Programming 3. practice exercise for ZH1

Create a single-layer console app that will be able to work with concert location data stored in a local database and with random ticket sales data. From the ticket sales you must create statistics, and using the stats you have to decide if there are any errors with the sales data or not (did we sell more tickets than the concert venue's capacity?).

Creating the Database and the Entities

Using the Code First approach, create a local database into a separated Class Library with three tables: `venues`, `sectors`, `sellers`.

a) The three entities:
   a. `Venue (concert venue)`
      i. `int Id` – primary key, database provided using automatically incremented values
      ii. `string Name` – maximum length is 100 characters, obligatory field
   b. `Sector (a sector inside a venue)`
      i. `int Id` – primary key, database provided using automatically incremented values
      ii. `int Capacity`
      iii. `int VenueId` – foreign key that references the venues table
      iv. `string Code` – maximum length is 13 characters, obligatory field
   c. `Seller (a ticket seller authorized for the venue)`
      i. `int Id` – primary key, database provided using automatically incremented values
      ii. `string Name` – maximum length is 100 characters, obligatory field
      iii. `int VenueId` – foreign key that references the venues table

b) Use attributes to define the required table structure, and use the fluent syntax to define foreign keys / navigational properties (you can fully use the fluent api instead of attributes if you want, but that is longer and requires more typing).

c) Define sample data into the database using data seeding (*Venue: Pap László Sports Arena; Sectors: A, B, C, with 1100, 2500, 1500 capacities; Sellers: Broadway, Eventim*).

Generating sales statistics

Create a Class Library into the application that is capable of randomly generate daily sales statistics.

a) A single day is stored in the `DailySale` class
   a. `DateTime Date`
   b. `string SellerName`
   c. `string SectorCode`
   d. `int TicketsSold`

b) Create a generator class called `DailyStatGenerator` that has
   a. a constructor where it receives all possible sellers and sectors in separated string arrays;
   b. a `GenerateList(int numDays, int numInstances, int maxSold)` method that will return with a list of `DailySale` instances. The `numInstances` parameter will define how many `DailySale` instances should be randomly generated; the `numDays` defines the maximum number of days in the past that can be used for the random date generation; the `maxSold` defines the biggest possible `TicketsSold` value in the random instances. Using these (and by randomly picking one seller and one sector from the ones received in the constructor) you should generate the list, filled with random ticket sale statistics.

XML GENERATION

Add a Class Library, that is capable of generating an XML file from a list of `DailySale` instances.

a) The XML root node should be `stats`;
b) For every day it should aggregate the sales in the lists for every different sectors and sellers (nodes: `day`, `sector`, `seller`, `sold`). The day should be displayed with the ToShortDateString(), the identification of the date, the sector and the seller should be done using XML attributes;
c) Example XML:

```
<stats>
  <day date="2020. 10. 01.">
    <sector code="B">
      <sold>149</sold>
    </sector>
    <sector code="A">
      <sold>58</sold>
    </sector>
    <sector code="C">
      <sold>199</sold>
    </sector>
    <seller name="Eventim">
      <sold>222</sold>
    </seller>
    <seller name="Broadway">
      <sold>184</sold>
    </seller>
  </day>
  <day date="2020. 10. 02.">
    <sector code="B">
      <sold>59</sold>
    </sector>
    <sector code="A">
      <sold>136</sold>
    </sector>
    <sector code="C">
      <sold>93</sold>
    </sector>
    <seller name="Eventim">
      <sold>141</sold>
    </seller>
    <seller name="Broadway">
      <sold>147</sold>
    </seller>
  </day>
…
</stats>
```

PROGRAM

a) Use the database to display all sectors and sellers;
b) Use the `DailyStatGenerator` to generate a list of sales transactions;
c) Use XML generator to craete the daily statistics in XML format;
d) Then, use the XML to:
    a. Determine the total number of ticket sales for every seller;
    b. Determine the total number of ticket sales for every sector;
    c. Determine the number of remaining tickets for every sector;
    d. Decide if there is any errors in the sales statistics (if we sold more tickets in any sector than its capacity).

Have fun while developing!