**Отчет по Лабораторной работе №3**

Студентка группы ИУ5-34

Лескина Виктория

Дата: 15.11.17

**Формулировка 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 Matrix (представлен в разделе «Вспомогательные материалы для выполнения лабораторных работ») 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 (представлен в разделе 9 «Вспомогательные материалы для выполнения лабораторных работ»). Необходимо добавить в класс методы:

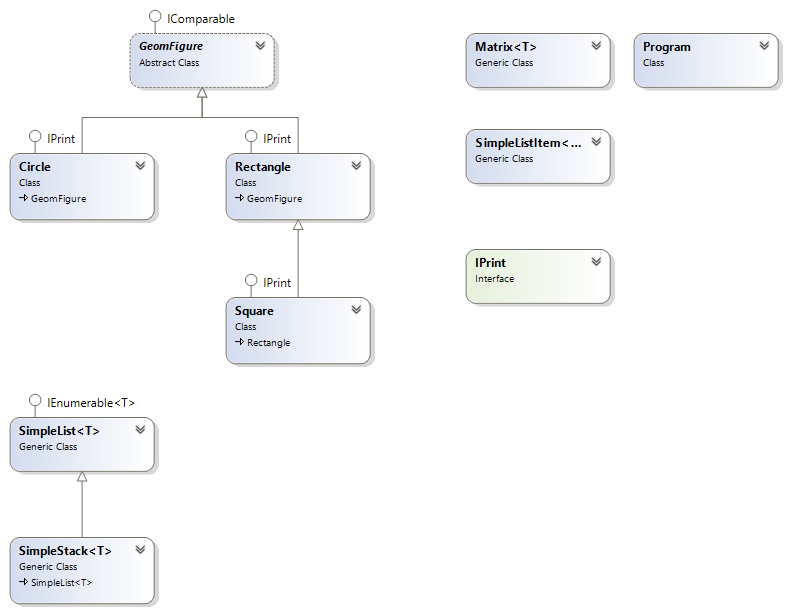
- public void Push(T element) – добавление в стек;

- public T Pop() – чтение с удалением из стека.

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

**Код**

|  |
| --- |
| UsingSystem; |
|  | using System.Collections.Generic; |
|  | using System.Collections; |
|  | using System.Linq; |
|  | using System.Text; |
|  | using System.Threading.Tasks; |
|  |  |
|  |  |
|  | namespace Lab3 |
|  | { |
|  | public class Matrix<T> |
|  | { |
|  | /// <summary> |
|  | /// Dictionary to store data |
|  | /// </summary> |
|  | Dictionary<string, T> \_matrix = new Dictionary<string, T>(); |
|  | int maxX; // max number of columns |
|  | int maxY; // max number of strings |
|  | int maxZ; |
|  | string name; |
|  |  |
|  | T nullElement; // Empty element, returns if there is no element with needed coordinats |
|  | public Matrix(string name, int px, int py, int pz, T nullElementParam) //Constructor |
|  | { |
|  | this.maxX = px; |
|  | this.maxY = py; |
|  | this.maxZ = pz; |
|  | this.name = name; |
|  | this.nullElement = nullElementParam; |
|  | } |
|  |  |
|  | void CheckBounds(int x, int y, int z) |
|  | { |
|  | if (x < 0 || x >= this.maxX) throw new Exception("x=" + x + " out of bounds"); |
|  | if (y < 0 || y >= this.maxY) throw new Exception("y=" + y + " out of bounds"); |
|  | if (z < 0 || z >= this.maxZ) throw new Exception("z=" + z + " out of bounds"); |
|  | } |
|  | /// <summary> |
|  | /// Form the key |
|  | /// </summary> |
|  | string DictKey(int x, int y, int z) |
|  | { |
|  | return x.ToString() + "\_" + y.ToString() + "\_" + z.ToString(); |
|  | } |
|  |  |
|  | /// <summary> |
|  | /// Indexer to access data |
|  | /// </summary> |
|  | 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); |
|  | } |
|  | } |
|  |  |
|  | public override string ToString() |
|  | { |
|  | //Class StringBuilder - for long strings, faster then making from lots of normal str |
|  | StringBuilder b = new StringBuilder(); |
|  | for (int i = 0; i < this.maxZ; i++) |
|  | { |
|  | for (int j = 0; j < this.maxY; j++) |
|  | { |
|  |  |
|  | for (int z = 0; z < this.maxX; z++) |
|  | { |
|  | if (z > 0) b.Append("\t"); |
|  | b.Append(this.name + "[" + i + "," + j + "," + z + "]: "); |
|  | if (this[i, j, z].Equals(nullElement)) b.Append("-" + "\t\t"); |
|  | else b.Append(this[i, j, z].ToString() + "\t"); |
|  | } |
|  | b.Append("\n"); |
|  | } |
|  | } |
|  | return b.ToString(); |
|  | } |
|  | } |
|  | interface IPrint |
|  | { |
|  | void Print(); |
|  | } |
|  | /// <summary> |
|  | /// Abstract class |
|  | /// </summary> |
|  | abstract class GeomFigure : IComparable |
|  | { |
|  | public abstract double Area(); |
|  | public int CompareTo(object ob) //with IComparable |
|  | { |
|  | GeomFigure f = (GeomFigure)ob; |
|  | if (this.Area() == f.Area()) return 0; |
|  | else if (this.Area() > f.Area()) return 1; |
|  | else return -1; |
|  |  |
|  | } |
|  |  |
|  | } |
|  | class Rectangle : GeomFigure, IPrint |
|  | { |
|  | //------------------properties------------------------------ |
|  | private double \_width; //data storage for properties |
|  | public double width //definition |
|  | { |
|  | get { return \_width; } |
|  | set { \_width = value; } |
|  | } |
|  |  |
|  | public double height { get; set; } //autodefinition |
|  | //---------------------------------------------------------- |
|  |  |
|  | public Rectangle(double w, double h) { \_width = w; height = h; } //constructor |
|  |  |
|  | public override double Area() |
|  | { |
|  | return \_width \* height; |
|  | } |
|  |  |
|  | public override string ToString() |
|  | { |
|  | return "Rectangle area = " + Area().ToString(); |
|  | } |
|  |  |
|  | public void Print() |
|  | { |
|  | System.Console.WriteLine(ToString()); |
|  | } |
|  |  |
|  | } |
|  | class Square : Rectangle, IPrint |
|  | { |
|  | public Square(double s) : base(s, s) { } //call of rect constructor in square constructor |
|  | public override string ToString() |
|  | { |
|  | return "Square area = " + Area().ToString(); |
|  | } |
|  |  |
|  | public void Print() |
|  | { |
|  | System.Console.WriteLine(ToString()); |
|  | } |
|  |  |
|  | } |
|  | class Circle : GeomFigure, IPrint |
|  | { |
|  | public double r { get; set; } |
|  | public Circle(double r) { this.r = r; } |
|  |  |
|  | public override double Area() |
|  | { |
|  | return 2 \* Math.PI \* r \* r; |
|  | } |
|  |  |
|  | public override string ToString() |
|  | { |
|  | return "Circle area = " + Area().ToString(); |
|  | } |
|  |  |
|  | public void Print() |
|  | { |
|  | System.Console.WriteLine(ToString()); |
|  | } |
|  | } |
|  |  |
|  | /// <summary> |
|  | /// Element in List |
|  | /// </summary> |
|  | public class SimpleListItem<T> |
|  | { |
|  | public T data { get; set; } |
|  | public SimpleListItem<T> nextElement { get; set; } //like pointer |
|  | public SimpleListItem(T param) { this.data = param; } //constructor |
|  |  |
|  | } |
|  | /// <summary> |
|  | /// List |
|  | /// </summary> |
|  | public class SimpleList<T> : IEnumerable<T> |
|  | where T : IComparable |
|  | { |
|  | protected SimpleListItem<T> firstElement = null; |
|  | protected SimpleListItem<T> lastElement = null; |
|  | int \_count; |
|  | public int numberOfElements |
|  | { |
|  | get { return \_count; } |
|  | protected set { \_count = value; } |
|  | } |
|  |  |
|  | public void AddElement(T element) |
|  | { |
|  | SimpleListItem<T> newItem = new SimpleListItem<T>(element); |
|  | this.numberOfElements++; |
|  | if (lastElement == null) //add first element |
|  | { |
|  | this.firstElement = newItem; |
|  | this.lastElement = newItem; |
|  | } |
|  |  |
|  | else //add following elements |
|  | { |
|  | this.lastElement.nextElement = newItem; |
|  | this.lastElement = newItem; |
|  | } |
|  | } |
|  | /// <summary> |
|  | ///Get container by number |
|  | /// </summary> |
|  | public SimpleListItem<T> GetItem(int number) |
|  | { |
|  | if ((number < 0) || (number >= this.numberOfElements)) |
|  | { |
|  | throw new Exception("Out of bounds"); |
|  | } |
|  | SimpleListItem<T> current = this.firstElement; |
|  | int i = 0; |
|  | while (i < number) |
|  | { |
|  | current = current.nextElement; //go to the next element |
|  | i++; |
|  | } |
|  | return current; |
|  | } |
|  | /// <summary> |
|  | /// Read element by number |
|  | /// </summary> |
|  | public T Get(int number) |
|  | { |
|  | return GetItem(number).data; |
|  | } |
|  | /// <summary> |
|  | /// Для перебора коллекции |
|  | /// </summary> |
|  | public IEnumerator<T> GetEnumerator() |
|  | { |
|  | SimpleListItem<T> current = this.firstElement; |
|  | //Перебор элементов |
|  | while (current != null) |
|  | { |
|  | yield return current.data; |
|  | current = current.nextElement; |
|  | } |
|  | } |
|  | //Реализация обощенного IEnumerator<T> требует реализации необобщенного интерфейса |
|  | //Данный метод добавляется автоматически при реализации интерфейса |
|  | System.Collections.IEnumerator |
|  | System.Collections.IEnumerable.GetEnumerator() |
|  | { |
|  | return GetEnumerator(); |
|  | } |
|  |  |
|  | public void Sort() |
|  | { |
|  | QuickSort(0, this.numberOfElements - 1); |
|  | } |
|  |  |
|  | private void QuickSort(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) QuickSort(low, j); |
|  | if (i < high) QuickSort(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; |
|  | } |
|  | } |
|  |  |
|  | public class SimpleStack<T> : SimpleList<T> where T : IComparable |
|  | { |
|  | public void Push(T element) |
|  | { |
|  | AddElement(element); |
|  | } |
|  |  |
|  | public T Pop() |
|  | { |
|  | T res = default(T); |
|  | if (this.numberOfElements == 0) return res; |
|  |  |
|  | if (this.numberOfElements == 1) |
|  | { |
|  | res = this.firstElement.data; |
|  | this.firstElement = null; |
|  | this.lastElement = null; |
|  |  |
|  | } |
|  | else |
|  | { |
|  | SimpleListItem<T> newLast = this.GetItem(this.numberOfElements - 2); |
|  | res = newLast.nextElement.data; |
|  | this.lastElement = newLast; |
|  | newLast.nextElement = null; |
|  |  |
|  | } |
|  | this.numberOfElements--; |
|  | return res; |
|  |  |
|  | } |
|  | } |
|  |  |
|  | class Program |
|  | { |
|  |  |
|  | static void Main(string[] args) |
|  | { |
|  | double a = 3, b = 4, r = 5; |
|  | Rectangle rec = new Rectangle(a, b); |
|  | Square sq = new Square(a); |
|  | Circle circ = new Circle(r); |
|  |  |
|  | ArrayList al = new ArrayList(); |
|  | al.Add(rec); |
|  | al.Add(sq); |
|  | al.Add(circ); |
|  | al.Sort(); |
|  | Console.WriteLine("ArrayList"); |
|  | foreach (object obj in al) Console.WriteLine(obj.ToString()); //or through str type = obj.GetType().Name + ifs |
|  |  |
|  | Rectangle rec2 = new Rectangle(4, 5); |
|  | Square sq2 = new Square(4); |
|  | Circle circ2 = new Circle(1); |
|  | List<GeomFigure> list = new List<GeomFigure>(); |
|  | list.Add(rec2); |
|  | list.Add(sq2); |
|  | list.Add(circ2); |
|  | list.Sort(); |
|  | Console.WriteLine("\nList with " + list.Count() + " elements"); |
|  | foreach (GeomFigure figure in list) |
|  | { |
|  | Console.WriteLine(figure.ToString()); |
|  | } |
|  |  |
|  | //----------------------------Sparse Matrix----------------------------------- |
|  | Console.WriteLine("\nMatrix"); |
|  | Square nullFigure = new Square(0); |
|  | Matrix<GeomFigure> matr = new Matrix<GeomFigure>("matr", 3, 3, 3, nullFigure); |
|  | matr[0, 0, 0] = rec; |
|  | matr[1, 1, 1] = sq; |
|  | matr[2, 2, 2] = circ; |
|  | Console.WriteLine(matr.ToString()); |
|  | //-------------------------------------------------------------------- |
|  |  |
|  | //---------------SimpleList------------------------------------- |
|  | Console.WriteLine("\nSimpleList"); |
|  | SimpleList<GeomFigure> SimpleL = new SimpleList<GeomFigure>(); |
|  | SimpleL.AddElement(rec); |
|  | SimpleL.AddElement(circ); |
|  | SimpleL.AddElement(sq); |
|  | foreach (var x in SimpleL) Console.WriteLine(x); |
|  | //-------------------------------------------------------------- |
|  |  |
|  | //---------------SimpleStack------------------------------------- |
|  | Console.WriteLine("\nSimpleStack"); |
|  | SimpleStack<GeomFigure> stack = new SimpleStack<GeomFigure>(); |
|  | stack.Push(rec); |
|  | stack.Push(sq); |
|  | stack.Push(sq); |
|  | stack.Push(circ); |
|  | foreach (var x in stack) Console.WriteLine(x); |
|  | Console.WriteLine("\nStack Pop result: " + stack.Pop().ToString()); |
|  | Console.WriteLine("\nStack after Pop"); |
|  | foreach (var x in stack) Console.WriteLine(x); |
|  | //-------------------------------------------------------------- |
|  |  |
|  |  |
|  | Console.ReadKey(); |
|  | } |
|  |  |
|  | } |
|  | }  **Диаграмма классов** |

****

**Результаты работы**

