Московский государственный технический университет им. Н.Э. Баумана

Факультет «Информатика и системы управления»
Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Базовые компоненты интернет-технологий»

Отчет по лабораторной работе №3
«Работа с коллекциями»

Выполнил:
студент группы ИУ5-33
Сергеев МЮ
Подпись и дата:
29.12.20

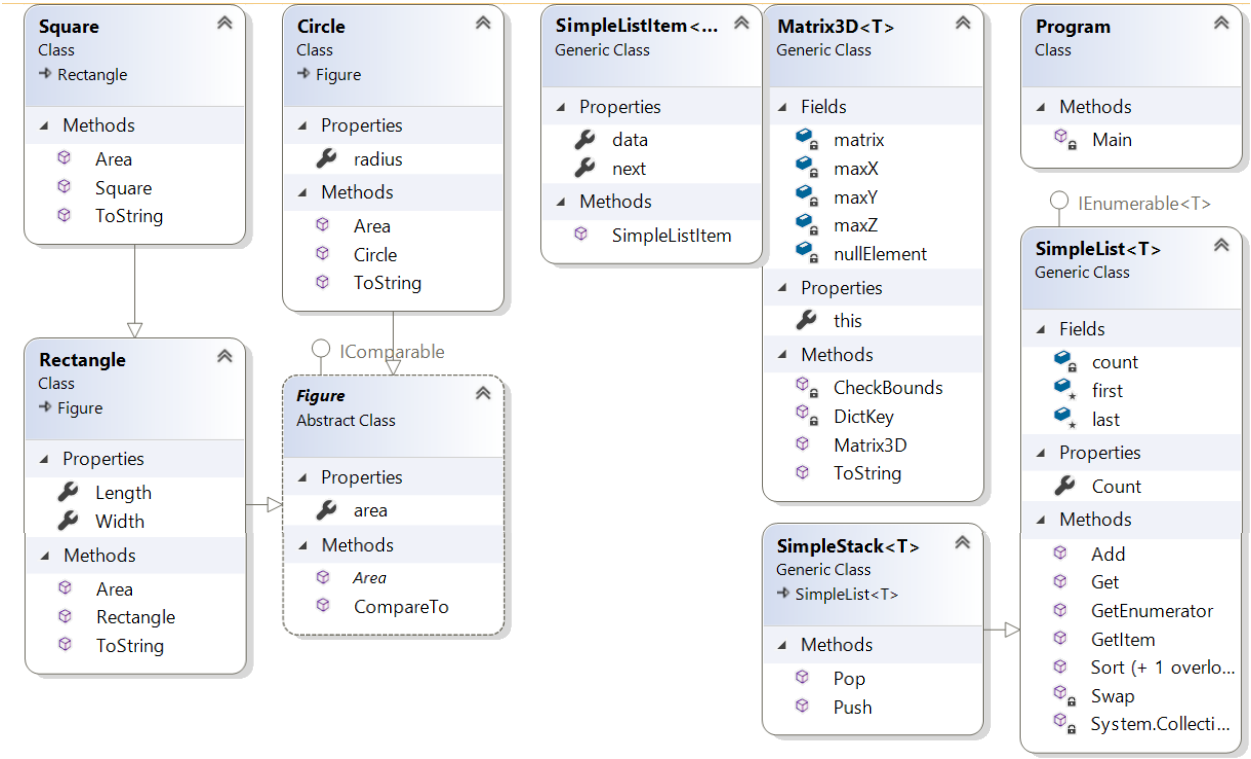
Проверил:
Подпись и дата:

Москва, 2020 г.

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

Диаграмма классов:



Текст программы

Program.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
namespace Lab3
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.WriteLine("Sergeev Maxim IU5-33B");
         Rectangle rect = new Rectangle(5, 4);
         Square square = new Square(5);
         Circle circle = new Circle(5);
         Console.WriteLine("\nArrayList");
         ArrayList a1 = new ArrayList();
         a1.Add(circle);
         a1.Add(rect);
         a1.Add(square);
```

```
            foreach (var x in a1) Console.WriteLine(x.ToString());
            Console.WriteLine("\nArrayList-sort");
            a1.Sort();
            foreach (var x in a1) Console.WriteLine(x.ToString());

            Console.WriteLine("\nList<Figure>");
            List<Figure> f1 = new List<Figure>();
            f1.Add(circle);
            f1.Add(rect);
            f1.Add(square);
            foreach (var x in f1) Console.WriteLine(x.ToString());
            Console.WriteLine("\nList<Figure>-sort");
            f1.Sort();
            foreach (var x in f1) Console.WriteLine(x.ToString());

            Console.WriteLine("\nMatrix");
            Matrix3D<Figure> cube = new Matrix3D<Figure>(3, 3, 3, null);
            cube[0, 0, 0] = rect;
            cube[1, 1, 1] = square;
            cube[2, 2, 2] = circle;
            Console.WriteLine(cube.ToString());

            Console.WriteLine("\nStack");
            SimpleStack<Figure> stack = new SimpleStack<Figure>();
            stack.Push(rect);
            stack.Push(square);
            stack.Push(circle);
            foreach (var x in stack) Console.WriteLine(x.ToString());
            Console.WriteLine("\nStack-pop");
            stack.Pop();
            foreach (var x in stack) Console.WriteLine(x.ToString());
        }
    }
}
```

Simple stack.cs

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

namespace Lab3
{
```

```
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("out of bounds index");
```

```
            }
            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(i).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;
      }
   }
   public class SimpleStack<T> : SimpleList<T>
      where T : IComparable
   {
      public void Push(T el)
      {
         Add(el);
      }
      public object Pop()
      {
         var tmp = first;
         while (tmp != null && tmp.next != last)
            tmp = tmp.next;
         last = tmp;
         T data = last.next.data;
         last.next = null;
         Count--;
         return data;
      }
   }
}
```

Matrix3D.cs

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

namespace Lab3
{
   public class Matrix3D<T>
   {
      Dictionary<string, T> matrix = new Dictionary<string, T>();
      int maxX, maxY, maxZ;
      T nullElement;
      public Matrix3D(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 + "going out
    of bounds");
            if (y < 0 || y >= this.maxY) throw new Exception("y=" + y + "going out
    of bounds");
            if (z < 0 || z >= this.maxZ) throw new Exception("z=" + z + "going out of
    bounds");
        }
        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 k = 0; k < this.maxZ; k++)
            {
                b.Append("[");
                for (int j = 0; j < this.maxY; j++)
                {
                    if (j > 0) b.Append(",  ");
                    b.Append("[");
                    for (int i = 0; i < this.maxX; i++)
                    {
                        if (i > 0) b.Append(";  ");
                        if (this[i, j, k] != null)
                            b.Append(this[i, j, k].ToString());
                        else
                            b.Append("null");
                    }
                    b.Append("]");
                }
                b.Append("]\n");
            }
            return b.ToString();
        }
    }
}
```

Figure.cs

```
using System;

namespace Lab3
{
   public abstract class Figure : IComparable
   {
      public float area { get; set; }
      public abstract void Area();
      public int CompareTo(object obj)
      {
         Figure f = obj as Figure;
         return this.area.CompareTo(f.area);
      }
   }
   public class Rectangle : Figure
   {
      public float Width { get; set; }
      public float Length { get; set; }
      public Rectangle(float a, float b)
      {
         Width = a;
         Length = b;
         area = Width * Length;
      }
      public override void Area()
      {
         area = Width * Length;
      }
      public override string ToString()
      {
         string str = "Rectangle Width=" + Width.ToString() + " Length=" + Length.ToString() +
" Area=" + area.ToString();
         return str;
      }
   }
   public class Square : Rectangle
   {
      public Square(float a) : base(a, a)
      {

      }
      public override void Area()
      {
         area = Width * Length;
      }
      public override string ToString()
```

```
        {
            string str = "Square Width=" + Width.ToString() + " Area=" + area.ToString();
            return str;
        }
    }
    public class Circle : Figure
    {
        public float radius { get; set; }
        public Circle(float a)
        {
            radius = a;
            area = (float)Math.PI * radius * radius;
        }
        public override void Area()
        {
            area = (float)Math.PI * radius * radius;
        }
        public override string ToString()
        {
            string str = "Circle Radius=" + radius.ToString() + " Area=" + area.ToString();
            return str;
        }
    }
}
```

Экранные формы с примерами выполнения программы: