ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

УПРАВЛЕНИЕ ДИСТАНЦИОННОГО ОБУЧЕНИЯ И ПОВЫШЕНИЯ КВАЛИФИКАЦИИ

Кафедра «Вычислительные системы и информационная безопасность»

Методические указания

к проведению лабораторных работ
по дисциплине

«Технологии и методы программирования»

Автор
Айдинян А.Р.

Ростов-на-Дону, 2017

Аннотация

«Методические указания» предназначены для студентов очной формы обучения направления подготовки 10.03.01 «Информационная безопасность» для выполнения лабораторной работы №1 по дисциплине «Технологии и методы программирования»

Автор

доцент, к.т.н.,
доцент кафедры
«Вычислительные системы и информационная безопасность»
Айдинян А.Р.

Оглавление

ОРГАНИЗАЦИЯ ХРАНЕНИЯ ОБЪЕКТОВ В СПИСКЕ НА ЯЗЫКЕ C# С ВОЗМОЖНОСТЬЮ РЕДАКТИРОВАНИЯ

Цель работы: изучить возможность создания, отображения и редактирования списков на платформе .NET.

Краткие теоретические сведения

Распространенным средством хранения набора данных является объект класса List. Он позволяет добавлять в него новые элементы, удалять, сортировать и так далее.

List является шаблонным классом. Поэтому при описании объекта необходимо указать тип данных, которые он будет содержать.

Используем объект list класса List для хранения граждан – объектов класса Grajdanin.

Класс гражданин содержит 3 свойства типа string.

```
class Grajdanin
{
    public string Fam { get; set; }  //фамилия
    public string Nam { get; set; }  //имя
    public string Country { get; set; }  //страна
}
```

Запись

```
public string Fam { get; set; }  //фамилия
```

вводит свойство Fam и автоматически создаваемое поле и является более компактной и удобной формой ранее изучаемой записи

```
private string m_Fam;
public string Fam
{
        get { return m_Fam; }
        set { m_Fam = value; }
}
```

В классе автоматически создается конструктор по умолчанию если он не имеет ни одного явно описанного конструктора.

Для создания списка list для хранения граждан необходимо записать

```
List<Grajdanin> list = new List<Grajdanin>();
```

При таком описании создается пустой список, в который можно добавлять объекты класса Grajdanin, используя метод Add класса List, и конструктор по умолчанию класса Grajdanin.

Добавление объекта класса Grajdanin в список записывается следующим образом

list.Add(new Grajdanin());

К сожалению, конструктор по умолчанию инициализирует поля пустыми строками, что потребует им явное присваивание значений следующим образом:

list[0].Fam = “Иванов”;
list[0].Nam = “Иван”;
list[0].Country = “Россия”;

или инициализировать поля до добавления в список так:

Grajdanin g = new Grajdanin();
g.Fam = “Иванов”;
g.Nam = “Иван”;
g.Country = “Россия”;
list.Add(g);

В языке C#, начиная с Visual Studio 2008, имеется возможность добавлять объекты в список и инициализировать свойства с помощью более компактной и удобной записи, которая выполняет то же, что и предыдущий программный код:

```
List<Grajdanin> list = new List<Grajdanin>()
{
   new Grajdanin { Fam = "Иванов", Nam = "Иван", Country ="Russia"},
   new Grajdanin { Fam = "Петров", Nam = "Петр", Country ="Russia"},
   new Grajdanin { Fam = "Сидоров", Nam = "Сема", Country ="Germany"},
   new Grajdanin { Fam = "Кизин", Nam = "Киз", Country ="Russia"}
};
```

Обратите внимание на то, что новая коллекция заполняется непосредственно внутри фигурных скобок. Даже, несмотря на то, что тип коллекции — List<T>, а не массив, мы все равно имеем возможность добавлять элементы без непосредственного вызова метода Add(). Даже при отсутствии конструктора с тремя параметрами для класса Grajdanin все равно имеется возможность создавать объекты этого типа как выражения, в которых внутри фигурных скобок задаются значения свойств этого объекта, указав имена этих свойств в произвольном порядке.

Для отображения полученного списка можно использовать

DataGridView и BindingNavigator, которые свяжем с BindingSource, а его в свою очередь — с list.

```
BindingSource bindingSource = new BindingSource();
bindingSource.DataSource = list;
dataGridView1.AutoGenerateColumns = true;
dataGridView1.DataSource = bindingSource;
bindingNavigator1.BindingSource = bindingSource;
```

Рекомендуется к классу Grajdanin добавить необходимые конструкторы и метод ToString(), например, так

```
class Grajdanin
{
   public string Fam { get; set; }
   public string Nam { get; set; }
   public string Country { get; set; }
   public override string ToString()
   {
      return Fam + " " + Nam + " " + Country;
   }

   public Grajdanin()// конструктор по умолчанию
   {
      Fam = "Иванов"; Nam = "Иван"; Country = "Russia";
   }
   public Grajdanin(Grajdanin g) // конструктор кописрования
   {
      Fam = g.Fam; Nam = g.Nam; Country = g.Country;
   }
   // конструкторы с параметрами – 2 штуки
   public Grajdanin(string fam, string nam)
   {
      Fam = fam; Nam = nam; Country = "Russia";
   }
   public Grajdanin(string fam, string nam, string contry)
   {
      Fam = fam; Nam = nam; Country = contry;
   }
}
```

Порядок выполнения работы

1. Создать приложение типа Window Forms.

