Вам выдана папка **NewVariant**, содержимое которой изменять и дополнять не нужно.

Ознакомьтесь с содержимым папки NewVariant/Models, в ней находится готовая модель базы данных (БД), которая состоит из 4 сущностей. Иными словами, БД содержит 4 таблицы.

- **Shop** – оффлайн магазин, оборудованный для продажи товаров. Каждый магазин имеет уникальный номер, а также содержит информацию о своем названии и месторасположении.
- **Buyer** – покупатель – обычный человек, совершающий покупки. Сущность покупателя в базе данных содержит информацию о своем уникальном идентификаторе, имени, фамилии и месте проживания.
- **Good** – товар, лежащий на полке определенного магазина. Каждый товар описывается собственным уникальным номером, идентификатором магазина, в котором его можно купить, а также категорией, к которой этот товар относится и, конечно же, ценой.
- **Sale** – совершенная в магазине покупка определенного товара. Покупка содержит информацию о покупателе, совершившем эту покупку (его id), магазине, в котором покупка была совершена (его id), товаре, который был куплен (его id), и количестве этого товара.

Ознакомьтесь с содержимым папки NewVariant/Exceptions, в ней находится один класс **DataBaseException**, содержащий реализацию пользовательского исключения, генерируемого при ошибках во время работы с таблицами БД.

Ознакомьтесь с содержимым папки Interfaces, в ней находятся 3 интерфейса (подробности о методах интерфейса см. ниже в варианте):

- **IEntity** – этот интерфейс описывает сущность, хранимую в БД. Его реализуют все наши модельные сущности.
- **IDataBase** – этот интерфейс содержит список обобщенных методов, которые позволяют работать с таблицами из БД (создавать и получать таблицу, добавлять записи в таблицу, а также сериализовать и десериализовать содержимое). Методы этого интерфейса вам необходимо реализовать самостоятельно.
- **IDataAccessLayer** – данный интерфейс содержит набор методов, которые обращаются к БД. Фактически они представляют собой LINQ-запросы, которые вам необходимо реализовать самостоятельно.

Ваше задание:

На основании полученного решения создать проект. Создать библиотеку классов.
Библиотека должна содержать два класса:

1. **DataBase**, реализующий **IDataBase.**
2. **DataAccessLayer**, реализующий **IDataAccessLayer.**

Важно: в обоих классах должен быть конструктор по умолчанию.
Для тестирования реализации создайте вспомогательный проект, который не сдается и не оценивается. Самостоятельно сгенерируйте экземпляры

модельных сущностей и заполните ими БД. Проверьте работоспособность всех методов из классов **DataBase и DataAccessLayer.**
Подробнее о методах из **IDataBase**:

1. **public void CreateTable<T>() where T : IEntity;**

   Создает новую таблицу типа T. Если такая таблица уже существует, выбрасывает исключение типа DataBaseException.

2. **public void InsertInto<T>(Func<T> getEntity) where T : IEntity;**

   Добавляет новую строку в таблицу типа T. Параметр getEntity возвращает экземпляр, соответствующего типа. В случае попытки добавления строки в несуществующую таблицу, выбрасывает исключение типа DataBaseException.

3. **public IEnumerable<T> GetTable<T>() where T : IEntity;**

   Возвращает ссылку на таблицу типа T. Если такой таблицы не существует, выбрасывает исключение типа DataBaseException.

4. **public void Serialize<T>(string path) where T : IEntity;**

   Сериализует таблицу типа T в файл, расположенный по переданному пути(path). Использует JSON-сериализацию. Если такой таблицы не существует, выбрасывает исключение типа DataBaseException.

5. **public void Deserialize<T>(string path) where T : IEntity;**

   Десериализует и сохраняет таблицу типа T из файла, расположенного по переданному пути(path). Использует JSON-сериализацию. Если таблица такого типа уже существовала ранее, то содержимое перезаписывается.

Подробнее о методах **IDataAccessLayer**:

1. **public IEnumerable<Good> GetAllGoodsOfLongestNameBuyer(IDataBase dataBase);**

   Возвращает список всех товаров, купленных покупателем с самым длинным именем. Если самых длинных имен несколько, то возвращается список для последнего в лексикографическом порядке имени.

2. **public string? GetMostExpensiveGoodCategory(IDataBase dataBase);**

   Возвращает название категории самого дорогого товара. Если таких товаров несколько, то возвращается категория любого из них.

3. **public string? GetMinimumSalesCity(IDataBase dataBase);**

   Получить название города, в котором было потрачено меньше всего денег

(= с наименьшей совокупной суммой покупок). Если таких городов несколько, то возвращается любой из них.

4. **public IEnumerable<Buyer> GetMostPopularGoodBuyers(IDataBase dataBase);**

   Возвращает список покупателей, которые купили самый популярный товар – такой товар, чьих единиц приобретено максимальное число.
   Если популярных товаров несколько, то вернуть список покупателей для любого из них.

5. **public int GetMinimumNumberOfShopsInCountry(IDataBase dataBase);**

   Для каждой страны определяет количество ее магазинов. Возвращает наименьшее из полученных значений.

6. **public IEnumerable<Sale> GetOtherCitySales(IDataBase dataBase);**

   Возвращает список покупок, совершенных покупателями во всех городах, отличных от города их проживания.

7. **public long GetTotalSalesValue(IDataBase dataBase);**

   Возвращает общую стоимость покупок, совершенных всеми покупателями.

На 9-10 баллов стоит реализовать веб-приложение, демонстрирующее вышеперечисленный функционал.