#### Московский государственный технический университет имени Н.Э. Баумана

## Лабораторная работа по дисциплине БКИТ №3

*Выполнил: Ханмурзин Тагир, ИУ5-34, 24.10.2017*

**1. 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<Figure>. Сохранить объекты в коллекцию. Отсортировать коллекцию. Вывести в цикле содержимое коллекции.
6. Модифицировать класс разреженной матрицы (проект SparseMatrix) 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 (разобранного в пособии). 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.

**2. Листинг программного кода**

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Laba4

{

    interface IPrint

    {

        void Print();

    }

    public interface IMatrixCheckEmpty<T>

    {

        T getEmptyElement(); // Кидаем пустой элемент

        bool checkEmpty(T element); // Проверка на пустой элемент

    }

    abstract class Figure : IComparable

    {

        public string Type { get; protected set; }

        public abstract double Area();

        public override string ToString()

        {

            return "Площадь = " + this.Area().ToString();

        }

        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;

        }

    }

    class Rectangle : Figure, IPrint

    {

        public Rectangle(double w, double h)

        {

            this.width = w;

            this.length = h;

        }

        private double \_length = 0;

        public double length

        {

            get

            {

                return \_length;

            }

            set

            {

                \_length = value;

            }

        }

        private double \_width = 0;

        public double width

        {

            get

            {

                return \_width;

            }

            set

            {

                \_width = value;

            }

        }

        public override string ToString()

        {

            return "Прямоугольник [" + this.width + ", " + this.length + "] S = " + this.Area();

        }

        public override double Area()

        {

            double Result = this.length \* this.width;

            return Result;

        }

        public void Print()

        {

            Console.WriteLine(ToString());

        }

    }

    class Square : Rectangle, IPrint

    {

        public double size { get; set; }

        public Square(double s) : base(s, s)

        {

            this.size = s;

        }

        public override string ToString()

        {

            return "Квадрат [" + this.width + ", " + this.length + "] S = " + this.Area();

        }

    }

    class Circle : Figure, IPrint

    {

        private double \_radius = 0;

        public double radius

        {

            get

            {

                return \_radius;

            }

            set

            {

                \_radius = value;

            }

        }

        public Circle(double r)

        {

            this.radius = r;

        }

        public override double Area()

        {

            double Result = Math.PI \* this.radius \* this.radius;

            return Result;

        }

        public void Print()

        {

            Console.WriteLine(ToString());

        }

        public override string ToString()

        {

            return "Круг [" + this.radius + "] S = " + this.Area();

        }

    }

    //Класс для разреженной проверки пустоты

    class FigureMatrixCheckEmpty : IMatrixCheckEmpty<Figure>

    {

        // В качестве пустого элемента возвращается null

        public Figure getEmptyElement()

        {

            return null;

        }

        // Проверка на равенство null

        public bool checkEmpty(Figure element)

        {

            bool Result = false;

            if (element == null)

            {

                Result = true;

            }

            return Result;

        }

    }

    //Класс матрицы (методичка)

    public class Matrix<T>

    {

        // Словарь для хранения значений

        Dictionary<string, T> \_matrix = new Dictionary<string, T>();

        // Mаксимальное количество столбцов

        int maxX;

        // Mаксимальное количество строк

        int maxY;

        // Реализация интерфейса для проверки пустого элемента

        IMatrixCheckEmpty<T> сheckEmpty;

        // Конструктор

        public Matrix(int px, int py, IMatrixCheckEmpty<T> сheckEmptyParam)

        {

            this.maxX = px;

            this.maxY = py;

            this.сheckEmpty = сheckEmptyParam;

        }

        // Индексатор для доступа к данных

        public T this[int x, int y]

        {

            set

            {

                //Проверка выхода за границы матрицы

                CheckBounds(x, y);

                //Передача ключу значения координат элемента в матрице

                string key = x.ToString() + "\_" + y.ToString();

                //Передача элемента в матрицу

                this.\_matrix.Add(key, value);

            }

            get

            {

                CheckBounds(x, y);

                string key = x.ToString() + "\_" + y.ToString();

                if (this.\_matrix.ContainsKey(key))

                {

                    return this.\_matrix[key];

                }

                else

                {

                    return this.сheckEmpty.getEmptyElement();

                }

            }

        }

        // Проверка границ

        void CheckBounds(int x, int y)

        {

            if (x < 0 || x >= this.maxX)

            {

                //Выброс новых исключений

                throw new ArgumentOutOfRangeException

                    ("x", "x=" + x + " выходит за границы");

            }

            if (y < 0 || y >= this.maxY)

            {

                throw new ArgumentOutOfRangeException

                    ("y", "y=" + y + " выходит за границы");

            }

        }

        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.сheckEmpty.checkEmpty(this[i, j]))

                    {

                        //Добавить приведенный к строке текущий элемент

                        b.Append(this[i, j].ToString());

                    }

                    else

                    {

                        //Иначе добавить признак пустого значения

                        b.Append(" -------NULL------- ");

                    }

                }

                b.Append("]\n");

            }

            return b.ToString();

        }

    }

    //Простой односвязный список (методичка)

    //Элемент списка

    public class SimpleListItem<T>

    {

        public T data { set; get; }

        public SimpleListItem<T> next { set; get; }

        public SimpleListItem(T param)

        {

            this.data = param;

        }

    }

    //Список

    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;

            }

        }

        // Чтение контейнера с заданным номером

        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;

            //Пропускаем нужное количество элементов

            while (i < number)

            {

                //Переход к следующему элементу

                current = current.next;

                //Увеличение счетчика

                i++;

            }

            return current;

        }

        // Чтение элемента с заданным номером

        public T Get(int number)

        {

            return GetItem(number).data;

        }

        // Для перебора коллекции

        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 обобщенного IEnumerator<T> требует реализации необобщенного интерфейса

        //Данный метод добавляется автоматически при реализации интерфейса

        System.Collections.IEnumerator

            System.Collections.IEnumerable.GetEnumerator()

        {

            return GetEnumerator();

        }

        // Cортировка

        public void Sort()

        {

            Sort(0, this.Count - 1);

        }

        // Алгоритм быстрой сортировки

        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);

        }

        // Вспомогательный метод для обмена элементов при сортировке

        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;

        }

    }

    //Класс стек

    class SimpleStack<T> : SimpleList<T> where T : IComparable

    {

        public void Push(T element) //Добавление в стек

        {

            Add(element);

        }

        public T Pop() //Удаление и чтение из стека

        {

            T Result = default(T);

            if (this.Count == 0) return Result;

            if (this.Count == 1)

            {

                Result = this.first.data;

                this.first = null;

                this.last = null;

            }

            else

            {

                SimpleListItem<T> newLast = this.GetItem(this.Count - 2);

                Result = newLast.next.data;

                this.last = newLast;

                newLast.next = null;

            }

            this.Count--;

            return Result;

        }

    }

    class Program

    {

        static void Main(string[] args)

        {

            Console.ForegroundColor = ConsoleColor.Green;

            Console.WriteLine("# ArrayList ~~~~~~~~~~~~~~~~~~");

            Console.WriteLine("# Создаём объекты..."); Console.ResetColor();

            Rectangle rec1 = new Rectangle(3, 4);

            Square sq1 = new Square(5);

            Circle c1 = new Circle(2);

            Rectangle rec2 = new Rectangle(5, 4);

            Console.ForegroundColor = ConsoleColor.Green; Console.WriteLine("# Кладём объекты в ArrayList..."); Console.ResetColor();

            ArrayList fig1 = new ArrayList();

            fig1.Add(rec1);

            fig1.Add(sq1);

            fig1.Add(c1);

            fig1.Add(rec2);

            foreach (object o in fig1)

            {

                Console.WriteLine(o.ToString());

            }

            Console.ForegroundColor = ConsoleColor.Green; Console.WriteLine("\n# Сортировка ArrayList..."); Console.ResetColor();

            fig1.Sort();

            foreach (object o in fig1)

            {

                Console.WriteLine(o.ToString());

            }

            Console.ForegroundColor = ConsoleColor.Green;

            Console.WriteLine("\n# List ~~~~~~~~~~~~~~~~~~~~~~~");

            Console.WriteLine("# Создаём объекты..."); Console.ResetColor();

            Rectangle rect = new Rectangle(5, 8);

            Square square = new Square(4);

            Circle circle = new Circle(6);

            Rectangle rec3 = new Rectangle(7, 4);

            Console.ForegroundColor = ConsoleColor.Green; Console.WriteLine("# Кладём объекты в List..."); Console.ResetColor();

            List<Figure> fig2 = new List<Figure>()

            {

                circle, square, rect, rec3

            };

            foreach (object o in fig1)

            {

                Console.WriteLine(o.ToString());

            }

            Console.ForegroundColor = ConsoleColor.Green; Console.WriteLine("\n# Сортировка List..."); Console.ResetColor();

            fig2.Sort();

            foreach (object o in fig2)

            {

                Console.WriteLine(o.ToString());

            }

            Console.ForegroundColor = ConsoleColor.Green;

            Console.WriteLine("\n# Матрица ~~~~~~~~~~~~~~~~~~~~");

            Console.WriteLine("# Создаём матрицу...");

            Console.WriteLine("# Кладём объекты в матрицу..."); Console.ResetColor();

            Matrix<Figure> matrix = new Matrix<Figure>(3, 4, new FigureMatrixCheckEmpty());

            matrix[0, 0] = rect;

            matrix[1, 1] = square;

            matrix[2, 2] = circle;

            matrix[0, 3] = rec3;

            Console.WriteLine(matrix.ToString());

            Console.ForegroundColor = ConsoleColor.Green;

            Console.WriteLine("\n# Проверка стека ~~~~~~~~~~~~~");

            Console.WriteLine("# Создаём объекты...");

            Console.WriteLine("# Кладём объекты в List..."); Console.ResetColor();

            SimpleList<Figure> list = new SimpleList<Figure>();

            list.Add(circle);

            list.Add(rect);

            list.Add(square);

            foreach (var x in list)

            {

                Console.WriteLine(x);

            }

            Console.ForegroundColor = ConsoleColor.Green; Console.WriteLine("\n# Сортировка List..."); Console.ResetColor();

            list.Sort();

            foreach (var x in list)

            {

                Console.WriteLine(x);

            }

            Console.ForegroundColor = ConsoleColor.Green; Console.WriteLine("\n# Стэк с фигурами");

            Console.WriteLine("# Кладём объекты в стек (Push)..."); Console.ResetColor();

            SimpleStack<Figure> stack = new SimpleStack<Figure>();

            stack.Push(rect);

            stack.Push(square);

            stack.Push(circle);

            Console.ForegroundColor = ConsoleColor.Green; Console.WriteLine("# Вывод объектов из стека (Pop)..."); Console.ResetColor();

            while (stack.Count > 0)

            {

                Figure f = stack.Pop();

                Console.WriteLine(f);

            }

            Console.Read();

        }

    }

}

**3. Тест работы**

# ArrayList ~~~~~~~~~~~~~~~~~~

# Создаём объекты...

# Кладём объекты в ArrayList...

Прямоугольник [3, 4] S = 12

Квадрат [5, 5] S = 25

Круг [2] S = 12,5663706143592

Прямоугольник [5, 4] S = 20

# Сортировка ArrayList...

Прямоугольник [3, 4] S = 12

Круг [2] S = 12,5663706143592

Прямоугольник [5, 4] S = 20

Квадрат [5, 5] S = 25

# List ~~~~~~~~~~~~~~~~~~~~~~~

# Создаём объекты...

# Кладём объекты в List...

Прямоугольник [3, 4] S = 12

Круг [2] S = 12,5663706143592

Прямоугольник [5, 4] S = 20

Квадрат [5, 5] S = 25

# Сортировка List...

Квадрат [4, 4] S = 16

Прямоугольник [7, 4] S = 28

Прямоугольник [5, 8] S = 40

Круг [6] S = 113,097335529233

# Матрица ~~~~~~~~~~~~~~~~~~~~

# Создаём матрицу...

# Кладём объекты в матрицу...

[Прямоугольник [5, 8] S = 40 -------NULL------- -------NULL------- ]

[ -------NULL------- Квадрат [4, 4] S = 16 -------NULL------- ]

[ -------NULL------- -------NULL------- Круг [6] S = 113,097335529233]

[Прямоугольник [7, 4] S = 28 -------NULL------- -------NULL------- ]

# Проверка стека ~~~~~~~~~~~~~

# Создаём объекты...

# Кладём объекты в List...

Круг [6] S = 113,097335529233

Прямоугольник [5, 8] S = 40

Квадрат [4, 4] S = 16

# Сортировка List...

Квадрат [4, 4] S = 16

Прямоугольник [5, 8] S = 40

Круг [6] S = 113,097335529233

# Стэк с фигурами

# Кладём объекты в стек (Push)...

# Вывод объектов из стека (Pop)...

Круг [6] S = 113,097335529233

Квадрат [4, 4] S = 16

Прямоугольник [5, 8] S = 40