Отчет по лабораторной работе № 3 по курсу “Базовые компоненты интернет-технологий”

ИСПОЛНИТЕЛЬ:

студент группы ИУ5-33 _____

(подпись)

Желанкина А.С.

"__"_____2017 г.

Москва, МГТУ - 2017

Описание задания

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. 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.

Текст программы

Program.cs

```
using System;
using System.Collections.Generic;
using System.Collections;

namespace Laba2
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("2), 3) Test");

            Circle circle = new Circle(13);
            circle.Print();

            Rectangle rectangle = new Rectangle(3, 5);
            rectangle.Print();

            Square square = new Square(7);
            square.Print();

            Console.WriteLine("4) ArrayList");
            ArrayList array = new ArrayList();

            array.Add(circle);
            array.Add(rectangle);
            array.Add(square);
            foreach (var x in array)
                Console.WriteLine(x);

            Console.WriteLine("Sort: ");
            array.Sort();
            foreach (var x in array)
                Console.WriteLine(x);

            Console.WriteLine("5) List<Figure>");
            List<Figure> list = new List<Figure>();

            list.Add(circle);
            list.Add(rectangle);
            list.Add(square);
            foreach (var x in list)
                Console.WriteLine(x);

            Console.WriteLine("Sort: ");
            list.Sort();
            foreach (var x in list)
                Console.WriteLine(x);

            Console.WriteLine("6) Matrix");
            Matrix<Figure> matrix = new Matrix<Figure>(2, 2, 2, rectangle);
            Console.WriteLine(matrix.ToString());

            Console.WriteLine("7), 8) SimpleList");
            SimpleList<Figure> simple = new SimpleList<Figure>();

            simple.Add(circle);
            simple.Add(rectangle);
            simple.Add(square);
```

```
            foreach (var x in simple)
                Console.WriteLine(x);

            simple.Sort();
            Console.WriteLine("Sort");
            foreach (var x in simple)
                Console.WriteLine(x);

            Console.WriteLine("7), 8) SimpleStack");
            SimpleStack<Figure> stack = new SimpleStack<Figure>();
            stack.Push(circle);
            stack.Push(rectangle);
            stack.Push(square);

            while (stack.Count > 0)
            {
                Figure figure = stack.Pop();
                Console.WriteLine(figure);
            }

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

Circle.cs

```
using System;

namespace Laba2
{
    class Circle : Figure, IPrint
    {
        /// <summary>
        /// Ширина
        /// </summary>
        double radius;
        /// <summary>
        /// Основной конструктор
        /// </summary>
        /// <param name="ph">Высота</param>
        /// <param name="pw">Ширина</param>
        public Circle(double pr)
        {
            radius = pr;
            Type = "Circle";
        }
        public override double Area()
        {
            double Result = Math.PI * radius * radius;
            return Result;
        }
        public void Print()
        {
            Console.WriteLine(ToString());
        }
    }

}
```

FigureCollections.cs

```
using System;

namespace Laba2
```

```
{
        abstract class Figure : IComparable
        {
            /// <summary>
            /// Тип фигуры
            /// </summary>
            public string Type
            {
                get
                {
                    return _Type;
                }
                protected set
                {
                    _Type = value;
                }
            }
            string _Type;
            /// <summary>
            /// Вычисление площади
            /// </summary>
            /// <returns></returns>
            public abstract double Area();
            /// <summary>
            /// Приведение к строке, переопределение метода Object
            /// </summary>
            /// <returns></returns>
            public override string ToString()
            {
                return Type + " has an area equal to " + Area().ToString();
            }
            /// <summary>
            /// Сравнение элементов (для сортировки списка)
            /// </summary>
            /// <param name="obj"></param>
            /// <returns></returns>
            public int CompareTo(object obj)
            {
                Figure p = (Figure)obj;
                if (Area() < p.Area()) return -1;
                else if (Area() == p.Area()) return 0;
                else return 1; //(this.Area() > p.Area())
            }
        }
}
```

IPrint.cs

```
namespace Laba2
{
    interface IPrint
    {
        void Print();
    }
}
```

Rectangle.cs

```
using System;

namespace Laba2
{
    class Rectangle : Figure, IPrint
    {
```

```
        /// <summary>
        /// Высота
        /// </summary>
        double height;
        /// <summary>
        /// Ширина
        /// </summary>
        double width;
        /// <summary>
        /// Основной конструктор
        /// </summary>
        /// <param name="ph">Высота</param>
        /// <param name="pw">Ширина</param>
        public Rectangle(double ph, double pw)
        {
            height = ph;
            width = pw;
            Type = "Rectangle";
        }
        /// <summary>
        /// Вычисление площади
        /// </summary>
        public override double Area()
        {
            double Result = width * height;
            return Result;
        }
        public void Print()
        {
            Console.WriteLine(ToString());
        }
    }
}
```

Square.cs

