**Hashtable:**

**class HashtableDivision<T, V> {**

array table[10]

function put(key, value) {

index = hashDivision(key)

if (table[index] == null) {

table[index] = new ArrayList<Passenger<T, V>>()

}

for each passenger in table[index] {

if (passenger.key.equals(key)) {

passenger.value = value

return

}

}

table[index].add(new Passenger<T, V>(key, value))

**}**

**function get(key) {**

index = hashDivision(key)

if (table[index] != null) {

for each passenger in table[index] {

if (passenger.key.equals(key)) {

return passenger.value

}

}

}

return null

}

**}**

**class Passenger<T, V> {**

T key

V value

function Passenger(key, value) {

this.key = key

this.value = value

}

**}**

**function hashDivision(key) {**

hashCode = key.hashCode()

index = hashCode % 10

if (index < 0) {

index = index + 10

}

return index

**}**

**Priority Queue:**

**public void add(T element) {**

if (size == heap.length - 1) {

            resize();

        }

        heap[++size] = element;

        int pos = size;

        while (pos > 1 && heap[pos].compareTo(heap[pos/2]) > 0) {

            swap(pos, pos/2);

            pos /= 2;

        }

**}**

**public T peek() {**

if (size == 0) {

            return null;

        }

        return heap[1];

**}**

**public T poll() {**

    if (size == 0) {

            return null;

        }

        T result = heap[1];

        heap[1] = heap[size--];

        heapify(1);

        return result;

**}**

**private void swap(int i, int j) {**

        T temp = heap[i];

        heap[i] = heap[j];

        heap[j] = temp;

**}**

**private void resize() {**

        T[] newHeap = (T[]) new Comparable[heap.length \* 2];

        System.arraycopy(heap, 1, newHeap, 1, size);

        heap = newHeap;

**}**

**public int size() {**

  return size;

**}**

**public boolean isEmpty() {**

return size == 0;

**}**

**public static <T extends Customer> int compareCustomers(T c1, T c2, Airplane airplane) {**

 if (hashDivision(c1.getKey()) == hashDivision(c2.getKey())) {

        if (hashDivision(c1.getKey()) > airplane.getFirstClass()) {

            if (c1.getSpecialAtention() != c2.getSpecialAtention()){

switch (c1.getSpecialAtention()) {

                    case true:

                        return 1;

                    case false:

                        return -1;

}

} else if (c1.getAge()>=60 && c2.getAge()>=60 || c1.getAge()>=60 && c2.getAge()<60 || c1.getAge()<60 && c2.getAge()>=60) {

                return c1.getAge() - c2.getAge();

} else if (c1.getMillas() != c2.getMillas()) {

return c1.getMillas() - c2.getMillas();

                }

            } else if (c1.getArrivalTime().compareTo(c2.getArrivalTime()) != 0) {

                return c1.getArrivalTime().compareTo(c2.getArrivalTime());

            }

        } else {

            if (c1.getArrivalTime().compareTo(c2.getArrivalTime()) != 0) {

                return c1.getArrivalTime().compareTo(c2.getArrivalTime());

            }

        }

    } else {

        return hashDivision(c1.getKey()) - hashDivision(c2.getKey());

    }

    return 0;

**}**

**Json:**

**import java.io.FileReader;**

**import java.io.FileWriter;**

**import org.json.JSONObject;**

**public class JsonUtils<T> {**

**public void saveDataToJsonFile(T data, String filename) {**

JSONObject json = new JSONObject();

        json.put("data", data);

        try {

            FileWriter fileWriter = new FileWriter(filename);

            fileWriter.write(json.toString());

            fileWriter.close();

        } catch (Exception e) {

            e.printStackTrace();

        }

**}**

**public T loadDataFromJsonFile(String filename) {**

 try {

            FileReader fileReader = new FileReader(filename);

            StringBuilder fileContents = new StringBuilder();

            int character;

            while ((character = fileReader.read()) != -1) {

                fileContents.append((char) character);

            }

            fileReader.close();

            JSONObject json = new JSONObject(fileContents.toString());

            return (T) json.get("data");

        } catch (Exception e) {

            e.printStackTrace();

            return null;

        }

**}**

**}**