**Министерство транспорта Российской Федерации**

**Федеральное государственное автономное образовательное  
учреждение высшего образования  
«РУТ (МИИТ»**

**Институт транспортной техники и систем управления**

**Кафедра «Управление и защита информации»**

ПРАКТИЧЕСКАЯ РАБОТА

по дисциплине

**«Основы построения защищенных баз данных»**

**на тему  
«Создание ORM для реляционной БД «Компания по услугам связи»»**

Выполнил: ст. гр. ТКИ–511  
Лисицына А.В.  
Проверил: Васильева М.А.

Москва 2021

Содержание

[Задание 3](#_Toc91584859)

[Описание 3](#_Toc91584860)

[Листинг программы 3](#_Toc91584861)

[Результаты работы программы 13](#_Toc91584862)

[Результаты прохождения тестов 16](#_Toc91584863)

# Задание

Разработать ORM на языке C# для хранения экземпляров классов базы данных компании по услугам связи. Компания предоставляет услуги связи. Каждый абонент обслуживается по выбранному тарифу. Реализовать принцип CRUD.

# Описание

Было создано одно общее решение CommunicationServicesORM, в составе которого имеются проекты. Проект Demo, содержащий главный метод, через который запускается решение (является консольным приложением). Проект Domain необходим для реализации классов сущностей реализуемой базы данных. Выбраны сущности *абонент* (subscriber) и *тариф* (tariff). DataAccess проект, содержащий классы маппинга, репозитория и фабрики сессий. В Staff представлены классы для интерфейса IEnumerable, представляющим собой перечислитель, а так же классы для определения и сокращения пустых строк. В завершении были добавлены модульные тесты Demo.Tests на сущностный класс.

# Листинг программы

1) Demo.Program:

namespace Demo

{

using System;

using System.Linq;

using DataAccess;

using DataAccess.Repositories;

using Domain;

internal class Program

{

/// <summary>

/// Главный метод

/// </summary>

/// <param name="args"></param>

private static void Main()

{

var subscriber = new Subscriber(1, "Рябова", "Дария", "1990.04.30", "Кирилловна");

var subscriber1 = new Subscriber(2, "Уварова", "Вероника", "1997.06.04");

var tariff = new Tariff(1, "Оптимальный", subscriber);

var tariff1 = new Tariff(2, "Стандартный", subscriber1);

Console.WriteLine($"{tariff} {subscriber}");

var settings = new Settings();

settings.AddDatabaseServer(@"LAPTOP-57BV0I6T\SQLEXPRESS");

settings.AddDatabaseName("CommunicationServ");

using var sessionFactory = Configurator.GetSessionFactory(settings, showSql: true);

using (var session = sessionFactory.OpenSession())

{

session.Save(tariff);

session.Save(tariff1);

session.Save(subscriber);

session.Save(subscriber1);

session.Flush();

}

using (var session = sessionFactory.OpenSession())

{

var repoTariff = new TariffRepository();

Console.WriteLine("All tariffs:");

repoTariff.GetAll(session)

.ToList().ForEach(Console.WriteLine);

Console.WriteLine(new string('-', 25));

var repoSubscriber = new SubscriberRepository();

Console.WriteLine("All sibscribers:");

repoSubscriber.GetAll(session)

.ToList().ForEach(Console.WriteLine);

Console.WriteLine(new string('-', 25));

}

}

}

}

2) Domain.Subscriber:

namespace Domain

{

using System;

using System.Collections.Generic;

using Staff.Extensions;

/// <summary>

/// Абонент

/// </summary>

public class Subscriber : IEquatable<Subscriber>

{

/// <summary>

/// Инициализирует новый экземпляр класса <see cref="Subscriber"/>

/// </summary>

/// <param name="id"> Идентефикатор </param>

/// <param name="lastName"> Фамилия </param>

/// <param name="firstName"> Имя </param>

/// <param name="patronymic"> Отчество </param>

/// <param name="dateOfBirth"> День рождения </param>

/// <exception cref="ArgumentOutOfRangeException">

/// В случае если <paramref name="lastName"/> или <paramref name="firstName"/> или <paramref name="dateOfBirth"/> <see langword="null"/>,

/// пустая строка или строка, содержащая только пробельные символы.

