Московский Государственный Технический Университет имени Н. Э. Баумана

Базовые компоненты интернет-технологий

Лабораторная работа №3

ИСПОЛНИТЕЛЬ:

ФИО Болотов Н.А.

Группа РТ5-31

ПРЕПОДАВАТЕЛЬ:

ФИО Гапанюк Ю. Е.

Москва 2017

Описание задания лабораторной работы.

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.

Код программы:

```
using System;
          using System.Collections.Generic;
          using System.Linq;
          using System.Text;
          using System.Threading.Tasks;
          namespace _laba
          {
              public interface IMatrixCheckEmpty<T>
              {
                  T getEmptyElement();
                  bool checkEmptyElement(T element);
              }
              class FigureMatrixCheckEmpty : IMatrixCheckEmpty<Figure>
              {
                  public Figure getEmptyElement()
                  {
                      return null;
                  }
                  public bool checkEmptyElement(Figure element)
                  {
                      bool Result = false;
                      if (element == null)
                      {
                          Result = true;
                      }
                      return Result;
                  }
              }
              public class Matrix<T>
              {
                  Dictionary<string, T> _matrix = new Dictionary<string, T>();
                  int maxX;
                  int maxY;
                  int maxZ;
                  IMatrixCheckEmpty<T> checkEmpty;
                  public Matrix(int x, int y, int z, IMatrixCheckEmpty<T> checkEmptyParam)
                  {
                      this.maxX = x;
                      this.maxY = y;
                      this.maxZ = z;
                      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 + " out of
bounds");
            }
            if (y < 0 || y >= this.maxY)
            {
                throw new ArgumentOutOfRangeException("y", "y=" + y + " out of
bounds");
            }
            if (z < 0 || z >= this.maxZ)
            {
                throw new ArgumentOutOfRangeException("z", "z=" + z + "  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 i = 0; i < this.maxX; i++)
            {
```

```
            b.Append("[");
            for (int j = 0; j < this.maxY; j++)
            {
                b.Append("[");
                for (int k = 0; k < this.maxZ; k++)
                {
                    if (!this.checkEmpty.checkEmptyElement(this[i, j, k]))
                        b.Append(this[i, j, k].ToString());
                    else
                        b.Append("");
                }
                if (j < this.maxY - 1)
                    b.Append("], ");
                else
                    b.Append("]");
            }
            b.Append("]\n");
        }
        return b.ToString();
    }
}
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 index's bound");
    }
    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(j).CompareTo(x) > 0) --j;
            if (i <= j)
            {
                Swap(i, j);
                i++; j--;
            }
        }
        while (i <= j);
        if (low < j) Sort(low, j);
        if (high > j) 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;
    }
}
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)
        {
```

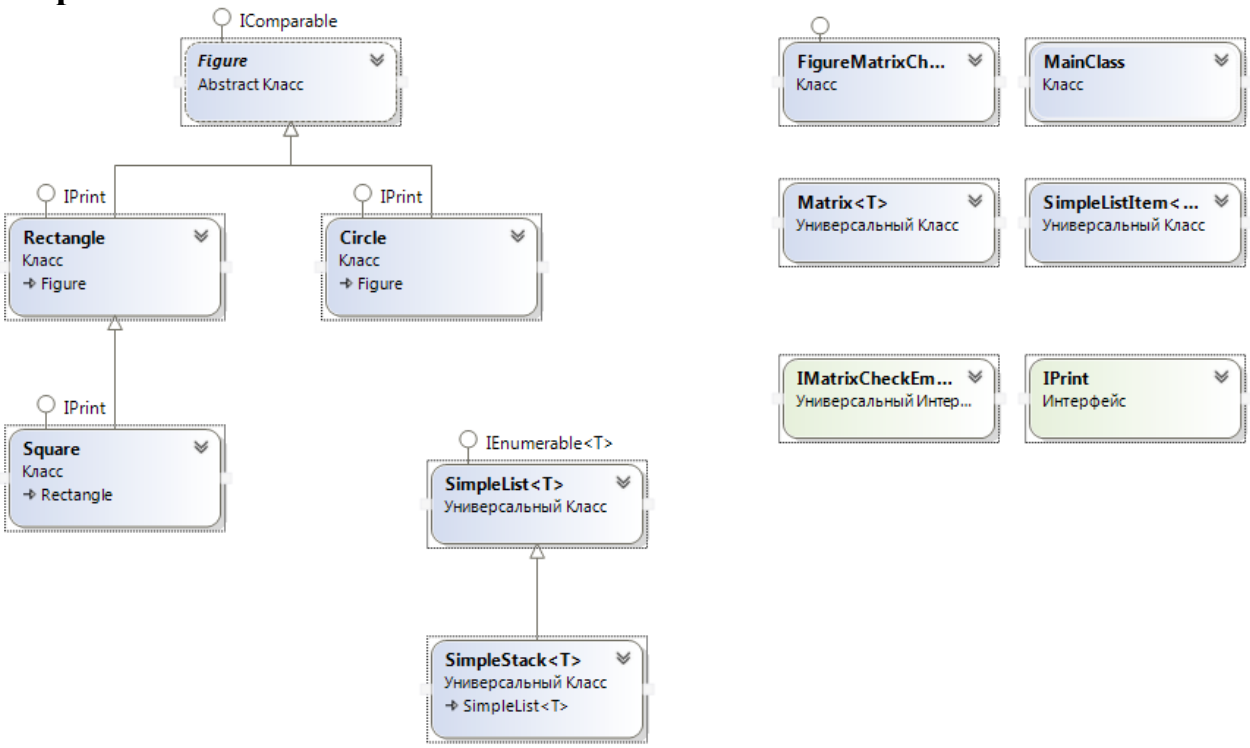
```
            Result = this.first.data;
            this.first = null;
            this.last = null;
        }
        else
        {
            SimpleListItem<T> newLast = this.GetItem(this.Count - 2);
            Result = newLast.next.data;
            this.last = newLast;
            newLast = null;
        }
        this.Count--;
        return Result;
    }
}
/// <summary>
/// Figure.
/// </summary>
abstract class Figure : IComparable
{
    public string Type
    { get; set; }
    public abstract double Area();
    public override string ToString()
    {
        return this.Type + "area " + 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;
    }
}
/// <summary>
/// Print.
/// </summary>
interface IPrint
{
    void Print();
}
/// <summary>
/// Rectangle.
/// </summary>
```

```
class Rectangle : Figure, IPrint
{
    double height;
    double width;
    public Rectangle(double ph, double pw)
    {
        this.height = ph;
        this.width = pw;
        this.Type = " rectangle ";
    }
    public override double Area()
    {
        double Result = this.width * this.height;
        return Result;
    }
    public void Print()
    {
        Console.WriteLine(this.ToString());
    }
}
/// <summary>
/// Square.
/// </summary>
class Square : Rectangle, IPrint
{
    public Square(double size) : base(size, size)
    {
        this.Type = " square ";
    }
}
/// <summary>
/// Circle.
/// </summary>
class Circle : Figure, IPrint
{
    double rad;
    public Circle(double pr)
    {
        this.rad = pr;
        this.Type = " Circle ";
    }
    public override double Area()
    {
        double Result = Math.PI * this.rad * this.rad;
        return Result;
```

```
        }
        public void Print()
        {
            Console.WriteLine(this.ToString());
        }
    }
    /// <summary>
    /// Main class.
    /// </summary>
    class MainClass
    {
        public static void Main(string[] args)
        {
            Matrix<Figure> matrix = new Matrix<Figure>(3, 3, 3, new
FigureMatrixCheckEmpty());
            Rectangle rect = new Rectangle(5, 4);
            Square square = new Square(5);
            Circle cir = new Circle(5);
            rect.Print();
            square.Print();
            cir.Print();
            List<Figure> fl = new List<Figure>();
            fl.Add(cir);
            fl.Add(rect);
            fl.Add(square);
            Console.WriteLine("Before sort");
            foreach (var x in fl) Console.WriteLine(x);
            fl.Sort();
            Console.WriteLine("After sort");
            foreach (var x in fl) Console.WriteLine(x);
            matrix[0, 0, 0] = rect;
            matrix[1, 1, 1] = square;
            matrix[2, 2, 2] = cir;
            Console.WriteLine(matrix.ToString());
            SimpleStack<Figure> stack = new SimpleStack<Figure>();
            stack.Push(rect);
            stack.Push(square);
            stack.Push(cir);
            while(stack.Count>0)
            {
                Figure f = stack.Pop();
                Console.WriteLine(f);
            }
            Console.ReadKey();
        }
```

```
        }
    }
```

Диаграмма классов:



Пример консольного вывода: