Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

«Московский государственный технический университет имени Н.Э. Баумана»   
(МГТУ им. Н.Э. Баумана)

**Отчет**

**по дисциплине**

**“Базовые компоненты интернет технологий”**

**по лабораторной работе №3**

Выполнил: Окопный Марк Олегович

Группа: ИУ5 – 34

**Описание задания**

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.

**Текст программы**

**Program.cs**

|  |
| --- |
| using System; |
|  | using System.Collections.Generic; |
|  | using System.Linq; |
|  | using System.Text; |
|  | using System.Threading.Tasks; |
|  | using System.Collections; |
|  | using Classes; |
|  |  |
|  | namespace Lab3 |
|  | { |
|  | class Program |
|  | { |
|  | static void Main(string[] args) |
|  | { |
|  | Rectangle rect = new Rectangle(10, 5); |
|  | Square sq = new Square(5); |
|  | Circle sc = new Circle(5); |
|  | Console.WriteLine("\nArraylist"); |
|  | ArrayList list = new ArrayList(); |
|  | list.Add(rect); |
|  | list.Add(sq); |
|  | list.Add(sc); |
|  | foreach (var x in list) Console.WriteLine(x); |
|  | Console.WriteLine("\nList Figure"); |
|  | List<Figure> lst = new List<Figure>(); |
|  | lst.Add(rect); |
|  | lst.Add(sq); |
|  | lst.Add(sc); |
|  | foreach (var i in lst) Console.WriteLine(i); |
|  | Console.WriteLine("\nList<Figure> sort"); |
|  | lst.Sort(); |
|  | foreach (var i in lst) Console.WriteLine(i); |
|  | Console.WriteLine("\nMatrix3D"); |
|  | Matrix3D<Figure> matr = new Matrix3D<Figure>(3,3,3,null); |
|  | matr[0, 0, 0] = sq; |
|  | matr[1, 1, 1] = sc; |
|  | matr[2, 2, 2] = rect; |
|  | Console.WriteLine(matr[1, 1, 1].ToString()); |
|  | Console.WriteLine(matr.ToString()); |
|  | Console.WriteLine("\nSimple List"); |
|  | SimpleList<Figure> list1 = new SimpleList<Figure>(); |
|  | list1.Add(sq); |
|  | list1.Add(sc); |
|  | list1.Add(rect); |
|  | list1.Add(sq); |
|  | list1.Add(sc); |
|  | foreach (var x in list1) Console.WriteLine(x); |
|  | list1.Sort(); |
|  | Console.WriteLine("\n Simple List sort"); |
|  | foreach (var x in list1) Console.WriteLine(x); |
|  | Console.WriteLine("\nMy shit kek"); |
|  | //list1.DeleteLast(); |
|  | foreach (var x in list1) Console.WriteLine(x); |
|  | Console.WriteLine("\nSimple Stack");// реализовать |
|  | SimpleStack<Figure> stck = new SimpleStack<Figure>(); |
|  | stck.push(sq); |
|  | stck.push(sc); |
|  | stck.push(rect); |
|  | foreach (var x in stck) Console.WriteLine(x); |
|  | Console.WriteLine("\npop"); |
|  | stck.pop(); |
|  | stck.pop(); |
|  | stck.pop(); |
|  | Console.ReadKey(); |
|  |  |
|  | } |
|  | } |
|  | } |

