بسم تعالی

نام و نام خانوادگی : سهیل نعمت الهی

موضوع : project 2 / **Software Engineering** /clean architecture

دانشکده ملی و مهارت آیت الله خامنه ای

```
namespace Core.Entities
{
    public class Ticket
    {
        public int TicketId { get; set; }
        public int FlightId { get; set; }
        public string PassengerName { get; set; }
        public DateTime DateOfPurchase { get; set; }
    }
}
```

1. این خط اعلام می‌کند که کد داخل این فضا (namespace) مربوط به موجودیت‌ها (Entities) پروژه است .

2. شناسه بلیط که از نوع عدد صحیح (int) است و برای شناسایی بلیط به‌طور منحصر به فرد استفاده می‌شود.

3. شناسه پرواز که بلیط مربوطه را به یک پرواز خاص متصل می‌کند.

4. نام مسافر که بلیط به آن اختصاص یافته است.

5. تاریخ خرید بلیط که نشان‌دهنده زمانی است که بلیط خریداری شده است.

این کلاس داده‌های مربوط به بلیط‌ها را مدل می‌کند و به عنوان یک موجودیت در پروژه عمل می‌کند.

```
namespace Core.Entities
{
    public class Flight
    {
        public int FlightId { get; set; }
        public string FlightName { get; set; }
        public DateTime DepartureDate { get; set; }
        public string DepartureCity { get; set; }
        public string ArrivalCity { get; set; }
        public decimal Price { get; set; }
    }
}
```

1. فضای نام این کد به بخش موجودیت‌ها مربوط می‌شود.

2. شناسه پرواز که یک عدد صحیح است و به‌طور منحصر به فرد پرواز را شناسایی می‌کند.

3. نام پرواز که می‌تواند شامل اطلاعاتی مانند شماره پرواز یا نام شرکت هواپیمایی باشد.

4. شهر مبدا که پرواز از آنجا آغاز می‌شود.

5. شهر مقصد که پرواز به آنجا می‌رود.

6. تاریخ و زمان پرواز که زمان دقیق پرواز را مشخص می‌کند.

7. قیمت بلیط که معمولاً به صورت عدد اعشاری مشخص می‌شود.

این کلاس برای ذخیره‌سازی اطلاعات پروازها استفاده می‌شود.

```
namespace Core.Repositories
{
    public interface ITicketRepository
    {

        void AddTicket(Ticket ticket);
        IEnumerable<Ticket> GetAllTickets();
        Ticket GetTicketById(int ticketId);
    }
}
```

1. فضای نام اینترفیس ITicketRepository که در لایه ریپوزیتوری‌ها قرار دارد.

2. متدی برای افزودن بلیط به ذخیره‌سازی. این متد یک شی از نوع Ticketمی‌گیرد و آن را ذخیره می‌کند.

3. متدی برای دریافت تمامی بلیط‌ها. این متد مجموعه‌ای از بلیط‌ها را برمی‌گرداند.

4. متدی برای دریافت یک بلیط خاص بر اساس شناسه بلیط (ticketId). این متد یک شی از نوع Ticket باز می‌گرداند.

این اینترفیس روش‌های استاندارد برای کار با بلیط‌ها را تعریف می‌کند.

```
namespace Core.Repositories
{
    public class TicketRepository : ITicketRepository
    {
        private readonly List<Ticket> _tickets = new List<Ticket>();

        public void AddTicket(Ticket ticket)
        {
            _tickets.Add(ticket);
        }

        public IEnumerable<Ticket> GetAllTickets()
        {
            return _tickets;
```

```
        public Ticket GetTicketById(int ticketId)
        {
            return _tickets.FirstOrDefault(t => t.TicketId == ticketId);
        }
    }
}
```

1. فضای نام این کد که مربوط به ریپوزیتوری‌ها است.
2. کلاس TicketRepository که اینترفیس ITicketRepository را پیاده‌سازی می‌کند.
3. یک لیست در حافظه به نام _tickets برای ذخیره‌سازی بلیط‌ها تعریف شده است.
4. متدی برای افزودن بلیط به لیست _tickets .
5. متدی برای بازگشت تمامی بلیط‌ها که در لیست _tickets ذخیره شده‌اند.
6. متدی برای جستجو در لیست _tickets و بازگشت بلیط خاصی بر اساس شناسه (ticketId).

