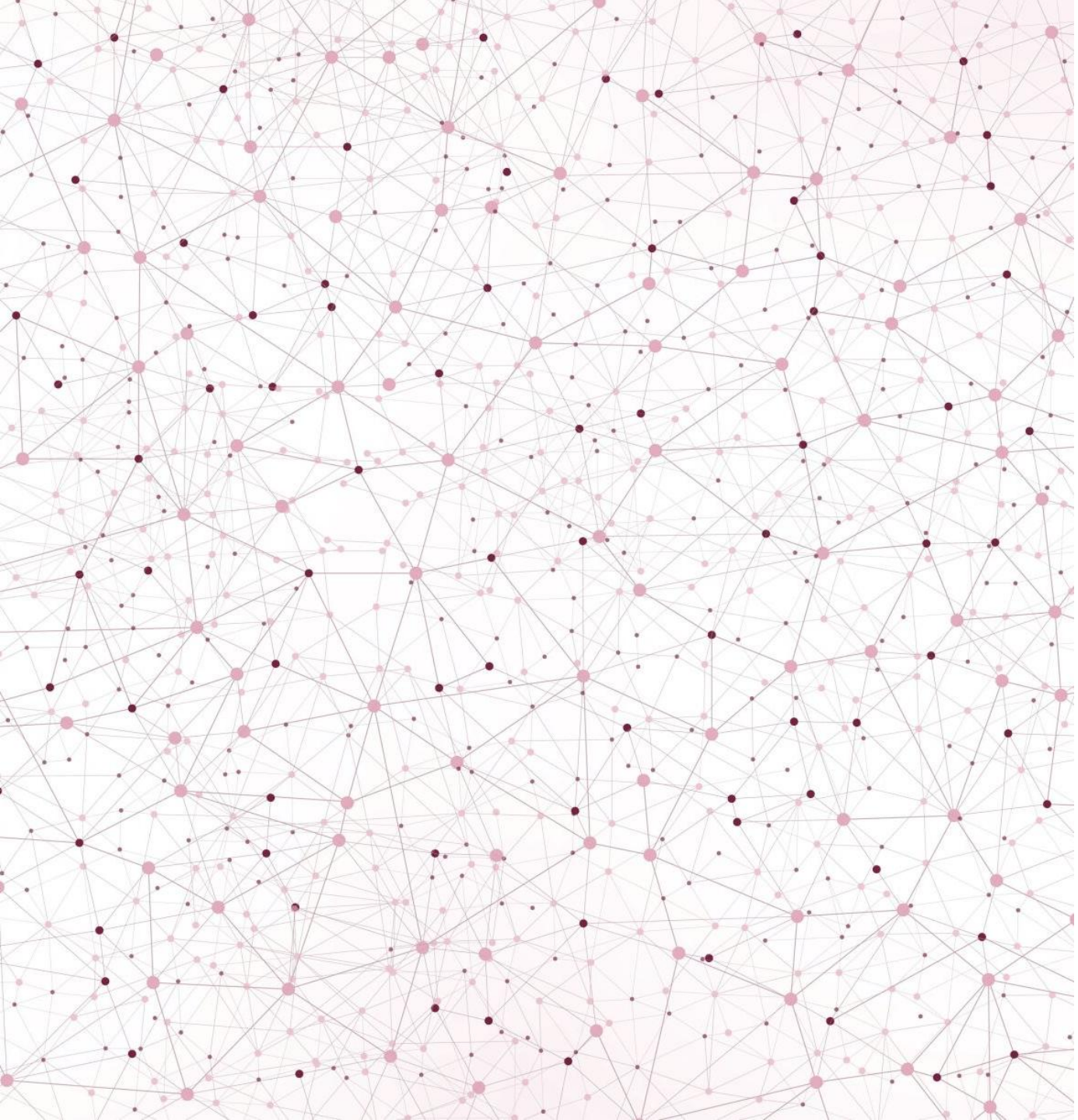


Занятие 8
Domain-Driven Design

Что такое DDD?

Что такое DDD?

Domain-Driven Design (DDD) - это подход к проектированию программного обеспечения, который фокусируется на моделировании предметной области (domain) и ее представлении в виде моделей данных (entities, value objects, aggregates, repositories, etc.).

Основные понятия

- Domain;
- Model;
- Bounded context;
- Entity;
- Value Object;
- Aggregate;
- Repository;
- Domain Service.

