МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий и робототехники

Кафедра программного обеспечения информационных систем и технологий

**Отчет**

**по лабораторной работе № 1**

# **«**Объектно-Ориентированное Программирование. Абстракция**»**

по дисциплине: «КПО»

Выполнили: студенты гр. 10701221

Крупкин Тимофей Сергеевич

Кульгейко Степан Александрович

Принял: преподаватель Станкевич С.

Минск 2023

**Цель работы:**

Научиться грамотно анализировать предметную область и с помощью абстракции выделять существенные детали, на базе которых в дальнейшем проектируются классы и объекты будущей программной системы согласно методологии ООП, а также практически закрепить данные навыки при решении соответствующих задач (бизнес проблем).

**Постановка задачи**

Необходимо решить задачу с использованием методологии ООП. Для чего необходимо подобрать самостоятельно соответствующую проблемную (предметную/доменную) область, которая базируется на объектах и событиях реального мира (примеры соответствующих предметных областей приведены ниже). Спроектировать классы (собственные пользовательские типы данных) в языке C# для программного представления данных объектов и основной логики будущей программной системы. Система должна решать, как минимум, два полезных действия и иметь дополнительно следующие опции: − не менее 3 разнообразных классов предметной области; − не менее 5 атрибутов (состояния) и методов (поведения) в классе-сущности; − не менее 3 методов, которые реализуют бизнес-логику программы, в соответствующих функциональных классах; − хранить глобальные характеристики системы или характеристики уровня отдельных классов. На базе спроектированной программной системы реализовать программу и продемонстрировать её работоспособность

**Задание**

**Вариант 17**

Железнодорожный транспорт (Railway Transport). Имеются поезда, которые состоят из соответствующих вагонов. Необходимо подсчитать общую длину конкретного поезда, а также найти самый длинный (короткий) вагон

**1. Совместная работа с использованием службы размещения репозиториев GitHub**

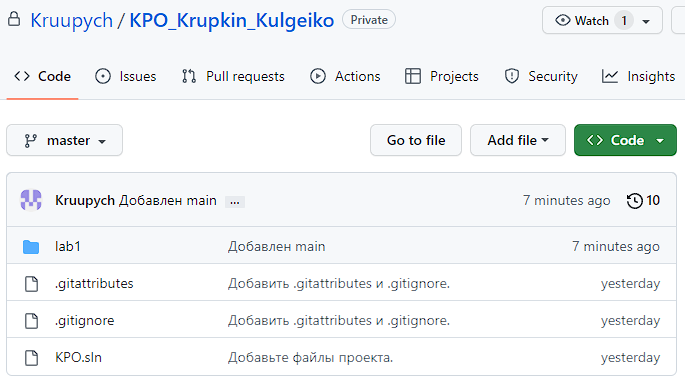
****

Рис. 1 – Файлы проекты размещены в репозитории GitHub

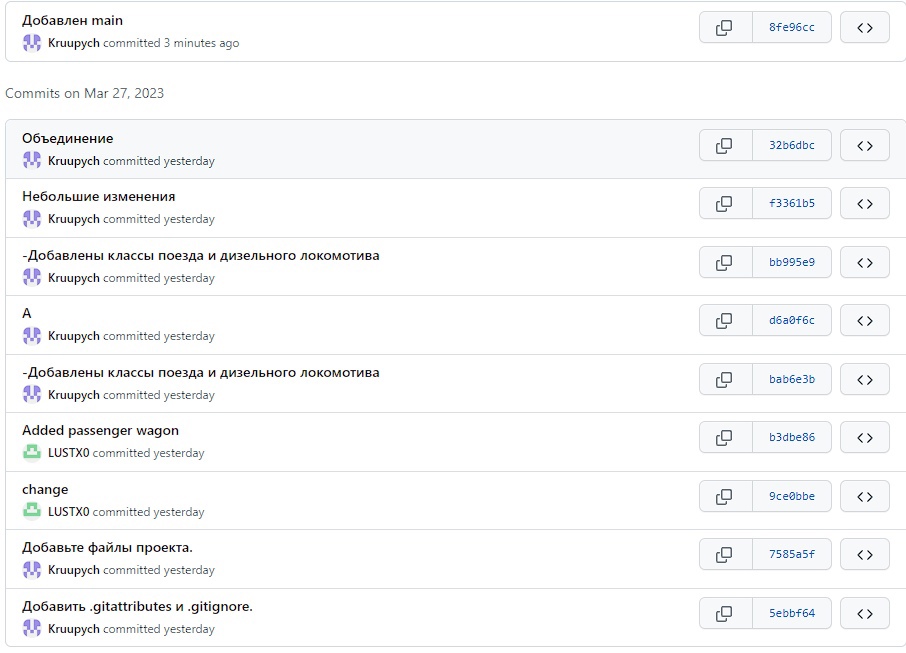


Рис. 2 – Список коммитов, сделанных за время выполнения задачи

**2.1. Разработка предметной области**

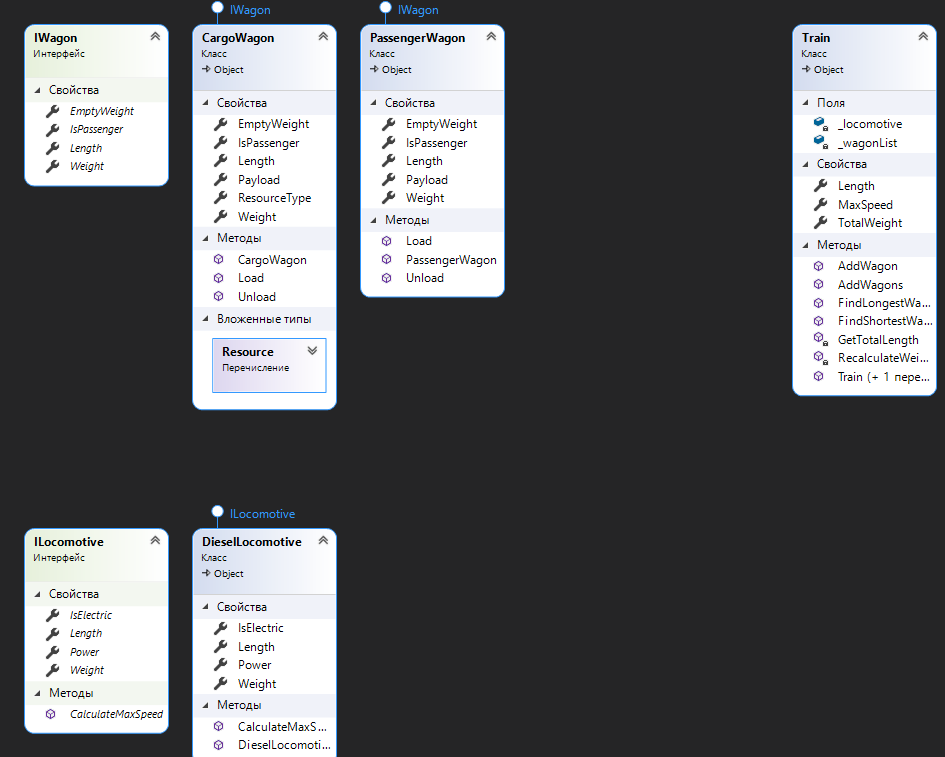
****

Рис. 3 – Диаграмма классов, созданная в среде Visual Studio

**2.2. Листинг программы**

Интерфейс Iwagon

namespace lab1

{

public interface IWagon

{

public double Length { get; }

public bool IsPassenger { get; }

public double EmptyWeight { get; }

public double Weight { get; }

}

}

Класс CargoWagon

namespace lab1

{

public class CargoWagon : IWagon

{

public enum Resource

{

Nothing, Coal, Wood, AgroCombine, Fuel, Animals

}

public double Length { get; private set; }

public bool IsPassenger { get; private set; }

public double EmptyWeight { get; private set; }

public double Weight { get; private set; }

public double Payload { get; private set; }

public Resource ResourceType { get; private set; }

public CargoWagon(double length, double emptyWeight, double payload)

{

Length = length;

EmptyWeight = emptyWeight;

IsPassenger = false;

Weight = emptyWeight;

Payload = payload;

}

public void Unload()

{

ResourceType = Resource.Nothing;

Weight = EmptyWeight;

}

public void Load(Resource resource, int loadPercentage)

{

ResourceType = resource;

loadPercentage = loadPercentage > 100? 100 : loadPercentage < 0 ? 0 : loadPercentage;

Weight += Payload \* loadPercentage / 100d;

}

public override string ToString()

{

return $"Грузовой вагон, везущий {ResourceType.ToString()}" +

$"\nДлина: {Length} метров" +

$"\nВес: {Weight.ToString("f2")} тонн";

}

}

}

Класс PassengerWagon

namespace lab1

{

public class PassengerWagon : IWagon

{

public double Length { get; private set; }

public bool IsPassenger { get; private set; }

public double EmptyWeight { get; private set; }

public double Weight { get; private set; }

public int Payload { get; private set; }

public PassengerWagon(double length, double emptyWeight, int maxPassengers)

{

Length = length;

EmptyWeight = emptyWeight;

IsPassenger = false;

Weight = emptyWeight;

Payload = maxPassengers;

}

public void Unload()

{

Weight = EmptyWeight;

}

public void Load(int passengersCount)

{

passengersCount = passengersCount > Payload ? Payload : passengersCount < 0 ? 0 : passengersCount;

Weight += passengersCount \* 0.07d;

}

}

}

Интерфейс ILocomotive

namespace lab1

{

public interface ILocomotive

{

double Length { get; }

double Power { get; }

double Weight { get; }

bool IsElectric { get; }

public double CalculateMaxSpeed(double totalWeight);

}

}

Класс DieselLocomotive

namespace lab1

{

public class DieselLocomotive : ILocomotive

{

public double Length { get; private set; }

public double Power { get; private set; }

public double Weight { get; private set; }

public bool IsElectric { get; private set; }

public DieselLocomotive(double length, double power, double weight)

{

Length = length;

Power = power;

Weight = weight;

IsElectric = false;

}

public double CalculateMaxSpeed(double totalWeight)

{

return Power / totalWeight \* 24d;

}

}

}

Класс Train

namespace lab1

{

public class Train

{

List<IWagon> \_wagonList;

ILocomotive \_locomotive;

public double TotalWeight { get; private set; }

public double Length { get => GetTotalLength(); }

public double MaxSpeed { get => \_locomotive.CalculateMaxSpeed(TotalWeight); }

public Train(ILocomotive locomotive)

{

\_locomotive = locomotive;

\_wagonList = new List<IWagon>();

RecalculateWeight();

}

public Train(ILocomotive locomotive, IEnumerable<IWagon> wagons) : this(locomotive)

{

AddWagons(wagons);

}

public void AddWagon(IWagon wagon)

{

\_wagonList.Add(wagon);

RecalculateWeight();

}

public void AddWagons(IEnumerable<IWagon> wagons)

{

foreach (var wagon in wagons)

{

\_wagonList.Add(wagon);

}

RecalculateWeight();

}

private void RecalculateWeight()

{

double weight = \_locomotive.Weight;

foreach (var wagon in \_wagonList)

{

weight += wagon.Weight;

}

TotalWeight = weight;

}

public IWagon FindShortestWagon()

{

if (\_wagonList.Count == 0) return null;

IWagon shortestWagon = \_wagonList[0];

foreach (var wagon in \_wagonList)

{

if (wagon.Length < shortestWagon.Length)

{

shortestWagon = wagon;

}

}

return shortestWagon;

}

public IWagon FindLongestWagon()

{

if (\_wagonList.Count == 0) return null;

IWagon longestWagon = \_wagonList[0];

foreach (var wagon in \_wagonList)

{

if (wagon.Length > longestWagon.Length)

{

longestWagon = wagon;

}

}

return longestWagon;

}

private double GetTotalLength()

{

double totalLength = \_locomotive.Length;

foreach (var wagon in \_wagonList)

{

totalLength += wagon.Length;

}

return totalLength;

}

}

}

Класс Program

namespace lab1

{

internal class Program

{

static void Main(string[] args)

{

//Инициализируем локомотив

DieselLocomotive dieselLocomotive = new DieselLocomotive(length: 15, power: 9100, weight: 100);

//Рандомно генерируем 20 вагонов

int wagonsCount = 20;

List<CargoWagon> cargoWagons = new();

Random random = new Random();

for (int i = 0; i < wagonsCount; i++)

{

//Генерируем параметры

double length = random.Next(11, 17);

double emptyWeight = random.Next(18, 31);

double payLoad = random.Next(45, 96);

CargoWagon.Resource resource = (CargoWagon.Resource)random.Next(1, 6);

int loadPercentage = random.Next(1, 101);

//Инициализируем вагон

CargoWagon wagon = new CargoWagon(length, emptyWeight, payLoad);

//Загружаем вагон

wagon.Load(resource, loadPercentage);

//Добавляем в список

cargoWagons.Add(wagon);

}

//Инициализируем поезд

Train train = new Train(dieselLocomotive, cargoWagons);

Console.WriteLine("Длина поезда: " + train.Length + "м");

Console.WriteLine("\nСамый длинный вагон:\n" + train.FindLongestWagon().ToString());

Console.WriteLine("\nСамый короткий вагон:\n" + train.FindShortestWagon().ToString());

Console.WriteLine("\nВес поезда: " + train.TotalWeight.ToString("f2") + " т");

Console.WriteLine("\nМаксимальная скорость: " + train.MaxSpeed.ToString("f1") + " км/ч");

}

}

}

**3. Выполнение программы**

В программе реализован интерфейс IWagon с двумя дочерними классами PassengerWagon и CargoWagon, интерфейс Ilocomotive с дочерним классом DieselLocomotive, класс Train, где реализована бизнес логика проекта. Также реализован класс Program, созданный для демонстрации функционала проекта. При запуске программы создается поезд, вычисляется общая длина состава, выводится информация о самом длинном и самом коротком вагонах, а так же общий вес состава и рассчитывается его максимальная скорость.

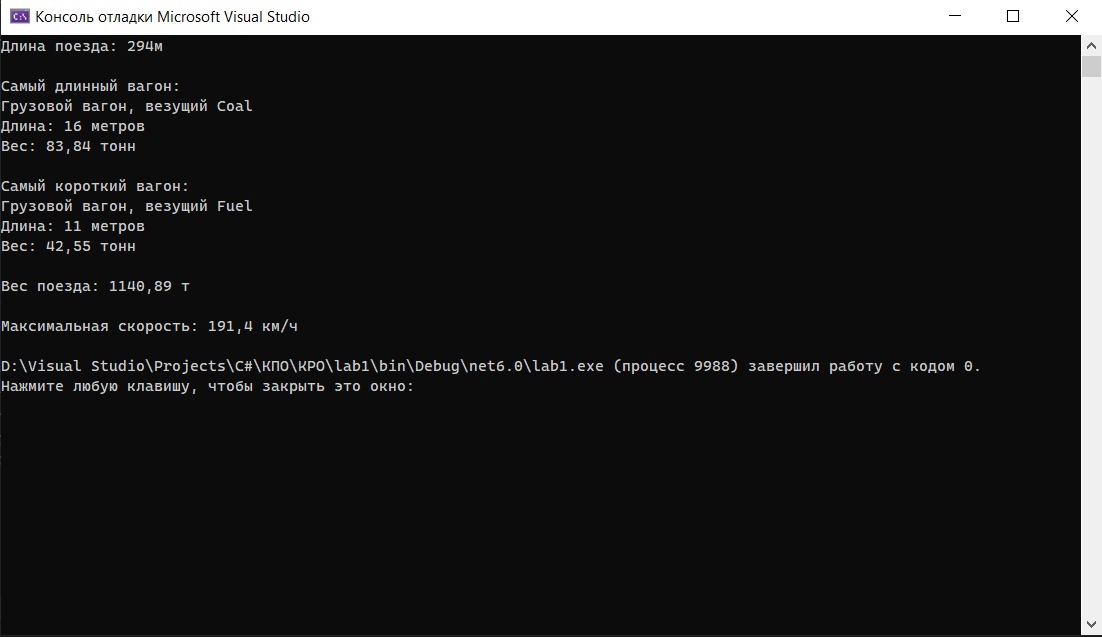


Рис. 4 – Результат выполнения метода Main

**4. Выводы**

Мы научились анализировать предметную область и с помощью абстракции выделять существенные детали, на базе которых в дальнейшем проектируются классы и объекты будущей программной системы согласно методологии ООП, а также практически закрепить данные навыки при решении соответствующих задач (бизнес проблем). В данной лабораторной работе предметной областью выступал железнодорожный транспорт, а бизнес функции заключались в поиске самого длинного/короткого состава, расчёте общей длины состава и его максимальной скорости.

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий и робототехники

Кафедра программного обеспечения информационных систем и технологий

**Отчет**

**по лабораторной работе № 2-4**

# **«**Объектно-Ориентированное Программирование. Инициализация состояния объекта. Статические компоненты класса. Инкапсуляция. Повторное использование кода. Наследование**»**

по дисциплине: «КПО»

Выполнили: студенты гр. 10701221

Крупкин Тимофей Сергеевич

Кульгейко Степан Александрович

Принял: преподаватель Станкевич С.Н.

