**Stack Interface**

/\*\*

\* @author Richelin Metellus

\* A stack is a collection of object that insert and removed objects

\* according to the Last-in First-out principle(LIFO)

\*/

public interface Stack <E>{

/\*\*

\*

\* @return number of elements in the stack

\*/

int size();

/\*\*

\*

\* @return true if stack is empty.

\*/

boolean isEmpty();

/\*\*

\*

\* @param e element/object to be inserted in the stack

\*/

void push(E e);

/\*\*

\* return does not remove, the last element in the stack

\* @return the top object

\*/

E top();

/\*\*

\*

\* @return and removed the last inserted element or null if list is empty.

\*/

E pop();

}

**ArrayStack Class**

/\*\*

\*

\* @author Richelin Metellus

\* @version 02/24/2017

\* @param <E> generic type for object

\*/

public class ArrayStack <E> implements Stack<E> {

public static final int CAPACITY = 1000;

private E[] data;

private int t = -1; // index of the top element in stack

public ArrayStack(){ this(CAPACITY);}

public ArrayStack(int capacity)

{

data = (E[]) new Object[capacity];

}

@Override

public int size()

{

return t+1;

}

public boolean isEmpty()

{

return t==-1;

}

public void push(E e) throws IllegalStateException

{

if (size() == data.length) throw new IllegalStateException("Stack is full");

data[++t] = e;

}

public E top()

{

if (isEmpty()) return null;

return data[t];

}

public E pop()

{

if(isEmpty()) return null;

E answer = data[t];

data[t] = null;

t--;

return answer;

}

}

**LinkedStack Class**

/\*\*

\* This is based on textbook code fragment 6.4

\* This stack implementation add the last in element at the head.

\* @author Richelin Metellus

\* @param <E>

\*/

public class LinkedStack<E> implements Stack<E> {

private SinglyLinkedList<E> list = new SinglyLinkedList<>();

public LinkedStack(){} //empty list

@Override

public int size()

{

return list.size();

}

@Override

public boolean isEmpty()

{

return list.isEmpty();

}

@Override

public void push(E element)

{

list.addFirst(element);

}

@Override

public E top(){return list.first();}

@Override

public E pop(){return list.removeFirst();}

}

**ArrayListStack Class**

/\*\*

\*The implementation of a stack using array of the reference of the object

\* in the list.

\* @author Rich Metelus

\* @version 2/24/2017

\*/

public class ArrayListStack<E> implements Stack<E> {

private ArrayList<E> list = new ArrayList<>();

public ArrayListStack(){} //empty list

@Override

public int size()

{

return list.size();

}

@Override

public boolean isEmpty()

{

return list.isEmpty();

}

@Override

public void push(E element)

{

list.add(0,element);

}

@Override

public E top(){return list.get(list.size()-1);}

@Override

public E pop(){return list.remove(0);}

}

**ArrayStackBad Class**

/\*\*

\* Implementation of the stack where the newest element is always added at index 0

\* @author Richelin Metellus

\* @version 02/24/2017

\* @param <E>

\*/

public class ArrayStackBad<E> implements Stack<E> {

public static final int CAPACITY = 1000;

private E[] data;

private int top\_oldest = -1; //oldest element = indexReference/ position of first element added in the array.

// so last element added always at indext 0

public ArrayStackBad(){ this(CAPACITY);}

public ArrayStackBad(int capacity)

{

data = (E[]) new Object[capacity];

}

@Override

public int size()

{

return top\_oldest+1;

}

@Override

public boolean isEmpty()

{

return top\_oldest==-1;

}

@Override

public void push(E e) throws IllegalStateException

{

if (size() == data.length) throw new IllegalStateException("Stack is full");

if(!(isEmpty()))

{

int temp = top\_oldest+1;

for(int i = top\_oldest; i>=0; --i)

{

data[temp]= data[i]; // shifting each element right to a higher index.

temp--;

}

data[temp] = e; // temp is = 0 after loop exit. this is similar to data[i]= e, i is = 0 after loop exit

// new element/last item in the stack is added at index 0.

top\_oldest++; // update index since one more element is added.

}

else

data[++top\_oldest] = e;

}

@Override

public E top()

{

if (isEmpty()) return null;

return data[0];

}

@Override

public E pop()

{

if(isEmpty()) return null;

E answer = data[0];

for(int i=0; i <top\_oldest; ++i)

{

int j = i+1;

data[i] = data[j];

}

top\_oldest--;

return answer;

}

}

**LinkedStackBad Class**

/\*\*

\* Using the adapter pattern to implement a stack as

\* a linked list. Each new element is added as the last node and the

\* last node element is removed.(LiFO)

\* @author Rich

\* @version 02/24/2017

\*/

public class LinkedStackBad<E> implements Stack<E> {

private SinglyLinkedList<E> list = new SinglyLinkedList<>();

public LinkedStackBad(){} //empty list

public int size()

{

return list.size();

}

public boolean isEmpty()

{

return list.isEmpty();

}

public void push(E element)

{

list.addLast(element);

}

public E top(){return list.first();}

public E pop(){return list.removeLast();}

}

**Client Class**

import java.util.Random;

/\*\*

\*Testing the various runtime of each stack implementation.

\* @author Richelin Metellus

\* @version 02/24/2017

\*/

public class StackTest {

public static void main(String[] args) {

ArrayStack<Integer> goodArrayStack = new ArrayStack<>(100000);

LinkedStack<Integer> goodLinkedStack = new LinkedStack<>();

ArrayListStack<Integer> ArrListStack = new ArrayListStack<>();

ArrayStackBad<Integer> badArrayStack = new ArrayStackBad<>(100000);

LinkedStackBad<Integer> badLinkedStack = new LinkedStackBad<>();

Random rand = new Random();

// running time check for ArrayStack

long startTime1 = System.currentTimeMillis();

for ( int i = 0; i < 100000; ++i)

{

int randNumber = rand.nextInt(100);

goodArrayStack.push(randNumber);

}

for ( int i = 0; i < 100000; ++i)

goodArrayStack.pop();

long endTime1 = System.currentTimeMillis();

long elapsedTime1 = endTime1 - startTime1;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// runing time for LinkedStack

long startTime2 = System.currentTimeMillis();

for ( int i = 0; i < 100000; ++i)

{

int randNumber = rand.nextInt(100);

goodLinkedStack.push(randNumber);

}

for ( int i = 0; i < 100000; ++i)

goodLinkedStack.pop();

long endTime2 = System.currentTimeMillis();

long elapsedTime2 = endTime2 - startTime2;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// runtime for ArrayListStack

long startTime3 = System.currentTimeMillis();

for ( int i = 0; i < 100000; ++i)

{

int randNumber = rand.nextInt(100);

ArrListStack.push(randNumber);

}

for ( int i = 0; i < 100000; ++i)

ArrListStack.pop();

long endTime3 = System.currentTimeMillis();

long elapsedTime3 = endTime3 - startTime3;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// running time for ArrayStackBad

long startTime4 = System.currentTimeMillis();

for ( int i = 0; i < 100000; ++i)

{

int randNumber = rand.nextInt(100);

badArrayStack.push(randNumber);

}

for ( int i = 0; i < 100000; ++i)

badArrayStack.pop();

long endTime4 = System.currentTimeMillis();

long elapsedTime4 = endTime4 - startTime4;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// running time for LinkedStackBad

long startTime5 = System.currentTimeMillis();

for ( int i = 0; i < 100000; ++i)

{

int randNumber = rand.nextInt(100);

badLinkedStack.push(randNumber);

}

for ( int i = 0; i < 100000; ++i)

badLinkedStack.pop();

long endTime5 = System.currentTimeMillis();

long elapsedTime5 = endTime5 - startTime5;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

System.out.printf("push/pop ArrayStack \tfor N\t= 100,000 \ttime \t= %,10d miliseconds \n", elapsedTime1);

System.out.printf("push/pop LinkedStack \tfor N\t= 100,000 \ttime \t= %,10d miliseconds \n", elapsedTime2);

System.out.printf("push/pop ArrayListStack for N\t= 100,000 \ttime \t= %,10d miliseconds \n", elapsedTime3);

System.out.printf("push/pop ArrayStackBad \tfor N\t= 100,000 \ttime \t= %,10d miliseconds \n", elapsedTime4);

System.out.printf("push/pop LinkedStackBad for N\t= 100,000 \ttime \t= %,10d miliseconds \n", elapsedTime5);

}

}

**Output**

run:

push/pop ArrayStack for N = 100,000 time = 22 miliseconds

push/pop LinkedStack for N = 100,000 time = 16 miliseconds

push/pop ArrayListStack for N = 100,000 time = 11,045 miliseconds

push/pop ArrayStackBad for N = 100,000 time = 10,710 miliseconds

push/pop LinkedStackBad for N = 100,000 time = 14,989 miliseconds

BUILD SUCCESSFUL (total time: 36 seconds)