این کلاس پیاده‌سازی واقعی عملیات ذخیره‌سازی بلیط‌ها است.

```
namespace Core.Repositories
{
    public interface IFlightRepository
    {
        IEnumerable<Flight> GetAllFlights();
        Flight GetFlightById(int flightId); }}
```

1. فضای نام اینترفیس IFlightRepository که در لایه ریپوزیتوری‌ها قرار دارد.

2. متدی برای دریافت تمامی پروازها. این متد مجموعه‌ای از پروازها را باز می‌گرداند.

3. متدی برای دریافت یک پرواز خاص بر اساس شناسه پرواز (flightId) .

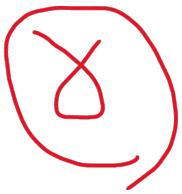
این اینترفیس روش‌های استاندارد برای کار با پروازها را تعریف می‌کند.

```
namespace Core.Repositories
{
```

```
    public class FlightRepository : IFlightRepository
    {
```

```
        private readonly List<Flight> _flights = new List<Flight>
        {
            new Flight { FlightId = 1, FlightName = "Flight 101", DepartureCity
= "New York", ArrivalCity = "London", DepartureDate =
DateTime.Now.AddDays(1), Price = 500 },
            new Flight { FlightId = 2, FlightName = "Flight 102", DepartureCity
= "Los Angeles", ArrivalCity = "Tokyo", DepartureDate =
DateTime.Now.AddDays(2), Price = 700 }
        };
```

```
        public IEnumerable<Flight> GetAllFlights()
        {
            return _flights;
        }
```



```
    public Flight GetFlightById(int flightId)
    {
        return _flights.FirstOrDefault(f => f.FlightId == flightId);
    }
  }

}
```

1. فضای نام این کد که مربوط به ریپوزیتوری‌ها است.
2. کلاس FlightRepository که اینترفیس IFlightRepository را پیاده‌سازی می‌کند.
3. یک لیست در حافظه به نام flights_برای ذخیره‌سازی پروازها تعریف شده است.
4. متدی برای بازگشت تمامی پروازها که در لیست flights_ذخیره شده‌اند.
5. متدی برای جستجو در لیست flights_ و بازگشت پرواز خاصی بر اساس شناسه (flightId) .

این کلاس پیاده‌سازی واقعی عملیات ذخیره‌سازی پروازها است.

```
namespace Core.UseCases
{
    public interface ITicketService
    {

        void ReserveTicket(int flightId, string passengerName);
    }
}
```

1. فضای نام اینترفیس ITicketService که مربوط به منطق تجاری پروژه است.

2. اینترفیس ITicketService که مشخص می‌کند کلاس‌هایی که آن را پیاده‌سازی می‌کنند باید متدهایی برای رزرو بلیط داشته باشند.

3. متدی برای رزرو بلیط که شناسه پرواز (flightId) و نام مسافر (passengerName) را به عنوان ورودی می‌گیرد و بلیط را رزرو می‌کند.

این اینترفیس شامل منطق تجاری رزرو بلیط است.

```
namespace Core.UseCases
{
    public class TicketService : ITicketService
    {
        private readonly ITicketRepository _ticketRepository;
        private readonly IFlightRepository _flightRepository;
```

```
        public TicketService(ITicketRepository ticketRepository,
IFlightRepository flightRepository)
        {
            _ticketRepository = ticketRepository;
            _flightRepository = flightRepository;
        }


        public void ReserveTicket(int flightId, string passengerName)
        {
            var flight = _flightRepository.GetFlightById(flightId);
            if (flight == null)
                throw new Exception("Flight not found");

            var ticket = new Ticket
            {
                FlightId = flightId,
                PassengerName = passengerName,
                DateOfPurchase = DateTime.Now
            };

            _ticketRepository.AddTicket(ticket); }}}
```

1. فضای نام کلاس TicketService که مربوط به منطق تجاری است.

