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
«Объектно-ориентированное программирование»
для студентов направлений подготовки бакалавриата
09.03.03 «Прикладная информатика»,
09.03.04 «Программная инженерия»

Ульяновск

УлГТУ

2024

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. 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 работу и 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 создать тестовое desktop-приложение по отслеживанию перемещения объекта по координатам. Необходимо реализовать следующие требования:*
    a. *Создание объекта с определенными характеристиками по нажатию кнопки. В характеристиках должны быть прописаны: скорость, вес, 2 поля-цвета, 2-3 поля-признака. Для тестового приложения характеристики можно задавать случайным образом.*
    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.*
    в. *Прорисовку объекта на форме при каждом действии (создание или перемещение). Объект отображать двумерно (вид сверху или сбоку). Координаты расположения объекта считать **левой верхней** точкой области его прорисовки. Некоторые элементы объекта должны прорисовываться опционально, в зависимости от значения характеристик объекта (полей-признаков). При прорисовке использовать 2 поля-цвета: один для прорисовки основных элементов, второй – для прорисовки опциональных*

элементов. Размеры объекта не должны превышать 150 на 150.

Проектирование

В качестве примера будет разработано desktop-приложение для работы с объектом-гоночным автомобилем.

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, будем называть его класс-сущность. Выделим следующие характеристики: скорость, вес (будут влиять на расстояние перемещения), основной и дополнительный цвета (будут использоваться при прорисовке), признак наличия обвесов, признак наличия антикрыла, признак наличия гоночной полосы (опциональные признаки прорисовки определенных элементов). Второй класс будет отвечать непосредственно за прорисовку сущности на форме, его перемещения по ней, будем называть этот класс класс-прорисовка. Класс-прорисовка, в частности, будет в себя включать объект класса-сущности. Также класс будет хранить информацию по координатам объекта, его размерам и границам поля (эти данные будут использоваться для проверки, что объект не «уходит» за границы). Также второй класс будет отвечать за прорисовку объекта на поле, его перемещение по нему (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-сущность. Описанные параметры зададим как свойства (напоминаю, свойство можно представить как закрытое поле и два публичных метода для работы с этим полем: для сохранения в поле значения и для получения из поля значения), 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).

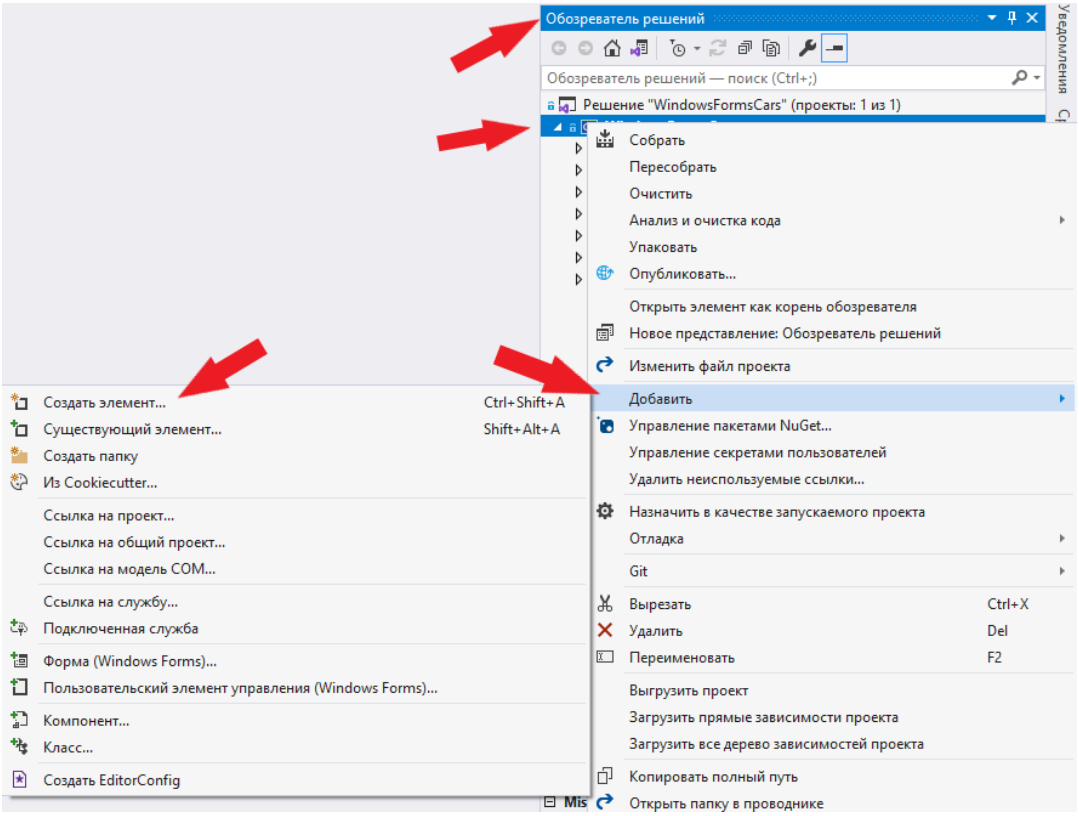


Рисунок 1.1 – Добавление элемента в проект

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).

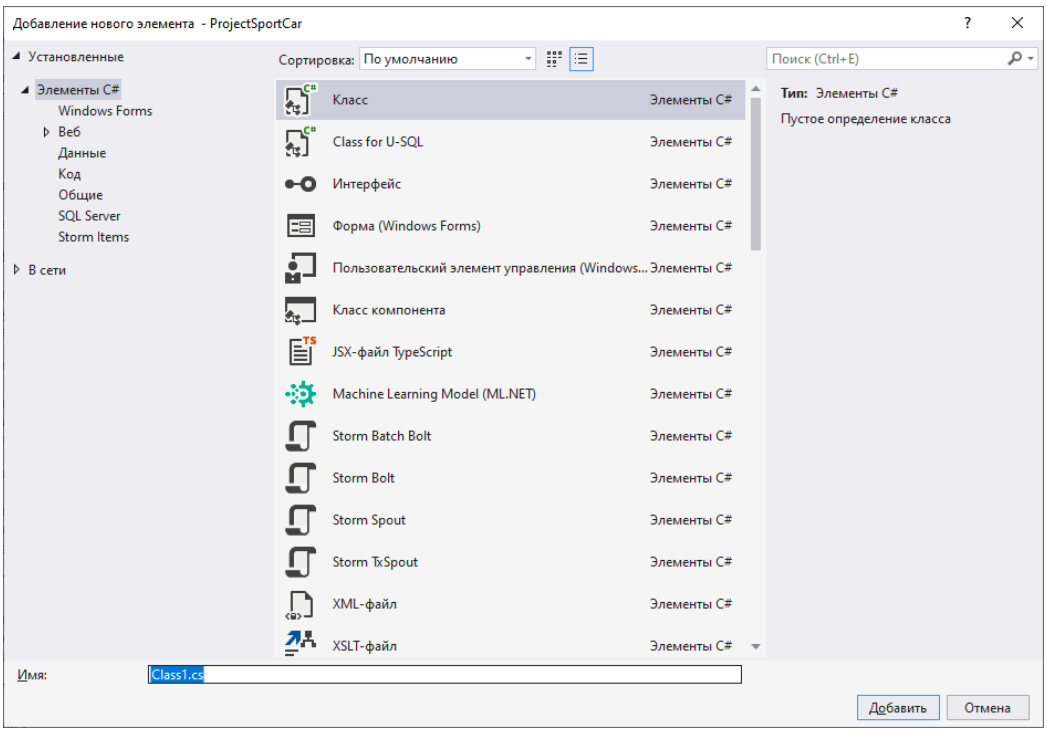


Рисунок 1.2 – Создание класса

Остается наполнить класс логикой (листинг 1.1).

```
namespace ProjectSportCar;

/// <summary>
/// Класс-сущность "Спортивный автомобиль"
/// </summary>
public class EntitySportCar
{
        /// <summary>
        /// Скорость
        /// </summary>
        public int Speed { get; private set; }

        /// <summary>
        /// Вес
        /// </summary>
        public double Weight { get; private set; }

        /// <summary>
        /// Основной цвет
        /// </summary>
        public Color BodyColor { get; private set; }

        /// <summary>
        /// Дополнительный цвет (для опциональных элементов)
        /// </summary>
        public Color AdditionalColor { get; private set; }

        /// <summary>
        /// Признак (опция) наличия обвеса
        /// </summary>
        public bool BodyKit { get; private set; }

        /// <summary>
```

```
        /// Признак (опция) наличия антикрыла
        /// </summary>
        public bool Wing { get; private set; }

        /// <summary>
        /// Признак (опция) наличия гоночной полосы
        /// </summary>
        public bool SportLine { get; private set; }

        /// <summary>
        /// Шаг перемещения автомобиля
        /// </summary>
        public double Step => Speed * 100 / Weight;

        /// <summary>
        /// Инициализация полей объекта-класса спортивного автомобиля
        /// </summary>
        /// <param name="speed">Скорость</param>
        /// <param name="weight">Вес автомобиля</param>
        /// <param name="bodyColor">Основной цвет</param>
        /// <param name="additionalColor">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 void Init(int speed, double weight, Color bodyColor, Color
additionalColor, bool bodyKit, bool wing, bool sportLine)
        {
                Speed = speed;
                Weight = weight;
                BodyColor = bodyColor;
                AdditionalColor = additionalColor;
                BodyKit = bodyKit;
                Wing = wing;
                SportLine = sportLine;
        }
}
```

Листинг 1.1 – Класс-сущность объекта «Спортивный 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 «DirectionType» и создаем класс (рисунок 1.3). Далее просто ключевое слово «class» меняем на «enum» и прописываем значения для него (листинга 1.2).

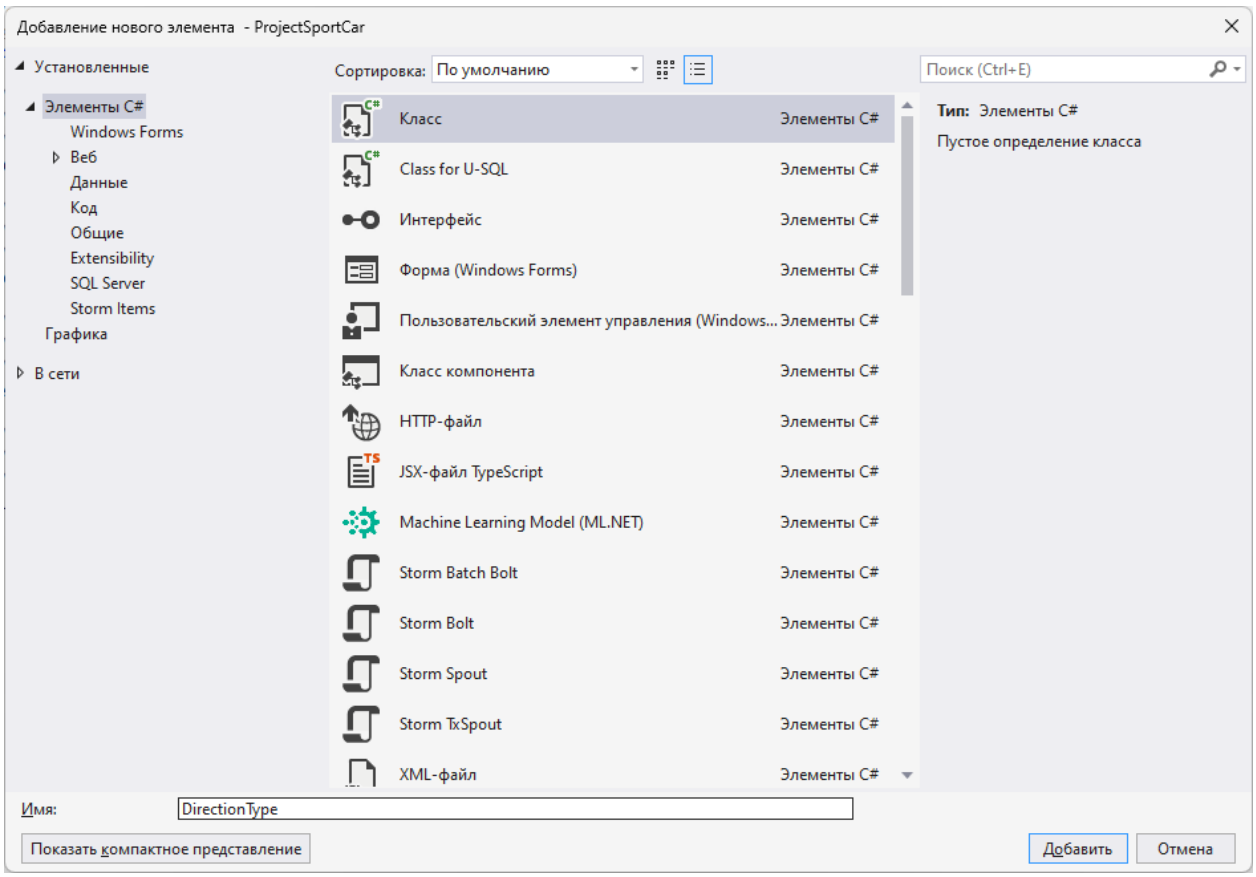


Рисунок 1.3 – Создание класса DirectionType

```
namespace ProjectSportCar;

/// <summary>
/// Направление перемещения
/// </summary>
public enum DirectionType
{
        /// <summary>
        /// Вверх
        /// </summary>
        Up = 1,

        /// <summary>
        /// Вниз
        /// </summary>
        Down = 2,

        /// <summary>
        /// Влево
        /// </summary>
        Left = 3,

        /// <summary>
        /// Вправо
        /// </summary>
        Right = 4
}
```

Листинг 1.2 – Перечисление направлений движения

Перейдем к классу-прорисовки. Назовем класс DrawningSportCar. Исходя из задания, сразу можно выделить 2 метода: для прорисовки объекта и для перемещения объекта. Дополнительно потребуется метод для заполнения параметров объекта (скорость, вес, цвета и прочее) и методы для указания размеров поля для прорисовки и установки позиции.

Рассмотрим детально методы:

- Инициализация свойств. Принимает параметры (сколько свойств мы описали) для заполнения свойств. По результату работы метода создается объект от класса-сущности.
- Передача размеров поля для прорисовки. Метод будет принимать 2 параметра – размеры поля, на котором будет рисоваться объект. Метод будет возвращать true, если объект можно вписать в размеры поля и false – если нельзя. Если размеры поля позволяют вписать в него объект, то запоминаем новые размеры и, если у нас уже имеются координаты размещения объекта, то возможно потребуется их корректировка, если в новых размерах объект начинает выходить за границы.
- Установка начальной позиции. Метод будет принимать 2 параметра – координаты левой верхней точки, от которой будет идти прорисовка объекта. Метод ничего не будет возвращать. Метод будет сохранять полученные параметры в полях класса «координаты прорисовки». Важное условие, что объект, расположенный по этим координатам, не выходит за границы поля, иначе меняем их на такие, чтобы объект был «внутри» поля.
- Перемещение объекта. Метод будет принимать 1 параметр – направление перемещения. Метод будет возвращать true, если перемещение удалось и false в противном случае. На первом шаге будет выполняться проверка, что у нас есть объект (а значит операция Init прошла успешно) и координаты размещения

объекта. Если объект или одна из координат равна null, то перемещение выполняться не будет. Далее, в зависимости от значения параметра, а также «шага» объекта, изменять значение у одного из полей «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 такая же проверка, что у нас есть объект и заданы координаты. Если объект или координаты равны null, то прорисовка выполняться не будет. Метод будет выполнять прорисовку объекта с использованием значений полей «координаты прорисовки», а также свойств объекта-сущности.

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

```
namespace ProjectSportCar;

/// <summary>
/// Класс, отвечающий за прорисовку и перемещение объекта-сущности
/// </summary>
public class DrawningSportCar
{
        /// <summary>
        /// Класс-сущность
        /// </summary>
        public EntitySportCar? EntitySportCar { get; private set; }

        /// <summary>
        /// Ширина окна
        /// </summary>
        private int? _pictureWidth;

        /// <summary>
        /// Высота окна
        /// </summary>
        private int? _pictureHeight;

        /// <summary>
        /// Левая координата прорисовки автомобиля
        /// </summary>
        private int? _startPosX;

        /// <summary>
        /// Верхняя кооридната прорисовки автомобиля
        /// </summary>
```

```
        private int? _startPosY;

        /// <summary>
        /// Ширина прорисовки автомобиля
        /// </summary>
        private readonly int _drawningCarWidth = 110;

        /// <summary>
        /// Высота прорисовки автомобиля
        /// </summary>
        private readonly int _drawningCarHeight = 60;

        /// <summary>
        /// Инициализация свойств
        /// </summary>
        /// <param name="speed">Скорость</param>
        /// <param name="weight">Вес</param>
        /// <param name="bodyColor">Основной цвет</param>
        /// <param name="additionalColor">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 void Init(int speed, double weight, Color bodyColor, Color
additionalColor, bool bodyKit, bool wing, bool sportLine)
        {
                EntitySportCar = new EntitySportCar();
                EntitySportCar.Init(speed, weight, bodyColor, additionalColor,
bodyKit, wing, sportLine);
                _pictureWidth = null;
                _pictureHeight = null;
                _startPosX = null;
                _startPosY = null;
        }

        /// <summary>
        /// Установка границ поля
        /// </summary>
        /// <param name="width">Ширина поля</param>
        /// <param name="height">Высота поля</param>
        /// <returns>true - границы заданы, false - проверка не пройдена, нельзя
разместить объект в этих размерах</returns>
        public bool SetPictureSize(int width, int height)
        {
                // TODO проверка, что объект "влезает" в размеры поля
                // если влезает, сохраняем границы и корректируем позицию объекта,
если она была уже установлена
                _pictureWidth = width;
                _pictureHeight = height;
                return true;
        }

        /// <summary>
        /// Установка позиции
        /// </summary>
        /// <param name="x">Координата X</param>
        /// <param name="y">Координата Y</param>
        public void SetPosition(int x, int y)
        {
                if (!_pictureHeight.HasValue || !_pictureWidth.HasValue)
                {
                        return;
                }
```

```
            // TODO если при установке объекта в эти координаты, он будет
"выходить" за границы формы
            // то надо изменить координаты, чтобы он оставался в этих границах
            _startPosX = x;
            _startPosY = y;
      }

      /// <summary>
      /// Изменение направления перемещения
      /// </summary>
      /// <param name="direction">Направление</param>
      /// <returns>true - перемещене выполнено, false - перемещение
невозможно</returns>
      public bool MoveTransport(DirectionType direction)
      {
            if (EntitySportCar == null || !_startPosX.HasValue ||
!_startPosY.HasValue)
            {
                  return false;
            }

            switch (direction)
            {
                  //влево
                  case DirectionType.Left:
                        if (_startPosX.Value - EntitySportCar.Step > 0)
                        {
                              _startPosX -= (int)EntitySportCar.Step;
                        }
                        return true;
                  //вверх
                  case DirectionType.Up:
                        if (_startPosY.Value - EntitySportCar.Step > 0)
                        {
                              _startPosY -= (int)EntitySportCar.Step;
                        }
                        return true;
                  // вправо
                  case DirectionType.Right:
                        //TODO прописать логику сдвига в право
                        return true;
                  //вниз
                  case DirectionType.Down:
                        //TODO прописать логику сдвига в вниз
                        return true;
                  default:
                        return false;
            }
      }

      /// <summary>
      /// Прорисовка объекта
      /// </summary>
      /// <param name="g"></param>
      public void DrawTransport(Graphics g)
      {
            if (EntitySportCar == null || !_startPosX.HasValue ||
!_startPosY.HasValue)
            {
                  return;
            }

            Pen pen = new(Color.Black);
```

```
            Brush additionalBrush = new
SolidBrush(EntitySportCar.AdditionalColor);

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

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

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

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

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

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

            //границы автомобиля
```

```
            g.DrawEllipse(pen, _startPosX.Value + 10, _startPosY.Value + 5, 20,
20);
            g.DrawEllipse(pen, _startPosX.Value + 10, _startPosY.Value + 35, 20,
20);
            g.DrawEllipse(pen, _startPosX.Value + 80, _startPosY.Value + 5, 20,
20);
            g.DrawEllipse(pen, _startPosX.Value + 80, _startPosY.Value + 35, 20,
20);
            g.DrawRectangle(pen, _startPosX.Value + 9, _startPosY.Value + 15, 10,
30);
            g.DrawRectangle(pen, _startPosX.Value + 90, _startPosY.Value + 15,
10, 30);
            g.DrawRectangle(pen, _startPosX.Value + 20, _startPosY.Value + 4, 70,
52);

            //задние фары
            Brush brRed = new SolidBrush(Color.Red);
            g.FillEllipse(brRed, _startPosX.Value + 10, _startPosY.Value + 5, 20,
20);
            g.FillEllipse(brRed, _startPosX.Value + 10, _startPosY.Value + 35,
20, 20);

            //передние фары
            Brush brYellow = new SolidBrush(Color.Yellow);
            g.FillEllipse(brYellow, _startPosX.Value + 80, _startPosY.Value + 5,
20, 20);
            g.FillEllipse(brYellow, _startPosX.Value + 80, _startPosY.Value + 35,
20, 20);

            //кузов
            Brush br = new SolidBrush(EntitySportCar.BodyColor);
            g.FillRectangle(br, _startPosX.Value + 10, _startPosY.Value + 15, 10,
30);
            g.FillRectangle(br, _startPosX.Value + 90, _startPosY.Value + 15, 10,
30);
            g.FillRectangle(br, _startPosX.Value + 20, _startPosY.Value + 5, 70,
50);

            //стекла
            Brush brBlue = new SolidBrush(Color.LightBlue);
            g.FillRectangle(brBlue, _startPosX.Value + 70, _startPosY.Value + 10,
5, 40);
            g.FillRectangle(brBlue, _startPosX.Value + 30, _startPosY.Value + 10,
5, 40);
            g.FillRectangle(brBlue, _startPosX.Value + 35, _startPosY.Value + 8,
35, 2);
            g.FillRectangle(brBlue, _startPosX.Value + 35, _startPosY.Value + 51,
35, 2);

            //выделяем рамкой крышу
            g.DrawRectangle(pen, _startPosX.Value + 35, _startPosY.Value + 10,
35, 40);
            g.DrawRectangle(pen, _startPosX.Value + 75, _startPosY.Value + 15,
25, 30);
            g.DrawRectangle(pen, _startPosX.Value + 10, _startPosY.Value + 15,
15, 30);

            // спортивная линия
            if (EntitySportCar.SportLine)
            {
                  g.FillRectangle(additionalBrush, _startPosX.Value + 75,
_startPosY.Value + 23, 25, 15);
```

```
                    g.FillRectangle(additionalBrush, _startPosX.Value + 35,
_startPosY.Value + 23, 35, 15);
                    g.FillRectangle(additionalBrush, _startPosX.Value + 10,
_startPosY.Value + 23, 20, 15);
                }

                // крыло
                if (EntitySportCar.Wing)
                {
                    g.FillRectangle(additionalBrush, _startPosX.Value,
_startPosY.Value + 5, 10, 50);
                    g.DrawRectangle(pen, _startPosX.Value, _startPosY.Value + 5,
10, 50);
                }
        }
}
```

Листинг 1.3 – Класс-прорисовка

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 удаляем и создаем новую, согласно заданию. Во вкладке «Обозреватель решения» два раза кликнем по файлу новой формы. Откроется интерфейс формы. Изменим размер формы (либо мышкой за нижний правый угол, либо через свойства найти там «Size») и подпись формы на соответствующее тематике задания (свойство «Text»). Размеры зададим 900 на 500 (необязательно, можно выставить иные).

Опционально. 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, кнопка для создания объекта и кнопки для перемещения объекта. Рассмотрим все элементы отдельно.

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). Перенесем из вкладки «Панель элементов» этот элемент на форму. Далее в свойствах у этого элемента найдем свойство «Dock» и выберем центральный квадратик (рисунок 1.4). И выставляем свойство у «SizeMode» в значение «AutoSize».

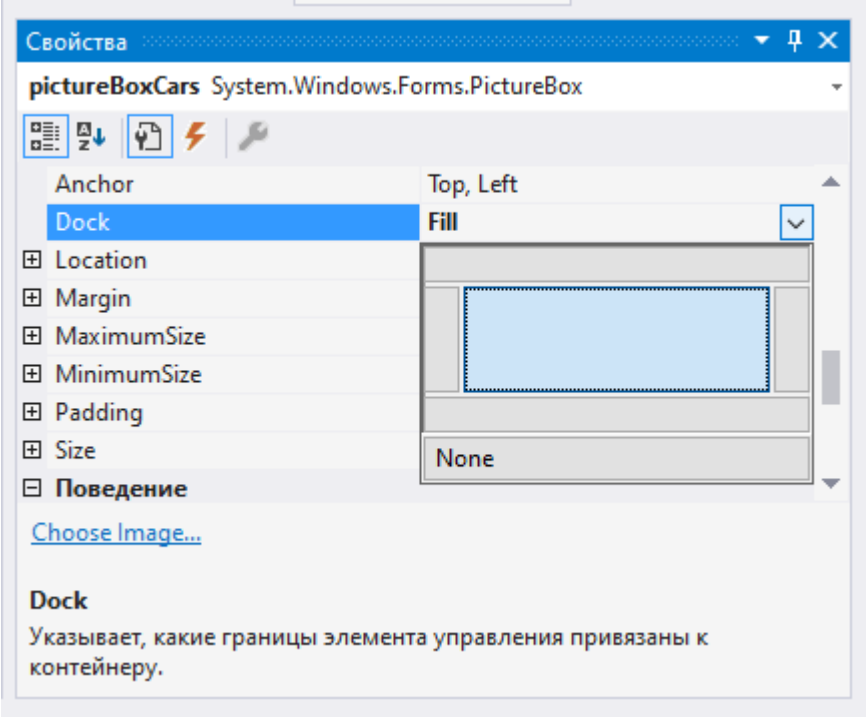


Рисунок 1.4 – Настройка элемента PictureBox

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 1.5).

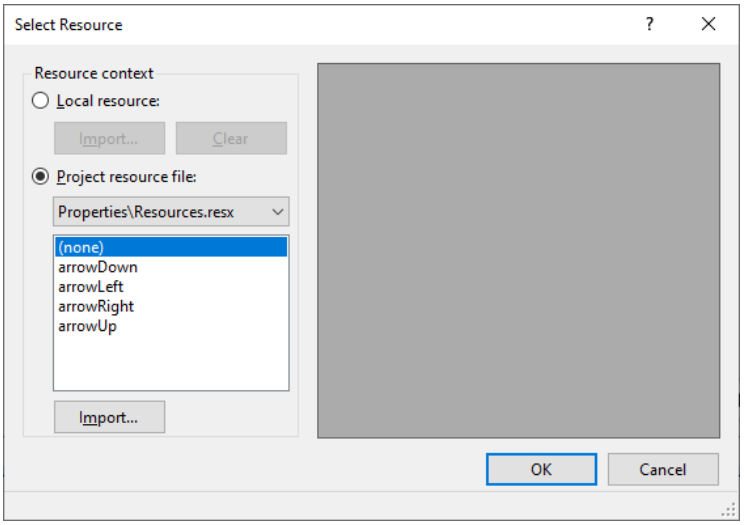


Рисунок 1.5 – Добавление ресурсов

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, для кнопки – это будет как раз нужное нам действие нажатия на кнопку);
- Во вкладке свойства для выбранного элемента перейти на события (значок молния), 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 1.6).

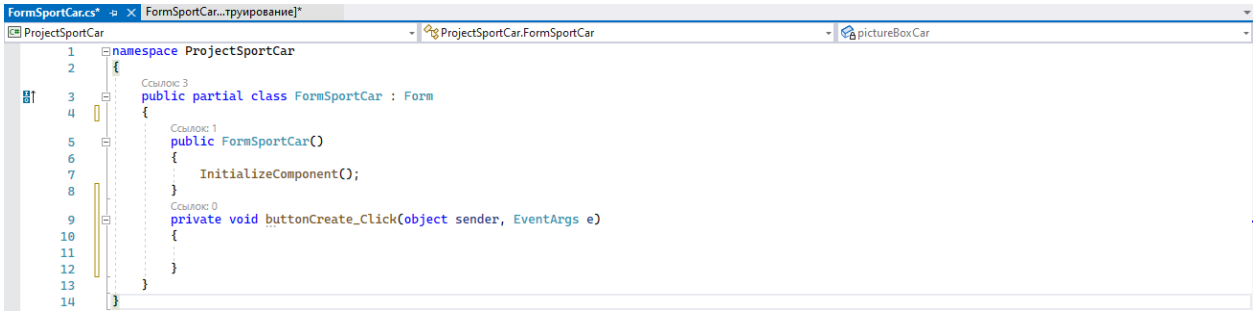


Рисунок 1.6 – Созданный метод обработки нажатия кнопки

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 (sender), который вызвал метод, вторым – информация по событию (e). 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 1.7).

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

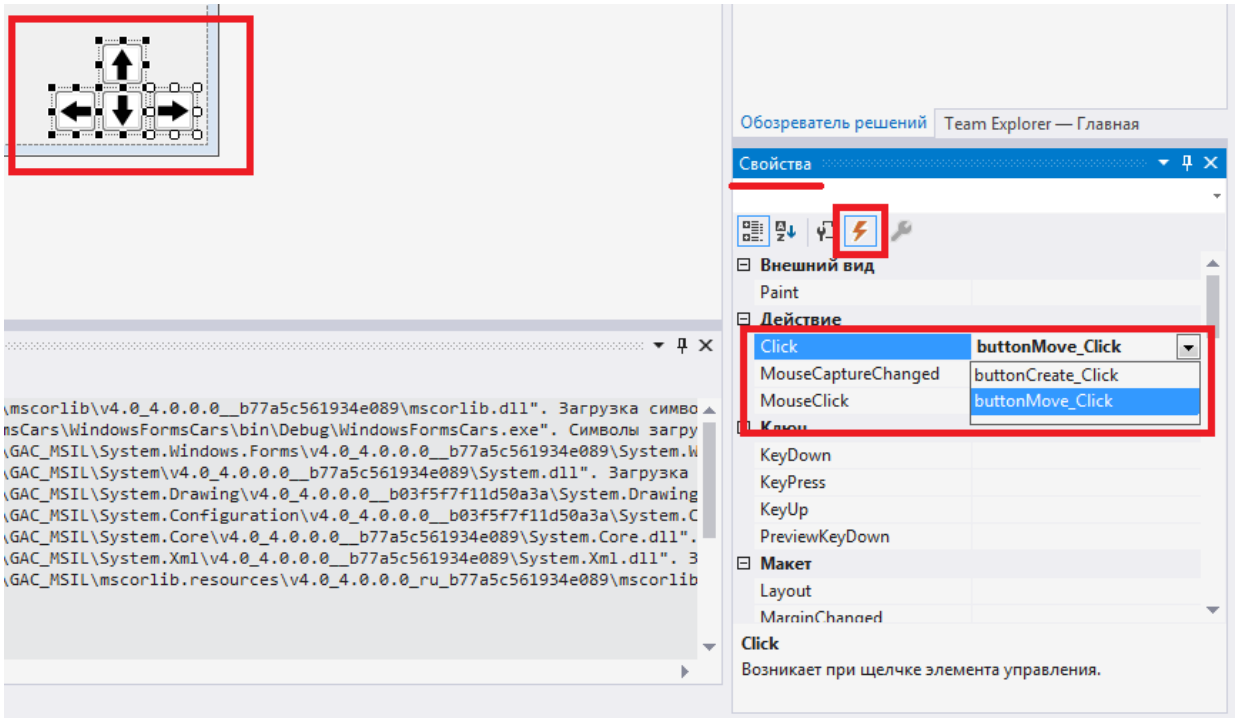


Рисунок 1.7 – Назначение метода обработки для кнопок управления

Замечание! Возможна ситуация, когда требуется удалить метод обработки за ненадобностью и после удаления метода из логики дизайнер формы может выдать такую картину (рисунок 1.8).

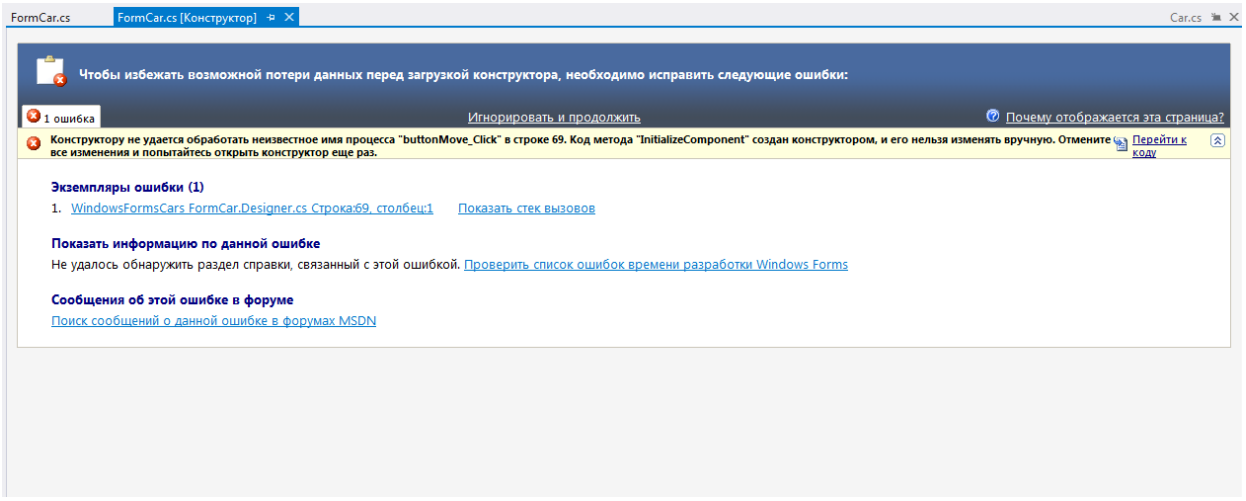


Рисунок 1.8 – Ошибка в дизайнере формы

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 1.9).

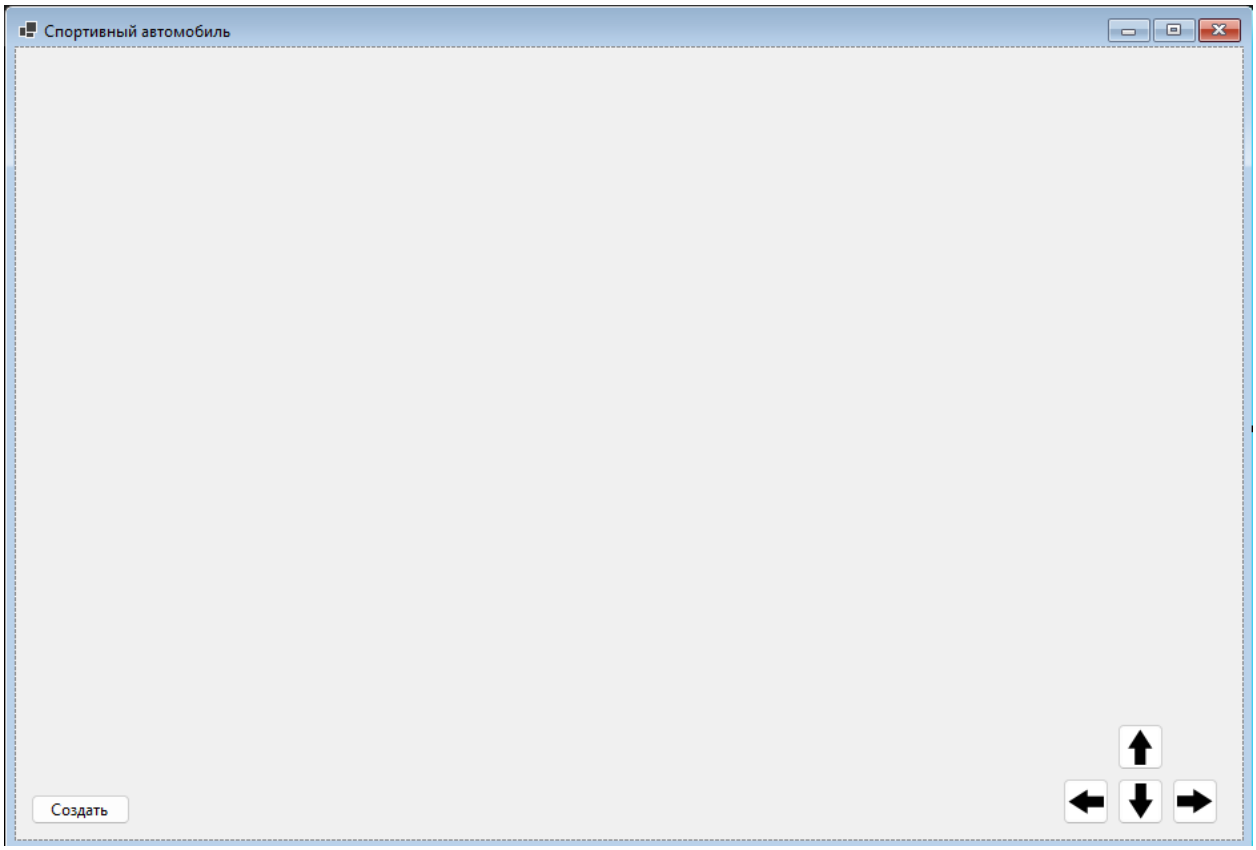


Рисунок 1.9 – Форма для отображения объекта-автомобиля

Теперь логика формы. Для нормальной работы с объектом в разных методах логики формы, вводится поле от класса-прорисовки. Во всех методах

идет работа с этим полем. Задается отдельный метод для прорисовки объекта, так как прорисовка требуется при всех возможных действиях (создание, перемещение). 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-прорисовки для рисования объекта на полотне;
- передать полученный рисунок на PictureBox.

