Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет

информатики и радиоэлектроники»

Факультет компьютерного проектирования

Кафедра Информатики

Дисциплина «Программирование»

**ОТЧЕТ**

к лабораторной работе №9

на тему:

**«множественное наследование»**

БГУИР 6-05-0612-02 35

|  |
| --- |
| Выполнила студентка группы 353504  ГУСЕНЦОВА Екатерина Андреевна |
|  |
| (дата, подпись студента) |
| Проверил ассистент каф. Информатики  РОМАНЮК Максим Валерьевич |
|  |
| (дата, подпись преподавателя) |

Минск 2024

# 1 Индивидуальное задание

**Задание 1. Вариант 5.** Предметная область: Фирма доставки товаров. Пример общих свойств: название, тип (городской, междугородний, международный …). Общие функции: GetInfo (в каждом классе реализуется посвоему), Deliver (варианты описаны в других классах, примеры: автомобилем, DHL, AutoLight Express …). Примеры дополнительных функций: сопровождение груза, страхование, отслеживание этапов доставки …

# 2 Выполнение работы

В рамках заданной предметной области программа содержит семейство классов, имеющих общий функционал. Общий функционал описывается в виде набора четырёх интерфейсов. Требуется использовать шаблоны проектирования “Bridge” и “Builder”. Диаграмма классов представлена на рисунке 1.

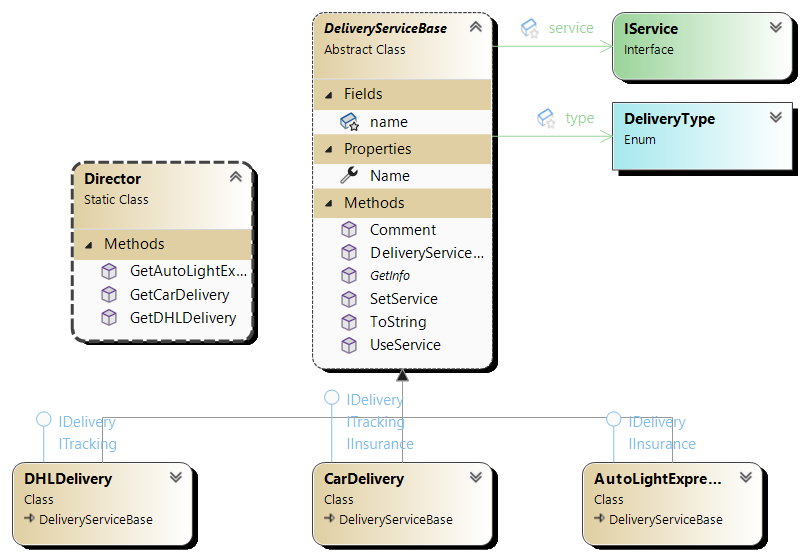


Рисунок 1 – Диаграмма классов

DeliveryType: это перечисление, которое определяет типы доставки - городской, междугородний и международный. Ниже представлен листинг кода.

namespace ConsoleApp1

{

public enum DeliveryType

{

City,

InterCity,

International

}

}

В программе описаны четыре интерфейса.

IDelivery: Этот интерфейс предназначен для определения способа доставки. Каждый класс, реализующий этот интерфейс, должен определить свой способ доставки. Ниже представлен листинг кода.

namespace ConsoleApp1

{

internal interface IDelivery

{

void DeliveryMethod();

}

}

ITracking: Этот интерфейс предназначен для отслеживания процесса доставки. Каждый класс, реализующий этот интерфейс, должен определить, как он отслеживает доставку. Ниже представлен листинг кода.

namespace ConsoleApp1

{

internal interface ITracking

{

void Track();

}

}

IInsurance: Этот интерфейс предназначен для страхования доставки. Каждый класс, реализующий этот интерфейс, должен определить, как он страхует доставку. Ниже представлен листинг кода.

namespace ConsoleApp1

{

internal interface IInsurance

{

void Insure();

}

}

IService: Этот интерфейс предназначен для предоставления информации о доставке. Каждый класс, реализующий этот интерфейс, должен определить, как он предоставляет информацию о доставке. Ниже представлен листинг кода.

namespace ConsoleApp1

{

internal interface IService

{

void GetInfo();

}

}

Для применения шаблона проектирования "Bridge" необходимо выделить абстрактный класс DeliveryServiceBase, описывающий общие члены классов различных служб доставки, и взаимодействующий с видами доставки через интерфейс IService. На рисунке 1 представлена диаграмма классов.

