Отчет по лабораторной работе №2 и №3 по курсу C#

10

(количество листов)

Студент группы ИУ5-32

Яценко Антон

Дата: 21.11.2017

Руководитель:

Гапанюк Ю.Е.

Подпись:

Дата:

Задание ко 2 лабораторной работе:

Разработать программу, реализующую работу с классами.

1. Программа должна быть разработана в виде консольного приложения на языке C#.

2. Абстрактный класс «Геометрическая фигура» содержит виртуальный метод для вычисления площади фигуры.

3. Класс «Прямоугольник» наследуется от класса «Геометрическая фигура». Ширина и высота объявляются как свойства (property). Класс должен содержать конструктор по параметрам «ширина» и «высота».

4. Класс «Квадрат» наследуется от класса «Прямоугольник». Класс должен содержать конструктор по длине стороны.

5. Класс «Круг» наследуется от класса «Геометрическая фигура». Радиус объявляется как свойство (property). Класс должен содержать конструктор по параметру «радиус».

6. Для классов «Прямоугольник», «Квадрат», «Круг» переопределить виртуальный метод Object.ToString(), который возвращает в виде строки основные параметры фигуры и ее площадь.

7. Разработать интерфейс IPrint. Интерфейс содержит метод Print(), который не принимает параметров и возвращает void. Для классов «Прямоугольник», «Квадрат», «Круг» реализовать наследование от интерфейса IPrint. Переопределяемый метод Print() выводит на консоль информацию, возвращаемую переопределенным методом ToString().

Задание к 3 лабораторной работе:

Разработать программу, реализующую работу с коллекциями.

1. Программа должна быть разработана в виде консольного приложения на языке C#.

2. Создать объекты классов «Прямоугольник», «Квадрат», «Круг».

3. Для реализации возможности сортировки геометрических фигур для класса «Геометрическая фигура» добавить реализацию интерфейса IComparable. Сортировка производится по площади фигуры.

4. Создать коллекцию класса ArrayList. Сохранить объекты в коллекцию. Отсортировать коллекцию. Вывести в цикле содержимое коллекции.

5. Создать коллекцию класса List<Figure>. Сохранить объекты в коллекцию. Отсортировать коллекцию. Вывести в цикле содержимое коллекции.

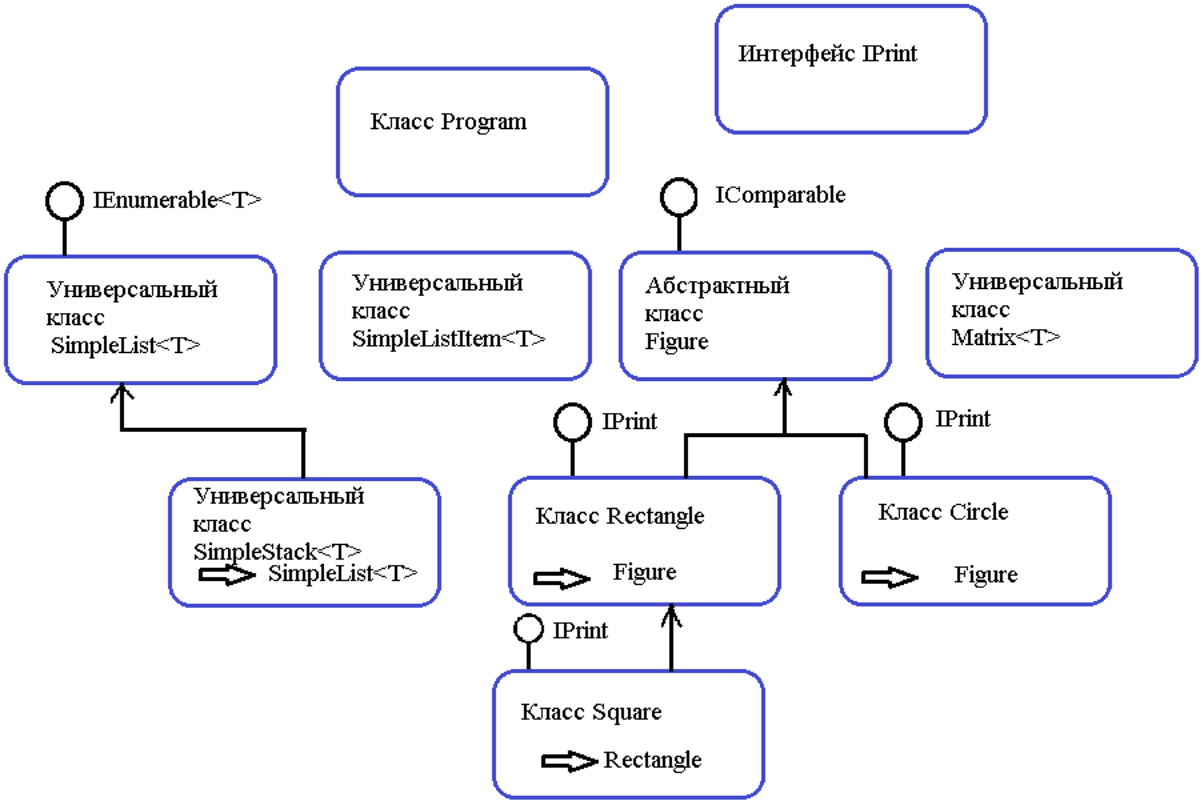
6. Модифицировать класс разреженной матрицы Matrix (представлен в разделе «Вспомогательные материалы для выполнения лабораторных работ») для работы с тремя измерениями – x,y,z. Вывод элементов в методе ToString() осуществлять в том виде, который Вы считаете наиболее удобным. Разработать пример использования разреженной матрицы для геометрических фигур.

7. Реализовать класс «SimpleStack» на основе односвязного списка. Класс SimpleStack наследуется от класса SimpleList (представлен в разделе «Вспомогательные материалы для выполнения лабораторных работ»). Необходимо добавить в класс методы:

- public void Push(T element) – добавление в стек;
- public T Pop() – чтение с удалением из стека.

8. Пример работы класса SimpleStack реализовать на основе геометрических фигур.

Диаграмма классов:



Текст программы:

(Square.cs)

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

namespace lab2.Csh
{
    class Square: Rectangle, Print
    {
        public Square(double size): base(size, size)
        {
            this.Type = "Квадрат";
        }
    }
}
```

(SimpleStack.cs)

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

namespace lab2.Csh
{
    class SimpleStack<T> : SimpleList<T> where T : IComparable
    {

        public void push(T element)
        {
            Add(element);
        }
        public T pop()
        {
            T test=Get(Count-1);
            last = null;
            if (Count > 0) { Count--; }
            return test;
        }
    }
}
```

(SimpleList.cs)

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

namespace lab2.Csh
{
    public class SimpleListItem<T>
    {
        public T data { get; set; }
        public SimpleListItem<T> next { get; set; }
        public SimpleListItem(T param)
        {
```

```
        this.data = param;
    }
}

public class SimpleList<T>: IEnumerable<T> where T : IComparable
{
    protected SimpleListItem<T> first = null;
    protected SimpleListItem<T> last = null;
    public int Count
    {
        get { return _count; }
        protected set { _count = value; }
    }
    int _count;
    public void Add(T element)
    {
        SimpleListItem<T> newItem = new SimpleListItem<T>(element);
        this.Count++;
        if (last == null)
        {
            this.first = newItem;
            this.last = newItem;
        }
        else
        {
            this.last.next = newItem;
            this.last = newItem;
        }
    }

    public SimpleListItem<T> GetItem(int number)
    {
        if ((number < 0) || (number >= this.Count))
        {
            throw new Exception("Выход за границу индекса");
        }
        SimpleListItem<T> current = this.first;
        int i = 0;
        while (i < number)
        {
            current = current.next;
            i++;
        }
        return current;
    }
    public T Get(int number)
    {
        return GetItem(number).data;
    }

    public IEnumerator<T> GetEnumerator()
    {
        SimpleListItem<T> current = this.first;
        while (current != null)
        {
            yield return current.data;
            current = current.next;
        }
    }
    System.Collections.IEnumerator System.Collections.IEnumerable.GetEnumerator()
    {
        return GetEnumerator();
    }
    public void Sort()
    {
```

```
            Sort(0, this.Count - 1);
        }
        private void Sort(int low, int high)
        {
            int i = low;
            int j = high;
            T x = Get((low + high) / 2);
            do
            {
                while (Get(i).CompareTo(x) < 0) ++i;
                while (Get(j).CompareTo(x) > 0) --j;
                if (i <= j)
                {
                    Swap(i, j);
                    i++; j--;
                }
            } while (i <= j);
            if (low < j) Sort(low, j);
            if (i < high) Sort(i, high);
        }
        private void Swap(int i, int j)
        {
            SimpleListItem<T> ci = GetItem(i);
            SimpleListItem<T> cj = GetItem(j);
            T temp = ci.data;
            ci.data = cj.data;
            cj.data = temp;
        }
    }
}
```

(Rectangle.cs)

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

namespace lab2.Csh
{
    class Rectangle: Figure, Print
    {
        double height;
        double width;
        public Rectangle(double ph, double pw)
        {
            this.height = ph;
            this.width = pw;
            this.Type = "Прямоугольник";
        }
        public override double Area()
        {
            double Result = this.width * this.height;
            return Result;
        }
        public void Print()
        {
            Console.WriteLine(this.ToString());
        }
    }
}
```

