Московский государственный технический университет им. Н.Э. Баумана

Факультет «Радиотехнический»
Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Базовые компоненты интернет-технологий»

Отчет по лабораторной работе №3

Выполнил:
студент группы РТ5-31Б
Борисочкин М.И.
Подпись и дата:

Проверил:
к.т.н., доцент
Гапанюк Ю. Е.
Подпись и дата:

Москва, 2020 г.

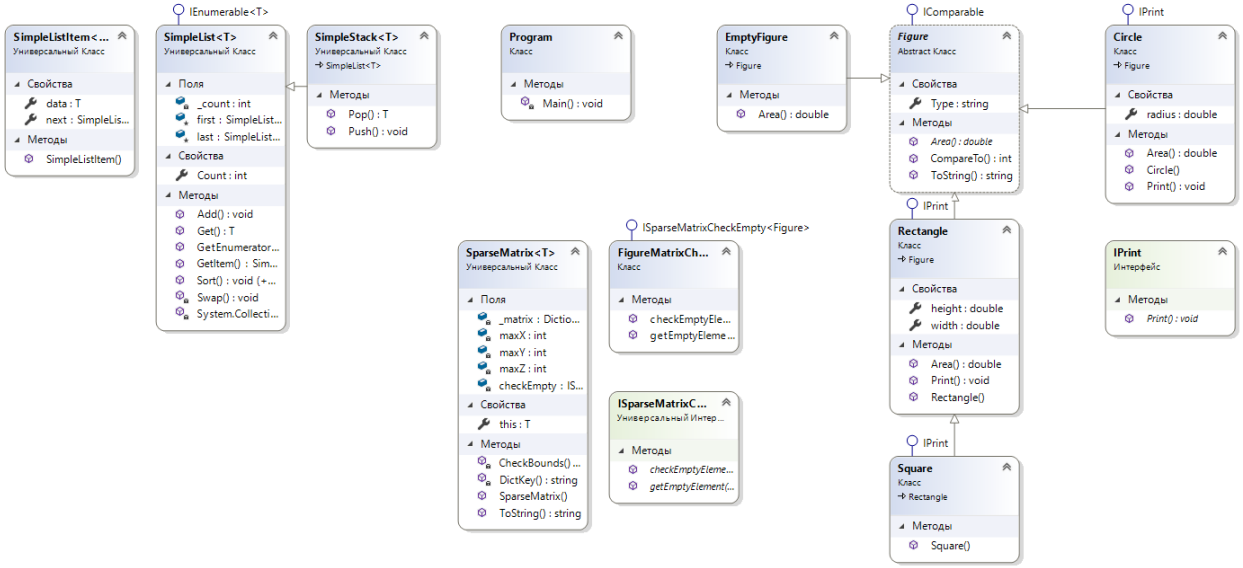
Описание задания

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 (проект SimpleListProject). Необходимо добавить в класс методы:
    - public void Push(T element) – добавление в стек;
    - public T Pop() – чтение с удалением из стека.

Пример работы класса SimpleStack реализовать на основе геометрических фигур.

Диаграмма классов



Текст программы

Figure.cs

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

namespace Borisochkin_Lab3
{
    /// <summary>
    /// Класс фигура
    /// </summary>
    abstract class Figure: IComparable
    {
        ///<summary>
        ///Тип фигуры
        ///</summary>
        public string Type { get; protected set;}

        ///<summary>
        ///Вычисление площади
        ///</summary>
        public abstract double Area();

        ///<summary>
        ///Приведение к строке
        ///</summary>
        public override string ToString()
        {
            return this.Type + " площадью " + this.Area().ToString();
        }

        /// <summary>
        /// Сравнение элементов (для сортировки)
        /// </summary>
        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;
        }
    }
}
```

EmptyFigure.cs

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

namespace Borisochkin_Lab3
{
    class EmptyFigure : Figure
    {
        public override double Area()
        {
            return 0;
        }
    }
}
```