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 методы передачи размера поля прорисовки (это будут размеры PictureBox) и установки изначальной позиции (будем использовать случайные числа так, чтобы объект оказался 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 (листинг 1.4).

```
namespace ProjectSportCar;

/// <summary>
/// Форма работы с объектом "Спортивный автомобиль"
/// </summary>
public partial class FormSportCar : Form
{
        /// <summary>
        /// Поле-объект для прорисовки объекта
        /// </summary>
        private DrawningSportCar? _drawningSportCar;

        /// <summary>
        /// Конструктор формы
        /// </summary>
        public FormSportCar()
        {
```

```
            InitializeComponent();
      }

      /// <summary>
      /// Метод прорисовки машины
      /// </summary>
      private void Draw()
      {
            if (_drawningSportCar == null)
            {
                  return;
            }

            Bitmap bmp = new(pictureBoxSportCar.Width,
pictureBoxSportCar.Height);
            Graphics gr = Graphics.FromImage(bmp);
            _drawningSportCar.DrawTransport(gr);
            pictureBoxSportCar.Image = bmp;
      }

      /// <summary>
      /// Обработка нажатия кнопки "Создать"
      /// </summary>
      /// <param name="sender"></param>
      /// <param name="e"></param>
      private void ButtonCreateSportCar_Click(object sender, EventArgs e)
      {
            Random random = new();
            _drawningSportCar = new DrawningSportCar();
            _drawningSportCar.Init(random.Next(100, 300), random.Next(1000,
3000),
                  Color.FromArgb(random.Next(0, 256), random.Next(0, 256),
random.Next(0, 256)),
                  Color.FromArgb(random.Next(0, 256), random.Next(0, 256),
random.Next(0, 256)),
                  Convert.ToBoolean(random.Next(0, 2)),
Convert.ToBoolean(random.Next(0, 2)), Convert.ToBoolean(random.Next(0, 2)));
            _drawningSportCar.SetPictureSize(pictureBoxSportCar.Width,
pictureBoxSportCar.Height);
            _drawningSportCar.SetPosition(random.Next(10, 100), random.Next(10,
100));
            Draw();
      }

      /// <summary>
      /// Перемещение объекта по форме (нажатие кнопок навигации)
      /// </summary>
      /// <param name="sender"></param>
      /// <param name="e"></param>
      private void ButtonMove_Click(object sender, EventArgs e)
      {
            if (_drawningSportCar == null)
            {
                  return;
            }

            string name = ((Button)sender)?.Name ?? string.Empty;
            bool result = false;
            switch (name)
            {
                  case "buttonUp":
                        result =
_drawningSportCar.MoveTransport(DirectionType.Up);
```

```
                        break;
                    case "buttonDown":
                        result =
_drawningSportCar.MoveTransport(DirectionType.Down);
                        break;
                    case "buttonLeft":
                        result =
_drawningSportCar.MoveTransport(DirectionType.Left);
                        break;
                    case "buttonRight":
                        result =
_drawningSportCar.MoveTransport(DirectionType.Right);
                        break;
            }

            if (result)
            {
                Draw();
            }
        }
}
```

Листинг 1.4 – Логика формы для работы с объектом

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 FormSportCar());) то их следует прописывать в методе Main.

```
namespace ProjectSportCar
{
    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 FormSportCar());
        }
    }
}
```

Листинг 1.5 – Код класса Program.cs

Тестирование

Запустим проект и нажмем кнопку «Создать». На форме должен появиться объект (рисунок 1.10).

Рисунок 1.10 – Создание объекта

Далее нажмем кнопку «Создать» еще раз, чтобы создался другой объект с другими цветами и набором опций (рисунок 1.11).

Рисунок 1.11 – Создание другого объекта

Далее будем наживать кнопку-стрелку влево, пока объект не упрется в левый край формы (рисунок 1.12).

Рисунок 1.12 – Перемещение объекта к левому краю

Далее будем наживать кнопку-стрелку вверх, пока объект не упрется в верхний край формы (рисунок 1.13).

Рисунок 1.13 – Перемещение объекта к верхнему краю

Затем будем наживать кнопку-стрелку вправо, пока объект не упрется в правый край формы (рисунок 1.14).

Рисунок 1.14 – Перемещение объекта к правому краю

В данном случае такое больше расстояние до края обуславливается тем, что не все признаки прорисованы, в частности, обвесы. С обвесами объект был бы шире и тогда бы он казался ближе к правому краю.

Чтобы проверить правильность работы по нижней границе сперва переместим объект чуть левее, чтобы он под кнопки не «заезжал» и далее «спустим» его вниз (рисунок 1.15).

Рисунок 1.15 – Перемещение объекта к нижнему краю

Таким образом все поставленные задачи выполнены. Работа готова.

Требования

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

   ==========

4. В качестве основы для отображения объекта по заданию использовать изображения, представленные во вариантах. Прорисовку объекта выстраивать таким образом, чтобы она была **ниже и правее** от координат прорисовки (x и y). Для прорисовки НЕопциональных элементов использовать основной цвет, а для прорисовки опциональных – дополнительный. Размеры объекта не

должны превышать 150 на 150. Объект на форме не должен выходить за границы формы при перемещении.

5. В классе-сущности должно быть минимум 2 свойства-признака (можно выделить исходя из описания объекта в задании).
6. Доделать логику в методе MoveTransport, но имеющуюся логику в MoveTransport не менять! Доделать проверку в методе SetPictureSize (размеры форм должны вмещать объект) и методе SetPosition (если объект выходит за границы формы, то «возвращать» его в границы).

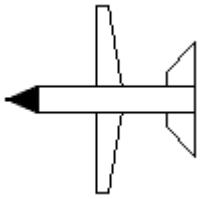
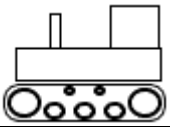
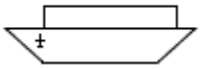
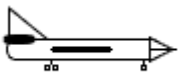
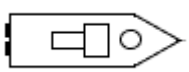
Проверка работы и порядок сдачи

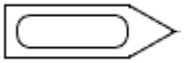
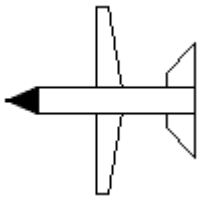
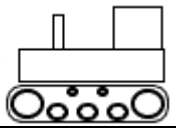
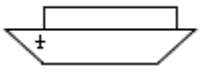
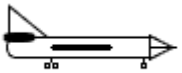
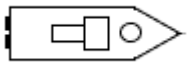
1. Создать объект несколько раз, показать, что опциональные признаки прорисовываются не каждый раз.
2. Довести объект до левой границы поля, показать, что он не выходит за нее.
3. Довести объект до верхней границы поля, показать, что он не выходит за нее.
4. Довести объект до правой границы поля, показать, что он не выходит за нее.
5. Довести объект до нижней границы поля, показать, что он не выходит за нее.

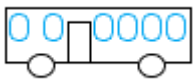
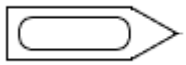
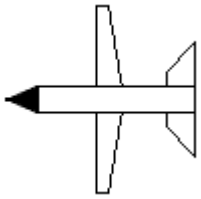
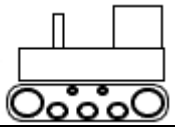
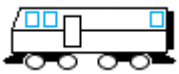
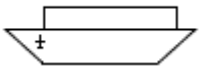
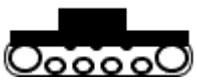
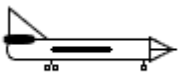
Контрольные вопросы к базовой части

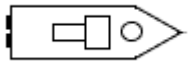
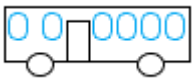
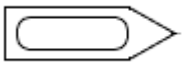
1. Показать объявление и инициализацию объекта от любого класса, созданного в рамках лабораторной работы.
2. Где задается начальное положение объекта на форме?
3. Перечислить элементы класса, которые имеются в классе-сущности.

Варианты

Вар.	Объект	Изображение
1.	Бомбардировщик (с бомбами и дополнительными топливными баками)	
2.	Самосвал (с кузовом и тентом)	
3.	Бульдозер (с отвалом спереди и рыхлителем сзади)	
4.	Электровоз (с «рогами» для подключения к проводам и отсеком под электрические батареи)	
5.	Теплоход (с трубами и отсеком под топливо)	
6.	Танк (с башней с орудием и зенитным пулеметом на башне)	
7.	Аэробус (с дополнительным «отсеком» для пассажиров спереди сверху и с дополнительными двигателями)	
8.	Линкор (с орудийной башней и отсеком под ракеты)	
9.	Автобус двухэтажный (со вторым этажом и лестницей, ведущей на него)	

Вар.	Объект	Изображение
10.	Катер (с мотором в корме и защитным стеклом спереди)	
11.	Истребитель (с ракетами и дополнительными крыльями для лучшей маневренности)	
12.	Бензовоз (с баком под бензин и сигнальным маяком на кабине)	
13.	Экскаватор (с ковшом и опорами для фиксации)	
14.	Тепловоз (с трубой и отсеком под топливо)	
15.	Контейнеровоз (с контейнерами и краном для разгрузки)	
16.	Самоходная арт. установка (с башней с орудием и залповой батарей сзади)	
17.	Самолет с радаром (с радаром и дополнительными топливными баками)	
18.	Крейсер (с ракетными шахтами и площадкой под вертолет)	

Вар.	Объект	Изображение
19.	Автобус с гармошкой (с дополнительным отсеком, соединенным гармошкой и отдельным входом для него)	
20.	Катамаран (с поплавками слева и справа от основного корпуса и парусом)	
21.	Штурмовик (с ракетами и бомбами)	
22.	Подметально-уборочная машина (с баком под воду и подметательной щеткой)	
23.	Подъемный кран (с краном и противовесом к нему)	
24.	Монорельс (с магнитной рельсой и второй кабиной в задней части)	
25.	Лайнер (с дополнительной палубой и бассейном)	
26.	Зенитная установка (с башней с зенитными орудиями и радаром)	
27.	Гидросамолет (с поплавками и надувной лодкой)	

Вар.	Объект	Изображение
28.	Авианосец (с палубой для взлета самолетов и рубкой управления)	
29.	Троллейбус (с «рогами» для подключения к проводам и отсеком под электрические батареи)	
30.	Парусник (с парусом и усиленным корпусом)	

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:*
    a. *Реализовать возможность создания «простой» версии объекта, без дополнительных (опциональных) элементов (не просто не выводить, а чтобы вообще не фигурировали поля, используемые для опциональных элементов). Использовать логику из классов, созданных на первом этапе. Инициализацию перенести в конструкторы классов.*
    б. *«Подружить» классы-прорисовки с набором стратегии перемещения. В набор входят: два перечисления (направление перемещения и статус выполнения стратегии), класс, хранящий координаты перемещаемого объекта, интерфейс, описывающий манипуляции с перемещаемым объектом и абстрактный класс, описывающий стратегию перемещения. Для тестирования работы, создать реализации абстрактного класса, отвечающего за перемещение объекта по полю. Одна реализация отвечает за перемещение объекта в центр формы, вторая – в правый нижний угол формы.*
    в. *На имеющейся форме сделать возможным создавать объекты простого и «продвинутого» объекта, а также*

сделать возможным выбирать способа перемещения и пошаговое выполнение стратегии.

Проектирование

Первым делом следует создать «простой» объект. У нас 2 класса, один отвечает за данные (класс-сущность), второй – за прорисовку (класс-прорисовка). Для каждого класса создадим родительский. Так как «простой» объект – это объект без всевозможных дополнительных элементов, то в родительский класс-сущность следует перенести все общие признаки: вес, скорость, основной цвет. В исходном классе-сущности останутся только поля-признаки и поле с дополнительным цветом, так как оно используется только в прорисовке дополнительных элементов. Для класса-прорисовки примерно все также: все общее в родительский класс (логика получения параметров, перемещения, прорисовки). В дочернем классе останется только логика для прорисовки дополнительных элементов (для этого в родительском классе этот метод пометим виртуальным, для переопределения в дочернем классе).

По инициализации также ничего сложного, 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).

Также имеется набор классов для задания различных стратегий перемещения объекта. Как указывалось выше, в набор входят: перечисление направления перемещения (аналогичное, имеющемуся у нас, листинг 2.1), перечисление-статус выполнения стратегии (листинг 2.2), класс, хранящий координаты перемещаемого объекта (листинг 2.3), интерфейс, описывающий

манипуляции с перемещаемым объектом (листинг 2.4) и абстрактный класс, описывающий стратегию перемещения (листинг 2.5).

```
namespace ProjectSportCar.MovementStrategy;

/// <summary>
/// Направление перемещения
/// </summary>
public enum MovementDirection
{
	/// <summary>
	/// Вверх
	/// </summary>
	Up = 1,

	/// <summary>
	/// Вниз
	/// </summary>
	Down = 2,

	/// <summary>
	/// Влево
	/// </summary>
	Left = 3,

	/// <summary>
	/// Вправо
	/// </summary>
	Right = 4
}
```

Листинг 2.1 – Код перечисления направления перемещения

```
namespace ProjectSportCar.MovementStrategy;

/// <summary>
/// Статус выполнения операции перемещения
/// </summary>
public enum StrategyStatus
{
	/// <summary>
	/// Все готово к началу
	/// </summary>
	NotInit,

	/// <summary>
	/// Выполняется
	/// </summary>
	InProgress,

	/// <summary>
	/// Завершено
	/// </summary>
	Finish
}
```

Листинг 2.2 – Код перечисления статуса выполнения стратегии

```
namespace ProjectSportCar.MovementStrategy;

/// <summary>
/// Параметры-координаты объекта
```

```
/// </summary>
public class ObjectParameters
{
	/// <summary>
	/// Координата X
	/// </summary>
	private readonly int _x;

	/// <summary>
	/// Координата Y
	/// </summary>
	private readonly int _y;

	/// <summary>
	/// Ширина объекта
	/// </summary>
	private readonly int _width;

	/// <summary>
	/// Высота объекта
	/// </summary>
	private readonly int _height;

	/// <summary>
	/// Левая граница
	/// </summary>
	public int LeftBorder => _x;

	/// <summary>
	/// Верхняя граница
	/// </summary>
	public int TopBorder => _y;

	/// <summary>
	/// Правая граница
	/// </summary>
	public int RightBorder => _x + _width;

	/// <summary>
	/// Нижняя граница
	/// </summary>
	public int DownBorder => _y + _height;

	/// <summary>
	/// Середина объекта
	/// </summary>
	public int ObjectMiddleHorizontal => _x + _width / 2;

	/// <summary>
	/// Середина объекта
	/// </summary>
	public int ObjectMiddleVertical => _y + _height / 2;

	/// <summary>
	/// Конструктор
	/// </summary>
	/// <param name="x">Координата X</param>
	/// <param name="y">Координата Y</param>
	/// <param name="width">Ширина объекта</param>
	/// <param name="height">Высота объекта</param>
	public ObjectParameters(int x, int y, int width, int height)
	{
		_x = x;
```

```
            _y = y;
            _width = width;
            _height = height;
        }
}
```

Листинг 2.3 – Код класса, хранящего координаты перемещаемого объекта

```
namespace ProjectSportCar.MovementStrategy;

/// <summary>
/// Интерфейс для работы с перемещаемым объектом
/// </summary>
public interface IMoveableObject
{
        /// <summary>
        /// Получение координаты объекта
        /// </summary>
        ObjectParameters? GetObjectPosition { get; }

        /// <summary>
        /// Шаг объекта
        /// </summary>
        int GetStep { get; }

        /// <summary>
        /// Попытка переместить объект в указанном направлении
        /// </summary>
        /// <param name="direction">Направление</param>
        /// <returns>true - объект перемещен, false - перемещение
невозможно</returns>
        bool TryMoveObject(MovementDirection direction);
}
```

Листинг 2.4 – Код интерфейса, описывающего манипуляции с перемещаемым объектом

```
namespace ProjectSportCar.MovementStrategy;

/// <summary>
/// Класс-стратегия перемещения объекта
/// </summary>
public abstract class AbstractStrategy
{
        /// <summary>
        /// Перемещаемый объект
        /// </summary>
        private IMoveableObject? _moveableObject;

        /// <summary>
        /// Статус перемещения
        /// </summary>
        private StrategyStatus _state = StrategyStatus.NotInit;

        /// <summary>
        /// Ширина поля
        /// </summary>
        protected int FieldWidth { get; private set; }

        /// <summary>
```

```
        /// Высота поля
        /// </summary>
        protected int FieldHeight { get; private set; }

        /// <summary>
        /// Статус перемещения
        /// </summary>
        public StrategyStatus GetStatus() { return _state; }

        /// <summary>
        /// Установка данных
        /// </summary>
        /// <param name="moveableObject">Перемещаемый объект</param>
        /// <param name="width">Ширина поля</param>
        /// <param name="height">Высота поля</param>
        public void SetData(IMoveableObject moveableObject, int width, int height)
        {
                if (moveableObject == null)
                {
                        _state = StrategyStatus.NotInit;
                        return;
                }

                _state = StrategyStatus.InProgress;
                _moveableObject = moveableObject;
                FieldWidth = width;
                FieldHeight = height;
        }

        /// <summary>
        /// Шаг перемещения
        /// </summary>
        public void MakeStep()
        {
                if (_state != StrategyStatus.InProgress)
                {
                        return;
                }

                if (IsTargetDestinaion())
                {
                        _state = StrategyStatus.Finish;
                        return;
                }

                MoveToTarget();
        }

        /// <summary>
        /// Перемещение влево
        /// </summary>
        /// <returns>Результат перемещения (true - удалось переместиться, false -
неудача)</returns>
        protected bool MoveLeft() => MoveTo(MovementDirection.Left);

        /// <summary>
        /// Перемещение вправо
        /// </summary>
        /// <returns>Результат перемещения (true - удалось переместиться, false -
неудача)</returns>
        protected bool MoveRight() => MoveTo(MovementDirection.Right);

        /// <summary>
```

```
        /// Перемещение вверх
        /// </summary>
        /// <returns>Результат перемещения (true - удалось переместиться, false -
неудача)</returns>
        protected bool MoveUp() => MoveTo(MovementDirection.Up);

        /// <summary>
        /// Перемещение вниз
        /// </summary>
        /// <returns>Результат перемещения (true - удалось переместиться, false -
неудача)</returns>
        protected bool MoveDown() => MoveTo(MovementDirection.Down);

        /// <summary>
        /// Параметры объекта
        /// </summary>
        protected ObjectParameters? GetObjectParameters =>
_moveableObject?.GetObjectPosition;

        /// <summary>
        /// Шаг объекта
        /// </summary>
        /// <returns></returns>
        protected int? GetStep()
        {
                if (_state != StrategyStatus.InProgress)
                {
                        return null;
                }
                return _moveableObject?.GetStep;
        }

        /// <summary>
        /// Перемещение к цели
        /// </summary>
        protected abstract void MoveToTarget();

        /// <summary>
        /// Достигнута ли цель
        /// </summary>
        /// <returns></returns>
        protected abstract bool IsTargetDestinaion();

        /// <summary>
        /// Попытка перемещения в требуемом направлении
        /// </summary>
        /// <param name="movementDirection">Направление</param>
        /// <returns>Результат попытки (true - удалось переместиться, false -
неудача)</returns>
        private bool MoveTo(MovementDirection movementDirection)
        {
                if (_state != StrategyStatus.InProgress)
                {
                        return false;
                }

                return _moveableObject?.TryMoveObject(movementDirection) ?? false;
        }
}
```

Листинг 2.5 – Код абстрактного класса, описывающего стратегию перемещения

Для внедрения этого набора классов в программу потребуется «подружить» интерфейс для работы с перемещаемым объектом с классом-прорисовки (если «подружим» родительский класс, то дочерний будет «дружить по умолчанию»). Есть 3 варианта:

1. В классе-прорисовке переименовать методы под объявленные в интерфейсы.
2. Сделать в классе-прорисовке 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.

Минус первого варианта в том, что требуется везде менять уже существующие вызовы методов класса, что противоречит принципам разработки (принципы SOLID). Второй вариант породит кучу однотипных методов в классе, что усложнит, запутает его использование. Третий вариант является самым правильным, он позволит, с одной стороны, использовать класс прорисовки в работе набора, а с другой – позволит его и дальше использовать независимо от набора, отвечающего за перемещение.

Перейдем к форме. Для возможности создания на форме объекта «простого» варианта потребуется добавить кнопку. Для работы со стратегиями перемещения потребуется добавить выпадающий список, для выбора стратегии и кнопку для выполнения шага стратегии.

Дополнительно. Так как классов и иных элементов проекта становится много, разнес все по папкам: классы-сущности будем хранить в папке Entities, классы-прорисовки и перечисление DirectionType в папке Drawnings. Набор стратегии перемещения в папке MovementStrategy.

Реализация

Первым делом введем базовые классы. Для класса-сущности все просто (листинг 2.6).

```
namespace ProjectSportCar.Entities;

/// <summary>
/// Класс-сущность "Автомобиль"
/// </summary>
public class EntityCar
{
	/// <summary>
	/// Скорость
	/// </summary>
	public int Speed { get; private set; }

	/// <summary>
	/// Вес
	/// </summary>
	public double Weight { get; private set; }

	/// <summary>
	/// Основной цвет
	/// </summary>
	public Color BodyColor { get; private set; }

	/// <summary>
	/// Шаг перемещения автомобиля
	/// </summary>
	public double Step => Speed * 100 / Weight;

	/// <summary>
	/// Конструктор сущности
	/// </summary>
	/// <param name="speed">Скорость</param>
	/// <param name="weight">Вес автомобиля</param>
	/// <param name="bodyColor">Основной цвет</param>
	public EntityCar(int speed, double weight, Color bodyColor)
	{
		Speed = speed;
		Weight = weight;
		BodyColor = bodyColor;
	}
}
```

Листинг 2.6 – Класс базовой сущности с конструктором

С базовым классом прорисовки чуть посложнее, так как требуется перенести не только поля, но и методы, а один сделать виртуальным и оставить в нем только часть логики, которая была ранее. Также размеры «продвинутого» объекта больше размеров «простого» объекта и это надо тоже учитывать. Изменим логику прорисовки с учетом, что нам не требуется прорисовывать опциональные элементы, так что элементы прорисовки все

чуть-чуть сдвигаются. Но возникает проблема: размеры задаются константами и могут меняться только в конструкторе, и их изменения невозможны в классе-наследнике. Поэтому потребуется сделать еще один конструктор, для изменения константных параметров. Конструктор сделаем доступным только классам-наследникам. А чтобы не дублировать логику задания определенных параметров, сделаем еще и приватный конструктор, который будем вызывать при вызове публичного конструктора и конструктора для классов-наследников через ключевое слово this. (листинг 2.7).

```
using ProjectSportCar.Entities;

namespace ProjectSportCar.Drawnings;

/// <summary>
/// Класс, отвечающий за прорисовку и перемещение базового объекта-сущности
/// </summary>
public class DrawningCar
{
        /// <summary>
        /// Класс-сущность
        /// </summary>
        public EntityCar? EntityCar { get; protected set; }

        /// <summary>
        /// Ширина окна
        /// </summary>
        private int? _pictureWidth;

        /// <summary>
        /// Высота окна
        /// </summary>
        private int? _pictureHeight;

        /// <summary>
        /// Левая координата прорисовки автомобиля
        /// </summary>
        protected int? _startPosX;

        /// <summary>
        /// Верхняя кооридната прорисовки автомобиля
        /// </summary>
        protected int? _startPosY;

        /// <summary>
        /// Ширина прорисовки автомобиля
        /// </summary>
        private readonly int _drawningCarWidth = 90;

        /// <summary>
        /// Высота прорисовки автомобиля
        /// </summary>
        private readonly int _drawningCarHeight = 50;

        /// <summary>
```

```
        /// Пустой конструктор
        /// </summary>
        private DrawningCar()
        {
                _pictureWidth = null;
                _pictureHeight = null;
                _startPosX = null;
                _startPosY = null;
        }

        /// <summary>
        /// Конструктор
        /// </summary>
        /// <param name="speed">Скорость</param>
        /// <param name="weight">Вес</param>
        /// <param name="bodyColor">Основной цвет</param>
        public DrawningCar(int speed, double weight, Color bodyColor) : this()
        {
                EntityCar = new EntityCar(speed, weight, bodyColor);
        }

        /// <summary>
        /// Конструктор для наследников
        /// </summary>
        /// <param name="drawningCarWidth">Ширина прорисовки автомобиля</param>
        /// <param name="drawningCarHeight">Высота прорисовки автомобиля</param>
        protected DrawningCar(int drawningCarWidth, int drawningCarHeight) : this()
        {
                _drawningCarWidth = drawningCarWidth;
                _pictureHeight = drawningCarHeight;
        }

        /// <summary>
        /// Установка границ поля
        /// </summary>
        /// <param name="width">Ширина поля</param>
        /// <param name="height">Высота поля</param>
        /// <returns>true - границы заданы, false - проверка не пройдена, нельзя
разместить объект в этих размерах</returns>
        public bool SetPictureSize(int width, int height)
        {
                // TODO проверка, что объект "влезает" в размеры поля
                // если влезает, сохраняем границы и корректируем позицию объекта,
если она была уже установлена
                _pictureWidth = width;
                _pictureHeight = height;
                return true;
        }

        /// <summary>
        /// Установка позиции
        /// </summary>
        /// <param name="x">Координата X</param>
        /// <param name="y">Координата Y</param>
        public void SetPosition(int x, int y)
        {
                if (!_pictureHeight.HasValue || !_pictureWidth.HasValue)
                {
                        return;
                }

                // TODO если при установке объекта в эти координаты, он будет
"выходить" за границы формы
```

```
            // то надо изменить координаты, чтобы он оставался в этих границах
            _startPosX = x;
            _startPosY = y;
      }

      /// <summary>
      /// Изменение направления перемещения
      /// </summary>
      /// <param name="direction">Направление</param>
      /// <returns>true - перемещене выполнено, false - перемещение
невозможно</returns>
      public bool MoveTransport(DirectionType direction)
      {
            if (EntityCar == null || !_startPosX.HasValue ||
!_startPosY.HasValue)
            {
                  return false;
            }

            switch (direction)
            {
                  //влево
                  case DirectionType.Left:
                        if (_startPosX.Value - EntityCar.Step > 0)
                        {
                              _startPosX -= (int)EntityCar.Step;
                        }
                        return true;
                  //вверх
                  case DirectionType.Up:
                        if (_startPosY.Value - EntityCar.Step > 0)
                        {
                              _startPosY -= (int)EntityCar.Step;
                        }
                        return true;
                  // вправо
                  case DirectionType.Right:
                        //TODO прописать логику сдвига в право
                        return true;
                  //вниз
                  case DirectionType.Down:
                        //TODO прописать логику сдвига в вниз
                        return true;
                  default:
                        return false;
            }
      }

      /// <summary>
      /// Прорисовка объекта
      /// </summary>
      /// <param name="g"></param>
      public virtual void DrawTransport(Graphics g)
      {
            if (EntityCar == null || !_startPosX.HasValue ||
!_startPosY.HasValue)
            {
                  return;
            }

            Pen pen = new(Color.Black);

            //границы автомобиля
```

```
            g.DrawEllipse(pen, _startPosX.Value, _startPosY.Value, 20, 20);
            g.DrawEllipse(pen, _startPosX.Value, _startPosY.Value + 30, 20, 20);
            g.DrawEllipse(pen, _startPosX.Value + 70, _startPosY.Value, 20, 20);
            g.DrawEllipse(pen, _startPosX.Value + 70, _startPosY.Value + 30, 20,
20);
            g.DrawRectangle(pen, _startPosX.Value - 1, _startPosY.Value + 10, 10,
30);
            g.DrawRectangle(pen, _startPosX.Value + 80, _startPosY.Value + 10,
10, 30);
            g.DrawRectangle(pen, _startPosX.Value + 10, _startPosY.Value - 1, 70,
52);

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

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

            //кузов
            Brush br = new SolidBrush(EntityCar.BodyColor);
            g.FillRectangle(br, _startPosX.Value, _startPosY.Value + 10, 10, 30);
            g.FillRectangle(br, _startPosX.Value + 80, _startPosY.Value + 10, 10,
30);
            g.FillRectangle(br, _startPosX.Value + 10, _startPosY.Value, 70, 50);

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

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

Листинг 2.7 – Класс прорисовки базовой сущности

Далее перейдем к «продвинутому» объекту. Унаследуем классы от новых, уберем дублирующие поля и методы и переопределим метод прорисовки. Правильный подход перегрузки методов подразумевает использование исходного метода. Но с методом прорисовки проблема, мы же

«сдвинули» «простой» объект чуть выше и левее (на 10 единиц влево и на 5 вверх) и теперь, если мы просто в дочернем оставим код прорисовки опциональных элементов и сделаем вызов прорисовки базовой части, то у нас будет наложение фигур. Для избежания этого сделаем хитрость, перед вызовом базового метода прорисовки изменим значения координат, а после выполнения базового метода, вернем их как было(2.8).

```
using ProjectSportCar.Entities;

namespace ProjectSportCar.Drawnings;

/// <summary>
/// Класс, отвечающий за прорисовку и перемещение объекта-сущности
/// </summary>
public class DrawningSportCar : DrawningCar
{
    /// <summary>
    /// Конструктор
    /// </summary>
    /// <param name="speed">Скорость</param>
    /// <param name="weight">Вес</param>
    /// <param name="bodyColor">Основной цвет</param>
    /// <param name="additionalColor">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, double weight, Color bodyColor, Color
additionalColor, bool bodyKit, bool wing, bool sportLine) : base(110, 60)
    {
        EntityCar = new EntitySportCar(speed, weight, bodyColor, additionalColor,
bodyKit, wing, sportLine);
    }

      public override void DrawTransport(Graphics g)
      {
            if (EntityCar == null || EntityCar is not EntitySportCar sportCar ||
!_startPosX.HasValue || !_startPosY.HasValue)
            {
                  return;
            }

            Pen pen = new(Color.Black);
            Brush additionalBrush = new SolidBrush(sportCar.AdditionalColor);

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

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

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

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

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

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

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

            // спортивная линия
            if (sportCar.SportLine)
            {
                    g.FillRectangle(additionalBrush, _startPosX.Value + 75,
_startPosY.Value + 23, 25, 15);
                    g.FillRectangle(additionalBrush, _startPosX.Value + 35,
_startPosY.Value + 23, 35, 15);
                    g.FillRectangle(additionalBrush, _startPosX.Value + 10,
_startPosY.Value + 23, 20, 15);
            }

            // крыло
```

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

Листинг 2.8 – Обновленный класс прорисовки «продвинутого» объекта

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

Перейдем к реализации интерфейса IMoveableObject. Сперва перенесем оба перечисления и класс, хранящий координаты перемещаемого объекта. Тут все просто и известно. Далее 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).

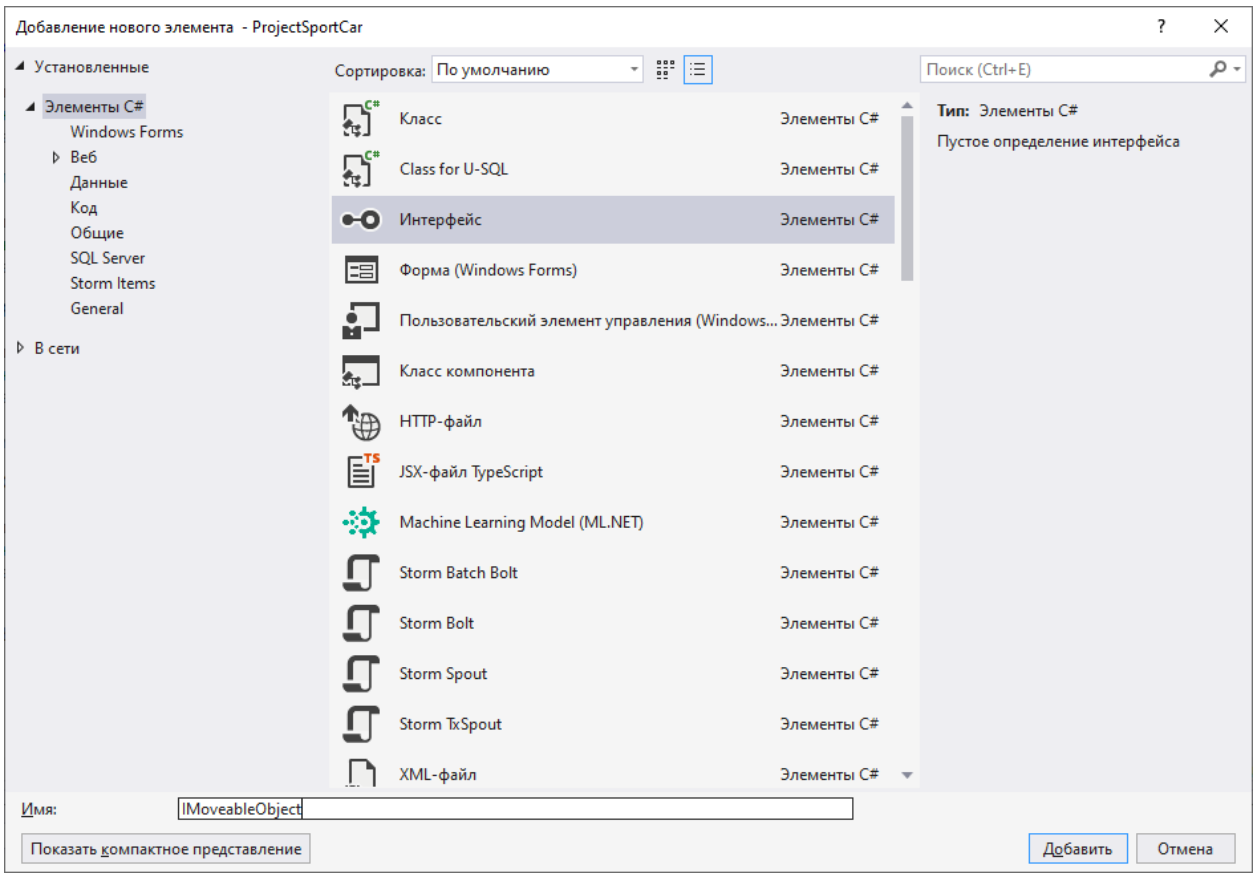


Рисунок 2.1 – Создание интерфейса

Второй вариант, добавить простой класс и в коде заменить «class» на «interface». Код интерфейса представлен был в листинге 2.2.

Создадим класс-наследник. Данный класс будет в реализации использовать публичные методы класса DrawningCar. Для этого класс DrawningCar необходимо доработать методами, которых не хватает для реализации методов интерфейса. В частности – это будут методы (или свойства) для получения координат расположения объекта, а также его длины и ширины (листинг 2.9).

```
using ProjectSportCar.Entities;

namespace ProjectSportCar.Drawnings;

/// <summary>
/// Класс, отвечающий за прорисовку и перемещение объекта-сущности
/// </summary>
public class DrawningCar
{
        …

        /// <summary>
        /// Координата X объекта
```

```
        /// </summary>
        public int? GetPosX => _startPosX;

        /// <summary>
        /// Координата Y объекта
        /// </summary>
        public int? GetPosY => _startPosY;

        /// <summary>
        /// Ширина объекта
        /// </summary>
        public int GetWidth => _drawningCarWidth;

        /// <summary>
        /// Высота объекта
        /// </summary>
        public int GetHeight => _drawningCarHeight;

        …
}
```

Листинг 2.9 – Новые методы класса DrawningCar

Теперь можно приступить к созданию класса-реализации интерфейса IMoveableObject. Класс будет содержать в себе поле от класса DrawningCar (передавать будем через конструктор) и в методах будем обращаться к методом класса DrawningCar (листинг 2.10).

```
using ProjectSportCar.Drawnings;

namespace ProjectSportCar.MovementStrategy;

/// <summary>
/// Класс-реализация IMoveableObject с использованием DrawningCar
/// </summary>
public class MoveableCar : IMoveableObject
{
        /// <summary>
        /// Поле-объект класса DrawningCar или его наследника
        /// </summary>
        private readonly DrawningCar? _car = null;

        /// <summary>
        /// Конструктор
        /// </summary>
        /// <param name="car">Объект класса DrawningCar</param>
        public MoveableCar(DrawningCar car)
        {
                _car = car;
        }

        public ObjectParameters? GetObjectPosition
        {
                get
                {
                        if (_car == null || _car.EntityCar == null ||
!_car.GetPosX.HasValue || !_car.GetPosY.HasValue)
                        {
                                return null;
```

```
                    }
                    return new ObjectParameters(_car.GetPosX.Value,
_car.GetPosY.Value, _car.GetWidth, _car.GetHeight);
                }
        }

        public int GetStep => (int)(_car?.EntityCar?.Step ?? 0);

        public bool TryMoveObject(MovementDirection direction)
        {
                if (_car == null || _car.EntityCar == null)
                {
                        return false;
                }

                return _car.MoveTransport(GetDirectionType(direction));
        }

        /// <summary>
        /// Конвертация из MovementDirection в DirectionType
        /// </summary>
        /// <param name="direction">MovementDirection</param>
        /// <returns>DirectionType</returns>
        private static DirectionType GetDirectionType(MovementDirection direction)
        {
                return direction switch
                {
                        MovementDirection.Left => DirectionType.Left,
                        MovementDirection.Right => DirectionType.Right,
                        MovementDirection.Up => DirectionType.Up,
                        MovementDirection.Down => DirectionType.Down,
                        _ => DirectionType.Unknow,
                };
        }
}
```

Листинг 2.10 – Класс MoveableCar

Нам потребуется преобразовывать значение переменной-перечисления MovementDirection в переменную перечисления DirectionType. Вынесем это в отдельный метод, а для красоты switch оформим в свернутом виде. Правда, для этого потребуется ввести в DirectionType еще одно значение – Unknow (листинг 2.11).

```
namespace ProjectSportCar.Drawnings;

/// <summary>
/// Направление перемещения
/// </summary>
public enum DirectionType
{
        /// <summary>
        /// Неизвестное направление
        /// </summary>
        Unknow = -1,

        /// <summary>
        /// Вверх
```

```
        /// </summary>
        Up = 1,

    /// <summary>
    /// Вниз
    /// </summary>
    Down = 2,

    /// <summary>
    /// Влево
    /// </summary>
    Left = 3,

    /// <summary>
    /// Вправо
    /// </summary>
    Right = 4
}
```

Листинг 2.11 – Перечисление DirectionType с новым значением

Далее перейдем к абстрактному классу. Создается простой класс, у которого добавляется модификатор abstract. Класс является реализацией паттерна поведения – Стратегия. В данном случае, задача класса – через перемещение объекта «привести» его в определенную точку. Имеется ряд методов с логикой, отвечающих за манипуляции с объектом, а также абстрактные методы, отвечающие за продвижение к точке назначения. У каждой реализации абстрактного класса будет своя точка назначения, чем они и будут отличаться, при этом методика (стратегия) достижения этой цели у всех будет одинаковая, так как она прописана в абстрактном классе. Для примера создадим реализацию, где точка назначения будет центром формы. В реализации потребуется переопределить 2 абстрактных метода: метод проверки, что объект в точке назначения, и метод, перемещающий объект к точке назначения. В каждом методе потребуется получать параметры объекта (потребуется проверка, что параметры есть, на всякий пожарный) и либо выполнять проверку, насколько близко объект от центра, либо определять в какую сторону следует двигаться (листинг 2.12).

```
namespace ProjectSportCar.MovementStrategy;

/// <summary>
/// Стратегия перемещения объекта в центр экрана
/// </summary>
public class MoveToCenter : AbstractStrategy
{
```

```
        protected override bool IsTargetDestinaion()
        {
                ObjectParameters? objParams = GetObjectParameters;
                if (objParams == null)
                {
                        return false;
                }

                return objParams.ObjectMiddleHorizontal - GetStep() <= FieldWidth / 2
&& objParams.ObjectMiddleHorizontal + GetStep() >= FieldWidth / 2 &&
                        objParams.ObjectMiddleVertical - GetStep() <= FieldHeight / 2
&& objParams.ObjectMiddleVertical + GetStep() >= FieldHeight / 2;
        }

        protected override void MoveToTarget()
        {
                ObjectParameters? objParams = GetObjectParameters;
                if (objParams == null)
                {
                        return;
                }

                int diffX = objParams.ObjectMiddleHorizontal - FieldWidth / 2;
                if (Math.Abs(diffX) > GetStep())
                {
                        if (diffX > 0)
                        {
                                MoveLeft();
                        }
                        else
                        {
                                MoveRight();
                        }
                }

                int diffY = objParams.ObjectMiddleVertical - FieldHeight / 2;
                if (Math.Abs(diffY) > GetStep())
                {
                        if (diffY > 0)
                        {
                                MoveUp();
                        }
                        else
                        {
                                MoveDown();
                        }
                }
        }
}
```

Листинг 2.12 – Реализация абстрактного класса AbstractStrategy

Последнее, что осталось сделать – внести изменения в форму. Там появились новая кнопка: для создания «простого» объекта, элемент «выпадающий список» для выбора стратегии перемещения и кнопка для выполнения шага перемещения (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 два пункта (одна реализация есть, еще одну нужно будет сделать). 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 «продвинутой» версии на объект по прорисовке «простой» версии. Так как «продвинутая» версия наследуется от простой, то при необходимости не составит труда проинициализировать объект от класса прорисовки «простой» версии классом прорисовки «продвинутой» версии. Также в логику вводим новое поле – объект от абстрактного класса. Работу с этим классом сведем в логику метода кнопки выполнения шага перемещения. Логика будет следующей: если объект от абстрактного класса не проинициализирован, то в зависимости от выбранного значения в выпадающем списке, инициализируем одним из классов-наследников от абстрактного класса. Далее, если объект проинициализирован, делаем шаг, выполняем проверку, закончили ли алгоритм и, если да, обнуляем объект абстрактного класса (листинг 2.13).

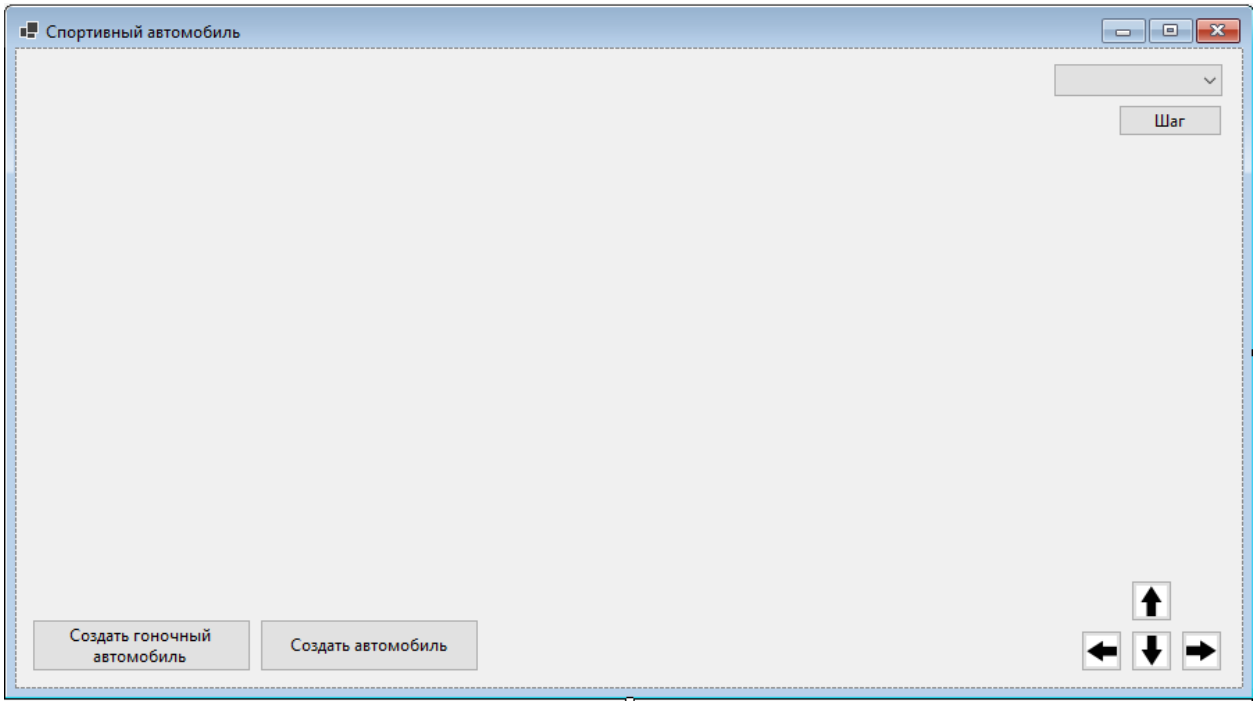


Рисунок 2.2 – Обновленная форма

```
using ProjectSportCar.Drawnings;
using ProjectSportCar.MovementStrategy;

namespace ProjectSportCar;

/// <summary>
/// Форма работы с объектом "Спортивный автомобиль"
/// </summary>
public partial class FormSportCar : Form
{
	/// <summary>
	/// Поле-объект для прорисовки объекта
	/// </summary>
	private DrawningCar? _drawningCar;

	/// <summary>
	/// Стратегия перемещения
	/// </summary>
	private AbstractStrategy? _strategy;

	/// <summary>
	/// Конструктор формы
	/// </summary>
	public FormSportCar()
	{
		InitializeComponent();
		_strategy = null;
	}

	/// <summary>
	/// Метод прорисовки машины
	/// </summary>
	private void Draw()
	{
```

```
            if (_drawningCar == null)
            {
                  return;
            }

            Bitmap bmp = new(pictureBoxSportCar.Width,
pictureBoxSportCar.Height);
            Graphics gr = Graphics.FromImage(bmp);
            _drawningCar.DrawTransport(gr);
            pictureBoxSportCar.Image = bmp;
      }

      /// <summary>
      /// Создание объекта класса-перемещения
      /// </summary>
      /// <param name="type">Тип создаваемого объекта</param>
      private void CreateObject(string type)
      {
            Random random = new();
            switch (type)
            {
                  case nameof(DrawningCar):
                        _drawningCar = new DrawningCar(random.Next(100, 300),
random.Next(1000, 3000),
                              Color.FromArgb(random.Next(0, 256),
random.Next(0, 256), random.Next(0, 256)));
                        break;
                  case nameof(DrawningSportCar):
                        _drawningCar = new DrawningSportCar(random.Next(100,
300), random.Next(1000, 3000),
                              Color.FromArgb(random.Next(0, 256),
random.Next(0, 256), random.Next(0, 256)),
                              Color.FromArgb(random.Next(0, 256),
random.Next(0, 256), random.Next(0, 256)),
                              Convert.ToBoolean(random.Next(0, 2)),
Convert.ToBoolean(random.Next(0, 2)), Convert.ToBoolean(random.Next(0, 2)));
                        break;
                  default:
                        return;
            }

            _drawningCar.SetPictureSize(pictureBoxSportCar.Width,
pictureBoxSportCar.Height);
            _drawningCar.SetPosition(random.Next(10, 100), random.Next(10, 100));
            _strategy = null;
            comboBoxStrategy.Enabled = true;
            Draw();
      }

      /// <summary>
      /// Обработка нажатия кнопки "Создать спортивный автомобиль"
      /// </summary>
      /// <param name="sender"></param>
      /// <param name="e"></param>
      private void ButtonCreateSportCar_Click(object sender, EventArgs e) =>
CreateObject(nameof(DrawningSportCar));

      /// <summary>
      /// Обработка нажатия кнопки "Создать автомобиль"
      /// </summary>
      /// <param name="sender"></param>
      /// <param name="e"></param>
```

```
        private void ButtonCreateCar_Click(object sender, EventArgs e) =>
CreateObject(nameof(DrawningCar));

        /// <summary>
        /// Перемещение объекта по форме (нажатие кнопок навигации)
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void ButtonMove_Click(object sender, EventArgs e)
        {
                if (_drawningCar == null)
                {
                        return;
                }

                string name = ((Button)sender)?.Name ?? string.Empty;
                bool result = false;
                switch (name)
                {
                        case "buttonUp":
                                result = _drawningCar.MoveTransport(DirectionType.Up);
                                break;
                        case "buttonDown":
                                result = _drawningCar.MoveTransport(DirectionType.Down);
                                break;
                        case "buttonLeft":
                                result = _drawningCar.MoveTransport(DirectionType.Left);
                                break;
                        case "buttonRight":
                                result =
_drawningCar.MoveTransport(DirectionType.Right);
                                break;
                }

                if (result)
                {
                        Draw();
                }
        }

        /// <summary>
        /// Обработка нажатия кнопки "Шаг"
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void ButtonStrategyStep_Click(object sender, EventArgs e)
        {
                if (_drawningCar == null)
                {
                        return;
                }

                if (comboBoxStrategy.Enabled)
                {
                        _strategy = comboBoxStrategy.SelectedIndex switch
                        {
                                0 => new MoveToCenter(),
                                1 => new MoveToBorder(),
                                _ => null,
                        };
                        if (_strategy == null)
                        {
                                return;
```

```
                    }
                    _strategy.SetData(new MoveableCar(_drawningCar),
pictureBoxSportCar.Width, pictureBoxSportCar.Height);
                }

                if (_strategy == null)
                {
                        return;
                }

                comboBoxStrategy.Enabled = false;
                _strategy.MakeStep();
                Draw();

                if (_strategy.GetStatus() == StrategyStatus.Finish)
                {
                        comboBoxStrategy.Enabled = true;
                        _strategy = null;
                }
        }
}
```

Листинг 2.13 – Обновленная логика формы

Тестирование

Запустим проект и нажмем кнопку «Создать автомобиль». На форме должен появиться «простой» объект (рисунок 2.3).

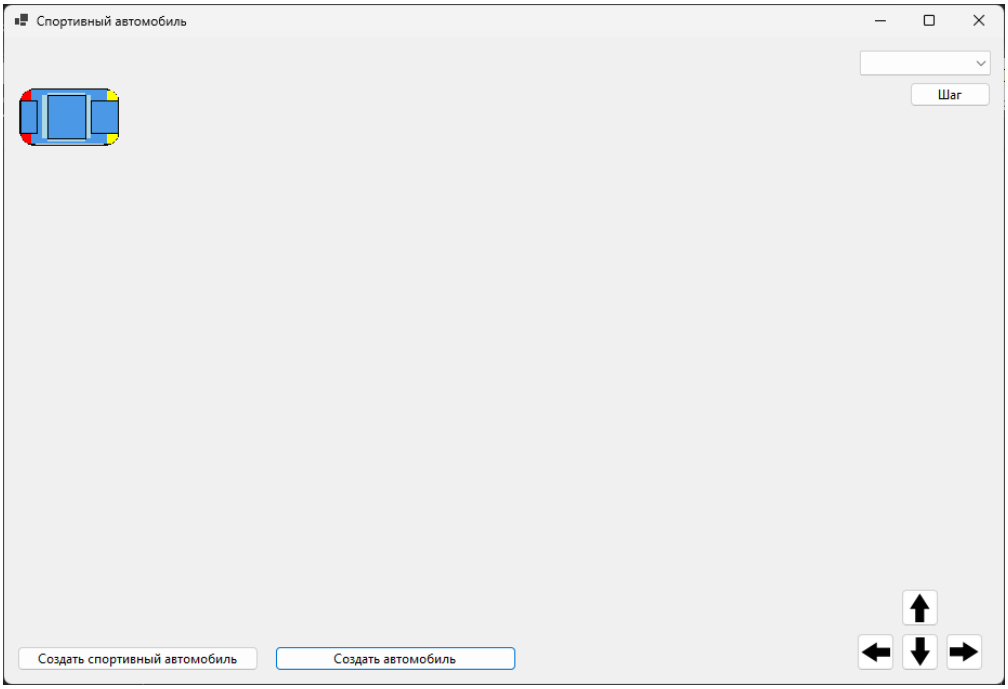


Рисунок 2.3 – Создание «простого» объекта

Далее изменим его положение по обеим осям, чтобы убедиться, что рисунок не «едет» (рисунок 2.4).

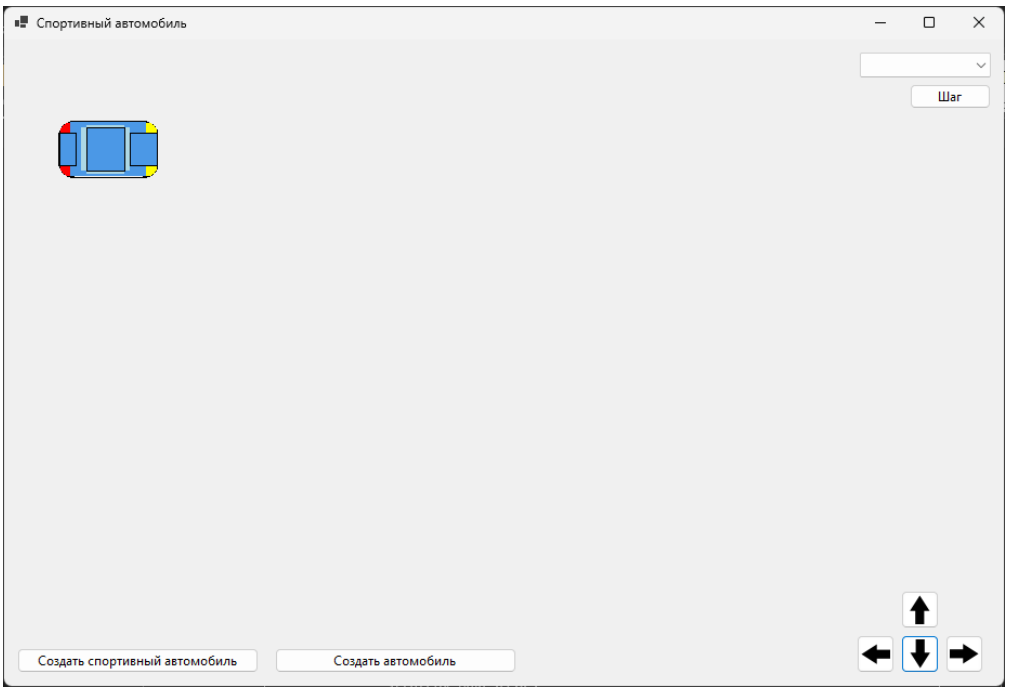


Рисунок 2.4 – Проверка перемещения «простого» объекта

Далее нажмем кнопку «Создать спортивный автомобиль». На форме должен появиться «продвинутый» объект (рисунок 2.5).

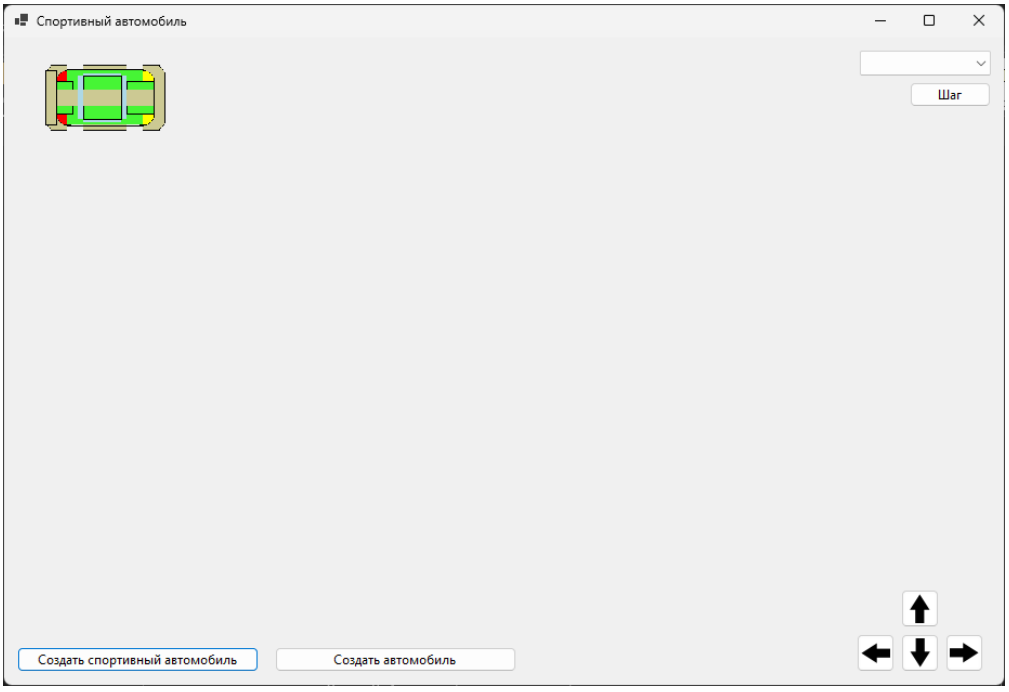


Рисунок 2.5 – Создание «продвинутого» объекта

Далее из выпадающего списка выберем стратегию «К центру» и, нажимая кнопку «Шаг» переместим объект в центр формы (рисунок 2.6).

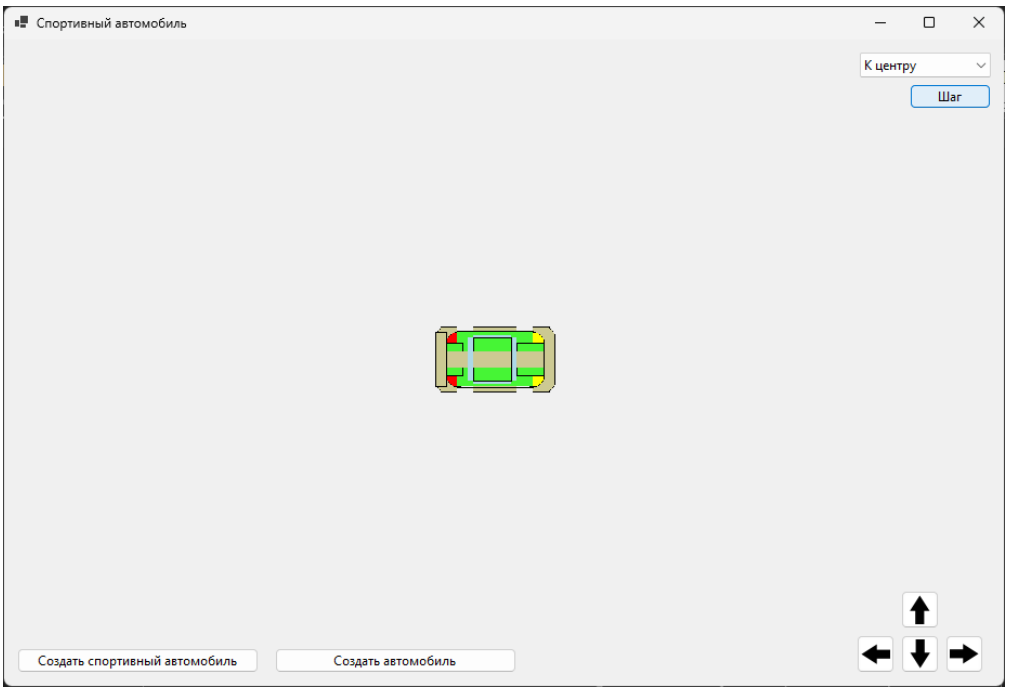


Рисунок 2.6 – Перемещение объекта в центр формы

Затем выберем стратегию «К краю» и переместим объект в правый нижний угол (рисунок 2.7).

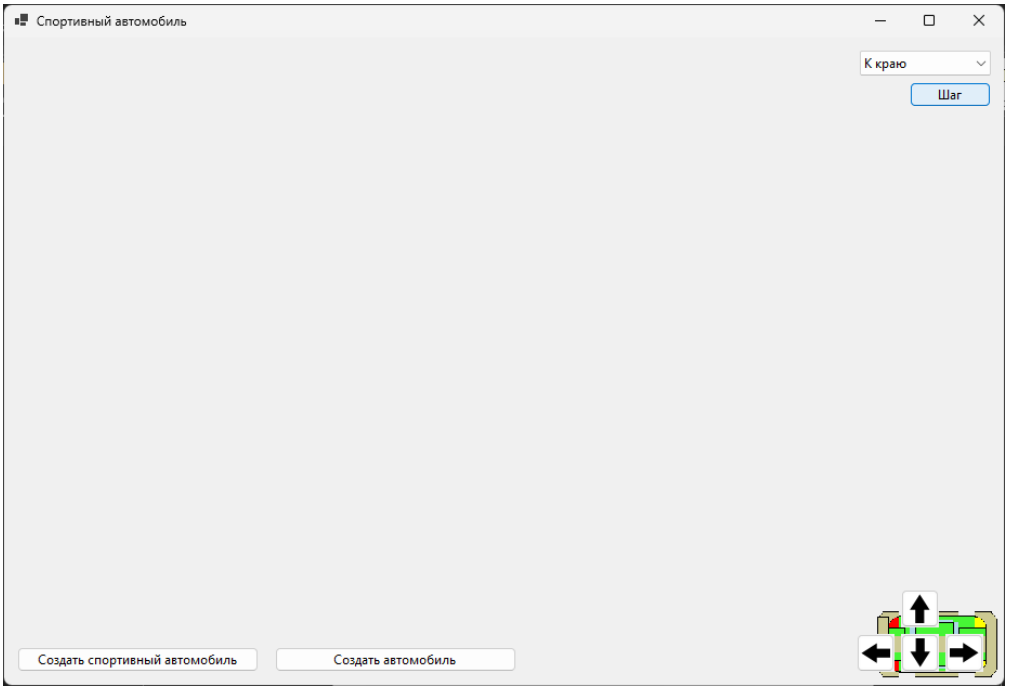


Рисунок 2.7 – Перемещение объекта к правому краю

Дополнительно проверим, что перемещение в центр работает со всех направлений и, выбрав стратегию «К центру» вернем объект в центр формы (рисунок 2.8).

Рисунок 2.8 – Возврат объекта в центр формы

Таким образом все поставленные задачи выполнены. Работа готова.

Требования

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

   ==========

4. Родительский класс должен реализовывать объект, указанный в задании.
5. Унаследовать класс-сущность «продвинутого» объекта от класса-сущности «простого» объекта.
6. Сделать вторую реализацию абстрактного класса с точкой назначения – правый нижний край формы.

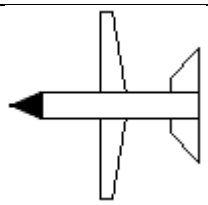
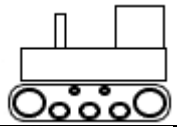
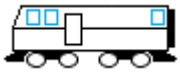
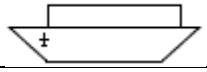
Проверка работы и порядок сдачи

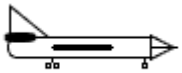
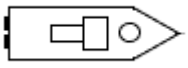
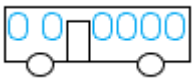
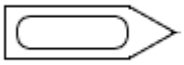
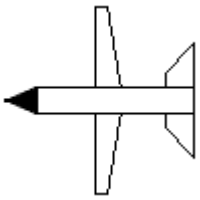
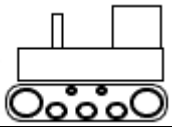
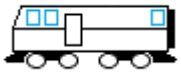
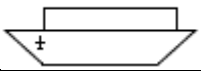
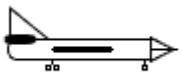
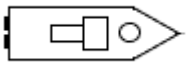
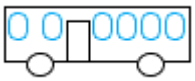
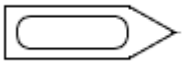
1. Создать базовый объект. Показать, как он двигается (буквально 2-3 шага в горизонтальной плоскости и вертикальной).
2. Создать продвинутый объект.
3. Выбрать первую стратегию, довести объект до центра.
4. Выбрать вторую стратегию, довести объект до края.

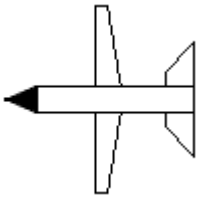
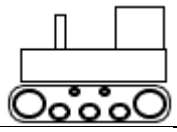
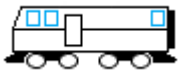
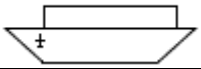
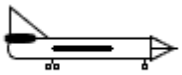
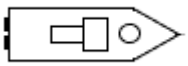
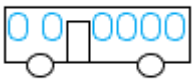
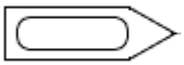
Контрольные вопросы к базовой части

1. Показать определения методов интерфейса. Обосновать, что в этом месте именно определение методов интерфейса, а не просто определение методов с идентичными именами и передаваемыми параметрами.
2. Показать переопределение методов.
3. Показать места вызовов абстрактных методов. Какова особенность вызова абстрактных методов?

Варианты

Вар.	Простой объект	Изображение
1.	Военный самолет	
2.	Грузовик	
3.	Гусеничная машина	
4.	Локомотив	
5.	Корабль	

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

Вар.	Простой объект	Изображение
21.	Военный самолет	
22.	Грузовик	
23.	Гусеничная машина	
24.	Локомотив	
25.	Корабль	
26.	Бронированная машина	
27.	Самолет	
28.	Военный корабль	
29.	Автобус	
30.	Лодка	

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 универсальный интерфейс, описывающий хранение коллекции объектов определенного типа. Создать реализацию на основе массива. При вставке по позиции искать свободное место сперва после места вставки (если оно занято), потом до места вставки (если после места вставки все занято). При удалении зачищать место, массив не сдвигать.*
    б. *Создать абстрактный класс и его реализацию, описывающую компанию, имеющую в своем распоряжении некую коллекцию объектов. Добавление и удалении объектов из коллекции абстрактного класса сделать через перегрузку операторов сложения и вычитания.*
    в. *Создать форму, для работы с абстрактным классом. В форме в отдельной панели сделать возможность добавления объекта (просто или продвинутый), удаления объекта по позиции, просмотр всего набора, передачу*

объекта в первую форму (первую форму потребуется доработать, в частности, убрать методы создания объекта там).

Проектирование

Интерфейс сделаем параметризованным (универсальным) и объявим в нем следующие методы:

- Вставка
- Вставка по позиции
- Удаление по позиции
- Получение объекта по позиции
- Количество объектов в коллекции
- Установка максимального количества элементов в коллекции

Параметр будет использоваться непосредственно в коллекции, которая будет хранить объекты.

Реализация на массивах. Для фиксации пустого места в массиве зададим ограничение, что тип-параметр должен быть ссылочным. Таким образом присвоив элементу массива значение Null, обозначим его как пустое. Логика вставки уже описана, если место, куда хотим вставить пустое, ищем ближайшее свободное, сперва справа от места вставки, потом слева, если справа все занято. В методе вставки без указания позиции вставлять будем в начало массива. При удалении и получении объекта следует сперва проверять, что указанная позиция не вызовет ошибку выхода за границы.

Далее займемся абстрактным классом. Для хранения объектов в абстрактном классе будем использовать описанный выше интерфейс. При перегрузке оператора необходимо, чтобы один из операндов являлся объект от класса, где мы перегружаем оператор. Вторым операндом будет: для сложения – добавляемый в коллекцию объект, а для вычитания – номер позиции, с которой будет удалять. Результатом работы обеих операций будет булевское

значение. Помимо перегрузки операторов сложения и вычитания, пропишем еще метод получения случайного объекта из коллекции, а также вывод на форму. При этом логику прорисовки заднего фона, а также логику расстановки объектов отдадим на откуп реализациям абстрактного класса.

Остается создать форму. На форму добавляем элемент-панель, на которую помещаем выпадающий список для выбора реализации абстрактного класса, кнопки для добавления объекта (простой и продвинутый), удаления объекта (к ней еще потребуется текстовое поле для ввода номера позиции), передачу объекта в другую форму и обновления формы. Панель разместим в правой части формы, а на оставшуюся часть – элемент PictureBox для вывода коллекции.

Реализация

Интерфейс назовем ICollectionGenericObjects. В принципе, мы все описали, что в нем должно быть. Единственно, заменим метод получения количества объектов в коллекции на свойство с геттером, а метод установки максимального количества элементов в коллекции на свойство с сеттером (листинг 3.1).

```
namespace ProjectSportCar.CollectionGenericObjects;

/// <summary>
/// Интерфейс описания действий для набора хранимых объектов
/// </summary>
/// <typeparam name="T">Параметр: ограничение - ссылочный тип</typeparam>
public interface ICollectionGenericObjects<T>
       where T : class
{
       /// <summary>
       /// Количество объектов в коллекции
       /// </summary>
       int Count { get; }

       /// <summary>
       /// Установка максимального количества элементов
       /// </summary>
       int SetMaxCount { set; }

       /// <summary>
       /// Добавление объекта в коллекцию
       /// </summary>
       /// <param name="obj">Добавляемый объект</param>
```

```
        /// <returns>true - вставка прошла удачно, false - вставка не
удалась</returns>
        bool Insert(T obj);

        /// <summary>
        /// Добавление объекта в коллекцию на конкретную позицию
        /// </summary>
        /// <param name="obj">Добавляемый объект</param>
        /// <param name="position">Позиция</param>
        /// <returns>true - вставка прошла удачно, false - вставка не
удалась</returns>
        bool Insert(T obj, int position);

        /// <summary>
        /// Удаление объекта из коллекции с конкретной позиции
        /// </summary>
        /// <param name="position">Позиция</param>
        /// <returns>true - удаление прошло удачно, false - удаление не
удалось</returns>
        bool Remove(int position);

        /// <summary>
        /// Получение объекта по позиции
        /// </summary>
        /// <param name="position">Позиция</param>
        /// <returns>Объект</returns>
        T? Get(int position);
}
```

Листинг 3.1 – Интерфейс, описывающий работу с коллекцией

Теперь создадим реализацию этого интерфейса (листинг 3.2).

```
namespace ProjectSportCar.CollectionGenericObjects;

/// <summary>
/// Параметризованный набор объектов
/// </summary>
/// <typeparam name="T">Параметр: ограничение - ссылочный тип</typeparam>
public class MassiveGenericObjects<T> : ICollectionGenericObjects<T>
        where T : class
{
        /// <summary>
        /// Массив объектов, которые храним
        /// </summary>
        private T?[] _collection;

        public int Count => _collection.Length;

        public int SetMaxCount
        {
                set
                {
                        if (value > 0)
                        {
                                if (_collection.Length > 0)
                                {
                                        Array.Resize(ref _collection, value);
                                }
                                else
                                {
                                        _collection = new T?[value];
                                }
```

```
                    }
                }
        }
        /// <summary>
        /// Конструктор
        /// </summary>
        public MassiveGenericObjects()
        {
                _collection = Array.Empty<T?>();
        }

        public T? Get(int position)
        {
                // TODO проверка позиции
                return _collection[position];
        }

        public bool Insert(T obj)
        {
                // TODO вставка в свободное место набора
                return false;
        }

        public bool Insert(T obj, int position)
        {
                // TODO проверка позиции
                // TODO проверка, что элемент массива по этой позиции пустой, если
нет, то
                //           ищется свободное место после этой позиции и идет вставка
туда
                //           если нет после, ищем до
                // TODO вставка
                return false;
        }

        public bool Remove(int position)
        {
                // TODO проверка позиции
                // TODO удаление объекта из массива, присвоив элементу массива
значение null
                return true;
        }
}
```

Листинг 3.2 – Класс-реализация интерфейса на массиве

Перейдем к абстрактному классу. По сути, за счет использования в абстрактном классе вместо конкретной коллекции интерфейс, получим более универсальную конструкцию, позволяя создавать различные экземпляры с различными коллекциями хранения объектов. Теперь, если у нас появится езе один способ хранения коллекции, не потребуется создавать еще классы-наследники от абстрактного класса под новые способы хранения. Такой шаблон в проектировании называется «Мост» (листинг 3.3).

```
using ProjectSportCar.Drawnings;
```

```
namespace ProjectSportCar.CollectionGenericObjects;

/// <summary>
/// Абстракция компании, хранящий коллекцию автомобилей
/// </summary>
public abstract class AbstractCompany
{
	/// <summary>
	/// Размер места (ширина)
	/// </summary>
	protected readonly int _placeSizeWidth = 210;

	/// <summary>
	/// Размер места (высота)
	/// </summary>
	protected readonly int _placeSizeHeight = 80;

	/// <summary>
	/// Ширина окна
	/// </summary>
	protected readonly int _pictureWidth;

	/// <summary>
	/// Высота окна
	/// </summary>
	protected readonly int _pictureHeight;

	/// <summary>
	/// Коллекция автомобилей
	/// </summary>
	protected ICollectionGenericObjects<DrawningCar>? _collection = null;

	/// <summary>
	/// Вычисление максимального количества элементов, который можно разместить
в окне
	/// </summary>
	private int GetMaxCount => _pictureWidth * _pictureHeight /
(_placeSizeWidth * _placeSizeHeight);

	/// <summary>
	/// Конструктор
	/// </summary>
	/// <param name="picWidth">Ширина окна</param>
	/// <param name="picHeight">Высота окна</param>
	/// <param name="collection">Коллекция автомобилей</param>
	public AbstractCompany(int picWidth, int picHeight,
ICollectionGenericObjects<DrawningCar> collection)
	{
		_pictureWidth = picWidth;
		_pictureHeight = picHeight;
		_collection = collection;
		_collection.SetMaxCount = GetMaxCount;
	}

	/// <summary>
	/// Перегрузка оператора сложения для класса
	/// </summary>
	/// <param name="company">Компания</param>
	/// <param name="car">Добавляемый объект</param>
	/// <returns></returns>
	public static bool operator +(AbstractCompany company, DrawningCar car)
	{
		return company._collection?.Insert(car) ?? false;
```

```
        }

        /// <summary>
        /// Перегрузка оператора удаления для класса
        /// </summary>
        /// <param name="company">Компания</param>
        /// <param name="position">Номер удаляемого объекта</param>
        /// <returns></returns>
        public static bool operator -(AbstractCompany company, int position)
        {
                return company._collection?.Remove(position) ?? false;
        }

        /// <summary>
        /// Получение случайного объекта из коллекции
        /// </summary>
        /// <returns></returns>
        public DrawningCar? GetRandomObject()
        {
                Random rnd = new();
                return _collection?.Get(rnd.Next(GetMaxCount));
        }

        /// <summary>
        /// Вывод всей коллекции
        /// </summary>
        /// <returns></returns>
        public Bitmap? Show()
        {
                Bitmap bitmap = new(_pictureWidth, _pictureHeight);
                Graphics graphics = Graphics.FromImage(bitmap);
                DrawBackgound(graphics);

                SetObjectsPosition();
                for (int i = 0; i < (_collection?.Count ?? 0); ++i)
                {
                        DrawningCar? obj = _collection?.Get(i);
                        obj?.DrawTransport(graphics);
                }

                return bitmap;
        }

        /// <summary>
        /// Вывод заднего фона
        /// </summary>
        /// <param name="g"></param>
        protected abstract void DrawBackgound(Graphics g);

        /// <summary>
        /// Расстановка объектов
        /// </summary>
        protected abstract void SetObjectsPosition();
}
```

Листинг 3.3 – Абстрактный класс, описывающий компанию

Под абстрактный класс потребуется сделать реализацию, в которой нужно будет переопределить 2 метода (какой класс – описано в задании). В

данном случае делаем реализацию – типа каршеринговая компания (листинг 3.4).