2. Создать класс в соответствии с вариантом. Добавить в него указанные свойства и дополнительно Id (уникальный номер). Также добавить конструкторы и метод ToString.

3. Создать объект класса List для хранения списка объектов созданного класса. Заполнить его тремя-четырьмя объектами программно.

4. Отобразить список на форме в элементе управления DataGridView с использованием BindingNavigator.

5. Проверить возможность добавления, редактирования и удаления элементов в DataGridView с использованием клавиш клавиатуры и через BindingNavigator.

Варианты индивидуальных заданий

1. Разработать класс CWorker (Сотрудник), содержащий свойства PersonID (Табельный номер сотрудника), Family (Фамилия сотрудника), Birth (Дата рождения сотрудника).

2. Разработать класс CJobless (Безработный), содержащий свойства класса JoblessID (регистрационный номер безработного), LastName (Фамилия безработного), FirstName (Имя безработного), Age (возраст безработного).

3. Разработать класс CDocument (Документ), содержащий свойства класса: Serial (серия документа), Number (номер документа), Date (Дата составления), Who (кем составлен)

4. Разработать класс CZakazchik (Заказчик), содержащий свойства класса: IdZakazchika (Идентификатор заказчика), Phone (телефон заказчика), Address (адрес заказчика).

5. Разработать класс CHoz (Владелец автомобиля), содержащий свойства класса: Family (фамилия владельца), Number (номер паспорта владельца).

6. Разработать класс CRoute (Маршрут), содержащий свойства класса: RouteID (идентификатор маршрута), RouteName (название маршрута), Period (срок пребывания), Cost (стоимость путевки).

7. Разработать класс CDoctor (Врач), содержащий свойства класса: RouteID (идентификатор врача), Family (фамилия врача), Specialnost (специальность врача), Room (номер кабинета).

8. Разработать класс CCust (Арендатор), содержащий свойства класса: CustomerID (ИНН арендатора), Customer (название арендатора), AddressCust (адрес арендатора), Room (номер кабинета), Chief (фамилия руководителя).

9. Разработать класс CVlad (Владелец), содержащий свойства класса: FIO (ФИО владельца), Phone (телефон владельца),

Registr (Регистрационный номер клиента).

10. Разработать класс CTelephonNumber (Телефонный номер), содержащий свойства класса: ID (идентификатор клиента), Family (фамилия клиента), PhoneAddress (адрес клиента), PhoneNumber (номер телефона), Value (Ежемесячная плата за телефон).

11. Разработать класс CCount (Счет-фактура), содержащий свойства класса: CountNumber (Номер счет-фактуры), Date (Дата выписки счет-фактуры), Value (Сумма к уплате), Cost (Стоимость приобретения).

12. Разработать класс CRabotnik (Работник автовокзала), содержащий свойства класса: FIO (ФИО работника), Dolg (Должность), Zarplata (оклад).

13. Разработать класс CAgreement (Договор), содержащий свойства класса: Number (Номер договора), Date (Дата договора).

14. Разработать класс CClient (Клиент), содержащий свойства класса: Family (Фамилия), BeginDate (дата въезда), EndDate (Дата отъезда), Summa (Сумма оплаты), NKvit (номер квитанции).

15. Разработать класс CBuy (покупка), содержащий свойства класса: Number (Номер покупателя), Sum (Сумма покупки), Otdel (название отдела магазина, где совершена покупка).

16. Разработать класс CSpec (специальность), содержащий свойства класса: Spec (название специальности), FIO (ФИО работника), Balls (количество баллов, необходимых для поступления).

17. Разработать класс CBilet (Билет), содержащий свойства класса: Time1 (время вылета), Time2 (Время прилета), Cost (цена билета), Name (владелец билета), Nomer (номер места).

18. Разработать класс CHousePit (Домашний питомец), содержащий свойства класса: Nam (Имя животного), Typ (Тип животного), Color (Цвет), Poroda (Порода), Year (Год рождения).

19. Разработать класс CKursWork (Курсовая работа), содержащий свойства класса: Dis (дисциплина), Author (Автор курсовой), Prep (Преподаватель), Met (Название методического пособия), Ball (Оценка).

20. Разработать класс CRoom (Комната для аренды), содержащий свойства класса: Address (Адрес помещения), P (Площадь помещения), Tel (Телефон владельца), Price (Цена аренды).

Контрольные вопросы

1. Что представляет собой класс List<T>?
2. Что представляет собой DataGridView и BindingNavigator?

3. Какой программный код позволит связать экземпляры классов DataGridView и List для отображения списка?

4. Что означает программный код
dataGridView1.AutoGenerateColumns = true?