DeliveryServiceBase: это абстрактный класс, который представляет базовую структуру для всех служб доставки. Он содержит общие свойства, такие как name, type и service, а также общие методы, такие как Comment(), SetService(), UseService() и GetInfo(). Каждый производный класс должен предоставить свою реализацию метода GetInfo(). Ниже представлен листинг кода.

using System;

namespace ConsoleApp1

{

internal abstract class DeliveryServiceBase

{

protected string name;

protected DeliveryType type;

protected IService service;

public string Name { get; set; }

public DeliveryServiceBase(string name, DeliveryType type, IService service)

{

this.name = name;

this.type = type;

this.service = service;

}

public void Comment(string comment)

{

Console.WriteLine(name + comment);

}

public void SetService(IService service)

{

this.service = service;

}

public IService UseService()

{

return service;

}

public abstract void GetInfo();

public override string ToString()

{

return string.Empty;

}

}

}

SomeService: Это конкретный класс, который реализует интерфейс IService. В этом классе определен метод GetInfo(), который выводит информацию о сервисе. Ниже представлен листинг кода.

using System;

namespace ConsoleApp1

{

internal class SomeService : IService

{

public void GetInfo()

{

Console.WriteLine("Информация о сервисе: SomeService");

}

}

}

CarDelivery: это конкретный класс доставки, который наследуется от DeliveryServiceBase и реализует интерфейсы IDelivery, ITracking и IInsurance. Этот класс представляет службу доставки, которая использует автомобиль для доставки, предоставляет возможность отслеживания и страхования груза. Ниже представлен листинг кода.

using System;

namespace ConsoleApp1

{

internal class CarDelivery : DeliveryServiceBase, IDelivery, ITracking, IInsurance

{

public CarDelivery(string name, DeliveryType type, IService service)

: base(name, type, service)

{

}

public void DeliveryMethod()

{

Console.WriteLine("Способ доставки: автомобиль.");

}

public void Track()

{

Console.WriteLine("Отслеживание этапов доставки.");

}

public void Insure()

{

Console.WriteLine("Страхование груза.");

}

public override void GetInfo()

{

Console.WriteLine($"Название: {name}, Тип: {type}");

}

}

}

DHLDelivery: это еще один конкретный класс доставки, который также наследуется от DeliveryServiceBase и реализует интерфейсы IDelivery и ITracking. Этот класс представляет службу доставки DHL, которая предоставляет возможность отслеживания груза. Ниже представлен листинг кода.

using System;

namespace ConsoleApp1

{

internal class DHLDelivery : DeliveryServiceBase, IDelivery, ITracking

{

public DHLDelivery(string name, DeliveryType type, IService service)

: base(name, type, service)

{

}

public void DeliveryMethod()

{

Console.WriteLine("Способ доставки: DHL.");

}

public void Track()

{

Console.WriteLine("Отслеживание этапов доставки.");

}

public override void GetInfo()

{

Console.WriteLine($"Название: {name}, Тип: {type}");

}

}

}

AutoLightExpressDelivery: Этот класс наследуется от DeliveryServiceBase и реализует интерфейсы IDelivery и IInsurance. Этот класс представляет службу доставки AutoLight Express, которая предоставляет страхование груза. Ниже представлен листинг кода.

using System;

namespace ConsoleApp1

{

internal class AutoLightExpressDelivery : DeliveryServiceBase, IDelivery, IInsurance

{

public AutoLightExpressDelivery(string name, DeliveryType type, IService service)

: base(name, type, service)

{

}

public void DeliveryMethod()

{

Console.WriteLine("Способ доставки: AutoLight Express.");

