Advanced Programming Techniques (java or c #) 2

Databases and EF Core

Initial tip: you will be printing your results to the console. In entity classes, override ToString method, so you can see actual data that they contain. Create separate functions for printing your results.

1. Set up a database environment.
   a. Go to Tools > NuGet Package Manager > Package Manager Console
   b. In opened terminal, run
   ```
   Install-Package Microsoft.EntityFrameworkCore.Sqlite
   ```
   c. Add a class that extends DbContext and implement it to configure your database
   ```
   public class MyDatabaseContext : DbContext {
           public string DbPath { get; }
           public BloggingContext()
           {
                   var folder =
   Environment.SpecialFolder.LocalApplicationData;
                   var path = Environment.GetFolderPath(folder);
                   DbPath = System.IO.Path.Join(path, "blogging.db"); }

                   // The following configures EF to create a Sqlite database
   file in the             // special "local" folder for your platform.
                   protected override void
   OnConfiguring(DbContextOptionsBuilder                options) =>
   options.UseSqlite($"Data Source={DbPath}");
   }
   ```
   d. In the same file add classes that will model a Student and a Class. Student class should include id, first name and last name. Class should include id and name.
   Use your classes in a database: in MyDatabaseContext add a DbSet fields for both classes
   ```
   public DbSet<Student> Students { get; set; }
   ```
   e. Create your database and create initial migrations. Run in Packet Manager Console:
   ```
   Install-Package Microsoft.EntityFrameworkCore.Tools
   Add-Migration InitialCreate
   ```

Update-Database

   f. In your main file, create a static instance of MyDatabaseContext and use it to add some students and classes.
      In order to interact with an entity, use DbContext.*Entity* syntax on MyDatabaseContext instance, ie db.Student.Add
      Remember to call SaveChanges function to apply change.
      Retrieve your saved data and print it in console. Use DbContext.*Entity* syntax and ToList function.
   g. Running your program multiple times will keep adding entries to a database. To prevent that, remove your database contents before every run. You can iterate over every entry and call Remove function.

2. Add a Teacher entity to your database. Add an one-to-one relationship between Teacher and a Class. In order to do this, add an int property to the Class, to represent foreign key with Teacher id. Add a Teacher field so, you will be able to reference a connected Teacher, once you retrieve the Class.
   a. In your main file, once a class is created, retrieve it by name (you can use Linq and *Entity* property on resulting EntityEntry). Once it's retrieved, create a new teacher and assign it to the class. Remember to save changes to the database. Retrieve class again to confirm that Teacher is added to the Class.
   b. Add another Teacher and another Class. At the end, remove the Teacher and print all Classes again to see how foreign key reference works for deletion.
      Remove the remaining class and print Teachers.
3. Teacher – Class relationship is de facto a one-to-many. In order to be able to access teachers classes, add an ICollection<Class> field to a Techer class. In the main file, retrieve a teacher and print all their classes.
4. Add a many-to-many relationship to connect students and classes. Add a collection of students in a Class and vice versa.
   You don't need to explicitly create a join entity as you would in sql, but you still need to create and apply a migration, because Entity Framework still needs to create it under the hood.