Министерство образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Пермский национальный исследовательский политехнический университет»

Кафедра «Информационные технологии и автоматизированные системы»

Отчёт

по лабораторной работе №9

Дисциплина: Объектно-ориентированное программирование.

Тема: Классы и объекты.

Выполнил:

Cтудент группы ИВТ-20-2б

Тедеев Александр Зурабович

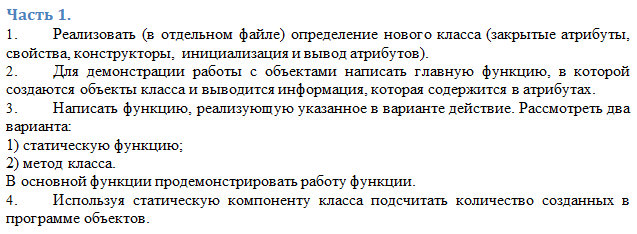
Проверила:

Доцент кафедры ИТАС

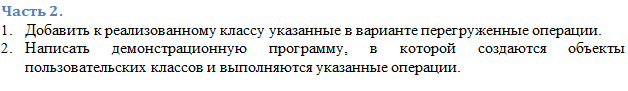
Викентьева Ольга Леонидовна

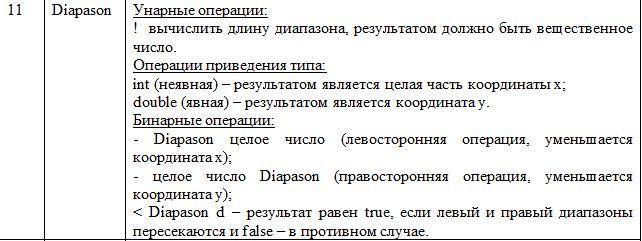
Пермь 2021

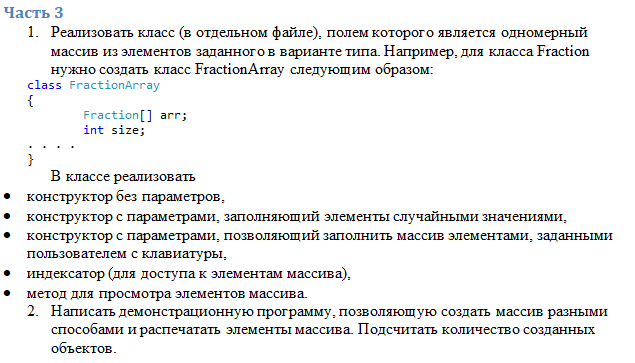
**Постановка задачи**















**Анализ**

* 1. Для выполнения данной задачи необходимо описать класс Diapason с полями \_x, \_y типа double и статическим полем counter (для подсчёта количества экземпляров класса Diapason), свойствами X, Y с методами get и set для получения значения полей \_x, \_y и установления значений к данным полям соответственно.

double \_x, \_y;

static int counter = 0;

public double X

{

get { return \_x; }

set

{

\_x = value;

}

}

public double Y

{

get { return \_y; }

set

{

if (value > X)

\_y = value;

else

throw new ArithmeticException();

}

}

* 1. Для свойства Y необходимо учесть случай, когда пользователь будет вводить значение для правой границы меньшее, чем для левой. В данном случае необходимо выбрасывать исключение, например, ArithmeticException.
  2. Чтобы узнать, пересекаются ли два диапазона, необходимо описать метод Intersect() (в качестве параметров передаётся второй диапазон), в котором сравниваются границы двух данных диапазонов.

if (Y < diaposon2.X || X > diaposon2.Y)

return false;

return true;

* 1. Данное действие необходимо повторить для статической функции класса Diapason, однако в данном случае оба диапазона передаются как параметры.
  2. Для задачи 2 необходимо в класс Diapason добавить перегрузки операторов. Для операторов неявного и явного преобразования необходимо использовать ключевые слова implicit и explicit.
  3. При перегрузке оператора – необходимо возвращать экземпляр класса Diapason с изменёнными границами (условия к изменению в задании).
  4. При перегрузке операторов сравнения и равенства/неравенства необходимо возвращать значение типа bool.
  5. Для выполнения задания 3 необходимо описать класс DiapasonArray, в котором основное поле – массив типа Diapason – arr, \_size типа int и статическое поле counter (для подсчёта количества экземпляров класса).