Основные понятия

- Domain - предметная область, которая является основой для проектирования программного обеспечения;
- Model - абстракция, отражающая предметную область и ее основные понятия;
- Bounded context - область, в пределах которой термины и модели однозначно интерпретируются;
- Entity - объект, который имеет уникальный идентификатор и идентифицируется по нему;
- Value Object - объект, который не имеет уникального идентификатора и идентифицируется по своим атрибутам;
- Aggregate - группа entities и value objects, которые управляются как единое целое;
- Repository - интерфейс для доступа к aggregate;
- Domain Service - сервис, который реализует бизнес-логику, не принадлежащую ни одной из сущностей.

Какие есть примеры сущностей?

Какие есть примеры сущностей?

- Студент;
- Автомобиль;
- Сотрудник;
- И т.д.

Почему это сущности?

Почему это сущности?

- Студент – однозначно идентифицируется номером студенческого билета;
- Автомобиль – однозначно идентифицируется VIN;
- Сотрудник – однозначно идентифицируется табельным номером.

Какие есть примеры value objects?

Какие есть примеры value objects?

- Размер;
- Цена;
- Цвет;
- И т.д.

Практическая задача

Есть некоторая компания AutoMarket - она специализируется на продаже подержанных автомобилей. Сейчас весь учет ведется в Excel и бизнес хочет автоматизировать процессы.

Функциональные требования

1. Мы хотим иметь возможность регистрировать автомобили, которые поступают к нам на площадку, и видеть их текущий статус: “в наличии”, “зарезервирован”, “продан”;
2. Клиент может зарезервировать автомобиль на определенное время - зарезервированный автомобиль не может быть продан другому клиенту;
3. Клиент может купить автомобиль - купить можно либо зарезервированный автомобиль, либо автомобиль, который находится в наличии;
4. Мы хотим понимать, кто покупает автомобили - имена клиентов, телефоны и почты.

Какие сущности можно выделить?

Какие сущности можно выделить?

- Автомобиль;
- Клиент.

Являются ли продажа и резерв сущностями?

Являются ли продажа и резерв сущностями?

В нашем случае, скорее, нет, так как бизнес рассматривает их только в контексте учета автомобилей, но не как самостоятельные единицы.

А когда они будут являться сущностями?

А когда они будут являться сущностями?

Например, если у нас появится требование о предоставлении отчетов о продажах или резервах. В этом случае уже имеет смысл выделять их как отдельные сущности.

А какие Value Objects возможны в нашей задаче?

А какие Value Objects возможны в нашей задаче?

- VIN – имеет определенный формат, в связи с чем имеет смысл создать для него отдельный value object;
- Sale – можно описать продажу как value object для упрощения работы с ней;
- Reservation – резерв также можно описать как value object;
- Phone – телефон клиента, имеет определенный формат и является хорошим кандидатом для value object;
- Email – адрес электронной почты клиента, который тоже обычно соответствует определенному формату.

Теперь к коду

Создадим проект

```
$ dotnet new solution -n 08-ddd

$ dotnet sln migrate

$ rm 08-ddd.sln

$ dotnet new console -n AutoMarketApp -o ./AutoMarketApp

$ dotnet sln add ./AutoMarketApp/AutoMarketApp.csproj
```

Структура проекта

- Корень
  - Domain – здесь будет вся доменная логика
    - Models – доменные модели
    - Repositories – репозитории доменных сущностей
  - Infrastructure – здесь храним различные инфраструктурные вещи
  - Program.cs – точка входа в приложение.

Опишем модели

Опишем наши Value Objects

В каталоге Domain / Models создадим класс Email

```
/// <summary>
/// Value Object representing an email address
/// </summary>
public sealed record Email
{
    public string Value { get; }

    public Email(string email)
    {
        if (string.IsNullOrWhiteSpace(email))
            throw new ArgumentException("Email cannot be empty", nameof(email));

        if (!email.Contains('@'))
            throw new ArgumentException("Email must contain @ symbol", nameof(email));

        Value = email;
    }

    public override string ToString()
    {
        return Value;
    }
}
```

