HW08 Code

Playlist.java:

```
/** 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() {
        size = 0;
        for (int i = 0; i < tracks.length; i++) {
            if (tracks[i] != null) {
                size++;
            }
        }
        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) {
        for (int i = 0; i < tracks.length; i++) {
            if (tracks[i] == null) {
                tracks[i] = track;
                return true;
            }
        }

        return false;
    }

    /** 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 playlist = "";
        for (int i = 0; i < tracks.length; i++) {
            if (tracks[i] != null) {
                playlist = tracks[i].getArtist() + ", ";
                playlist = tracks[i].getTitle() + ", ";
                playlist = tracks[i].getDuration() + "\n";
            }
        }
        return playlist;
    }

    /** Removes the last track from this list. If the list is empty, does
nothing. */
     public void removeLast() {
        for (int i = 1; i <= tracks.length; i++) {
            if (tracks[tracks.length - i] != null) {
                tracks[tracks.length - i] = null;
```

```
                break;
            }
        }
    }

    /** Returns the total duration (in seconds) of all the tracks in this
list.*/
    public int totalDuration() {
        int totalDuration = 0;
        for (int i = 0; i < tracks.length; i++) {
            if (tracks[i] != null) {
                totalDuration = totalDuration + tracks[i].getDuration();
            }
        }
        return totalDuration;
    }

    /** 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 < tracks.length; i++) {
            if (tracks[i] != null && title.equals(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) {
```

```
        int space = 0;
        Track[] newTracks = new Track[maxSize];
        for (int j = 0; j < tracks.length; j++) {
            if (j == i) {
                if (tracks[j] == null) {
                    newTracks[j] = track;
                    return true;
                } else {
                    for (int k = j; k < tracks.length; k++) {
                        if (tracks[k] == null) {
                            space = 1;
                        }
                    }
                    if (space == 1) {
                        for (int k = 0; k < tracks.length - 1 -j; k++) {
                            newTracks[k+1] = tracks[k];
                        }
                        newTracks[j] = track;

                        for (int z = 0; z < tracks.length; z++) {
                            tracks[z] = newTracks[z];
                        }
                        size = getSize();
                        return true;
                    }
                }
            } else {
                newTracks[j] = tracks[j];
            }
        }
        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) {
        for (int j = 0; j < tracks.length; j++) {
            if (j == i) {
```

```
                tracks[j] = null;
                for (int z = j; z < tracks.length-1; z++) {
                    tracks[z] = tracks[z+1];
                }
            }

        }
    }

    /** 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) {
        for (int i = 0; i < tracks.length; i++) {
            if (title.equals(tracks[i].getTitle())) {
                tracks[i] = null;
            }
            for (int z = i; z < tracks.length-1; z++) {
                tracks[z] = tracks[z+1];
            }
        }

    }

    /** Removes the first track from this list. If the list is empty, does
nothing. */
    public void removeFirst() {
        if (tracks[0] != null) {
            tracks[0] = null;
            for (int z = 0; z < tracks.length-1; z++) {
                tracks[z] = tracks[z+1];
            }
        }
    }

    /** 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) {
```

```
        int amountOfSpace = 0;
        for (int i = 0; i < tracks.length; i++) {
            if (tracks[i] == null) {
                amountOfSpace++;
            }
        }

        if (amountOfSpace >= other.tracks.length){
            for (int j = 0; j < other.tracks.length; j++) {
                tracks[tracks.length-amountOfSpace+j] = other.tracks[j];
            }
        }
    }

    /** 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) {
        int min = 10000000;
        int minIndex = 0;
        for (int i = start; i < tracks.length; i++) {
            if (tracks[i] != null) {
                if (tracks[i].getDuration() < min) {
                    min = tracks[i].getDuration();
                    minIndex = i;
                }
            }
        }
        return minIndex;
    }

    /** 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.
        Track[] newTracks = new Track[maxSize];
        for (int i = 0; i < newTracks.length; i++) {
            if (tracks == null) {
                break;
            }
            newTracks[i] = tracks[minIndex(0)];
            remove(minIndex(0));
        }
        for (int i = 0; i < tracks.length; i++) {
            tracks[i] = newTracks[i];
            size = getSize();
        }
    }
}
```