Name: Dhruv Sharma

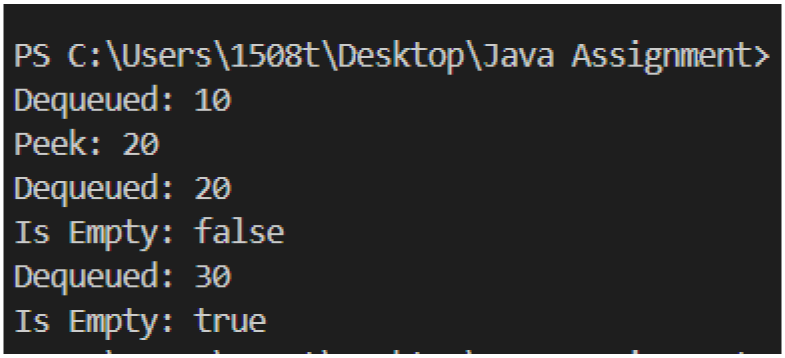
UID: 22BCS13922

Section: 612-“B”

AP ASSIGNMENT 6

Implement Queue using Stack

```
import java.util.Stack;

public class Tutorial {
    Stack<Integer> stack1 = new Stack<>();
    Stack<Integer> stack2 = new Stack<>();

    public void enqueue(int value) {
        stack1.push(value);
    }

    public int dequeue() {
        if (isEmpty()) {
            throw new RuntimeException("Queue is empty");
        }
        if (stack2.isEmpty()) {
            while (!stack1.isEmpty()) {
                stack2.push(stack1.pop());
            }
        }
        return stack2.pop();
    }

    public int peek() {
        if (isEmpty()) {
            throw new RuntimeException("Queue is empty");
        }
        if (stack2.isEmpty()) {
            while (!stack1.isEmpty()) {
                stack2.push(stack1.pop());
            }
        }
        return stack2.peek();
    }

    public boolean isEmpty() {
        return stack1.isEmpty() && stack2.isEmpty();
    }

    public static void main(String[] args) {
        Tutorial queue = new Tutorial();

        queue.enqueue(10);
        queue.enqueue(20);
        queue.enqueue(30);

        System.out.println("Dequeued: " + queue.dequeue());
        System.out.println("Peek: " + queue.peek());
        System.out.println("Dequeued: " + queue.dequeue());
        System.out.println("Is Empty: " + queue.isEmpty());
        System.out.println("Dequeued: " + queue.dequeue());
        System.out.println("Is Empty: " + queue.isEmpty());
    }
}
```

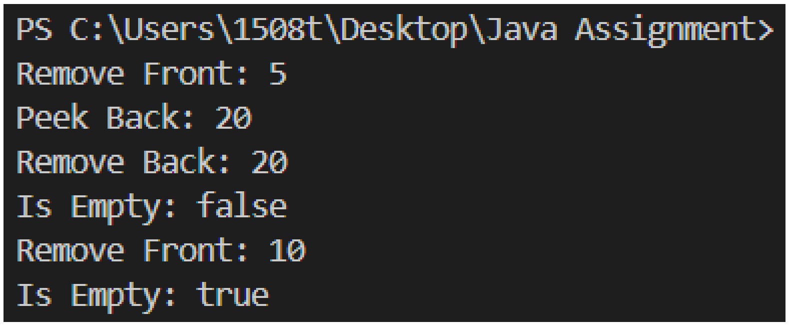


```
PS C:\Users\1508t\Desktop\Java Assignment>
Dequeued: 10
Peek: 20
Dequeued: 20
Is Empty: false
Dequeued: 30
Is Empty: true
```

Implement Deque using Stack

```
import java.util.Stack;

public class Tutorial {
    Stack<Integer> frontStack = new Stack<>();
    Stack<Integer> backStack = new Stack<>();

    public void addFront(int value) {
        frontStack.push(value);
    }

    public void addBack(int value) {
        backStack.push(value);
    }

    public int removeFront() {
        if (isEmpty()) {
            throw new RuntimeException("Deque is empty");
        }
        if (frontStack.isEmpty()) {
            while (!backStack.isEmpty()) {
                frontStack.push(backStack.pop());
            }
    }
    return frontStack.pop();
}

public int removeBack() {
    if (isEmpty()) {
        throw new RuntimeException("Deque is empty");
    }
    if (backStack.isEmpty()) {
        while (!frontStack.isEmpty()) {
            backStack.push(frontStack.pop());
        }
    }
    return backStack.pop();
}

public int peekFront() {
    if (isEmpty()) {
        throw new RuntimeException("Deque is empty");
    }
    if (frontStack.isEmpty()) {
        while (!backStack.isEmpty()) {
            frontStack.push(backStack.pop());
        }
    }
    return frontStack.peek();
}

public int peekBack() {
    if (isEmpty()) {
        throw new RuntimeException("Deque is empty");
    }
    if (backStack.isEmpty()) {
        while (!frontStack.isEmpty()) {
            backStack.push(frontStack.pop());
        }
    }
    return backStack.peek();
}

public boolean isEmpty() {
    return frontStack.isEmpty() && backStack.isEmpty();
}

public static void main(String[] args) {
    Tutorial deque = new Tutorial();

    deque.addFront(10);
    deque.addBack(20);
    deque.addFront(5);

    System.out.println("Remove Front: " + deque.removeFront());
    System.out.println("Peek Back: " + deque.peekBack());
    System.out.println("Remove Back: " + deque.removeBack());
    System.out.println("Is Empty: " + deque.isEmpty());
    System.out.println("Remove Front: " + deque.removeFront());
    System.out.println("Is Empty: " + deque.isEmpty());
}
```