2. کلاس TicketService که اینترفیس ITicketService را پیاده‌سازی می‌کند.

3. یک شیء از نوع ITicketRepository که برای تعامل با داده‌های بلیط‌ها استفاده می‌شود.

4. یک شیء از نوع IFlightRepository که برای تعامل با داده‌های پروازها استفاده می‌شود.

5. سازنده که وابستگی‌های ITicketRepository و IFlightRepository را از طریق تزریق وابستگی دریافت می‌کند.

6. متد رزرو بلیط که شناسه پرواز و نام مسافر را به‌عنوان ورودی می‌گیرد و بلیط را رزرو می‌کند.

7. پرواز مربوطه را از ریپوزیتوری پروازها دریافت می‌کند.

8. اگر پرواز یافت نشد، یک استثنا پرتاب می‌کند.

9. یک شی جدید از نوع ITicketایجاد می‌کند.

10. بلیط رزرو شده را به ریپوزیتوری بلیط‌ها اضافه می‌کند.

این کلاس منطق تجاری مربوط به رزرو بلیط‌ها را پیاده‌سازی می‌کند.

```
namespace Core.UseCases
{
    public interface IFlightService
    {
        IEnumerable<Flight> GetAvailableFlights();
    }
```

```
public interface ITicketService
    {
        void ReserveTicket(int flightId, string passengerName);
    }
}
```

1. فضای نامی که اینترفیس‌ها را در بخشی منطقی به نام موارد استفاده دسته‌بندی می‌کند . Core.UseCases

2. یک اینترفیس عمومی که قرارداد مدیریت پروازها را تعریف می‌کند. این اینترفیس فقط متدهای لازم برای دریافت اطلاعات پروازها را مشخص می‌کند و پیاده‌سازی آن بر عهده کلاس‌های مرتبط است.

3. متدی که لیستی از اشیای Flight(پروازها) را برمی‌گرداند. این متد برای نمایش پروازهای موجود به کاربران استفاده می‌شود.

4. یک اینترفیس عمومی که قرارداد مدیریت بلیط‌ها را تعریف می‌کند. این اینترفیس خدمات رزرو بلیط را مشخص می‌کند.

5. متدی که مسئول رزرو بلیط است. این متد شناسه پرواز (flightId) و نام مسافر (passengerName) را به‌عنوان ورودی می‌گیرد و عملیات رزرو را انجام می‌دهد.

```
namespace Core.UseCases
{
    public class FlightService : IFlightService
    {
        private readonly IFlightRepository _flightRepository;

        public FlightService(IFlightRepository flightRepository)
        {
            _flightRepository = flightRepository;
        }

        public IEnumerable<Flight> GetAvailableFlights()
        {
            return _flightRepository.GetAllFlights();
        }
    }

    public class TicketService : ITicketService
    {
        private readonly ITicketRepository _ticketRepository;
        private readonly IFlightRepository _flightRepository;
```

```
        public TicketService(ITicketRepository ticketRepository,
IFlightRepository flightRepository)
        {
            _ticketRepository = ticketRepository;
            _flightRepository = flightRepository;
        }

        public void ReserveTicket(int flightId, string passengerName)
        {
            var flight = _flightRepository.GetFlightById(flightId);
            if (flight == null)
                throw new Exception("Flight not found");

            var ticket = new Ticket
            {
                FlightId = flightId,
                PassengerName = passengerName,
                DateOfPurchase = DateTime.Now
            };

            _ticketRepository.AddTicket(ticket);}}}
```

1. دسته‌بندی کلاس‌ها در فضای نام برای معماری موارد استفاده.

