1.A

class Node {

int data;

Node next;

public Node(int data) {

this.data = data;

this.next = null;

}

}

class CustomLinkedList {

Node head;

public CustomLinkedList() {

this.head = null;

}

public void insertAtBeginning(int data) {

Node new\_node = new Node(data);

if (head == null) {

head = new\_node;

} else {

new\_node.next = head;

head = new\_node;

}

}

public void insertAtEnd(int data) {

Node new\_node = new Node(data);

if (head == null) {

head = new\_node;

} else {

Node current = head;

while (current.next != null) {

current = current.next;

}

current.next = new\_node;

}

}

public void insertAtPosition(int position, int data) {

if (position == 0) {

insertAtBeginning(data);

} else {

Node new\_node = new Node(data);

Node current = head;

for (int i = 0; i < position - 1; i++) {

if (current.next != null) {

current = current.next;

} else {

System.out.println("Position out of range");

return;

}

}

new\_node.next = current.next;

current.next = new\_node;

}

}

public void deleteNode(int data) {

if (head == null) {

return;

}

if (head.data == data) {

head = head.next;

return;

}

Node current = head;

while (current.next != null) {

if (current.next.data == data) {

current.next = current.next.next;

return;

}

current = current.next;

}

}

public void updateNode(int old\_data, int new\_data) {

Node current = head;

while (current != null) {

if (current.data == old\_data) {

current.data = new\_data;

return;

}

current = current.next;

}

}

public void displayAllNodes() {

Node current = head;

while (current != null) {

System.out.print(current.data + " ");

current = current.next;

}

System.out.println();

}

}

public class Main {

public static void main(String[] args) {

CustomLinkedList linked\_list = new CustomLinkedList();

linked\_list.insertAtEnd(1);

linked\_list.insertAtEnd(2);

linked\_list.insertAtEnd(3);

linked\_list.insertAtBeginning(0);

linked\_list.insertAtPosition(2, 4);

linked\_list.displayAllNodes(); // Output: 0 1 4 2 3

linked\_list.updateNode(1, 10);

linked\_list.displayAllNodes(); // Output: 0 10 4 2 3

linked\_list.deleteNode(2);

linked\_list.displayAllNodes(); // Output: 0 10 4 3

}

}

2.A

class CustomStack {

int[] stack;

int top;

public CustomStack(int size) {

stack = new int[size];

top = -1;

}

public void push(int value) {

if (top >= stack.length - 1) {

System.out.println("Stack is full. Cannot push more elements.");

} else {

stack[++top] = value;

}

}

public int pop() {

if (top < 0) {

System.out.println("Stack is empty. Cannot pop elements.");

return -1;

} else {

return stack[top--];

}

}

public int peek() {

if (top < 0) {

System.out.println("Stack is empty. Cannot peek.");

return -1;

} else {

return stack[top];

}

}

public boolean isEmpty() {

return top < 0;

}

}

public class Main {

public static void main(String[] args) {

CustomStack stack = new CustomStack(5);

stack.push(1);

stack.push(2);

stack.push(3);

stack.push(4);

stack.push(5);

System.out.println("Stack after pushing elements:");

while (!stack.isEmpty()) {

System.out.print(stack.pop() + " ");

}

System.out.println();

}

}

Output:

Stack after pushing elements:

5 4 3 2 1

2.B

class CustomQueue<T> {

private T[] queue;

private int front;

private int rear;

private int size;

public CustomQueue(int capacity) {

this.queue = (T[]) new Object[capacity];

this.front = 0;

this.rear = -1;

this.size = 0;

}

public void enqueue(T item) {

if (rear == queue.length - 1) {

System.out.println("Queue is full. Cannot enqueue more elements.");

return;

}

queue[++rear] = item;

size++;

}

public T dequeue() {

if (size == 0) {

System.out.println("Queue is empty. Cannot dequeue elements.");

return null;

}

T item = queue[front];

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

size--;

return item;

}

public T peek() {

if (size == 0) {

System.out.println("Queue is empty. Cannot peek.");

return null;

}

return queue[front];

}

public boolean isEmpty() {

return size == 0;

}

}

public class Main {

public static void main(String[] args) {

CustomQueue<String> stringQueue = new CustomQueue<>(5);

CustomQueue<Integer> intQueue = new CustomQueue<>(5);

// Enqueue strings

stringQueue.enqueue("apple");

stringQueue.enqueue("banana");

stringQueue.enqueue("cherry");

// Enqueue integers

intQueue.enqueue(1);

intQueue.enqueue(2);

intQueue.enqueue(3);

System.out.println("String queue:");

while (!stringQueue.isEmpty()) {

System.out.print(stringQueue.dequeue() + " ");

}

System.out.println();

System.out.println("Integer queue:");

while (!intQueue.isEmpty()) {

System.out.print(intQueue.dequeue() + " ");

}

System.out.println();

}

}

Output:

String queue:

apple banana cherry

Integer queue:

1 2 3

3.A

class Patient implements Comparable<Patient> {

private String name;

private int urgency;

public Patient(String name, int urgency) {

this.name = name;

this.urgency = urgency;

}

public String getName() {

return name;

}

public int getUrgency() {

return urgency;

}

public int compareTo(Patient other) {

return Integer.compare(other.urgency, this.urgency);

}

}

class EmergencyRoom {

private PriorityQueue<Patient> priorityQueue;

public EmergencyRoom() {

priorityQueue = new PriorityQueue<>();

}

public void admit(Patient patient) {

priorityQueue.offer(patient);

}

public Patient serve() {

return priorityQueue.poll();

}

public boolean isEmpty() {

return priorityQueue.isEmpty();

}

}

public class Main {

public static void main(String[] args) {

EmergencyRoom emergencyRoom = new EmergencyRoom();

emergencyRoom.admit(new Patient("John", 5));

emergencyRoom.admit(new Patient("Alice", 3));

emergencyRoom.admit(new Patient("Bob", 7));

emergencyRoom.admit(new Patient("Charlie", 2));

while (!emergencyRoom.isEmpty()) {

Patient patient = emergencyRoom.serve();

System.out.println("Serving patient: " + patient.getName() + " (urgency: " + patient.getUrgency() + ")");

}

}

}

Output:

Serving patient: Bob (urgency: 7)

Serving patient: John (urgency: 5)

Serving patient: Alice (urgency: 3)

Serving patient: Charlie (urgency: 2)