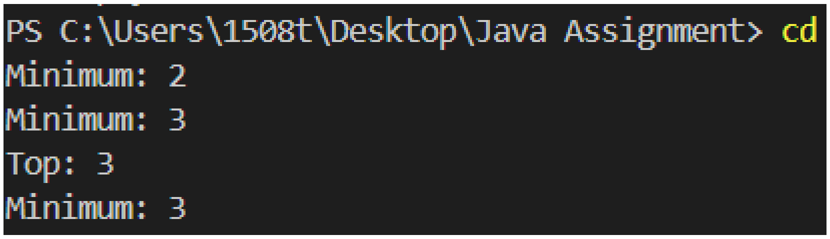
```
}
```



```
PS C:\Users\1508t\Desktop\Java Assignment>
Remove Front: 5
Peek Back: 20
Remove Back: 20
Is Empty: false
Remove Front: 10
Is Empty: true
```

Implement Min Stack using Two Stacks

```
import java.util.Stack;

public class Tutorial {
    Stack<Integer> mainStack = new Stack<>();
    Stack<Integer> minStack = new Stack<>();

    public void push(int value) {
        mainStack.push(value);
        if (minStack.isEmpty() || value <= minStack.peek()) {
            minStack.push(value);
        }
    }

    public int pop() {
        if (mainStack.isEmpty()) {
            throw new RuntimeException("Stack is empty");
        }
        int value = mainStack.pop();
        if (value == minStack.peek()) {
            minStack.pop();
        }
        return value;
    }

    public int getMin() {
        if (minStack.isEmpty()) {
            throw new RuntimeException("Stack is empty");
        }
        return minStack.peek();
    }

    public int top() {
        if (mainStack.isEmpty()) {
            throw new RuntimeException("Stack is empty");
        }
        return mainStack.peek();
    }

    public static void main(String[] args) {
        Tutorial minStack = new Tutorial();

        minStack.push(5);
        minStack.push(3);
        minStack.push(7);
        minStack.push(2);

        System.out.println("Minimum: " + minStack.getMin()); // Output: 2
        minStack.pop();
        System.out.println("Minimum: " + minStack.getMin()); // Output: 3
        minStack.pop();
        System.out.println("Top: " + minStack.top());         // Output: 3
        System.out.println("Minimum: " + minStack.getMin()); // Output: 3
    }
```



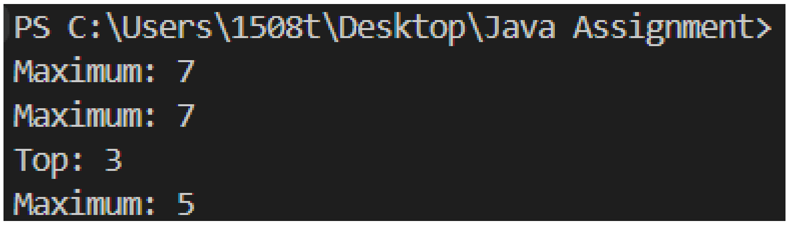
```
PS C:\Users\1508t\Desktop\Java Assignment> cd
Minimum: 2
Minimum: 3
Top: 3
Minimum: 3
```

```
}
```

.Implement Max Stack using Two Stacks

```
import java.util.Stack;

public class Tutorial {
    Stack<Integer> mainStack = new Stack<>();
    Stack<Integer> maxStack = new Stack<>();

    public void push(int value) {
        mainStack.push(value);
        if (maxStack.isEmpty() || value >= maxStack.peek()) {
            maxStack.push(value);
        }
    }

    public int pop() {
        if (mainStack.isEmpty()) {
            throw new RuntimeException("Stack is empty");
        }
        int value = mainStack.pop();
        if (value == maxStack.peek()) {
            maxStack.pop();
        }
    return value;
}

public int getMax() {
    if (maxStack.isEmpty()) {
        throw new RuntimeException("Stack is empty");
    }
    return maxStack.peek();
}

public int top() {
    if (mainStack.isEmpty()) {
        throw new RuntimeException("Stack is empty");
    }
    return mainStack.peek();
}

public static void main(String[] args) {
    Tutorial maxStack = new Tutorial();

    maxStack.push(5);
    maxStack.push(3);
    maxStack.push(7);
    maxStack.push(2);

    System.out.println("Maximum: " + maxStack.getMax()); // Output: 7
    maxStack.pop();
    System.out.println("Maximum: " + maxStack.getMax()); // Output: 7
    maxStack.pop();
    System.out.println("Top: " + maxStack.top());        // Output: 3
```

