```
/** 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(this.size == this.maxSize){
            return false;
    }

    this.tracks[this.size] = track;
    this.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() {

    String str = "";

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

        str += this.tracks[i].toString() + "\n";
    }

    return "\n" + str;
}

/** Removes the last track from this list. If the list is empty, does nothing. */
 public void removeLast() {

    if(this.size != 0){

        this.tracks[this.size] = null;
        this.size--;

    }

}

/** Returns the total duration (in seconds) of all the tracks in this list.*/
public int totalDuration() {

    int duration = 0;

    for (int i = 0; i < this.size; i++){
        duration += this.tracks[i].getDuration();

    }

    return duration;
}

/** 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) {

    String str = title.toLowerCase();

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

        if (((this.tracks[i].getTitle()).toLowerCase()).equals(str)){

            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) {

    int counter = 0;

    if (i == this.size){

        this.add(track);
        return true;
    } else if(i <= this.maxSize && i >= 0 && this.size != this.maxSize){

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

            counter++;
        }

        int r = counter;

        for (int l = counter-1; l >= 0; l--){

            this.tracks[i+r] = this.tracks[i+l];
            r--;

        }

            this.tracks[i] = track;
            this.size++;

            return true;
    }

    return false;
}

/** 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 void remove(int i) {

    int counter = 0;;

    if (this.size > 0 && this.size != this.maxSize && i < this.size && i >= 0){

        this.tracks[i] = null;


        for (int j = i+1; j < this.size; j++){
            counter++;
        }

        int r = 1;

        for (int l = 0; l < counter; l++){

            this.tracks[i+l] = this.tracks[i+r];
            r++;

        }


        this.size--;
    }

}

/** 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) {

    int i = this.indexOf(title);

    if(this.size > 0 && i != -1){

        this.remove(i);

    }

}

/** Removes the first track from this list. If the list is empty, does nothing. */
public void removeFirst() {

    this.tracks[0] = null;

    int r = 1;
    for (int l = 0; l < this.size-1; l++){

        this.tracks[l] = this.tracks[r];
        r++;

    }

    this.size--;

}

/** 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(other.size + this.size < this.maxSize){

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

            this.add(other.tracks[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 <= this.size-1){

        int[] arr = new int[this.size];
        for(int i = 0; i < arr.length; i++){

            arr[i] = this.tracks[i].getDuration();

        }

        int min = arr[start];
        int minIndex = start;

        for (int i = start; i < arr.length; i++){

            if (arr[i] < min){

                min = arr[i];
                minIndex = i;
            }

        }

        return minIndex;
    }

    return -1;
}

/** 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 < this.size; i++){

            Track temp = this.tracks[i];
            int minIndex = this.minIndex(i);
            tracks[i] = tracks[minIndex];
            tracks[minIndex] = temp;

        }

    }
}
```

```
/** 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() {
        //// Replace the following statement with code that returns
        //// the data of this track according to the method's documentation.
        return artist + ", " + title + ", " + duration;
    }

    /** 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) {

            String str = "";
            int minutes;
            int seconds;

            minutes = totalSeconds/60;
            seconds = totalSeconds % 60;

            if (seconds == 0){

                str = minutes + ":00";

            } else if (seconds < 10){

                str = minutes + ":0" + seconds;

            } else {

                str = minutes + ":" + seconds;

            }

        return str;
    }
}
```