Министерство науки и высшего образования Российской Федерации

Казанский национальный исследовательский технический университет – КАИ им. А.Н. Туполева

Институт компьютерных технологий и защиты информации

Отделение СПО ИКТЗИ «Колледж информационных технологий»

ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЕ

Методические указания к лабораторной работе №8

Тема: «Создание и использование библиотеки классов для графических примитивов»

Казань 2022

Составитель преподаватель СПО ИКТЗИ Мингалиев Заид Зульфатович

Методические указания к лабораторным работам по дисциплине «ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЕ» предназначены для студентов направления подготовки 09.02.07 «Информационные системы и программирование»

КНИТУ-КАИ, 2022

ПРОЦЕСС СДАЧИ ВЫПОЛНЕННОЙ РАБОТЫ

По итогам выполнения работы студент:

1. демонстрирует преподавателю правильно работающие программы;
2. демонстрирует приобретённые знания и навыки отвечает на пару небольших вопросов преподавателя по составленной программе, возможностям её доработки;
3. демонстрирует отчет по выполненной лабораторной работе.

Итоговая оценка складывается из оценок по трем указанным составляющим.

ЛАБОРАТОРНАЯ РАБОТА №8
ТЕМА: «СОЗДАНИЕ И ИСПОЛЬЗОВАНИЕ БИБЛИОТЕКИ КЛАССОВ ДЛЯ ГРАФИЧЕСКИХ ПРИМИТИВОВ»

ЦЕЛЬ РАБОТЫ

Приобрести умения и практические навыки для разработки приложения по созданию иерархии классов графических примитивов.

ХОД РАБОТЫ

1. Пространство System.Drawing

На примере графики наглядно видны преимущества ООП, смысл использования классов, их методов и свойств. Добавляя в пространство имен своего проекта соответствующие библиотеки, вы получаете сразу набор инструментов, необходимых для графики. Это графические примитивы (линии, прямоугольники, эллипсы и т.п.), перо для черчения, кисть для закраски и много других полезных объектов и методов.

2D-графика делится на растровую и векторную. Растровое изображение – это набор цветных пикселей, заданных в прямоугольной области, хранящихся в файлах *.bmp, *.jpg, *.png и т.п. Самый простой растровый редактор — программа Paint. Векторная графика намного экономнее (по объемам памяти) растровой. Так для рисования прямоугольника достаточно задать координаты двух точек (левого верхнего и правого нижнего углов) и цвет и толщину линии.

Для рисования графики используется **интерфейс GDI+** – это модель рисования общего назначения для приложений .NET. В среде .NET интерфейс GDI+ используется в нескольких местах, в том числе при отправке документов на принтер, отображения графики в Windows-приложениях и визуализации графических элементов на веб-странице.

Ядром программирования GDI+ является пространство имен **System.Drawing**, которое обеспечивает доступ к функциональным возможностям графического интерфейса GDI+, используя около 50 (!) классов, в том числе класс **Graphics**.

Класс **Graphics** предоставляет методы рисования на устройстве отображения (другие термины — графический контекст, «холст»).

2. Создание абстрактного суперкласса

Для создания классов под графические примитивы изначально необходимо создать абстрактный класс «Фигура». Абстрактной фигуры самой по себе не существует. В то же время может потребоваться определить для всех фигур какой-то общий класс, который будет содержать общую для всех функциональность. И для описания подобных сущностей используются абстрактные классы.

Создаем внутри проекта новый класс Figure.cs и описываем его свойства и методы, которые будут являться общими для всех дочерних от него классов:

Листинг 8.1 – Абстрактный класс «Фигура»

1	`abstract public class Figure`	При определении абстрактных классов используется ключевое слово `abstract`
2	`{`	
3	`    public int x;`	Координата базовой точки фигуры по оси абсцисс
4	`    public int y;`	Координата базовой точки фигуры по оси ординат
5	`    public int w;`	Ширина фигуры
6	`    public int h;`	Высота фигуры
7	`    abstract public void Draw();`	Абстрактный метод прорисовки фигуры
8	`    abstract public void MoveTo(int x, int y);`	Абстрактный метод по перемещению фигуры
9	`}`	

3. Рисование простых графических примитивов

Для рисования фигур используется визуальная компонента из пространства имен `System.Windows.Forms` – `PictureBox`. Для задания графического контекста на данном элементе используется растровый объект класса `Bitmap`. В классе формы задается два объекта:

Листинг 8.2 – Создание инструментов рисования

```
1  Bitmap bitmap = new
   Bitmap(pictureBox1.ClientSize.Width,
   pictureBox1.ClientSize.Height);
2  Pen pen = new Pen(Color.Black, 5);
```

Класс `Bitmap` предоставляет объект, являющийся буфером для хранения bitmap-изображений. При вызове конструктора класса указывается ширина и длина элемента. В листинге 8.2 в качестве первого аргумента вызываемого конструктора используется свойство `ClientSize.Width`, которое возвращает значение ширины клиентской области элемента управления, а в качестве второго аргумента – `ClientSize.Height`, возвращающее значение длины элемента.

Класс `Pen` (перо) используется для рисования линий и кривых. В листинге 8.2 в качестве первого аргумента вызываемого конструктора класса `Pen` передается цвет пера, используемый для рисования фигур. Второй параметр конструктора – толщина пера в пикселях.

Для удобства реализации методов прорисовки фигур создадим отдельный статический класс `Init`, который будет хранить ссылки на объект `Bitmap`, `PictureBox` и `Pen`, которые будут являться статическими полями. Нельзя создавать экземпляры статического класса, поэтому доступ к членам статического класса в дальнейшей будет осуществляться с использованием самого имени класса `Init`. Статический класс содержит только статические поля, а на уровне памяти для них будет создаваться участок в памяти, который будет общим для всех объектов класса. Благодаря этому свойство можно получить доступ к инструментам рисования из любого участка кода.

Листинг 8.3 – Инициализация инструментов рисования

```
1  static class Init
2  {
3      public static Bitmap bitmap;
4      public static PictureBox pictureBox;
5      public static Pen pen;
6  }
```

Инициализацию описанных в листинге 8.3 статических свойств произведем в конструкторе главной формы `Form1.cs`.

Листинг 8.4 – Конструктор главной формы

```
1  public Form1()
2  {
3          InitializeComponent();
4          this.bitmap = new
   Bitmap(pictureBox1.ClientSize.Width,
   pictureBox1.ClientSize.Height);
5          this.pen = new Pen(Color.Black, 5);
6          Init.bitmap = this.bitmap;
7          Init.pictureBox = pictureBox1;
8          Init.pen = this.pen;
9  }
```

Далее создается графический интерфейс для прорисовки фигур (Рисунок 1.1).

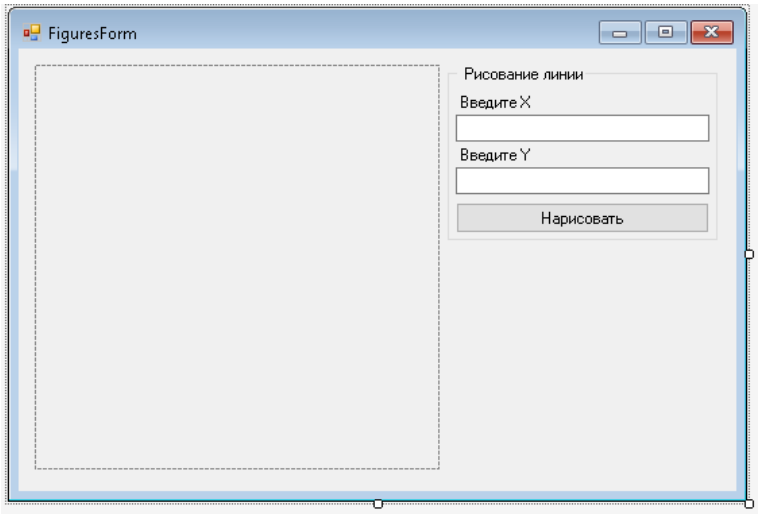

Рисунок 8.1 – Окно создания фигур

В окне размещаются следующие элементы: визуальная компонента `PictureBox` для рисования, контейнер `GroupBox` для вставки в него элементов управления, два элемента `TextBox` для ввода координат по оси

абсцисс и ординат базовой точки фигуры и кнопка Button «Нарисовать» для прорисовки фигуры.

После заполнения формы создается обработчик события нажатия кнопки «Нарисовать». В метод обработчик события добавляется следующий программный код для создания и прорисовки фигуры:

Листинг 8.5 – Метод создания и прорисовки фигуры

```
1  private void buttonDraw_Click(object sender,
   EventArgs e)
2  {
3          Graphics g = Graphics.FromImage(Init.bitmap);
4          g.DrawLine(Init.pen,
   float.Parse(textBoxCoordX.Text),
   float.Parse(textBoxCoordY.Text),
   float.Parse(textBoxCoordX.Text)          *          2,
   float.Parse(textBoxCoordY.Text));
5          Init.pictureBox.Image = Init.bitmap;
6  }
```

В строке 5создается объект g класса Graphics, представляющий собой некий «холст» для рисования фигур. Метод FromImage возвращает новый объект класса Graphics из объекта bitmap. На основе bitmap будет создан новый объект Graphics.

В строке 6 вызывается метод DrawLine, который проводит линию, соединяющую две точки, задаваемые парами координат. В качестве первого аргумента при вызове метода передается структура Pen, определяющая цвет, ширину и стиль линии, второй-третий аргумент – координаты X-Y первой точки, четвертый-пятый аргумент – координаты X-Y второй точки.

Результат работы разработанного метода представлен на рисунке 8.2.

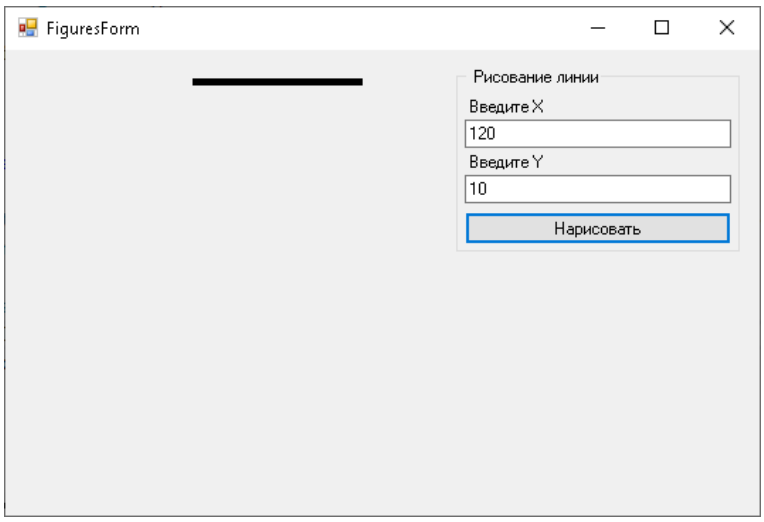

Рисунок 8.1 – Прорисовка линии

Стоим отметить, что начальной точкой в системе координат и **базовой точкой фигуры** является верхняя левая точка.