/// </exception>

public Subscriber(int id, string lastName, string firstName, string dateOfBirth, string patronymic = null)

{

this.ID = id;

this.LastName = lastName.TrimOrNull() ?? throw new ArgumentOutOfRangeException(nameof(lastName));

this.FirstName = firstName.TrimOrNull() ?? throw new ArgumentOutOfRangeException(nameof(firstName));

this.DateOfBirth = dateOfBirth.TrimOrNull() ?? throw new ArgumentOutOfRangeException(nameof(dateOfBirth));

this.Patronymic = patronymic.TrimOrNull();

}

/// <summary>

/// Инициализирует новый экземпляр класса <see cref="Subscriber"/>

/// </summary>

[Obsolete("For ORM", true)]

protected Subscriber()

{

}

/// <summary>

/// Уникальный идентефикатор

/// </summary>

public virtual int ID { get; protected set; }

/// <summary>

/// Фамилия

/// </summary>

public virtual string LastName { get; protected set; }

/// <summary>

/// Имя

/// </summary>

public virtual string FirstName { get; protected set; }

/// <summary>

/// День рождения

/// </summary>

public virtual string DateOfBirth { get; protected set; }

/// <summary>

/// Отчество

/// </summary>

public virtual string Patronymic { get; protected set; }

/// <summary>

/// Полная информация об абоненте

/// </summary>

public virtual string FullData => $"{this.LastName} {this.FirstName[0]}. {this.DateOfBirth[0..10]}. {this.Patronymic?[0]}.".Trim();

/// <summary>

/// Множество тарифов

/// </summary>

public virtual ISet<Tariff> Tariffs { get; protected set; } = new HashSet<Tariff>();

/// <summary>

/// Метод, добавляющий тариф абоненту.

/// </summary>

/// <param name="tariff"> Добавляемый тариф. </param>

/// <returns>

/// Флаг успешности выполнения операции:

/// <see langword="true"/> – тариф был успешно добавлена,

/// <see langword="false"/> в противном случае.

/// </returns>

/// <exception cref="ArgumentNullException">

/// В случае если <paramref name="tariff"/> – <see langword="null"/>.

/// </exception>

public virtual bool AddTariff(Tariff tariff)

{

return tariff == null

? throw new ArgumentNullException(nameof(tariff))

: this.Tariffs.Add(tariff);

}

public override string ToString() => this.FullData;

public override bool Equals(object obj)

{

return !ReferenceEquals(null, obj) && (ReferenceEquals(this, obj) || this.Equals(obj as Subscriber));

}

public virtual bool Equals(Subscriber other)

{

return !ReferenceEquals(null, other) && (ReferenceEquals(this, other) || this.ID == other.ID);

}

public override int GetHashCode() => this.ID;

}

}

3) Domain.Tariff:

namespace Domain

{

using System;

using System.Collections.Generic;

using System.Linq;

using Staff.Extensions;

/// <summary>

/// Тариф

/// </summary>

public class Tariff

{

/// <summary>

/// Инициализирует новый экземпляр класса <see cref="Tariff"/>.

/// </summary>

/// <param name="id"> Идентификатор. </param>

/// <param name="tariffName"> Название. </param>

/// <param name="subscribers"> А,jytyns. </param>

public Tariff(int id, string tariffName, params Subscriber[] subscribers)

: this(id, tariffName, new HashSet<Subscriber>(subscribers))

{

}

/// <summary>

/// Инициализирует новый экземпляр класса <see cref="Tariff"/>.

/// </summary>

/// <param name="id"> Идентификатор. </param>

/// <param name="tariffName"> Название. </param>

/// <param name="subscribers"> Множество абонентов. </param>

public Tariff(int id, string tariffName, ISet<Subscriber> subscribers = null)

{

this.ID = id;

this.TariffName = tariffName.TrimOrNull() ?? throw new ArgumentOutOfRangeException(nameof(tariffName));

foreach (var subscriber in subscribers ?? Enumerable.Empty<Subscriber>())

{

this.Subscribers.Add(subscriber);

subscriber.AddTariff(this);

}

}

/// <summary>

/// Инициализирует новый экземпляр класса <see cref="Tariff"/>.

/// </summary>

[Obsolete("For ORM", true)]

protected Tariff()

{

}

/// <summary>

/// Идентификатор.

