```
package Stack_Examples;

import java.util.Scanner;

public class Stack_Class
{
   int stack[];
   int tos,MaxSize;
   void create_Stack(int size)
   {
      tos=-1;
      MaxSize=size;
      stack=new int[MaxSize];
   }
   //push:accepts an element on stack
   //tos+1
   void push(int e)
   {
      tos++;
      stack[tos]=e;
      //stack[++tos]=e;
   }
   boolean is_full()
   {
      if(tos==MaxSize-1)
         return true;
      else
         return false;

      //return(tos==MaxSize-1);

   }
   int pop()
   {
      int temp=stack[tos];
      tos--;
      return(temp);
      //return(stack[tos--]);
   }
   boolean is_empty()
   {
      if(tos==-1)
         return true;
      else
         return false;

      //return(tos==-1);
```

```
}
void print_stack()//in LIFO
{
    for(int i=tos;i>=0;i--)
        System.out.println(stack[i]);
}

int peek()//only return element on top
{
    return(stack[tos]);
}

public static void main(String[] args)
{
    Stack_Class obj=new Stack_Class();
    int choice;
    Scanner in=new Scanner(System.in);
    System.out.println("Enter size of stack:");
    int size=in.nextInt();
    obj.create_Stack(size);
    do
    {
        System.out.println("\nStack Menu");
        System.out.println("==========");
        System.out.println("1.Push");
        System.out.println("2.Pop");
        System.out.println("3.Peek");
        System.out.println("4.Print");
        System.out.println("0.Exit");
        System.out.println("--------");
        System.out.print(":");
        choice=in.nextInt();
        switch(choice)
        {
            case 1:
                    if(!obj.is_full())//if not full then
                    {
                        System.out.println("Enter element to push:");
                        int e=in.nextInt();
                        obj.push(e);
                        System.out.println("Element pushed");
                    }
                    else
                    {
                        System.out.println("Stack Full");
                    }
                    break;
            case 2:
                    if(!obj.is_empty())//if not empty then
                    {
                        System.out.println("Element poped:"+obj.pop());
                    }
                    else
                    {
                        System.out.println("Stack Empty");
                    }
                    break;
                case 3:
                    if(!obj.is_empty())//if not empty then
                    {
                        System.out.println("Element @ Peek is:"+obj.peek());
                    }
                    else
                    {
                        System.out.println("Stack Empty");
                    }
                    break;
                case 4:
                    if(!obj.is_empty())//if not empty then
                    {
                        System.out.println("Stack has:");
                        obj.print_stack();
                    }
                    else
                    {
                        System.out.println("Stack Empty");
                    }
                    break;
                case 0:
                        System.out.println("Thanks for using the code: amar.career");
                    break;
                default:
                    System.out.println("check the option selected.");
                    break;

            }
        }while(choice!=0);
    }
}
```

The Java Collections Choice: Deque (Double-Ended Queue)

While Java has a java.util.Stack class, it is considered **legacy** and **suboptimal** because:

1. It extends Vector, which makes its methods unnecessarily **synchronized** (slower in single-threaded apps).
2. It allows elements to be accessed by index (breaking the fundamental LIFO principle).

The official Java documentation recommends using the **Deque (Double-Ended Queue) interface** for stack operations, typically implemented by the **ArrayDeque** class, as it is faster and has a cleaner API for LIFO behavior.

3. Essential Stack Methods via the Deque Interface

The Deque interface provides a consistent and performance-optimized set of methods that perfectly map to the three primary stack operations by treating one end (the "head" or "first" element) as the **top** of the stack.

Stack Operation	Deque Method	Behavior (Stack LIFO)	Return Type / Throws
Push (Add to Top)	push(E e)	Adds the element to the front (head) of the deque (the **top of the stack**).	void / Throws IllegalStateException if capacity-restricted (not for ArrayDeque).
Pop (Remove from Top)	pop()	Retrieves and **removes** the element from the front (head) of the deque (the **top of the stack**).	E (the element) / Throws NoSuchElementException if empty.
Peek (View Top)	peek()	Retrieves, but **does NOT remove**, the	E (the element) / Returns null if

		element from the front (head) of the deque (the **top of the stack**).	empty.
Check Empty	isEmpty()	Returns true if the Deque has no elements.	boolean
Size	size()	Returns the number of elements in the Deque.	int

4. Implementation: Java Code Example (ArrayDeque)

This example demonstrates how to use the recommended ArrayDeque to implement a stack of String elements, showing the three core operations.