Опишем наши Value Objects

В каталоге Domain / Models создадим класс MobilePhone

```
/// <summary>
/// Value Object representing a mobile phone number
/// </summary>
public sealed record MobilePhone
{
    public string Value { get; }

    public MobilePhone(string phone)
    {
        if (string.IsNullOrWhiteSpace(phone))
            throw new ArgumentException("Phone number cannot be empty", nameof(phone));

        // Simple validation: phone should contain only digits, spaces, +, -, (), or be empty after cleaning
        var cleaned :string = phone
            .Replace(" ", "")
            .Replace("-", "")
            .Replace("(", "")
            .Replace(")", "")
            .Replace("+", "");

        if (cleaned.Length < 7)
            throw new ArgumentException("Phone number is too short", nameof(phone));

        if (!cleaned.All(char.IsDigit))
            throw new ArgumentException(
                "Phone number must contain only digits and common formatting characters",
                nameof(phone));

        Value = phone;
    }

    public override string ToString()
    {
        return Value;
    }
}
```

Опишем модель клиента

В каталоге Domain / Models создадим класс Customer

```
/// <summary>
/// Entity representing a customer
/// </summary>
public class Customer
{
    public Guid Id { get; }
    public string Name { get; }
    public Email Email { get; }
    public MobilePhone Phone { get; }

    public Customer(Guid id, string name, Email email, MobilePhone phone)
    {
        if (id == Guid.Empty)
            throw new ArgumentException("Customer ID cannot be empty", nameof(id));

        if (string.IsNullOrWhiteSpace(name))
            throw new ArgumentException("Customer name cannot be empty", nameof(name));

        Id = id;
        Name = name;
        Email = email ?? throw new ArgumentNullException(nameof(email));
        Phone = phone ?? throw new ArgumentNullException(nameof(phone));
    }
}
```

Опишем факт продажи

В каталоге Domain / Models создадим класс Sale

```
/// <summary>
/// Value Object representing a sale of a car
/// </summary>
public sealed record Sale
{
    public DateTimeOffset SaleDate { get; }
    public Customer Customer { get; }

    public Sale(DateTimeOffset saleDate, Customer customer)
    {
        SaleDate = saleDate;
        Customer = customer ?? throw new ArgumentNullException(nameof(customer));
    }
}
```

Опишем факт резерва

В каталоге Domain / Models создадим класс Reservation

```
/// <summary>
/// Value Object representing a reservation of a car
/// </summary>
public sealed record Reservation
{
    public DateTimeOffset ReservationDate { get; }
    public Customer Customer { get; }
    public DateTimeOffset ExpirationDate { get; }

    public bool IsActive => DateTimeOffset.UtcNow <= ExpirationDate;

    public Reservation(DateTimeOffset reservationDate, Customer customer, DateTimeOffset expirationDate)
    {
        if (expirationDate <= reservationDate)
            throw new ArgumentException("Expiration date must be after reservation date", nameof(expirationDate));

        ReservationDate = reservationDate;
        Customer = customer ?? throw new ArgumentNullException(nameof(customer));
        ExpirationDate = expirationDate;
    }
}
```

Опишем автомобиль

В каталоге Domain / Models
создадим класс Car

```
/// <summary>
/// Entity representing a car in the AutoMarket
/// </summary>
public sealed class Car
{
    public string Vin { get; }
    public Reservation? Reservation { get; private set; }
    public Sale? Sale { get; private set; }

    public Car(string vin)
    {
        if (string.IsNullOrWhiteSpace(vin))
            throw new ArgumentException("VIN cannot be empty", nameof(vin));

        Vin = vin;
    }

    public void Reserve(Reservation reservation)
    {
        if (Sale is not null)
            throw new InvalidOperationException("Cannot reserve a sold car");

        if (Reservation is not null && Reservation.IsActive)
            throw new InvalidOperationException("Car is already reserved");

        Reservation = reservation;
    }

    public void Sell(Sale sale)
    {
        if (Sale is not null)
            throw new InvalidOperationException("Car is already sold");

        if (Reservation is not null && Reservation.IsActive && Reservation.Customer.Id != sale.Customer.Id)
            throw new InvalidOperationException("Cannot sell car to different customer when reserved");

        Sale = sale;
    }
}
```

Модели описаны – описываем репозиторий

Опишем репозиторий клиентов

Создадим интерфейс ICustomerRepository в каталоге Domain / Repositories

```
public interface ICustomerRepository
{
    IReadOnlyList<Customer> List();
    Customer? GetById(Guid id);
    void Save(Customer customer);
}
```

Реализация репозитория клиентов

Для хранения будем использовать JSON-файл, чтобы не разбираться со сложными ORM на данном этапе.

Начнем с добавления DTO для хранения клиентов.

