**STACKS AND QUEUES IN JAVA**

* Stack and queues are fundamental data structures in Java Collections framework
* They are used to store the same type of data and retrieve the data in specific order. Stack and Queue both are Linear Data Structures.
* Basically Stack follows the LIFO principle i.e. Last In First Out.
* The operations are denoted as push and pop for insertion and deletion in stacks.
* Queue follows the FIFO principle i.e. First In First Out. Insertion operation is known as enqueue and deletion operation is known as dequeue.
* Stacks are used in many applications. One such application is **undo mechanism** in text editors. All the changes are stored in a stack. When we undo something, it pops the most recent action. When we make changes, it pushes changes onto the stack.
* Back and Forward buttons in browsers are also performed using stacks.

Example:

import java.util.\*;

class Stack

{

    private int arr[];

    private int top;

    private int capacity;

    // Constructor to initialize the stack

    Stack(int size)

    {

        arr = new int[size];

        capacity = size;

        top = -1;

    }

  public void push(int x)//inserting

    {

        if (isFull())

        {

            System.out.println("Overflow\nProgram Terminated\n");

            System.exit(-1);

        }

 System.out.println("Inserting " + x);

arr[++top] = x;

 public int pop()

    {

           if (isEmpty())

        {

            System.out.println("Underflow\nProgram Terminated");

            System.exit(-1);

        }

   System.out.println("Removing " + peek());

        // decrease stack size by 1 and (optionally) return the popped element

        return arr[top--];

    }

    // Utility function to return the top element of the stack

    public int peek()

    {

        if (!isEmpty())

{

            return arr[top];

        }

        else

{

            System.exit(-1);

        }

return -1;

    }

// Utility function to return the size of the stack

    public int size() {

        return top + 1;

    }

   public boolean isEmpty() {

    return top == -1;

    }

      public boolean isFull() {

        return top == capacity - 1;     // or return size() == capacity;

    }

}

Main.java

class Main

{

    public static void main (String[] args)

    {

        Stack stack = new Stack(3);

        stack.push(1);      // inserting 1 in the stack

        stack.push(2);      // inserting 2 in the stack

        stack.pop();        // removing the top element (2)

        stack.pop();        // removing the top element (1)

        stack.push(3);      // inserting 3 in the stack

        System.out.println("The top element is " + stack.peek());

        System.out.println("The stack size is " + stack.size());

        stack.pop();        // removing the top element (3)

        // check if the stack is empty

        if (stack.isEmpty()) {

            System.out.println("The stack is empty");

        }

        else {

            System.out.println("The stack is not empty");

        }

    }

}