5. Какое преимущество имеют public-свойства по сравнению с использованием public-полей в объектно-ориентированном программировании?

6. В чем отличие между следующими двумя способами создания объекта:
g=new Grajdanin(«Иванов», «Иван»);
и
g=new Grajdanin{Fam=«Иванов», Nam=«Иван»};

7. В чем отличие между конструктором по умолчанию и конструктором с параметрами?

ИНТЕРФЕЙСЫ И АБСТРАКТНЫЕ КЛАССЫ

Цель работы: изучить способы и преимущества использования интерфейсов и абстрактных классов при реализации наследования.

Краткие теоретические сведения

Наследование, вместе с инкапсуляцией и полиморфизмом, является одним из базовых понятий объектно-ориентированного программирования. Наследование позволяет создавать новые классы, которые повторно используют, расширяют и изменяют поведение, определенное в других классах. Класс, члены которого наследуются, называется базовым классом, а класс, который наследует эти члены, называется производным классом. Производный класс может иметь только один непосредственный базовый класс.

Введем абстрактный класс Shape. Объекты абстрактного класса созданы быть не могут.

Для того, чтобы экземпляры класса Kvadrat и экземпляры других производных классов имели возможность быть прорисованными, необходимо добавить виртуальный метод (virtual-метод) Draw в родительском классе Shape. Параметр g класса Graphics метода Draw предназначен для определения графического контекста, на котором будет осуществляться прорисовка. Однако, поскольку абстрактная фигура Shape не имеет конкретной формы, то необходимо метод Draw сделать абстрактным. По сути абстрактный метод является виртуальным, но без реализации.

```
abstract class Shape
{
	public string Name { get; set; }
	public int X { get; set; }
	public int Y { get; set; }
	abstract public void Draw(Graphics g);
	public Shape()
	{
		Name="Figura";
		X=0;
		Y=0;
	}

	public Shape(string name, int x, int y)
	{
		Name=name;
		X=x;
		Y=y;
	}
}
```

На основе класса Shape будем разрабатывать классы конкретных фигур, например, класс Kvadrat.

```
class Kvadrat : Shape
{
	public int Dlina { get; set; }
	public Kvadrat(): base()
	{
		Dlina = 1;
	}
	public Kvadrat(string name, int x, int y, int dlina)
	: base(name, x, y)
	{
		Dlina = dlina;
	}

	public override void Draw(Graphics g)
	{
		g.DrawPolygon(new Pen(new SolidBrush(Color.Black)),
		new Point[4]
		{
```

```
                new Point(X, Y),
                new Point(X+Dlina, Y),
                new Point(X+Dlina, Y+Dlina),
                new Point(X, Y+Dlina)
            }
            );
        }
}
```

Метод Draw лучше включить в интерфейс IDraw:

```
interface IDraw
{
    void Draw(Graphics g);
}
```

Класс Shape необходимо сделать наследником IDraw. Поскольку в классе, который является наследником интерфейса необходимо описать все методы интерфейса, то в классе Shape необходимо также объявить метод Draw из интерфейса IDraw.

Тогда класс Shape имеет вид:

```
abstract class Shape : IDraw
{
    public string Name { get; set; }
    public int X { get; set; }
    public int Y { get; set; }
    public Shape()
    {
        Name="Figura";
        X=0;
        Y=0;
    }

    public Shape(string name, int x, int y)
    {
        Name = name;
        X = x;
        Y = y;
    }

    public abstract void Draw(Graphics g);
}
```

Для прорисовки на форме необходимо получить графический контекст формы, создав его с помощью программного кода, например, в конструкторе формы:

```
Graphics g = this.CreateGraphics();
```

Метод Draw в классе Shape является абстрактным, поскольку, неизвестно, как рисовать абстрактную фигуру. Однако в классе Kvadrat необходимо перегрузить метод Draw и описать ее реализацию:

```
public override void Draw(Graphics g)
{
    g.DrawPolygon
    (
        new Pen(Color.Black),
        new Point[4]
        {
            new Point(X, Y),
            new Point(X+Dlina, Y),
            new Point(X+Dlina, Y+Dlina),
            new Point(X, Y+Dlina)
        }
    );
}
```

Для создания объекта класса Kvadrat можно использовать следующий код:

```
Kvadrat k = new Kvadrat("Квадрат 1", -63, 4, 122);
```

Этот код можно поместить в обработчик события Click кнопки на форме.

На базе класса Kvadrat можно создать класс «Цветной квадрат», добавив в него поле для хранения цвета фигуры:

```
class ColorKvadrat : Kvadrat
{
    public Color Color { get; set; }
    public ColorKvadrat ( )  : base( )
    {
        Color = Color.Blue;
    }

    public ColorKvadrat(string name, int x, int y, int dlina, Color color)  : base(name, x, y, dlina)
    {
        Color = color;
    }

    public override void Draw(Graphics g)
```

```
        {
            g.DrawPolygon
            (   new Pen(Color),
                new Point[4]
                {
                        new Point(X, Y),
                        new Point(X + Dlina, Y),
                        new Point(X + Dlina, Y + Dlina),
                        new Point(X, Y + Dlina)
                }
            );
        }
}
```

Для вычисления периметра и площади фигуры необходимо добавить интерфейс с методами для вычисления площади и периметра фигуры. Класс Shape сделать наследником интерфейса ICalcPS.

```
interface ICalcPS
{
        /// <summary>
        /// Вычисление периметра
        /// </summary>
        /// <returns>Периметр</returns>
        float P( );

        /// <summary>
        /// Вычисление площади
        /// </summary>
        /// <returns>Площадь</returns>
        float S( );
}
```

В класс Shape добавить абстрактные методы интерфейса ICalcPS с модификатором доступа public:

```
public abstract float P ( );
public abstract float S ( );
```

Реализовать методы интерфейса ICalcPS в классе Kvadrat, перегрузив их:

```
public override float P ( )
{
    return Dlina * 4;
```

```
    }

    public override float S ( )
    {
        return Dlina * Dlina;
    }
```

Класс ColorKvadrat наследует методы P и S без изменений, поскольку формулы для вычисления периметра и площади не меняются с добавлением цвета, а метод Draw необходимо переопределить в связи с необходимостью учета цвета прорисовки. При переопределении метода Draw необходимо прорисовывать геометрическую фигуру закрашенной (метод

g.FillPolygon(new SolidBrush(Color), ….).

Для хранения списка геометрических фигур Kvadrat и ColorKvadrat необходимо создать список, являющийся полем формы.

```
List<Shape> list = new List<Shape>
{
    new Kvadrat(),
    new ColorKvadrat("Цв. квадрат 1", 5, 88, 34, Color.Yellow),
    new ColorKvadrat("Цв. квадрат 2", 45, 58, 134, Color.Green),
    new Kvadrat("Квадрат 1", 145, 158, 64)
};
```

Благодаря тому, что список List<Shape> содержит элементы класса Shape, в него можно помещать элементы любого производного от него класса: в данном случае Kvadrat и ColorKvadrat. Для вызова «правильных методов», то есть определения метода по типу ссылки объекта, реально находящегося в элементе списка, а не по типу, указанному при описании элементов контейнера, методы Draw, P и S в наследниках класса Shape описаны как перегруженные (override).

Задание для выполнения работы

1. Реализовать абстрактный класс Shape (геометрическая фигура) со свойствами, предназначенными для описания координат фигуры X и Y, и имени фигуры.
2. В классе Shape определить конструктор по умолчанию, конструктор с параметрами и конструктор копирования.
3. Определить производный класс «Конкретная геометрическая фигура 1» в соответствии с вариантом на базе класса Shape. Реализовать в полученном классе свойства для описания конкретной фигуры и необходимые конструкторы. Конструкторы

производного класса должны обращаться к конструкторам базового класса и конструкторам текущего класса.

Для обращения к методам базового класса используется ключевое слово base, а текущего класса – this.

При реализации конструкторов класса «Конкретная геометрическая фигура» и свойств необходимо учесть «Способ задания фигуры» в соответствии с индивидуальным заданием.

4. Сделать возможным вывод фигуры на форму. Для этого определить интерфейс IDraw с методом Draw, предназначенным для рисования геометрической фигуры на форму. Учесть, что метод рисования должен получить информацию о том, на какой поверхности нарисовать фигуру (объект класса Graphics). Разработать класс Shape и реализовать в нем интерфейс IDraw. Добавить метод рисования как абстрактный метод в класс Shape.

5. Реализовать методы интерфейса IDraw в производном классе «Конкретная геометрическая фигура 1».

6. В классе Form1 создать объект класса «Конкретная геометрическая фигура» и сохранить ссылку на него в объекте класса Shape и нарисовать на экране.

7. Описать интерфейс ICalcPS с методами для вычисления периметра и площади геометрической фигуры. Наследовать базовый класс Shape от интерфейса ICalcPS.

8. Реализовать методы интерфейса ICalcPS в классе «Конкретная геометрическая фигура 1».

9. Реализовать класс «Конкретная геометрическая фигура2» в соответствии с вариантом. класс «Конкретная геометрическая фигура2» должен быть наследником класса «Конкретная геометрическая фигура1». В классе описать необходимые поля, свойства, конструкторы, методы.

10. Создать список объектов класса Shape и записать в него объекты классов «Конкретная геометрическая фигура 1» и «Конкретная геометрическая фигура2» в произвольном порядке.

11. Нарисовать на экране фигуры из списка с помощью оператора foreach следующим образом

list.ForEach(a => a.Draw(g));

где g — графический контекст формы.

12. Поместить на форму элемент управления DataGridView, добавить в него 2 столбца и вывести в них периметры и площади всех фигур из списка.

Пример для вывода информации о площади фигур в первый столбец DataGridView приведен ниже:

```
int j = 0;
```

```
foreach (var i in list)
{
        s =  i.S().ToString() ;
        dataGridView1.Rows.Add(new DataGridViewRow());
        dataGridView1.Rows[j].Cells[0].Value = s;
        j++;
}
```

13. Реализовать для классов «Конкретная геометрическая фигура 1» и «Конкретная геометрическая фигура 2» метод ToString() для получения строки с описанием фигуры. В описание фигуры включить:

тип, название фигуры, координаты, площадь и периметр.