/// </summary>

public virtual int ID { get; protected set; }

/// <summary>

/// Название

/// </summary>

public virtual string TariffName { get; protected set; }

/// <summary>

/// Абоненты

/// </summary>

public virtual ISet<Subscriber> Subscribers { get; protected set; } = new HashSet<Subscriber>();

/// <inheritdoc/>

public override string ToString() => $"{this.TariffName} {this.Subscribers.Join()}".Trim();

}

}

4) DataAccess.Configurator:

namespace DataAccess

{

using System.Reflection;

using FluentNHibernate.Cfg;

using FluentNHibernate.Cfg.Db;

using NHibernate;

using NHibernate.Tool.hbm2ddl;

/// <summary>

/// Настройщик подключения к БД и поставщик фабрики сессий.

/// </summary>

public static class Configurator

{

/// <summary>

/// Конфигурация.

/// </summary>

private static FluentConfiguration fluentConfiguration;

/// <summary>

/// Генерирует фабрику сессий (<see cref="ISessionFactory"/>).

/// </summary>

/// <param name="settings"> Настройки. </param>

/// <param name="assembly"> Целевая сборка. </param>

/// <param name="showSql"> Показывать генерируемый SQL-код. </param>

/// <returns> Фабрику сессий. </returns>

public static ISessionFactory GetSessionFactory(

Settings settings,

Assembly assembly = null,

bool showSql = false)

{

return GetConfiguration(settings, assembly ?? Assembly.GetExecutingAssembly(), showSql)

.BuildSessionFactory();

}

/// <summary>

/// Возвращаем конфигурацию по правилам.

/// </summary>

/// <param name="settings"> Установки названия сервера БД и имени БД. </param>

/// <param name="assembly"> Целевая сборка. </param>

/// <param name="showSql"> Показывать генерируемый SQL-код. </param>

/// <returns> конфигурацию по правилам. </returns>

private static FluentConfiguration GetConfiguration(

Settings settings,

Assembly assembly,

bool showSql = false)

{

if (fluentConfiguration is null)

{

var databaseConfiguration = MsSqlConfiguration.MsSql2012.ConnectionString(

x => x

.Server(settings.GetDatabaseServer())

.Database(settings.GetDatabaseName())

.TrustedConnection());

if (showSql)

{

databaseConfiguration = databaseConfiguration.ShowSql().FormatSql();

}

fluentConfiguration = Fluently.Configure()

.Database(databaseConfiguration)

.Mappings(m => m.FluentMappings.AddFromAssembly(assembly))

.ExposeConfiguration(BuildSchema);

}

return fluentConfiguration;

}

/// <summary>

/// Метод, порождающий таблицы (если их не было в БД) по конфигурации.

/// </summary>

/// <remarks> Необходимо только для создания схемы БД из ничего. </remarks>

/// <param name="configuration"> Конфигурация ORM, содержащая правила отображения. </param>

private static void BuildSchema(NHibernate.Cfg.Configuration configuration)

{

new SchemaExport(configuration).Execute(true, true, false);

}

}

}

5) DataAccess.Settings:

namespace DataAccess

{

using System;

using System.Configuration;

public sealed class Settings

{

private string databaseServerName;

private string databaseName;

public void AddDatabaseServer(string serverName)

{

this.databaseServerName = serverName;

}

public string GetDatabaseServer()

{

return this.databaseServerName;

}

public void AddDatabaseName(string databaseName)

{

this.databaseName = databaseName;

}

public string GetDatabaseName()

{

return this.databaseName;

}

}

}

6) DataAccess.Mappings.SubscriberMap:

namespace DataAccess.Mappings

{

using FluentNHibernate.Mapping;

using Domain;

internal class SubscriberMap : ClassMap<Subscriber>

{

public SubscriberMap()

{

this.Table("Subscribers");

this.Id(x => x.ID);

this.Map(x => x.LastName)

.Not.Nullable();

this.Map(x => x.FirstName)

.Not.Nullable();

this.Map(x => x.DateOfBirth)

.Not.Nullable();

this.Map(x => x.Patronymic)

.Nullable();

this.HasManyToMany(x => x.Tariffs)

.Cascade.Delete();

}

}

}

7) DataAccess.Mappings.TariffMap:

namespace DataAccess.Mappings

{

using FluentNHibernate.Mapping;

using Domain;

internal class TariffMap : ClassMap<Tariff>

{

public TariffMap()

{

this.Table("Tariffs");

this.Id(x => x.ID);

this.Map(x => x.TariffName)