Создадим класс CustomerDto в каталоге Infrastructure / Dtos

```
internal sealed record CustomerDto(
    Guid Id,
    string Name,
    string Email,
    string Phone
);
```

Реализация репозитория клиентов

Чтобы самостоятельно не писать и не поддерживать код маппинга между доменными сущностями и DTO, можно использовать специализированные инструменты. Один из них называется Mapperly. Добавим его в наш проект указанной командой.

```
$  dotnet add .\AutoMarketApp\AutoMarketApp.csproj package Riok.Mapperly
```

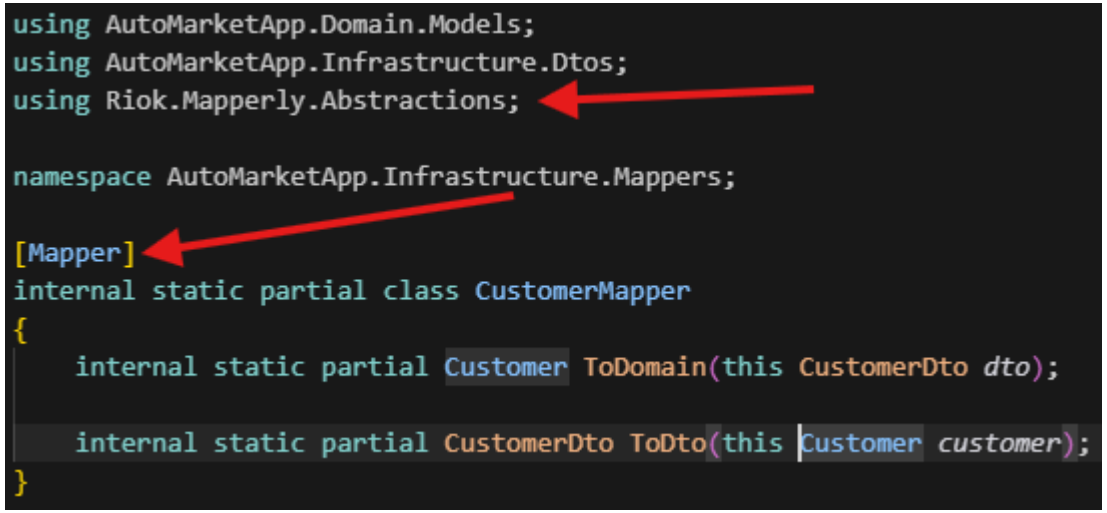
Реализация репозитория клиентов

Добавим класс-заглушку для генерации маппера клиентов.

Для этого в каталоге Infrastructure / Mappers создадим класс CustomerMapper.

Обратите внимание на пространство имен и атрибут, выделенные стрелками.

Они отвечают за то, чтобы активировать механизм автоматической генерации код маппинга.



Реализация репозитория клиентов

Теперь создадим класс JsonCustomerRepository в каталоге Infrastructure

```
internal sealed class JsonCustomerRepository
{
    private readonly string _filePath;
    private readonly List<CustomerDto> _customers;

    public JsonCustomerRepository(string filePath = "customers.json")
    {
        _filePath = filePath;
        _customers = LoadFromFile();
    }

    private List<CustomerDto> LoadFromFile()
    {
        if (!File.Exists(_filePath))
            return new List<CustomerDto>();

        try
        {
            var json :string = File.ReadAllText(_filePath);
            return JsonSerializer.Deserialize<List<CustomerDto>>(json) ?? new List<CustomerDto>();
        }
        catch
        {
            return new List<CustomerDto>();
        }
    }

    private void SaveToFile()
    {
        var json :string = JsonSerializer.Serialize(_customers, new JsonSerializerOptions { WriteIndented = true });
        File.WriteAllText(_filePath, contents: json);
    }
}
```

Реализация репозитория клиентов

В созданный класс добавим реализацию добавленного ранее интерфейса

```
internal sealed class JsonCustomerRepository : ICustomerRepository
{
    private readonly string _filePath;
    private readonly List<CustomerDto> _customers;

    public JsonCustomerRepository(string filePath = "customers.json")
    {
        _filePath = filePath;
        _customers = LoadFromFile();
    }

    public IReadOnlyList<Customer> List()
    {
        return _customers.Select(CustomerMapper.ToDomain).ToList();
    }

    public Customer? GetById(Guid id)
    {
        var dto = _customers.FirstOrDefault(c :CustomerDto => c.Id == id);
        if (dto == null)
            return null;

        return dto.ToDomain();
    }

    public void Save(Customer customer)
    {
        var dto = _customers.FirstOrDefault(c :CustomerDto => c.Id == customer.Id);
        var customerDto = customer.ToDto();

        if (dto == null)
        {
            _customers.Add(customerDto);
        }
        else
        {
            var index = _customers.IndexOf(dto);
            _customers[index] = customerDto;
        }

        SaveToFile();
    }
```