**Classes.cs**

|  |
| --- |
| using System; |
|  | using System.Collections.Generic; |
|  | using System.Linq; |
|  | using System.Text; |
|  | using System.Threading.Tasks; |
|  |  |
|  | namespace Classes |
|  | { |
|  |  |
|  | abstract class Figure: IComparable |
|  | { |
|  | public string Type; |
|  | public abstract double Area(); |
|  | public override string ToString() |
|  | { |
|  | return this.Type + " с 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; //(this.Area() > p.Area()) |
|  | } |
|  |  |
|  |  |
|  | } |
|  | interface IPrint |
|  | { |
|  | void Print(); |
|  | } |
|  | class Rectangle : Figure, IPrint |
|  | { |
|  | double height; |
|  | double width; |
|  | public Rectangle(double wid, double hei) |
|  | { |
|  | this.width = wid; |
|  | this.height = hei; |
|  | this.Type = "Прямоугольник"; |
|  | } |
|  | public override double Area() |
|  | { |
|  | double Result = this.width \* this.height; |
|  | return Result; |
|  | } |
|  | public void Print() |
|  | { |
|  | Console.WriteLine(this.ToString()); |
|  | } |
|  | } |
|  | class Square : Rectangle, IPrint |
|  | { |
|  | public Square(double len) |
|  | : base(len, len) |
|  | { |
|  | this.Type = "Квадрат"; |
|  | } |
|  | } |
|  | class Circle : Figure, IPrint |
|  | { |
|  | double radius; |
|  | public Circle(double rad) |
|  | { |
|  | this.radius = rad; |
|  | this.Type = "Круг"; |
|  | } |
|  | public override double Area() |
|  | { |
|  | const double pi = 3.14; |
|  | double Result = pi \* this.radius \* this.radius; |
|  | return Result; |
|  | } |
|  | public void Print() |
|  | { |
|  | Console.WriteLine(ToString()); |
|  | } |
|  | } |
|  | public class Matrix3D<T> |
|  | { |
|  | Dictionary<string, T> \_matrix = new Dictionary<string, T>(); |
|  | int maxX; |
|  | int maxY; |
|  | int maxZ; |
|  | T nullElement; |
|  | public Matrix3D(int px, int py, int pz, T nullElementParam) |
|  | { |
|  | this.maxX = px; |
|  | this.maxY = py; |
|  | this.maxZ = pz; |
|  | this.nullElement = nullElementParam; |
|  | } |
|  | 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); |
|  | } |
|  | } |
|  |  |
|  | 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.maxZ) throw new Exception("z=" + z + " выходит за границы"); |
|  |  |
|  | } |
|  | public string DictKey(int x, int y, int z) |
|  | { |
|  | return x.ToString() + "\_" + y.ToString() + "\_" + z.ToString(); |
|  | } |
|  | public override string ToString() |
|  | { |
|  | StringBuilder br = new StringBuilder(); |
|  | for (int i = 0; i < this.maxX; i++) |
|  | { |
|  | for (int j = 0; j < this.maxY; j++) |
|  | { |
|  | for (int k = 0; k < this.maxZ; k++) |
|  | { |
|  | br.Append("[" + i + "," + j + "," + k + "]"); |
|  | br.Append(this[i, j, k]+"\n"); |
|  | } |
|  |  |
|  |  |
|  | } |
|  | } |
|  | return br.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 num) |
|  | { |
|  | if (num < 0 || num > this.count) |
|  | { |
|  | throw new Exception("Выход за границу массива"); |
|  | } |
|  | SimpleListItem<T> current = this.first; |
|  | int i = 0; |
|  | while (i < num) |
|  | { |
|  | current = current.next; |
|  | i++; |
|  | } |
|  | return current; |
|  | } |
|  | public T Get(int num) |
|  | { |
|  | return GetItem(num).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 (i < high) Sort(i, high); |
|  | } |
|  | public void Swap(int i, int j) |
|  | { |
|  | SimpleListItem<T> xi = GetItem(i); |
|  | SimpleListItem<T> xj = GetItem(j); |
|  | T st = xi.Data; |
|  | xi.Data = xj.Data; |
|  | xj.Data = st; |
|  | } |
|  | public void printLast() |
|  | { |
|  | Console.WriteLine(this.last.Data.ToString()); |
|  | } |
|  | public void DeleteLast() |
|  | { |
|  | SimpleListItem<T> xn = this.first; |
|  | for (int i = 1; i < count - 1; i++) |
|  | { xn = xn.next; } |
|  | xn.next = this.last.next; |
|  | this.last = xn; |
|  | count--; |
|  | } |
|  |  |
|  | } |
|  | public class SimpleStack<T> : SimpleList<T> |
|  | where T : IComparable |
|  | { |
|  | public void push(T Data) |
|  | { |
|  | this.Add(Data); |
|  | } |
|  | public void pop() |
|  | { |
|  | this.printLast(); |
|  | this.DeleteLast(); |
|  | } |
|  |  |
|  | } |
|  |  |
|  | } |