static Random rnd = new Random();

Diapason[] arr;

int \_size;

static int counter = 0;

* 1. Также необходимо использовать свойства для данных полей.

public int Size

{

get { return \_size; }

}

public int Count

{

get { return counter; }

}

* 1. Для перегрузки индексатора используется особое свойство, пример которого можно увидеть ниже. Добавлены проверки выхода за пределы массива Diapason, в случае выхода за пределы массива выбрасывается исключение IndexOutOfRangeException.

public Diapason this[int index]

{

get

{

if (index >= 0 && index < arr.Length)

return arr[index];

else

throw new IndexOutOfRangeException();

}

set

{

if (index >= 0 && index < arr.Length)

arr[index] = value;

else

throw new IndexOutOfRangeException();

}

}

* 1. Так как необходимо реализовать заполнение массива через ДСЧ и ручной ввод, используются два конструктора с параметром, где конструктор, который необходим для заполнения массива через ДСЧ, получает в качестве параметра только переменную с размером массива, а конструктор для заполнения получает фиктивную переменную.
  2. Для поиска максимального диапазона в массиве диапазонов необходима перегрузка индексатора. Объявляется преременная max типа Diapason, в цикле производится сравнение по критерию длины (перегрузка оператора !, который возвращает длину диапазона) диапазона; если найден элемент с большей длиной диапазона, то данное значение присваивается в переменную max.

Diapason max = diapasons1[0];

for (int i = 1; i < diapasons1.Size; i++)

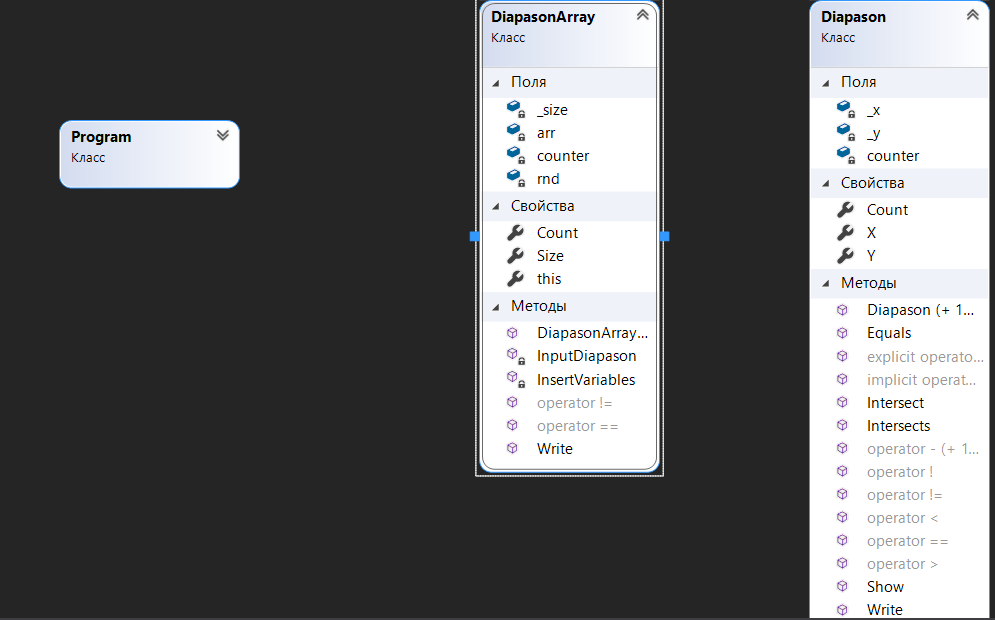
{

if (!max < !diapasons1[i])

max = diapasons1[i];

}

**Диаграммы классов**



**Код**

**Задача 1**

**Класс Diapason**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab9

{

public class Diapason

{

double \_x, \_y;

static int counter = 0;

public int Count

{

get { return counter; }

}

public double X

{

get { return \_x; }

set

{

\_x = value;