3. Структура «Точка»

Структура `Point` представляет упорядоченную пару целых чисел — координат X и Y, определяющую точку на двумерной плоскости.

Экземпляры данной структуры используются для проектирования графических примитивов.

Рассмотрим пример прорисовки линии с помощью точек, соединяющих точки фигуры. Для этого используем метод

```
public void DrawLine (System.Drawing.Pen pen,
System.Drawing.Point pt1, System.Drawing.Point pt2);
```

где соединяются два экземпляра `pt1` и `pt2` структуры в линию.

Листинг 8.6

```
1  public void DrawLinePoint(PaintEventArgs e)
2  {
3      Graphics g = Graphics.FromImage(Init.bitmap);
4      Point point1 = new Point(100, 100);
5      Point point2 = new Point(500, 100);
6      g.DrawLine(Init.pen, point1, point2);
7      pictureBox1.Image = Init.bitmap;
8  }
```

Код выполняет следующие действия.

- Создает объект `Graphics` для применения средств рисования;
- Создает две точки, являющихся конечными точками линии;
- Отображает линию на экране.

4. Методы класса `Graphics` для прорисовки фигур

В таблице 8.1 представлены наиболее важные методы класса `Graphics` для прорисовки фигур.

Таблица 8.1 – Методы прорисовки фигур[1]

Прототип метода	**Описание параметров метода**			**Описание метода**
	Параметр	**Тип данных параметра**	**Описание**	
`public void DrawArc (System.Drawing.Pen pen, int x, int y, int width, int height, int startAngle, int sweepAngle);`	pen	Pen	Объект Pen, определяющий цвет, ширину и стиль дуги	Рисует дугу, которая является частью эллипса, заданного парой координат, шириной и высотой
	x	Int32	Координата X верхнего левого угла прямоугольника, определяющего эллипс	
	y	Int32	Координата Y верхнего левого угла прямоугольника, определяющего эллипс	
	width	Int32	Ширина прямоугольника, определяющего эллипс	
	height	Int32	Высота прямоугольника, определяющего эллипс	
	startAngle	Int32	Угол (в градусах), который измеряется по	

[1] Полный список свойств и методов класса Graphics представлен здесь https://docs.microsoft.com/ru-ru/dotnet/api/system.drawing.graphics?view=netframework-4.8

			часовой стрелке, начиная от оси X и заканчивая начальной точкой дуги	
	sweepAngle	Int32	Угол (в градусах), который измеряется по часовой стрелке, начиная от значения параметра startAngle и заканчивая конечной точкой дуги	
`public void DrawRectangle (System.Drawing.Pen pen, int x, int y, int width, int height);`	pen	Pen	Структура Pen, определяющая цвет, ширину и стиль прямоугольника	Рисует прямоугольник, определяемый парой координат, шириной и высотой
	x	Int32	Координата X верхнего левого угла прямоугольника для рисования	
	y	Int32	Координата Y верхнего левого угла прямоугольника для рисования	
	width	Int32	Ширина прямоугольника для рисования	
	height	Int32	Высота прямоугольника для рисования	

public void DrawEllipse (System.Drawing .Pen pen, int x, int y, int width, int height);	pen	Pen	Структура Pen, определяющая цвет, ширину и стиль эллипса	Рисует эллипс, определяемый ограничивающим прямоугольником, заданным с помощью координат верхнего левого угла прямоугольника, высоты и ширины
	x	Int32	Координата X верхнего левого угла ограничивающего прямоугольника, который определяет эллипс	
	y	Int32	Координата Y верхнего левого угла ограничивающего прямоугольника, который определяет эллипс	
	width	Int32	Ширина ограничивающего прямоугольника, который определяет эллипс	
	height	Int32	Высота ограничивающего прямоугольника, который определяет эллипс	
	s	String	Строка для рисования	

public void DrawString (string s, System.Drawing. Font font, System.Drawing. Brush brush, System.Drawing. PointF point, System.Drawing. StringFormat format);	font	Font	Объект Font (шрифт текста), определяющий формат текста строки	Создает заданную текстовую строку в указанном месте с помощью заданных объектов Brush и Font, используя атрибуты форматирования заданного формата StringFormat.
	brush	Brush	Объект Brush, определяющий цвет и текстуру создаваемого текста	
	point	PointF	Структура PointF (объект «Точка»), задающая верхний левый угол создаваемого текста	
	format	StringFormat	Объект StringFormat, определяющий атрибуты форматирования, такие как межстрочный интервал и выравнивание, которые применяются к создаваемому тексту (необязательный параметр)	
public void DrawPolygon (System.Drawing .Pen pen, System.Drawing.	pen	Pen	Структура Pen, определяющая цвет, ширину и стиль многоугольника	Рисует многоугольник, определяемый массивом
	points	Point[]	Массив структур Point, которые представляют	

Point[] points);			вершины многоугольника	структур Point

5. Разработка класса «Квадрат»

Рассмотрим пример описания класса Square.cs. Класс «Квадрат» будет дочерним классом от базового класса «Прямоугольник». Поэтому предварительно опишем класс «Прямоугольник».

Класс «Прямоугольник» (Rectangle) будет наследников абстрактного класса Figure. Он будет наследовать следующие свойства:

Таблица 8.2

`public int x;`	Координата по оси абсцисс верхнего левого угла прямоугольника для рисования
`public int y;`	Координата по оси ординат верхнего левого угла прямоугольника для рисования
`public int w;`	Ширина прямоугольника для рисования
`public int h;`	Длина прямоугольника для рисования

Создадим для класса `Rectangle` два конструктора: конструктор по умолчанию, создающий прямоугольник нулевой размерности, и конструктор инициализации, создающий конструктор с заданными значениями координат базовой точки, ширины и длины.

Листинг 8.7 – Конструкторы класса «Прямоугольник»

```
1  public Rectagle(int x, int y, int w, int h)
2  {
3  this.x = x;
4  this.y = y;
5  this.w = w;
6  this.h = h;
7  }
8  public Rectagle()
9  {
10 this.x = 0;
11 this.y = 0;
12 this.w = 0;
13 this.h = 0;
14 }
```

Далее создаем переопределенный метод прорисовки прямоугольника `Draw`. Для этого будем использовать метод `DrawRectangle`.

Листинг 8.8 – Метод прорисовки

```
1  public override void Draw()
2  {
3  Graphics g = Graphics.FromImage(Init.bitmap);
4  g.DrawRectangle(Init.pen, this.x, this.y, this.w,
   this.h);
5  Init.pictureBox.Image = Init.bitmap;
6  }
```

Форма для задания значений параметров прямоугольника представлена на рисунке 8.3.

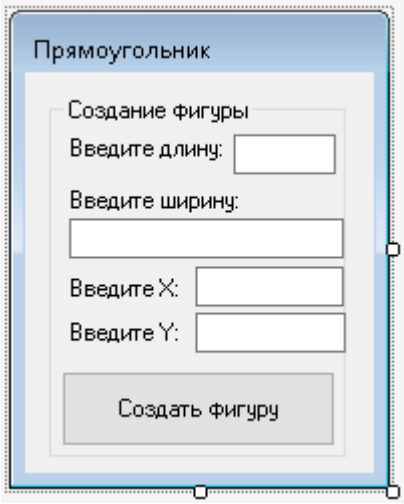

Рисунок 8.2 – Форма для создания прямоугольника

Использование метода прорисовки может выглядеть следующим образом:

Листинг 8.9 – Прорисовка прямоугольника

```
1  Rectagle rectagle = new
   Rectagle(int.Parse(textBoxRectX.Text),
   int.Parse(textBoxRectY.Text),
   int.Parse(textBoxW.Text), int.Parse(textBoxH.Text));
2  ShapeContainer.AddFigure(this.rectagle);
3  rectagle.Draw();
```

Здесь `ShapeContainer` представляет собой контейнерный класс для хранения коллекций графических примитивов.

Листинг 8.10 – Контейнер фигур

```
1  class ShapeContainer
2  {
3      public static List <Figure> figureList;
4      public ShapeContainer()
```

```
5       {
6           figureList = new List<Figure>();
7       }
8       public static void AddFigure(Figure figure)
9       {
10          figureList.Add(figure);
11      }
12  }
```

В строке 3 создается типизированная коллекция, где в качестве хранимого типа указывается базовый абстрактный класс `Figure`. Поскольку существует неявное преобразование производного класса в его базовый класс, то можно в данном контейнере хранить любые экземпляры классов, производных от `Figure`.

В классе `Square`, который является производным от класса `Rectagle`, достаточно реализовать метод-конструктор. Используя геометрическое свойство равенства сторон квадрата, можно реализовать его создание через вызов соответствующего конструктора, определив в его списке инициализации конструктор родительского класса:

Листинг 8.11

```
1   public class Square : Rectagle
2   {
3       public Square(int x, int y, int w) : base(x, y, w,
    w)
4       {   }
5   }
```

В листинге 8.11 параметр `w` используется как сторона квадрата, поэтому он передается в конструктор родительского класса дважды – как ширина и длина прямоугольника одновременно.

6. Удаление и перемещение графического примитива

Для реализации методов удаления и перемещения в классе «Квадрат» необходимо переопределить виртуальные методы абстрактного класса в классе «Прямоугольник», который является базовым.

Метод удаления прямоугольника `DeleteF` будет реализован как член абстрактного класса `Figure`:

Листинг 8.12

```
1  public void DeleteF(Figure figure, bool flag = true)
2  {
3  if (flag == true)
4  {
5  Graphics g = Graphics.FromImage(Init.bitmap)
6  ShapeContainer.arrayList.Remove(figure);
7  this.Clear();
8  Init.pictureBox.Image = Init.bitmap;
9  foreach (Figure f in ShapeContainer.arrayList)
10 {
11 f.Draw();
12 }
13 }
14 else
15 {
16 Graphics g = Graphics.FromImage(Init.bitmap)
17 ShapeContainer.arrayList.Remove(figure);
18 this.Clear();
19 pictureBox1.Image = Init.bitmap;
20 foreach (Figure f in ShapeContainer.arrayList)
21 {
22 f.Draw();
23 }
24 ShapeContainer.arrayList.Add(figure);
25 }
26 }
```

Метод будет универсальным, выполняющим удаление любой фигуры из иерархии классов. В качестве первого параметра метод принимает любой объект, который является производным от класса `Figure`. Вторым параметров является логическая переменная `flag`. Если `flag` принимает истинностное значение, то происходит полное удаление фигуры. При ложном значении

логической переменной происходит временное удаление фигуры с целью перемещения ее базовой точки и повторное сохранение ее в контейнер фигур.

При удалении выбранной фигуры изначально удаляем фигуру из контейнера (строка 6).

Далее производим очистку всего «холста» вызовом метода `Clear` (строка 7).

Листинг 8.13 – Реализация метода Clear

```
1  public void Clear()
2  {
3  Graphics g = Graphics.FromImage(Init.bitmap);
4  g.Clear(Color.White);
5  }
```

После очистки холста на визуальную компоненту накладывается пустая битовая карта (строка 8) и на ней прорисовываются все фигуры, которые остались в контейнере (строка 9-12).

При реализации альтернативной ветки метода удаления, который используется при перемещении фигуры, повторяются все вышеописанные инструкции и добавляется добавление в контейнер временно удаленной фигуры, но уже с новыми координатами базовой точки (строка 24).

При реализации метода перемещения прямоугольника необходимо переопределить в классе `Rectagle.cs` виртуальный метод `MoveTo` абстрактного класса.

Листинг 8.14 – Переопределения виртуального метода перемещения прямоугольника

```
1  public override void MoveTo(int x, int y)
2  {
3  if (!((this.x + x < 0 && this.y + y < 0)
   || (this.y + y < 0)
   || (this.x + x > Init.pictureBox.Width && this.y + y <
   0) || (this.x + this.w + x > Init.pictureBox.Width)
   || (this.x + x > Init.pictureBox.Width && this.y + y >
   Init.pictureBox.Height)
   || (this.y + this.h + y > Init.pictureBox1.Height)
```

```
    || (this.x + x < 0 && this.y + y > Init.pictureBox.Height) || (this.x + x < 0)))
4   {
5   this.x += x;
6   this.y += y;
7   this.DeleteF(this, false);
8   this.Draw();
9   }
10  }
```

Метод принимает два параметра – смещение координаты базовой точки по оси абсцисс и по оси ординат. В строке 3 представлено условие, не позволяющее прямоугольнику «выйти» за границы «холста» при изменении координат его базовой точки. В строке 5-6 производится изменение координат базовой точки на обозначенную величину. В строке 7 происходит временное удаление фигуры. После временного удаления производится новая прорисовка перемещенной фигуры через вызов метода `Draw` в строке 8.

7. Создание сложной фигуры

Создадим из графических примитивов более сложную фигуру. В качестве примера возьмем фигуру «Машина», представленную на рисунке 8.4.

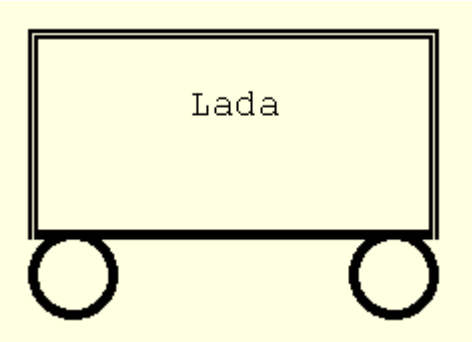

Рисунок 8.4 – Фигура «Машина»

Данная фигура состоит из следующих более простых элементов из разработанной иерархии классов: прямоугольник, строка и две окружности равного радиуса.

Создадим новый класс «Машина» (`Car`), унаследуем его от абстрактного класса `Figure` и инкапсулируем в нем следующие поля:

Листинг 8.15 – Поля класса `Car`

1	`public Rectagle r1`	Прямоугольник – корпус
2	`public Circle c1`	Окружность 1 – шина левая
3	`public Circle c2`	Окружность 2 – шина правая
4	`public String str`	Строка с маркой

Добавим конструктор класса, который будет инициализировать поля класса согласно заданным в форме значениям:

Листинг 8.16 – Конструктор класса `Car`

```
1  public Car(int x, int y, int w, int h, string nameCar)
2  {
3  this.r1 = new Rectagle(x, y, w, h);
4  this.c1 = new Circle(x, y + h, w / 5);
5  this.c2 = new Circle((x + w) - w / 5, y + h, w / 5);
6  this.str = new String(this.r1.x + this.r1.w/2 - 25,
   this.r1.y + this.r1.h/2 - 25, nameCar);
7  }
```

Переопределенные версии метода прорисовки и перемещения фигуры будут выглядеть следующим образом:

Листинг 8.17

```
1  public override void Draw()
2  {
3  this.r1.Draw();
4  this.c1.Draw();
5  this.c2.Draw();
6  this.str.Draw();
7  Init.pictureBox1.Image = Init.bitmap1;
8  }
9  public override void MoveTo(int x, int y)
10 {
11 if (!((this.r1.x + x < 0 && this.r1.y + y < 0) || (this.r1.y + y < 0) || (this.r1.x + x > Init.pictureBox1.Width && this.r1.y + y < 0) || (this.r2.x + this.r1.w + x > Init.pictureBox1.Width) || (this.r1.x + x > Init.pictureBox1.Width && this.r1.y + y > Init.pictureBox1.Height) || (this.r1.y + this.r2.h + y > Init.pictureBox1.Height) || (this.r1.x + x < 0 && this.r1.y+y > Init.pictureBox1.Height) || (this.r1.x + x < 0)))
12 {
13 this.r1.x += x;
14 this.r1.y += y;
15 this.c1.x += x;
16 this.c1.y += y;
17 this.c2.x += x;
18 this.c2.y += y;
19 this.str.x += x;
20 this.str.y += y;
21 this.DeleteF(this, false);
22 this.Draw();
23 }
24 }
```

В методе прорисовки последовательно вызываются методы прорисовки отдельных составных элементов фигуры-машины, а в методе перемещения последовательно на одинаковую величину меняются координаты базовой точки каждого составного элемента.

8. Разработка класса «Многоугольник»

Для реализации прорисовки многоугольников используется метод, имеющий следующий прототип:

```
public void DrawPolygon (System.Drawing.Pen pen,
System.Drawing.PointF[] points);
```

Здесь `points` – это массив структур `Point`, которые представляют вершины многоугольника.

Для задания значений свойств многоугольника создается форма, представленная на рисунке 8.5.

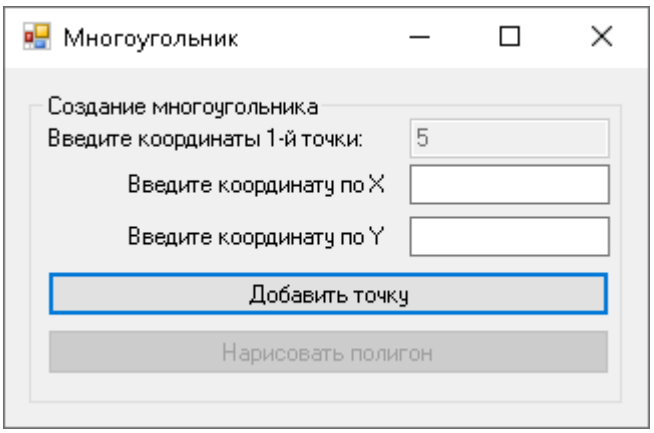

Рисунок 8.5 – Форма создания многоугольника

Порядок создания многоугольника следующий:

1. Вводится количество вершин многоугольника;
2. Нажимается кнопка «Добавить точку», после чего блокируется поле для ввода количества вершин;
3. Поочередно вводятся координаты вершин многоугольника и для вставки создаваемой вершины в массив точек нажимается кнопка «Добавить точку»;
4. После ввода координат всех вершин многоугольника активируется кнопка «Нарисовать полигон», на которую следует нажать для прорисовки многоугольника на битовой карте.

После выполнения вышеописанного порядка действий на форме появится многоугольник.

Метод прорисовки полигона может выглядеть следующим образом:

Листинг 8.18 – Метод прорисовки полигона

```
1  public override void Draw()
2  {
3  Graphics g = Graphics.FromImage(Init.bitmap1);
4  g.DrawPolygon(Init.pen1, pointFs);
5  Init.pictureBox1.Image = Init.bitmap1;
6  }
```

Вышеописанный метод будет вызывать при нажатии кнопки «Нарисовать полигон».

Реализация метода обработчика события нажатия на кнопку «Добавить точку» показана ниже:

Листинг 8.19 – Метод добавления точки

```
1  private void buttonCreatePoint_Click(object sender, EventArgs e)
2  {
3  if (flag == false)
4  {
5  numPoints = int.Parse(textBoxNumPoints.Text);
6  this.pointFs = new PointF[numPoints];
7  flag = true;
8  textBoxNumPoints.Enabled = false;
9  textBoxCoordX.Enabled = true;
10 textBoxCoordY.Enabled = true;
11 label1.Text = $"Введите координаты {i + 1}-й точки: ";
12 }
13 else
14 {
15 if (i != numPoints - 1)
16 {
17 i++;
18 label1.Text = $"Введите координаты {i + 1}-й точки: ";
19 pointFs[i].X = float.Parse(textBoxCoordX.Text);
20 pointFs[i].Y = float.Parse(textBoxCoordY.Text);
21 }
22 else
23 {
24 buttonCreatePoint.Enabled = false;
25 buttonDrawPolygon.Enabled = true;
26 flag = false;
27 }
```

```
28  }
29  }
30  }
```

Если логическая переменная `flag` равна `true`, то происходит определение количества вершин многоугольника. В противном случае происходит добавление точек в массив `pointFs`.

В блоке метода, отвечающего за определение количества вершин многоугольника, целочисленная переменная `numPoints` хранит количество вершин, которое используется в строке 6 для создания массива с размерностью, равной ее значению. В строке 7 изменяется значение логической переменной, так как после определения количества вершин многоугольника необходимо переходить к заданию координат ее вершин. Свойство `Enabled` указывает, может ли элемент управления получить фокус и реагировать на события, вызванные пользователем. Если свойству `Enabled` присвоено значение `True`, то элемент управления может получать фокус и реагировать на события, вызванные пользователем. В противном случае пользователь не может взаимодействовать с элементом управления при помощи мыши и клавиатуры, и он остается доступным только через код.

В блоке метода, отвечающего за добавление точек в массив вершин многоугольника, в строке 18-19 идет перебор всех точек и инициализация их координат значениями, задаваемыми пользователем в форме приложения.

9. Создание библиотеки классов

Библиотека классов определяет типы и методы, которые могут быть вызваны из любого приложения.

Нередко различные классы и структуры оформляются в виде отдельных библиотек, которые компилируются в файлы **dll** и затем могут подключать в другие проекты. Благодаря этому мы можем определить один и тот же функционал в виде библиотеки классов и подключать в различные проекты или передавать на использование другим разработчикам.

Создадим и подключим библиотеку классов для функционирования классов, которые будут присутствовать и в клиентской, и в серверной части приложения.

Возьмем имеющийся проект, созданный на предыдущей лабораторной работе. В структуре проекта нажмем правой кнопкой на название решения и далее в появившемся контекстном меню выберем **Добавить → Создать проект**:

Далее в списке шаблонов проекта найдем пункт **Библиотека классов (.NET Framework)**:

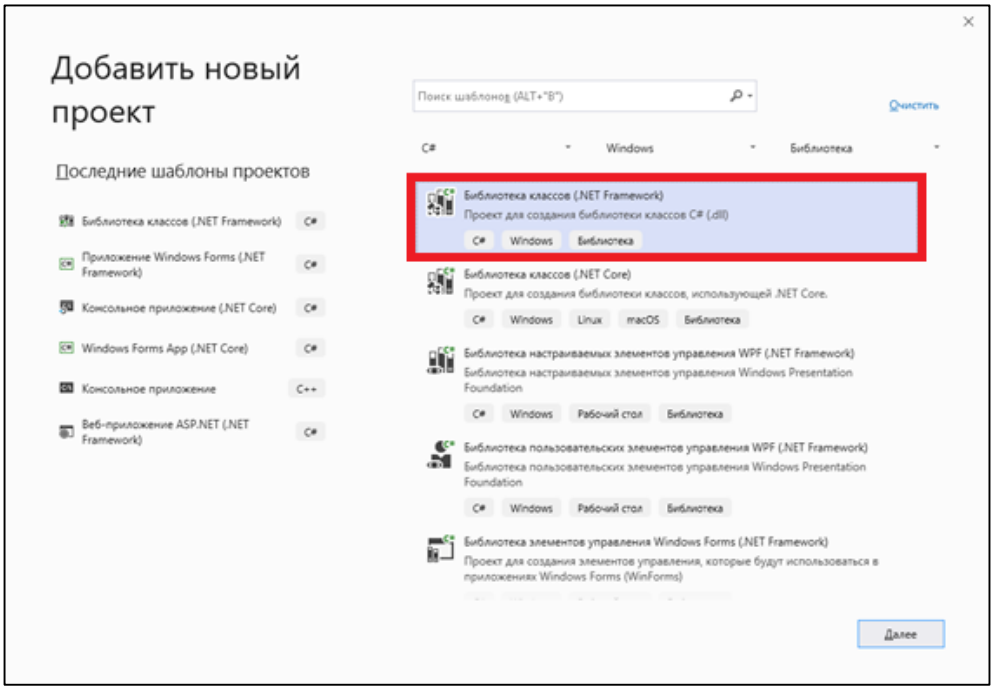

Рисунок 8.6 – Создание библиотеки классов

Затем дадим новому проекту какое-нибудь название, например, `MyLib`.

После этого в решение будет добавлен новый проект, в нашем случае с названием `MyLib`:

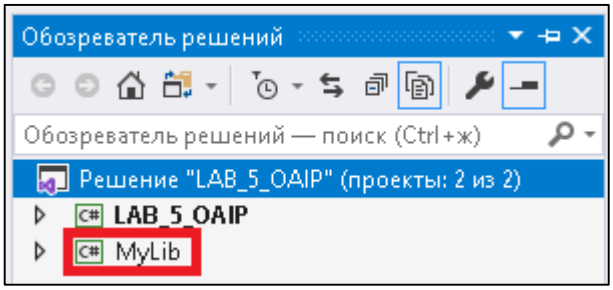

Рисунок 8.7

По умолчанию новый проект имеет один пустой класс `Class1` в файле `Class1.cs`. Мы можем этот файл удалить или переименовать, как нам больше нравится.

Например, переименуем файл `Class1.cs` в `User.cs`, а класс `Class1` в `User`. Определим в классе `User` простейший код для сущности «Пользователь»:

Листинг 8.20 – Класс «Пользователь»

```
1  namespace MyLib
2  {
3  public partial class User
4  {
5  public int Id { get; set; }
6  public string Login { get; set; }
7  public string Password { get; set; }
8  public string Role { get; set; }
9  }
10 }
```

Теперь скомпилируем библиотеку классов. Для этого нажмем правой кнопкой на проект библиотеки классов и в контекстном меню выберем пункт **Перестроить**.

После компиляции библиотеки классов в папке проекта в каталоге **bin/Debug/** мы сможем найти скомпилированный файл **dll** (MyLib.dll). Подключим его в основной проект. Для этого в основном проекте нажмем правой кнопкой на узел **Ссылки** и в контекстном меню выберем пункт **Добавить ссылку…**:

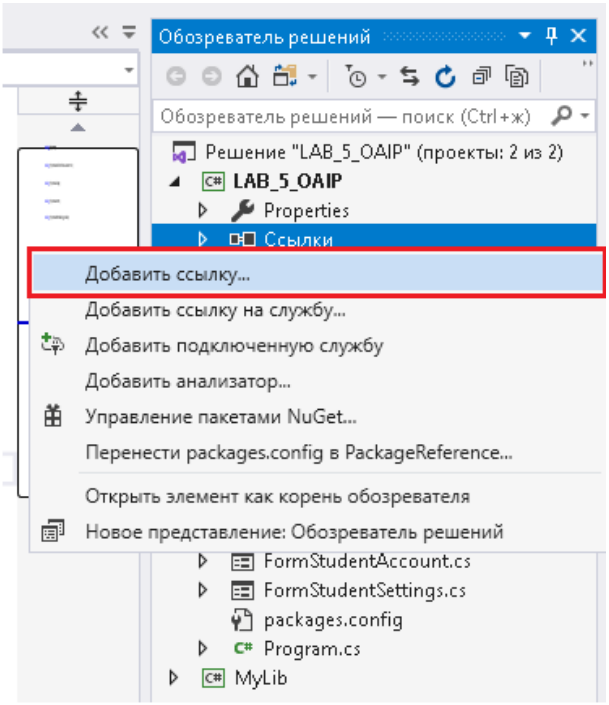

Рисунок 8.8 – Добавление ссылки на библиотеку в основной проект

Далее нам откроется окно для добавления библиотек. В этом окне выберем пункт **Проекты**, который позволяет увидеть все библиотеки классов из проектов текущего решения, поставим отметку рядом с нашей библиотекой и нажмем на кнопку **OK**:

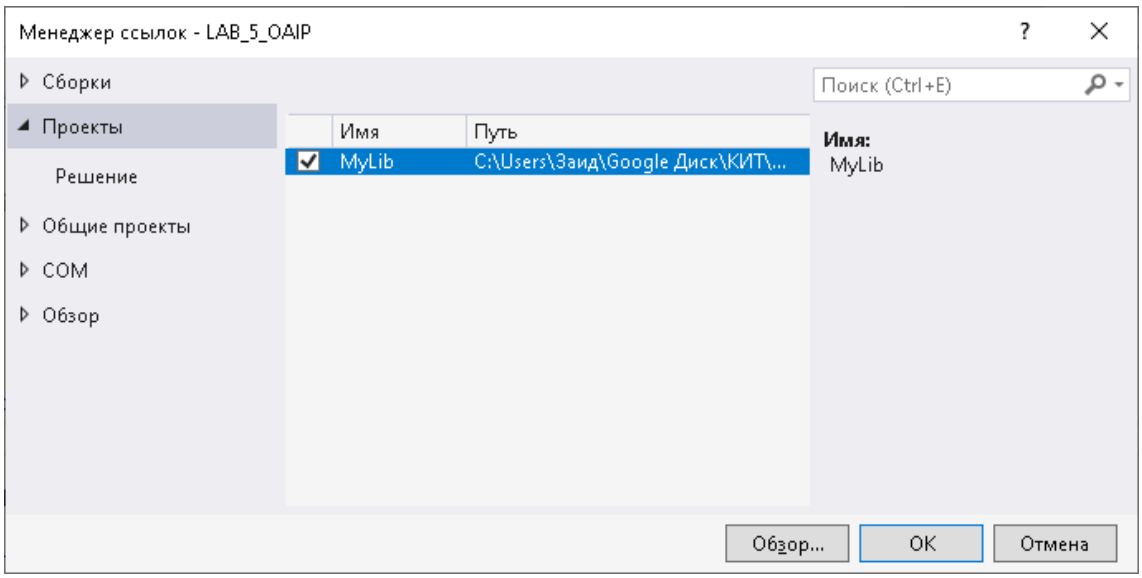

Рисунок 8.9 – Менеджер ссылок

Если наша библиотека вдруг представляет файл dll, который не связан ни с каким проектом в нашем решении, то с помощью кнопки Обзор мы можем найти местоположение файла dll и также его подключить.

После успешного подключения библиотеки в главном проекте изменим класс главной формы `Form1`, чтобы он использовал класс `User` из библиотеки классов:

Листинг 8.21

```
1  using MyLib;
2  using System;
3  using System.IO;
4  using System.Net.Sockets;
5  using System.Windows.Forms;
6  namespace LAB_5_OAIP
7  {
8  public partial class Form1 : Form
9  {
10 …
11 }
12 }
```

В строке 1 происходит подключение пространства имен `MyLib` из библиотеки классов.

После успешного подключения библиотеки в главном проекте можно обращаться к элементам классов, которые определены в библиотеке.

ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ

Требуется создать небольшую иерархию классов, описывающих основные графические примитивы: эллипс, окружность, прямоугольник, квадрат.

Библиотека должна включать следующий **минимальный** набор классов:

- корневой класс фигур;
- дочерний класс эллипсов, наследующий классу фигур (первый уровень наследования);
- дочерний класс прямоугольников, наследующий классу фигур (первый уровень наследования);
- дочерний класс окружностей, наследующий классу эллипсов (второй уровень наследования);
- дочерний класс квадратов, наследующий классу прямоугольников (второй уровень наследования).

Корневой класс фигур должен определять общие свойства и поведение всех объектов-примитивов:

1. координаты базовой точки примитива;
2. конструктор;
3. методы доступа;
4. абстрактные метод прорисовки **Draw**;
5. абстрактный метод перемещения **MoveTo.**

В каждом классе необходимо реализовать:

- конструктор;
- методы прорисовки фигуры;
- метод удаления выбранной фигуры;
- метод перемещения выбранной фигуры.

При реализации метода перемещении необходимо предусмотреть проверку невозможности выхода фигуры за границы области рисования.

Кроме того, классы должны содержать методы, уникальные только для соответствующего поддерева:

- изменение радиуса окружности;
- изменение линейных размеров прямоугольника.

Вся библиотека оформляется в виде одного или нескольких модулей, которые подключаются к основной программе для демонстрации возможностей этой библиотеки.

Добавить в созданную библиотеку классов для графических примитивов следующий набор классов:

- дочерний класс многоугольников, наследующий классу фигур (первый уровень наследования),
- дочерний класс треугольников, наследующий классу многоугольников (второй уровень наследования).

Реализовать класс сложной фигуры, состоящей из простых фигур из иерархии классов. Вид сложной фигуры выбирается согласно индивидуальному варианту, определенного преподавателем.

В каждом классе необходимо реализовать:

- конструктор;
- методы прорисовки фигуры;
- метод удаления выбранной фигуры;
- метод перемещения выбранной фигуры.

Выполнить модификацию созданной ранее библиотеки классов для графических примитивов на основе использования механизма виртуальных методов. Цель – устранение ситуации повторения в каждом классе одинаковых методов перемещения и тем самым реализация универсального метода для перемещения любых графических объектов.