Опишем репозиторий автомобилей

Создадим интерфейс ICarRepository в каталоге Domain / Repositories

```
public interface ICarRepository
{
    IReadOnlyList<Car> List();
    Car? GetByVin(string vin);
    void Save(Car car);
}
```

Реализация репозитория автомобилей

Для хранения аналогично с клиентами будем использовать JSON-файл. Для начала добавим соответствующие DTO.

Создадим файл CarDto.cs в каталоге Infrastructure / Dtos

```
internal sealed record CarDto(
    string Vin,
    ReservationDto? Reservation = null,
    SaleDto? Sale = null
);

internal sealed record ReservationDto(
    DateTimeOffset ReservationDate,
    Guid CustomerId,
    DateTimeOffset ExpirationDate
);

internal sealed record SaleDto(
    DateTimeOffset SaleDate,
    Guid CustomerId
);
```

Реализация репозитория автомобилей

Теперь добавим класс для маппинга.

Создадим класс CarMapper в каталоге Infrastructure / Mappers.

Обратите внимание на использование генератора маппингов.

```
[Mapper]
internal static partial class CarMapper
{
    private static partial SaleDto ToDto(this Sale sale);
    [MapperIgnoreSource(nameof(Reservation.IsActive))]
    private static partial ReservationDto ToDto(this Reservation reservation);
    [MapperIgnoreSource(nameof(SaleDto.CustomerId))]
    private static partial Sale ToDomain(this SaleDto dto, Customer customer);
    [MapperIgnoreSource(nameof(ReservationDto.CustomerId))]
    private static partial Reservation ToDomain(this ReservationDto dto, Customer customer);
}
```

Реализация репозитория автомобилей

В тот же класс добавим методы для маппинга самого автомобиля.

Здесь маппинг уже довольно сложный – поэтому нам проще написать его самостоятельно.

```
[Mapper]
internal static partial class CarMapper
{
    internal static Car ToDomain(this CarDto dto, ICustomerRepository customerRepository)
    {
        var car = new Car(dto.Vin);

        if (dto.Reservation != null)
        {
            var customer = customerRepository.GetById(dto.Reservation.CustomerId)
                ?? throw new InvalidOperationException("Customer not found");
            car.Reserve(dto.Reservation.ToDomain(customer));
        }

        if (dto.Sale != null)
        {
            var customer = customerRepository.GetById(dto.Sale.CustomerId)
                ?? throw new InvalidOperationException("Customer not found");
            car.Sell(dto.Sale.ToDomain(customer));
        }

        return car;
    }

    public static CarDto ToDto(this Car car)
    {
        var dto = new CarDto(Vin: car.Vin);

        if (car.Reservation != null)
        {
            dto = dto with
            {
                Reservation = car.Reservation.ToDto()
            };
        }

        if (car.Sale != null)
        {
            dto = dto with
            {
                Sale = car.Sale.ToDto()
            };
        }

        return dto;
    }
```

Реализация репозитория автомобилей

Наконец, добавляем реализацию самого репозитория.

Создаем класс JsonCarRepository в каталоге Infrastructure

```
internal sealed class JsonCarRepository
{
    private readonly string _filePath;
    private readonly List<CarDto> _cars;
    private readonly ICustomerRepository _customerRepository;

    public JsonCarRepository(ICustomerRepository customerRepository, string filePath = "cars.json")
    {
        _customerRepository = customerRepository;
        _filePath = filePath;
        _cars = LoadFromFile();
    }

    private List<CarDto> LoadFromFile()
    {
        if (!File.Exists(_filePath))
            return new List<CarDto>();

        try
        {
            var json :string = File.ReadAllText(_filePath);
            return JsonSerializer.Deserialize<List<CarDto>>(json) ?? new List<CarDto>();
        }
        catch
        {
            return new List<CarDto>();
        }
    }

    private void SaveToFile()
    {
        var json :string = JsonSerializer.Serialize(_cars, new JsonSerializerOptions { WriteIndented = true });
        File.WriteAllText(_filePath, contents: json);
    }
}
```

Реализация репозитория автомобилей

И добавляем в созданный класс реализацию интерфейса репозитория.

```
internal sealed class JsonCarRepository : ICarRepository
{
    private readonly string _filePath;
    private readonly List<CarDto> _cars;
    private readonly ICustomerRepository _customerRepository;

    public JsonCarRepository(ICustomerRepository customerRepository, string filePath = "cars.json")
    {
        _customerRepository = customerRepository;
        _filePath = filePath;
        _cars = LoadFromFile();
    }

    public IReadOnlyList<Car> List()
    {
        return _cars.Select(dto => dto.ToDomain(_customerRepository)).ToList();
    }

    public Car? GetByVin(string vin)
    {
        var dto = _cars.FirstOrDefault(c => c.Vin == vin);
        if (dto == null)
            return null;

        return dto.ToDomain(_customerRepository);
    }

    public void Save(Car car)
    {
        var dto = car.ToDto();
        var existingIndex = _cars.FindIndex(c => c.Vin == car.Vin);

        if (existingIndex < 0)
        {
            _cars.Add(dto);
        }
        else
        {
            _cars[existingIndex] = dto;
        }

        SaveToFile();
    }
```

Прикладные сервисы

Сервис для работы с клиентами

Добавим сервис для работы с клиентами. Для этого добавим класс CustomerService в каталоге Application / Services

```
public sealed class CustomerService
{
    private readonly ICustomerRepository _customerRepository;

    public CustomerService(ICustomerRepository customerRepository)
    {
        _customerRepository = customerRepository;
    }

    public IReadOnlyList<Customer> List()
    {
        return _customerRepository.List();
    }

    public Customer Add(string name, Email email, MobilePhone phone)
    {
        var customer = new Customer(id: Guid.NewGuid(), name, email, phone);
        _customerRepository.Save(customer);
        return customer;
    }
}
```

Сервис для работы с автомобилями

Добавим сервис для работы с автомобилями.

Для этого добавим класс CarService в каталог Application / Services

```
public sealed class CarService
{
    private readonly ICarRepository _carRepository;

    public CarService(ICarRepository carRepository)
    {
        _carRepository = carRepository;
    }

    public Car Add(string vin)
    {
        var existingCar = _carRepository.GetByVin(vin);
        if (existingCar != null)
            throw new InvalidOperationException($"Car with VIN {vin} already exists");

        var car = new Car(vin);
        _carRepository.Save(car);
        return car;
    }

    public IReadOnlyList<Car> List()
    {
        return _carRepository.List();
    }
}
```

Сервис для работы с автомобилями

Дополним созданный сервис, добавив туда методы для резервирования и продажи автомобилей

```
public sealed class CarService
{
    private readonly ICarRepository _carRepository;
    private readonly ICustomerRepository _customerRepository;

    public CarService(ICarRepository carRepository, ICustomerRepository customerRepository)
    {
        _carRepository = carRepository;
        _customerRepository = customerRepository;
    }

    // ... existing methods ...

    public void Reserve(string vin, Guid customerId, DateTimeOffset expirationDate)
    {
        var car = _carRepository.GetByVin(vin)
            ?? throw new InvalidOperationException($"Car with VIN {vin} not found");

        var customer = _customerRepository.GetById(customerId)
            ?? throw new InvalidOperationException($"Customer with ID {customerId} not found");

        var reservationDate = DateTimeOffset.UtcNow;
        var reservation = new Reservation(reservationDate, customer, expirationDate);

        car.Reserve(reservation);
        _carRepository.Save(car);
    }

    public void Sell(string vin, Guid customerId)
    {
        var car = _carRepository.GetByVin(vin)
            ?? throw new InvalidOperationException($"Car with VIN {vin} not found");

        var customer = _customerRepository.GetById(customerId)
            ?? throw new InvalidOperationException($"Customer with ID {customerId} not found");

        var saleDate = DateTimeOffset.UtcNow;
        var sale = new Sale(saleDate, customer);

        car.Sell(sale);
        _carRepository.Save(car);
    }
}
```

Объединяем компоненты

DI-контейнер

Мы сделаем простое приложение с консольным интерфейсом.

Для начала используем изученный ранее DI-контейнер для построения компонентов приложения.