Минск 2023

**Цель работы:**

**1)**Научиться грамотно использовать соответствующие средства, предоставляемые языком Java, для первоначальной инициализации состояния объекта, а также изучить истинное предназначение статических компонентов класса в языке Java.

**2)** Углубить свои фундаментальные знания в использовании методологии ООП, а также научиться практически применять инкапсуляцию с использованием средств, которые предоставляет язык Java.

**3)** Углубить свои фундаментальные знания в использовании методологии ООП, а также научиться практически применять инструменты языка Java для повторного использования кода в виде ассоциации, наследования, агрегации, композиции и делегирования.

**Задание**

**1)** Необходимо в проект, который был спроектирован и разработан в предыдущей лабораторной работе, внести следующие дополнения:

− для грамотной инициализации состояния объектов соответствующей предметной области добавить всевозможные средства инициализации, которые предоставляет язык Java (блоки инициализации, конструктор по умолчанию, конструкторы с параметрами, конструктор-копирования и т.д.);

− проанализировав соответствующую предметную область добавить в проект статические компоненты класса и возможность их первоначальной инициализации с помощью средств, который предоставляет язык Java. Дополнительно необходимо проанализировать стадии и способы инициализации как состояния объектов, так и состояния соответствующих объектов классов (объектов класса Class), а также их очередность вызова JVM. Привести анализ результатов и соответствующие выводы в отчёте.

**2)** Необходимо в проект, который был спроектирован и разработан в предыдущей лабораторной работе, внести следующие изменения и дополнения:

− скрыть реализацию всех компонентов и структур данных проекта, т.е. инкапсулировать все поля классов и методы, которые предназначены для внутреннего использования, с использованием модификаторов доступа языка Java, и предоставить только интерфейсную часть для внешнего взаимодействия;

− ввести, где это необходимо, высокоуровневые объекты-контейнеры, которые инкапсулируют структуру хранения множества объектов предметной области;   
− убрать из класса-контроллера код по инициализации объектов предметной области и ввести соответствующие программные компоненты, которые и будут предназначены для создания и инициализации объектов предметной области, т.е. использовать компоненты в виде фабрик ил строителей («креаторов»).

**3)** Необходимо произвести рефакторинг программной системы, созданной в предыдущей лабораторной работе, следующим образом:

− классы, описывающие объекты соответствующей предметной области (бизнес-объекты), должны быть сведены в иерархическую структуру (произвести, где это необходимо, классификацию типов); к примеру, в предыдущей лабораторной работе была только сущность автомобиль/автотранспорт, а теперь должна быть иерархия автотранспорта с соответствующими характеристиками: легковой автомобиль (седан, универсал, хэтчбэк, …), грузовой автомобиль (фура, самосвал, бетономешалка ,…), пассажирский автомобиль (автобус, микроавтобус, минивэн, …) и т.д., никто никого не ограничивает в фантазиях;

− логика системы должна быть реализована внутри соответствующих функциональных классов;

− логика системы и большинство других компонентов должны зависеть преимущественно только от абстракции, а не от реализаций;

− необходимо дополнительно для безопасности выполнения кода добавить по возможности в методы бизнес логики проверку входящих объектов на соответствие типа, с которым должна взаимодействовать данная логика.

**Результаты выполнения работы**

**1. Совместная работа с использованием службы размещения репозиториев GitHub**

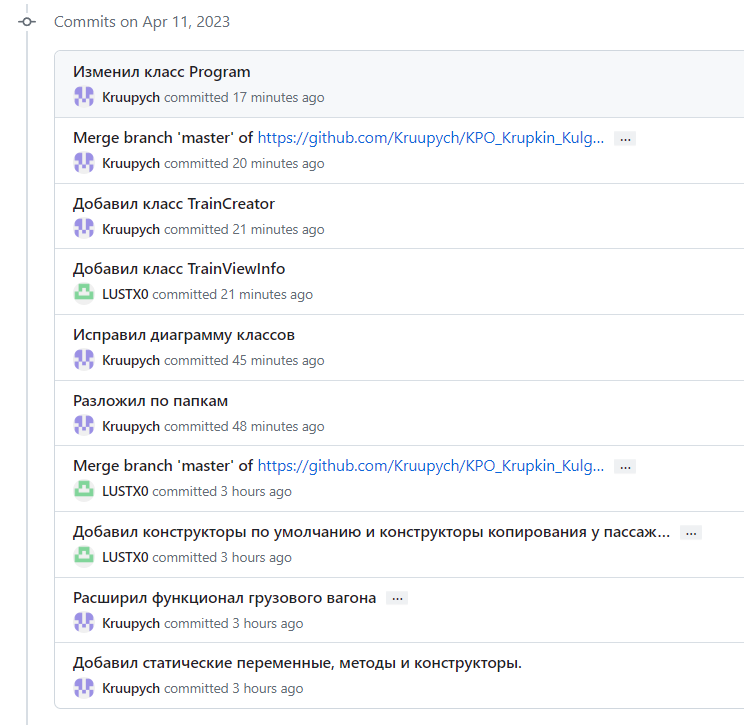
****

Рис. 1 – Список коммитов, сделанных за время выполнения задачи

**2.1. Разработка предметной области**

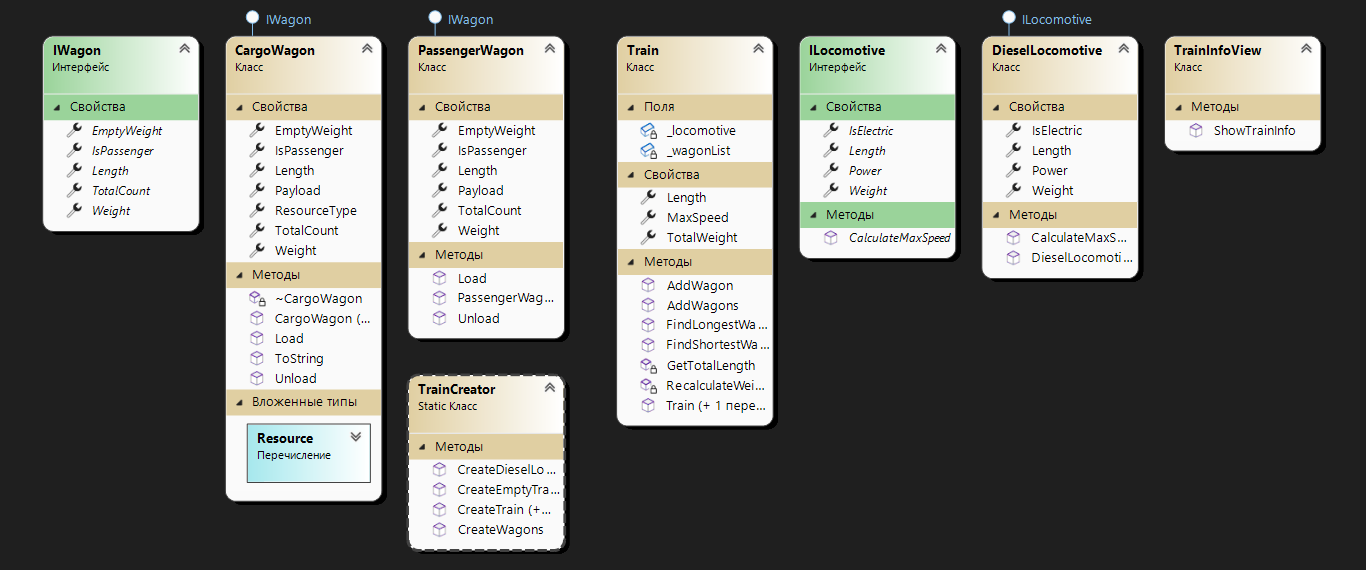
****

Рис. 2 – Диаграмма классов, созданная в среде Visual Studio

**2.2. Листинг программы**

**/Controller**

**//TrainCreator**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace RailwayTransport.Controller

{

public static class TrainCreator

{

public static List<CargoWagon> CreateWagons(int count)

{

count = count < 1 ? 1 : count;

List<CargoWagon> cargoWagons = new();

Random random = new Random();

for (int i = 0; i < count; i++)

{

//Генерируем параметры

double length = random.Next(11, 17);

double emptyWeight = random.Next(18, 31);

double payLoad = random.Next(45, 96);

CargoWagon.Resource resource = (CargoWagon.Resource)random.Next(1, 6);

int loadPercentage = random.Next(1, 101);

