3.1) Создание мультисписка на Java

import java.util.\*;

public class Multilist {

private List<List<Object>> sublists;

public Multilist() {

this.sublists = new ArrayList<>();

}

public void addSublist(List<Object> items) {

sublists.add(new ArrayList<>(items));

}

public List<Object> flatten() {

List<Object> result = new ArrayList<>();

for (List<Object> sublist : sublists) {

result.addAll(sublist);

}

return result;

}

public List<Object> getSublist(int index) {

return new ArrayList<>(sublists.get(index));

}

}

// Использование

Multilist ml = new Multilist();

ml.addSublist(Arrays.asList(1, 2, 3));

ml.addSublist(Arrays.asList("a", "b"));

3.2) Создание очереди на Java  
  
import java.util.LinkedList;

public class Queue<T> {

private LinkedList<T> list;

public Queue() {

this.list = new LinkedList<>();

}

public void enqueue(T item) {

list.addLast(item);

}

public T dequeue() {

if (is\_empty()) {

throw new RuntimeException("Queue is empty");

}

return list.removeFirst();

}

public boolean is\_empty() {

return list.isEmpty();

}

public int size() {

return list.size();

}

}

3.3) Создание Дек на Java

import java.util.ArrayDeque;

public class Deque<T> {

private ArrayDeque<T> deque;

public Deque() {

this.deque = new ArrayDeque<>();

}

public void add\_front(T item) {

deque.addFirst(item);

}

public void add\_rear(T item) {

deque.addLast(item);

}

public T remove\_front() {

if (is\_empty()) {

throw new RuntimeException("Deque is empty");

}

return deque.removeFirst();

}

public T remove\_rear() {

if (is\_empty()) {

throw new RuntimeException("Deque is empty");

}

return deque.removeLast();

}

public boolean is\_empty() {

return deque.isEmpty();

}

public int size() {

return deque.size();

}

}

3.4) Создание приоритетной очереди на Java

import java.util.\*;

public class PriorityQueue<T> {

private PriorityQueue<Item<T>> heap;

private int indexCounter;

private static class Item<T> implements Comparable<Item<T>> {

T value;

int priority;

int index;

Item(T value, int priority, int index) {

this.value = value;

this.priority = priority;

this.index = index;

}

@Override

public int compareTo(Item<T> other) {

if (this.priority == other.priority) {

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

}

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

}

}

public PriorityQueue() {

this.heap = new PriorityQueue<>();

this.indexCounter = 0;

}

public void push(T item, int priority) {

heap.offer(new Item<>(item, priority, indexCounter++));

}

public T pop() {

if (is\_empty()) {

throw new RuntimeException("Priority queue is empty");

}

return heap.poll().value;

}

public boolean is\_empty() {

return heap.isEmpty();

}

public int size() {

return heap.size();

}

}