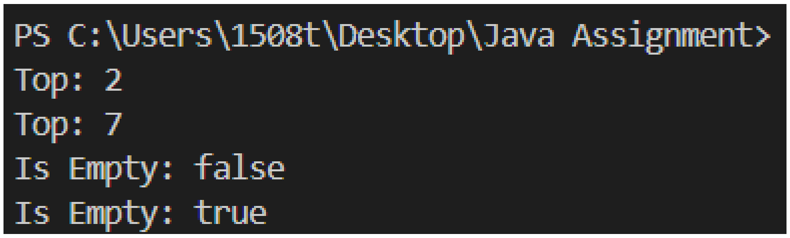
```
        System.out.println("Maximum: " + maxStack.getMax()); // Output: 5
    }
}
```



```
PS C:\Users\1508t\Desktop\Java Assignment>
Maximum: 7
Maximum: 7
Top: 3
Maximum: 5
```

Implement Stack using Queue

```
import java.util.LinkedList;
import java.util.Queue;

public class Tutorial {
    Queue<Integer> queue1 = new LinkedList<>();
    Queue<Integer> queue2 = new LinkedList<>();

    public void push(int value) {
        queue2.add(value);
        while (!queue1.isEmpty()) {
            queue2.add(queue1.remove());
        }
        Queue<Integer> temp = queue1;
        queue1 = queue2;
        queue2 = temp;
    }

    public int pop() {
        if (queue1.isEmpty()) {
            throw new RuntimeException("Stack is empty");
        }
    return queue1.remove();
}

public int top() {
    if (queue1.isEmpty()) {
        throw new RuntimeException("Stack is empty");
    }
    return queue1.peek();
}

public boolean isEmpty() {
    return queue1.isEmpty();
}

public static void main(String[] args) {
    Tutorial stack = new Tutorial();

    stack.push(5);
    stack.push(3);
    stack.push(7);
    stack.push(2);

    System.out.println("Top: " + stack.top()); // Output: 2
    stack.pop();
    System.out.println("Top: " + stack.top()); // Output: 7
    stack.pop();
    System.out.println("Is Empty: " + stack.isEmpty()); // Output: false
    stack.pop();
    stack.pop();
    System.out.println("Is Empty: " + stack.isEmpty()); // Output: true
    }
```



```
PS C:\Users\1508t\Desktop\Java Assignment>
Top: 2
Top: 7
Is Empty: false
Is Empty: true
```

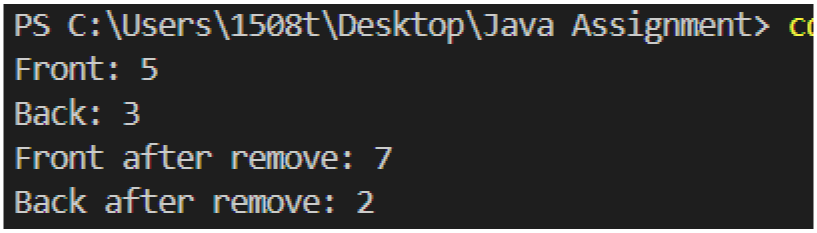
```
}
```

Implement Deque using Queue

```
import java.util.LinkedList;
import java.util.Queue;

public class Tutorial {
    Queue<Integer> frontQueue = new LinkedList<>();
    Queue<Integer> backQueue = new LinkedList<>();

    public void addFront(int value) {
        frontQueue.add(value);
    }

    public void addBack(int value) {
        backQueue.add(value);
    }

    public int removeFront() {
        if (!frontQueue.isEmpty()) {
            return frontQueue.remove();
        } else if (!backQueue.isEmpty()) {
            while (backQueue.size() > 1) {
                frontQueue.add(backQueue.remove());
            }
            return backQueue.remove();
        }
        throw new RuntimeException("Deque is empty");
    }

    public int removeBack() {
        if (!backQueue.isEmpty()) {
            return backQueue.remove();
        } else if (!frontQueue.isEmpty()) {
            while (frontQueue.size() > 1) {
                backQueue.add(frontQueue.remove());
            }
            return frontQueue.remove();
        }
        throw new RuntimeException("Deque is empty");
    }

    public int peekFront() {
        if (!frontQueue.isEmpty()) {
            return frontQueue.peek();
        } else if (!backQueue.isEmpty()) {
            while (!backQueue.isEmpty()) {
                frontQueue.add(backQueue.remove());
            }
            return frontQueue.peek();
        }
        throw new RuntimeException("Deque is empty");
    }

    public int peekBack() {
```

```
    if (!backQueue.isEmpty()) {
        return backQueue.peek();
    } else if (!frontQueue.isEmpty()) {
        while (!frontQueue.isEmpty()) {
            backQueue.add(frontQueue.remove());
        }
        return backQueue.peek();
    }
    throw new RuntimeException("Deque is empty");
}

public static void main(String[] args) {
    Tutorial deque = new Tutorial();

    deque.addFront(5);
    deque.addBack(3);
    deque.addFront(7);
    deque.addBack(2);

    System.out.println("Front: " + deque.peekFront()); // Output: 7
    System.out.println("Back: " + deque.peekBack());   // Output: 2

    deque.removeFront();
    System.out.println("Front after remove: " + deque.peekFront()); // Output: 5

    deque.removeBack();
    System.out.println("Back after remove: " + deque.peekBack());   // Output: 3
}
```

```
}}
```

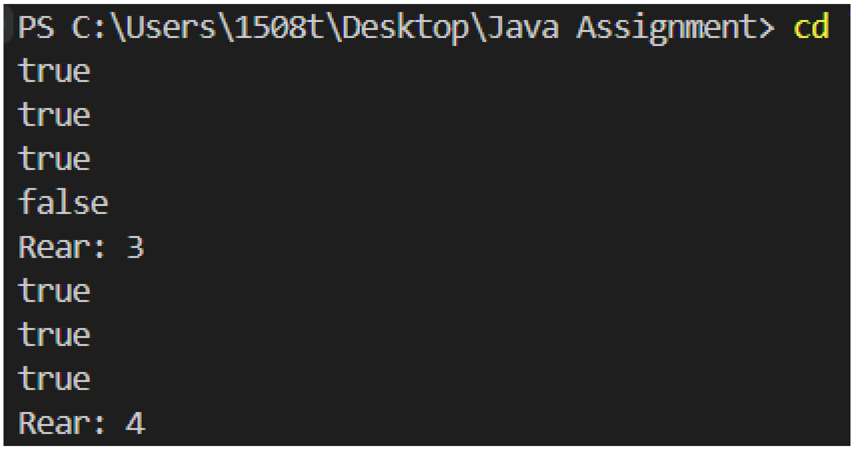


```
PS C:\Users\1508t\Desktop\Java Assignment> co
Front: 5
Back: 3
Front after remove: 7
Back after remove: 2
```

Implement Circular Queue using Queue

```
import java.util.LinkedList;
import java.util.Queue;

public class Tutorial {
    private int[] queue;
    private int front;
    private int rear;
    private int size;
    private int capacity;

    public Tutorial(int k) {
        capacity = k;
        queue = new int[k];
        front = -1;
        rear = -1;
        size = 0;
    }

    public boolean enQueue(int value) {
        if (isFull()) {
            return false;
        }
    if (isEmpty()) {
        front = 0;
    }
    rear = (rear + 1) % capacity;
    queue[rear] = value;
    size++;
    return true;
}

public boolean deQueue() {
    if (isEmpty()) {
        return false;
    }
    if (front == rear) {
        front = -1;
        rear = -1;
    } else {
        front = (front + 1) % capacity;
    }
    size--;
    return true;
}

public int Front() {
    if (isEmpty()) {
        return -1;
    }
    return queue[front];
}
```

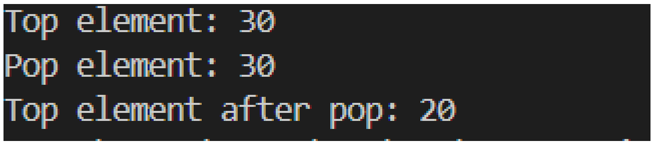
```
    public int Rear() {
        if (isEmpty()) {
            return -1;
        }
        return queue[rear];
    }

    public boolean isEmpty() {
        return size == 0;
    }

    public boolean isFull() {
        return size == capacity;
    }

    public static void main(String[] args) {
        Tutorial circularQueue = new Tutorial(3);
        System.out.println(circularQueue.enQueue(1)); // Output: true
        System.out.println(circularQueue.enQueue(2)); // Output: true
        System.out.println(circularQueue.enQueue(3)); // Output: true
        System.out.println(circularQueue.enQueue(4)); // Output: false (Queue is full)

        System.out.println("Rear: " + circularQueue.Rear()); // Output: 3
        System.out.println(circularQueue.isFull());          // Output: true

        System.out.println(circularQueue.deQueue());         // Output: true
        System.out.println(circularQueue.enQueue(4));        // Output: true
        System.out.println("Rear: " + circularQueue.Rear()); // Output: 4
    }
}
```



```
PS C:\Users\1508t\Desktop\Java Assignment> cd
true
true
true
false
Rear: 3
true
true
true
Rear: 4
```

Implement Stack using an Array

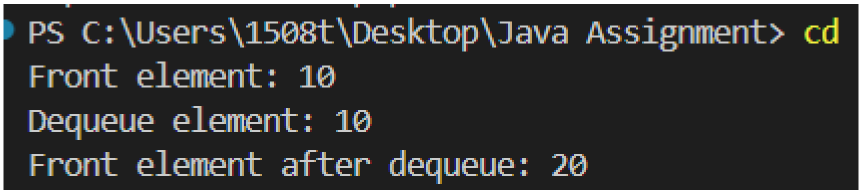
```
public class Tutorial {
    private int[] stack;
    private int top;
    private int capacity;

    public Tutorial(int size) {
        stack = new int[size];
        top = -1;
        capacity = size;
    }

    public boolean push(int value) {
        if (isFull()) {
            return false;
        }
        stack[++top] = value;
        return true;
    }

    public int pop() {
        if (isEmpty()) {
            throw new RuntimeException("Stack is empty");
        }
        return stack[top--];
    }

    public int peek() {
        if (isEmpty()) {
            throw new RuntimeException("Stack is empty");
        }
        return stack[top];
    }

    public boolean isEmpty() {
        return top == -1;
    }

    public boolean isFull() {
        return top == capacity - 1;
    }

    public static void main(String[] args) {
        Tutorial stack = new Tutorial(5);
        stack.push(10);
        stack.push(20);
        stack.push(30);

        System.out.println("Top element: " + stack.peek()); // Output: 30
        System.out.println("Pop element: " + stack.pop());  // Output: 30
        System.out.println("Top element after pop: " + stack.peek()); // Output: 20
    }
}
```



```
Top element: 30
Pop element: 30
Top element after pop: 20
```

Implement Queue using an Array

```
public class Tutorial {
    private int[] queue;
    private int front;
    private int rear;
    private int capacity;
    private int size;

    public Tutorial(int capacity) {
        this.capacity = capacity;
        this.queue = new int[capacity];
        this.front = 0;
        this.rear = -1;
        this.size = 0;
    }

    public boolean enqueue(int value) {
        if (isFull()) {
            return false;
        }
        rear = (rear + 1) % capacity;
        queue[rear] = value;
        size++;
        return true;
    }

    public int dequeue() {
        if (isEmpty()) {
            throw new RuntimeException("Queue is empty");
        }
        int value = queue[front];
        front = (front + 1) % capacity;
        size--;
        return value;
    }

    public int front() {
        if (isEmpty()) {
            throw new RuntimeException("Queue is empty");
        }
        return queue[front];
    }

    public boolean isEmpty() {
        return size == 0;
    }

    public boolean isFull() {
        return size == capacity;
    }

    public static void main(String[] args) {
        Tutorial queue = new Tutorial(5);
        queue.enqueue(10);
        queue.enqueue(20);
        queue.enqueue(30);

        System.out.println("Front element: " + queue.front()); // Output: 10
        System.out.println("Dequeue element: " + queue.dequeue()); // Output: 10
        System.out.println("Front element after dequeue: " + queue.front()); // Output: 20
    }
```



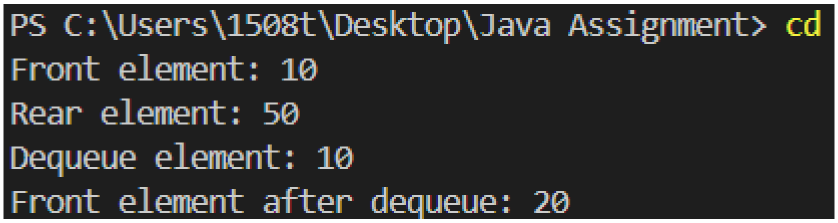
```
PS C:\Users\1508t\Desktop\Java Assignment> cd
Front element: 10
Dequeue element: 10
Front element after dequeue: 20
```

```
}
```

Implement Circular Queue using an Array

```
public class Tutorial {
    private int[] queue;
    private int front;
    private int rear;
    private int capacity;
    private int size;

    public Tutorial(int capacity) {
        this.capacity = capacity;
        this.queue = new int[capacity];
        this.front = -1;
        this.rear = -1;
        this.size = 0;
    }

    public boolean enqueue(int value) {
        if (isFull()) {
            return false;
        }
        if (isEmpty()) {
            front = 0;
        }
        rear = (rear + 1) % capacity;
    queue[rear] = value;
    size++;
    return true;
}

public int dequeue() {
    if (isEmpty()) {
        throw new RuntimeException("Queue is empty");
    }
    int value = queue[front];
    if (front == rear) {
        front = -1;
        rear = -1;
    } else {
        front = (front + 1) % capacity;
    }
    size--;
    return value;
}

public int front() {
    if (isEmpty()) {
        throw new RuntimeException("Queue is empty");
    }
    return queue[front];
}

public int rear() {
    if (isEmpty()) {
        throw new RuntimeException("Queue is empty");
```

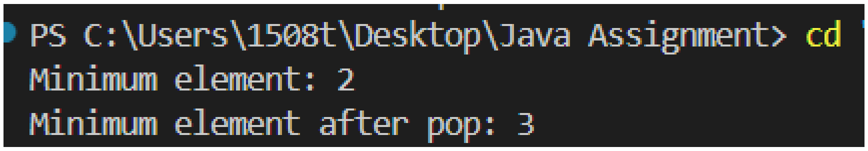
```java
        }
        return queue[rear];
    }

    public boolean isEmpty() {
        return size == 0;
    }

    public boolean isFull() {
        return size == capacity;
    }

    public static void main(String[] args) {
        Tutorial circularQueue = new Tutorial(5);
        circularQueue.enqueue(10);
        circularQueue.enqueue(20);
        circularQueue.enqueue(30);
        circularQueue.enqueue(40);
        circularQueue.enqueue(50);

        System.out.println("Front element: " + circularQueue.front()); // Output: 10
        System.out.println("Rear element: " + circularQueue.rear());   // Output: 50

        System.out.println("Dequeue element: " + circularQueue.dequeue()); // Output: 10
        System.out.println("Front element after dequeue: " + circularQueue.front()); // Output: 20
    }
}
```



```
PS C:\Users\1508t\Desktop\Java Assignment> cd
Front element: 10
Rear element: 50
Dequeue element: 10
Front element after dequeue: 20
```

Implement Min Stack using Linked List

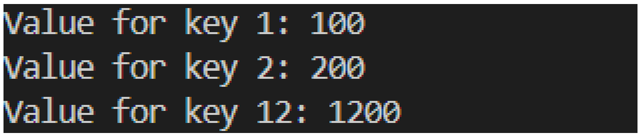
```
class Node {
    int value;
    int min;
    Node next;

    Node(int value, int min) {
        this.value = value;
        this.min = min;
        this.next = null;
    }
}

public class Tutorial {
    private Node head;

    public Tutorial() {
        head = null;
    }

    public void push(int value) {
        if (head == null) {
            head = new Node(value, value);
        } else {
            Node newNode = new Node(value, Math.min(value, head.min));
            newNode.next = head;
            head = newNode;
        }
    }

    public int pop() {
        if (head == null) {
            throw new RuntimeException("Stack is empty");
        }
        int value = head.value;
        head = head.next;
        return value;
    }

    public int top() {
        if (head == null) {
            throw new RuntimeException("Stack is empty");
        }
        return head.value;
    }

    public int getMin() {
        if (head == null) {
            throw new RuntimeException("Stack is empty");
        }
        return head.min;
    }

    public static void main(String[] args) {
        Tutorial minStack = new Tutorial();
        minStack.push(5);
        minStack.push(3);
        minStack.push(7);
        minStack.push(2);

        System.out.println("Minimum element: " + minStack.getMin()); // Output: 2
        minStack.pop();
        System.out.println("Minimum element after pop: " + minStack.getMin()); // Output: 3
    }
}
```



```
PS C:\Users\1508t\Desktop\Java Assignment> cd
Minimum element: 2
Minimum element after pop: 3
```

Implement Hash Table using Linked List (Chaining Method)

```
import java.util.LinkedList;
import java.util.List;

class HashNode {
    int key;
    int value;

    HashNode(int key, int value) {
        this.key = key;
        this.value = value;
    }
}

public class Tutorial {
    private List<HashNode>[] table;
private int capacity;

public Tutorial(int capacity) {
    this.capacity = capacity;
    table = new LinkedList[capacity];
    for (int i = 0; i < capacity; i++) {
        table[i] = new LinkedList<>();
    }
}

private int hash(int key) {
    return key % capacity;
}

public void put(int key, int value) {
    int index = hash(key);
    for (HashNode node : table[index]) {
        if (node.key == key) {
            node.value = value;
            return;
        }
    }
    table[index].add(new HashNode(key, value));
}

public int get(int key) {
    int index = hash(key);
    for (HashNode node : table[index]) {
        if (node.key == key) {
            return node.value;
            }
        }
        throw new RuntimeException("Key not found");
    }

    public void remove(int key) {
        int index = hash(key);
        table[index].removeIf(node -> node.key == key);
    }

    public static void main(String[] args) {
        Tutorial hashTable = new Tutorial(10);
        hashTable.put(1, 100);
        hashTable.put(2, 200);
        hashTable.put(12, 1200);

        System.out.println("Value for key 1: " + hashTable.get(1)); // Output: 100
        System.out.println("Value for key 2: " + hashTable.get(2)); // Output: 200
        System.out.println("Value for key 12: " + hashTable.get(12)); // Output: 1200

        hashTable.remove(2);
        System.out.println("Value for key 2 after removal: " + (hashTable.get(2))); // Throws exception
    }
}
```

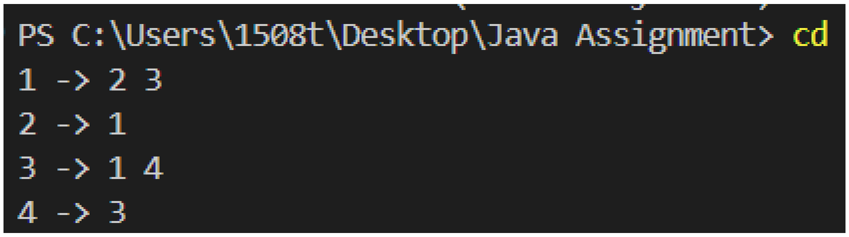


```
Value for key 1: 100
Value for key 2: 200
Value for key 12: 1200
```

Implement Graph using Linked List

```
import java.util.LinkedList;
import java.util.List;

class GraphNode {
    int value;
    List<GraphNode> neighbors;

    GraphNode(int value) {
        this.value = value;
        this.neighbors = new LinkedList<>();
    }
}

public class Tutorial {
    private List<GraphNode> graph;

    public Tutorial() {
        graph = new LinkedList<>();
    }

    public void addNode(int value) {
        graph.add(new GraphNode(value));
    }

    public void addEdge(int src, int dest) {
        GraphNode srcNode = findNode(src);
        GraphNode destNode = findNode(dest);

        if (srcNode != null && destNode != null) {
            srcNode.neighbors.add(destNode);
            destNode.neighbors.add(srcNode); // Undirected graph
        }
    }

    public void displayGraph() {
        for (GraphNode node : graph) {
            System.out.print(node.value + " -> ");
            for (GraphNode neighbor : node.neighbors) {
                System.out.print(neighbor.value + " ");
            }
            System.out.println();
        }
    }

    private GraphNode findNode(int value) {
        for (GraphNode node : graph) {
            if (node.value == value) {
                return node;
            }
        }
        return null;
    }

    public static void main(String[] args) {
        Tutorial graph = new Tutorial();
        graph.addNode(1);
        graph.addNode(2);
        graph.addNode(3);
        graph.addNode(4);
```

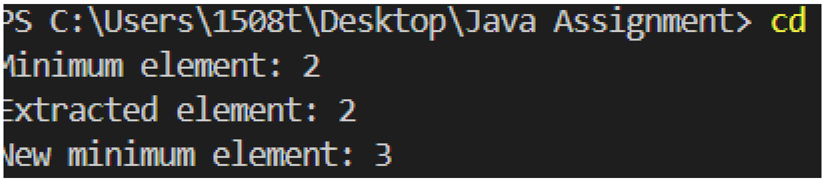
```
        graph.addEdge(1, 2);
        graph.addEdge(1, 3);
        graph.addEdge(3, 4);

        graph.displayGraph();
    }
}
```



```
PS C:\Users\1508t\Desktop\Java Assignment> cd
1 -> 2 3
2 -> 1
3 -> 1 4
4 -> 3
```

Implement Priority Queue using Heap

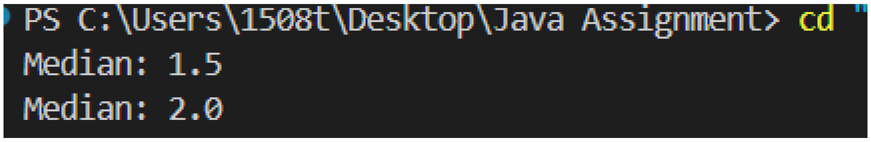
```
import java.util.PriorityQueue;

public class Tutorial {
    private PriorityQueue<Integer> heap;

    public Tutorial() {
        heap = new PriorityQueue<>();
    }

    public void insert(int value) {
        heap.offer(value);
    }

    public int extractMin() {
        if (heap.isEmpty()) {
            throw new RuntimeException("Priority Queue is empty");
        }
        return heap.poll();
    }

    public int getMin() {
        if (heap.isEmpty()) {
            throw new RuntimeException("Priority Queue is empty");
        }
        return heap.peek();
    }

    public boolean isEmpty() {
        return heap.isEmpty();
    }

    public static void main(String[] args) {
        Tutorial pq = new Tutorial();
        pq.insert(5);
        pq.insert(3);
        pq.insert(7);
        pq.insert(2);

        System.out.println("Minimum element: " + pq.getMin()); // Output: 2
        System.out.println("Extracted element: " + pq.extractMin()); // Output: 2
        System.out.println("New minimum element: " + pq.getMin()); // Output: 3
    }
}
```



```
PS C:\Users\1508t\Desktop\Java Assignment> cd
Minimum element: 2
Extracted element: 2
New minimum element: 3
```

Implement Median Finder using Two Heaps (Min Heap + Max Heap)

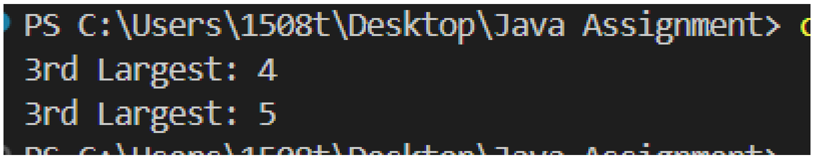
```java
import java.util.Collections;
import java.util.PriorityQueue;

public class Tutorial {
    private PriorityQueue<Integer> maxHeap;
    private PriorityQueue<Integer> minHeap;

    public Tutorial() {
        maxHeap = new PriorityQueue<>(Collections.reverseOrder());
        minHeap = new PriorityQueue<>();
    }

    public void addNum(int num) {
        if (maxHeap.isEmpty() || num <= maxHeap.peek()) {
            maxHeap.offer(num);
        } else {
            minHeap.offer(num);
        }

        if (maxHeap.size() > minHeap.size() + 1) {
            minHeap.offer(maxHeap.poll());
        } else if (minHeap.size() > maxHeap.size()) {
            maxHeap.offer(minHeap.poll());
        }
    }

    public double findMedian() {
        if (maxHeap.size() == minHeap.size()) {
            return (maxHeap.peek() + minHeap.peek()) / 2.0;
        } else {
            return maxHeap.peek();
        }
    }

    public static void main(String[] args) {
        Tutorial medianFinder = new Tutorial();
        medianFinder.addNum(1);
        medianFinder.addNum(2);
        System.out.println("Median: " + medianFinder.findMedian()); // Output: 1.5
        medianFinder.addNum(3);
        System.out.println("Median: " + medianFinder.findMedian()); // Output: 2
    }
}
```



```
PS C:\Users\1508t\Desktop\Java Assignment> cd "
Median: 1.5
Median: 2.0
```

Implement Kth Largest Element Finder using Heap

```
import java.util.PriorityQueue;

public class Tutorial {
    private PriorityQueue<Integer> minHeap;
    private int k;

    public Tutorial(int k) {
        this.k = k;
        minHeap = new PriorityQueue<>();
    }

    public void add(int num) {
        if (minHeap.size() < k) {
            minHeap.offer(num);
        } else if (num > minHeap.peek()) {
            minHeap.poll();
            minHeap.offer(num);
        }
    }

    public int getKthLargest() {
        if (minHeap.size() < k) {
            throw new IllegalStateException("Less than " + k + " elements present.");
        }
        return minHeap.peek();
    }

    public static void main(String[] args) {
        Tutorial kthLargestFinder = new Tutorial(3);
        kthLargestFinder.add(4);
        kthLargestFinder.add(5);
        kthLargestFinder.add(8);
        kthLargestFinder.add(2);
        System.out.println("3rd Largest: " + kthLargestFinder.getKthLargest()); // Output: 4
        kthLargestFinder.add(10);
        System.out.println("3rd Largest: " + kthLargestFinder.getKthLargest()); // Output: 5
    }
}
```



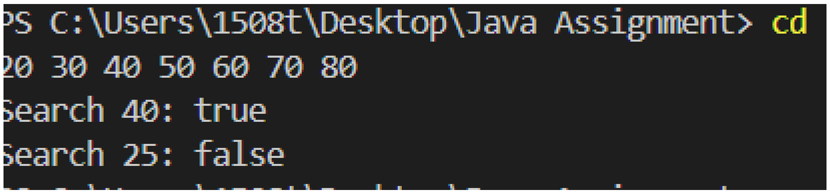
```
PS C:\Users\1508t\Desktop\Java Assignment> c
3rd Largest: 4
3rd Largest: 5
PS C:\Users\1508t\Desktop\Java Assignment>
```

Implement BST using Linked List

```
class TreeNode {
    int value;
    TreeNode left, right;

    TreeNode(int value) {
        this.value = value;
        left = right = null;
    }
}

public class Tutorial {
    private TreeNode root;

    public void insert(int value) {
        root = insertRec(root, value);
    }

    private TreeNode insertRec(TreeNode root, int value) {
        if (root == null) {
            return new TreeNode(value);
        }
        if (value < root.value) {
            root.left = insertRec(root.left, value);
        } else if (value > root.value) {
            root.right = insertRec(root.right, value);
        }
        return root;
    }

    public boolean search(int value) {
        return searchRec(root, value);
```

```
}

private boolean searchRec(TreeNode root, int value) {
    if (root == null) {
        return false;
    }
    if (root.value == value) {
        return true;
    }
    return value < root.value ? searchRec(root.left, value) : searchRec(root.right, value);
}

public void inorderTraversal() {
    inorderRec(root);
    System.out.println();
}

private void inorderRec(TreeNode root) {
    if (root != null) {
        inorderRec(root.left);
        System.out.print(root.value + " ");
        inorderRec(root.right);
    }
}

public static void main(String[] args) {
    Tutorial bst = new Tutorial();
    bst.insert(50);
    bst.insert(30);
    bst.insert(70);
```

```
        bst.insert(20);
        bst.insert(40);
        bst.insert(60);
        bst.insert(80);

        bst.inorderTraversal();

        System.out.println("Search 40: " + bst.search(40)); // Output: true
        System.out.println("Search 25: " + bst.search(25)); // Output: false
    }
}
```



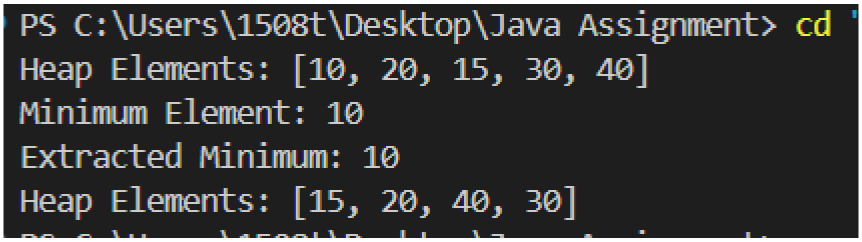
```
PS C:\Users\1508t\Desktop\Java Assignment> cd
20 30 40 50 60 70 80
Search 40: true
Search 25: false
```

Implement Heap using BST

```
import java.util.PriorityQueue;

class TreeNode {
    int value;
    TreeNode left, right;

    TreeNode(int value) {
        this.value = value;
        left = right = null;
    }
}

public class Tutorial {
    private PriorityQueue<Integer> heap;
```

```
public Tutorial() {
    heap = new PriorityQueue<>();
}

public void insert(int value) {
    heap.offer(value);
}

public int extractMin() {
    if (heap.isEmpty()) {
        throw new IllegalStateException("Heap is empty");
    }
    return heap.poll();
}

public int getMin() {
    if (heap.isEmpty()) {
        throw new IllegalStateException("Heap is empty");
    }
    return heap.peek();
}

public void displayHeap() {
    System.out.println("Heap Elements: " + heap);
}

public static void main(String[] args) {
    Tutorial heap = new Tutorial();
    heap.insert(10);
```

```
        heap.insert(20);
        heap.insert(15);
        heap.insert(30);
        heap.insert(40);

        heap.displayHeap();

        System.out.println("Minimum Element: " + heap.getMin()); // Output: 10
        System.out.println("Extracted Minimum: " + heap.extractMin()); // Output: 10
        heap.displayHeap();
    }
}
```



```
PS C:\Users\1508t\Desktop\Java Assignment> cd
Heap Elements: [10, 20, 15, 30, 40]
Minimum Element: 10
Extracted Minimum: 10
Heap Elements: [15, 20, 40, 30]
```