//Инициализируем вагон

CargoWagon wagon = new CargoWagon(length, emptyWeight, payLoad);

//Загружаем вагон

wagon.Load(resource, loadPercentage);

//Добавляем в список

cargoWagons.Add(wagon);

}

return cargoWagons;

}

public static DieselLocomotive CreateDieselLocomotive()

{

return new DieselLocomotive();

}

public static Train CreateEmptyTrain()

{

return new Train(CreateDieselLocomotive());

}

public static Train CreateTrain(int wagonsCount)

{

wagonsCount = wagonsCount > 100? 100 : wagonsCount;

var locomotive = CreateDieselLocomotive();

var wagons = CreateWagons(wagonsCount);

return new Train(locomotive, wagons);

}

public static Train CreateTrain(DieselLocomotive locomotive, IEnumerable<IWagon> wagons)

{

return CreateTrain(locomotive, wagons);

}

}

}

**/Model**

**//Locomotives**

**///DieselLocomotive**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace RailwayTransport

{

public class DieselLocomotive : ILocomotive

{

public double Length { get; private set; }

public double Power { get; private set; }

public double Weight { get; private set; }

public bool IsElectric { get; private set; }

public DieselLocomotive(double length, double power, double weight)

{

Length = length;

Power = power;

Weight = weight;

IsElectric = false;

}

public DieselLocomotive()

{

Length = 13.5;

Power = 7500;

Weight = 100;

IsElectric = false;

}

public DieselLocomotive(DieselLocomotive locomotive)

{

Length = locomotive.Length;

Power = locomotive.Power;

Weight = locomotive.Weight;

IsElectric = locomotive.IsElectric;

}

public double CalculateMaxSpeed(double totalWeight)

{

return Power / totalWeight \* 24d;

}

public static double CalculateMaxSpeed(double totalWeight, double power)

{

return power / totalWeight \* 24d;

}

}

}

**///ILocomotive**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace RailwayTransport

{

public interface ILocomotive

{

double Length { get; }

double Power { get; }

double Weight { get; }

bool IsElectric { get; }

public double CalculateMaxSpeed(double totalWeight);

}

}

**//Trains**

**///Train**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace RailwayTransport

{

public class Train

{

List<IWagon> \_wagonList;

ILocomotive \_locomotive;

public double TotalWeight { get; private set; }

public double Length { get => GetTotalLength(); }

public double MaxSpeed { get => \_locomotive.CalculateMaxSpeed(TotalWeight); }

public Train(ILocomotive locomotive)

{

\_locomotive = locomotive;

\_wagonList = new List<IWagon>();

RecalculateWeight();

}

public Train(ILocomotive locomotive, IEnumerable<IWagon> wagons) : this(locomotive)

{

AddWagons(wagons);

}

public void AddWagon(IWagon wagon)

{

\_wagonList.Add(wagon);

RecalculateWeight();

}

public void AddWagons(IEnumerable<IWagon> wagons)

{

foreach (var wagon in wagons)

{

\_wagonList.Add(wagon);

}

RecalculateWeight();

}

private void RecalculateWeight()

{

double weight = \_locomotive.Weight;

foreach (var wagon in \_wagonList)

{

weight += wagon.Weight;

}

TotalWeight = weight;

}

public IWagon FindShortestWagon()

{

if (\_wagonList.Count == 0) return null;

IWagon shortestWagon = \_wagonList[0];

foreach (var wagon in \_wagonList)

{

if (wagon.Length < shortestWagon.Length)

{

shortestWagon = wagon;

}

}

return shortestWagon;

}

public IWagon FindLongestWagon()

{

if (\_wagonList.Count == 0) return null;

IWagon longestWagon = \_wagonList[0];

foreach (var wagon in \_wagonList)

{

if (wagon.Length > longestWagon.Length)

{

longestWagon = wagon;

}

}

return longestWagon;

}

private double GetTotalLength()

{

double totalLength = \_locomotive.Length;

foreach (var wagon in \_wagonList)

{

totalLength += wagon.Length;

}

return totalLength;

}

}

}

**//Wagons**

**///CargoWagon**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace RailwayTransport

{

public class CargoWagon : IWagon

{

public enum Resource

{

Nothing, Coal, Wood, AgroCombine, Fuel, Animals

}

public double Length { get; private set; }

public bool IsPassenger { get; private set; }

public double EmptyWeight { get; private set; }

public double Weight { get; private set; }

public double Payload { get; private set; }

public Resource ResourceType { get; private set; }

public static int TotalCount { get; private set; }

public CargoWagon()

{

Length = 15d;

EmptyWeight = 25d;

Weight = EmptyWeight;

Payload = 75d;

ResourceType = Resource.Nothing;

IsPassenger = false;

TotalCount++;

}

public CargoWagon(double length, double emptyWeight, double payload)

{

Length = length;

EmptyWeight = emptyWeight;

Weight = emptyWeight;

Payload = payload;

ResourceType = Resource.Nothing;

IsPassenger = false;

TotalCount++;

}

public CargoWagon(CargoWagon wagon)

{

Length = wagon.Length;

EmptyWeight = wagon.EmptyWeight;

Weight = wagon.Weight;

Payload = wagon.Payload;

ResourceType = wagon.ResourceType;

IsPassenger = false;

TotalCount++;

}

static CargoWagon()

{

TotalCount = 0;

}

public void Unload()

{

ResourceType = Resource.Nothing;

Weight = EmptyWeight;

}

public void Load(Resource resource, int loadPercentage)

{

ResourceType = resource;

loadPercentage = loadPercentage > 100? 100 : loadPercentage < 0 ? 0 : loadPercentage;

Weight += Payload \* loadPercentage / 100d;

}

public override string ToString()

{

return $"Грузовой вагон, везущий {ResourceType.ToString()}" +

$"\nДлина: {Length} метров" +

$"\nВес: {Weight.ToString("f2")} тонн";

}

~CargoWagon()

{

TotalCount--;

}

}

}

**///PassengerWagon**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace RailwayTransport

{

public class PassengerWagon : IWagon

{

public double Length { get; private set; }

public bool IsPassenger { get; private set; }

public double EmptyWeight { get; private set; }

public double Weight { get; private set; }

public int Payload { get; private set; }

public static int TotalCount { get; private set; }

public PassengerWagon(double length, double emptyWeight, int maxPassengers)

{

Length = length;

EmptyWeight = emptyWeight;

IsPassenger = true;

Weight = emptyWeight;

Payload = maxPassengers;

TotalCount++;

}

public PassengerWagon()

{

Length = 25.0;

EmptyWeight = 56.0;

IsPassenger = true;

Weight = EmptyWeight;

Payload = 64;

TotalCount++;

}

public PassengerWagon(PassengerWagon wagon)

{

Length = wagon.Length ;

EmptyWeight = wagon.EmptyWeight;

IsPassenger = wagon.IsPassenger;

Weight = wagon.Weight;

Payload = wagon.Payload;

TotalCount++;

}

static PassengerWagon()

{

TotalCount = 0;

}

public void Unload()

{

Weight = EmptyWeight;

}

public void Load(int passengersCount)

{

passengersCount = passengersCount > Payload ? Payload : passengersCount < 0 ? 0 : passengersCount;

Weight += passengersCount \* 0.07d;

}

~PassengerWagon()

{

TotalCount--;

}

}

}

**///IWagon**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace RailwayTransport

{

public interface IWagon

{

public double Length { get; }

public bool IsPassenger { get; }

public double EmptyWeight { get; }

public double Weight { get; }

public static int TotalCount { get; }

}

}

**/View**

**//TrainInfoView**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace RailwayTransport.View

{

internal class TrainInfoView

{

public static void ShowTrainInfo(Train train)

{

Console.WriteLine("Длина поезда: " + train.Length + "м");

Console.WriteLine("\nСамый длинный вагон:\n" + train.FindLongestWagon().ToString());

Console.WriteLine("\nСамый короткий вагон:\n" + train.FindShortestWagon().ToString());

Console.WriteLine("\nВес поезда: " + train.TotalWeight.ToString("f2") + " т");

Console.WriteLine("\nМаксимальная скорость: " + train.MaxSpeed.ToString("f1") + " км/ч");

}

}

}

**//Program**

using RailwayTransport.Controller;

using RailwayTransport.View;

namespace RailwayTransport

{

internal class Program

{

static void Main(string[] args)

{

var train = TrainCreator.CreateTrain(wagonsCount: 30);

TrainInfoView.ShowTrainInfo(train);

}

}

}

**3. Выполнение программы**

- Была реализована архитектура MVC. Добавлен класс «TrainCreator.cs» в модуле Controller для инициализации компонентов поезда, а также класс «TrainInfoView.cs» в модуле View для вывода информации о поезде в консоль.

- Были добавлены конструкторы по умолчанию, копирования деструкторы для классов «CargoWagon», «PassengerWagon» и «DieselLocomotive».

- Были добавлены статические переменные и методы.

- Также была введена проверка некоторых входных параметров на корректность.

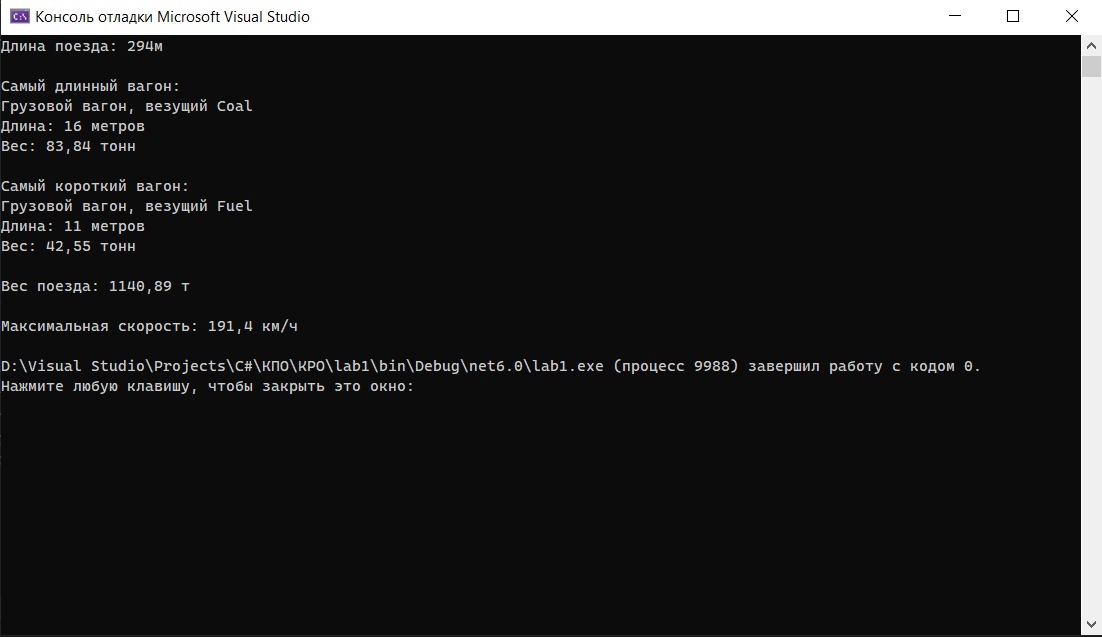


Рис. 4 – Результат выполнения метода Main

**4. Выводы**

Научились грамотно использовать соответствующие средства, предоставляемые языком C#, для первоначальной инициализации состояния объекта, а также изучили истинное предназначение статических компонентов класса в языке C#. Углубили свои фундаментальные знания в использовании методологии ООП, а также научились практически применять инкапсуляцию с использованием средств, которые предоставляет язык C#. Углубили свои фундаментальные знания в использовании методологии ООП, а также научились практически применять инструменты языка C# для повторного использования кода в виде ассоциации, наследования, агрегации, композиции и делегирования.

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий и робототехники

Кафедра программного обеспечения информационных систем и технологий

**Отчет**

**по лабораторной работе № 5-6**

# **«**Объектно-Ориентированное Программирование. Полиморфизм. Методология разработки Test Driven Development. Unit тестирование в C#**»**

по дисциплине: «КПО»

Выполнили: студенты гр. 10701221

Крупкин Тимофей Сергеевич

Кульгейко Степан Александрович

Принял: преподаватель Станкевич С.Н.

Минск 2023

**Цель работы:**

**1)** Углубить свои фундаментальные знания в использовании методологии ООП, а также научиться практически применять динамический полиморфизм с использованием средств, которые предоставляет язык C#.

**2)** Освоить методологию разработки ПО через тестирование (TDD), а также изучить соответствующие тестовые фреймворки для модульного тестирования jUnit и TestNG и закрепить их практически.

**Задание**

**1)**  Необходимо произвести рефакторинг программной системы, созданной в предыдущей лабораторной работе, следующим образом:

− наделить сущности (бизнес-объекты), которыми манипулирует бизнеслогика, соответствующим полиморфным поведением таким образом, чтобы не пришлось переписывать саму бизнес-логику;

− используя поведенческий шаблон проектирования стратегия (The Strategy Pattern) внедрить логику по поиску соответствующих данных в предметную область, а также добавить различные типы сортировок (сортировки по разным критериям сущностей из предметной области);

− на уровне всей программы добавить глобальные характеристики, на базе которых реализуется бизнес-логика (на пример, механизм отслеживания количества создаваемых объектов предметной области, которыми манипулирует система при выполнении основных расчётов программной системы).

**2)** Необходимо произвести рефакторинг программной системы, созданной в предыдущей лабораторной работе, следующим образом:

− создать отдельный тестовый подпроект в рамках текущего программного решения;

− максимально покрыть основную бизнес-логику модульными тестами;

− реализовать тестовые планы, которые будут включать как методы, которые будут вызываться перед и после каждого тестового случая, так и методы, которые должны вызываться на уровне всего класса;

− предусмотреть также тестирование методов на исключительные ситуации и временные интервалы.

**Результаты выполнения работы**

**1. Совместная работа с использованием службы размещения репозиториев GitHub**

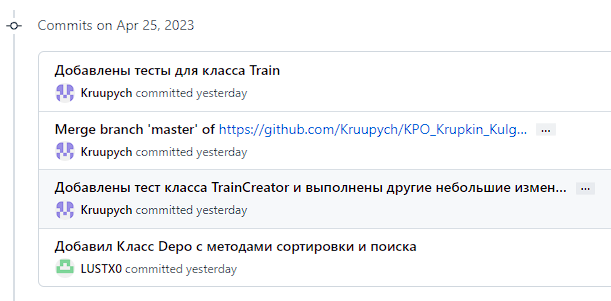
****

Рис. 1 – Список коммитов, сделанных за время выполнения задачи

**2.1. Разработка предметной области**

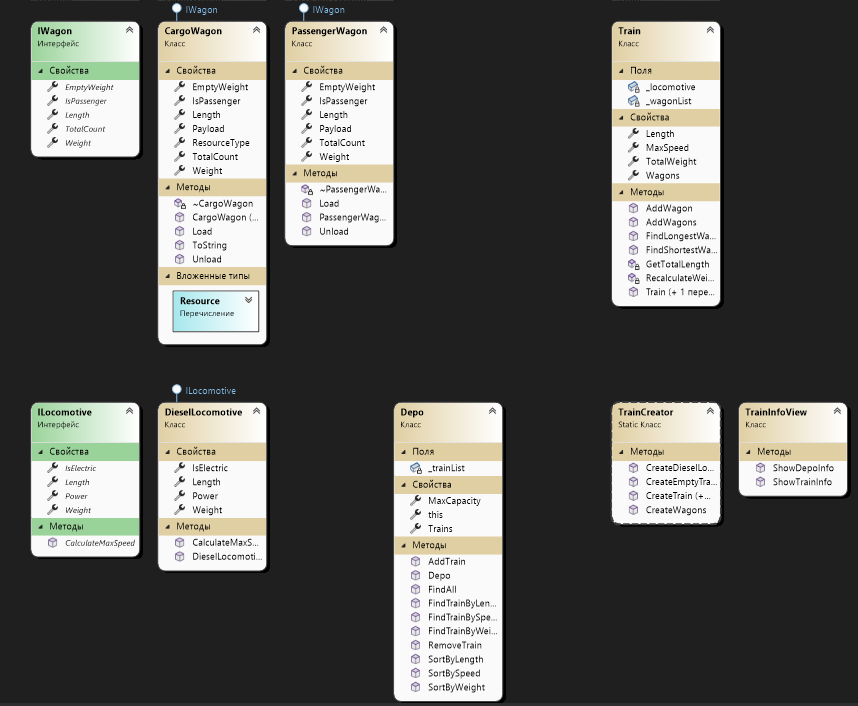
****

Рис. 2 – Диаграмма классов, созданная в среде Visual Studio

**2.2. Листинг программы**

**/Controller**

**//TrainCreator**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace RailwayTransport.Controller

{

public static class TrainCreator

{

public static List<CargoWagon> CreateWagons(int count)

{

if (count < 1)

{

throw new ArgumentOutOfRangeException();

}

List<CargoWagon> cargoWagons = new();

Random random = new Random();

for (int i = 0; i < count; i++)

