Eric Roberts
CS 106A

Handout #60A
May 30, 2012

Solution to Section #8

```
/*
 * File: FTDataBase.java
 * ---------------------
 * This file implements the data structure for the entire FlyTunes
 * database.
 */

import acm.util.*;
import java.io.*;
import java.util.*;

/**
 * This class represents the entire FlyTunes database.
 */
public class FTDataBase {

/**
 * Creates a new database by reading the contents of the specified
 * file.  Errors are reported by throwing <code>ErrorException</code>.
 *
 * @param filename The name of the data file
 */
   public FTDataBase(String filename) {
      try {
         BufferedReader rd = new BufferedReader(new FileReader(filename));
         albumMap = new TreeMap<String,FTAlbum>();
         artistIndex = new TreeMap<String,ArrayList<FTTrack>>();
         while (true) {
            FTAlbum album = FTAlbum.readAlbum(rd);
            if (album == null) break;
            albumMap.put(album.getTitle(), album);
            updateIndexData(album);
         }
      } catch (IOException ex) {
         throw new ErrorException(ex);
      }
   }

/**
 * Returns an array of all the albums in the database.
 *
 * @return An array of all the albums in the database
 */
   public FTAlbum[] getAlbums() {
      return albumMap.values().toArray(new FTAlbum[albumMap.size()]);
   }

/**
 * Gets an album by title.  If no album exists with that
 * title, this method returns <code>null</code>.
 *
 * @param title The title of the album
 * @return The album with that title, or <code>null</code>
 */
   public FTAlbum getAlbumByTitle(String title) {
      return albumMap.get(title);
   }
```

☞

```
/**
 * Returns an array of all the tracks recorded by a particular
 * artist.
 *
 * @param artist The name of the artist
 * @return An array of the tracks recorded by that artist
 */
   public FTTrack[] getTracksByArtist(String artist) {
      ArrayList<FTTrack> trackList = artistIndex.get(artist);
      if (trackList == null) trackList = new ArrayList<FTTrack>();
      return trackList.toArray(new FTTrack[trackList.size()]);
   }

/**
 * Updates the global artist index by adding data for the
 * specific track.
 */
   private void updateIndexData(FTAlbum album) {
      for (FTTrack track : album.getTracks()) {
         String artist = track.getArtist();
         ArrayList<FTTrack> trackList = artistIndex.get(artist);
         if (trackList == null) trackList = new ArrayList<FTTrack>();
         trackList.add(track);
         artistIndex.put(artist, trackList);
      }
   }

   private TreeMap<String,FTAlbum> albumMap;
   private Map<String,ArrayList<FTTrack>> artistIndex;
}
```

```
/*
 * File: FTAlbum.java
 * ------------------
 * This file implements the data structure for an album in the
 * FlyTunes database.
 */

import acm.util.*;
import java.io.*;
import java.util.*;

/**
 * This class represents an album in the FlyTunes database.
 */

public class FTAlbum {

/**
 * Returns the title of the album.
 *
 * @return The title of the album
 */
   public String getTitle() {
      return title;
   }

/**
 * Returns the total running time of the album.
 *
 * @return The total running time in seconds of the album
 */
   public int getTotalRunningTime() {
      int time = 0;
      for (FTTrack track : tracks) {
         time += track.getRunningTime();
      }
      return time;
   }

/**
 * Returns an array containing the tracks on the album.
 *
 * @param number The question number
 * @return The <code>TMQuestion</code> object with that number
 */
   public FTTrack[] getTracks() {
      /* The argument to toArray makes sure the array has the correct type */
      return tracks.toArray(new FTTrack[tracks.size()]);
   }
```

```
/**
 * Reads the information for an album from the reader.  The external
 * form of the album data consists of the album title followed by
 * individual lines representing single tracks.  The tracks are
 * terminated by the end of file or a blank line.
 */
   public static FTAlbum readAlbum(BufferedReader rd) {
      try {
         String line = rd.readLine();
         if (line == null) return null;
         FTAlbum album = new FTAlbum();
         album.title = line;
         album.tracks = new ArrayList<FTTrack>();
         while (true) {
            FTTrack track = FTTrack.readTrack(rd);
            if (track == null) break;
            track.setAlbum(album);
            album.tracks.add(track);
         }
         return album;
      } catch (IOException ex) {
         throw new ErrorException(ex);
      }
   }

/* Instance variables */

   private String title;
   private ArrayList<FTTrack> tracks;

}
```

```
/*
 * File: FTTrack.java
 * ------------------
 * This file implements the data structure for a single track in the
 * FlyTunes database.
 */

import acm.util.*;
import java.io.*;

/**
 * This class represents a single track in the FlyTunes database.
 */

class FTTrack {

/**
 * Returns the title of this track.
 *
 * @return The title of this track
 */
   public String getTitle() {
      return title;
   }

/**
 * Returns the artist for this track.
 *
 * @return The artist for this track
 */
   public String getArtist() {
      return artist;
   }

/**
 * Returns the running time of the track in seconds.
 *
 * @return The running time of this track
 */
   public int getRunningTime() {
      return runningTime;
   }

/**
 * Returns the album to which this track belongs.
 *
 * @return The album to which this track belongs
 */
   public FTAlbum getAlbum() {
      return album;
   }
```

```
/**
 * Converts a Track object to its string representation, which is chosen
 * to be the same as the external form used by readTrack.
 *
 * @return The string representation of this track
 */
   public String toString() {
      String str = artist + ": " + '"' + title + '"';
      if (runningTime != 0) {
         str += " " + convertSecondsToTimeString(runningTime);
      }
      return str;
   }

/**
 * Reads the information from a track from the specified reader.
 * The line is in the form
 *
 *      Artist: "Title" (time)
 *
 * The time specification is optional.  If it is missing altogether,
 * the running time of the track is listed as 0.
 *
 * If readTrack reaches the end of file or encounters a blank line,
 * this method returns null.
 */
   public static FTTrack readTrack(BufferedReader rd) {
      try {
         String line = rd.readLine();
         if (line == null || line.length() == 0) return null;
         int colon = line.indexOf(':');
         if (colon == -1) {
            throw new ErrorException("Missing colon after artist");
         }
         int quote1 = line.indexOf('"', colon + 1);
         int quote2 = line.indexOf('"', quote1 + 1);
         if (quote1 == -1 || quote2 == -1) {
            throw new ErrorException("Missing quotes around title");
         }
         if (line.substring(colon + 1, quote1).trim().length() != 0) {
            throw new ErrorException("Unexpected text on line");
         }
         FTTrack track = new FTTrack();
         track.artist = line.substring(0, colon).trim();
         track.title = line.substring(quote1 + 1, quote2);
         track.runningTime
         = convertTimeStringToSeconds(line.substring(quote2 + 1));
         return track;
      } catch (IOException ex) {
         throw new ErrorException(ex);
      } catch (NumberFormatException ex) {
         throw new ErrorException("Illegal numeric value");
      }
   }
```

```
/**
 * Converts a string to a running time.  The time is expressed
 * either as a number of seconds or as a combination of minutes
 * and seconds separated by a colon to the total number of seconds.
 * The time string may also be enclosed in parentheses.  An empty
 * string is interpreted as 0.
 *
 * @param str The time value string
 * @return The number of seconds specified by the string
 */
   public static int convertTimeStringToSeconds(String str) {
      str = str.trim();
      if (str.length() == 0) return 0;
      if (str.startsWith("(") && str.endsWith(")")) {
         str = str.substring(1, str.length() - 1);
      }
      int colon = str.indexOf(':');
      if (colon == -1) return Integer.parseInt(str);
      int min = Integer.parseInt(str.substring(0, colon));
      int sec = Integer.parseInt(str.substring(colon + 1));
      return 60 * min + sec;
   }

/**
 * Converts a number of seconds into a running time string.  The
 * string is always expressed in the form min:sec.
 *
 * @param time The running time in seconds
 * @return A string in the form "min:sec"
 */
   public static String convertSecondsToTimeString(int time) {
      int min = time / 60;
      int sec = time % 60;
      if (sec < 10) {
         return min + ":0" + sec;
      } else {
         return min + ":" + sec;
      }
   }

/**
 * Sets the album to which this track belongs.  This method is intended
 * to be called by the FTAlbum class and not by clients.  Leaving off
 * the public designation means that this method can be called only
 * from within the package that contains it.
 */
   void setAlbum(FTAlbum album) {
      this.album = album;
   }

/* Instance variables */
   private FTAlbum album;
   private String title;
   private String artist;
   private int runningTime;
}
```