**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

**}**

**HeapSort:**

**public void sort(T[] arr) {**

  int n = arr.length;

        for (int i = n / 2 - 1; i >= 0; i--) {

            heapify(arr, n, i);

        }

        for (int i = n - 1; i >= 0; i--) {

            T temp = arr[0];

            arr[0] = arr[i];

            arr[i] = temp;

            heapify(arr, i, 0);

        }

**}**

**private void heapify(T[] arr, int n, int i) {**

    int largest = i;

        int l = 2 \* i + 1;

        int r = 2 \* i + 2;

        if (l < n && arr[l].compareTo(arr[l], arr[largest], airplane) > 0) {

            largest = l;

        }

        if (r < n && arr[r].compareTo(arr[r], arr[largest], airplane) > 0) {

            largest = r;

        }

        if (largest != i) {

            T temp = arr[i];

            arr[i] = arr[largest];

            arr[largest] = temp;

            heapify(arr, n, largest);

        }

**}**

**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.getMillas() != c2.getMillas()) {

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

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

                switch (c1.getSpecialAtention()) {

                    case true:

                        return 1;

                    case false:

                        return -1;

                }

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

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

            } 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;

**}**