Q1:

/\*//*TODO: uncomment if needed*  
import java.util.ArrayList;  
import java.util.Objects;  
  
public class DirectoryEntry {  
 private String name; // person’s name  
 private String number; // person’s phone number  
 public void setNumber(String phoneNum) { number = phoneNum; }  
 public String getNumber() { return number; }  
 public void setName(String phoneName) {name = phoneName;}  
 public String getName() {return name;}  
  
  
  
}  
  
class Test {  
 ArrayList<DirectoryEntry> theDirectory = new ArrayList<>();  
\*///*TODO: uncomment if needed* // aName: The name of the person being added or changed  
 // newNumber: The new number to be assigned  
 public String addOrChangeEntry(String name, String newNumber) {  
 boolean nameFound = false;  
 String oldNumber = null;  
 for (DirectoryEntry i: theDirectory) {  
 if (Objects.equals(i.getName(), name)) {  
 oldNumber = i.getNumber();  
 i.setNumber(newNumber);  
 nameFound = true;  
 }  
 }  
 if (!nameFound) {  
 DirectoryEntry newDirectory = new DirectoryEntry();  
 newDirectory.setName(name);  
 newDirectory.setNumber(newNumber);  
 theDirectory.add(newDirectory);  
 }  
 return oldNumber;  
 }  
  
 // aName: The name of the person to be removed  
 public DirectoryEntry remove (String aName) {  
 DirectoryEntry oldDirectory = null;  
 for (DirectoryEntry i: theDirectory) {  
 if (Objects.equals(i.getName(), aName)){  
 oldDirectory = i;  
 theDirectory.remove(i);  
 }  
 }  
 return oldDirectory;  
 }  
/\*//*TODO: uncomment if needed*  
 public static void main(String[] args) {  
 Test t = new Test();  
 DirectoryEntry d1 = new DirectoryEntry(); d1.setName("Jack"); d1.setNumber("111");  
 DirectoryEntry d2 = new DirectoryEntry(); d2.setName("Mohamed"); d2.setNumber("222");  
 t.theDirectory.add(d1); t.theDirectory.add(d2);  
 System.out.println(t.addOrChangeEntry("Juntang", "333"));  
 System.out.println(t.remove("Mohamed"));  
 for (DirectoryEntry d : t.theDirectory) {  
 System.out.println(d.getName() + "\t" +d.getNumber());  
 }  
 }  
}  
\*///*TODO: uncomment if needed*

DirectoryEntry d1 = new DirectoryEntry(); d1.setName("Jack"); d1.setNumber("111");  
 DirectoryEntry d2 = new DirectoryEntry(); d2.setName("Mohamed"); d2.setNumber("222");  
 t.theDirectory.add(d1); t.theDirectory.add(d2);  
 System.out.println(t.addOrChangeEntry("Juntang", "333"));  
 System.out.println(t.remove("Mohamed"));  
 for (DirectoryEntry d : t.theDirectory) {  
 System.out.println(d.getName() + "\t" +d.getNumber());  
 }  
 }  
}  
\*///*TODO: uncomment if needed*

Q2.

import java.util.ArrayList;  
import java.util.Scanner;  
  
public class PINCombinations {  
 // Array mapping digits to their corresponding letters  
 private static final String[] *KEYPAD* = {"", "", "ABC", "DEF", "GHI", "JKL", "MNO", "PQRS", "TUV", "WXYZ"};  
  
 public static void main(String[] args) {  
 try (Scanner scanner = new Scanner(System.*in*)) {  
 int attemptCount = 0;  
  
 while (true) {  
 System.*out*.print("Enter a PIN with a maximum of 6 digits: ");  
 String input = scanner.nextLine();  
  
 // Check if the input is valid  
 if (input.matches("[2-9]{1,6}")) {  
 ArrayList<String> combinations = new ArrayList<>();  
 *generateCombinations*(input, 0, "", combinations);  
 System.*out*.println("Possible strings: " + combinations);  
 break;  
 } else {  
 attemptCount++;  
 if (attemptCount >= 10) {  
 throw new IllegalArgumentException("Too many incorrect attempts. Please enter a valid PIN.");  
 }  
 System.*out*.println("Invalid input. Please try again.");  
 }  
 }  
 } catch (IllegalArgumentException e) {  
 System.*out*.println("IllegalArgumentException thrown" + e);  
 }  
 }  
  
 private static void generateCombinations(String digits, int index, String current, ArrayList<String> result) {  
 if (index == digits.length()) {  
 result.add(current);  
 return;  
 }  
  
 String letters = *KEYPAD*[digits.charAt(index) - '0'];  
 for (int i = 0; i < letters.length(); i++) {  
 *generateCombinations*(digits, index + 1, current + letters.charAt(i), result);  
 }  
 }  
}

Q3.

import java.util.Stack;  
  
public class MyQueue<E> {  
 // define attributes required  
 private Stack<E> inStack = new Stack<>();  
 private Stack<E> outStack = new Stack<>();  
  
