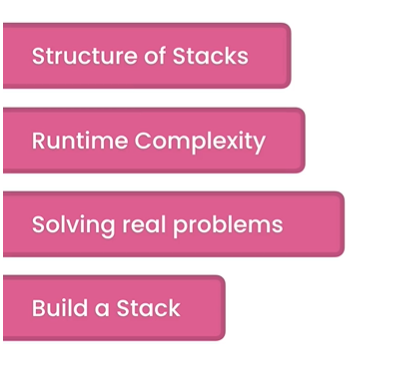
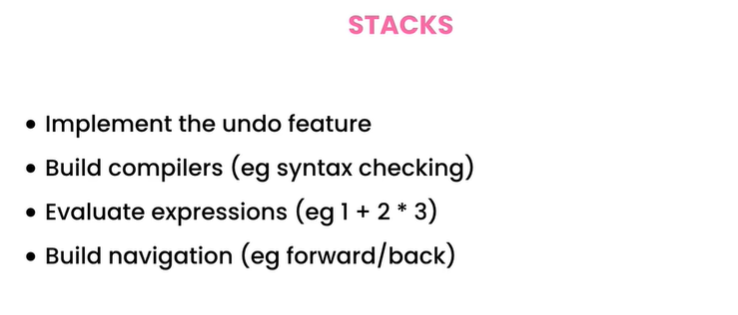
Stacks



(V-1) What Are Stacks?

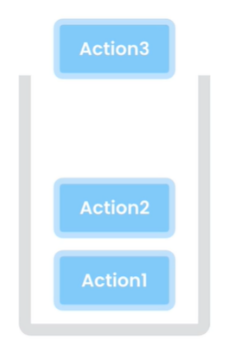
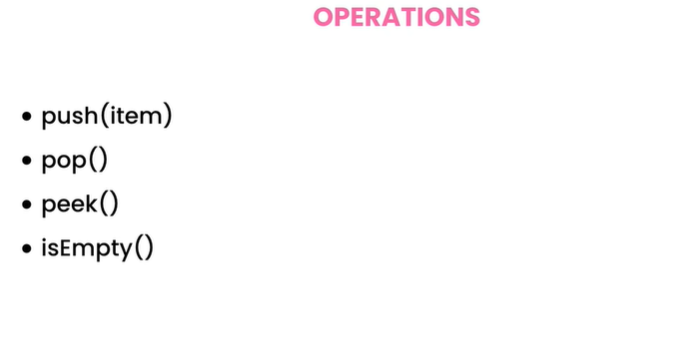


If we want to get the first item

We have to remove all the existing items

From tops to get the first item, that is called

Last In First Out **(LIFO)**



See The List of operations

carefully we don’t have look up

here. That means we don’t use

stacks for store any list of customers, products . For lookup we use Array and Linked List .

**The Last Action Can Be undone using Stacks**

**Undo**

All Operations has a time complexity of O(1) no matter it is implemented using Array or List.

(V-3)-Working with stacks

Stack <Integer> stack = new Stack<>();  
stack.push(10);// [10] -> top -> 10  
stack.push(20);// [10,20] -> top -> 20  
stack.push(30);// [10,20,30] -> top -> 30  
System.*out*.println(stack);  
var pop = stack.pop();// pop store 30 as 30 is in the top of the stack  
System.*out*.println(pop);  
System.*out*.println(stack);// [10, 20] -> top -> 20  
stack.pop(); // 20 will be popped as 20 is in the top  
System.*out*.println(stack.peek()); // it will return the top value without removing any items form stack. So, it will print 10

(V-4 & 5)-Reversing a String:-

import java.util.Stack;  
  
public class StringReverser {  
 // We can use stack DS to reverse a String Cause it is LIFO Structured  
 public String reverse(String input){  
 if (input==null)  
 throw new IllegalArgumentException();  
 Stack<Character> stack2 = new Stack<>();  
 /\* One Way using traditional For loop  
 for(int i=0;i<input.length();i++)  
 stack2.push(input.charAt(i));  
 \* \*/  
 //but for each loop is better and tricky  
 for(char ch : input.toCharArray()){//input.toCharArray() is written cause in java for each loop we can't iterate through String thats why first we make it in chararry  
 stack2.push(ch);  
 }  
 // Now half of the operation is done just left popping the elements from stacks  
 // But if we declare a String var and everytime concatanating it with new characters  
 // it will become a more time consuming cause in JAVA String is immutable  
 // That means we can't directly manipulate the String using same memory  
 // Every time String changes new memory will be allocated for the same operations  
 // So we can use StringBuffer class . It is best for String Manipulation when too much string manipulation is needed  
 StringBuffer reversed = new StringBuffer();  
 while(!stack2.empty())// all characters will be popped and string will be reversed  
 reversed.append(stack2.pop());// all charactes popped doing like concatanation but in same memory. Memory is saved now.  
 return reversed.toString();// charecter converted to String and then returned  
 }  
}

V-(6-10)-(Balance Parenthesis):-[All Basic Operation’s and uses of stacks are covered in this part]

Mosh Solution 1:-

public boolean isBalanced(String input){  
 // The idea is for checking the parenthesis are balanced or not using undo operation  
 // We iterate through the whole String (Character's) and push if we found opening bracket  
 // if we found closing bracket we then pop  
 // if the stack is empty then all parenthesis are balanced  
 // if we have anything left on the stack that means expression is not balanced  
 Stack <Character> stack = new Stack<>();// Call Stack  
 for(char ch : input.toCharArray()){// for each loop we can't iterate through string so we convert it into characters  
 if(ch == '(' || ch == '{' || ch == '[' || ch == '<')  
 stack.push(ch);  
 if(ch == ')' || ch == '}' || ch == ']' || ch == '>' ){  
 if(stack.empty()) return false;  
 var top = stack.pop();  
 if (// As we have total 4 types of bracket so we pop for every closing bracket and check with the corresponding current right bracket if they don't match as per required we return false immediately  
 (ch == ')' && top!= '(') ||  
 (ch == '}' && top!='{') ||  
 (ch == ']' && top!='[') ||  
 (ch == '>' && top!='<')  
 ) return false;  
 }  
 }  
 return stack.empty();  
}

Mosh Solution 2(More Readable Refactored):-

public boolean isBalanced2(String input1){  
 Stack<Character> stack1 = new Stack<>();  
 for(char ch : input1.toCharArray()){//pura string charcter array te niye iterate  
 if(isLeftBracket(ch))// left bracket (,{,[,< paile push  
 stack1.push(ch);  
 if(isRightBracket(ch)) {// right bracket ),},],> paile pop kore compare  
 if(stack1.empty()) return false;  
 var top = stack1.pop();  
 if((BracketMatch(top,ch))) return false; //eikhane BracketMatch false  
 }  
 }  
 return stack1.empty();  
}

private boolean isLeftBracket(char ch){// opening bracket gula thakle push korbe //call hoya method theke  
 return ch == '(' || ch == '{' || ch == '[' || ch == '<';  
}  
private boolean isRightBracket(char ch){  
 return ch == ')' || ch == '}' || ch == ']' || ch == '>';  
}  
private boolean BracketMatch(char left,char right){//pop korbe opening bracket

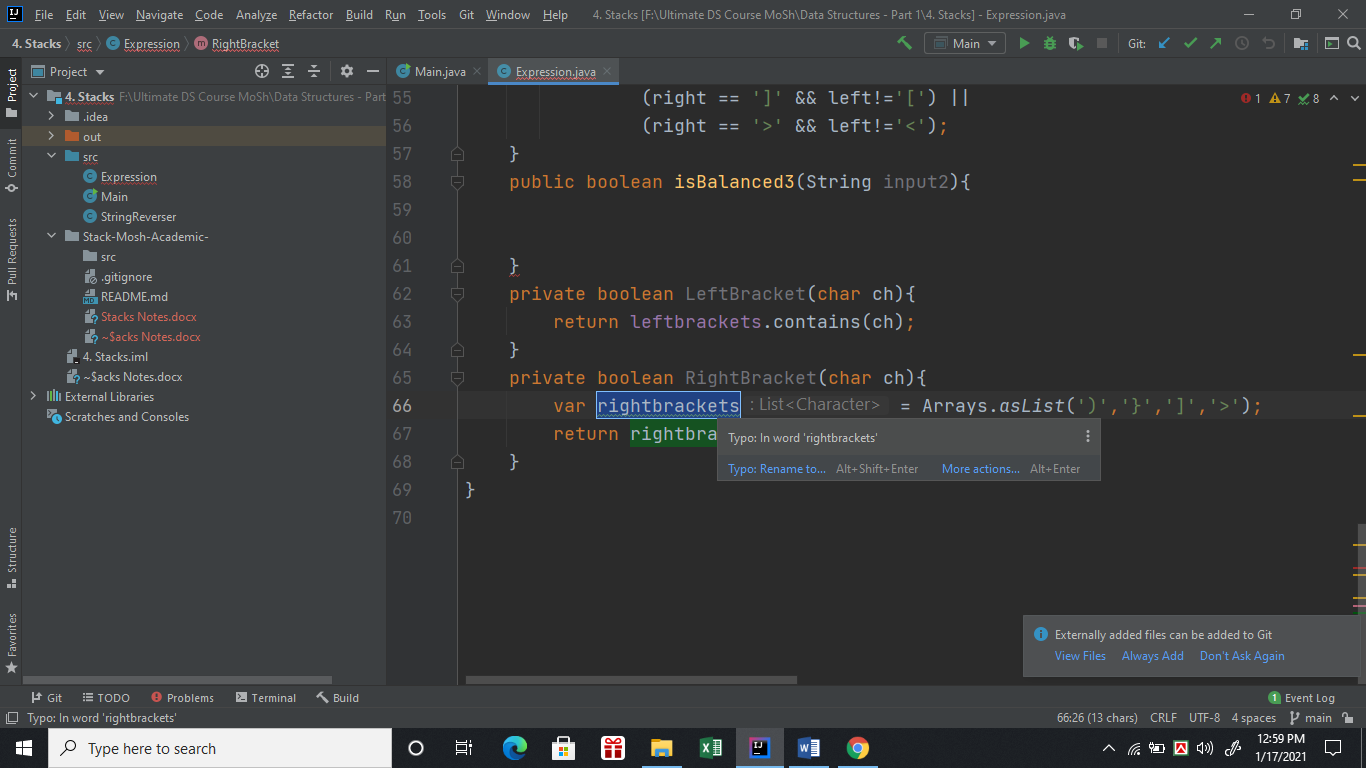
// (,[,{,< aigula left bracket tai namkoron e char left dicci

// ),],},> aigula right(closing) bracket jeigula stack e push hobe nah but compare //hobe tai jeikhan theke call hocce seikhan theke current character patano hoyese   
 return (right == ')' && left!= '(') ||  
 (right == '}' && left!='{') ||  
 (right == ']' && left!='[') ||  
 (right == '>' && left!='<');  
}

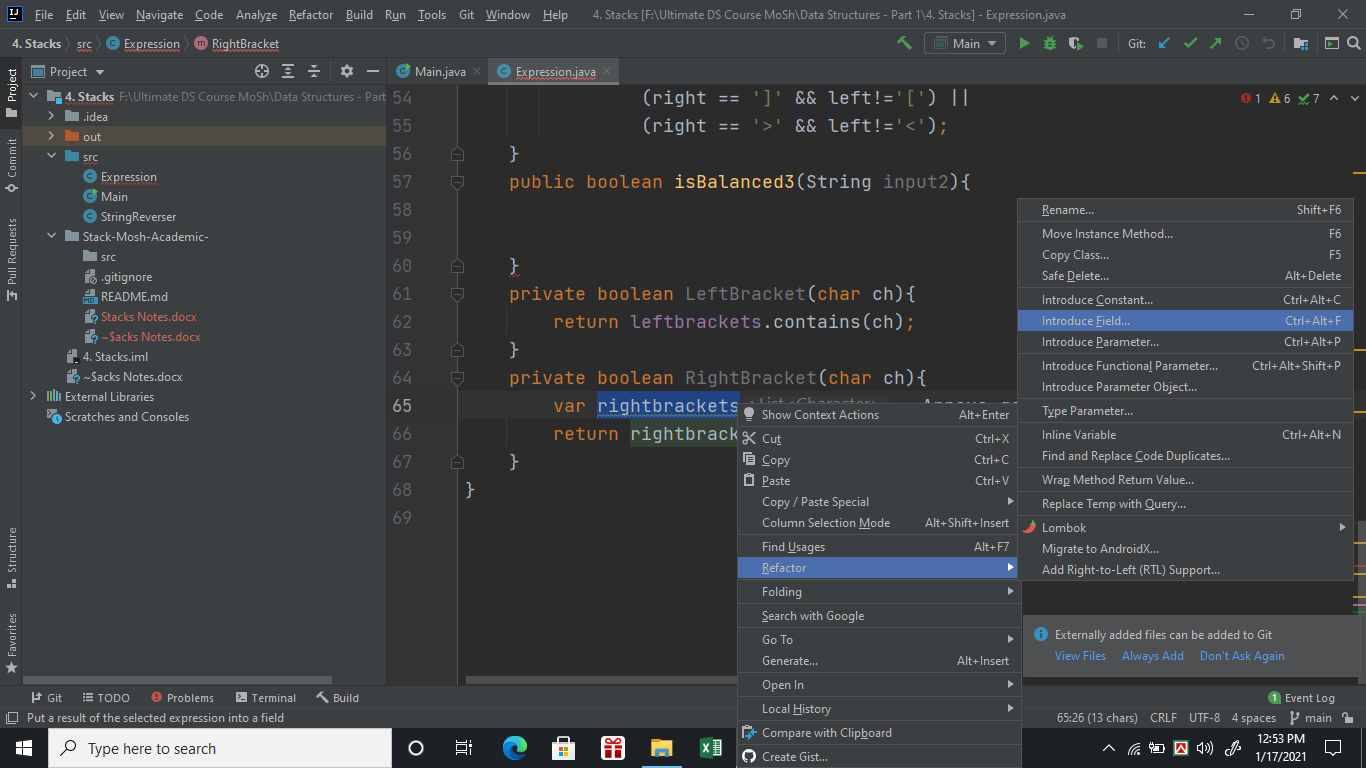
Mosh Solution 3(More Readable Refactored+):-

