Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Институт информационных технологий

Факультет компьютерных технологий

Кафедра информационных систем и технологий

Контрольная работа

по дисциплине

«Конструирования программного обеспечения»

|  |  |
| --- | --- |
|  | Выполнил студент группы 181073  Шерстнёв Артур Александрович |
|  |  |

Минск 2022

Основная цель выполнения контрольной работы состоит в практической реализации теоретических вопросов курса при изучении дисциплины «Конструирование программного обеспечения».

**Часть 1: «Проектирование классов»**

**Предметная область –** здравоохранение**. Необходимо спроектировать:**

**‒ Не менее 5 классов**

**‒ Не менее 3 полей для каждого класса.**

**‒ Не менее 3 операций (методов) для каждого класса.**

**‒ Имена классов – существительные.**

**Варианты заданий:**

1. Здравоохранение.

**Листинг кода решения:**

namespace ConsoleApplicatio11

{

public class Program

{

static void Main(string[] args)

{

Console.WriteLine("Hello World!");

}

}

public interface IPerson

{

public string Name{ get; }

public DateOnly DateOfBirth{ get; }

}

public class Patient : IPerson

{

public string Name{ get; }

public DateOnly DateOfBirth{ get; }

public string OutpatientCardNumber{ get; }

private List<Voucher> \_vouchers{ get; }

public Patient(string name, DateOnly dateOfBirth, string outpatientCardNumber)

{

OutpatientCardNumber = outpatientCardNumber;

Name = name;

DateOfBirth = dateOfBirth;

\_vouchers = new List<Voucher>();

}

public void GetVouchers(Voucher voucher)

{

\_vouchers.Add(voucher);

}

public void ViewAllVoucher()

{

\_vouchers.ForEach(Console.WriteLine);

}

public override string ToString()

{

return $"Patient {Name}\nDate of Birth = {DateOfBirth}\nOutpatient card number = {OutpatientCardNumber}";

}

}

public class Voucher

{

public DateTime Time{ get; }

public DoctorType DoctorType{ get; }

public Voucher(DateTime time, DoctorType doctorType)

{

Time = time;

DoctorType = doctorType;

}

public override string ToString()

{

return $"Appointment to the {DoctorType} at {Time}";

}

}

public enum DoctorType

{

Surgeon = 0,

Therapist = 1,

Dentist = 2

}

public abstract class Doctor : IPerson

{

protected Doctor(string name, DateOnly dateOfBirth, DoctorType doctorType)

{

Name = name;

DateOfBirth = dateOfBirth;

DoctorType = doctorType;

}

public string Name{ get; }

public DateOnly DateOfBirth{ get; }

public DoctorType DoctorType{ get; }

public abstract void Examination(Patient patient);

public abstract void WritePrescription(Patient patient);

public abstract void IssueSickLeave(Patient patient);

public void CloseSickLeave(Patient patient)

{

Console.WriteLine("Close sick leave");

}

}

public class Surgeon : Doctor

{

public Surgeon(string name, DateOnly dateOfBirth, DoctorType doctorType) : base(name, dateOfBirth, doctorType)

{

}

public override void Examination(Patient patient)

{

Console.WriteLine($"Surgeon start examination {patient.Name}");

}

public override void WritePrescription(Patient patient)

{

Console.WriteLine($"Surgeon start write prescription {patient.Name}");

}

public override void IssueSickLeave(Patient patient)

{

Console.WriteLine($"Surgeon start issue sick leave {patient.Name}");

}

public override string ToString()

{

return $"Surgeon {Name}\nDate of Birth = {DateOfBirth}";

}

}

public class Therapist : Doctor

{

public Therapist(string name, DateOnly dateOfBirth, DoctorType doctorType) : base(name, dateOfBirth, doctorType)

{

}

public override void Examination(Patient patient)

{

Console.WriteLine($"Therapist start examination {patient.Name}");

}

public override void WritePrescription(Patient patient)

{

Console.WriteLine($"Therapist start write prescription {patient.Name}");

}

public override void IssueSickLeave(Patient patient)

{

Console.WriteLine($"Therapist start issue sick leave {patient.Name}");

}

public override string ToString()

{

return $"Therapist {Name}\nDate of Birth = {DateOfBirth}";

}

}

public class Dentist : Doctor

{

public Dentist(string name, DateOnly dateOfBirth, DoctorType doctorType) : base(name, dateOfBirth, doctorType)

{

}

public override void Examination(Patient patient)

{

Console.WriteLine($"Dentist start examination {patient.Name}");

}

public override void WritePrescription(Patient patient)

{

Console.WriteLine($"Dentist start write prescription {patient.Name}");

}

public override void IssueSickLeave(Patient patient)

{

Console.WriteLine($"Dentist start issue sick leave {patient.Name}");

}

public override string ToString()

{

return $"Dentist {Name}\nDate of Birth = {DateOfBirth}";

}

}

public static class DataBase

{

private static List<Patient> \_patients = new List<Patient>();

private static List<Doctor> \_doctors = new List<Doctor>();

private static List<Voucher> \_vouchers = new List<Voucher>();

public static IReadOnlyList<Patient> GetPatients()

{

return \_patients;

}

public static void AddPatients(params Patient[] patients)

{

\_patients.AddRange(patients);

}

public static IReadOnlyList<Doctor> GetDoctors()

{

return \_doctors;

}

public static void AddDoctors(params Doctor[] doctors)

{

\_doctors.AddRange(doctors);

}

public static IReadOnlyList<Doctor> GetVoucher()

{

return \_doctors;

}

public static void AddVoucher(params Voucher[] vouchers)

{

\_vouchers.AddRange(vouchers);

}

}

}

Часть 2: «Реализация классов»

Организовать проверку существования необходимой фигуры.

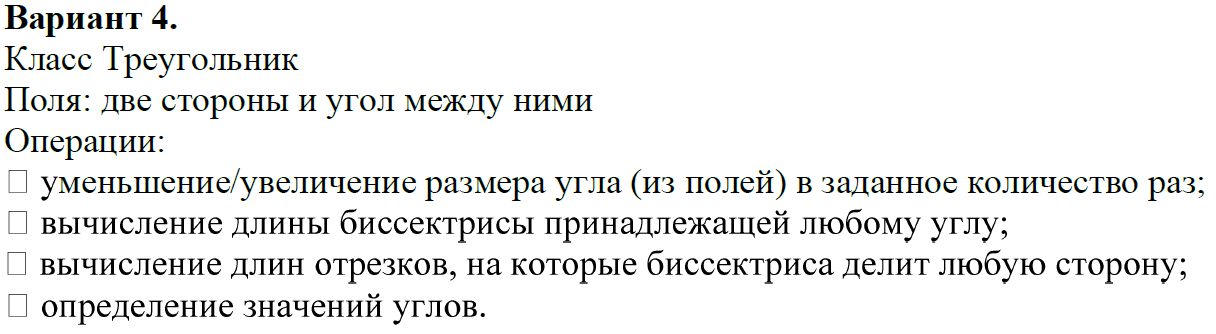
В каждом классе необходимо дополнительно реализовать:

‒ закрытые поля;

‒ к каждому полю свойства;

‒ 3 конструктора;

‒ метод ToString().



Triangle ABC = new Triangle();

Console.WriteLine("Введите стороный треугольника");

ABC.A = Convert.ToDouble(Console.ReadLine());

ABC.B = Convert.ToDouble(Console.ReadLine());

Console.WriteLine("Введите угол между сторонами треугольника");

ABC.ANGL = Convert.ToDouble(Console.ReadLine());

Console.WriteLine("\nУвеличение или уменьшение угол между сторонами треугольника:\n1)Увеличение\n2)уменьшение");

int upOrDown = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("Ведите колличество раз увеличения:");

int valueUpOrDown = Convert.ToInt32(Console.ReadLine());

switch (upOrDown)

{

case 1:

Console.WriteLine("Угол = {0}", ABC.upAngl(valueUpOrDown));

break;

case 2:

Console.WriteLine("Угол = {0}", ABC.downAngl(valueUpOrDown));

break;

default:

Console.WriteLine("default");

break;

}

double c = Math.Round(ABC.valueC(ABC.A, ABC.B, ABC.ANGL));

double betta = ABC.Beta((Convert.ToInt32(ABC.Gamma(ABC.A, ABC.B, Math.Round(ABC.valueC(ABC.A, ABC.B, ABC.ANGL))))), (Convert.ToInt32(ABC.ANGL)));

double gamma = ABC.Gamma(ABC.A, ABC.B, ABC.valueC(ABC.A, ABC.B, ABC.ANGL));

Console.Write("\nДлинные бессектрис треугольника:");

Console.WriteLine("\nУгола альфа = {0}", ABC.bisector(ABC.A, c,ABC.ANGL));

Console.WriteLine("Угола бетта = {0}", ABC.bisector(ABC.A, ABC.B, betta));

Console.WriteLine("Угола гамма = {0}", ABC.bisector(ABC.A, c, gamma));

Console.Write("\nДлинны сторон");

Console.WriteLine("\nДлинны отрезка а = {0} и {1}", Math.Round(ABC.lengthFirst(ABC.B, c, ABC.A)), Math.Round(ABC.lengthSecond(ABC.B, c, ABC.A)));

Console.WriteLine("Длинны отрезка b = {0} и {1}", Math.Round(ABC.lengthFirst(c, ABC.A, ABC.B)), Math.Round(ABC.lengthSecond(c, ABC.A, ABC.B)));

Console.WriteLine("Длинны отрезка c = {0} и {1}", Math.Round(ABC.lengthFirst(ABC.A, ABC.B, c)), Math.Round(ABC.lengthSecond(ABC.A, ABC.B, c)));

Console.WriteLine("\nУгол альфа = {0}", Convert.ToInt32(ABC.ANGL));

Console.WriteLine("Угол бетта = {0}", betta);

Console.WriteLine("Угол гама = {0}", Convert.ToInt32(gamma));

class Triangle

{

private double a, b, anglAB;

public double A

{

get { return a; } // возвращаем

set { a = value; } // устанавливаем новое

}

public double B

{

get { return b; } // возвращаем

set { b = value; } // устанавливаем новое

}

public double ANGL

{

get { return anglAB; } // возвращаем

set { anglAB = value; } // устанавливаем новое

}

public Triangle(Triangle Triangle)

{

a = Triangle.a;

b = Triangle.b;

anglAB = Triangle.anglAB;

}

public Triangle(double a, double b, double anglAB)

{

this.a = a;

this.b = b;

this.anglAB = anglAB;

}

public Triangle()

{

a = 0;

b = 0;

anglAB = 0;

}

public override string ToString()

{

return "\r\n" + base.ToString() + "\r\n" + "1 сторона = " + a + "\r\n" + "2 сторона = " + b + "\r\n" + "Угол межбу сторонами = " + anglAB;

}

public double upAngl(double p)

{

return this.anglAB = anglAB \* p;

}

public double downAngl(double p)

{

return this.anglAB = anglAB / p;

}

public double valueC(double a, double b, double anglAB)

{

return Math.Sqrt((a \* a) + (b \* b) - ((2 \* a \* b) \* Math.Cos(Math.PI \* anglAB / 180.0)));

}

public double Beta(double Gamma, double Alpha)

{

double x = 180;

return (x - (Gamma + Alpha));

}

public double Gamma(double a, double b, double c)

{

return (((Math.Acos(((b \* b) + (c \* c) - (a \* a)) / (2 \* c \* b))) \* 180) / 3.14);

}

public double bisector(double b, double c, double angl)

{

return ((2 \* b \* c)/(b+c))\*Math.Cos((Math.PI \* angl / 180.0)/2) ;

}

public double lengthFirst(double a,double b,double c)

{

return (a \* c) / (a + b);

}

public double lengthSecond(double a, double b, double c)

{

return (b \* c) / (a + b);

}

}

Вывод:Проектирование классов с соблюдением принципов ООП является основной задачей программиста для создания понятного, удобного в расширении и гибкого кода.