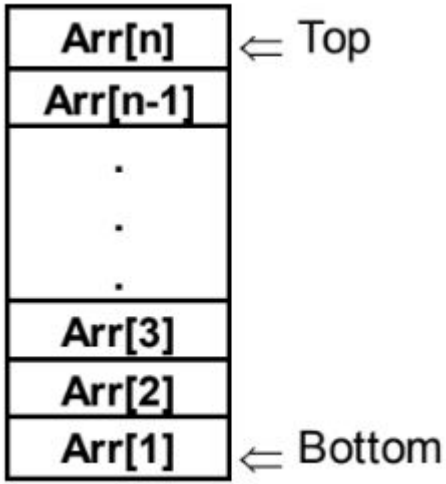
- Balancing of symbols
- Infix to Postfix Notation or vice versa
- Redo-undo features at many places like editors, photoshop
- Forward and backward feature in web browsers
- Used in many algorithms like Towers of Hanoi, tree traversals
- Backtracking
- Memory management
- String reversal

Stack Representation

There are two ways to represent a stack:

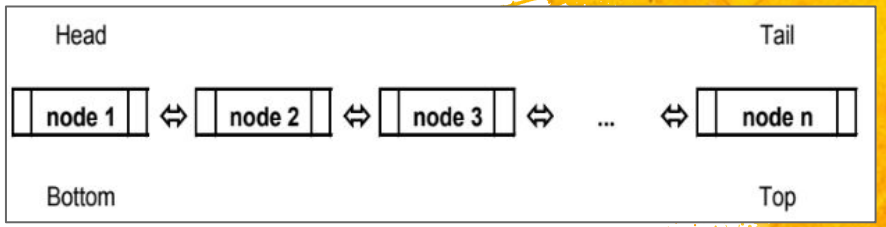
- **One-dimensional array**

Insertion, deletion, and retrieval in using one-dimensional array, it is done from the last element in the array.



- **Doubly-Linked List**

Insertion, deletion, and retrieval in using doubly-linked list, it is done from the tail.



Evaluation of Expressions

- An expressions is made up of operators and operands.

  Example: J / K * L + M - O

  Operator: / * + -
  Operand: J K L M O

- The operations to be performed on the operands are described by the associated operator.

- Operators of a higher precedence are processed first
- Evaluation is performed from left-to-right

Evaluation of Expressions : Infix and Postfix Notation

- Infix Notation

  operand operator operand

- Postfix Notation

  operand operand operator

Infix expression may be directly translated to postfix form by beginning with the conversion of the sub expression with the highest precedence

Evaluation of Expressions : Infix Notation to Postfix Notation

- A + B = AB+
- (A + B) * C = AB+
AB+C*
- (A + B) * (C + D) = AB+
AB+CD+*
- ((A / B / C) * (D + E)) = ABC//
ABC//DE+
ABC//DE+*
- ((A + B) / (C - A)) + (D * E) = AB+
AB+CA-/
AB+CA-/DE*+

Exercise Problem : Infix Notation to Postfix Notation

Convert the following infix notation to postfix notation

1.) (A * B + C) / D – E * F

2.) P / O + D – S * R

3.) J * Q + S – V / B – N

4.) H – (A+ K) + E * D

5.) L / (F – R) + (O * R – (W / D))

Postfix Notation

The algorithm for Postfix performs the following operations on expressions written in infix notation:

- Detects errors in the syntax of the infix expression.
- Transforms the infix expression into postfix notation.

Algorithm

- If recognize an operand, push it on the stack.
- If recognize an operator, pop its operands, apply the operator and push the value on the stack.
- Upon conclusion, the value of the postfix expression is on the top of the stack.

Postfix Notation to Infix Notation

- A B C D * + / E * F -

TOKENS	OPERATION	STACKS			
A	Push A	A			
B	Push B	B	A		
C	Push C	C	B	A	
D	Push D	D	C	B	A
*	Pop the top two element, multiply push result	C * D	B	A	
+	Pop the top two element, add, push the result	B + (C*D)	A		
/	Pop the top two element, divide push the result	A / (B+ (C*D))			
E	Push E	E	A / (B+ (C*D)		
*	Pop the top two element, multiply, push result	A / (B+ (C*D)) * E			
F	Push F	F	A / (B+ (C*D) * E		
-	Pop the top two element	A / (B+ (C*D)) * E - F			

Postfix Notation to Infix Notation

- A B + A + B
- X Y Z / * X * (Y / Z)
- A B C / - J K / L - * A - (B / C)
  (A – (B / C)) * ((J / K) – L)

Exercise Problem : Postfix Notation to Infix Notation

Convert the following postfix notation to infix notation

1.) A B * C + D E F * - /

2.) P O / D + S R * -

3.) J Q * S + V B / - N -

4.) H A K + - E D * -

5.) L F R - / O R * W D / - +

Stack Implementation

In order to create a stack, import the **java.util.Stack** package first.

To create a stack in Java:

```
Stack<Type> stacks = new Stack<>();
```

Type indicates the stack's type.
Example:

```
Stack<Integer> stacks = new Stack<>();
Stack<String> stacks = new Stack<>();
```

Stack Implementation

push() and pop() Method


```java
import java.util.Stack;

class StackFruitsPushPop {
  public static void main(String[] args) {

    // create an object of Stack class
    Stack<String> fruits = new Stack<>();

    // push elements to top of stack
    fruits.push("Apple");
    fruits.push("Banana");
    fruits.push("Orange");
    fruits.push("Grape");
    fruits.push("Watermelon");
    System.out.println("Stack: " + fruits);

    // pop elements from the top
    fruits.pop();
    fruits.pop();
    System.out.println("Stack after pop: " + fruits);
    }
}
```

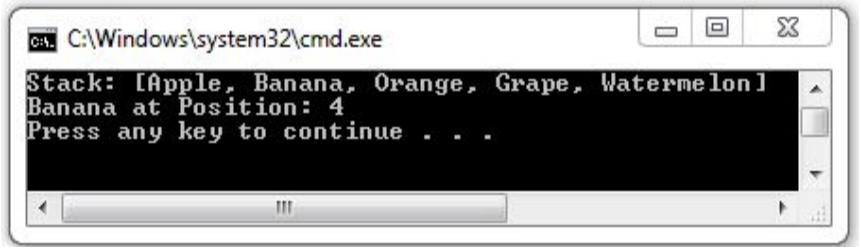
Stack Implementation

peek() Method


```java
import java.util.Stack;

class StackFruitsPeek {
  public static void main(String[] args) {

    // create an object of Stack class
    Stack<String> fruits = new Stack<>();

    // push elements to top of stack
    fruits.push("Apple");
    fruits.push("Banana");
    fruits.push("Orange");
    fruits.push("Grape");
    fruits.push("Watermelon");
    System.out.println("Stack: " + fruits);

    // access element from the top
    String element = fruits.peek();
    System.out.println("Element at top: " + element);
    }
}
```

Stack Implementation

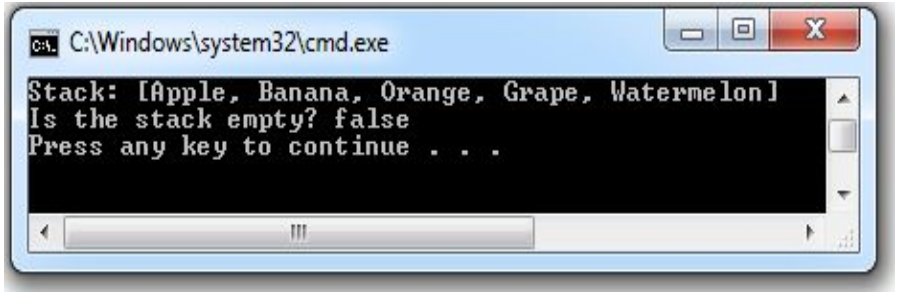
search() Method




```java
import java.util.Stack;

class StackFruitsSearch {
  public static void main(String[] args) {

    // create an object of Stack class
    Stack<String> fruits = new Stack<>();

    // push elements to top of stack
    fruits.push("Apple");
    fruits.push("Banana");
    fruits.push("Orange");
    fruits.push("Grape");
    fruits.push("Watermelon");
    System.out.println("Stack: " + fruits);

    // search an element
    int position = fruits.search("Banana");
    System.out.println("Banana at Position: " + position);
    }
}
```

Stack Implementation

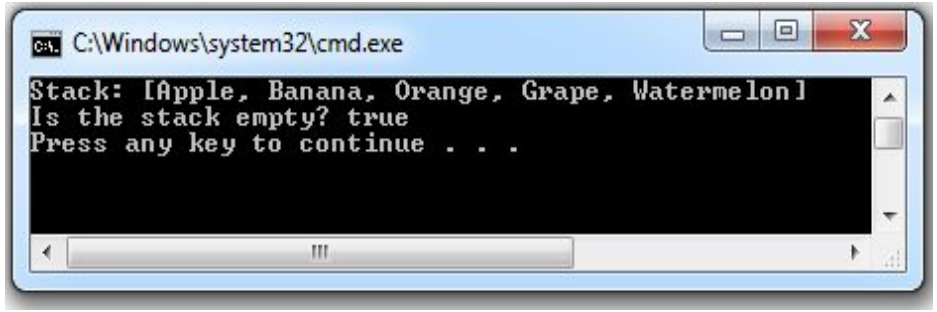
empty() Method - False




```java
import java.util.Stack;

class StackFruitsEmpty {
  public static void main(String[] args) {

    // create an object of Stack class
    Stack<String> fruits = new Stack<>();

    // push elements to top of stack
    fruits.push("Apple");
    fruits.push("Banana");
    fruits.push("Orange");
    fruits.push("Grape");
    fruits.push("Watermelon");
    System.out.println("Stack: " + fruits);

    // check if stack is empty
    boolean result = fruits.empty();
    System.out.println("Is the stack empty? " + result);
    }
}
```

Stack Implementation

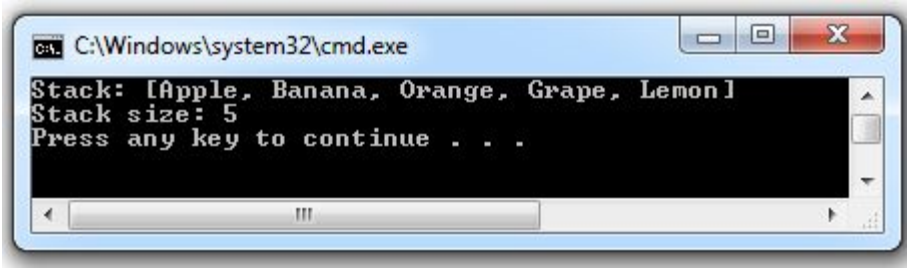
empty() Method - True




```java
import java.util.Stack;

class StackFruitsEmpty {
  public static void main(String[] args) {

    // create an object of Stack class
    Stack<String> fruits = new Stack<>();

    // push elements to top of stack
    fruits.push("Apple");
    fruits.push("Banana");
    fruits.push("Orange");
    fruits.push("Grape");
    fruits.push("Watermelon");
    System.out.println("Stack: " + fruits);

    fruits.pop();
    fruits.pop();
    fruits.pop();
    fruits.pop();
    fruits.pop();

    // check if stack is empty
    boolean result = fruits.empty();
    System.out.println("Is the stack empty? " + result);
    }
}
```

Stack Implementation

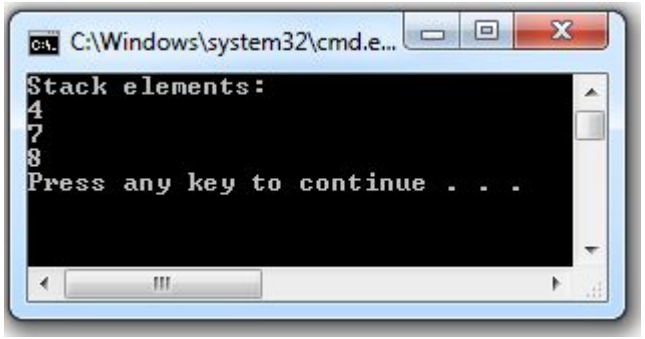
java.util.Stack.size() Method




```java
import java.util.Stack;

class StackFruitsSize {
  public static void main(String[] args) {

    // create an object of Stack class
    Stack<String> fruits = new Stack<>();

    // push elements to top of stack
    fruits.push("Apple");
    fruits.push("Banana");
    fruits.push("Orange");
    fruits.push("Grape");
    fruits.push("Lemon");

    System.out.println("Stack: " + fruits);

    //java.util.Stack.size() method
    System.out.println("Stack size: " + fruits.size());
    }
}
```

Stack Implementation

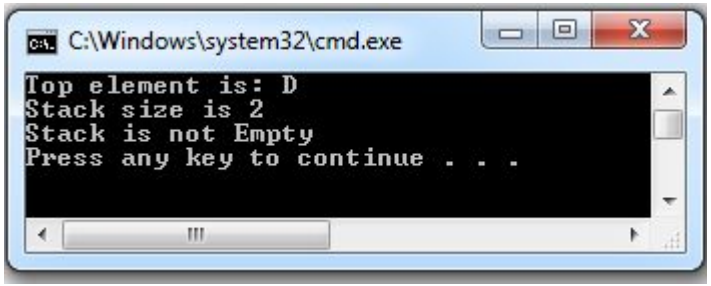
Print/Iterate Stack elements using Iterator




```java
import java.util.*;

public class StackIterate
{
    public static void main(String[] args) {

        //declare and initialize a stack object
        Stack<Integer> stack = new Stack<Integer>();

        stack.push(4);
        stack.push(7);
        stack.push(8);

        System.out.println("Stack elements:");

        //get an iterator for the stack
        Iterator iterator = stack.iterator();

        //traverse the stack using iterator in a loop and print each element
        while(iterator.hasNext()){
        System.out.println(iterator.next() + " ");
        }

    }
}
```

Stack Implementation

Stack applying push, pop and determining if stack is empty or not.




```java
import java.util.*;

class StackTheory
{
    public static void main(String[] args)
    {
        Stack<String> stack = new Stack<String>();

        stack.push("A");    // Insert "A" in the stack
        stack.push("B");    // Insert "B" in the stack
        stack.push("C");    // Insert "C" in the stack
        stack.push("D");    // Insert "D" in the stack

        // Prints the top of the stack ("D")
        System.out.println("Top element is: " + stack.peek());

        stack.pop();    // removing the top ("D")
        stack.pop();    // removing the next top ("C")

        // Returns the number of elements present in the stack
        System.out.println("Stack size is " + stack.size());

        // check if stack is empty
        if (stack.empty())
            System.out.println("Stack is Empty");
        else
            System.out.println("Stack is not Empty");
    }
}
```

Stack Implementation

Stack implementing Java

```
class StackImplemJava {


  private int arr[];          // store elements of stack
  private int top;            // represent top of stack
  private int capacity;       // total capacity of the stack


  StackImplemJava(int size) {           // Creating a stack
    arr = new int[size];      // initialize the array
    capacity = size;          // initialize the stack variables
    top = -1;
  }


  public void push(int x) {  // push elements to the top of stack
    if (isFull()) {
      System.out.println("Stack OverFlow");
      System.exit(1);          // terminates the program
    }

  // insert element on top of stack
    System.out.println("Inserting " + x);
    arr[++top] = x;
  }
```

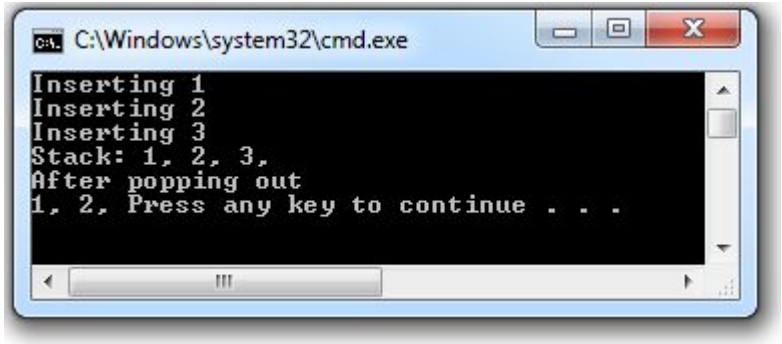
Stack Implementation

Stack implementing Java

```
// pop elements from top of stack
public int pop() {

  // if stack is empty
  // no element to pop
  if (isEmpty()) {
    System.out.println("STACK EMPTY");
    // terminates the program
    System.exit(1);
  }

  // pop element from top of stack
  return arr[top--];
}

// return size of the stack
public int getSize() {
  return top + 1;
}

// check if the stack is empty
public Boolean isEmpty() {
  return top == -1;
}

// check if the stack is full
public Boolean isFull() {
  return top == capacity - 1;
}
```