Firstly, We need to know about Extracting fields form any method:

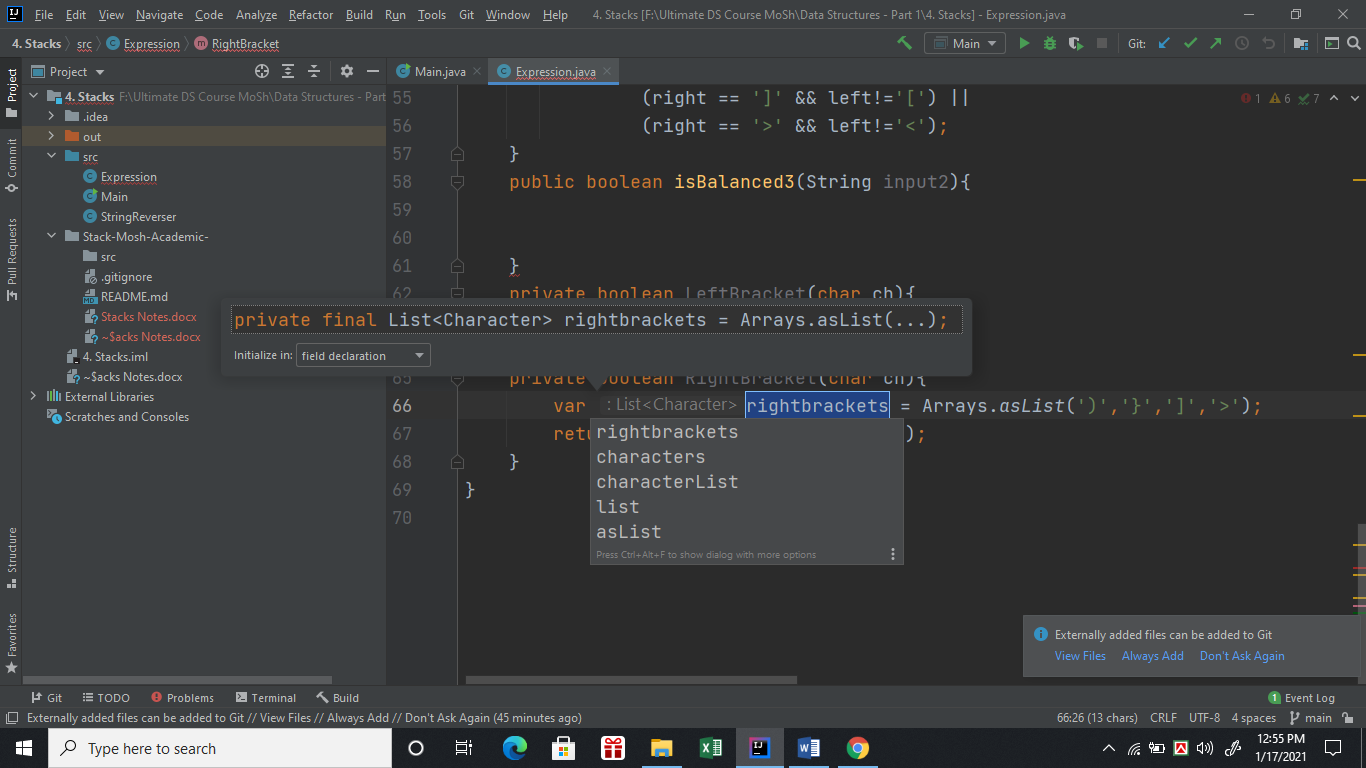
Step 1:- Take cursor to the field that I am going to extract or select the field . In the picture below I want rightbrackets field form RightBracket method to be extracted. So I select “rightbrackets”.