{

//Генерируем параметры

double length = random.Next(11, 17);

double emptyWeight = random.Next(18, 31);

double payLoad = random.Next(45, 96);

CargoWagon.Resource resource = (CargoWagon.Resource)random.Next(1, 6);

int loadPercentage = random.Next(1, 101);

//Инициализируем вагон

CargoWagon wagon = new CargoWagon(length, emptyWeight, payLoad);

//Загружаем вагон

wagon.Load(resource, loadPercentage);

//Добавляем в список

cargoWagons.Add(wagon);

}

return cargoWagons;

}

public static DieselLocomotive CreateDieselLocomotive()

{

return new DieselLocomotive();

}

public static Train CreateEmptyTrain()

{

return new Train(CreateDieselLocomotive());

}

public static Train CreateTrain(int wagonsCount)

{

wagonsCount = wagonsCount > 100? 100 : wagonsCount;

var locomotive = CreateDieselLocomotive();

var wagons = CreateWagons(wagonsCount);

return new Train(locomotive, wagons);

}

public static Train CreateTrain(DieselLocomotive locomotive, IEnumerable<IWagon> wagons)

{

return new Train(locomotive, wagons);

}

}

}

**/Model**

**//Depo**

**///Depo**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace RailwayTransport

{

public class Depo

{

private List<Train> \_trainList;

public int MaxCapacity { get; private set; }

public List<Train> Trains { get => \_trainList; }

public Depo(int maxCapacity)

{

MaxCapacity = maxCapacity;

\_trainList = new List<Train>();

}

public Train this[int index]

{

get => \_trainList[index];

private set => \_trainList[index] = value;

}

public bool AddTrain(Train train)

{

if (\_trainList.Count < MaxCapacity)

{

\_trainList.Add(train);

return true;

}

return false;

}

public bool RemoveTrain(Train train)

{

return \_trainList.Remove(train);

}

public Depo SortByWeight()

{

\_trainList.Sort((x, y) => x.TotalWeight.CompareTo(y.TotalWeight));

return this;

}

public Depo SortBySpeed()

{

\_trainList.Sort((x, y) => x.MaxSpeed.CompareTo(y.MaxSpeed));

return this;

}

public Depo SortByLength()

{

\_trainList.Sort((x, y) => x.Length.CompareTo(y.Length));

return this;

}

public Train FindTrainBySpeed(bool IsSmallest)

{

if(\_trainList== null)

{

throw new Exception("\_trainList is null");

}

if (IsSmallest)

{

var minm = \_trainList.Min(p => p.MaxSpeed);

return \_trainList.Find(p => p.MaxSpeed == minm);

}

else

{

var maxm = \_trainList.Max(p => p.MaxSpeed);

return \_trainList.Find(p => p.MaxSpeed == maxm);

}

}

public Train FindTrainByWeight(bool IsSmallest)

{

if (\_trainList == null)

{

throw new Exception("\_trainList is null");

}

if (IsSmallest)

{

var minm = \_trainList.Min(p => p.TotalWeight);

return \_trainList.Find(p => p.TotalWeight == minm);

}

else

{

var maxm = \_trainList.Max(p => p.TotalWeight);

return \_trainList.Find(p => p.TotalWeight == maxm);

}

}

public Train FindTrainByLength(bool IsSmallest)

{

if (\_trainList == null)

{

throw new Exception("\_trainList is null");

}

if (IsSmallest)

{

var minm = \_trainList.Min(p => p.Length);

return \_trainList.Find(p => p.Length == minm);

}

else

{

var maxm = \_trainList.Max(p => p.Length);

return \_trainList.Find(p => p.Length == maxm);

}

}

public List<Train> FindTrainBySpeed(bool IsBiggerThan,int value)

{

if (\_trainList == null)

{

throw new Exception("\_trainList is null");

}

if (IsBiggerThan)

{

return \_trainList.FindAll(p => p.MaxSpeed >= value);

}

else

{

return \_trainList.FindAll(p => p.MaxSpeed < value);

}

}

public List<Train> FindTrainByWeight(bool IsBiggerThan, double value)

{

if (\_trainList == null)

{

throw new Exception("\_trainList is null");

}

if (IsBiggerThan)

{

return \_trainList.FindAll(p => p.TotalWeight >= value);

}

else

{

return \_trainList.FindAll(p => p.TotalWeight < value);

}

}

public List<Train> FindTrainByLength(bool IsBiggerThan, double value)

{

if (\_trainList == null)

{

throw new Exception("\_trainList is null");

}

if (IsBiggerThan)

{

return \_trainList.FindAll(p => p.Length >= value);

}

else

{

return \_trainList.FindAll(p => p.Length < value);

}

}

public List<Train> FindAll(Predicate<Train> match)

{

return \_trainList.FindAll(match);

}

}

}

**//Locomotives**

**///DieselLocomotive**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace RailwayTransport

{

public class DieselLocomotive : ILocomotive

{

public double Length { get; private set; }

public double Power { get; private set; }

public double Weight { get; private set; }

public bool IsElectric { get; private set; }

public DieselLocomotive(double length, double power, double weight)

{

Length = length;

Power = power;

Weight = weight;

IsElectric = false;

}

public DieselLocomotive()

{

Length = 13.5;

Power = 7500;

Weight = 100;

IsElectric = false;

}

public DieselLocomotive(DieselLocomotive locomotive)

{

Length = locomotive.Length;

Power = locomotive.Power;

Weight = locomotive.Weight;

IsElectric = locomotive.IsElectric;

}

public double CalculateMaxSpeed(double totalWeight)

{

return Power / totalWeight \* 24d;

}

public static double CalculateMaxSpeed(double totalWeight, double power)

{

return power / totalWeight \* 24d;

}

}

}

**///ILocomotive**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace RailwayTransport

{

public interface ILocomotive

{

double Length { get; }

double Power { get; }

double Weight { get; }

bool IsElectric { get; }

public double CalculateMaxSpeed(double totalWeight);

}

}

**//Trains**

**///Train**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace RailwayTransport

{

public class Train

{

List<IWagon> \_wagonList;

ILocomotive \_locomotive;

public double TotalWeight { get; private set; }

public double Length { get => GetTotalLength(); }

public double MaxSpeed { get => \_locomotive.CalculateMaxSpeed(TotalWeight); }

public Train(ILocomotive locomotive)

{

\_locomotive = locomotive;

\_wagonList = new List<IWagon>();

RecalculateWeight();

}

public Train(ILocomotive locomotive, IEnumerable<IWagon> wagons) : this(locomotive)

{

AddWagons(wagons);

}

public void AddWagon(IWagon wagon)

{

\_wagonList.Add(wagon);

RecalculateWeight();

}

public void AddWagons(IEnumerable<IWagon> wagons)

{

foreach (var wagon in wagons)

{

\_wagonList.Add(wagon);

}

RecalculateWeight();

}

private void RecalculateWeight()

{

double weight = \_locomotive.Weight;

foreach (var wagon in \_wagonList)

{

weight += wagon.Weight;

}

TotalWeight = weight;

}

public IWagon FindShortestWagon()

{

if (\_wagonList.Count == 0) return null;

IWagon shortestWagon = \_wagonList[0];

foreach (var wagon in \_wagonList)

{

if (wagon.Length < shortestWagon.Length)

{

shortestWagon = wagon;

}

}

return shortestWagon;

}

public IWagon FindLongestWagon()

{

if (\_wagonList.Count == 0) return null;

IWagon longestWagon = \_wagonList[0];

foreach (var wagon in \_wagonList)

{

if (wagon.Length > longestWagon.Length)

{

longestWagon = wagon;

}

}

return longestWagon;

}

private double GetTotalLength()

{

double totalLength = \_locomotive.Length;

foreach (var wagon in \_wagonList)

{

totalLength += wagon.Length;

}

return totalLength;

}

}

}

**//Wagons**

**///CargoWagon**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace RailwayTransport

{

public class CargoWagon : IWagon

{

public enum Resource

{

Nothing, Coal, Wood, AgroCombine, Fuel, Animals

}

public double Length { get; private set; }

public bool IsPassenger { get; private set; }

public double EmptyWeight { get; private set; }

public double Weight { get; private set; }

public double Payload { get; private set; }

public Resource ResourceType { get; private set; }

public static int TotalCount { get; private set; }

public CargoWagon()

{

Length = 15d;

EmptyWeight = 25d;

Weight = EmptyWeight;

Payload = 75d;

ResourceType = Resource.Nothing;

IsPassenger = false;

TotalCount++;

}

public CargoWagon(double length, double emptyWeight, double payload)

{

Length = length;

EmptyWeight = emptyWeight;

Weight = emptyWeight;

Payload = payload;

ResourceType = Resource.Nothing;

IsPassenger = false;

TotalCount++;

}

public CargoWagon(CargoWagon wagon)

{

Length = wagon.Length;

EmptyWeight = wagon.EmptyWeight;

Weight = wagon.Weight;

Payload = wagon.Payload;

ResourceType = wagon.ResourceType;

IsPassenger = false;

TotalCount++;

}

static CargoWagon()

{

TotalCount = 0;

}

public void Unload()

{

ResourceType = Resource.Nothing;

Weight = EmptyWeight;

}

public void Load(Resource resource, int loadPercentage)

{

ResourceType = resource;

loadPercentage = loadPercentage > 100? 100 : loadPercentage < 0 ? 0 : loadPercentage;

Weight += Payload \* loadPercentage / 100d;

}

public override string ToString()

{

return $"Грузовой вагон, везущий {ResourceType.ToString()}" +

$"\nДлина: {Length} метров" +

$"\nВес: {Weight.ToString("f2")} тонн";

}

~CargoWagon()

{

TotalCount--;

}

}

}

**///PassengerWagon**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace RailwayTransport

{

public class PassengerWagon : IWagon

{

public double Length { get; private set; }

public bool IsPassenger { get; private set; }

public double EmptyWeight { get; private set; }

public double Weight { get; private set; }

public int Payload { get; private set; }

public static int TotalCount { get; private set; }

public PassengerWagon(double length, double emptyWeight, int maxPassengers)

{

Length = length;

EmptyWeight = emptyWeight;

IsPassenger = true;

Weight = emptyWeight;

Payload = maxPassengers;

TotalCount++;

}

public PassengerWagon()

{

Length = 25.0;

EmptyWeight = 56.0;

IsPassenger = true;

Weight = EmptyWeight;

Payload = 64;

TotalCount++;

}

public PassengerWagon(PassengerWagon wagon)

{

Length = wagon.Length ;

EmptyWeight = wagon.EmptyWeight;

IsPassenger = wagon.IsPassenger;

Weight = wagon.Weight;

Payload = wagon.Payload;

TotalCount++;

}

static PassengerWagon()

{

TotalCount = 0;

}

public void Unload()

{

Weight = EmptyWeight;

}

public void Load(int passengersCount)

{

passengersCount = passengersCount > Payload ? Payload : passengersCount < 0 ? 0 : passengersCount;

Weight += passengersCount \* 0.07d;

}

~PassengerWagon()

{

TotalCount--;

}

}

}

**///IWagon**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace RailwayTransport

{

public interface IWagon

{

public double Length { get; }

public bool IsPassenger { get; }

public double EmptyWeight { get; }

public double Weight { get; }

public static int TotalCount { get; }

}

}

**/View**

**//TrainInfoView**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace RailwayTransport.View

{

internal class TrainInfoView

{

public static void ShowTrainInfo(Train train)

{

Console.WriteLine("Длина поезда: " + train.Length + "м");

Console.WriteLine("\nСамый длинный вагон:\n" + train.FindLongestWagon().ToString());

Console.WriteLine("\nСамый короткий вагон:\n" + train.FindShortestWagon().ToString());

Console.WriteLine("\nВес поезда: " + train.TotalWeight.ToString("f2") + " т");

Console.WriteLine("\nМаксимальная скорость: " + train.MaxSpeed.ToString("f1") + " км/ч");

}

public static void ShowDepoInfo(Depo depo)

{

for (int i = 0; i < depo.Trains.Count; i++)

{

Console.WriteLine($"\n\nПоезд №{i + 1}: \n");

ShowTrainInfo(depo[i]);

}

}

}

}

**//Program**

using RailwayTransport.Controller;

using RailwayTransport.View;

namespace RailwayTransport

{

internal class Program

{

static void Main(string[] args)

{

var train0 = TrainCreator.CreateTrain(wagonsCount: 30);

var train1 = TrainCreator.CreateTrain(wagonsCount: 11);

var train2 = TrainCreator.CreateTrain(wagonsCount: 22);

var train3 = TrainCreator.CreateTrain(wagonsCount: 10);

var train4 = TrainCreator.CreateTrain(wagonsCount: 8);

Depo depo = new Depo(5);

depo.AddTrain(train0);

depo.AddTrain(train1);

depo.AddTrain(train2);

depo.AddTrain(train3);

depo.AddTrain(train4);

Console.WriteLine("\n\t---До сортировки:---\n");

TrainInfoView.ShowDepoInfo(depo);

Console.WriteLine("\n\n\t---Сортировка по скорости:---\n");

TrainInfoView.ShowDepoInfo(depo.SortBySpeed());

Console.WriteLine("\n\n\t---Сортировка по длине:---\n");

TrainInfoView.ShowDepoInfo(depo.SortByLength());

Console.WriteLine("\n\n\t---Сортировка по весу:---\n");

TrainInfoView.ShowDepoInfo(depo.SortByWeight());

}

}

}

//TrainCreatorTests

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace RailwayTransport.Tests

{

[TestClass]

public class TrainCreatorTests

{

[TestMethod]

public void CreateWagons\_30\_ListWith30WagonsReturn()

{

//arrange

int input = 30;

int expected = 30;

//act

var wagonsList = TrainCreator.CreateWagons(input);

//assert

Assert.AreEqual(expected, wagonsList.Count);

}

[TestMethod]

public void CreateTrain\_30\_TrainWith30WagonsReturn()

{

//arrange

int input = 30;

int expected = 30;

//act

int actual = TrainCreator.CreateTrain(input).Wagons.Count;

//assert

Assert.AreEqual(expected, actual);

}

[TestMethod]

public void CreateTrain\_CreateLocomotiveAndCreate30Wagons\_TrainWith30WagonsReturn()

{

//arrange

int input = 30;

int expected = 30;

//act

int actual = TrainCreator.CreateTrain(TrainCreator.CreateDieselLocomotive(), TrainCreator.CreateWagons(input)).Wagons.Count;

//assert

Assert.AreEqual(expected, actual);

}

[ExpectedException(typeof(ArgumentOutOfRangeException))]

[TestMethod]

public void CreateWagons\_LessThanOne\_ArgumentOutOfRangeException()

{

int[] input = { 0, -1, -214 };

for (int i = 0; i < 3; i++)

{

TrainCreator.CreateWagons(input[i]);

}

}

[TestMethod]

public void CreateEmptyTrain\_Null()

{

IWagon expected = null;

var train = TrainCreator.CreateEmptyTrain();

var actual = train.FindLongestWagon();

Assert.AreEqual(expected, actual);

actual = train.FindShortestWagon();

Assert.AreEqual(expected, actual);

}

}

}

//TrainTests

namespace RailwayTransport.Tests

{

[TestClass]

public class TrainTests

{

private static IWagon \_expectedShortestWagon;

private static IWagon \_expectedLongestWagon;

private static List<IWagon> \_wagonsList;

private static double \_expectedTotalLength;

private Train \_train;

[ClassInitialize]

public static void ClassInitialize(TestContext context)

{

\_expectedShortestWagon = new CargoWagon(12, 18, 40);

\_expectedLongestWagon = new CargoWagon(20, 34, 100);

CargoWagon wagon = new CargoWagon();

\_expectedTotalLength = TrainCreator.CreateDieselLocomotive().Length + \_expectedShortestWagon.Length + \_expectedLongestWagon.Length + wagon.Length;

\_wagonsList = new List<IWagon> { \_expectedShortestWagon, wagon, \_expectedLongestWagon };

}

[TestInitialize]

public void TestInitialize()

{

\_train = TrainCreator.CreateTrain(TrainCreator.CreateDieselLocomotive(), \_wagonsList);

}

[ExpectedException(typeof(ArgumentNullException))]

[TestMethod]

public void AddWagon\_NullWagon\_ArgumentNullExceptionThrow()

{

TrainCreator.CreateEmptyTrain().AddWagon(null);

}

[TestMethod]

public void FindShortestWagonTest()

{

var actualShortestWagon = \_train.FindShortestWagon();

Assert.AreSame(\_expectedShortestWagon, actualShortestWagon);

}

[TestMethod]

public void FindLongestWagonTest()

{

var actualLongestWagon = \_train.FindLongestWagon();

Assert.AreSame(\_expectedLongestWagon, actualLongestWagon);

}

[TestMethod]

public void LengthTest()

{

var actualTotalLength = \_train.Length;

Assert.AreEqual(\_expectedTotalLength, actualTotalLength);

}

}

}

**3. Выполнение программы**

- Добавлен класс «Depo.cs» в модуле Model для инициализации компонентов поезда, была внедрена логика по поиску соответствующих данных в предметную область, а также добавить различные типы сортировок (сортировки по разным критериям сущностей из предметной области);

- Был добавлен набор unit тестов для тестирования классов TrainCreator и Train.