(Print.cs)

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

namespace lab2.Csh
{
    interface Print
    {
        void Print();
    }
}
```

(Matrix.cs)

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

namespace lab2.Csh
{
    public class Matrix<T>
    {
        Dictionary<string, T> _matrix = new Dictionary<string, T>();
        int maxX;
        int maxY;
        int maxZ;
        T nullElement;
        public Matrix(int px, int py, int pz, T nullElementParam)
        {
            this.maxX = px;
            this.maxY = py;
            this.maxZ = pz;
            this.nullElement = nullElementParam;
        }

        public T this[int x, int y, int z]
        {
            get
            {
                CheckBounds(x, y, z);
                string key = DictKey(x, y,z);
                if (this._matrix.ContainsKey(key))
                {
                    return this._matrix[key];
                }
                else
                {
                    return this.nullElement;
                }
            }
            set
            {
                CheckBounds(x, y, z);
                string key = DictKey(x, y,z);
                this._matrix.Add(key, value);
            }
        }
```

```
            void CheckBounds(int x, int y, int z)
        {
            if (x < 0 || x >= this.maxX) throw new Exception("x=" + x + " выходит за
границы");
            if (y < 0 || y >= this.maxY) throw new Exception("y=" + y + " выходит за
границы");
            if (z < 0 || z >= this.maxY) throw new Exception("z=" + z + " выходит за
границы");
        }
        string DictKey(int x, int y, int z)
        {
            return x.ToString() + "_" + y.ToString()+"_"+z.ToString();
        }
        public override string ToString()
        {
            StringBuilder b = new StringBuilder();
            for (int i = 0; i < this.maxY; i++)
            {

                for (int j = 0; j < this.maxX; j++)
                {
                    b.Append("x=" + i + ", y=" + j+"   ");
                    b.Append("[");
                    for (int k = 0; k < this.maxZ; k++)
                    {
                        if (k > 0) b.Append("\t");
                        b.Append(this[i, j,k].ToString());
                    }
                    b.Append("]\n");
                }
            }
            return b.ToString();
        }
    }
}
```

(Figure.cs)

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

namespace lab2.Csh
{
    abstract class Figure : IComparable
    {
        string _Type;

        public int CompareTo(object obj){
            Figure p = (Figure)obj;
            if (this.Area() < p.Area()) return -1;
            else if (this.Area() == p.Area()) return 0;
            else return 1; //(this.Area() > p.Area())
        }
        public string Type
        {
            get
            {
                return this._Type;
            }
            protected set
            {
                this._Type = value;
```

```
            }
        }
        public override string ToString()
        {
            return this.Type + " площадью " + this.Area().ToString();
        }
        public abstract double Area();
    }
}
```

(Circle.cs)

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

namespace lab2.Csh
{
    class Circle: Figure, Print
    {
        double radius;
        public Circle(double pr)
        {
            this.radius = pr;
            this.Type = "Круг";
        }
        public override double Area()
        {
            double Result = Math.PI * this.radius * this.radius;
            return Result;
        }
        public void Print()
        {
            Console.WriteLine(this.ToString());
        }
    }
}
```

(Program.cs)

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

namespace lab2.Csh
{
    class Program
    {
        static void Main(string[] args)
        {
            Circle c1 = new Circle(3);
            Square s1 = new Square(4);
            Rectangle r1 = new Rectangle(3, 5);
            ArrayList ar = new ArrayList(3);
            ar.Add(c1); ar.Add(s1); ar.Add(r1);
            ar.Sort();
            Console.WriteLine("Отсортированный ArrayList:");
```

```
            foreach(object o in ar)
            {
                Console.WriteLine(o);
            }

            List<Figure> list = new List<Figure>();
            list.Add(c1); list.Add(s1); list.Add(r1);
            list.Sort();
            Console.WriteLine("");
            Console.WriteLine("Отсортированный List<Figure>:");
            foreach (Figure f in list)
            {
                Console.WriteLine(f);
            }

            Console.WriteLine("");
            Matrix<int> m = new Matrix<int>(3, 3, 3, 0);
            Console.Write(m.ToString());

            Console.WriteLine("");
            Console.WriteLine("SimpleStack:");
            SimpleStack<Figure> stack = new SimpleStack<Figure>();
            stack.push(c1);  stack.push(s1); stack.push(r1);
            for (int i = 0; i < 3; i++)
            {
                Console.WriteLine(stack.pop());
            }

            Console.ReadKey();
        }
    }
}
```

Экранная форма: