МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Е.Н. ЭГОВ

РАЗРАБОТКА ПРОГРАММЫ ХРАНИЛИЩА ОБЪЕКТОВ

практикум по дисциплинам
«Технологии программирования» и
«Разработка профессиональных приложений»
для студентов направлений подготовки бакалавриата
09.03.03 «Прикладная информатика»,
09.03.04 «Программная инженерия»

Ульяновск

УлГТУ

2023

СОДЕРЖАНИЕ

ЛАБОРАТОРНАЯ РАБОТА №0.
РАБОТА В СЕМЕСТРЕ

Цель

Ознакомиться с процессом выполнения работы в семестре.

Работа в семестре

В учебном плане предусмотрено 8 лабораторных работ. Работы выстроены так, что в течении семестра идет постепенное развитие **одного** проекта. В рамках первой лабораторной работы (или до нее) **создается проект,** в который в последующих лабораторных работах **вносятся изменения**, дополнения.

На лабораторных будут рассмотрены следующие темы:

1) Классы, объекты, инкапсуляция. В рамках этой лабораторной ознакомимся как создавать desktop-приложения, создавать классы по сущностям, наполнять классы элементами, а также создавать объекты от классов.
2) Наследование. Абстрактные классы. Интерфейсы. В рамках этой лабораторной ознакомимся как устроен механизм наследования от интерфейсов и абстрактных классов.
3) Полиморфизм. Перегрузка операций. Параметрические классы. В рамках этой лабораторной ознакомимся с параметризованными классами, а также перегрузкой методов.
4) Коллекции. Индексаторы. В рамках этой лабораторной ознакомимся с разнообразными коллекциями, применяемыми в программировании.
5) Делегаты, события. Drag&Drop. В рамках этой лабораторной ознакомимся с принципами событийного программирования, делегатами, а также технологией Drag&Drop.

6) Файлы и потоки. Сохранение и загрузка данных. В рамках этой лабораторной ознакомимся с основами работы с файлами, сохранением данных и их загрузкой.
7) Обработка исключений. Логирование действий. В рамках этой лабораторной ознакомимся как правильно обрабатывать исключения и логировать информацию.
8) Стандартные интерфейсы. В рамках этой лабораторной ознакомимся с основами применения стандартных интерфейсов.

Алгоритм работы в семестре

1. Ознакомится с методическими материалами по лабораторной работе. Просмотреть/прочитать лекцию по теме лабораторной работы. Просмотреть, прочитать методические указания к лабораторной работе. Для лабораторной расписан и этап проектирования (по сути, как планируется писать код, какие классы/интерфейсы и т.п. могут потребоваться, для каких целей), и этап разработки (приводится код с основной логикой, оставшееся можно дописать по аналогии). Изучить код, приводимый в лабораторной работе, **понять**, как он работает, чтобы потом дописать оставшиеся фрагменты кода.
2. Реализовать код, согласно заданию, к лабораторной работе с учетом всех требований.
3. Продемонстрировать лабораторную работу. Ответить на вопросы преподавателя.

Варианты

Номер варианта	Название проекта	Номер варианта	Название проекта

1.	AirBomber	2.	DumpTruck
3.	Bulldozer	4.	ElectricLocomotive
5.	WarmlyShip	6.	Tank
7.	Airbus	8.	Battleship
9.	DoubleDeckerBus	10.	MotorBoat
11.	AirFighter	12.	GasolineTanker
13.	Excavator	14.	WarmlyLocomotive
15.	ContainerShip	16.	SelfPropelledArtilleryUnit
17.	AirplaneWithRadar	18.	Cruiser
19.	AccordionBus	20.	Catamaran
21.	Stormtrooper	22.	RoadTrain
23.	HoistingCrane	24.	Monorail
25.	Liner	26.	AntiAircraftGun
27.	Seaplane	28.	AircraftCarrier
29.	Trolleybus	30.	Sailboat

ЛАБОРАТОРНАЯ РАБОТА № 01.
КЛАССЫ, ОБЪЕКТЫ, ИНКАПСУЛЯЦИЯ

Цель

Научиться проводить объектную декомпозицию задачи. Познакомиться с понятиями классов, объектов и инкапсуляции. Изучить Windows Forms.

Задание

1. *Требуется сделать тестовое приложение по отслеживанию перемещения объекта по координатам. Необходимо реализовать следующую функциональность:*
    a. *Прорисовку объекта, согласно заданию. Объект отображать двумерно (вид сверху или сбоку, в зависимости от задания). Координаты расположения объекта считать левой верхней точкой области его прорисовки.*
    б. *Перемещение объекта по форме (вверх, вниз, вправо, влево), с учетом его характеристик (вес и скорость, определяющие «шаг» объекта) по нажатию кнопок. Объект не должен выходить за границы формы. Границы формы могут меняться.*

Приложение должно быть оформлено в виде desktop-приложения.

Проектирование

Планируется сделать движок, который будет задействован в нескольких проектах и частью этого движка будет отображение объекта на некой «карте» и перемещение этого объекта по ней. На данном этапе следует разработать набор методов для прорисовки объекта по координатам и возможности смены этих координат. Для отладки методов создадим отдельное приложение, на

котором будет возможность как отладить прорисовку объекта, так и проверить логику перемещения объекта в ограниченном пространстве.

Правильный подход к разработке ПО подразумевает, что каждый класс должен отвечать строго за один аспект поведения/работы. Поэтому выделим один класс, который будет отвечать за так называемую «сущность» – это объект, как правило, реального мира, который будет закодирован в программе. В частности, для нашей программы – это объект, который будет отображаться и перемещаться по «карте». Этот класс будет отвечать только за хранение информации по сущности, его характеристик. Второй класс будет отвечать непосредственно за прорисовку сущности на форме, его перемещения по ней. Второй класс, в частности, будет в себя включать объект первого класса. По сущности выделим следующие параметры: скорость, вес (будут влиять на расстояние перемещения) и цвет (будет использоваться при прорисовке). За хранение этих параметров будет отвечать первый класс. Второй класс будет хранить информацию по координатам объекта, его размерам и границам поля (эти данные будут использоваться для того, чтобы объект не «уходил» за границы). Также второй класс будет отвечать за прорисовку объекта на поле, его перемещение по нему, изменению границ формы (методы).

В задании было указано 4 возможных направления перемещения. Получается, в зависимости от того, какое направление выбрано, объект должен сместиться на форме на один шаг в выбранном направлении, если позволяют границы. Указание, в каком направлении двигаться объекту можно передавать по-разному:

- Строкой. Можно передавать строку с названием направления, но, если опечатались в названии при передаче, либо при сравнении в обработке, то перемещаться объект не будет.
- Числом. Под каждое направление можно завести соответствующее число, например, влево – это «1». Тогда объекту передаем число и он, в зависимости от значения перемещается в

нужном направлении. Но, тут есть вероятность запутаться, какое число за какое направление отвечает и передать ошибочное значение, либо значение, к которому не привязано направление.

- Перечисление. Самый лучший вариант. Под каждое направление движения задается конкретное значение в перечислении в виде названия. При передачи параметром легко определить какое значение передается за счет того, что оно написано словом, а так как значение выбирается строго из описанных вариантов в перечислении, то опечатки тут быть не может.

Исходя из описанных вариантов перемещения, получается, что в перечислении должно быть 4 значения возможных: вверх, вниз, вправо, влево.

Также для отладки отображения объекта, и проверки правильности работы логики перемещения, потребуется форма, на который объект будет отображаться и перемещаться (перемещение сделаем по кнопкам).

Реализация

У нас уже есть созданный проект, приступим к его наполнению. Первым делом, зададим класс для описания объекта. Описанные параметры зададим как свойства, но set у этих свойств зададим закрытый, чтобы только внутри класса можно было их менять, а для установления значений создадим отдельный метод. В проект можно добавлять новые элементы путем вызова меню через правую кнопку на названии проекта (не решения!) на вкладке «Обозреватель решения» (если нет такой вкладки, ищите в пункте меню «Вид»). В меню будет пункт «Добавить», в нем можно выбрать «Создать элемент…» или сразу «Создать класс» внизу списка (рисунок 1.1).

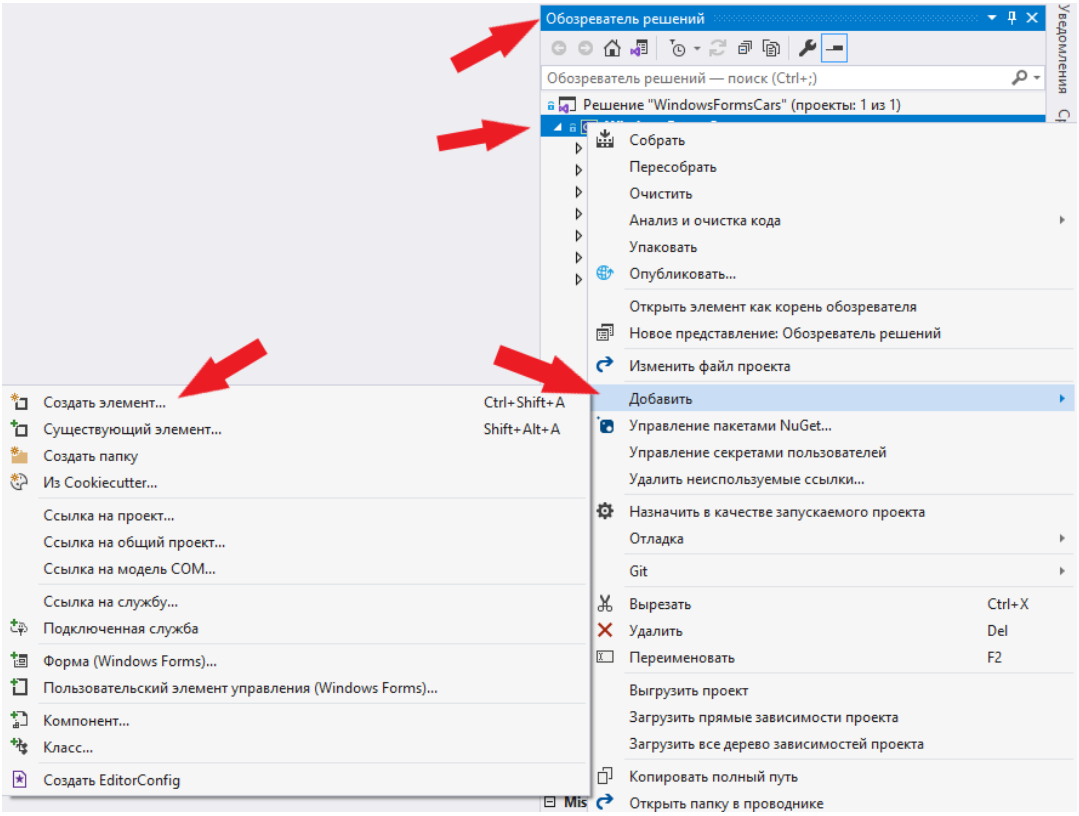


Рисунок 1.1 – Добавление элемента в проект

Далее выбираем пункт меню «Класс», внизу вводим название и создается в проекте файл и в нем класс. У файла и класса одинаковое название (рисунок 1.2).

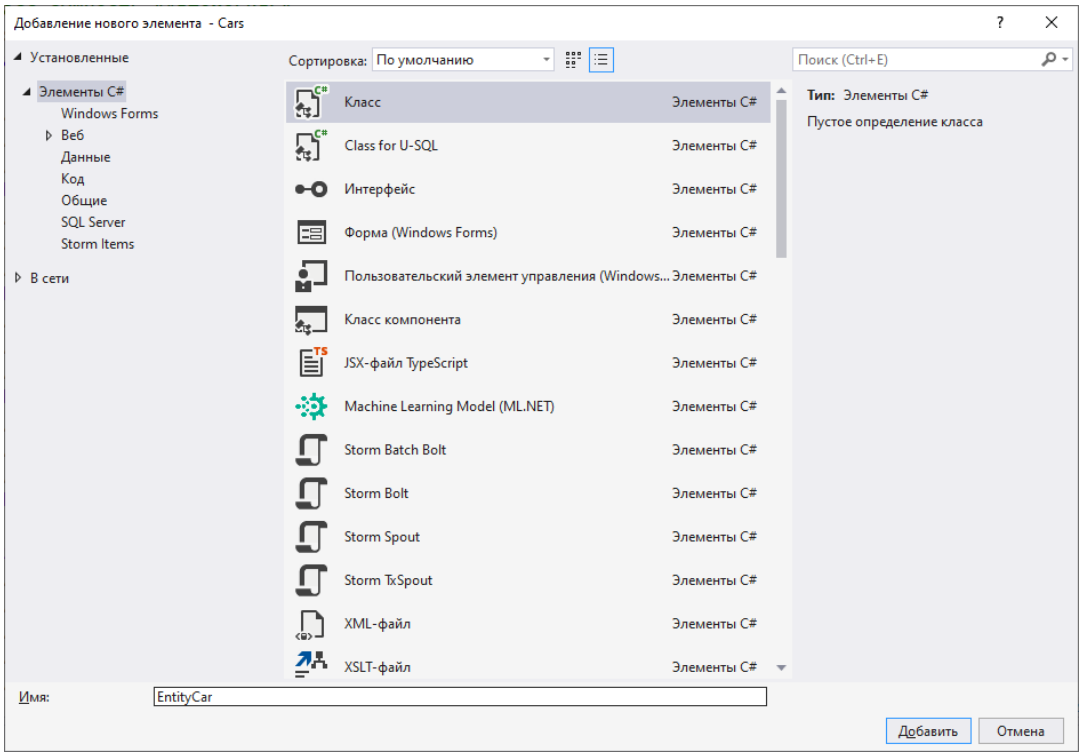


Рисунок 1.2 – Создание класса

Остается наполнить класс логикой (листинг 1.1).

```
namespace Cars
{
    /// <summary>
    /// Класс-сущность "Автомобиль"
    /// </summary>
    internal class EntityCar
    {
        /// <summary>
        /// Скорость
        /// </summary>
        public int Speed { get; private set; }
        /// <summary>
        /// Вес
        /// </summary>
        public float Weight { get; private set; }
        /// <summary>
        /// Цвет кузова
        /// </summary>
        public Color BodyColor { get; private set; }
        /// <summary>
        /// Шаг перемещения автомобиля
        /// </summary>
        public float Step => Speed * 100 / Weight;
        /// <summary>
        /// Инициализация полей объекта-класса автомобиля
        /// </summary>
        /// <param name="speed"></param>
        /// <param name="weight"></param>
        /// <param name="bodyColor"></param>
        /// <returns></returns>
        public void Init(int speed, float weight, Color bodyColor)
        {
            Random rnd = new();
            Speed = speed <= 0 ? rnd.Next(50, 150) : speed;
            Weight = weight <= 0 ? rnd.Next(40, 70) : weight;
            BodyColor = bodyColor;
        }
    }
}
```

Листинг 1.1 – Класс-сущность объекта «Автомобиль»

Далее нам потребуется перечисление, описывающие возможные направления движения. Для добавления перечисления в проект придется немного повозиться. Снова проходим этап добавления в проект нового элемента. В форме с перечнем возможных для создания элементов нет варианта «Перечисление». Потому делаем следующее, выбираем элемент класс, вводим название, как у нас называется перечисление «Direction» и создаем класс (рисунок 1.3). Далее просто ключевое слово «class» меняем на «enum» и прописываем значения для него (листинга 1.2).

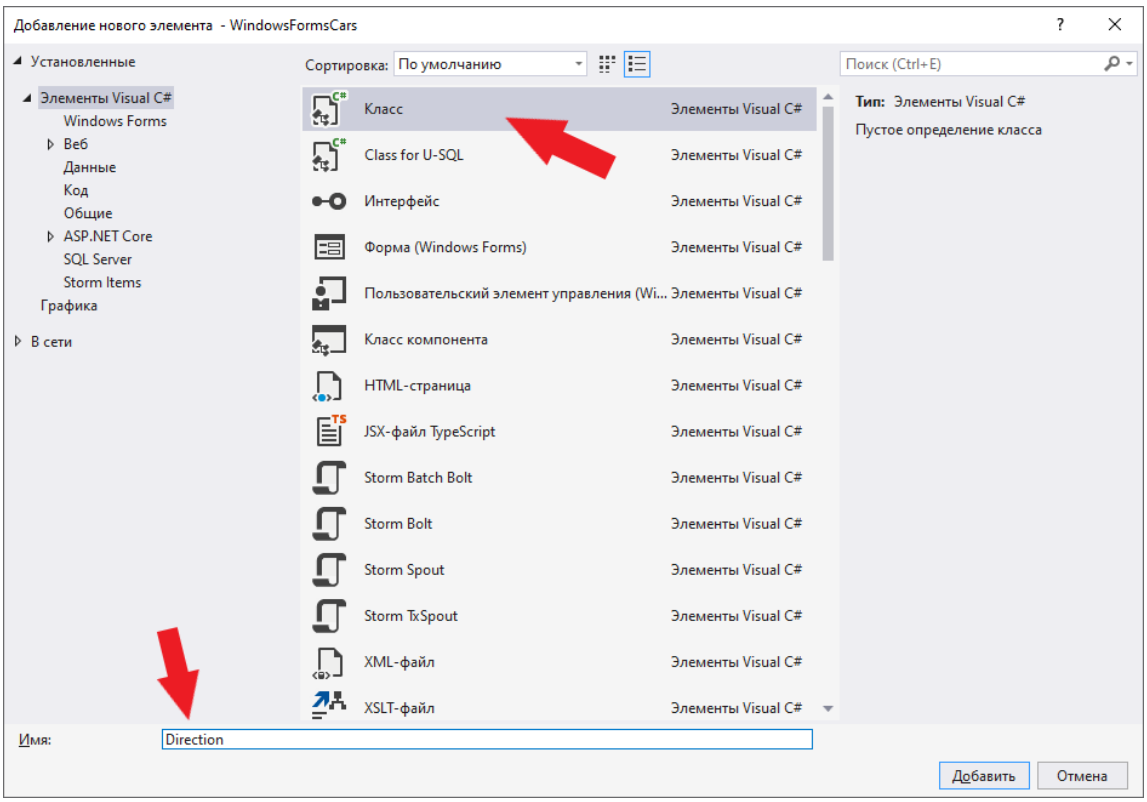


Рисунок 1.3 – Создание класса Direction

```
namespace Cars
{
    /// <summary>
    /// Направление перемещения
    /// </summary>
    internal enum Direction
    {
        Up = 1,
        Down = 2,
        Left = 3,
        Right = 4
    }
}
```

Листинг 1.2 – Перечисление направлений движения

Перейдем ко второму классу, ответственному за прорисовку объекта. Исходя из задания, сразу можно выделить 2 метода: для прорисовки объекта и для перемещения объекта. Дополнительно потребуется метод для заполнения параметров объекта (скорость, вес, цвет), метод для установки позиции и метод для смены границ формы прорисовки (если они будут меняться).

Рассмотрим детально методы:

- Инициализация свойств. Принимает 3 параметра (сколько свойств мы описали) для заполнения свойств.

- Установка начальной позиции. Метод будет принимать 4 параметра – координаты левой верхней точки, от которой будет идти прорисовка объекта и ширину, и высоту всего поля, на котором можно будет прорисовывать объект. Метод ничего не будет возвращать. Метод будет сохранять полученные параметры в полях класса «координаты прорисовки» и «границы поля». Важное условие, что все параметры больше 0 и координаты не выходят за границы полей.
- Метод изменения границ формы прорисовки. Метод будет принимать 2 параметра – новые границы формы прорисовки. Метод ничего не будет возвращать. Метод будет сохранять полученные параметры в полях класса «границы поля». Дополнительно, следует проверять, что объект не выходит за новые границы. Если выходит, то смещать объект, чтобы не выходил.
- Перемещение объекта. Метод будет принимать 1 параметр – направление перемещения. Метод ничего не будет возвращать. На первом шаге будет выполняться проверка, что у нас заполнена информация о границах поля (для этого поля завернуты в Nullable, т.е. они могут принимать значение null). Если этих данных нет, то перемещение выполняться не будет. Далее, в зависимости от значения параметра, изменять значение у одного из свойств класса «координаты прорисовки», если позволяют «границы поля».
- Прорисовка объекта. Метод будет получать объект от класса Graphics через который будет происходить прорисовка. Метод ничего не будет возвращать. В начале будет проверка, что у нас заполнены координаты, по которым будет прорисовываться объект. Если этих данных нет, то прорисовка выполняться не будет. Метод будет выполнять прорисовку объекта с

использованием значений свойств класса «координаты прорисовки».

Часть логики в классе DrawningCar не доделана, потребуется дописывать самостоятельно, на основе имеющейся логики (листинг 1.3).

```
namespace Cars
{
    /// <summary>
    /// Класс, отвечающий за прорисовку и перемещение объекта-сущности
    /// </summary>
    internal class DrawningCar
    {
        /// <summary>
        /// Класс-сущность
        /// </summary>
        public EntityCar Car { private set; get; }
        /// <summary>
        /// Левая координата отрисовки автомобиля
        /// </summary>
        private float _startPosX;
        /// <summary>
        /// Верхняя кооридната отрисовки автомобиля
        /// </summary>
        private float _startPosY;
        /// <summary>
        /// Ширина окна отрисовки
        /// </summary>
        private int? _pictureWidth = null;
        /// <summary>
        /// Высота окна отрисовки
        /// </summary>
        private int? _pictureHeight = null;
        /// <summary>
        /// Ширина отрисовки автомобиля
        /// </summary>
        private readonly int _carWidth = 80;
        /// <summary>
        /// Высота отрисовки автомобиля
        /// </summary>
        private readonly int _carHeight = 50;
        /// <summary>
        /// Инициализация свойств
        /// </summary>
        /// <param name="speed">Скорость</param>
        /// <param name="weight">Вес автомобиля</param>
        /// <param name="bodyColor">Цвет кузова</param>
        public void Init(int speed, float weight, Color bodyColor)
        {
            Car = new EntityCar();
            Car.Init(speed, weight, bodyColor);
        }
        /// <summary>
        /// Установка позиции автомобиля
        /// </summary>
        /// <param name="x">Координата X</param>
        /// <param name="y">Координата Y</param>
        /// <param name="width">Ширина картинки</param>
        /// <param name="height">Высота картинки</param>
        public void SetPosition(int x, int y, int width, int height)
```

```
        {
            // TODO проверки
            _startPosX = x;
            _startPosY = y;
            _pictureWidth = width;
            _pictureHeight = height;
        }
        /// <summary>
        /// Изменение направления перемещения
        /// </summary>
        /// <param name="direction">Направление</param>
        public void MoveTransport(Direction direction)
        {
            if (!_pictureWidth.HasValue || !_pictureHeight.HasValue)
            {
                return;
            }
            switch (direction)
            {
                // вправо
                case Direction.Right:
                    if (_startPosX + _carWidth + Car.Step.Value < _pictureWidth)
                    {
                        _startPosX += Car.Step.Value;
                    }
                    break;
                //влево
                case Direction.Left:
                    // TODO: Продумать логику
                    break;
                //вверх
                case Direction.Up:
                    // TODO: Продумать логику
                    break;
                //вниз
                case Direction.Down:
                    if (_startPosY + _carHeight + Car.Step.Value < _pictureHeight)
                    {
                        _startPosY += Car.Step.Value;
                    }
                    break;
            }
        }
        /// <summary>
        /// Отрисовка автомобиля
        /// </summary>
        /// <param name="g"></param>
        public void DrawTransport(Graphics g)
        {
            if (_startPosX < 0 || _startPosY < 0
                || !_pictureHeight.HasValue || !_pictureWidth.HasValue)
            {
                return;
            }
            Pen pen = new(Color.Black);
            //границы автомобиля
            g.DrawEllipse(pen, _startPosX, _startPosY, 20, 20);
            g.DrawEllipse(pen, _startPosX, _startPosY + 30, 20, 20);
            g.DrawEllipse(pen, _startPosX + 70, _startPosY, 20, 20);
            g.DrawEllipse(pen, _startPosX + 70, _startPosY + 30, 20, 20);
            g.DrawRectangle(pen, _startPosX - 1, _startPosY + 10, 10, 30);
            g.DrawRectangle(pen, _startPosX + 80, _startPosY + 10, 10, 30);
            g.DrawRectangle(pen, _startPosX + 10, _startPosY - 1, 70, 52);
```

```
            //задние фары
            Brush brRed = new SolidBrush(Color.Red);
            g.FillEllipse(brRed, _startPosX, _startPosY, 20, 20);
            g.FillEllipse(brRed, _startPosX, _startPosY + 30, 20, 20);

            //передние фары
            Brush brYellow = new SolidBrush(Color.Yellow);
            g.FillEllipse(brYellow, _startPosX + 70, _startPosY, 20, 20);
            g.FillEllipse(brYellow, _startPosX + 70, _startPosY + 30, 20, 20);

            //кузов
            Brush br = new SolidBrush(Car?.BodyColor ?? Color.Black);
            g.FillRectangle(br, _startPosX, _startPosY + 10, 10, 30);
            g.FillRectangle(br, _startPosX + 80, _startPosY + 10, 10, 30);
            g.FillRectangle(br, _startPosX + 10, _startPosY, 70, 50);

            //стекла
            Brush brBlue = new SolidBrush(Color.LightBlue);
            g.FillRectangle(brBlue, _startPosX + 60, _startPosY + 5, 5, 40);
            g.FillRectangle(brBlue, _startPosX + 20, _startPosY + 5, 5, 40);
            g.FillRectangle(brBlue, _startPosX + 25, _startPosY + 3, 35, 2);
            g.FillRectangle(brBlue, _startPosX + 25, _startPosY + 46, 35, 2);

            //выделяем рамкой крышу
            g.DrawRectangle(pen, _startPosX + 25, _startPosY + 5, 35, 40);
            g.DrawRectangle(pen, _startPosX + 65, _startPosY + 10, 25, 30);
            g.DrawRectangle(pen, _startPosX, _startPosY + 10, 15, 30);
        }
        /// <summary>
        /// Смена границ формы отрисовки
        /// </summary>
        /// <param name="width">Ширина картинки</param>
        /// <param name="height">Высота картинки</param>
        public void ChangeBorders(int width, int height)
        {
            _pictureWidth = width;
            _pictureHeight = height;
            if (_pictureWidth <= _carWidth || _pictureHeight <= _carHeight)
            {
                _pictureWidth = null;
                _pictureHeight = null;
                return;
            }
            if (_startPosX + _carWidth > _pictureWidth)
            {
                _startPosX = _pictureWidth.Value - _carWidth;
            }
            if (_startPosY + _carHeight > _pictureHeight)
            {
                _startPosY = _pictureHeight.Value - _carHeight;
            }
        }
    }
}
```

Листинг 1.3 – Класс, отвечающий за прорисовку и перемещение объекта-автомобиля

Замечание! Метод прорисовки объекта следует реализовывать на последнем (или предпоследнем) этапе разработки, чтобы была возможность видеть результат на форме.

Остается последний элемент проекта – форма для вывода объекта. Существующую форму Form1 переименовываем согласно заданию. Для этого на вкладке «Обозреватель решения» вызовем меню на файле Form1 и найдем пункт «Переименовать». Изменим имя и нажмем «Enter». Может последовать вопрос, переименовать ли класс (имя файла может не совпадать с именем класса, который в нем описан). В таком случае нажимаем «Ok». Во вкладке «Обозреватель решения» два раза кликнем по файлу формы. Откроется интерфейс формы. Изменим размер формы (либо мышкой за нижний правый угол, либо через свойства найти там «Size») и подпись формы с «Form1» на соответствующее тематике задания (свойство «Text»). Размеры зададим 900 на 500 (необязательно, можно выставить иные). Рядом есть интересной свойство «StartPosition», которое отвечает за место отображения формы на экране. По умолчанию стоит случайное место. Можно выбрать по центру экрана или по центру родителя и т.д. Выставим «По центру экрана». Также есть свойство «WindowState» которое отвечает за размер формы, либо по заданным размерам, либо на весь экран, либо в свернутом режиме.

На форме потребуется: элемент для вывода изображения, элемент для отображения характеристик объекта, кнопка для создания объекта и кнопки для перемещения объекта. Рассмотрим все элементы отдельно.

Для отображения значений характеристик объекта на форме используется элемент StatusStrip. В нем добавлены 3 элемента ToolStripStatusLabel, каждый их которых будет отвечать за вывод одного из свойств объекта (скорость, вес, цвет). StatusStrip по умолчанию располагается внизу формы и привязан к его нижней границе.

Перенесем на форму элемент StatusStrip из вкладки «Панель элементов», добавим в него 3 StatusLabel, каждому зададим корректное имя и подпись (рисунок 1.4).