.Not.Nullable();

this.HasManyToMany(x => x.Subscribers)

.Cascade.Delete()

.Inverse();

}

}

}

8) DataAccess.Repositories.SubscriberRepository:

namespace DataAccess.Repositories

{

using System;

using System.Collections.Generic;

using System.Linq;

using DataAccess.Repositories.Abstraction;

using Domain;

using NHibernate;

public class SubscriberRepository : IRepository<Subscriber>

{

public bool Create<TEntity>(TEntity entity)

{

throw new NotImplementedException();

}

public bool Delete<TEntity>(TEntity entity)

{

throw new NotImplementedException();

}

public IQueryable<Subscriber> Filter(ISession session, System.Linq.Expressions.Expression<Func<Subscriber, bool>> predicate)

{

return this.GetAll(session)

.Where(predicate);

}

public Subscriber Find(ISession session, System.Linq.Expressions.Expression<Func<Subscriber, bool>> predicate)

{

return this.GetAll(session)

.FirstOrDefault(predicate);

}

public Subscriber Get(ISession session, int id) =>

session?.Get<Subscriber>(id);

public IQueryable<Subscriber> GetAll(ISession session) =>

session?.Query<Subscriber>();

public bool Update<TEntity>(TEntity oldEntity, TEntity newEntity)

{

throw new NotImplementedException();

}

}

}

9) DataAccess.Repositories.TariffRepository:

namespace DataAccess.Repositories

{

using System;

using System.Collections.Generic;

using System.Linq;

using System.Linq.Expressions;

using DataAccess.Repositories.Abstraction;

using Domain;

using NHibernate;

public class TariffRepository : IRepository<Tariff>

{

public Tariff Get(ISession session, int id) =>

session?.Get<Tariff>(id);

public Tariff Find(ISession session, Expression<Func<Tariff, bool>> predicate)

{

return this.GetAll(session).FirstOrDefault(predicate);

}

public IQueryable<Tariff> GetAll(ISession session) =>

session?.Query<Tariff>();

public IQueryable<Tariff> Filter(ISession session, Expression<Func<Tariff, bool>> predicate)

{

return this.GetAll(session).Where(predicate);

}

public bool Create<TEntity>(TEntity entity)

{

throw new NotImplementedException();

}

public bool Delete<TEntity>(TEntity entity)

{

throw new NotImplementedException();

}

public bool Update<TEntity>(TEntity oldEntity, TEntity newEntity)

{

throw new NotImplementedException();

}

}

}

10) DataAccess.Repositories.Abstraction.IRepository:

namespace DataAccess.Repositories.Abstraction

{

using System;

using System.Linq;

using System.Linq.Expressions;

using NHibernate;

public interface IRepository<TEntity>

{

TEntity Get(ISession session, int id);

TEntity Find(ISession session, Expression<Func<TEntity, bool>> predicate);

IQueryable<TEntity> GetAll(ISession session);

IQueryable<TEntity> Filter(ISession session, Expression<Func<TEntity, bool>> predicate);

bool Create<TEntity>(TEntity entity);

bool Delete<TEntity>(TEntity entity);

bool Update<TEntity>(TEntity oldEntity, TEntity newEntity);

}

}

11) Staff.Extension.IEnumerable:

namespace Staff.Extensions

{

using System.Collections.Generic;

public static class IEnumerableExtension

{

public static string Join<T>(this IEnumerable<T> collection, string separator = ", ") => string.Join(separator, collection);

}

}

12) Staff.Extension.StringExtensions:

namespace Staff.Extensions

{

public static class StringExtensions

{

public static bool IsNullOrEmpty(this string value) => string.IsNullOrEmpty(value);

public static string TrimOrNull(this string value)

{

var trimmedValue = value?.Trim(' ', '\n', '\t', '\r', '\0');

return trimmedValue.IsNullOrEmpty()

? null

: trimmedValue;