```
using ProjectSportCar.Drawnings;

namespace ProjectSportCar.CollectionGenericObjects;

/// <summary>
/// Реализация абстрактной компании - каршеринг
/// </summary>
public class CarSharingService : AbstractCompany
{
	/// <summary>
	/// Конструктор
	/// </summary>
	/// <param name="picWidth"></param>
	/// <param name="picHeight"></param>
	/// <param name="collection"></param>
	public CarSharingService(int picWidth, int picHeight,
ICollectionGenericObjects<DrawningCar> collection) : base(picWidth, picHeight,
collection)
	{
	}

	protected override void DrawBackgound(Graphics g)
	{
		throw new NotImplementedException();
	}

	protected override void SetObjectsPosition()
	{
		throw new NotImplementedException();
	}
}
```

Листинг 3.4 – Реализация абстрактного класса, описывающий компанию

Теперь создадим новую форму (рисунок 3.1).

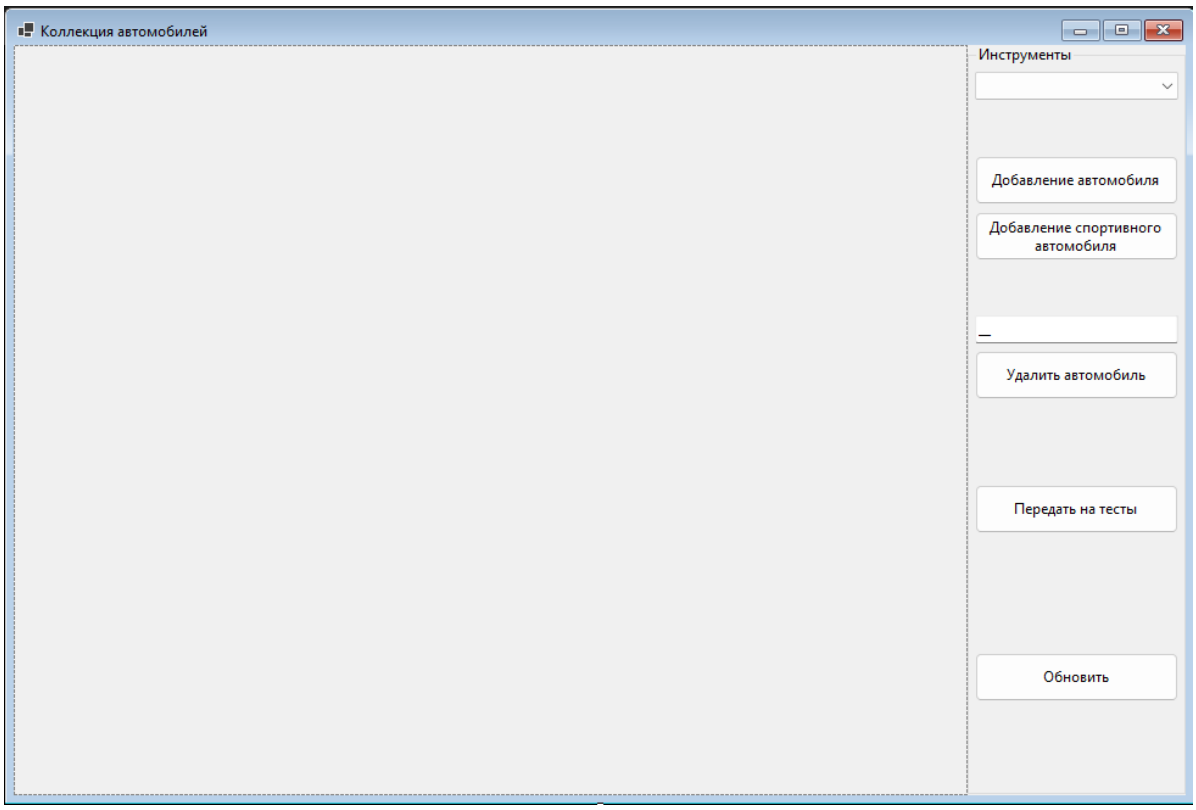


Рисунок 3.1 – Форма для работы с компаниями

Для работы кнопки «Передача на тесты» необходимо внести изменений в первую форму FormSportCar. Для этого сделаем ряд манипуляций: уберем кнопки создания объектов, выпилим весь код, который был с этим связан и пропишем свойство с сеттером, через которое будет принимать объект класса-прорисовки (листинг 3.5).

```
using ProjectSportCar.Drawnings;
using ProjectSportCar.MovementStrategy;

namespace ProjectSportCar;

/// <summary>
/// Форма работы с объектом "Спортивный автомобиль"
/// </summary>
public partial class FormSportCar : Form
{
        /// <summary>
        /// Поле-объект для прорисовки объекта
        /// </summary>
        private DrawningCar? _drawningCar;

        /// <summary>
        /// Стратегия перемещения
        /// </summary>
        private AbstractStrategy? _strategy;
```

```
        /// <summary>
        /// Получение объекта
        /// </summary>
        public DrawningCar SetCar
        {
                set
                {
                        _drawningCar = value;
                        _drawningCar.SetPictureSize(pictureBoxSportCar.Width,
pictureBoxSportCar.Height);
                        comboBoxStrategy.Enabled = true;
                        _strategy = null;
                        Draw();
                }
        }

        /// <summary>
        /// Конструктор формы
        /// </summary>
        public FormSportCar()
        {
                InitializeComponent();
                _strategy = null;
        }

        /// <summary>
        /// Метод прорисовки машины
        /// </summary>
        private void Draw()
        {
                if (_drawningCar == null)
                {
                        return;
                }

                Bitmap bmp = new(pictureBoxSportCar.Width,
pictureBoxSportCar.Height);
                Graphics gr = Graphics.FromImage(bmp);
                _drawningCar.DrawTransport(gr);
                pictureBoxSportCar.Image = bmp;
        }

        /// <summary>
        /// Перемещение объекта по форме (нажатие кнопок навигации)
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void ButtonMove_Click(object sender, EventArgs e)
        {
                if (_drawningCar == null)
                {
                        return;
                }

                string name = ((Button)sender)?.Name ?? string.Empty;
                bool result = false;
                switch (name)
                {
                        case "buttonUp":
                                result = _drawningCar.MoveTransport(DirectionType.Up);
                                break;
                        case "buttonDown":
                                result = _drawningCar.MoveTransport(DirectionType.Down);
```

```
                        break;
                    case "buttonLeft":
                        result = _drawningCar.MoveTransport(DirectionType.Left);
                        break;
                    case "buttonRight":
                        result =
_drawningCar.MoveTransport(DirectionType.Right);
                        break;
            }

            if (result)
            {
                Draw();
            }
        }

        /// <summary>
        /// Обработка нажатия кнопки "Шаг"
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void ButtonStrategyStep_Click(object sender, EventArgs e)
        {
            if (_drawningCar == null)
            {
                return;
            }

            if (comboBoxStrategy.Enabled)
            {
                _strategy = comboBoxStrategy.SelectedIndex switch
                {
                    0 => new MoveToCenter(),
                    1 => new MoveToBorder(),
                    _ => null,
                };
                if (_strategy == null)
                {
                    return;
                }
                _strategy.SetData(new MoveableCar(_drawningCar),
pictureBoxSportCar.Width, pictureBoxSportCar.Height);
            }

            if (_strategy == null)
            {
                return;
            }

            comboBoxStrategy.Enabled = false;
            _strategy.MakeStep();
            Draw();

            if (_strategy.GetStatus() == StrategyStatus.Finish)
            {
                comboBoxStrategy.Enabled = true;
                _strategy = null;
            }
        }
}
```

Листинг 3.5 – Новая логика формы FormSportCar

Остается логика самой формы работы с набором. Там все просто, нам понадобится объект от абстрактного класса, инициализацию которого сделаем при выборе элемента из выпадающего списка, и реализовать обработку нажатия 5 кнопок... ой, нам теперь тут создавать объекты, а красивый код создания мы удалили, что же делать, не писать же его заново… А на это у нас есть Git. Открываем вкладку «Изменения Git». Находим там в списке файл FormSportCar.cs, правой кнопкой вызываем пункт «Сравнить с неизменным» и вот наш код (он даже красным будет подсвечен, так как был удален).

Остается его грамотно скопировать и перенести в логику новой формы. Попутно добавив вызов диалогового окна для выбора цвета (или цветов, если создаем «продвинутый» объект) создаваемого объекта (листинг 3.6).

```
using ProjectSportCar.CollectionGenericObjects;
using ProjectSportCar.Drawnings;

namespace ProjectSportCar;

/// <summary>
/// Форма работы с компанией и ее коллекцией
/// </summary>
public partial class FormCarCollection : Form
{
	/// <summary>
	/// Компания
	/// </summary>
	private AbstractCompany? _company = null;

	/// <summary>
	/// Конструктор
	/// </summary>
	public FormCarCollection()
	{
		InitializeComponent();
	}

	/// <summary>
	/// Выбор компании
	/// </summary>
	/// <param name="sender"></param>
	/// <param name="e"></param>
	private void ComboBoxSelectorCompany_SelectedIndexChanged(object sender,
EventArgs e)
	{
		switch (comboBoxSelectorCompany.Text)
		{
			case "Хранилище":
				_company = new CarSharingService(pictureBox.Width,
pictureBox.Height, new MassiveGenericObjects<DrawningCar>());
				break;
		}
	}
```

```
        /// <summary>
        /// Добавление обычного автомобиля
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void ButtonAddCar_Click(object sender, EventArgs e) =>
CreateObject(nameof(DrawningCar));

        /// <summary>
        /// Добавление спортивного автомобиля
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void ButtonAddSportCar_Click(object sender, EventArgs e) =>
CreateObject(nameof(DrawningSportCar));

        /// <summary>
        /// Создание объекта класса-перемещения
        /// </summary>
        /// <param name="type">Тип создаваемого объекта</param>
        private void CreateObject(string type)
        {
                if (_company == null)
                {
                        return;
                }

                Random random = new();
                DrawningCar drawningCar;
                switch (type)
                {
                        case nameof(DrawningCar):
                                drawningCar = new DrawningCar(random.Next(100, 300),
random.Next(1000, 3000), GetColor(random));
                                break;
                        case nameof(DrawningSportCar):
                                // TODO вызов диалогового окна для выбора цвета
                                drawningCar = new DrawningSportCar(random.Next(100,
300), random.Next(1000, 3000),
                                        Color.FromArgb(random.Next(0, 256),
random.Next(0, 256), random.Next(0, 256)),
                                        Color.FromArgb(random.Next(0, 256),
random.Next(0, 256), random.Next(0, 256)),
                                        Convert.ToBoolean(random.Next(0, 2)),
Convert.ToBoolean(random.Next(0, 2)), Convert.ToBoolean(random.Next(0, 2)));
                                break;
                        default:
                                return;
                }

                if (_company + drawningCar)
                {
                        MessageBox.Show("Объект добавлен");
                        pictureBox.Image = _company.Show();
                }
                else
                {
                        MessageBox.Show("Не удалось добавить объект");
                }
        }

        /// <summary>
```

```
        /// Получение цвета
        /// </summary>
        /// <param name="random">Генератор случайных чисел</param>
        /// <returns></returns>
        private static Color GetColor(Random random)
        {
                Color color = Color.FromArgb(random.Next(0, 256), random.Next(0,
256), random.Next(0, 256));
                ColorDialog dialog = new();
                if (dialog.ShowDialog() == DialogResult.OK)
                {
                        color = dialog.Color;
                }

                return color;
        }

        /// <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) || _company ==
null)
                {
                        return;
                }

                if (MessageBox.Show("Удалить объект?", "Удаление",
MessageBoxButtons.YesNo, MessageBoxIcon.Question) != DialogResult.Yes)
                {
                        return;
                }

                int pos = Convert.ToInt32(maskedTextBoxPosition.Text);
                if (_company - pos)
                {
                        MessageBox.Show("Объект удален");
                        pictureBox.Image = _company.Show();
                }
                else
                {
                        MessageBox.Show("Не удалось удалить объект");
                }
        }

        /// <summary>
        /// Передача объекта в другую форму
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void ButtonGoToCheck_Click(object sender, EventArgs e)
        {
                if (_company == null)
                {
                        return;
                }

                DrawningCar? car = null;
                int counter = 100;
                while (car == null)
```

```
            {
                car = _company.GetRandomObject();
                counter--;
                if (counter <= 0)
                {
                    break;
                }
            }

            if (car == null)
            {
                return;
            }

            FormSportCar form = new()
            {
                SetCar = car
            };
            form.ShowDialog();
        }

        /// <summary>
        /// Перерисовка коллекции
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void ButtonRefresh_Click(object sender, EventArgs e)
        {
            if (_company == null)
            {
                return;
            }

            pictureBox.Image = _company.Show();
        }
}
```

Листинг 3.6 – Логика формы работы с компаний

И самый последний шаг, в классе Program нужно изменить форму, с которой будет начинаться программа (листинг 3.7).

```
namespace ProjectSportCar
{
    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 FormCarCollection());
        }
    }
```

```
}
```

Листинг 3.7 – Обновленная логика класса Program

Тестирование

Запустим проект, выберем в выпадающем списке пункт «Хранилище» и нажмем кнопку «Добавить автомобиль», должно появится диалоговое окно выбора цвета (рисунок 3.2).

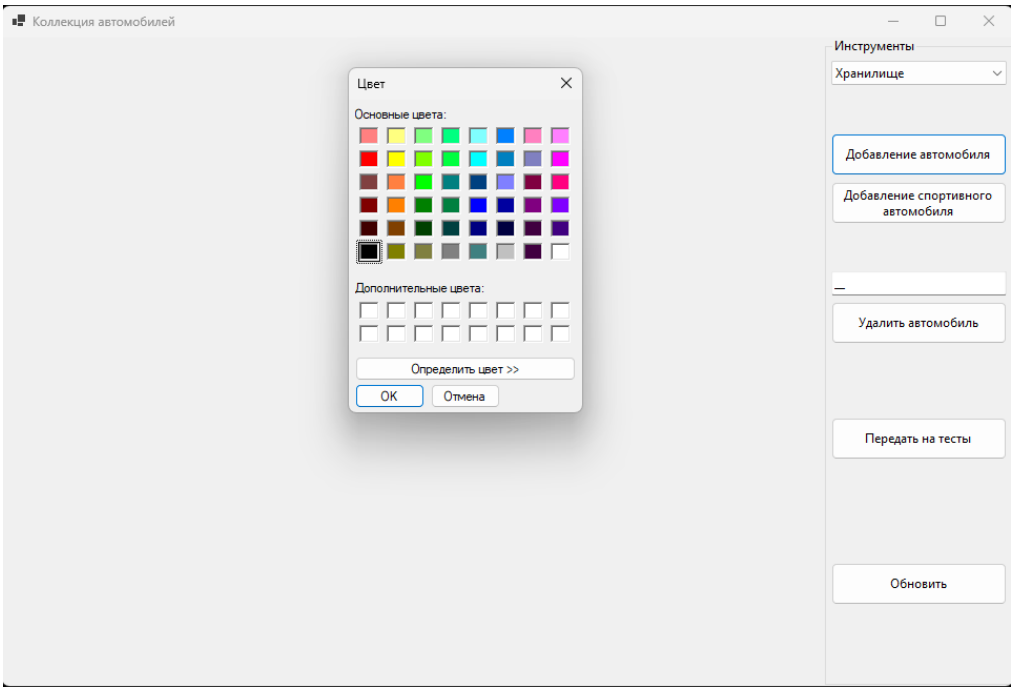


Рисунок 3.2 – Диалоговое окно для выбора цвета

Выбираем любой цвет и убеждаемся, что создается объект именно с таким цветом, а также, что он располагается в границах линий и не пересекает их (рисунок 3.3).

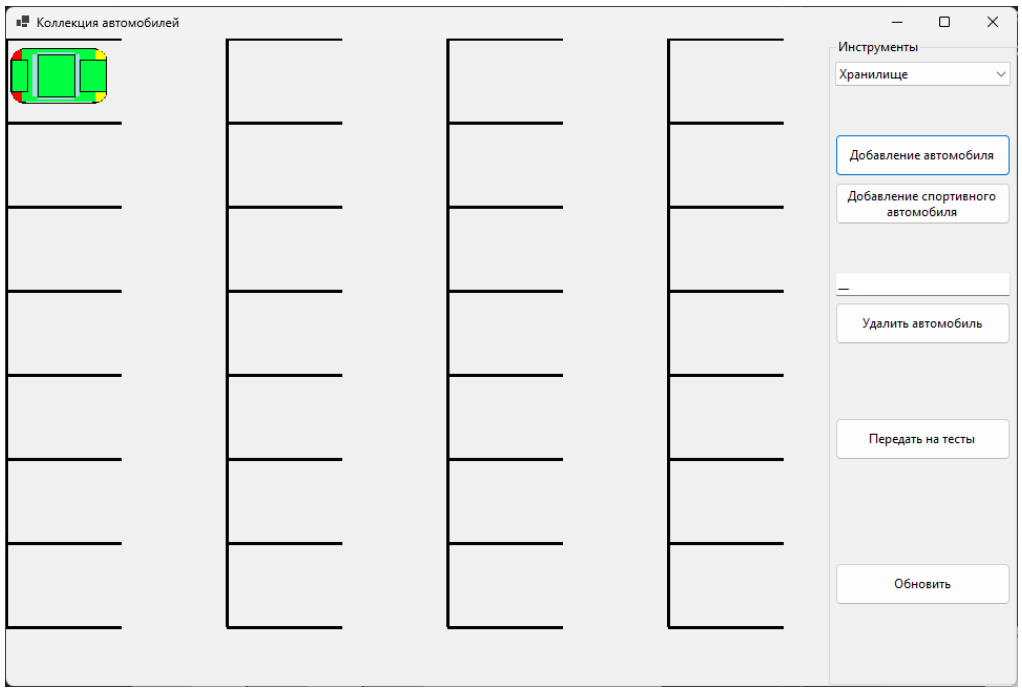


Рисунок 3.3 – Добавление «простого» объекта в хранилище

Далее нажмем кнопку «Добавление спортивного автомобиля». Выбираем 2 цвета через диалоговые окна, убеждаемся, что создается объект именно с такими цветами, а также, что он располагается в границах линий и не пересекает их (рисунок 3.4).

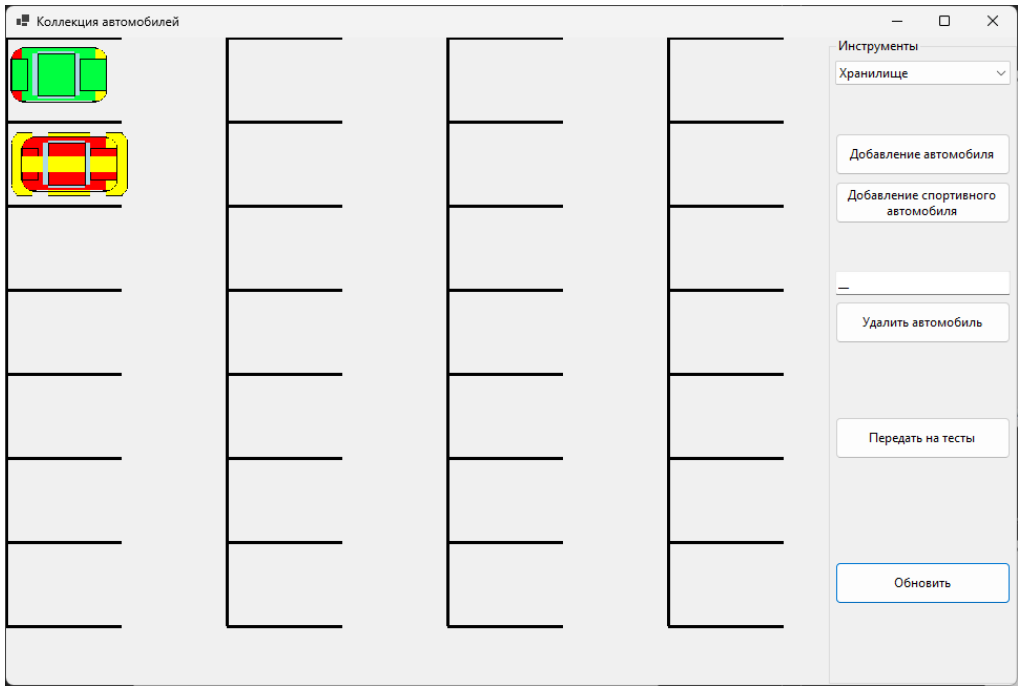


Рисунок 3.4 – Добавление «продвинутого» объекта в хранилище

Далее добавляем объекты (можно случайных цветов) так, чтобы одна линия заполнилась полностью и 1-2 объекта добавились на следующую (рисунок 3.5).

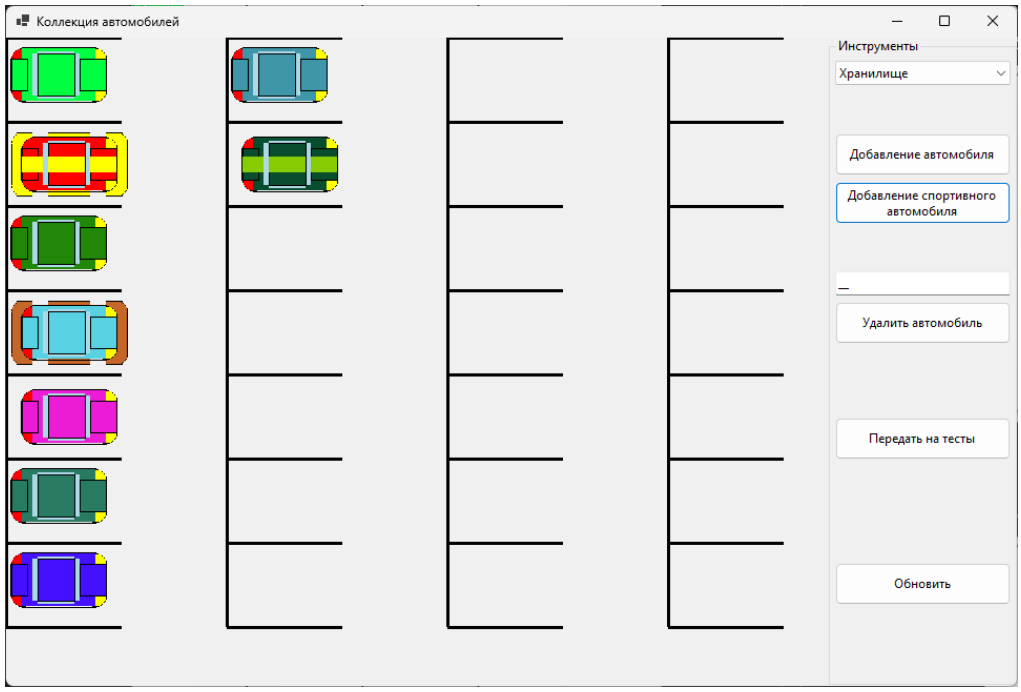


Рисунок 3.5 – Добавленные объекты в хранилище

Затем введем позицию автомобиля для удаления (чтобы он был не первым и не последним в коллекции) и удалим его. Должно остаться пустой место (рисунок 3.6).

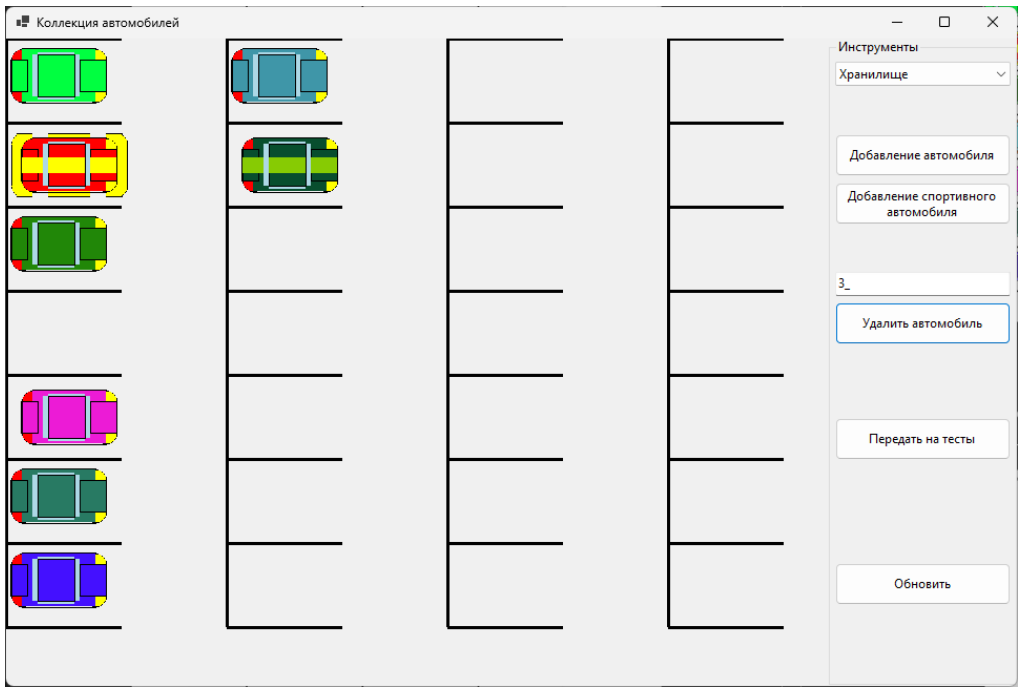


Рисунок 3.6 – Удаление объекта из хранилища

Нажмем кнопку «Передать на тесты» и убедимся, что объект передается на другую форму (рисунок 3.7).

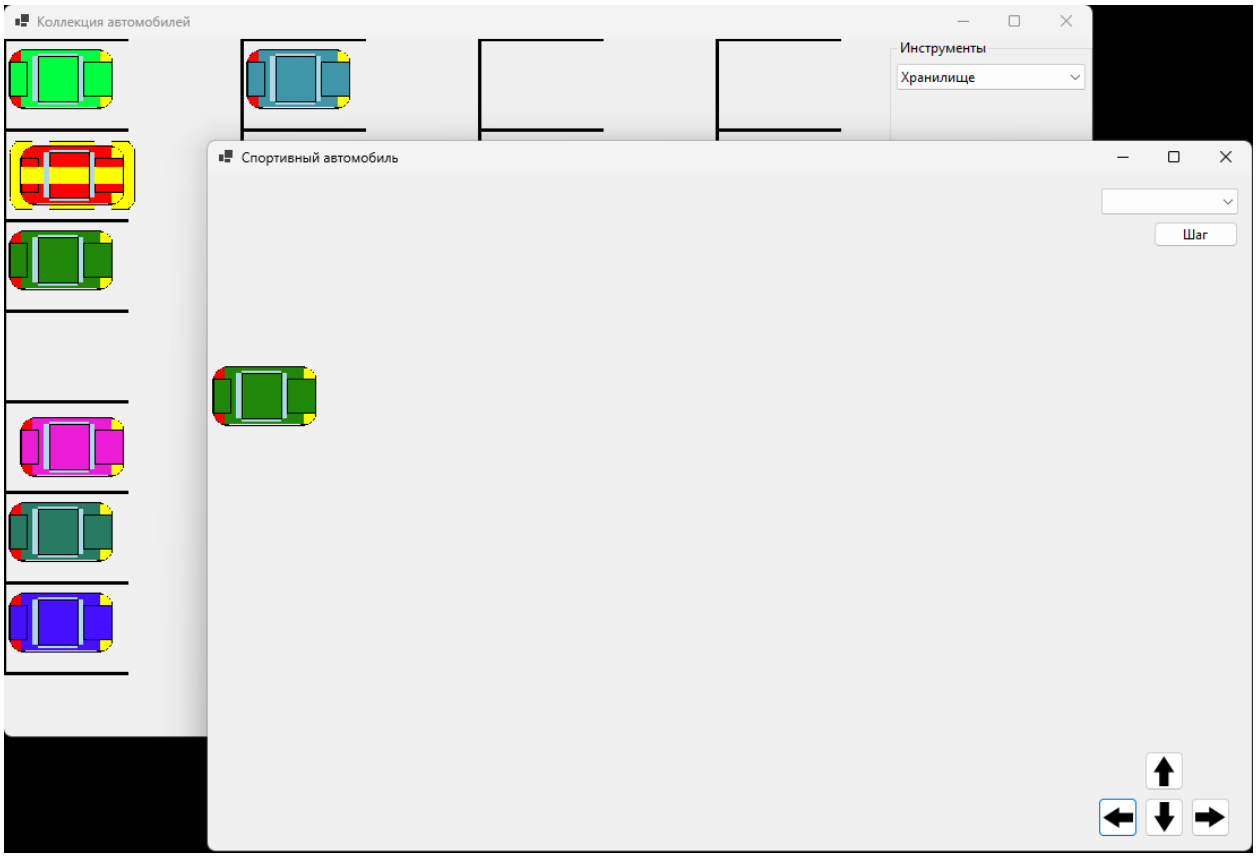


Рисунок 3.7 – Передача объекта на другую форму

Закроем форму и убедимся, что объект из хранилища не удалился, нажав на кнопку «Обновить» (рисунок 3.8).

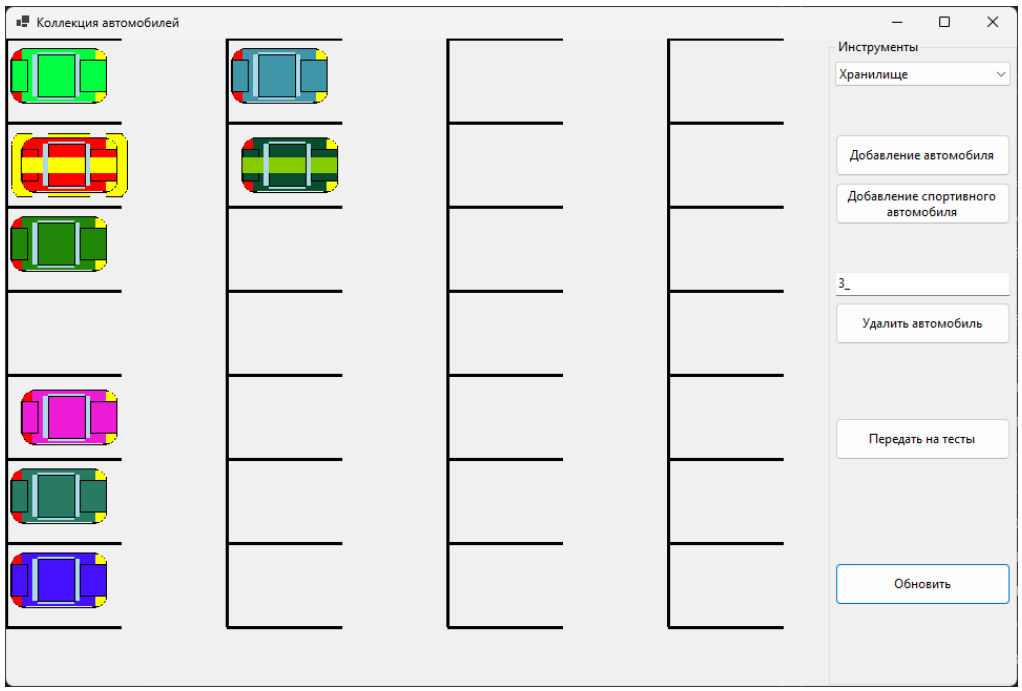


Рисунок 3.8 – Обновление коллекции

Таким образом все поставленные задачи выполнены. Работа готова.

Требования

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

   ==========

4. Дописать логику в методах вставки, удаления и получения элемента класса MassiveGenericObjects, а также дописать логику создания «продвинутого объекта» с вызовом диалоговых окон для выбора цветов на первой форме.
5. В классе абстрактной компании поменять типы возвращаемых значений для операций сложения и вычитания (тип возвращаемого значения должен логически согласовываться с результатом операции

для типов передаваемых операндов, возможно, потребуется поменять типы возвращаемых значений в методах интерфейса).

6. Создать реализацию абстрактного класса, согласно варианту. Направление вывода объектов указано также во вариантах. **Важно!** Линни разметки должны ограничивать объект (т.е. прорисовываться вокруг объекта). Объекты на форме не должны «накладываться» на линии разметки и друг на друга. На форме коллекции задать такую ширину формы, чтобы там можно было добавить минимум 4 объекта в ряд.

Проверка работы и порядок сдачи

1. Создать простой объект с выбранным цветом.
2. Создать продвинутый объект с выбранными цветами.
3. Создать ряд объектов так, чтобы было заполнен один ряд полностью и второй на 1-2 объекта.
4. Удалить один объект (не первый и не последний).
5. Один из объектов передать на первую форму.
6. Закрыть первую форму, убедиться, что объект не был удален с формы коллекции.

Контрольные вопросы к базовой части

1. Где перегрузка методов, операторов (описание и вызов).
2. Показать параметризованный класс, доказать, что он параметризованный, показать объект от параметризованного класса.
3. Где используется полиморфизм подтипов, показать на любом примере в коде.

Варианты

Вар.	Продвинутый / простой объект	Реализация абстрактного класса	Направление
1.	Бомбардировщик / Военный самолет	Ангар	Влево, вниз
2.	Самосвал / Грузовик	Автопарк	Вправо, вниз
3.	Бульдозер / Гусеничная машина	Гараж	Влево, вверх
4.	Электровоз / Локомотив	Депо	Вправо, вверх
5.	Теплоход / Корабль	Пристань	Влево, вниз
6.	Танк / Бронированная машина	База	Вправо, вниз
7.	Аэробус / Самолет	Аэродром	Влево, вверх
8.	Линкор / Военный корабль	Доки	Вправо, вверх
9.	Автобус двухэтажный / Автобус	Автовокзал	Влево, вниз
10.	Катер / Лодка	Гавань	Вправо, вниз
11.	Истребитель / Военный самолет	Ангар	Влево, вверх
12.	Бензовоз / Грузовик	Автопарк	Вправо, вверх
13.	Экскаватор / Гусеничная машина	Гараж	Влево, вниз
14.	Тепловоз / Локомотив	Депо	Вправо, вниз
15.	Контейнеровоз / Корабль	Пристань	Влево, вверх
16.	Самоходная арт. установка / Бронированная машина	База	Вправо, вверх
17.	Самолет с радаром / Самолет	Аэродром	Влево, вниз

Вар.	Продвинутый / простой объект	Реализация абстрактного класса	Направление
18.	Крейсер / Военный корабль	Доки	Вправо, вниз
19.	Автобус с гармошкой / Автобус	Автовокзал	Влево, вверх
20.	Катамаран / Лодка	Гавань	Вправо, вверх
21.	Штурмовик / Военный самолет	Ангар	Влево, вниз
22.	Подметально-уборочная машина / Грузовик	Автопарк	Вправо, вниз
23.	Подъемный кран / Гусеничная машина	Гараж	Влево, вверх
24.	Монорельс / Локомотив	Депо	Вправо, вверх
25.	Лайнер / Корабль	Пристань	Влево, вниз
26.	Зенитная установка / Бронированная машина	База	Вправо, вниз
27.	Гидросамолет / Самолет	Аэродром	Влево, вверх
28.	Авианосец / Военный корабль	Доки	Вправо, вверх
29.	Троллейбус / Автобус	Автовокзал	Влево, вниз
30.	Парусник / Лодка	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:*
    а. *Создать реализацию интерфейса работы с коллекцией с использованием List<T> в качестве хранилища. При простой вставке, вставлять в конец списка. При удалении просто удалять объект из списка (без замены на* null*).*
    б. *Создать класс-хранилище для объектов-реализаций интерфейса работы с коллекциями (каждому объекту сопоставлять уникальное имя). Реализовать добавление в хранилище коллекции, удаление по имени и получение по имени коллекции (через индексатор).*
    в. *Модифицировать форму, по работе с компанией. Реализовать возможность работы с класс-хранилищем: добавления в хранилище классов-коллекций, переключения между объектами, удаление объекта из хранилища. А при создании компании подкидывать туда объект из этого хранилища.*

Проектирование

Первым шагом сделаем новую реализацию для интерфейса ICollectionGenericObjects. Логика там будет простая, в основном

использоваться методы класса List. Единственно что, в отличии от массива у списка нет ограничения по количеству элементов (он ограничен только доступной памятью), так что потребуется отдельное поле для хранения этого значения.

Далее создадим класс-хранилище. Так как каждой коллекции нужно будет сопоставлять уникальное имя, то логично использовать словарь, где ключ – строка – название, а значение – объект от класса-коллекции. При вставке нужно будет проверять, что такой записи нет в словаре. При удалении и получении, что такая запись есть.

И последнее – поработаем с формой. Сперва на самой форме добавим элементы для работы с хранилищем. Добавим текстовое поле для ввода названия, 2 элемента для выбора типа коллекции (массив или список, radioButton), кнопки для добавления объекта класса-коллекции а и удаления объекта класса-коллекции, а также элемент для вывода списка (воспользуемся простым – ListBox). Также инициализацию объекта абстрактного класса вынесем из выбора элемента выпадающего списка в обработку нажатия кнопки (добавим отдельную кнопку). В логике добавим объект от класса-хранилища и логику в новые методы.

Реализация

Первый шаг довольно простой, все уже было обсуждено, так что просто нужно реализовать (листинг 4.1).

```
namespace ProjectSportCar.CollectionGenericObjects;

/// <summary>
/// Параметризованный набор объектов
/// </summary>
/// <typeparam name="T">Параметр: ограничение – ссылочный тип</typeparam>
public class ListGenericObjects<T> : ICollectionGenericObjects<T>
       where T : class
{
       /// <summary>
       /// Список объектов, которые храним
       /// </summary>
       private readonly List<T?> _collection;

       /// <summary>
```

```
        /// Максимально допустимое число объектов в списке
        /// </summary>
        private int _maxCount;

        public int Count => _collection.Count;

        public int SetMaxCount { set { if (value > 0) { _maxCount = value; } } }

        /// <summary>
        /// Конструктор
        /// </summary>
        public ListGenericObjects()
        {
                _collection = new();
        }

        public T? Get(int position)
        {
                // TODO проверка позиции
                return _collection[position];
        }

        public bool Insert(T obj)
        {
                // TODO проверка, что не превышено максимальное количество элементов
                // TODO вставка в конец набора
                return true;
        }

        public bool Insert(T obj, int position)
        {
                // TODO проверка, что не превышено максимальное количество элементов
                // TODO проверка позиции
                // TODO вставка по позиции
                return true;
        }

        public bool Remove(int position)
        {
                // TODO проверка позиции
                // TODO удаление объекта из списка
                return true;
        }
}
```

Листинг 4.1 – Реализация работы с коллекцией на списках

Перейдём к классу-хранилищу. Сделаем так, что класс сам отвечает за создание объекта от нужной реализации интерфейса-коллекции. Для этого нужно будет понимать, какую коллекцию требуется создать и добавить в набор. Сделаем новое перечисление, тип коллекции, которое будем передавать параметром в метод добавления в хранилище новой записи. Пока в перечислении будет 2 варианта: массив и список (листинг 4.2).

```
namespace ProjectSportCar.CollectionGenericObjects;

/// <summary>
```

```
/// Тип коллекции
/// </summary>
public enum CollectionType
{
	/// <summary>
	/// Неопределено
	/// </summary>
	None = 0,

	/// <summary>
	/// Массив
	/// </summary>
	Massive = 1,

	/// <summary>
	/// Список
	/// </summary>
	List = 2
}
```

Листинг 4.2 – Перечисление тип коллекции

Теперь сам класс-хранилище (листинг 4.3).

```
namespace ProjectSportCar.CollectionGenericObjects;

/// <summary>
/// Класс-хранилище коллекций
/// </summary>
/// <typeparam name="T"></typeparam>
public class StorageCollection<T>
	where T : class
{
	/// <summary>
	/// Словарь (хранилище) с коллекциями
	/// </summary>
	readonly Dictionary<string, ICollectionGenericObjects<T>> _storages;

	/// <summary>
	/// Возвращение списка названий коллекций
	/// </summary>
	public List<string> Keys => _storages.Keys.ToList();

	/// <summary>
	/// Конструктор
	/// </summary>
	public StorageCollection()
	{
		_storages = new Dictionary<string, ICollectionGenericObjects<T>>();
	}

	/// <summary>
	/// Добавление коллекции в хранилище
	/// </summary>
	/// <param name="name">Название коллекции</param>
	/// <param name="collectionType">тип коллекции</param>
	public void AddCollection(string name, CollectionType collectionType)
	{
		// TODO проверка, что name не пустой и нет в словаре записи с таким
ключом
		// TODO Прописать логику для добавления
	}
```

```
        /// <summary>
        /// Удаление коллекции
        /// </summary>
        /// <param name="name">Название коллекции</param>
        public void DelCollection(string name)
        {
                // TODO Прописать логику для удаления коллекции
        }

        /// <summary>
        /// Доступ к коллекции
        /// </summary>
        /// <param name="name">Название коллекции</param>
        /// <returns></returns>
        public ICollectionGenericObjects<T>? this[string name]
        {
                get
                {
                        // TODO Продумать логику получения объекта
                        return null;
                }
        }
}
```

Листинг 4.3 – Класс StorageCollection

Перейдем к форме. В первую очередь, добавим элементов на форму, предварительно увеличив ее размеры (если потребуется), чтобы вместить новые элементы. Также, все элементы, завязанные на работу с компанией (добавление объектов, удаление, передачу на тесты и обновление) занесем в отдельную панель, которую изначально сделаем неактивной (выставим значение в свойстве Enabled панели в false) (рисунок 4.1).

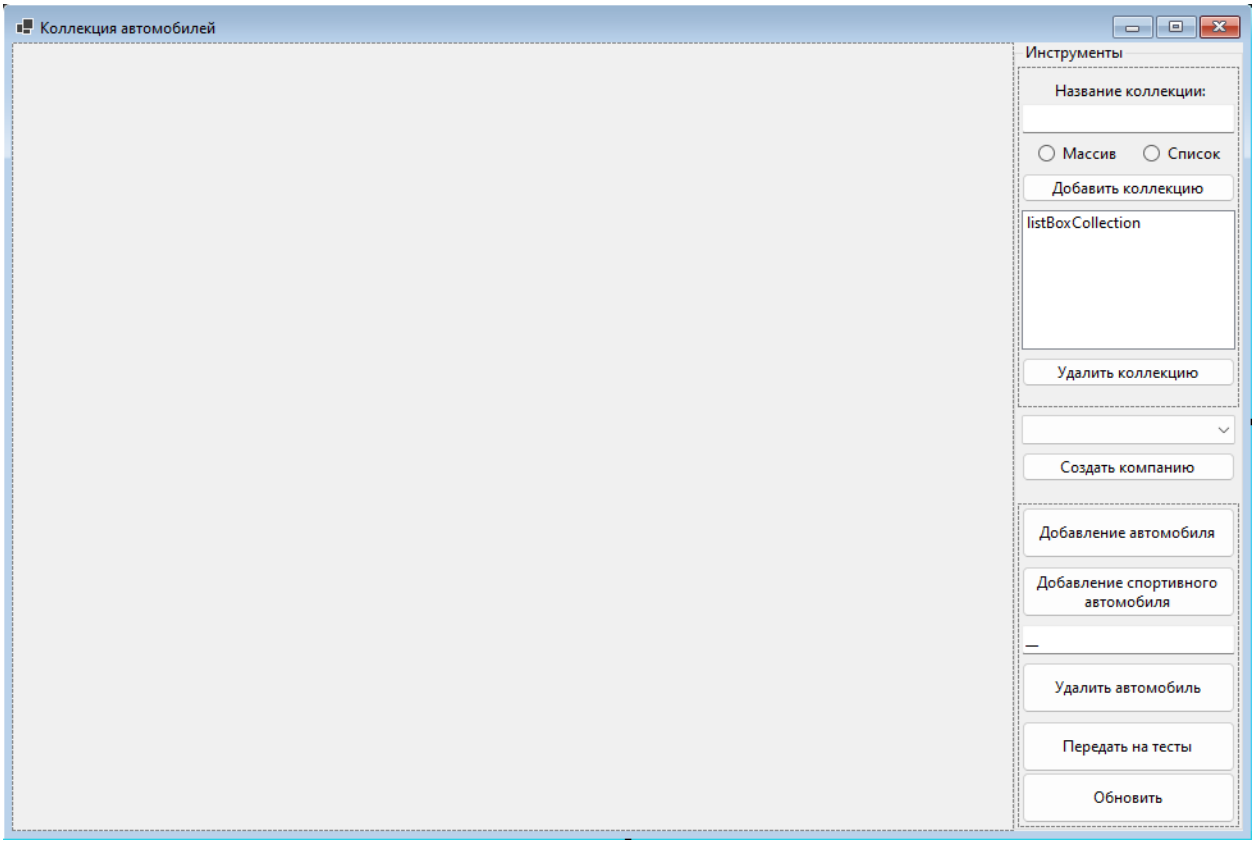


Рисунок 4.1 – Обновленный дизайн формы

Далее пойдем в логику. Все меняем, как было описано на этапе проектирования (листинг 4.4).

```
using ProjectSportCar.CollectionGenericObjects;
using ProjectSportCar.Drawnings;

namespace ProjectSportCar;

/// <summary>
/// Форма работы с компанией и ее коллекцией
/// </summary>
public partial class FormCarCollection : Form
{
	/// <summary>
	/// Хранилище коллекций
	/// </summary>
	private readonly StorageCollection<DrawningCar> _storageCollection;

	/// <summary>
	/// Компания
	/// </summary>
	private AbstractCompany? _company = null;

	/// <summary>
	/// Конструктор
	/// </summary>
	public FormCarCollection()
	{
		InitializeComponent();
		_storageCollection = new();
```

```
        }

        /// <summary>
        /// Выбор компании
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void ComboBoxSelectorCompany_SelectedIndexChanged(object sender,
EventArgs e)
        {
                panelCompanyTools.Enabled = false;
        }

        /// <summary>
        /// Добавление обычного автомобиля
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void ButtonAddCar_Click(object sender, EventArgs e) =>
CreateObject(nameof(DrawningCar));

        /// <summary>
        /// Добавление спортивного автомобиля
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void ButtonAddSportCar_Click(object sender, EventArgs e) =>
CreateObject(nameof(DrawningSportCar));

        /// <summary>
        /// Создание объекта класса-перемещения
        /// </summary>
        /// <param name="type">Тип создаваемого объекта</param>
        private void CreateObject(string type)
        {
                if (_company == null)
                {
                        return;
                }

                Random random = new();
                DrawningCar drawningCar;
                switch (type)
                {
                        case nameof(DrawningCar):
                                drawningCar = new DrawningCar(random.Next(100, 300),
random.Next(1000, 3000), GetColor(random));
                                break;
                        case nameof(DrawningSportCar):
                                // TODO выбор цветов
                                drawningCar = new DrawningSportCar(random.Next(100,
300), random.Next(1000, 3000),
                                        Color.FromArgb(random.Next(0, 256),
random.Next(0, 256), random.Next(0, 256)),
                                        Color.FromArgb(random.Next(0, 256),
random.Next(0, 256), random.Next(0, 256)),
                                        Convert.ToBoolean(random.Next(0, 2)),
Convert.ToBoolean(random.Next(0, 2)), Convert.ToBoolean(random.Next(0, 2)));
                                break;
                        default:
                                return;
                }
```

```
            if (_company + drawningCar)
            {
                MessageBox.Show("Объект добавлен");
                pictureBox.Image = _company.Show();
            }
            else
            {
                MessageBox.Show("Не удалось добавить объект");
            }
        }

        /// <summary>
        /// Получение цвета
        /// </summary>
        /// <param name="random">Генератор случайных чисел</param>
        /// <returns></returns>
        private static Color GetColor(Random random)
        {
            Color color = Color.FromArgb(random.Next(0, 256), random.Next(0,
256), random.Next(0, 256));
            ColorDialog dialog = new();
            if (dialog.ShowDialog() == DialogResult.OK)
            {
                color = dialog.Color;
            }

            return color;
        }

        /// <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) || _company ==
null)
            {
                return;
            }

            if (MessageBox.Show("Удалить объект?", "Удаление",
MessageBoxButtons.YesNo, MessageBoxIcon.Question) != DialogResult.Yes)
            {
                return;
            }

            int pos = Convert.ToInt32(maskedTextBoxPosition.Text);
            if (_company - pos)
            {
                MessageBox.Show("Объект удален");
                pictureBox.Image = _company.Show();
            }
            else
            {
                MessageBox.Show("Не удалось удалить объект");
            }
        }

        /// <summary>
        /// Передача объекта в другую форму
        /// </summary>
```

```
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void ButtonGoToCheck_Click(object sender, EventArgs e)
        {
                if (_company == null)
                {
                        return;
                }

                DrawningCar? car = null;
                int counter = 100;
                while (car == null)
                {
                        car = _company.GetRandomObject();
                        counter--;
                        if (counter <= 0)
                        {
                                break;
                        }
                }

                if (car == null)
                {
                        return;
                }

                FormSportCar form = new()
                {
                        SetCar = car
                };
                form.ShowDialog();
        }

        /// <summary>
        /// Перерисовка коллекции
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void ButtonRefresh_Click(object sender, EventArgs e)
        {
                if (_company == null)
                {
                        return;
                }

                pictureBox.Image = _company.Show();
        }

        /// <summary>
        /// Добавление коллекции
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void ButtonCollectionAdd_Click(object sender, EventArgs e)
        {
                if (string.IsNullOrEmpty(textBoxCollectionName.Text) ||
(!radioButtonList.Checked && !radioButtonMassive.Checked))
                {
                        MessageBox.Show("Не все данные заполнены", "Ошибка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                        return;
                }
```

```
            CollectionType collectionType = CollectionType.None;
            if (radioButtonMassive.Checked)
            {
                  collectionType = CollectionType.Massive;
            }
            else if (radioButtonList.Checked)
            {
                  collectionType = CollectionType.List;
            }

            _storageCollection.AddCollection(textBoxCollectionName.Text,
collectionType);
            RerfreshListBoxItems();
      }

      /// <summary>
      /// Удаление коллекции
      /// </summary>
      /// <param name="sender"></param>
      /// <param name="e"></param>
      private void ButtonCollectionDel_Click(object sender, EventArgs e)
      {
            // TODO прописать логику удаления элемента из коллекции
            // нужно убедиться, что есть выбранная коллекция
            // спросить у пользователя через MessageBox, что он подтверждает, что
хочет удалить запись
            // удалить и обновить ListBox
      }

      /// <summary>
      /// Обновление списка в listBoxCollection
      /// </summary>
      private void RerfreshListBoxItems()
      {
            listBoxCollection.Items.Clear();
            for (int i = 0; i < _storageCollection.Keys?.Count; ++i)
            {
                  string? colName = _storageCollection.Keys?[i];
                  if (!string.IsNullOrEmpty(colName))
                  {
                        listBoxCollection.Items.Add(colName);
                  }
            }

      }

      /// <summary>
      /// Создание компании
      /// </summary>
      /// <param name="sender"></param>
      /// <param name="e"></param>
      private void ButtonCreateCompany_Click(object sender, EventArgs e)
      {
            if (listBoxCollection.SelectedIndex < 0 ||
listBoxCollection.SelectedItem == null)
            {
                  MessageBox.Show("Коллекция не выбрана");
                  return;
            }

            ICollectionGenericObjects<DrawningCar>? collection =
_storageCollection[listBoxCollection.SelectedItem.ToString() ?? string.Empty];
            if (collection == null)
```

```
            {
                MessageBox.Show("Коллекция не проинициализирована");
                return;
            }

            switch (comboBoxSelectorCompany.Text)
            {
                case "Хранилище":
                    _company = new CarSharingService(pictureBox.Width,
pictureBox.Height, collection);
                    break;
            }

            panelCompanyTools.Enabled = true;
            RerfreshListBoxItems();
        }
}
```

Листинг 4.4 – Измененный код формы FormCarCollection

Тестирование

Запустим проект, попытаемся создать коллекцию без имени, получим сообщение с ошибкой (рисунок 4.2).

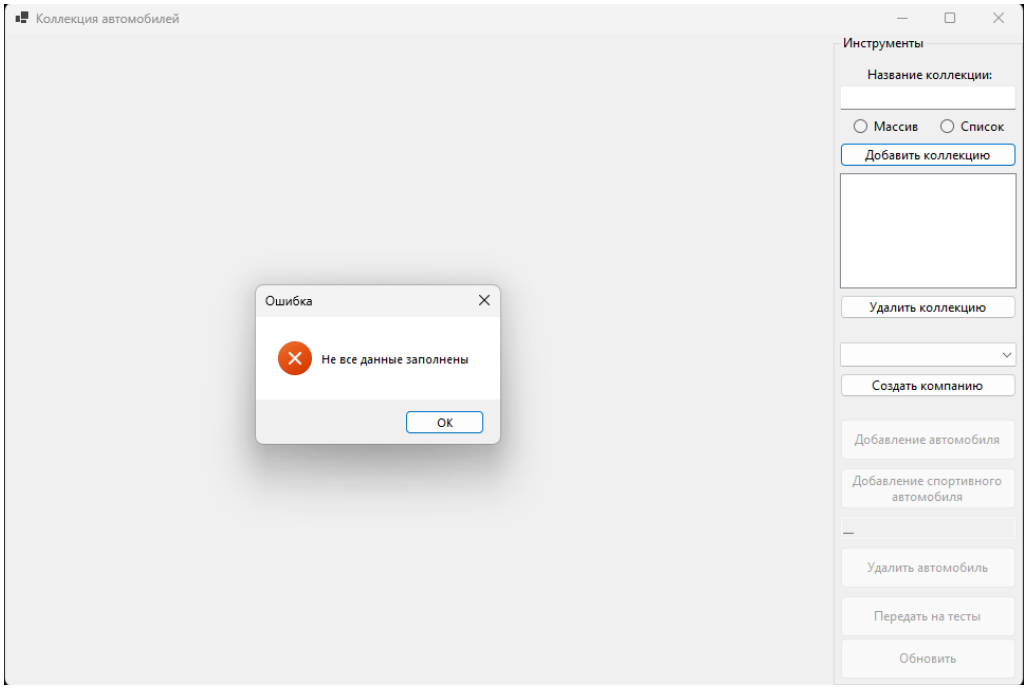


Рисунок 4.2 – Попытка создания коллекции без имени

Создадим коллекцию с массивом (рисунок 4.3).

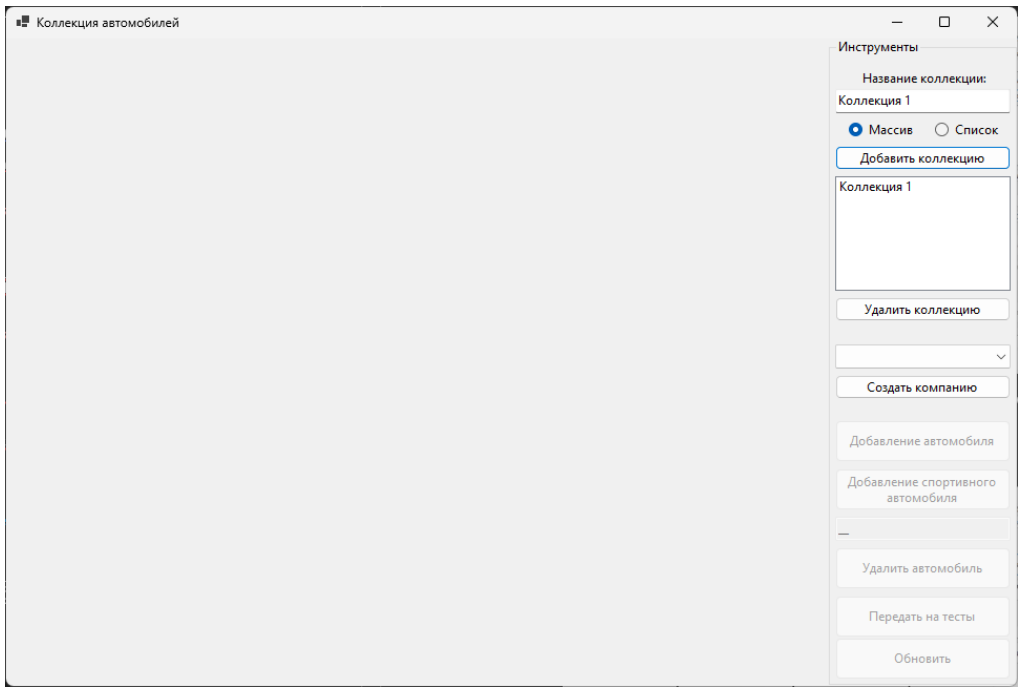


Рисунок 4.3 – Добавление коллекции на массиве

Попытаемся добавить еще раз, с тем же именем (ничего не должно произойти). Выберем компанию («Хранилище») и добавим пару объектов туда (рисунок 4.4).

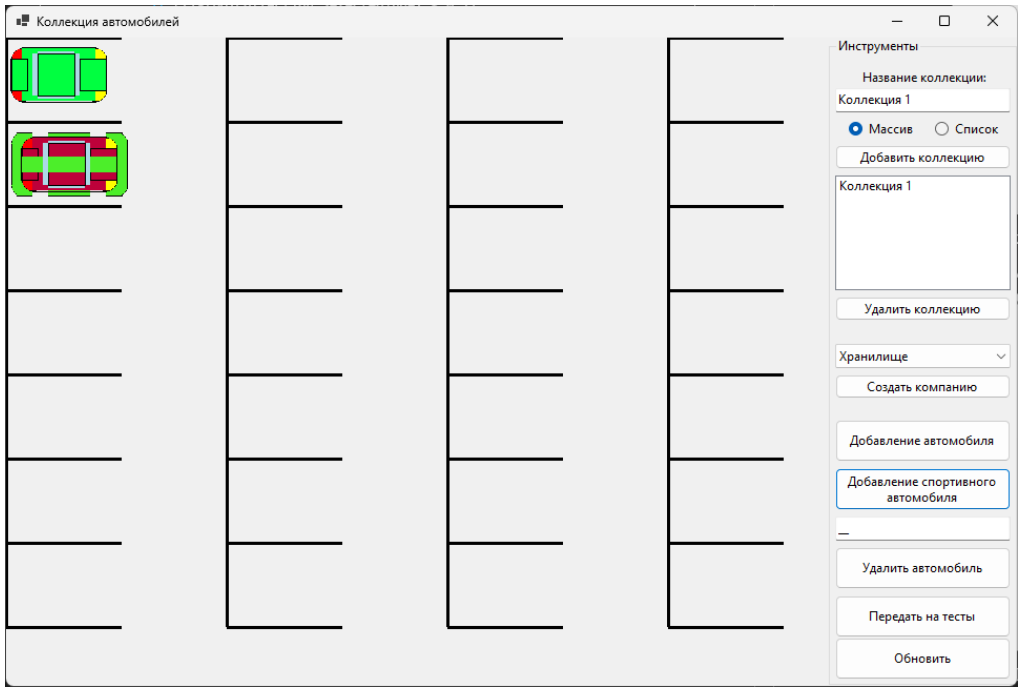


Рисунок 4.4 – Хранилище с объектами

Добавим коллекцию на списках, также создадим компанию и добавим несколько объектов (рисунок 4.5).

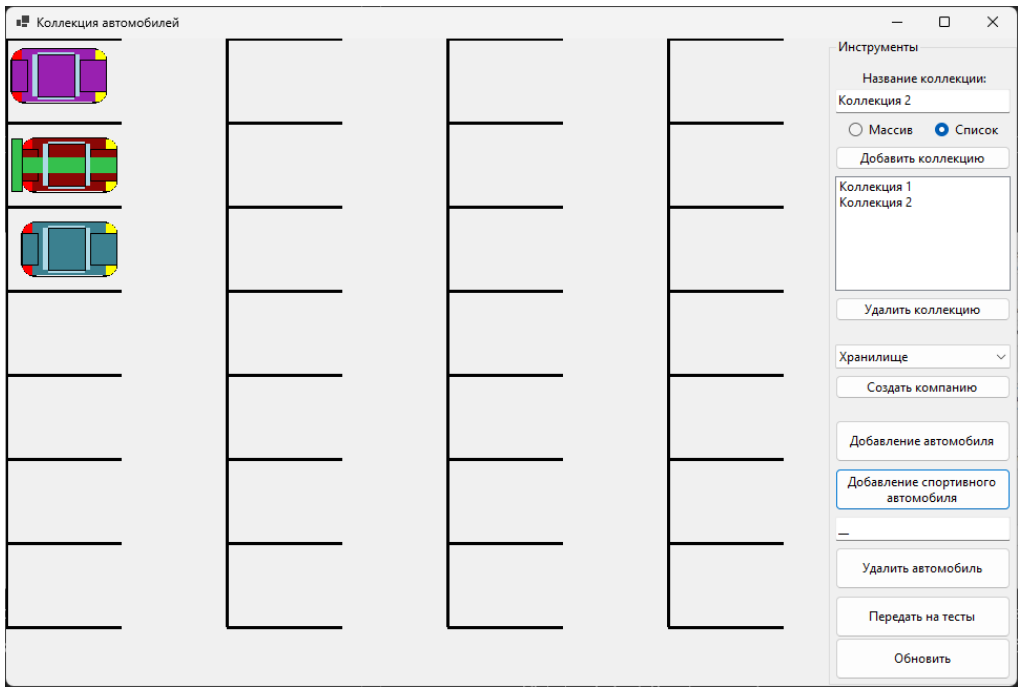


Рисунок 4.5 – Другое хранилище с объектами

Теперь снова создадим компанию с коллекцией на массивах и просто обновим форму (рисунок 4.6).

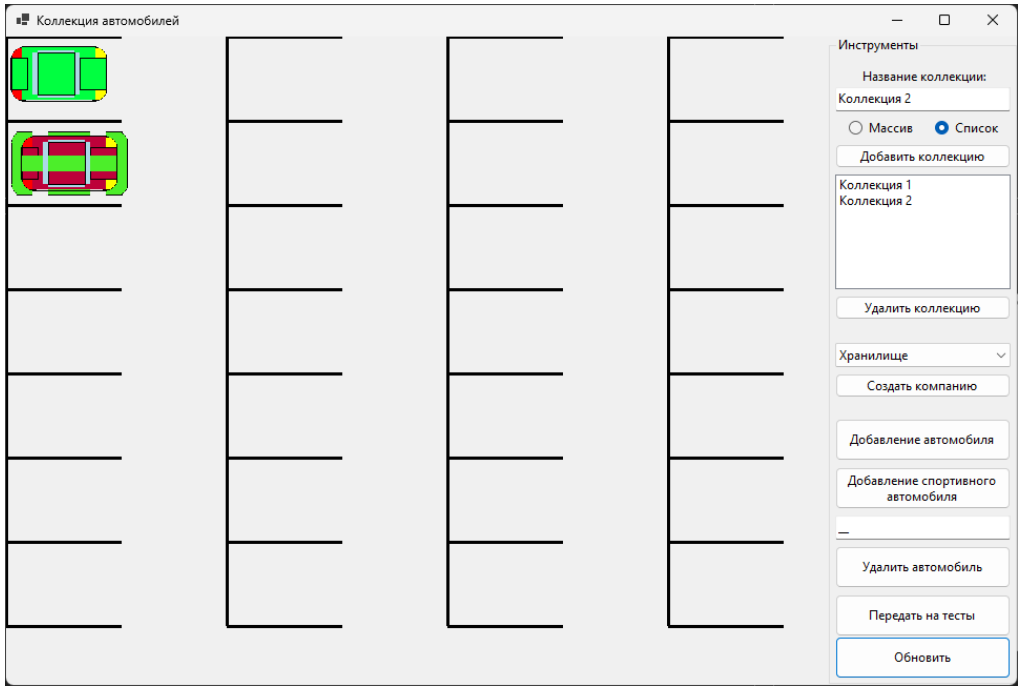


Рисунок 4.6 – Новая компания на массиве

А теперь создадим компанию с коллекций на списках и обновим его (рисунок 4.7).

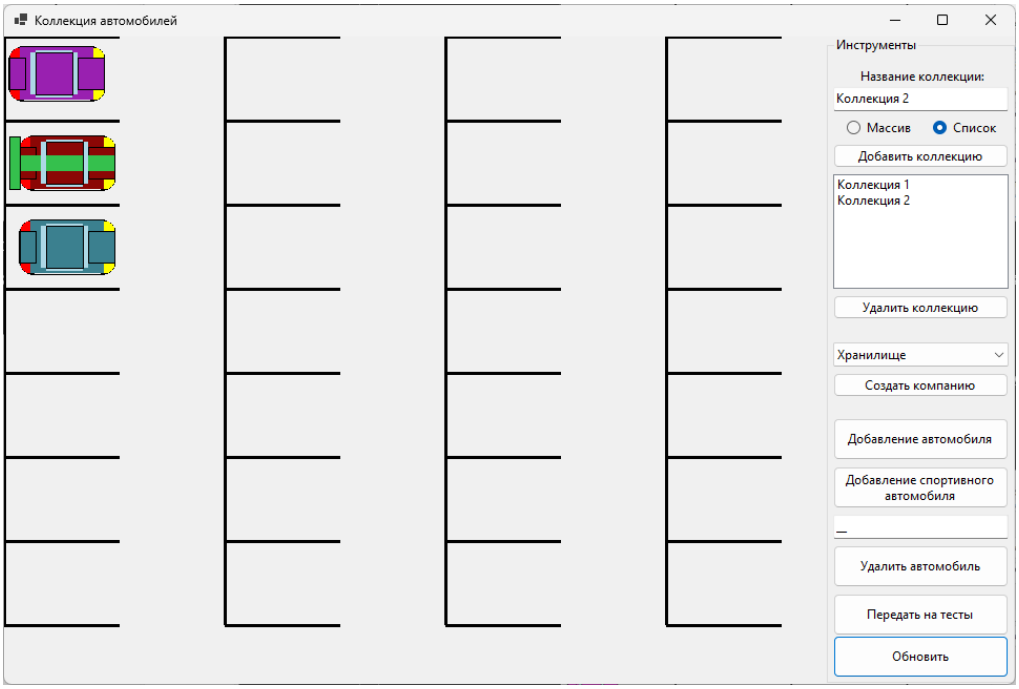


Рисунок 4.7 – Новая компания на списке

Удалить коллекцию из списка (рисунок 4.8).

Рисунок 4.8 – Удаление коллекции

Таким образом все поставленные задачи выполнены. Работа готова.

Требования

1. Платформа для проектов – .Net версии 6.0

2. Название проекта форм, классов, свойств классов должно соответствовать логике задания.
3. Все элементы форм (заголовки форм, текст в label и т.д.) должны иметь подписи на одном языке (или все русским, или все английским).

   ==========

4. Дописать логику в методах вставки, удаления и получения элемента класса ListGenericObjects.
5. Дописать логику в методах вставки, удаления и получения элемента класса-хранилища.
6. Дописать на форме логику удаления записи из хранилища.

Проверка работы и порядок сдачи

1. Попытаться создать коллекцию без имени.
2. Создать коллекцию на массиве.
3. Создать коллекцию на списке.
4. Создать объект абстрактного класса на основе массива.
5. Добавить 2 объекта.
6. Создать объект абстрактного класса на основе списка.
7. Добавить 2 объекта.
8. Создать объект абстрактного класса на основе массива.
9. Обновить форму.
10. Создать объект абстрактного класса на основе списка.
11. Обновить форму.
12. Удалить одну из коллекций из хранилища.

Контрольные вопросы к базовой части

1. Перечислить методы и свойства словаря, которые используются в решении.

2. Перечислить методы и свойства List, которые используются в решении.
3. Показать реализацию индексатора и место, где происходит его вызов.

Варианты

Вар.	**Продвинутый / простой объект**	**Реализация абстрактного класса**	**Коллекция усложнения**
1.	Бомбардировщик / Военный самолет	Ангар	LinkedList
2.	Самосвал / Грузовик	Автопарк	Stack
3.	Бульдозер / Гусеничная машина	Гараж	Queue
4.	Электровоз / Локомотив	Депо	LinkedList
5.	Теплоход / Корабль	Пристань	Stack
6.	Танк / Бронированная машина	База	Queue
7.	Аэробус / Самолет	Аэродром	LinkedList
8.	Линкор / Военный корабль	Доки	Stack
9.	Автобус двухэтажный / Автобус	Автовокзал	Queue
10.	Катер / Лодка	Гавань	LinkedList
11.	Истребитель / Военный самолет	Ангар	Stack
12.	Бензовоз / Грузовик	Автопарк	Queue
13.	Экскаватор / Гусеничная машина	Гараж	LinkedList
14.	Тепловоз / Локомотив	Депо	Stack

Вар.	Продвинутый / простой объект	Реализация абстрактного класса	Коллекция усложнения
15.	Контейнеровоз / Корабль	Пристань	Queue
16.	Самоходная арт. установка / Бронированная машина	База	LinkedList
17.	Самолет с радаром / Самолет	Аэродром	Stack
18.	Крейсер / Военный корабль	Доки	Queue
19.	Автобус с гармошкой / Автобус	Автовокзал	LinkedList
20.	Катамаран / Лодка	Гавань	Stack
21.	Штурмовик / Военный самолет	Ангар	Queue
22.	Подметально-уборочная машина / Грузовик	Автопарк	LinkedList
23.	Подъемный кран / Гусеничная машина	Гараж	Stack
24.	Монорельс / Локомотив	Депо	Queue
25.	Лайнер / Корабль	Пристань	LinkedList
26.	Зенитная установка / Бронированная машина	База	Stack
27.	Гидросамолет / Самолет	Аэродром	Queue
28.	Авианосец / Военный корабль	Доки	LinkedList
29.	Троллейбус / Автобус	Автовокзал	Stack
30.	Парусник / Лодка	Гавань	Queue

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. *В классах-сущностях задать возможность сменить цвета.*
    б. *Создать новую форму для создания объекта с возможностью редактирования его характеристик. 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 с формы по работе с коллекцией.*

Проектирование

Потребуется сделать 2 действия (по поводу 1 пункта, там все просто, так что на нем останавливаться не будем):

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).

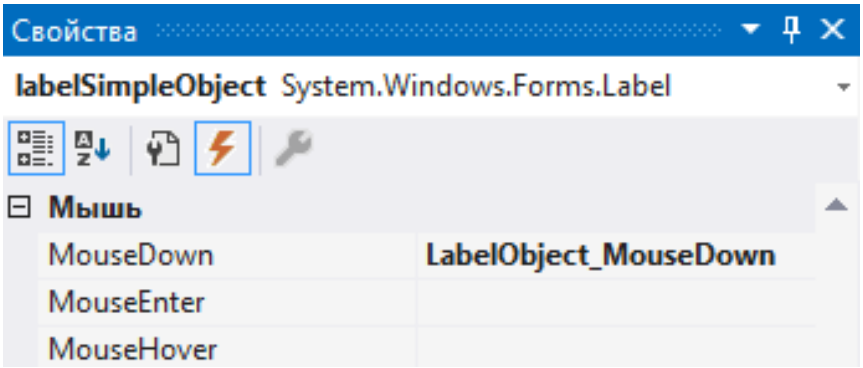


Рисунок 5.1 – Событие нажатия кнопки мышки

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.

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).

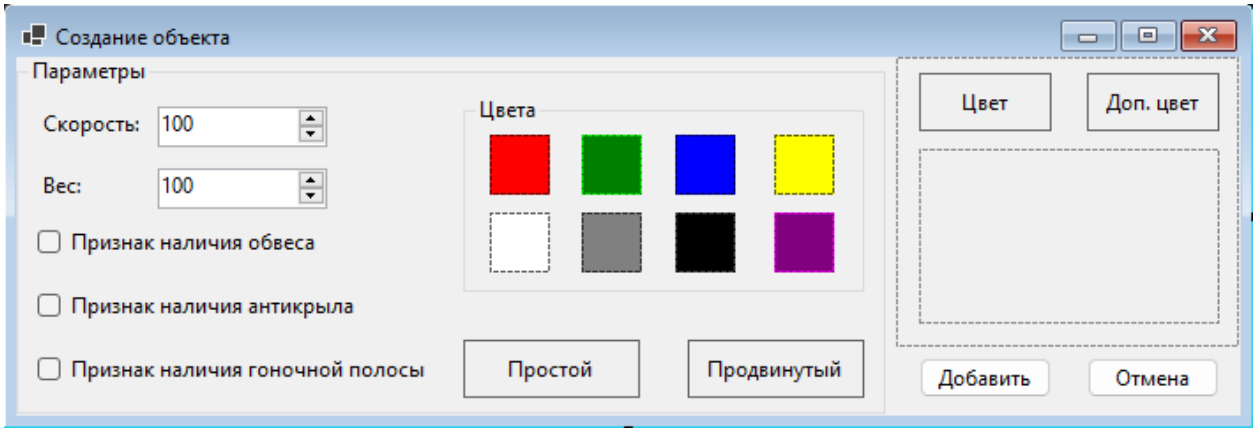


Рисунок 5.2 – Форма создания объекта.

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

```
using ProjectSportCar.Drawnings;

namespace ProjectSportCar;

/// <summary>
/// Форма конфигурации объекта
/// </summary>
public partial class FormCarConfig : Form
{
	/// <summary>
	/// Объект - прорисовка автомобиля
	/// </summary>
	private DrawningCar? _car = null;

	/// <summary>
	/// Конструктор
	/// </summary>
	public FormCarConfig()
	{
		InitializeComponent();
	}

	/// <summary>
	/// Прорисовка объекта
	/// </summary>
	private void DrawObject()
	{
		Bitmap bmp = new(pictureBoxObject.Width, pictureBoxObject.Height);
		Graphics gr = Graphics.FromImage(bmp);
		_car?.SetPictureSize(pictureBoxObject.Width,
pictureBoxObject.Height);
		_car?.SetPosition(15, 15);
		_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 ??
string.Empty, 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)
	{
```

```
            e.Effect = e.Data?.GetDataPresent(DataFormats.Text) ?? false ?
DragDropEffects.Copy : 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,
(double)numericUpDownWeight.Value, Color.White);
                        break;
                  case "labelModifiedObject":
                        _car = new
DrawningSportCar((int)numericUpDownSpeed.Value, (double)numericUpDownWeight.Value,
Color.White,
                              Color.Black, checkBoxBodyKit.Checked,
checkBoxWing.Checked, checkBoxSportLine.Checked);
                        break;
            }

            DrawObject();
      }
}
```

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).

```
using ProjectSportCar.Drawnings;

namespace ProjectSportCar;

/// <summary>
/// Делегат передачи объекта класса-прорисовки
/// </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).

```
using ProjectSportCar.Drawnings;

namespace ProjectSportCar;

/// <summary>
/// Форма конфигурации объекта
/// </summary>
public partial class FormCarConfig : Form
{
	…

	/// <summary>
	/// Событие для передачи объекта
	/// </summary>
	private event CarDelegate? CarDelegate;

	…

	/// <summary>
	/// Привязка внешнего метода к событию
	/// </summary>
	/// <param name="carDelegate"></param>
	public void AddEvent(CarDelegate carDelegate)
	{
		CarDelegate += carDelegate;
	}

	…

	/// <summary>
	/// Передача объекта
	/// </summary>
	/// <param name="sender"></param>
	/// <param name="e"></param>
	private void ButtonAdd_Click(object sender, EventArgs e)
	{
		if (_car != null)
		{
			CarDelegate?.Invoke(_car);
			Close();
```

```
            }
        }
}
```

Листинг 5.3 – Реализация события в форме FormCarConfig

Итоговый код формы FormCarConfig следующий (листинг 5.4).

```
using ProjectSportCar.Drawnings;

namespace ProjectSportCar;

/// <summary>
/// Форма конфигурации объекта
/// </summary>
public partial class FormCarConfig : Form
{
        /// <summary>
        /// Объект - прорисовка автомобиля
        /// </summary>
        private DrawningCar? _car;

        /// <summary>
        /// Событие для передачи объекта
        /// </summary>
        private event CarDelegate? CarDelegate;

        /// <summary>
        /// Конструктор
        /// </summary>
        public FormCarConfig()
        {
                panelRed.MouseDown += Panel_MouseDown;
                panelGreen.MouseDown += Panel_MouseDown;
                panelBlue.MouseDown += Panel_MouseDown;
                panelYellow.MouseDown += Panel_MouseDown;
                panelWhite.MouseDown += Panel_MouseDown;
                panelGray.MouseDown += Panel_MouseDown;
                panelBlack.MouseDown += Panel_MouseDown;
                panelPurple.MouseDown += Panel_MouseDown;

                // TODO buttonCancel.Click привязать анонимный метод через lambda с
закрытием формы
                InitializeComponent();
        }

        /// <summary>
        /// Привязка внешнего метода к событию
        /// </summary>
        /// <param name="carDelegate"></param>
        public void AddEvent(CarDelegate carDelegate)
        {
                CarDelegate += carDelegate;
        }

        /// <summary>
        /// Прорисовка объекта
        /// </summary>
        private void DrawObject()
        {
                Bitmap bmp = new(pictureBoxObject.Width, pictureBoxObject.Height);
                Graphics gr = Graphics.FromImage(bmp);
```

```
            _car?.SetPictureSize(pictureBoxObject.Width,
pictureBoxObject.Height);
            _car?.SetPosition(15, 15);
            _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 ??
string.Empty, 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)
      {
            e.Effect = e.Data?.GetDataPresent(DataFormats.Text) ?? false ?
DragDropEffects.Copy : 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,
(double)numericUpDownWeight.Value, Color.White);
                        break;
                  case "labelModifiedObject":
                        _car = new
DrawningSportCar((int)numericUpDownSpeed.Value, (double)numericUpDownWeight.Value,
Color.White,
                              Color.Black, checkBoxBodyKit.Checked,
checkBoxWing.Checked, checkBoxSportLine.Checked);
                        break;
            }

            DrawObject();
      }

      /// <summary>
      /// Передаем информацию при нажатии на Panel
      /// </summary>
      /// <param name="sender"></param>
      /// <param name="e"></param>
      private void Panel_MouseDown(object? sender, MouseEventArgs e)
      {
            // TODO отправка цвета в Drag&Drop
      }
```

```
        // TODO Реализовать логику смены цветов: основного и дополнительного (для
продвинутого объекта)

        /// <summary>
        /// Передача объекта
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void ButtonAdd_Click(object sender, EventArgs e)
        {
                if (_car != null)
                {
                        CarDelegate?.Invoke(_car);
                        Close();
                }
        }
}
```

Листинг 5.4 – Логика формы FormCarConfig

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

```
        /// <summary>
        /// Добавление автомобиля
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void ButtonAddCar_Click(object sender, EventArgs e)
        {
                FormCarConfig form = new();
                // TODO передать метод
                form.Show();
        }

        /// <summary>
        /// Добавление автомобиля в коллекцию
        /// </summary>
        /// <param name="car"></param>
        private void SetCar(DrawningCar? car)
        {
                if (_company == null || car == null)
                {
                        return;
                }

                if (_company + car)
                {
                        MessageBox.Show("Объект добавлен");
                        pictureBox.Image = _company.Show();
                }
                else
                {
                        MessageBox.Show("Не удалось добавить объект");
                }
        }
```

Листинг 5.5 – Новая логика метода добавления объекта в форме FormMapWithSetCars

Тестирование

Запустим проект, создаем коллекцию, компанию и вызываем форму создания объекта (рисунок 5.3).

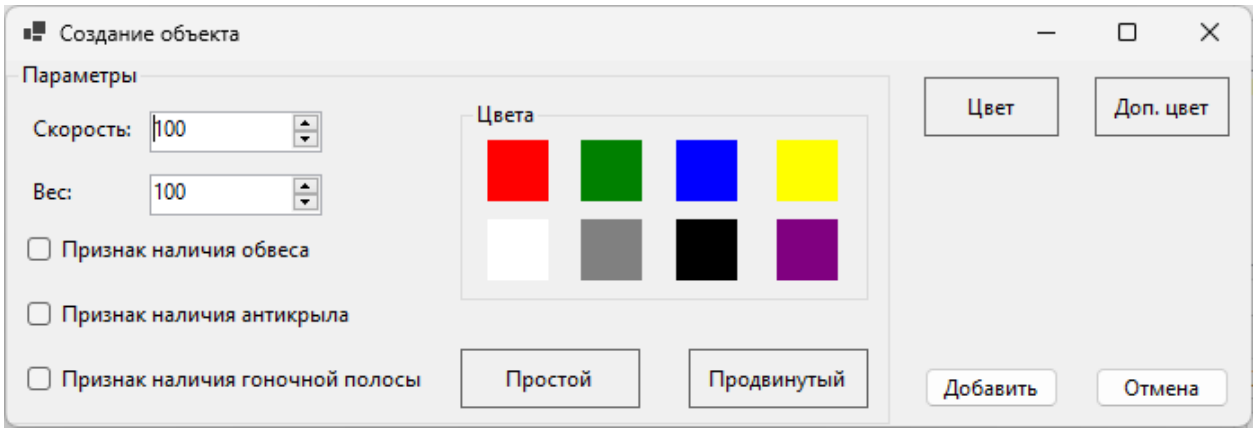


Рисунок 5.3 – Форма создания объекта

Перетаскиванием выбираем «простой» объект (рисунок 5.4).

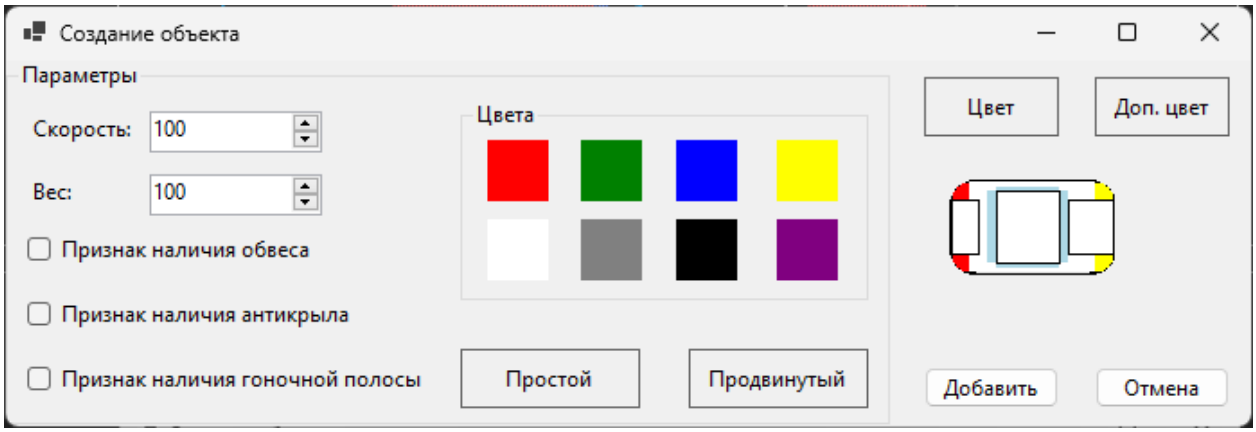


Рисунок 5.4 – Создание «простого» объекта

Задаем ему цвет (рисунок 5.5).

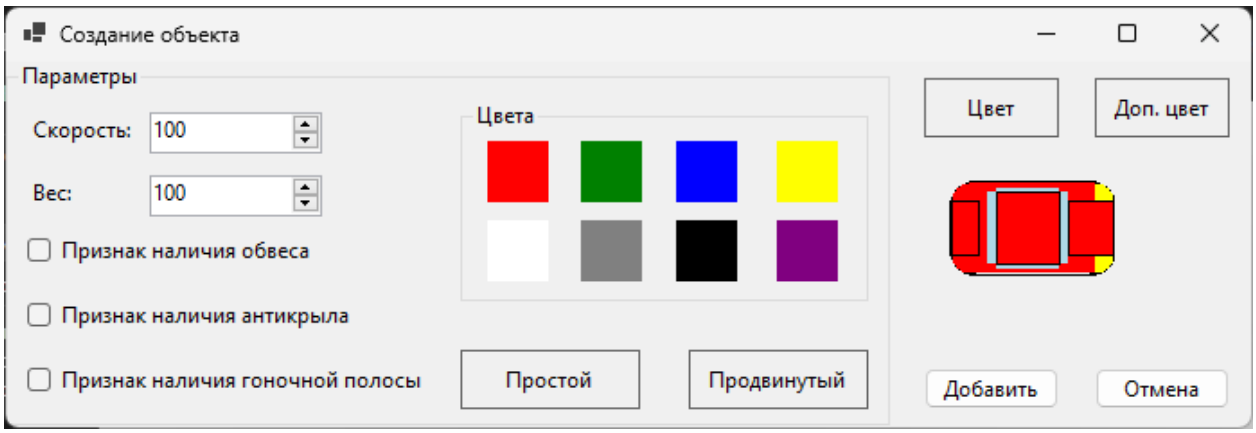


Рисунок 5.5 – Выбор цвета для объекта

Пытаемся добавить «Доп. цвет». Ничего не должно произойти. Нажимаем кнопку «Добавить» и объект должен появится в коллекции (рисунок 5.6).

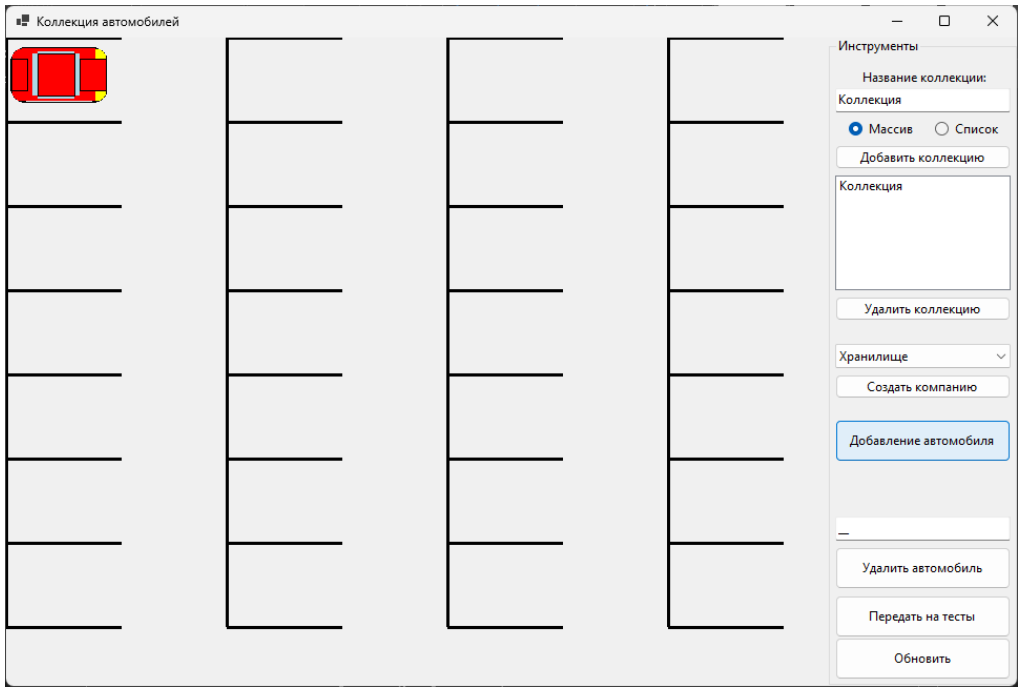


Рисунок 5.6 – Объект в коллекции

Теперь снова вызываем форму, ставим одну из опций активной и выбираем «продвинутый» объект (рисунок 5.7).

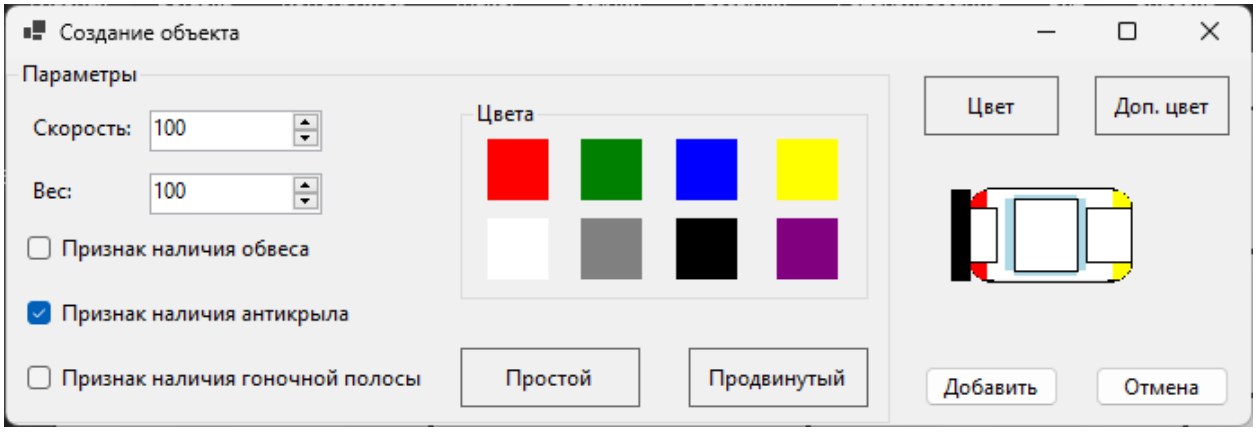


Рисунок 5.7 – «Продвинутый» объект с одной активной опцией

Выбираем все опции и снова создаем «продвинутый» объект (рисунок 5.8).

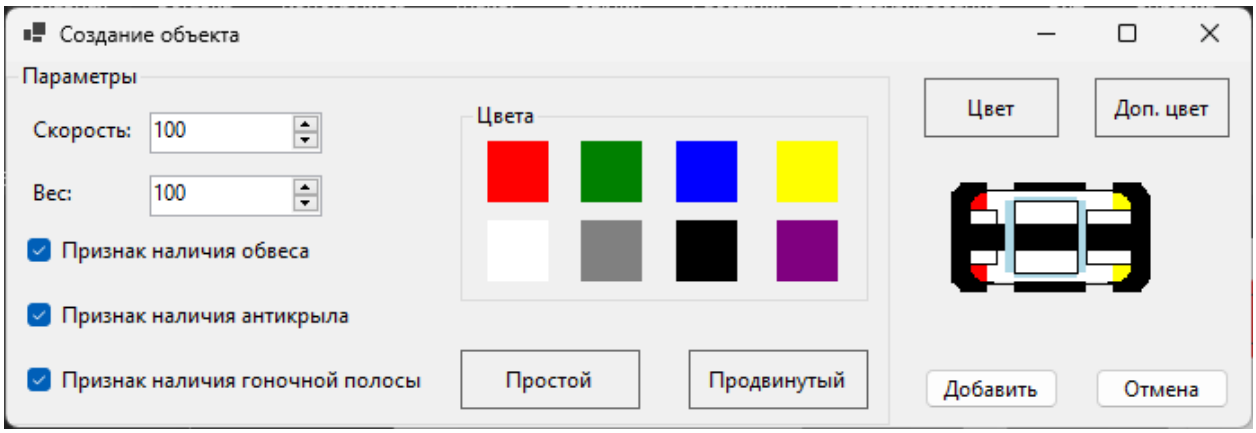


Рисунок 5.8 – «Продвинутый» объект со всеми опциями

Задаем 2 цвета, основной и дополнительный (рисунок 5.9).

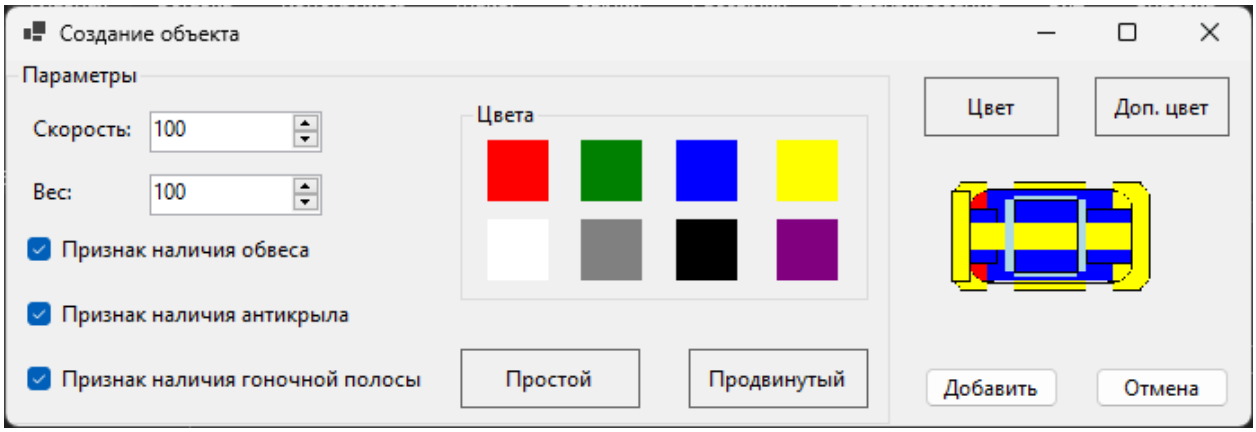


Рисунок 5.9 – «Продвинутый» объект со всеми опциями и цветами

Добавляем объект в коллекцию (рисунок 5.10).

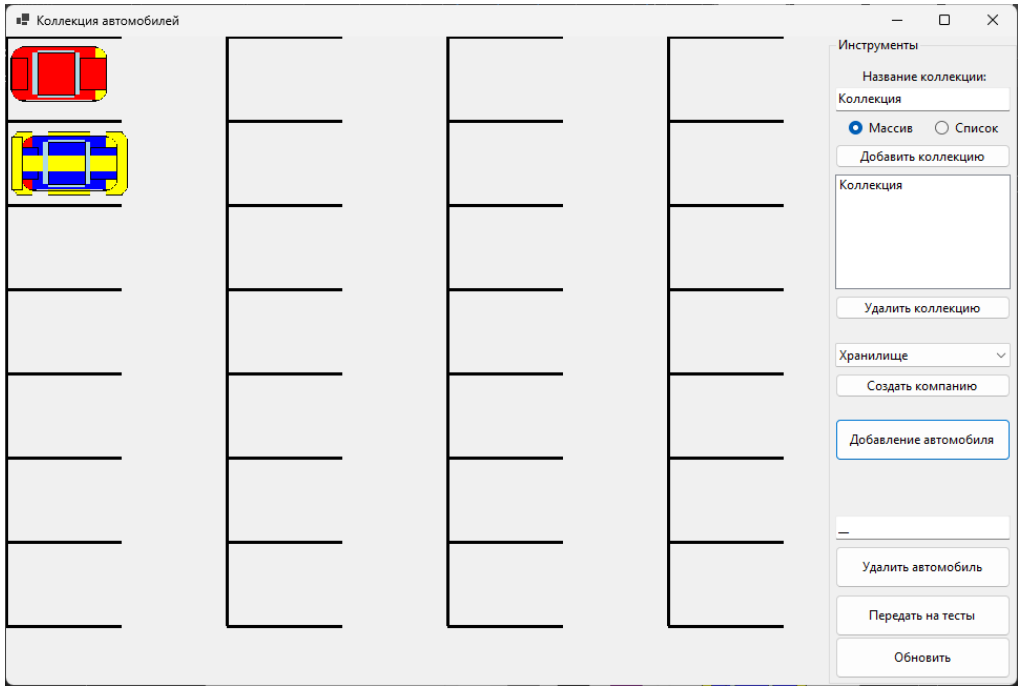


Рисунок 5.10 – Объекты в коллекции

Таким образом все поставленные задачи выполнены. Работа готова.

Требования

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

   ==========
4. В классах-сущностях смены основного и дополнительного цветов прописать в виде методов.
5. В форме конфигурации заменить использование собственного делегата на встроенный делегат. Прописать логику обработки события Click у кнопки «Отмена» с использованием лямбда-выражений. Прописать логику drag&drop для выбора цвета. Учитывать, что смена дополнительного цвета может быть только у продвинутого объекта.
6. В форме-коллекции передавать метод обработки добавления объекта в форму конфигурации.

Проверка работы и порядок сдачи

1. Вызвать новую форму.
2. Выбрать простой объект.
3. Задать ему цвет. Попытаться задать ему доп. цвет.
4. Добавить в коллекцию.
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 коллекцию.

Контрольные вопросы к базовой части

1. Как определяется, какой объект создавать, при drag&drop в форме конфигурации?
2. Как новый объект попадает в коллекцию?
3. Покажите пример лямбда-выражения в коде.

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)

Вар.	Базовый объект	«Продвинутый» объект
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)
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)

Вар.	Базовый объект	«Продвинутый» объект
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)
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:*
    а. *Реализовать сохранение и загрузку данных по всем объектам, созданным в классе-хранилище.*
    б. *Путь до файла и имя файла сохранения/загрузки указывать через диалоговое окно. Вызов диалоговых окон реализовать через пункты меню. Дополнительно применить фильтр к диалоговому окну, чтобы отображались файлы только определенного формата (*.txt).*
    в. *В интерфейсе-коллекции описать метод получения всех элементов коллекций для сохранения (в реализации делать так, чтобы каждый элемент выдавался на обработку поочередно). Получение данных для сохранения с классов-прорисовки и создание объектов при загрузке реализовать через методы расширения. Создание объектов классов-сущностей реализовать через статические методы.*

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 данные из класса-хранилища. По каждой записи слова сохранять следует следующую информацию:

- ключ;
- информацию по объекту класса-коллекции.

По объекту класса-коллекции хранить будем следующую информацию:

- тип класса-коллекции;
- максимальное количество элементов;
- информацию по набору объектов от классов прорисовки.

В классе прорисовки важным будет только информация о том, какой объект класса-сущности («простого» или «продвинутого») там используется. Координаты, по которым расположен объект не важны, ибо они определяются при выводе на форму исходя из размеров форм, куда будут выводиться.

По объекту класса-сущности «простого» типа хранить будем:

- тип
- скорость;
- вес;
- цвет.

А по объекту класс-сущности «продвинутого» типа:

- информацию по базовому классу (только изменить тип);
- 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 класса-хранилища:
  - ключ
  - тип коллекции
  - максимальное количество элементов
  - объект класса-коллекции:
    - список объектов классов прорисовки:
      - тип
      - скорость;
      - вес;
      - цвет;
      - дополнительный цвет (если есть);
      - признак наличия обвеса (если есть);
      - признак наличия антикрыла (если есть);
      - признак наличия гоночной полосы (если есть).

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 информация по содержимому коллекции. В языках программирования у строк есть замечательный метод разбиения строки на массив строк по символу-разделителю (может даже не символ, а набор символов быть). В принципе, мы можем обойтись тремя символами-разделителями, чтобы отделить данные. Первый символ поможет нам разделить ключ, информацию по классу-коллекции и набор объектов класса-коллекции, второй ключ позволит разделить записи объектов, а третий ключ – информацию по объекту «простого» или «продвинутого» типа. 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|тип коллекции|максимальное количество
записей|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.

Также в интерфейсе-коллекции потребуется прописать и изменить ряд методов и свойств.

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

Реализация

Первым делом займемся классами-сущностями. В базовом пропишем метод создания массива строк с данными по объекту (он будет виртуальный и переопределяться в классе-наследнике), а также статический метод, который позволяет на основе массива строк создавать объект (листинг 6.1).

```
namespace ProjectSportCar.Entities;

/// <summary>
/// Класс-сущность "Автомобиль"
/// </summary>
public class EntityCar
{
        /// <summary>
        /// Скорость
        /// </summary>
        public int Speed { get; private set; }

        /// <summary>
        /// Вес
        /// </summary>
        public double Weight { get; private set; }
```

```
        /// <summary>
        /// Основной цвет
        /// </summary>
        public Color BodyColor { get; private set; }

        /// <summary>
        /// Шаг перемещения автомобиля
        /// </summary>
        public double Step => Speed * 100 / Weight;

        /// <summary>
        /// Конструктор сущности
        /// </summary>
        /// <param name="speed">Скорость</param>
        /// <param name="weight">Вес автомобиля</param>
        /// <param name="bodyColor">Основной цвет</param>
        public EntityCar(int speed, double weight, Color bodyColor)
        {
                Speed = speed;
                Weight = weight;
                BodyColor = bodyColor;
        }

        /// <summary>
        /// Получение строк со значениями свойств объекта класса-сущности
        /// </summary>
        /// <returns></returns>
        public virtual string[] GetStringRepresentation()
        {
                return new[] { nameof(EntityCar), Speed.ToString(),
Weight.ToString(), BodyColor.Name };
        }

        /// <summary>
        /// Создание объекта из массива строк
        /// </summary>
        /// <param name="strs"></param>
        /// <returns></returns>
        public static EntityCar? CreateEntityCar(string[] strs)
        {
                if (strs.Length != 4 || strs[0] != nameof(EntityCar))
                {
                        return null;
                }

                return new EntityCar(Convert.ToInt32(strs[1]),
Convert.ToDouble(strs[2]), Color.FromName(strs[3]));
        }
}
```

Листинг 6.2 – Дополнения в базовом классе-сущности

Аналогичное потребуется сделать для дочернего класса-сущности.

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. При получении строки из объекта будем получать массив строк – данных по сущности и объединять их

по разделителю в единую строку. А при получении объекта из строки будем по разделителю разбивать строку на массив и пытаться создать объект сперва продвинутого типа, если не удастся, то базового типа (листинг 6.2).

```
using ProjectSportCar.Entities;

namespace ProjectSportCar.Drawnings;

/// <summary>
/// Расширение для класса EntityCar
/// </summary>
public static class ExtentionDrawningCar
{
	/// <summary>
	/// Разделитель для записи информации по объекту в файл
	/// </summary>
	private static readonly string _separatorForObject = ":";

	/// <summary>
	/// Создание объекта из строки
	/// </summary>
	/// <param name="info">Строка с данными для создания объекта</param>
	/// <returns>Объект</returns>
	public static DrawningCar? CreateDrawningCar(this string info)
	{
		string[] strs = info.Split(_separatorForObject);
		EntityCar? car = EntitySportCar.CreateEntitySportCar(strs);
		if (car != null)
		{
			return new DrawningSportCar(car);
		}

		car = EntityCar.CreateEntityCar(strs);
		if (car != null)
		{
			return new DrawningCar(car);
		}

		return null;
	}

	/// <summary>
	/// Получение данных для сохранения в файл
	/// </summary>
	/// <param name="drawningCar">Сохраняемый объект</param>
	/// <returns>Строка с данными по объекту</returns>
	public static string GetDataForSave(this DrawningCar drawningCar)
	{
		string[]? array = drawningCar?.EntityCar?.GetStringRepresentation();

		if (array == null)
		{
			return string.Empty;
		}

		return string.Join(_separatorForObject, array);
	}
}
```

Листинг 6.2 – Класс ExtentionDrawningCar

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. А самих классах прорисовки пропишем еще конструкторы, которые на вход будут принимать объект класса-сущности.

Далее для того, чтобы в классе-хранилище получать и сохранять информацию по объектам надо расширить интерфейс-коллекцию (листинг 6.3).

```
namespace ProjectSportCar.CollectionGenericObjects;

/// <summary>
/// Интерфейс описания действий для набора хранимых объектов
/// </summary>
/// <typeparam name="T">Параметр: ограничение - ссылочный тип</typeparam>
public interface ICollectionGenericObjects<T>
	where T : class
{
	/// <summary>
	/// Количество объектов в коллекции
	/// </summary>
	int Count { get; }

	/// <summary>
	/// Установка максимального количества элементов
	/// </summary>
	int MaxCount { get; set; }

	/// <summary>
	/// Добавление объекта в коллекцию
	/// </summary>
	/// <param name="obj">Добавляемый объект</param>
	/// <returns>true - вставка прошла удачно, false - вставка не
удалась</returns>
	bool Insert(T obj);

	/// <summary>
	/// Добавление объекта в коллекцию на конкретную позицию
	/// </summary>
	/// <param name="obj">Добавляемый объект</param>
	/// <param name="position">Позиция</param>
	/// <returns>true - вставка прошла удачно, false - вставка не
удалась</returns>
	bool Insert(T obj, int position);

	/// <summary>
	/// Удаление объекта из коллекции с конкретной позиции
	/// </summary>
	/// <param name="position">Позиция</param>
	/// <returns>true - удаление прошло удачно, false - удаление не
удалось</returns>
	bool Remove(int position);

	/// <summary>
```

```
        /// Получение объекта по позиции
        /// </summary>
        /// <param name="position">Позиция</param>
        /// <returns>Объект</returns>
        T? Get(int position);

        /// <summary>
        /// Получение типа коллекции
        /// </summary>
        CollectionType GetCollectionType { get; }

        /// <summary>
        /// Получение объектов коллекции по одному
        /// </summary>
        /// <returns>Поэлементый вывод элементов коллекции</returns>
        IEnumerable<T?> GetItems();
}
```

Листинг 6.3 – Доработанный интерфейс-коллекция

Соответственно, потребуются изменения в классах-наследниках (листинг 6.4).

```
namespace ProjectSportCar.CollectionGenericObjects;

/// <summary>
/// Параметризованный набор объектов
/// </summary>
/// <typeparam name="T">Параметр: ограничение - ссылочный тип</typeparam>
public class MassiveGenericObjects<T> : ICollectionGenericObjects<T>
        where T : class
{
        /// <summary>
        /// Массив объектов, которые храним
        /// </summary>
        private T?[] _collection;

        public int Count => _collection.Length;

        public int MaxCount
        {
                get
                {
                        return _collection.Length;
                }
                set
                {
                        if (value > 0)
                        {
                                if (_collection.Length > 0)
                                {
                                        Array.Resize(ref _collection, value);
                                }
                                else
                                {
                                        _collection = new T?[value];
                                }
                        }
                }
        }
        public CollectionType GetCollectionType => CollectionType.Massive;
```

```
        /// <summary>
        /// Конструктор
        /// </summary>
        public MassiveGenericObjects()
        {
                _collection = Array.Empty<T?>();
        }

        public T? Get(int position)
        {
                // TODO проверка позиции
                return _collection[position];
        }

        public bool Insert(T obj)
        {
                // TODO вставка в свободное место набора
                return false;
        }

        public bool Insert(T obj, int position)
        {
                // TODO проверка позиции
                // TODO проверка, что элемент массива по этой позиции пустой, если
нет, то
                //           ищется свободное место после этой позиции и идет вставка
туда
                //           если нет после, ищем до
                // TODO вставка
                return false;
        }

        public bool Remove(int position)
        {
                // TODO проверка позиции
                // TODO удаление объекта из массива, присвоив элементу массива
значение null
                return true;
        }

        public IEnumerable<T?> GetItems()
        {
                for (int i = 0; i < _collection.Length; ++i)
                {
                        yield return _collection[i];
                }
        }
}
```

Листинг 6.4 – Изменённый класс-коллекция на массиве

В классе-хранилище пропишем методы непосредственной записи и чтения данных из файла. Для записи и чтения будем использовать FileStream. Логика метода сохранения будет следующий: если есть файл, удаляем его, далее получаем набор всех данных в виде одной строки и после записываем в

файл первым слово «CollectionsStorage» (это чтобы потом, при чтении понимать, нужный ли файл открываем на чтение) и затем, строку с данными.

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 «CollectionsStorage». Если есть, все следующие строки – это записи словаря. Очищаем словарь и добавляем записи. Единственно что, чтобы применять методы расширения, который мы создали выше придётся тип T ограничить уже не просто от ссылочного типа, а именно от базового класса прорисовки (листинг 6.5).

```
using ProjectSportCar.Drawnings;
using System.Text;

namespace ProjectSportCar.CollectionGenericObjects;

/// <summary>
/// Класс-хранилище коллекций
/// </summary>
/// <typeparam name="T"></typeparam>
public class StorageCollection<T>
	where T : DrawningCar
{
	…

	/// <summary>
	/// Ключевое слово, с которого должен начинаться файл
	/// </summary>
	private readonly string _collectionKey = "CollectionsStorage";

	/// <summary>
	/// Разделитель для записи ключа и значения элемента словаря
	/// </summary>
	private readonly string _separatorForKeyValue = "|";

	/// <summary>
	/// Разделитель для записей коллекции данных в файл
	/// </summary>
	private readonly string _separatorItems = ";";

	…

	/// <summary>
	/// Сохранение информации по автомобилям в хранилище в файл
	/// </summary>
	/// <param name="filename">Путь и имя файла</param>
	/// <returns>true - сохранение прошло успешно, false - ошибка при
сохранении данных</returns>
	public bool SaveData(string filename)
	{
		if (_storages.Count == 0)
		{
```

```
                return false;
            }

            if (File.Exists(filename))
            {
                File.Delete(filename);
            }

            StringBuilder sb = new();

            sb.Append(_collectionKey);
            foreach (KeyValuePair<string, ICollectionGenericObjects<T>> value in
_storages)
            {
                sb.Append(Environment.NewLine);
                // не сохраняем пустые коллекции
                if (value.Value.Count == 0)
                {
                    continue;
                }

                sb.Append(value.Key);
                sb.Append(_separatorForKeyValue);
                sb.Append(value.Value.GetCollectionType);
                sb.Append(_separatorForKeyValue);
                sb.Append(value.Value.MaxCount);
                sb.Append(_separatorForKeyValue);

                foreach (T? item in value.Value.GetItems())
                {
                    string data = item?.GetDataForSave() ?? string.Empty;
                    if (string.IsNullOrEmpty(data))
                    {
                        continue;
                    }

                    sb.Append(data);
                    sb.Append(_separatorItems);
                }
            }

            using FileStream fs = new(filename, FileMode.Create);
            byte[] info = new UTF8Encoding(true).GetBytes(sb.ToString());
            fs.Write(info, 0, info.Length);
            return true;
        }

        /// <summary>
        /// Загрузка информации по автомобилям в хранилище из файла
        /// </summary>
        /// <param name="filename">Путь и имя файла</param>
        /// <returns>true - загрузка прошла успешно, false - ошибка при загрузке
данных</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);
                    }
                }

                string[] strs = bufferTextFromFile.Split(new char[] { '\n', '\r' },
StringSplitOptions.RemoveEmptyEntries);
                if (strs == null || strs.Length == 0)
                {
                    return false;
                }

                if (!strs[0].Equals(_collectionKey))
                {
                    //если нет такой записи, то это не те данные
                    return false;
                }

                _storages.Clear();
                foreach (string data in strs)
                {
                    string[] record = data.Split(_separatorForKeyValue,
StringSplitOptions.RemoveEmptyEntries);
                    if (record.Length != 4)
                    {
                        continue;
                    }

                    CollectionType collectionType =
(CollectionType)Enum.Parse(typeof(CollectionType), record[1]);
                    ICollectionGenericObjects<T>? collection =
StorageCollection<T>.CreateCollection(collectionType);
                    if (collection == null)
                    {
                        return false;
                    }

                    collection.MaxCount = Convert.ToInt32(record[2]);

                    string[] set = record[3].Split(_separatorItems,
StringSplitOptions.RemoveEmptyEntries);
                    foreach (string elem in set)
                    {
                        if (elem?.CreateDrawningCar() is T car)
                        {
                            if (!collection.Insert(car))
                            {
                                return false;
                            }
                        }
                    }

                    _storages.Add(record[0], collection);
                }

                return true;
            }

            /// <summary>
            /// Создание коллекции по типу
```

```
        /// </summary>
        /// <param name="collectionType"></param>
        /// <returns></returns>
        private static ICollectionGenericObjects<T>?
CreateCollection(CollectionType collectionType)
        {
                return collectionType switch
                {
                        CollectionType.Massive => new MassiveGenericObjects<T>(),
                        CollectionType.List => new ListGenericObjects<T>(),
                        _ => null,
                };
        }
}
```

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).

```
using ProjectSportCar.CollectionGenericObjects;
using ProjectSportCar.Drawnings;

namespace ProjectSportCar;

/// <summary>
/// Форма работы с компанией и ее коллекцией
/// </summary>
public partial class FormCarCollection : 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 (_storageCollection.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

Тестирование

Запустим проект, создаем коллекцию на массиве, компанию и добавляем туда несколько объектов разного типа и обязательно разных цветов (рисунок 6.1).

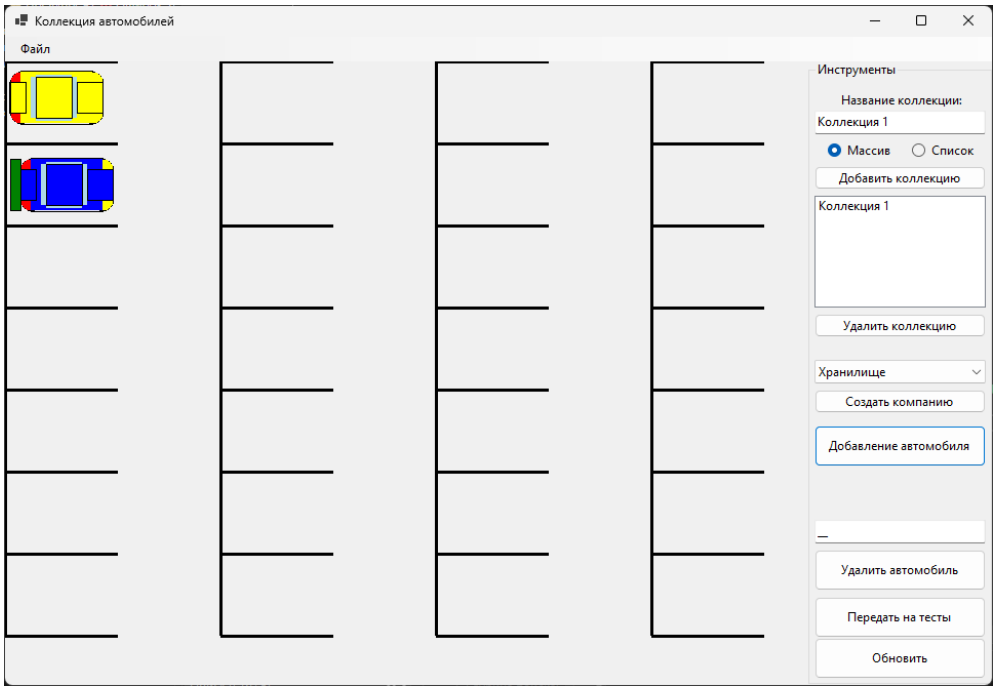


Рисунок 6.1 – Первая коллекция на массиве

Создаем вторую коллекцию на списке, также создаем компанию и добавляем туда несколько объектов разного типа и обязательно разных цветов (рисунок 6.2).

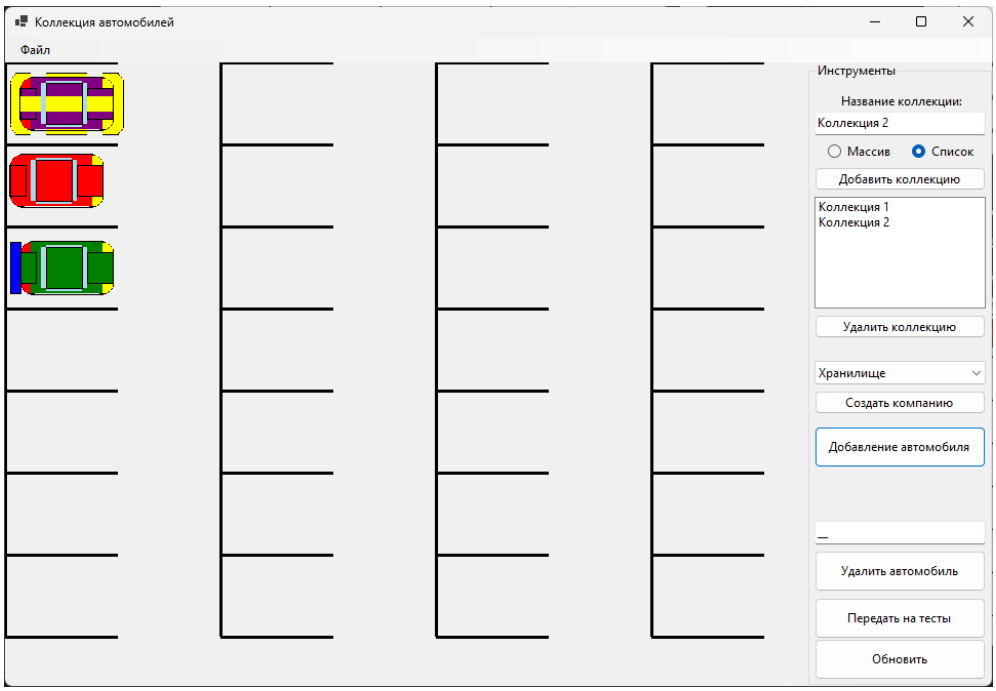


Рисунок 6.2 – Вторая коллекция на списке

Сохраняем через пункт меню в файл (рисунок 6.3).

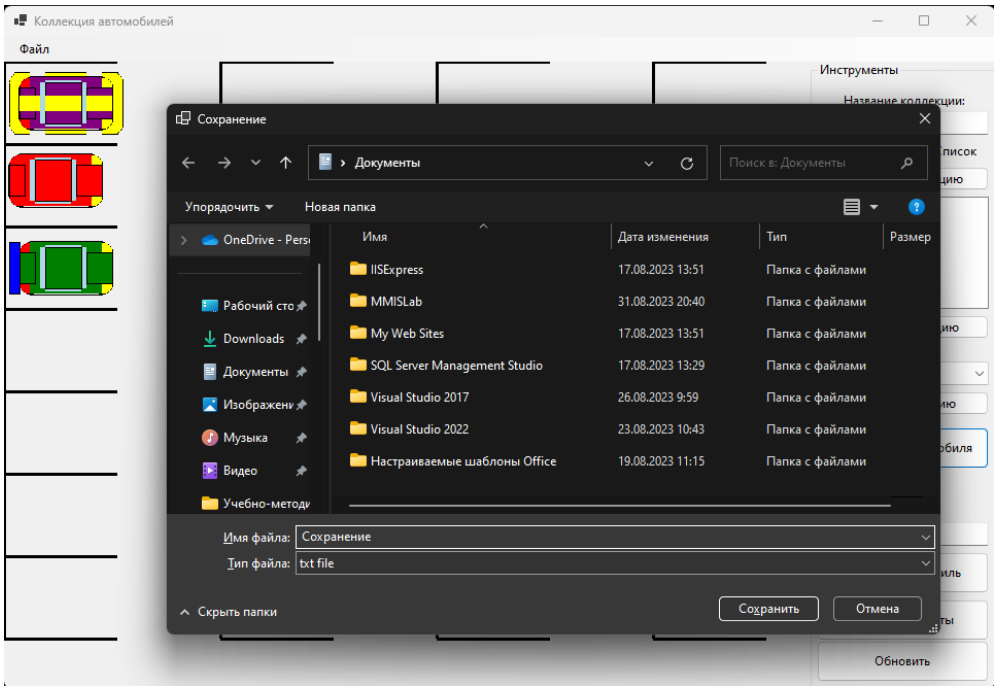


Рисунок 6.3 – Сохранение в файл

Можно посмотреть результат в файле (рисунок 6.4).