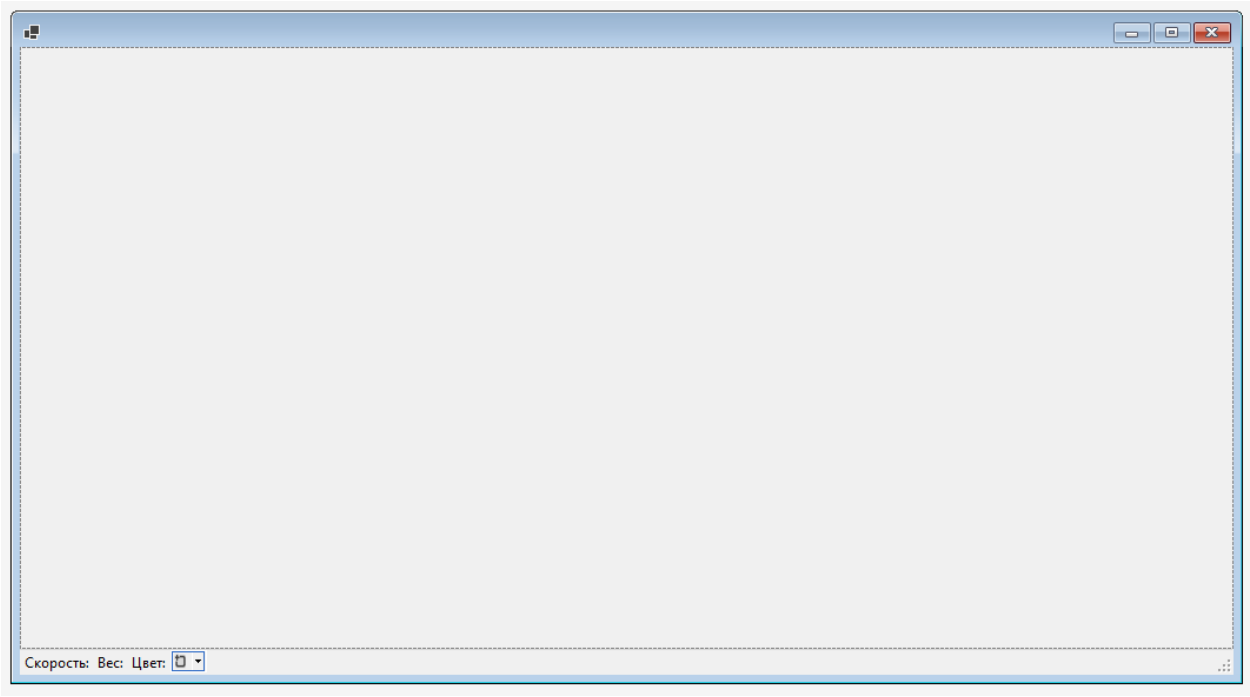


Рисунок 1.4 – Добавление StatusStrip ан форму

Элементом для вывода изображения будет выступать элемент PictureBox, который может выводить в частности, рисунки, формируемые в коде. Ряд его свойств были изменены для лучшего отображения изображения. Во-первых, свойство Dock выставлено значение Fill, чтобы PictureBox автоматически растягивался на всю возможную площадь родительского элемента (в данном случае, этим элементом выступает сама форма). Также, чтобы в коде было возможно получать актуальную информацию о ширине и высоте элемента нужно в свойстве «SizeMode» выставить значение «AutoSize». Перенесем из вкладки «Панель элементов» этот элемент на форму. Далее в свойствах у этого элемента найдем свойство «Dock» и выберем центральный квадратик (рисунок 1.5). И выставляем свойство у «SizeMode» в значение «AutoSize». Также, чтобы PictureBox не уходил под StatusStrip вызовем меню у PictureBox (правая кнопка мыши по элементу PictureBox на форме) и выберем вариант отображения «поверх» остальных.

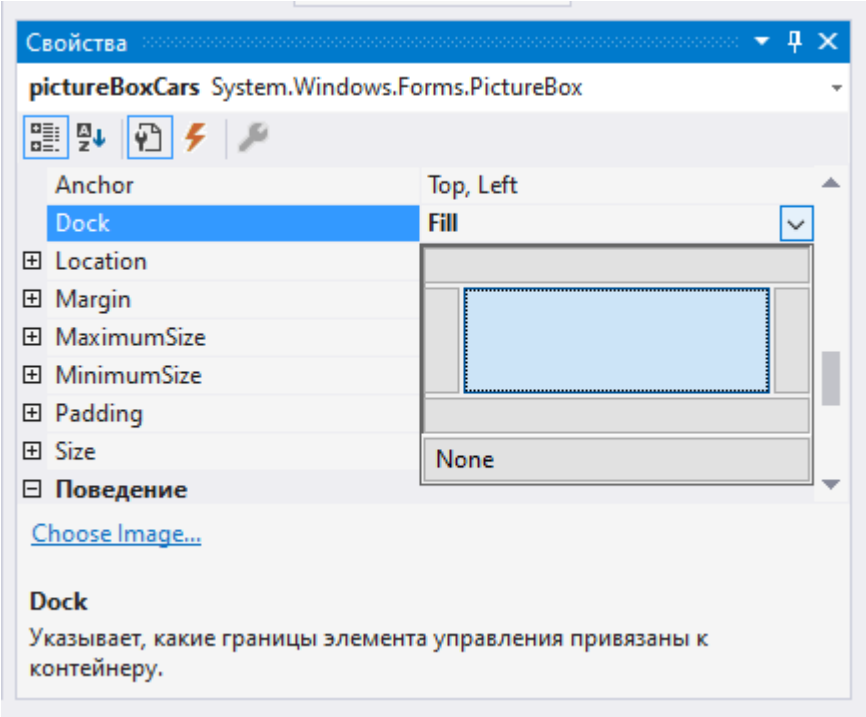


Рисунок 1.5 – Настройка элемента PictureBox

Для кнопки, отвечающей за создание объекта, каких-то особых настроек не требуется, только задать подпись.

Для кнопок, отвечающих за перемещения, задали картинки-стрелочки, которые указывают направление перемещения.

Добавим на форму 5 кнопок. Для кнопки создания объекта укажем имя buttonCreate и в свойстве «Text» изменим на «Создать». Кнопку разместим в левом нижнем углу. У кнопки найдем свойство «Anchor» и изменим его свойства с «Top, Left» на «Bottom, Left». Это свойство отвечает также за привязку элемента к краям формы, как и «Dock», но имеет другую логику работы. Для четырех других кнопок зададим размер 30 на 30 и разместим их в правом нижнем углу. Дадим им соответствующие имена: buttonLeft, buttonRight, buttonUp, buttonDown. Также у этих 4 кнопок найдем свойство «Anchor» и изменим его свойства с «Top, Left» на «Bottom, Right». В интернете найти и скачать картинку «Стрелка» (любую). Сделать (или скачать) 4 версии стрелки в разные направления. Выберем одну из 4 кнопок, найдем у нее свойство «BackgroundImage» и нажмем кнопку «…» у нее. Откроется окошка

выбора ресурса. Внизу есть кнопка «Импорт». Нажимаем ее и загружаем все 4 рисунка (рисунок 1.6).

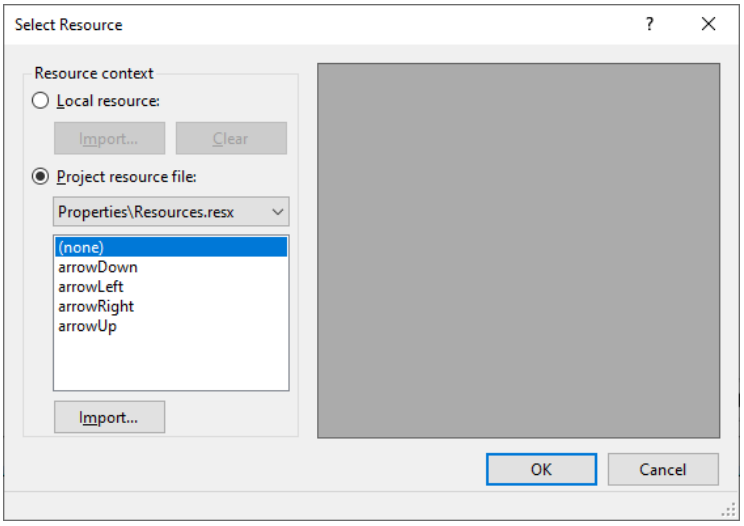


Рисунок 1.6 – Добавление ресурсов

Далее для кнопок выбираем соответствующие стрелки. В свойстве «BackgroundImageLayout» указать значение «Zoom». И в свойстве «Text» отчистить значение.

Для того, чтобы задать логику на какое-то действие пользователя или событие на форме (например, загрузка формы или нажатие кнопки, или ввод текста в текстовое поле) есть два варианта:

- Два раза «кликнуть» по элементу на дизайнере форм (в таком случае создастся метод обработки только на одно, предопределенное действие для элемента);
- Во вкладке свойства для выбранного элемента перейти на события, найти нужное событие и дважды кликнуть по полю справа (будем рассмотрено ниже);

Для кнопки «Создать» два раза кликнем по ней и у нас откроется файл с логикой для формы и там будет создан метод обработки клика по кнопке (рисунок 1.7).

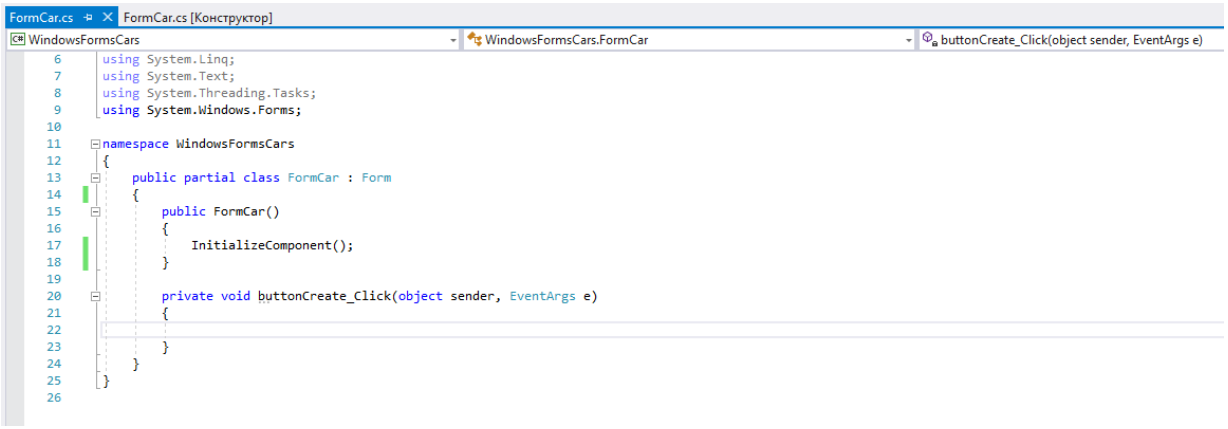


Рисунок 1.7 – Созданный метод обработки нажатия кнопки

Для обработок нажатия кнопок управления сделаем другой алгоритм. Сперва в логике пропишем метод, назовем его ButtonMove_Click (скопируем метод ButtonCreate_Click и поменяем название, а логику метода оставим пока пустой). Если внимательно посмотреть на передаваемые в метод параметры, то первым будет идти объект, который вызвал метод, вторым – информация по событию. Второй параметр нас пока не будет интересовать, а вот из первого будем получать информацию о кнопке, которая вызвала событие. Будем из него вытаскивать имя кнопки по нему определять, куда надо сдвинуть объект. В зависимости от имени будет вызываться метод класса-работы с объектом для перемещения объекта и передаваться параметр-перечисление, в каком направлении переместить объекта и затем вызовем метод прорисовки.

Вернемся на форму. Выделим все 4 кнопки и в свойствах перейдем на события. Найдем событие «Click» в правом поле вызовем выпадающий список и выберем метод ButtonMove_Click (рисунок 1.8).

Если метода нет, а нужно его создать, то достаточно два раза кликнуть по правой колонке напротив события, которое требуется обработать.

Останется отлавливать изменение формы, чтобы менять границы прорисовки. Для этого есть 2 варианта:

1. Ловить изменения формы
2. Ловить изменения PictureBox

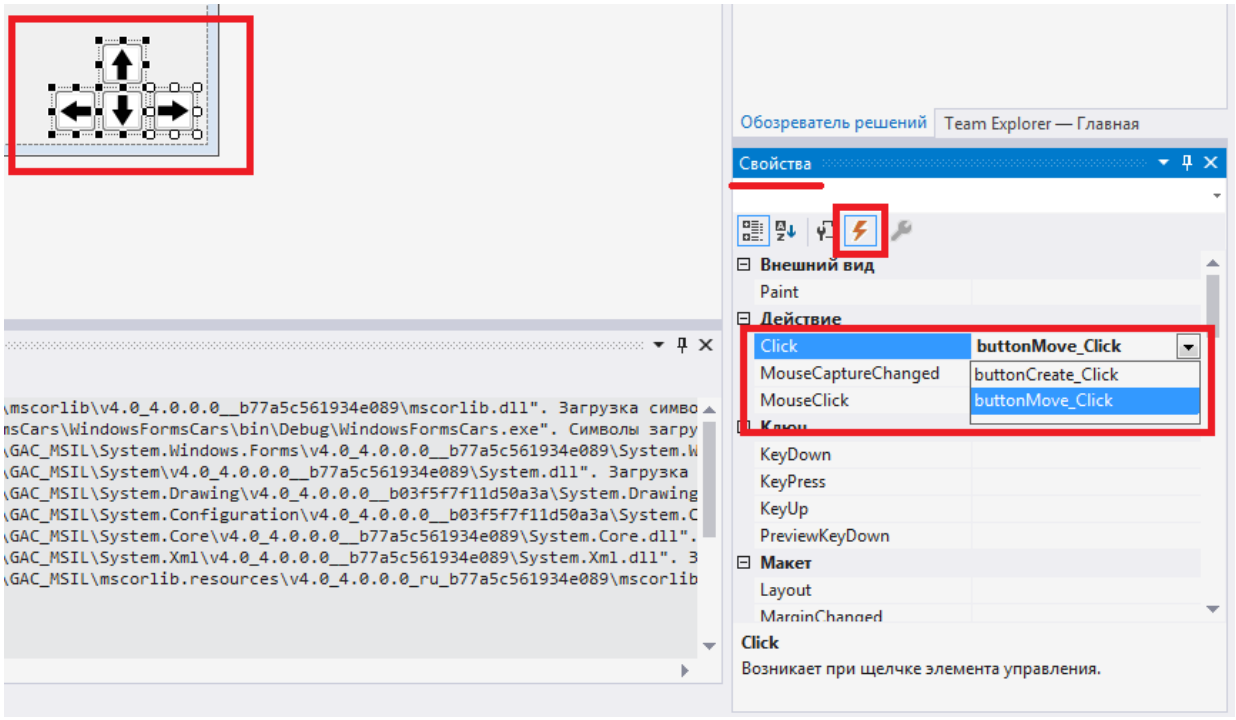


Рисунок 1.8 – Назначение метода обработки для кнопок управления

Так как PictureBox привязан к размерам формы любой из вариантов возможен. В данном случае, выберем 2 вариант. У PictureBox есть событие Resize и к нему можно привязать метод с логикой (как привязывать разобрали чуть выше).

Замечание! Возможна ситуация, когда требуется удалить метод обработки за ненадобностью и после удаления метода из логики дизайнер формы может выдать такую картину (рисунок 1.9).

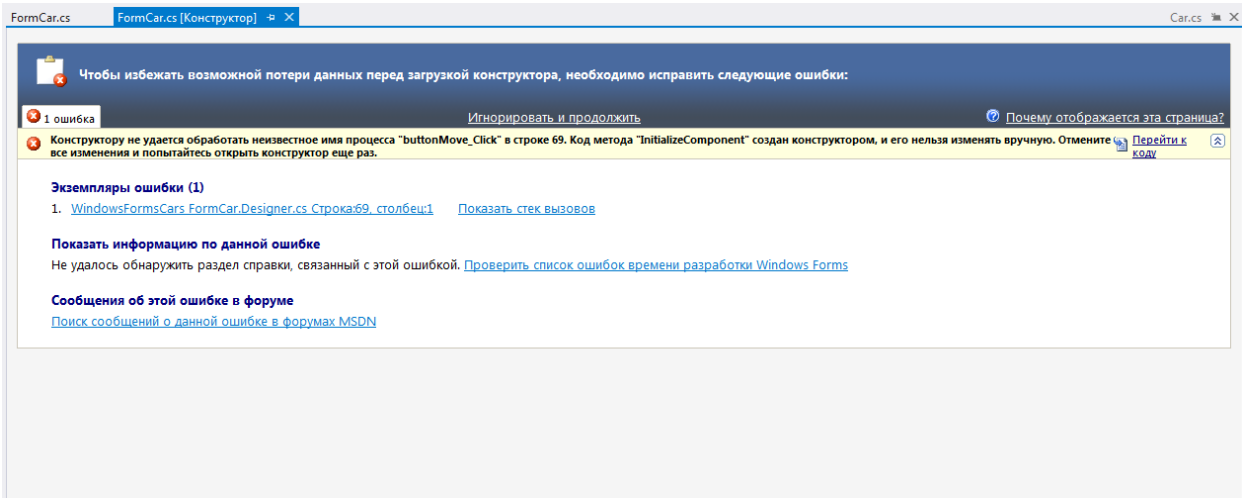


Рисунок 1.9 – Ошибка в дизайнере формы

В таком случае нужно перейти в класс дизайнера формы (например, FormCar.Designer.cs), найти там строку (или строки) с ошибками и устранить их.

Таким образом получилась следующая форма (рисунок 1.10).

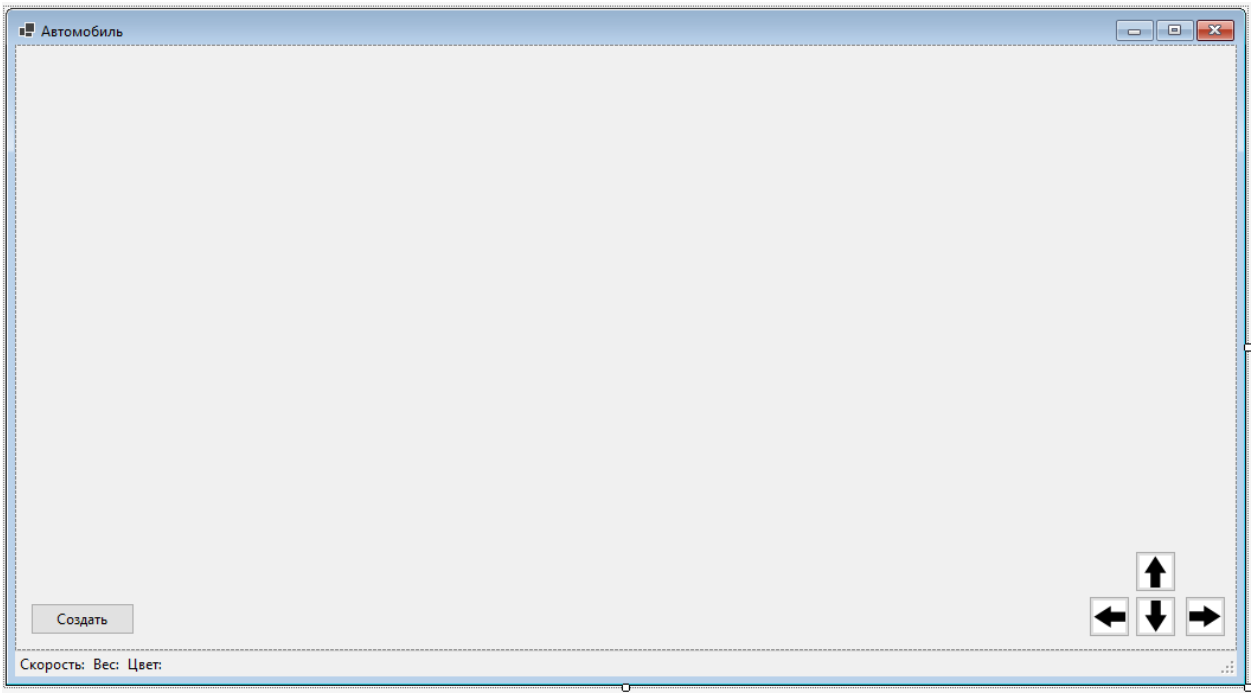


Рисунок 1.10 – Форма для отображения объекта-автомобиля

Теперь логика формы. Для нормальной работы с объектом в разных методах логики формы, вводится поле от класса для работы с объектом. Во всех методах идет работа с этим полем. Задается отдельный метод для прорисовки объекта, так как прорисовка требуется при всех возможных действиях (создание, перемещение, изменение границ формы). Метод прорисовки будет состоять из следующих действий:

- создать объект от класса Bitmap, которое будет представлять из себя «полотно», на котором будем рисовать объект;
- от объекта Bitmap получить объект Graphics, с помощью которого будет прорисовываться объект;
- вызвать метод прорисовки класса DrawningCar для рисования объекта на полотне;
- передать полученный рисунок на pictureBoxCar.

В методе создания объекта (вызывается при нажатии на кнопку «Создать») поле-объект инициализируется и вызывается метод инициализации его свойств (значения свойств задаются случайным образом) и метод установки изначальной позиции (примерно в левом верхнем углу формы), после чего вызывает метод прорисовки. Для перемещения задан один метод на все 4 кнопки, исходя из того какая кнопка вызывает метод (определяется через название кнопки), вызывается метод перемещения объекта и передается параметр – направление перемещения из перечисления, после чего вызывает метод прорисовки. Метод изменения размеров форм вызывает соответствующий метод поля-объекта и передает туда новые параметры PictureBox, после чего вызывает метод прорисовки объекта (листинг 1.4).

```
namespace Cars
{
    public partial class FormCar : Form
    {
        private DrawningCar _car;

        public FormCar()
        {
            InitializeComponent();
        }
        /// <summary>
             /// Метод прорисовки машины
             /// </summary>
             private void Draw()
        {
            Bitmap bmp = new(pictureBoxCar.Width, pictureBoxCar.Height);
            Graphics gr = Graphics.FromImage(bmp);
            _car?.DrawTransport(gr);
            pictureBoxCar.Image = bmp;
        }
        /// <summary>
        /// Обработка нажатия кнопки "Создать"
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void ButtonCreate_Click(object sender, EventArgs e)
        {
            Random rnd = new();
            _car = new DrawningCar();
            _car.Init(rnd.Next(100, 300), rnd.Next(1000, 2000),
Color.FromArgb(rnd.Next(0, 256), rnd.Next(0, 256), rnd.Next(0, 256)));
            _car.SetPosition(rnd.Next(10, 100), rnd.Next(10, 100),
pictureBoxCar.Width, pictureBoxCar.Height);
            toolStripStatusLabelSpeed.Text = $"Скорость: {_car.Car.Speed}";
            toolStripStatusLabelWeight.Text = $"Вес: {_car.Car.Weight}";
            toolStripStatusLabelBodyColor.Text = $"Цвет:
{_car.Car.BodyColor.Name}";
```

```
            Draw();
        }
        /// <summary>
        /// Изменение размеров формы
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void ButtonMove_Click(object sender, EventArgs e)
        {
            //получаем имя кнопки
            string name = ((Button)sender)?.Name ?? string.Empty;
            switch (name)
            {
                case "buttonUp":
                    _car?.MoveTransport(Direction.Up);
                    break;
                case "buttonDown":
                    _car?.MoveTransport(Direction.Down);
                    break;
                case "buttonLeft":
                    _car?.MoveTransport(Direction.Left);
                    break;
                case "buttonRight":
                    _car?.MoveTransport(Direction.Right);
                    break;
            }
            Draw();
        }
        /// <summary>
        /// Изменение размеров формы
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void PictureBoxCar_Resize(object sender, EventArgs e)
        {
            _car?.ChangeBorders(pictureBoxCar.Width, pictureBoxCar.Height);
            Draw();
        }
    }
}
```

Листинг 1.4 – Логика формы для работы с объектом

В результате, при запуске и нажатии кнопки «Создать», получаем следующую картину (рисунок 1.11).

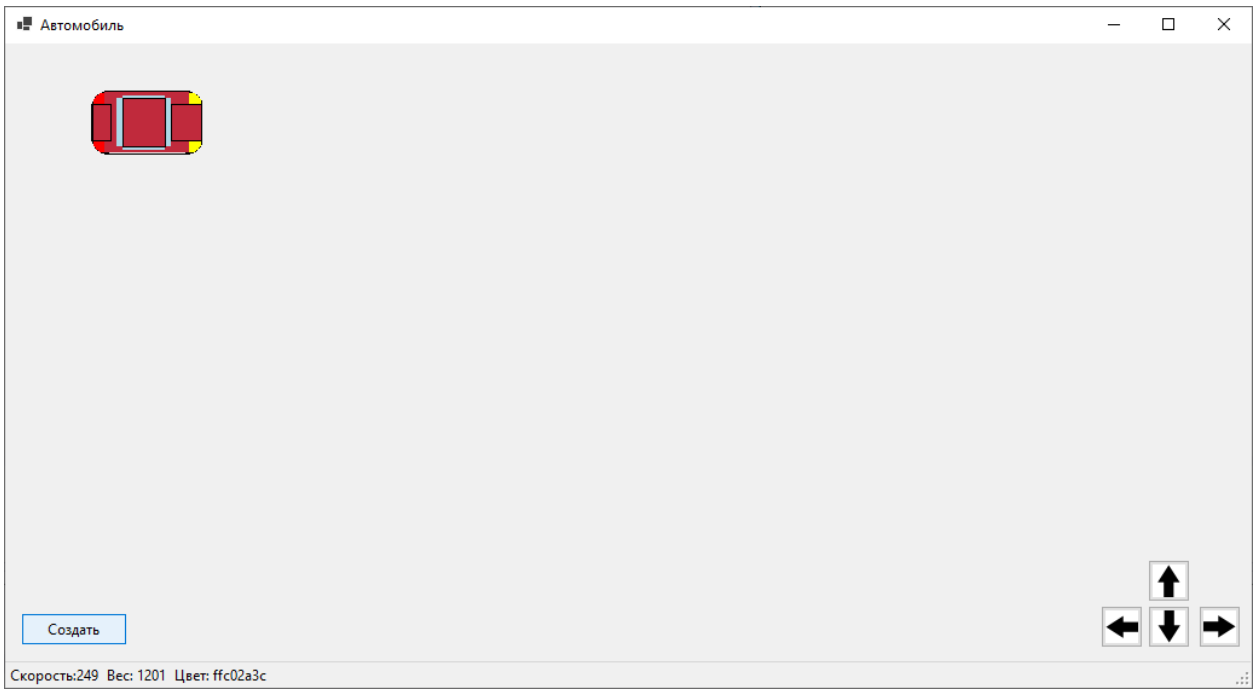


Рисунок 1.11 – Пример работы программы

Последний момент. Посмотрим класс Program.cs (листинг 1.5).

При запуске программы всегда вызывается метод Main класса Program. Поэтому, если требуется выполнять какие-то логические действия (например, загрузка конфигурации или вызов формы ввода логина/пароля) до вызова основной формы (строка Application.Run(new FormCar());) то их следует прописывать в методе Main.

```
namespace Cars
{
    internal static class Program
    {
        /// <summary>
        ///  The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            // To customize application configuration such as set high DPI
settings or default font,
            // see https://aka.ms/applicationconfiguration.
            ApplicationConfiguration.Initialize();
            Application.Run(new FormCar());
        }
    }
}
```

Листинг 1.5 – Код класса Program.cs

Требования

1. Платформа для проектов – .Net версии 6.0
2. Название проекта должно соответствовать логике задания.
3. Название форм, классов, свойств классов должно соответствовать логике задания.
4. Все элементы форм (заголовки форм, текст в label и т.д.) должны иметь подписи на одном языке (или все русским, или все английским).
5. В качестве базы для отображения объекта по заданию использовать изображения, представленные во вариантах.
6. Прорисовку объекта выстраивать таким образом, чтобы она была **ниже и правее** от координат прорисовки (x и y).
7. Объект на форме не должен выходить за границы формы при перемещении.
8. Доделать проверку в методе SetPosition (размеры форм должны вмещать объект и, если объект выходит за границы формы, то «возвращать» его в границы).
9. Доделать логику в методе MoveTransport (имеющуюся логику в MoveTransport не менять!)

Порядок сдачи базовой части

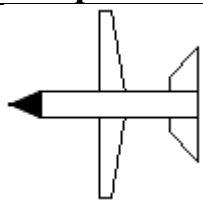
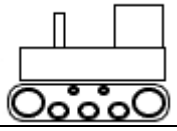
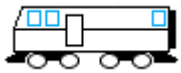
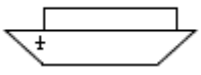
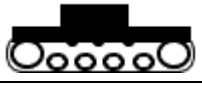
1. Создать объект.
2. Довести объект до левой границы поля, показать, что он не выходит за нее.
3. Довести объект до верхней границы поля, показать, что он не выходит за нее.
4. Довести объект до правой границы поля, показать, что он не выходит за нее.

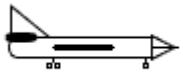
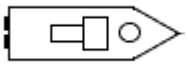
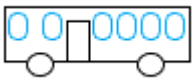
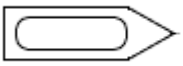
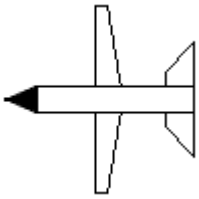
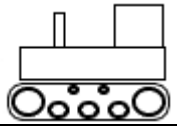
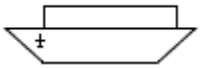
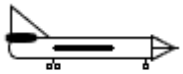
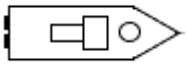
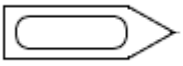
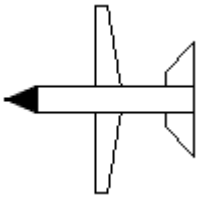
5. Довести объект до нижней границы поля, показать, что он не выходит за нее.

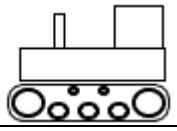
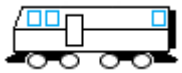
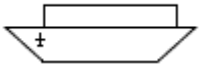
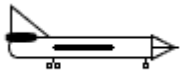
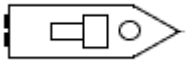
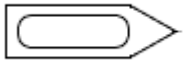
Контрольные вопросы к базовой части

1. Показать объявление и инициализацию объекта / Прописать в логике класса прорисовки возможность перемещения по диагонали.
2. Где задается начальное положение объекта на форме? / Сделать размещение объекта в другом углу формы (в зависимости от размеров формы).
3. Где задается шаг перемещения объекта? / В метод Init класса прорисовки вместо имеющихся параметров передавать готовый объект от класса-сущности.

Варианты

Вар.	Объект	Изображение
1.	Военный самолет	
2.	Грузовик	
3.	Трактор	
4.	Локомотив	
5.	Корабль	
6.	Бронированная машина	

Вар.	Объект	Изображение
7.	Самолет	
8.	Военный корабль	
9.	Автобус	
10.	Лодка	
11.	Военный самолет	
12.	Грузовик	
13.	Трактор	
14.	Локомотив	
15.	Корабль	
16.	Бронированная машина	
17.	Самолет	
18.	Военный корабль	
19.	Автобус	
20.	Лодка	
21.	Военный самолет	

Вар.	Объект	Изображение
22.	Грузовик	
23.	Трактор	
24.	Локомотив	
25.	Корабль	
26.	Бронированная машина	
27.	Самолет	
28.	Военный корабль	
29.	Автобус	
30.	Лодка	

ЛАБОРАТОРНАЯ РАБОТА №2.
НАСЛЕДОВАНИЕ. АБСТРАКТНЫЕ КЛАССЫ. ИНТЕРФЕЙСЫ

Цель

Познакомится с понятиями наследование. Научиться создавать абстрактные классы и интерфейсы и наследоваться от них.

Задание

1. *В тестовое приложение по отслеживанию перемещения объекта по координатам необходимо внести следующие изменения:*
    а. *Инициализацию перенести в конструкторы классов.*
    б. *Создать классы, которые позволяют прорисовывать «продвинутую» версию объекта. Для прорисовки использовать логику прорисовки исходного объекта с дополнением новых элементов. Использовать отдельный цвет для новых элементов, а также определять прорисовывать ли их через свойства-признаки (отдельное свойство на каждый новый элемент прорисовки).*
    в. *Классы по прорисовке и перемещению объектов привязать к интерфейсу IDrawningObject (будет приведен код).*
    г. *Создать абстрактный класс «Карта» для отображения фона. На карте должны быть представлены участки 2-х типов, по которым объект IDrawningObject может перемещаться и по которым не может перемещаться. Наследник должен отвечать за раскраску участков и алгоритм расстановки участков.*
    д. *На имеющейся форме сделать возможным создавать объекты простого и «продвинутого» объекта. Создать*

форму, аналогичную существующей, где работа с объектами будет вестись через класс «Карта».

Проектирование

По инициализации все просто, вызов отдельных методов для инициализации полей класса – плохой подход, так как программист, использующий такой класс может попросту забыть прописать вызов метода и поля класса в таком случае останутся незаполненными. Куда проще сделать конструктор, который будет заполнять поля класса значениями, в таком случае программист просто не сможет пропустить этап инициализации полей класса (особенно, если у класса указан только один конструктор).

Далее следует создать «продвинутый» объект. Так как этот объект будет являться усовершенствованной версией имеющегося объекта, то логично будет воспользоваться механизмами наследования, чтобы не дублировать код. Для нового объекта также потребуется 2 класса: класс-сущность и класс для прорисовки. Новый класс-сущность наследуем от имеющего класса-сущности и расширяем его новыми полями:

- дополнительный цвет;
- признак отображения обвеса;
- признак отображения заднего спойлера;
- признак отображения гоночной полосы.

Также делаем конструктор для инициализации полей с вызовом конструктора базового класса.

Класс для прорисовки и перемещения «продвинутого» объекта также унаследуем от класса работы с обычным объектом. Но предварительно исходный класс надо немного модифицировать. В исходном классе были методы инициализации (заменен на конструктор), метод установки начальной позиции, метод перемещения, метод смены границ форм и метод прорисовки. Для работы первых трех методов из этого списка необходимо знать размеры отображаемого объекта. Так как «продвинутый» объект может быть больше

исходного, то чтобы не дублировать логику, сделаем еще один конструктор, который будет вызывать только в дочерних классах и передавать в этот конструктор дополнительно и новые размеры объекта. Последний метод будет более значительно отличаться в классе-наследнике, но при этом, он будет использовать логику исходного метода. Чтобы сделать все правильно, сделаем метод виртуальным, чтобы его корректно переопределить в классе-наследнике.

Для связывания нашего проекта с будущими решениями вводится интерфейс, который определят поведение объекта при перемещении и прорисовке (листинг 2.1). Через данный интерфейс наше решение будет «вводится» в другие проект.

```
namespace Cars
{
    /// <summary>
    /// Интерфейс для работы с объектом, прорисовываемым на форме
    /// </summary>
    internal interface IDrawningObject
       {
              /// <summary>
              /// Шаг перемещения объекта
              /// </summary>
              public float Step { get; }
              /// <summary>
              /// Установка позиции объекта
              /// </summary>
              /// <param name="x">Координата X</param>
              /// <param name="y">Координата Y</param>
              /// <param name="width">Ширина полотна</param>
              /// <param name="height">Высота полотна</param>
              void SetObject(int x, int y, int width, int height);
              /// <summary>
              /// Изменение направления пермещения объекта
              /// </summary>
              /// <param name="direction">Направление</param>
              void MoveObject(Direction direction);
              /// <summary>
              /// Отрисовка объекта
              /// </summary>
              /// <param name="g"></param>
              void DrawningObject(Graphics g);
              /// <summary>
              /// Получение текущей позиции объекта
              /// </summary>
              /// <returns></returns>
              (float Left, float Right, float Top, float Bottom)
GetCurrentPosition();
       }
}
```

Листинг 2.1 – Код интерфейса для работы с прорисовываемым объектом

Если посмотреть по методам, то в нашем классе прорисовки и перемещения простого объекта методы отличаются только названием (правда, один из наших методов еще и виртуальный). При наследовании нашего класса от интерфейса можно пойти 3 путями:

1. Переименовать наши методы под объявленные в интерфейсы.
2. Сделать в классе новые методы, объявленные в интерфейсе и вызывать там уже имеющиеся.
3. Сделать отдельный класс, который будет наследником от интерфейса и при этом в логике работать с методами нашего класса прорисовки через поле-объект.

Мы пойдем третьим вариантом, через отдельный класс.

Перейдем к абстрактному классу. В нем будет реализована большая часть логики работы с картой: ее генерация, работа с объектом. За наследниками останется только небольшая роль раскрашивать карту. В классе потребуется как минимум, массив двумерный под карту, поле под объект от IDrawningObject и методы для создания карты, прорисовки карты с объектом и перемещение объекта по карте. Также будут 3 абстрактных метода: прорисовка участка, по которому можно двигаться и прорисовка участка, по которому нельзя двигаться и генерация карты.

Для возможности создания на форме объекта «продвинутого» варианта потребуется добавить кнопку.

Также сделаем форму, на основе существующей (можно сделать аккуратный copy-paste). В логике новый формы меняем общий подход на работу с абстрактным классом. Для выбора варианта реализации поместим на форму выпадающий список.

Реализация

Первым делом введем конструкторы. Тут все просто (листинги 2.2, 2.3).

```
namespace Cars
{
```

```
    /// <summary>
    /// Класс-сущность "Автомобиль"
    /// </summary>
    internal class EntityCar
    {
        /// <summary>
        /// Скорость
        /// </summary>
        public int Speed { get; private set; }
        /// <summary>
        /// Вес
        /// </summary>
        public float Weight { get; private set; }
        /// <summary>
        /// Цвет кузова
        /// </summary>
        public Color BodyColor { get; private set; }
        /// <summary>
        /// Шаг перемещения автомобиля
        /// </summary>
        public float Step => Speed * 100 / Weight;
        /// <summary>
        /// Инициализация полей объекта-класса автомобиля
        /// </summary>
        /// <param name="speed"></param>
        /// <param name="weight"></param>
        /// <param name="bodyColor"></param>
        /// <returns></returns>
        public EntityCar(int speed, float weight, Color bodyColor)
        {
            Random rnd = new();
            Speed = speed <= 0 ? rnd.Next(50, 150) : speed;
            Weight = weight <= 0 ? rnd.Next(40, 70) : weight;
            BodyColor = bodyColor;
        }
    }
}
```

Листинг 2.2 – Класс базовой сущности с конструктором

```
        /// <summary>
        /// Инициализация свойств
        /// </summary>
        /// <param name="speed">Скорость</param>
        /// <param name="weight">Вес автомобиля</param>
        /// <param name="bodyColor">Цвет кузова</param>
        public DrawningCar(int speed, float weight, Color bodyColor)
        {
            Car = new EntityCar(speed, weight, bodyColor);
        }
```

Листинг 2.3 – Конструктор класса прорисовки объекта

Далее перейдем к «продвинутому» объекту. Создадим новый класс-сущность, унаследуем его от имеющегося и пропишем новые поля (листинг 2.4).

```
namespace Cars
{
    /// <summary>
    /// Класс-сущность "Спортивный автомобиль"
```

```
    /// </summary>
    internal class EntitySportCar : EntityCar
    {
        /// <summary>
        /// Дополнительный цвет
        /// </summary>
        public Color DopColor { get; private set; }
        /// <summary>
        /// Признак наличия обвеса
        /// </summary>
        public bool BodyKit { get; private set; }
        /// <summary>
        /// Признак наличия антикрыла
        /// </summary>
        public bool Wing { get; private set; }
        /// <summary>
        /// Признак наличия гоночной полосы
        /// </summary>
        public bool SportLine { get; private set; }
        /// <summary>
        /// Инициализация свойств
        /// </summary>
        /// <param name="speed">Скорость</param>
        /// <param name="weight">Вес автомобиля</param>
        /// <param name="bodyColor">Цвет кузова</param>
        /// <param name="dopColor">Дополнительный цвет</param>
        /// <param name="bodyKit">Признак наличия обвеса</param>
        /// <param name="wing">Признак наличия антикрыла</param>
        /// <param name="sportLine">Признак наличия гоночной полосы</param>
        public EntitySportCar(int speed, float weight, Color bodyColor, Color
dopColor, bool bodyKit, bool wing, bool sportLine) :
            base(speed, weight, bodyColor)
        {
            DopColor = dopColor;
            BodyKit = bodyKit;
            Wing = wing;
            SportLine = sportLine;
        }
    }
}
```

Листинг 2.4 – Класс «продвинутой» сущности

Как видим, для корректной работы требуется через ключевое слово base явно вызывать конструктор базового класса и передавать туда параметры.

Далее в классе прорисовки базового объекта метод DrawTransport делаем виртуальным, а также добавляем новый конструктор, через который можно будет менять размеры объекта (листинг 2.5).

```
        /// <summary>
        /// Инициализация свойств
        /// </summary>
        /// <param name="speed">Скорость</param>
        /// <param name="weight">Вес автомобиля</param>
        /// <param name="bodyColor">Цвет кузова</param>
        /// <param name="carWidth">Ширина отрисовки автомобиля</param>
        /// <param name="carHeight">Высота отрисовки автомобиля</param>
```

```
        protected DrawningCar(int speed, float weight, Color bodyColor, int
carWidth, int carHeight) :
            this(speed, weight, bodyColor)
        {
            _carWidth = carWidth;
            _carHeight = carHeight;
        }
        /// <summary>
        /// Отрисовка автомобиля
        /// </summary>
        /// <param name="g"></param>
        public virtual void DrawTransport(Graphics g)
        {
            if (_startPosX < 0 || _startPosY < 0
                || !_pictureHeight.HasValue || !_pictureWidth.HasValue)
            {
                return;
            }
            Pen pen = new(Color.Black);
            //границы автомобиля
            g.DrawEllipse(pen, _startPosX, _startPosY, 20, 20);
            g.DrawEllipse(pen, _startPosX, _startPosY + 30, 20, 20);
            g.DrawEllipse(pen, _startPosX + 70, _startPosY, 20, 20);
            g.DrawEllipse(pen, _startPosX + 70, _startPosY + 30, 20, 20);
            g.DrawRectangle(pen, _startPosX - 1, _startPosY + 10, 10, 30);
            g.DrawRectangle(pen, _startPosX + 80, _startPosY + 10, 10, 30);
            g.DrawRectangle(pen, _startPosX + 10, _startPosY - 1, 70, 52);

            //задние фары
            Brush brRed = new SolidBrush(Color.Red);
            g.FillEllipse(brRed, _startPosX, _startPosY, 20, 20);
            g.FillEllipse(brRed, _startPosX, _startPosY + 30, 20, 20);

            //передние фары
            Brush brYellow = new SolidBrush(Color.Yellow);
            g.FillEllipse(brYellow, _startPosX + 70, _startPosY, 20, 20);
            g.FillEllipse(brYellow, _startPosX + 70, _startPosY + 30, 20, 20);

            //кузов
            Brush br = new SolidBrush(Car?.BodyColor ?? Color.Black);
            g.FillRectangle(br, _startPosX, _startPosY + 10, 10, 30);
            g.FillRectangle(br, _startPosX + 80, _startPosY + 10, 10, 30);
            g.FillRectangle(br, _startPosX + 10, _startPosY, 70, 51);

            //стекла
            Brush brBlue = new SolidBrush(Color.LightBlue);
            g.FillRectangle(brBlue, _startPosX + 60, _startPosY + 5, 5, 40);
            g.FillRectangle(brBlue, _startPosX + 20, _startPosY + 5, 5, 40);
            g.FillRectangle(brBlue, _startPosX + 25, _startPosY + 3, 35, 2);
            g.FillRectangle(brBlue, _startPosX + 25, _startPosY + 46, 35, 2);

            //выделяем рамкой крышу
            g.DrawRectangle(pen, _startPosX + 25, _startPosY + 5, 35, 40);
            g.DrawRectangle(pen, _startPosX + 65, _startPosY + 10, 25, 30);
            g.DrawRectangle(pen, _startPosX, _startPosY + 10, 15, 30);
        }
```

Листинг 2.5 – Метод DrawTransport класса прорисовки базовой сущности

И создаем класс для прорисовки «продвинутого» объекта (листинг 2.6).

```
namespace Cars
{
```

```
    /// <summary>
    /// Класс, отвечающий за прорисовку и перемещение объекта-сущности
    /// </summary>
    internal class DrawningSportCar : DrawningCar
    {
        /// <summary>
        /// Инициализация свойств
        /// </summary>
        /// <param name="speed">Скорость</param>
        /// <param name="weight">Вес автомобиля</param>
        /// <param name="bodyColor">Цвет кузова</param>
        /// <param name="dopColor">Дополнительный цвет</param>
        /// <param name="bodyKit">Признак наличия обвеса</param>
        /// <param name="wing">Признак наличия антикрыла</param>
        /// <param name="sportLine">Признак наличия гоночной полосы</param>
        public DrawningSportCar(int speed, float weight, Color bodyColor, Color
dopColor, bool bodyKit, bool wing, bool sportLine) :
            base(speed, weight, bodyColor, 110, 60)
        {
            Car = new EntitySportCar(speed, weight, bodyColor, dopColor, bodyKit,
wing, sportLine);
        }
        public override void DrawTransport(Graphics g)
        {
            if (Car is not EntitySportCar sportCar)
            {
                return;
            }

            Pen pen = new(Color.Black);
            Brush dopBrush = new SolidBrush(sportCar.DopColor);

            if (sportCar.BodyKit)
            {
                g.DrawEllipse(pen, _startPosX + 90, _startPosY, 20, 20);
                g.DrawEllipse(pen, _startPosX + 90, _startPosY + 40, 20, 20);
                g.DrawRectangle(pen, _startPosX + 90, _startPosY + 10, 20, 40);
                g.DrawRectangle(pen, _startPosX + 90, _startPosY, 15, 15);
                g.DrawRectangle(pen, _startPosX + 90, _startPosY + 45, 15, 15);

                g.FillEllipse(dopBrush, _startPosX + 90, _startPosY, 20, 20);
                g.FillEllipse(dopBrush, _startPosX + 90, _startPosY + 40, 20, 20);
                g.FillRectangle(dopBrush, _startPosX + 90, _startPosY + 10, 20,
40);
                g.FillRectangle(dopBrush, _startPosX + 90, _startPosY + 1, 15,
15);
                g.FillRectangle(dopBrush, _startPosX + 90, _startPosY + 45, 15,
15);

                g.DrawEllipse(pen, _startPosX, _startPosY, 20, 20);
                g.DrawEllipse(pen, _startPosX, _startPosY + 40, 20, 20);
                g.DrawRectangle(pen, _startPosX, _startPosY + 10, 20, 40);
                g.DrawRectangle(pen, _startPosX + 5, _startPosY, 14, 15);
                g.DrawRectangle(pen, _startPosX + 5, _startPosY + 45, 14, 15);

                g.FillEllipse(dopBrush, _startPosX, _startPosY, 20, 20);
                g.FillEllipse(dopBrush, _startPosX, _startPosY + 40, 20, 20);
                g.FillRectangle(dopBrush, _startPosX + 1, _startPosY + 10, 25,
40);
                g.FillRectangle(dopBrush, _startPosX + 5, _startPosY + 1, 15, 15);
                g.FillRectangle(dopBrush, _startPosX + 5, _startPosY + 45, 15,
15);
```

```
                g.DrawRectangle(pen, _startPosX + 35, _startPosY, 39, 15);
                g.DrawRectangle(pen, _startPosX + 35, _startPosY + 45, 39, 15);

                g.FillRectangle(dopBrush, _startPosX + 35, _startPosY + 1, 40,
15);
                g.FillRectangle(dopBrush, _startPosX + 35, _startPosY + 45, 40,
15);
            }

            _startPosX += 10;
            _startPosY += 5;
            base.DrawTransport(g);
            _startPosX -= 10;
            _startPosY -= 5;

            if (sportCar.SportLine)
            {
                g.FillRectangle(dopBrush, _startPosX + 75, _startPosY + 23, 25,
15);
                g.FillRectangle(dopBrush, _startPosX + 35, _startPosY + 23, 35,
15);
                g.FillRectangle(dopBrush, _startPosX, _startPosY + 23, 20, 15);
            }

            if (sportCar.Wing)
            {
                g.FillRectangle(dopBrush, _startPosX, _startPosY + 5, 10, 50);
                g.DrawRectangle(pen, _startPosX, _startPosY + 5, 10, 50);
            }
        }
    }
}
```

Листинг 2.6 – Класс прорисовки «продвинутой» сущности

Как видим тут кода по минимуму: конструктор и перегруженный метод прорисовки. В конструкторе мы присваиваем полю Car новое значение – объект класса-«продвинутой» сущности. Так как конструктор дочернего класса будет вызываться после вызова конструктора базового класса, то в поле Car сохранится «продвинутый» объект. А в методе прорисовки делаем хитрость. Правильный подход перегрузки методов подразумевает использование исходного метода. В нашем исходном методе выполнялась прорисовка базового объекта. Воспользуемся ею, дополнив элементами, которые есть в «продвинутом». Однако, из-за новых элементов необходимо «сместить» прорисовку базовой части чуть ниже и левее. Так как в методе в качестве координат используются поля класса, а не передаваемые в метод параметры, то, чтобы не писать отдельный метод прорисовки с передачей

координат через параметры, мы изменим значения координат перед вызовом метода и вернем их в прежнее значение после вызова.

Перейдем к реализации интерфейса IDrawningObject. Первым делом надо перенести код интерфейса в проект. Для этого в проекте создаем новый элемент «Интерфейс». **Важно**: по негласному правилу в названии интерфейса первой буквой всегда идет буква I. Пример: ITransport, IAnimal. Добавить интерфейс в проект можно так же, как и класс: на вкладке «Обозреватель решения» вызовем меню через правую кнопку на названии проекта, найдем пункт «Добавить» и в нем «Создать элемент…». В появившемся окне выбрать «Интерфейс» и ввести имя (рисунок 2.1).

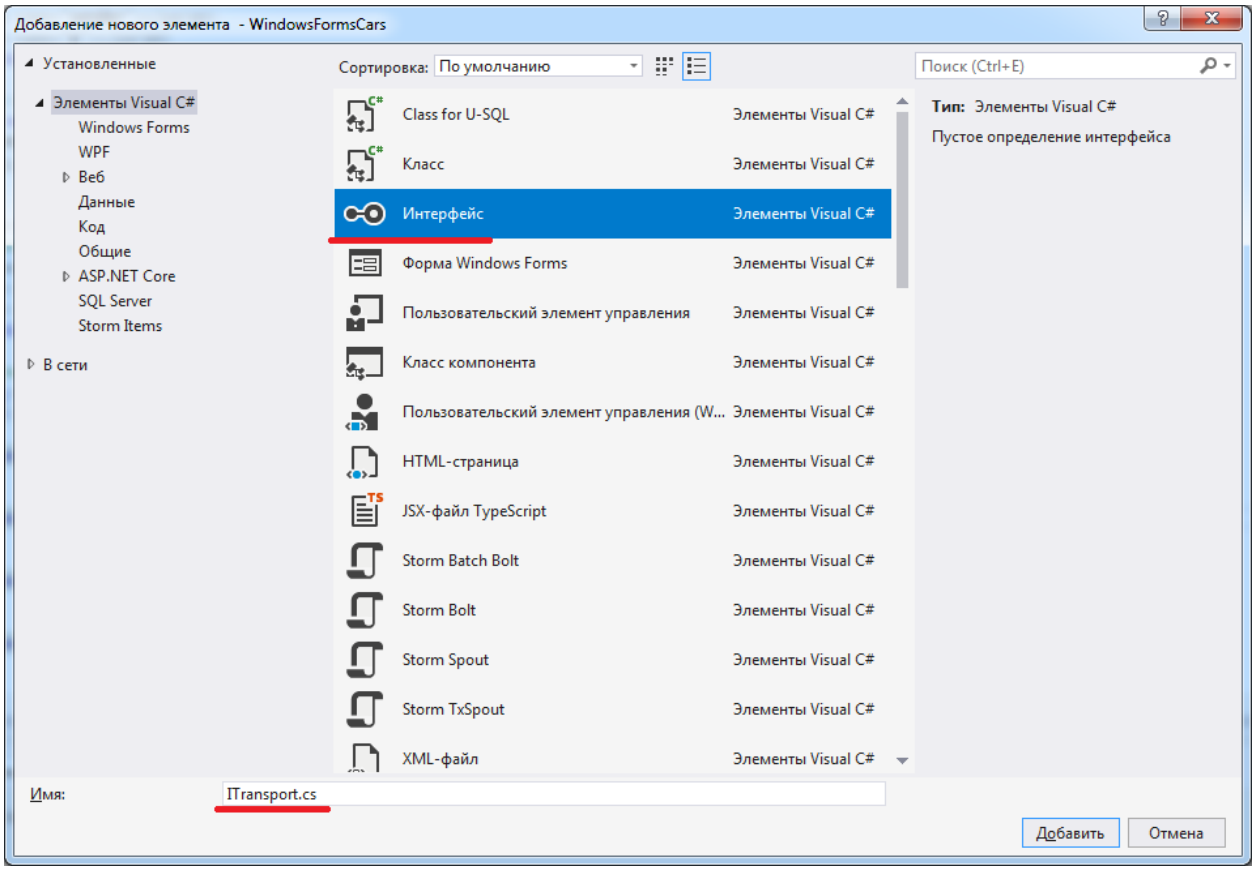


Рисунок 2.1 – Создание интерфейса

Второй вариант, добавить простой класс и в коде заменить «class» на «interface». Код интерфейса представлен был в листинге 2.1.

Создадим класс-наследник (листинг 2.7).

```
namespace Cars
{
```

```
    internal class DrawningObjectCar : IDrawningObject
    {
        private DrawningCar _car = null;

        public DrawningObjectCar(DrawningCar car)
        {
            _car = car;
        }

        public float Step => _car?.Car?.Step ?? 0;

        public (float Left, float Right, float Top, float Bottom)
GetCurrentPosition()
        {
            return _car?.GetCurrentPosition() ?? default;
        }

        public void MoveObject(Direction direction)
        {
            _car?.MoveTransport(direction);
        }

        public void SetObject(int x, int y, int width, int height)
        {
            _car.SetPosition(x, y, width, height);
        }

        public void DrawningObject(Graphics g)
        {
            // TODO
        }
    }
}
```

Листинг 2.7 – Класс DrawningObjectCar

В методе получения текущей позиции объекта должны возвращаться координаты прямоугольника. Чтобы не создавать отдельный класс с 4 полями, тут используется кортеж на 4 параметра, которые обозначают параметры прямоугольника (левую верхнюю точку и нижнюю правую). Для его корректной работы в классе прорисовки объекта добавляем метод для получения координат (листинг 2.8).

```
        /// <summary>
        /// Получение текущей позиции объекта
        /// </summary>
        /// <returns></returns>
        public (float Left, float Right, float Top, float Bottom)
GetCurrentPosition()
        {
            return (_startPosX, _startPosY, _startPosX + _carWidth, _startPosY +
_carHeight);
        }
```

Листинг 2.7 – Метод получения текущей позиции в классе прорисовки объекта

Останется только решить, как вызывать метод прорисовки с учетом, что он виртуальный и может потребуется вызывать переопределенный в классе-наследнике.

Далее перейдем к абстрактному классу. Создается простой класс, у которого добавляется модификатор abstract. Сам класс имеется ряд методов для создания карты, помещение объекта на карту, перемещение объекта по карте, прорисовку карты с объектом (листинг 2.9).

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Cars
{
    internal abstract class AbstractMap
    {
        private IDrawningObject _drawningObject = null;
        protected int[,] _map = null;
        protected int _width;
        protected int _height;
        protected float _size_x;
        protected float _size_y;
        protected readonly Random _random = new();
        protected readonly int _freeRoad = 0;
        protected readonly int _barrier = 1;

        public Bitmap CreateMap(int width, int height, IDrawningObject
drawningObject)
        {
            _width = width;
            _height = height;
            _drawningObject = drawningObject;
            GenerateMap();
            while (!SetObjectOnMap())
            {
                GenerateMap();
            }
            return DrawMapWithObject();
        }
        public Bitmap MoveObject(Direction direction)
        {
            // TODO проверка, что объект может переместится в требуемом
направлении
            if (true)
            {
                _drawningObject.MoveObject(direction);
            }
            return DrawMapWithObject();
        }
        private bool SetObjectOnMap()
        {
            if (_drawningObject == null || _map == null)
            {
                return false;
```

```
            }
            int x = _random.Next(0, 10);
            int y = _random.Next(0, 10);
            _drawningObject.SetObject(x, y, _width, _height);
            // TODO првоерка, что объект не "накладывается" на закрытые участки
            return true;
        }
        private Bitmap DrawMapWithObject()
        {
            Bitmap bmp = new(_width, _height);
            if (_drawningObject == null || _map == null)
            {
                return bmp;
            }
            Graphics gr = Graphics.FromImage(bmp);
            for (int i = 0; i < _map.GetLength(0); ++i)
            {
                for (int j = 0; j < _map.GetLength(1); ++j)
                {
                    if (_map[i, j] == _freeRoad)
                    {
                        DrawRoadPart(gr, i, j);
                    }
                    else if (_map[i, j] == _barrier)
                    {
                        DrawBarrierPart(gr, i, j);
                    }
                }
            }
            _drawningObject.DrawningObject(gr);
            return bmp;
        }

        protected abstract void GenerateMap();
        protected abstract void DrawRoadPart(Graphics g, int i, int j);
        protected abstract void DrawBarrierPart(Graphics g, int i, int j);
    }
}
```

Листинг 2.9 – Абстрактный класс «Карта»

Потребуется дописать ряд проверок:

1. При помещении объекта на карту следует проверять, что он не попадет на закрытый участок и изменять координаты, так, чтобы он в итоге попал на свободное пространство.
2. При перемещении объекта следует проверять, что при смене координаты он не попадет на закрытый участок.

Реализацию класса пока сделаем самую простую: закрашиваем прямоугольниками сплошным цветом. Один цвет для участков, по которым можно перемещаться, другой цвет для закрытых участков. Генерация карты

будет заключаться в инициализации массива, далее случайным образом ставим ряд ячеек заблокированными (листинг 2.10).

```
namespace Cars
{
    /// <summary>
    /// Простая реализация абсрактного класса AbstractMap
    /// </summary>
    internal class SimpleMap : AbstractMap
    {
        /// <summary>
        /// Цвет участка закрытого
        /// </summary>
        private readonly Brush barrierColor = new SolidBrush(Color.Black);
        /// <summary>
        /// Цвет участка открытого
        /// </summary>
        private readonly Brush roadColor = new SolidBrush(Color.Gray);

        protected override void DrawBarrierPart(Graphics g, int i, int j)
        {
            g.FillRectangle(barrierColor, i * _size_x, j * _size_y, i * (_size_x +
1), j * (_size_y + 1));
        }
        protected override void DrawRoadPart(Graphics g, int i, int j)
        {
            g.FillRectangle(roadColor, i * _size_x, j * _size_y, i * (_size_x +
1), j * (_size_y + 1));
        }
        protected override void GenerateMap()
        {
            _map = new int[100, 100];
            _size_x = (float)_width / _map.GetLength(0);
            _size_y = (float)_height / _map.GetLength(1);
            int counter = 0;
            for (int i = 0; i < _map.GetLength(0); ++i)
            {
                for (int j = 0; j < _map.GetLength(1); ++j)
                {
                    _map[i, j] = _freeRoad;
                }
            }
            while (counter < 50)
            {
                int x = _random.Next(0, 100);
                int y = _random.Next(0, 100);
                if (_map[x, y] == _freeRoad)
                {
                    _map[x, y] = _barrier;
                    counter++;
                }
            }
        }
    }
}
```

Листинг 2.10 – Реализация абстрактного класса «Карта»

Последнее, что осталось рассмотреть – новая форма по работе с картой. Там появились новая кнопка: для создания «продвинутого» объекта и элемент «Выпадающий список» для выбора вида карты (рисунок 2.2).

Сперва следует заполнить элемент ComboBox (выпадающий список). На форме, в свойствах этого элемента есть пункт «Items», при его выборе открывается формочка для ввода строковых значений, каждая строка – пункт в выпадающем списке. На данный момент там будет один пункт (так как у нас одна реализация карты имеется). Чтобы пользователь мог только выбирать из списка, не вводя свои значения в свойствах найдем «DropDownList» и выставим в нем значение «DropDownList». Теперь к логике. В логике меняем работу с объекта класса по прорисовке на объект от абстрактного класса. Метод прорисовки удаляем, так как он будет теперь в абстрактном классе. Вместо него вводится метод заполнения информации по объекту (так как у нас теперь 2 варианта создания). Также вводится метод смены карты при смене выбранного значения в выпадающем списке. А вот метод отслеживания изменения размеров уходит, так как в абстрактном классе не предусмотрена такая возможность (листинг 2.1).

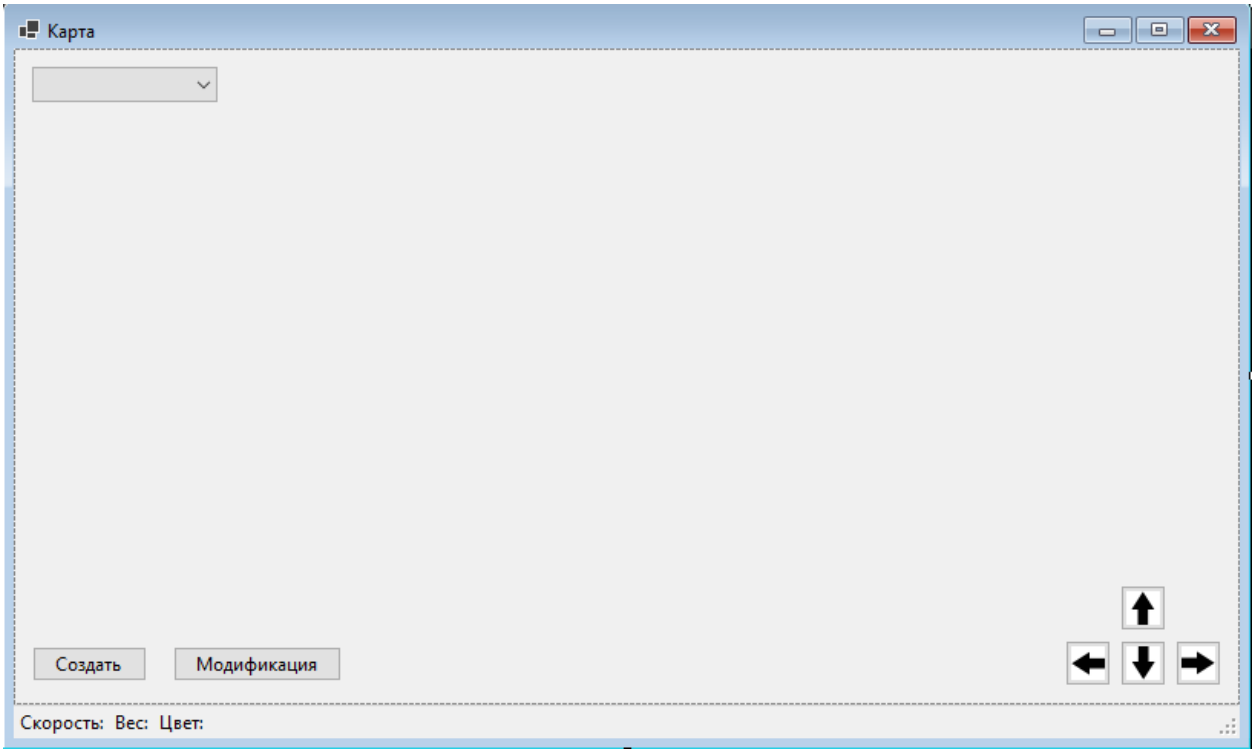


Рисунок 2.2 – Форма для работы с картой

```
namespace Cars
{
    public partial class FormMap : Form
    {
        private AbstractMap _abstractMap;

        public FormMap()
        {
            InitializeComponent();
            _abstractMap = new SimpleMap();
        }
        /// <summary>
        /// Заполнение информации по объекту
        /// </summary>
        /// <param name="car"></param>
        private void SetData(DrawningCar car)
        {
            toolStripStatusLabelSpeed.Text = $"Скорость: {car.Car.Speed}";
            toolStripStatusLabelWeight.Text = $"Вес: {car.Car.Weight}";
            toolStripStatusLabelBodyColor.Text = $"Цвет:
{car.Car.BodyColor.Name}";
            pictureBoxCar.Image = _abstractMap.CreateMap(pictureBoxCar.Width,
pictureBoxCar.Height,
                new DrawningObjectCar(car));
        }
        /// <summary>
        /// Обработка нажатия кнопки "Создать"
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void ButtonCreate_Click(object sender, EventArgs e)
        {
            Random rnd = new();
            var car = new DrawningCar(rnd.Next(100, 300), rnd.Next(1000, 2000),
Color.FromArgb(rnd.Next(0, 256), rnd.Next(0, 256), rnd.Next(0, 256)));
            SetData(car);
        }
        /// <summary>
        /// Изменение размеров формы
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void ButtonMove_Click(object sender, EventArgs e)
        {
            //получаем имя кнопки
            string name = ((Button)sender)?.Name ?? string.Empty;
            Direction dir = Direction.None;
            switch (name)
            {
                case "buttonUp":
                    dir = Direction.Up;
                    break;
                case "buttonDown":
                    dir = Direction.Down;
                    break;
                case "buttonLeft":
                    dir = Direction.Left;
                    break;
                case "buttonRight":
                    dir = Direction.Right;
                    break;
```

```
            }
            pictureBoxCar.Image = _abstractMap?.MoveObject(dir);
        }
        /// <summary>
        /// Обработка нажатия кнопки "Модификация"
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void ButtonCreateModif_Click(object sender, EventArgs e)
        {
            Random rnd = new();
            var car = new DrawningSportCar(rnd.Next(100, 300), rnd.Next(1000,
2000),
                Color.FromArgb(rnd.Next(0, 256), rnd.Next(0, 256), rnd.Next(0,
256)),
                Color.FromArgb(rnd.Next(0, 256), rnd.Next(0, 256), rnd.Next(0,
256)),
                Convert.ToBoolean(rnd.Next(0, 2)), Convert.ToBoolean(rnd.Next(0,
2)), Convert.ToBoolean(rnd.Next(0, 2)));
            SetData(car);
        }
        /// <summary>
        /// Смена карты
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void ComboBoxSelectorMap_SelectedIndexChanged(object sender,
EventArgs e)
        {
            switch (comboBoxSelectorMap.Text)
            {
                case "Простая карта":
                    _abstractMap = new SimpleMap();
                    break;
            }
        }
    }
}
```

Листинг 2.11 – Логика формы работы с картой

Требования

1. Платформа для проектов – .Net версии 6.0
2. Название проекта должно соответствовать логике задания.
3. Название форм, классов, свойств классов должно соответствовать логике задания.
4. Все элементы форм (заголовки форм, текст в label и т.д.) должны иметь подписи на одном языке (или все русским, или все английским).

5. Дочерний класс должен реализовывать объект, указанный в задании.
6. В дочернем классе должно быть **минимум 2 свойства-признака**, которые по логике задания отличают его от базового класса и свойство дополнительного цвета. Свойства должны влиять на прорисовку объекта дочернего класса.
7. Сделать 2 реализации абстрактного класса с различной прорисовкой карты и генерацией полей.
8. Доделать логику в методах MoveObject и SetObjectOnMap абстрактного класса.
9. Доделать логику в методе DrawningObject класса DrawningObjectCar.

Порядок сдачи базовой части

1. Выбрать карту.
2. Создать базовый объект. Показать, как он двигается по карте с учетом препятствий.
3. Создать «продвинутый» объект. Показать, как он двигается по карте с учетом препятствий.
4. Показать остальные варианты карт.

Контрольные вопросы к базовой части

1. Показать определения методов интерфейса. / Добавить метод в интерфейс.
2. Показать перегрузку методов. / Изменить перемещение для дочернего класса.
3. Показать вызовы абстрактных методов. / Сделать простой класс-наследник от интерфейса.

Варианты

Вар.	Базовый объект	«Продвинутый» объект
1.	Военный самолет	Бомбардировщик (с бомбами и дополнительными топливными баками)
2.	Грузовик	Самосвал (с кузовом и тентом)
3.	Гусеничная машина	Бульдозер (с отвалом спереди и рыхлителем сзади)
4.	Локомотив	Электровоз (с «рогами» для подключения к проводам и отсеком под электрические батареи)
5.	Корабль	Теплоход (с трубами и отсеком под топливо)
6.	Бронированная машина	Танк (с башней с орудием и зенитным пулеметом на башне)
7.	Самолет	Аэробус (с дополнительным «отсеком» для пассажиров спереди сверху и с дополнительными двигателями)
8.	Военный корабль	Линкор (с орудийной башней и отсеком под ракеты)
9.	Автобус	Автобус двухэтажный (со вторым этажом и лестницей, ведущей на него)
10.	Лодка	Катер (с мотором в корме и защитным стеклом спереди)
11.	Военный самолет	Истребитель (с ракетами и дополнительными крыльями для лучшей маневренности)
12.	Грузовик	Бензовоз (с баком под бензин и сигнальным маяком на кабине)
13.	Гусеничная машина	Экскаватор (с ковшом и опорами для фиксации)

Вар.	Базовый объект	«Продвинутый» объект
14.	Локомотив	Тепловоз (с трубой и отсеком под топливо)
15.	Корабль	Контейнеровоз (с контейнерами и краном для разгрузки)
16.	Бронированная машина	Самоходная арт. установка (с башней с орудием и залповой батарей сзади)
17.	Самолет	Самолет с радаром (с радаром и дополнительными топливными баками)
18.	Военный корабль	Крейсер (с ракетными шахтами и площадкой под вертолет)
19.	Автобус	Автобус с гармошкой (с дополнительным отсеком, соединенным гармошкой и отдельным входом для него)
20.	Лодка	Катамаран (с поплавками слева и справа от основного корпуса и парусом)
21.	Военный самолет	Штурмовик (с ракетами и бомбами)
22.	Грузовик	Подметально-уборочная машина (с баком под воду и подметательной щеткой)
23.	Гусеничная машина	Подъемный кран (с краном и противовесом к нему)
24.	Локомотив	Монорельс (с магнитной рельсой и второй кабиной в задней части)
25.	Корабль	Лайнер (с дополнительной палубой и бассейном)
26.	Бронированная машина	Зенитная установка (с башней с зенитными орудиями и радаром)
27.	Самолет	Гидросамолет (с поплавками и надувной лодкой)

Вар.	Базовый объект	«Продвинутый» объект
28.	Военный корабль	Авианосец (с палубой для взлета самолетов и рубкой управления)
29.	Автобус	Троллейбус (с «рогами» для подключения к проводам и отсеком под электрические батареи)
30.	Лодка	Парусник (с парусом и усиленным корпусом)

ЛАБОРАТОРНАЯ РАБОТА №3.
ПОЛИМОРФИЗМ. ПЕРЕГРУЗКА МЕТОДОВ И ОПЕРАЦИЙ. ПАРАМЕТРИЧЕСКИЕ КЛАССЫ

Цель

Познакомится с понятиями полиморфизма. Изучить его виды. Научиться работать с параметризованными классами и параметрическим полиморфизмом. Познакомиться с понятием специализированного полиморфизма. Изучить перегрузку методов и операций.

Задание

1. *В тестовое приложение по отслеживанию перемещения объекта по координатам необходимо внести следующие изменения:*
    a. *Создать generic-класс, в котором будет хранится карта и набор объектов под эту карту. Добавление и удаление объектов реализовать через перегрузку операторов сложения и вычитания. Добавить методы для прорисовки всего набора объектов, а также карты с первым объектом из набора.*
    б. *Создать форму, для работы с generic-классом. В форме в отдельной панели сделать возможность выбора реализации абстрактного класса, добавления объекта к нему (сделать через вызов формы FormCar), удаления объекта по позиции, просмотр всего набора, карты и перемещение по ней объекта (использовать логику формы FormMap, а саму форму удалить).*

Проектирование

Для работы с набором объектов под карту создадим отдельный generic-класс от одного параметра. Параметр задействуем в массиве (массив будет от этого типа-параметра). В конструкторе класса будем передавать количество элементов в массиве. Для фиксации пустого места в массиве ограничим тип-параметр как ссылочный. Таким образом присвоив элементу массива значение Null, обозначим его как пустое. Пропишем ряд методов для вставки в массив объектов, извлечения и удаления. При вставке будет сдвигать массив вправо от места вставки до первого null-элемента, если такой есть, иначе не вставляем. Удаление будет простое – элементу присваивается значение null.

Теперь создадим еще один generic-класс, в этот раз уже под 2 параметра. Первый будет использоваться для создания объекта от нашего первого generic-класса. Этот параметр дополнительно ограничим от интерфейса IDrawningObject, чтобы можно было вызывать методы установки объекта и его прорисовки. Второй параметр ограничим от абстрактного класса AbstractMap. Можно было вместо параметра сделать и поле от этого типа, в данном случае, разница была бы минимальной. В конструктор будем передавать размеры поля и объект от класса-наследника AbstractMap. Через размеры поля и размеры, необходимые под прорисовку одного объекта, определяем максимально допустимое количество объектов (значение передаем в конструктор первого параметризованного класса). Далее, делаем что требовалось по заданию, перегружаем операторы сложения и вычитания. В оба оператора одним из параметров будет передавать объект от generic-класса с 2 параметрами. Для оператора сложения вторым параметром будет объект, добавляемый в набор. Для оператора вычитания вторым параметром будет номер позиции, с которой удаляется объект. Результатом обеих операций будет bool-значение, обозначающее успех или неудачу выполнения операции. Последним шагом

будет добавление метода для прорисовки всего набора объектов, и метода для прорисовки карты с первым объектом из набора.

Остается создать форму. На форму добавляем элемент-панель, на которую помещаем выпадающий список для выбора карты (берем с формы FormMap), кнопки для добавления объекта, удаления объекта (к ней еще потребуется текстовое поле для ввода номера позиции), просмотра набора объектов и карты с объектом. Также с формы FormMap перенесем кнопки управления. Панель размести в правой части формы, а на оставшуюся часть – элемент PictureBox для вывода картинок.

Реализация

Первый класс назовем SetCarsGeneric (набор автомобилей параметризованный). Сделаем 2 метода для вставки, один будет вставлять в начало набора, второй по позиции и метода для удаления (по позиции). Также отдельный метод получения объекта из набора по позиции, но без удаления. И последнее, что потребуется – свойство для получения количества элементов в массиве (листинг 3.1).

```
namespace Cars
{
    /// <summary>
    /// Параметризованный набор объектов
    /// </summary>
    /// <typeparam name="T"></typeparam>
    internal class SetCarsGeneric<T>
            where T : class
      {
            /// <summary>
            /// Массив объектов, которые храним
            /// </summary>
            private readonly T[] _places;
            /// <summary>
            /// Количество объектов в массиве
            /// </summary>
            public int Count => _places.Length;
            /// <summary>
            /// Конструктор
            /// </summary>
            /// <param name="count"></param>
            public SetCarsGeneric(int count)
            {
                  _places = new T[count];
            }
            /// <summary>
```

```
        /// Добавление объекта в набор
        /// </summary>
        /// <param name="car">Добавляемый автомобиль</param>
        /// <returns></returns>
        public bool Insert(T car)
        {
            // TODO вставка в начало набора
            return true;
        }
        /// <summary>
        /// Добавление объекта в набор на конкретную позицию
        /// </summary>
        /// <param name="car">Добавляемый автомобиль</param>
        /// <param name="position">Позиция</param>
        /// <returns></returns>
        public bool Insert(T car, int position)
        {
            // TODO проверка позиции
            // TODO проверка, что элемент массива по этой позиции пустой,
если нет, то
            //           проверка, что после вставляемого элемента в
массиве есть пустой элемент
            //           сдвиг всех объектов, находящихся справа от
позиции до первого пустого элемента
            // TODO вставка по позиции
            _places[position] = car;
            return true;
        }
        /// <summary>
        /// Удаление объекта из набора с конкретной позиции
        /// </summary>
        /// <param name="position"></param>
        /// <returns></returns>
        public bool Remove(int position)
        {
            // TODO проверка позиции
            // TODO удаление объекта из массива, присовив элементу массива
значение null
            return true;
        }
        /// <summary>
        /// Получение объекта из набора по позиции
        /// </summary>
        /// <param name="position"></param>
        /// <returns></returns>
        public T Get(int position)
        {
            // TODO проверка позиции
            return _places[position];
        }
    }
}
```

Листинг 3.1 – Параметризованный класс с набором объектов

Второму классу дадим имя MapWithSetCarsGeneric. В нем, как уже отмечалось выше, перегрузим два оператора, и пропишем 2 метода. Дополнительно потребуется метод для получения объекта-карты для манипуляции с ним на форме. Также для метода прорисовки сделаем 3

внутренних метода, один будет отвечать за перенос всех объектов в начало массива, второй – за прорисовку фона, третий – за прорисовку объектов (листинг 3.2).

```
namespace Cars
{
    /// <summary>
    /// Карта с набром объектов под нее
    /// </summary>
    /// <typeparam name="T"></typeparam>
    /// <typeparam name="U"></typeparam>
    internal class MapWithSetCarsGeneric<T, U>
        where T : class, IDrawningObject
        where U : AbstractMap
    {
        /// <summary>
        /// Ширина окна отрисовки
        /// </summary>
        private readonly int _pictureWidth;
        /// <summary>
        /// Высота окна отрисовки
        /// </summary>
        private readonly int _pictureHeight;
        /// <summary>
        /// Размер занимаемого объектом места (ширина)
        /// </summary>
        private readonly int _placeSizeWidth = 210;
        /// <summary>
        /// Размер занимаемого объектом места (высота)
        /// </summary>
        private readonly int _placeSizeHeight = 90;
        /// <summary>
        /// Набор объектов
        /// </summary>
        private readonly SetCarsGeneric<T> _setCars;
        /// <summary>
        /// Карта
        /// </summary>
        private readonly U _map;
        /// <summary>
        /// Конструктор
        /// </summary>
        /// <param name="picWidth"></param>
        /// <param name="picHeight"></param>
        /// <param name="map"></param>
        public MapWithSetCarsGeneric(int picWidth, int picHeight, U map)
        {
            int width = picWidth / _placeSizeWidth;
            int height = picHeight / _placeSizeHeight;
            _setCars = new SetCarsGeneric<T>(width * height);
            _pictureWidth = picWidth;
            _pictureHeight = picHeight;
            _map = map;
        }
        /// <summary>
        /// Перегрузка оператора сложения
        /// </summary>
        /// <param name="map"></param>
        /// <param name="car"></param>
        /// <returns></returns>
```

```
        public static bool operator +(MapWithSetCarsGeneric<T, U> map, T car)
        {
            return map._setCars.Insert(car);
        }
        /// <summary>
        /// Перегрузка оператора вычитания
        /// </summary>
        /// <param name="map"></param>
        /// <param name="position"></param>
        /// <returns></returns>
        public static bool operator -(MapWithSetCarsGeneric<T, U> map, int
position)
        {
            return map._setCars.Remove(position);
        }
        /// <summary>
        /// Вывод всего набора объектов
        /// </summary>
        /// <returns></returns>
        public Bitmap ShowSet()
        {
            Bitmap bmp = new(_pictureWidth, _pictureHeight);
            Graphics gr = Graphics.FromImage(bmp);
            DrawBackground(gr);
            DrawCars(gr);
            return bmp;
        }
        /// <summary>
        /// Просмотр объекта на карте
        /// </summary>
        /// <returns></returns>
        public Bitmap ShowOnMap()
        {
            Shaking();
            for (int i = 0; i < _setCars.Count; i++)
            {
                var car = _setCars.Get(i);
                if (car != null)
                {
                    return _map.CreateMap(_pictureWidth, _pictureHeight, car);
                }
            }
            return new(_pictureWidth, _pictureHeight);
        }
        /// <summary>
        /// Перемещение объекта по крате
        /// </summary>
        /// <param name="direction"></param>
        /// <returns></returns>
        public Bitmap MoveObject(Direction direction)
        {
            if (_map != null)
            {
                return _map.MoveObject(direction);
            }
            return new(_pictureWidth, _pictureHeight);
        }
        /// <summary>
        /// "Взбалтываем" набор, чтобы все элементы оказались в начале
        /// </summary>
        private void Shaking()
        {
            int j = _setCars.Count - 1;
```

```
            for (int i = 0; i < _setCars.Count; i++)
            {
                if (_setCars.Get(i) == null)
                {
                    for (; j > i; j--)
                    {
                        var car = _setCars.Get(j);
                        if (car != null)
                        {
                            _setCars.Insert(car, i);
                            _setCars.Remove(j);
                            break;
                        }
                    }
                    if (j <= i)
                    {
                        return;
                    }
                }
            }
        }
        /// <summary>
        /// Метод отрисовки фона
        /// </summary>
        /// <param name="g"></param>
        private void DrawBackground(Graphics g)
        {
            Pen pen = new(Color.Black, 3);
            for (int i = 0; i < _pictureWidth / _placeSizeWidth; i++)
            {
                for (int j = 0; j < _pictureHeight / _placeSizeHeight + 1; ++j)
                {//линия рамзетки места
                    g.DrawLine(pen, i * _placeSizeWidth, j * _placeSizeHeight, i *
_placeSizeWidth + _placeSizeWidth / 2, j * _placeSizeHeight);
                }
                g.DrawLine(pen, i * _placeSizeWidth, 0, i * _placeSizeWidth,
(_pictureHeight / _placeSizeHeight) * _placeSizeHeight);
            }
        }
        /// <summary>
        /// Метод прорисовки объектов
        /// </summary>
        /// <param name="g"></param>
        private void DrawCars(Graphics g)
        {
            for (int i = 0; i < _setCars.Count; i++)
            {
                // TODO установка позиции
                _setCars.Get(i)?.DrawningObject(g);
            }
        }
    }
}
```

Листинг 3.2 – Параметризованный класс карты с набором объектов

Теперь создадим новую форму (рисунок 3.1).

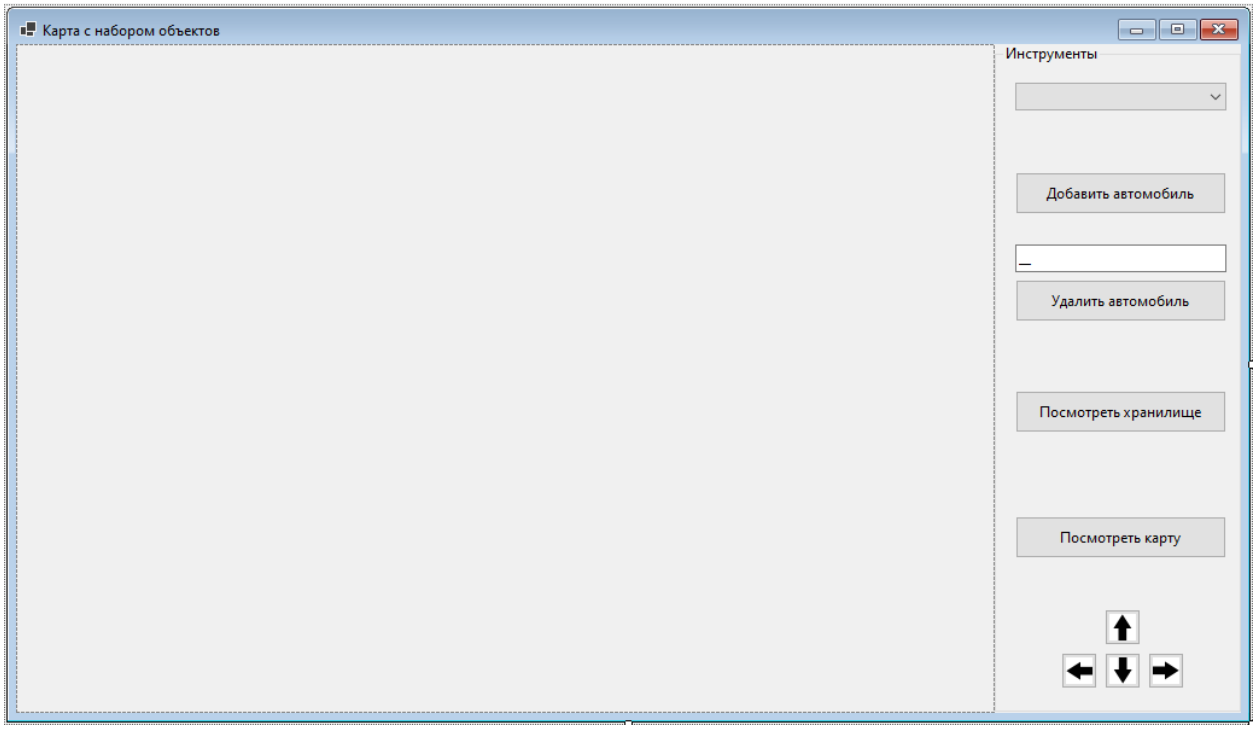


Рисунок 3.1 – Форма для работы с картой и набором объектов к ней

Для добавления объекта в набор будем вызывать форму FormCar. Для этого сделаем в ее логике ряд манипуляций: добавим свойство для получения объекта, а на самой форме – кнопку, при нажатии на которую будем понимать, что объект выбран. Также для выбора цвета объекта воспользуемся специальным диалоговым окном (листинг 3.3). Так как форма имеет публичный модификатор доступа, то у классов DrawningCar, EntityCar и перечисления Direction также придется сменить модификатор доступа на public.

```
namespace Cars
{
    public partial class FormCar : Form
    {
        private DrawningCar _car;

        public DrawningCar SelectedCar { get; private set; }

        public FormCar()
        {
            InitializeComponent();
        }
        /// <summary>
             /// Метод прорисовки машины
             /// </summary>
             private void Draw()
        {
            Bitmap bmp = new(pictureBoxCar.Width, pictureBoxCar.Height);
```

```
            Graphics gr = Graphics.FromImage(bmp);
            _car?.DrawTransport(gr);
            pictureBoxCar.Image = bmp;
        }
        /// <summary>
             /// Метод установки данных
             /// </summary>
             private void SetData()
        {
            Random rnd = new();
            _car.SetPosition(rnd.Next(10, 100), rnd.Next(10, 100),
pictureBoxCar.Width, pictureBoxCar.Height);
            toolStripStatusLabelSpeed.Text = $"Скорость: {_car.Car.Speed}";
            toolStripStatusLabelWeight.Text = $"Вес: {_car.Car.Weight}";
            toolStripStatusLabelBodyColor.Text = $"Цвет:
{_car.Car.BodyColor.Name}";
        }
        /// <summary>
        /// Обработка нажатия кнопки "Создать"
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void ButtonCreate_Click(object sender, EventArgs e)
        {
            Random rnd = new();
            Color color = Color.FromArgb(rnd.Next(0, 256), rnd.Next(0, 256),
rnd.Next(0, 256));
            ColorDialog dialog = new();
            if (dialog.ShowDialog() == DialogResult.OK)
            {
                color = dialog.Color;
            }
            _car = new DrawningCar(rnd.Next(100, 300), rnd.Next(1000, 2000),
color);
            SetData();
            Draw();
        }
        /// <summary>
        /// Изменение размеров формы
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void ButtonMove_Click(object sender, EventArgs e)
        {
            //получаем имя кнопки
            string name = ((Button)sender)?.Name ?? string.Empty;
            switch (name)
            {
                case "buttonUp":
                    _car?.MoveTransport(Direction.Up);
                    break;
                case "buttonDown":
                    _car?.MoveTransport(Direction.Down);
                    break;
                case "buttonLeft":
                    _car?.MoveTransport(Direction.Left);
                    break;
                case "buttonRight":
                    _car?.MoveTransport(Direction.Right);
                    break;
            }
            Draw();
        }
```

```
        /// <summary>
        /// Изменение размеров формы
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void PictureBoxCar_Resize(object sender, EventArgs e)
        {
            _car?.ChangeBorders(pictureBoxCar.Width, pictureBoxCar.Height);
            Draw();
        }
        /// <summary>
        /// Обработка нажатия кнопки "Модификация"
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void ButtonCreateModif_Click(object sender, EventArgs e)
        {
            Random rnd = new();
            Color color = Color.FromArgb(rnd.Next(0, 256), rnd.Next(0, 256),
rnd.Next(0, 256));
            ColorDialog dialog = new();
            if (dialog.ShowDialog() == DialogResult.OK)
            {
                color = dialog.Color;
            }
            Color dopColor = Color.FromArgb(rnd.Next(0, 256), rnd.Next(0, 256),
rnd.Next(0, 256));
            ColorDialog dialogDop = new();
            if (dialogDop.ShowDialog() == DialogResult.OK)
            {
                dopColor = dialogDop.Color;
            }
            _car = new DrawningSportCar(rnd.Next(100, 300), rnd.Next(1000, 2000),
color, dopColor,
                Convert.ToBoolean(rnd.Next(0, 2)), Convert.ToBoolean(rnd.Next(0,
2)), Convert.ToBoolean(rnd.Next(0, 2)));
            SetData();
            Draw();
        }
        /// <summary>
        /// Обработка нажатия кнопки "Выбрать"
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void ButtonSelectCar_Click(object sender, EventArgs e)
        {
            SelectedCar = _car;
            DialogResult = DialogResult.OK;
        }
    }
}
```

Листинг 3.3 – Новая логика формы FormCar

Теперь перейдем к логике формы FormMapWithSetCars. Там создадим объект от второго параметризованного класса. В качестве параметров укажем типы DrawningObjectCar и AbstractMap (листинг 3.4).

```
namespace Cars
{
    public partial class FormMapWithSetCars : Form
```

```
    {
        /// <summary>
        /// Объект от класса карты с набором объектов
        /// </summary>
        private MapWithSetCarsGeneric<DrawningObjectCar, AbstractMap>
_mapCarsCollectionGeneric;
        /// <summary>
        /// Конструктор
        /// </summary>
        public FormMapWithSetCars()
        {
            InitializeComponent();
        }
        /// <summary>
        /// Выбор карты
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void ComboBoxSelectorMap_SelectedIndexChanged(object sender,
EventArgs e)
        {
            AbstractMap map = null;
            switch (comboBoxSelectorMap.Text)
            {
                case "Простая карта":
                    map = new SimpleMap();
                    break;
            }
            if (map != null)
            {
                _mapCarsCollectionGeneric = new
MapWithSetCarsGeneric<DrawningObjectCar, AbstractMap>(
                    pictureBox.Width, pictureBox.Height, map);
            }
            else
            {
                _mapCarsCollectionGeneric = null;
            }
        }
        /// <summary>
        /// Добавление объекта
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void ButtonAddCar_Click(object sender, EventArgs e)
        {
            if(_mapCarsCollectionGeneric == null)
            {
                return;
            }
            FormCar form = new();
            if (form.ShowDialog() == DialogResult.OK)
            {
                DrawningObjectCar car = new(form.SelectedCar);
                if (_mapCarsCollectionGeneric + car)
                {
                    MessageBox.Show("Объект добавлен");
                    pictureBox.Image = _mapCarsCollectionGeneric.ShowSet();
                }
                else
                {
                    MessageBox.Show("Не удалось добавить объект");
                }
```

```
            }
        }
        /// <summary>
        /// Удаление объекта
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void ButtonRemoveCar_Click(object sender, EventArgs e)
        {
            if (string.IsNullOrEmpty(maskedTextBoxPosition.Text))
            {
                return;
            }
            if (MessageBox.Show("Удалить объект?", "Удаление",
MessageBoxButtons.YesNo, MessageBoxIcon.Question) == DialogResult.No)
            {
                return;
            }
            int pos = Convert.ToInt32(maskedTextBoxPosition.Text);
            if (_mapCarsCollectionGeneric - pos)
            {
                MessageBox.Show("Объект удален");
                pictureBox.Image = _mapCarsCollectionGeneric.ShowSet();
            }
            else
            {
                MessageBox.Show("Не удалось удалить объект");
            }
        }
        /// <summary>
        /// Вывод набора
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void ButtonShowStorage_Click(object sender, EventArgs e)
        {
            if (_mapCarsCollectionGeneric == null)
            {
                return;
            }
            pictureBox.Image = _mapCarsCollectionGeneric.ShowSet();
        }
        /// <summary>
        /// Вывод карты
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void ButtonShowOnMap_Click(object sender, EventArgs e)
        {
            if (_mapCarsCollectionGeneric == null)
            {
                return;
            }
            pictureBox.Image = _mapCarsCollectionGeneric.ShowOnMap();
        }
        /// <summary>
        /// Перемещение
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void ButtonMove_Click(object sender, EventArgs e)
        {
            if (_mapCarsCollectionGeneric == null)
```

```
            {
                return;
            }
            //получаем имя кнопки
            string name = ((Button)sender)?.Name ?? string.Empty;
            Direction dir = Direction.None;
            switch (name)
            {
                case "buttonUp":
                    dir = Direction.Up;
                    break;
                case "buttonDown":
                    dir = Direction.Down;
                    break;
                case "buttonLeft":
                    dir = Direction.Left;
                    break;
                case "buttonRight":
                    dir = Direction.Right;
                    break;
            }
            pictureBox.Image = _mapCarsCollectionGeneric.MoveObject(dir);
        }
    }
}
```

Листинг 3.4 – Логика формы FormMapWithSetCars

Остается удалить форму FormMap, а в классе Program прописать вызов новой формы.

Требования

1. Платформа для проектов – .Net версии 6.0
2. Название проекта должно соответствовать логике задания.
3. Название форм, классов, свойств классов должно соответствовать логике задания.
4. Все элементы форм (заголовки форм, текст в label и т.д.) должны иметь подписи на одном языке (или все русским, или все английским).
5. Дописать логику в методах вставки, удаления и получения элемента класса SetCarsGeneric.
6. Поменять тип возвращаемого значения для операций сложения и вычитания (тип возвращаемого значения должен логически согласовываться с результатом операции для типов передаваемых

операндов, потребуется поменять тип возвращаемого значения в методах первого параметризованного класса).

7. Метод прорисовки фона DrawBackground должен учитывать тип карты, указанный во вариантах. Направление вывода объектов (метод DrawCars) также указан во вариантах.

Порядок сдачи базовой части

1. Выбрать карту.
2. Создать несколько объектов.
3. Вывести один объект на карту и поуправлять им на ней.
4. Удалить один объект (не первый и не последний).
5. Добавить новый объект.

Контрольные вопросы к базовой части

1. Где перегрузка методов, операторов (описание и вызов). / Перегрузить оператор сравнения для двух объектов от первого параметризованного класса (по количеству мест, не используя свойство Count).
2. Показать параметризованный класс, доказать, что он параметризованный, показать объект от параметризованного класса. / Добавить второй параметр в первый параметризованный класс (должен быть простым типом).
3. Где используется полиморфизм подтипов, показать на любом примере в коде. / Создать параметризованный класс, ограничив параметр только объектами прорисовки усложненного класса, либо его наследников.

Варианты

Вар.	**Тип карты**	**Направление**	**Вар.**	**Тип карты**	**Направление**

1.	Ангар	Влево, вниз	2.	Гараж	Вправо, вниз
3.	Стоянка	Влево, вверх	4.	Депо	Вправо, вверх
5.	Пристань	Влево, вниз	6.	База	Вправо, вниз
7.	Аэродром	Влево, вверх	8.	Доки	Вправо, вверх
9.	Автовокзал	Влево, вниз	10.	Гавань	Вправо, вниз
11.	Ангар	Влево, вверх	12.	Гараж	Вправо, вверх
13.	Стоянка	Влево, вниз	14.	Депо	Вправо, вниз
15.	Пристань	Влево, вверх	16.	База	Вправо, вверх
17.	Аэродром	Влево, вниз	18.	Доки	Вправо, вниз
19.	Автовокзал	Влево, вверх	20.	Гавань	Вправо, вверх
21.	Ангар	Влево, вниз	22.	Гараж	Вправо, вниз
23.	Стоянка	Влево, вверх	24.	Депо	Вправо, вверх
25.	Пристань	Влево, вниз	26.	База	Вправо, вниз
27.	Аэродром	Влево, вверх	28.	Доки	Вправо, вверх
29.	Автовокзал	Влево, вниз	30.	Гавань	Вправо, вниз

ЛАБОРАТОРНАЯ РАБОТА №4.
КОЛЛЕКЦИИ. ИНДЕКСАТОРЫ

Цель

Познакомится с коллекциями (в частности, словарь и список). Познакомиться с индексаторами.

Задание

1. *В тестовое приложение по отслеживанию перемещения объекта по координатам необходимо внести следующие изменения:*
    a. *В качестве хранилища объектов в универсальном классе заменить массив на список.*
    б. *Добавить класс для хранения набора карт с объектами к нему (класс MapWithSetCarsGeneric), при этом у каждого объекта MapWithSetCarsGeneric в наборе должно быть уникальное имя.*
    в. *Модифицировать форму, для работы с generic-классом. Реализовать возможность указания имени для карты, просмотра карт, а также удаления карты.*

Проектирование

Все работы разобьем на 3 этапа, по этапу на каждый пункт задачи.

На первом этапе заменим массив в классе SetCarsGeneric на список. Также заменим метод Get на индексатор для обращения к элементу списка через объект класса (это уже больше баловство, так как большой разницы между индексатором и методом нет). В MapWithSetCarsGeneric потребуется поправить возникшие ошибки после внесения изменений в класс SetCarsGeneric.

На втором этапе создадим новый класс для работы с набором карт. В классе для хранения будем использовать словарь. В качестве типа ключа будет выступать тип string, так как он позволит хранить уникальные названия для карт, в качестве типа значений будет выступать класс MapWithSetCarsGeneric. В классе потребуется прописать несколько методов для взаимодействия со словарем: добавление в словарь запись, удаление записи из словаря, обращение к элементу словаря по ключу (также через индексатор сделаем).

На третьем этапе поработаем с формой. Сперва на самой форме добавим элементы для работы с создаваемым набором карт. Добавим текстовое поле для ввода названия, кнопки для добавления карты и удаления карты, а также элемент для вывода списка (воспользуемся простым – ListBox). В логике заменим объект от класса MapWithSetCarsGeneric на объект от нового класса и переработаем взаимодействие с объектом в методах класса.

Реализация

Замена массива на список довольно простая операция. Дополнительно введем поле для хранения максимально возможного количества элементов, которые можем добавить в список (а то сам список не имеет ограничения, в отличии от массива) и сделаем возможность прохода по списку через оператор yield до первого не пустого, чтобы выдавать объект и сразу его обрабатывать (листинг 4.1).

```
namespace Cars
{
    /// <summary>
    /// Параметризованный набор объектов
    /// </summary>
    /// <typeparam name="T"></typeparam>
    internal class SetCarsGeneric<T>
            where T : class
      {
            /// <summary>
            /// Список объектов, которые храним
            /// </summary>
            private readonly List<T> _places;
            /// <summary>
            /// Количество объектов в списке
            /// </summary>
            public int Count => _places.Count;
```

```
        private readonly int _maxCount;
        /// <summary>
        /// Конструктор
        /// </summary>
        /// <param name="count"></param>
        public SetCarsGeneric(int count)
        {
            _maxCount = count;
            _places = new List<T>();
        }
        /// <summary>
        /// Добавление объекта в набор
        /// </summary>
        /// <param name="car">Добавляемый автомобиль</param>
        /// <returns></returns>
        public bool Insert(T car)
        {
            // TODO вставка в начало набора
            // TODO проверка на _maxCount
            return true;
        }
        /// <summary>
        /// Добавление объекта в набор на конкретную позицию
        /// </summary>
        /// <param name="car">Добавляемый автомобиль</param>
        /// <param name="position">Позиция</param>
        /// <returns></returns>
        public bool Insert(T car, int position)
        {
            // TODO проверка позиции
            // TODO вставка по позиции
            _places[position] = car;
            return true;
        }
        /// <summary>
        /// Удаление объекта из набора с конкретной позиции
        /// </summary>
        /// <param name="position"></param>
        /// <returns></returns>
        public bool Remove(int position)
        {
            // TODO проверка позиции
            return true;
        }
        /// <summary>
        /// Получение объекта из набора по позиции
        /// </summary>
        /// <param name="position"></param>
        /// <returns></returns>
        public T this[int position]
        {
            get
            {
                // TODO проверка позиции
                return _places[position];
            }
            set
            {
                // TODO проверка позиции
                // TODO вставка в список по позиции
            }
        }
```

```
            /// <summary>
            /// Проход по набору до первого пустого
            /// </summary>
            /// <returns></returns>
            public IEnumerable<T> GetCars()
            {
                  foreach (var car in _places)
                  {
                        if (car != null)
                        {
                              yield return car;
                        }
                        else
                        {
                              yield break;
                        }
                  }
            }
      }
}
```

Листинг 4.1 – Измененный код класса SetCarsGeneric

Также, потребуется внести правки в класс MapWithSetCarsGeneric. Там будут только точечные изменения, связанные с удалением метода Get и вводом метода прохода по списку через перечисление, поэтому приведем только фрагмент кода (листинг 4.2).

```
        /// <summary>
        /// Просмотр объекта на карте
        /// </summary>
        /// <returns></returns>
        public Bitmap ShowOnMap()
        {
            Shaking();
            foreach (var car in _setCars.GetCars())
            {
                return _map.CreateMap(_pictureWidth, _pictureHeight, car);
            }
            return new(_pictureWidth, _pictureHeight);
        }
        /// <summary>
        /// Перемещение объекта по крате
        /// </summary>
        /// <param name="direction"></param>
        /// <returns></returns>
        public Bitmap MoveObject(Direction direction)
        {
            if (_map != null)
            {
                return _map.MoveObject(direction);
            }
            return new(_pictureWidth, _pictureHeight);
        }
        /// <summary>
        /// "Взбалтываем" набор, чтобы все элементы оказались в начале
        /// </summary>
        private void Shaking()
        {
```

```
            int j = _setCars.Count - 1;
            for (int i = 0; i < _setCars.Count; i++)
            {
                if (_setCars[i] == null)
                {
                    for (; j > i; j--)
                    {
                        var car = _setCars[j];
                        if (car != null)
                        {
                            _setCars.Insert(car, i);
                            _setCars.Remove(j);
                            break;
                        }
                    }
                    if (j <= i)
                    {
                        return;
                    }
                }
            }
        }
        /// <summary>
        /// Метод отрисовки фона
        /// </summary>
        /// <param name="g"></param>
        private void DrawBackground(Graphics g)
        {
            Pen pen = new(Color.Black, 3);
            for (int i = 0; i < _pictureWidth / _placeSizeWidth; i++)
            {
                for (int j = 0; j < _pictureHeight / _placeSizeHeight + 1; ++j)
                {//линия рамзетки места
                    g.DrawLine(pen, i * _placeSizeWidth, j * _placeSizeHeight, i *
_placeSizeWidth + _placeSizeWidth / 2, j * _placeSizeHeight);
                }
                g.DrawLine(pen, i * _placeSizeWidth, 0, i * _placeSizeWidth,
(_pictureHeight / _placeSizeHeight) * _placeSizeHeight);
            }
        }
        /// <summary>
        /// Метод прорисовки объектов
        /// </summary>
        /// <param name="g"></param>
        private void DrawCars(Graphics g)
        {
            foreach (var car in _setCars.GetCars())
            {
                // TODO установка позиции
                car.DrawningObject(g);
            }
        }
```

Листинг 4.2 – Фрагмент класса MapWithSetCarsGeneric

Далее создадим новый класс. Он будет отвечать за коллекцию карт, так что назовем его MapsCollection. Как и обговаривалось, внутри будет словарь и ряд методов для работы с ним (листинг 4.3).

```
namespace Cars
{
```

```
    /// <summary>
    /// Класс для хранения коллекции карт
    /// </summary>
    internal class MapsCollection
    {
        /// <summary>
        /// Словарь (хранилище) с картами
        /// </summary>
        readonly Dictionary<string, MapWithSetCarsGeneric<DrawningObjectCar,
AbstractMap>> _mapStorages;
        /// <summary>
        /// Возвращение списка названий карт
        /// </summary>
        public List<string> Keys => _mapStorages.Keys.ToList();
        /// <summary>
        /// Ширина окна отрисовки
        /// </summary>
        private readonly int _pictureWidth;
        /// <summary>
        /// Высота окна отрисовки
        /// </summary>
        private readonly int _pictureHeight;
        /// <summary>
        /// Конструктор
        /// </summary>
        /// <param name="pictureWidth"></param>
        /// <param name="pictureHeight"></param>
        public MapsCollection(int pictureWidth, int pictureHeight)
        {
            _mapStorages = new Dictionary<string,
MapWithSetCarsGeneric<DrawningObjectCar, AbstractMap>>();
            _pictureWidth = pictureWidth;
            _pictureHeight = pictureHeight;
        }
        /// <summary>
        /// Добавление карты
        /// </summary>
        /// <param name="name">Название карты</param>
        /// <param name="map">Карта</param>
        public void AddMap(string name, AbstractMap map)
        {
            // TODO Прописать логику для добавления
        }
        /// <summary>
        /// Удаление карты
        /// </summary>
        /// <param name="name">Название карты</param>
        public void DelMap(string name)
        {
            // TODO Прописать логику для удаления
        }
        /// <summary>
        /// Доступ к парковке
        /// </summary>
        /// <param name="ind"></param>
        /// <returns></returns>
        public MapWithSetCarsGeneric<DrawningObjectCar, AbstractMap> this[string
ind]
        {
            get
            {
                // TODO Продумать логику получения объекта
                return null;
```

```
            }
        }
    }
}
```

Листинг 4.3 – Класс MapsCollection

Перейдем к форме. В первую очередь, добавим элементов на форму, предварительно увеличив ее размеры, чтобы вместить новые элементы (рисунок 4.1).

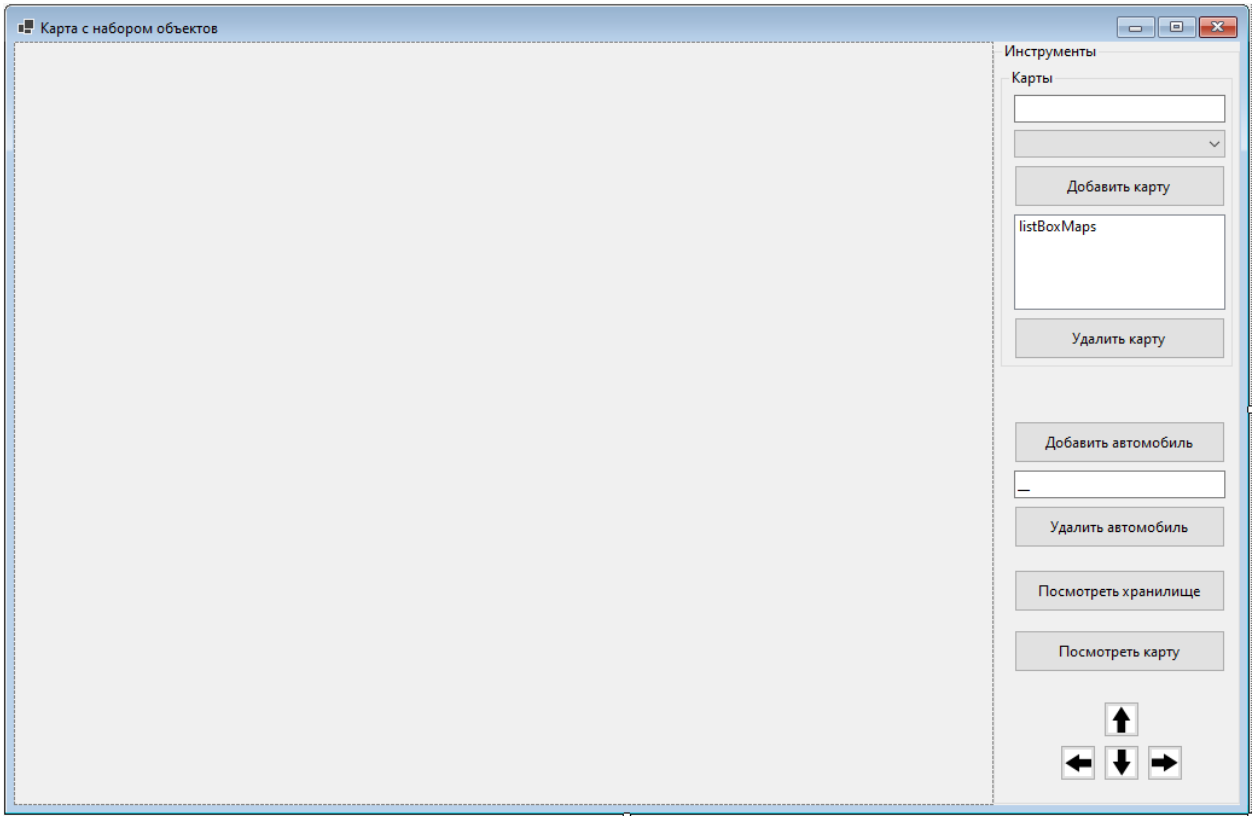


Рисунок 4.1 – Обновленный дизайн формы

Далее пойдем в логику. Для выпадающего списка (comboBox) сделаем в логике словарь, который бы сопоставлял название и дочерний класс от AbstractMap, который бы ему соответствовал. Сам выпадающий список заполним ключами из этого словаря. Все остальное меняем, как было описано на этапе проектирования (листинг 4.4). Дополнительно сделаем метод для обновления списка в ListBox, так как обновление требуется в 2-х случаях. Прочие изменения затронули, в основном, работу с замененным объектом и поменялось ряд проверок.

```
namespace Cars
{
```

```
    public partial class FormMapWithSetCars : Form
    {
        /// <summary>
        /// Словарь для выпадающего списка
        /// </summary>
        private readonly Dictionary<string, AbstractMap> _mapsDict = new()
        {
            { "Простая карта", new SimpleMap() }
        };
        /// <summary>
        /// Объект от коллекции карт
        /// </summary>
        private readonly MapsCollection _mapsCollection;
        /// <summary>
        /// Конструктор
        /// </summary>
        public FormMapWithSetCars()
        {
            InitializeComponent();
            _mapsCollection = new MapsCollection(pictureBox.Width,
pictureBox.Height);
            comboBoxSelectorMap.Items.Clear();
            foreach(var elem in _mapsDict)
            {
                comboBoxSelectorMap.Items.Add(elem.Key);
            }
        }
        /// <summary>
        /// Заполнение listBoxMaps
        /// </summary>
        private void ReloadMaps()
        {
            int index = listBoxMaps.SelectedIndex;

            listBoxMaps.Items.Clear();
            for (int i = 0; i < _mapsCollection.Keys.Count; i++)
            {
                listBoxMaps.Items.Add(_mapsCollection.Keys[i]);
            }

            if (listBoxMaps.Items.Count > 0 && (index == -1 || index >=
listBoxMaps.Items.Count))
            {
                listBoxMaps.SelectedIndex = 0;
            }
            else if (listBoxMaps.Items.Count > 0 && index > -1 && index <
listBoxMaps.Items.Count)
            {
                listBoxMaps.SelectedIndex = index;
            }
        }
        /// <summary>
        /// Добавление карты
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void ButtonAddMap_Click(object sender, EventArgs e)
        {
            if (comboBoxSelectorMap.SelectedIndex == -1 ||
string.IsNullOrEmpty(textBoxNewMapName.Text))
            {
                MessageBox.Show("Не все данные заполнены", "Ошибка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
```

```
                return;
            }
            if (!_mapsDict.ContainsKey(comboBoxSelectorMap.Text))
            {
                MessageBox.Show("Нет такой карты", "Ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
                return;
            }
            _mapsCollection.AddMap(textBoxNewMapName.Text,
_mapsDict[comboBoxSelectorMap.Text]);
            ReloadMaps();
        }
        /// <summary>
        /// Выбор карты
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void ListBoxMaps_SelectedIndexChanged(object sender, EventArgs e)
        {
            pictureBox.Image =
_mapsCollection[listBoxMaps.SelectedItem?.ToString() ?? string.Empty].ShowSet();
        }
        /// <summary>
        /// Удаление карты
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void ButtonDeleteMap_Click(object sender, EventArgs e)
        {
            if (listBoxMaps.SelectedIndex == -1)
            {
                return;
            }

            if (MessageBox.Show($"Удалить карту {listBoxMaps.SelectedItem}?",
"Удаление", MessageBoxButtons.YesNo, MessageBoxIcon.Question) == DialogResult.Yes)
            {
                _mapsCollection.DelMap(listBoxMaps.SelectedItem?.ToString() ??
string.Empty);
                ReloadMaps();
            }
        }
        /// <summary>
        /// Добавление объекта
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void ButtonAddCar_Click(object sender, EventArgs e)
        {
            if (listBoxMaps.SelectedIndex == -1)
            {
                return;
            }
            FormCar form = new();
            if (form.ShowDialog() == DialogResult.OK)
            {
                DrawningObjectCar car = new(form.SelectedCar);
                if (_mapsCollection[listBoxMaps.SelectedItem?.ToString() ??
string.Empty] + car)
                {
                    MessageBox.Show("Объект добавлен");
                    pictureBox.Image =
_mapsCollection[listBoxMaps.SelectedItem?.ToString() ?? string.Empty].ShowSet();
```

```
                }
                else
                {
                    MessageBox.Show("Не удалось добавить объект");
                }
            }
        }
        /// <summary>
        /// Удаление объекта
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void ButtonRemoveCar_Click(object sender, EventArgs e)
        {
            if (listBoxMaps.SelectedIndex == -1)
            {
                return;
            }
            if (string.IsNullOrEmpty(maskedTextBoxPosition.Text))
            {
                return;
            }
            if (MessageBox.Show("Удалить объект?", "Удаление",
MessageBoxButtons.YesNo, MessageBoxIcon.Question) == DialogResult.No)
            {
                return;
            }
            int pos = Convert.ToInt32(maskedTextBoxPosition.Text);
            if (_mapsCollection[listBoxMaps.SelectedItem?.ToString() ??
string.Empty] - pos)
            {
                MessageBox.Show("Объект удален");
                pictureBox.Image =
_mapsCollection[listBoxMaps.SelectedItem?.ToString() ?? string.Empty].ShowSet();
            }
            else
            {
                MessageBox.Show("Не удалось удалить объект");
            }
        }
        /// <summary>
        /// Вывод набора
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void ButtonShowStorage_Click(object sender, EventArgs e)
        {
            if (listBoxMaps.SelectedIndex == -1)
            {
                return;
            }
            pictureBox.Image =
_mapsCollection[listBoxMaps.SelectedItem?.ToString() ?? string.Empty].ShowSet();
        }
        /// <summary>
        /// Вывод карты
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void ButtonShowOnMap_Click(object sender, EventArgs e)
        {
            if (listBoxMaps.SelectedIndex == -1)
            {
```

