Much of the complexity of a project like Adventure comes in the design of the data structure. To make sure that everyone starts out on a reasonable path, we’ve done that design work for you, which makes the assignment quite a bit easier. At the same time, taking the design phase out of the assignment means that you get relatively little practice with one of the more important aspects of programming. To remedy that, this week’s section consists of two problems involving data structure design. The first is similar to the type of data structure question that we like to ask on the final exam; the second is a bit more open-ended and allows you to think about a more significant problem in data representation than you could solve in an exam setting.

1. Designing a CalendarDate class

Implement a class named CalendarDate that exports the following methods:

- A constructor that allows the client to create a new date by supplying the month, day, and year as integer values. Thus, it should be possible to write the following declarations that use the CalendarDate class:

  ```java
  CalendarDate independenceDay = new CalendarDate(7, 4, 1776);
  CalendarDate bastilleDay = new CalendarDate(7, 14, 1789);
  CalendarDate today = new CalendarDate(3, 2, 2010);
  CalendarDate tomorrow = new CalendarDate(3, 3, 2010);
  ```

- Three getter methods—`getMonth`, `getDay`, `getYear`—that allow clients to retrieve these components of a CalendarDate object.

- A `compareTo` method that compares the current CalendarDate object with a second CalendarDate object supplied as an argument. The `compareTo` method should return—in much the same way as it does for the String class—an integer that is less than 0 if this date comes before the argument date, an integer greater than 0 if this date comes after the argument date, and 0 if the two dates are the same. Thus, given the earlier declarations, the expression

  ```java
  independenceDay.compareTo(bastilleDay)
  ```

  should return a negative integer, because the original Independence Day occurred earlier than the original Bastille Day. Using the same logic

  ```java
  tomorrow.compareTo(today)
  ```

  should return a positive integer, and

  ```java
  today.compareTo(today)
  ```

  should return 0.

- A `toString` method that converts a CalendarDate object into a String in the following form:

  ```java
  Month day, year
  ```

  where Month is the full name of the month, day is the day of the month, and year is the year. For example, the call
independenceDay.toString()
should return the string "July 4, 1776".

As you write this program, you should keep the following points in mind:

• You may assume that all arguments to the constructor are valid and need not check them to see if they represent a legal date.
• Your class definition should not include any instance variables that are visible outside the CalendarDate class itself.

2. Designing a data structure for a FlyTunes music program

Most of you have presumably used a digital music application such as iTunes, probably within the last 24 hours. Such systems allow you to store a database consisting of a number of albums, each of which contains a number of tracks. Your job in this problem is to design the data structure for a new music application called FlyTunes that does much the same thing.

In FlyTunes, the data structure for each track must include the following information:

• The title of the song recorded on that track
• The artist who recorded it
• The running time in seconds

Each album then consists of a number of tracks, along with the title of the album. The FlyTunes database records information for as many albums as you choose to store in it.

In this problem, your mission is to define three classes—FTDataBase, FTAlbum, and FTTrack—that maintain these data structures. Your job therefore consists of the following steps:

1. Figure out what information you need to store with each of these structures.
2. Determine what information clients need to be able to obtain from the data structure.
3. Write the prototypes for a set of methods that retrieve this information.
4. Design an external representation that allows you to store FlyTunes data as a text file.
5. Code the implementation for each of the three classes.

It is important to note that this problem is wildly underspecified and there are many ways to solve it, depending on what data you choose to store and what methods you offer to the client. The code in the solution is only one possibility.