

**Код програми**

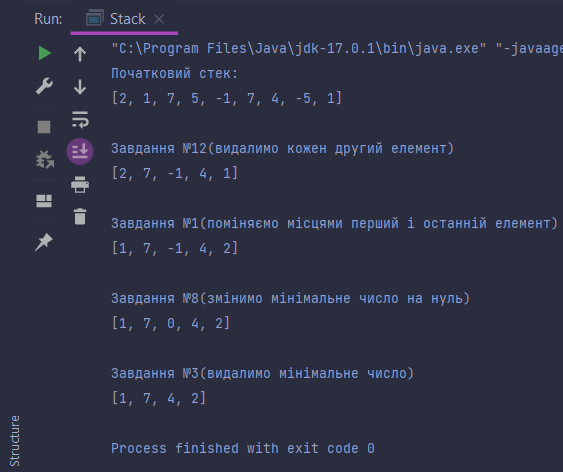
**Інтерфейс стеку**

*public interface InterfaceStack*<T> *extends Iterable*<T> {  
 *int* size();*//метод для виведення розміру стеку* T pop();*//Повертає елемент, що знаходиться у верхній частині стека, видаляючи його у процесі* T peek();*//Повертає елемент, що знаходиться у верхній частині стека, але не видаляє його.* T push(T elem);*//додає елемент у стек* T get(*int* index);*//повертає елемент за індексом  
  
 int* search(T element);*//повертає індекс елемента у стеку  
  
 boolean* isEmpty();*//Перевіряє, чи стек є порожнім.*}

**Класс Stack**

*import* java.util.*Iterator*;  
*import* java.util.NoSuchElementException;  
  
*public class* Stack<T> *implements InterfaceStack*<T> {  
  
  
 *public static void* main(String[] args) {  
 Stack<Integer> stack = *new* Stack<>();  
  
  
 stack.push(1);  
 stack.push(-5);  
 stack.push(4);  
 stack.push(7);  
 stack.push(-1);  
 stack.push(5);  
 stack.push(7);  
 stack.push(1);  
 stack.push(2);  
  
 System.***out***.println("Початковий стек: ");  
 System.***out***.println(stack);  
  
  
 System.***out***.println("\nЗавдання №12(видалимо кожен другий елемент)");  
 stack.deleteEverySecond();  
 System.***out***.println(stack);  
  
 System.***out***.println("\nЗавдання №1(поміняємо місцями перший і останній елемент)");  
 stack.swapLastAndFirst();  
 System.***out***.println(stack);  
  
 System.***out***.println("\nЗавдання №8(змінимо мінімальне число на нуль)");  
 stack.searchMin(0);  
 System.***out***.println(stack);  
  
 System.***out***.println("\nЗавдання №3(видалимо мінімальне число)");  
 stack.deleteMin();  
 System.***out***.println(stack);  
 }  
  
 *private* Node<T> firstNode;  
 *private int* size = 0;  
  
 *public* Stack() {*//створимо конструктор* firstNode = *new* Node<T>(*null*, *null*);  
 }  
  
  
 @Override  
 *public int* size() {*//повертає розмір стека  
 return* size;  
 }  
  
 @Override  
 *public* T pop() {*//Повертає елемент, що знаходиться у верхній частині стека, видаляючи його у процесі  
 if* (isEmpty()) *throw new* NoSuchElementException("Stack is empty");  
 T removeElem = firstNode.getNext().getElem();  
 firstNode.setElem(*null*);  
 firstNode = getNext(firstNode);  
 size--;  
 *return* removeElem;  
 }  
  
 @Override  
 *public* T peek() {  
 *return* firstNode.getNext().getElem();*//Повертає елемент, що знаходиться у верхній частині стека, але не видаляє його.* }  
  
 @Override  
 *public* T push(T elem) {*//додає елемент у стек* Node<T> next = firstNode;  
 next.setElem(elem);  
 firstNode = *new* Node<>(*null*, next);  
 size++;  
 *return* elem;  
 }  
  
 @Override  
 *public* T get(*int* index) {*//повертає елемент за індексом* Node<T> temp = firstNode.getNext();  
 *for* (*int* i = 0; i < index; i++) {  
 temp = getNext(temp);  
 }  
  
 *return* temp.getElem();  
 }  
  
 @Override  
 *public int* search(T element) {*//повертає індекс елемента у стеку  
 int* index = -1;  
 Node<T> temp = firstNode.getNext();  
 *for* (*int* i = 0; i < size(); i++) {  
 *if* (temp.getElem() == element) {  
 index = i;  
 *break*;  
 }  
 temp = getNext(temp);  
 }  
 *return* index;  
 }  
  