```
                return;
            }
            pictureBox.Image =
_mapsCollection[listBoxMaps.SelectedItem?.ToString() ?? string.Empty].ShowOnMap();
        }
        /// <summary>
        /// Перемещение
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void ButtonMove_Click(object sender, EventArgs e)
        {
            if (listBoxMaps.SelectedIndex == -1)
            {
                return;
            }
            //получаем имя кнопки
            string name = ((Button)sender)?.Name ?? string.Empty;
            Direction dir = Direction.None;
            switch (name)
            {
                case "buttonUp":
                    dir = Direction.Up;
                    break;
                case "buttonDown":
                    dir = Direction.Down;
                    break;
                case "buttonLeft":
                    dir = Direction.Left;
                    break;
                case "buttonRight":
                    dir = Direction.Right;
                    break;
            }
            pictureBox.Image =
_mapsCollection[listBoxMaps.SelectedItem?.ToString() ??
string.Empty].MoveObject(dir);
        }
    }
}
```

Листинг 4.1 – Измененный код формы FormMapWithSetCars

Требования

1. Платформа для проектов – .Net версии 6.0
2. Название проекта должно соответствовать логике задания.
3. Название форм, классов, свойств классов должно соответствовать логике задания.
4. Все элементы форм (заголовки форм, текст в label и т.д.) должны иметь подписи на одном языке (или все русским, или все английским).

5. Изменить логику в методах вставки, удаления и получения элемента класса SetCarsGeneric.
6. Дописать логику в методах вставки, удаления и получения элемента класса MapsCollection.
7. На форме, при переходе с карты на карту, сразу должна идти ее прорисовка.

Порядок сдачи базовой части

1. Создать 2 карты.
2. Создать несколько объектов на первой карте.
3. Переключиться на вторую карту.
4. Создать несколько объектов на второй карте
5. Переключиться на первую карту.
6. Удалить первую карту.

Контрольные вопросы к базовой части

1. Перечислить методы и свойства словаря, которые используются в решении. / Заменить словарь mapsDict на список.
2. Перечислить методы и свойства ListBox, которые используются в решении. / Изменить цикл заполнения ListBox на foreach.
3. Показать реализацию индексатора и место, где происходит его вызов. / Сделать индексатор с числом в качестве параметра.

Варианты

Вар.	**Проект**	**Вар.**	**Проект**
1.	Бомбардировщик	2.	Самосвал
3.	Бульдозер	4.	Электровоз
5.	Теплоход	6.	Танк

7.	Аэробус	8.	Линкор
9.	Автобус двухэтажный	10.	Катер
11.	Истребитель	12.	Бензовоз
13.	Экскаватор	14.	Тепловоз
15.	Контейнеровоз	16.	САУ
17.	Самолет с радаром	18.	Крейсер
19.	Автобус с гармошкой	20.	Катамаран
21.	Штурмовик	22.	Подметально-уборочная машина
23.	Подъемный кран	24.	Монорельс
25.	Лайнер	26.	Зенитная установка
27.	Гидросамолет	28.	Авианосец
29.	Троллейбус	30.	Парусник

ЛАБОРАТОРНАЯ РАБОТА №5.
ДЕЛЕГАТЫ, СОБЫТИЯ. DRAG&DROP

Цель

Познакомится с событиями и делегатами. Ознакомиться с работой механизма drag&drop.

Задание

1. *В тестовое приложение по отслеживанию перемещения объекта по координатам необходимо внести следующие изменения:*
    a. *Создать новую форму для создания объекта.*
    б. *При выборе типа объекта (обычного или продвинутого), а также цвета использовать механизм Drag&Drop. Остальные поля заполнять через обычные элементы формы (TextBox, NumericUpDown, CheckBox и т.п.).*
    в. *Предусмотреть возможность вызова сразу нескольких форм создания объектов.*

Проектирование

Потребуется сделать 2 действия:

1. Создать новую форму.
2. Настроить вызов новой формы из формы работы с коллекцией.

В новой форме потребуется следующий набор функций: ввод свойств объекта (как для простого объекта класса, так и для продвинутого), выбор типа объекта (простой или продвинутый) и выбор цветов (только основной или еще дополнительный для объекта продвинутого класса).

Добавим на форму GroupBox, куда поместим все элементы для задания параметров объекта. В него добавим элементы Label для вывода подписей (например, «Скорость» и «Вес»). Для числовых параметров будем

использовать элемент NumericUpDown, которые позволяют вводить числа в определенном диапазоне (ограничим от 100 до 1000). Также добавим в GroupBox несколько элементов CheckBox для установки меток по свойствам-признакам для продвинутого класса.

Для отображения объекта поместим на форму PictureBox. Рядом поместим GroupBox куда добавим 2 Label с вариантами выбора объекта, которые мы можем сделать (простого или продвинутого). Выбор типа объекта будет происходить путем «перетаскивания» элементов. Для этого будем применять технологию Drag&Drop.

Для работы с цветами, зададим 8 возможных цветов для объекта. Для этого на форме зададим 8 маленьких панелей, каждую из которых раскрасим в определенный цвет, используя у панелей свойство BackColor. Рядом с PictureBox зададим 2 Label'а, один, куда мы будем перетаскивать (технология Drag&Drop) цвет для основного цвета, второй – для дополнительного цвета.

Второй шаг – взаимодействие форм. На данный момент мы вызывали одну форму из другой по средствам вызова у второй формы метода ShowDialog(). Данный вариант плох тем, что основная форма, из который идет вызов второй формы становится не активной, пока открыта вторая форма. Если посмотреть по коду основной формы через отладку, то при вызове метода ShowDialog() второй формы, выполнение кода основной формы прерывается и возобновится когда вторая форма будет закрыта.

Однако есть иной вариант вызова второй формы, через метод Show(). В таком случае основная форма не будет блокироваться, а при отладке мы увидим, что выполнение кода основной формы не прерывается в месте вызова метода Show(), а идет дальше. Однако, возникает проблема. После закрытия второй формы, мы должны получить от нее информацию о выбранном объекте и добавит его в хранилище. Но как это сделать, если мы не знаем, когда закроется эта форма? С вариантом через ShowDialog() проще в таком плане.

Выполнение кода метода продолжится после строки form.ShowDialog() только когда закроется форма.

А для варианта с Show() придется создать делегат, от него во второй форме событие, метод добавления к событию метода, и в основной форме метод-обработчик, который будет навязываться событию. В основной форме в логике вызова второй формы будет создаваться объект от второй формы, далее у этого объекта будет вызываться метод привязки к событию обработчика (это еще один метод основной формы) и далее будет вызываться метод Show() второй формы. Когда во второй форме будет нажиматься кнопка «Добавить», в ее логике будет вызываться событие и метод-обработчик, который к ней привязан (в него будет передаваться сформированный объект), и вторая форма закроется. А метод-обработчик добавит новый объект в хранилище.

Реализация

Создадим форму (назовем ее FormCarConfig), в которой будет настраивать добавляемый объект. Для ввода параметров объекта добавим на форму GroupBox. В него добавим 2 Label для вывода текста «Скорость» и «Вес». Рядом с ними поместим 2 NumericUpDown, которые позволяют вводить числа в определенном диапазоне (ограничим от 100 до 1000). Также добавим в GroupBox несколько элементов checkbox для установки меток по свойствам-признакам для продвинутого класса (наличие обвесов, антикрыла и полоски гоночной).

В GroupBox добавим еще один GroupBox под цвета. В него добавим 8 маленьких панелей, каждую из которых раскрасим в определенный цвет, используя у панелей свойство BackColor.

Под внутренний GroupBox поместим 2 Label (чтобы удобнее с ними было работать, зададим им жестко размеры, выставив в свойстве AutoSize

значение False, и обведем границы через свойство BorderStyle) с вариантами выбора объекта, которые мы можем сделать (простого или продвинутого).

Рядом с внешним GroupBox добавим на новую форму PictureBox для отображения объекта.

Для выполнения операций перетаскивания в приложениях Windows необходимо обрабатывать последовательность событий **DragEnter** и **DragDrop**.

Рассмотрим на примере. Начнем с Label с надписью: «Простой». Все операции перетаскивания начинаются с переноса данных. Функциональные возможности, позволяющие собрать данные при начале перетаскивания, реализуются в методе **DoDragDrop**. Как и где это применять? Все начинается, когда мы нажимаем кнопкой мыши на объекте, который хотим перетащить. У каждого элемента формы есть событие, которое срабатывает при нажатии кнопки мыши на него, называется MouseDown. Создадим у Label это событие (рисунок 5.1).

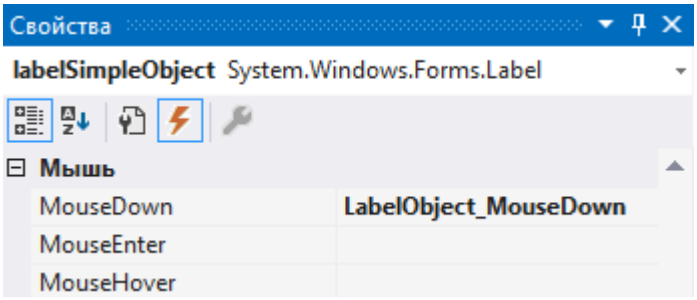


Рисунок 5.1 – Событие нажатия кнопки мышки

В методе-обработки этого события будет вызываться метод **DoDragDrop** для нашего Label, в который будем передавать информацию. Первым параметром, передаваемым в этот метод, является объект, который будет перемещаться. В нашем случае, это будет строка с текстом, размещенным в Label’е. Вторым параметром – возможные результаты операции перетаскивания. В нашем случае это будут операции перемещения и копирования. Результаты можно комбинировать, используя логические операторы.

Теперь на получателе нужно оформить 2 метода, для приема. Но тут возникает проблема. На элементе, который будет принимать перетаскиваемый объект, должно быть свойство AllowDrop. Но его нет у элемента PictureBox. Поэтому сделаем хитрость, поместим PictureBox в Panel. У Panel задаем значение свойства AllowDrop значением true. И задаем методы для 2-х событий: **DragEnter** и **DragDrop**. Событие DragEnter позволяет убедиться в корректности происходящих действий, в нашем случае, это должен быть текст (строка). Событие DragDrop будет основным, в нем будет реализовываться основная логика. Логика заключается в следующем: в зависимости от переданного текста создать либо объект базового класса, либо дочернего. Цвет зададим по умолчанию: основной – белый, дополнительный – черный. Также, создадим метод, который будет прорисовывать созданный объект. Для этого переменную car сделаем полем класса-формы (листинг 5.1).

Для работы с цветами добавим на Panel 2 Label (также зададим им жестко размеры, выставив в свойстве AutoSize значение False, и обведем границы через свойство BorderStyle), один для основного цвета, второй для дополнительного у продвинутого объекта. Под Panel разместим 2 Button, один будет отвечать за добавление готового объекта в хранилище, второй – за отмену добавления (рисунок 5.2).

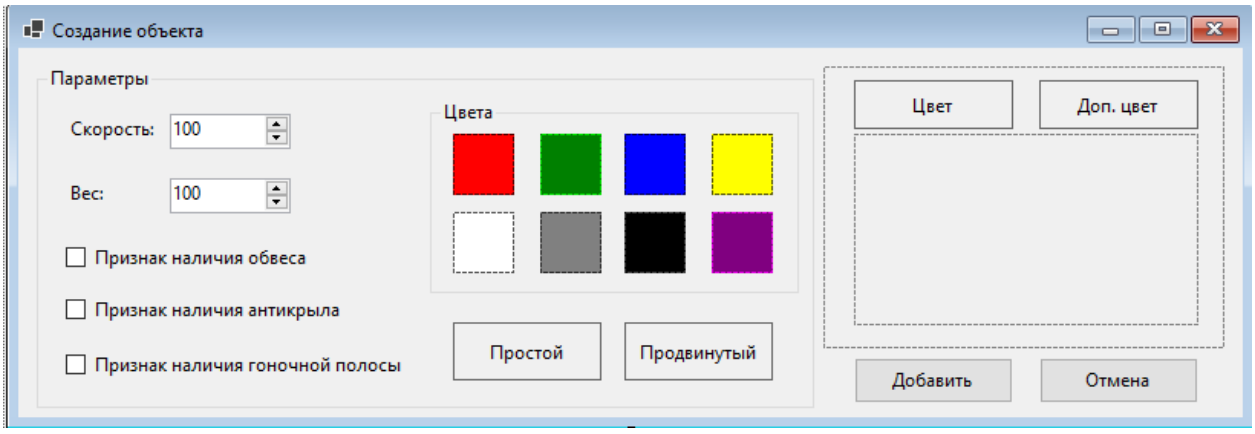


Рисунок 5.2 – Форма создания объекта.

Логика формы по работе с Drag&Drop на примере выбора создаваемого объекта представлена в листинге 5.1.

```
namespace Cars
{
    /// <summary>
    /// Форма создания объекта
    /// </summary>
    public partial class FormCarConfig : Form
    {
        /// <summary>
        /// Переменная-выбранная машина
        /// </summary>
        DrawningCar _car = null;
        /// <summary>
        /// Конструктор
        /// </summary>
        public FormCarConfig()
        {
            InitializeComponent();
        }
        /// <summary>
        /// Отрисовать машину
        /// </summary>
        private void DrawCar()
        {
            Bitmap bmp = new(pictureBoxObject.Width, pictureBoxObject.Height);
            Graphics gr = Graphics.FromImage(bmp);
            _car?.SetPosition(5, 5, pictureBoxObject.Width,
pictureBoxObject.Height);
            _car?.DrawTransport(gr);
            pictureBoxObject.Image = bmp;
        }
        /// <summary>
        /// Передаем информацию при нажатии на Label
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void LabelObject_MouseDown(object sender, MouseEventArgs e)
        {
            (sender as Label).DoDragDrop((sender as Label).Name,
DragDropEffects.Move | DragDropEffects.Copy);
        }
        /// <summary>
        /// Проверка получаемой информации (ее типа на соответствие требуемому)
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void PanelObject_DragEnter(object sender, DragEventArgs e)
        {
            if (e.Data.GetDataPresent(DataFormats.Text))
            {
                e.Effect = DragDropEffects.Copy;
            }
            else
            {
                e.Effect = DragDropEffects.None;
            }
        }
        /// <summary>
        /// Действия при приеме перетаскиваемой информации
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void PanelObject_DragDrop(object sender, DragEventArgs e)
        {
```

```
            switch (e.Data.GetData(DataFormats.Text).ToString())
            {
                case "labelSimpleObject":
                    _car = new DrawningCar((int)numericUpDownSpeed.Value,
(int)numericUpDownWeight.Value, Color.White);
                    break;
                case "labelModifiedObject":
                    _car = new DrawningSportCar((int)numericUpDownSpeed.Value,
(int)numericUpDownWeight.Value, Color.White, Color.Black,
                        checkBoxBodyKit.Checked, checkBoxWing.Checked,
checkBoxSportLine.Checked);
                    break;
            }
            DrawCar();
        }
}
```

Листинг 5.1 – Реализация технологии Drag&Drop на примере создания объекта

В методе LabelObject_MouseDown мы вызываем у контрола (элемент формы, в нашем случае – Label) метод DoDragDrop и передаем туда информацию, в частности – строку с названием контрола. В принципе, передавать можно все что угодно, например, объект типа Color (это вам понадобится для варианта Drag&Drop по цветам). Так как метод сделали без привязки к конкретному элементу формы, то его можно использовать как для Label, отвечающего за простой объект, так и для Label, отвечающего за продвинутый объект. Далее в методе PanelObject_DragEnter делаем проверку, что у нас при перетаскивании передается верная информация в плане ожидаемого типа данных. А в методе PanelObject_DragDrop уже получаем информацию и на ее основе создаем объект.

По аналогичной схеме надо сделать работу с цветами. Только привязку к событию MouseDown для цветовых панелей, для разнообразия, вынесем в код, а не сделаем через дизайнер формы.

Для реализации механизма событий на первом шаге сделаем делегат (листинг 5.2).

```
namespace Cars
{
    /// <summary>
    /// Делегат для передачи объекта-автомобиля
    /// </summary>
    /// <param name="car"></param>
    public delegate void CarDelegate(DrawningCar car);
```

```
}
```

Листинг 5.2 – Делегат CarDelegate

Далее в форме FormCarConfig добавим событие, метод привязки к событию, а также логику для кнопки «Добавить», в которой будем вызывать событие (листинг 5.3).

```
namespace Cars
{
    /// <summary>
    /// Форма создания объекта
    /// </summary>
    public partial class FormCarConfig : Form
    {
        /// <summary>
        /// Событие
        /// </summary>
        private event CarDelegate EventAddCar;
        /// <summary>
        /// Добавление события
        /// </summary>
        /// <param name="ev"></param>
        public void AddEvent(CarDelegate ev)
        {
            if (EventAddCar == null)
            {
                EventAddCar = new CarDelegate(ev);
            }
            else
            {
                EventAddCar += ev;
            }
        }
        /// <summary>
        /// Добавление машины
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void ButtonOk_Click(object sender, EventArgs e)
        {
            EventAddCar?.Invoke(_car);
            Close();
        }
    }
}
```

Листинг 5.3 – Реализация события в форме FormCarConfig

Итоговый код формы FormCarConfig следующий (листинг 5.4).

```
namespace Cars
{
    /// <summary>
    /// Форма создания объекта
    /// </summary>
    public partial class FormCarConfig : Form
    {
        /// <summary>
        /// Переменная-выбранная машина
        /// </summary>
```

```
        DrawningCar _car = null;
        /// <summary>
        /// Событие
        /// </summary>
        private event CarDelegate EventAddCar;
        /// <summary>
        /// Конструктор
        /// </summary>
        public FormCarConfig()
        {
            InitializeComponent();
            panelBlack.MouseDown += PanelColor_MouseDown;
            panelPurple.MouseDown += PanelColor_MouseDown;
            panelGray.MouseDown += PanelColor_MouseDown;
            panelGreen.MouseDown += PanelColor_MouseDown;
            panelRed.MouseDown += PanelColor_MouseDown;
            panelWhite.MouseDown += PanelColor_MouseDown;
            panelYellow.MouseDown += PanelColor_MouseDown;
            panelBlue.MouseDown += PanelColor_MouseDown;

            // TODO buttonCancel.Click with lambda
        }
        /// <summary>
        /// Отрисовать машину
        /// </summary>
        private void DrawCar()
        {
            Bitmap bmp = new(pictureBoxObject.Width, pictureBoxObject.Height);
            Graphics gr = Graphics.FromImage(bmp);
            _car?.SetPosition(5, 5, pictureBoxObject.Width,
pictureBoxObject.Height);
            _car?.DrawTransport(gr);
            pictureBoxObject.Image = bmp;
        }
        /// <summary>
        /// Добавление события
        /// </summary>
        /// <param name="ev"></param>
        public void AddEvent(CarDelegate ev)
        {
            if (EventAddCar == null)
            {
                EventAddCar = new CarDelegate(ev);
            }
            else
            {
                EventAddCar += ev;
            }
        }
        /// <summary>
        /// Передаем информацию при нажатии на Label
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void LabelObject_MouseDown(object sender, MouseEventArgs e)
        {
            (sender as Label).DoDragDrop((sender as Label).Name,
DragDropEffects.Move | DragDropEffects.Copy);
        }
        /// <summary>
        /// Проверка получаемой информации (ее типа на соответствие требуемому)
        /// </summary>
        /// <param name="sender"></param>
```

```
        /// <param name="e"></param>
        private void PanelObject_DragEnter(object sender, DragEventArgs e)
        {
            if (e.Data.GetDataPresent(DataFormats.Text))
            {
                e.Effect = DragDropEffects.Copy;
            }
            else
            {
                e.Effect = DragDropEffects.None;
            }
        }
        /// <summary>
        /// Действия при приеме перетаскиваемой информации
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void PanelObject_DragDrop(object sender, DragEventArgs e)
        {
            switch (e.Data.GetData(DataFormats.Text).ToString())
            {
                case "labelSimpleObject":
                    _car = new DrawningCar((int)numericUpDownSpeed.Value,
(int)numericUpDownWeight.Value, Color.White);
                    break;
                case "labelModifiedObject":
                    _car = new DrawningSportCar((int)numericUpDownSpeed.Value,
(int)numericUpDownWeight.Value, Color.White, Color.Black,
                        checkBoxBodyKit.Checked, checkBoxWing.Checked,
checkBoxSportLine.Checked);
                    break;
            }
            DrawCar();
        }
        /// <summary>
        /// Отправляем цвет с панели
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void PanelColor_MouseDown(object sender, MouseEventArgs e)
        {
            (sender as Control).DoDragDrop((sender as Control).BackColor,
DragDropEffects.Move | DragDropEffects.Copy);
        }
        /// <summary>
        /// Проверка получаемой информации (ее типа на соответствие требуемому)
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void LabelBaseColor_DragEnter(object sender, DragEventArgs e)
        {
            if (e.Data.GetDataPresent(typeof(Color)))
            {
                e.Effect = DragDropEffects.Copy;
            }
            else
            {
                e.Effect = DragDropEffects.None;
            }
        }
        /// <summary>
        /// Принимаем основной цвет
        /// </summary>
```

```
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void LabelBaseColor_DragDrop(object sender, DragEventArgs e)
        {
            // TODO Call method from object _car and set color
        }
        /// <summary>
        /// Принимаем дополнительный цвет
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void LabelDopColor_DragDrop(object sender, DragEventArgs e)
        {
            // TODO Call method from object _car if _car is DrawningSportCar and
set dop color
        }
        /// <summary>
        /// Добавление машины
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void ButtonOk_Click(object sender, EventArgs e)
        {
            EventAddCar?.Invoke(_car);
            Close();
        }
    }
}
```

Листинг 5.4 – Логика формы FormCarConfig

Остается в логике формы FormMapWithSetCars поменять логику в методе добавления объекта (листинг 5.5).

```
        /// <summary>
        /// Добавление объекта
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void ButtonAddCar_Click(object sender, EventArgs e)
        {
            var formCarConfig = new FormCarConfig();
            // TODO Call method AddEvent from formCarConfig
            formCarConfig.Show();
        }
```

Листинг 5.5 – Новая логика метода добавления объекта в форме FormMapWithSetCars

Требования

1. Платформа для проектов – .Net версии 6.0
2. Название проекта должно соответствовать логике задания.
3. Название форм, классов, свойств классов должно соответствовать логике задания.

4. Все элементы форм (заголовки форм, текст в label и т.д.) должны иметь подписи на одном языке (или все русским, или все английским).
5. Использовать встроенные делегаты вместо собственных.
6. Прописать логику обработки события Click у кнопки «Отмена» с использованием лямбда-выражений.
7. Выполнять проверку DragEnter для дополнительного цвета.
8. Прописать логику в методах LabelBaseColor_DragDrop и LabelDopColor_DragDrop.
9. В форме FormMapWithSet… привязываться к событию новой формы.

Порядок сдачи базовой части

1. Вызвать новую форму.
2. Выбрать простой объект.
3. Задать ему цвет. Попытаться задать ему доп. цвет.
4. Добавить в хранилище.
5. Вызвать новую форму.
6. Выбрать продвинутый объект (проставить часть признаков активными).
7. Выбрать продвинутый объект (проставить все признаки активными).
8. Задать ему цвет. Задать ему доп. цвет.
9. Добавить в хранилище.

Контрольные вопросы к базовой части

1. Как определяется, какой объект прорисовывать. / Передавать объект вместо текста при выборе типа объекта.

2. Как новый объект попадает в хранилище. / Создать событие, через вызов которого в форме можно было бы получать объект извне.
3. Покажите пример лямбда-выражения в коде. / Переделать логику обработки нажатия кнопки «Добавить» на анонимный метод.

Варианты

Вар.	**Базовый объект**	**«Продвинутый» объект**
1.	Военный самолет	Бомбардировщик (с бомбами и дополнительными топливными баками)
2.	Грузовик	Самосвал (с кузовом и тентом)
3.	Гусеничная машина	Бульдозер (с отвалом спереди и рыхлителем сзади)
4.	Локомотив	Электровоз (с «рогами» для подключения к проводам и отсеком под электрические батареи)
5.	Корабль	Теплоход (с трубами и отсеком под топливо)
6.	Бронированная машина	Танк (с башней с орудием и зенитным пулеметом на башне)
7.	Самолет	Аэробус (с дополнительным «отсеком» для пассажиров спереди сверху и с дополнительными двигателями)
8.	Военный корабль	Линкор (с орудийной башней и отсеком под ракеты)
9.	Автобус	Автобус двухэтажный (со вторым этажом и лестницей, ведущей на него)
10.	Лодка	Катер (с мотором в корме и защитным стеклом спереди)
11.	Военный самолет	Истребитель (с ракетами и дополнительными крыльями для лучшей маневренности)

Вар.	Базовый объект	«Продвинутый» объект
12.	Грузовик	Бензовоз (с баком под бензин и сигнальным маяком на кабине)
13.	Гусеничная машина	Экскаватор (с ковшом и опорами для фиксации)
14.	Локомотив	Тепловоз (с трубой и отсеком под топливо)
15.	Корабль	Контейнеровоз (с контейнерами и краном для разгрузки)
16.	Бронированная машина	Самоходная арт. установка (с башней с орудием и залповой батарей сзади)
17.	Самолет	Самолет с радаром (с радаром и дополнительными топливными баками)
18.	Военный корабль	Крейсер (с ракетными шахтами и площадкой под вертолет)
19.	Автобус	Автобус с гармошкой (с дополнительным отсеком, соединенным гармошкой и отдельным входом для него)
20.	Лодка	Катамаран (с поплавками слева и справа от основного корпуса и парусом)
21.	Военный самолет	Штурмовик (с ракетами и бомбами)
22.	Грузовик	Подметально-уборочная машина (с баком под воду и подметательной щеткой)
23.	Гусеничная машина	Подъемный кран (с краном и противовесом к нему)
24.	Локомотив	Монорельс (с магнитной рельсой и второй кабиной в задней части)
25.	Корабль	Лайнер (с дополнительной палубой и бассейном)

Вар.	Базовый объект	«Продвинутый» объект
26.	Бронированная машина	Зенитная установка (с башней с зенитными орудиями и радаром)
27.	Самолет	Гидросамолет (с поплавками и надувной лодкой)
28.	Военный корабль	Авианосец (с палубой для взлета самолетов и рубкой управления)
29.	Автобус	Троллейбус (с «рогами» для подключения к проводам и отсеком под электрические батареи)
30.	Лодка	Парусник (с парусом и усиленным корпусом)

ЛАБОРАТОРНАЯ РАБОТА №6.
ФАЙЛЫ И ПОТОКИ. СОХРАНЕНИЕ И ЗАГРУЗКА ДАННЫХ

Цель

Познакомится с файлами и потоками. Освоить вызов диалоговых окон для загрузки и сохранения данных.

Задание

1. *В тестовое приложение по отслеживанию перемещения объекта по координатам необходимо внести следующие изменения:*
    a. *Реализовать сохранение и загрузку данных по всем объектам, созданным в классе-коллекции объектов.*
    б. *Путь до файла и имя файла сохранения/загрузки указывать через диалоговое окно. Вызов диалоговых окон реализовать через пункты меню.*
    в. *Дополнительно применить фильтр к диалоговому окну, чтобы отображались файлы только определенного формата (*.txt).*

Проектирование

Когда встает вопрос хранения данных в файлах, необходимо решить 2 момента: в каком формате будут данные хранится (бинарный или текстовый), а также, для сложных данных, – в каком порядке буду записываться значения полей объектов сохраняемых.

С первым вопросом в нашем случае все ясно – хранить будем в текстовом формате. Остается решить вопрос как будем сохранять информацию по объекту. И вообще, какую информацию будем сохранять, а какую пропускать. Сохранять будем набор из класса MapsCollection. По каждой записи слова сохранять следует следующую информацию:

- ключ;
- информацию по объекту класса MapWithSetCarsGeneric.

По объекту класса MapWithSetCarsGeneric хранить будем следующую информацию:

- информацию по объекту класса-реализации от абстрактного класса AbstractMap;
- информацию по набору объектов от классов-реализации интерфейса IDrawningObject.

По объекту класса-реализации от абстрактного класса AbstractMap будем хранить только название самого класса, так как иной важной информации по классу нам не требуется.

Классов-реализации интерфейса IDrawningObject у нас не так много, всего один – это DrawningObjectCar. И в этом классе важным будет только информация о том, какой объект там используется, от класса DrawningCar или DrawningSportCar. По объекту DrawningCar или DrawningSportCar важны будет только хранимый там объект EntityCar или EntitySportCar соответственно, а вот координатами, по которым расположен объект пренебрежем, будем их определять при загрузке исходя из размеров форм, куда будем выводить загруженные объекты. В итоге хранить нужно будет только информацию по объекту от класса EntityCar или класса EntitySportCar.

По объекту класса EntityCar хранить будем:

- скорость;
- вес;
- цвет кузова.

А по объекту класса EntitySportCar:

- информацию по базовому классу;
- дополнительный цвет;
- признак наличия обвеса;

- признак наличия антикрыла;
- признак наличия гоночной полосы.

В итоге получается следующая структура данных, сохраняемых в файл:

- по каждой записи словаря из MapsCollection:
    - ключ
    - объект класса MapWithSetCarsGeneric:
        - название класса-реализации AbstractMap
        - список объектов класса DrawningObjectCar:
            - тип объекта (EntityCar или EntitySportCar)
            - скорость;
            - вес;
            - цвет кузова;
            - дополнительный цвет (если объект EntitySportCar);
            - признак наличия обвеса (если объект EntitySportCar);
            - признак наличия антикрыла (если объект EntitySportCar);
            - признак наличия гоночной полосы (если объект EntitySportCar).

Остается определить, как будем записывать эту информацию в файл и потом считывать. Можно каждый элемент записать в отдельной строке, но тогда количество строк будет просто огромным. С другой стороны, каждую запись словаря можно хранить в одной строке, но тогда нужно понимать, где кончается, например, ключ и начинается название класса-реализации AbstractMap. В языках программирования у строк есть замечательный метод разбиения строки на массив строк по символу-разделителю (может даже не символ, а набор символов быть, но так далеко мы не будем заходить). В

принципе, мы можем обойтись тремя символами-разделителями, чтобы отделить данные. Первый символ поможет нам разделить ключ, название класса-реализации AbstractMap и набор объектов класса DrawningObjectCar, второй ключ позволит разделить записи объектов, а третий ключ – информацию по объекту класса EntityCar или EntitySportCar. В качестве таких символов-разделителей выберем знаки пунктуации. В качестве первого символа возьмем символ «|». В качестве второго – «;». В качестве третьего – «:». Таким образом, строка в файле будет представлять собой следующую последовательность:

«ключ|название_класса|тип:скорость:вес:цвет;

тип:скорость:вес:цвет:доп_цвет:признак:признак:признак»

Потребуется прописать методы, которые будут создавать строки на основе информации объектов класса, а также заполнять поля и свойства объектов класса на основании строк.

На основной форме FormMapWithSetCars сделать меню, прописать там 2 пункта: «Сохранить» и «Загрузить». В логике вызывать диалоговые окна сохранения файла или загрузки и вызывать методы, заранее прописанные в MapsCollection, которые и будут записывать данные в файлы или считывать их оттуда.

Реализация

Для получения строк из данных объекта и объектов из строк воспользуемся, в том числе, статичными методами. Для работы с классами DrawningCar и DrawningSportCar сделаем методы-расширения для получения объекта из строки и строку из информации объекта. Методы пропишем в отдельном классе (листинг 6.1).

```
namespace Cars
{
    /// <summary>
    /// Расширение для класса DrawningCar
    /// </summary>
    internal static class ExtentionCar
```

```
    {
        /// <summary>
        /// Разделитель для записи информации по объекту в файл
        /// </summary>
        private static readonly char _separatorForObject = ':';
        /// <summary>
        /// Создание объекта из строки
        /// </summary>
        /// <param name="info"></param>
        /// <returns></returns>
        public static DrawningCar CreateDrawningCar(this string info)
        {
            string[] strs = info.Split(_separatorForObject);
            if (strs.Length == 3)
            {
                return new DrawningCar(Convert.ToInt32(strs[0]),
                    Convert.ToInt32(strs[1]), Color.FromName(strs[2]));
            }
            if (strs.Length == 7)
            {
                return new DrawningSportCar(Convert.ToInt32(strs[0]),
                    Convert.ToInt32(strs[1]), Color.FromName(strs[2]),
                    Color.FromName(strs[3]), Convert.ToBoolean(strs[4]),
                    Convert.ToBoolean(strs[5]), Convert.ToBoolean(strs[6]));
            }
            return null;
        }
        /// <summary>
        /// Получение данных для сохранения в файл
        /// </summary>
        /// <param name="drawningCar"></param>
        /// <returns></returns>
        public static string GetDataForSave(this DrawningCar drawningCar)
        {
            var car = drawningCar.Car;
            var str =
$"{car.Speed}{_separatorForObject}{car.Weight}{_separatorForObject}{car.BodyColor.
Name}";
            if (car is not EntitySportCar sportCar)
            {
                return str;
            }
            return
$"{str}{_separatorForObject}{sportCar.DopColor.Name}{_separatorForObject}{sportCar
.BodyKit}{_separatorForObject}{sportCar.Wing}{_separatorForObject}{sportCar.SportL
ine}";
        }
    }
}
```

Листинг 6.1 – Класс ExtentionCar

С одной стороны, эти методы можно прописать и в классе DrawningCar, но, в таком случае, класс DrawningCar должен «узнать» про дочерний класс DrawningSportCar, что нарушает принципы ООП. Потому, выносим в отдельный класс-расширение.

Далее, для того, чтобы в классе MapWithSetCarsGeneric получать и сохранять информацию по объектам надо расширить интерфейс IDrawningObject методом получения строки из объекта (листинг 6.2), а также в классе DrawningObjectCar добавить статический метод создания объекта IDrawningObject из строки (листинг 6.3).

```
namespace Cars
{
    /// <summary>
    /// Интерфейс для работы с объектом, прорисовываемым на форме
    /// </summary>
    internal interface IDrawningObject
       {
              …
              /// <summary>
              /// Получение информации по объекту
              /// </summary>
              /// <returns></returns>
              string GetInfo();
       }
}
```

Листинг 6.2 – Добавленный метод в интерфейс IDrawningObject

```
namespace Cars
{
    internal class DrawningObjectCar : IDrawningObject
    {
        …
        public string GetInfo() => _car?.GetDataForSave();

        public static IDrawningObject Create(string data) => new
DrawningObjectCar(data.CreateDrawningCar());
    }
}
```

Листинг 6.3 – Добавленные методы класса DrawningObjectCar

Определение статического метода по созданию объекта класса в самом классе довольно распространенная практика. Правда, применяется, в основном, в классах, которые отвечают за описание данных (в нашем случае это будут классы EntityCar и EntitySportCar). Статический метод принимает на вход один или несколько параметров, которые нужно обработать, прежде, чем на их основе создать объект от класса. Также, создание такого метода в классе позволяет оперативно вносить правки в метод в случае изменения полей (свойств) класса (добавление новых, удаление, смена типа данных).

Далее в классе MapWithSetCarsGeneric пропишем также 2 метода все для того же, получение и сохранение данных. При сохранении будем сохранять тип класса-реализации AbstractMap и набор объектов к ней. А при получении будем наполнять набор данных по строкам. При этом при получении уже не нужно передавать тип класса-реализации AbstractMap, так как он будет извлечен при создании объекта класса MapWithSetCarsGeneric (листинг 6.4).

```
namespace Cars
{
    /// <summary>
    /// Карта с набром объектов под нее
    /// </summary>
    /// <typeparam name="T"></typeparam>
    /// <typeparam name="U"></typeparam>
    internal class MapWithSetCarsGeneric<T, U>
        where T : class, IDrawningObject
        where U : AbstractMap
    {
        …
        /// <summary>
        /// Получение данных в виде строки
        /// </summary>
        /// <param name="sep"></param>
        /// <returns></returns>
        public string GetData(char separatorType, char separatorData)
        {
            string data = $"{_map.GetType().Name}{separatorType}";
            foreach (var car in _setCars.GetCars())
            {
                data += $"{car.GetInfo()}{separatorData}";
            }
            return data;
        }
        /// <summary>
        /// Загрузка списка из массива строк
        /// </summary>
        /// <param name="records"></param>
        public void LoadData(string[] records)
        {
            foreach (var rec in records)
            {
                _setCars.Insert(DrawningObjectCar.Create(rec) as T);
            }
        }
    }
}
```

Листинг 6.4 – Новые методы класса MapWithSetCarsGeneric

Далее уже в классе MapsCollection пропишем методы непосредственной записи и чтения данных из файла. Для записи и чтения будем использовать FileStream. Так как для записи надо преобразовывать строку в массив байт, вынесем эту процедуру в отдельный метод. Логика метода сохранения будет

следующий: если есть файл, удаляем его, далее в файл первым записываем слово «MapsCollection», чтобы потом, при чтении понимать, нужный ли файл открываем на чтение.

Логика загрузки будет следующей: проверяем, есть ли файл, если он есть, считываем все в буфер и преобразуем в строку. Строку разбиваем на массив строк по признаку конца строки и проверяем, что в первой строке есть фраза «MapsCollection». Если есть, все следующие строки – это записи словаря. Очищаем словарь и добавляем записи (листинг 6.5).

```
using System.Text;

namespace Cars
{
    /// <summary>
    /// Класс для хранения коллекции карт
    /// </summary>
    internal class MapsCollection
    {
        …
        /// <summary>
        /// Метод записи информации в файл
        /// </summary>
        /// <param name="text">Строка, которую следует записать</param>
        /// <param name="stream">Поток для записи</param>
        private static void WriteToFile(string text, FileStream stream)
        {
            byte[] info = new UTF8Encoding(true).GetBytes(text);
            stream.Write(info, 0, info.Length);
        }
        /// <summary>
        /// Сохранение информации по автомобилям в хранилище в файл
        /// </summary>
        /// <param name="filename">Путь и имя файла</param>
        /// <returns></returns>
        public bool SaveData(string filename)
        {
            if (File.Exists(filename))
            {
                File.Delete(filename);
            }
            using (FileStream fs = new(filename, FileMode.Create))
            {
                WriteToFile($"MapsCollection{Environment.NewLine}", fs);
                foreach (var storage in _mapStorages)
                {

WriteToFile($"{storage.Key}{separatorDict}{storage.Value.GetData(separatorDict,
separatorData)}{Environment.NewLine}", fs);
                }
            }
            return true;
        }

        /// <summary>
```

```
        /// Загрузка нформации по автомобилям на парковках из файла
        /// </summary>
        /// <param name="filename"></param>
        /// <returns></returns>
        public bool LoadData(string filename)
        {
            if (!File.Exists(filename))
            {
                return false;
            }
            string bufferTextFromFile = "";
            using (FileStream fs = new(filename, FileMode.Open))
            {
                byte[] b = new byte[fs.Length];
                UTF8Encoding temp = new(true);
                while (fs.Read(b, 0, b.Length) > 0)
                {
                    bufferTextFromFile += temp.GetString(b);
                }
            }
            var strs = bufferTextFromFile.Split(new char[] { '\n', '\r' },
StringSplitOptions.RemoveEmptyEntries);
            if (!strs[0].Contains("MapsCollection"))
            {
                //если нет такой записи, то это не те данные
                return false;
            }
            //очищаем записи
            _mapStorages.Clear();
            for (int i = 1; i < strs.Length; ++i)
            {
                var elem = strs[i].Split(separatorDict);
                AbstractMap map = null;
                switch (elem[1])
                {
                    case "SimpleMap":
                        map = new SimpleMap();
                        break;
                }
                _mapStorages.Add(elem[0], new
MapWithSetCarsGeneric<IDrawningObject, AbstractMap>(_pictureWidth, _pictureHeight,
map));
                _mapStorages[elem[0]].LoadData(elem[2].Split(separatorData,
StringSplitOptions.RemoveEmptyEntries));
            }
            return true;
        }
    }
}
```

Листинг 6.5 – Методы загрузки и сохранения в файл

Остается на форме добавить меню (элемент MenuStrip), в нем прописать пункт «Файл», а в нем подпункты «Сохранение» и «Загрузка». Также добавим два диалоговых окна openFileDialog и saveFileDialog. У этих окно в свойствах есть элемент Filter, в котором можно указать файлы с каким расширением

отображать. Пропишем в нем свойства так, чтобы отображались только текстовые файлы: «txt file | *.txt».

Остается в логике формы прописать вызовы диалоговых окон при сохранении и загрузки (листинг 6.6).

```
namespace Cars
{
    public partial class FormMapWithSetCars : Form
    {
        …
        /// <summary>
        /// Обработка нажатия "Сохранение"
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void SaveToolStripMenuItem_Click(object sender, EventArgs e)
        {
            if (saveFileDialog.ShowDialog() == DialogResult.OK)
            {
                if (_mapsCollection.SaveData(saveFileDialog.FileName))
                {
                    MessageBox.Show("Сохранение прошло успешно", "Результат",
MessageBoxButtons.OK, MessageBoxIcon.Information);
                }
                else
                {
                    MessageBox.Show("Не сохранилось", "Результат",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                }
            }
        }
        /// <summary>
        /// Обработка нажатия "Загрузка"
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void LoadToolStripMenuItem_Click(object sender, EventArgs e)
        {
            // TODO продумать логику
        }
    }
}
```

Листинг 6.6 – Обработка вызовов пунктов меню

Требования

1. Платформа для проектов – .Net версии 6.0
2. Название проекта должно соответствовать логике задания.
3. Название форм, классов, свойств классов должно соответствовать логике задания.

4. Все элементы форм (заголовки форм, текст в label и т.д.) должны иметь подписи на одном языке (или все русским, или все английским).
5. При сохранении данных использовать StreamWriter.
6. При загрузке использовать StreamReader. Обработку делать построчную (не загружать текст из файла целиком в программу).
7. Прописать логику метода LoadToolStripMenuItem_Click.

Порядок сдачи базовой части

1. Добавить 2 карты (2 элемента словаря).
2. На каждую карту добавить по 2-3 объекта разного типа и характеристик.
3. Сохранить в файл.
4. Закрыть программу.
5. Запустить заново.
6. Загрузить из файла.

Контрольные вопросы к базовой части

1. Где и как вызываются статические методы, используемые при сохранении и загрузке. / Сделать создание объекта по данным из строки через перегрузку метода ToString.
2. Рассказать пошаговый алгоритм сохранения. / Сделать сохранение отдельной карты (передается название карты и имя файла).
3. Рассказать пошаговый алгоритм загрузки. / Сделать сохранение/загрузку только названий карт без содержимого.

Варианты

Вар.	Базовый объект	«Продвинутый» объект
1.	Военный самолет	Бомбардировщик (с бомбами и дополнительными топливными баками)
2.	Грузовик	Самосвал (с кузовом и тентом)
3.	Гусеничная машина	Бульдозер (с отвалом спереди и рыхлителем сзади)
4.	Локомотив	Электровоз (с «рогами» для подключения к проводам и отсеком под электрические батареи)
5.	Корабль	Теплоход (с трубами и отсеком под топливо)
6.	Бронированная машина	Танк (с башней с орудием и зенитным пулеметом на башне)
7.	Самолет	Аэробус (с дополнительным «отсеком» для пассажиров спереди сверху и с дополнительными двигателями)
8.	Военный корабль	Линкор (с орудийной башней и отсеком под ракеты)
9.	Автобус	Автобус двухэтажный (со вторым этажом и лестницей, ведущей на него)
10.	Лодка	Катер (с мотором в корме и защитным стеклом спереди)
11.	Военный самолет	Истребитель (с ракетами и дополнительными крыльями для лучшей маневренности)
12.	Грузовик	Бензовоз (с баком под бензин и сигнальным маяком на кабине)
13.	Гусеничная машина	Экскаватор (с ковшом и опорами для фиксации)
14.	Локомотив	Тепловоз (с трубой и отсеком под топливо)

Вар.	Базовый объект	«Продвинутый» объект
15.	Корабль	Контейнеровоз (с контейнерами и краном для разгрузки)
16.	Бронированная машина	Самоходная арт. установка (с башней с орудием и залповой батарей сзади)
17.	Самолет	Самолет с радаром (с радаром и дополнительными топливными баками)
18.	Военный корабль	Крейсер (с ракетными шахтами и площадкой под вертолет)
19.	Автобус	Автобус с гармошкой (с дополнительным отсеком, соединенным гармошкой и отдельным входом для него)
20.	Лодка	Катамаран (с поплавками слева и справа от основного корпуса и парусом)
21.	Военный самолет	Штурмовик (с ракетами и бомбами)
22.	Грузовик	Подметально-уборочная машина (с баком под воду и подметательной щеткой)
23.	Гусеничная машина	Подъемный кран (с краном и противовесом к нему)
24.	Локомотив	Монорельс (с магнитной рельсой и второй кабиной в задней части)
25.	Корабль	Лайнер (с дополнительной палубой и бассейном)
26.	Бронированная машина	Зенитная установка (с башней с зенитными орудиями и радаром)
27.	Самолет	Гидросамолет (с поплавками и надувной лодкой)
28.	Военный корабль	Авианосец (с палубой для взлета самолетов и рубкой управления)

Вар.	Базовый объект	«Продвинутый» объект
29.	Автобус	Троллейбус (с «рогами» для подключения к проводам и отсеком под электрические батареи)
30.	Лодка	Парусник (с парусом и усиленным корпусом)

ЛАБОРАТОРНАЯ РАБОТА №7.
ОБРАБОТКА ИСКЛЮЧЕНИЙ. ЛОГИРОВАНИЕ ДЕЙСТВИЙ

Цель

Познакомится с логгерами. Познакомится с обработкой исключений, вызовами исключений и созданием собственных исключений.

Задание

1. *В тестовое приложение по отслеживанию перемещения объекта по координатам необходимо внести следующие изменения:*
    a. *Реализовать генерацию и обработку собственных исключений. Создать для этого 2 собственных исключения.*
    б. *Выдавать исключения при работе с файлами, используя существующие классы исключений.*
    в. *Добавить логирование действий пользователя.*

Проектирование

Для генерации собственных исключений создадим новые исключения от класса Exception. Есть негласные правила по созданию собственных классов-исключений:

- название должно оканчиваться на Exception;
- наследоваться от ApplicationException;
- сопровождаться атрибутом [System.Serializable];
- иметь конструктор по умолчанию (Exception());
- иметь конструктор, который устанавливает значение унаследованного свойства Message (Exception(String));
- иметь конструктор для обработки «внутренних исключений» (Exception(String, Exception));
- иметь конструктор для обработки сериализации типа.

В проекте есть два места, где уместно вызывать собственные исключения: при попытке добавить объект в переполненное хранилище и при попытке получить объект с пустого места. Обе эти проверки выполняются в классе SetCarsGeneric в методах Insert и Remove.

В классе MapsCollection изменим логику сохранения и загрузки. Методы теперь не будут возвращать булевское значение. Вместо этого будут генерироваться ошибки, если что-то не так. В данном случае не будем создавать собственные классы ошибок, а воспользуемся стандартным классом Exception или его наследниками.

Перейдем к логированию. Логировать будем действия пользователя и возникающие ошибки. Действия пользователя пойдут по уровню логирования Info, а возникающие ошибки – по уровню логирования Warning. Для реализации задачи логирования действий пользователя воспользуемся готовыми средствами. Первое из этих средств – это набор пакетов Microsoft.Extensions, в которых описана логика работы с логгером. В частности, объявлен интерфейс ILogger, в котором описаны методы логирования сообщений. Помимо этих пакетов потребуется пакет непосредственной реализации ILogger. В примере воспользуемся решением от NLog. Для этого в проект нужно добавить пакет NLog.Extensions.Logging.

Реализация

Первым делом делаем свои классы-наследники от Exception. В классе нам необходимо объявить ряд конструкторов (листинги 7.1, 7,2).

```
using System.Runtime.Serialization;

namespace Cars
{
    [Serializable]
    internal class CarNotFoundException : ApplicationException
    {
        public CarNotFoundException(int i) : base($"Не найден объект по позиции
{i}") { }
        public CarNotFoundException() : base() { }
        public CarNotFoundException(string message) : base(message) { }
        public CarNotFoundException(string message, Exception exception) :
base(message, exception) { }
```

```
        protected CarNotFoundException(SerializationInfo info, StreamingContext
contex) : base(info, contex) { }
    }
}
```

Листинг 7.1 – Код класса CarNotFoundException

```
using System.Runtime.Serialization;

namespace Cars
{
    [Serializable]
    internal class StorageOverflowException : ApplicationException
    {
        public StorageOverflowException(int count) : base($"В наборе превышено
допустимое количество: {count}") { }
        public StorageOverflowException() : base() { }
        public StorageOverflowException(string message) : base(message) { }
        public StorageOverflowException(string message, Exception exception) :
base(message, exception) { }
        protected StorageOverflowException(SerializationInfo info,
StreamingContext contex) : base(info, contex) { }
    }
}
```

Листинг 7.2 – Код класса StorageOverflowException

Далее в классе SetCarsGeneric в методах Insert и Remove делаем выброс этих исключений. В методе Insert, в случае, если размерность списка достигла значения, записанного в _maxCount. В методе Remove, в случае, если по указанной позиции уже пустой элемент находится.

В классе MapWithSetCarsGeneric делать ничего не будем, если возникнет ошибка при добавлении или удалении, она просто пройдет по стеку выше.

А вот в форме уже пропишем оператор try-catch, чтобы отлавливать эти ошибки в методах добавления объекта и извлечения объекта.

Но сперва еще в классе MapsCollection пропишем вызов исключений в методах сохранения и загрузки. В методе сохранения выбрасывать специфические ошибки не требуется, так что там просто меняем bool на void. А вот в методе загрузке, там, где раньше возвращался false делаем выброс ошибок. В первом случае, что не найден файл, во втором, что неверный формат файла (листинг 7.3).

```
using System.Text;

namespace Cars
{
    /// <summary>
    /// Класс для хранения коллекции карт
```

```
    /// </summary>
    internal class MapsCollection
    {
        …
        /// <summary>
        /// Сохранение информации по автомобилям в хранилище в файл
        /// </summary>
        /// <param name="filename">Путь и имя файла</param>
        /// <returns></returns>
        public void SaveData(string filename)
        {
            if (File.Exists(filename))
            {
                File.Delete(filename);
            }
            using (FileStream fs = new(filename, FileMode.Create))
            {
                WriteToFile($"MapsCollection{Environment.NewLine}", fs);
                foreach (var storage in _mapStorages)
                {

WriteToFile($"{storage.Key}{separatorDict}{storage.Value.GetData(separatorDict,
separatorData)}{Environment.NewLine}", fs);
                }
            }
        }

        /// <summary>
        /// Загрузка нформации по автомобилям на парковках из файла
        /// </summary>
        /// <param name="filename"></param>
        /// <returns></returns>
        public void LoadData(string filename)
        {
            if (!File.Exists(filename))
            {
                throw new Exception("Файл не найден");
            }
            string bufferTextFromFile = "";
            using (FileStream fs = new(filename, FileMode.Open))
            {
                byte[] b = new byte[fs.Length];
                UTF8Encoding temp = new(true);
                while (fs.Read(b, 0, b.Length) > 0)
                {
                    bufferTextFromFile += temp.GetString(b);
                }
            }
            var strs = bufferTextFromFile.Split(new char[] { '\n', '\r' },
StringSplitOptions.RemoveEmptyEntries);
            if (!strs[0].Contains("MapsCollection"))
            {
                //если нет такой записи, то это не те данные
                throw new Exception("Формат данных в файле не правильный");
}
            //очищаем записи
            _mapStorages.Clear();
            for (int i = 1; i < strs.Length; ++i)
            {
                var elem = strs[i].Split(separatorDict);
                AbstractMap map = null;
                switch (elem[1])
                {
```

```
                    case "SimpleMap":
                        map = new SimpleMap();
                        break;
                }
                _mapStorages.Add(elem[0], new
MapWithSetCarsGeneric<IDrawningObject, AbstractMap>(_pictureWidth, _pictureHeight,
map));
                _mapStorages[elem[0]].LoadData(elem[2].Split(separatorData,
StringSplitOptions.RemoveEmptyEntries));
            }
        }
    }
}
```

Листинг 7.3 – Обновленные методы класса MapsCollection

Само собой, в логике формы потребуется внести изменения в методы загрузки и сохранения, а также добавить туда операторы try-catch (листинг 7.4).

```
using Microsoft.Extensions.Logging;

namespace Cars
{
    public partial class FormMapWithSetCars : Form
    {
        …
        /// <summary>
        /// Удаление объекта
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void ButtonRemoveCar_Click(object sender, EventArgs e)
        {
            if (listBoxMaps.SelectedIndex == -1)
            {
                return;
            }
            if (string.IsNullOrEmpty(maskedTextBoxPosition.Text))
            {
                return;
            }
            if (MessageBox.Show("Удалить объект?", "Удаление",
MessageBoxButtons.YesNo, MessageBoxIcon.Question) == DialogResult.No)
            {
                return;
            }
            int pos = Convert.ToInt32(maskedTextBoxPosition.Text);
            try
            {
                if (_mapsCollection[listBoxMaps.SelectedItem?.ToString() ??
string.Empty] - pos)
                {
                    MessageBox.Show("Объект удален");
                    pictureBox.Image =
_mapsCollection[listBoxMaps.SelectedItem?.ToString() ?? string.Empty].ShowSet();
                    _logger.LogInformation("Добавлен объект");
                }
                else
                {
                    MessageBox.Show("Не удалось удалить объект");
```

```
                }
            }
            catch (CarNotFoundException ex)
            {
                MessageBox.Show($"Ошибка удаления: {ex.Message}");
            }
            catch (Exception ex)
            {
                MessageBox.Show($"Неизвестная ошибка: {ex.Message}");
            }
        }
        …
        /// <summary>
        /// Обработка нажатия "Сохранение"
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void SaveToolStripMenuItem_Click(object sender, EventArgs e)
        {
            if (saveFileDialog.ShowDialog() == DialogResult.OK)
            {
                try
                {
                    _mapsCollection.SaveData(saveFileDialog.FileName);
                    MessageBox.Show("Сохранение прошло успешно", "Результат",
MessageBoxButtons.OK, MessageBoxIcon.Information);
                }
                catch(Exception ex)
                {
                    MessageBox.Show($"Не сохранилось: {ex.Message}", "Результат",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                }
            }
        }
    }
}
```

Листинг 7.4 – Обновленные методы логики формы FormMapWithSetCars

Остается разобраться с логгером. Для этого сперва через Nuget-пакеты подключим ряд пакетов:

- Microsoft.Extensions.DependencyInjection
- Microsoft.Extensions.Logging
- Microsoft.Extensions.Logging.Abstractions
- NLog.Extensions.Logging

Далее в логике формы объявим логгер и в конструкторе укажем, что будем принимать логгер в качестве передаваемого параметра. А в методах пропишем логирование действий пользователя. В качестве уровня логирования выберем уровень Info (листинг 7.5).

```
using Microsoft.Extensions.Logging;
```

```
namespace Cars
{
    public partial class FormMapWithSetCars : Form
    {
        …
        /// <summary>
        /// Логер
        /// </summary>
        private readonly ILogger _logger;
        /// <summary>
        /// Конструктор
        /// </summary>
        public FormMapWithSetCars(ILogger<FormMapWithSetCars> logger)
        {
            InitializeComponent();
            _logger = logger;
            …
        }
        /// <summary>
        /// Добавление карты
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void ButtonAddMap_Click(object sender, EventArgs e)
        {
            if (comboBoxSelectorMap.SelectedIndex == -1 ||
string.IsNullOrEmpty(textBoxNewMapName.Text))
            {
                MessageBox.Show("Не все данные заполнены", "Ошибка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                return;
            }
            if (!_mapsDict.ContainsKey(comboBoxSelectorMap.Text))
            {
                MessageBox.Show("Нет такой карты", "Ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
                return;
            }
            _mapsCollection.AddMap(textBoxNewMapName.Text,
_mapsDict[comboBoxSelectorMap.Text]);
            ReloadMaps();
            _logger.LogInformation($"Добавлена карта {textBoxNewMapName.Text}");
        }
        …
}
```

Листинг 7.5 – Работа с логгером в логике формы FormMapWithSetCars

Теперь надо в классе Program в месте вызова конструктора формы FormMapWithSetCars передавать логгер. А для этого нужно сделать экземпляр класса логгера. В современном программировании эту задачу возложили на механизм внедрения зависимостей (Dependency Injection или DI). С принципом работы этого механизма мы ознакомимся в другом семестре, поэтому просто рассмотрим код работы этого механизма (листинг 7.6).

```
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Logging;
```

```
using NLog.Extensions.Logging;

namespace Cars
{
    internal static class Program
    {
        /// <summary>
        ///  The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            // To customize application configuration such as set high DPI
settings or default font,
            // see https://aka.ms/applicationconfiguration.
            ApplicationConfiguration.Initialize();
            var services = new ServiceCollection();
            ConfigureServices(services);
            using (ServiceProvider serviceProvider =
services.BuildServiceProvider())
            {

Application.Run(serviceProvider.GetRequiredService<FormMapWithSetCars>());
            }
        }

        private static void ConfigureServices(ServiceCollection services)
        {
            services.AddSingleton<FormMapWithSetCars>()
                    .AddLogging(option =>
                    {
                        option.SetMinimumLevel(LogLevel.Information);
                        option.AddNLog("nlog.config");
                    });
        }
    }
}
```

Листинг 7.6 – Класс Program

В методе Main создается объект от класса ServiceCollection. Далее в методе ConfigureServices выполняется его настройка, в частности, настройка логгера. Тут указывается, что конфигурация логгера будет выполнена через NLog, а файл конфига называется nlog.config. Далее создается объект класса ServiceProvider и через его метод GetRequiredService получается экземпляр формы FormMapWithSetCars, в конструктор которого автоматически будет подставлен объект-реализация интерфейса ILogger.

Остается только создать файл-конфигурацию nlog.config в проекте (листинг 7.7).

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
      <nlog xmlns="http://www.nlog-project.org/schemas/NLog.xsd"
```

```
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        autoReload="true" internalLogLevel="Info">

              <targets>
                    <target xsi:type="File" name="tofile" fileName="carlog-
${shortdate}.log" />
              </targets>

              <rules>
                    <logger name="*" minlevel="Debug" writeTo="tofile" />
              </rules>
       </nlog>
</configuration>
```

Листинг 7.7 – Файл nlog.config

В данной конфигурации прописано сохранение логов в файл с определенным названием. При этом, в названии файла с логами должна фигурировать дата. Далее в правилах указали, что для всех логов уровня Debug и выше должна происходить запись в этот файл.

Требования

1. Платформа для проектов – .Net версии 6.0
2. Название проекта должно соответствовать логике задания.
3. Название форм, классов, свойств классов должно соответствовать логике задания.
4. Все элементы форм (заголовки форм, текст в label и т.д.) должны иметь подписи на одном языке (или все русским, или все английским).
5. Прописать вызов исключений в методах Insert и Remove класса SetCarsGeneric.
6. Добавить логирование ошибок (уровень Warning). Логи ошибок выводить в тот же файл, что и выводятся логи уровня Info.
7. В MapsCollection в методе загрузки в местах генерации ошибок вместо класса Exception использовать его наследников, которые будут соответствовать возникающей ошибке.