}

public void Insure()

{

Console.WriteLine("Страхование груза.");

}

public override void GetInfo()

{

Console.WriteLine($"Название: {name}, Тип: {type}");

}

}

}

Для применения шаблона проектирования "Builder" необходимо создать абстрактный класс DeliveryBuilder. Ниже представлен листинг кода.

namespace ConsoleApp1

{

internal abstract class DeliveryBuilder

{

protected string name;

protected DeliveryType type;

protected IService service;

public DeliveryBuilder SetName(string name)

{

this.name = name;

return this;

}

public DeliveryBuilder SetType(DeliveryType type)

{

this.type = type;

return this;

}

public DeliveryBuilder SetService(IService service)

{

this.service = service;

return this;

}

public abstract DeliveryServiceBase Build();

}

}

CarDeliveryBuilder, DHLDeliveryBuilder, AutoLightExpressDeliveryBuilder: Это строители, которые наследуются от DeliveryBuilderBase. Они используются для создания объектов CarDelivery, DHLDelivery и AutoLightExpressDelivery соответственно. Каждый из них имеет метод Build(), который возвращает новый объект доставки. Ниже представлены листинги кода каждого из них соответственно.

namespace ConsoleApp1

{

internal class CarDeliveryBuilder : DeliveryBuilder

{

public override DeliveryServiceBase Build()

{

return new CarDelivery(name, type, service);

}

}

}

namespace ConsoleApp1

{

internal class DHLDeliveryBuilder : DeliveryBuilder

{

public override DeliveryServiceBase Build()

{

return new DHLDelivery(name, type, service);

}

}

}

namespace ConsoleApp1

{

internal class AutoLightExpressDeliveryBuilder : DeliveryBuilder

{

public override DeliveryServiceBase Build()

{

return new AutoLightExpressDelivery(name, type, service);

}

}

}

Класс Director: Этот класс используется для создания объектов различных типов доставки. Он содержит статические методы, которые принимают имя и строителя (builder) и возвращают объект доставки. Ниже представлен листинг кода.

namespace ConsoleApp1

{

internal static class Director

{

public static DeliveryServiceBase GetCarDelivery(string name, DeliveryBuilder builder)

{

return builder.SetName(name).SetType(DeliveryType.City).SetService(new SomeService()).Build();

}

public static DeliveryServiceBase GetDHLDelivery(string name, DHLDeliveryBuilder builder)

{

return builder.SetName(name).SetType(DeliveryType.InterCity).SetService(new SomeService()).Build();

}

public static DeliveryServiceBase GetAutoLightExpressDelivery(string name, AutoLightExpressDeliveryBuilder builder)

{

return builder.SetName(name).SetType(DeliveryType.International).SetService(new SomeService()).Build();

}

}

}

Класс Program: Этот класс содержит основной метод Main, который является точкой входа в ваше приложение. В этом методе создается коллекция объектов доставки, используя различные строители, и затем для каждого элемента коллекции вызываются все доступные методы. Это соответствует вашему заданию. Ниже представлен листинг кода.

using System;

using System.Collections.Generic;

namespace ConsoleApp1

{

internal class Program

{

static void Main(string[] args)

{

var services = new List<DeliveryServiceBase>();

var carDeliveryBuilder = new CarDeliveryBuilder();

var dhlDeliveryBuilder = new DHLDeliveryBuilder();

var autoLightExpressDeliveryBuilder = new AutoLightExpressDeliveryBuilder();

services.AddRange(new DeliveryServiceBase[] {

Director.GetCarDelivery("Service №1", carDeliveryBuilder),

Director.GetDHLDelivery("Service №2", dhlDeliveryBuilder),

Director.GetAutoLightExpressDelivery("Service №3", autoLightExpressDeliveryBuilder),

});

foreach (var service in services)

{

Console.WriteLine(service.ToString());

}

}

}

}

# Вывод

В ходе лабораторной работы были получены навыки проектирования приложения, состоящих из нескольких взаимосвязанных классов и интерфейсов, а также были изучены концепции множественного наследования в объектно-ориентированном программировании.