  
 @Override  
 *public boolean* isEmpty() {*//перевіряє чи стек пустий  
 return* size == 0;  
 }  
  
  
 *public void* swapLastAndFirst() {*//Завдання №1 Поміняти місцями перший і останній елементи* Node<T> temp1 = firstNode.getNext();  
 Node<T> temp2 = firstNode.getNext();  
  
  
 *for* (*int* i = 0; i < size; i++) {  
  
 *if*(temp2.getElem()==*null*){  
 i--;  
 temp2 = getNext(temp2);  
 *continue*;  
 }  
 *if*(temp2.getNext() !=*null*)temp2 = getNext(temp2);  
  
  
 }  
  
  
 T temp = temp1.getElem();  
 temp1.setElem(temp2.getElem());  
 temp2.setElem(temp);  
  
 }  
  
 *public void* deleteMin() {*//Завдання №3 Видалити мінімальний елемент стеку* Node<T> temp = firstNode.getNext();  
 Node<T> delete = firstNode.getNext();  
  
 *int* a = (*int*) temp.getElem();  
  
  
 *while* (temp != *null*) {  
  
 *if* (temp.getElem() == *null*) {  
 temp = getNext(temp);  
  
 *continue*;  
 }  
 *int* b = (*int*) temp.getElem();  
 *if* (b < a) {  
 delete = temp;  
 a = b;  
 }  
 temp = getNext(temp);  
  
 }  
  
 delete.setElem(*null*);  
 delete = getNext(delete);  
 size--;  
  
  
 }  
  
  
 *public void* searchMin(T zero) {*//Завдання №8 Знайти мінімальний елемент і вставити на його місце нуль* Node<T> temp = firstNode.getNext();  
 Node<T> delete = firstNode.getNext();  
  
  
 *int* a = (*int*) temp.getElem();  
  
 *for* (*int* i = 0; i < size(); i++) {  
  
 *if*(temp.getElem()==*null*){  
 i--;  
 temp = getNext(temp);  
 *continue*;  
 }  
 *int* b = (*int*) temp.getElem();  
 *if* (b < a) {  
 delete = temp;  
 a = b;  
  
 }  
 *if*(temp.getNext() !=*null*)temp = getNext(temp);  
  
 }  
  
  
 delete.setElem(zero);  
  
  
 }  
  
  
 *public void* deleteEverySecond() {*//Завдання №12 Видалити кожен другий елемент стеку* Node<T> temp = firstNode.getNext();  
 *int* len = *this*.size;  
 *for* (*int* i = 0; i < len; i++) {  
 *if* (i % 2 != 0) {  
 temp.setElem(*null*);  
 size--;  
 }  
 temp = getNext(temp);  
 }  
  
  
 }  
  
  
 @Override  
 *public* String toString() {*// Перевизначимо метод toString  
  
 if* (size == 0) *return* "Stack is clear";  
 StringBuilder sb = *new* StringBuilder();  
 sb.append("[");  
  
 Node<T> current = firstNode.getNext();  
 *while* (current != *null*) {  
  
 *if* (current.getElem() == *null*) {  
 current = getNext(current);  
 *continue*;  
 }  
  
  
 sb.append(current.getElem());  
 *if* (current.next == *null*) {  
 *break*;  
 }  
 current = current.next;  
  
 *if* (current != *null*) sb.append(", ");  
 }  
 sb.append("]");  
 *return* sb.toString();  
 }  
  
 *private* Node<T> getNext(Node<T> current) {  
 *return* current.getNext();  
 }  
  
 *private class* Node<T> {  
  
 *private* T elem;  
 *private* Node<T> next;  
  
 *private* Node(T elem, Node<T> next) {*//створимо конструктор  
 this*.elem = elem;  
 *this*.next = next;  
 }  
  
 *public* T getElem() {*//створимо сетери і гетери  
 return* elem;  
 }  
  
 *public void* setElem(T elem) {  
 *this*.elem = elem;  
 }  
  
  
 *public* Node<T> getNext() {  
 *return* next;  
 }  
  
 *public void* setNext(Node<T> next) {  
 *this*.next = next;  
 }  
  
  
 }  
  
 @Override  
 *public Iterator*<T> iterator() {*//створимо ітератор  
 return new* Iterator() {  
 *int* counter = 0;  
  
 @Override  
 *public boolean* hasNext() {  
 *return* counter < size;  
 }  
  
 @Override  
 *public* Object next() {  
 *return* get(counter++);  
 }  
 };  
 }  
}

**Результат програми**

****