Метод ToString имеет следующий синтаксис:

```
public override string ToString( )
{
        return <строка_с_информацией_об_объекте>;
}
```

14. Вывести информацию о фигуре, возвращаемую методом ToString(), в третий столбец элемента управления DataGridView.

Варианты индивидуальных заданий

Варианты заданий приведены в таблице 1.

Таблица 1 — Варианты заданий

№	Конкретная геометрическая фигура 1	Конкретная геометрическая фигура 2	Способ задания фигуры
1	Квадрат	Квадрат с заданным цветом заливки	Задается координатами всех четырех вершин
2	Квадрат со сторонами параллельными осям координат	Квадрат с заданным цветом заливки	Задается длиной стороны
3	Треугольник	Цветной треугольник	Задается координатами трех вершин
4	Треугольник	Цветной треугольник	Задается длинами сторон
5	Квадрат с указанной ориентацией	Квадрат с заданным цветом заливки	Задается длиной стороны и углом ориентации
6	Круг	Цветной круг со	Задается длиной

		сплошной заливкой	окружности
7	Круг	Цветной круг с несплошной заливкой	Задается площадью
8	Окружность	Цветная окружность	Задается радиусом
9	Овал	Цветной овал	Задается двумя радиусами
10	Овал	Цветной овал	Задается габаритами
11	Четырехугольник произвольный	Цветной четырехугольник произвольный	Задается координатами четырех вершин
12	Сектор	Цветной сектор	Задается радиусом и углом
13	Сектор	Цветной сектор	Задается радиусом и углом начала и углом конца сектора
14	Ромб	Цветной ромб	Задается длиной двух диагоналей
15	Окружность	Цветная окружность	Задается периметром
16	Овал	Цветной овал	Задается отношением радиусов и наибольшим радиусом
17	Круг	Цветной овал	Задается радиусами
18	Эллиптический сектор	Цветной эллиптический сектор	Задается количеством квадрантов и радиусами
19	Круг с описанным квадратом	Круг с описанным и вписанным квадратом	Задается радиусом
20	4 концентрические окружности	Цветные четыре концентрические окружности. Для каждой окружности задается цвет.	Задается каждым из 4 радиусов

Контрольные вопросы

1. Что такое абстрактный класс?

2. Что такое абстрактный метод?
3. Какие ограничения имеются у абстрактных классов?
4. С какой целью в программу добавлен класс Shape?
5. Что такое интерфейс? Что они могут содержать?
6. Какова цель использования интерфейсов?
7. Объясните назначение класса Graphics.
8. Можно ли в объект list программы включить объекты класса Shape?
9. Что означают ключевые слова virtual и override?

РАБОТА С КОЛЛЕКЦИЯМИ БИБЛИОТЕКИ .NET

Цель работы: изучить методы работы с динамическими списками библиотеки .NET.

Краткие теоретические сведения

Коллекция в C# – совокупность объектов. В среде .NET Framework имеется немало интерфейсов и классов, в которых определяются и реализуются различные типы коллекций. Коллекции упрощают решение различных задач программирования, требующих сложных структур данных. К ним относятся списки, стеки, динамические массивы, очереди, хеш-таблицы.

Главное преимущество коллекций заключается в том, что они стандартизируют обработку групп объектов в программе. В среде .NET Framework поддерживаются четыре типа коллекций: необобщенные, специальные, с поразрядной организацией и обобщенные.

Первоначально существовали только классы необобщенных коллекций. Затем появились обобщенные версии классов и интерфейсов.

Необобщенные коллекции оперируют данными типа Object (базовый класс для всех классов). Таким образом, они служат для хранения объектов любого типа, причем в одной коллекции допускается наличие объектов разных типов. Это преимущество, если нужно оперировать объектами разных типов в одном списке или если типы объектов заранее неизвестны. Но эти коллекции не обеспечивают типовую безопасность. Они размещаются в пространстве имен System.Collections.

Специальные коллекции оперируют данными конкретного типа (например, для символьных строк StringCollection и StringDictionary) или же делают это каким-то особым образом (для бито-

вых элементов длиной 32 бита – BitVector32). Они размещаются в пространстве имен System.Collections.Specialized.

Коллекции с поразрядной организацией поддерживают поразрядные операции над двоичными разрядами (И, ИЛИ). К ним относится только класс BitArray. Они размещаются в пространстве имен System.Collections.

Обобщенные коллекции обеспечивают обобщенную реализацию нескольких стандартных структур данных, включая связанные списки, стеки, очереди и словари. Такие коллекции являются типизированными. Это означает, что в обобщенной коллекции могут храниться только такие элементы данных, которые совместимы по типу с данной коллекцией. Это исключает случайное несовпадение типов и обеспечивают типовую безопасность. Они размещаются в пространстве имен System.Collections.Generic.