8. Убедиться, что логируются следующие действия пользователя: добавление карты, удаление карты, переход на карту, добавление объекта, удаление объекта, сохранение данных, загрузка данных.
9. Вместо пакета NLog использовать пакет Serilog.

Порядок сдачи базовой части

1. Добавить карту (элемент словаря).
2. На карту добавить объекты в таком количестве, чтобы при добавлении следующего возникала ошибка переполнения.
3. Сохранить в файл.
4. Закрыть программу.
5. Запустить заново.
6. Загрузить из файла.
7. Показать вывод ошибки переполнения.
8. Показать ошибку извлечения с пустого места.
9. Закрыть программу.
10. Показать файл с логами.

Контрольные вопросы к базовой части

1. Как обрабатываются ошибки (свой класс-исключение, выброс ошибки, обработка) / В классе MapWithSetCarsGeneric сделать проброс ошибки.
2. Как записывать лог? / Выдавать ошибку, если пытаемся добавить пустой объект в хранилище.
3. Как и где настроить формат записи логов? / Изменить формат записи на «Уровень логирования/дата/текст».

Варианты

Вар.	Базовый объект	«Продвинутый» объект
1.	Военный самолет	Бомбардировщик (с бомбами и дополнительными топливными баками)
2.	Грузовик	Самосвал (с кузовом и тентом)
3.	Гусеничная машина	Бульдозер (с отвалом спереди и рыхлителем сзади)
4.	Локомотив	Электровоз (с «рогами» для подключения к проводам и отсеком под электрические батареи)
5.	Корабль	Теплоход (с трубами и отсеком под топливо)
6.	Бронированная машина	Танк (с башней с орудием и зенитным пулеметом на башне)
7.	Самолет	Аэробус (с дополнительным «отсеком» для пассажиров спереди сверху и с дополнительными двигателями)
8.	Военный корабль	Линкор (с орудийной башней и отсеком под ракеты)
9.	Автобус	Автобус двухэтажный (со вторым этажом и лестницей, ведущей на него)
10.	Лодка	Катер (с мотором в корме и защитным стеклом спереди)
11.	Военный самолет	Истребитель (с ракетами и дополнительными крыльями для лучшей маневренности)
12.	Грузовик	Бензовоз (с баком под бензин и сигнальным маяком на кабине)
13.	Гусеничная машина	Экскаватор (с ковшом и опорами для фиксации)
14.	Локомотив	Тепловоз (с трубой и отсеком под топливо)

Вар.	Базовый объект	«Продвинутый» объект
15.	Корабль	Контейнеровоз (с контейнерами и краном для разгрузки)
16.	Бронированная машина	Самоходная арт. установка (с башней с орудием и залповой батарей сзади)
17.	Самолет	Самолет с радаром (с радаром и дополнительными топливными баками)
18.	Военный корабль	Крейсер (с ракетными шахтами и площадкой под вертолет)
19.	Автобус	Автобус с гармошкой (с дополнительным отсеком, соединенным гармошкой и отдельным входом для него)
20.	Лодка	Катамаран (с поплавками слева и справа от основного корпуса и парусом)
21.	Военный самолет	Штурмовик (с ракетами и бомбами)
22.	Грузовик	Подметально-уборочная машина (с баком под воду и подметательной щеткой)
23.	Гусеничная машина	Подъемный кран (с краном и противовесом к нему)
24.	Локомотив	Монорельс (с магнитной рельсой и второй кабиной в задней части)
25.	Корабль	Лайнер (с дополнительной палубой и бассейном)
26.	Бронированная машина	Зенитная установка (с башней с зенитными орудиями и радаром)
27.	Самолет	Гидросамолет (с поплавками и надувной лодкой)
28.	Военный корабль	Авианосец (с палубой для взлета самолетов и рубкой управления)

Вар.	Базовый объект	«Продвинутый» объект
29.	Автобус	Троллейбус (с «рогами» для подключения к проводам и отсеком под электрические батареи)
30.	Лодка	Парусник (с парусом и усиленным корпусом)

ЛАБОРАТОРНАЯ РАБОТА №8.
СТАНДАРТНЫЕ ИНТЕРФЕЙСЫ

Цель

Ознакомится с возможностью применения стандартных интерфейсов.

Задание

1. *В тестовое приложение по отслеживанию перемещения объекта по координатам необходимо внести следующие изменения:*
    а. *При добавлении объекта реализовать проверку уникальности этого объекта в хранилище, куда его добавляют (т.е. нет второго объекта с такими же характеристиками).*
    б. *Реализовать возможность сортировки объектов в хранилище по двум разным критериям: по типу объектов (простой или продвинутый), по основному цвету. В случае равенства первого критерия далее объекты сортируются по скорости и по весу.*
    в. *Реализовать возможность создания словаря, где в качестве ключа будет выступать объект от класса-наследника AbstractMap, а значением – коллекция объектов DrawningCar.*

Проектирование

Для проверки уникальности объекта в хранилище при добавления нового объекта его нужно сравнивать со всеми уже существующими в хранилище. Делать будем это с помощью метода Contains класса List<T>. Этот метод «знает» как сравнивать простые типы (int, double), но не знает, как сравнивать объекты классов. Чтобы «научить» его сравнивать объекты класса,

этот класс надо унаследовать от интерфейса IEquatable. В нем требуется реализовать метод Equals. Наш список задается в классе SetCarsGeneric от параметра T, который ограничен в классе MapWithSetCarsGeneric интерфейсом IDrawningObject. Поэтому сам интерфейс унаследуем от интерфейса IEquatable.

Для реализации второго пункта потребуется сделать 2 класса-наследника интерфейса IComparer, так как нам потребуется 2 различных способа сортировки списка. Вызывать сортировку будем опять же у самого списка через его метод Sort, куда будем передавать объект от одного из этих двух классов. В зависимости от передаваемого объекта класса-наследника интерфейса IComparer будет меняться порядок объектов в списке. Либо они будут упорядочены по типу (сперва, например, продвинутые, а потом обычные), либо по цвету.

У словаря главным является уникальность ключей. Если ключ задается каким-то классом, то этот класс должен реализовывать тот же интерфейс IEquatable. Таким образом, класс AbstractMap надо будет унаследовать от IEquatable. Метод определим прям в этом классе и в качестве определения по равенству будем сравнивать параметры класса (размеры поля, размеры ячейки, значения в массивах).

Реализация

Первым шагом будет унаследование интерфейса IDrawningObject от интерфейса IEquatable. Выберем параметризованный вариант интерфейса IEquatable. В качестве параметра укажем IDrawningObject (листинг 8.1).

```
namespace Cars
{
	/// <summary>
	/// Интерфейс для работы с объектом, прорисовываемым на форме
	/// </summary>
	internal interface IDrawningObject : IEquatable<IDrawningObject>
	{
		…
	}
}
```

Листинг 8.1 – Интерфейс IDrawningObject

Далее перейдем в реализацию интерфейса: класс DrawningObjectCar. Тут потребуется реализовать новый метод Equals (листинг 8.2).

```
namespace Cars
{
    internal class DrawningObjectCar : IDrawningObject
    {
        …
        public bool Equals(IDrawningObject? other)
        {
            if (other == null)
            {
                return false;
            }
            var otherCar = other as DrawningObjectCar;
            if (otherCar == null)
            {
                return false;
            }
            var car = _car.Car;
            var otherCarCar = otherCar._car.Car;
            if (car.Speed != otherCarCar.Speed)
            {
                return false;
            }
            if (car.Weight != otherCarCar.Weight)
            {
                return false;
            }
            if (car.BodyColor != otherCarCar.BodyColor)
            {
                return false;
            }
            // TODO доделать проверки в случае продвинутого объекта
            return true;
        }
    }
}
```

Листинг 8.2 – Метод Equals класса DrawningObjectCar

Так как метод сравнения должен вызываться в классе SetCarsGeneric, то дополнительно ограничим параметр T, чтобы он был унаследован также от интерфейса IEquatable. Ну и в методе Insert потребуется проверка, есть ли в List уже такой объект или нет (листинг 8.3).

```
namespace Cars
{
    /// <summary>
    /// Параметризованный набор объектов
    /// </summary>
    /// <typeparam name="T"></typeparam>
    internal class SetCarsGeneric<T>
            where T : class, IEquatable<T>
      {
            …
```

```
            /// <summary>
            /// Добавление объекта в набор на конкретную позицию
            /// </summary>
            /// <param name="car">Добавляемый автомобиль</param>
            /// <param name="position">Позиция</param>
            /// <returns></returns>
            public bool Insert(T car, int position)
            {
                // TODO проверка на уникальность
                // TODO проверка на _maxCount
                // TODO проверка позиции
                // TODO вставка по позиции
                _places[position] = car;
                return true;
            }
            …
        }
}
```

Листинг 8.3 – Обновленный класс SetCarsGeneric

Первый шаг готов, теперь перейдем к сортировке. Так как потребуется 2 варианта сортировки, то сделаем 2 класса-наследника от IComparer. Первый будет отвечать за сортировку по типам, второй – по цветам. Логика в них будет примерно одинаковая, сперва проверка, что оба объекта null или не null. Затем для первого класса проверка по типу, для второго – по цвету. Далее проверка по остальным параметрам, если первый совпадает (листинги 8.4 и 8.5).

```
namespace Cars
{
    internal class CarCompareByType : IComparer<IDrawningObject>
    {
        public int Compare(IDrawningObject? x, IDrawningObject? y)
        {
            if (x == null && y == null)
            {
                return 0;
            }
            if (x == null && y != null)
            {
                return 1;
            }
            if (x != null && y == null)
            {
                return -1;
            }
            var xCar = x as DrawningObjectCar;
            var yCar = y as DrawningObjectCar;
            if (xCar == null && yCar == null)
            {
                return 0;
            }
            if (xCar == null && yCar != null)
            {
                return 1;
            }
            if (xCar != null && yCar == null)
```

```
            {
                return -1;
            }
            if (xCar.GetCar.GetType().Name != yCar.GetCar.GetType().Name)
            {
                if (xCar.GetCar.GetType().Name == "DrawningCar")
                {
                    return -1;
                }
                return 1;
            }
            var speedCompare =
xCar.GetCar.Car.Speed.CompareTo(yCar.GetCar.Car.Speed);
            if (speedCompare != 0)
            {
                return speedCompare;
            }
            return xCar.GetCar.Car.Weight.CompareTo(yCar.GetCar.Car.Weight);
        }
    }
}
```

Листинг 8.4 – Класс CarCompareByType

```
namespace Cars
{
    internal class CarCompareByColor : IComparer<IDrawningObject>
    {
        public int Compare(IDrawningObject? x, IDrawningObject? y)
        {
            // TODO реализовать логику сравнения
            throw new NotImplementedException();
        }
    }
}
```

Листинг 8.5 – Класс CarCompareByColor

Для нормально работы этого класса потребуется в классе DrawningObjectCar сделать метод или свойство с get, через которое можно было бы получить ссылку на объект DrawningCar, чтобы сравнить по типу, скорости и весу. В данном случае сделаем свойство (листинг 8.6).

```
namespace Cars
{
    internal class DrawningObjectCar : IDrawningObject
    {
        …

        public DrawningCar GetCar => _car;

        …
        }
    }
}
```

Листинг 8.6 – Добавление свойства в класс DrawningObjectCar

Далее в классах SetCarsGeneric и MapWithSetCarsGeneric создаем методы для сортировки. В классе SetCarsGeneric буде выполняться непосредственно сортировка, так как список объявлен там. А в классе MapWithSetCarsGeneric будет вызываться новый метод класса SetCarsGeneric (листинги 8.7 и 8.8).

```
namespace Cars
{
    /// <summary>
    /// Параметризованный набор объектов
    /// </summary>
    /// <typeparam name="T"></typeparam>
    internal class SetCarsGeneric<T>
        where T : class, IEquatable<T>
    {
        …
        /// <summary>
        /// Сортировка набора объектов
        /// </summary>
        /// <param name="comparer"></param>
        public void SortSet(IComparer<T> comparer)
        {
            if (comparer == null)
            {
                return;
            }
            _places.Sort(comparer);
        }
    }
}
```

Листинг 8.7 – Добавление метода сортировки в класс SetCarsGeneric

```
namespace Cars
{
    /// <summary>
    /// Карта с набром объектов под нее
    /// </summary>
    /// <typeparam name="T"></typeparam>
    /// <typeparam name="U"></typeparam>
    internal class MapWithSetCarsGeneric<T, U>
        where T : class, IDrawningObject, IEquatable<T>
        where U : AbstractMap
    {
        …
        /// <summary>
        /// Сортировка
        /// </summary>
        /// <param name="comparer"></param>
        public void Sort(IComparer<T> comparer)
        {
            _setCars.SortSet(comparer);
        }
        …
    }
}
```

Листинг 8.8 – Добавление метода сортировки в класс MapWithSetCarsGeneric

Остается на форме сделать 2 кнопки: «сортировка по типу» и «сортировка по цвету». И вызывать метод сортировки класса MapWithSetCarsGeneric передавая туда объект от класса CarCompareByType или от класса CarCompareByColor.

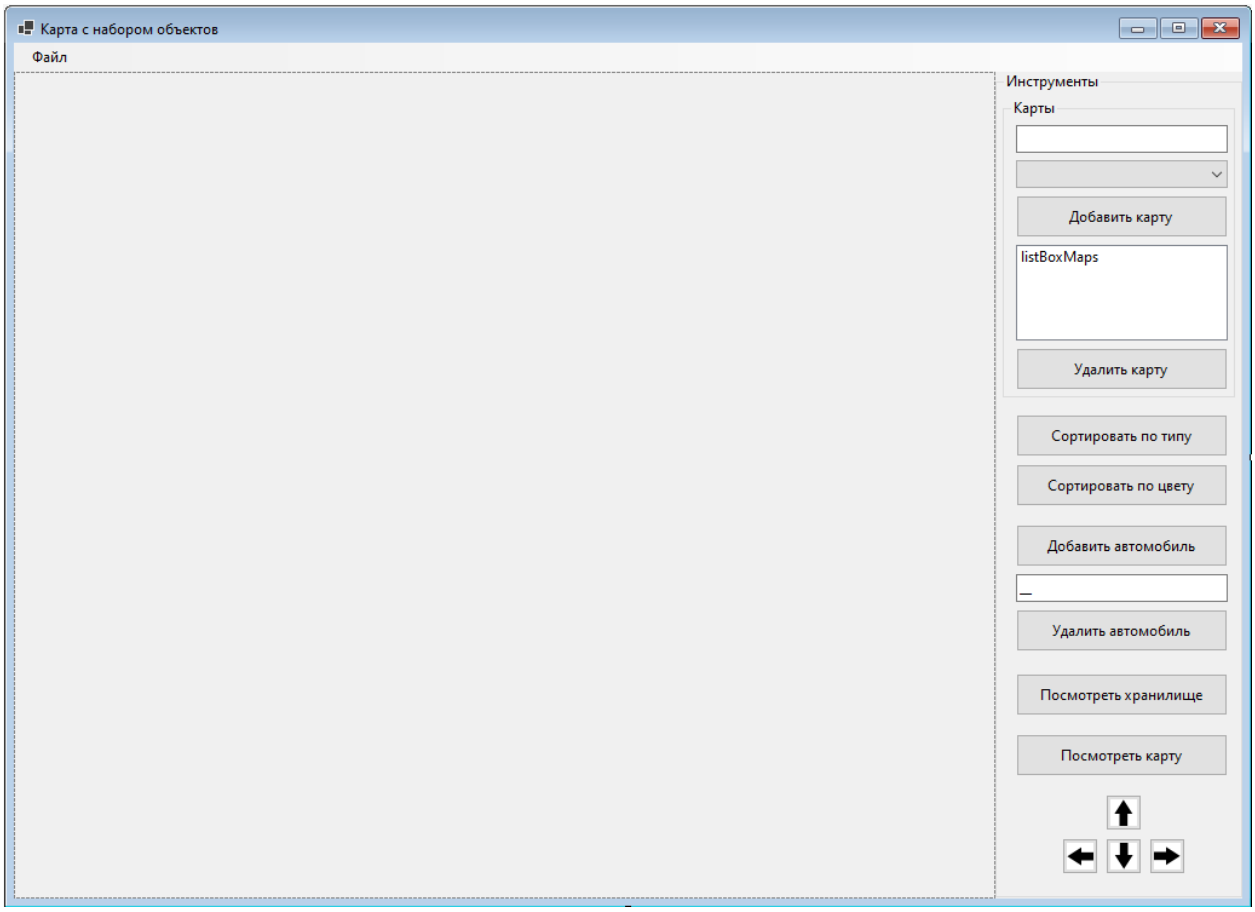


Рисунок 8.1 – Обновленная форма FormMapWithSetCars

Логика будет следующая (листинг 8.9).

```
using Microsoft.Extensions.Logging;

namespace Cars
{
    public partial class FormMapWithSetCars : Form
    {
        …
        /// <summary>
        /// Сортировка по типу
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void ButtonSortByType_Click(object sender, EventArgs e)
        {
            if (listBoxMaps.SelectedIndex == -1)
            {
                return;
            }
```

```
            _mapsCollection[listBoxMaps.SelectedItem?.ToString() ??
string.Empty].Sort(new CarCompareByType());
            pictureBox.Image =
_mapsCollection[listBoxMaps.SelectedItem?.ToString() ?? string.Empty].ShowSet();
        }
        /// <summary>
        /// Сортировка по цвету
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void ButtonSortByColor_Click(object sender, EventArgs e)
        {
            // TODO прописать логику
        }
    }
}
```

Листинг 8.9 – Добавление методов в логику формы FormMapWithSetCars

Второй шаг готов. Для третьего шага требуется класс AbstractMap унаследовать от IEquatable. Такое мы уже делали для класса DrawningObjectCar. Все будет аналогичным, только проверки будут свои. Потребуется проверить, что у объекта other, который будет поступать на вход метода Equals значения полей _width, _height, _size_x, _size_y одинаковые и значения в массивах _map также одинаковы.

Требования

1. Платформа для проектов – .Net версии 6.0
2. Название проекта должно соответствовать логике задания.
3. Название форм, классов, свойств классов должно соответствовать логике задания.
4. Все элементы форм (заголовки форм, текст в label и т.д.) должны иметь подписи на одном языке (или все русским, или все английским).
5. Дописать логику метода Equals класса DrawningObjectCar.
6. Прописать логику метода Compare класса CarCompareByColor.
7. Прописать логику метода ButtonSortByColor_Click формы FormMapWithSetCars.

8. Унаследовать класс AbstractMap от IEquatable и прописать логику метода Equals.

Порядок сдачи базовой части

1. Добавить карту (элемент словаря).
2. На карту добавить простой объект определенного цвета.
3. Попытаться добавить такой же объект.
4. Добавить объект продвинутого типа с таким же цветом.
5. Попытаться добавить еще один такой же продвинутый объект.
6. Добавить еще один простой объект другого цвета.
7. Отсортировать объекты по типу.
8. Отсортировать объекты по цвету.

Контрольные вопросы к базовой части

1. Как реализована сортировка по типу или цвету? / Прописать логику для сравнения двух объектов класса MapsCollection.
2. Как реализована проверка на уникальность объектов в наборе? / Прописать логику для общего (единого) варианта сортировки объектов класса MapsCollection.
3. Как реализована возможность использования класса AbstractMap в качестве ключа для словаря? / Прописать логику для частного (конкретного) варианта сортировки объектов класса MapsCollection.

Варианты

Вар.	**Базовый объект**	**«Продвинутый» объект**
1.	Военный самолет	Бомбардировщик (с бомбами и дополнительными топливными баками)

Вар.	Базовый объект	«Продвинутый» объект
2.	Грузовик	Самосвал (с кузовом и тентом)
3.	Гусеничная машина	Бульдозер (с отвалом спереди и рыхлителем сзади)
4.	Локомотив	Электровоз (с «рогами» для подключения к проводам и отсеком под электрические батареи)
5.	Корабль	Теплоход (с трубами и отсеком под топливо)
6.	Бронированная машина	Танк (с башней с орудием и зенитным пулеметом на башне)
7.	Самолет	Аэробус (с дополнительным «отсеком» для пассажиров спереди сверху и с дополнительными двигателями)
8.	Военный корабль	Линкор (с орудийной башней и отсеком под ракеты)
9.	Автобус	Автобус двухэтажный (со вторым этажом и лестницей, ведущей на него)
10.	Лодка	Катер (с мотором в корме и защитным стеклом спереди)
11.	Военный самолет	Истребитель (с ракетами и дополнительными крыльями для лучшей маневренности)
12.	Грузовик	Бензовоз (с баком под бензин и сигнальным маяком на кабине)
13.	Гусеничная машина	Экскаватор (с ковшом и опорами для фиксации)
14.	Локомотив	Тепловоз (с трубой и отсеком под топливо)
15.	Корабль	Контейнеровоз (с контейнерами и краном для разгрузки)

Вар.	Базовый объект	«Продвинутый» объект
16.	Бронированная машина	Самоходная арт. установка (с башней с орудием и залповой батарей сзади)
17.	Самолет	Самолет с радаром (с радаром и дополнительными топливными баками)
18.	Военный корабль	Крейсер (с ракетными шахтами и площадкой под вертолет)
19.	Автобус	Автобус с гармошкой (с дополнительным отсеком, соединенным гармошкой и отдельным входом для него)
20.	Лодка	Катамаран (с поплавками слева и справа от основного корпуса и парусом)
21.	Военный самолет	Штурмовик (с ракетами и бомбами)
22.	Грузовик	Подметально-уборочная машина (с баком под воду и подметательной щеткой)
23.	Гусеничная машина	Подъемный кран (с краном и противовесом к нему)
24.	Локомотив	Монорельс (с магнитной рельсой и второй кабиной в задней части)
25.	Корабль	Лайнер (с дополнительной палубой и бассейном)
26.	Бронированная машина	Зенитная установка (с башней с зенитными орудиями и радаром)
27.	Самолет	Гидросамолет (с поплавками и надувной лодкой)
28.	Военный корабль	Авианосец (с палубой для взлета самолетов и рубкой управления)
29.	Автобус	Троллейбус (с «рогами» для подключения к проводам и отсеком под электрические батареи)

Вар.	Базовый объект	«Продвинутый» объект
30.	Лодка	Парусник (с парусом и усиленным корпусом)

КУРСОВАЯ РАБОТА. ЗАДАНИЕ НА 3

Выполнить 8 лабораторных работ.

КУРСОВАЯ РАБОТА. ЗАДАНИЕ НА 4

Реализовать задание на «3». Дополнительно:

- Реализовать работу с БД. Хранить информацию об уровнях в системе и о событиях в системе (добавление и извлечение объектов в хранилище). Уровни должны задаваться при создании БД и выводится на основной форме (5 уровней).
- Создать форму для вывода отчета, на которой можно выводить информацию о добавленных и извлеченных объектов из хранилища на определенном уровне (выбирает пользователь) и за определенный период (2 даты, выбирает пользователь). Форму вызывать с основной формы через меню.

КУРСОВАЯ РАБОТА. ЗАДАНИЕ НА 5

Реализовать задание на «4». Дополнительно:

- На форме формирования отчета добавить 2 кнопки для выгрузки данных в Word/Excel (выбрать один из вариантов самостоятельно) и в PDF. Должен запрашиваться куда (путь) и как (имя файла) сохраняем отчет. В отчете выводить заголовок (продумать самим как будет называться), параметры фильтрации (выбранный уровень и даты диапазона) и таблица с результатом. Если данных за период нет, вместо таблицы соответствующая надпись. По завершению выгрузки оповестить об этом пользователя через MessageBox.

ОФОРМЛЕНИЕ ЗАПИСКИ И ПРЕЗЕНТАЦИИ

На основе разработанного программного продукта, оформить пояснительную записку.

Основной текст записки: Times New Roman, 14 шрифт, полуторный интервал, отступ первой строк 1.27.

Пояснительная записка должна содержать следующие пункты:

- Титульный лист (приложение 1).
- Задание на курсовую работу (приложение 2).
- Отзыв руководителя (приложение 3).
- Введение. Описывается актуальность задачи.
- Первая глава. Теоретическая. Описывается предметная область, ТЗ.
- Вторая глава. Руководства. Приводится руководство пользователя для разработанного проекта.
- Третья глава. Руководства. Приводится руководство программиста для разработанного проекта.
- Четвертая глава. Тестирование. Приводится пример работы всего функционала.
- Заключение. Выводы по проделанной работе, какие технологии были применены и освоены.
- Список литературы. Не менее **10** источников с ссылками в тексте.
- Приложение. Листинг кода (8 шрифт, 3 колонки).

Записка предоставляется в электронном виде преподавателю для проверки.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. METANIT.COM. Сайт о программировании [Электронный ресурс] / Режим доступа: https://metanit.com/sharp/. – Загл. с экрана.
2. ProfessorWeb. .Net & Web Programming [Электронный ресурс] / Режим доступа: https://professorweb.ru/. – Загл. с экрана.
3. Tiberiu Covaci, Rod Stephens, Vincent Varallo, Gerry O'Brien. MCSD Certification Toolkit (Exam 70-483) // Published by John Wiley & Sons, Inc. – 2013. – 656p.
4. MCTS Self-Paced Training Kit (Exam 70-536): Microsoft .NET Framework– Application Development Foundation, Second Edition eBook // Published by Microsoft Press. – 2009. – 829 p.
5. Решения и проекты в Visual Studio [Электронный ресурс] / Режим доступа: https://docs.microsoft.com/ru-ru/visualstudio/ide/solutions-and-projects-in-visual-studio
6. Каковы файлы obj и bin (созданные Visual Studio)? [Электронный ресурс] / Режим доступа: http://qaru.site/questions/48688/what-are-the-obj-and-bin-folders-created-by-visual-studio-used-for

ПРИЛОЖЕНИЕ 1

Титульный лист

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет ИСТ
Кафедра «Информационные системы»
Дисциплина «Разработка профессиональных приложений»

КУРСОВАЯ РАБОТА

Тема указать свою тему по приказу

Выполнил студент _____ /Фамилия И.О./
подпись инициалы, фамилия

Курс второй Группа ИСЭбд-21

Направление/специальность 09.03.03 «Прикладная информатика» (профиль «Прикладная информатика в экономике»)

Руководитель ст. преподаватель Эгов Е.Н.
должность, ученая степень, ученое звание фамилия, имя, отчество

Дата сдачи:
«____»_____2020 г.

Дата защиты:
«____»_____2020 г.

Оценка: _____

Ульяновск
2020 г.

ПРИЛОЖЕНИЕ 2

Задание на курсовую работу

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет ИСТ
Кафедра «Информационные системы»
Дисциплина «Разработка профессиональных приложений»

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Студенту ИСЭбд-21 Фамилия И.О.
группа фамилия, инициалы
Тема работы указать свою тему по приказу
Срок сдачи законченной работы «___» _____2020 г.
Исходные данные к работе: описание задания по теме, утвержденной распоряжением деканата ФИСТ
Рекомендуемая литература: курс лекций по дисциплине «Технологии программирования», методические указания к лабораторным работам по дисциплине «Технологии программирования», интернет-источники.
Содержание пояснительной записки (перечень подлежащих разработке вопросов)
Введение. Описание актуальности задачи.
Первая глава. Описание предметной области, поиск аналогов, ТЗ.
Вторая глава. Представление диаграмм с их описанием.
Третья глава. Представление руководств пользователя и программиста для разработанного проекта
Четвертая глава. Тестирование. Приводится пример работы всего функционала, озвученного в ТЗ и use-case диаграмме.
Перечень графического материала (с точным указанием обязательных чертежей)
Диаграммы UML: диаграмма вариантов использования (use-case), диаграмма последовательности (sequence), диаграмма развертывания (deployment).
ER-диаграмма.
Скриншоты разработанного программного продукта

Руководитель ст. преподаватель _____ /Эгов Е.Н./
должность подпись инициалы, фамилия

«_____»_____2020 г.

Студент _____ /Фамилия И.О./
подпись инициалы, фамилия

«_____»_____2020 г.

ПРИЛОЖЕНИЕ 3

Отзыв руководителя

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

ОТЗЫВ
руководителя на курсовую работу

студента Фамилия Имя Отчество
фамилия, имя и отчество
Факультет ИСТ группа ИСЭбд-21 курс второй
Дисциплина «Разработка профессиональных приложений»
Тема работы указать свою тему по приказу

Отмечаются следующие моменты: актуальность темы исследования; соответствие содержания и структуры курсовой работы ее теме; степень разработанности проблемы, наиболее интересно исследованные вопросы. Оценивается степень самостоятельности и инициативы студента; умение пользоваться различными источниками информации; уровень его теоретической подготовки; умение анализировать научные материалы, делать практические выводы; знание основных концепций, научной и специальной литературы по избранной теме. Содержится оценка проекта (работы) руководителем.

Руководитель ст. преподаватель _____ /Эгов Е.Н./
должность, учёная степень, ученое звание подпись инициалы, фамилия

«_____»_____2020 г.