<https://github.com/aboman17/CS240HW.git>

\*\*

David Roura

\*/

public final class LinkedStack<T> implements StackInterface<T>

{

private Node topNode;

public LinkedStack()

{

topNode = null;

}

public void push(T newEntry)

{

topNode = new Node(newEntry, topNode);

}

public T top()

{

T top = peek();

assert topNode != null;

topNode = topNode.getNextNode();

return top;

}

public T peek()

{

if(isEmpty())

throw new EmptyStackException();

else

return topNode.getData();

}

public boolean isEmpty()

{

return topNode == null;

}

public void clear()

{

topNode = null;

}

}

/\*\*

David Roura

\*/

public final class VectorStack<T> implements StackInterface<T>

{

private Vector<T> stack;

private boolean initialized = false;

private static final int DEFAULT\_CAPACITY = 25;

private static final int MAX\_CAPACITY = 10000;

public VectorStack()

{

this(DEFAULT\_CAPACITY);

}

public VectorStack(int initialCapacity)

{

checkCapacity(initialCapacity);

stack = new Vector<>(initialCapacity);

initialized = true;

}

public void push(T newEntry)

{

checkInitialization();

stack.add(newEntry);

}

public T pop()

{

checkInitialization();

if(isEmpty())

throw new EmptyStackException();

else

return stack.remove(stack.size()-1);

}

public boolean isEmpty()

{

return stack.isEmpty();

}

public void clear()

{

stack.clear();

}

}

/\*\*

David Roura

\*/

public final class ArrayStack<T> implements StackInterface<T>

{

private T[] stack;

private int topIndex;

private boolean initialized = false;

private static final int DEFAULT\_CAPACITY = 10;

private static final int MAX\_CAPACITY = 10000;

public ArrayStack()

{

this(DEFAULT\_CAPACITY);

}

public ArrayStack(int intialCapacity)

{

checkCapacity(initialCapacity);

@SuppressWarnings("unchecked")

T[]ctempStack = (T[])new Object[initialCapacity];

stack = tempStack;

topIndex = -1;

initialized = true;

}

public void push(T newEntry)

{

checkInitialization();

ensureCapacity();

stack[topIndex + 1] = newEntry;

topIndex++;

}

private void ensureCapacity()

{

if (topIndex == stack.length -1)

{

int newLength = 2 \* stack.length;

checkCapacity(newLength);

stack = Arrays.copyOf(stack, newLength);

}

}

public T pop()

{

checkInitialization();

if (isEmpty())

throw new EmptyStackException();

else

{

T top = stack[topIndex];

stack[topIndex] = null;

topIndex--;

return top;

}

}

public T peek()

{

checkInitialization();

if (isEmpty())

throw new EmptyStackException();

else

return stack[topIndex];

}

public boolean isEmpty()

{

return topIndex < 0;

}

public void clear()

{

topIndex = -1

}

}

/\*\*

David Roura

\*/

public final class ArrayQueue<T> implements QueueInterface<T>

{

private T[] queue;

private int frontIndex;

private int backIndex;

private boolean initialized = false;

private static final int DEFAULT\_CAPACITY = 10;

private static final int MAX\_CAPACITY = 10000;

public ArrayQueue(int initialCapacity)

{

checkCapacity(initialCapacity)

@SuppressWarnings("unchecked")

T[] tempQueue = (T[]) new Object[initialCapacity + 1];

queue = tempQueue;

frontIndex = 0;

backIndex = initialCapacity;

initialized = true;

}

public void enqueue(T newEntry)

{

checkInitialization();

ensureCapacity();

backIndex = (backIndex + 1) % queue.length;

queue[backIndex] = newEntry;

}

public T dequeue()

{

checkInitialization();

if(isEmpty())

throw new EmptyQueueException();

else

{

T front = queue[frontIndex];

queue[frontIndex] = null;

frontIndex = (frontIndex + 1) % queue.length;

return front;

}

}

public T getFront()

{

checkInitialization();

if(isEmpty())

throw new EmptyQueueException();

else

return queue[frontIndex];

}

public boolean isEmpty()

{

return frontIndex == ((backIndex + 1) % queue.length);

}

public void clear()

{

frontIndex = 0;

backIndex = queue.length-1;

}

}

/\*\*

David Roura

\*/

public final class LinkedQueue<T> implements QueueInterface<T>

{

private Node firstNode;

private Node latNode;

public LinkedQueue()

{

firstNode = null;

lastNode = null;

}

public void enqueue (T newEntry, null);

{

if(isEmpty())

firstNode = newNode;

else

lastNode = newNode;

}

public T getFront()

{

if (isEmpty())

throw new EmptyQueueException();

else

return firstNode.getData();

}