2. کلاس مدیریت خدمات پرواز که از قرارداد اینترفیس IFlightService پیروی می‌کند.

3. ذخیره وابستگی به ریپوزیتوری برای مدیریت داده‌های پرواز.

4. سازنده برای تزریق ریپوزیتوری پرواز و مقداردهی به متغیر وابستگی.

5. متدی برای دریافت پروازهای موجود که داده‌ها را از ریپوزیتوری دریافت می‌کند.

6. پروازهای موجود را از ریپوزیتوری می‌گیرد و بازمی‌گرداند.

7. کلاس مدیریت خدمات بلیط که از قرارداد اینترفیس ITicketService پیروی می‌کند.

8. ذخیره وابستگی به ریپوزیتوری بلیط‌ها.

9. ذخیره وابستگی به ریپوزیتوری پروازها برای دریافت اطلاعات مرتبط.

10. سازنده برای تزریق وابستگی‌ها و مقداردهی به ریپوزیتوری‌های مربوطه.

11. متدی که عملیات رزرو بلیط را انجام می‌دهد.

12. پرواز را با استفاده از شناسه از ریپوزیتوری دریافت می‌کند.

13. بررسی می‌کند که آیا پرواز موجود است، و در صورت عدم وجود، استثنا پرتاب می‌کند.

14. شیء بلیط جدید ایجاد شده و مقادیر پر می‌شوند.

15. بلیط جدید در ریپوزیتوری مربوطه ذخیره می‌شود.

```
using Core.Repositories;
using Core.UseCases;
using Infrastructure.Data;
using Microsoft.Extensions.DependencyInjection;

class Program
{
    static void Main(string[] args)
    {

        var serviceProvider = new ServiceCollection()
            .AddSingleton<IFlightRepository, FlightRepository>()
            .AddSingleton<ITicketRepository, TicketRepository>()
            .AddSingleton<IFlightService, FlightService>()
            .AddSingleton<ITicketService, TicketService>()
            .BuildServiceProvider();


        var flightService = serviceProvider.GetService<IFlightService>();
        var ticketService = serviceProvider.GetService<ITicketService>();
```

```
        Console.WriteLine("Available Flights:");
        foreach (var flight in flightService.GetAvailableFlights())
        {
            Console.WriteLine($"Flight {flight.FlightId}: {flight.DepartureCity}
-> {flight.ArrivalCity}, Price: {flight.Price}");
        }



        Console.Write("Enter Flight ID to reserve: ");
        int flightId = int.Parse(Console.ReadLine());
        Console.Write("Enter Passenger Name: ");
        string passengerName = Console.ReadLine();
        ticketService.ReserveTicket(flightId, passengerName);



        Console.WriteLine("\nReserved Tickets:");
        foreach (var ticket in ticketService.GetReservedTickets())
        {
            Console.WriteLine($"Ticket ID: {ticket.TicketId}, Passenger:
{ticket.PassengerName}");
        }
    }
}
```

1. استفاده از کلاس‌های موجودیت‌ها(Entities) .

2. استفاده از ریپوزیتوری‌ها.

3. استفاده از منطق تجاری(Use Cases) .

4. استفاده از لایه زیرساختی.

5. استفاده از کتابخانه برای **Dependency Injection** .

6. ایجاد و پیکربندی یک شیء ServiceCollection برای مدیریت تزریق وابستگی‌ها.

7. دریافت خدمات مربوط به پروازها از طریق DI .

8. نمایش لیست پروازهای موجود.

9. نمایش تمامی پروازها با اطلاعات مختلف.

10. درخواست شناسه پرواز از کاربر.

11. راخوانی متد رزرو بلیط.

12. نمایش بلیط‌های رزرو شده.

13. نمایش جزئیات بلیط‌های رزرو شده.

14. پیامی برای تایید رزرو موفقیت‌آمیز.

این کد بخش اصلی برنامه است که به کاربر امکان می‌دهد پروازها را مشاهده کرده و بلیط‌ها را رزرو کند.