МГТУ им. Баумана
Факультет “Информатика и системы управления”
Кафедра “Автоматизированные системы обработки информации и управления”

Отчет по лабораторной работе №3

Выполнил:
Студент группы ИУ5-31
Фонканц Р.В.
Преподаватель:
Гапанюк Ю.Е.

Задание

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.

Текст программы

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

namespace Лаб3
{
    class Program
    {
        static void Main(string[] args)
        {
```

```
            Rectangle r = new Rectangle(4, 6);
            Square s = new Square(2);
            Circle c = new Circle(5);
            Console.WriteLine("ArrayList: ");
            ArrayList al = new ArrayList();
            al.Add(r);
            al.Add(s);
            al.Add(c);
            foreach (var x in al) Console.WriteLine(x);
            Console.WriteLine("ArrayList после сортировки: ");
            al.Sort();
            foreach (var x in al) Console.WriteLine(x);

            Console.WriteLine("List <Figure>: ");
            List<Figure> f = new List<Figure>();
            f.Add(r);
            f.Add(s);
            f.Add(c);
            foreach (var x in f) Console.WriteLine(x);
            Console.WriteLine("List <Figure> после сортировки: ");
            f.Sort();
            foreach (var x in f) Console.WriteLine(x);

            Console.WriteLine("Матрица: ");

            Matrix<Figure> matr = new Matrix<Figure>(3, 3, new
FigureMatrixCheckEmpty());
            matr[0, 0] = r;
            matr[1, 1] = s;
            matr[2, 2] = c;
            Console.WriteLine(matr.ToString());
            Console.WriteLine("Список");
            SimpleList<Figure> list = new SimpleList<Figure>();
            list.Add(c);
            list.Add(r);
            list.Add(s);
            foreach (var x in list) Console.WriteLine(x);
            list.Sort();
            Console.WriteLine("Список после сортировки:");
            foreach (var x in list) Console.WriteLine(x);
```

```
        Console.WriteLine("\nСтек: ");
        SimpleStack<Figure> stack = new SimpleStack<Figure>();
        stack.Push(r);
        stack.Push(s);
        stack.Push(c);
        while (stack.Count > 0)
        {
            Figure ff = stack.Pop();
            Console.WriteLine(ff);
        }
        Console.ReadLine();


    }
}
interface IPrint
{
    void Print();
}
///       <summary>
///       Класс Геометрическая фигура
///       </summary>
abstract class Figure : IComparable
{
    ///   <summary>
    ///   Тип фигуры
    ///   </summary>
    public string Type
    {
        get
        {
            return this._Type;
        }
        protected set
        {
            this._Type = value;
        }
    }
    string _Type;
    ///   <summary>
```

```
    ///   Вычисление площади
    ///   </summary>
    ///   <returns></returns>
    public abstract double Area();
    ///   <summary>
    ///   Переопределение метода Object
    ///   </summary>
    ///   <returns></returns>
    public override string ToString()
    {
        return this.Type + " площадью " + this.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 (this.Area() < p.Area()) return -1;
        else if (this.Area() == p.Area()) return 0;
        else return 1;
    }

}
///       <summary>
///       Класс Прямоугольник
///       </summary>
class Rectangle : Figure, IPrint
{
    double height;
    double width;
    ///   <summary>
    ///   Основной конструктор
    ///   </summary>
    public Rectangle(double ph, double pw)
    {
        this.height = ph;
        this.width = pw;
```

```
        this.Type = "Прямоугольник";
    }
    ///   <summary>
    ///   Вычисление площади
    ///   </summary>
    public override double Area()
    {
        double Res = this.width * this.height;
        return Res;
    }
    public void Print()
    {
        Console.WriteLine(this.ToString());
    }
}
///       <summary>
///       Класс Квадрат
///       </summary>
class Square : Rectangle, IPrint
{
    public Square(double size) : base(size, size)
    {
        this.Type = "Квадрат";
    }
}
///       <summary>
///       Класс Круг
///       </summary>
class Circle : Figure, IPrint
{

    double radius;
    ///   <summary>
    ///   Основной конструктор
    ///   </summary>
    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());
        }
    }
    ///       <summary>
    ///       Класс Разреженная матрица
    ///       </summary>

    ///       <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)
        {
            this.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);
    this.Count++;
    //Добавление первого элемента
    if (last == null)
    {
        this.first = newItem;
        this.last = newItem;

    }
    //Добавление следующих элементов
    else
    {
        //Присоединение элемента к цепочке
        this.last.next = newItem;
        //Просоединенный элемент считается последним
        this.last = newItem;
    }
}
```

```
///   <summary>
///   Чтение контейнера с заданным номером
///   </summary>
public SimpleListItem<T> GetItem(int number)
{
    if ((number < 0) || (number >= this.Count))
    {
        //Можно создать собственный класс исключения
        throw new Exception("Выход за границу индекса");
    }
    SimpleListItem<T> current = this.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 = this.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, this.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;
    }
 }
 ///<summary>
 ///Класс стек
 ///</summary>
class SimpleStack<T>:SimpleList<T> where T : IComparable
 {
    ///<summary>
    ///   Добавление в стек
    ///   </summary>
    public void Push(T element)
    { //Добавление в конец списка уже реализовано
        Add(element);
    }
    ///   <summary>
    ///   Удаление и чтение из стека
    ///   </summary>
    public T Pop()
    {
        //default(T) - значение для типа T по
        умолчанию T Result = default(T);
                //Если стек пуст, возвращается значение по умолчанию для
типа
```

```
        if (this.Count == 0) return Result;
        //Если элемент единственный
        if (this.Count == 1)
        {
            //то из него читаются данные
            Result = this.first.data;
            //обнуляются указатели начала и конца списка
            this.first = null;
            this.last = null;
        }
        //В списке более одного элемента
        else
        { //Поиск предпоследнего элемента
            SimpleListItem<T> newLast = this.GetItem(this.Count - 2);
            //Чтение значения из последнего элемента
            Result = newLast.next.data;
            //предпоследний элемент считается последним
            this.last = newLast;
            //последний элемент удаляется из списка
            newLast.next = null;
        }
        //Уменьшение количества элементов в списке
        this.Count--;
        //Возврат результата
        return Result;
    }
}
public interface IMatrixCheckEmpty<T>
{ /// <summary>
   ///     Возвращает пустой элемент
   ///     </summary>
   T  getEmptyElement();
  ///     <summary>
  /// Проверка что элемент является пустым
  ///     </summary>
  bool checkEmptyElement(T element); }
class FigureMatrixCheckEmpty : IMatrixCheckEmpty<Figure>
{ /// <summary>
  ///     В качестве пустого элемента возвращается null
  ///     </summary>
```

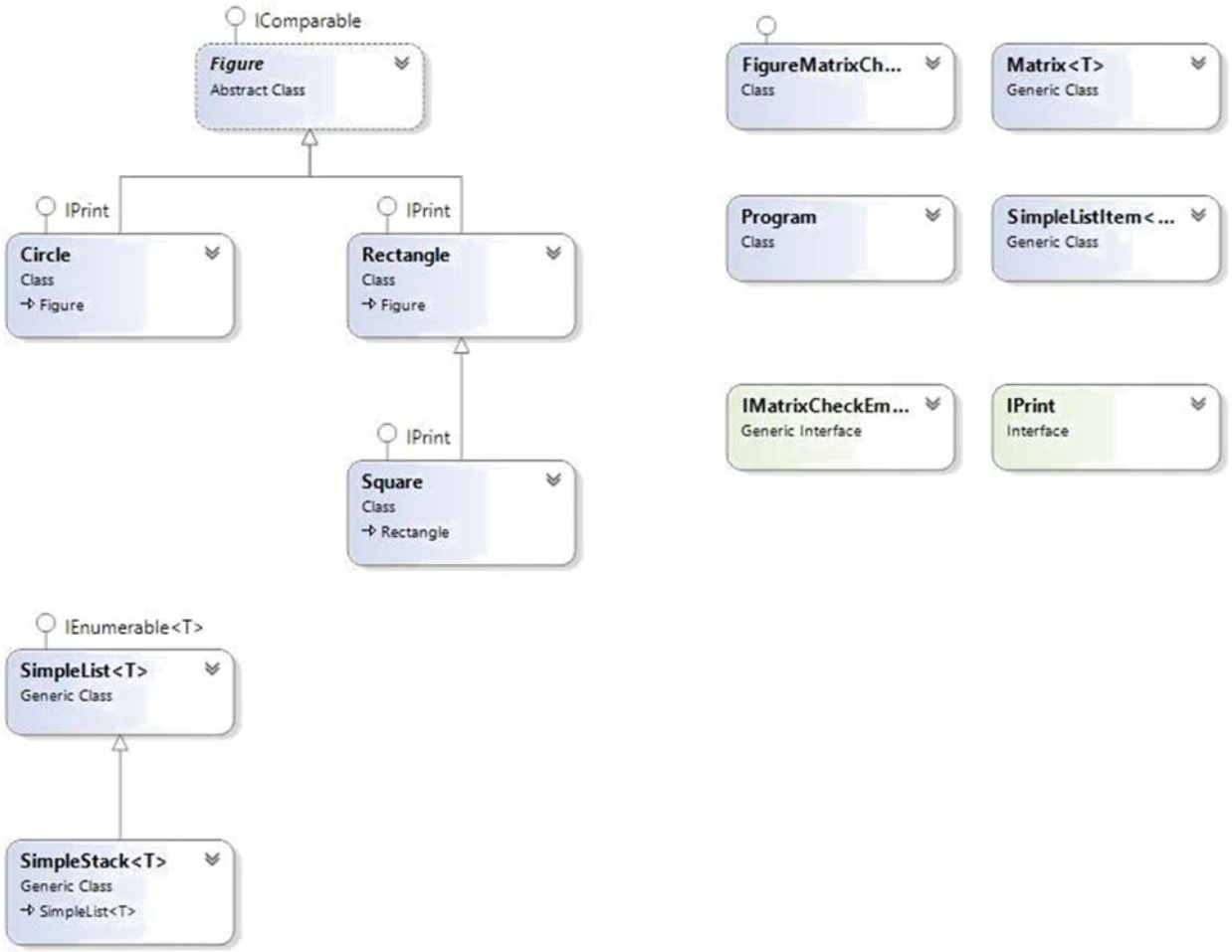
```
        public Figure getEmptyElement()
          { return null; }
        ///     <summary>
        ///     Проверка что переданный параметр равен null
        ///     </summary>
        public bool checkEmptyElement(Figure element)
          { bool Result =
              false; if
              (element ==
              null) { Result =
              true; } return
              Result;
          }
    }


    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>
        ///   Реализация интерфейса для проверки пустого элемента
        ///   </summary>

        ///   <summary>
        ///   Конструктор
        ///   </summary>
```

```
        public Matrix(int px, int py,
IMatrixCheckEmpty<T> checkEmptyParam)
        {
            this.maxX = px;
            this.maxY = py;
            this.checkEmpty = checkEmptyParam;
        }
        ///   <summary>
        ///   Индексатор для доступа к данных
        ///   </summary>
        public T this[int x, int y]
        {
            set {
                CheckBounds(x, y);
                string key = DictKey(x, y);
                this._matrix.Add(key, value);
            }
            get
            {
                CheckBounds(x, y);
                string key = DictKey(x, y);
                if (this._matrix.ContainsKey(key))
                { return this._matrix[key];
                }
                else {
                return this.checkEmpty.getEmptyElement();
                        }
            }
        }
        ///   <summary>
        ///   Проверка границ
        ///   </summary>
        void CheckBounds(int x, int y)
{

        if (x< 0 || x >= this.maxX)
         {
            throw new ArgumentOutOfRangeException("x", "x=" + x + "
выходит за границы");
          }
```

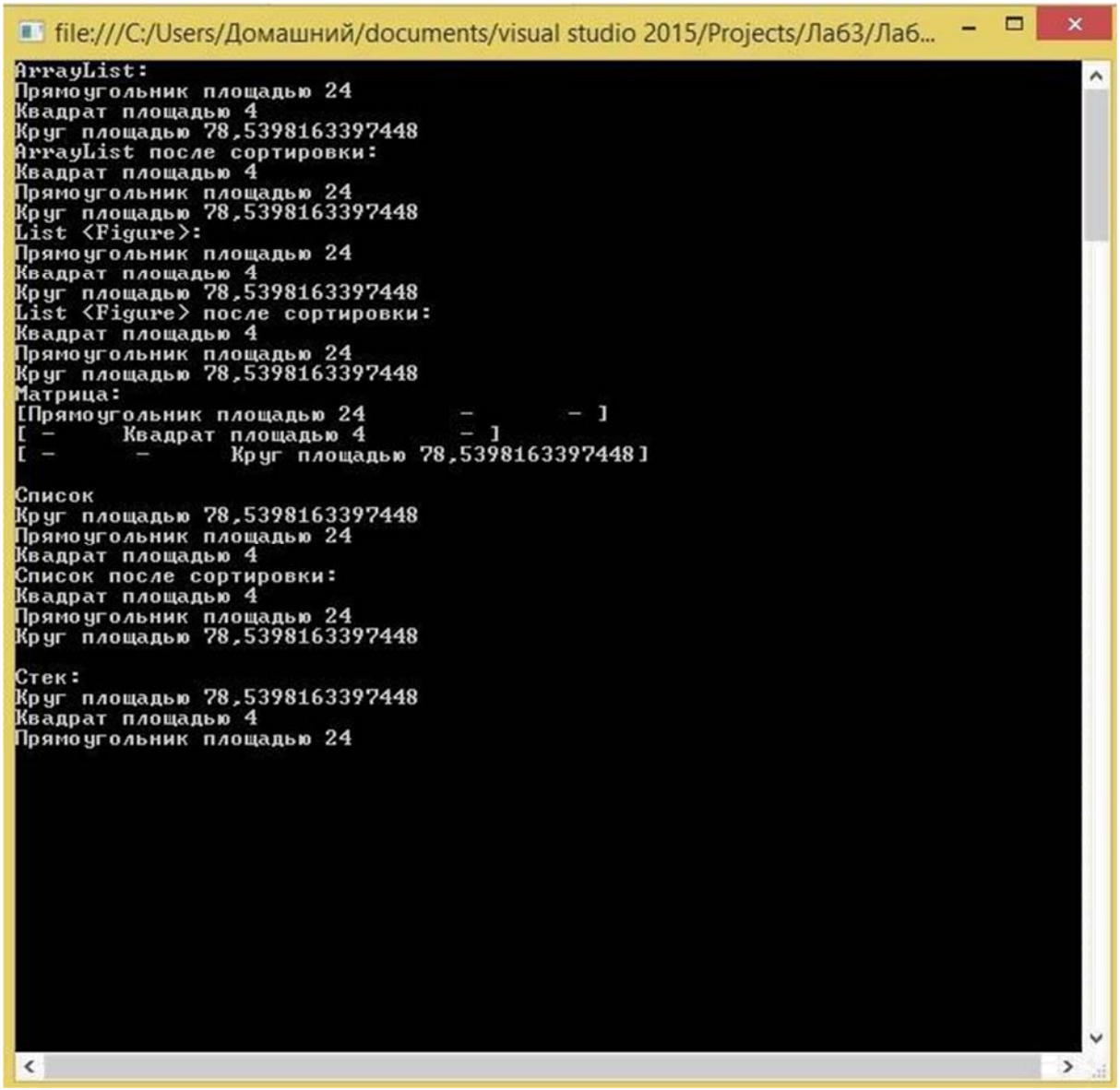
```
 if (y< 0 || y >= this.maxY)
{
 throw new ArgumentOutOfRangeException("y", "y=" + y + " выходит
за границы");
}
        }
/// <summary>
/// Формирование ключа
/// </summary>
string DictKey(int x, int y)
{
 return x.ToString() + "_" +
y.ToString(); }
/// <summary>
/// Приведение к строке
/// </summary>
/// <returns></returns>
public override string ToString()
{
    StringBuilder b = new StringBuilder(); for
     (int j = 0; j < this.maxY; j++) {

        b.Append("[");
        for (int i = 0; i < this.maxX; i++)
        {
            //Добавление разделителя-табуляции
            if (i > 0)
            {
             b.Append("\t");
              }
            //Если текущий элемент не пустой
            if (!this.checkEmpty.checkEmptyElement(this[i, j]))
            {
                //Добавить приведенный к строке текущий элемент
                b.Append(this[i, j].ToString());
             }
               else {
                //Иначе добавить признак пустого значения
                        b.Append(" - ");
                          }
```

```
                }
                b.Append("]\n");

            }
            return b.ToString();
        }
    }

    }
```

Диаграмма классов



Результат


file:///C:/Users/Домашний/documents/visual studio 2015/Projects/Лаб3/Лаб...
ArrayList:
Прямоугольник площадью 24
Квадрат площадью 4
Круг площадью 78,5398163397448
ArrayList после сортировки:
Квадрат площадью 4
Прямоугольник площадью 24
Круг площадью 78,5398163397448
List <Figure>:
Прямоугольник площадью 24
Квадрат площадью 4
Круг площадью 78,5398163397448
List <Figure> после сортировки:
Квадрат площадью 4
Прямоугольник площадью 24
Круг площадью 78,5398163397448
Матрица:
[Прямоугольник площадью 24 - -]
[- Квадрат площадью 4 -]
[- - Круг площадью 78,5398163397448]
Список
Круг площадью 78,5398163397448
Прямоугольник площадью 24
Квадрат площадью 4
Список после сортировки:
Квадрат площадью 4
Прямоугольник площадью 24
Круг площадью 78,5398163397448
Стек:
Круг площадью 78,5398163397448
Квадрат площадью 4
Прямоугольник площадью 24