Инициализация коллекций осуществляется следующим образом:

```
List<int> myList = new List<int>{0,1,2,3,4,5,6,7,8};
```

Если контейнер управляет коллекцией классов и структур, то можно смешивать синтаксис инициализации объектов с инициализации объектов с синтаксисом инициализации коллекций, создавая некоторый функциональный код:

```
List<Point> m = new List<int>
{
    new Point {X=2, Y=3},
    new Point {X=6, Y=4},
    new Point (Color.Red) {X=2, Y=3}
};
```

Класс List<T>

Класс List<T> представляет собой односвязный список с возможностью обращения к элементам по номерам и имеет следующие методы:

1. Метод AddRange() – добавление множества элементов в коллекцию за один прием. Метод AddRange принимает один аргумент типа IEnumerable<T>

```
points.AddRange(new Point[]
{
    new Point {X=5, Y=6},
    new Point(5,8)
}
);
```

2. Метод Insert() – вставка элемента в определенную позицию.

Например, points.Insert(3, new Point(3,5));

Если указывается индекс, превышающий количество элементов в коллекции, то генерируется исключение типа ArgumentOutOfRangeException.

3. Метод InsertRange() – вставка нескольких элементов в определенную позицию за один прием аналогично AddRange.

4. Доступ к элементам осуществляется с помощью индексатора []. Первый элемент доступен по индексу 0.

Например, Point p = points[0];

4. Проход по элементам коллекции возможен с помощью оператора foreach благодаря реализации интерфейса IEnumerable. Например,

```
foreach(Point p in points)
    Console.WriteLine(p);
```

5. Метод ForEach(), который может использоваться вместо оператора foreach также, объявленный с параметром Action<T>. Action должен быть объявлен как метод с параметром того же типа, что и элемент коллекции и возвращающий void.

Например,

```
points.ForEach(Console.WriteLine);
points.ForEach(p=>Console.WriteLine(p));
```

6. Метод RemoveAt() – удаление элемента по индексу.

7. Метод Remove() – удаление элемента по ссылке.

8. Метод RemoveRange() удаляет множество элементов из коллекции. Первый параметр определяет индекс, начиная с которого располагаются удаляемые элементы, а второй параметр задает количество удаляемых элементов.

10. Метод Clear() – удаление всех элементов коллекции.

11. Метод IndexOf(), LastIndexOf(), FindIndex(), FindLastIndex(), Find(), FindLast() – поиск элемента коллекции.

16. Метод Exists() – проверка существования элемента.

17. Метод Sort() осуществляет сортировку элементов.

Класс LinkedList<T>

Класс LinkedList<T> является представителем двунаправленного списка, где каждый элемент списка содержит ссылки на предыдущий и следующий элементы (рис. 1). Итератор поддерживает обход в обе стороны. Реализует методы получения, удаления и вставки в начало, середину и конец списка.

Рис. 1. Двусвязный список

Преимущество связного списка проявляется в том, что операция вставки элемента в середину выполняется очень быстро. При этом только ссылки Next предыдущего элемента и Previous следующего элемента должны быть изменены так, чтобы указывать на вставляемый элемент (рис. 2).

Рис. 2. Добавление элемента

Естественно, у связных списков есть и свои недостатки. Так, например, все элементы связных списков доступны лишь друг за другом. Поэтому для нахождения элемента, находящегося в середине или конце списка, требуется довольно много времени. Двусвязный список не может просто хранить элементы внутри себя. Вместе с каждым из них ему необходимо иметь информацию о следующем и предыдущем элементах. Для этого LinkedList<T> содержит элементы класса LinkedListNode<T>. Класс LinkedListNode<T> определяет свойства List, Next, Previous и Value. Свойство List возвращает объект LinkedList<T>, ассоциированный с узлом. Свойства Next и Previous предназначены для итераций по списку и для доступа к следующему и предыдущему элементам. Свойство Value типа T возвращает элемент, ассоциированный с узлом.

Сам класс LinkedList<T> определяет члены для доступа к первому (First) и последнему (Last) элементам в списке, для

вставки элементов в определенные позиции (AddAfter(), AddBefore(), AddFirst(), AddLast()), для удаления элементов из заданных позиций (Remove(), RemoveFirst(), RemoveLast()) и для нахождения элементов, начиная поиск либо с начала (Find()), либо с конца (FindLast()) списка.

Пример использования связных списков:

```
// Создание двусвязного списка
LinkedList<int> link = new LinkedList<int>();
// Добавление элементов
link.AddLast(1);
link.AddLast(2);
link.AddFirst(0);  //Будет получен список  «0  1  2»
//Перебор элементов в прямом направлении
LinkedListNode<string> node;
for (node = link.First; node != null; node = node.Next)
    Console.Write(node.Value + "\t");
//Перебор элементов в обратном направлении
for (node = link.Last; node != null; node = node.Previous)
    Console.Write(node.Value + "\t");
//Вывод пятого элемента
for (node = link.First; i<5; node = node.Next)
    Console.Write(node.Value + "\t");
//Добавление после элемента node числа 2
link.AddAfter(node, 2);
```

Класс Stack<T>

Класс Stack<T> представляет коллекцию элементов, работающую по алгоритму «последний вошел – первый вышел» (LIFO) и имеет следующие методы:

Метод Push() — вставка элемента в вершину стека.

Метод Pop() — извлечение элемента из вершины стека.

Метод Peek() — вернуть элемент из вершины стека без его удаления.

При попытке извлечения элемента из пустого стека генерируется исключение InvalidOperationException.

Класс Queue<T>

Класс Queue<T> представляет коллекцию элементов, работающую по принципу «первый вошел – первый вышел» (FIFO) и имеет следующие методы:

Метод Enqueue () — вставка элемента в конец очереди.

Метод Dequeue() — извлечение элемента из начала очереди.

Метод Peek() — вернуть элемент из начала очереди без его удаления.

При попытке извлечения элемента из пустой очереди генерируется исключение InvalidOperationException.

BitArray

Класс BitArray служит для хранения отдельных битов в коллекции. Изменение количества элементов можно осуществить через свойство Length.

```
BitArray bits2 = new BitArray(7);
bits1.SetAll(false);
bits1[3] = true;
bits1.Length = bits1.Length + 1;
bits1[7] = true;
```

Порядок выполнения работы

1. Создайте приложение Windows Forms в среде C#.
2. Создайте исходные и результирующие контейнеры с элементами, являющимися целыми числами, в соответствии с индивидуальным заданием. Обычно необходимо создать 1-2 исходных контейнера и 1-2 — результирующих.
3. Напишите метод для ввода исходных данных и заполнения исходных контейнеров начальными значениями, заданными пользователем.

В случае ввода некорректных значений необходимо, чтобы программа не завершалась аварийно.

4. Напишите метод для выполнения действий, заданных индивидуальным заданием в соответствии с индивидуальным заданием.
5. В случае невозможности выполнения необходимых действий выдать соответствующее сообщение.
6. Выведите содержимое результирующих контейнеров и данные в соответствии с индивидуальным заданием на форму.

Индивидуальные задания

Вариант 1. Исходный контейнер: стек. Результирующий контейнер: стек. Перемещать элементы из исходного контейнера во выходной, пока значение вершины исходного стека не станет четным. Если в исходном стеке нет элементов с четными значениями, то переместить из исходного стека в выходной все элементы. Вывести на экран значения вершин обеих стеков. Если исходный стек стал пустым, то вывести «null».

Вариант 2. Исходный контейнер: стек. Результирующий контейнер: 2 стека. Перемещать элементы из исходного контейнера в первый выходной контейнер все элементы с четными значениями, а во второй – с нечетными. Один из результирующих контейнеров может остаться пустым. Вывести значения вершин обеих выходных контейнеров. Для каждого пустого результирующего контейнера вывести «null».

Вариант 3. Исходный контейнер: очередь. Результирующий контейнер: 2 стека. Переместить в первый из результирующих контейнеров элементы исходного контейнера с четными значениями, а во второй – с нечетными. Один из результирующих контейнеров может остаться пустым. Вывести значения вершин обеих результирующих контейнеров. Для каждого пустого результирующего контейнера вывести «null».

Вариант 4. Исходный контейнер: очередь. Результирующий контейнер: двусвязный список. Извлекать из очереди элементы, пока значение первого элемента очереди не станет четным и заносить значения извлеченных элементов в двусвязный список. Если очередь не содержит элементов с четными значениями, то извлечь все ее элементы. Вывести на экран «null», если очередь стала пустой.

Вариант 5. Исходный контейнер: очередь. Результирующий контейнер: очередь. Переместить из исходного контейнера в результирующий контейнер N элементов. Если исходная очередь содержит менее N элементов, то перенести все элементы из исходной очереди в результирующую. Вывести на экран «null», если первая очередь стала пустой.

Вариант 6. Исходный контейнер: очередь. Результирующий контейнер: очередь. Перемещать элементы исходной очереди в конец результирующей очереди, пока значение начального элемента исходного контейнера не станет четным. Если исходный контейнер не содержит четных элементов, то перенести все элементы из него в результирующий. Вывести на экран «null», если исходный контейнер стал пустым.

Вариант 7. Исходный контейнер: две очереди с одинаковым количеством элементов. Результирующий контейнер: очередь. Перенести в выходной контейнер элементы из исходных контейнеров поочередно. Вывести на экран первый и последний элементы результирующего контейнера.

Вариант 8. Исходный контейнер: двусвязный список. Результирующий контейнер: список. Найти в исходном контейнере первое встретившееся число D (D задается пользователем) и вы-

вести два числа перед найденным и два числа после найденного в результирующий контейнер и на экран. Если перед и после найденного числа не имеется двух элементов, то вместо отсутствующих элементов занести в контейнер 0, а на экран вывести «null».