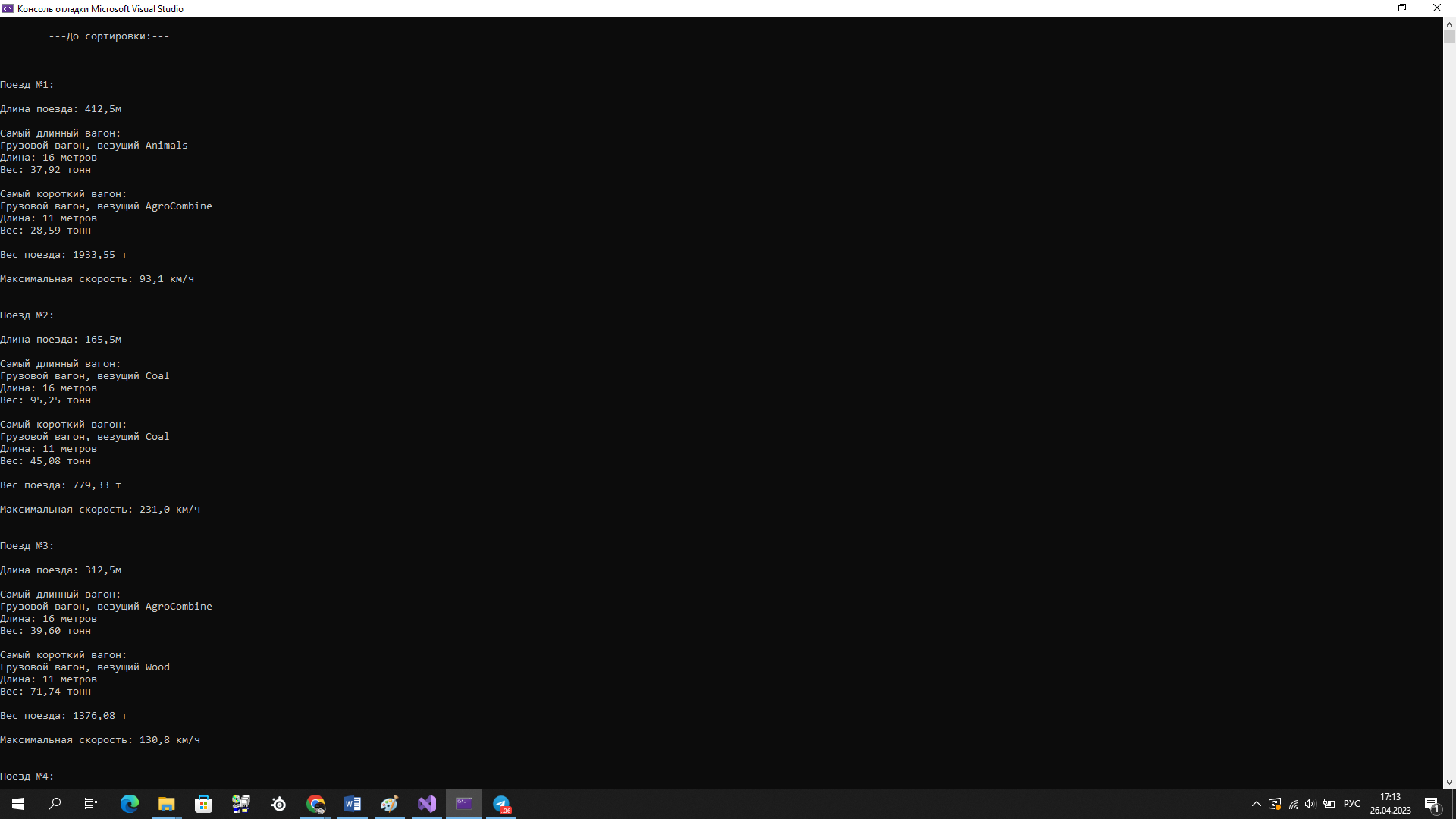


Рис. 4 – Результат выполнения метода Main

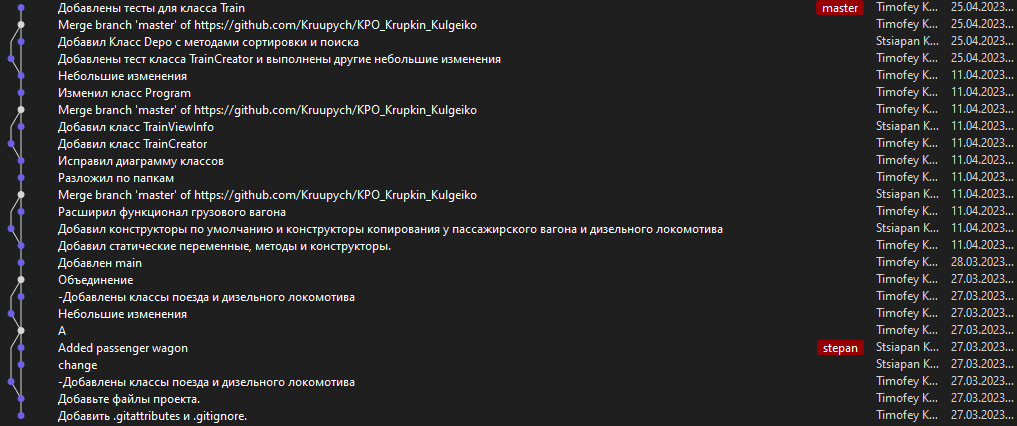


Рис. 5 – Демонстрация совместного рефакторинга проекта. Вид в Visual Studio

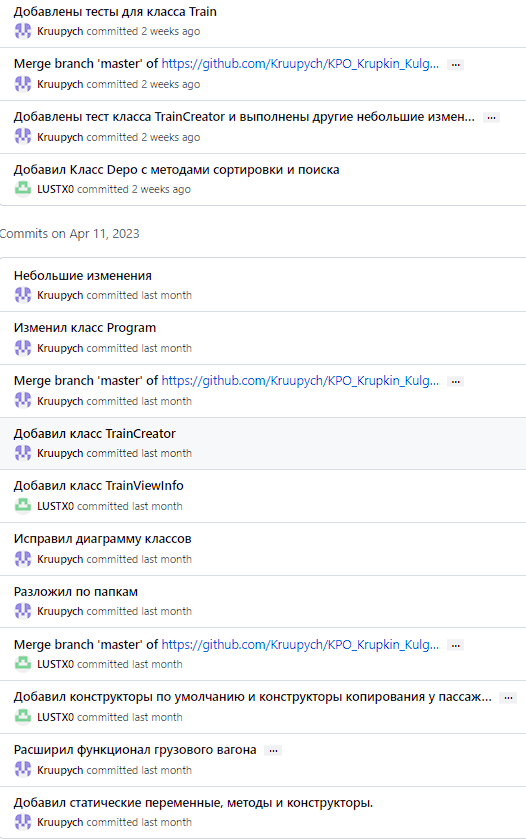
****

Рис. 6 – Демонстрация совместного рефакторинга проекта. Вид в GitHub

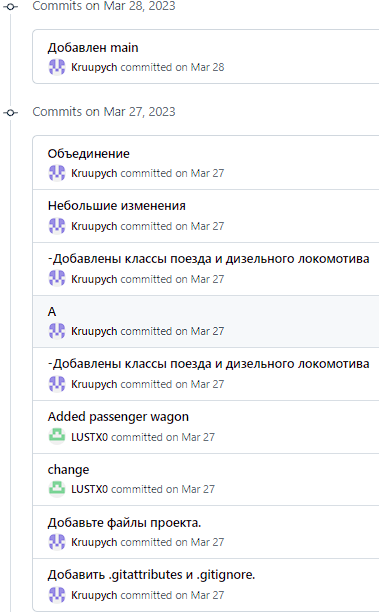
****

Рис. 7 – Демонстрация совместного рефакторинга проекта. Вид в GitHub

**4. Выводы**

Научились грамотно использовать соответствующие средства, предоставляемые языком C#, для первоначальной инициализации состояния объекта, а также изучили истинное предназначение статических компонентов класса в языке C#. Углубили свои фундаментальные знания в использовании методологии ООП, а также научились практически применять инкапсуляцию с использованием средств, которые предоставляет язык C#. Углубили свои фундаментальные знания в использовании методологии ООП, а также научились практически применять инструменты языка C# для повторного использования кода в виде ассоциации, наследования, агрегации, композиции и делегирования.