```
CollectionsStorage
Коллекция 1|Massive|28|EntityCar:100:100:Yellow;EntitySportCar:100:100:Blue:Green:False:True:False;
Коллекция 2|List|28|EntitySportCar:100:100:Purple:Yellow:True:False:True;EntityCar:100:100:Red;EntitySportCar:100:100:Green:Blue:False:True:False;
```

Рисунок 6.4 – Сохраненные данные в файле

Закрываем приложение, запускаем заново, загружаем данные из файла, получаем 2 коллекции (рисунок 6.5).

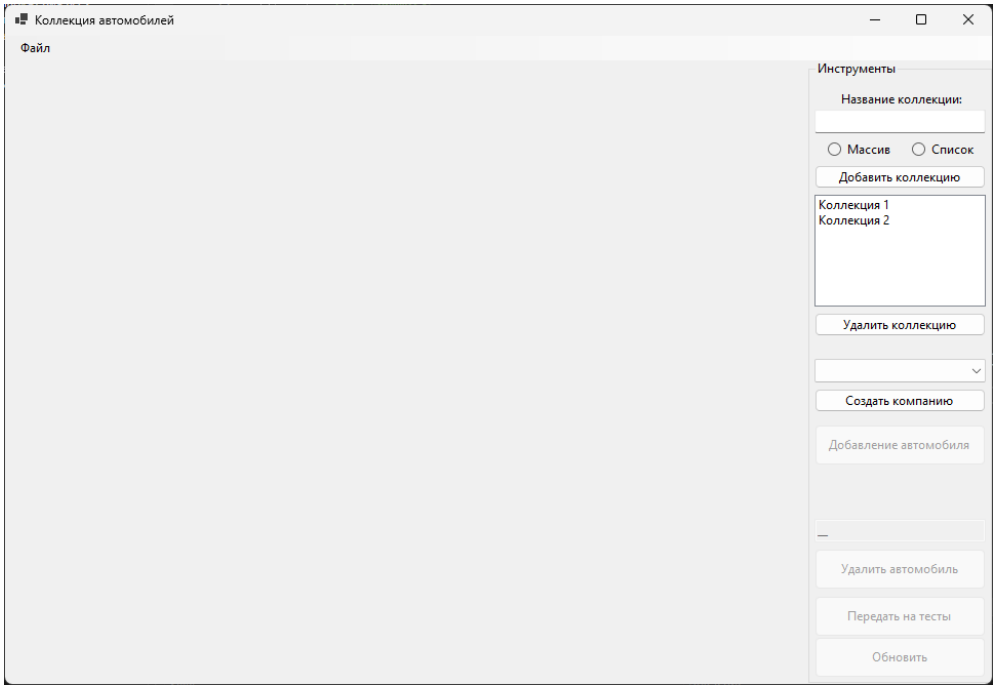


Рисунок 6.5 – Загрузка из файла

Выбираем коллекцию, создаем компанию, обновляем и убеждаемся, что все загружено корректно (рисунок 6.6).

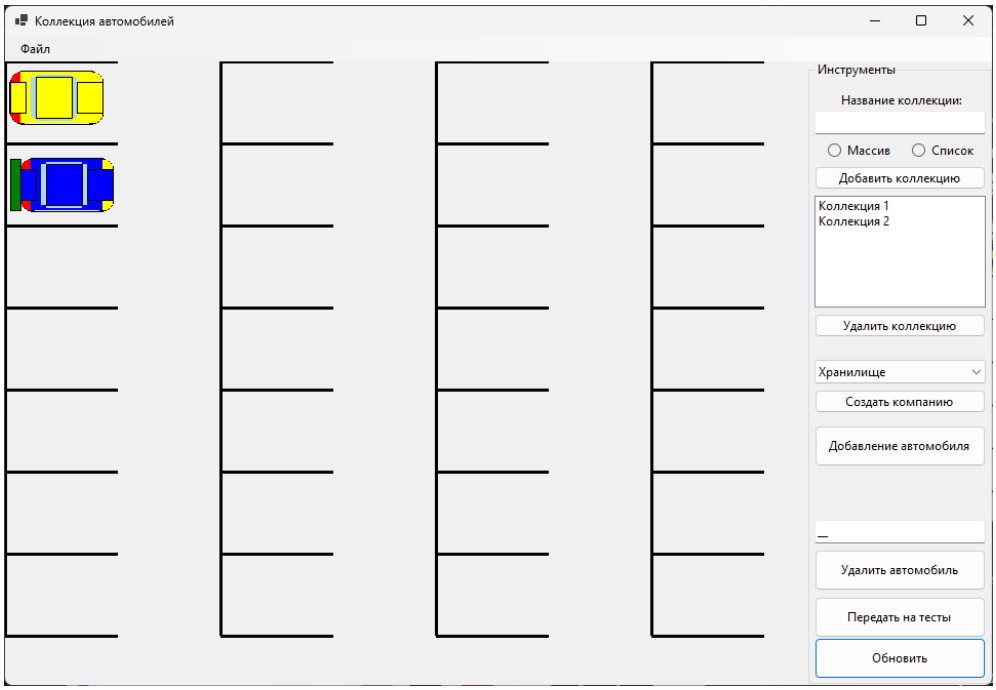


Рисунок 6.6 – Загруженная коллекция на массиве

Тоже самое проделываем со второй коллекцией (рисунок 6.7).

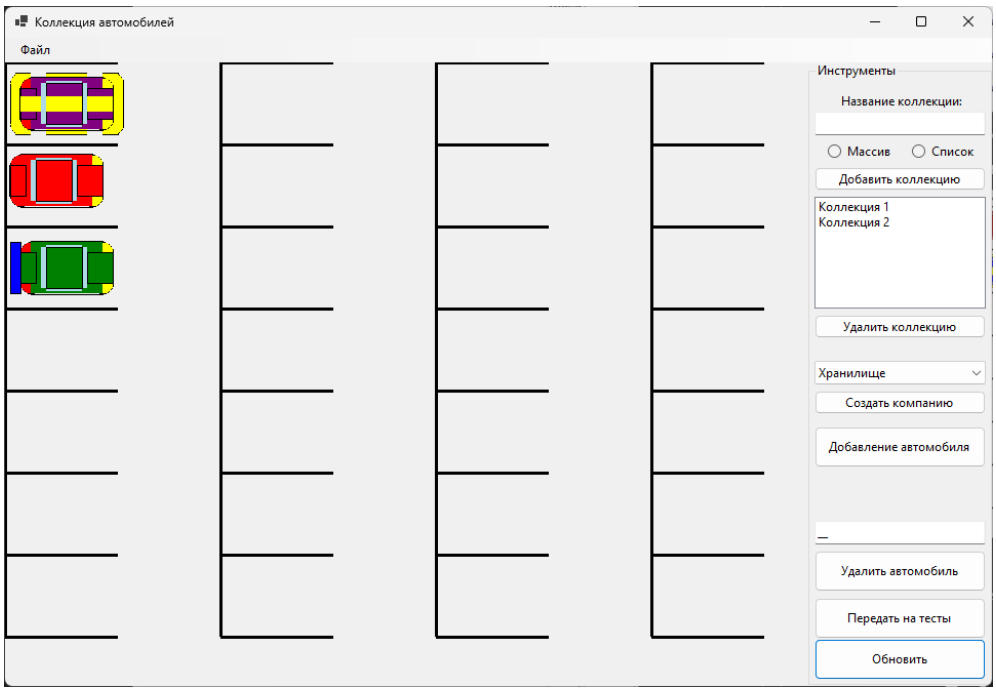


Рисунок 6.7 – Загруженная коллекция на списке

Таким образом все поставленные задачи выполнены. Работа готова.

Требования

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

   ==========

4. В дочернем классе-сущности перегрузить метод получения строк по данным и создать метод создания объекта по массиву строк. Доработать классы-прорисовки конструкторами, принимающими готовые объекты класса-сущности. Доработать логику класса-коллекции на списках.
5. В классе-хранилище. При сохранении данных использовать StreamWriter. Сохранение делать поэлементно, а не разом в конце.

При загрузке использовать StreamReader. Обработку делать построчную (не загружать текст из файла целиком в программу).

6. В логике формы по работе с коллекцией прописать логику метода LoadToolStripMenuItem_Click.

Проверка работы и порядок сдачи

1. Предварительно:
    a. Добавить 2 коллекции.
    б. В каждую коллекцию добавить по 2-3 объекта разного типа и характеристик.
2. Показать первую коллекцию.
3. Показать вторую коллекцию.
4. Сохранить в файл.
5. Закрыть программу.
6. Запустить заново.
7. Загрузить из файла.
8. Показать первую коллекцию.
9. Показать вторую коллекцию.

Контрольные вопросы к базовой части

1. Где и как вызываются статические методы, используемые при сохранении и загрузке?
2. Рассказать пошаговый алгоритм сохранения.
3. Рассказать пошаговый алгоритм загрузки.

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:*
    а. *Реализовать генерацию и обработку собственных исключений. Собственные исключения выбрасывать в классах по работе с коллекциями, если не находится элемент по позиции, если указанная позиция выходит за границы коллекции и, если нельзя уже вставить в коллекцию новый элемент по причине заполненности коллекции.*
    б. *Выбрасывать исключения в классе-хранилище при работе с файлами, используя существующие классы исключений, прописанные в .Net.*
    в. *Добавить логирование действий пользователя. Логи писать в специальные файлы. Выделять логи действий пользователя и логи возникающих ошибок.*

Проектирование

Для генерации собственных исключений создадим новые исключения от класса ApplicationException. 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 классах-коллекциях есть три случая, когда уместно вызывать исключения: при попытке добавить объект в переполненную коллекцию, при попытке получить объект с пустого места и когда пытаемся получить объект по позиции, выходящий за границы коллекции.

В классе-хранилище 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. ILogger – это лишь абстракция, описывающая возможные действия логгера (запись логов разного типа). А в качестве реализации этой абстракции используют различные готовые решения. 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, 7.3).

```
using System.Runtime.Serialization;

namespace ProjectSportCar.Exceptions;

/// <summary>
/// Класс, описывающий ошибку переполнения коллекции
/// </summary>
[Serializable]
internal class CollectionOverflowException : ApplicationException
{
	public CollectionOverflowException(int count) : base("В коллекции превышено
допустимое количество: " + count) { }

	public CollectionOverflowException() : base() { }

	public CollectionOverflowException(string message) : base(message) { }

	public CollectionOverflowException(string message, Exception exception) :
base(message, exception) { }

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

Листинг 7.1 – Код класса CollectionOverflowException

```
using System.Runtime.Serialization;

namespace ProjectSportCar.Exceptions;

/// <summary>
/// Класс, описывающий ошибку, что по указанной позиции нет элемента
/// </summary>
[Serializable]
internal class ObjectNotFoundException : ApplicationException
{
	public ObjectNotFoundException(int i) : base("Не найден объект по позиции "
+ i) { }

	public ObjectNotFoundException() : base() { }

	public ObjectNotFoundException(string message) : base(message) { }

	public ObjectNotFoundException(string message, Exception exception) :
base(message, exception) { }

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

Листинг 7.2 – Код класса ObjectNotFoundException

```
using System.Runtime.Serialization;

namespace ProjectSportCar.Exceptions;

/// <summary>
```

```
/// Класс, описывающий ошибку выхода за границы коллекции
/// </summary>
[Serializable]
internal class PositionOutOfCollectionException : ApplicationException
{
	public PositionOutOfCollectionException(int i) : base("Выход за границы
коллекции. Позиция " + i) { }

	public PositionOutOfCollectionException() : base() { }

	public PositionOutOfCollectionException(string message) : base(message) { }

	public PositionOutOfCollectionException(string message, Exception
exception) : base(message, exception) { }

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

Листинг 7.3 – Код класса PositionOutOfCollectionException

Далее в классах-коллекциях в методах Insert, Remove и Get делаем выброс этих исключений. В методе Insert в случае, если размерность коллекции достигла максимального значения, либо указанная позиция при вставке выходит за границы коллекции. В методах Remove и Get в случае, если по указанной позиции уже пустой элемент находится (потребуется добавить проверки в соответствующих методах) либо указанная позиция выходит за границы коллекции.

В классе-хранилище пропишем вызов исключений в методах сохранения и загрузки. В методе сохранения (там, где раньше возвращался false) будем выбрасывать ошибку, если нет данных для сохранения. А в методе загрузке делаем выброс ошибок, если нет файла, либо неверный формат данных (листинг 7.4). Само собой, в логике формы потребуется внести изменения в методы загрузки и сохранения, а также добавить туда операторы try-catch.

```
using ProjectSportCar.Drawnings;
using ProjectSportCar.Exceptions;
using System.Text;

namespace ProjectSportCar.CollectionGenericObjects;

/// <summary>
/// Класс-хранилище коллекций
/// </summary>
/// <typeparam name="T"></typeparam>
public class StorageCollection<T>
	where T : DrawningCar
```

```
{
	…

	/// <summary>
	/// Сохранение информации по автомобилям в хранилище в файл
	/// </summary>
	/// <param name="filename">Путь и имя файла</param>
	public void SaveData(string filename)
	{
		if (_storages.Count == 0)
		{
			throw new Exception("В хранилище отсутствуют коллекции для 
сохранения");
		}

		if (File.Exists(filename))
		{
			File.Delete(filename);
		}

		StringBuilder sb = new();

		sb.Append(_collectionKey);
		foreach (KeyValuePair<string, ICollectionGenericObjects<T>> value in 
_storages)
		{
			sb.Append(Environment.NewLine);
			// не сохраняем пустые коллекции
			if (value.Value.Count == 0)
			{
				continue;
			}

			sb.Append(value.Key);
			sb.Append(_separatorForKeyValue);
			sb.Append(value.Value.GetCollectionType);
			sb.Append(_separatorForKeyValue);
			sb.Append(value.Value.MaxCount);
			sb.Append(_separatorForKeyValue);

			foreach (T? item in value.Value.GetItems())
			{
				string data = item?.GetDataForSave() ?? string.Empty;
				if (string.IsNullOrEmpty(data))
				{
					continue;
				}

				sb.Append(data);
				sb.Append(_separatorItems);
			}
		}

		using FileStream fs = new(filename, FileMode.Create);
		byte[] info = new UTF8Encoding(true).GetBytes(sb.ToString());
		fs.Write(info, 0, info.Length);
	}

	/// <summary>
	/// Загрузка информации по автомобилям в хранилище из файла
	/// </summary>
	/// <param name="filename">Путь и имя файла</param>
	public void LoadData(string filename)
```

```
      {
            if (!File.Exists(filename))
            {
                  throw new Exception("Файл не существует");
            }

            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);
                  }
            }

            string[] strs = bufferTextFromFile.Split(new char[] { '\n', '\r' },
StringSplitOptions.RemoveEmptyEntries);
            if (strs == null || strs.Length == 0)
            {
                  throw new Exception("В файле нет данных");
            }

            if (!strs[0].Equals(_collectionKey))
            {
                  throw new Exception("В файле неверные данные");
            }

            _storages.Clear();
            foreach (string data in strs)
            {
                  string[] record = data.Split(_separatorForKeyValue,
StringSplitOptions.RemoveEmptyEntries);
                  if (record.Length != 4)
                  {
                        continue;
                  }

                  CollectionType collectionType =
(CollectionType)Enum.Parse(typeof(CollectionType), record[1]);
                  ICollectionGenericObjects<T>? collection =
StorageCollection<T>.CreateCollection(collectionType) ??
                        throw new Exception("Не удалось определить тип
коллекции:" + record[1]);
                  collection.MaxCount = Convert.ToInt32(record[2]);

                  string[] set = record[3].Split(_separatorItems,
StringSplitOptions.RemoveEmptyEntries);
                  foreach (string elem in set)
                  {
                        if (elem?.CreateDrawningCar() is T car)
                        {
                              try
                              {
                                    if (!collection.Insert(car))
                                    {
                                          throw new Exception("Объект не
удалось добавить в коллекцию: " + record[3]);
                                    }
                              }
                              catch (CollectionOverflowException ex)
                              {
```

```
                                        throw new Exception("Коллекция
переполнена", ex);
                                }
                            }
                        }

                        _storages.Add(record[0], collection);
                    }
                }
                …
}
```

Листинг 7.4 – Обновленные методы класса-хранилища

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

- Microsoft.Extensions.Logging
- 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;
using ProjectSportCar.CollectionGenericObjects;
using ProjectSportCar.Drawnings;

namespace ProjectSportCar;

/// <summary>
/// Форма работы с компанией и ее коллекцией
/// </summary>
public partial class FormCarCollection : Form
{
        …

        /// <summary>
        /// Логер
        /// </summary>
        private readonly ILogger _logger;

        /// <summary>
        /// Конструктор
        /// </summary>
        public FormCarCollection(ILogger<FormCarCollection> logger)
        {
                InitializeComponent();
                _storageCollection = new();
                _logger = logger;
        }

        …

        /// <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
                        {
                                _storageCollection.SaveData(saveFileDialog.FileName);
                                MessageBox.Show("Сохранение прошло успешно",
"Результат", MessageBoxButtons.OK, MessageBoxIcon.Information);
                                _logger.LogInformation("Сохранение в файл: {filename}",
saveFileDialog.FileName);
                        }
                        catch(Exception ex)
                        {
                                MessageBox.Show(ex.Message, "Результат",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                                _logger.LogError("Ошибка: {Message}", ex.Message);
                        }
                }
        }

        …
}
```

Листинг 7.5 – Пример работы с логгером в логике формы FormCarCollection

Теперь надо в классе Program в месте вызова конструктора формы FormCarCollection 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 ProjectSportCar
{
        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();

                        ServiceCollection services = new();
```

```
                    ConfigureServices(services);
                    using ServiceProvider serviceProvider =
services.BuildServiceProvider();

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

            /// <summary>
            /// Конфигурация сервиса DI
            /// </summary>
            /// <param name="services"></param>
            private static void ConfigureServices(ServiceCollection services)
            {
                  services.AddSingleton<FormCarCollection>()
                              .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 FormCarCollection, 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.

Останется только в свойствах файла nlog.config указываем, чтобы он всегда копировался при сборке (рисунок 7.1).

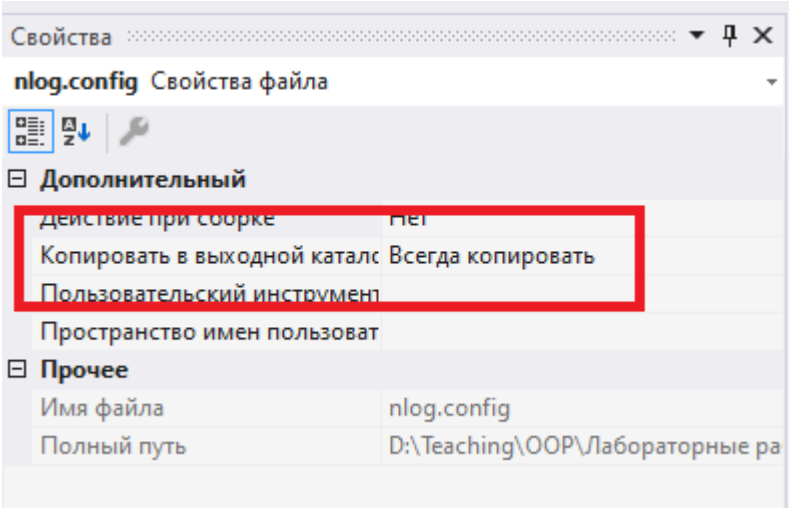


Рисунок 7.1 – Настройка копирования файла конфигурации в папку сборки

Тестирование

Запустим проект, создаем коллекцию на массиве, компанию и добавляем туда объектов, пока есть свободные места (рисунок 7.2).

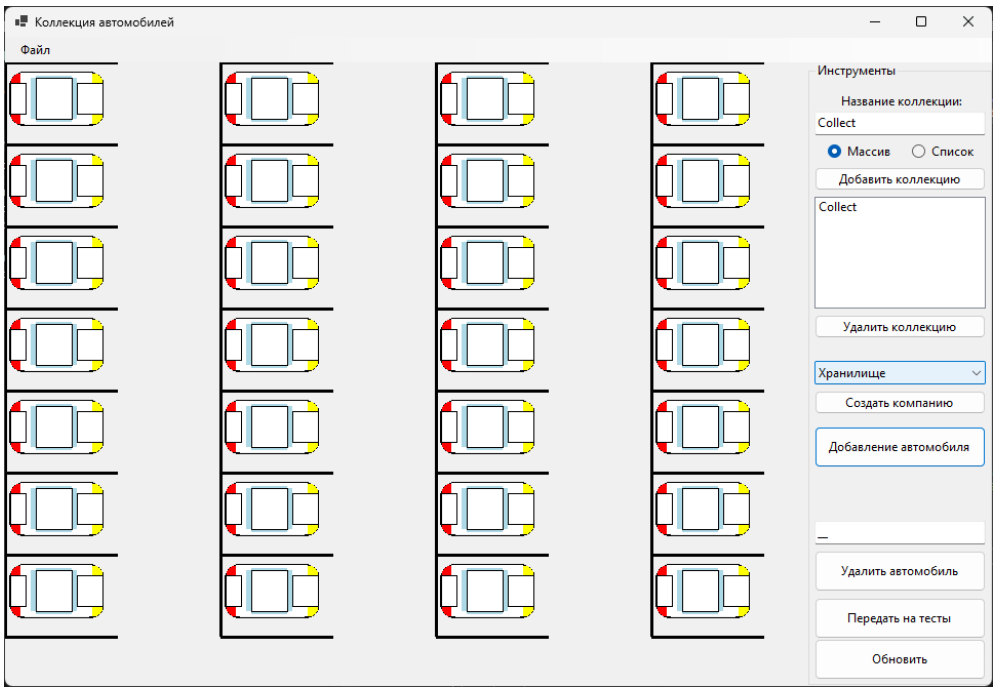


Рисунок 7.2 – Заполненная коллекция на массиве

Коллекцию лучше сохранить, чтобы, если что, всегда можно было загрузить и не тратить время на повторное заполнение. Далее, пытаемся добавить еще один элемент, получаем ошибку переполнения (рисунок 7.3).

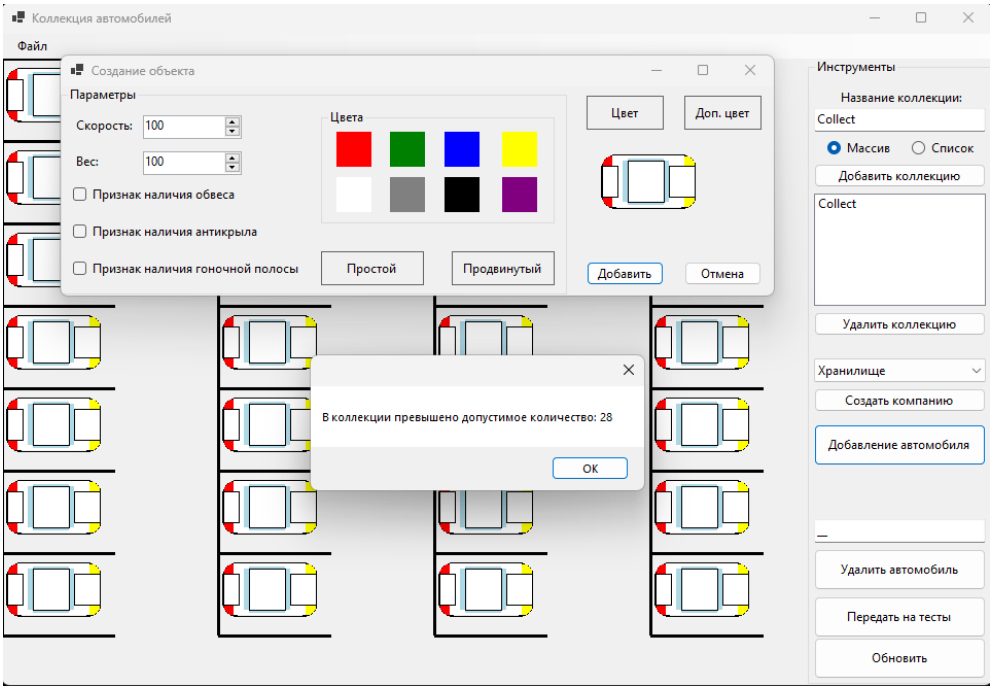


Рисунок 7.3 – Ошибка переполнения

Теперь удалим с какой-нибудь позиции объект (рисунок 7.4).

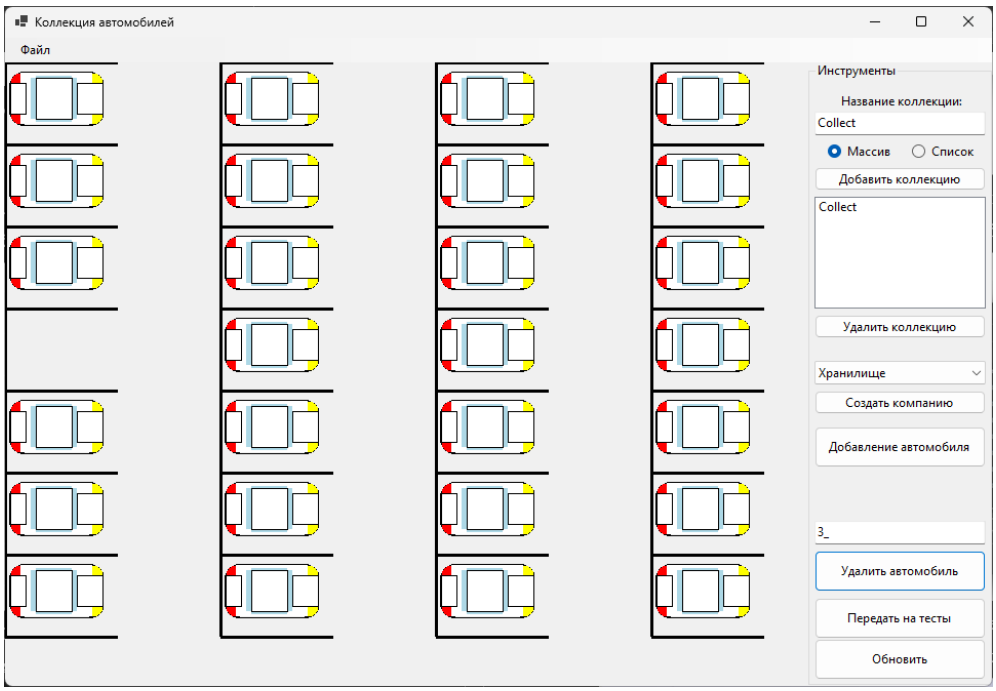


Рисунок 7.4 – Удаление объекта

Повторно попытаемся удалить с этой позиции объект, получаем ошибку удаления пустого объекта (рисунок 7.5).

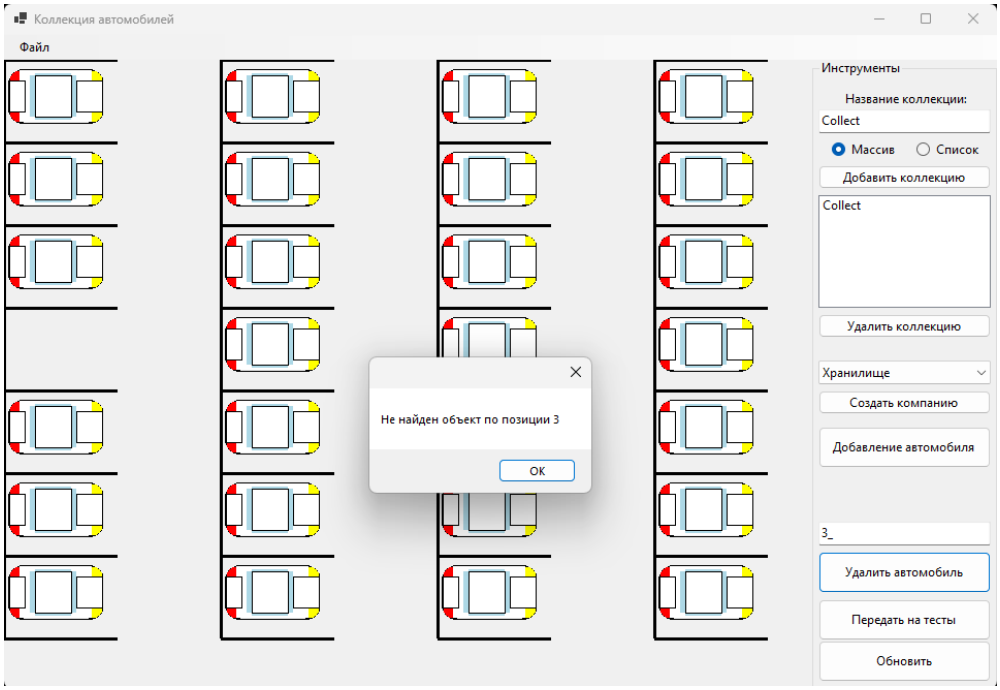


Рисунок 7.5 – Удаление с пустого места

Посмотрим лог-файл. Там будут записи, как добавления объектов, так и записи по ошибкам (рисунок 7.6).

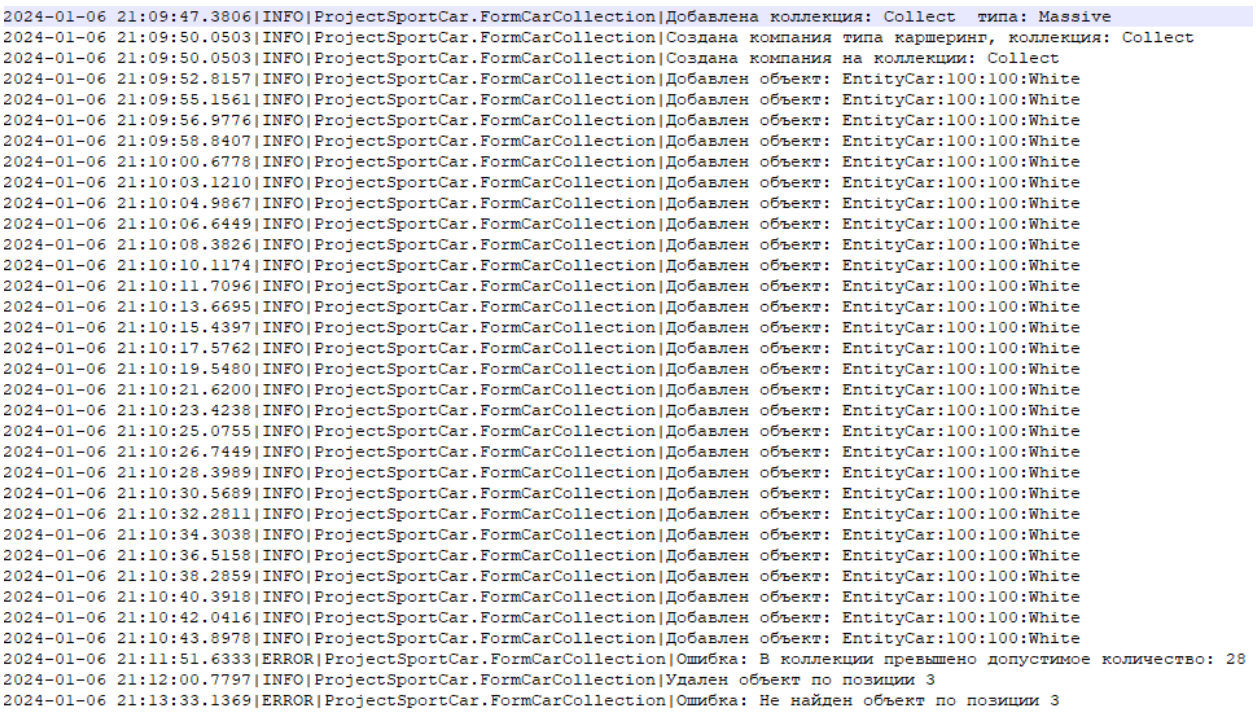


Рисунок 7.6 – Лог-файл

Таким образом все поставленные задачи выполнены. Работа готова.

Требования

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

   ==========

4. Прописать вызов исключений в методах Insert, Remove и Get классов-коллекций.
5. В классе-хранилище в методах сохранения и загрузки в местах генерации ошибок вместо класса Exception использовать его наследников, которые будут соответствовать возникающей ошибке.

6. В логике формы обрабатывать ошибки, выбросы которых прописаны в классах-коллекциях и классе-хранилище (обработка – запись соответствующих логов в файлы). Убедиться, что логируются следующие действия пользователя: добавление коллекции, удаление коллекции, добавление объекта, удаление объекта, сохранение данных, загрузка данных. Вместо пакета NLog использовать пакет Serilog.

Проверка работы и порядок сдачи

1. Предварительно:
    а. Добавить коллекцию.
    б. В коллекцию добавить объекты в таком количестве, чтобы при добавлении следующего возникала ошибка переполнения.
    в. Сохранить в файл (чтобы не каждый раз заполнять всю коллекцию).
2. Показать вывод ошибки переполнения.
3. Показать ошибку извлечения с пустого места (использовать коллекцию-массив).
4. Закрыть программу.
5. Показать файл с логами.

Контрольные вопросы к базовой части

1. Как обрабатываются ошибки (свой класс-исключение, где выброс ошибки, где обработка)?
2. Как данные записываются в лог-файл?
3. Как и где настроить формат записи логов?

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:*
    a. *При добавлении объекта реализовать проверку уникальности этого объекта в классах-коллекциях, куда его добавляют (т.е. нет второго объекта с такими же характеристиками).*
    б. *Реализовать возможность сортировки объектов в классе-коллекции 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. *Создать класс для описания коллекции. В класс включить свойства: название, описание. Заменить ключ в словаре в классе-хранилище на этот класс.*

Проектирование

Для проверки уникальности объекта в коллекции 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. У нас два класса-коллекции и оба от параметра T. В абстрактном классе-компании этот параметр заменяется типом DrawningCar. Поэтому логично тип унаследовать от интерфейса IEquatable. Однако, есть проблема, что в списке вместо объекта DrawningCar может быть объект от дочернего класса DrawningSportCar. В таком случае, при реализации интерфейса IEquatable в классе DrawningCar мы рискуем потерять часть логики, если одним из сравниваемых объектов будет объект от DrawningSportCar. Потому, сделаем отдельный класс, унаследуем его от интерфейса IEqualityComparer и будем в нем выполнять операцию сравнения.

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, новый класс надо будет унаследовать от IEquatable. Метод определим прям в этом классе и в качестве определения по равенству будем сравнивать значения в свойствах «название» в классе.

Реализация

Первым шагом будет создание класса-реализации интерфейса IEqualityComparer. Тут будет следующая логика: если хотя бы один объект равен null, то возвращаем false. Далее сравниваем типы объектов (через вызов метода GetType и получение свойства Name у типа, будет возвращено как раз

имя класса) и остальные параметры. Если что-то не совпадает возвращаем false. Если все совпадает, то возвращаем true (листинг 8.1).