Для этого добавим соответствующий NuGet-пакет в приложение.

```
$  dotnet add package Microsoft.Extensions.DependencyInjection
```

DI-контейнер

После добавления пакета настроим DI-контейнер и получим из него прикладные сервисы.

Для этого в Program.cs добавим соответствующий код.

```
var services = new ServiceCollection();

// Register repositories
services.AddSingleton<ICustomerRepository, JsonCustomerRepository>();
services.AddSingleton<ICarRepository, JsonCarRepository>();

// Register application services
services.AddSingleton<CustomerService>();
services.AddSingleton<CarService>();

var serviceProvider = services.BuildServiceProvider();

var customerService = serviceProvider.GetRequiredService<CustomerService>();
var carService = serviceProvider.GetRequiredService<CarService>();
```

Обработчики команд

Наше приложение будет уметь обрабатывать следующие команды:

- customers – для вывода списка клиентов
- add customer – для добавления нового клиента
- cars – для вывода списка автомобилей
- add car – для добавления нового автомобиля
- reserve – для резервирования автомобиля
- sell – для продажи автомобиля

Каждая из команд будет реализована при помощи своего собственного обработчика

Обработчики команд

Начнем с вывода списка клиентов.

Добавим соответствующий метод в Program.cs.

```
void ListCustomers()
{
    var customers :IReadOnlyList<Customer> = customerService.List();
    if (customers.Count == 0)
    {
        Console.WriteLine("No customers found.");
        return;
    }

    Console.WriteLine("Customers:");
    foreach (var customer in customers)
    {
        Console.WriteLine($"  ID: {customer.Id}");
        Console.WriteLine($"  Name: {customer.Name}");
        Console.WriteLine($"  Email: {customer.Email}");
        Console.WriteLine($"  Phone: {customer.Phone}");
        Console.WriteLine();
    }
}
```

Обработчики команд

Далее добавляем метод для добавления покупателя.

```
void AddCustomer()
{
    Console.Write("Name: ");
    var name = Console.ReadLine()?.Trim();
    if (string.IsNullOrWhiteSpace(name))
    {
        Console.WriteLine("Name cannot be empty.");
        return;
    }

    Console.Write("Email: ");
    var emailInput :string? = Console.ReadLine()?.Trim();
    if (string.IsNullOrWhiteSpace(emailInput))
    {
        Console.WriteLine("Email cannot be empty.");
        return;
    }

    Console.Write("Phone: ");
    var phoneInput :string? = Console.ReadLine()?.Trim();
    if (string.IsNullOrWhiteSpace(phoneInput))
    {
        Console.WriteLine("Phone cannot be empty.");
        return;
    }

    try
    {
        var email = new Email(emailInput);
        var phone = new MobilePhone(phoneInput);
        var customer = customerService.Add(name, email, phone);
        Console.WriteLine($"Customer added successfully. ID: {customer.Id}");
    }
    catch (ArgumentException ex)
    {
        Console.WriteLine($"Invalid input: {ex.Message}");
    }
}
```

Обработчики команд

Метод для добавления автомобиля

```
void AddCar()
{
    Console.Write("VIN: ");
    var vin :string? = Console.ReadLine()?.Trim();
    if (string.IsNullOrWhiteSpace(vin))
    {
        Console.WriteLine("VIN cannot be empty.");
        return;
    }

    try
    {
        var car = carService.Add(vin);
        Console.WriteLine($"Car added successfully. VIN: {car.Vin}");
    }
    catch (InvalidOperationException ex)
    {
        Console.WriteLine($"Error: {ex.Message}");
    }
}
```

Обработчики команд

Метод для вывода списка автомобилей

```
void ListCars()
{
    var cars :IReadOnlyList<Car> = carService.List();
    if (cars.Count == 0)
    {
        Console.WriteLine("No cars found.");
        return;
    }

    Console.WriteLine("Cars:");
    foreach (var car in cars)
    {
        Console.WriteLine($"  VIN: {car.Vin}");

        if (car.Sale != null)
        {
            Console.WriteLine("  Status: Sold");
            Console.WriteLine($"  Customer: {car.Sale.Customer.Name} ({car.Sale.Customer.Email})");
            Console.WriteLine($"  Sale Date: {car.Sale.SaleDate:yyyy-MM-dd}");
        }
        else if (car.Reservation != null && car.Reservation.IsActive)
        {
            Console.WriteLine("  Status: Reserved");
            Console.WriteLine($"  Customer: {car.Reservation.Customer.Name} ({car.Reservation.Customer.Email})");
            Console.WriteLine($"  Reserved Until: {car.Reservation.ExpirationDate:yyyy-MM-dd}");
        }
        else
        {
            Console.WriteLine("  Status: Available");
        }

        Console.WriteLine();
    }
}
```

