Московский государственный технический
университет имени Н.Э. Баумана

Парадигмы и конструкции языков программирования
Отчёт по лабораторной работе №3

Работу выполнил
Студент группы ИУ5-35Б
Шиленок Д.А.

2023 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. 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.

Текст программы

Program.cs

```
using FigureCollections;
using System.Collections;

namespace lab_3
{
   internal class Program
   {
      static void Main(string[] args)
      {
         Rectangle rectangle1 = new Rectangle(10, 2);
         Rectangle rectangle2 = new Rectangle(5, 8);
         Square square = new Square(4);
         Circle circle = new Circle(1);

         Console.WriteLine("\n\t\t---ARRAYLIST---\n");
         ArrayList arrayList = new ArrayList();
         arrayList.Add(rectangle1);
         arrayList.Add(rectangle2);
         arrayList.Add(square);
         arrayList.Add(circle);
         arrayList.Sort();
         foreach (Shape shape in arrayList) { shape.Print(); }

         Console.WriteLine("\n\t\t---LIST---\n");
         List<Shape> list = new List<Shape>();
         foreach (Shape shape in arrayList) { list.Add(shape); }
         list.Sort();
         foreach(Shape shape in list) { shape.Print(); };

         Console.WriteLine("\n\t\t---MATRIX---\n");
         Matrix<Shape> matrix = new Matrix<Shape>(3, 3, 3, new ShapeMatrixCheckEmpty());
         matrix[0, 0, 0] = rectangle1;
         matrix[1, 1, 1] = rectangle2;
         matrix[2, 2, 2] = square;
         matrix[1, 1, 0] = circle;
         Console.WriteLine(matrix.ToString());

         Console.WriteLine("\t\t---STACK---\n");
         SimpleStack<Shape> stack = new SimpleStack<Shape>();
         stack.Push(rectangle1);
         stack.Push(rectangle2);
         stack.Push(square);
         stack.Push(circle);
         while(stack.Count > 0) {  Console.WriteLine(stack.Pop()); }
      }
   }
}
```

Shape.cs

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

namespace lab_3
{
   interface IPrint
   {
      void Print();
   }

   abstract class Shape : IComparable, IPrint
   {
      public double Area { get; set; }
      public abstract double CalculateArea();
      public int CompareTo(object? obj)
      {
         if (obj == null) return 1;

         if (obj is Shape otherShape)
            return this.Area.CompareTo(otherShape.Area);
         else
            throw new ArgumentException("Object is not a Shape");
      }

      public abstract void Print();
   }

   class Rectangle : Shape
   {
      private double _width;
      private double _height;
      public double Width
      {
         get { return this._width; }
         set { this._width = (value > 0 ? value : 0); Area = CalculateArea(); }
      }
      public double Height
      {
         get { return this._height; }
         set { this._height = (value > 0 ? value : 0); Area = CalculateArea(); }
      }
      public Rectangle(double width, double height)
      {
         Width = width;
         Height = height;
      }
```

```
    public override double CalculateArea()
    {
        return Width * Height;
    }

    public override string ToString()
    {
        return $"Rectangle, Area = {Area}";
    }

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

}

class Square : Rectangle
{
    public Square(double width) : base(width, width)
    {
    }
    public override string ToString()
    {
        return $"Square, Area = {Area}";
    }
}

class Circle : Shape
{
    private double _radius;
    public double Radius
    {
        get { return this._radius; }
        set { this._radius = (value > 0 ? value : 0); Area = CalculateArea(); }
    }
    public Circle(double radius)
    {
        Radius = radius;
    }
    public override double CalculateArea()
    {
        return Math.PI * Radius * Radius;
    }
    public override string ToString()
    {
        return $"Circle, Area = {Math.Round(Area, 2)}";
    }
    public override void Print()
    {
        Console.WriteLine(this.ToString());
```

```
            }

        }
    }
```

IMatrixCheckEmpty.cs

```
using System;

    namespace lab_3
    {
        /// <summary>
        /// Проверка пустого элемента матрицы
        /// </summary>
        public interface IMatrixCheckEmpty<T>
        {
            /// <summary>
            /// Возвращает пустой элемент
            /// </summary>
            T getEmptyElement();

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

ShapeMatrixCheckEmpty.cs

```
using lab_3;
using System;

namespace lab_3
{
    class ShapeMatrixCheckEmpty : IMatrixCheckEmpty<Shape>
    {
        /// <summary>
        /// В качестве пустого элемента возвращается null
        /// </summary>
        public Shape getEmptyElement()
        {
            return null;
        }

        /// <summary>
        /// Проверка что переданный параметр равен null
        /// </summary>
        public bool checkEmptyElement(Shape element)
        {
            bool Result = false;
            if (element == null)
            {
```

```
            Result = true;
        }
        return Result;
      }
    }
}
```

Matrix.cs

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

namespace lab_3
{
  public class Matrix<T>
  {
    Dictionary<string, T> _matrix = new Dictionary<string, T>();

    int maxX;
    int maxY;
    int maxZ;

    IMatrixCheckEmpty<T> checkEmpty;

    public Matrix(int px, int py, int pz, IMatrixCheckEmpty<T> checkEmptyParam)
    {
      this.maxX = px;
      this.maxY = py;
      this.maxZ = pz;
      this.checkEmpty = checkEmptyParam;
    }

    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();
        }
    }
}

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");
    }
}

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++)
    {
        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");
        }
        b.Append("\n");
    }

    return b.ToString();
  }
 }
}
```

SimpleListItem.cs

```
using System;

namespace FigureCollections
{
    /// <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;
        }
    }
}
```

SimpleList.cs

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

namespace FigureCollections
{
   /// <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;
        }
    }
}
```

SimpleStack.cs

```
using System;

namespace FigureCollections
{
    class SimpleStack<T> : SimpleList<T> where T : IComparable
    {
        public void Push(T element)
        {
            Add(element);
        }

        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;
        }
    }
}
```

Вывод программы