IPrint.cs

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

namespace Borisochkin_Lab_2
{
    interface IPrint
    {
        void Print();
    }
}
```

Rectangle.cs

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

namespace Borisochkin_Lab_2
{
    /// <summary>
    /// Класс прямоугольник
    /// </summary>
    class Rectangle : Figure, IPrint
    {
        /// <summary>
        /// Высота
        /// </summary>
       public double height { get; set;}

        /// <summary>
        /// Ширина
        /// </summary>
       public double width { get; set;}

        /// <summary>
        /// Основной конструктор
        /// </summary>
        /// <param name="rectheight">Высота</param>
        ///  <param name="rectwidth">Ширина</param>
        public Rectangle(double rectheight, double rectwidth)
        {
            this.height = rectheight;
            this.width = rectwidth;
            this.Type = "Прямоугольник";
        }

        /// <summary>
        /// Вычисление площади
        /// </summary>
        public override double Area()
        {
            double Result = this.width * this.height;
            return Result;
        }

        public void Print()
        {
            Console.WriteLine(this.ToString());
        }
    }
}
```

Square.cs

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

namespace Borisochkin_Lab_2
{
    /// <summary>
    /// Класс квадрат
    /// </summary>
    class Square : Rectangle, IPrint
    {
        /// <summary>
        /// Основной конструктор
        /// </summary>
        /// <param name="size">Длина стороны квадрата</param>
        public Square(double size): base(size,size)
        {
            this.Type = "Квадрат";
        }
    }
}
```

Circle.cs

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

namespace Borisochkin_Lab_2
{
    /// <summary>
    /// Класс круг
    /// </summary>
    class Circle : Figure, IPrint
    {
        /// <summary>
        /// Радиус
        /// </summary>
        public double radius { get; set; }

        /// <summary>
        /// Основной конструктор
        /// </summary>
        /// <param name="r">Радиус</param>
        public Circle(double r)
        {
            this.radius = r;
            this.Type = "Круг";
        }

        /// <summary>
        /// Вычисление площади
        /// </summary>
        public override double Area()
        {
            double Result = Math.PI * this.radius * this.radius;
            return Result;
        }

        public void Print()
        {
            Console.WriteLine(this.ToString());
        }
    }
}
```

SparseMatrix.cs

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

namespace Borisochkin_Lab3
{
    class SparseMatrix<T>
    {
        /// <summary>
        /// Словарь для хранения значений
        /// </summary>
        Dictionary<string, T> _matrix = new Dictionary<string, T>();

        /// <summary>
        /// Количество элементов по X
        /// </summary>
        int maxX;

        /// <summary>
        /// Количество элементов по Y
        /// </summary>
        int maxY;

        /// <summary>
        /// Количество элементов по Z
        /// </summary>
        int maxZ;

        /// <summary>
        /// Реализация интерфейса для проверки пустого элемента
        /// </summary>
        ISparseMatrixCheckEmpty<T> checkEmpty;

        /// <summary>
        /// Конструктор
        /// </summary>
        public SparseMatrix(int x, int y, int z, ISparseMatrixCheckEmpty<T>
checkEmptyParam)
        {
            this.maxX = x;
            this.maxY = y;
            this.maxZ = z;
            this.checkEmpty = checkEmptyParam;
        }

        /// <summary>
        /// Индексатор для доступа к данных
        /// </summary>
        public T this[int x, int y, int z]
        {
            set
            {
                CheckBounds(x, y, z);
                string key = DictKey(x, y, z);
                this._matrix.Add(key, value);
            }
            get
            {
                CheckBounds(x, y, z);
                string key = DictKey(x, y, z);
```

```
                if (this._matrix.ContainsKey(key))
                {
                    return this._matrix[key];
                }
                else
                {
                    return this.checkEmpty.getEmptyElement();
                }
            }
        }

        /// <summary>
        /// Проверка границ
        /// </summary>
        void CheckBounds(int x, int y, int z)
        {
            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");
            }

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

        /// <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()
        {
            StringBuilder b = new StringBuilder();

            for(int k =0; k< this.maxZ; k++)
            {
                b.Append("\nz = " + k + "\n");
                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, k]))
                        {
                            //Добавить приведенный к строке текущий элемент
                            b.Append(this[i, j, k].ToString());
                        }
                        else
                        {
                            //Иначе добавить признак пустого значения
                            b.Append(" - ");
                        }
                    }
                    b.Append("]\n");
                }
            }
            return b.ToString();
        }
    }
}
```

ISparseMatrixCheckEmpty.cs

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

namespace Borisochkin_Lab3
{
    public interface ISparseMatrixCheckEmpty<T>
    {
        /// <summary>
        /// Возвращает пустой элемент
        /// </summary>
        T getEmptyElement();

        /// <summary>
        /// Проверка что элемент является пустым
        /// </summary>
        bool checkEmptyElement(T element);
    }
}
```

FigureMatrixCheckEmpty.cs

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

namespace Borisochkin_Lab3
{
    class FigureMatrixCheckEmpty : ISparseMatrixCheckEmpty<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;
        }
    }
}
```

SimpleList.cs

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

namespace Borisochkin_Lab3
{
    /// <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>
        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 обобщенного 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>
```

```
        /// Алгоритм быстрой сортировки
        /// </summary>
        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;
        }
    }
}
```

SimpleListItem.cs

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

namespace Borisochkin_Lab3
{
    /// <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;
        }
    }
}
```

SimpleStack.cs

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

namespace Borisochkin_Lab3
{
    /// <summary>
    /// Класс стек
    /// </summary>
    class SimpleStack<T> : SimpleList<T> where T : IComparable
    {
        /// <summary>
        /// Добавление в стек
        /// </summary>
        public void Push(T element)
        {
            Add(element);
        }

        /// <summary>
        /// Удаление и чтение из стека
        /// </summary>
        public T Pop()
        {
            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;
        }
    }
}
```

Program.cs

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

namespace Borisochkin_Lab3
{
    class Program
    {
        static void Main(string[] args)
        {
            Rectangle rect = new Rectangle(3, 2);
            Square square = new Square(6);
            Circle circle = new Circle(2);

            // Коллекция класса ArrayList
            ArrayList f1 = new ArrayList();

            // Добавление элементов
            f1.Add(rect);
            f1.Add(square);
            f1.Add(circle);

            // Вывод коллекции до сортировки
            Console.WriteLine("ArrayList: \n");
            Console.WriteLine("До сортировки:");
            foreach (var i in f1)
                Console.WriteLine(i);

            // Сортировка и вывод коллекции после сортировки
            Console.WriteLine("\nПосле сортировки:");
            f1.Sort();
            foreach (var i in f1)
                Console.WriteLine(i);

            // Коллекция класса List<Figure>
            List<Figure> f2 = new List<Figure>();

            // Добавление элементов
            f2.Add(square);
            f2.Add(circle);
            f2.Add(rect);

            // Вывод коллекции до сортировки
            Console.WriteLine("\nList<Figure>:");
            Console.WriteLine("\nДо сортировки:");
            foreach (var i in f2)
                Console.WriteLine(i);

            // Сортировка и вывод коллекции после сортировки
            Console.WriteLine("\nПосле сортировки:");
            f2.Sort();
            foreach (var i in f2)
                Console.WriteLine(i);

            // Матрица
            Console.WriteLine("\nМатрица:");
            SparseMatrix<Figure> fmatrix = new SparseMatrix<Figure>(3, 3, 3, new
FigureMatrixCheckEmpty());
            fmatrix[0, 0, 0] = rect;
```

```
            fmatrix[1, 1, 1] = square;
            fmatrix[2, 2, 2] = circle;
            Console.WriteLine(fmatrix.ToString());

            //Стек
            Console.WriteLine("\nСтек:");
            SimpleStack<Figure> fstack = new SimpleStack<Figure>();

            // Добавление элементов в стек
            fstack.Push(rect);
            fstack.Push(square);
            fstack.Push(circle);

            // Чтение из стека и вывод элементов
            while(fstack.Count > 0)
            {
                Figure f = fstack.Pop();
                Console.WriteLine(f);
            }

            Console.ReadLine();
        }
    }
}
```

Примеры выполнения программы