}

}

public double Y

{

get { return \_y; }

set

{

if (value > X)

\_y = value;

else

throw new ArithmeticException();

}

}

public Diapason()

{

this.\_x = 0;

this.\_y = 0;

counter++;

}

public Diapason(double x, double y)

{

X = x;

Y = y;

counter++;

}

public bool Intersect(Diapason diaposon2)

{

if (Y < diaposon2.X || X > diaposon2.Y)

return false;

return true;

}

public void Show()

{

Console.WriteLine($"[{X};{Y}]");

}

public static bool Intersects(Diapason diapason1, Diapason diapason2)

{

if (diapason1.Y < diapason2.X || diapason1.X > diapason2.Y)

return false;

return true;

}

}

}

**Класс Program**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab9

{

class Program

{

static void Main(string[] args)

{

Diapason diapason1 = new Diapason();

InputDiapason(ref diapason1);

Diapason diapason2 = new Diapason();

InputDiapason(ref diapason2);

Console.WriteLine("Диапазон 1 пересекает диапазон 2 (метод класса) - " + diapason1.Intersect(diapason2));

Console.WriteLine("То же самое через статическую функцию - " + Diapason.Intersects(diapason1, diapason2));

Console.WriteLine("Количество объектов класса Diapason - " + diapason1.Count);

Console.WriteLine("1-й диапазон");

diapason1.Show();

Console.WriteLine("2-й диапазон");

diapason2.Show();

}

static void InputDiapason(ref Diapason diapason)

{

double begin, end;

begin = InsertVariables("Введите значение х");

diapason.X = begin;

bool caughtError = false;

do

{

end = InsertVariables("Введите значение y");

try

{

diapason.Y = end;

caughtError = true;

}

catch (ArithmeticException)

{

Console.WriteLine("Ошибка!");

}

} while (!caughtError);

}

static int InsertVariables(string toAsk)

{

int x;

bool isParsed = false;

do

{

Console.WriteLine(toAsk);

if (int.TryParse(Console.ReadLine(), out x))

isParsed = true;

else

Console.WriteLine("Ошибка ввода! Попробуйте ввести заново!");

} while (!isParsed);

return x;

}

}

}

**Задача 2**

**Класс Diapason**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Part2

{

public class Diapason

{

double \_x, \_y;

static int counter = 0;

public int Count

{

get { return counter; }

}

public double X

{

get { return \_x; }

set

{

\_x = value;

}

}

public double Y

{

get { return \_y; }

set

{

if (value > X)

\_y = value;

else

throw new ArithmeticException();

}

}

public Diapason()

{

this.\_x = 0;

this.\_y = 0;

counter++;

}

public Diapason(double x, double y)

{

X = x;

Y = y;

counter++;

}

public bool Intersect(Diapason diaposon2)

{

if (Y < diaposon2.X || X > diaposon2.Y)

return false;

return true;

}

public static bool Intersects(Diapason diapason1, Diapason diapason2)

{

if (diapason1.Y < diapason2.X || diapason1.X > diapason2.Y)

return false;

return true;

}

public void Show()

{

Console.WriteLine($"[{X};{Y}]");

}

public string Write()

{

return "[" + X + "; " + Y + "]";

}

//////////////// OVERLOADING \\\\\\\\\\\\\\\\\

public static double operator !(Diapason toCount)

{

return toCount.Y - toCount.X;

}

//неявное преобразование

public static implicit operator int(Diapason toFloor)

{

return (int)toFloor.X;

}

//явное преобразование

public static explicit operator double(Diapason toFloor)

{

return toFloor.Y;

}

public static Diapason operator - (double number, Diapason diapason1)

{

return new Diapason(diapason1.X - number, diapason1.Y);

}

public static Diapason operator - (Diapason diapason1, double number)

{

return new Diapason(diapason1.X, diapason1.Y - number);

}

public static bool operator <(Diapason diapason1, Diapason diapason2)

{

if (diapason1.Y < diapason2.X || diapason1.X > diapason2.Y)

return false;

return true;

}

public static bool operator >(Diapason diapason1, Diapason diapason2)