Stack Implementation

Stack implementing Java




```
  // display elements of stack
  public void printStack() {
    for (int i = 0; i <= top; i++) {
      System.out.print(arr[i] + ", ");
    }
  }

  public static void main(String[] args) {
    StackImplemJava stack = new StackImplemJava(5);

    stack.push(1);
    stack.push(2);
    stack.push(3);

    System.out.print("Stack: ");
    stack.printStack();

    // remove element from stack
    stack.pop();
    System.out.println("\nAfter popping out");
    stack.printStack();

  }
}
```

Stack Implementation

Stack implementing Java

```
class StackBasic {

    static final int MAX = 100;
    int top;
    int a[] = new int[MAX]; // Maximum size of Stack

    boolean isEmpty()
    {
        return (top < 0);
    }
    StackBasic()
    {
        top = -1;
    }

    boolean push(int num)
    {
        if (top >= (MAX - 1)) {
            System.out.println("Stack Overflow");
            return false;
        }
        else {
            a[++top] = num;
            System.out.println(num + " pushed into stack");
            return true;
        }
    }
```

Stack Implementation

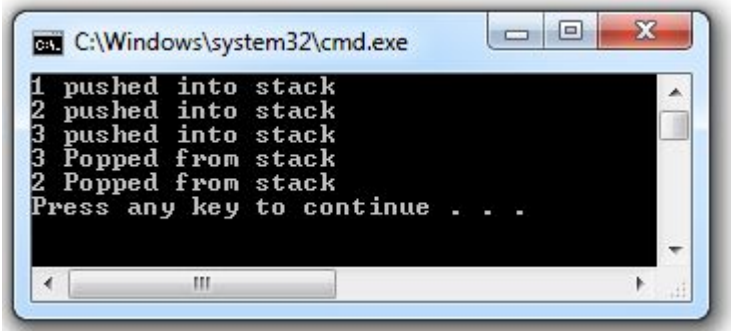
Stack implementing Java

```
int pop()
{
    if (top < 0) {
        System.out.println("Stack Underflow");
        return 0;
    }
    else {
        int num = a[top--];
        return num;
    }
}

int peek()
{
    if (top < 0) {
        System.out.println("Stack Underflow");
        return 0;
    }
    else {
        int num = a[top];
        return num;
    }
}
```

Stack Implementation

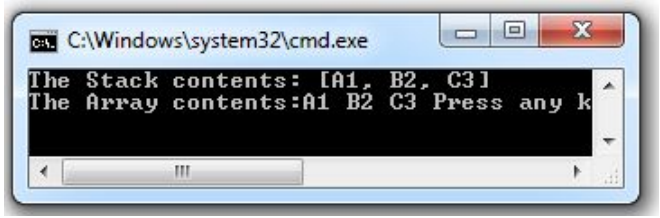
Stack implementing Java

```
    public static void main(String args[])
    {
        StackBasic s = new StackBasic();
        s.push(1);
        s.push(2);
        s.push(3);
        System.out.println(s.pop() + " Popped from stack");
        System.out.println(s.pop() + " Popped from stack");
    }
}
```



Stack Implementation

The stack data structure can be converted to an Array using 'toArray()' method of the Stack class.




```java
import java.util.*;

public class StackToArray
{
    public static void main(String[] args) {

        //declare and initialize a stack object
        Stack<String> stack = new Stack<String>();
        stack.push("A1");
        stack.push("B2");
        stack.push("C3");

        //print the stack
        System.out.println("The Stack contents: " + stack);

        // Create the array and use toArray() method to convert stack to array
        Object[] strArray = stack.toArray();

        //print the array
        System.out.print("The Array contents:");
        for (int j = 0; j < strArray.length; j++)
            System.out.print(strArray[j]+ " ");
    }
}
```

RIZAL TECHNOLOGICAL UNIVERSITY
COLLEGE OF ENGINEERING ARCHITECTURE AND TECHNOLOGY

Thank You 😊
Keep safe
and God bless!