```
namespace Laba2
{
    class Square : Rectangle, IPrint
    {
        public Square(double size)
        : base(size, size)
        {
            Type = "Square";
        }
    }
}
```

matrix.cs

```
using System;
using System.Collections.Generic;
using System.Text;

namespace Laba2
{
    public class Matrix<T>
    {
        /// <summary>
        /// Словарь для хранения значений
        /// </summary>
        Dictionary<string, T> _matrix = new Dictionary<string, T>();
        /// <summary>
        /// Количество элементов по горизонтали (максимальное количество столбцов)
```

```
        /// </summary>
        int maxX;
        /// <summary>
        /// Количество элементов по вертикали (максимальное количество строк)
        /// </summary>
        int maxY;
        /// <summary>
        /// Количество элементов в глубину
        /// </summary>
        int maxZ;
        /// <summary>
        /// Пустой элемент, который возвращается если элемент с нужными координатами не
был задан
        /// </summary>
        T nullElement;
        /// <summary>
        /// Конструктор
        /// </summary>
        public Matrix(int px, int py, int pz, T nullElementParam)
        {
            maxX = px;
            maxY = py;
            maxZ = pz;
            nullElement = nullElementParam;
        }
        /// <summary>
        /// Индексатор для доступа к данных
        /// </summary>
        public T this[int x, int y, int z]
        {
            get
            {
                CheckBounds(x, y, z);
                string key = DictKey(x, y, z);
                if (_matrix.ContainsKey(key))
                {
                    return _matrix[key];
                }
                else
                {
                    return nullElement;
                }
            }
            set
            {
                CheckBounds(x, y, z);
                string key = DictKey(x, y, z);
                _matrix.Add(key, value);
            }
        }
        /// <summary>
        /// Проверка границ
        /// </summary>
        void CheckBounds(int x, int y, int z)
        {
            if (x < 0 || x >= maxX) throw new Exception("x = " + x + " comes out");
            if (y < 0 || y >= maxY) throw new Exception("y = " + y + " comes out");
            if (z < 0 || z >= maxZ) throw new Exception("z = " + z + " comes out");
        }
        /// <summary>
        /// Формирование ключа
        /// </summary>
        string DictKey(int x, int y, int z)
        {
            return x.ToString() + "_" + y.ToString() + "_" + z.ToString();
```

```
        }
        /// <summary>
        /// Приведение к строке
        /// </summary>
        /// <returns></returns>
        public override string ToString()
        {
            //Класс StringBuilder используется для построения длинных строк
            //Это увеличивает производительность по сравнению с созданием и склеиванием
            //большого количества обычных строк
            StringBuilder b = new StringBuilder();
            for (int k = 0; k < maxZ; k++)
            {
                b.Append("[");
                for (int j = 0; j < maxY; j++)
                {
                    if (j > 0) b.Append("\t");
                    b.Append("[");
                    for (int i = 0; i < maxX; i++)
                    {
                        b.Append(this[i, j, k].ToString());
                        if (i != (maxX - 1)) b.Append(", ");
                    }
                    b.Append("]");
                }
                b.Append("]\n");
            }
            return b.ToString();
        }
    }
}
```

SimpleList.cs

```
using System;
using System.Collections.Generic;

namespace Laba2
{
    /// <summary>
    /// Элемент списка
    /// </summary>
    public class SimpleListItem<T>
    {
        /// <summary>
        /// Данные
        /// </summary>
        public T data { get; set; }
        /// <summary>
        /// Следующий элемент
        /// </summary>
        public SimpleListItem<T> next { get; set; }
        ///конструктор
        public SimpleListItem(T param)
        {
            data = param;
        }
    }
    /// <summary>
    /// Список
    /// </summary>
    public class SimpleList<T> : IEnumerable<T>
    where T : IComparable
    {
```

```
/// <summary>
/// Первый элемент списка
/// </summary>
protected SimpleListItem<T> first = null;
/// <summary>
/// Последний элемент списка
/// </summary>
protected SimpleListItem<T> last = null;
/// <summary>
/// Количество элементов
/// </summary>
public int Count
{
    get { return _count; }
    protected set { _count = value; }
}
int _count;
/// <summary>
/// Добавление элемента
/// </summary>
/// <param name="element"></param>
public void Add(T element)
{
    SimpleListItem<T> newItem = new SimpleListItem<T>(element);
    Count++;
    //Добавление первого элемента
    if (last == null)
    {
        first = newItem;
        last = newItem;
    }
    //Добавление следующих элементов
    else
    {
        //Присоединение элемента к цепочке
        last.next = newItem;
        //Просоединенный элемент считается последним
        last = newItem;
    }
}
/// <summary>
/// Чтение контейнера с заданным номером
/// </summary>
public SimpleListItem<T> GetItem(int number)
{
    if ((number < 0) || (number >= Count))
    {
        //Можно создать собственный класс исключения
        throw new Exception("Come out!");
    }
    SimpleListItem<T> current = first;
    int i = 0;
    //Пропускаем нужное количество элементов
    while (i < number)
    {
        //Переход к следующему элементу
        current = current.next;
        //Увеличение счетчика
        i++;
    }
    return current;
}
/// <summary>
/// Чтение элемента с заданным номером
/// </summary>
```

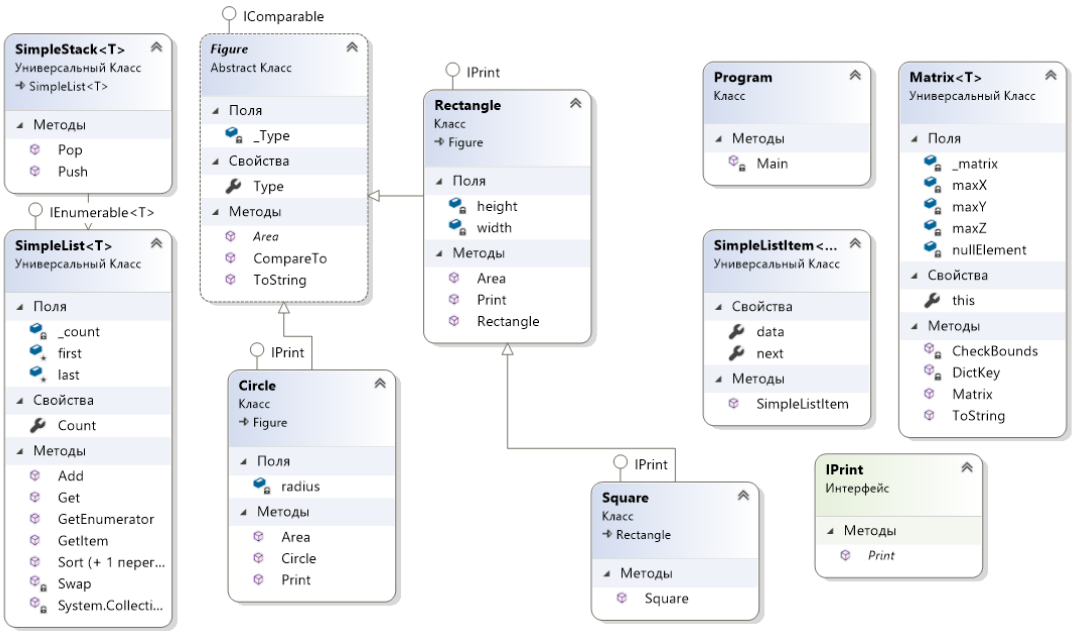
```
        public T Get(int number)
        {
            return GetItem(number).data;
        }
        /// <summary>
        /// Для перебора коллекции
        /// </summary>
        public IEnumerator<T> GetEnumerator()
        {
            SimpleListItem<T> current = first;
            //Перебор элементов
            while (current != null)
            {
                //Возврат текущего значения
                yield return current.data;
                //Переход к следующему элементу
                current = current.next;
            }
        }
        //Реализация обощенного IEnumerator<T> требует реализации необобщенного
интерфейса
        //Данный метод добавляется автоматически при реализации интерфейса
        System.Collections.IEnumerator
       System.Collections.IEnumerable.GetEnumerator()
        {
            return GetEnumerator();
        }
        /// <summary>
        /// Сортировка
        /// </summary>
        public void Sort()
        {
            Sort(0, Count - 1);
        }
        /// <summary>
        /// Реализация алгоритма быстрой сортировки
        /// </summary>
        /// <param name="low"></param>
        /// <param name="high"></param>
        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);
        }
        /// <summary>
        /// Вспомогательный метод для обмена элементов при сортировке
        /// </summary>
        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;
        }
    }
}
```

SimpleStack.cs

```
using System;

namespace Laba2
{
    class SimpleStack<T> : SimpleList<T>
    where T:IComparable
    {
        /// <summary>
        /// Добавление в стек
        /// </summary>
        public void Push(T element)
        {
            Add(element);
        }

        /// <summary>
        /// Чтение с удалением из стека
        /// </summary>
        public T Pop()
        {
            T element = Get(Count - 1);

            SimpleListItem<T> listItem = GetItem(Count - 1);
            listItem = null;

            Count--;

            return element;
        }
    }
}
```

Диаграмма классов



Результаты выполнения