Лабораторная работа № 3 по курсу
“Базовые компоненты интернет-технологий”

Алексеев А.В.
РТ5-31
МГТУ им. Баумана

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:

Figure:

```
abstract class Figure: IComparable
    {
        public string Shape { get; set; }
        public abstract double Area();
        public override string ToString()
        {
            return this.Shape + " площадью " + 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;
        }


    }
    #region Прямоугольник
    class Rectangle : Figure, IPrint
    {
        protected double Width { get; set; }
        protected double Length { get; set; }
        public Rectangle(double Width, double Length)
        {
            this.Shape = "Прямоугольник";
            this.Width = Width;
            this.Length = Length;
        }
        public override double Area()
        {
            double result = this.Width * this.Length;
            return result;
        }
        public void Print()
        {
            Console.WriteLine(this.ToString());
        }
    }
    #endregion
    #region Квадрат
    class Square : Rectangle, IPrint
    {
        public Square(double Length) : base(Length, Length)
        {
            this.Shape = "Квадрат";
            this.Length = Length;

        }
        public override double Area()
        {
            double result = Math.Pow(Length, 2);
            return result;
        }
        public override string ToString()
        {
            return this.Shape + " площадью " + this.Area().ToString();
        }
```

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

    }
    #endregion
    #region Круг
    class Circle : Figure, IPrint
    {
        private double radius;
        public Circle(double radius)
        {
            this.Shape = "Круг";
            this.radius = radius;
        }
        public override double Area()
        {
            double result = Math.Pow(radius, 2) * 3.14;
            return result;
        }
        public override string ToString()
        {
            return this.Shape + " площадью " + this.Area().ToString();
        }
        public void Print()
        {
            Console.WriteLine(this.ToString());
        }
    }
    #endregion
```

Matrix:

```
public class Matrix<T>
    {
        Dictionary<string, T> mtrx = new Dictionary<string, T>();
        int maxX;//максимальное колличесвто элементов по X
        int maxY;//максимальное колличесвто элементов по Y
        int maxZ;//максимальное колличесвто элементов по Z
        T nullElement;
        public Matrix(int py,int px,int pz,T pnullElement)
        {
            this.maxX = px;
            this.maxY = py;
            this.maxZ = pz;
            this.nullElement = pnullElement;
        }
        //индексатор для доступа к данным
        public T this[int x,int y,int z]
        {
            set
            {
                CheckBounds(x, y, z);
                string key = DictKey(x, y, z);
                this.mtrx.Add(key, value);
            }
            get
            {
                CheckBounds(x, y, z);
                string key = DictKey(x, y, z);
                if (this.mtrx.ContainsKey(key))
                {
```

```
                    return this.mtrx[key];
                }
                else {
                    return this.nullElement;
                        }


            }
        }
        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=" + x + " выходит за границы.");
            if ((z < 0) || (z > this.maxZ)) throw new ArgumentOutOfRangeException("Z",
"Z=" + x + " 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.maxY; k++)
            {
                b.Append("[");
                for (int j = 0; j < maxY; j++)
                {
                    if (j > 0) b.Append("\t");
                    b.Append("[");
                    for (int i = 0; i < maxX; i++)
                    {
                        if (this[i, j, k] != null)
                            b.Append(this[i, j, k].ToString());
                        else
                            b.Append("0");
                        if (i != (maxX - 1)) b.Append(", ");
                    }
                    b.Append("]");
                }

                b.Append("]\n");
            }
            return b.ToString();
        }
    }
```

Simple List:

```
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("Выход за границу ");
    }
    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; //возврат тек. знач.
        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 (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 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;
    }
}
```

Program:

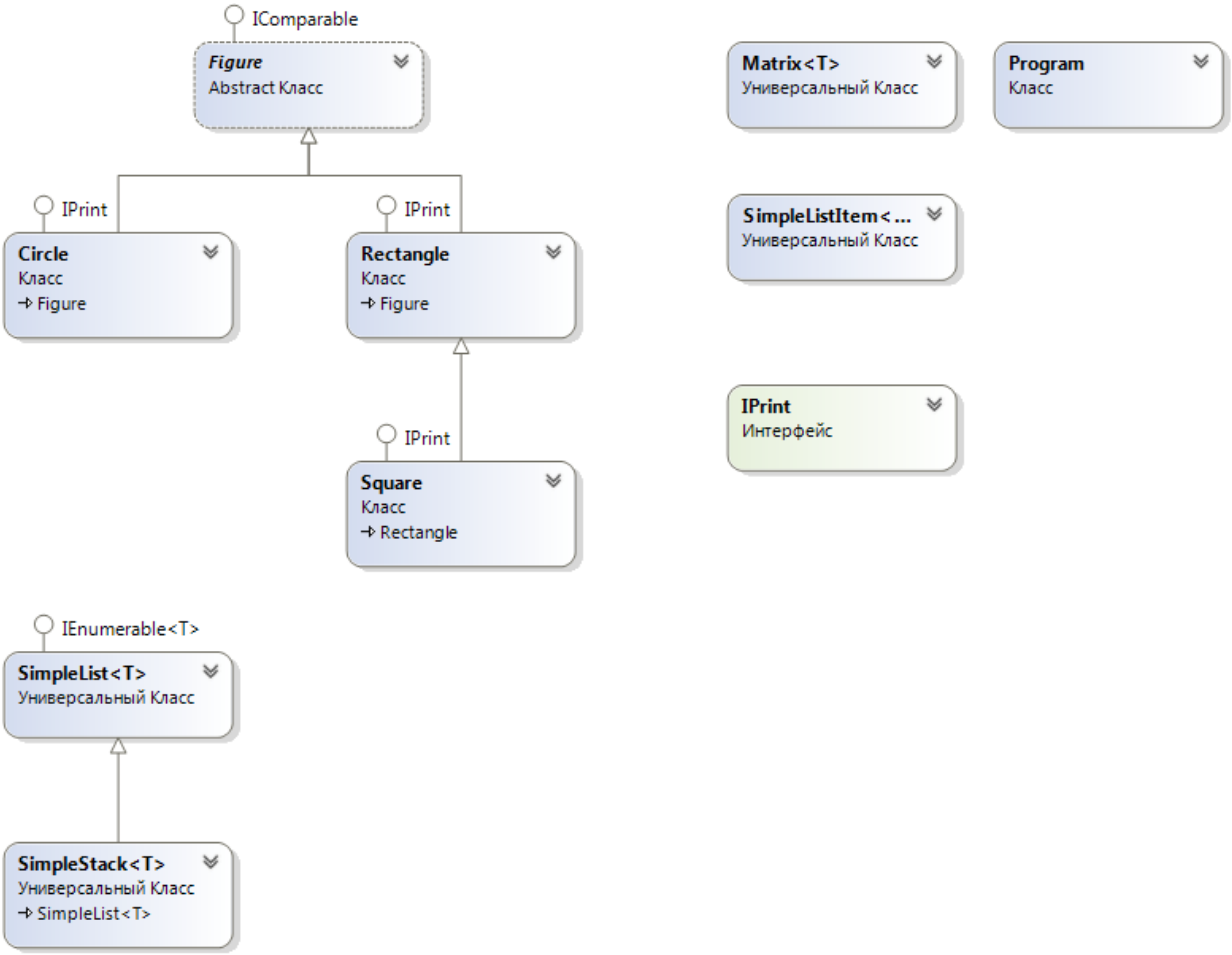
```
class Program
    {
        static void Main(string[] args)
        {
            Rectangle rect = new Rectangle(5, 5);
            Square square = new Square(5);
            Circle circle = new Circle(1);
           // rect.Print();
           // square.Print();
           // circle.Print();
            ArrayList GFa = new ArrayList();//создание необобщенной коллекции.
            GFa.Add(rect);
```

```
            GFa.Add(square);
            GFa.Add(circle);
            Console.WriteLine("\nПеред сортировкой необобщённой коллекции");
            foreach (object i in GFa)  Console.WriteLine(i);
            GFa.Sort();
            Console.WriteLine("\nПосле сортировки");
            foreach (object i in GFa)
            Console.WriteLine(i);
            {
                string type = rect.GetType().Name;
                Console.WriteLine("Определение типа данных метода прямоугольник");
                if (type == "Int32")
                {
                    Console.WriteLine("Целое число:" + rect.ToString());
                }
                else if (type == "String")
                {
                    Console.WriteLine("Строка:" + rect.ToString());
                }
                else Console.WriteLine("Другой тип");
            }
            List< Figure> GFl = new List<Figure>();//создание обобщенной коллекции.
            GFl.Add(rect);
            GFl.Add(square);
            GFl.Add(circle);
            Console.WriteLine("\nПеред сортировкой обобщённой коллекции");
            foreach (var i in GFl) Console.WriteLine(i);
            GFl.Sort();
            Console.WriteLine("\nПосле сортировки");
            foreach (var i in GFl) Console.WriteLine(i);
            Console.WriteLine("Разреженная матрица");
        Matrix<Figure> cub = new Matrix<Figure>(3,3,3,null);
        cub[0, 0, 0] = rect;
        cub[1, 1, 1] = square;
        cub[2, 2, 2] = circle;
        Console.WriteLine(cub.ToString());
        Console.WriteLine("Стэк");
        SimpleStack<Figure> stack = new SimpleStack<Figure>();
        stack.Push(rect);
        stack.Push(square);
        stack.Push(circle);
        while(stack.Count>0)
        {
            Figure f = stack.Pop();
            Console.WriteLine(f);
        }
            Console.ReadKey();
        }
    }
```

Интерефейс IPrint:

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

Диаграмма классов:



Пример консольного вывода:

```
Круг площадью 3,14
Квадрат площадью 25
Прямоугольник площадью 25
Определение типа данных метода прямоугольник
Другой тип

Перед сортировкой обобщённой коллекции
Прямоугольник площадью 25
Квадрат площадью 25
Круг площадью 3,14

После сортировки
Круг площадью 3,14
Квадрат площадью 25
Прямоугольник площадью 25
Разреженная матрица
[[Прямоугольник площадью 25, 0, 0]      [0, 0, 0]       [0, 0, 0]]
[[0, 0, 0]      [0, Квадрат площадью 25, 0]     [0, 0, 0]]
[[0, 0, 0]      [0, 0, 0]       [0, 0, Круг площадью 3,14]]

Стэк
Круг площадью 3,14
Квадрат площадью 25
Прямоугольник площадью 25
```