Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)

Факультет «Информатика и вычислительная техника»
Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Базовые компоненты интернет-технологий»

Отчет по лабораторной работе №1
«Коллекции»

Выполнил:
студент группы ИУ5-22Б

Веревкина Диана В.

Подпись и дата:

Проверил:
преподаватель каф. ИУ5
Гапанюк Ю.Е.

Подпись и дата:

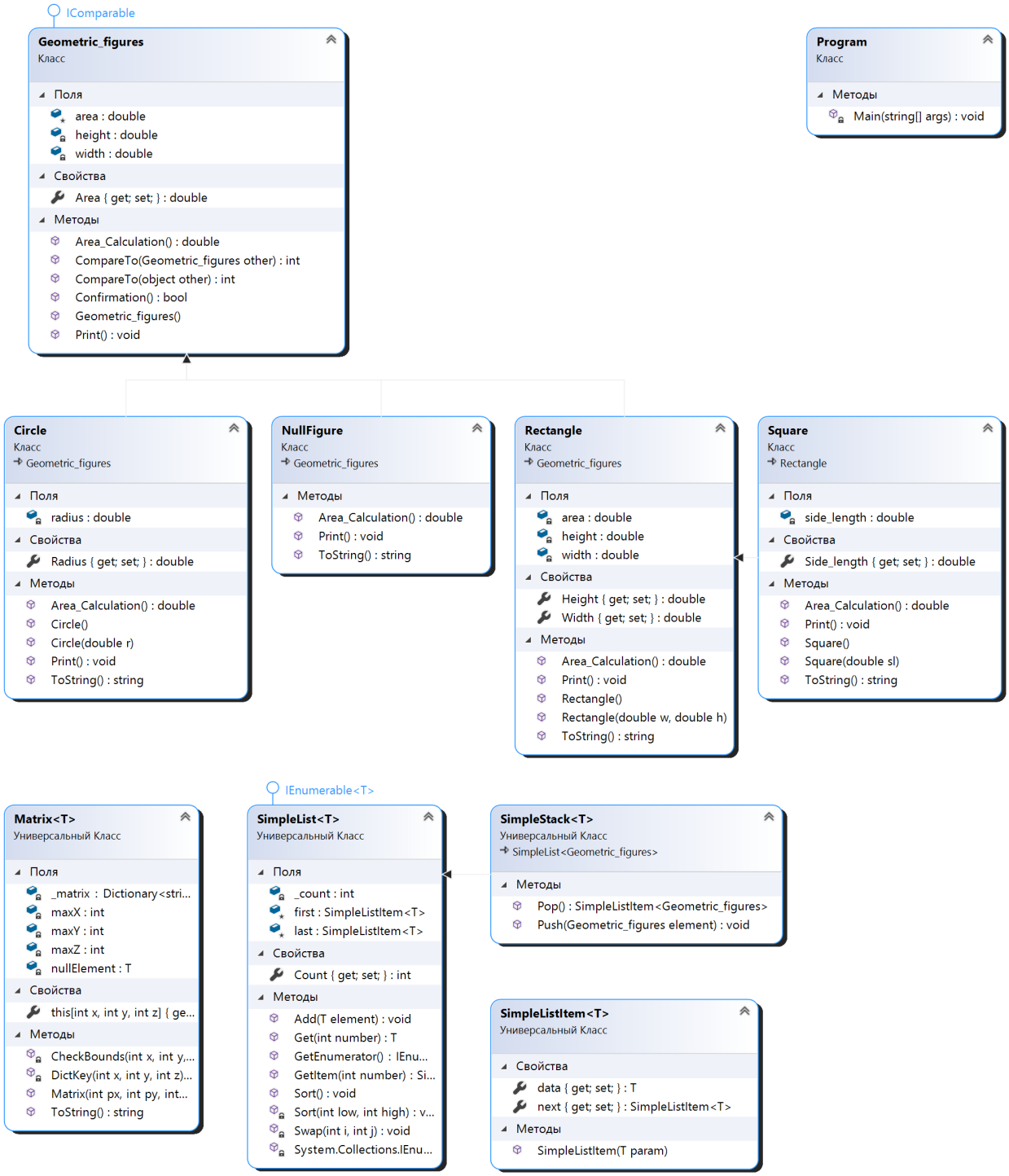
Москва, 2020 г.

Постановка задачи

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

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

```
namespace BCIT_Lab3
{
   #region Geometric_figures
   public class Geometric_figures : IComparable //IComparable - интерфейс для
сортировки
   {
      protected double area;
      private double width, height;

      public int CompareTo(Geometric_figures other)
      {
         if (other==null)
            throw new Exception("Ошибка! Невозможно сравнить объекты!");
         return area.CompareTo(other.area);
      }
      public int CompareTo(object other)
      {
         if (other == null)
            throw new Exception("Ошибка! Невозможно сравнить объекты!");
         return CompareTo(other as Geometric_figures);
      }
     public Geometric_figures ()
      {
         width = 1;
         height = 1;
      }
      public virtual double Area_Calculation() //Виртуальный метод вычисления
площади
      {
         //area = w * h;
         return 0;
      }
      public double Area //свойство
      {
         get
         {
            return area;
         }
         set
         {
            area = value;
         }
      }
```

```
        public bool Confirmation() //метод, определяющий желание пользователя продолжить работу
        {
            string Answer;
            do
            {
                Console.WriteLine();
                Console.WriteLine("Хотите продолжить? (Y/N)");

                Answer = Console.ReadLine();
                Console.WriteLine();
            }
            while ((Answer != "Y") & (Answer != "N") & (Answer != "y") & (Answer != "n"));
            return (Answer == "Y") || (Answer == "y");
        }
        public virtual void Print() {}
    }
    #endregion
    #region Rectangle
    public class Rectangle : Geometric_figures //Прямоугольник
    {
        private double width, height, area; //ширина, высота, площадь
        public double Width //свойство
        {
            get
            {
                return width;
            }
            set
            {
                width = value;
            }
        }
        public double Height //свойство
        {
            get
            {
                return height;
            }
            set
            {
                height = value;
            }
        }
```

```
        public Rectangle() //пустой конструктор
        {
            width = 1;
            height = 1;
        }
        public Rectangle(double w, double h) //конструктор
        {
            width = w;
            height = h;
        }
        public override double Area_Calculation() //вычисление площади
        {
            //double Rec_area;
            area = width * height;
            return area;
        }
        public override string ToString() //переопределение метода ToString
        {
            return ("Прямоугольник:    Ширина = " + Width + "    Высота = " +
Height + "    Площадь = " + Area_Calculation());
        }

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

    }
    #endregion
    #region Square
    public class Square : Rectangle //Квадрат
    {
        private double side_length; //длина стороны квадрата
        public double Side_length //свойство
        {
            get
            {
                return side_length;
            }
            set
            {
                side_length = value;
            }
        }
        public Square() //пустой конструктор
```

```
        {
            side_length = 1;
        }
        public Square(double sl) // конструктор
        {
            side_length = sl;
        }
        public override double Area_Calculation() //площадь
        {
          // double Sq_area;
            area = side_length * side_length;
            return area;
        }

        public override string ToString()
        {
            return "Квадрат:    Длина стороны = " + Side_length + "    Площадь = " +
Area_Calculation();
        }

        public  override void Print()
        {
            Console.WriteLine(ToString());
        }
    }
    #endregion
    #region Circle
    public class Circle : Geometric_figures //Круг
    {
        private double radius;
        public double Radius //свойство
        {
            get
            {
                return radius;
            }
            set
            {
                radius = value;
            }
        }
        public Circle() //пустой конструктор
        {
            radius = 1;
        }
```

```
        public Circle(double r) //конструктор
        {
            radius = r;
        }
        public override double Area_Calculation()
        {
            //double Ci_area;
            area = 3.14 * radius * radius;
            return area;
        }
        public override string ToString()
        {
            return "Круг:    Радиус = " + Radius + "    Площадь = " +
Area_Calculation();
        }
        public override void Print()
        {
            Console.WriteLine(ToString());
        }
    }
    #endregion
    #region SparseMatrix
    public class Matrix<T>
    {
        /// Словарь для хранения значений
        Dictionary<string, T> _matrix = new Dictionary<string, T>();

        /// Количество элементов по горизонтали (максимальное количество
столбцов)
        int maxX;

        /// Количество элементов по вертикали (максимальное количество строк)
        int maxY;

        /// Количество элементов по Z (максимальное количество строк)
        int maxZ;


        /// Пустой элемент, который возвращается, если элемент с нужными
координатами не был задан
        T nullElement;

        /// Конструктор

        public Matrix(int px, int py, int pz, T nullElementParam)
```

```
        {
            this.maxX = px;
            this.maxY = py;
            this.maxZ = pz;
            this.nullElement = nullElementParam;
        }

        /// Индексатор для доступа к данных
        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);
            }
        }

        /// Проверка границ
        void CheckBounds(int x, int y, int z)
        {
            if (x < 0 || x >= this.maxX) throw new Exception("x=" + x + " выходит за
границы");
            if (y < 0 || y >= this.maxY) throw new Exception("y=" + y + " выходит за
границы");
            if (z < 0 || z >= this.maxY) throw new Exception("z=" + z + " выходит за
границы");

        }

        /// Формирование ключа
        string DictKey(int x, int y, int z)
```

```
        {
            return x.ToString() + "_" + y.ToString() + "_" + z.ToString();
        }

        /// Приведение к строке
        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)
            {
                for (int j = 0; j != maxY; ++j)
                {
                    b.Append("[  ");
                    for (int i = 0; i != maxX; ++i)
                    {
                        if (i > 0) b.Append("  |  ");
                        b.Append(this[i, j, k].ToString());
                    }
                    b.Append("  ]\n");
                }
                b.Append("\n");
            }
            return b.ToString();
        }
    }
    #endregion
    #region SimpleStack
    public class SimpleListItem<T>
    {
        /// Данные
        public T data { get; set; }
        /// Следующий элемент
        public SimpleListItem<T> next { get; set; }
        ///конструктор
        public SimpleListItem(T param)
        {
            this.data = param;
        }
```

```
}

public class SimpleList<T> : IEnumerable<T>
  where T : IComparable
{
   /// Первый элемент списка
   protected SimpleListItem<T> first = null;

   /// Последний элемент списка
   protected SimpleListItem<T> last = null;

   /// Количество элементов
   public int Count
   {
      get { return _count; }
      protected set { _count = value; }
   }
   int _count;

   /// Добавление элемента
   /// <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;
      }
   }

   /// Чтение контейнера с заданным номером
   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;
        }

        /// 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 обощенного IEnumerator<T> требует реализации
необобщенного интерфейса
        //Данный метод добавляется автоматически при реализации интерфейса
```

```
        System.Collections.IEnumerator
System.Collections.IEnumerable.GetEnumerator()
        {
            return GetEnumerator();
        }

        /// Сортировка
        public void Sort()
        {
            Sort(0, this.Count - 1);
        }

        /// Алгоритм быстрой сортировки
        /// <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);
        }

        /// 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;
        }
    }
```

```
public class SimpleStack<T>: SimpleList<Geometric_figures>
{
   public void Push(Geometric_figures element)
   {
      Add(element);
   }
   public SimpleListItem<Geometric_figures> Pop()
   {
      SimpleListItem<Geometric_figures> item;
      if (this.Count==0)
         throw new Exception("В стеке больше нет элементов");
      if (this.Count==1)
      {
         item = this.first;
         this.first = null;
         this.last = null;
         this.Count = 0;
         return item;
      }
      //если элементов >1
      item = this.last;
      //this.last = null;
      this.last = this.GetItem(this.Count - 2);
      this.Count--;
      return item;
   }

}

#endregion
class NullFigure : Geometric_figures
{
   public override double Area_Calculation()
   {
      return 0;
   }
   public override string ToString()
   {
      return "None";
   }
   public override void Print()
   {
      Console.WriteLine(ToString());
   }
}
```

```
class Program
{
    static void Main(string[] args)
    {
        double w, h, l, r; //вводимые ширина, высота, длина стороны квадрата и радиус
        Console.WriteLine("\\Ввод данных\\");
        Console.WriteLine();
        #region Ввод данных для прямоугольника
        Rectangle rec;
        Console.WriteLine("__Прямоугольник");
        Console.WriteLine();
        //заполнение полей объекта
        Console.Write("Введите ширину прямоугольника: ");
        while (!double.TryParse(Console.ReadLine(), out w)) //проверка на корректность введенного значения
            Console.WriteLine("Ошибка! Введите число!");
        Console.Write("Введите высоту прямоугольника: ");
        while (!double.TryParse(Console.ReadLine(), out h))
            Console.WriteLine("Ошибка! Введите число!");
        rec = new Rectangle(w, h);
        rec.Area_Calculation();
        rec.Print();
        #endregion
        #region Ввод данных для квадрата
        Square squ;
        Console.WriteLine();

        Console.WriteLine("__Квадрат");
        Console.WriteLine();
        //заполнение полей объекта
        Console.Write("Введите длину стороны квадрата: ");
        while (!double.TryParse(Console.ReadLine(), out l))
            Console.WriteLine("Ошибка! Введите число!");
        squ = new Square(l);
        squ.Area_Calculation();
        squ.Print();
        #endregion
        #region Ввод данных для круга
        Circle cir;
        Console.WriteLine();
        Console.WriteLine("__Круг");
        Console.WriteLine();
        //заполнение полей объекта
        Console.Write("Введите радиус круга: ");
```

```
        while (!double.TryParse(Console.ReadLine(), out r))
            Console.WriteLine("Ошибка! Введите число!");
        cir = new Circle(r);
        cir.Area_Calculation();
        cir.Print();
        #endregion
        Console.WriteLine("-------------------------------------------------");
        #region ArrayList
        ArrayList arlist = new ArrayList() {rec,squ,cir};

        arlist.Sort();
        Console.WriteLine();
        Console.WriteLine("Сортировка ArrayList по возрастанию");
        Console.WriteLine();
        foreach (object other in arlist)
        {
            Console.WriteLine(other);
            Console.WriteLine();
        }
        #endregion
        Console.WriteLine("-------------------------------------------------");

        #region List<Figure>
        List<Geometric_figures> list = new List<Geometric_figures>() { rec, squ,
cir};
        list.Sort();
        Console.WriteLine();
        Console.WriteLine("Сортировка List по возрастанию");
        Console.WriteLine();
        foreach (object other in arlist)
        {
            Console.WriteLine(other);
            Console.WriteLine();
        }
        #endregion
        Console.WriteLine("-------------------------------------------------");

        #region SparseMatrix
        Console.WriteLine("SparseMatrix");
        Console.WriteLine();
        Matrix<Geometric_figures> MatrixFig = new
Matrix<Geometric_figures>(4, 3, 2, new NullFigure());
        MatrixFig[0, 0, 1] = rec;
        MatrixFig[2, 1, 0] = squ;
        MatrixFig[3, 0, 1] = cir;
```

```
            MatrixFig[1, 2, 0] = rec;

            Console.WriteLine("\n" + MatrixFig);
            #endregion
            Console.WriteLine("-----------------------------------------------------");

            #region SimpleStack
            Console.WriteLine("SimpleStack");
            Console.WriteLine();
            SimpleStack<Geometric_figures> StackFig = new
SimpleStack<Geometric_figures>();
            StackFig.Push(rec);
            StackFig.Push(cir);
            StackFig.Push(squ);
            while (StackFig.Count != 0)
                StackFig.Pop().data.Print();
              // SimpleListItem<Geometric_figures> item = StackFig.Pop();
              // (item.data as Geometric_figures).Print();

            #endregion
        }
    }
}
```

Примеры выполнения программы

Консоль отладки Microsoft Visual Studio

```
\Ввод данных\

__Прямоугольник

Введите ширину прямоугольника: 3
Введите высоту прямоугольника: 4
Прямоугольник:    Ширина = 3    Высота = 4    Площадь = 12

__Квадрат

Введите длину стороны квадрата: 6
Квадрат:    Длина стороны = 6    Площадь = 36

__Круг

Введите радиус круга: 4
Круг:    Радиус = 4    Площадь = 50,24
-----------------------------------------------------

Сортировка ArrayList по возрастанию

Прямоугольник:    Ширина = 3    Высота = 4    Площадь = 12

Квадрат:    Длина стороны = 6    Площадь = 36

Круг:    Радиус = 4    Площадь = 50,24

-----------------------------------------------------

Сортировка List по возрастанию

Прямоугольник:    Ширина = 3    Высота = 4    Площадь = 12

Квадрат:    Длина стороны = 6    Площадь = 36

Круг:    Радиус = 4    Площадь = 50,24

-----------------------------------------------------
```

```
-----------------------------------------------------
SparseMatrix


[  None  |  None  |  None  |  None  ]
[  None  |  None  |  Квадрат:    Длина стороны = 6    Площадь = 36  |  None  ]
[  None  |  Прямоугольник:    Ширина = 3    Высота = 4    Площадь = 12  |  None  |  None  ]

[  Прямоугольник:    Ширина = 3    Высота = 4    Площадь = 12  |  None  |  None  |  Круг:    Радиус = 4    Площадь = 50,24  ]
[  None  |  None  |  None  |  None  ]
[  None  |  None  |  None  |  None  ]


-----------------------------------------------------
SimpleStack

Квадрат:    Длина стороны = 6    Площадь = 36
Круг:    Радиус = 4    Площадь = 50,24
Прямоугольник:    Ширина = 3    Высота = 4    Площадь = 12

C:\Users\diva2\Desktop\C#_(2 курс)\BCIT_Lab3\BCIT_Lab3\bin\Debug\netcoreapp3.1\BCIT_Lab3.exe (процесс 6340) завершил работу с ко
Чтобы автоматически закрывать консоль при остановке отладки, включите параметр "Сервис" ->"Параметры" ->"Отладка" -> "Автоматиче
дки".
Нажмите любую клавишу, чтобы закрыть это окно…
```