/\*\* Represnts a list of musical tracks. The list has a maximum capacity (int),

 \*  and an actual size (number of tracks in the list, an int). \*/

class PlayList {

    private Track[] tracks;  // Array of tracks (Track objects)

    private int maxSize;     // Maximum number of tracks in the array

    private int size;        // Actual number of tracks in the array

    /\*\* Constructs an empty play list with a maximum number of tracks. \*/

    public PlayList(int maxSize) {

        this.maxSize = maxSize;

        tracks = new Track[maxSize];

        size = 0;

    }

    /\*\* Returns the maximum size of this play list. \*/

    public int getMaxSize() {

        return maxSize;

    }

    /\*\* Returns the current number of tracks in this play list. \*/

    public int getSize() {

        return size;

    }

    /\*\* Method to get a track by index \*/

    public Track getTrack(int index) {

        if (index >= 0 && index < size) {

            return tracks[index];

        } else {

            return null;

        }

    }

    /\*\* Appends the given track to the end of this list.

     \*  If the list is full, does nothing and returns false.

     \*  Otherwise, appends the track and returns true. \*/

    public boolean add(Track track) {

        if (size == maxSize) {

            return false;

        }

        tracks[size] = track;

        size++;

        return true;

    }

    /\*\* Returns the data of this list, as a string. Each track appears in a separate line. \*/

    //// For an efficient implementation, use StringBuilder.

    public String toString() {

        StringBuilder sb = new StringBuilder();

        for (int i = 0; i < size; i++){

            sb.append(tracks[i].toString() + "\n");

        }

        return sb.toString();

    }

    /\*\* Removes the last track from this list. If the list is empty, does nothing. \*/

     public void removeLast() {

        remove(size);

    }

    /\*\* Returns the total duration (in seconds) of all the tracks in this list.\*/

    public int totalDuration() {

        int total = 0;

        for (int i = 0; i < size; i++) {

            total += tracks[i].getDuration();

        }

        return total;

    }

    /\*\* Returns the index of the track with the given title in this list.

     \*  If such a track is not found, returns -1. \*/

    public int indexOf(String title) {

        for (int i = 0; i < size; i++){

            if (title.equalsIgnoreCase(tracks[i].getTitle())) {

                return i;

            }

        }

        return -1;

    }

    /\*\* Inserts the given track in index i of this list. For example, if the list is

     \*  (t5, t3, t1), then just after add(1,t4) the list becomes (t5, t4, t3, t1).

     \*  If the list is the empty list (), then just after add(0,t3) it becomes (t3).

     \*  If i is negative or greater than the size of this list, or if the list

     \*  is full, does nothing and returns false. Otherwise, inserts the track and

     \*  returns true. \*/

    public boolean add(int i, Track track) {

        if (i < 0 || i > size || size == maxSize) {

            return false;

        }

        for (int j = size; j > i; j--) {

            tracks[j] = tracks[j - 1];

        }

        tracks[i] = track;

        size++;

        return true;

    }

    /\*\* Removes the track in the given index from this list.

     \*  If the list is empty, or the given index is negative or too big for this list,

     \*  does nothing and returns -1. \*/

    public int remove(int i) {

        if (i < 0 || i > size || size == 0) {

            return -1;

        }

        if (i != maxSize) {

            for (int j = i; j < size - 1;j++) {

                tracks[j] = tracks[j + 1];

            }

        }

        size--;

        return 0;

    }

    /\*\* Removes the first track that has the given title from this list.

     \*  If such a track is not found, or the list is empty, or the given index

     \*  is negative or too big for this list, does nothing. \*/

    public void remove(String title) {

       remove(indexOf(title));

    }

    /\*\* Removes the first track from this list. If the list is empty, does nothing. \*/

    public void removeFirst() {

        remove(0);

    }

    /\*\* Adds all the tracks in the other list to the end of this list.

     \*  If the total size of both lists is too large, does nothing. \*/

    //// An elegant and terribly inefficient implementation.

     public void add(PlayList other) {

        if (size + other.getSize() <= maxSize) {

            for (int i = 0; i < other.getSize(); i++) {

                add(other.getTrack(i));

            }

        }

    }

    /\*\* Returns the index in this list of the track that has the shortest duration,

     \*  starting the search in location start. For example, if the durations are

     \*  7, 1, 6, 7, 5, 8, 7, then min(2) returns 4, since this the index of the

     \*  minimum value (5) when starting the search from index 2.

     \*  If start is negative or greater than size - 1, returns -1.

     \*/

    private int minIndex(int start) {

        if (start < 0 || start > size - 1) {

            return -1;

        }

        int min = tracks[start].getDuration();

        int min\_index = start;

        for (int i = start + 1; i < size; i++) {

            if (tracks[i].getDuration() < min) {

                min = tracks[i].getDuration();

                min\_index = i;

            }

        }

        return min\_index;

    }

    /\*\* Returns the title of the shortest track in this list.

     \*  If the list is empty, returns null. \*/

    public String titleOfShortestTrack() {

        return tracks[minIndex(0)].getTitle();

    }

    /\*\* Sorts this list by increasing duration order: Tracks with shorter

     \*  durations will appear first. The sort is done in-place. In other words,

     \*  rather than returning a new, sorted playlist, the method sorts

     \*  the list on which it was called (this list). \*/

    public void sortedInPlace() {

        // Uses the selection sort algorithm,

        // calling the minIndex method in each iteration.

        for (int i = 0; i < size; i++) {

            Track tmpTrack = tracks[i];

            int min\_index = minIndex(i);

            tracks[i] = tracks[min\_index];

            tracks[min\_index] = tmpTrack;

        }

    }

}

/\*\* Represents a music track. A track has a title (String), an artist (String),

 \*  and a duration (int), in seconds. \*/

class Track {

    private String title;

    private String artist;

    private int duration;

    /\*\* Constructs a track from the given values. \*/

    public Track(String title, String artist, int duration) {

        this.title = title;

        this.artist = artist;

        this.duration = duration;

    }

    /\*\* Returns this track's data as "artist, title, minutes:seconds".

     \*  For example, "John Lennon, Imagine, 3:07" \*/

    public String toString() {

        return (getArtist() + ", " + getTitle() + ", " + formattedDuration(getDuration()));

    }

    /\*\* Returns this track's title. \*/

    public String getTitle() {

        return title;

    }

    /\*\* Returns this track's artist. \*/

    public String getArtist() {

        return artist;

    }

    /\*\* Returns this track's duration. \*/

    public int getDuration() {

        return duration;

    }

    /\*\* If this track's duration is shorter than the other track's duration

     \*  returns true; otherwise returns false. \*/

    public boolean isShorterThan(Track other) {

        return duration < other.duration;

    }

    // Returns a string that represents the totalSeconds as "minutes:seconds",

    // Where seconds is always two digits. For example, "3:17" or "12:05".

    private String formattedDuration(int totalSeconds) {

        int minutes = (totalSeconds / 60);

        int seconde = (totalSeconds % 60);

        if (seconde < 10){

            return (Integer.toString(minutes) + ":0"+  Integer.toString(seconde));

        }

        return (Integer.toString(minutes) + ":" + Integer.toString(seconde));

    }

}