16.1-CRUD Operations in Entity Framework

Program

```
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;
using System.Linq;

namespace InventorySystem
{
    internal class Product
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public decimal Price { get; set; }
    }

    internal class AppDbContext : DbContext
    {
        public DbSet<Product> Products { get; set; }

        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            optionsBuilder.UseSqlServer("yourconnection");
        }
    }
```

```
internal class Program
{
    static void Main()
    {
        using (var context = new AppDbContext())
        {
            context.Database.Migrate();
        }

        // CRUD Operations
        using (var context = new AppDbContext())
        {
            // Create and insert sample data
            var newProduct1 = new Product { Name = "Laptop", Price = 999.99M };
            var newProduct2 = new Product { Name = "Smartphone", Price = 699.99M };
            var newProduct3 = new Product { Name = "Tablet", Price = 399.99M };

            context.Products.AddRange(newProduct1, newProduct2, newProduct3);
            context.SaveChanges();

            // Read and display all products
            var allProducts = context.Products.ToList();
            Console.WriteLine("All Products:");
            foreach (var product in allProducts)
            {
                Console.WriteLine($"Id: {product.Id}, Name: {product.Name}, Price: {product.Price:C}");
            }
```

```
            // Update one product
                    var productToUpdate = context.Products.FirstOrDefault(product => product.Id ==
newProduct1.Id);
            if (productToUpdate != null)
            {
                productToUpdate.Name = "Updated Laptop";
                productToUpdate.Price = 1099.99M;
            }

            // Delete one product
                    var productToDelete = context.Products.FirstOrDefault(product => product.Id ==
newProduct2.Id);
            if (productToDelete != null)
            {
                context.Products.Remove(productToDelete);
            }

            // Save changes once after all modifications
            context.SaveChanges();
        }
    }
  }
}
```

Exercise

Write a C# program that create an inventory management system implemented in C# using Entity Framework Core (EF Core) for database operations. It consists of three main components: the Product class representing individual products, the AppDbContext class representing the database context, and the Program class containing the main method where CRUD (Create, Read, Update, Delete) operations are performed on products.

The Product class defines properties such as Id, Name, and Price to represent product attributes. The AppDbContext class inherits from DbContext and includes a DbSet<Product> property to interact with the Product table in the database. It also overrides the OnConfiguring method to specify the database connection string.

Within the Main method of the Program class, the program first ensures that the database is up to date by calling context.Database.Migrate(), which applies any pending migrations. Then, it performs CRUD operations on products using the AppDbContext. It creates and inserts sample product data into the database, reads all products from the database, updates the name and price of one product, deletes another product, and finally saves the changes to the database.

Hint

Entity Framework Core Integration: The program utilizes Entity Framework Core (EF Core) to interact with a database. It defines a database context (AppDbContext) inheriting from DbContext and a model class (Product) representing products in the database.

Database Migration: It includes code to ensure that the database schema is up to date using migrations. The context.Database.Migrate() method is called to apply any pending migrations to the database when the application starts.

CRUD Operations: The program demonstrates CRUD operations (Create, Read, Update, Delete) on products. It creates sample product data, reads and displays all products, updates the name and price of one product, and deletes another product from the database. Finally, it saves the changes to the database.

Explanation

This program represents a simple inventory management system implemented in C# using Entity Framework Core (EF Core) for database interaction. It consists of three main components: the Product class, the AppDbContext class inheriting from DbContext, and the Program class containing the main logic.

Firstly, the Product class defines the properties of a product, including an auto-incrementing Id, the Name, and the Price. These properties represent the fields stored in the database for each product.

Secondly, the AppDbContext class serves as the database context, where a DbSet<Product> property named Products is declared to represent the collection of products in the database. The OnConfiguring method configures the database connection using SQL Server.

The Program class contains the main entry point and logic of the application. Within the Main method, database migration is performed to ensure that the database schema is up to date. Then, CRUD operations are demonstrated:

Create: Three sample products (laptop, smartphone, and tablet) are created and inserted into the database using the AddRange method. These new products are then saved to the database using SaveChanges.

Read: All products from the database are retrieved using ToList() and displayed on the console, showing their Id, Name, and Price.

Update: One product (in this case, the first product created) is selected for updating. Its Name is changed to "Updated Laptop", and its Price is adjusted. The changes are then saved to the database using SaveChanges.

Delete: Another product (in this case, the second product created) is selected for deletion. It is removed from the Products set and subsequently deleted from the database using Remove. Again, changes are saved to the database using SaveChanges.