}

}

}

# Результаты работы программы

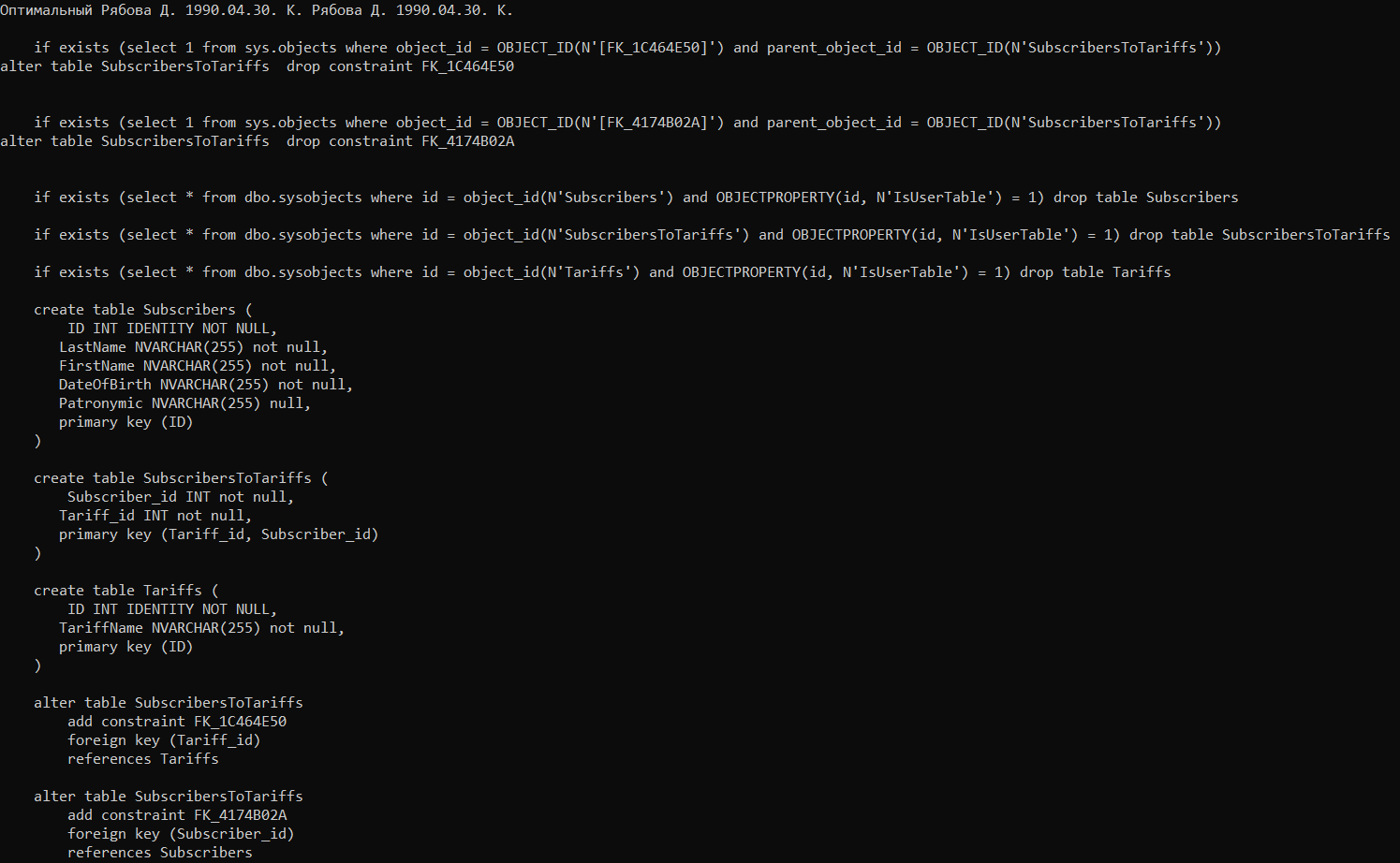


Рисунок 1

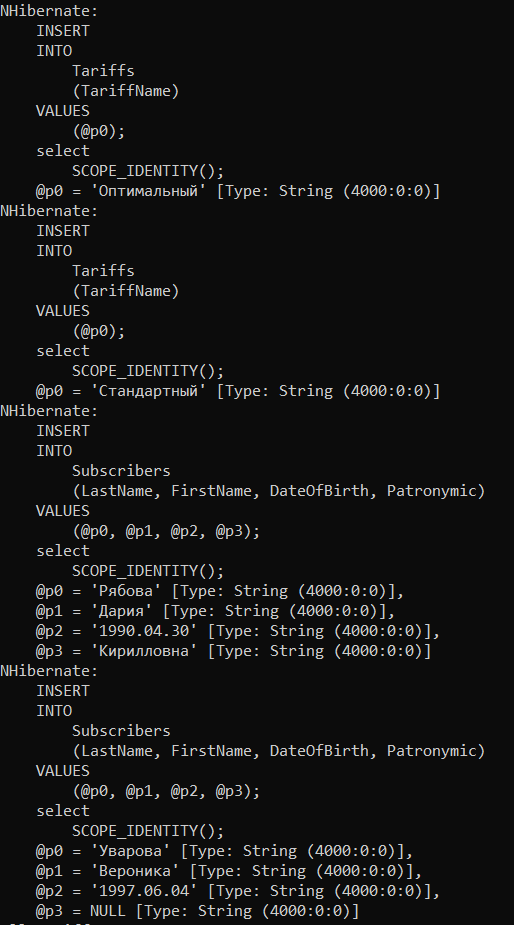


Рисунок 2

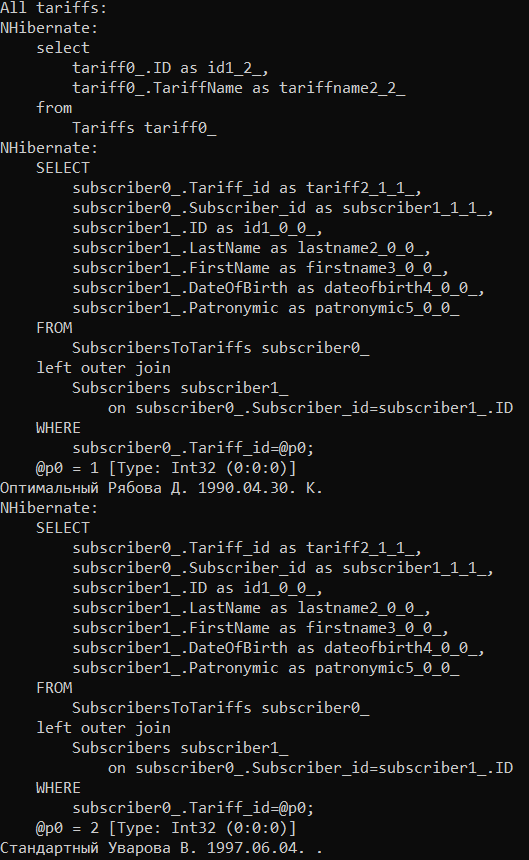