Вариант 9. Исходный контейнер: двусвязный список. Результирующий контейнер: очередь. Найти в исходном контейнере первое встретившееся число D (D задается пользователем) и вывести два числа перед найденным и два числа после найденного в обратном порядке в результирующий контейнер и на экран. Если перед и после найденного числа не имеется двух элементов, то вместо отсутствующих элементов занести в контейнер 0, а на экран вывести «null».

Вариант 10. Исходный контейнер: очередь и массив с необязательно одинаковым количеством элементов. Результирующий контейнер: двусвязный список. Перенести элементы из исходных контейнеров поочередно в выходной контейнер. В конец результирующего контейнера перенести все оставшиеся элементы из более длинного исходного контейнера.

Вариант 11. Исходный контейнер: двусвязный список. Найти в исходном контейнере элемент равный числу D, и занести число D1 перед ним, а число D2 после него (числа D, D1 и D2 задаются пользователем). На экран вывести количество чисел равных D1 в контейнере.

Вариант 12. Исходный контейнер: двусвязный список. Найти в контейнере все элементы равные числу D1, и занести число D2 перед и после каждого из них (числа D1 и D2 задаются пользователем). На экран вывести количество элементов равных D1 в контейнере. Если их нет, то вывести «null».

Вариант 13. Исходный контейнер: двусвязный список и очередь. Найти в двусвязном списке первое попавшееся число D (число D задаются пользователем) и занести перед ним элементы из очереди до тех пор, пока первым элементом очереди не станет число D. Если в очереди нет числа, равного D, то перенести из очереди все элементы. На экран вывести количество и первый элемент очереди. Если очередь пуста, то вывести «null».

Вариант 14. Исходный контейнер: двусвязный список. Продублировать в нем первый и последний элементы. Продублированные элементы добавлять перед соответствующими существующими элементами.

Вариант 15. Исходный контейнер: двусвязный список. Продублировать в нем все элементы с нечетными номерами. Продуб-

лированные элементы добавлять перед соответствующими существующими элементами с такими же значениями.

Вариант 16. Исходный контейнер: двусвязный список. Удалить из него все элементы с нечетными номерами и перенести их в очередь. Вывести на экран значения этих элементов.

Вариант 17. Исходный контейнер: двусвязный список. Результирующий контейнер: список. Переместить в выходной контейнер все элементы из него, равные числу D (число D задаются пользователем). Вывести на экран количество перенесенных элементов. Если перенесенных элементов нет, то на экран вывести «null».

Вариант 18. Исходный контейнер: список. Результирующий контейнер: битовый массив такой же размерности. В выходном контейнере элементы в позиции i должны принимать значение true, если число i-ой позиции исходного контейнера отрицательное и false – в противном случае. Если в исходном контейнере нет отрицательных чисел, то необходимо на экран вывести «null».

Вариант 19. Исходный контейнер: список. Результирующий контейнер: битовый массив такой же размерности. В выходном контейнере элементы в позиции i должны принимать значение true, если число в i-ой позиции исходного контейнера меньше числа в i+1 позиции и false – в противном случае. В последний элемент битового массива занести true, если последний элемент списка положительный и false – в противном случае. Если в результирующем контейнере все элементы являются одинаковыми, то необходимо на экран вывести «null».

Вариант 20. Исходный контейнер: стек и очередь одинаковых размерностей. Результирующий контейнер: битовый массив такой же размерности. В выходном контейнере элементы в позиции i должны принимать значение true, если оба числа в i-ых позициях исходных контейнеров положительные и false – в противном случае. Если в результирующем контейнере не все элементы являются одинаковыми, то необходимо на экран вывести «null».

Контрольные вопросы

1. Какие методы имеются у класса List<T>?
2. Какие методы имеются у класса LinkedList<T>?
3. Какие методы имеются у класса Queue<T>?
4. Какие методы имеются у класса Stack<T>?
5. Какие методы имеются у класса BitArray?
6. Объектами какого класса являются элементы контейнера LinkedList<T>?

7. Как получить третий элемент контейнера LinkedList без использования циклов?

8. Как добавить перед элементом p двусвязного списка LinkedList<int> (p — ссылка на элемент двусвязного списка) элемент t (t — ссылка на экземпляр класса LinkedListNode<int>) без использования метода AddBefore?

9. Как удалить элемент перед элементом двусвязного списка LinkedList<int>, хранящим число 4?

СПИСОК ЛИТЕРАТУРЫ

1. Хейлсберг А., Торгерсен, М., Вилтамут, С., Голд, П. Язык программирования C#. — СПб.: Питер, 2012.

2. Зыков С.В. Введение в теорию программирования. Объектно-ориентированный подход. — М.: ИНТУИТ, 2016.

3. Иванова Г.С. Объектно-ориентированное программирование: учебник. — М.: Изд-во МГТУ, 2003.

4. Кариев Ч.А. Разработка Windows-приложений на основе Visual C#. — М.: ИНТУИТ, 2007.

5. Подбельский В.В. Язык C#. Базовый курс. — М.: Финансы и статистика, 2010.

6. Котов О.М. Язык C#: краткое описание и введение в технологии программирования: учебное пособие. — Екатеринбург: Изд-во Уральского университета, 2014.