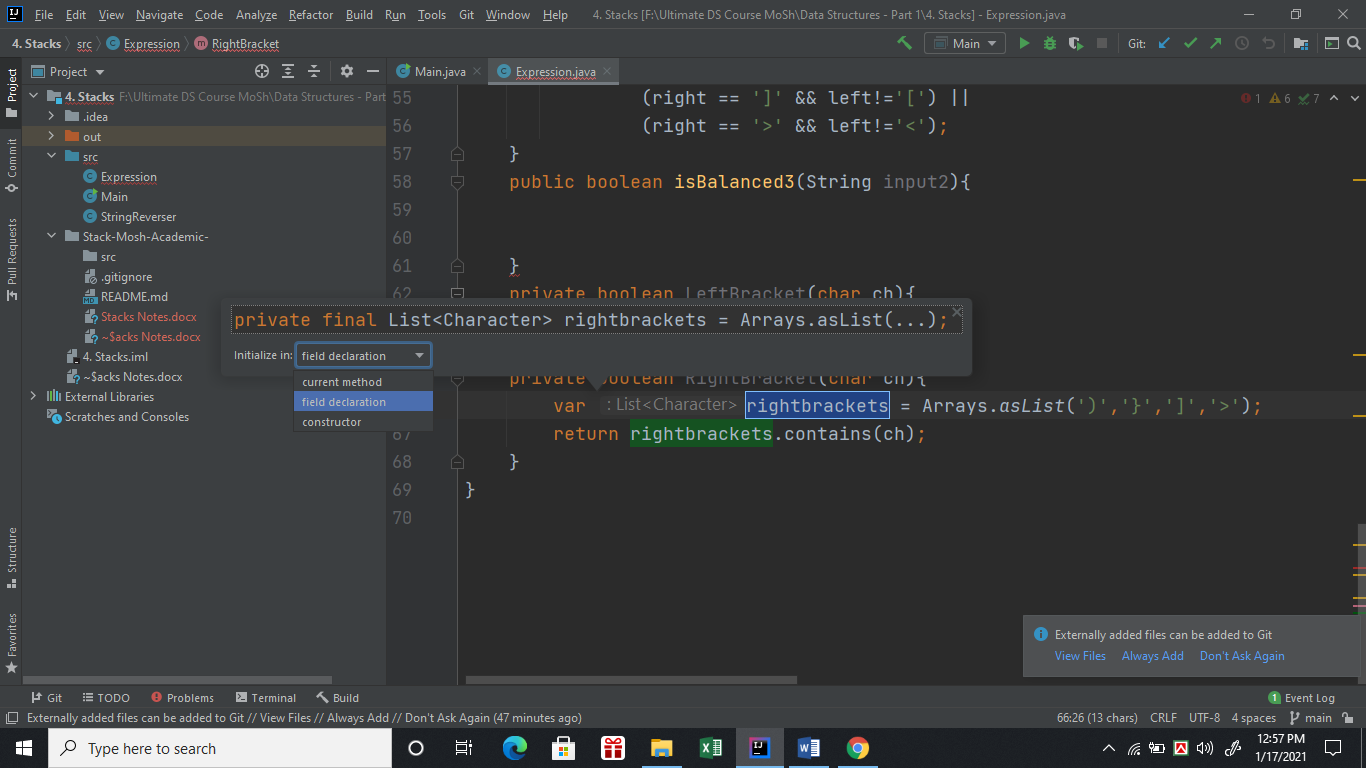
Step 2:- Right Click on the mouse then Reactor🡪 Introduce Field.



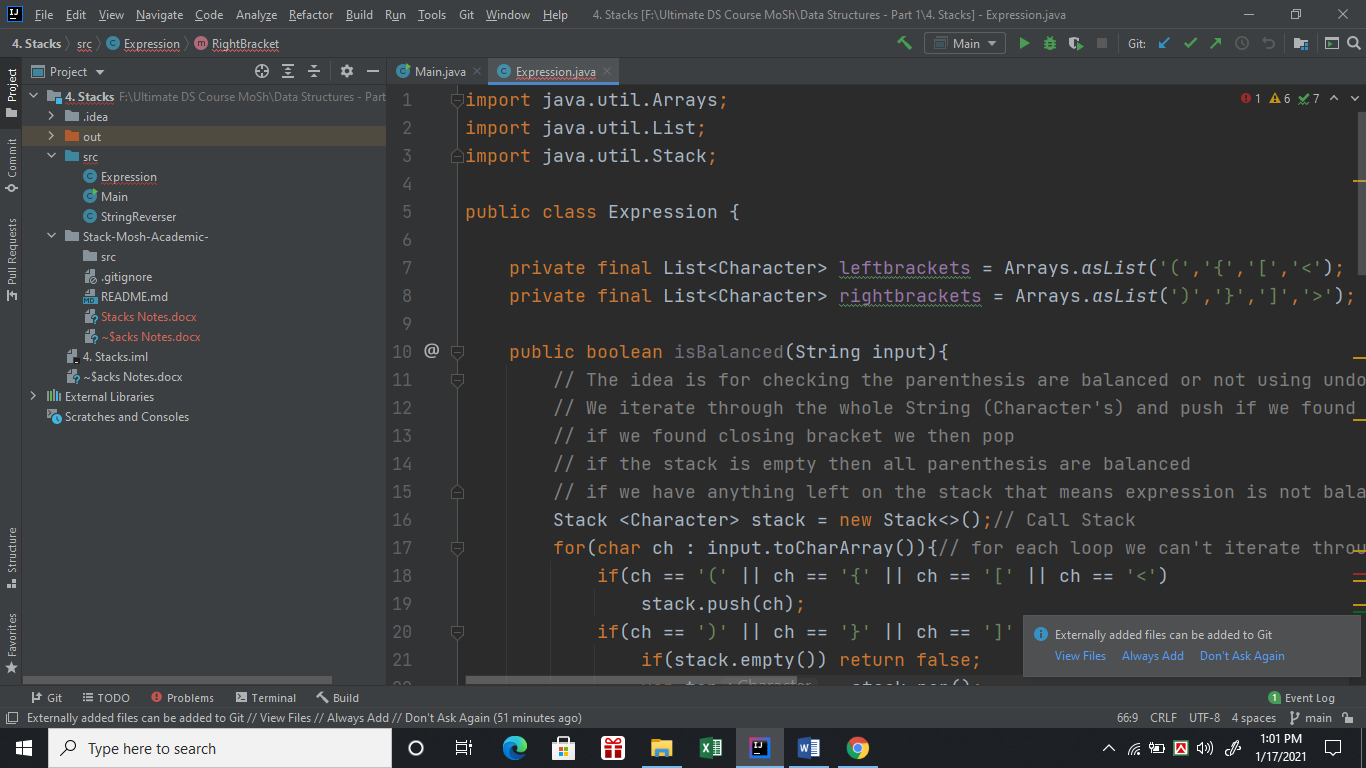
Step 3: Then we got something like the image below-



Step 4: Form initialize in dropdown menu select field declaration and hit enter



Step 5: We got our rightbrackets filed extracted at the top of the class like below:-



After Completing All the steps and extract the fields we can reuse the two arraylist as much time as we want memory is save and time also. Otherwise, as many times isBalanced method is called the ArrayList would have been initialized multiple times.

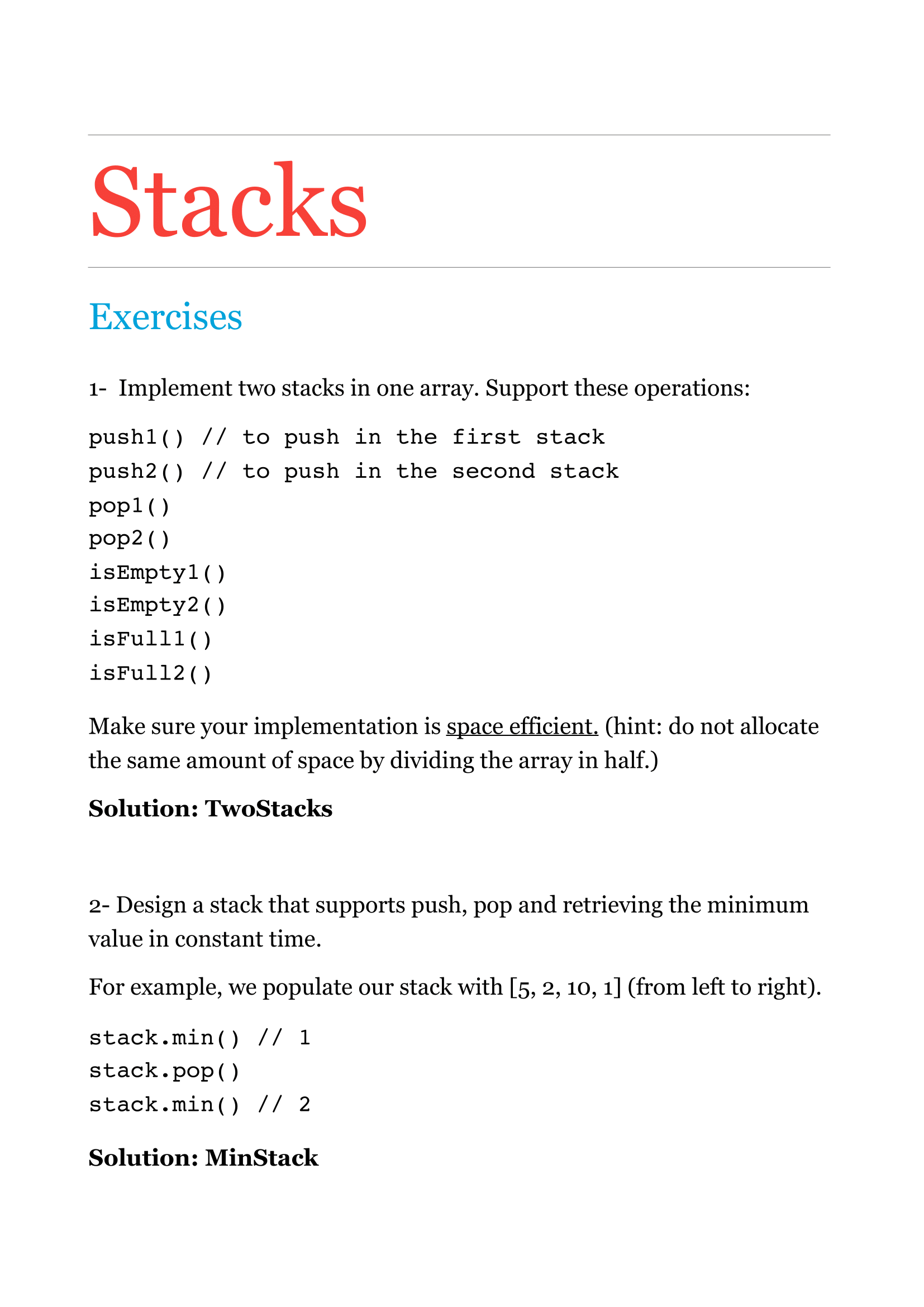
So, Now Mosh Solution 3(More Readable Refactored+):-

public boolean isBalanced3(String input2){  
 Stack<Character> stack2 = new Stack<>();  
 for(char ch : input2.toCharArray()){//pura string charcter array te niye iterate  
 if(LeftBracket(ch))// left bracket (,{,[,< paile push  
 stack2.push(ch);  
 if(RightBracket(ch)) {// right bracket ),},],> paile pop kore compare  
 if(stack2.empty()) return false;  
  
 var top = stack2.pop();  
 if(!BracketMatched(top,ch)) return false;  
 }  
 }  
 return stack2.empty();  
  
}  
private boolean LeftBracket(char ch){  
 return leftbrackets.contains(ch);  
}  
private boolean RightBracket(char ch){  
 return rightbrackets.contains(ch);  
}  
private boolean BracketMatched(char left,char right){  
 return leftbrackets.indexOf(left)==rightbrackets.indexOf(right);// index same hole match korse  
}

(V-(11+12))- Now We will be built stack using array

import java.util.Arrays;  
  
public class Stack {  
 // Basically we will be going to implement the stacks using array  
 // where below methods should be there  
 //push  
 //pop  
 //peek  
 //isEmpty  
 private int count;  
 int [] items = new int[5];// stack size limit 5  
 public void push(int item){  
 if (count==items.length)  
 throw new StackOverflowError();  
  
 items[count++]=item;  
 }  
 public int peek(){  
 if(count==0)  
 throw new IllegalStateException();  
 return items[count-1];// cause top er value count er cheye 1 kom  
 //kenona item add hoye count er value 1 bere jabe but last item jeta  
 // stack e add hoise seta toh count - 1 ei thakbe   
 }  
 public int pop(){  
 if(count==0)  
 throw new IllegalStateException();  
  
 return items[--count];// cause top er value count er cheye 1 kom  
 }  
 public boolean isEmpty(){  
 return count==0;  
 }  
 @Override  
 public String toString(){  
 var contents = Arrays.*copyOfRange*(items,0,count);// eikhane count porjonto joto gula items ase seigula copy holo  
 return Arrays.*toString*(contents);  
 }  
}

Mosh Stack Exercises-(V-13):- Seems Very Easy But actually implementation is tricky.



Code of TwoStacks:-

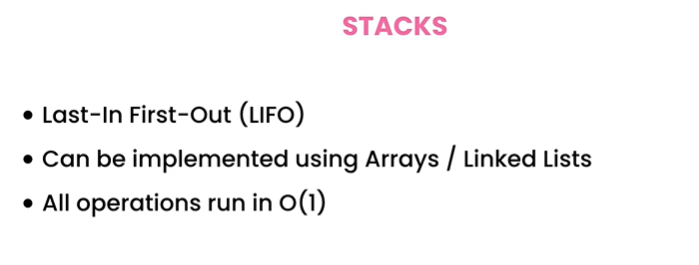
import java.util.Arrays;  
  
public class TwoStacks {  
 private int top1;  
 private int top2;  
 private int[] items;  
  
 public TwoStacks(int capacity) {  
 if (capacity <= 0)  
 throw new IllegalArgumentException("capacity must be 1 or greater.");  
  
 items = new int[capacity];  
 top1 = -1;  
 top2 = capacity;  
 }  
  
 public void push1(int item) {  
 if (isFull1())  
 throw new IllegalStateException();  
  
 items[++top1] = item;  
 }  
  
 public int pop1() {  
 if (isEmpty1())  
 throw new IllegalStateException();  
  
 return items[top1--];  
 }  
  
 public boolean isEmpty1() {  
 return top1 == -1;  
 }  
  
 public boolean isFull1() {  
 return top1 + 1 == top2;  
 }  
  
 public void push2(int item) {  
 if (isFull2())  
 throw new IllegalStateException();  
  
 items[--top2] = item;  
 }  
  
 public int pop2() {  
 if (isEmpty2())  
 throw new IllegalStateException();  
  
 return items[top2++];  
 }  
  
 public boolean isEmpty2() {  
 return top2 == items.length;  
 }  
 public boolean isFull2() {  
 return top2 - 1 == top1;  
 }  
  
 @Override  
 public String toString() {  
 return Arrays.toString(items);  
 }  
}

Code of Minstack :-

// We need two stacks to implement a min stack.  
// One stack holds the values, the other stack  
// (called minStack) holds the minimums.  
public class MinStack {  
 private Stack stack = new Stack();  
 private Stack minStack = new Stack();  
  
 public void push(int item) {  
 stack.push(item);  
  
 if (minStack.isEmpty())  
 minStack.push(item);  
 else if (item < minStack.peek())  
 minStack.push(item);  
 }

public int pop() {  
 if (stack.isEmpty())  
 throw new IllegalStateException();  
  
 var top = stack.pop();  
  
 if (minStack.peek() == top)  
 minStack.pop();  
  
 return top;  
 }  
  
 public int min() {  
 return minStack.peek();  
 }  
}

Summary-(v-14)



**[Undo or doing operations in reverse order]**