Обработчики команд

Метод для резервирования

```
void ReserveCar()
{
    Console.Write("Car VIN: ");
    var vin :string? = Console.ReadLine()?.Trim();
    if (string.IsNullOrWhiteSpace(vin))
    {
        Console.WriteLine("VIN cannot be empty.");
        return;
    }

    Console.Write("Customer ID: ");
    var customerIdInput :string? = Console.ReadLine()?.Trim();
    if (string.IsNullOrWhiteSpace(customerIdInput) || !Guid.TryParse(customerIdInput, out var customerId :Guid))
    {
        Console.WriteLine("Invalid customer ID.");
        return;
    }

    Console.Write("Expiration date (YYYY-MM-DD): ");
    var dateInput :string? = Console.ReadLine()?.Trim();
    if (string.IsNullOrWhiteSpace(dateInput) || !DateTimeOffset.TryParse(dateInput, out var expirationDate))
    {
        Console.WriteLine("Invalid date format.");
        return;
    }

    carService.Reserve(vin, customerId, expirationDate);
    Console.WriteLine($"Car {vin} reserved successfully.");
}
```

Обработчики команд

Метод для продажи

```
void SellCar()
{
    Console.Write("Car VIN: ");
    var vin :string? = Console.ReadLine()?.Trim();
    if (string.IsNullOrWhiteSpace(vin))
    {
        Console.WriteLine("VIN cannot be empty.");
        return;
    }

    Console.Write("Customer ID: ");
    var customerIdInput :string? = Console.ReadLine()?.Trim();
    if (string.IsNullOrWhiteSpace(customerIdInput) || !Guid.TryParse(customerIdInput, out var customerId :Guid))
    {
        Console.WriteLine("Invalid customer ID.");
        return;
    }

    carService.Sell(vin, customerId);
    Console.WriteLine($"Car {vin} sold successfully.");
}
```

Основной цикл обработки команд

Основной цикл обработки команд

Теперь добавим основной каркас, который будет считывать ввод от пользователя и запускать обработчики.

Добавим соответствующий код сразу после кода построения нашим сервисов и до кода обработчиков.

```
var serviceProvider = services.BuildServiceProvider();

var customerService = serviceProvider.GetRequiredService<CustomerService>();
var carService = serviceProvider.GetRequiredService<CarService>();

Console.WriteLine("AutoMarket Application");
Console.WriteLine("Available commands: customers, add customer, add car, cars, reserve, sell, exit");
Console.WriteLine();

while (true)
{
    Console.Write("> ");
    var input :string? = Console.ReadLine()?.Trim();

    if (string.IsNullOrWhiteSpace(input))
        continue;

    var parts :string[] = input.Split(' ', StringSplitOptions.RemoveEmptyEntries);
    var command :string = parts[0].ToLower();

    try
    {
        // handlers will be called here
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Error: {ex.Message}");
    }

    Console.WriteLine();
}
```

Основной цикл обработки команд

И, наконец, добавим вызов самих обработчиков.

```
var parts :string[] = input.Split(' ', StringSplitOptions.RemoveEmptyEntries);
var command :string = parts[0].ToLower();

try
{
    switch (command)
    {
        case "customers":
            ListCustomers();
            break;

        case "add" when parts.Length >= 2 && parts[1].ToLower() == "customer":
            AddCustomer();
            break;

        case "add" when parts.Length >= 2 && parts[1].ToLower() == "car":
            AddCar();
            break;

        case "cars":
            ListCars();
            break;

        case "reserve":
            ReserveCar();
            break;

        case "sell":
            SellCar();
            break;

        case "exit":
            return;

        default:
            Console.WriteLine($"Unknown command: {input}");
            Console.WriteLine("Available commands: customers, add customer, add car, cars, reserve, sell, exit");
            break;
    }
}
catch (Exception ex)
{
    Console.WriteLine($"Error: {ex.Message}");
}
```

Запустим проект

```
$ dotnet run --project .\AutoMarketApp\
```

Проверочные вопросы

- Подойдет ли описанная нами модель для другой предметной области? Например, для описания сервисного центра для автомобилей?