Java

```java
import java.util.ArrayDeque;
import java.util.Deque;
import java.util.NoSuchElementException;

/**
 * Modern Stack Implementation using the Deque interface and ArrayDeque class.
 * This is the preferred way to implement LIFO behavior in Java.
 */
public class ModernStackDemo {

    public static void main(String[] args) {
        // Declaration: Use the Deque interface for flexibility
        // Instantiation: Use ArrayDeque for an efficient, non-synchronized implementation
        Deque<String> mySoftwareStack = new ArrayDeque<>();

        System.out.println("--- Stack Initial State ---");
        System.out.println("Is the stack empty? " + mySoftwareStack.isEmpty());
        System.out.println("---------------------------");

        // 1. PUSH Operation: Adding elements (LIFO: Last In)
        System.out.println("1. PUSHING Elements (Last In, First Out):");
        mySoftwareStack.push("Java Fundamentals");
        mySoftwareStack.push("Spring Boot");
        mySoftwareStack.push("Microservices (TOP)"); // This is the last element pushed

        System.out.println("Stack after Pushes: " + mySoftwareStack);
        System.out.println("Stack size: " + mySoftwareStack.size());
        System.out.println("---------------------------");

        // 2. PEEK Operation: Viewing the top element
        String topElement = mySoftwareStack.peek();
        System.out.println("2. PEEK: The top element is: " + topElement);
        System.out.println("Stack size (after peek): " + mySoftwareStack.size()); // Size remains the
same
        System.out.println("---------------------------");

        // 3. POP Operation: Removing the top element (LIFO: First Out)
        String poppedItem = mySoftwareStack.pop(); // Removes "Microservices (TOP)"
        System.out.println("3. POP: Removed item (LIFO): " + poppedItem);
        System.out.println("Stack after 1st Pop: " + mySoftwareStack);

        poppedItem = mySoftwareStack.pop(); // Removes "Spring Boot"
        System.out.println("4. POP: Removed item (Next LIFO): " + poppedItem);
        System.out.println("Stack after 2nd Pop: " + mySoftwareStack);
        System.out.println("---------------------------");

        // 5. Handling an Empty Stack (Good Practice)
        mySoftwareStack.pop(); // Remove the last item: "Java Fundamentals"
        System.out.println("Stack after final Pop. Is it empty? " + mySoftwareStack.isEmpty());

        try {
            // This will throw a NoSuchElementException, which is a key part of the Deque API
            System.out.print("Attempting to pop from an empty stack... ");
            mySoftwareStack.pop();
        } catch (NoSuchElementException e) {
            System.out.println("Caught Exception: Cannot pop from an empty stack! (Correct
Behavior)");
        }
    }
}
```

```
package Stack_Examples;

import java.util.ArrayDeque;
import java.util.Deque;

public class Stack_Collection
{
    public static void main(String[] args)
    {
        Deque<Integer> stack=new ArrayDeque<>();//faster stack
        stack.push(100);
        stack.push(200);
        stack.push(300);
        stack.push(400);
        System.out.println("Elements on stack:"+stack);
        System.out.println("Total Elements on stack:"+stack.size());
        System.out.println("Elements at peek in stack:"+stack.peek());
        System.out.println("Elements poped from stack:"+stack.pop());
        System.out.println("Elements on stack:"+stack);
        System.out.println("Total Elements on stack:"+stack.size());
        System.out.println("is stack empty:"+stack.isEmpty());


    }

}
```

```
package Stack_Examples;

import java.util.ArrayDeque;
import java.util.Deque;

public class Word_reversal
{
    public static void main(String[] args)
    {
        String str = "career credentials";
        Deque<Character> stack = new ArrayDeque<>();

        // push each char
        for(char ch : str.toCharArray())
        {
            stack.push(ch);
        }

        // reverse using for-each
        String sb = "";
        while(!stack.isEmpty())   // till not empty
        {
            sb+=stack.pop();
        }

        System.out.println("Original  : " + str);
        System.out.println("Reversed  : " + sb);
    }
}
```

```
package Stack_Examples;
import java.util.Scanner;
import java.util.Deque;
import java.util.ArrayDeque;
public class Dec_To_Binary
{
    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a decimal number: ");
        int dec = sc.nextInt();     // input decimal number

        Deque<Integer> stack = new ArrayDeque<>();

        int num = dec;

        // Step 1: Push remainders to stack
        while(num > 0)
        {
            int rem = num % 2;      // remainder when divided by 2
            stack.push(rem);        // push remainder into stack
            num = num / 2;          // divide number by 2
        }

        System.out.print("Binary of " + dec + " is: ");

        // Step 2: Pop and print (gives correct binary order)
        for(int bit : stack)//while (!stack.isEmpty())
        {
            System.out.print(bit);//stack.pop()
        }

        sc.close();
    }
}
```

```
package Stack_Examples;

import java.util.ArrayDeque;
import java.util.Deque;
import java.util.Scanner;

public class Check_Wellness
{
    public static void main(String[] args)
    {
        Scanner in=new Scanner(System.in);
        System.out.println("Enter pattern to check:");
        String str = in.next();
        Deque<Character> stack = new ArrayDeque<>();
        // push each char
        boolean flag=true;
        for(char ch : str.toCharArray()) {
            if (ch == '{')
                stack.push(ch);
            else if (ch == '}') {
                if (!stack.isEmpty())
                    stack.pop();
                else {
                    flag = false;//error
                    break;
                }
            }
        }
        if(!stack.isEmpty())
            flag=false;
        System.out.println("Pattern "+str+" is "+flag);

    }

}
```

```
package Stack_Examples;

import java.util.ArrayDeque;
import java.util.Deque;
import java.util.Scanner;

public class Check_Wellness
{//modify code to use boolean evaluate(pattern)--->true/false
    static boolean evaluate(String str)
    {
        Deque<Character> stack = new ArrayDeque<>();
        // push each char
        boolean flag=true;
        for(char ch : str.toCharArray()) {
            if (ch == '{')
                stack.push(ch);
            else if (ch == '}') {
                if (!stack.isEmpty())
                    stack.pop();
                else {
                    return false;
                }
            }
        }
            return (stack.isEmpty());//if empty true-->true else -->false

    }
    public static void main(String[] args)
    {
        Scanner in=new Scanner(System.in);
        System.out.println("Enter pattern to check:");
        String str = in.next();

        System.out.println("Pattern "+str+" is "+Check_Wellness.evaluate(str));

    }

}
```