```
using System.Diagnostics.CodeAnalysis;

namespace ProjectSportCar.Drawnings;

/// <summary>
/// Реализация сравнения двух объектов класса-прорисовки
/// </summary>
public class DrawiningCarEqutables : IEqualityComparer<DrawningCar?>
{
	public bool Equals(DrawningCar? x, DrawningCar? y)
	{
		if (x == null || x.EntityCar == null)
		{
			return false;
		}

		if (y == null || y.EntityCar == null)
		{
			return false;
		}

		if (x.GetType().Name != y.GetType().Name)
		{
			return false;
		}

		if (x.EntityCar.Speed != y.EntityCar.Speed)
		{
			return false;
		}

		if (x.EntityCar.Weight != y.EntityCar.Weight)
		{
			return false;
		}

		if (x.EntityCar.BodyColor != y.EntityCar.BodyColor)
		{
			return false;
		}

		if (x is DrawningSportCar && y is DrawningSportCar)
		{
			// TODO доделать логику сравнения дополнительных параметров
		}

		return true;
	}

	public int GetHashCode([DisallowNull] DrawningCar obj)
	{
		return obj.GetHashCode();
	}
}
```

Листинг 8.1 – Класс-реализация интерфейса IEqualityComparer

Далее надо бы в классах-коллекциях при вставке выполнять проверку. Однако, в таком случае класс должен узнать, что тип T как-то ограничен. Чтобы этого избежать, добавим еще один передаваемый параметр в методы вставки. При этом, параметр сделаем необязательным (листинг 8.2).

```
using ProjectSportCar.Drawnings;

namespace ProjectSportCar.CollectionGenericObjects;

/// <summary>
/// Интерфейс описания действий для набора хранимых объектов
/// </summary>
/// <typeparam name="T">Параметр: ограничение - ссылочный тип</typeparam>
public interface ICollectionGenericObjects<T>
       where T : class
{
       …

       /// <summary>
       /// Добавление объекта в коллекцию
       /// </summary>
       /// <param name="obj">Добавляемый объект</param>
       /// <param name="comparer">Сравнение двух объектов</param>
       /// <returns>true - вставка прошла удачно, false - вставка не
удалась</returns>
       bool Insert(T obj, IEqualityComparer<T?>? comparer = null);

       /// <summary>
       /// Добавление объекта в коллекцию на конкретную позицию
       /// </summary>
       /// <param name="obj">Добавляемый объект</param>
       /// <param name="position">Позиция</param>
       /// <param name="comparer">Сравнение двух объектов</param>
       /// <returns>true - вставка прошла удачно, false - вставка не
удалась</returns>
       bool Insert(T obj, int position, IEqualityComparer<T?>? comparer = null);

       …
}
```

Листинг 8.2 – Правки в объявлении методов вставки интерфейса-коллекции

Останется только передавать объект от класса DrawiningCarEqutables при вызове методов Insert в абстрактном классе-компании.

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 не null. Затем для первого класса проверка по типу, для второго – по цвету. Далее проверка по остальным параметрам, если первый совпадает (листинги 8.3 и 8.4).

```
namespace ProjectSportCar.Drawnings;

/// <summary>
/// Сравнение по типу, скорости, весу
/// </summary>
public class DrawningCarCompareByType : IComparer<DrawningCar?>
{
    public int Compare(DrawningCar? x, DrawningCar? y)
    {
            if (x == null || x.EntityCar == null)
            {
                return -1;
            }

            if (y == null || y.EntityCar == null)
            {
                return 1;
            }

            if (x.GetType().Name != y.GetType().Name)
            {
                return x.GetType().Name.CompareTo(y.GetType().Name);
            }

            var speedCompare = x.EntityCar.Speed.CompareTo(y.EntityCar.Speed);
            if (speedCompare != 0)
            {
                return speedCompare;
            }

            return x.EntityCar.Weight.CompareTo(y.EntityCar.Weight);
      }
}
```

Листинг 8.3 – Класс DrawningCarCompareByType

```
namespace ProjectSportCar.Drawnings;

/// <summary>
/// Сравнение по цвету, скорости, весу
/// </summary>
public class DrawningCarCompareByColor : IComparer<DrawningCar?>
{
      public int Compare(DrawningCar? x, DrawningCar? y)
      {
            // TODO прописать логику сравения по цветам, скорости, весу
            throw new NotImplementedException();
      }
}
```

Листинг 8.4 – Класс DrawningCarCompareByColor

Далее в интерфейсе-коллекции и абстрактном классе-компании создаем методы для сортировки (листинги 8.5 и 8.6).

```
using ProjectSportCar.Drawnings;

namespace ProjectSportCar.CollectionGenericObjects;

/// <summary>
/// Интерфейс описания действий для набора хранимых объектов
/// </summary>
```

```
/// <typeparam name="T">Параметр: ограничение - ссылочный тип</typeparam>
public interface ICollectionGenericObjects<T>
	where T : class
{
	…

	/// <summary>
	/// Сортировка коллекции
	/// </summary>
	/// <param name="comparer">Сравнитель объектов</param>
	void CollectionSort(IComparer<T?> comparer);
}
```

Листинг 8.5 – Объявление метода сортировки в интерфейсе-коллекции

```
using ProjectSportCar.Drawnings;

namespace ProjectSportCar.CollectionGenericObjects;

/// <summary>
/// Абстракция компании, хранящий коллекцию автомобилей
/// </summary>
public abstract class AbstractCompany
{
	…

	/// <summary>
	/// Перегрузка оператора сложения для класса
	/// </summary>
	/// <param name="company">Компания</param>
	/// <param name="car">Добавляемый объект</param>
	/// <returns></returns>
	public static bool operator +(AbstractCompany company, DrawningCar car)
	{
		return company._collection?.Insert(car, new DrawiningCarEqutables())
?? false;
	}

	…

	/// <summary>
	/// Сортировка
	/// </summary>
	/// <param name="comparer">Сравнитель объектов</param>
	public void Sort(IComparer<DrawningCar?> comparer) =>
_collection?.CollectionSort(comparer);

	…
}
```

Листинг 8.6 – Добавление метода сортировки в абстрактный класс-компанию

Остается на форме сделать 2 кнопки: «сортировка по типу» и «сортировка по цвету» (рисунок 8.1). И вызывать метод сортировки класса-компании передавая туда объект от класса DrawiningCarCompareByType или от класса DrawiningCarCompareByColor (листинг 8.7).

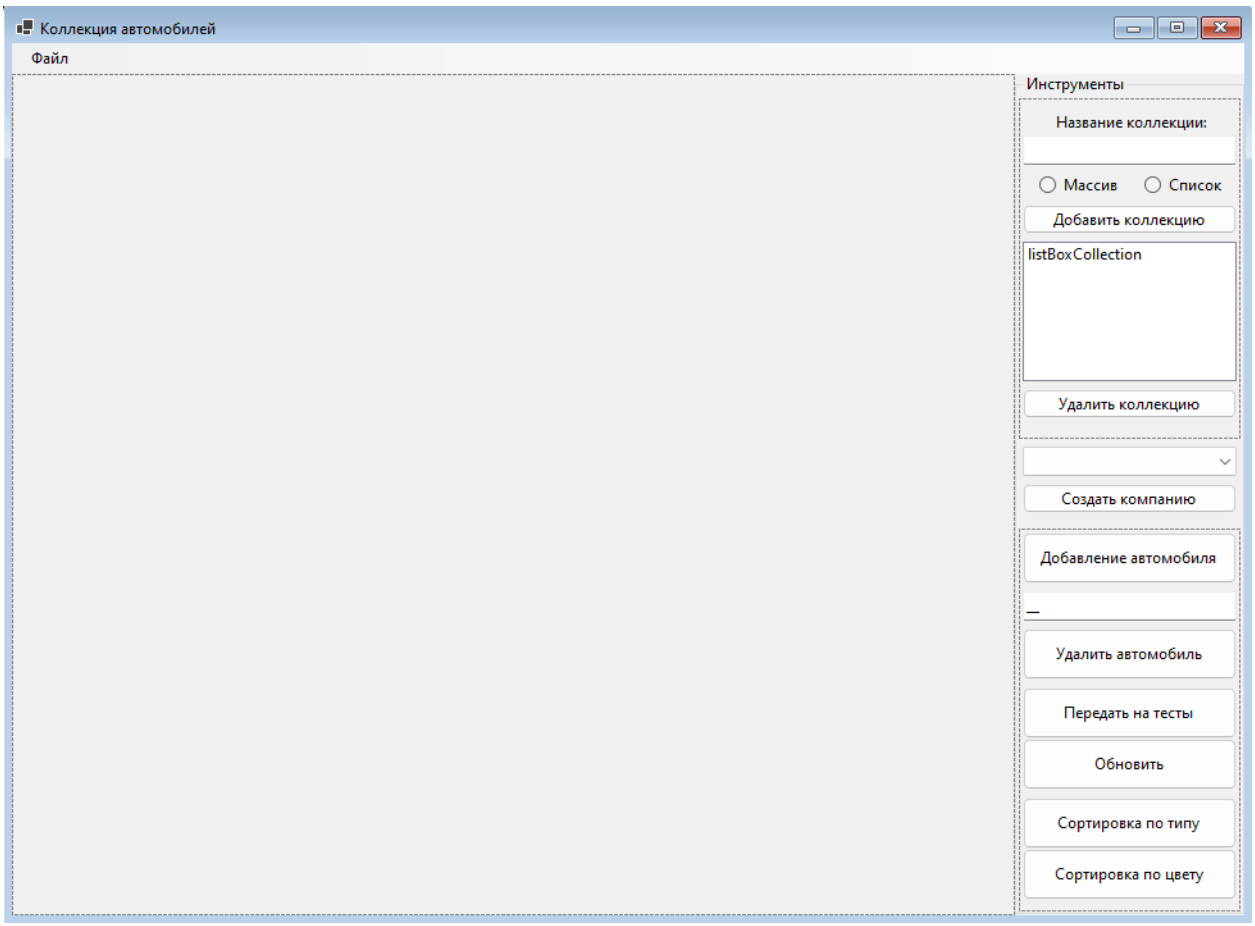


Рисунок 8.1 – Обновленная форма FormCarCollection

```
using Microsoft.Extensions.Logging;
using ProjectSportCar.CollectionGenericObjects;
using ProjectSportCar.Drawnings;

namespace ProjectSportCar;

/// <summary>
/// Форма работы с компанией и ее коллекцией
/// </summary>
public partial class FormCarCollection : Form
{
	…

	/// <summary>
	/// Сортировка по типу
	/// </summary>
	/// <param name="sender"></param>
	/// <param name="e"></param>
	private void ButtonSortByType_Click(object sender, EventArgs e)
	{
		CompareCars(new DrawningCarCompareByType());
	}

	/// <summary>
	/// Сортировка по цвету
	/// </summary>
	/// <param name="sender"></param>
	/// <param name="e"></param>
	private void ButtonSortByColor_Click(object sender, EventArgs e)
```

```
        {
                CompareCars(new DrawningCarCompareByColor());
        }

        /// <summary>
        /// Сортировка по сравнителю
        /// </summary>
        /// <param name="comparer">Сравнитель объектов</param>
        private void CompareCars(IComparer<DrawningCar?> comparer)
        {
                if (_company == null)
                {
                        return;
                }

                _company.Sort(comparer);
                pictureBox.Image = _company.Show();
        }
}
```

Листинг 8.7 – Добавление методов в логику формы FormCarCollection

Второй шаг готов. Для третьего шага потребуется класс для хранения описания коллекции (листинг 8.8). Пока что ограничимся тремя полями (свойствами) в классе: название, тип коллекции и описание. Класс нужно будет унаследовать от IEquatable. Потребуется проверить, что у объекта other, который будет поступать на вход метода Equals значения полей «Название» и «Тип коллекции» будут совпадать с полями «Название» и «Тип коллекции» у текущего объекта.

```
namespace ProjectSportCar.CollectionGenericObjects;

/// <summary>
/// Класс, хранящий информацию по коллекции
/// </summary>
public class CollectionInfo : IEquatable<CollectionInfo>
{
        /// <summary>
        /// Название
        /// </summary>
        public string Name { get; private set; }

        /// <summary>
        /// Тип
        /// </summary>
        public CollectionType CollectionType { get; private set; }

        /// <summary>
        /// Описание
        /// </summary>
        public string Description { get; private set; }

        /// <summary>
        /// Разделитель для записи информации по объекту в файл
        /// </summary>
        private static readonly string _separator = "-";
```

```
        /// <summary>
        /// Конструктор
        /// </summary>
        /// <param name="name">Название</param>
        /// <param name="collectionType">Тип</param>
        /// <param name="description">Описание</param>
        public CollectionInfo(string name, CollectionType collectionType, string
description)
        {
                Name = name;
                CollectionType = collectionType;
                Description = description;
        }

        /// <summary>
        /// Создание объекта из строки
        /// </summary>
        /// <param name="data">Строка</param>
        /// <returns>Объект или null</returns>
        public static CollectionInfo? GetCollectionInfo(string data)
        {
                string[] strs = data.Split(_separator,
StringSplitOptions.RemoveEmptyEntries);
                if (strs.Length < 1 || strs.Length > 3)
                {
                        return null;
                }

                return new CollectionInfo(strs[0],
(CollectionType)Enum.Parse(typeof(CollectionType), strs[1]), strs.Length > 2 ?
strs[2] : string.Empty);
        }

        public override string ToString()
        {
                return Name + _separator + CollectionType + _separator + Description;
        }

        public bool Equals(CollectionInfo? other)
        {
                return Name == other?.Name;
        }

        public override bool Equals(object? obj)
        {
                return Equals(obj as CollectionInfo);
        }

        public override int GetHashCode()
        {
                return Name.GetHashCode();
        }
}
```

Листинг 8.8 – Класс для описания коллекции

В классе-хранилище заменим ключ с типа string на этот класс. При добавлении набора, удалении и получению по ключу, будем передавать объект от нового класса (листинг 8.9).

```
using ProjectSportCar.Drawnings;
using ProjectSportCar.Exceptions;
using System.Text;

namespace ProjectSportCar.CollectionGenericObjects;

/// <summary>
/// Класс-хранилище коллекций
/// </summary>
/// <typeparam name="T"></typeparam>
public class StorageCollection<T>
	where T : DrawningCar
{
	/// <summary>
	/// Словарь (хранилище) с коллекциями
	/// </summary>
	readonly Dictionary<CollectionInfo, ICollectionGenericObjects<T>>
_storages;

	/// <summary>
	/// Возвращение списка названий коллекций
	/// </summary>
	public List<CollectionInfo> Keys => _storages.Keys.ToList();

	/// <summary>
	/// Ключевое слово, с которого должен начинаться файл
	/// </summary>
	private readonly string _collectionKey = "CollectionsStorage";

	/// <summary>
	/// Разделитель для записи ключа и значения элемента словаря
	/// </summary>
	private readonly string _separatorForKeyValue = "|";

	/// <summary>
	/// Разделитель для записей коллекции данных в файл
	/// </summary>
	private readonly string _separatorItems = ";";

	/// <summary>
	/// Конструктор
	/// </summary>
	public StorageCollection()
	{
		_storages = new Dictionary<CollectionInfo,
ICollectionGenericObjects<T>>();
	}

	/// <summary>
	/// Добавление коллекции в хранилище
	/// </summary>
	/// <param name="name">Название коллекции</param>
	/// <param name="collectionType">тип коллекции</param>
	public void AddCollection(string name, CollectionType collectionType)
	{
		// TODO проверка, что name не пустой и нет в словаре записи с таким
ключом
		// TODO Прописать логику для добавления
	}

	/// <summary>
	/// Удаление коллекции
	/// </summary>
```

```
        /// <param name="name">Название коллекции</param>
        public void DelCollection(string name)
        {
                // TODO Прописать логику для удаления коллекции
        }

        /// <summary>
        /// Доступ к коллекции
        /// </summary>
        /// <param name="name">Название коллекции</param>
        /// <returns></returns>
        public ICollectionGenericObjects<T>? this[string name]
        {
                get
                {
                        // TODO Продумать логику получения объекта
                        return null;
                }
        }

        /// <summary>
        /// Сохранение информации по автомобилям в хранилище в файл
        /// </summary>
        /// <param name="filename">Путь и имя файла</param>
        public void SaveData(string filename)
        {
                if (_storages.Count == 0)
                {
                        throw new Exception("В хранилище отсутствуют коллекции для
сохранения");
                }

                if (File.Exists(filename))
                {
                        File.Delete(filename);
                }

                StringBuilder sb = new();

                sb.Append(_collectionKey);
                foreach (KeyValuePair<CollectionInfo, ICollectionGenericObjects<T>>
value in _storages)
                {
                        sb.Append(Environment.NewLine);
                        // не сохраняем пустые коллекции
                        if (value.Value.Count == 0)
                        {
                                continue;
                        }

                        sb.Append(value.Key);
                        sb.Append(_separatorForKeyValue);
                        sb.Append(value.Value.MaxCount);
                        sb.Append(_separatorForKeyValue);

                        foreach (T? item in value.Value.GetItems())
                        {
                                string data = item?.GetDataForSave() ?? string.Empty;
                                if (string.IsNullOrEmpty(data))
                                {
                                        continue;
                                }
```

```
                        sb.Append(data);
                        sb.Append(_separatorItems);
                    }
                }

                using FileStream fs = new(filename, FileMode.Create);
                byte[] info = new UTF8Encoding(true).GetBytes(sb.ToString());
                fs.Write(info, 0, info.Length);
        }

        /// <summary>
        /// Загрузка информации по автомобилям в хранилище из файла
        /// </summary>
        /// <param name="filename">Путь и имя файла</param>
        public void LoadData(string filename)
        {
                if (!File.Exists(filename))
                {
                        throw new Exception("Файл не существует");
                }

                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);
                        }
                }

                string[] strs = bufferTextFromFile.Split(new char[] { '\n', '\r' },
StringSplitOptions.RemoveEmptyEntries);
                if (strs == null || strs.Length == 0)
                {
                        throw new Exception("В файле нет данных");
                }

                if (!strs[0].Equals(_collectionKey))
                {
                        throw new Exception("В файле неверные данные");
                }

                _storages.Clear();
                foreach (string data in strs)
                {
                        string[] record = data.Split(_separatorForKeyValue,
StringSplitOptions.RemoveEmptyEntries);
                        if (record.Length != 3)
                        {
                                continue;
                        }

                        CollectionInfo? collectionInfo =
CollectionInfo.GetCollectionInfo(record[0]) ??
                                throw new Exception("Не удалось определить информацию
коллекции:" + record[0]);
                        ICollectionGenericObjects<T>? collection =
StorageCollection<T>.CreateCollection(collectionInfo.CollectionType) ??
                                throw new Exception("Не удалось создать коллекцию");
                        collection.MaxCount = Convert.ToInt32(record[1]);
```

```
                    string[] set = record[2].Split(_separatorItems,
StringSplitOptions.RemoveEmptyEntries);
                    foreach (string elem in set)
                    {
                        if (elem?.CreateDrawningCar() is T car)
                        {
                            try
                            {
                                if (!collection.Insert(car))
                                {
                                    throw new Exception("Объект не
удалось добавить в коллекцию: " + record[3]);
                                }
                            }
                            catch (CollectionOverflowException ex)
                            {
                                throw new Exception("Коллекция
переполнена", ex);
                            }
                        }
                    }

                    _storages.Add(collectionInfo, collection);
                }
            }

            /// <summary>
            /// Создание коллекции по типу
            /// </summary>
            /// <param name="collectionType"></param>
            /// <returns></returns>
            private static ICollectionGenericObjects<T>?
CreateCollection(CollectionType collectionType)
            {
                return collectionType switch
                {
                    CollectionType.Massive => new MassiveGenericObjects<T>(),
                    CollectionType.List => new ListGenericObjects<T>(),
                    _ => null,
                };
            }
}
```

Листинг 8.9 – Класс-хранилище с новым ключом

Соответственно, потребуется внести корректировки в логику работы формы.

Тестирование

Запустим проект, создаем коллекцию на массиве, компанию и добавляем туда объект, задав ему определенный цвет, не по умолчанию (рисунок 8.2).

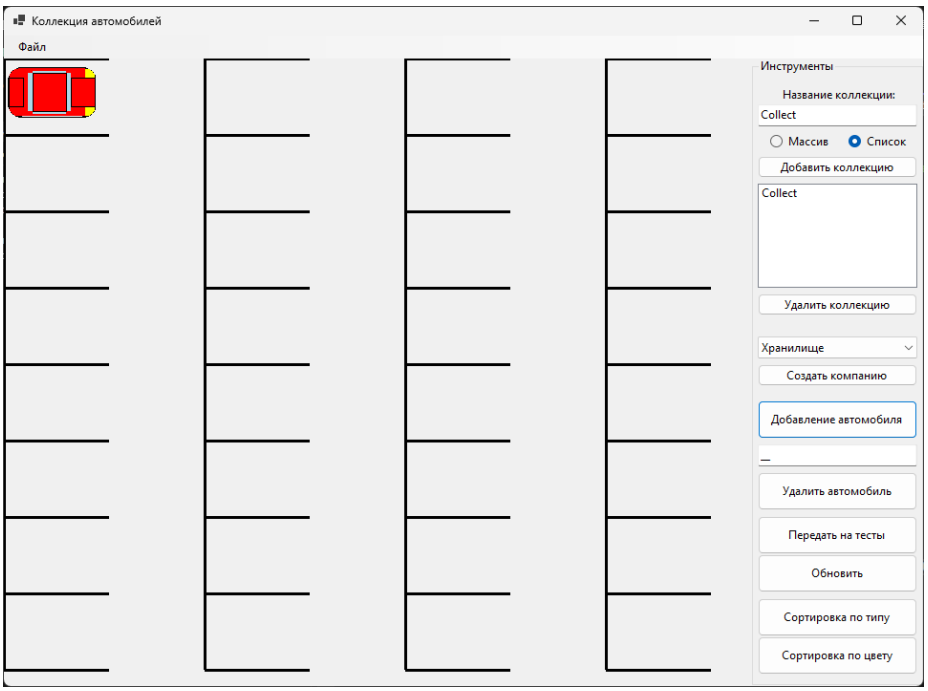


Рисунок 8.2 – Создание объекта определенного цвета

Пытаемся добавить туда еще один такой же объект, получаем ошибку (рисунок 8.3).

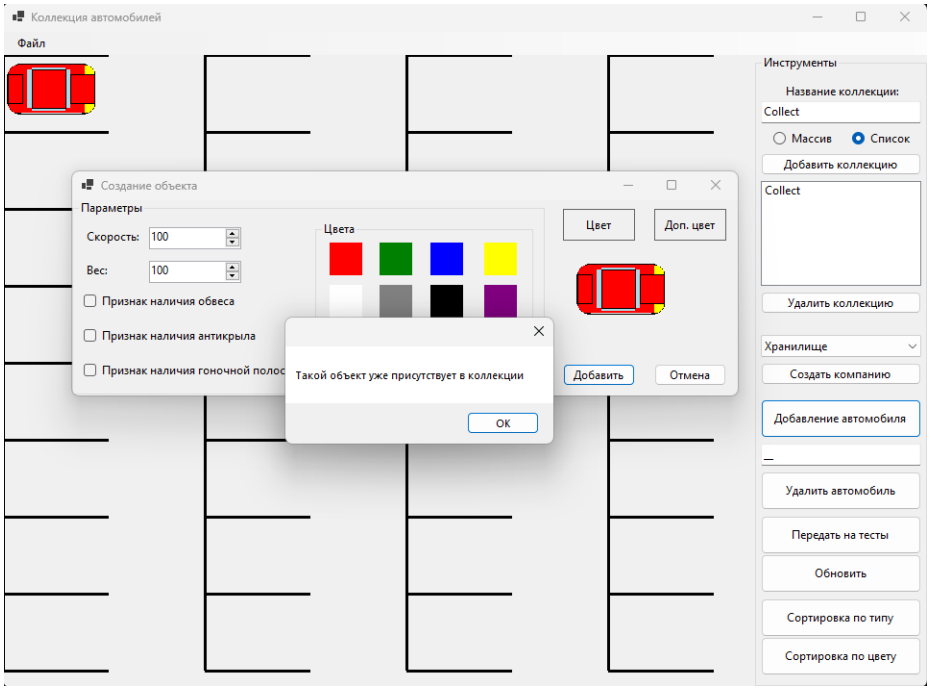


Рисунок 8.3 – Ошибка наличия такого же простого объекта

Добавляем в коллекцию «продвинутый» объект, указав у него все признаки, выставим базовый цвет, такой же как и у «простого» объекта и задав дополнительный цвет (рисунок 8.4).

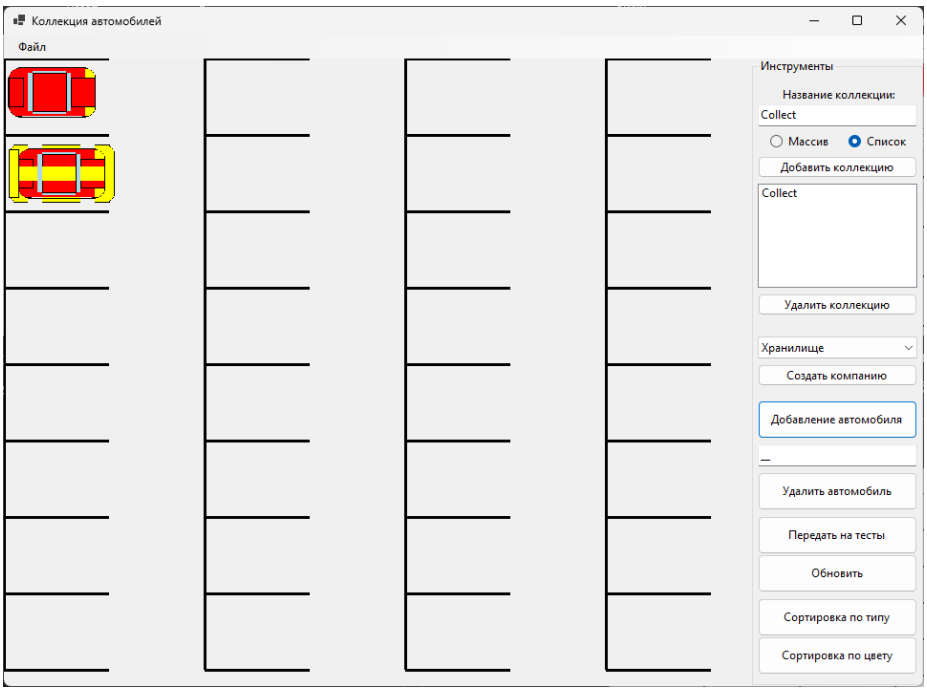


Рисунок 8.4 – Два объекта в коллекции

Пытаемся добавить такой же продвинутый объект, получаем ошибку (рисунок 8.5).

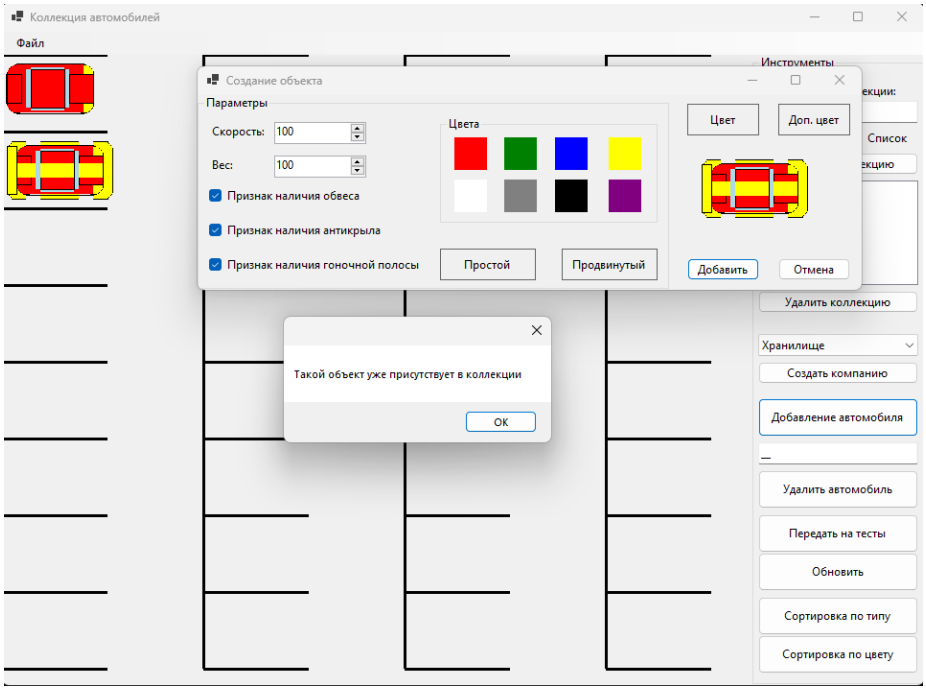


Рисунок 8.5 – Еще одна ошибка наличия такого же простого объекта

Добавляем еще один объект «простого» типа (рисунок 8.6).

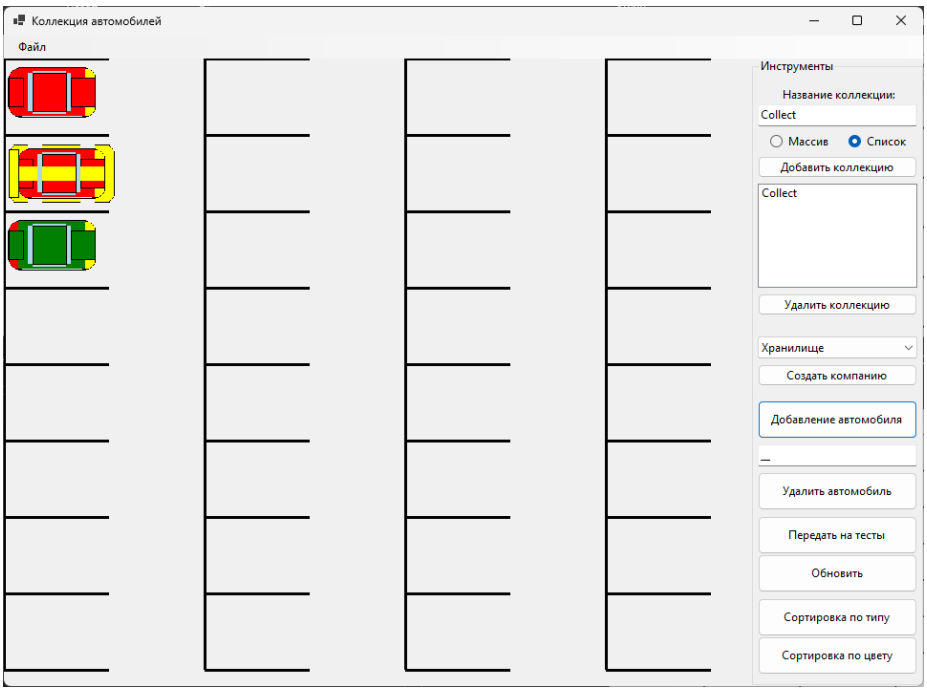


Рисунок 8.6 – Три объекта в коллекции

Нажимаем кнопку «Сортировка по типу» (рисунок 8.7).

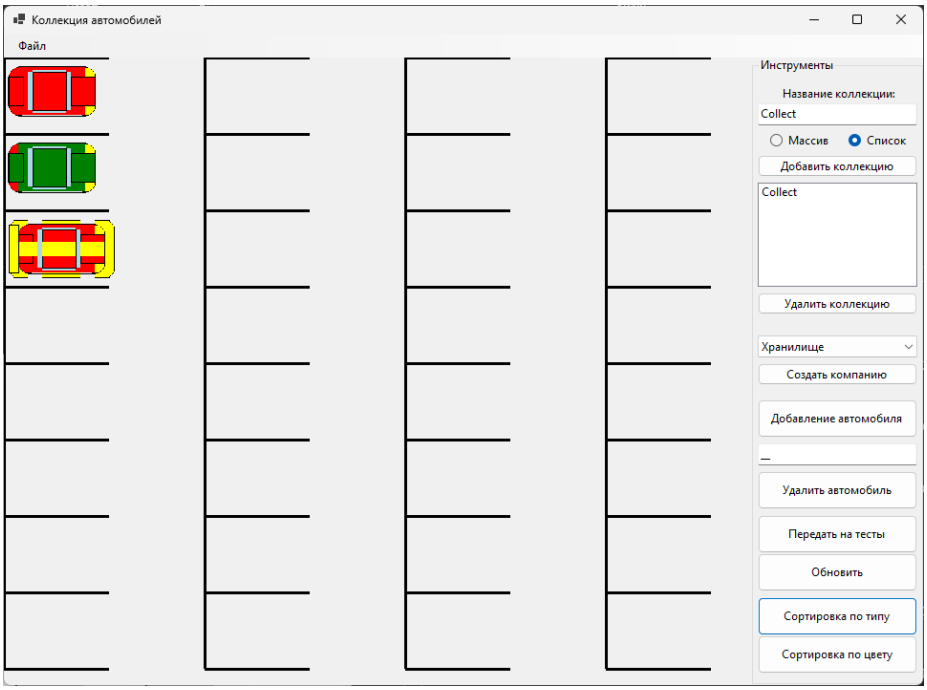


Рисунок 8.7 – Сортировка по типу

Нажимаем кнопку «Сортировка по цвету» (рисунок 8.8).

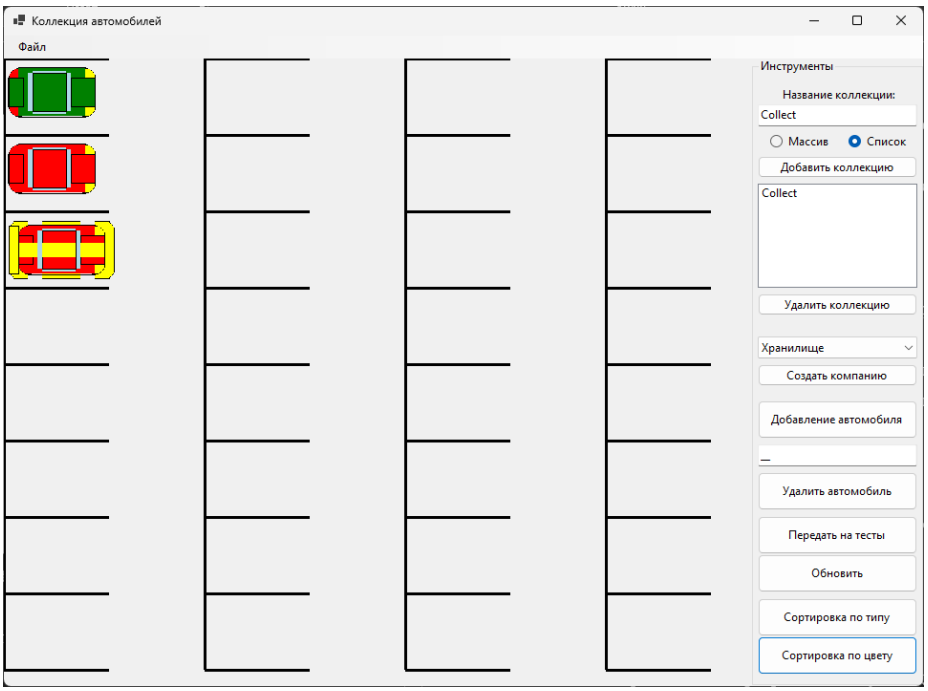


Рисунок 8.8 – Сортировка по цвету

Таким образом все поставленные задачи выполнены. Работа готова.

Требования

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

   ==========
4. Дописать логику метода Equals класса сравнения объектов классов-прорисовок.
5. В классах-коллекциях прописать выброс ошибки (создать свою ошибку), если в коллекции уже есть такой элемент.
6. Прописать логику метода Compare класса сортировки по цвету.

Проверка работы и порядок сдачи

1. Добавить коллекцию.
2. В коллекцию добавить простой объект определенного (выбрать цвет из имеющейся палитры) цвета.
3. Попытаться добавить такой же объект, получить ошибку.
4. Добавить объект продвинутого типа с таким же основным цветом и определенным дополнительным цветом.
5. Попытаться добавить еще один такой же продвинутый объект, получить ошибку.
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?
2. Как реализована проверка на уникальность объектов в коллекции?
3. Как реализована возможность использования класса-информатора в качестве ключа для словаря?

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)

Вар.	Базовый объект	«Продвинутый» объект
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)
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)

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

КУРСОВАЯ РАБОТА. ЗАДАНИЕ НА 3

Выполнить 6 лабораторных работ.

КУРСОВАЯ РАБОТА. ЗАДАНИЕ НА 4

Выполнить 7 лабораторных работ.

КУРСОВАЯ РАБОТА. ЗАДАНИЕ НА 5

Выполнить 8 лабораторных работ.

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 1.25.

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.
- 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 «Объектно-ориентированное программирование»

КУРСОВАЯ РАБОТА

Тема указать свою тему по приказу

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

Курс первый Группа ИСЭбд-11

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

Руководитель доцент _____ Эгов Е.Н.
должность, ученая степень, ученое звание фамилия, имя, отчество

Дата сдачи:
«____» _____2024 г.

Дата защиты:
«____» _____2024 г.

Оценка: _____

Ульяновск
2024 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 «Объектно-ориентированное программирование»

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Студенту ИСЭбд-11 Фамилия И.О.
группа фамилия, инициалы
Тема работы указать свою тему по приказу
Срок сдачи законченной работы «___» _____2024 г.
Исходные данные к работе: описание задания по теме, утвержденной распоряжением деканата ФИСТ
Рекомендуемая литература: курс лекций по дисциплине «Объектно-ориентированное программирование», методические указания к лабораторным работам по дисциплине «Объектно-ориентированное программирование», 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
Перечень графического материала (с точным указанием обязательных чертежей)
Скриншоты разработанного программного продукта

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

«_____»_____2024 г.

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

«_____»_____2024 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-11 курс первый
Дисциплина «Объектно-ориентированное программирование»
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 доцент _____ /Эгов Е.Н./
должность, учёная степень, ученое звание подпись инициалы, фамилия

«_____»_____2024 г.