{

return !(diapason1 < diapason2);

}

public static bool operator ==(Diapason diapason1, Diapason diapason2)

{

return diapason1.X == diapason2.X && diapason1.Y == diapason2.Y;

}

public static bool operator !=(Diapason diapason1, Diapason diapason2)

{

return !(diapason1 == diapason2);

}

public override bool Equals(object obj)

{

if (obj is Diapason)

{

Diapason d = (Diapason)obj;

return X == d.X && Y == d.Y;

}

return false;

}

}

}

**Класс Program**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Part2

{

class Part2

{

static void Main(string[] args)

{

Diapason diapason1 = new Diapason();

InputDiapason(ref diapason1);

Console.WriteLine($"Длина диапазона: {!diapason1}");

int flooredX = diapason1;

Console.WriteLine($"Неявное преобразование типа: {flooredX}");

Console.WriteLine($"Явное преобразование типа: {(double)diapason1}");

ShowMinusOverloading(diapason1);

Console.WriteLine("Введите второй диапазон");

Diapason diapason2 = new Diapason();

InputDiapason(ref diapason2);

Console.WriteLine($"Диапазон 1 пересекает диапазон 2: {diapason1 < diapason2}");

}

static void ShowMinusOverloading(Diapason diapason1)

{

bool correctSubstaction = false;

do

{

double toSubtract = InsertVariables("\nВведите число, которое нужно отнять");

try

{

Diapason check2 = toSubtract - diapason1;

Console.WriteLine($"Перегрука опеатора -: {check2.Write()}");

Diapason check = diapason1 - toSubtract;

Console.WriteLine($"Перегрука опеатора -: {check.Write()}");

correctSubstaction = true;

}

catch (ArithmeticException)

{

Console.WriteLine("Ошибка");

}

} while (!correctSubstaction);

}

static void InputDiapason(ref Diapason diapason)

{

double begin, end;

begin = InsertVariables("Введите значение х");

diapason.X = begin;

bool caughtError = false;

do

{

end = InsertVariables("Введите значение y");

try

{

diapason.Y = end;

caughtError = true;

}

catch (ArithmeticException)

{

Console.WriteLine("Ошибка!");

}

} while (!caughtError);

}

static double InsertVariables(string toAsk)

{

double x;

bool isParsed = false;

do

{

Console.WriteLine(toAsk);

if (double.TryParse(Console.ReadLine(), out x))

isParsed = true;

else

Console.WriteLine("Ошибка ввода! Попробуйте ввести заново!");

} while (!isParsed);

return x;

}

}

}

**Задача 3**

**Класс Diapason**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Part3

{

public class Diapason

{

double \_x, \_y;

static int counter = 0;

public int Count

{

get { return counter; }

}

public double X

{

get { return \_x; }

set

{

\_x = value;

}

}

public double Y

{

get { return \_y; }

set

{

if (value > X)

\_y = value;

else

throw new ArithmeticException();

}

}

public Diapason()

{

this.\_x = 0;

this.\_y = 0;

counter++;

}

public Diapason(double x, double y)

{

X = x;

Y = y;

counter++;

}

public bool Intersect(Diapason diaposon2)

{

if (Y < diaposon2.X || X > diaposon2.Y)

return false;

return true;

}

public static bool Intersects(Diapason diapason1, Diapason diapason2)

{

if (diapason1.Y < diapason2.X || diapason1.X > diapason2.Y)

return false;

return true;

}

public void Show()

{

Console.WriteLine($"[{X};{Y}]");

}

public string Write()

{

return "[" + X + "; " + Y + "]";

}

//////////////// OVERLOADING \\\\\\\\\\\\\\\\\

public static double operator !(Diapason toCount)

{

return toCount.Y - toCount.X;

}

//неявное преобразование

public static implicit operator int(Diapason toFloor)

{

return (int)toFloor.X;

}

//явное преобразование

public static explicit operator double(Diapason toFloor)

{

return toFloor.Y;

}

public static Diapason operator -(double number, Diapason diapason1)

{

return new Diapason(diapason1.X - number, diapason1.Y);

}

public static Diapason operator -(Diapason diapason1, double number)

{

return new Diapason(diapason1.X, diapason1.Y - number);

}

public static bool operator <(Diapason diapason1, Diapason diapason2)

{

if (diapason1.Y < diapason2.X || diapason1.X > diapason2.Y)

return false;

return true;

}

public static bool operator >(Diapason diapason1, Diapason diapason2)

{

return !(diapason1 < diapason2);

}

public static bool operator ==(Diapason diapason1, Diapason diapason2)

{

return diapason1.X == diapason2.X && diapason1.Y == diapason2.Y;

}

public static bool operator !=(Diapason diapason1, Diapason diapason2)

{

return !(diapason1 == diapason2);

}

public override bool Equals(object obj)

{

if (obj is Diapason)

{

Diapason d = (Diapason)obj;

return X == d.X && Y == d.Y;

}

return false;

}

}

}

**Класс DiapasonArray**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Part3

{

public class DiapasonArray

{

static Random rnd = new Random();

Diapason[] arr = null;

int \_size;

static int counter = 0;

public int Count

{

get { return counter; }

}

public Diapason this[int index]

{

get

{

if (index >= 0 && index < arr.Length)

return arr[index];

else

throw new IndexOutOfRangeException();

}

set

{

if (index >= 0 && index < arr.Length)

arr[index] = value;

else

throw new IndexOutOfRangeException();

}

}

public int Size

{

get { return \_size; }

}

public DiapasonArray()

{

arr = null;

counter++;

}

public DiapasonArray(int size)

{

\_size = size;

arr = new Diapason[size];

for (int i = 0; i < size; i++)

{

int value = rnd.Next(-50, 50), val2 = rnd.Next(25, 100);

arr[i] = new Diapason(value, value + val2);

}

counter++;

}

public DiapasonArray(int size, int h)

{

\_size = size;

arr = new Diapason[size];

for (int i = 0; i < size; i++)

{

arr[i] = new Diapason();

InputDiapason(ref arr[i]);

}

counter++;

}

public void Write()

{

int count = 1;

foreach (Diapason item in arr)

{

Console.WriteLine($"{count}) {item.Write()}");

count++;

}

}

public static bool operator ==(DiapasonArray diapason1, DiapasonArray diapason2)

{

return diapason1.Size == diapason2.Size;

}

public static bool operator !=(DiapasonArray diapason1, DiapasonArray diapason2)

{

return !(diapason1 == diapason2);

}

static void InputDiapason(ref Diapason diapason)

{

double begin, end;

begin = InsertVariables("Введите значение х");

diapason.X = begin;

bool caughtError = false;

do

{

end = InsertVariables("Введите значение y");

try

{

diapason.Y = end;

caughtError = true;

}

catch (ArithmeticException)

{

Console.WriteLine("Ошибка!");

}

} while (!caughtError);

}

static double InsertVariables(string toAsk)

{

double x;

bool isParsed = false;

do

{

Console.WriteLine(toAsk);

if (double.TryParse(Console.ReadLine(), out x))

isParsed = true;

else

Console.WriteLine("Ошибка ввода! Попробуйте ввести заново!");

} while (!isParsed);

return x;

}

}

}

**Класс Program**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Part3

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Диапазоны сгенерированные с помощью ДСЧ");

DiapasonArray diapasons1 = new DiapasonArray(5);

diapasons1.Write();

Console.WriteLine("\nДиапазоны вводимые пользователем");

int count = InsertVariables("Введите количество диапазонов");

DiapasonArray diapasons2 = new DiapasonArray(count, 0);

diapasons2.Write();

Console.WriteLine($"Количество экземпляров класса DiapasonArray: { diapasons2.Count }");

Console.WriteLine("Максимальное значение в массиве диапазонов 1");

Diapason max = diapasons1[0];

for (int i = 1; i < diapasons1.Size; i++)

{

if (!max < !diapasons1[i])

max = diapasons1[i];

}

max.Show();

}

static int InsertVariables(string toAsk)

{

int x;

bool isParsed = false;

do

{

Console.WriteLine(toAsk);

if (int.TryParse(Console.ReadLine(), out x) & x > 0)

isParsed = true;

else

Console.WriteLine("Ошибка ввода! Попробуйте ввести заново!");

} while (!isParsed);

return x;

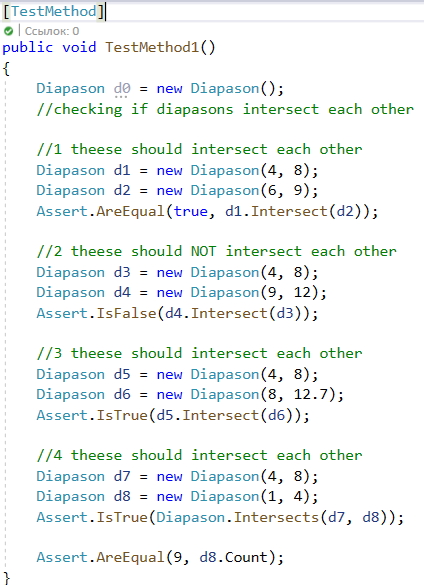
}

}

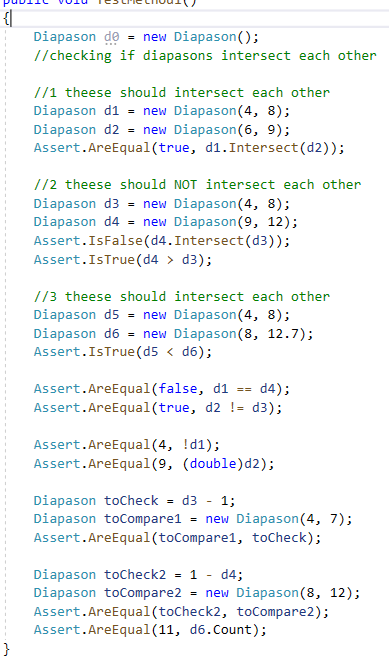
}

**Тесты**

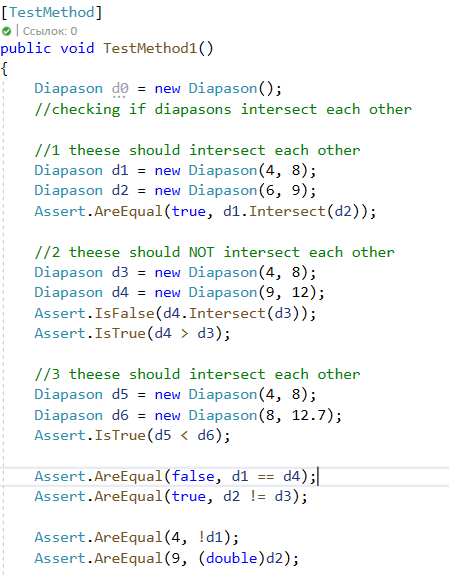
**Задача 1**

****

**Задача 2**

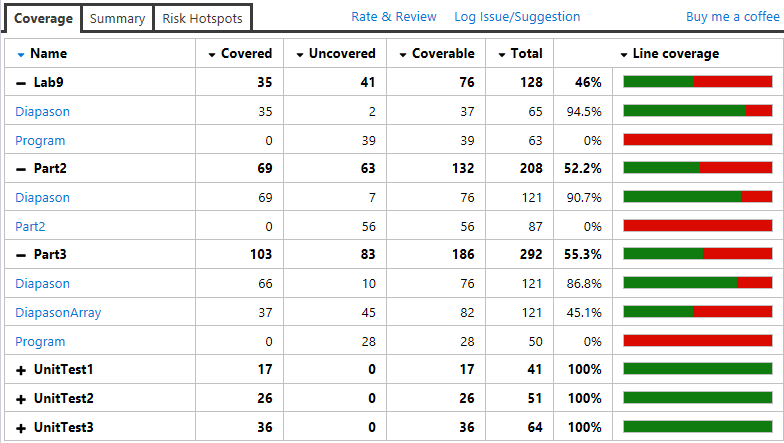
****

**Задача 3**

****

****

**Таблица покрытия тестами**

****