 // add ‘e’ to the back of the queue  
 public void enQueue(E e) {  
 inStack.push(e);  
 }  
 // remove the element from the front of the queue and return it  
 public E deQueue() {  
 if (isEmpty()) {  
 throw new IllegalStateException("Queue is empty");  
 }  
  
 if (outStack.isEmpty()) {  
 while (!inStack.isEmpty()) {  
 outStack.push(inStack.pop());  
 }  
 }  
 return outStack.pop();  
 }  
 // return the element at the front of the queue  
 public E peek() {  
 if (isEmpty()) {  
 throw new IllegalStateException("Queue is empty");  
 }  
  
 if (outStack.isEmpty()) {  
 while (!inStack.isEmpty()) {  
 outStack.push(inStack.pop());  
 }  
 }  
 return outStack.peek();  
 }  
 // return true if the queue is empty and not otherwise  
 public boolean isEmpty() {  
 return inStack.isEmpty() && outStack.isEmpty();  
 }  
 // test the methods above, print if necessary [5 points]  
 public static void main(String[] str) {  
 MyQueue<Integer> queue = new MyQueue<>();  
  
 queue.enQueue(1);  
 queue.enQueue(2);  
 queue.enQueue(3);  
  
 System.*out*.println("First element: " + queue.peek());  
 System.*out*.println("Removed element: " + queue.deQueue());  
 System.*out*.println("Next element: " + queue.peek());  
  
 queue.enQueue(4);  
 while (!queue.isEmpty()) {  
 System.*out*.println("Dequeue: " + queue.deQueue());  
 }  
 }  
}

while (!queue.isEmpty()) {  
 System.*out*.println("Dequeue: " + queue.deQueue());  
 }  
 }  
}

Q4.

public class MyLinkedList<E> {  
 private static class MyNode<E> {  
 E item;  
 MyNode<E> next;  
  
 MyNode(E element) {  
 this.item = element;  
 this.next = null;  
 }  
 }  
  
 private MyNode<E> head;  
  
 // Reverse elements in the list from position start to end  
 public boolean reverse(int start, int end) throws Exception {  
 if (start >= end || start < 0) {  
 throw new IllegalArgumentException("Invalid start and end positions");  
 }  
  
 // Dummy node to simplify edge cases  
 MyNode<E> dummy = new MyNode<E>(null);  
 dummy.next = head;  
 MyNode<E> prev = dummy;  
  
 for (int i = 0; i < start; i++) {  
 if (prev.next == null) {  
 throw new Exception("Start position is out of bounds");  
 }  
 prev = prev.next;  
 }  
  
 MyNode<E> current = prev.next;  
 MyNode<E> next;  
  
 for (int i = start; i < end; i++) {  
 if (current.next == null) {  
 throw new Exception("End position is out of bounds");  
 }  
 next = current.next;  
 current.next = next.next;  
 next.next = prev.next;  
 prev.next = next;  
 }  
  
 if (start == 0) {  
 head = dummy.next;  
 }  
  
 return true;  
 }  
  
 // Add at the beginning for simplicity  
 public void addFirst(E item) {  
 MyNode<E> newNode = new MyNode<>(item);  
 newNode.next = head;  
 head = newNode;  
 }  
  
 // Print the list for testing  
 public void printList() {  
 MyNode<E> current = head;  
 while (current != null) {  
 if (current.next == null) {  
 System.*out*.println(current.item);  
 break;  
 }  
 System.*out*.print(current.item + " -> ");  
 current = current.next;  
 }  
 //System.out.println("null");  
 }  
  
 // Test method  
 public static void main(String[] args) {  
 MyLinkedList<String> list = new MyLinkedList<>();  
 list.addFirst("E");  
 list.addFirst("D");  
 list.addFirst("C");  
 list.addFirst("B");  
 list.addFirst("A");  
  
 System.*out*.println("Original list:");  
 list.printList();  
  
 try {  
 list.reverse(1, 3);  
 System.*out*.println("Reversed list:");  
 list.printList();  
 } catch (Exception e) {  
 System.*out*.println("Error: " + e.getMessage());  
 }  
 }  
}

public void addFirst(E item) {  
 MyNode<E> newNode = new MyNode<>(item);  
 newNode.next = head;  
 head = newNode;  
 }  
  
 // Print the list for testing  
 public void printList() {  
 MyNode<E> current = head;  
 while (current != null) {  
 if (current.next == null) {  
 System.*out*.println(current.item);  
 break;  
 }  
 System.*out*.print(current.item + " -> ");  
 current = current.next;  
 }  
 //System.out.println("null");  
 }  
  
 // Test method  
 public static void main(String[] args) {  
 MyLinkedList<String> list = new MyLinkedList<>();  
 list.addFirst("E");  
 list.addFirst("D");  
 list.addFirst("C");  
 list.addFirst("B");  
 list.addFirst("A");  
  
 System.*out*.println("Original list:");  
 list.printList();  
  
 try {  
 list.reverse(1, 3);  
 System.*out*.println("Reversed list:");  
 list.printList();  
 } catch (Exception e) {  
 System.*out*.println("Error: " + e.getMessage());  
 }  
 }  
}