Рисунок 3

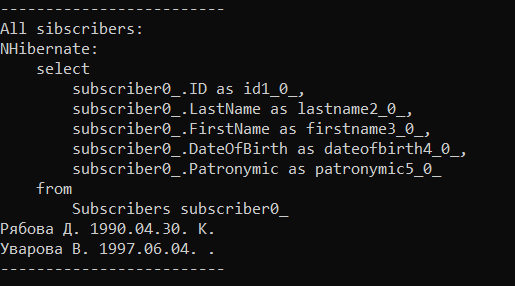


Рисунок 4

# Результаты прохождения тестов

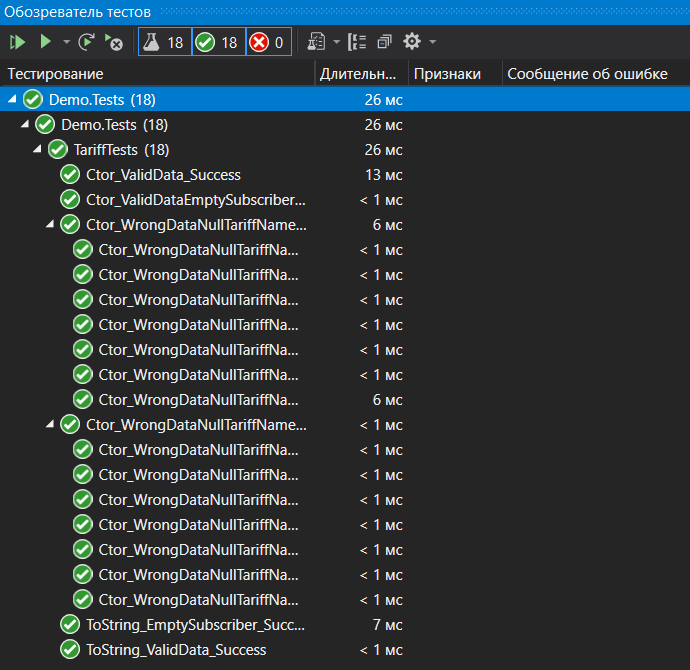


Рисунок 5